The Journal of Supercomputing (2022) 78:5479–5506
https://doi.org/10.1007/s11227-021-04066-y

An optimized FP-growth algorithm for discovery of association rules

Mai Shawkat[1] · Mahmoud Badawi[2,3] · Sally El-ghamrawy[4] · Reham Arnous[5] · Ali El-desoky[3]

Accepted: 1 September 2021 / Published online: 24 September 2021

Abstract
Association rule mining (ARM) is a data mining technique to discover interesting associations between datasets. The frequent pattern-growth (FP-growth) is an effective ARM algorithm for compressing information in the tree structure. However, it tends to suffer from the performance gap when processing large databases because of its mining procedure. This study presents a modified FP-growth (MFP-growth) algorithm to enhance the efficiency of the FP-growth by obviating the need for recurrent creation of conditional subtrees. The proposed algorithm uses a header table configuration to reduce the complexity of the whole frequent pattern tree. Four experimental series are conducted using different benchmark datasets to analyze the operating efficiency of the proposed MFP-growth algorithm compared with state-of-the-art machine learning algorithms in terms of runtime, memory consumption, and the effectiveness of generated rules. The experimental results confirm the superiority of the MFP-growth algorithm, which focuses on its potential implementations in various contexts.

Keyword Association rule mining · FP-tree. FP-growth · Frequent itemset mining · Particle swarm optimization

1 Introduction

Currently, association rule mining (ARM) has attracted significant attention in the research areas of data mining. It involves defining the frequency of items and evaluating conditional inference rules between them. Applications of data mining are scattered internally and externally over numerous techniques [1]. Data mining is a method by which beneficial data are collected from some large datasets, results

✉ Sally El-ghamrawy
Sally_elghamrawy@ieee.org; sally@mans.edu.eg

Extended author information available on the last page of the article

analysis, and stored information processing. It is generally exploring and analyzing massive information blocks to discover significant patterns and correlations. Besides, it also discovers hidden relationships between different items in large datasets. Frequent itemsets are necessary for frequent item mining (FIM) from transaction databases. It is essential to investigate association rules in various real-life applications, such as cross-marketing, retail, market basket analysis, fraud detection, and spam email filtering. Frequent itemsets may be transactional or relational [2, 3].

Discovering behavioral patterns has been widely used in association analysis. It focuses on the support and confidence threshold to comprehend whether the data are correlated. Support is the occurrences of items in the transaction dataset. Confidence is the indication of the strength of a rule in terms of percentage. An itemset is called frequent when its support level is exceeded or equals the minimum support defined by the user. The methodology is to first find the frequent items by scanning the dataset according to the support and confidence level and then extract association rules [4]. Discovered association rules with confidence values less than the minimum confidence are eliminated. ARM [5] is one of the widely used techniques for association analysis. Based on the ARM, we can identify potential relationships between items in a large transaction database. The ARM seeks to recognize the transactional behaviors that conform to the primary metrics used to build an association rule [6]. Minimum support and confidence are the primary metrics for evaluating the effectiveness of the generated rules in the database.

An association rule can be identified by $I=\{i1, i2, i3 ....in\}$ including several different items, and a database of transactions $DB=\{T1, T2, T3 ...Tm\}$ consists of several transactions where every transaction T is a subset of items list I. The form of an association rule is $X \rightarrow Y$ *and* $X \cap Y = \varphi$, where X, Y is a subset of I. The set of items X is referred to as antecedent and Y is referred to as the consequent. The support of X will be dependent on the occurrence frequency in the database, and *freq(X)/DB* is determined from the fraction of every row including the items specified in support of itemset X. An association rule support is determined from the combination of the antecedent and the consequent *(X, Y)* as follows:

$$Support(X \rightarrow Y) = (X \cup Y)/DB$$

An association rule confidence is determined by the percentage of rows, which includes itemset X and Y as follows:

$$Confident(X \rightarrow Y) = P(X/Y) = \sup port(X \cup Y)/Support(X)$$

ARM seeks to recognize all rules, which meet user-specific constraints. Association mining aims to derive similarities with attention grabbing, frequent patterns, and association rules in transaction databases or various data repositories. This data industry has created numerous data mining applications that can be used to showcase the product in retail shopping, telecommunication networks [7], textile industry [8], physical education [9], and the Internet of Things (IoT) [10]. These applications work for all items that have specific attributes—patient/symptoms, restaurants/menus, internet/keywords, basket/products, and railways /timings.

Several aspects are of great concern, such as analyzing the market basket, promoting financial products, planning the warehouse, recommending news, and analyzing the network faults. Market basket analysis indicates the products that are frequently purchased together and it might provide data for promotion strategies. For example, the purchase of shampoo is accompanied by the purchase of a conditioner. The promotion of shampoo might increase the sale of the conditioner. The users themselves specify the primary metrics for discovering the association rules of traditional algorithms. Whether the value is smaller or larger as the results will be affected.

With frequent pattern mining techniques, there are two major problems. First, the database is scanned several times. Second, the complicated candidate method for generating a candidate with too many itemsets is generated. These two problems are the bottleneck of efficiency in FIM.

Many FIM techniques are discussed extensively, and the studies are shown in the literature survey of frequent itemset mining [11, 12]. One of the most famous algorithms is Apriori which uses to find frequent itemsets from large databases and extending them to larger frequent itemsets [13]. The frequent patterns identified by Apriori are used to generate association rules. However, the Apriori algorithm repeatedly scans the database in the mining process, which influences the mining running speed and the average search space for all frequent itemset is large. Many enhancement algorithms [14, 15] have been proposed to increase the Apriori efficiency based on the original algorithm. Several useful algorithms for distributed and parallel frequent itemset mining have been proposed. The AprioriTID algorithm [16] is a variation of the Apriori algorithm. The TID list is a list that stores the IDs of transactions containing that itemset. In the AprioriTID algorithm, the former uses the encoding of the candidates created in the previous iteration. So, after every pass, the encoding size is reduced, which reduces the complexity of the mining to a certain extent. However, many challenges still exist [17]. The management of unbalanced and cost-sensitive information is also an essential concern. The mining of massive data algorithms is a challenge. Furthermore, the creation of efficient algorithms for valuable patterns for iterative and collaborative mining is a major interest in big data applications [18].

A further achievement in FIM is the FP-growth algorithm [19, 20] which proposes a method for compressing the required information for the frequent pattern mining in FP-tree without candidate generation and recurrently builds FP-trees for all frequent patterns. The prefix-trees are used to store the database in the FP-tree compact model. It is divided into subtrees called conditional pattern bases, and then, it separately mines the frequent itemsets.

To compensate for this deficit in the mining procedure, this study proposes the modified FP-growth (MFP-growth) algorithm based on existing methods. The proposed MFP-growth is defined based on three main features. Firstly, it employs the structure of an address table to reduce the mapping difficulty of recurrent itemsets in the MFP-tree. Secondly, a modern FP-tree* structure eliminates the need to construct conditional FP-trees. Thirdly, the proposed algorithm has lower memory demand and superior efficiency than FP-growth-based algorithms of using the header address table and FP-tree *.

Experimental results indicate the proposed method and associate them with various existing algorithms, including the FP-growth particle swarm optimization algorithm (PSOFP-growth) [21]. PSOFP-growth is an enhanced algorithm implemented into a new algorithm to find the appropriate support level and mining rules using the FP-growth algorithm. The MFP-growth and PSOFP-growth algorithms applied to the association analysis of a social security incident, which was not easy to solve. The experimental results showed that the MFP-growth analyzes effectively the frequent itemsets with less need for storage requirements using filter data to provide an accurate output of association rules that meet the specific requirements. The MFP-growth outperforms other algorithms under various minimal supports in execution time.

The applicability of traditional association rule algorithms has decreased as a result of the explosive growth of data, and it is hard to discover the rules we need directly from a large amount of data. Thus, in the context of large-scale data, the proposed MFP-growth enhances the mined rule efficiency and decreases the algorithm running time and memory consumption. The major contributions of this paper are listed below:

- Proposing the MFP-growth algorithm to boost FP-growth efficiency.
- An address table layout is used to present frequent itemsets in an MFP-tree, which reduces the mapping complexity of the MFP-tree.
- Presenting the structure of the MFP-tree mining methodology to obviate the requirement of constructing conditional pattern bases facilitates the configuration of the MFP-tree.
- Evaluating the performance of the modified algorithm on twenty various datasets with different minimum support levels.
- Testing the validity of the proposed algorithm efficiency by conducting comparative experiments with numerous advanced algorithms, such as FP-growth and PSOFP-growth algorithms.
- Presenting a simplified model willing to increase the reliability of association rules, and boost user experience greatly.
- The MFP-growth algorithm is mined using filter data to measure the strength of generated rules.

The rest of the paper is arranged as follows. Section 2 introduces the related work with a brief survey of current FIM and association rule mining algorithms. The MFP-growth algorithm and MFP-tree construction are discussed in Sect. 3. The comparative experiments using real and synthetic datasets such as dataset description, runtime consumption, and memory consumption are analyzed in Sect. 4. Finally, Sect. 5 concludes the paper and presents future work.

2 Related works

Nowadays, the construction of a conditional FP-tree involves space and time complexity [17, 19–47]. Many attempts [20–36, 47] have been made to resolve this issue. Zaki [22] has developed the equivalence class transformation (ECLAT) algorithm for frequent itemset mining. It relies on the intersection property and can perform both sequential and parallel computing. Unlike Apriori and FP-growth, ECLAT works vertically and creates a TID list for each item. Unfortunately, ECLAT also suffers from the drawbacks of generating too many itemset candidates, like Apriori. Hyper-structure mining of frequent patterns (H-Mine) algorithm [23] proposed to overcome the FP-growth performance bottlenecks by using queues rather than a tree data structure. H-Mine organizes transaction items into distinct queues and uses hyperlinks to connect transactions having the same first item name. It is considered good for sparse datasets and is efficient in terms of memory and runtime consumption than Apriori and FP-growth. But in the case of dense datasets, FP-growth significantly overpowers H-Mine. Borgelt [24] developed an effective implementation of an FP-growth projection design to accelerate the conditional FP-tree construction process. Grahne et al. [25] proposed a new FP-array approach to minimize tree traversing time, but the recurring construction of conditional FP-trees still exists. Lin et al. [26] utilized a new address table improved FP-growth (IFP-growth) structure to boost FP-growth efficiency; however, it still requires much computation to generate a conditional FP-tree. Tanbeer et al. [27] introduced a new compressed pattern tree structure (CP-tree) capturing one scan (insertion phase) database information. It has been proved that CP-tree enhances frequent pattern mining efficiency and can be useful in applications, such as interactive and accumulative mining. Current methods of parallelization are based largely on the methodology of multithreading and distributed architecture. For example, the work in [28] parallels FP-growth with shared memory using a multithreading approach. FIM workflows are conducted on separate computing devices and performed the last results by analyzing output data. Another successful distributed computing framework implemented at Hadoop, Map-Reduce [29] provided a secure, powerful, and fault-tolerant data processing service. For instance, the parallel FP-growth (PFP) algorithm [30] employed three Map-Reduce phases to extract tasks of the FP-growth and combine intermediate data. The frequent node set-based boundary POC-tree (FNBP) algorithm [31] employed the Map-Reduce approach to identify mining tasks by establishing an early-stage boundary to remove the irrelevant patterns. The mining medical aggregation behavior based on distributed computing (DCMMAB) methodology [32] incorporated the distributed Map-Reduce computing model with the ARM techniques to identify abnormal activities during the medical insurance process. It used a detection algorithm throughout the medical cluster which focuses on distinct behaviors of patients in medical treatment based on frequent pattern mining. However, Map-Reduce would be generally not preferred to implement workflows with iterative operations that are extremely typical of the ARM algorithm due to the increased usage of the Apache Spark framework [33]. Many methods have been proposed to accelerate Spark's memory-based workflow engine, such as yet another frequent itemset mining (YAFIM) [34]. These algorithms also suffer due to the inherent Apriori limitation from the complexity of time and memory. Zhang et al. [35] introduced the distributed frequent itemset mining (DFIMA) pruning procedure using a matrix that may minimize the computation complexity of the pattern growth stage. The parallel Spark workflow version of the array prefix-tree growth (PAPT-growth) algorithm [36] facilitated the processing of large data as a Spark workflow. There is also the associative classification (AC) method [37]; however, it suffers from many regulations, which is an issue inherited from the ARM. The classification-based association (CBA) [37] is one of the AC-developed algorithms. It employs the Apriori's generating candidates function for exploring association rules from databases. The active pruning rules (APR) in [38] as a new method of classification showed an improvement in classification precision. The technique of sequential minimally optimized (SMO) ontology of context (EAA-SMO) employed an enhanced Apriori algorithm (EAA) for mining and analyzing collected data by using the discovered patterns and rules [39]. The fast incremental updating frequent pattern growth algorithm (FIUFP-growth) is a new approach that focuses on incremental frequent itemset mining by using a new incremental conditional pattern tree structure (ICP-tree) [40]. Table 1 summarizes all of the above-mentioned related works.

The above-related works provide us a better understanding of previous issues. Therefore, it is necessary to improve the operating efficiency of the FIM algorithms to mine the high-dimensional datasets. The contribution of this paper focused on how to enhance the efficiency of the execution time and the association rules of the algorithm. As irrelevant association rules will misguide the decision-making, and this is must be considered. This paper concentrates on improving the reliability and reduction of the memory space specifications of the mined association rules.

3 The proposed modified FP-growth algorithm

The proposed MFP-growth algorithm aims to decrease both runtime and memory usage. The FP-growth algorithm requires traversing the header table at item frequency in descending order to collect each item and then scan the current FP-tree to collect the conditional pattern base of each item. Then, recurrently create the conditional FP-tree without making the best use of some pruning strategies. Primarily, the pattern growth is derived through the series of suffix patterns with the frequently generated itemsets. Figure 1 presents an example of the recursive mining operation in the FP-growth algorithm [41]. These FP-tree recursive constructions require more time and memory. This, a new structure is required to avoid these costly operations to improve runtime performance.

The MFP-tree is a prefix-tree with a node layout and a frequent item address table consists of three fields: One field is indicated by a node list for the frequent item, whereas the other two fields carry the item and the total frequent count for every node throughout the node list. It also maintains the data between itemsets.

The structure of the proposed MFP-tree can be designed as shown as follows:

Table 1 Summary of the literature review from 2000 to 2021 related work

Author	Method	Objective	Advantages	Limitations
R. Agrawal et al. [13]	Apriori	Apriori is a data mining methodology that recognizes frequent patterns and association rules in big transactional databases	-Simple and easy-to-understand algorithm -Multiple scans	-Time-consuming due to expensive candidate generation - Excessive usage of memory
R. Agrawal et al. [16]	Apriori TID	Apriori TID uses TID lists for the encoding of itemset candidates to reduce memory usage	-It is not used in the database for counting the support of itemsets	-It requires more computational time
J. Han et al. [19]	FP-growth	FP-growth proposes a method for Finding frequent patterns without candidate generation	-Faster than Apriori -No candidate generation	-Running time complexity -FP-tree mapping complexity
Zaki et al.[22]	Eclat	Equivalence class transformation (ECLAT) is a bottom-up algorithm. It is suitable for both sequential and parallel computing	-It requires less memory than Apriori -It is suitable for small datasets	-Memory space complexities -It needs more time for intersecting TID lists
J. Pei [23]	H-Mine	H-Mine is a novel mining algorithm with a hyperlinked data structure	-Runs fast in memory-based settings -Performs well with sparse datasets	- It has problems when dealing with dense datasets
G. Grahne et al. [25]	FP-array	FP-array scheme proposed to decrease the traversing time of the tree, but that does not obviate the need to build conditional FP-trees recursively	-Reducing the need for traversing FP-tree	- Consumes more memory when the datasets are sparse
L. Ke-Chung et al. [26]	IFP-growth	IFP-growth is an improved frequent pattern growth technique. It is used for association rule mining using an effective data configuration with less memory requirement	-Efficient and scalable algorithm -does not generate subtrees	-Large amount of time and memory
S. Tanbeer et al. [27]	CP-tree	Compact pattern tree (CP-tree) is a frequent pattern mining method with single pass (insertion phase) using a prefix-tree	-One scan (insertion phase)	-Overfitting -Time complexity

Table 1 (continued)

Author	Method	Objective	Advantages	Limitations
L. Liu et al. [28]	Parallel FP-growth	Parallel FP-growth divides the mining task into some independent subtasks, generated by using a multiple-core processor	-Eliminates the locks in building a single FP-tree	-Runtime and memory complexities
J. Dean et al. [29]	Map-Reduce	The Map-Reduce approach provides a simplified and secured data processing service on large clusters, but it would be generally not preferred to implement with iterative operations workflows	-Parallel processing -Simple programming model -Powerful tool with big data	-Requiring a large amount of memory -Not efficient enough for iterative computation
D. Zhang et al. [30]	PFP	Parallel FP-growth employed three Map-Reduce phases for query recommendation	-Scalability -Parallel implementation	-Load balance while dealing with large datasets
E. El-Elshafeiy, et al. [31]	FNBP	A frequent node set-based boundary POC tree (FNBP) algorithm is a big data framework that uses Hadoop. It identifies mining tasks by establishing an early-stage boundary to remove the infrequent itemsets	-High scalability -Parallel computing	-The cost in hardware - Not efficient enough for iterative computation
S. Zhou et al. [32]	DCMMAB	The methodology of mining medical aggregation behavior based on distributed computing (DCMMAB) incorporates the Map-Reduce framework and ARM to identify frauds in health insurance	-Increase the fraud detection accuracy	- The excessive number of rules discovered even on small datasets

Table 1 (continued)

Author	Method	Objective	Advantages	Limitations
H. Qiu et al. [34]	YAFIM	Yet another frequent itemset mining (YAFIM) is a frequent itemset mining in a parallel technique suggested to boost the memory of the Spark workflow engine	-Designed especially to deal with iterative and interactive data processing	-Time and memory complexities
F. Zhang et al. [35]	DFIMA	A distributed frequent itemset mining algorithm (DFIMA) uses a matrix pruning procedure of Spark for big data analytics	-It is used for big data analytics -Improves iterative computation efficiency	-It needs further optimizations to be suitable for more mining cases -no support for real-time processing
X. Niu et al. [33]	PAPT-growth	A parallel Spark workflow version of array prefix-tree growth (APT-growth) algorithm for finding frequent itemsets	-Improves running time performance	-It requires large memory -Workload imbalance issue for distributed operations
B. Liu et al. [37]	CBA	Classification-based association (CBA) explores new association rules from databases using the Apriori candidate's generation technique	-Solving classification problems in mining datasets with many different class labels	-It needs accuracy, the rule set size and training time improvements
K. D. Rajab [38]	APR	Active pruning rules (APR) is a new method of associative classifiers based on rule pruning for the dataset classification that shows an increase in the accuracy of classification	-Improves predictive accuracy and reducing rule redundancy	-It needs improvement in dealing with sparse variables datasets

Table 1 (continued)

Author	Method	Objective	Advantages	Limitations
X. Zhou et al. [21]	PSOFP-growth	Particle swarm optimization-based association rule mining (PSOFP-growth) is an enhanced algorithm introduced into an intelligent algorithm, to discover the optimal support, and mine the association rules with the aid of the FP-growth algorithm	-It has a short computational time -It finds the optimal support which has a great influence on the discovered association rules	- It has a low convergence rate when dealing with complex problems -It has problems in recognizing primary design parameters
M. Sornalakshmi et al. [39]	Sequential minimal optimized (SMO)	The technique of sequential minimal optimized (SMO) ontology of context (EAA-SMO) is a hybrid approach for mining association rules that relies on an enhanced Apriori algorithm (EAA) optimizer	-It reduces the quadratic programming (QP) problem at every iteration	-The training process of support vector machine (SVM)
W. Thurachon et al. [40]	FIUFP-growth	A fast incremental updating frequent pattern growth algorithm (FIUFP-growth) is a new ARM algorithm for mining incremental association rules by retrieving discovered frequent patterns with their support counts and using them again to mine new frequent patterns from the updated dataset	-It constructed a small number of subtrees -It has a short execution time	-It requires a large space to store the constructed subtrees

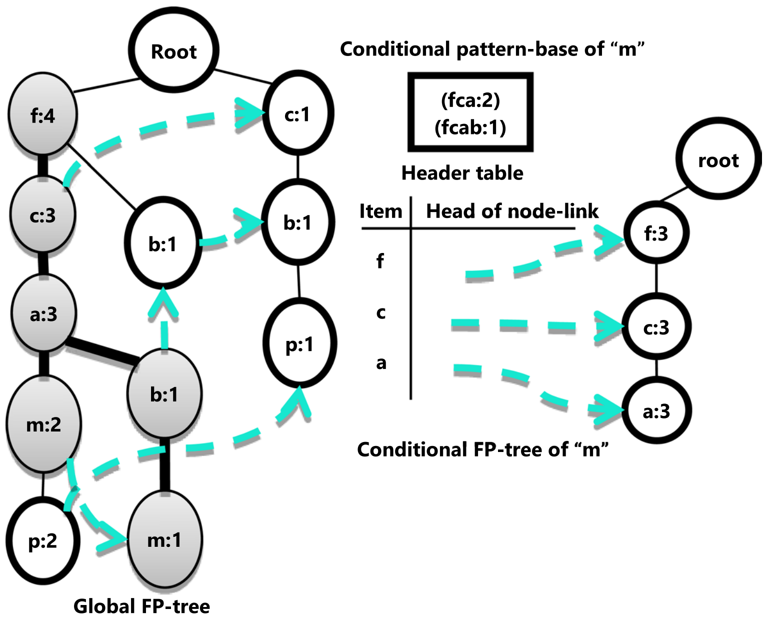


Fig. 1 Recursive mining process of FP-growth algorithm

It consists of three attributes namely:

- Root node: defined as "null."
- Child node: a set of prefix subtrees.
- Header table: header table of frequencies of items.

There are three attributes in every node in the FP-tree:

- Item name: registers the names of the items stored in this file.
- Count: registers the number of item occurrences.
- Node link: prefers to the next node that holds the same item or is labeled with null if there is none.

There are two characteristics for every node in the FP-tree*:

- Item -name
- Head of node link: points to an address table.

The MFP-tree primarily builds over an FP-tree structure, which is developed to keep the information in the dataset without the need for a new generation of FP-tree. Besides, the MFP-growth algorithm eliminates the generation of subtrees and conditional pattern base that consumes more time and memory.

Accordingly, the proposed algorithm has quickly traversed the tree. The FP-tree* technique is presented in this paper. It is used in every conditional pattern mining process to minimize the need for reconstructing FP-trees. By combining the FP-tree* approach with the conditional FP-tree methodology, an MFP-tree mining approach has been proposed to effectively discover frequent itemsets. The FP-tree* mining concept is similar to that of the conditional FP-trees, but the FP-tree* mining direction differs from that of the conditional FP-tree. Every FP-tree* can be built on the original FP-tree which reduces the memory demands.

The FP-tree* construction is presented as follows: Each node in the FP-tree* consists of the item name, node count, and node link. Algorithm 1 shows the procedure of constructing an FP-tree*. Each parent node (PNode) support is accumulated from the children nodes (CNode). The node link of every node connects the node with the same item name through its node link form.

```
Algorithm 1: The FP-Tree* Construction
Input: Tree: a frequent pattern tree, count of frequency of T as freq;
Output: FP-Tree*;
Parameter: HTable: header-table;
Construct: Construct a new FP-Tree;
Temp=FP-Tree. Root;
while ti ∈ T do
        HTable[i].count=0;
        HTable[i].nodelink=NULL;
end
Assume HTable[n].name=freq;
CNode=HTable[n].nodelink;
while CNode= NULL do
        PNode=CNode.parent;
end
while PNode != ROOT do
        Assume HTable[x].name=PNode.name;
HTable[x].count +=CNode.count;
        if PNode has only one child node then
          PNode.count= CNode.count;
          PNode.nodelink=HTable[x].nodelink;
          HTable[x].nodelink=PNode;
      else
          PNode.count+= CNode.count;
          PNode=PNode.parent;
      end
      CNode=CNode.nodelink;
      return FP - Tree;
end
```

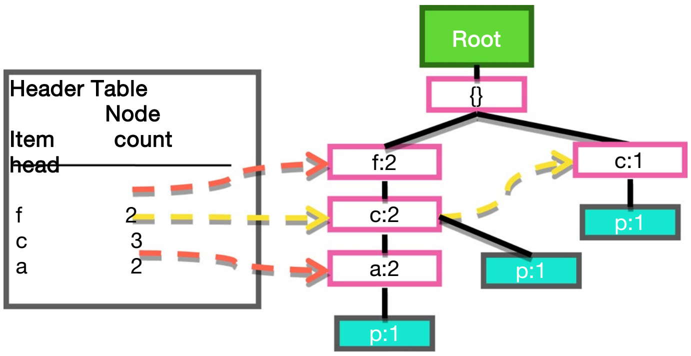


Fig. 2 FP-tree* of item p

Figure 2 shows the mining scheme of the FP-tree* of item p. The frequent itemsets associated with f, c, and p are recursively mined. The frequent item "p" is derived and the following paths are drawn as follows: {<f:1, c:1, a:1, p:1>, <f:1, c:1, p:1>, <c:1, p:1>} of node p. The supports of nodes f and c are 2 in the tree are assembled from path <f, c, a, p> and <f, c, p>. In FP-tree*, the frequent item is in the address table, a frequent itemset {pc} is output, and the FP-tree* of {pc} is constructed for mining consecutively. There are no frequent itemsets derived from {pc} in the example. Therefore, the associated frequent itemsets of p are {p, pc}.

The FP-tree* procedure will be defined as follows: First, the memory use is saved without reconstructing conditional FP-trees as FP-growth. Second, the construction of the FP-tree* is quick because no extra costs are needed to eliminate infrequent items.

The MFP-tree generates and extracts frequent patterns. Algorithm 2 shows the basic steps of the MFP-tree construction, namely: (1) Find frequent itemsets by scanning the DB once, (2) sort the frequency of items in descending order, and (3) scan the DB and build the MFP-tree again. The MFP-growth adopts a hybrid method for frequent itemset mining. The frequent itemsets (*P)* are the output whenever new frequent itemsets are found. At the beginning of the algorithm, the frequent itemsets in a single path are discovered. Then, the MFP-growth mining method recognized the conditional pattern tree type. And depending on the conditional pattern tree (β) type, the algorithm generates all the frequent itemsets from the MFP-growth tree by performing the MFP-growth mining procedure shown in Algorithm 3. The explored frequent itemsets from MFP-tree are from last to first using the "bottom-up" approach.

Algorithm 2: The MFP-growth Algorithm

```
Input: FP-tree*, DB and a minimum support threshold, a: "NUll" initially;
Output: A set of frequent itemsets P ;
//Call MFP-growth (MFP-tree, null);
Procedure MFP-growth (Tree, a);
if Tree contains a single path P then
 foreach combination (denoted as $\beta$) of the nodes in the path P do
         generate pattern $\beta \cup a$ with support = minimum support of nodes in P;
   end
else
     for each ai in the HTable of Tree do
      generate pattern $\beta = ai \cup a$ with support = ai .support;
   end
end
if Tree $\beta \neq \theta$ then
Call MFP-growth(Tree$\beta$, $\beta$);
Return(freq itemset (P))
End
```

Algorithm 3: The MFP-tree construction

```
  MFP-Growth Algorithm: F [1] (MFP-tree)
  $F[I]= \emptyset$;
  Foreach $i \in \tau$ that is in D in frequency increasing order do
  $F[I]=F[I] \cup \{I \cup \{i\}\}$ ;
  $Di = \emptyset$ ;
  $H= \emptyset$;
    Foreach $j \in \tau$ in D such that $j < i$ do
  Select j for which support $(I \cup \{i, j\}) \geq minsup$ ;
  $H = H \cap \{j\}$;
    end
  foreach$(T\,id, X) \in D$ with $i \in X$do
  $Di = Di \cup \{T\,id, \{X\setminus\{i\}\} \cap H)\}$ ;
  Construct MFP-tree from Di , F;
  Call $F[I \cup \{i\}]$ (MFP-tree) ;
  $F[I]=F[I] \cup F[I \cup \{i\}]$ (MFP-tree) ;
  end
end
```

Figure 3 shows the flowchart of the MFP-growth algorithm. The construction of MFP-growth is as follows. Table 2 presents the transaction database. Table 3 displays the list of frequent items with their support count. A frequent item is

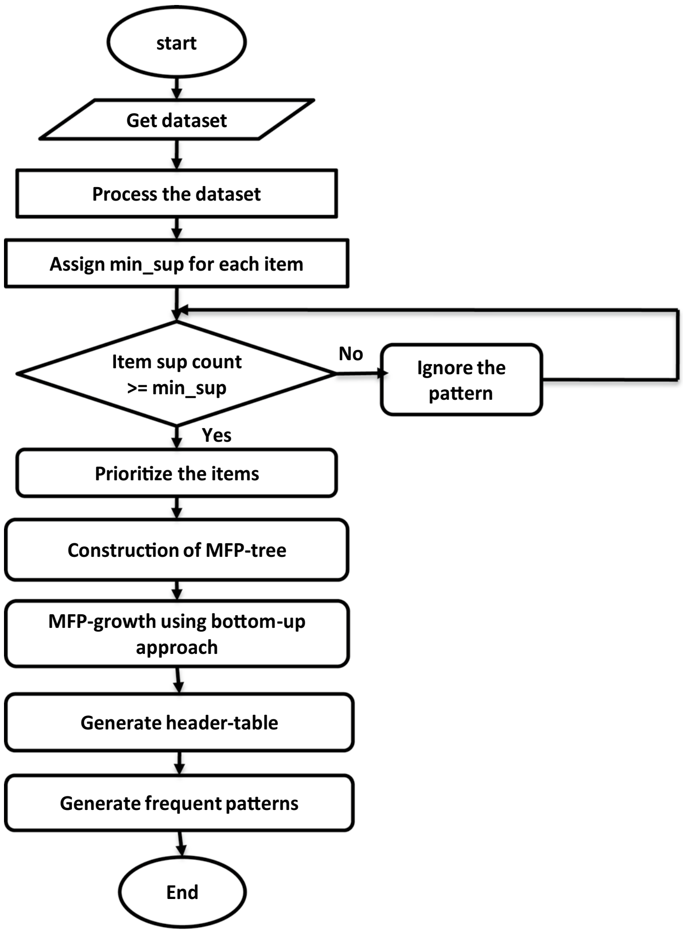


Fig. 3 Flowchart of the MFP-growth algorithm

Table 2 Transaction database

TID	Item bought	Ordered (frequent) items
100	f, a, c, d, g, i, m, p	f, c, a, m, p
200	a, b, c, f, l, m, o	f, c, a, b, m
300	b, f, h, j, o, w	f, b
400	b, c, k, s, p	cb, p
500	a, f, c, e, l, p, m, n	f, c, a, m, p

Table 3 Items priority

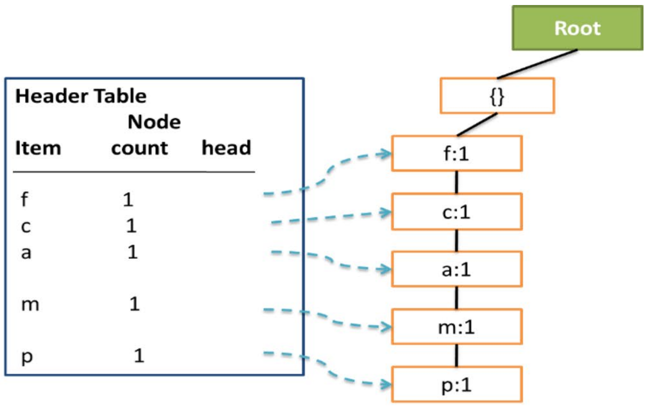
Items	Support
{f}	4
{c}	4
{a}	3
{m}	3
{p}	3
{b}	3

arranged in the descending sequence of support count. Items with lesser support value are pruned.

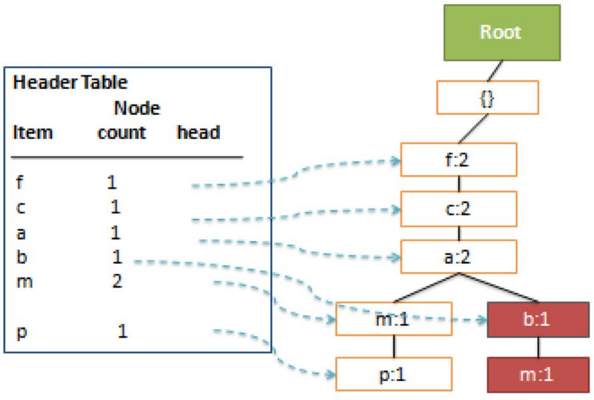
A simple example of the MFP-tree construction process is illustrated in Fig. 4. At first, the MFP-algorithm generates a node called “root.” The second scan starts generating nodes for each transaction. Figure 4 shows that the first transaction of the database contains five items f, c, a, m, and p arranged in descending order, building the first wing (f: 1) (c: 1) (a: 1) (m: 1) (p: 1) in MFP-tree with five nodes. The second transaction contains five items (f: 1) (c: 1) (a: 1) (b: 1) (m: 1), where the five nodes are generated. Here this branch shares the prefix ‘f, ‘c, ‘a with the existing path of transaction “01.” Following this approach, the count of the nodes is incremented to “2.” For items ‘b, ‘m new nodes are generated from item ‘a. It continues for the remaining transactions. Figure 3(c) shows the complete MFP-tree.

4 Results analysis and discussion

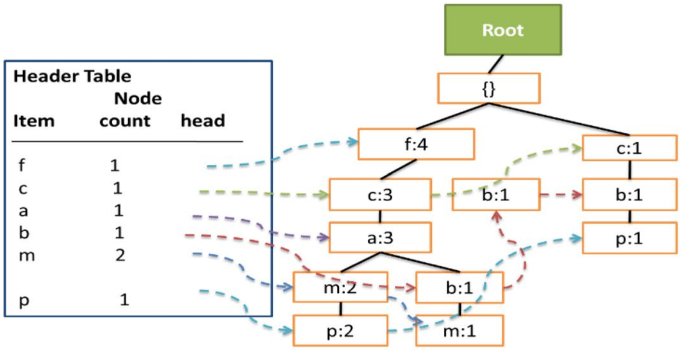
The validity of the proposed MFP-growth algorithm was examined through four experiments. The first experiment was performed on MFP-growth, CBA, and APR algorithms. The second experiment was conducted to test the MFP-growth algorithm superiority compared with the four recent frequent itemset mining algorithms. The third experiment is a comparative analysis of the MFP-growth and five FIM algorithms in terms of their memory consumption. Finally, the fourth experiment was performed to examine the runtime of the modified new algorithm MFP-growth, PSOFP-growth, and the famous algorithms Apriori and FP-growth. The runtime is the time to mine recurrent itemsets of entirely different transactions. For performance evaluation, datasets of various domains are chosen. Different support levels for each dataset were used. The Weka 3.8 workbench tool [42] for data mining tasks is used to conduct the evaluations. It is open-source machine learning software built in Java, which implements different data mining methods such as classification, regression, association rule mining, and visualization. The experimental setup was performed using a desktop computer with 4 GB of free RAM and a Core i5 processor running Windows 10. The experiment measures the execution time against the “minsup” threshold parameter.



(a) Insertion of TID=100.



(b) Insertion of TID=200.



(c) Final MFP-Tree.

Fig. 4 Construction of MFP-tree

4.1 EXP 1: classifier processing time

The proposed MFP-growth algorithm performance was evaluated using various repository datasets from the University of California Irvine (UCI) [43]. Table 4 presents the entire information of the datasets used for the implementation where all numerical attributes have been described. Several established classification algorithms were used to conduct the experiments, which generate rule-based classifiers. In all experiments, the minimum support and minimum confidence for APR, CBA, and MFP-growth were set at 2% and 30%, respectively, according to [38]. The minimum support value has a significant influence in controlling the number of generated candidates. Furthermore, the minimum confidence value has a minimal influence on the execution and is set to 30%.

Figure 5 shows the time algorithms required for the classifier construction on UCI datasets. The proposed algorithm consistently outperforms the CBA and APR in terms of the required training time to generate the classifier. This is because of the MFP-growth classifiers pruning procedure discards the generation of conditional subtrees. This is not the case with CBA, because it assumes that all associated rules specified the confidence threshold. Thus, the CBA will test all the candidates until the training dataset is empty during the classifier construction. On the other hand, the APR classifier needed a lower execution time than CBA to generate the datasets. That is noticeable on the "Mushroom" and "Led" datasets compared with the other datasets, which have a significant number of data examples. However, the proposed algorithm took less processing time to establish the classifier compared to CBA and APR which provides clear evidence of the MFP-growth pruning procedure effectiveness.

Table 4 UCI dataset characteristics

Dataset	Size	No. of classes
Glass	214	7
Iris	150	3
Tic-tac	958	2
Breast	699	2
Zoo	101	7
Heart	270	2
Diabetes	768	2
Pima	768	2
Vote	435	2
Balance scale	625	3
Mushroom	8124	3
Led	3200	3

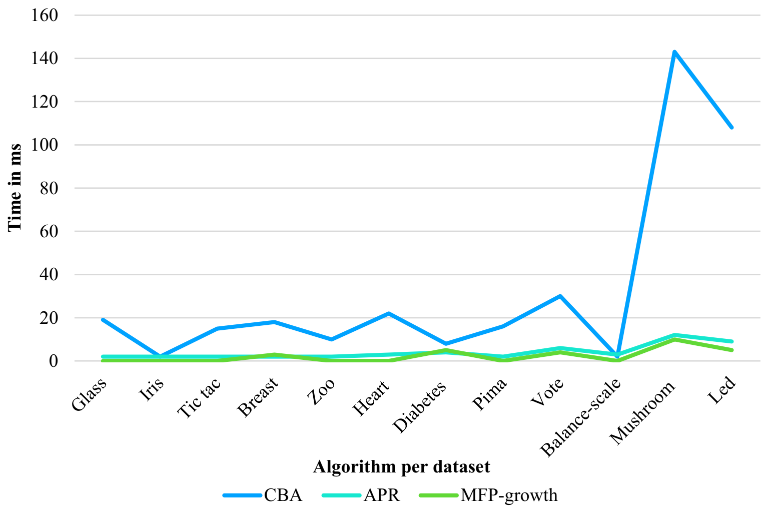


Fig. 5 Processing time needed for creating the UCI dataset classifier

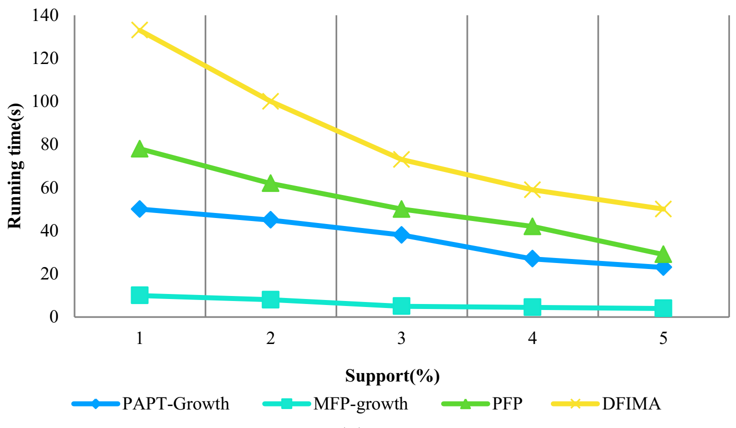
4.2 EXP 2: execution time evaluation

Several workflow experiments were conducted on a Spark platform to evaluate the MFP-growth with the PFP, PAPT-growth, DFIMA, and YAFIM algorithms to test the execution time for the processing of large-scale databases [33]. Various datasets were used in the experiments, such as mushroom, accidents, KddCup99, and Kosarak, downloaded from [43], whereas the synthetic datasets, such as T10I4D100K and T40I10D100K were obtained from [44] using the IBM data generator for the latter. Table 5 presents the properties and characteristics of every dataset. Different minimum support levels run on every dataset in this experiment. Figures 6 and 7 show the experimental results for each dataset.

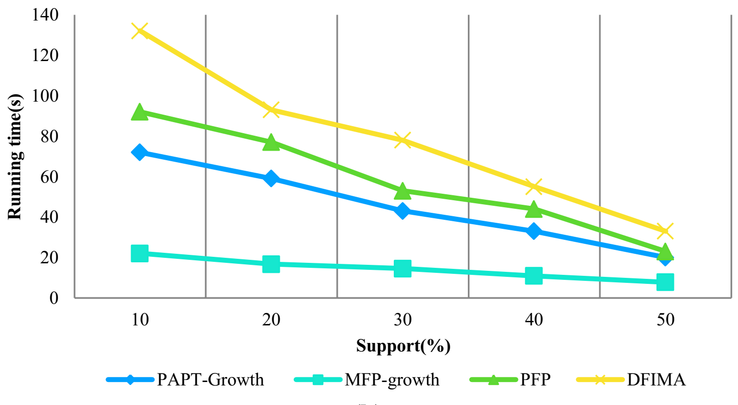
The mushroom dataset is not big but highly complex, and transactions were mostly replicated and irrelevant, indicating that it involves some possible frequent items within that dataset. Figure 6a shows the algorithm runtime on the mushroom dataset for different levels of support. Due to Apriori's inherent limitation, the running time of YAFIM is far beyond the axis range. This is why the YAFIM curve is

Table 5 Properties of the real-life and synthetic datasets

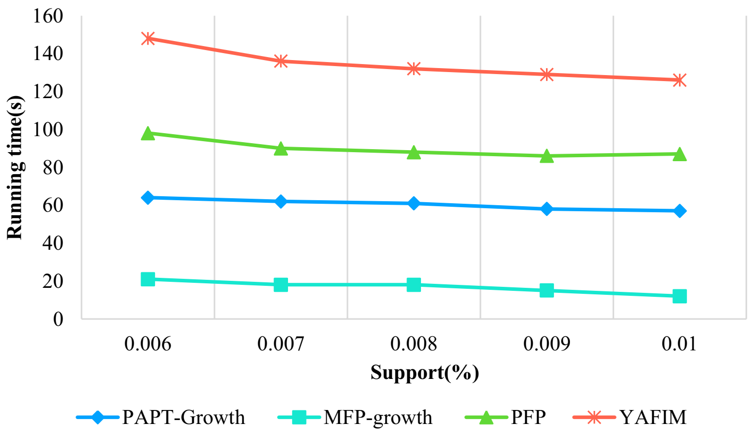
Dataset	Items	Average	Transaction	Size
Mushroom	119	23	8124	507 KB
Accidents	468	34	340,183	33.85 MB
KddCup99	135	16	10,000,000	46.7 MB
Kosarak	41,270	8.1	990,002	30.5 MB
T10I4D100K	870	10	100,000	30.83 MB
T40I10D100K	1000	40	100,000	14.76 MB



(a) Mushroom

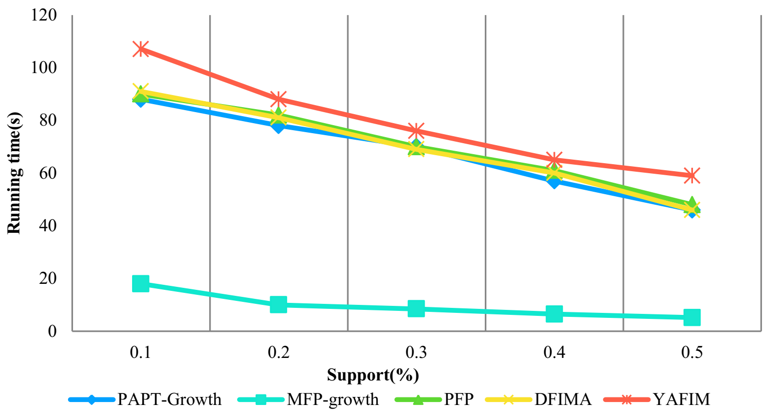


(b) Accidents



(c) KddCup99

Fig. 6 Performance of running time on real-life datasets



(d) Kosarak

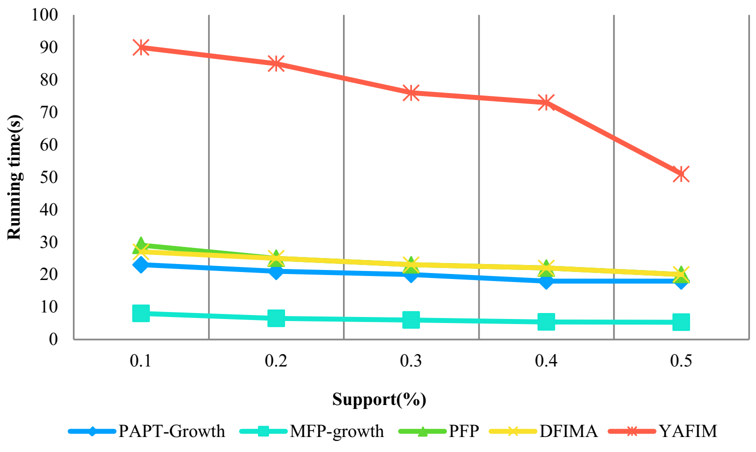
Fig. 6 (continued)

not presented in the figure. As shown in the figure, the PFP curve tends to rise with the threshold reaches 1% probably because there are a lot of items with a support count between 1 and 3%. The curves of MFP-growth, PAPT-growth, and PFP are quite smooth with a support count between 4 and 5%. In this experiment, the MFP-growth outperforms the other three algorithms and the superiority was very obvious because the dataset is dense, which has a major influence on the operation of the candidate generation required by their pruning procedure.

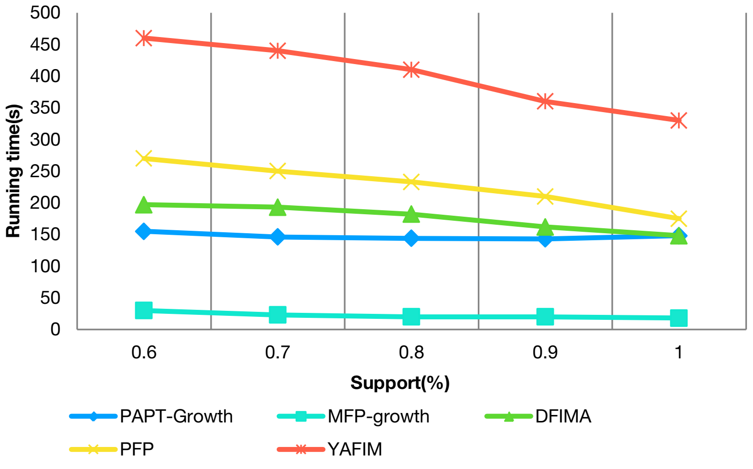
The accidents database is a quite dense database that contains data about traffic accidents. Figure 6b shows the experimental results on the accident dataset. The increase in minimum support value influences the runtime in all algorithms. The reason is that the associated patterns number is decreased with the increase in the level of support, which makes a significant improvement of the recursive process.

KddCup99 and Kosarak, known as UCI repository, are both unique real-life databases. KddCup99 consists of several short transactions and small separate objects, making it fairly easy to process. Figure 6c and d shows the measurement of runtime on KddCup99 and Kosarak, respectively. As shown in Fig. 6c, all the output curves increase gradually because the support threshold decreases. The MFP-growth outperforms the other state-of-the-art algorithms. Kosarak is a sparse dataset that contains a lot of distinct items. All performance curves gradually increase with the support threshold count decreasing, as shown in Fig. 6d. The MFP-growth has a better time performance than PFP, PAPT-growth, DFIMA, and YAFIM algorithms.

Figure 7 shows the measures of effectiveness for synthetic datasets, i.e., T10I4D100K and T40I10D100K. These datasets are created with different conditions by the IBM data generator. Both are similar, but they vary in size and complexity. All algorithms have a consistent efficiency with different thresholds, as seen in Fig. 7a and b. Because T10I4D100K consists of several distinct objects, there is little effect on the frequent itemset number due to the variability in the support value. Thus, the curves appear to be smooth. As shown in Fig. 7b, the MFP-growth



(a) T10I4D100K



(b) T40I10D100K.

Fig. 7 Performance of runtime on synthetic datasets

algorithm's running time is very low; thus, increasing the mining speed. These results show that MFP-growth using the MFP-tree is superior based on the time cost for FIM over PAPT-growth, PFP, DFIMA, and YAFIM. Even if the minimum support is reduced, the MFP-growth provides an effective method. Therefore, the MFP-growth is very appropriate for high-performance computing.

4.3 EXP 3: memory consumption evaluation

This experiment illustrates the performance evaluation of six frequent itemset mining algorithms on the zoo dataset. Zoo is a dense dataset that contains a lot of common items. The algorithms have been compared based on their memory usage for

Table 6 Dataset description

Dataset	No. of transactions	Items	Average transaction size	Type
Zoo	101	36	17	Dense

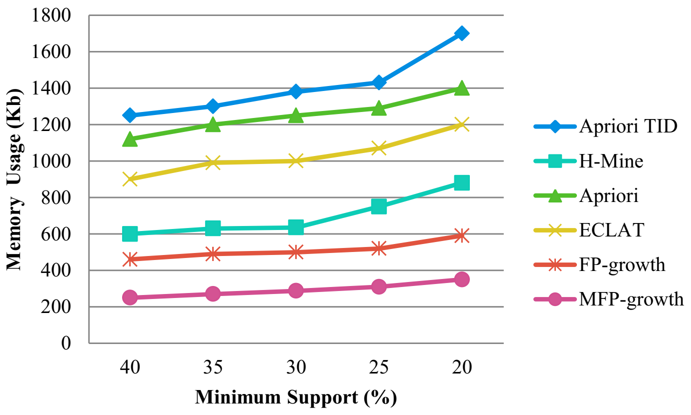


Fig. 8 Performance of memory usage on a dense dataset

discovering the set of all valid association rules [45]. The entire information of the zoo dataset is shown in Table 6. The results obtained are shown in Fig. 8.

The graph longitudinal axis shows the memory in MB, and the latitudinal axis shows the different support threshold values. The charts portrayed in Fig. 8 show the memory consumption of the MFP-growth, Apriori TID, H-Mine, Apriori, ECLAT, and FP-growth algorithms. The support threshold varies from 20 to 40% to record the maximum utilization of memory at different counts of support. Since zoo is a dense dataset and it contains a lot of frequent items, the size of the constructed MFP-tree will be significantly lower due to the overlapping of frequent items. Moreover, the proposed algorithm eliminates the generation of conditional pattern base and conditional subtrees. This eliminates the need for space to store conditional patterns. Therefore, the experimental results proved the superiority of the proposed MFP-growth algorithm in terms of memory consumption. The FP-growth is the next best in memory usage due to the compact and small FP-trees construction. On the other hand, Apriori, and Apriori TID are the most expensive algorithms in terms of memory usage due to the storage of the huge number of candidate generations during each pass and multiple database scans.

4.4 EXP 4: number of generated association rules

This study investigates the association rules for social security incidents. We compare the proposed MFP-growth algorithm with the PSOPF-growth [21], the standard Apriori, and the FP-growth algorithms to confirm the superiority of

Table 7 Comparison of the number of rules that are produced with the conventional algorithm

Support	Apriori	FP-growth	PSOFP-growth	MFP-growth
20	2150	2030	1480	1451
40	1385	1289	855	941
50	898	975	558	451
80	350	367	105	0

our algorithm and analyze the output association rules. Table 7 shows that the association rules explored by the MFP-growth algorithm are approximately 29% lower than that of the Apriori and FP-growth algorithms by adjusting the present support value and using filter data to the association rule that satisfies the specific requirements. Some irrelevant rules were removed that reduced the memory space requirements, making it easy to find the rules we need from massive datasets. These results showed that the modified algorithm could construct a new set of rules with high confidence and improve efficiency, indicating the significant effect of using a data filter on the output.

The experiments were conducted with the adapted new algorithm MFP-growth, PSOFP-growth, Apriori, and FP-growth algorithms. The runtime comparison of the algorithms is presented in Fig. 9. The MFP-growth algorithm requires even less time than the PSOFP-growth, the Apriori, and FP-growth traditional algorithms. As shown in the figure, the greater the support is set, the lesser the execution time of the algorithm, indicating the significant impact of support levels on the algorithm performance. These results showed that, in terms of time cost, the MFP-growth, using the MFP-tree is optimal for the ARM over the PSOFP-growth, Apriori, and FP-growth algorithm.

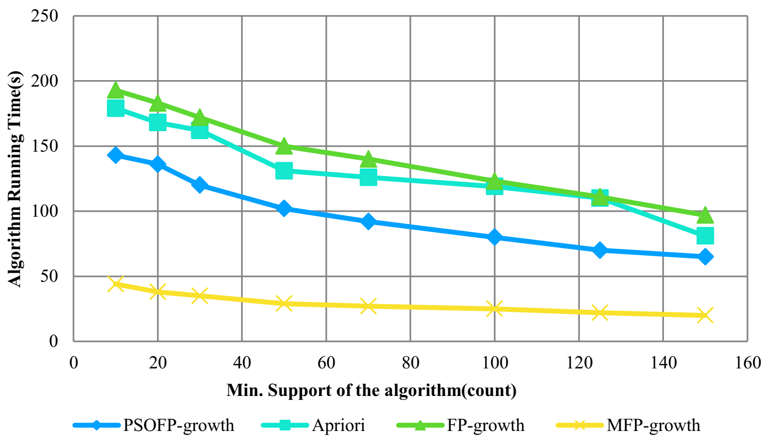


Fig. 9 Comparison of runtime with different minimum support

5 Conclusion

This paper presented a new scheme to discover associations from a wide range of relations across the dataset. Besides, we developed an improved MFP-growth algorithm that discovers frequent associations is developed by combining the FP-tree* mining method and the header table of the FP-growth. The major advantages of the MFP-growth and MFP-tree are that they eliminate the requirement for reconstructing conditional pattern base, subtrees, and simplify the function of tree construction. Extensive workflow experiments through different types of datasets showed that the MFP-growth significantly improves the recent mining speed FIM algorithms and the association rules generated with various support levels and confidence thresholds. The first experiment was evaluated on a set of 12 different UCI datasets and compared its performance with the CBA and APR algorithms. To further evaluate the MFP-growth efficiency, the second experiment was compared with four recent frequent itemset mining algorithms: PAPT-growth, PFP, YAFIM, and DFIMA. The third experiment describes the association rules that satisfy the degree of interest and the runtime of the enhanced MFP-growth algorithm, PSOFP-growth, Apriori, and FP-growth algorithms to the ARM of social security incidents. Different support levels for each dataset were used. Finally, data filters were also used to prove the effectiveness of association rules, facilitating the collection of productive knowledge. The findings confirmed the MFP-growth efficiency and ensured that it is a suitable algorithm for dense datasets. However, MFP-growth can also have the same constraint on efficiency as FP-growth because both need a prefix-tree to state transaction information. The MFP-tree cannot be stored on one node in a transaction database greater than a certain size. Although the information is in a compact form in the MFP-tree structure, long-running times may render it impractical for time-critical applications. Thus, when the database is very large, the algorithm may not fit in the shared memory.

It is our future interest to further calculate the accuracy and performance evaluation by applying the modified algorithm on other real-life datasets with a massive size. Moreover, association rule discovery can be a powerful weapon in fighting pandemics when it is utilized in data mining to develop sciences and to provide new insights into health care.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s11227-021-04066-y.

References

1. Fisch D, Kalkowski E, Sick B (2014) Knowledge fusion for probabilistic generative classifiers with data mining applications. IEEE Trans Knowl Data Eng 26(3):652–666
2. Ceglar A, Roddick JF (2006) Association mining. ACM Comput Surv 38:5
3. Han X, Liu X, Chen J, Lai G, Gao H, Li J (2019) Efficiently mining frequent itemsets on massive data. IEEE Access 7:31409–31421

4. Coenen F, Leng P, Ahmed S (2004) Data structure for association rule mining: T-trees and P-trees. IEEE Trans Knowl Data Eng 16(6):774–778
5. Han J, Fu Y (1999) Mining multiple-level association rules in large databases. IEEE Transact Knowl Data Eng 11(5):798–805
6. Son LH, Chiclana F, Kumar R, Mittal M, Khari M, Chatterjee JM, Baik SW (2018) ARM–AMO: An efficient association rule mining algorithm based on animal migration optimization. Knowl Based Syst 154:68–80
7. Li T-Y, Li X-M (2011) Preprocessing expert system for mining association rules in telecommunication networks. Expert Syst Appl 38:1709–1715. https://doi.org/10.1016/j.eswa.2010.07.096
8. Yildirim P, Birant D, Alpyildis T (2017) Discovering the relationships between yarn and fabric properties using association rule mining. Turk J Elect Eng Comput Sci 25:4788–4804. https://doi.org/10.3906/elk-1611-16
9. Zhang T (2018) Automatic evaluation model of physical education based on association rules algorithm. Wirel Pers Commun. https://doi.org/10.1007/s11277-018-5304-6
10. Khedr AM, Osamy W, Salim A, Abbas S (2020) A novel association rule-based data mining approach for Internet of Things based wireless sensor networks. IEEE Access 8:151574–151588. https://doi.org/10.1109/ACCESS.2020.3017488
11. Viger F, Lin JCW, Vo B, Chi TT, Zhang J, Le HB (2017) A survey of itemset mining. WIREs Data Mining Knowl Discovery. https://doi.org/10.1002/widm.1207
12. Sinthuja M, Puviarasan N, Arun P (2019) Comparative analysis of association rule mining algorithms in mining frequent patterns. Int J Adv Comput Res 8:1839–1846
13. Agrawal R, Mannila H, Srikanth R, Toivonen H, Verkamo AI (1996) Fast discovery of association rules. In: Fayyad UM, Piatetsky-Shapiro G, Smyth P, Uthurusamy R (Eds.) Advances in knowledge discovery and data mining, pp. 307–328
14. Wu H, Lu Z, Pan L, Xu R, Jiang W (2009) An improved apriori based algorithm for association rules mining. In: Sixth International Conference on Fuzzy Systems and Knowledge Discovery, IEEE, vol. 2, pp. 51–55, 2009, https://doi.org/10.1109/FSKD.2009.193
15. Yabing J (2013) Research of an improved apriori algorithm in data mining association rules. Int J Comput Commun Eng 2(1):25
16. Agrawal R, Srikant R (1994) Fast algorithms for mining association rules. In: Proc. 20th int. conf. very large databases, VLDB, vol. 1215, pp. 487–499
17. Gan W, Lin CW, Chao HC, Zhan J (2017) Data mining in distributed environment: a survey. Wiley Interdiscip Rev Data Mining Knowl Discov 7(6):e1216
18. Abdel-Hamid NB, ElGhamrawy S, El Desouky A, Arafat H (2018) A dynamic spark-based classification framework for imbalanced big data. J Grid Comput 16(4):607–626
19. Han J, Pei J, Yin Y (2000) Mining frequent patterns without candidate generation. In: ACM SIGMOD International Conference on Management of Data, pp. 1–12
20. Zhong R, Wang H (2011) Research of commonly used association rules mining algorithm in data mining. In: Proc. IEEE Inter. Conf. Internet Comput. Inf. Services, Hong Kong, pp. 219–222, Sep. 2011
21. Su T, Xu H, Zhou X (2019) Particle swarm optimization based association rule mining in Big Data environment. IEEE Access. https://doi.org/10.1109/ACCESS.2019.2951195
22. Zaki MJ (1997) Fast mining of sequential patterns in very large databases. University of Rochester Computer Science Department, New York
23. Pei J, Han J, Lu H, Nishio S, Tang S, Yang D (2001) H-mine: hyper-structure mining of frequent patterns in large databases. In Data Mining. In: Proc.s IEEE Inter. Conf., IEEE, pp. 441–448
24. Borgelt C (2005) An implementation of the FP-growth algorithm. In: Proceedings of the 1st International Workshop on Open Source Data Mining: Frequent Pattern Mining Implementations, ACM
25. Grahne G, Zhu J (2005) Fast algorithms for frequent itemset mining using FP-trees. IEEE Trans Knowl Data Eng 17(10):1347–1362. https://doi.org/10.1109/TKDE.2005.166
26. Ke-Chung L, Liao IE, Sheng C (2011) An improved frequent pattern growth method for mining association rules. Expert Syst Appl 38(5):5154
27. Tanbeer S, Farhan A, Jeong B, Lee Y (2008) Efficient single-pass frequent pattern mining using a prefix-tree. Inf Sci 179:559–583
28. Liu L, Li E (2007) Optimization of frequent itemset mining on multiple-core processor. In: International Conference on Very Large Databases, University of Vienna, Austria, pp.1275–1285
29. Dean J, Ghemawat S (2004) MapReduce: simplified data processing on large clusters. In: Conference on Symposium on Operating Systems Design and Implementation

30. Li H, Wang Y, Zhang D, Zhang M, Chang EY (2009) PFP: parallel FP-growth for query recommendation. In: ACM Conference on Recommender Systems, pp. 107–114
31. El-Elshafeiy E, El-desouky A (2017) A Big Data framework for mining sensor data using hadoop. Stud Inf Control 26(3):365–376
32. Zhou S, He J, Yang H, Chen D, Zhang R (2020) Big Data-driven abnormal behavior detection in healthcare based on association rules. IEEE Access 8:129002–129011. https://doi.org/10.1109/ACCESS.2020.3009006
33. Apache. Apache spark repository, 2016.
34. Qiu H, Gu R, Yuan C, Huang, Y (2014) YAFIM: a parallel frequent itemset mining algorithm with spark. In: Parallel and Distributed Processing Symposium Workshops, pp. 1664–1671
35. Zhang F, Liu M, Gui F, Shen W, Shami A, Ma Y (2015) A distributed frequent itemset mining algorithm using spark for big data analytics. Clust Comput 18(4):1493–1501
36. Niu X, Qian M, Wu C, Hou A (2019) On a parallel spark workflow for frequent itemset mining based on array prefix-tree," IEEE/ACM Workflows in Support of Large-Scale Science (WORKS), Denver, CO, USA, pp. 50-59, 2019
37. Ma BLWH, Liu B (1998) Integrating classification and association rule mining," in Proc. 4th KDD, pp. 80–86
38. Rajab KD (2019) New associative classification method based on rule pruning for classification of datasets. IEEE Access 7:157783
39. Sornalakshmi M, Balamurali S, Venkatesulu M et al (2020) Hybrid method for mining rules based on enhanced Apriori algorithm with sequential minimal optimization in healthcare industry. Neural Comput Applic. https://doi.org/10.1007/s00521-020-04862-2
40. Thurachon W, Kreesuradej W (2021) Incremental association rule mining with a fast incremental updating frequent pattern growth algorithm. IEEE Access 9:55726–55741. https://doi.org/10.1109/ACCESS.2021.3071777
41. Cheng H, Han J (2009) Pattern-growth methods. In: Liu L, Özsu MT (eds) Encyclopedia of database systems. Springer, Boston
42. Weka Data Mining Tool, (1999), http:// www.cs.waikato.ac.nz/ml/weka
43. UCI.Ucimachinelearningrepository, (2013)
44. Goethals B, Zaki M (2004) Advances in frequent itemset mining implementations: Report on FIMI'03," SIGKDD Explorations, pp. 109–117
45. Borah A, Nath B (2021) Comparative evaluation of pattern mining techniques: an empirical study. Complex Intell. Syst. 7:589–619
46. ElGhamrawy SM (2016) A knowledge management framework for imbalanced data using frequent pattern mining based on bloom filter. 2016 11th International Conference on Computer Engineering & Systems (ICCES), IEEE, 2016
47. Hassib EM, El-Desouky A, El-Kenawy S, El-Ghamrawy S (2019) An imbalanced big data mining framework for improving optimization algorithms performance. IEEE Access 7:170774–170795

Authors and Affiliations

Mai Shawkat[1] · Mahmoud Badawi[2,3] · Sally El-ghamrawy[4] · Reham Arnous[5] · Ali El-desoky[3]

Mai Shawkat
shawkatmai@gmail.com

Mahmoud Badawi
engbadawy@mans.edu.eg

[1] Communications and Information Engineering Department, Faculty of Engineering, Mansoura University, Mansoura, Egypt

[2] Department of Computer Science and Informatics, Taibah University, Medina, Saudi Arabia

[3] Department of Computer Engineering and Systems, Faculty of Engineering, Mansoura University, Mansoura, Egypt

[4] Computer Engineering Department, MISR Higher Institute for Engineering and Technology, Scientific Research Group in Egypt (SRGE), Mansoura, Egypt

[5] Computer and Systems Engineering Department, Delta Higher Institute for Engineering and Technology (DHIET), Mansoura, Egypt