

Top- K Miner: top- K identical frequent itemsets discovery without user support threshold

Saif-Ur-Rehman¹ · Jawad Ashraf¹ · Asad Habib¹ ·
Abdus Salam²

Received: 14 July 2014 / Revised: 12 September 2015 / Accepted: 24 November 2015 /
Published online: 31 December 2015
© Springer-Verlag London 2015

Abstract Frequent itemsets (FIs) mining is a prime research area in association rule mining. The customary techniques find FIs or its variants on the basis of either support threshold value or by setting two generic parameters, i.e., N (topmost itemsets) and K_{\max} (size of the itemsets). However, users are unable to mine the absolute desired number of patterns because they tune these approaches with their approximate parameters settings. We proposed a novel technique, top- K Miner that does not require setting of support threshold, N and K_{\max} values. Top- K Miner requires the user to specify only a single parameter, i.e., K to find the desired number of frequent patterns called identical frequent itemsets (IFIs). Top- K Miner uses a novel candidate production algorithm called join-FI algorithm. This algorithm uses frequent 2-itemsets to yield one or more candidate itemsets of arbitrary size. The join-FI algorithm follows bottom-up recursive technique to construct candidate-itemsets-search tree. Finally, the generated candidate itemsets are manipulated by the *Maintain-Top- K -List* algorithm to produce *Top- K -List* of the IFIs. The proposed top- K Miner algorithm significantly outperforms the generic benchmark techniques even when they are running with the ideal parameters settings.

Keywords Frequent itemsets · Association rules · Identical frequent itemsets (IFIs) · Candidate- itemsets-search tree

✉ Saif-Ur-Rehman
saifyabbas@yahoo.com

Jawad Ashraf
jawad.ashraf@kust.edu.pk

Asad Habib
asadsan@gmail.com

Abdus Salam
dr.salam@abasyn.edu.pk

¹ Institute of Information Technology, Kohat University of Science and Technology, Kohat, Pakistan

² Computer Science Department, Abasyn University, Peshawar, Pakistan

1 Introduction

Frequent itemsets (FIs) mining has been among the most challenging researched areas in association rule mining (ARM) over the past decade. ARM is a two steps process; (1) finding the FIs from the data sets on the basis of support followed by (2) deduction of association rules from the already mined FIs [1]. The FIs discovery process is a NP-Complete problem that demands extensive computational resources [2]. FIs generation is useful in many computing problems such as consumer market basket analysis [1], network intrusion detection [3], Web page access-log analysis [4], document analysis [5], telecom data analysis [6], and biological data analysis [7].

Agrawal et al. [1] in his seminal work on ARM proposed Apriori algorithm to compute the frequent itemsets (FIs). Apriori algorithm discovers set of all FIs that qualify a given support threshold using bottom-up strategy. The promising feature of this algorithm is the Apriori property, i.e., all subsets of FIs are also frequent. Thus resultant frequent itemset search space includes all frequent itemsets including frequent sub-itemsets. The subsequent research work introduced different variants of FIs to reduce the result set. The main categories of these variants include maximal frequent itemsets (MFIs) [8–10], closed frequent itemsets (CFIs) [11–14], maximal length frequent itemsets (MLFIs) [15], and colossal patterns [16–18].

A FI having support greater than the given support threshold value and is not subset of any other FI is called MFI [8]. Max-Miner [8] performs breadth-first, bottom-up traversal of search enumeration tree for mining MFIs. Burdick et al. introduced another MFIs mining algorithm called maximal frequent itemsets algorithm (MAFIA) [9]. This method performs depth first traversal of the search space with parent equivalence, look ahead, and superset frequency pruning in order to enhance performance. Another method called GenMax discovers all MFIs using backtrack search tree method [10]. It employ items reordering to minimize the size of the combine set and to remove lower frequency nodes early in backtrack search tree. GenMax also uses progressive focusing technique making superset checking practical on dense data sets. The second variant of FI is CFIs mining technique [11–14]. The CFI is compressed form of FIs. An itemset X is called CFI, if there is no proper superset Y of X such that X and Y have the same support in transaction data set D [11]. Given a support threshold any CFI and all its subsets have the same support. Hence, a user can find CFIs and all subsets of CFIs along with its frequency on a given data set. CLOSET is one of the primary techniques proposed to mine all CFIs on a given support threshold value [12]. It uses frequent pattern tree (FP-tree) structure to represent all transactions in a given data set in RAM [19]. CLOSET is a partition-based projection and divide-and-conquer strategy to determine all CFIs. CHARM enumerates all CFIs by using a novel Itemset-Tidset tree (IT-tree) search space structure [13]. It avoids computation of many levels of the IT-tree by applying efficient hybrid search approach. CHARM also uses *diffsets* to efficiently store the transaction id's (tid's) of the itemsets in memory while searching for CFIs. MLFI is another variant of FI representing maximum number of items in an itemset [15]. LFIMiner and LFIMinerALL are MLFI finding techniques using FP-growth algorithm [19]. LFIMiner finds a single MLFI, whereas LFIMinerAll finds set of MLFIs based on support threshold. The basic FI mining methods [1, 19] and its variants [8–14, 16–18] mine traditional commercial data sets more efficiently as compared to high-dimensional data sets. The commercial data sets have large number of transactions and small average transaction length. Applying traditional mining methods (basic FI mining, CFI, MFI) to high-dimensional data sets is inefficient. Only long-sized patterns persisting in high-dimensional data sets (such as gene expression data sets in bioinformatics and program

trace data sets in software engineering etc.) represent complete and required information. Thus different frequent pattern mining methodologies were introduced to mine very long-sized patterns called colossal patterns [16–18]. The colossal pattern algorithms directly mine high-dimensional data sets to produce only the colossal patterns, ignoring the small- and mid-sized patterns.

FIs and its variants such as MFI, CFI, MLFI, and colossal patterns techniques require tuning of support threshold parameter [1, 8–14, 16–19]. Regulating the right support threshold parameter value to get the only required frequent itemsets is considered a challenging task for the user [20]. It is difficult to determine the ideal value of the support threshold parameter without having prior knowledge about the characteristics of data set. Failure to tune the right support threshold parameter can cause the whole process of data mining worthless. Inappropriate setting of the support threshold parameter may cause the following issues.

- Analysts have to assess the resultant patterns manually to decide whether these patterns hold significant importance for decision-making. Exponential number of frequent patterns may be generated due to setting low support threshold poses great challenge for user while using them in decision-making process.
- The user may be naïve and fail to understand role of support threshold in the mining process. When user sets lower support threshold, it may cause an algorithm to produce either exponential number of frequent patterns or may cause an algorithm to run out of memory or reasonable disk space.
- The FIs mining process is used as a preliminary step for ARM. When the FIs mining algorithm runs out of memory or disk space, then the association rule discovery step may not be achieved.
- A very small value of support threshold may cause the FIs mining algorithm to produce spurious patterns, not desired by the user.
- A very large value of support threshold may cause the FIs mining algorithm to skip the desired patterns in the FIs result set.
- The generated frequent patterns may subsequently be used as input to other data mining tasks such as clustering, classification, and outlier detection. The time and space complexity of these data mining tasks may increase due to exponential number of frequent patterns.
- The frequent itemset mining is an exploratory data mining task. The support threshold-based FIs mining algorithm imposes our own presumptions and prejudices on the problem and do not allow the data itself speak to us.

Several techniques of frequent itemsets mining have been proposed by the researchers to avoid the support threshold parameter. For the first time Shen et al. [33] introduced Apriori-inspired support threshold-free topmost frequent patterns mining. This algorithm finds the N (number) of itemsets of highest frequency in a given data set. Fu et. al proposed another Apriori-inspired support threshold-free Itemset-Loop/iLoop algorithms. These approaches find the topmost N itemsets of highest frequency in each set of K -itemsets [22]. All the Apriori-inspired approaches for topmost frequent itemsets mining have the same demerits as that in Apriori approach [34]. There are other support threshold-free procedures to perform top- K frequent itemsets mining using the classical FP-growth methodology (i.e., FP-tree) [21, 24, 34–36]. Apart from the other demerits in every FP-growth inspired technique for top- K frequent itemsets mining, they encompass two common issues. First, FP-growth-inspired top- K frequent itemsets mining scan the given data set at least twice. Second, it transforms the entire data set to FP-tree.

Our top- K Miner approach initiates the top- K frequent itemsets discovery process by scanning the given data set only once. During this process top- K Miner creates a set of frequent 2-itemsets. This set is sorted in descending support order. Top- K Miner then iteratively selects frequent 2-itemset to create a CIS-tree using the *join-FI* procedure. This procedure returns the set of possible candidate itemsets (along with their support count) C in CIS-tree for a selected frequent 2-itemset. The set of candidate itemsets C is subsequently evaluated by the *Maintain-Top-K-List* procedure to return the updated list of top- K FIs named *Top-K-List* so far. This list is updated with the top- K FIs for every frequent 2-itemset with the set of possible candidate itemsets C . The top- K FIs mining process is stopped when the minimum support count of an itemset in the *Top-K-List* is greater than the maximum support count of an itemset in the set of possible candidate itemsets C .

The rest of the paper is organized in the following manner. Related work is discussed in the next section. Section 3 contains preliminaries for top- K frequent pattern mining. The top- K Miner algorithmic description is given in Sect. 4. Experimental results are reported in Sect. 5, also containing brief discussion over the performance trends of top- K Miner with other benchmark techniques like BOMO [21] and FP-growth [19]. Conclusion and future work is presented in Sect. 6.

2 Related work

Numerous algorithms have been presented in the last decade to eradicate the support threshold parameter from frequent itemsets mining process. The support threshold-free techniques need minimum two parameters to tune them, which is a mind-boggling exercise for the user. To the best of our knowledge Itemset-Loop and Itemset-iLoop were the first algorithms, to put limits on the number of frequent itemsets obtained rather than on support threshold [22]. Itemsets-Loop and Itemsets-iLoop algorithms are used to produce N -most interesting itemsets. The N -most interesting itemsets is the union of N -most interesting K -itemsets for each $1 \leq K \leq m$ [22], where N is the number of most interesting itemsets required by the user and m is the upper bound on the size of interesting itemsets. Both parameters are required by Itemsets-Loop and Itemsets-iLoop at the start. A Build-Once and Mine-Once approach named as BOMO is used to find N -most interesting FIs. BOMO needs to tune two parameters N and K_{\max} [21]. BOMO represents all transactions in the given data set in compressed format using FP-tree structure. In mining phase, it uses NFP-Mine algorithm to extract patterns from the already constructed FP-tree. NFP-Mine mines the patterns for each element in the header of the tree by constructing conditional pattern base and conditional FP-tree recursively [21]. BOMO carry out lots of recursive calls while mining N -most interesting K -itemsets from FP-tree. COFI+BOMO is an algorithm to mine N -most interesting itemsets without support threshold value [23]. COFI+BOMO technique is inspired mainly from the BOMO and uses co-occurrence frequent-item-trees (COFI-tree) structure [24]. COFI+BOMO technique avoids large number of recursive call while performing mining but requires two parameters [24]. COFI+BOMO algorithm consume more memory than BOMO at any particular instance of time. Slam and Khayal proposed a technique to find out topmost maximal frequent itemset (TMMFI) and top- K maximal frequent itemsets (TKMFI) [25]. TMMFI is a FI of length equal or >3 having maximum support than all other itemsets of length equal or >3 [25]. TKMFI refers to the set of k number of maximal FIs having support greater or equal to the other members of TKMFI set [25]. This technique mines TKMFI by adopting novel AS-graph based approach [25]. Li et al. [26] proposed TGP algorithm to produce top- K frequent

closed graph patterns without minimum support threshold. TGP uses a novel lexicographic pattern net structure to store candidate graph patterns and their relation with other candidate patterns. TGP needs two parameters, i.e., K representing number frequent closed graph patterns and min_size representing the size of the patterns greater than the specified size constraint. TGP is a NP-Complete problem and a graph based top- K frequent closed pattern mining technique. TGP results significant performance degradation if applied to the data sets with larger transaction length and produces huge graph search space [26]. *Max-Clique* is a technique to find top- K maximal frequent patterns proposed by Xie and Yu [27]. Max-Clique discovers the K number of maximal frequent patterns having largest length than the other existing maximal frequent patterns. Max-Clique follows top-down search strategy for the discovery of frequent patterns rather than conventional bottom-up frequent pattern discovery approaches [27]. Okubo and Haraguchi [28] proposed branch-and-bound algorithm to mine top- N colossal patterns. top- N colossal patterns are those maximal patterns having N (user specified) largest length in the available maximal patterns. Candidate colossal patterns are detected from the pattern graph as the maximum clique. These candidate colossal patterns are subsequently evaluated using specified N and frequency threshold values to determine top- N colossal patterns [28].

3 Ground work

The FIs computing without support threshold is the most feasible choice to find out the desired number of patterns. Based on the discussion in the previous section, it is quite obvious that all FIs mining techniques need two types of parameters. The primitive group of FIs mining technique demands support threshold parameter. These FIs mining techniques make use of the support threshold parameter to generate either lesser or redundant number of FIs. The second class of technique requires parameter other than support threshold, to generate user specified number of FIs. But these techniques have tendency to miss some important FIs because of user's parameter settings. Therefore, we need to revisit user's needed FIs finding mechanism to produce only the user required number of patterns. For this purpose, important definitions are given below.

Definition 1 Identical frequent itemset (IFI)

A set of frequent itemsets $F = \{X_1, X_2, X_3, \dots, X_m\}$ is called IFI if $X_i \in F, \forall i = 1, 2, 3, \dots, m$ have same support count where the length of X_i is arbitrary.

Definition 2 Top- K identical frequent itemsets (Top- K IFIs)

Top- K IFIs is a set of IFIs of highest support to the K th support to be found from a given data set D is stated as follows

$$\{F_j | F_j \subset I \text{ and } support(F_j) > support(F_{j+1})\}$$

$$\forall j = 1, 2, 3, \dots, K$$

$$\text{And } \forall i = 1, 2, 3, \dots, K - 1$$

where I is the set of all IFIs and F_i or F_j is any IFI in I . For the user desired number of K the $support(F_1) > support(F_2) > support(F_3) > \dots > support(F_k)$.

Example 1 Consider the transaction data set given in Table 1 that contains six different items A, B, C, D, E and F . The data set contains 10 transactions. For the given value of $K = 7, 6, 4$ the data set is mined for top-7, top-6, or top-4 IFIs as shown in Table 2.

Table 1 Transactional data set

TID	Transactions
1	<i>A B D F</i>
2	<i>A B F</i>
3	<i>A D F</i>
4	<i>B C D E F</i>
5	<i>B C D E F</i>
6	<i>A B F</i>
7	<i>A B D F</i>
8	<i>A B F</i>
9	<i>A B C D F</i>
10	<i>A B C E F</i>

Table 2 Generated top-7, top-6, and top-4 IFIs

Top-7 IFIs	Top-6 IFIs	Top-4 IFIs
1) $F = 10$	1) $F = 10$	1) $F = 10$
2) $B = 9, BF = 9$	2) $B = 9, BF = 9$	2) $B = 9, BF = 9$
3) $A = 8, AF = 8$	3) $A = 8, AF = 8$	3) $A = 8, AF = 8$
4) $AB = 7, FBA = 7$	4) $AB = 7, FBA = 7$	4) $AB = 7, FBA = 7$
5) $D = 6, DF = 6$	5) $D = 6, DF = 6$	
6) $BD = 5, FBD = 5$	6) $BD = 5, FBD = 5$	
7) $C = 4, AD = 4, ADF = 4$		

The Example 1 illustrates that the user only provide a parameter, i.e., K to find the required number of IFIs.

It is clear from the Example 1 that the top- $(K = i)$ are the subset of top- K IFIs for any $1 \leq i < K$. It can be deduced from Example 1 that building top- K IFIs would automatically result construction of all previous IFIs. The number of FIs produced by the FIs mining methods [1, 8–14, 16–19] entirely depends upon the support threshold parameter by the user. If the user sets a high value of the support threshold parameter, less number of FIs will qualify. The FIs result set produced on high support threshold setting does not contain the FIs result set that can be obtained on low support threshold as is the case for IFIs mining. The IFIs mining with a parameter K , is simple and straight forward approach. The exact numbers of IFIs are discovered by specifying the number of IFIs required by the user in the form of K . The problem of mining top- K IFIs on a given data set can be defined as follows.

3.1 Problem definition

Given a transactional data set D , every transaction is associated with a distinct transaction identifier (tid). A set X is called an itemset. The support of an itemset X is the number of transactions in which an itemset X appears. Let F be the set of all such X itemsets having similar support called IFI, and I is the set of all IFIs such that $F \subset I$. For the given value of K , the problem of mining top- K IFIs is to find the Top- K _List containing all IFIs of highest support to the K th support.

4 Top-K Miner approach to find top-K IFIS

To determine top- K patterns from the transactional data set containing large number of items is not an easy task. Most of techniques mentioned in the related work one way or the other check every item in the data set to discover topmost FIs. Our proposed technique uses such methodology that once all frequent 2-itemsets are found, can easily deduce top- K IFIs by only picking few frequent 2-itemsets. To eradicate the support threshold parameter, all the topmost FIs mining techniques utilize two parameters, i.e., N for top most patterns and K_{\max} to set the size of patterns. Although the support threshold is only one such parameter in conventional FIs mining that the user has to predict, it is difficult for the user to guess its optimum value [20]. The previous topmost FIs mining technique has made the life of the user little easier but still the user has to tune them with one extra parameter, i.e., K_{\max} hence restricting the size of the patterns to be mined. Our top- K IFIs mining approach will not demand for the K_{\max} from the user hence allowing the given data set to speak to us that what relationship exists in the patterns at the top- K levels.

The working of the top- K Miner is based on seven steps as shown in Fig. 1. First, the top- K Miner approach finds out all the frequent 2-itemsets that exist in the given data set D . Secondly, the discovered frequent 2-itemsets are sorted in descending order as shown in step 2 of Fig. 1.

Example 2 Consider the transactional data set example in Table 1. For the given data set in Table 1 with support greater than zero, all frequent 2-itemsets are as shown in Table 3.

The step 3 and rest of the steps in top- K Miner use candidate-itemsets-search tree (CIS-tree). Before describing the remaining steps of the top- K Miner CIS-tree is defined in the following.

Definition 3 Candidate-itemsets-search tree (CIS-tree)

A tree T is called CIS-tree if and only if every node $N \in T$ is an extension of its parent node $P \in T$ with one of the item $i \in P.Tail-Items$ and $supp(N.Head-Itemset) \leq supp(P.Head-Itemset)$ excluding the root node of the tree T . where *Head-Itemset* and *Tail-Items* are two fields of a node.

All nodes in CIS-tree have identical node structure. A node in CIS-tree includes five fields as shown in the Table 4. The first two fields are *Head-Itemset* and *Tail-Items*. The *Head-Itemset* field of a node is actually a candidate itemset. The *Tail-Items* field of a node contains the prospective items that can be combined with the *Head-Itemset* field of the corresponding node, forming *Head-Itemset* field of a child node. An item having highest support count is removed from the parent node *Tail-Items* field. The removed item is joined to the parent node *Head-Itemset* field forming *Head-Itemset* field of the child node. A *Support-Count* is the third field of a node. It represents the support count of a candidate itemset (here as *Head-Itemset*). Fourth field in every node of the CIS-tree is the list of pointers to the child nodes. Fifth field in a node is *Diff-Set*. It represents the tids where *Head-Itemset* is not present.

Step 3 of the top- K Miner algorithm initializes the *Top-K_List* and *Head-Itemset* field of the root node of the CIS-tree to empty (\emptyset). The step 4 of the top- K Miner selects K frequent 2-itemset to initialize the *Tail-Items* field of the root node from the set of an already calculated frequent 2-itemsets F . The step 5 initially populates the *Top-K_List* with the frequent 1-itemsets of higher support to the K th support. The *Top-K_List* is subsequently adjusted with the other candidate itemsets, if their support is ranked higher or equal to the current top- K IFIs. The step 6 extends the root node of the CIS-tree with the child nodes. The number of items in *Tail-Items* field of the root node of CIS-tree represents the number

Algorithm Top-K Miner

T : Candidate-Itemset-Search Tree
 F : A set of all frequent 2-itemsets
 I_{\max} : Frequent 1-itemset of higher support in f
 I_{\min} : Frequent 1-itemset of support less than I_{\max} in f
 C : Candidate itemsets
 $F_{f(x_i, x_j)}$: Any set of frequent 2-itemset in F
 $f(x_i, x_j)$: A set of frequent 2-itemset

Input:

K : User desired number of IFIs
 D : Transactional data set

Output:

Top-K_List: Top-K list containing K-topmost IFIs

```

1) foreach transaction  $t \in D$ 
   foreach item  $x_i \in t, \forall i = 1, 2, 3, \dots \dots \dots |t| - 1$ 
     foreach item  $x_j \in t, \forall j = j + 1, \dots \dots \dots |t|$ 
        $F_{f(x_i, x_j)} = F_{f(x_i, x_j)} + 1$ 
2) Sort  $F$  in decreasing order of support
3)  $T.$ Head-Itemset =  $\emptyset$ , Top-K_List =  $\emptyset$ 
4) foreach frequent 2-itemset  $f_i(x, y) \in F, \forall i = 1, 2, 3, \dots \dots k$  { //loop start
    $I_{\max} = \text{MaxSupport\_1Itemset}(f_i(x, y))$ 
    $I_{\min} = f_i(x, y) - I_{\max}$ 
    $T.$ Tail-Items =  $T.$ Tail-Items  $\cup I_{\max}$ 
    $T.$ Tail-Items =  $T.$ Tail-Items  $\cup I_{\min}$ 
} // loop end
5) TopK_List = Top-K_List  $\cup T.$ Tail-Items
6) foreach item  $i \in T.$ Tail - Items { //loop start
   Create new node  $N$ 
    $N.$ Head-Itemset =  $i$ 
    $N.$ Tail-Items =  $T.$ Tail-Items -  $i$ 
    $T.$ down_link = addressof ( $N$ )
} // loop end
7) foreach frequent 2-itemset  $f \in F$  { //loop start
    $I_{\max} = \text{MaxSupport-1Itemset}(f)$ 
    $I_{\min} = f - I_{\max}$ ;
    $C = \emptyset$ ;
   Call Join_FI ( $T, I_{\max}, I_{\min}, C$ ); // to find set of candidate itemsets
   If ( $\min\_support(\text{Top} - K\_List) \leq \max\_support(C)$ )
     Maintain-TopK_List( $C, \text{Top-K\_List}$ );
   Else
     Break;
} // loop end
8) Return Top-K_List;
  
```

Fig. 1 Top-K Miner algorithm

of child nodes created in step 6. The step 7 of the top-K Miner selects frequent 2-itemset f having maximum support iteratively from a set of frequent 2-itemsets F . The maximum support item I_{\max} and minimum support item I_{\min} are deduced from f . The top-K Miner calls the join-FI algorithm with four parameters, CIS-tree T , maximum support item I_{\max} , minimum support item I_{\min} , and an empty set of candidate itemsets C . The join-FI procedure

Table 3 All frequent 2-itemsets

Sr. no.	Itemset	Support
1	<i>BF</i>	9
2	<i>AF</i>	8
3	<i>AB</i>	7
4	<i>DF</i>	6
5	<i>BD</i>	5
6	<i>AD</i>	4
7	<i>BC</i>	4
8	<i>CF</i>	4
9	<i>BE</i>	3
10	<i>CD</i>	3
11	<i>CE</i>	3
12	<i>EF</i>	3
13	<i>AC</i>	2
14	<i>DE</i>	2
15	<i>AE</i>	1

Table 4 CIS-tree node structure

Sr. no.	Field name
1	Head-Itemset
2	Tail-Items
3	Support-Count
4	Ptr-List
5	Diff-Set

is designed to return a set of candidate itemsets C for a frequent 2-itemset. The returned set of candidate itemsets C contains all possible candidate itemset having I_{\min} as the minimum support item. The top- K Miner calls *Maintain-TopK_List* procedure if the minimum support of the K th IFI in *Top-K_List* is greater or equal to the maximum support of the itemsets in C . The *Maintain-TopK_List* procedure is passed a set of candidate itemsets C and *Top-K_List* as parameters. The *Top-K_List* contains the top- K IFIs mined so far. The *Maintain-TopK_List* procedure fills in *Top-K_List* using a set of candidate itemsets C if the required condition is true.

4.1 Candidate IFIs discovery using join-FI algorithm

Top- K Miner mines the candidate itemsets using join-FI approach as shown in Fig. 2. The earliest support threshold-based FIs or topmost FIs mining techniques create lattice or tree structure for all the itemset present in the given data set and then perform topmost FIs mining on them [21, 23]. The join-FI technique is designed to perform recursive construction of the CIS-tree as shown in Fig. 2. The size of the CIS-tree is restricted to the selection of the few numbers of frequent 2-itemsets until all the top- K IFIs are generated. Hence the join-FI algorithm does not create the CIS-tree for all the items in the given data set.

```

Algorithm join_FI
Input  $T$  : CIS-tree
       $I_{max}$  : A node with item to be searched in CIS-tree
       $I_{min}$  : Item to append in new node of the CIS-tree
       $C$  : Candidate itemsets
Output  $C$ : Candidate itemsets
1) For every node  $N \in T$ 
2) If  $(\text{support}(\text{lastItem}(N.\text{Head} - \text{Itemset})) < \text{support}(I_{max}))$ 
   Continue; /* Backtrack from this path */

3) If  $(\text{support}(\text{lastItem}(N.\text{Head} - \text{Itemset})) > \text{support}(I_{max}))$ {
    $T = \text{addressof}(N)$ 
   Call  $\text{join\_FI}(T, I_{max}, I_{min}, C)$ ;
}

4) If  $(\text{lastItem}(N.\text{Head} - \text{Itemset}) == (I_{min}))$  {
   Create new node  $\hat{c}$ 
    $\hat{c}.\text{Head} - \text{Itemset} = N.\text{Head} - \text{Itemset} \cup I_{min}$ 
    $\hat{c}.\text{Tail} - \text{Item} = N.\text{Tail} - \text{Item} - I_{min}$ 
    $\hat{c}.\text{Support} - \text{Count} = \text{support}(N.\text{Head} - \text{Itemset} \cup I_{min})$ 
    $N.\text{Ptr} - \text{List} = N.\text{Ptr} - \text{List} \cup \text{addressof}(\hat{c})$ 
    $C = C \cup \hat{c}$ 
}

5) Exit

```

Fig. 2 Join-FI algorithm

The CIS-tree is constructed by considering build-once-mine-many (BOMM) strategy. The join_FI algorithm constructs CIS-tree by considering the K input. The same CIS-tree can be reused again for other K values less than the initial K set by the user. The structure of the CIS-tree remains the same for a data set used for mining process later on, because the frequent 2-itemsets involved in CIS-tree construction are independent of the support threshold parameter. Thus mining process can be performed over the existing CIS-tree without constructing CIS-tree from scratch for different K values on a same data set. Therefore, top- K Miner can be adopted to utilize BOMM approach.

The join-FI algorithm works with four parameters. The first parameter T points to root node of every sub-CIS-tree. The second parameter I_{max} is the least support item to be searched in the *Head-Itemset* field of every node in CIS-tree. The I_{min} is a third parameter. The I_{min} is least support item in the *Head-Itemset* field of every new node in CIS-tree. The join-FI creates new nodes for the given frequent 2-itemset in CIS-tree. The *Head-Itemset* field of the new node is $\text{parent node}.\text{Head-Itemset} \cup I_{min}$. The fourth parameter C is the set of candidate itemsets. This set is updated with the *Head-Itemset* field of every new node as candidate itemset. The join_FI recursively searches for the nodes having I_{max} as the least support item in *Head-Itemset* field of every node in a given CIS-tree. The join-FI procedure detects all such nodes recursively as shown in the step 4 in Fig. 2. A new node is appended to the found nodes. The support of *Head-Itemset* field of the new node is determined. The new node's only *Head-Itemset* field and its support are copied as candidate itemset to C . The biggest gain is achieved while searching for the destination-parent node because of using step 2 in join-FI procedure. If the least support of an item in *Head-Itemset* field of the current traverse node is less than the support of the I_{max} , it means there is no chance that the given sub-CIS-tree will contain a node having I_{max} as least support item. Hence the entire given sub-CIS-tree is skipped as shown in step 2 of the Fig. 2. In step 3 if the least support of an item in the *Head-Itemset* field of the current traversed node is greater than I_{max} , it is a clear indication that the given sub-CIS-tree will contain a node having I_{max} as the least support item. Hence

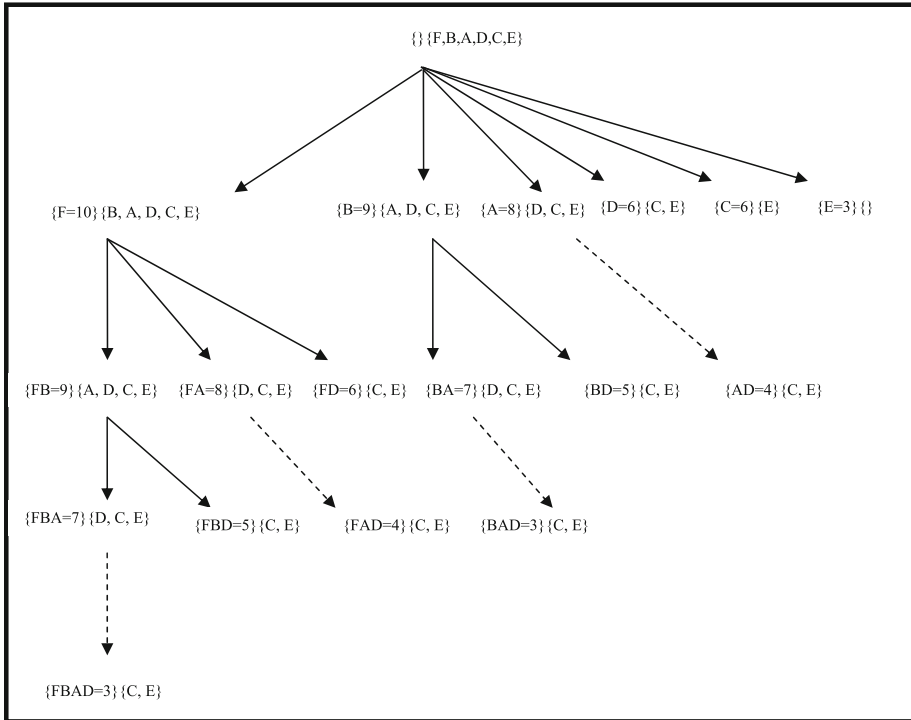


Fig. 3 Candidate itemsets production in CIS-tree

the join_FI is recursively called again with the given sub-CIS-tree as shown in step 3 of the Fig. 2.

Theorem 1 *join-FI algorithm returns at the most $2^{|X|}$ candidate itemsets for every frequent 2-itemsets in a data set D containing n total number of items, where X is the set of items arranged in descending support order.*

Proof let T be a CIS-tree and $R \in T$ be a root node of the CIS-tree where $R.Head-Itemset = \emptyset, R.Tail-Items = \{i_1, i_2, i_3, \dots, i_K\}$ and all items in $R.Tail-Items$ are in descending support order. Any frequent 2-itemset f contains an item $i_{max} \in f$ having maximum support than the next item $i_{min} \in f$ of minimum support. Both the itemsets belong to $R.Tail-Items$, i.e., $i_{max} \in R.Tail-Items$ and $i_{min} \in R.Tail-Items$. Let $X = \{i_1, i_2, i_3, \dots, i_j\}$ is a set of 1-itemset such that $X \subset R.Tail-Items, support(i_m) > support(i_{max}), \forall m = 1, 2, 3, \dots, j$, and $support(i_l) > support(i_{l+1}), \forall l = 1, 2, 3, \dots, m - 1$ where $j < K < n$. For an item I_{max} in every f , the CIS-tree contain up to $2^{|X|}$ nodes or itemsets (also shown in example 3). Every node in $2^{|X|}$ nodes contains I_{max} as the least support item in their $Head-Itemset$ field. The join-FI algorithm finds all such nodes containing I_{max} as least support item in their $Head-Itemset$ field using depth first traversal. The found nodes are extended by the join-FI algorithm with the child nodes, i.e., the $Head-Itemset$ field of the parent nodes and $i_{min} \in R.tail$ are copied to the $Head-Itemset$ field of the new nodes. Since all $2^{|X|}$ nodes are extended with child nodes, hence the join-FI returns $2^{|X|}$ $Head-Itemset$ field of the new nodes as the candidate itemsets. \square

Algorithm Maintain-TopK_List

Input :

Top-K_List: top-K IFIs
C: Candidate Itemsets

Output: *Top-K_List*

```

1) If ((min_support(C) < min_support(Top-K_List))){
           C = C - Cmin_support(Top-K_List)-min_support(C)
           Top-K_List = Top-K_List U C
       }
       return Top-K_List;
}
2) If ((min_support(C) > min_support(Top-K_List))){
           Top-K_List = Top-K_List - Top-K_Listmin_support(C)-min_support(Top-K_List)
           Top-K_List = Top-K_List U C
       }
       return Top-K_List;
}
3) If(((max_support(Top-K_List) < max_support(C)) && (min_support(Top-K_List) < min_support(C))){
           Top-K_List = Top-K_List U C
       }
       return Top-K_List;
}

```

Fig. 4 *Top-K_List* management algorithm

Example 3 Consider a set of frequent 2-itemset in sorted form $F = \{AF, AB, DF, \dots AC, DE, AE\}$ in Table 3. The CIS-tree constructed from F is presented in the Fig. 3. The produced candidate itemset $\{FBAD, FAD, BAD, AD\}$ are presented with the dotted line after positioning the frequent 2-itemset AD in the CIS-tree as shown in Fig. 3.

4.2 Itemset support computation by join-FI

There are two most common data set representation formats, i.e., horizontal data set representation and vertical data set representation. Horizontal data format consists of transactions. Every transaction has associated transaction identifiers (tids) followed by list of items [29]. In vertical format every item in the data set is associated with a corresponding list of tids representing the transactions where it appears [30]. Vertical database representation format was proposed later. The algorithms using this format outperform the algorithms following horizontal approach. The algorithm following vertical approach performs fast frequency computation on sparse data sets but starts to suffer from serious demerits when applied to dense data sets. Dense data sets have high items frequency and large number of patterns resulting gigantic list of tids, require data compression and writing of intermediate results to disk [31]. Therefore, the join-FI procedure in this paper follows another form of vertical data set representation approach called *diffsets* avoiding the drawbacks present in the earlier vertical data representation approach [31]. Diffsets only keeps track of the differences in the tids of all the patterns. Diffsets not only drastically cut down the memory required for large tids by the significant order of the magnitude but also allow extremely faster frequency computation than tids based vertical representation approach [31].

4.3 Top-K_List management procedure

The *Top-K_List* management procedure is used to maintain updated list of the top- K IFIs during mining process as shown in Fig. 4. When top- K Miner concludes the mining process, the *Top-K_List* contains all the top- K IFIs in the given data set. The *Top-K_List* management procedure is passed two parameters, the set of candidate itemsets C and the *Top-K_List*. The candidate itemsets having support less than the minimum support of the IFIs in *Top-K_List*

must be removed from C as shown in step 1 of the Fig. 4. The remaining candidate itemsets in C are the IFIs of the current mining process, copied to the global Top-K_List as shown in step 1 of the Fig. 4. If the support of one or more of the IFIs in the Top-K_List is less than the minimum support of the candidate itemsets in C all such IFIs are removed from the Top-K_List as shown in step 2 of the Fig. 4. In the next step C is copied to Top-K_List to return the actual top-K IFIs in the Top-K_List shown in step 2 of the Fig. 4. If the maximum and minimum support of the candidate itemsets in C is greater than the maximum and minimum support of the IFIs in the Top-K_List respectively, in this case the set of C is copied to Top-K_List and is returned to top-K Miner algorithm as shown in step 3 of the Fig. 4.

Theorem 2 Top-K Miner takes $2^{|I_{Root.Tail_{n-n'}}| - 2^{|X|}}$ time to find top-K IFIs. Where n is the total number of items in a given data set, n' are the items not present in Tail-Items field of the root node of CIS-tree, and X is the set of items in descending support order.

Proof For a frequent 2-itemset the join-FI algorithm perform depth first traversal of the CIS-tree to find all the nodes containing $I_{max} \in f$ as the least support item in their Head-Itemset fields. The CIS-tree is composed of the nodes representing all combinations (appear as Head-Itemset field of the nodes) for the items in the Tail-Items field of the root node excluding those combinations having I_{min} as the least support item. For a given frequent 2-itemset f the CIS-tree contains $2^{|X|}$ nodes having I_{max} as the least support item in Head-Itemset field as proved in theorem 1. The $2^{|X|}$ nodes are extended with a child node containing I_{min} as the least support item in their Head-Itemset field. The number of new nodes formed in a CIS-tree for a frequent 2-itemset f are $2^{|X|}$. The total number of nodes traversed by the join-FI using depth first search approach for a frequent 2-itemset is $2^{|I_{Root.Tail_{n-n'}}| - 2^{|X|}}$ excluding the newly formed $2^{|X|}$ in the worst case. □

5 Comparative evaluation

Top-K Miner is tested on a machine with 1.8GHZ Core 2 Duo processor running Windows 7 operating system. The data sets for the comparative evaluation are downloaded from the FIs mining data set repository [32]. These data sets include real and synthetic data sets. The performance of top-K Miner with other established algorithms is checked on both types of these data sets. The specification of these data sets is given in the Table 5.

The experimental evaluation of the top-K Miner with other established techniques is bifurcated into two phases. In the first phase we compare the results generated by top-K Miner with FP-growth [19] and BOMO [21]. We conducted an experiment to mine different real and synthetic data sets like chess, mushroom, connect, and T40I10D100K. To justify our

Table 5 Data sets for comparative analysis

Database	# Items	Avg length	# Transactions
Chess	75	37	3196
Mushroom	119	23	8124
Connect	129	43	67,557
Retail	16,469	10.3	88,162
T40I10D100K	1000	40	100,000
T10I4D100K	1000	10	100,000

Table 6 Top-10 IFIs on top- K Miner and BOMO (with $N = 3$, $K_{\max} = 3$)

Sr. no.	Data set	Top-10 IFIs by top- K Miner	FP-growth based FIs	BOMO $N = 3$ and $K_{\max} = 3$	IFIs not mined by BOMO
1	Chess	1) 58 = 3195 2) 52 = 3185 3) 58, 52 = 3184 4) 29 = 3181 5) 58, 29 = 3180 6) 40 = 3170, 52, 29 = 3170 7) 58, 52, 29 = 3169 58, 40 = 3169 8) 40 = 3159 9) 58, 52, 40 = 3158 10) 29, 40 = 3155	1) 58 = 3195 2) 52 = 3185 3) 29 = 3181 4) 40 = 3170 5) 52, 58 = 3184 6) 58, 29 = 3180 7) 52, 29 = 3170 8) 58, 40 = 3169 9) 52, 40 = 3159 10) 29, 40 = 3155 11) 52, 29, 58 = 3169 12) 52, 40, 58 = 3158	1) 58 = 3195 2) 52 = 3185 3) 29 = 3181 4) 58, 52 = 3184 5) 58, 29 = 3180 6) 52, 29 = 3170 7) 58, 52, 29 = 3169 8) 58, 52, 40 = 3158 9) 58, 29, 40 = 3154	40 = 3170 58, 40 = 3169 52, 40 = 3159
2	Connect	1) 91 = 67473 2) 109 = 67469 3) 127 = 67465 4) 91, 109 = 67385 5) 91, 127 = 67381 6) 109, 127 = 67377 7) 91, 109, 127 = 67293 8) 75 = 67245 9) 91, 75 = 67161 10) 109, 75 = 67157	1) 91 = 67473 2) 109 = 67469 3) 127 = 67465 4) 75 = 67245 5) 109, 91 = 67385 6) 91, 127 = 67381 7) 109, 127 = 67377 8) 109, 127, 91 = 67293 9) 91, 75 = 67161 10) 109, 75 = 67157	1) 91 = 67473 2) 109 = 67469 3) 127 = 67465 4) 91, 109 = 67385 5) 91, 127 = 67381 6) 109, 127 = 67377 7) 91, 109, 127 = 67293 8) 91, 109, 75 = 67073 9) 91, 127, 75 = 67069	1) 75 = 67245 2) 91, 75 = 67161 3) 109, 75 = 67157

Table 6 continued

Sr. no.	Data set	Top-10 IFIs by top-K Miner	FP-Growth Based FIs	BOMO $N = 3$ and $K_{max} = 3$	IFIs not mined by BOMO
3	Mushroom	1) 85 = 8124 2) 86 = 7924 85, 86 = 7924 3) 34 = 7914 34, 85 = 7914 4) 34, 86 = 7906 34, 85, 86 = 7906 5) 90 = 7488 90, 85 = 7488 6) 90, 34 = 7296 90, 85, 34 = 7296 7) 90, 86 = 7288 90, 34, 86 = 7288 90, 85, 86 = 7288 90, 85, 34, 86 = 7288 8) 36 = 6812 36, 85 = 6812 9) 36, 86 = 6620 36, 85, 86 = 6620 10) 36, 34 = 6602 36, 86, 34 = 6602 36, 85, 34 = 6602 3685 86 34 = 6602	1) 85 = 8124 2) 86 = 7924 3) 34 = 7914 4) 90 = 7488 5) 36 = 6812 6) 8685 = 7924 7) 3485 = 7914 8) 3486 = 7906 9) 9085 = 7488 10) 9034 = 7296 11) 9086 = 7288 12) 3685 = 6812 13) 3686 = 6620 14) 3634 = 6602 15) 348586 = 7906 16) 908534 = 7296 17) 903486 = 7288 18) 908586 = 7288 19) 368586 = 6620 20) 368634 = 6602 21) 368534 = 6602 22) 90853486 = 7288 23) 36858634 = 6602	1) 85 = 8124 2) 86 = 7924 3) 34 = 7914 4) 85, 86 = 7924 5) 85, 34 = 7914 6) 86, 34 = 7906 7) 85, 86, 34 = 7906 8) 34, 85, 90 = 7296 9) 85, 86, 90 = 7288	1) 90 = 7488 2) 85, 90 = 7488 3) 34 = 7914 5) 85, 86, 34, 90 = 7906 6) 86, 34, 90 = 7906 7) 34, 90 = 7296 8) 86, 90 = 7288

Table 6 continued

Sr. no.	Data set	Top-10 IFIs by top- K Miner	FP-growth based FIs	BOMO $N = 3$ and $K_{\max} = 3$	IFIs not mined by BOMO
4	T40I10D100K	1) 368 = 28738 2) 529 = 23384 3) 829 = 23121 4) 510 = 20125 5) 419 = 20216 6) 217 = 19326 7) 489 = 18921 8) 682 = 17427 9) 914 = 17343 10) 692 = 17203	1) 368 = 28738 2) 529 = 23384 3) 829 = 23121 4) 510 = 20125 5) 419 = 20216 6) 217 = 19326 7) 489 = 18921 8) 682 = 17427 9) 914 = 17343 10) 692 = 17203	1) 368 = 28738 2) 529 = 23384 3) 829 = 23121 4) 368, 529 = 7500 5) 368, 829 = 6957 6) 368, 682 = 6130 7) 682, 368, 489 = 2420 8) 368, 529, 692 = 2373 9) 368, 529, 829 = 2288	1) 510 = 20125 2) 419 = 20216 3) 217 = 19326 4) 489 = 18921 5) 682 = 17427 6) 914 = 17343 7) 692 = 17203

experiment we set $K=10$ for top- K Miner and for BOMO $N = 3$ and $K_{\max} = 3$ as shown in Table 6.

The parameters are tuned with the intention to highlight two issues. First, the FP-growth method uses different support threshold parameter to mine top- K IFIs on different data sets. Second, setting of N and K_{\max} for BOMO is simple and similar on all data sets to mine top- K IFIs but have the tendency to miss some of the IFIs. Top- K Miner does not skip the information about patterns available in the data sets as done by topmost frequent pattern mining techniques based on two parameters N , and K_{\max} [21,22]. The top- K Miner mines more and absolute desired patterns than the existing topmost frequent pattern finding techniques. Mostly topmost frequent pattern mining techniques varies N while keep K_{\max} as constant parameter. Whatever the value of N is set by the user, it is only used to find the N topmost patterns for $\forall 1, 2, 3, \dots, k_{\text{MAX}}$ sizes. The parameter N also sets the upper bound and lower bounds in terms of support for $\forall 1, 2, 3, \dots, k_{\text{MAX}}$ sizes patterns. The IFIs with greater support are dropped because of the miss tuning N value as shown in the Table 6. The same or greater support patterns having item sets size greater than K_{\max} are also missed because of K_{\max} . Now from the user's perspective, few of the IFIs will definitely miss whatever the value of N or K_{\max} is adjusted by the user. While on the other hand top- K Miner will find every pattern of greater support and of arbitrary size until K number of patterns are discovered. Our top- K IFIs mining technique is superior enough in the context that it is free from K_{\max} parameter setting. Top- K IFIs mining approach also does not need extra care to set top- K parameter, as it is demanded for adjusting N and K_{\max} .

The results generated by the top- K Miner are also compared with that of the FP-growth method [19]. The strength of the top- K Miner method is that it requires the only a value of the parameter K to mine top- K IFIs on different data set. The FP-growth method requires different support threshold parameter tuning on different data sets to get the same number of FIs as shown in Table 6. The FP-growth technique requires support threshold parameter tuning before FIs discovery. Setting a support threshold to get particular number of FIs is a challenging task for user [20]. Setting support threshold greatly depends on the characteristics (number of items, frequency of items, number of transactions, and average length of the transaction) of the data set. The top-10 IFIs generated by the top- K Miner are equivalent to the number of FIs produced by the FP-growth on connect and T40I10D100K data sets. The FP-growth method produces large number of FIs than the IFIs on chess and mushroom data sets as shown in Table 6.

In the second phase of evaluation top- K Miner is analyzed for time constraint with the established benchmarks, i.e., BOMO [21] and, FP-growth [19]. BOMO is basically the same class of algorithm constructed to mine topmost- N interesting itemsets with K_{\max} itemsets constraint setting. The second technique also takes two parameters, i.e., supports threshold and maximum size of patterns to find frequent itemsets. While experimenting on each data set, the ideal support threshold value is provided to FP-growth [19]. Top- K Miner produces same result by tuning the number of patterns required by the user without using support threshold value. We carried out performance experiments on two types of data sets, i.e., real and synthetic data sets. Real data sets include chess, mushroom, connect and retail. Synthetic data sets include T40I10D100K and T10I4D100K. Figures 5 and 6 demonstrate the performance results of the top- K Miner with BOMO and FP-growth on both types of data sets. The x-axis represents the top- K IFIs to be mined till K_{\max} size and y-axis represents time taken by algorithm.

Real data sets are dense in nature and can produce longer patterns if lower support threshold is provided. As illustrated in the Fig. 5a–d on real data sets which are smaller in terms of the number of items like chess, mushroom, and connect, the top- K Miner execution time

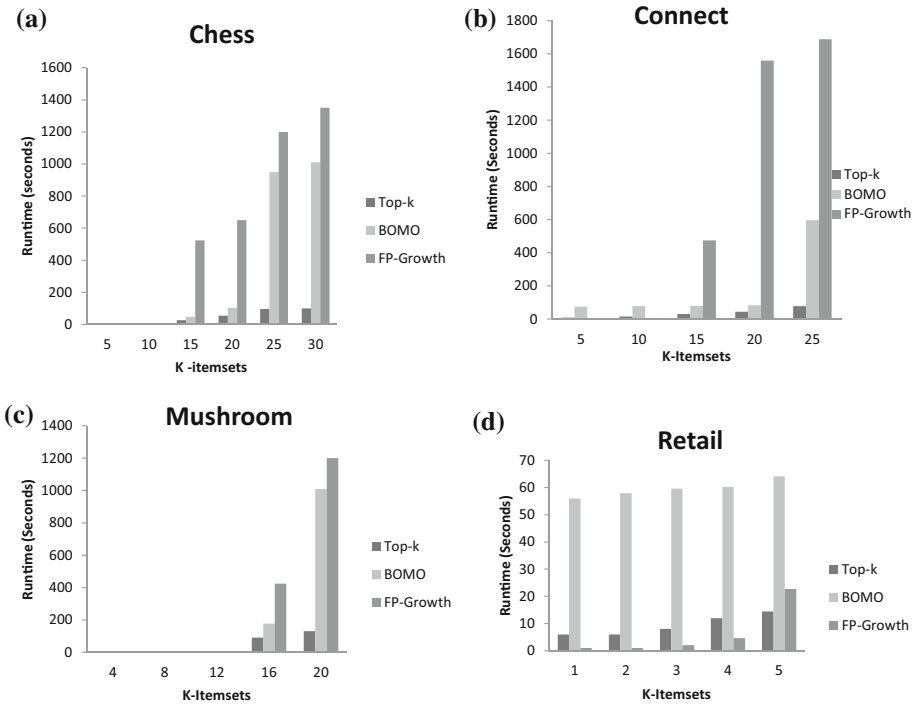


Fig. 5 Top- K Miner performance trends on real data sets

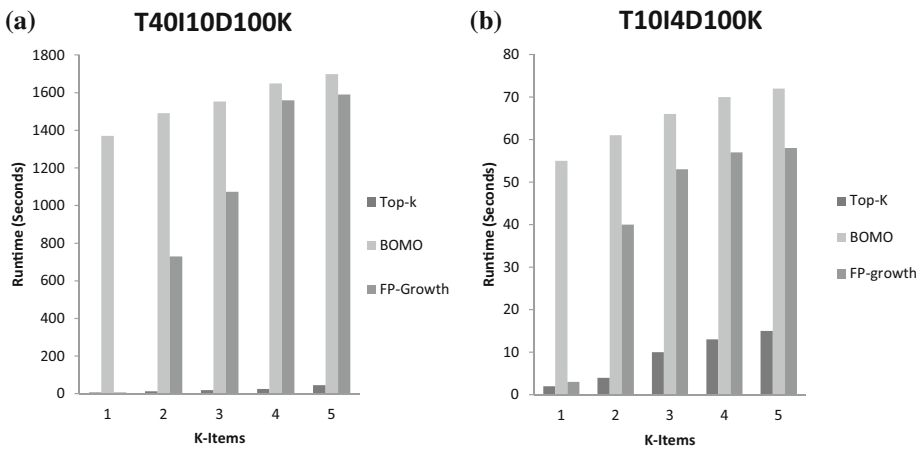


Fig. 6 Top- K Miner performance trends on synthetic data sets

is tantamount to BOMO and FP-growth for smaller K values. On chess, mushroom, and connect data sets, when the pattern length is increased, the performance of BOMO and FP-growth is significantly degraded because of the larger FP-tree size and subsequently mining process on the FP-tree. Top- K Miner is least affected by the increase in pattern length size as shown in Fig. 5a–c.

The top- K Miner is less efficient than FP-growth for small K values on retail data set as shown in Fig. 5d. The retail is a largest real data set in terms of the number items as shown in Table 5. In case of retail data set, the dominating factor for the top- K Miner is sorting of $n(n-1)/2$ number of items. The time required for sorting $n(n-1)/2$ number of items remains constant for mining any number of top- K IFIs over the retail data set using top- K Miner. For the FP-growth the dominating factor is the size of FP-tree (depends on the data set having transaction containing unique set of items) and the subsequent FP-tree mining process. The FP-tree mining process checks every combination of itemsets to be frequent or not and accept 1-itemsets (are Checked before FP-tree construction). The time required to mine FP-tree depends on the number of FIs to be mined using FP-growth algorithm. On a retail data set the time required by the top- K Miner for sorting $n(n-1)/2$ is greater than the time taken by the FP-growth algorithm for mining small number of top- K patterns. The FP-growth performance starts to degrade as compared to top- K Miner when K is increased.

The data set T40I10D100K and T10I4D100K are synthetic data sets shown in Fig. 6a, b. The length of the patterns is short in these data sets but have 1000 number of items. Top- K Miner takes extra amount of time in sorting large number of items as in the case of synthetic data set. Still the total execution time (sorting time + pattern mining time) for the top- K Miner is equivalent to FP-growth during the first few IFIs calculation as shown in the Fig. 6a, b. Furthermore BOMO and FP-growth start to suffer from drastic increase in execution time when IFI patterns of higher length are discovered.

There are few main reasons why top- K Miner showed significant performance over BOMO and FP-growth algorithms. These algorithms build complete FP-tree in memory for the whole data set. The BOMO and FP-growth algorithms scan the given data set twice. Top- K Miner performs single scan of the data set. The size of the CIS-tree is practically limited to the number of items selected for mining out of the total number of items in the data set.

BOMO and FP-growth algorithms perform mining of the frequent patterns from the entire FP-tree. The performance of the FP-growth and BOMO algorithms degrade significantly when provided small value of the support threshold parameter or mining large number of frequent pattern over a dense data set or mining large number of patterns over a data set with large number of items.

6 Conclusion

The top- K Miner is proposed in this paper to find top- K IFIs. To the best of our knowledge top- K Miner is the first approach which finds all the absolute desired required number of patterns and does not miss any pattern of highest support. Top- K Miner needs at the most only one parameter, whereas other techniques which also find topmost frequent pattern demand at least two parameters. Our proposed technique has shown improved performance than its counter parts because it follows build-and-mine (BM) strategy for CIS-tree construction, whereas other methods perform mining after building in-memory data structure for all the items and hence take more time to find required IFIs. There is ample room for the future work in the proposed technique. In future we plan to adapt this work to solve other data mining tasks like text mining, subspace clustering, and stream mining.

References

1. Agrawal R, Imielinski T, Swami A (1993) Mining association rules between sets of items in large databases. In: Proceedings of the 1993 ACM-SIGMOD international conference on management of data (SIGMOD'93). Washington, DC, pp 207–216

2. Grahne G, Zhu J (2003) High performance mining of maximal frequent itemsets. In: Proceeding of the 2003 SIAM international workshop on high performance data mining, pp 135–143
3. Lee W, Stolfo SJ, Mok KW (2000) Adaptive intrusion detection: a data mining approach. *Artif Intell Rev* 14(6):533–567
4. Pei J, Han J, Mortazavi-Asl B, Zhu H (2000) Mining access patterns efficiently from web logs. In: Proceeding of the 2000 Pacific-Asia conference on knowledge discovery and data mining. Kyoto, Japan, pp 396–407
5. Holt JD, Chung SM (1999) Efficient mining of association rules in text databases. In: Proceeding of the 1999 international conference on Information and knowledge management. Kansas City, Missouri, pp 234–242
6. Klemettinen M (1999) A knowledge discovery methodology for telecommunication network alarm databases. Ph.D. thesis, University of Helsinki
7. Satou K, Shibayama G, Ono T, Yamamura Y, Furuichi E, Kuhara S, Takagi T (1997) Finding associations rules on heterogeneous genome data. In: Proceeding of the 1997 Pacific symposium on biocomputing (PSB'97). Hawaii, pp 397–408
8. Bayardo RJ (1998) Efficiently mining long patterns from databases. In: Proceeding of the 1998 ACM-SIGMOD international conference on management of data (SIGMOD'98). Seattle, WA, pp 85–93
9. Burdick D, Calimlim M, Flannick J, Gehrke J, Yiu T (2005) MAFLIA: a maximal frequent itemset algorithm. *IEEE Trans Knowl Data Eng* 17(11):1490–1504
10. Gouda K, Zaki MJ (2005) GenMax: an efficient algorithm for mining maximal frequent itemsets. *Data Min Knowl Discov* 11(3):1–20
11. Pasquier N, Bastide Y, Taouil R, Lakhal L (1999) Discovering frequent closed itemsets for association rules. In: Proceeding of the 7th international conference on database theory (ICDT'99). Jerusalem, Israel, pp 398–416
12. Pei J, Han J, Mao R (2000) CLOSET: an efficient algorithm for mining frequent closed itemsets. In: Proceeding of the 2000 ACM-SIGMOD international workshop data mining and knowledge discovery (DMKD'00). Dallas, TX, pp 11–20
13. Zaki MJ, Hsiao CJ (2002) CHARM: an efficient algorithm for closed itemset mining. In: Proceeding of the 2002 SIAM international conference on data mining (SDM'02). Arlington, VA, pp 457–473
14. Borgelt C, Yang X, Nogales-Cadenas R, Carmona-Saez P, Pascual-Montano A (2011) Finding closed frequent item sets by intersecting transactions. In: Proceedings of the 2011 international conference on extending database technology (EDBT-11). Sweden, Uppsala, pp 367–376
15. Hu T, Sung SY, Xiong H, Fu Q (2008) Discovery of maximum length frequent itemsets. *Inf Sci Int J* 178(1):69–87
16. Zhu F, Yan X, Han J, Yu PS, Cheng H (2007) Mining colossal frequent patterns by core pattern fusion. In: Proceeding of the 2007 international conference on data engineering (ICDE'07). Istanbul, Turkey, pp 706–715
17. Dabbiru M, Shashi M (2010) An efficient approach to colossal pattern mining. *Int J Comput Sci Netw Secur (IJCSNS)* 10(1):304–312
18. Sohrabi MK, Barforoush AA (2012) Efficient colossal pattern mining in high dimensional datasets. *Knowl Based Syst* 33:41–52
19. Han J, Pei J, Yin Y (2000) Mining frequent patterns without candidate generation. In: Proceeding of the 2000 ACM-SIGMOD international conference on management of data (SIGMOD'00). Dallas, TX, pp 1–12
20. Han J, Cheng H, Xin D, Yan (2007) Frequent pattern mining—current status and future directions. *Data Min Knowl Discov* 15(1):55–86
21. Cheung YL, Fu AWC (2004) Mining frequent itemsets without support threshold: with and without item constraints. *IEEE Trans Knowl Data Eng* 16(9):1052–1069
22. Fu AWC, Kwong RWW, Tang J (2000) Mining N -most interesting itemsets. In: Proceedings of the 2000 international symposium on methodologies for intelligent systems, pp 59–67
23. Ngan SC, Lam T, Wong RCW, Fu AWC (2005) Mining N -most interesting itemsets without support threshold by the COFI-tree. *Int J Bus Intell Data Min* 1(1):88–106
24. El-Hajj M, Zaïane OR (2003) COFI-tree mining: a new approach to pattern growth with reduced candidacy generation. In: Workshop on frequent itemset mining implementations (FIMI 2003) in conjunction with IEEE-ICDM
25. Salam A, Khayal M (2011) Mining top- k frequent patterns without minimum support threshold. *Knowl Inf Syst* 30(1):112–142
26. Li Y, Lin Q, Li R, Duan D (2010) TGP: mining top- K frequent closed graph pattern without minimum support. In: Proceeding of the 2010 international conference on advanced data mining and applications (ADMA '10), pp 537–548

27. Xie Y, Yu PS (2010) Max-Clique: a top-down graph-based approach to frequent pattern mining. In: Proceeding of the 2010 IEEE international conference on data mining (ICDM '10). pp 1139–1144
28. Okubo Y, Haraguchi M (2012) Finding top- N colossal patterns based on clique search with dynamic update of graph. In: Proceeding of the 2012 international conference on formal concept analysis (ICFCA'12). Springer, pp 244–259
29. Agrawal R, Mannila H, Srikant R, Toivonen H, Verkamo A (1996) Fast discovery of association rules. In: Fayyad UM, Piatetsky G, Smyth P, Uthurusamy R (eds) Advances in KDD. MIT press
30. Holsheimer M, Kersten M, Mannila H, Toivonen H (1995) A perspective on database and data mining. In: Proceeding of the 1995 international conference on knowledge discovery and data mining (KDD'95). pp 150–155
31. Zaki MJ, Gouda K (2003) Fast vertical mining using diffsets. In: Proceedings of the 2003 ACM-SIGKDD international conference on knowledge discovery and data mining (SIGKDD'03). Washington, pp 326–335
32. Frequent itemset mining implementations repository. <http://fimi.cs.helsinki.fi/>
33. Shen L, Shen H, Pritchard P, Topor R (1998) Finding the N largest itemsets. In: Proceedings of international conference on data mining. pp 211–222
34. Quang TM, Oyanagi S, Yamazaki K (2006) ExMiner: an efficient algorithm for mining top- K frequent patterns, ADMA 2006, LNAI 4093. pp 436–447
35. Wang J, Han J (2005) TFP: an efficient algorithm for mining top- K frequent closed itemsets. IEEE Trans Knowl Data Eng 17(5):652–664
36. Hirate Y, Iwahashi E, Yamana H (2004) TF2P-growth: an efficient algorithm for mining frequent patterns without any thresholds. In: Proceedings of ICDM



Saif-Ur-Rehman is a Ph.D. student in Institute of Information Technology (IIT), Kohat University of Science and Technology, Kohat, Pakistan. Currently, He is working as lecturer in University of Peshawar, Peshawar, Pakistan. His main research interest includes data mining.



Jawad Ashraf has earned his Ph.D. in Computer Science from Leicester University, UK, in 2012. His main research interest includes scheduling scientific workflows in grid computing and data mining. Currently, he is working as assistant professor in Institute of Information Technology (IIT), Kohat University of Science and Technology, Kohat, Pakistan.



Asad Habib has received Doctor of Engineering degree from Graduate School of Information Science, NAIST (Nara Institute of Science and Technology), Japan. His research interests include Big Data, Natural Language Engineering and HCI. Currently, he is working as assistant professor in Institute of Information Technology (IIT), Kohat University of Science and Technology, Kohat, Pakistan.



Abdus Salam has a distinguished career in the field of computer science spanning over 27 years. This includes 14 years of professional experience and 13 years of teaching experience. He earned his M.Sc. in Computer Science from Quaid-e-Azam University in 1987. He also earned his Ph.D. in Computer Science from International Islamic University, Islamabad, in 2011. Presently, he is working as Head of Department of Computing & Technology, Abasyn University, Peshawar, Pakistan.