Frequent Pattern Mining under Multiple Support Thresholds

SADEQ DARRAB, BELGIN ERGENÇ
Computer Engineering
Izmir Institute of Technology
Urla/İzmir TURKEY
sadeqaldrab@gmail.com, belginergenc@iyte.edu.tr
dworld.iyte.edu.tr

Abstract: - Traditional methods use a single minimum support threshold to find out the complete set of frequent patterns. However, in real word applications, using single minimum item support threshold is not adequate since it does not reflect the nature of each item. If single minimum support threshold is set too low, a huge amount of patterns are generated including uninteresting patterns. On the other hand if it is set too high, many of interesting patterns (called rare items) may be lost. Recently, several methods have been studied to tackle the rare item problem by avoiding using single minimum item support threshold. The nature of each item is considered where different items are specified with different Minimum Item Support thresholds (MIS) instead of using single support threshold. By this, the complete set of frequent patterns is generated without creating uninteresting patterns and losing substantial patterns. In this paper, we propose an efficient method, Multiple Item Support Frequent Pattern growth algorithm, MISFP-growth, to mine the complete set of frequent patterns with multiple item support thresholds. In this method, Multiple Item Support Frequent Pattern Tree, MISFP-Tree, is constructed to store all crucial information to mine frequent patterns. Since the construction of the MISFP-Tree is done with respect to minimum of MIS; post pruning and reconstruction phases are not required. In order to show the efficiency of the proposed method, it is compared with a recent tree-based algorithm, CFP-growth++ and various experiments are conducted on both real and synthetic datasets. Experimental results reveal that MISFP-growth outperforms in terms of execution time and memory space while we vary MIS values of items.

Key-Words: - Association rule mining, Frequent patterns, Rare itemsets, Multiple support thresholds

Acknowledgements: This work is partially supported by the Scientific and Technological Research Council of Turkey (TUBITAK) under ARDEB 3501 Project No: 114E779

1 Introduction

Data mining is a nontrivial process of discovering interesting knowledge from large databases. Association Rule Mining (ARM) is an important pattern analysis technique of data mining that focuses on finding out the sequences of actions or events known as itemsets (patterns) that are items that occur together. It has drawn attention since it was first proposed in [1]. This is due to its applicability in various domains e.g. medical field, elections, telecommunication firms, etc. The overall goal of ARM is to discover all interesting rules from a dataset that have the form: $X \rightarrow Y \mid X \cap Y = \emptyset$ where X and Y are the set of items in the dataset. An interesting rule should satisfy two statistical measures known as minimum support threshold denoted as *minsup* and minimum confidence threshold denoted as *minconf* where *minsup* refers to the percentage of transactions in the dataset that contain $X \cup Y$ whereas *minconf* denotes to the conditional probability of finding $X \cup Y$ given the transactions that contain X.

Association rules can be found in two essential steps as follows;

1. Finding all frequent patterns that exceed a given *minsup*.
2. Generating association rules from frequent patterns that are found in step 1 where frequent pattern satisfies both of *minsup* and *minconf*.

Since the first step is expensive, almost all research on ARM focuses on generating the frequent patterns. Once the frequent patterns are generated, generating association rules is straightforward since confidence does not possess closure property as support.

As mentioned before, discovering frequent patterns is an essential step of ARM that aims to extract a set of items that frequently co-occur (itemset) in a database. Frequent itemsets (patterns) should satisfy user-specified minimum support threshold, *minsup*. Numerous methods have been

proposed in order to find out frequent patterns with a single *minsup* [1, 2, 3, 4, 5, 6]. Using the single *minsup* considerably reduces the search space and computation by avoiding a huge amount of infrequent itemsets from being handled. However, mining frequent patterns with the single *minsup* faces two problems: 1) extremely large amounts of meaningless patterns are generated if the *minsup* is set too low, 2) useful patterns may be lost when the *minsup* is set too high. This problem is called rare item problem [7]. Recently, several studies have been performed to mine both of frequent and interesting rare itemsets. Instead of using a single *minsup* threshold for all items, separate *minsup* threshold is assigned to each item based on the characteristic of the item/itemset.

Since MSapriori [7], many methods have been proposed to reduce search space and execution time while generating frequent patterns under multiple item support (MIS) [8, 9, 10, 11]. These methods discover all meaningful rules with MIS, including rarely occurring ones by applying different MIS to each item. They can be classified into two types: 1) Apriori-like methods [7, 8, 9] and 2) FP-growth-like methods [10, 11].

In Apriori-like methods, the databases have to be scanned many times to create all candidates items since they are based on Apriori algorithm. They consume an enormous amount of runtime, especially when databases contain too many transactions since they have to perform several database scans with their candidate generation-and-test approach. In order to overcome this weakness FP-growth-like methods were proposed. These methods require database scan at most twice as they use FP-Tree to hold all necessary information that is needed in mining process. They construct MIS-Tree with a single scan and then the tree is pruned and re-constructed in order to eliminate redundant nodes.

However, these methods are still far from being efficient since they require 1) huge amount of memory due to the management of irrelevant nodes, and 2) high execution time for post prune-and-reconstruct phase.

In this paper, we propose an efficient algorithm called Multiple Item Support Frequent Pattern growth, MISFP-growth, which is an extended version of FP-growth [3]. MISFP-growth is designed to mine frequent patterns under MIS over large databases. MISFP-growth use Multiple Item Support Frequent Pattern tree, MISFP-Tree, which is based on FP-Tree, to hold all necessary information that is used to discover the complete set of frequent patterns with MIS. At the same time, a frequent item header table, MIN-MIS-Frequent table, is generated with all items that have support no less than the minimum of minimum item support (MIS) thresholds (MIN-MIS). Since MISFP-Tree is constructed based on MIN-MIS, it does not need any pruning and reconstruction. MISFP-growth, extracts frequent patterns from MISFP-Tree. The experimental results on both sparse and dense datasets show the superiority of MISFP-growth in comparison to a similar recent algorithm in terms of runtime and memory while varying item support thresholds.

The remaining of this paper is organized as follows. In section 2, we give preliminaries of the challenge. In section 3, the proposed MISFP-growth algorithm is presented. Experimental results are shown in section 4 while the related work is discussed in section 5. Conclusion is given in section 6.

2 Preliminaries

In this section, we introduce the basic terminology related to frequent pattern mining under both of single and multiple thresholds.

Let $I = \{i_1, i_2, \dots, i_m\}$ represents the set of m distinct items, and $DB = \{T_1, T_2, \dots, T_n\}$ be a transaction database where Ti ($i \in [1 \dots n]$) is a transaction, which contains a set of items in I. Each transaction is associated with an identifier, called *TID*. A transaction can be defined as $T_i = (TID_i, X)$, which is a tuple has number *TID* and contains an itemset *{X}*. The itemset $X = \{x_1, x_2 \dots x_k\}$ is a set of k items in T. Thus, the itemset *{X}* have at least one item and at most all items in specific transaction. The itemset that contains *K* items is called *K*-itemset. If support of the itemset is greater than or equal to *minsup***,** then it is a frequent pattern. The support of the itemset *X*, denoted as sup*(X)*, is the number of transactions that contain *{X}* in *DB* as follows:

$$sup(X, DB) := \{TID | (TID, I) \in DB, X \subseteq I\}$$

Definition 1 (Frequent Pattern with Single Threshold): Let *DB* be a transaction database over a set of items *I*, and *minsup* is minimum support threshold given by the user. The set of frequent patterns in *DB*, which exceeds *minsup* is defined as follows:

$$F(DB, minsup) = \{X \subseteq I, \sup(X, DB)/|DB| \geq minsup\},$$

where *F* represents all the frequent itemsets in *DB* and $|DB|$ is the number of transactions in *DB*.

Example. Suppose there are two itemsets: *K = {x, y, z} and Z = {n, m}* with actual support = 70%, 40%, respectively in a given database and the *minsup* is set at *50%.*

According to definition 1, the K itemset is frequent as its support exceeds *minsup* = 50%, whereas Z is

infrequent itemset as its support does not satisfy *minsup*.

Definition 2 Multiple Item Support (*MIS*): Let I be a set of I items $I = \{i_1, \ldots, i_n\}$, an itemset $X = \{i_1, \ldots, i_k\}$, the minimum item support(*MIS*) of itemset X is defined as follows. $MIS(X) = \text{MIN}\{\text{MIS}(i_1), \text{MIS}(i_2), \ldots, \text{MIS}(i_k)\}$.

Example. Assume that an itemset $K = \{x, y, z\}$ has an actual support = 8% in a given database. Suppose that the *MIS* and the actual support of items are given as follows:

$MIS(x) = 5\%, MIS(y) = 10\%$ *and* $MIS\ (z) = 15\%,$
$sup(x) = 10\%, sup(y) = 9\%$ *and* $sup\ (z) = 11\%.$

Then the MIS of the itemset K can be defined as follows:

$MIS(K) = MIN\ \{MIS(x) = 5\%, MIS(y) = 10\%,\ MIS\ (z) = 15\%\} = 5\%.$ Thus, the itemset K is frequent with support = 8%, which exceeds *MIS of K = 5%.*

3 MISFP-growth algorithm

The proposed method, MISFP-growth, is extended version of FP-growth [3]. Its approach is similar to FP-growth with slight differences. Main differences between MISFP-growth and FP-growth are as follows;

1. FP-growth is used to create frequent patterns based on single threshold whereas MISFP-growth is used to mine frequent patterns with multiple item support thresholds.
2. Items in FP-growth method are arranged in descending order in terms of their actual support whereas in MISFP-growth items are sorted in descending order in terms of their support threshold values.

MISFP-growth reduces search space based on the minimum of minimum item support threshold, MIN-MIS. This idea plays a big role to reduce search space since it is used to discard unpromising items that play no role in creating frequent patterns at high order.

Discarding property (MIN-MIS): Any item that has support less than MIN-MIS is discarded and is not used in building up MISFP-Tree.

We build our tree by only those promising items that have support more or equal to MIN-MIS. MISFP-growth utilizes MISFP-tree, an extended prefix-tree, which compresses all transactions of database in horizontal data format in memory. This enables MISFP-growth to search for the complete set of frequent patterns without the requirement of generating a large number of candidate itemsets. MISFP-growth requires the following essential steps:

1. Scan database DB once to find out the actual support of each item.
2. Find the lowest minimum support threshold (MIN-MIS) among all items in database.
3. Scan DB once again to collect items that satisfy MIN-MIS in each transaction, sort them in the descending order of their predefined MIS and insert these items into the MISFP-Tree. If the appropriate node of an item exists, its count is increased by one. Otherwise, a new node is inserted in the MISFP-Tree.
4. Create MIN-MIS-frequent header table of MISFP-Tree, which is used to hold items with support no less than MIN-MIS in descending order of MIS values of items. It consists of item-name, MIS of item and the head of node-link that point to item's occurrences in the MISFP-Tree. Nodes that have the same item-name are linked in sequence. Such node-links simplify tree traversal.
5. Build the conditional pattern base and the conditional MISFP-Tree of each suffix item whose support is greater or equal to its predefined MIS. These two data structures represent the knowledge extracted from MISFP-Tree.

As a summary, MISFP-growth algorithm involves two main steps to create the whole set of frequent patterns and rare patterns with MIS as follows; construction of MISFP-Tree (steps 1-4) and mining frequent patterns from MISFP-Tree (step 5).

3.1 Construction of MISFP-Tree

A multiple support frequent pattern tree (MISFP-Tree) is a tree structure that can be defined as follows.

- It composes of a root named as null, a set of item prefix sub trees as the children of the root, and a MIN-MIS-frequent item header table which contains all items that have support more than MIN-MIS.
- Each node in the item prefix subtree composes of three fields: item-name, count and node link, where item-name represents the item that this node presents, count records the number of transactions represented by the portion of the branch reaching to this node, and node-link links to the next node in the MISFP-Tree carrying the same item-name, or null if there is none.
- Each entry in the MIN-MIS-frequent item header table consists of three fields: item-name, item's minimum support thresholds and head of node-link which points to the first node in the MISFP-Tree carrying the item-name.

- All the items in the table are sorted in descending order in terms of their minimum item support thresholds values.
- The MISFP-Tree and header table consist only of items that have support no less than MIN-MIS.

Table 1 (a). Transaction database

TID	Items	Items have support no less than MIN-MIS
1	d, c, a, f	a, c, f
2	g, c, a, f, e	a, c, f, g
3	b, a, c , f, h	a, b, c, f
4	g, b, f	b, f, g
5	b, c	b, c

Table 1 (b). MIS and actual support of items

Item	a	b	c	d	e	f	g	h
MIS	4	4	4	3	3	2	2	2
Sup	3	3	4	1	1	2	2	1

Example. Given a transaction database DB as shown in Table 1(a) and the multiple item supports of items in Table 1(b), construct the MISFP-Tree with MIS in DB.

To build MISFP-Tree from the data in Table 1 (a) with the multiple predefined minimum support values in Table 1(b), the process works as follows.

1. Scan database once to find the support of items in the database DB. The Table 1 (b) shows the items, MIS of items and their actual support in the consecutive rows of the table.
2. Find out the least minimum support threshold among all minimum item support thresholds of items: MIN-MIS = min {MIS(a), MIS(b), …, MIS(h)} = 2.
3. Compare the actual support of items with MIN-MIS value (i.e, 2) and each item has support less than 2 is discarded since they play no role in generation of frequent patterns. Hence, items {h, e, d} are discarded. The remaining items that have support no less than 2 are arranged according to their minimum item support thresholds in descending order as shown in the right column in Table 1 (a).

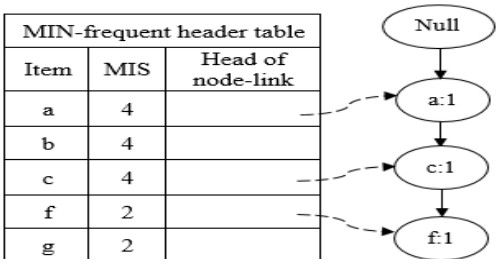


Fig.1 (a). After inserting the first transaction

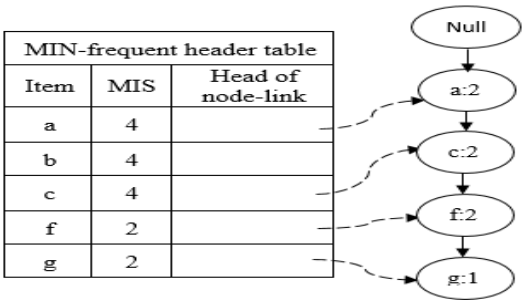


Fig.1 (b). After inserting the second transaction

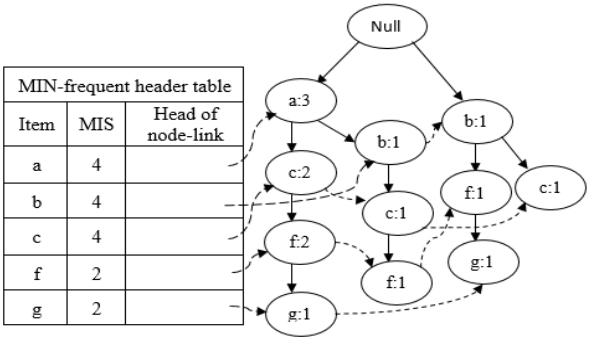


Fig.1 (c). MISFP-Tree with all transactions

4. Scan database once again to construct MISFP-Tree. We use items in the right column of the Table 1 a) to build our tree. The process of inserting transactions into the tree works as follows.
   - 4.1 The root of MISFP-Tree is created and labeled as "null".
   - 4.2 For the first transaction {a, c, f}; the first branch of MSFP-Tree is created as shown in Fig.1 (a). Notice that all items in the transaction are inserted into the tree in descending order in term of their minimum item support thresholds.
   - 4.3 For the second transaction {a, c, f, g}; since it shares the prefix {a, c, f} with the first transactions, the count of each node along the prefix is increased by 1, a new node (g: 1) is generated and linked as child of (f:2) as shown in Fig.1 (b).
   - 4.4 By repeating the steps 4.2 and 4.3 consecutive transactions are added to the

tree. Fig.1 (c) shows the complete MISFP-Tree after we insert all transactions.

3.2 Mining frequent patterns from MISFP-Tree

MISFP-growth is similar to CFP-growth++ [11] in extracting the complete set of frequent patterns from MISFP-Tree with the following difference.

Since we do not discard those items whose support greater than MIN-MIS and less than their predefined MIS, those items can be used to generate frequent patterns with others but from them no frequent pattern can be generated. Therefore, we avoid generating conditional pattern base and conditional MISFP-tree for these items by checking the item's support of each item against its predefined MIS.

In order to mine frequent patterns from MISFP-Tree shown in Fig.1 (c), we start to mine frequent patterns that can be created from item {g} since it has the least minimum threshold among all items in the MIN-MIS-frequent header table. Following the node-link of item {g}, there are two branches that contain that item {g}; { a:3, c: 2, f:2, g:1} and {b:1, f:1, g:1}. Considering the item "g" as suffix item, all frequent patterns are generated based on item "g" with support no less than MIS of suffix item, here item {g} (i.e, 2). Since items are ordered in descending order in terms of their minimum item support thresholds, the item {g} has the least minimum threshold among all items on these paths. Therefore, all patterns that have support less than 2 cannot be frequent.

Hence, the conditional pattern base of item {g} is { a:1, c:1, f:1} and {b:1, f:1}. Notice that since the counter value of {g} in each path is 1, the counter of the nodes in these two branches is set to 1. After the conditional pattern base of item {g} are identified, the g's conditional MISFP-Tree is created by adding the counts along the link and searching for patterns that exceed the minimum support threshold value of item {g}.

In the conditional MISFP-Tree for suffix item {g}, since only the item {f }has support no less than the minimum support threshold of item {g} (i.e., 2), the only conditional frequent pattern {(fg:2)} is generated. Since the support count of the remaining items {a, b, c} is 1, no frequent patterns can be created from them. Thus, we create g's conditional frequent pattern (fg:2). By repeating the same process for the remaining items in MIN-MIS frequent header table, we find out the whole set of frequent patterns as shown in Table 2.

Table 2. The complete set of frequent patterns

Suffix item	Minsup	Conditional pattern base	Conditional MISFP-Tree	Frequent patterns
g	2	{ a, c , f :1}, { b , f : 1}	{f:2}	fg:2
f	2	{a, c :2}, {a, b, c :1 }, {b :1 }	{a:2}, {c:2}, {b:2}, {ac:3}	af:2, cf:2, bf:2, acf:2
c	4	{a:2}, {a, h:1}, {b}	-	-
b	4	{a:1}	-	-
a	4	-	-	-

4 Performance Evaluation

In this section, the proposed method, MISFP-growth, is compared with the recent tree based method, CFP-growth++ [11], to discover frequent patterns under multiple support thresholds. To verify the effectiveness and efficiency of the proposed method, several experiments are conducted using five datasets with different characteristics. In these experiments, we measure the performance of two methods in terms of execution time and memory space.

4.1 Experimental environment and datasets

We conduct two experiments using different type of datasets to measure the performance and efficiency of the proposed method, MISFP-growth. All experiments are executed on an Intl(R) core i7 - 5500u CPU@ 3.40 GHz with 8GB main memory, running on Microsoft Windows 10 operating system. All the programs are implemented with C#.

Table 3. Characteristics of datasets

Datasets	Density (%)	Size (MB)	# of Distinct Items	Average Transaction Length	# of Transactions
Kosarak	0.002	30.5	41271	8.1	990002
Retail	0.006	4.2	16470	10.3	88126
Pumsb	3.5	16.3	2113	74	49046
Mushroom	19.3	0.56	119	23	8124
T10I4D100k	1,15	3.83	870	10.1	100000

We use two kinds of datasets in our experiments; one synthetic dataset (T10I4D100K) and four real world datasets (Kosarak, Pumsb, Retail, and Mushroom). The synthetic dataset T10I4D100K is created with the data generator [2] which is widely used for evaluating frequent pattern mining algorithms. The real world datasets are taken from FIMI repository [14]. The important characteristics of the synthetic and real word datasets are shown in Table 3. The density[1] of a dataset indicates the similarity of the transactions.

[1] Density (%) = (Average Transaction Length / # of Distinct Items) × 100

We use the following formula for assigning MIS to items that is based on their actual supports [7]. LS represents the least minimum item support, $\beta \in [0,1]$ represents the parameter used to control how the minimum support values of items should be related to their occurrences in the database and $f(i)$ represents the number of transactions that contain item i (the support of item i).

$$MIS(i) = \begin{cases} f(i) * \beta , & f(i) * \beta > LS \\ LS , & Otherwise \end{cases}$$

Notice, if $\beta = 1$ and $f(i) \geq$ LS, then the minimum item support threshold values of items are the actual support of items, $f(i)$, whereas if $\beta = 0$, then there is only one minimum support LS and the same process of mining with single threshold is implemented as FP-growth algorithm. In our experiments the β parameter is calculated by the following formula: $\beta = \frac{1}{\alpha}$.

According to this formula, increasing the value of **α** leads to decrease in MIS of items and that increases the number of frequent patterns that are generated. In these experiments, the value of **α** is increased and the value of LS is fixed**.** For the datasets Kosarak, Retail, T10I4D100k and Mushroom, **α** is varied from 1 to 10 and we set LS at 0.001, 0.001, 0.01 and 0.1 respectively. In quite dense dataset, Pumsb, **α** is varied from 1 to 1.9 and we set LS at 0.6. This is because it has a high number of distinct items and the average transaction length is large as well. For example, when the value of **α** is set to 1.5 and LS = 0.6, the number of frequent itemsets discovered from the Pumsb dataset is about 2 million as shown in Fig.2 (d). Figs.2 (a), (b), (c), (d) and (e) show the number of frequent patterns, which are generated with respect to **α.** It can be seen from the graphs that increasing **α** leads to increase in the number of frequent patterns for all datasets.

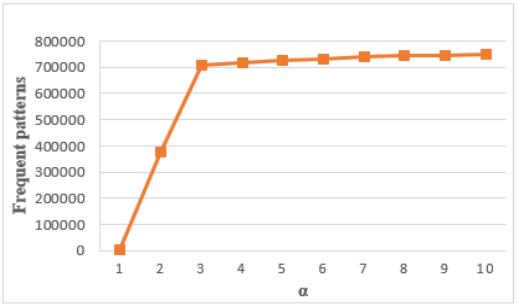
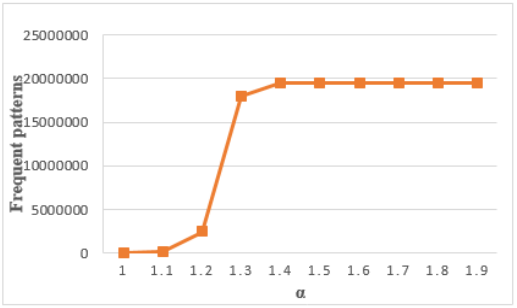


Fig.2 (a). Frequent patterns in Kosarak dataset.



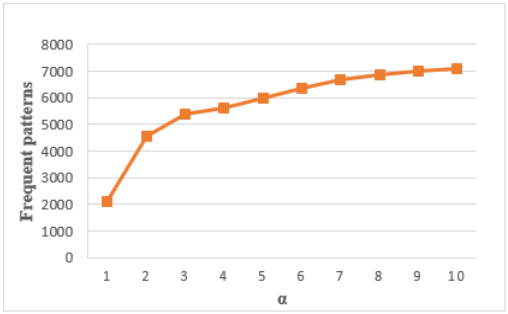


Fig.2 (b). Frequent patterns in Retail dataset.

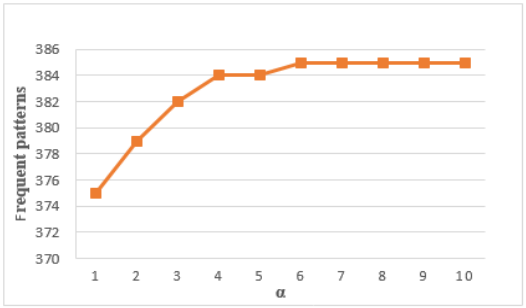


Fig.2 (c). Frequent patterns in T10I4D100K dataset.

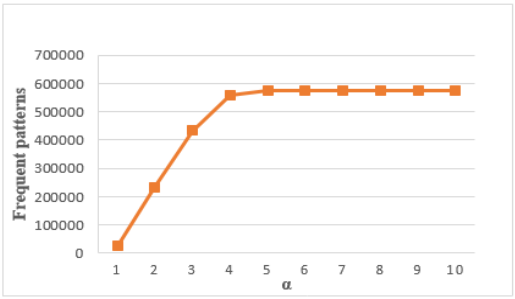


Fig.2 (e). Frequent patterns in Mushroom dataset.

4.2 Execution time

The execution time comparison of MISFP-growth and CFP-growth++ is shown in Figs.3 (a), (b), (c), (d), (e). The performance of two algorithms with various **α** is measured on the given five datasets. Note that, the execution time here means the total runtime time, which is the period between input and output. The experimental results reveal that our proposed method, MISFP-growth, is substantially faster than CFP-growth++ almost in all cases. This is due to the fact that CFP-growth++ method spends too much time to rebuild the MIS-Tree when there

are lots of useless items. It has to carry out an exhaustive search to prune unpromising items. Then, MIS-Tree has to be traversed to merge child nodes carrying the same name and linked to same parent.

Once MISFP-Tree of MISFP-growth is built, no pruning and reconstruction is needed since it contains only useful items that satisfy MIN-MIS. Furthermore, in the mining process we avoid generating patterns from unpromising items and we skip them in building conditional pattern base and conditional MISFP-Tree. For a very dense dataset like Mushroom, the performance of two methods is almost the same. This is due to the fact that there are few items that have to be discarded during rebuilding of MIS-Tree of CFP-growth++.

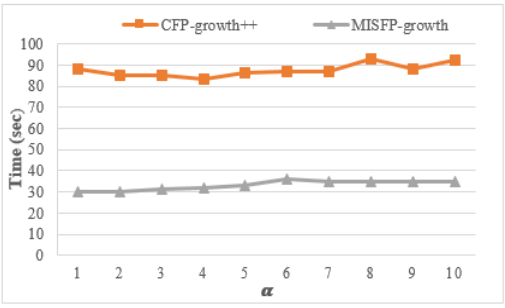


Fig.3 (a). Execution time for Kosarak dataset.

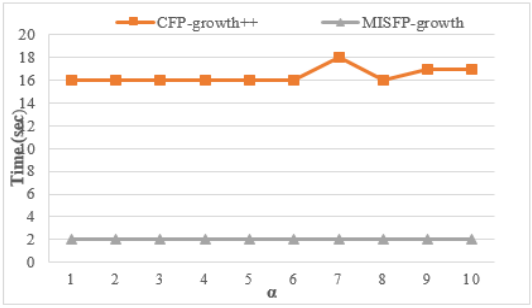


Fig.3 (b). Execution time for Retail dataset.

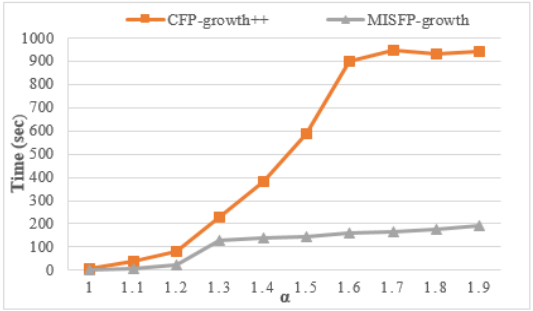


Fig.3 (c). Execution time for Pumsb dataset.

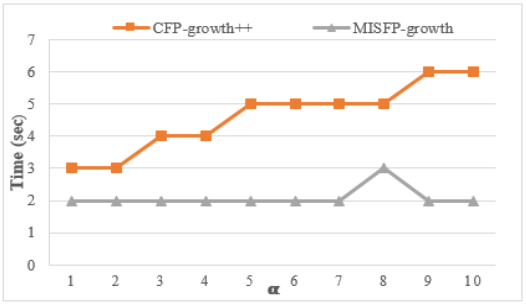


Fig.3 (d). Execution time for T10I4D100K dataset.

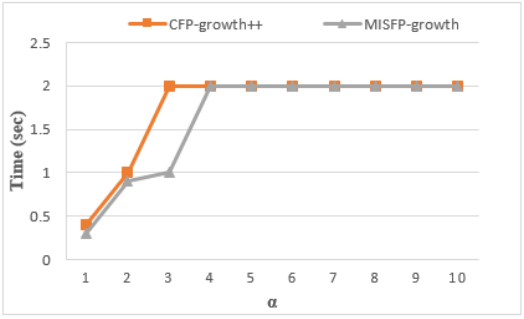


Fig.3 (e). Execution time for Mushroom dataset.

The speedup of the proposed method is summarized in Table 4. In this table, column 4 in contains the minimum speedup (MIN) and the maximum speedup (MAX) of MISFP-growth against the compared method. The speedup[2] is defined as the ratio between the execution time of CFP-growth++ and MISFP-growth. Speedup can be up to magnitude of 8-9 on sparse dataset like Retail. On the other hand, on a dense dataset like Mushroom, execution time of MISFP-growth can be half of CFP-growth++. As a summary, from this table it can be seen that MISFP-growth is more efficient than CFP-growth++ for all sparse and dense datasets.

Table 4. Speedup summary of MISFP-growth with varied **α**

Datasets	Density	Varied (α)	Speed-up [MIN, MAX]
Kosarak	0.002	[1, 2, …, 10]	[2.4, 2.9]
Retail	0.006	[1, 2, …, 10]	[8, 9]
Pumsb	3.5	[1.1, 1.2, …, 1.9]	[1.8, 6.5]
T10I4D100k	1.15	[1, 2, …, 10]	[1.5, 3]
Mushroom	19.3	[1, 2, …, 10]	[1, 2]

4.3 Memory usage

In this section we demonstrate the results of the experiment that is carried on to compare memory usage performance of MISFP-growth and CFP-growth++ on the datasets given in Table 3. Similar to execution time experiment, we change **α** and fix

[2] speedup = execution time of CFP-growth++ / execution time of MISFP-growth

the value of LS. Figs. 4. (a), (b), (c), (d) and (e) show memory consumption of the two algorithms on the given datasets. As it can be noticed from the graphs, MISFP-growth, consumes much less memory than CFP-growth++ for all five datasets. This is due to the absence of items that play no role to create any frequent patterns. MISFP-growth discards the items that play no role to generate any frequent patterns and constructs the MISFP-Tree by only items that can be utilized to create frequent patterns.

In CFP-growth++, MIS-Tree is built by items which cannot generate any frequent patterns and the tree is rebuilt once again to discard meaningless items by pruning and merging operations. In addition, CFP-growth++ consumes more memory to store all items from a dataset in MIN-frequent header table and then those useless items are discarded from this table during rebuilding the tree. In the proposed method, MISFP-growth, MIN-MIS-frequent header table contains only the useful items that can be utilized to create frequent patterns.

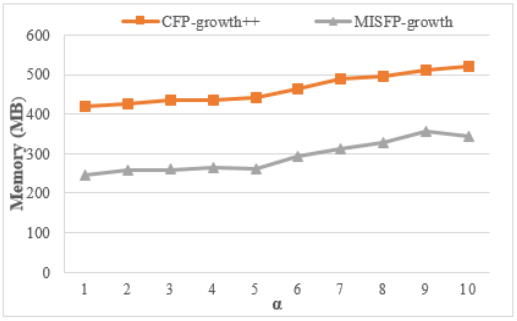


Fig.4 (a). Memory usage for Kosarak dataset.

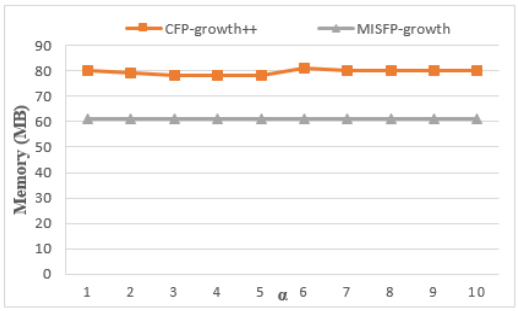


Fig.4 (b). Memory usage for Retail dataset.

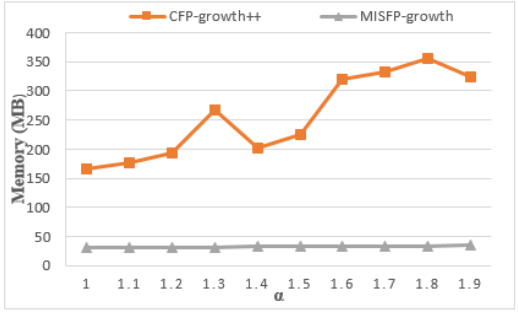


Fig.4 (c). Memory usage for Pumsb dataset.

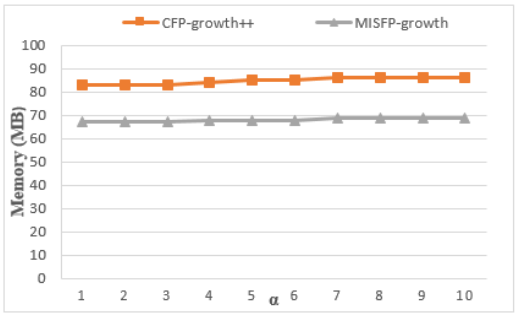


Fig.4 (d). Memory usage for T10I4D100K dataset.

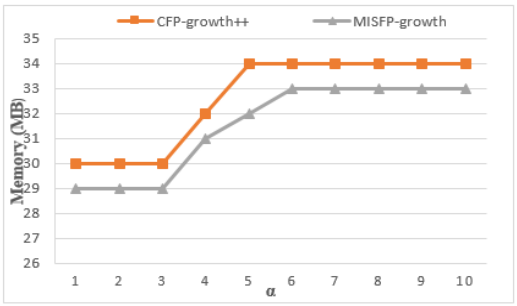


Fig.4 (e). Memory usage for Mushroom dataset.

Table 5 shows the memory gain[3] of the proposed method compared to CFP-growth++ under varied **α**. Column 4 in this table shows the minimum memory gain (MIN) and the maximum memory gain (MAX) of MISFP-growth against the compared method.

Table 5. Memory gain summary of MISFP-growth with varied **α**

Datasets	Density	Varied (α)	Memory Gain (%) [MIN, MAX]
Kosarak	0.002	[1, 2, …, 10]	[30, 41]
Retail	0.006	[1, 2, …, 10]	[22, 25]
Pumsb	3.5	[1.1, 1.2, …, 1.9]	[81, 90]
T10I4D100k	1.15	[1, 2, …, 10]	[19, 20]
Mushroom	19.3	[1, 2, …, 10]	[3, 6]

It can be noticed that MISFP-growth consumes less memory in all cases except in Mushroom dataset where the memory consumption of the proposed method is slightly less than memory consumption of the compared method. For a sparse dataset like Kosarak, memory gain can reach up to 41%, on the other hand for a dense dataset like Mushroom memory gain can be 6% only.

4.4 Discussion on Results

Experimental results show that the proposed method significantly outperforms CFP-growth++ on both real and synthetic datasets in terms of execution time, memory usage and scalability. As it can be

[3] Memory gain = (Memory consumption of CFP-growth++ - Memory consumption of MISFP-growth) / Memory consumption of CFP-growth++

observed from the Tables 4 and 5, the speedup in execution time and memory gain is quite high for the sparse datasets but not quite significant for very dense datasets (like Mushroom). On the sparse datasets, the generated trees are much bigger than those on dense datasets. Hence, the bigger is the tree, the more is time cost. Thus, MISFP-growth is much more efficient than CFP-growth++ on sparse datasets. With very dense datasets, MISFP-growth and CFP-growth++ work almost the same. This is quite expected, as in the case of very dense dataset, most of items are frequent and only a few pruning effort is needed in MIS-Tree built up. For example, in the case of Mushroom dataset, we can observe that the performance efficiency of two compared methods is almost the same.

5 Related work

An efficient heuristic algorithm has been proposed in order to mine frequent patterns [2]. Apriori algorithm deploys a breadth-first search to count the support of (k+1)-itemsets that are created from frequent k-itemsets. It achieves good performance by reducing the search space as the *downward closure* property is utilized. Since it is considered an innovation that opened new doors for many frequent patterns mining applications, many variants of Apriori have been proposed to enhance the performance of Apriori such as Matrix Apriori [12], BitApriori [13], etc. On the other hand, the multiple database scan approach of Apriori algorithm is very I/O expensive for large databases. In addition, due to the candidate generation-and-test approach, it requires huge computational time and memory usage when too many candidate itemsets are generated.

To handle these weaknesses, FP-growth [3] and its improvements [4, 5, 6] have been proposed to generate frequent patterns without creating a huge amount of candidate itemsets as Apriori. FP-growth methods utilizes FP-tree, an extended prefix-tree, which compresses all transactions of database in horizontal data format in memory. This enables FP-growth to search for the complete set of frequent patterns, which eliminates reduces the number of database scans.

Above methods are used to find frequent patterns with single *minsup* but using only a single *minsup* implicitly assumes that all items in the data are of the same nature or have similar frequencies in the database. In fact frequent pattern mining based on single threshold might cause abundance of meaningless frequent patterns (low threshold) or loss of useful patterns (high threshold). To tackle this problem which is called a rare itemset problem, MSapriori [7] has been proposed to discover frequent patterns with multiple support thresholds. It is an extension of Apriori algorithm. As *downward closure* property implies, an itemset is frequent if and only if all its subsets are frequent but this does not hold when we assign multiple *minsup* values to items/itemsets. Frequent itemsets are found if an itemset satisfies the lowest MIS value among items within it. In this method, the frequent items are assigned with a higher MIS value whereas rare items are assigned with a lower MIS value.

Two other methods have been proposed to mine frequent patterns with MIS based on Apriori [8, 9]. These methods work as MSapriori with the some differences as follows. In [8], it first finds all the frequent 1-itemsets for the given database by comparing the support of each item with its predefined minimum support. It then finds all the frequent k-itemsets for the database by comparing the support of each candidate k-itemset with the maximum of the minimum supports of the items contained in it. In [9], all the steps are same as that used in MSapriori with following exception: 1) it discovers frequent patterns (L) by basic Apriori with a single minimum support, 2) choose all frequent patterns from L that satisfy the definition of frequent patterns with multiple minimum supports from L. These methods [7, 8, 9] are based on Apriori algorithm. Therefore, they adopt an Apriori-like candidate set generation-and-test approach and it is always costly in terms of memory and execution time when the database is large and frequent patterns are long.

To address this problem, a multiple item support tree (MIS-Tree) which extends the FP-tree structure [3], has been proposed for storing compressed and crucial information about frequent patterns [10]. A MIS-Tree-based mining method, CFP-growth algorithm was developed for mining the complete set of frequent patterns with multiple minimum support thresholds. It finds out the whole set of frequent itemsets with a single scan of the transaction database. However, CFP-growth expends too much time for discovering the whole set of frequent patterns since it repeats growth process until each conditional pattern base becomes empty for each item. This is because *downward closure* property no longer holds in multiple item support framework.

To reduce the search space, an improved CFP-growth method called CFP-growth++ has been proposed [11]. In this method, four different pruning operations have been introduced to reduce the search space and avoid growth process until each conditional pattern base becomes empty.

CFP-growth++ has to conduct an exhaustive search in reconstruction of the tree structure. This is due to time that is consumed for punning useless items. Then, MIS-Tree has to be scanned to merge any child nodes linked to same parent node. In addition, initial tree occupies large memory space since it is built with all items in the database.

6 Conclusion

In this paper, we propose an efficient algorithm, MISFP-growth that is based on FP-growth algorithm and is designed to discover interesting patterns involving both of frequent and rare patterns. It constructs MISFP-Tree to hold all necessary information that are needed in mining process. This tree is efficiently constructed with only useful items that play role to generate frequent and rare patterns. Thus, reconstructing the tree is not needed. The experimental results indicate that MISFP-growth performs better than CFP-growth++ in term of both runtime and memory consumption.

Up to now, a few methods have been proposed to mine frequent patterns with MIS. We can sense that there is much more to do in this field. For upcoming studies, we plan 1) to carry on more experiments to understand the scalability performance of MISFP-growth, 2) to extend MISFP-growth with the capability of finding meaningful rare patterns with multiple thresholds without generating a huge number of frequent patterns, and 3) to extend MISFP-growth to mine frequent patterns under MIS in incremental databases.

References:

[1] Agrawal, Rakesh, Tomasz Imieliński, and Arun Swami. "Mining association rules between sets of items in large databases", ACM SIGMOD Record. Vol. 22. No. 2. ACM, 1993.

[2] Agrawal, Rakesh, and Ramakrishnan Srikant. "Fast algorithms for mining association rules", Proc. 20th int. conf. very large data bases, VLDB. Vol. 1215. 1994.

[3] Han, Jiawei, Jian Pei, and Yiwen Yin. "Mining frequent patterns without candidate generation", ACM SIGMOD Record. Vol. 29. No. 2. ACM, 2000.

[4] Grahne, Gosta, and Jianfei Zhu. "Fast algorithms for frequent itemset mining using fp-trees", Knowledge and Data Engineering, IEEE Transactions on 17.10 (2005): 1347-1362.

[5] Jalan, Shalini, Anurag Srivastava, and G. K. Sharma. "A non-recursive approach for FP-tree based frequent pattern generation", Research and Development (SCOReD), 2009 IEEE Student Conference on. IEEE, 2009.

[6] Zhang, Wei, Hongzhi Liao, and Na Zhao. "Research on the FP growth algorithm about association rule mining", Business and Information Management, 2008. ISBIM'08. International Seminar on. Vol. 1. IEEE, 2008.

[7] Liu, Bing, Wynne Hsu, and Yiming Ma. "Mining association rules with multiple minimum supports", Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 1999.

[8] Lee, Yeong-Chyi, Tzung-Pei Hong, and Wen-Yang Lin. "Mining association rules with multiple minimum supports using maximum constraints", International Journal of Approximate Reasoning 40.1 (2005): 44-54.

[9] Xu, Tiantian, and Xiangjun Dong. "Mining frequent patterns with multiple minimum supports using basic Apriori." Natural Computation (ICNC), 2013 Ninth International Conference on. IEEE, 2013.

[10] Hu, Ya-Han, and Yen-Liang Chen. "Mining association rules with multiple minimum supports: a new mining algorithm and a support tuning mechanism", Decision Support Systems 42.1 (2006): 1-24.

[11] Kiran, R. Uday, and P. Krishna Reddy. "Novel techniques to reduce search space in multiple minimum supports-based frequent pattern mining algorithms", Proceedings of the 14th International Conference on Extending Database Technology. ACM, 2011.

[12] Pavón, Judith, Sidney Viana, and Santiago Gómez. "Matrix Apriori: Speeding Up the Search for Frequent Patterns", Databases and Applications. 2006.

[13] Le, Thi, Thi Nguyen, and Tae Chong Chung. "BitApriori: an apriori-based frequent itemsets mining using bit streams", Information Science and Applications (ICISA), 2010 International Conference on. IEEE, 2010.

[14] Frequent Itemset Mining Implementations Repository http://fimi.ua.ac.be/data/