Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 112 (2017) 417–426

Procedia Computer Science

www.elsevier.com/locate/procedia

International Conference on Knowledge Based and Intelligent Information and Engineering Systems, KES2017, 6-8 September 2017, Marseille, France

# Vertical Pattern Mining Algorithm for Multiple Support Thresholds

Sadeq Darrab[a], Belgin Ergenc[a]*

[a]*Izmir Institute of Technology, Computer Engineering Department, Izmir, 35447, Turkey*

**Abstract**

Frequent pattern mining is an important task in discovering hidden items that co-occur (itemset) more than a predefined threshold in a database. Mining frequent itemsets has drawn attention although rarely occurring ones might have more interesting insights. In existing studies, to find these interesting patterns (rare itemsets), user defined single threshold should be set low enough but this results in generation of huge amount of redundant itemsets. We present Multiple Item Support-eclat; *MIS-eclat* algorithm, to mine frequent patterns including rare itemsets under multiple support thresholds (MIS) by utilizing a vertical representation of data. We compare *MIS-eclat* to our previous tree based algorithm, MISFP-growth[28] and another recent algorithm, CFP-growth++[22] in terms of execution time, memory usage and scalability on both sparse and dense databases. Experimental results reveal that *MIS-eclat* and MISFP-growth outperform CFP-growth++ in terms of execution time, memory usage and scalability.

*Keywords:* frequent pattern mining, multiple support thresholds, vertical mining, pattern growth tree

## 1. Introduction

Association rule mining has drawn a big attention since it was first proposed in[1] due to its applicability in various domains as medical studies, telecommunication data, market basket analysis, etc. Association rule mining focuses on finding co-occurrence of items in databases where the relationships between these items are expressed as association rules. The overall goal of this process is to discover all interesting rules from a database that have the

* Corresponding author. Tel.: +90-532-226-4125; fax: +90-232-750-7862
*E-mail address:* belginergenc@iyte.edu.tr

1877-0509 © 2017 The Authors. Published by Elsevier B.V.
Peer-review under responsibility of KES International
10.1016/j.procs.2017.08.051

form: $X \rightarrow Y \mid X \cap Y = \emptyset$ where X and Y are the set of items in the database. An interesting rule should satisfy two statistical measures defined by the user and known as minimum support threshold denoted as *minsup* and minimum confidence threshold denoted as *minconf. Minsup* refers to the percentage of transactions in the database that contain $X \cup Y$ whereas *minconf* refers to the conditional probability of finding $X \cup Y$ given the transactions that contain X. An itemset is frequent if its support exceeds *minsup,* an association rule is frequent if its confidence exceeds *minconf.*

Frequent association rules can be found in two consecutive steps; 1. Generation of all frequent patterns that satisfy a given *minsup*, 2. Generation of association rules which satisfy both of *minsup* and *minconf,* from frequent patterns found in step 1. Since the first step is computationally expensive almost all research on association rule mining focuses on generating frequent patterns that frequently co-occur (itemset) in a database. For the same reason, association rule mining, frequent pattern mining and frequent itemset mining are used interchangeably.

Since the introduction of frequent itemset mining[1], most of the well-studied algorithms, such as Join-based algorithms[1,2,3,4,5,6,7,8], Tree-based algorithms[9,10,11,12] or Vertical Mining [13,14] employ the uniform *minsup* at all levels. Using single *minsup* allows the utilization of downward closure property which says "any subset of frequent itemset should be frequent" and reduces the search space and computation cost considerably. Thus, these algorithms avoid a huge amount of infrequent itemsets from being processed. However these algorithms assume that all items in the database have the same nature and similar frequencies but this assumption is not true for real-life applications. In many applications, some items appear very frequently in the database whereas others hardly ever appear. Users require finding not only frequent itemsets but rare items as well. To identify the frequent and rare itemsets, single *minsup* is not adequate since when *minsup* is set low, the number of frequent itemsets goes up dramatically, and the performance of these algorithms degenerates quickly. If *minsup* is set too high, interesting rare patterns may be missed. This problem is called rare item problem[15]. Rare item problem is also studied in tasks like classification[30] or periodic pattern mining[31], in this paper we only consider mining both frequent patterns and rare ones by utilizing Multiple Item Support (MIS) thresholds instead of single support threshold, in association rule mining process.

Several algorithms are studied to reduce search space and execution time while generating frequent patterns under MIS[15,16,17,18,19,20,21,22,23,24,25,28]. These algorithms overcome the rare item problem by discovering the complete set of frequent patterns including rarely occurring ones since they apply different support threshold to each item. These algorithms can be classified in terms of search strategies as breadth-first search algorithms[15,16,17,18,19] and depth-first search algorithms[20, 21, 22, 23,24,25,28]. Breadth-first search algorithms which are based on Apriori algorithm scan the databases many times with their candidate generation-and-test approach. In order to overcome this weakness, depth-first search algorithms based on FP-Tree structure are proposed. These algorithms require database scan at most twice since FP-Tree holds all necessary information that is needed in mining process. However, these algorithms are still far from being efficient since they require huge amount of memory due to the management of irrelevant nodes and high execution time for tree operations in pre-mining phase as pruning and reconstruction. There is no algorithm that is devised for multiple item support thresholds that uses vertical representation of data and discarding property.

In this paper; we present a new itemset mining under MIS algorithm; *MIS-eclat* and compare it to our previous algorithm MISFP-growth[28] and another recent algorithm, CFP-growth++[22]. *MIS-eclat* 1) utilizes the vertical representation of data to find the support of itemsets, and 2) constructs AdjacencyMIS structure with useful items that have support greater than the least MIS in order to increase efficiency in terms of execution time and memory usage. In order to be self-contained, we revisit our previous algorithm, MISFP-growth[28] that is pattern growth tree based. To assess the performance of these algorithms, execution time, memory usage and scalability experiments are conducted on both sparse and dense databases in comparison to a recent tree based algorithm; CFP-growth++[22]. The experimental results show the superiority of *MIS-eclat* and MISFP-growth algorithms in comparison to CFP-growth++, in terms of runtime and memory usage. The results also show that *MIS-eclat* and MISFP-growth scale up much better than CFP-growth++ as the size of database increases except with dense databases.

The organization of the paper is as follows: in section 2, we give preliminaries of the challenge. In section 3, *MIS-eclat* is presented and MISFP-growth is revisited to be self-contained. Experimental results are shown in section 4 while the related work is discussed in section 5. Conclusion remarks are given in section 6.

## 2. Preliminaries

In this section, we introduce the basic terminology related to frequent pattern mining under single and multiple thresholds. Let I={$i_1$,$i_2$,…,$i_m$} represent the set of m distinct items, and D={$T_1$,$T_2$,…,$T_n$} be a transaction database where Ti (i ∈ [1…n]) is a transaction, which contains a set of items in I. 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= {$x_1$, $x_2$... $x_k$} 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.

**Frequent Pattern with Single Threshold:** Let D 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 D are the patterns which exceed *minsup.* As an 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 given definition above, the K itemset is frequent as its support exceeds *minsup* = 50%, whereas Z is infrequent its support does not satisfy *minsup*.

**Frequent Pattern with Multiple Thresholds:** Let I be a set of I items I = {$i_1$,…, $i_n$}, an itemset X = {$i_1$, …, $i_k$}, the minimum item support (MIS) of itemset X is defined as follows: MIS(X)=MIN{MIS($I_1$),MIS($I_2$),…,MIS($I_k$)}. As an example assume that an itemset K = {x, y, z} has an actual support = 8% in a given database. Suppose that the MIS of items are given as: MIS(x) = 5%, MIS(y) = 10%, MIS(z) = 15% and actual supports of items are given as: sup(x) = 10%, sup(y) = 9%, sup(z) = 11% then the MIS of the itemset K can be defined as: 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%. This is called downward closure property with MIS; in another words, any itemset containing an item with support less than the lowest minimum support threshold cannot be considered as frequent.

Table 1. Transaction database D

| TID | Item | Ordered items |
|---|---|---|
| 100 | d, c, a, f | a, c, d, f |
| 200 | g, c, a, f, e | a, c, e, f, g |
| 300 | b, a, c, f, h | a, b, c, f, h |
| 400 | g, b, f | b, f, g |
| 500 | b, c | b, c |

Table 2. MIS and actual support of items in D

| Item | a | b | c | d | e | f | g | H |
|---|---|---|---|---|---|---|---|---|
| MIS value | 4 | 4 | 4 | 3 | 3 | 2 | 2 | 2 |
| Actual support | 3 | 3 | 4 | 1 | 1 | 4 | 2 | 1 |

## 3. Frequent Pattern Mining for MIS

In this section, we explain our algorithms to mine frequent patterns with multiple support thresholds. In the first subsection, the new proposed algorithm, *MIS-eclat*, is introduced. In the second subsection the bottom-up tree based algorithm, MISFP-growth[28], is revisited to be self-contained. We start by explaining a motivating example that will be used in the presentation of the algorithms. A sample database is given in the Table 1. Multiple support threshold of each item is given in the Table 2. Last row of Table 2 shows actual support of each item in the database D. In the right most column of Table 1, items in the transactions are in decreasing order of their multiple support thresholds.

In *MIS-eclat* algorithm, the process of mining frequent patterns is done by utilizing a vertical representation of databases. Similar to MISFP-growth algorithm, this algorithm follows bottom-up approach to extract the complete set of frequent patterns. To mine the complete set of frequent patterns including rare ones, it avoids scanning database multiple times instead it scans at most twice. *MIS-eclat* is similar to Eclat algorithm[13]. *MIS-eclat* is used to mine frequent patterns including rare itemsets based on multiple support thresholds whereas the original one is used to mine frequent patterns under single threshold. In MISFP-growth[28], the process of mining frequent patterns is done by utilizing the growth tree and bottom-up search strategy. Bottom-up strategy builds itemset combinations from smallest to the largest like FP-growth[9]. Main difference of the subject algorithm from FP-growth is its capability of finding frequent patterns based on multiple support thresholds instead of unique minsup given by the user. MISFP-growth has two steps; 1) construction of the pattern growth tree 2) generating frequent patterns from the tree.

Both algorithms use discarding property which says any item that has support less than minimum of MIS (MIN-MIS) is discarded and is not used. Let us go through our motivating example to explain this property. The least minimum support threshold in this example is 2 as seen from MIS values in the second row of Table 2. Thus items {d, e, h} can be discarded since their actual support is less than 2. The discarding property is utilized to build AdjacencyMIS and MISFP-Tree by only promising items that have support more or equal to MIN-MIS.

## *3.1. MIS-eclat algorithm*

*MIS-eclat* algorithm is a novel algorithm for mining frequent patterns including rare patterns. It is an extended version of Eclat algorithm[13] to find frequent patterns under multiple support thresholds. The difference between our algorithm and Eclat is as follows; 1) the items in *MIS-eclat* is in increasing order according their MIS, 2) the Tidset itself is built with items that have support no less than MIN-MIS and 3) it works to mine frequent patterns including rare itemsets under MIS instead of single threshold. It finds frequent patterns under multiple support thresholds by utilizing a vertical representation of data called *AdjacencyMIS* structure. This structure is built to hold items and their Tidset. Items that have support less than MIN-MIS will not be added to this structure since they play no role to generate frequent patterns. Furthermore, we use *AdjacencyMIS* structure to find the support of all itemsets.

Table 3. AdjacencyMIS for the database in Table 1

| item | Tidset of transactions | | | |
|---|---|---|---|---|
| f | 100 | 200 | 300 | 400 |
| g | 200 | 400 | | |
| a | 100 | 200 | 300 | |
| b | 300 | 500 | 500 | |
| c | 100 | 200 | 300 | 500 |

The main processes of this algorithm can be summarized as follows;
1- Scan database once to find actual support of items,
2- Find the least minimum support threshold, MIN-MIS,
3- Build *AdjacencyMIS* of items that have support no less than MIN-MIS,
4- Find the single frequent patterns, FP, with items that have support greater or equal to MIN-MIS and order them in increasing order of their MIS,
5- Extract Frequent patterns, FPs, for each frequent item, i, whose support no less than its predefined MIS(i),
  5.1- Create a candidate itemsets for suffix item, i, for all items in step 4,
  5.2- Find the support of the candidate itemset using *AdjacencyMIS* in step 3,
  5.3- Add the frequent itemset that have support no less than MIS(i) to FPs,
  5.4- Repeat step 5.1 to 5.3 for all items found in step 4.

Let us understand how *MIS-eclat* algorithm with our motivating example given in Table 1 and Table 2.
1- Scan database once to find the support of items that can be found in the last line of Table 2.
2- Find the lowest minimum support threshold, MIN-MIS = 2.
3- Create the AdjacencyMIS of items that have support greater than MIN-MIS is shown in Table 3.
4- The single items that have support greater than MIN-MIS = {f, g, a, b, c}.
5- For all single frequent patterns, {f, g, c} that have support greater than their perspective MIS, extract the frequent patterns that can be generated with each frequent item. The items {a} is not considered since its support = 3 which is less than its MIS = 4. For the same reason, item {b} is not considered as well.

For item f, the process of generation all frequent patterns from f can be done as follows. All itemsets that can be generated from item f should satisfy the minimum support threshold of item, f = 2 since items is ordered in increasing order according their MIS. From the Table 3, item g occurs twice with item f, in transactions 200 and 400. Thus, itemset {fg} is frequent since its support not less than MIS value of item f. The itemset, {fa} with support

= 3 (intersection of item f and a) is frequent since it satisfies the minimum support value of item f = 2. Similarly, all frequent patterns that can be found from item f are {fg: 2, fa: 3, fb: 2, fc: 3, fac: 3}. The process is terminated, since there is no 2-frequent itemsets or more can be generated from either item g or item c. For example, ga just occurs once together. So, it is not frequent since it does not exceed MIS value of item g. Thus, the complete set of frequent patterns are {f: 4, g: 2, c: 4, fg:2, fa: 3, fb: 2, fc: 3, fac: 3}.

### 3.2. MISFP-growth algorithm

*MISFP-growth* (Multiple Item Support Frequent Pattern growth) algorithm finds frequent patterns under multiple support thresholds by utilizing a pattern growth tree called MISFP-Tree[28]. This tree is formed by scanning all the transactions of the given database. The items placed in the tree are the ones that obey discarding property which captures the idea of "the items that have support less than MIN-MIS can play no role to create any frequent patterns".

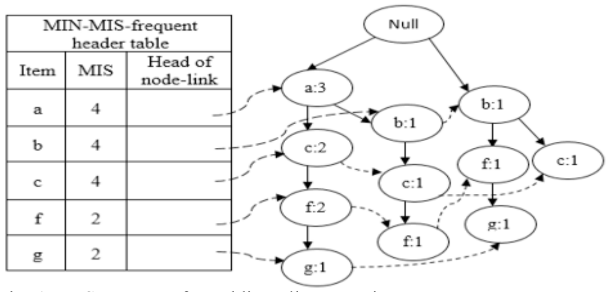


Fig. 1. MISFP-Tree after adding all transactions.

Table 4. The complete set of frequent patterns

| Suffix Item | *Minsup* | Conditional Pattern Base | Conditional trees | 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:3}, {c:3}, {b:2}, {ac:3} | af:3, cf:3, bf:2, acf:3 |
| c | 4 | {a:2}, {a,h:1}, {b} | - | - |
| b | 4 | {a:1} | - | - |
| a | 4 | - | - | - |

The steps of *MISFP-growth* algorithm can be best understood using our motivating example. MISFP-Tree is built from the database given in Table 1 and predefined minimum support values given in Table 2;

1. Scan database once to find out the support of items in the database D as shown in the second row of the Table 2.
2. Find out the least minimum support threshold among all minimum item support thresholds: MIN-MIS = 2.
3. Scan database once again to construct MISFP-Tree with the items in the right column of the Table 1. The process of inserting transactions into the tree works as follows;
4. The root of MISFP-Tree is created and labeled as "null".
5. For each transaction; items that have support greater or equal to 2, are inserted into the tree in descending order in term of their minimum item support thresholds. The first branch of MISFP-Tree is created by adding the first transaction {a, c, f}. The count of each node in this path is assigned by 1. 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). By repeating same steps, consecutive transactions are added to the tree. Nodes that have the same item-name are linked in sequence by node-links starting from head of node-link of MIN-MIS frequent header table as seen in Fig.1.
6. For the generation of frequent patterns under multiple item support thresholds MISFP-growth algorithm uses the similar procedure of CFP-growth++[22]. However it does do not require pruning and reconstruction of the growth trees since MISFP-Tree keeps items with support greater than MIN-MIS. This feature brings the advantage of not dealing with useless items in frequent pattern generation step. MISFP-Tree building algorithm and frequent pattern generation phase is explained in[28]. Complete set of frequent patterns is shown in Table 4.

## 4. Performance Evaluation

In this section, *MIS-eclat* and MISFP-growth are compared to a recent tree based algorithm, CFP-growth++ [22] using four real databases with different characteristics. We first explain experimental environment and the databases in the first subsection and then we show the results of execution time, memory usage and scalability.

### *4.1. Experimental Environment*

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. The java source code of *CFP-growth++* has been downloaded from[30]. All the programs are converted and implemented in C#. We conduct our experiments on four real word databases which are widely used for evaluating frequent pattern mining algorithms; Retail, BMS-POS, Kosarak and BMSWebView2. The important characteristics of these databases and execution parameters are shown in Table 5. The last column of this table stands for the range of frequent patterns that are generated under $\alpha = [1 \dots 10]$ by all algorithms. The density of database (second column of the table) is: Density (%) = (Average Trans. / # of Distinct Items) × 100.

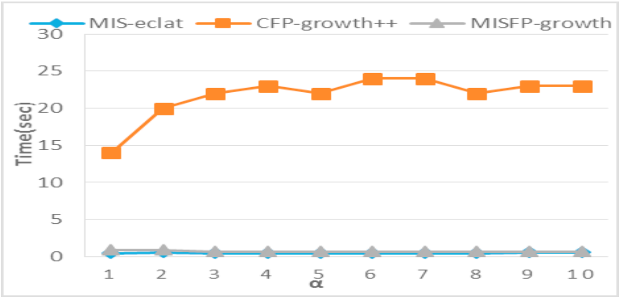
Table 5. Characteristics of databases

| **Database** | **Density (%)** | **Size (MB)** | **# of Distinct Items** | **Average Trans. Length** | **# of Transactions** | **LS** | **# of Frequent Patterns** |
|---|---|---|---|---|---|---|---|
| Retail | 0.006 | 4.2 | 16,470 | 10.3 | 88,126 | 0.01 | [2,117-7,097] |
| BMS-POS | 0.39 | 11.62 | 1,657 | 6.5 | 515,597 | 0.01 | [141-1,085] |
| Kosarak | 0.002 | 30.5 | 41,271 | 8.1 | 990,002 | 0.01 | [54- 383] |
| BMSWebView2 | 0.15 | 2.26 | 3,340 | 5 | 77,512 | 0.001 | [1,133-14,536] |

The following formula is used for assigning MIS to items that is based on their actual supports[15]. LS stands for 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 $f(i)$ refers to 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. 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 increase in the number of frequent patterns that are generated. In these experiments, the value of α is increased in the range of [1-10] and the value of LS is fixed to 0.01 for the three former databases, and 0.001 for BMSWebView2.



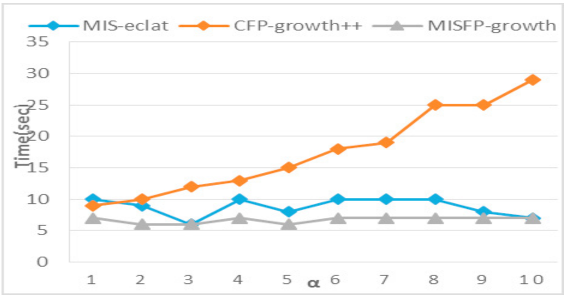
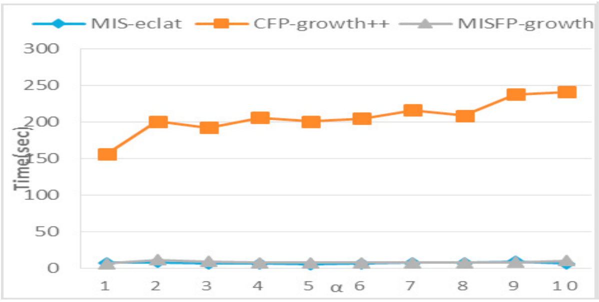


Fig. 2. (a) Execution time for Retail; (b) Execution time for BMS-POS.



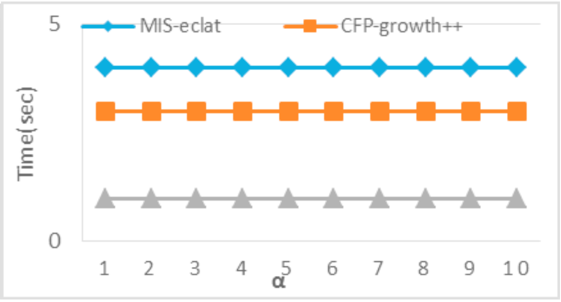


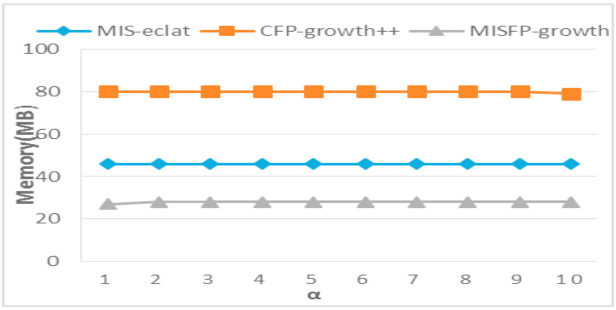
Fig. 2. (c) Execution time for Kosarak; (d) Execution time for BMSWebView2.

## *4.2. Execution Time*

In this subsection, we compare *MIS-eclat* and MISFP-growth to a state-of-the art algorithm, CFP-growth++, in terms of execution time for all databases shown in Table 5. For the Retail database, Fig.2 (a); MISFP-growth and *MIS-eclat* algorithms perform approximately same, their performance is about 25 orders of magnitude faster than CFP-growth++ for all α values. For the BMS-POS database, Fig.2 (b), MISFP-growth and *MIS-eclat* also perform better for all α values. For the Kosarak database, as seen in Fig.2, (c), both of MISFP-growth and *MIS-eclat* is about 250 orders of magnitude faster than CFP-growth++. MISFP-growth and *MIS-eclat* once again they achieve the similar performance for all α values. For BMSWebView2, Fig. 2 (d), we can observe that MISFP-growth performs better than the other two. CFP-growth++ is better than *MIS-eclat* on dense database.

## *4.3. Memory Usage*

In this subsection, we compare *MIS-eclat* and MISFP-growth to a state-of-the art algorithm, CFP-growth++, in terms of memory consumption on all databases shown in Table 5.  For Retail database, Fig.3 (a), shows that, memory cost of CFP-growth++ is highest one. The memory usage of CFP-growth++ is about 4 and 2 times of that consumed by MISFP-growth and *MIS-eclat* respectively. For BMS-POS database, the performance comparison of the three algorithms is shown by Fig.3 (b). In the graph, it can be noticed that the memory usage of *MIS-eclat* and CFP-growth++ is almost same. For Kosarak database, in Fig.3 (c) it can be observed that MISFP-growth consumes the lowest amount of memory. The memory consumption of MISF-growth and *MIS-eclat* is less than memory usage of CFP-growth++. For BMSWebView2, the memory usage of the three algorithms is shown by Fig.3 (d); once again the memory usage that is used by MISFP-growth is the lowest one among the other algorithms. Also, the deference between *MIS-eclat* and *CFP-growth* is slight.



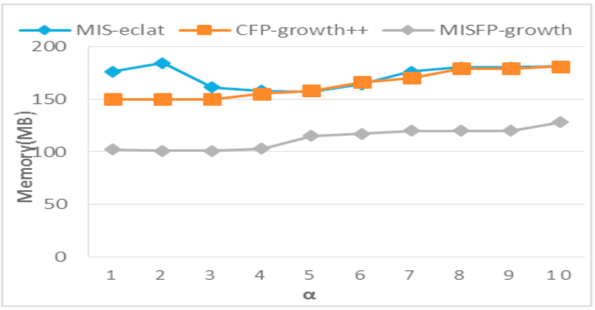
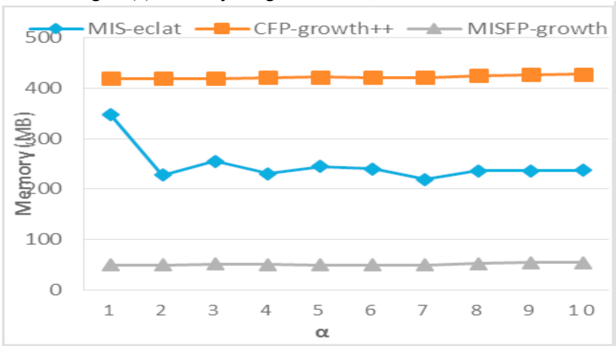


Fig.3. (a) Memory usage for Retail; (b) Memory usage of BMS-POS.



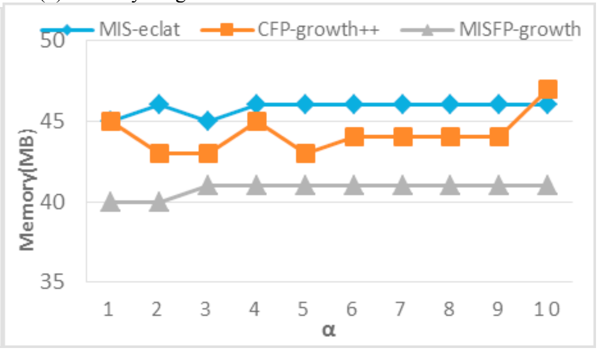


Fig.3. (c) Memory usage of Kosarak; (d) Memory usage for BMSWebView2.

## *4.4. Scalability*

In this subsection, we compare the performance of *MIS-eclat*, MISFP-growth and CFP-growth++ in terms of increasing the size of databases. In fact the aim of these measurements is to show how MISFP-Tree and AdjacencyMIS structures scale up with the increasing number of transactions. The databases given in Table 5 are decomposed into 10 evenly sections with approximately average 10% of database size and keep aggregating the parts. We set the parameter α to 4 as it is stated in[15] that this value is common in many real world applications. The LS threshold is set at 0.01, 0.01, 0.01 and 0.001 for Kosarak, Retail, BMS-POS and BMSWebView2 respectively.

Fig. 4. (a), (b), (c) and (d) reveal how the algorithms scale up as the size of the databases increases. As it can be noticed, both of MISFP-growth and *MIS-eclat* algorithms scale much better than the CFP-growth++ algorithm with respect to increasing the size of databases. This is due to the effect of increasing the number of useless items that CFP-growth++ must process by applying pruning and merging operations. Discarding infrequent items by searching through the tree becomes very expensive in MIS-Tree. It can also be seen that the performance of two algorithms while the size of transactions increase is almost the same in dense database like BMSWebView2, since few useless items are discarded and most of the items are frequent while mining with MIS.

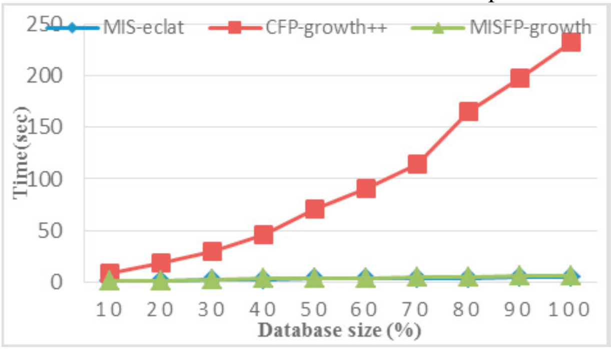
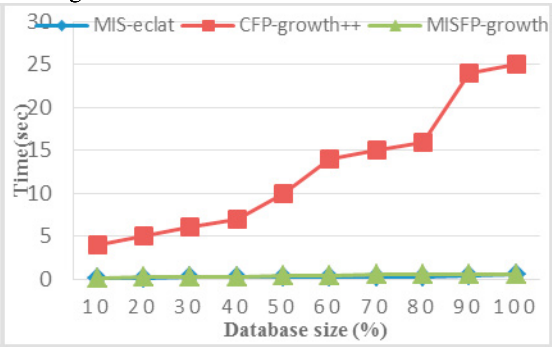


Fig.4 (a) Scalability for Kosarak database;



(b) Scalability for Retail database

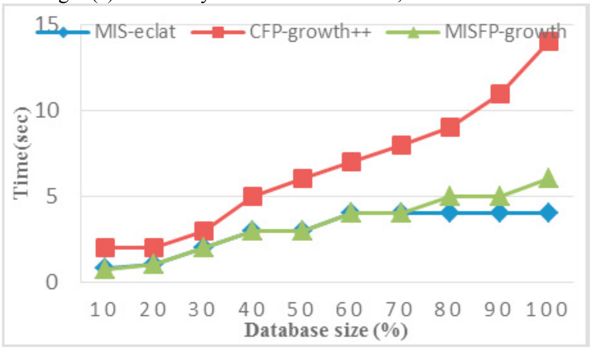
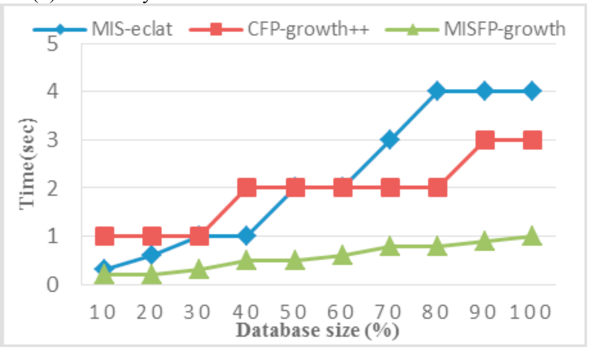


Fig.4 (c) Scalability for BMS-POS database;



(d) Scalability for BMSWebView2 database

## 5. Related Work

Numerous studies are proposed to mine the complete set of frequent patterns from large databases. These algorithms are compared in Table 6. In our context, main classification is done with the parameter of support threshold in the last column; "Single" means the algorithm considers unique support threshold to identify the pattern as frequent, "Multiple" means that the algorithm considers different support thresholds in frequent pattern generation. Third column indicates the algorithmic approach; "Join based", "Tree based" or "Vertical Mining".

Join-based algorithms seen in the first row depend on the breadth-first search strategy to find the whole set of frequent patterns that satisfy given *minsup*[1,2,3,4,5,6,7]. The frequent patterns that are found in the level k are used as seeds to generate the candidate patterns in the level k+1. New database scan is required for each level to find the support of candidate itemsets. Multiple database scan and the candidate generation-and-test approaches of these algorithms are very I/O and memory expensive. Another group of join based-single support algorithms are seen in the second row[8,29]; Matrix Apriori uses matrix to keep the signatures of the transactions to eliminate multiple database scans. Single-support algorithms seen in the third row of the Table 6 are classified as "Vertical Mining" where the data is represented in vertical layout1[3,14] where simple intersection operations are used in mining.

In the fourth row we see tree-based-single support algorithms that utilize FP-Tree structure[9] which is a prefix tree of the lists of frequent items in the transactions. That allows the traversal of the transactions in depth-first order. In these algorithms, the subtree of an itemset is searched only if the itemset is frequent and the search space is shrunk quickly. This enables tree-based algorithms to search for the complete set of frequent patterns by eliminating the number of database scans; it can be said that depth-first search algorithms are usually more efficient than the breadth-first search algorithms. Abovementioned algorithms find out the complete set of frequent patterns based on single *minsup*. Setting a proper minimum support threshold is a crucial issue; low *minsup* causes abundance of meaningless frequent patterns, high *minsup* causes loss of useful patterns (rare item problem)[15].

Table 6. Comparison of frequent pattern mining algorithms

| **Algorithm** | **Algorithmic Approach** | **Data Structure** | **Exploration Approach** | **Support Threshold** |
|---|---|---|---|---|
| Apriori and its optimizations[1,2,3,4, 5,6,7] | Join based | Hash tree | Breadth first | Single |
| Matrix Apriori[8,29] | Join based | Matrix | Breadth first | Single |
| Eclat[13,14] | Vertical mining | Hash tree | Vertical | Single |
| FP-growth and its optimizations[9,10,11,12] | Tree based | FP-Tree | Depth first | Single |
| MSapriori and its optimizations[15,16,17,18,19] | Join based | Hash Tree | Breadth first | Multiple |
| FP-ME$_{Diffest}$[27] | Tree based | FP-ME , Diffest | Vertical | Multiple |
| CFP-growth and its optimizations[20,21,22,23,24,25,26] | *Tree based* | FP-Tree | Depth first | Multiple |

The algorithms that find frequent patterns under multiple item support (MIS) assign *minsup* for each item (last column with the value "Multiple"). Frequent itemsets are found if an itemset satisfies the lowest MIS value among the respective items. In[15,16,17,18,19,] the rare item problem is addressed and this group can be classified as join based-multiple item support algorithms. These studies are based on Apriori algorithm[2]. Therefore, they adopt breadth-first search approach based on candidate-generation-and-test approach and have to scan database several times. In[27], another algorithm, FP-ME, is studied to mine frequent patterns under MIS using vertical representation of data. In this algorithm, the concept DiffSet and Set-Enumeration-tree structure with multiple minimum supports (ME-tree) is utilized in mining process.

To enhance the performance of join-based algorithms, several tree-based algorithms are proposed again focusing on the rare item problem[20,21,22,23,24,25,26]. They utilize FP-Tree structure[9] for storing compressed and crucial information about frequent patterns. These algorithms adopt a depth-first search approach by which they efficiently reduce the search space and avoid the process of candidate-generation-and-test approach. Although the tree-based algorithms try to reduce the search space and database scans and demonstrate better performance than join-based algorithms, they still have high execution times and memory consumption since they need to do heavy tree pruning and merging operations. To our best knowledge, there is only one vertical mining algorithm that is devised for multiple item support thresholds.

## 6. Conclusion

Numerous frequent pattern mining algorithms have been studied extensively in data mining field. The traditional algorithms mine frequent patterns with single *minsup* threshold and that is not adequate for many real life scenarios. In this paper, we propose a new vertical mining algorithm; *MIS-eclat* to discover the complete set of frequent patterns including the rarely occurred ones. We examine this algorithm in comparison to our previous algorithm, MISFP-growth and another recent algorithm, CFP-growth++*,* on dense and sparse databases. *MIS-eclat* and MISFP-growth run faster and need less memory than CFP-growth++ on all databases except dense database like BMSWebView2. Both algorithms provide this advantage by using discarding property; eliminating the processing useless items in constructing MIS-Tree and AdjacencyMIS data structures. For the scalability, *MIS-eclat* and MISFP-growth algorithms scale up much better than CFP-growth++ especially in sparse databases.

## Acknowledgements

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

## References

1. Agrawal R, Imielinski T, Swami A. Mining association rules between sets of items in large databases. ACM SIGMOD Record 1993;22(2): 207-216.
2. Agrawal R, Srikant R. Fast algorithms for mining association rules. 20th VLDB 1994; 1215: 487-499.
3. Agrawal C, Han J. Frequent pattern mining. Springer, 2014.
4. Nhan L, Thuy N, Chung T. BitApriori: an Apriori-based frequent itemsets mining using bit streams. International Conference on Information Science and Applications (ICISA) 2010; 1-6.
5. Ghanem A, Sallam H. Hybrid search based association rule mining. IEEE Pacific Rim Conference on Communications, Computers and Signal Processing 2011.
6. Park J, Chen M, Yu P. Using a hash-based method with transaction trimming for mining association rules. Knowledge and Data Engineering 1997; 9(5): 813-825.
7. Brin S, Motwani R, Ullman J, Tsur S. Dynamic itemset counting and implication rules for market basket data. ACM SIGMOD Record 1197; 26(2):255-264.
8. Pavón J, Viana S, Gómez S. Matrix Apriori: speeding up the search for frequent patterns. Databases and Applications 2006:75-82.
9. Han J, Pei J, Yin Y. Mining frequent patterns without candidate generation. ACM Sigmod Record 2000: 29(2).
10. Grahne G, Zhu J. Fast algorithms for frequent itemset mining using FP-Trees. Knowledge and Data Engineering 2005; 17(10): 1347-1362.
11. Jalan S, Srivastava A, Sharma G. A non-recursive approach for FP-Tree based frequent pattern generation. Research and Development (SCOReD) 2009: 160-163.
12. Zhang W, Liao H, Zhao N. Research on the FP-growth algorithm about association rule mining. Business and Information Management, 2008; 1:315-318.
13. Zaki M. Scalable algorithms for association mining. IEEE Transactions on Knowledge and Data Engineering 2000; 12(3): 372-390.
14. Thieme L. Algorithmic features of Eclat. FIMI 2004.
15. Liu B, Hsu W, Ma Y. Mining association rules with multiple minimum supports. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining 1999: 337-341.
16. Xu T, Dong X. Mining frequent patterns with multiple minimum supports using basic Apriori. In Ninth International Conference on Natural Computation (ICNC) 2013: 957-961.
17. Uday R, Reddy P. An improved multiple minimum support based approach to mine rare association rules. In Computational Intelligence and Data Mining 2009: 340-347.
18. Lee Y, Hong T, Lin W. Mining association rules with multiple minimum supports using maximum constraints. International Journal of Approximate Reasoning 2005; 40(1): 44-54.
19. Bansal A, Baghel N, Tiwari S. An novel approach to mine rare association rules based on multiple minimum support approach. International Journal of Advanced Electrical and Electronics Engineering 2013; 10: 75-80.
20. Hu Y, Chen Y. Mining association rules with multiple minimum supports: a new mining algorithm and a support tuning mechanism. Decision Support Systems 2006; 42(1):1-24.
21. Sinthuja M, Rachel S, Janani G. MIS-Tree algorithm for mining association rules with multiple minimum supports. Bonfring International Journal of Data Mining 2011; 1: 1-5.
22. Kiran R, Reddy P. Novel techniques to reduce search space in multiple minimum supports based-frequent pattern mining algorithms. International Conference on Extending Database Technology 2011: 11-20.
23. Hoque F, Debnath M, Easmin N, Rashed K. Frequent pattern mining for multiple minimum supports with support tuning and tree maintenance on incremental database. Research Journal of Information Technology 2011; 3(2): 79-90.
24. Chen Y, Lin G, Chan Y, Shih M. Mining frequent patterns with multiple item support thresholds in tourism information databases. Technologies and Applications of Artificial Intelligence 2014: 89-98.
25. Ryang H, Yun U, Ryu K. Discovering high utility itemsets with multiple minimum supports. Intelligent data analysis 2014; 18(6): 1027-1047.
26. Taktak W, Slimani Y. MS-FP-growth: a multi-support version of FP-growth algorithm. International Journal of Hybrid Information Technology 2014; 7(3): 155-166.
27. Gan W, Lin J, Viger P, Chao H. More efficient algorithm for mining frequent patterns with multiple minimum supports. International Conference on Web-Age Information Management 2016: 3-16.
28. Darrab S, Ergenç B. Frequent pattern mining under multiple support thresholds. WSEAS Transactions on Computer Research 2016; 4: 1-10.
29. B. Yıldız and B. Ergenç, Comparison of two association rule mining algorithms without candidate generation, 10th IASTED 2010: 450–457.
30. M. Vannucci, V.Colla, Smart under-sampling for the detection of rare patterns in unbalanced datasets, Smart Innovation, Systems and Technologies - 8th KES International Conference on Intelligent Decision Technologies (KES-IDT 2016), vol. 56, pp. 395-404.
31. Surana, Akshat, R. Kiran, and P. Reddy, An efficient approach to mine periodic-frequent patterns in transactional databases. New Frontiers in Applied Data Mining (2012), pp. 254-266.