Tree based frequent and rare pattern mining techniques: a comprehensive structural and empirical analysis

Anindita Borah¹ · Bhabesh Nath¹

© Springer Nature Switzerland AG 2019

Abstract

Pattern mining has emerged as a compelling field of research over the years due to its extensive popularity in several application domains. Researchers in this field have contributed ample endeavors, encompassing different issues encountered during pattern generation. The patterns are either extracted based on a level-wise exploration of the search space or by employing efficient tree data structures. Tree based approaches show remarkable performance over the former ones on various grounds and are therefore regarded as the most prominent techniques of pattern mining. A precise and impartial analysis of these eminent techniques is necessary to widen the scope of effective pattern mining using the notion of tree data structures. This paper is therefore an attempt to provide a comparative scrutiny of the fundamental tree based pattern mining techniques through performance analysis based on several decisive parameters. The paper provides a structural classification of the widely referenced tree based pattern mining techniques in four categories and an analytical comparison of these techniques using benchmark real and synthetic datasets. Through this empirical study, an endeavor has been made to enable the researchers identify the factors affecting the performance of the most well-known techniques of pattern mining: the tree based approaches.

Keywords Performance analysis · Pattern mining · Tree based approach · Empirical study

1 Introduction

Enormous quantity of data generated by organizations, emphasize on the discovery of valuable and significant information that led to the emergence of the field of data mining. Pattern mining has entrenched itself as the most indispensable area of data mining research and aims to determine the set of patterns (itemsets), disclosing significant information from databases. Since its inception, a considerable amount of research has been carried out in the field of pattern mining targeting different kinds of patterns as well as the issues and challenges faced during their extraction. Pattern mining is the initial phase of association rule mining with an aim to extract significant patterns and finally generating meaningful association rules from those patterns. These patterns and rules serve

different applications ranging from fraud detection, medical diagnosis, web mining to customer transaction analysis. Pattern mining techniques generate different types of patterns depending upon the requirements of the user. A wide range of pattern mining techniques are available in the literature covering different aspects of data mining applications.

The pioneer attempt for pattern mining initiated with the notion of frequent pattern extraction. A pattern is said to be frequent if its support value is equal to or higher than the user-defined support threshold. The usefulness and applicability of frequent patterns was first identified by Agrawal et al. [2] who managed to generate the desired patterns using a level-wise search strategy. Despite being the most straightforward and uncomplicated technique of frequent pattern generation, their algorithm named

✉ Anindita Borah, anindita01.borah@gmail.com | ¹Department of Computer Science and Engineering, Tezpur University, Tezpur, Assam, India.

Apriori failed to impress on several grounds. The algorithm performs multiple database scans and generates huge number of candidate itemsets. Both these factors affect the execution time and memory usage of the algorithm to a great extent. An efficient alternative for *Apriori* algorithm was developed by Han et al. [21], with a view to overcome its shortcomings. For efficiently generating the set of frequent patterns, they stored entire information of the database in a compact tree-like structure called *Frequent Pattern Tree* (FP-Tree). Instead of scanning the database time and again for frequent pattern generation, their algorithm called *FP-Growth* generates the complete set of frequent patterns in mere two database scans. *FP-Growth* achieved remarkable improvement in terms of execution time and memory usage over the former technique of frequent pattern mining. Keeping in view the popularity and significance of *FP-Growth*, several other tree based approaches were proposed, encompassing different issues pertaining to pattern mining. A literature review has been carried out on the various tree-based pattern mining techniques

that were developed to resolve different pattern mining issues over the years. The popularity and applicability of this eminent category of pattern mining techniques is evident from Fig. 1 that illustrates the number of tree based approaches developed over the years encompassing different pattern mining issues. Figure 1a presents a year-wise distribution of the tree-based approaches developed from the year 2000 to 2016. Figure 1b illustrates the number of tree based approaches developed to handle different pattern mining issues. After establishing itself as a compelling and fruitful approach of pattern mining for over a decade, there is a demand for an overview and re-examination of the various tree based techniques developed and let the researchers identify their pros and cons, in order to develop efficient techniques for pattern generation.

Even though a large number of theoretical as well as empirical reviews in the field of pattern mining can be found in the literature, no initiative has been taken to carry out a comparative evaluation of the tree based pattern

Fig. 1 Tree based approaches developed for solving pattern mining issues

(a) Number of articles published on tree based approaches

(b) Number of tree based and level-wise approaches developed for pattern mining issues

mining techniques generating different categories of patterns or itemsets using experimental validation [17, 36, 40]. This study offers an unbiased empirical study of the fundamental tree based techniques in the area of pattern mining, generating different categories of itemsets. To the best of our knowledge, till now no attempt has been made to perform an empirical study and experimental evaluation of the tree based pattern mining techniques developed under various constraints using different thresholds and datasets. The primary aim of this study is to perform a detailed analysis of the existing tree based pattern mining techniques widely referenced in the literature, under the following four themes: (1) frequent pattern generation, (2) closed frequent pattern generation, (3) maximal frequent pattern generation and (4) rare pattern generation. The major contributions of this paper includes:

- Evaluation of the most widely referenced tree based techniques generating different types of patterns using various support thresholds.
- Comparative analysis of the tree based techniques with level-wise approaches through experimental evaluation on a wide range of real and synthetic benchmark datasets.
- Analysis of the factors affecting the performance of tree based approaches using several synthetic datasets.

The remaining paper is organized as follows: Sect. 2 followed by introduction discusses the existing tree based pattern mining approaches in the literature. The performance analysis of some prominent tree based approaches extracting different patterns is provided in Sect. 3. Section 4 elucidates various factors that affect the performance of tree based approaches. Discussion and explanation on the results obtained from the experimental evaluation is provided in Sect. 5 followed by conclusion of this study in Sect. 6.

2 Literature review

Since its inception, a wide range of pattern mining techniques were developed by the researchers [6]. The pattern mining techniques emphasized on the extraction of different types of patterns based on the requirement of the user and the application domains.

The researchers initially concentrated on the extraction of frequent patterns. The patterns that appear frequently together in a database are considered to be interesting for many application domains. All those patterns are considered to be frequent whose *support* value lies above the user defined minimum *support* value. The *support* value of

a pattern is the percentage of transactions in the transaction database that consist of that particular pattern.

A major issue faced by frequent pattern mining algorithms is the generation of innumerable number of frequent patterns. Research on pattern mining identified a significant solution to solve this issue in the form of *closed frequent itemsets* (CFI) that possess the same potential as that of the large set of frequent itemsets generated [34, 38]. The CFI's despite having a smaller magnitude than the larger set of frequent itemsets, are capable of identifying the explicit frequencies of all the itemsets.

Extracting the entire set of frequent itemsets is a computationally challenging as well as extravagant phase of pattern mining. Research on pattern mining illustrates that subset of every frequent pattern must also be frequent. Thus, to overcome the drawback of huge number of frequent pattern generation, extraction of only the *maximal frequent pattern* is sufficient [4, 18]. A pattern is considered to be a *maximal frequent pattern* only if none of its supersets are frequent.

The paradigm of frequent pattern mining considers the rare patterns to be of least importance, demanding their removal during the phase of pattern generation. Of late, it has been identified that the rare patterns are significant for many application domains [7, 8, 10–12]. A pattern is considered to be rare if its *support* value lies below the pre-defined *support* value.

The pattern mining algorithms can be broadly classified into three different categories:

2.1 Join-based algorithms

Join-based algorithms make use of joins to generate $k + 1$ candidates from the k frequent itemsets generated. The primer algorithm under this category was introduced by Agrawal et al. [3]. To handle the vast quantity of itemsets produced and to generate only the fruitful frequent itemsets, their *Apriori* algorithm adopts a candidate generation approach in which the frequent 1-itemsets are initially generated by scanning the database and then proceeding towards the generation of candidate 2-itemsets from these frequent 1-itemsets. The same process of frequent itemset and candidate generation continues until no further frequent n -itemsets can be generated for some particular item n . Zaki et al. [39] developed the Apriori-like *Equivalence Class Transformation* (ECLAT) algorithm that employs a depth first search strategy for the generation of frequent itemsets. It is based on the property of set intersection and is capable of performing sequential as well as parallel execution. ECLAT operates on vertical data format and generates the TID set for each item. The intersection of the TID sets of current n -itemsets is used to find the TID sets of the next $n + 1$ -itemsets. The algorithm avoids rescanning of

the database for support counting of the $n + 1$ -itemsets, as the TID sets of n -itemsets contains the entire information necessary for calculating their support values. Similar to *Apriori*, *ECLAT* also suffers from the drawback of generating too many candidate itemsets. In spite of being the primer and one of the simplest techniques, level-wise search and huge number of candidate itemsets produced adds to the complexity of the algorithm. The performance of the algorithm is thus affected in terms of execution time and memory usage due to continuous generation of candidate itemsets.

For the purpose of generating the closed frequent itemsets, *AprioriClose* [26] initially generates a set of generators based on the closure properties of itemsets employing a level-wise search procedure. The support counts of all the generators are then obtained to remove the unwanted generators. The algorithm is based on the principle of pruning the closed itemset lattice instead of the itemset lattice. The closure of the generators are computed during the first iteration of the algorithm. The closed itemset lattice is determined using the mechanism based on *Galois Connection*. Since the closed itemset lattice is a sub-order of the itemset lattice, the number of itemsets considered will be considerably reduced. *Maximal Frequent Itemset Algorithm (MAFIA)* [14], on the other hand, employs a depth first search approach to generate the maximal frequent itemsets (MFI)'s. It obtains the frequency count of the itemsets based on a bitmap representation. The columns of the vertical bitmap represent the count of the itemsets. A bit vector and operation is applied on each bit vectors of individual items to obtain the bit vector for the entire itemset. The algorithm prunes the subsets and supersets based on their frequency counts. *MAFIA* is able to generate all the supersets of MFI's but does not perform well with datasets that have long average pattern length.

Szathmary et al. [31] developed *Another Rare Itemset Mining Algorithm (ARIMA)*, targeting the complete set of rare items. *ARIMA* employs an *Apriori*-like level-wise approach and works as combination of two algorithms for the generation of rare itemsets. To obtain the support count of a given itemset, it employs a subroutine. The algorithm produces all the minimal rare itemsets i.e the itemsets that are rare by themselves but have frequent proper subsets. However, it cannot find all the rare itemsets. To minimize the search space, it removes the zero generators and generates the rare itemsets. However, it spends a lot of execution time in the generation of rare itemsets. *Rarity* algorithm proposed by Troiano et al. [32] uses a level-wise search like *Apriori*. But unlike *Apriori* algorithm that performs a bottom up search for finding the frequent itemsets, *Rarity* follows a top down strategy. The algorithm first identifies the longest transaction and searches the rare itemsets in a top down manner. It do not

take into account the lower layers where frequent itemsets are present. The candidate items which are frequent are removed using the downward closure property and only those items are considered whose support count is below the threshold. The algorithm has a disadvantage that it may suffer from shortage of memory which relies on the dimension of itemsets and their number.

2.2 Tree-based algorithms

Tree-based algorithms are based on the concepts of set enumeration and assumes a lexicographic tree of candidate itemsets that is explored using traversal strategies like depth first search or breadth first search. To identify the candidate itemsets, a subgraph of the lattice of itemsets called lexicographic tree or enumeration tree is generated. Tree-based algorithms are based on the assumption that a lexicographic order exists among the items in a database and the lexicographic tree is constructed in a way that it provides a certain order of exploration.

The AIS algorithm developed by Agrawal et al. [2] was the first lexicographic tree algorithm. The frequency of the items at a particular level are counted using the transaction database and the lexicographic tree is constructed in a level-wise manner. The algorithm does not use any optimization to explore the search space of candidate itemsets and is not very efficient in counting the itemsets.

2.3 Tree projection-based algorithms

Tree projection-based algorithms employ the concept of recursive projection of transaction database. Database projection reduces the effort for itemset counting by restricting the size of the database used for support counting. The tree projection-based approaches are the most proficient one and have skillfully attempted to resolve the issues faced by the pattern mining community [9, 13].

FP-Growth [21] is the first frequent pattern mining technique that uses a recursive suffix-based approach for search space exploration. The algorithm generates the complete set of frequent patterns in two database scans. During first scan, the algorithm identifies the set of frequent items, retains those items and remove the rare ones. The tree structure called *FP-Tree* is constructed during the second scan which is a compact representation of the original database containing only frequent items. The desired set of frequent patterns are then extracted from *FP-Tree* without referring the original database time and again. This gave the algorithm huge advantage over level-wise approaches in terms of execution time and number of database scans. The memory space consumed is also less compared to previous approaches as it avoids the generation of candidate itemsets. To further improve the

space usage during frequent pattern generation, Sucahyo and Gopalan [30] developed the *CT-PRO* algorithm that makes use of a tree structure called *Compressed FP-Tree* (*CFP-Tree*). *CFP-Tree* is more compact and consists of lesser nodes than *FP-Tree*. In order to compress the *FP-Tree*, the identical subtrees are removed from the complete *FP-Tree* and the corresponding information is stored in the remaining nodes of the tree. The *CFP-Tree* is then generated by collecting together all the subtrees of the root of the *FP-Tree* in the leftmost branch. The algorithm performs better than *FP-Growth* in case of sparse datasets. Another variant of *FP-Growth* called *Improved FP-Growth* (*IFP-Growth*) was proposed in [25] to improve its efficiency. The algorithm uses two data structures: an address table and a tree structure called *FP-Tree*⁺ to avoid the generation of conditional *FP-Trees*. The algorithm uses a top-down traversal strategy to identify the set of frequent itemsets. Usage of two data structures and the avoidance of conditional tree building reduces the memory overhead of the algorithm to a great extent. Moreover, the construction of *FP-Tree*⁺ requires lesser time than that of *FP-Tree*.

Among the *closed frequent itemset* generation techniques, Grahne and Zhu [19] developed the *FPClose* algorithm that employs a tree structure called *Close Frequent Itemset Tree* (*CFI-Tree*) for storing the *closed frequent itemset*. The algorithm uses multiple conditional *CFI-Trees* instead of a single global prefix-tree to reduce the search time. *FPClose* performs efficiently due to the usage of array based implementation and compact tree structure. The array technique works well specifically for sparse datasets. *FP-Tree* generated for sparse datasets will be big and bushy. The array implementation saves time and is faster than other *FP-Tree* based implementations. Another tree based approach called *CLOSET* was developed by Pei et al. [27] for efficiently generating the *closed frequent itemsets* from large datasets. The algorithm employs a single large global prefix tree for searching the closed itemsets. A CFI is inserted into the tree by replacing the count of the node with the maximal updated count instead of incrementing it by one. *CLOSET* algorithm was further extended in their *CLOSET*⁺ algorithm [35] for improvement in execution by fast exploration of the tree structure.

For generating the *maximal frequent itemsets*, *FPMMax* algorithm developed by Grahne and Zhu [20] intends to generate the *maximal frequent itemset* with the help of a tree structure called *Maximal Frequent Itemset Tree* (*MFI-Tree*) that stores the desired MFI's. *FPMMax* employs the concept of depth first search and insert an identified frequent itemset into the *MFI-Tree* only if it is a subset of an itemset already present in the tree. The algorithm removes the non-maximal frequent itemsets from the supersets of frequent itemsets. *FPMMax* works well with large datasets and datasets having short average transaction length. Another

tree based approach called *Frequent Pattern Tree for Maximal Frequent Itemsets* (*FPMFI*) was developed by Yan et al. [37] that employs a projection based superset checking technique to find out the MFI's.

The primer tree based approach towards the extraction of rare itemsets called *Rare Pattern Tree* (*RP-Tree*), was developed by Tsang et al. [33]. For rare pattern generation, the algorithm considers only those transactions that contain at least one rare item. The reduced database is then used to build the rare pattern tree. The algorithm generates the conditional rare pattern tree similar to the *FP-Growth* algorithm. *RP-Tree* is faster than other rare pattern mining techniques that uses a level-wise approach to identify the rare itemsets due to the usage of the compact tree structure and reduced database scans. It however, managed to extract only a subset of the rare itemsets called *rare-item itemsets*. Bhatt and Patel [5] later modified the *RP-Tree* algorithm for enhancing its efficiency, using multiple threshold values. Their algorithm called *Maximum Constraint Based Rare Pattern Tree* (*MCRP*) employs a maximum constraint model for assigning individual support values to each item. Similar to *RP-Tree*, it considers only the transactions containing at least one rare item and performs only one database scan for tree construction. For finding the rare items, an *MIS* value is attached to each item and only those items are regarded as rare, support count of which is lower than the minimum frequent support but higher than its corresponding *MIS* value.

3 Performance analysis of tree based approaches

Even though tree based approaches have been identified as the most compelling category of pattern mining techniques, they still suffer from several drawbacks that hinders their performance. An in-depth scrutiny of the bottlenecks of tree based approaches is therefore necessary to enhance their performance for efficiently handling different pattern mining challenges. This section provides an extensive analysis of the tree based approaches generating different categories of patterns. For comparative analysis some widely referenced level-wise approaches have also been included in the experimental study. The criteria taken into consideration for measuring the effectiveness of tree based approaches is the respective execution time and memory space invested by the algorithms in pattern generation along with compactness of the tree data structure. The execution time considered for each algorithm is the average of multiple runs. It is to be noted that the reported execution time of algorithms is the total execution time, that is CPU, I/Os, tree construction and pattern mining.

Compactness is evaluated based on the reduction ratio achieved and amount of memory consumed by the tree structure. Reduction ratio is the ratio of total number of items present in the database to corresponding number of nodes generated. For rare pattern generation two support thresholds have been used. The minimum frequent support threshold has been considered to be 30% and minimum rare support threshold has been taken as 1%.

An extensive comparative analysis has been carried out on eighteen fundamental algorithms using eleven real and four synthetic benchmark datasets. Table 1 illustrates the characteristics of the datasets used for experimentation. For simplicity and more clarity, the graphs depicting the corresponding results were divided into four different categories. The first category contains five algorithms generating only the frequent patterns namely *Apriori*, *FP-Growth*, *CT-PRO*, *ECLAT* and *IFP-Growth*. The second category includes algorithms generating closed frequent patterns: *AprioriClose*, *FPClose*, *CLOSET* and *CLOSET+*. Four algorithms generating maximal frequent patterns namely *MAFIA*, *FPMMax*, *FPMMax** and *FPMFI* constitute the third category. The fourth category on the other hand, includes algorithms like *ARIMA*, *RP-Tree*, *RP-Tree-IG*, *MCRP* and *Rarity* that generates rare patterns. Some publicly available Java implementations of certain standard algorithms like *Apriori*, *ECLAT*, *FP-Growth*, *FPClose*, *AprioriClose*, *MAFIA* and *FPMMax* have been used for this study. Rest of the algorithms have been implemented in Java on a 64-bit machine of 4 GB RAM.

3.1 Frequent pattern generation

This section provides analysis of the tree based approaches that emphasize on the generation of frequent patterns. For experimental analysis, we considered three widely referenced tree based approaches in the area of frequent pattern mining: *FP-Growth*, *CFP-Growth* and *IFP-Growth*. For comparison, frequent pattern mining level-wise approaches *Apriori* and *ECLAT* have been considered. Figure 2 illustrates the performance analysis of the tree-based approaches with respect to level-wise approaches for generating frequent patterns.

(a) Execution Time

(b) Memory Usage

Table 1 Datasets used

Datasets	Number of transactions	Average transaction size	Category	Type
Zoo	101	17	Real	Dense
Hepatitis	137	35	Real	Sparse
Lymph	148	18	Real	Sparse
Tumor	339	17	Real	Dense
Soyabean	631	17	Real	Dense
Anneal	812	43	Real	Dense
Hypothyroid	3247	44	Real	Dense
Mushroom	8124	23	Real	Dense
Connect-4	67,557	43	Real	Dense
Gazelle	59,602	2.5	Real	Sparse
Retail	88,162	11.31	Real	Sparse
T10I20D10K	10,000	10	Synthetic	Dense
T25I10D10K	10,000	10	Synthetic	Sparse
T10I4D100K	100,000	3.7	Synthetic	Sparse
T40I10D100K	100,000	8.5	Synthetic	Sparse

Fig. 2 Performance analysis of frequent pattern mining tree based approaches

The performance of tree based approaches has been better than level-wise approaches in case of dense datasets. However, in case of sparse datasets level-wise approaches *Apriori* and *ECLAT* outperformed the tree based approaches. In terms of memory usage, tree based approaches displayed outstanding performance in case of dense datasets. Due to the presence of too many frequent items in dense dataset, the size of generated FP-tree will be very less due to the sharing of common prefixes. *Apriori* and *ECLAT* proved little expensive in these datasets due to the storage of candidate itemsets generated during each phase of itemset generation. For sparse datasets, the amount of node sharing is less due to which large number of nodes are generated in the tree structures. Storage of these nodes in main memory proved to be costly for tree based approaches in terms of memory usage than level-wise approaches.

Among the frequent pattern mining tree based approaches, *FP-Growth* has been found to consume the maximum amount of execution time for pattern generation. *CT-PRO* invested lesser time than *FP-Growth* as it avoids the generation of conditional pattern bases and conditional pattern trees during pattern mining. *FP-Growth* generates conditional FP-tree during each step of mining and recursively extracts the frequent patterns from the conditional pattern trees. *CT-PRO* on the other hand, generates a local *CFP-Tree* for each item and performs a non-recursive traversal for frequent pattern generation. *IFP-Growth* algorithm outperformed the previous two algorithms in terms of execution time. Similar to *CT-PRO*, *IFP-Growth* too prevents the recursive generation of conditional FP-trees. *IFP-Growth* employs an additional data structure called *address-table* to reduce the overhead of FP-Tree construction. The *address-table* stores the items along with pointers that points to their corresponding child nodes. Thus, the time spent in searching the child nodes in FP-Tree and CFP-Tree is considerably reduced in case of *IFP-Growth*.

The tree data structure *CFP-Tree* used by *CT-PRO*, is much more compact than *FP-Tree* due to which it generates almost half the nodes generated by *FP-Tree*. *CT-PRO* therefore, has been found to consume less amount of memory space than *FP-Growth*. *IFP-Growth* managed to perform better than both the algorithm in terms of memory usage as well. The usage of more compact data structures like *address-table* and *FP-Tree** further reduces the memory consumption.

To analyze the compactness of tree structures used by the frequent pattern mining techniques, reduction ratio achieved by each tree structure on different datasets has been evaluated. Figure 3 elicits the compactness achieved by the frequent pattern mining tree structures. *CFP-Tree* used by *CT-PRO* algorithm generates considerably lesser

Fig. 3 Compactness of frequent pattern mining tree structure

number of nodes than *FP-Tree* due to which its reduction ratio is higher than *FP-Tree*. However, *IFP-Tree* employed by *IFP-Growth* was the clear winner as it has produced almost half the nodes generated by *FP-Tree* and comparably lesser number of nodes than *CFP-Tree*. Thus, *IFP-Tree* achieved the maximum amount of compression compared to the other two tree data structures. Despite, their outstanding performance in dense datasets, the tree data structures could not achieve good compression in case of sparse datasets.

3.2 Closed frequent pattern generation

This section elucidates the comparative performance analysis of tree based and level-wise approaches that generates the closed frequent patterns. The tree based algorithms considered in this category are: *FPClose*, *CLOSET* and *CLOSET+*. For comparison with level-wise closed frequent pattern generation technique, *AprioriClose* has been considered. The performance evaluation of the tree based and level-wise closed frequent pattern mining approaches is illustrated in Fig. 4.

For dense datasets, all the tree based approaches performed better than level-wise approach *AprioriClose*. This is due to the sharing of similar nodes in the CFI-Trees, representing closed frequent itemsets. *AprioriClose* on the contrary, generates huge number of candidate itemsets by scanning the database multiple times. *AprioriClose* consumed the maximum amount of memory among the other algorithms for dense datasets. The reason behind this overhead is that *AprioriClose* tends to store each extracted *Minimal Frequent Generator (FMG)* in main memory until it obtains the entire set. Thus the closure computation for each generated *FMG* proves to be expensive. Tree based approaches however failed to impress both in terms

(a) Execution Time

(b) Memory Usage

Fig. 4 Performance analysis of closed frequent pattern mining tree based approaches

of execution time and memory usage in case of sparse datasets.

FPClose performs best in terms of execution time among the tree based closed frequent pattern mining techniques for all datasets. The difference in the execution times of the algorithms lies in the fact that *FPClose* retains only a part of the extracted CFI's in the recursively generated CFI-trees and the subsumption checking cost is also very less compared to *CLOSET* and *CLOSET+*. *CLOSET+* however, managed to become the best algorithm in terms of memory consumption as it does not retain the set of *FMG*'s in main memory. It however performs the closure computation of *FMG*'s during support counting, for which it needs to store the closed itemsets. *FPClose* proved to be

Fig. 5 Compactness of closed frequent pattern mining tree structure

a little expensive in terms of memory consumption. This is due to the fact that it tends to store the previously generated *Frequent Closed Itemsets* (FCI)'s in main memory. Even though, *FPClose* retains only a portion of the extracted FCI's in CFI-Tree but storing multiple CFI-Trees still requires ample amount of main memory.

From the compactness analysis given in Fig. 5, it can be observed that *FPClose* achieved the highest reduction ratio. As illustrated earlier, *FPClose* retains only a portion of the extracted FCI's in CFI-Tree due to which the number of nodes generated is very less compared to the total number of items in the dataset. The CFI-Tree in case of *FPClose* thus achieves higher compression than the tree structures of *CLOSET* and *CLOSET+*. However, the tree structures of all the three algorithms failed to achieve good compression in case of sparse datasets due to less sharing of nodes.

3.3 Maximal frequent pattern generation

The comparative performance analysis of tree based and level-wise maximal frequent pattern mining techniques is illustrated in this section. The tree based algorithms considered for evaluation are: *FPMFI*, *FPMMax* and *FPMMax**. Level-wise maximal frequent pattern mining approach *MAFIA* has been considered for comparative analysis. Figure 6 elucidates the performance of the tree based and level-wise approaches generating maximal frequent patterns.

Among this category of algorithms, the most effective ones in case of dense datasets are the tree based approaches, due to their obvious advantage of projection tree construction. The tree based algorithms will generate a small tree from dense datasets effectively generating

(a) Execution Time

(b) Memory Usage

Fig. 6 Performance analysis of maximal frequent pattern mining tree based approaches

Minimal Frequent Itemsets (MFI)'s from the extracted MFI-Tree. *MAFIA* on the other hand, performs expensive bit-vector operations and set intersections costing it higher execution time and memory for pattern generation. But for sparse datasets, it is still the best algorithm.

*FPMMax** performed better than its predecessor *FPMMax* due to the incorporation of an array based tree traversal technique and maximality checking function. The algorithm also stores the MFI's in a reduced form in the MFI-Tree due to which it achieves higher advantage in terms of execution time and space usage over *FPMMax*. *FPMFI* however, proved to be the best algorithm among the tree based approaches due to its additional feature of

Fig. 7 Compactness of maximal pattern mining tree structure

projection based superset checking that reduces the cost of counting the supersets and the tree traversal time to a great extent. It also has memory advantage over the former ones due to its compressed tree structure generated by removing the redundant itemsets.

Figure 7 elucidates the compactness evaluation of tree structures employed by maximal frequent pattern mining tree based approaches. Due to the removal of redundant itemsets and retaining only the significant maximal frequent itemsets in the tree structure, tree structure of *FPMFI* generates very lesser number of nodes compared to *FPMMax* and *FPMMax**. The reduction ratio achieved by *FPMFI* is therefore higher than the other two approaches.

3.4 Rare pattern generation

This section provides the analysis of tree based approaches with respect to level-wise approaches generating rare patterns. Three tree based rare pattern mining techniques: *RP-Tree*, *RP-Tree-IG* and *MCRP* have been considered for performance evaluation. For comprehensive comparative analysis, two level-wise rare pattern mining approaches: *ARIMA* and *Rarity* have been considered. The performance evaluation of the rare pattern mining tree based and level-wise approaches is elucidated in Fig. 8.

Performance analysis of the fourth category of algorithms illustrates that tree based approaches *RP-Tree*, *RP-Tree-IG* and *MCRP* expends the lowest amount of execution time and memory among others due to the generation of only the rare-item itemsets in both dense and sparse datasets. *ARIMA* proves to be the most expensive one, due to the fact that it spends a lot of time in the execution of *Apriori Rare*, generating the *Minimal Rare Itemsets* (MRI)'s and *MRG-Exp* extracting the *Minimal Rare Generators*

(a) Execution Time

(b) Memory Usage

Fig. 8 Performance analysis of rare pattern mining tree based approaches

(MRG)'s. On the other hand, *RP-Tree* along with its other variants *RP-Tree-IG* and *MCRP* spends less execution time and memory space due to the generation of only a subset of rare itemsets.

MCRP outperformed both *RP-Tree* and *RP-Tree-IG* algorithm in terms of execution time and memory usage. *RP-Tree-IG* spent lesser amount of time and memory than *RP-Tree* due to the retainment of only the significant rare items based on the information gain component. As per the compactness analysis provided in Fig. 9, tree structure employed by *RP-Tree-IG* achieved the highest reduction ratio. This is due to the fact that *RP-Tree-IG* retains the lowest number of rare items and hence generated the lowest number of nodes compared to other two algorithms.

Fig. 9 Compactness of rare pattern mining tree structure

As *MCRP* generates higher number of rare items than *RP-Tree* and *RP-Tree-IG*, its tree structure is less compact than others.

4 Factors affecting the performance of tree based approaches

The performance of tree based approaches is influenced by several parameters. This section provides an illustration of the factors affecting tree based approaches through comprehensive experimental evaluation on a range of datasets. The factors considered for the experimental study are: dimensions of dataset, change of threshold and ordering of items in tree structure. Synthetic datasets have been used for experimental evaluation which were generated using the data generation process described by Agrawal et al. [3].

4.1 Dataset dimensions

The dimensions of dataset affects the performance of tree based approaches to a great extent. This section provides a detailed analysis of the dataset dimensions influencing the performance of tree based approaches. The dataset dimensions considered for evaluation are: average items present per transactions, distinct items present in the database and number of transactions in the database. The experiments were conducted using synthetic datasets varying their dimensions. The notations used for datasets represent their characteristics: T represents average items present per transactions, I denotes the distinct items present in the database and D indicates number of transactions in the database. For instance, T50I100D10K indicates

Fig. 10 Performance affected by distinct items in the database

that there are 50 items per transaction, 100 distinct items and 10,000 transactions in the dataset. For experimental evaluation, we considered the best tree based approach in terms of execution time in all the four categories of pattern generation. *IFP-Growth* from the category of frequent pattern generation, *FPClose* from closed frequent pattern generation, *FPMFI* from maximal frequent pattern generation and *MCRP* from rare pattern generation have been considered for this study.

4.1.1 Distinct items present in the database

Several experiments using synthetic datasets were conducted for determining the affect of distinct items present in the database on the performance of tree based approaches. Table 2 illustrates the characteristics of

Table 2 Characteristics of synthetic datasets used

Dataset	# Transactions	# Items per transaction	# Distinct items
T10I10D1K	1000	10	10
T10I20D1K	1000	10	20
T10I50D1K	1000	10	50
T10I100D1K	1000	10	100
T10I200D1K	1000	10	200
T10I300D1K	1000	10	300

synthetic datasets used for this experimental analysis. For a detailed analysis, number of transactions and number of items present per transaction has been kept constant in every dataset varying only the number of distinct items.

From experimental analysis given in Fig. 10a, b, it has been observed that both execution time and memory usage for pattern mining increases for each algorithm when the number of distinct items in the dataset is increased keeping the other two factors constant. However, for the same setting the reduction ratio or the compactness of tree structure of each of these algorithms get reduced. For instance, it can be seen in Fig. 10c that upon varying the distinct items in the database from 10 to 300, reduction ratio of IFP-Tree reduces from 16.8 to 7.2. In other words, the performance of tree based approaches deteriorates when the number of distinct items increases with rest of the two dataset dimensions being constant.

4.1.2 Average items present per transaction

The next dimension of dataset affecting the performance of tree based approaches is average items present per transaction. The affect of this factor on the performance of tree based approaches is evaluated by varying the average items present per transaction and keeping rest of the two factors constant. Table 3 elucidates the characteristics of synthetic dataset used for evaluating the affect of average items per transactions.

Upon increasing the number of average items present per transaction and keeping the number of distinct items and number of transactions constant, execution time and memory usage of tree based approaches decreases. The performance enhancement of tree based approaches becomes evident from Fig. 11a, b. Similarly, from Fig. 11c, it can be observed that compactness of tree structures employed by tree based approaches increases with increase in the number of average items present per transaction. For example, upon increasing the average items present per transaction from 10 to 250, reduction ratio of IFP-Tree increases from 16.8 to 26.85. Therefore, it can be concluded that the increase in number of average items present per transaction has a positive impact on the performance of tree based approaches.

Table 3 Characteristics of synthetic datasets used

Dataset	# Transactions	# Items per transaction	# Distinct items
T10I10D1K	1000	10	10
T50I10D1K	1000	50	10
T100I10D1K	1000	100	10
T150I10D1K	1000	150	10
T200I10D1K	1000	200	10
T250I10D1K	1000	250	10

4.1.3 Number of transactions present in the database

The number of transactions present in the database is another factor highly influencing the efficiency of tree based approaches. The experiments for analyzing the effect of the same is conducted using synthetic datasets by changing the number of transactions and keeping the rest of the factors constant. Table 4 provides information about the characteristics of dataset used for this set of experiments.

From the set of experiments to visualize the affect of change in number of transactions, it has been identified that the increase in number of transactions of dataset has a positive effect on the performance of tree based approaches. For the same number of distinct items and average items present per transaction, execution time and memory usage of tree based approaches decreases. Execution time and memory space invested by tree based approaches on various synthetic datasets is illustrated in Fig. 12a, b. As shown in Fig. 12c, the tree based approaches achieve performance improvements in terms of compactness of tree structure as well when the number of transaction is increased keeping the other two factors constant.

4.2 Change in support threshold

To analyze the effect of threshold change on the performance of tree based approaches, we conducted experiments on a widely used synthetic dataset, T40I10D100K using different support threshold values. The comparative evaluation has been done based on the average execution time and memory space invested by the algorithms. The minimum frequent support threshold, *minFreq* has been varied from 5 to 30% and the minimum rare support threshold, *minRare* has been kept constant at 1%. The execution time and memory invested by the algorithms on T40I10D100K for different threshold values is illustrated in Fig. 13a, b.

From the figures, it can be observed that the execution time of frequent, maximal frequent and closed frequent pattern generation algorithms decreases with the increase in support threshold. When the value of support threshold is high, then very few items qualify as frequent, maximal frequent or closed frequent due to which the time spent for the generation of these itemsets decreases with increase in threshold. However, the case is reverse for the rare pattern mining algorithms. Algorithms *RP-Tree*, *RP-Tree-IG* and *MCRP* generates the rare itemsets based on two support thresholds. When the frequent support threshold is increased keeping the rare support threshold constant, then more number of items qualify as rare due to which the execution time increases with increase in the value of support threshold. Since, memory usage is

Fig. 11 Performance affected by average items in the database

Table 4 Characteristics of synthetic datasets used

Dataset	# Transactions	# Items per transaction	# Distinct items
T10I10D1K	1000	10	10
T10I10D2K	2000	10	10
T10I10D3K	3000	10	10
T10I10D4K	4000	10	10
T10I10D5K	5000	10	10
T10I10D6K	6000	10	10

highly affected by the size of tree structure and the number of itemsets generated, the amount of memory used by frequent, maximal frequent and closed frequent pattern algorithms decreases with the increase in support

threshold while it increases for the rare pattern mining algorithms.

4.3 Ordering of items in tree structure

The efficiency of a tree based pattern mining algorithms is highly affected by the size of tree structure which in turn is influenced by the ordering of items during tree construction. This section elicits the compactness evaluation of three most relevant frequent pattern mining tree data structures. The tree structures used for experimental analysis employs different item ordering and arrangement during their construction. Most of the pattern mining tree structures follow a frequency descending order of items while few tree structures employ frequency independent ordering of items. Three frequent pattern generation tree structures have been used for comparison: FP-Tree,

Fig. 12 Performance affected by number of transactions in the database

CFP-Tree and CanTree. CanTree is a canonical, frequency independent tree structure developed by Leung et al. [23] that maintains the complete information of a database. For a fair and unbiased comparison, CanTree has been modified to retain only the frequent items of the database. Compactness of tree data structures has been evaluated based on the total amount of memory consumed for storing the entire tree structure. The memory consumption of the algorithms is shown in Fig. 14.

Compactness of a tree structure depends on the degree of prefix sharing and the ordering of items. In frequency descending ordering of items, frequent items lie at the upper region of the tree near the root. Such an arrangement offers more prefix sharing ensuring higher compactness of tree structure. As both FP-Tree and CFP-Tree follows frequency descending order of items during their

construction, both the tree structures achieved almost similar amount of compactness, with CFP-Tree being slightly better than FP-Tree. In case of real and synthetic sparse datasets as shown in Fig. 14b, all the three tree structures consumed almost similar amount of memory. As discussed in earlier sections, sparse data contains very few common transactions that reduces the amount of prefix sharing among nodes when a frequency descending order is followed. A tree structure from such a dataset will therefore, be large and bushy thus consuming lot of memory. However, for dense datasets, FP-Tree and CFP-Tree achieved the maximum compactness as shown in Fig. 14a. The difference in compactness of CanTree with FP-Tree and CFP-tree is more prominent in case of dense datasets. Due to the construction of FP-Tree and CFP-Tree in frequency descending order, it is guaranteed to achieve

Fig. 13 Performance affected by threshold change

Fig. 14 Compactness of tree structure

maximum compactness. Therefore, it can be observed that FP-Tree and CFP-Tree consumed lesser amount of memory for almost all datasets compared to CanTree. CFP-Tree however, performed even better than FP-Tree in both categories of datasets.

5 Discussion

Several issues have been identified from this study that affects the performance of tree based approaches. This section summarizes the observations of this study and

discusses the pros and cons of tree based approaches identified from the experimental evaluation.

From the experimental analysis given in previous sections, it has been observed that the tree based approaches show outstanding performance in case of dense datasets but fails miserably in case of sparse datasets. Density of dataset highly affects the compactness of tree structure and hence performance of tree based approaches. In case of dense datasets, the transactions are very much similar to each other. A tree data structure based algorithm is suitable for dense datasets since many transactions will share common prefixes and henceforth, the database could be

compactly represented. However, in case of sparse data, tree data structures could not achieve good compression. Since sparse data contains lesser number of frequent items, tree data structures will be large and thicker as there will be fewer shared common prefixes.

As illustrated in Sect. 4.1.1, the performance of tree based approaches deteriorates when the number of distinct items increases. This is due to the fact that sparseness of data increases with the increase in number of distinct items. On the other hand, the dataset becomes more dense when the average number of items per transaction and number of transaction is increased. Tree based approaches thus show improvement in terms of execution time, memory usage and compactness of tree structure as shown in Sects. 4.1.2 and 4.1.3. To better comprehend the behavior of tree based approaches with respect to dataset characteristics, let us consider an example.

Figure 15a represents a dense dataset where there is a dense distribution of items. The dataset contains many common items appearing frequently. From the resultant tree given in Fig. 15b, it can be observed that many common items represented by nodes are being shared. For instance, there is only one node representing item 2 in the tree that is being shared four times. This kind of sharing, minimizes the nodes generated in the tree for dense dataset making it more compact. Thus for 16 items in the

dense dataset, only 10 corresponding nodes have been generated. On the contrary, all the 16 items in the example sparse dataset given in Fig. 15c are different. Due to this, the resultant tree in Fig. 15d will have 16 corresponding nodes, one node for each item in the dataset. The level-wise approaches perform well in case of sparse datasets compared to tree based approaches. Data structures like queues are faster and more space efficient in handling sparse datasets than tree data structures due to polynomial time complexity. An approach called *H-Mine* [28] employs queue data structure and has been found to perform appreciably well in case of sparse datasets. However, for dense datasets, tree based approaches are still the best option.

Mining performance is highly affected by the size of tree structure which relies on the arrangement or ordering of items. As elucidated in Sect. 4.3, a tree structure achieves maximum compactness when the items are arranged in a frequency descending order. Placing the most frequent items near the root guarantees the tree to be small and more compact. However, a tree structure failed to achieve such compactness when the items are arranged in a canonical order.

To better gauge the memory space efficiency of tree based approaches, we carried out several experiments on large datasets for each category of pattern generation

Fig. 15 Example of dense and sparse data

algorithms. In most of the cases, the experiments ran out of memory owing to memory shortage. A major drawback of tree based approaches identified in this study is their inefficiency in handling large datasets due to shortage of memory. The time taken to construct the tree structure is quite large, particularly when the dimensionality is large and it may not even fit into main memory if the database is very big and scattered. Compactness of tree structure is a major deciding factor of the efficiency of mining process. For instance, in case of Connect-4 dataset, number of nodes created in FP-Tree is 238,019, and if we consider the size of a node to be 36 bytes, total size of the FP-Tree for Connect-4 dataset will be $238,019 \times 36 = 8,568,684$ bytes which is approximately equal to 9 MB. Suppose the physical memory size is only 5 MB, then FP-Growth algorithm will fail for Connect-4 dataset. Connect-4 dataset is a small dataset with only 129 attributes and 67,554 instances, but if the dataset will have 10 millions of attributes and 100 millions of instances, then GB's of physical memory will not be sufficient. Limited amount of physical memory is a serious drawback of tree based approaches due to which it is necessary to develop new methods that do not fully rely on physical memory and utilize the secondary storage in the mining process. Several extensions have been proposed in the literature to handle this serious drawback of tree based approaches [1, 29].

In another set of experiments, we analyzed the efficiency of tree based approaches in handling incremental datasets over level-wise approaches. We carried out the experiments on the four categories of static techniques discussed in earlier sections and compared them with level-wise approaches. The time taken by tree based approaches to operate on the newly added data is considerably less than level-wise approaches. This is due to the fact that tree based approaches perform only two database scans in comparison to level-wise approaches that perform multiple database scans. To further enhance the performance of tree based approaches on incremental datasets, several variants have been proposed in the literature [15, 16, 22, 24].

6 Conclusion

Over the years, tree based approaches have emerged as the most eminent techniques in the field of pattern mining. In spite of their substantial prominence, tree based approaches become inconsistent at certain instances. An in-depth analysis of the performance bottlenecks of tree based approaches therefore becomes necessary for developing efficient pattern mining techniques employing tree data structures. The prime focus of this study is to enable the researchers gain an insight into the

performance and respective pros and cons of the fundamental tree based pattern mining techniques through a detailed theoretical and empirical analysis.

Initially, we provided a structural classification of the pattern mining techniques in four different categories based upon the type of patterns generated and a theoretical comparison of all these algorithms specifying their merits and demerits. The categorization has been done as: frequent, maximal frequent, closed frequent and rare pattern mining algorithms. To gauge the performance of the concerned algorithms, we presented an empirical analysis based on several decisive parameters. We performed experiments using different real and synthetic benchmark datasets. Both dense and sparse datasets have been considered for analyzing the behavior of the algorithms under different data characteristics. The performance is analyzed based upon the execution time invested, amount of memory consumed and the compactness achieved by the tree data structures. The empirical study initiated with an exhaustive scrutiny of the frequent pattern mining algorithms, followed by maximal frequent, closed frequent and rare pattern mining algorithms. Furthermore, an extensive study of the factors influencing the performance of tree based approaches has been done using experimental evaluation on several benchmark datasets. The factors identified as decisive parameters are: dimensions of dataset, change in support threshold and ordering of items in tree structure. A detailed analysis of the pros and cons of the tree based approaches has been provided in the discussion section.

Despite their preeminence, the inconsistency of tree based approaches at certain instances cannot be overlooked and need utmost attention. Through this structural and empirical analysis, an attempt has been made to exemplify the performance of tree based pattern mining approaches under various conditions. This study is an attempt to let the researchers analyze the merits as well as performance bottlenecks of tree based approaches, for developing efficient pattern mining techniques employing tree data structures.

Compliance with ethical standards

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. Adnan M, Alhaji R (2009) Drfp-tree: disk-resident frequent pattern tree. *Appl Intell* 30(2):84–97

2. Agrawal R, Imieliński T, Swami A (1993) Mining association rules between sets of items in large databases. In: *Acm sigmod record*, ACM 22:207–216
3. Agrawal R, Srikant R, et al (1994) Fast algorithms for mining association rules. In: *Proc. 20th int. conf. very large data bases, VLDB*, vol 1215, pp 487–499
4. Bayardo RJ Jr (1998) Efficiently mining long patterns from databases. *ACM Sigmod Rec* 27(2):85–93
5. Bhatt U, Patel P (2015) A novel approach for finding rare items based on multiple minimum support framework. *Procedia Comput Sci* 57:1088–1095
6. Borah A, Nath B (2017a) Mining patterns from data streams: an overview. In: *2017 International conference on I-SMAC (IoT in social, mobile, analytics and cloud) (I-SMAC)*. IEEE, pp 371–376
7. Borah A, Nath B (2017b) Mining rare patterns using hyper-linked data structure. In: *International conference on pattern recognition and machine intelligence*. Springer, pp 467–472
8. Borah A, Nath B (2017c) Rare association rule mining: a systematic review. *Int J Knowl Eng Data Min* 4(3–4):204–258
9. Borah A, Nath B (2018a) Fp-tree and its variants: towards solving the pattern mining challenges. In: *Proceedings of first international conference on smart system, innovations and computing*. Springer, pp 535–543
10. Borah A, Nath B (2018b) Identifying risk factors for adverse diseases using dynamic rare association rule mining. *Expert Syst Appl* 113:233–263
11. Borah A, Nath B (2018c) Rare association rule mining from incremental databases. *Pattern Anal Appl* 1–22
12. Borah A, Nath B (2018d) Rare pattern mining: challenges and future perspectives. *Complex Intell Syst* 1–23
13. Borah A, Nath B (2019) Performance analysis of tree-based approaches for pattern mining. In: *Computational intelligence in data mining*. Springer, pp 435–448
14. Burdick D, Calimlim M, Gehrke J (2001) Mafia: a maximal frequent itemset algorithm for transactional databases. In: *Proceedings of the 17th international conference on data engineering*, 2001. IEEE, pp 443–452
15. Cheung W, Zaiane OR (2003) Incremental mining of frequent patterns without candidate generation or support constraint. In: *Proceedings of the seventh international database engineering and applications symposium*, 2003. IEEE, pp 111–116
16. Ezeife CI, Su Y (2002) Mining incremental association rules with generalized fp-tree. In: *Conference of the Canadian Society for computational studies of intelligence*. Springer, pp 147–160
17. Goethals B (2003) Survey on frequent pattern mining. *Univ Helsinki* 19:840–852
18. Gouda K, Zaki MJ (2005) Genmax: an efficient algorithm for mining maximal frequent itemsets. *Data Min Knowl Discov* 11(3):223–242
19. Grahne G, Zhu J (2003a) Efficiently using prefix-trees in mining frequent itemsets. In: *FIMI*, vol 90
20. Grahne G, Zhu J (2003b) High performance mining of maximal frequent itemsets. In: *6th International workshop on high performance data mining*
21. Han J, Pei J, Yin Y, Mao R (2004) Mining frequent patterns without candidate generation: a frequent-pattern tree approach. *Data Min Knowl Discov* 8(1):53–87
22. Hong TP, Lin CW, Wu YL (2008) Incrementally fast updated frequent pattern trees. *Expert Syst Appl* 34(4):2424–2435
23. Leung CKS, Khan QI, Li Z, Hoque T (2007) Cantree: a canonical-order tree for incremental frequent-pattern mining. *Knowl Inf Syst* 11(3):287–311
24. Lin CW, Hong TP, Lu WH (2009) The pre-fupf algorithm for incremental mining. *Expert Syst Appl* 36(5):9498–9505
25. Lin KC, Liao IE, Chen ZS (2011) An improved frequent pattern growth method for mining association rules. *Expert Syst Appl* 38(5):5154–5161
26. Pasquier N, Bastide Y, Taouil R, Lakhal L (1999) Discovering frequent closed itemsets for association rules. In: *International conference on database theory*. Springer, pp 398–416
27. Pei J, Han J, Mao R et al (2000) Closet: an efficient algorithm for mining frequent closed itemsets. In: *ACM SIGMOD workshop on research issues in data mining and knowledge discovery*, vol 4, pp 21–30
28. 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: *Proceedings IEEE international conference on data mining*, 2001. ICDM 2001. IEEE, pp 441–448
29. Rácz B (2004) nonordfp: an fp-growth variation without rebuilding the fp-tree. In: *FIMI*
30. Sucahyo YG, Gopalan RP (2004) Ct-pro: a bottom-up non recursive frequent itemset mining algorithm using compressed fp-tree data structure. *FIMI* 4:212–223
31. Szathmary L, Napoli A, Valtchev P (2007) Towards rare itemset mining. In: *19th IEEE international conference on tools with artificial intelligence*, vol 1, 2007. ICTAI 2007. IEEE, pp 305–312
32. Troiano L, Scibelli G, Birtolo C (2009) A fast algorithm for mining rare itemsets. In: *2009 Ninth international conference on intelligent systems design and applications*. IEEE, pp 1149–1155
33. Tsang S, Koh YS, Dobbie G (2011) Rp-tree: rare pattern tree mining. In: *Data warehousing and knowledge discovery*. Springer, pp 277–288
34. Uno T, Asai T, Uchida Y, Arimura H (2003) Lcm: an efficient algorithm for enumerating frequent closed item sets. In: *FIMI*, vol 90
35. Wang J, Han J, Pei J (2003) Closet+: searching for the best strategies for mining frequent closed itemsets. In: *Proceedings of the ninth ACM SIGKDD international conference on knowledge discovery and data mining*. ACM, pp 236–245
36. Yahia SB, Hamrouni T, Nguifo EM (2006) Frequent closed itemset based algorithms: a thorough structural and analytical survey. *ACM SIGKDD Explor Newslett* 8(1):93–104
37. Yan YJ, Li ZJ, Chen HW (2005) Efficiently mining of maximal frequent item sets based on fp-tree. *Ruan Jian Xue Bao (J Softw)* 16(2):215–222
38. Zaki MJ, Hsiao CJ (2002) Charm: An efficient algorithm for closed itemset mining. In: *SDM, SIAM* 2:457–473
39. Zaki MJ, Parthasarathy S, Ogihara M, Li W et al (1997) New algorithms for fast discovery of association rules. *KDD* 97:283–286
40. Zheng Z, Kohavi R, Mason L (2001) Real world performance of association rule algorithms. In: *Proceedings of the seventh ACM SIGKDD international conference on knowledge discovery and data mining*. ACM, pp 401–406

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.