Complex & Intelligent Systems (2021) 7:589–619
https://doi.org/10.1007/s40747-020-00226-4

SURVEY AND STATE OF THE ART

Comparative evaluation of pattern mining techniques: an empirical study

Anindita Borah[1] · Bhabesh Nath[1]

Received: 9 December 2019 / Accepted: 27 October 2020 / Published online: 11 November 2020

Abstract
Pattern mining has emerged as a compelling field of data mining over the years. Literature has bestowed ample endeavors in this field of research ranging from frequent pattern mining to rare pattern mining. A precise and impartial analysis of the existing pattern mining techniques has therefore become essential to widen the scope of data analysis using the notion of pattern mining. This paper is therefore an attempt to provide a comparative scrutiny of the fundamental algorithms in the field of pattern mining through performance analysis based on several decisive parameters. The paper provides a structural classification of the widely referenced techniques in four pattern mining categories: frequent, maximal frequent, closed frequent and rare. It provides an analytical comparison of these techniques based on computational time and memory consumption using benchmark real and synthetic data sets. The results illustrate that tree based approaches perform exceptionally well over level wise approaches in case of dense data sets for all the categories. However, for sparse data sets, level wise approaches performed better than the former ones. This study has been carried out with an aim to analyze the pros and cons of the well known pattern mining techniques under different categories. Through this empirical study, an endeavor has been made to enable the researchers identify some fruitful and promising research directions in one of the most remarkable area of research, pattern mining.

Keywords Association rule mining · Frequent itemsets · Pattern mining · Performance analysis

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. Data mining has established itself as an inspiring area of database research whose prime concern is to extract hidden and meaningful information from databases. An imperative area of data mining research is pattern mining that aims to identify momentous patterns and correlations existing within a database. 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 [10,11,15]. After establishing itself as a compelling and fruitful research area for over a decade, pattern mining demands for an overview and re-examination of the various techniques developed and let the researchers identify their pros and cons, in order to establish it as a cornerstone approach in the field of data mining.

Pattern mining is the initial phase of association rule mining that extracts significant patterns and further generates meaningful association rules from those patterns. Pattern mining techniques generate different types of patterns depending upon the requirements of the user. These patterns and rules serve different applications ranging from fraud detection, medical diagnosis, web mining to customer transaction analysis. A wide range of pattern mining techniques are available in the literature covering different aspects of data mining applications. A careful study of all these techniques is therefore a necessary requirement to have complete grasp of the area of pattern mining.

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 techniques generating different categories of patterns or itemsets using experimental valida-

✉ Anindita Borah
anindita01.borah@gmail.com

Bhabesh Nath
bnath@tezu.ernet.in

1 Department of Computer Science and Engineering, Tezpur University, Napaam, Tezpur, Sonitpur, Assam 784028, India

tion. An empirical attempt can be found in [46] that analyzes only the closed frequent itemset generation algorithms. Few other attempts illustrate the behavior of only frequent itemset generation algorithms through experimental evaluation [20,51]. However, no initiative has been taken to perform an analytical study on the maximal frequent itemset and rare itemset generation techniques. This study offers an unbiased empirical study of the fundamental 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 pattern mining techniques developed under various constraints using different thresholds and data sets.

The pattern mining algorithms basically attempts to identify the four main categories of itemsets: frequent, maximal frequent, closed frequent and rare. This paper attempts to provide a comparative analysis of all the mainstream algorithms generating different categories of itemsets. The main purpose of this study is to let the researchers in the field of pattern mining identify the pros and cons of techniques generating all the main categories of patterns. To the best of our knowledge, it is the first attempt to do a comparative study of different categories of pattern mining techniques together. The primary aim of this study is to perform a detailed analysis of the existing techniques widely referenced in the literature, under the following four themes: (1) frequent itemset generation, (2) closed frequent itemset generation, (3) maximal frequent itemset generation and (4) rare itemset generation. Through this work, an attempt has been made to improve on the existing studies in the following way:

– evaluation of a large number (19) of widely referenced pattern mining techniques using different thresholds.
– analysis of the techniques through experimental evaluation on several real and synthetic benchmark data sets [19].
– comparison of the techniques through graphical analysis under different scenarios.

The remaining paper is organized as follows: Sect. 2 illustrates the algorithms considered for this experimental study. Section 3 elicits and discusses the experimental results obtained for different categories of algorithms. Finally, Sect. 6 summarizes and discusses the findings of this empirical study.

Algorithms considered for the study

This section illustrates the algorithms that have been considered for this study. Several mainstream algorithms in the field of pattern mining have been taken into account for comparison and evaluation using different thresholds and data sets [19]. Nineteen algorithms encompassing different pattern mining issues have been considered for this study. The algorithms have been chosen from the studies numerously referenced in the literature [4,6,18,21–24,26,29,32–34,39,43,44,49,50]. They can be distinguished depending upon the type of itemsets generated during itemset generation phase.

Frequent itemset generation

The notion of pattern mining was introduced, considering the usefulness and applicability of frequent patterns or itemsets present in the database [8]. Transaction databases usually contain huge number of distinct single items whose combination further tends to generate enormous quantity of itemsets [31]. Devising scalable techniques to handle such large number of itemsets itself is a challenging task. Thus existing pattern mining techniques employ several strategies to deal with this issue. For this study, five primer and most widely accepted frequent pattern mining techniques have been considered which are described as follows:

(a) **Apriori:** To handle the vast quantity of itemsets produced and to generate only the fruitful frequent itemsets, [4] introduced a downward closure property called *Apriori*. According to this property, *an itemset can be considered to be frequent, only if all its proper subsets are frequent*. 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*. In spite of being the primer and one of the simplest techniques, levelwise 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.

(b) **FP-Growth:** Multiple scanning of the database and generation of enormous number of candidate itemsets through pattern matching escalates the computational cost of *Apriori* algorithm. [24] in their algorithm managed to overcome the shortcomings faced by the levelwise pattern mining algorithms. Their well-known algorithm called *Frequent Pattern Growth (FP-Growth)* employs a trie-based tree data structure to reduce the number of database scans and avoid the generation of candidate itemsets. The algorithm uses divide and conquer strategy and generates a compressed tree representation of the original database using the frequent items generated during the initial scan of the database.

Pattern mining is then performed on the tree representation of the database by generating conditional pattern bases. Employing such a strategy enables the frequent itemsets to be mined using only two scans of the database and also avoids extravagant steps of pattern matching and candidate generation. This greatly minimizes the computational complexity of the algorithm, thus making it the most widely accepted pattern mining technique. Despite its eminence and numerous advantages, *FP-Growth* still suffers from certain drawbacks, most crucial being the limitation of memory. It fails miserably when huge number of frequent itemsets are generated in case of large data sets and henceforth the tree can no longer be accommodated in main memory.

(c) **AprioriTID:** The *AprioriTID* algorithm [4] was developed as an extension of the primer *Apriori* algorithm where the former does not use the database for support counting of candidate itemsets after the initial pass. The algorithm uses the encoding of the candidate itemsets generated in the previous pass. The size of the encoding tends to get reduced after each pass and becomes much smaller as compared to the original database that reduces the mining intricacy to a extent. However, in the initial passes *Apriori* performs better than *AprioriTID*.

(d) **ECLAT:** [49] 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 parallel execution. Unlike *Apriori* and *FP-Growth*, 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.

(e) **H-Mine:** In addition to memory limitation, *FP-Growth* suffers from performance bottlenecks in case of sparse data. Experimental results available in the literature illustrates that the performance of *FP-Growth* tends to deteriorate with the increase in sparseness of data. The number of frequent items decreases with growing sparseness of data thus reducing the probability of node sharing among frequent items in the tree. Henceforth, the resulting tree becomes very large affecting the performance of the algorithm in terms of execution time and main memory. [34] managed to overcome this demerit of *FP-Growth* by introducing their technique called *H-Mine* that makes use of queues instead of tree data structure. *H-Mine* stores the items of the transactions in separate queues and links transactions having same first item name using hyper-links. It performs appreciably well with sparse data set and is more efficient than *Apriori* and *FP-Growth* in terms of space usage and execution time. However, *FP-Growth* still overpowers *H-Mine* in case of dense data sets.

Numerous other frequent pattern mining techniques have been developed encompassing different issues associated with frequent pattern mining [2,30,47]. The issue of candidate generation and multiple database scans was resolved by FP-Growth algorithm. To further improve the FP-Growth algorithm, several variants of FP-Growth were proposed, the most efficient being the LP-Tree algorithm [35]. LP-Tree managed to reduce the tree traversal time with the help of additional data structures. LP-Tree algorithm was further enhanced by [38] that uses a top down approach to reduce the number of level searches and identify the frequent itemsets. Incremental mining is an important issue in the field of pattern mining as most of the techniques fail to efficiently handle the incremental data sets. FP-Growth managed to reduce the number of database scans to two but still could not handle the issue of incremental mining effectively. [41] modified the FP-Growth algorithm to generate the frequent itemsets in a single database scan for efficient incremental and interactive mining. Their algorithm called CP-Tree perform branch readjustment to obtain the FP-Tree structure. It however, cannot maintain the FP-Tree structure at all times and the resultant FP-Tree is generated only after the user defined criteria is satisfied. [37] enhanced the CP-Tree algorithm to maintain the FP-Tree structure at all times and employed two hash tables to minimize the number of branch comparisons.

Closed frequent itemset generation

An important issue to deal with during frequent pattern mining is the generation of inexpediently large number of frequent itemsets. Research on pattern mining identified a significant solution to solve this issue in the form of *closed frequent itemsets* (CFI) that posses the same potential as that of the large set of frequent itemsets generated. 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. The most widely referenced closed frequent mining techniques considered for this study are described as follows:

(a) **Apriori Close:** For the purpose of generating the *closed frequent itemsets*, *Apriori Close* [32] initially generates a set of generators based on the closure properties of itemsets employing a levelwise search procedure. The support counts of all the generators are then obtained to remove the unwanted generators. The closure of all the desirable generators are computed to obtain the set of CFI's. *Apriori Close* performs exceptionally well with dense data set but suffers from performance degradation while dealing with data set with long patterns at low support thresholds.

(b) **FPClose:** [22] proposed another variant of *FP-Growth* called *FPclose* to mine the *closed frequent itemsets*. The algorithm makes use of a tree data structure named *Closed Frequent Itemset Tree* (CFI-Tree) to store the CFI's obtained from the set of frequent itemsets. Whenever a new frequent itemset is generated, it is compared with the existing *closed frequent itemsets* in the CFI-Tree. A frequent itemset will be considered to be closed, provided it has no superset in the CFI-Tree with same support count. *FPclose* performs efficiently due to the usage of array based implementation and compact tree structure. The drawback however is the additional time spent by the algorithm in checking the *closedness* of frequent itemsets.

(c) **LCM:** To generate the frequent closed itemsets, [44] introduced the *Linear Time Closed Itemset Miner (LCM)* algorithm that employs a parent child relationship between the frequent closed itemsets. The algorithm builds set enumeration tree for the frequent itemsets and the closure of these itemsets is obtained by traversal of the enumeration tree. The algorithm obtains the frequent closed itemsets in linear computational time.

(d) **CHARM:** *CHARM* [50] introduced a novel data structure called *Itemset Tree* to examine the itemset space as well the transaction space. Employing a hybrid search method by the algorithm, allows faster generation of CFI's by avoiding search at several levels of the *itemset tree*. The algorithm further uses a hash based technique to remove the itemsets that do not satisfy the closure constraints. For efficiency in memory usage and faster frequency calculations, *CHARM* employs another data structure called *diffset*. It is however, not very efficient in case of data sets with long patterns

(e) **CLOSET:** *CLOSET* [33] algorithm was introduced with the purpose of developing a scalable technique that is capable of handling larger data sets. It extends the concept of pattern growth adopted previously by *FP-Growth* algorithm. To mine the CFI's, the algorithm generates a compressed tree representation of the database, similar to FP-Tree. Furthermore, it uses a partition based projection technique to lessen the search space as well as to achieve scalable pattern mining. *CLOSET* is faster and efficient than other CFI generation techniques over dense data sets. However, in case of sparse data sets, it still needs improvement.

Scalability and reduction of search space has always been a crucial issue while mining closed frequent itemsets. [36] developed an efficient technique that identifies the closed frequent itemsets using a tree structure. The algorithm is highly scalable and shows appreciable performance in terms of execution time. [28] attempted to identify the closed frequent itemsets using a vertical data structure. Their proposed vertical data structure reduces the storage space to a great extent and therefore can effectively handle large data sets. Another issue to deal with in case of closed frequent pattern mining is the generation of closed frequent patterns from data streams. [25] developed an algorithm to handle the concept drift problem and identify the closed frequent itemsets from data streams.

Maximal frequent itemset generation

Extracting the entire set of frequent itemsets is a computationally challenging as well as extravagant phase of pattern mining. Studies in the literature illustrate that if a particular itemset is frequent then its subset must also be frequent. Such a notion emphasizes that extraction of only the *maximal frequent itemsets* (MFI) will be sufficient rather than the complete set of frequent itemsets, thus minimizing the huge number of frequent itemsets generated. A frequent itemset qualifies to become a *maximal frequent itemset*, only if none of its superset is frequent. A brief discussion on the algorithms considered for this empirical study is given below:

(a) **Max-Miner:** In order to generate the long patterns or more specifically, the *maximal frequent itemsets*, [6] developed an extension of Apriori algorithm called *Max-Miner*. The algorithm reduces the search space by removing the candidate itemsets having an infrequent subset. *Max Miner* avoids expensive traversal of search space in a bottom up fashion and employs a look ahead search for faster identification of MFI's. However, it expends multiple database scans while searching for the long frequent patterns.

(b) **MAFIA:** *MAFIA* [18] employs a depth-first search approach to generate the *maximal frequent itemsets*. 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 bitvector *and* operation is applied on each bitvectors 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 data sets that have long average pattern length.

(c) **GenMax:** *GenMax* developed by [21] operates on vertical representation of data sets. It adopts the mechanism of *progressive focussing* to identify the set of MFI's. Based on this technique, the algorithm generates a list of *local maximal frequent itemsets* (LMFI). Whenever, a new frequent itemset is produced, instead of comparing it with all the previously generated MFI's, *GenMax* compares it with the list of *local maximal frequent itemsets*. It performs well with data sets having long average transaction length but demands performance improvement in case of data sets having long average pattern length.

(d) **FPMax:** *FPMax* [23] was developed as an extension of *FP-Growth* algorithm to find the MFI's. It initially constructs the *Maximal Frequent Itemset Tree* (MFI-Tree) to store the MFI's. Only those frequent itemsets will be inserted into the MFI-Tree that are subsets of itemsets already present in the tree. The algorithm generates the supersets of frequent itemsets and removes the non-maximal frequent itemsets. *FPMax* is highly scalable and works well with data sets having short average transaction length. However, it spends a lot of time in construction of the MFI-Tree.

To reduce the complexity of mining maximal frequent itemsets and minimize the cost of large execution time and memory usage, several techniques have been developed. [45] developed an algorithm that uses an N-List structure to compress the data set and mine the top-rank k maximal frequent patterns. They also proposed a pruning technique in order to reduce the search space. One of the challenges in the field of maximal frequent pattern mining is the generation of maximal frequent patterns from data streams. [48] developed a novel technique to identify the maximal frequent patterns over data streams by imposing some weight constraints. The algorithm performs a single scan of the database, thus minimizing the overhead of extracting the patterns from data streams. [27] developed a sliding window based technique to mine the maximal frequent patterns from data streams. The algorithm employs a strategy to avoid meaningless pattern generation by pruning the unnecessary operations.

Rare itemset generation

The paradigm of frequent pattern mining have always treated the rare patterns and itemsets to be of least importance and thus have always demanded for its removal or elimination during itemset generation phase. However, extraction of rare patterns or itemsets have recently gained much importance considering its manifold applications in several domains [10]. Substantial quantity of research has already been performed in the area of rare pattern mining that contributes enormous techniques looking for these previously unwanted rare itemsets [9,12–14,16,17]. The rare pattern mining techniques considered for comparative analysis have been described below.

(a) **MS Apriori:** The first endeavor towards rare pattern mining was taken by [29], who attempted to retain some rare items along with the frequent itemsets during the phase of itemset generation. They argued that using a single support threshold alone might not allow the fruitful rare items to be retained. This led them using separate support threshold for each item called their *Minimum Item Support* (MIS) value which is obtained by specifying an additional parameter β. With the usage of individual MIS values for the items, the algorithm satisfies different support requirements of the users. However, introduction of an additional user defined parameter increases the overhead of the algorithm.

(b) **Apriori Inverse:** *Apriori Inverse* proposed by [26] uses inverse downward closure property to obtain a special set of itemsets called *sporadic itemsets*. The support value of *sporadic itemsets* must be below a maximum support threshold but higher than a minimum support threshold. Despite its capability of find the *sporadic itemsets* faster than *Apriori*, *Apriori Inverse* fails to find the complete set of rare items.

(c) **Apriori Rare:** [40] introduced a modification of *Apriori* algorithm called *Apriori Rare*, with a view to generate both the frequent as well as rare itemsets. However, instead of generating the entire set of rare items, the algorithm is capable of generating a special class of itemset called *Minimal Rare Itemsets* (MRI).

(d) **ARIMA:** [39] developed *Another Rare Itemset Mining Algorithm* (ARIMA), targeting the complete set of rare items. ARIMA employs an *Apriori*-like levelwise approach and works as combination of two algorithms for the generation of rare itemsets. ARIMA uses Apriori Rare to find out the MRI's that were initially removed by *Apriori* algorithm. The algorithm then proceeds to find the *Mininimal Rare Generators* (MRG) from the *Frequent Generators* (FG) using another algorithm called *MRG-Exp*. With the MRI's as the input ARIMA generates the complete set of rare itemsets, those having zero as well as non- zero support. ARIMA however, fails to be time efficient as it spends a lot of time generating the MRI's and MRG's.

(e) **RP-Tree:** Keeping in view the significance of *FP-Growth* based approaches, [43] developed the first tree based algorithm for the extraction of rare itemsets called *Rare Pattern Tree* (RP-Tree). The algorithm during its initial phase obtains the rare items based on its support count. The next phase is the tree construction phase considering those transactions that contain at least one rare item in it. The pattern mining operation is same as that of *FP-Growth* by generating conditional pattern bases and conditional trees. In spite of its efficiency, RP-Tree is unable to generate the complete set of rare itemsets. It targets only a special class of rare itemsets called rare item itemset where in the entire itemset is composed of only rare items.

To resolve the issue of different support requirements of the user, RP-Tree algorithm is later extended by [7] using multiple support framework by assigning each item their individual support values. In order to minimize the number of searches, [42] proposed Rarity algorithm that selects the longest transaction within the database and performs a top down search for finding the rare itemsets thus avoiding the lower layers of the search space containing frequent itemsets. However, the mining intricacy of the algorithm is increased with the extraction of large number of candidates and level-wise search. AfRIM [1] uses the same top down strategy as Rarity algorithm for finding the rare itemsets. AfRIM initially searches for the itemset that contains all the items present in the database. During candidate generation, all the possible combinations of rare itemset pairs in the previous level are examined to find the common itemset subsets between them. The efficiency of the algorithm is highly affected by costly steps of candidate generation and pruning. A major issue with rare pattern mining techniques is inefficiency in handling incremental data sets. Several techniques have been proposed in the literature taking into account the issue of incremental rare pattern generation. Borah and Nath [13] developed an incremental rare pattern mining technique for earthquake trend analysis and anticipation. The technique could only handle the case of transaction insertion. Therefore, they further extended their approach in [12] to efficiently generate the rare patterns upon insertion as well as deletion of transactions. A queue data structure based rare pattern mining approach was proposed in [9] to handle the issue of generating rare patterns from sparse data set.

Table 1 shows a comparative analysis of the different category of algorithms considered for this study. The comparison has been done based on the strategy or mechanism used, number of database scans performed, type of itemset generated by the algorithms as well as their merits ans demerits.

Performance analysis

An extensive comparative analysis has been carried out on nineteen fundamental algorithms using three real and three synthetic data sets. Some publicly available Java implementations of certain standard algorithms like Apriori, FP-Growth, FPClose, CHARM, GenMax and FPMax have been used for this study. Rest of the algorithms have been implemented in Java on a 64-bit machine of 4 GB RAM.

Several benchmark real-life and synthetic data sets have been used for experimentation. The selected data sets are considered as benchmark data sets as these data sets are being widely used by the pattern mining techniques in the literature, specifically the techniques considered in this study. Table 2 illustrates the characteristics of the data sets used for evaluating the performance of pattern mining algorithms. The data sets are obtained from UCI Machine Learning Repository [19].

This section discusses the performance of selected pattern mining algorithms on the benchmark real and synthetic data sets. For simplicity and more clarity, the figures depicting the corresponding results were divided into four different categories. The first category contains five algorithms generating only the frequent itemsets namely the Apriori, FP-Growth, Apriori TID, ECLAT and H-Mine. The second category of algorithms include Apriori Close, FPClose, LCM, CHARM and CLOSET algorithms generating closed frequent itemsets. Four algorithms generating maximal frequent itemsets namely Max-Miner, MAFIA, GenMax and FPMax constitute the third category. The fourth category on the other hand, includes algorithms like MS Apriori, Apriori Inverse, Apriori Rare, ARIMA and RP-Tree that generates rare itemsets.

Zoo data set

This section illustrates the performance evaluation of the considered algorithms on zoo data set. The algorithms under the four categories have been compared based on their execution time, memory usage and number of itemsets generated.

Execution time

The execution time invested by the considered algorithms is illustrated through graphical analysis in Fig. 1. Figure 1a–d depicts the execution time analysis of the four categories of algorithms separately while Fig. 1d highlights the same for all the algorithms.

(a) *First Category:* In the first category of algorithms, *FP-Growth* outperforms all the other algorithms in terms of execution time. This is quite obvious since *Zoo* is a dense data set and performance of tree based approaches is best in case of dense data sets. *Zoo* data set contains a lot of frequent items due to which FP-Tree achieves good compression due to the overlapping of common items. Moreover, due to less database scans, execution time is greatly reduced in case of FP-Growth.
H-Mine manages to perform better than rest of the algorithms due to the usage of queue data structure for storing the frequent itemsets. It however fails to outperform *FP-Growth*, as the data set is dense where the performance of tree based approaches have proven to be the best. *ECLAT* manages to gain a spot between the levelwise algorithms *Apriori TID* and *Apriori* and pattern growth approaches *FP-Growth* and *H-Mine*. *Apriori* and *Apriori TID* compensates in terms of execution time due to huge number of candidate generation and multiple database scans. *Apriori TID* spends slightly higher execution time than *Apriori* due to the inclusion of the item ids along

Table 1 Comparison between pattern mining algorithms

Algorithm	Strategy	Database Scan	Itemset Generated	Merit	Demerit
Apriori [4]	Levelwise search	Multiple	Frequent	Generates more frequent itemsets	Huge memory consumption
FP-Growth [24]	Tree based	Two	Frequent	Less database scans	FP-Tree for large data may not fit in memory
Apriori TID [3]	Levelwise search	Multiple	Frequent	Database is not used for support counting of candidates	Higher execution time
ECLAT [49]	Levelwise search	Multiple	Frequent	Fast support counting	Huge memory consumption
H-Mine [34]	Queue based	Two	Frequent	Memory efficient	Not suitable for dense data sets
Apriori Close [32]	Levelwise search	Multiple	Closed Frequent	Generates more closed frequent itemsets	Costly when mining long patterns
FPClose [22]	Tree based	Two	Closed Frequent	Less execution time	FP-Tree for large data may not fit in memory
LCM [44]	Levelwise search	Multiple	Closed Frequent	Less execution time	Does not work well at lower supports
CHARM [50]	Tree based	Two	Closed Frequent	Quickly identifies the closed frequent itemsets	Does not work well at higher supports
CLOSET [33]	Tree based	Two	Closed Frequent	Avoids redundant computation	Does not work well at lower supports
Max-Miner [6]	Levelwise search	Multiple	Maximal Frequent	Quickly identifies long frequent itemsets	More memory consumption
MAFIA [18]	Levelwise search	Mutliple	Maximal Frequent	Efficient for databases having long itemsets	Performs expensive bit vector operations
GenMax [21]	Levelwise search	Multiple	Maximal Frequent	Fast support counting	Performs expensive bit vector operations
FPMax [23]	Tree based	Two	Maximal Frequent	Less execution time	Not suitable for large data sets
MS Apriori [29]	Levelwise search	Multiple	Frequent itemset having rare item	Generates more rare items	Determination of parameter β is cumbersome
Apriori Inverse [26]	Levelwise search	Multiple	Rare	Less execution time	Assignment of two thresholds
Apriori Rare [40]	Levelwise search	Multiple	Rare	Less Execution Time	Generates only the minimal rare itemsets
ARIMA [39]	Levelwise search	Multiple	Frequent and Rare	Generates the complete set of rare items	Huge memory consumption
RP-Tree [43]	Tree based	Two	Rare	Less execution time	Cannot generate the complete set of rare items

Table 2 Data sets used

Data sets	Number of transactions	Number of items	Average transaction size	Type
Zoo	101	36	17	Dense
Lymph	148	68	18	Dense
Mushroom	8124	119	23	Dense
Retail	88,162	16,470	10	Sparse
T10I4D100K	100,000	1000	10	Sparse
T40I10D100K	100,000	1000	40	Sparse

with the frequent itemsets. The execution time for all the algorithms tends to increase with decrease in the minimum support values due to generation of higher number of frequent itemsets.

(b) *Second Category:* The performance of the second category of algorithms are as follows. *FP-Close* performs better than *CLOSET* for all minimum support values. Both *FP-Close* and *CLOSET* recursively constructs the FP-Trees for dense data set *Zoo*. The difference in the execution times of the algorithms lies in the fact that *FP-Close* 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*. *LCM outperform CHARM* at all minimum support values due to its ability of duplicate detection and usage of several other optimization strategies. *CHARM* however managed to perform better than *Apriori Close*.

(c) *Third Category:* Among the third category of algorithms generating *maximal frequent itemsets*, the most effective one is the *FPMax* algorithm due to its obvious advantage of projection tree construction. Moreover, *Zoo* data set has a short average transaction length (ATL) of 17 and the average pattern length (APL) of the generated MFI's is comparable to ATL. *FPMax* will generate a small FP-Tree from this data set effectively generating MFI's from the extracted MFI-Tree. *GenMax* and *MAFIA* on the other hand, performs expensive bitvector operations and set intersections.Thus, *FP Max* shows better performance than *GenMax* and *MAFIA* in such data sets. Minimal difference can be seen in the execution times of *GenMax* and *MAFIA*. However, both the algorithms managed to perform better than *Max-Miner*.

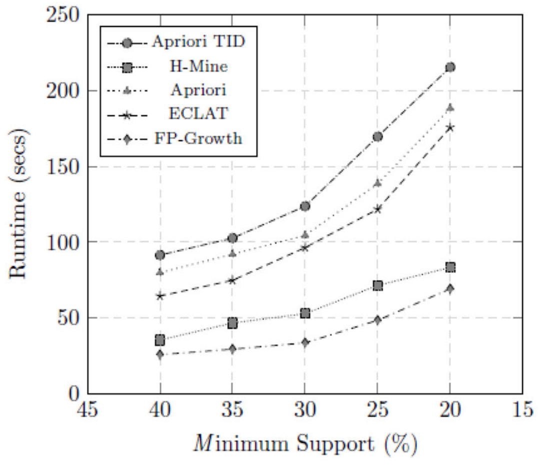
(d) *Fourth Category: ARIMA* proves to be the most expensive one, in the fourth category of algorithms generating rare itemsets. This is due to the fact that it spends a lot of time in the execution of *Apriori Rare* generating the MRI's and *MRG-Exp* extracting the MRG's. *RP-Tree* on the other hand, spends the least amount of execution time among all the algorithms due to the generation of only a subset of rare itemsets. Fixing the *Maximum Support* (maxsup) value at 60%, it has been observed that *Apriori Inverse* incurs execution time only slightly higher than *RP-Tree*. Using a β value of 0.1, *MS-Apriori* performs better than *ARIMA* but proves to be slightly expensive than *Apriori Rare*.

Memory Usage

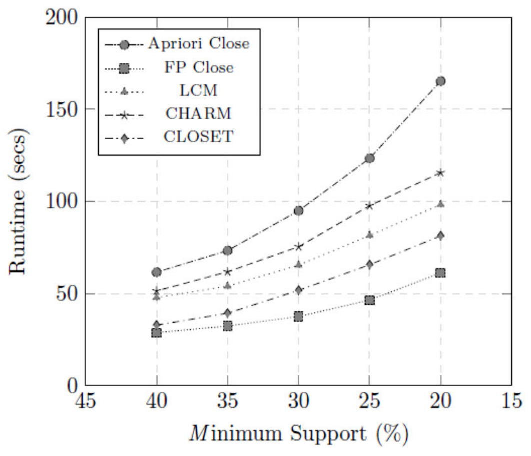
The memory usage of the different categories of algorithms is depicted in this section. Figure 2a–d, shows a graphical analysis of the four categories. Figure 2e shows a comparison of all the algorithms in terms of their memory usage.

(a) *First Category:* Among the algorithms belonging to the first category, *FP-Growth* proved to be the best algorithm in terms of memory usage. Due to the presence of too many frequent items in the data set, the size of generated FP-tree will be very less due to the sharing of common items. The next best in this catgory is the *H-Mine* algorithm. Employing queue data structures during frequent itemset generation gave an added advantage to *H-Mine* as queue data structures consume less memory. *ECLAT* has been found to be showing an average amount of memory consumption only slightly higher than *FP-Growth* and *H-Mine*. *Apriori TID* is the most expensive in this category followed by *Apriori* due to the storage of candidate itemsets generated during each phase of itemset generation.

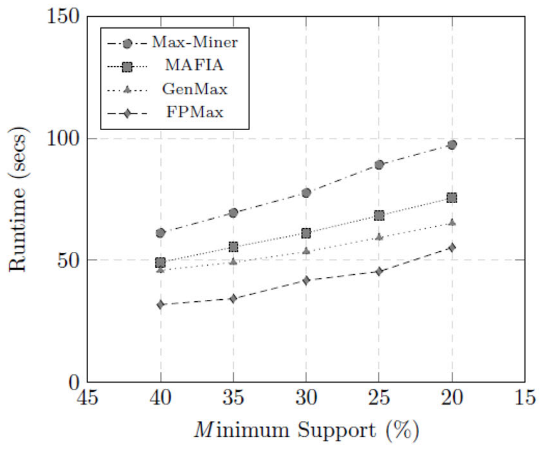
(b) *Second Category: AprioriClose* consumed the maximum amount of memory among the other algorithms under this category. The reason behind this overhead is that *AprioriClose* tends to store each extracted *Minimal Frequent Generator* (FMG) in main memory until it obtains the entire set. Thus the closure computation for each generated FMG proves to be expensive. *CLOSET* 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* (CI) with supports lower than the minimum support value in main memory. Since the number of CI's in case of sparse data sets is very large, *CLOSET* does not perform well in case of sparse data sets.



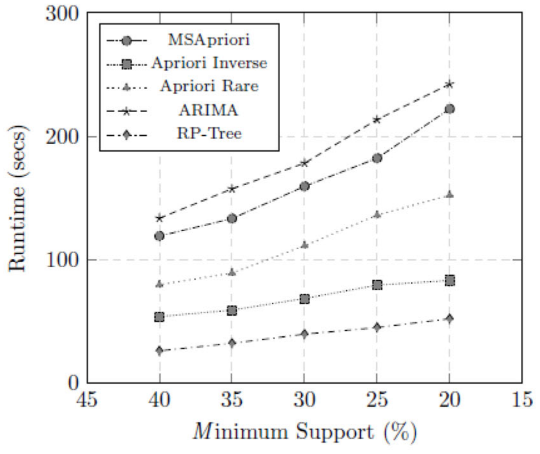
(a) Frequent Itemset Generation



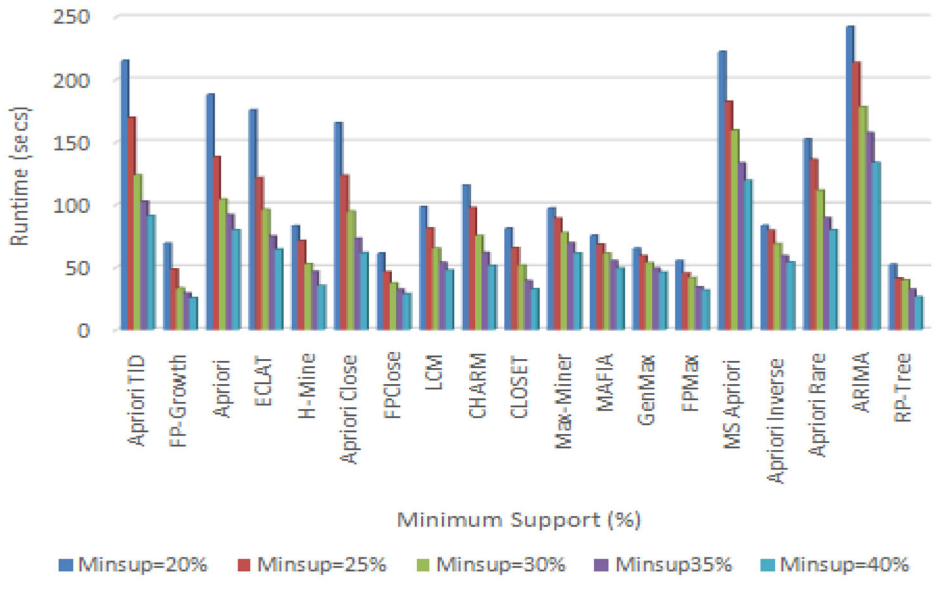
(b) Closed Frequent Itemset Generation



(c) Maximal Frequent Itemset Generation



(d) Rare Itemset Generation



(e) Execution Time of Algorithms

Fig. 1 Execution time on zoo data set. **a** Frequent itemset generation. **b** Closed frequent itemset generation. **c** Maximal frequent itemset generation. **d** Rare itemset generation. **e** Execution time of algorithms

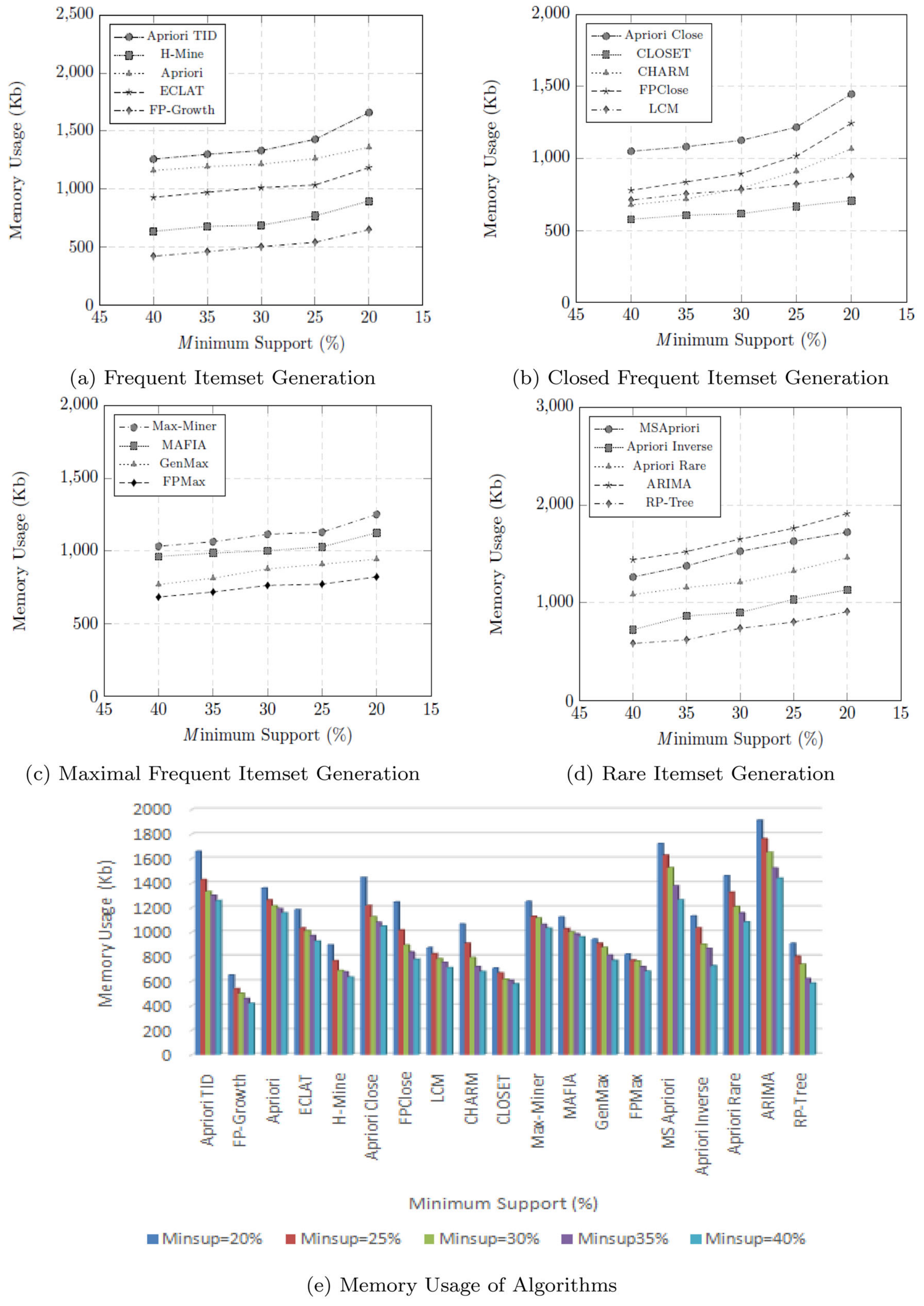


Fig. 2 Memory usage on zoo data set. **a** Frequent itemset generation. **b** Closed frequent itemset generation. **c** Maximal frequent itemset generation. **d** Rare itemset generation. **e** Memory usage of algorithms

LCM on the other hand, maintained a steady memory consumption at all support values as it avoids storing the previously extracted FCI's in main memory. Both *CHARM* and *FP-Close* proved to be a little expensive in terms of memory consumption at low support values. This is due to the fact that both the algorithms tend to store the previously generated 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.

(c) *Third Category: FPMax* outperformed the rest of the algorithms under this category. Since the FP-Tree generated for this data set will be too small due to short ATL which gives *FPMax* an advantage over others. *GenMax* consumes a slightly higher memory as it needs to retain the LMFI's for comparison with the generated MFI's. Storing the bit vectors for each itemset generated, made *MAFIA* a bit expensive than the previous two algorithms. *Max-Miner* on the other hand, consumed the maximum amount of memory among other algorithms.

(d) *Fourth Category:* In this category, the algorithm consuming the maximum amount of memory is *ARIMA* due to the storage of the MRI's and MRG's obtained from *Apriori Rare* and *MRG-Exp*. The next expensive algorithm in terms of memory consumption is *MSApriori*. Storage of the candidate itemsets at each phase of itemset generation demands more amount of main memory for this algorithm. *Apriori Rare* managed to consume less memory than *ARIMA* and *MS-Apriori* due to the retainment of only the MRI's. *RP-Tree* is undoubtedly the best algorithm in this regard due to the generation of small FP-Tree in this data set, that too storing only the rare-item itemsets. With maximum support threshold of 60% and for storing only the *sporadic itemsets*, *Apriori Invserse* managed to consume a marginal amount of main memory slightly higher than *RP-Tree*.

Itemsets generated

Figure 3 illustrates the different types of itemsets generated from *Zoo* data set. The maximum number of itemsets generated in this data set is that of the frequent itemsets. *Zoo* being a highly dense data set has maximum number of frequent itemsets. The next type of itemsets generated are the rare itemsets followed by the rare-item itemsets. Number of CFI's generated have been found to be quite higher than that of MFI's. MRI's and *sporadic itemsets* make up only a small portion of the data set.

Lymph

The performance evaluation of the algorithms under different criterias on *Lymph* data set is illustrated in this section.

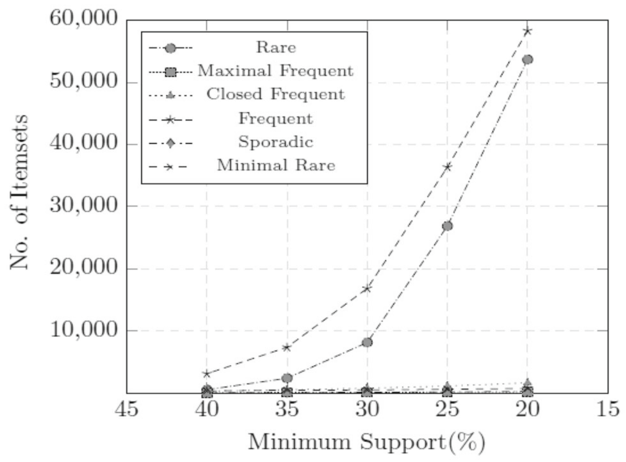


Fig. 3 Itemsets generated in zoo data set

Execution time

This section contains the performance analysis of the algorithms in terms of execution time. Figure 4a–d compares the four categories of algorithms separately while Fig. 4e examines the algorithms altogether through graphical analysis.

(a) *First Category:* The performance of the first category of algorithms in terms of execution time are as follows. The most expensive one under this category has been found to be *Apriori TID* with minimally higher execution time than *Apriori*. The idea of generating candidates and storing the identifiers of items escalates the execution time of *Apriori TID* making it the most extravagant algorithm. *ECLAT* performed better than the previous two algorithms and managed to present a moderate conduct. *FP-Growth* has again shown the best performance among all the other algorithms, for *Lymph* being a dense data set followed by *H-Mine*.

(b) *Second Category: FPClose* surpassed the rest of the algorithms under this category with only a nominal difference with that of *CLOSET*. The reduced subsumption checking cost and idea of storing only a fraction of CFI's facilitated *FPClose* to be the best among others. *LCM* displayed a consistent performance and *CHARM* managed a slot just below it. It, however, performed than *Apriori Close*.

(c) *Third Category:* In this category, the performance of *FPMax* outperformed the other algorithms. For the same reasons discussed previously, appreciable performance can be observed from *FPMax*. *GenMax* and *MAFIA* has slightly higher execution time due to bitvector operations and set intersections. *Max-Miner* again fail to show convincing performance as it spends a lot of time searching for the long MFI's.

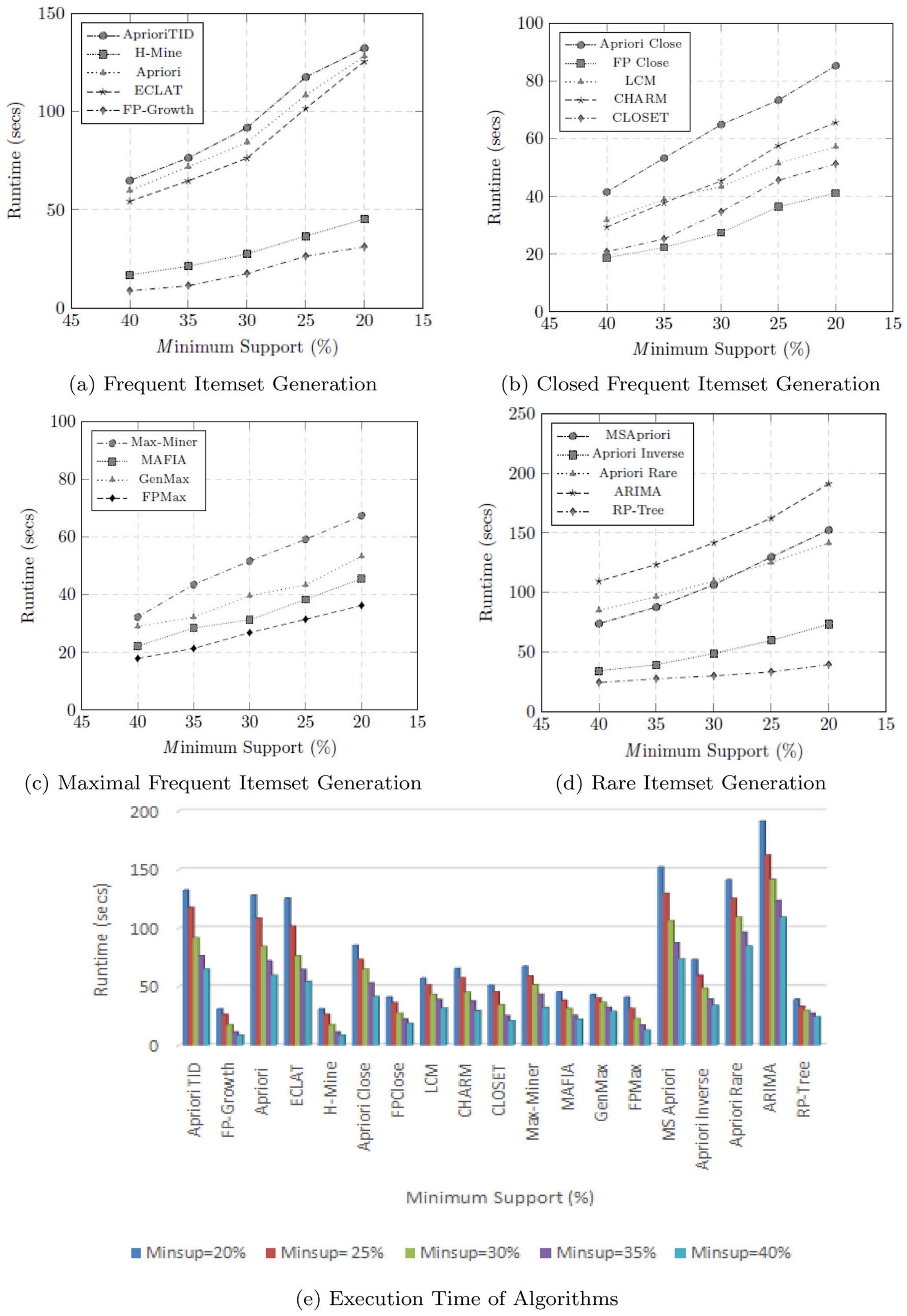
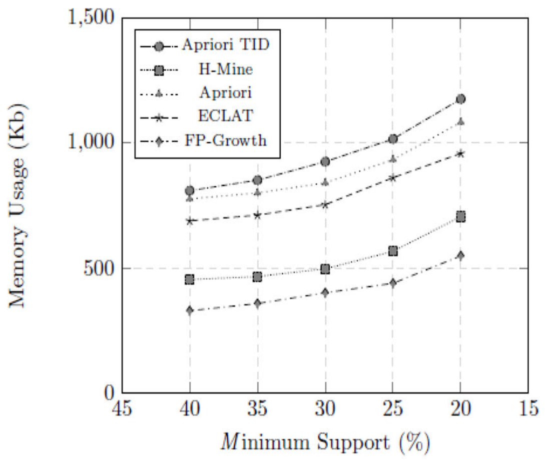
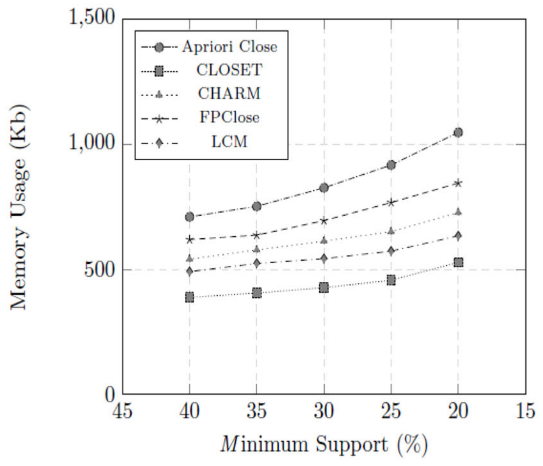


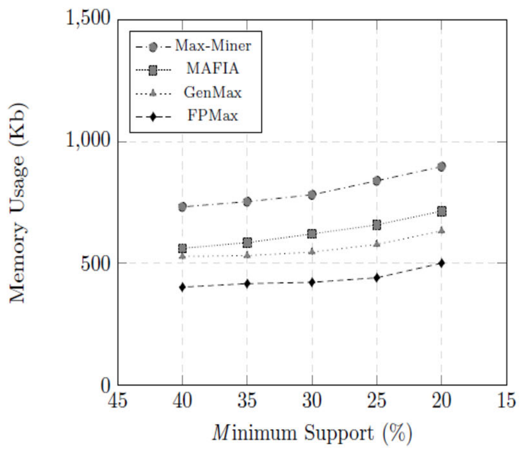
Fig. 4 Execution time on lymph data set. **a** Frequent itemset generation. **b** Closed frequent itemset generation. **c** Maximal frequent itemset generation. **d** Rare itemset generation. **e** Execution time of algorithms



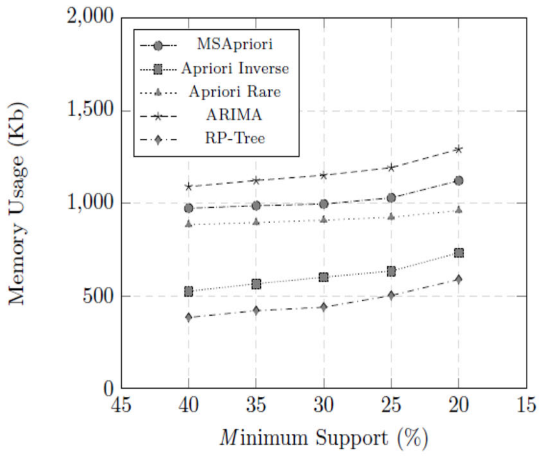
(a) Frequent Itemset Generation



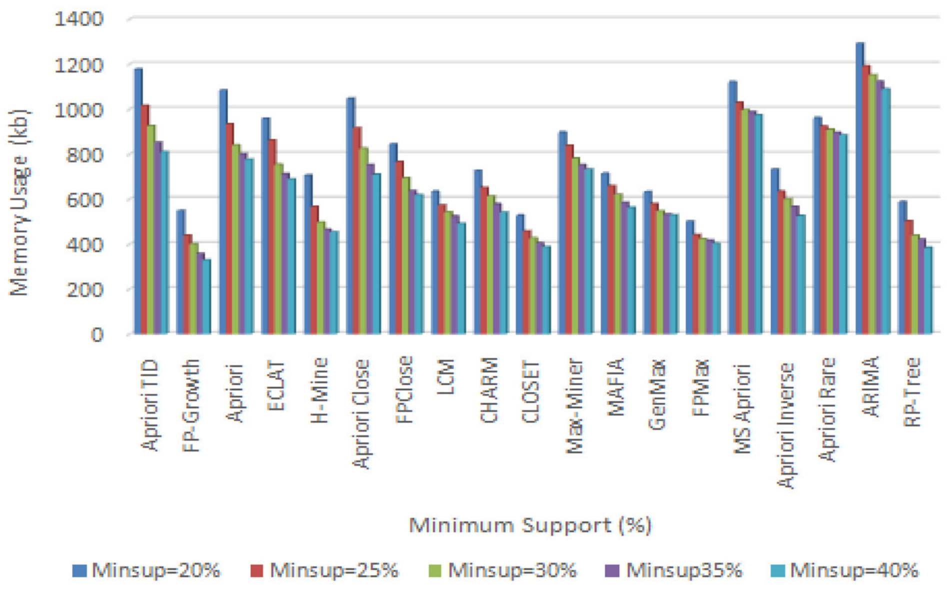
(b) Closed Frequent Itemset Generation



(c) Maximal Frequent Itemset Generation



(d) Rare Itemset Generation



(e) Memory Usage of Algorithms

Fig. 5 Memory usage on lymph data set. **a** Frequent itemset generation. **b** Closed frequent itemset generation. **c** Maximal frequent itemset generation. **d** Rare itemset generation. **e** Memory usage of algorithms

(d) *Fourth Category:* Under this category, *RP-Tree* has been found to spent lesser execution time in comparison to other algorithms. Reducing the database size and generating only the rare-item itemsets enabled *RP-Tree* to expend minor execution time. *Apriori Inverse* followed *RP-Tree* generating only the sporadic itemsets below the maximum support threshold. *ARIMA* generating the complete set of rare itemsets has been the most expensive one followed by *MSApriori* and *Apriori Rare*. *Apriori Rare* generating the MRI's managed to display a satisfying performance in terms of execution time.

Memory usage

This section elicits the memory usage of the algorithms. The memory efficiency of each category of algorithms is shown in Fig. 5a–d. Figure 5e on the other hand, examines the memory efficiency of all the algorithms together.

(a) *First Category:* The algorithm that consumed the lowest amount of memory under this category is *FP-Growth*. Just like the previous data set, *Lymph* is also a dense data set that led to the generation of compact and small FP-Trees resulting in less usage of main memory. *H-Mine* consumed a bit more memory than *FP-Growth* as its performance in case of dense data set is not very appreciable as that of sparse data set. The levelwise approaches *Apriori TID*, *Apriori* and *ECLAT* expended more memory as compared to the previous two approaches.
(b) *Second Category: CLOSET* dominated rest of the algorithms under this category due to similar reasons discussed in case of *Zoo* data set. *LCM* maintained its consistency of memory consumption in *Lymph* data set as well. *CHARM* and *FPClose* have been again found to be consuming more memory at low support values due to the retainment of FCI's at every pass. Nonetheless, their memory consumption have been still lower than that of *Apriori Close* as it attempts to store the FMG's in main memory before generating the entire FMG set.
(c) *Third Category:* Memory consumption by *FPMax* algorithm is the lowest one among the MFI generating algorithms due to the construction of small FP-Trees from a comparatively dense data set. Storage of MFI's in main memory by *MAFIA* and *GenMax*, increases the memory overhead for these two algorithms with only a minimal difference between the two. *Max-Miner* again consumed the highest amount of memory among all.
(d) *Fourth Category:* Despite the fact that *ARIMA* is the only algorithm generating the complete set of rare itemsets, it consumed the highest amount of memory particularly due to the storage of MRI's and FG's in main memory. *MSApriori* too consumed lot of memory due to the retainment of rare items along with the frequent ones during itemset generation. *RP-Tree* has shown the best performance among all in *Lymph* data set as well, followed by *Apriori Inverse* with a maximum support threshold of 60%. *Apriori Rare* generating the MRI's performed better than *ARIMA* and *MSApriori* but failed to surpass *RP-Tree* and *Apriori Inverse*.

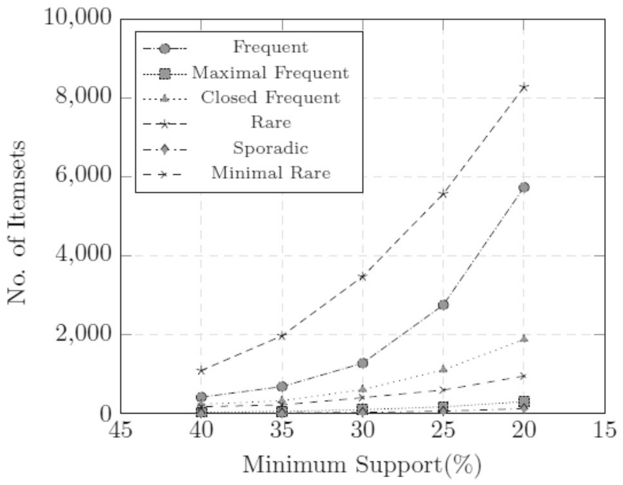


Fig. 6 Itemsets generated in lymph data set

Itemsets generated

Figure 6 illustrates the number of different itemsets generated for *Lymph* data set. The number of different types of itemsets generated in *Lymph* data set is lesser than *Zoo* data set. Rare itemsets generated from this data set is quite higher than that of the frequent itemsets. The closed frequent itemsets are the next in number followed by the MRI's. On the contrary, only a limited number of maximal frequent and sporadic itemsets have been obtained. Figure 6 shows a graphical representation of the different types of itemsets generated for the *Lymph* data set.

Mushroom

This section elicits the performance of the algorithms in terms of execution time, memory usage and number of itemsets generated on *Mushroom* data set.

Execution time

The execution time invested by different categories of algorithms considered in the study is illustrated in this section. Figure 7a to 7e presents a graphical analysis and comparison of the execution time expend by the algorithms.

(a) *First Category:* The same scenario is repeated in this case like the previous two data sets. *FP-Growth* still proved to be the best algorithm for frequent itemset mining among others. *H-Mine* had slightly higher execution time than *FP-Growth* since *Mushroom* is a comparatively denser data set. *Apriori TID* and *Apriori* used the highest execution time as expected due to multiple database scans and generation of candidates. *ECLAT* on the other hand, managed to perform better than these two algorithms.

(b) *Second Category:* In this category of algorithms, *FPClose* surpassed rest of the algorithms at all minimum support values specifically at low supports. Despite showing an appreciable performance, *CLOSET* failed to perform better than *FPClose*. Both the algorithms construct projected FP-Trees for dense data set *Mushroom*. *FPClose* on one hand stores a portion of the FCI's in the generated CFI Trees while *CLOSET* stores all the FCI's in a global prefix tree that made the difference in their performance. *LCM* outperformed *CHARM* due to its efficiency in duplicate detection. *Apriori Close* consumed the highest amount of execution time as predicted.

(c) *Third Category:* The third category of algorithms presented similar type of performances obtained in earlier data sets. *FPMax* performed exceptionally well at all minimum support values. Data set *Mushroom* has long average pattern length and a short average transaction length. Performance of *FPMax* is appreciable in case of data sets having short ATL and long APL. *GenMax* managed to have a decent execution time, slightly higher than *FPMax*. *MAFIA* and Max-Miner still continued to be the expensive ones in this category of algorithms.

(d) *Fourth Category:* Performance analysis of the fourth category of algorithms illustrates that RP-Tree expends the lowest amount of execution time among others due to the generation of only the rare-item itemsets. With a maxsup of 60%, *Apriori Inverse* is the next best algorithm under this category. *Apriori Rare* utilized a decent amount of execution time for the generation of the MRI's. Setting the β value to 0.1, *MS-Apriori* completed its execution before *ARIMA*. *ARIMA* despite generating the complete set of rare itemsets, proved to be the costliest.

Memory usage

Memory efficiency of the algorithms and their comparative scrutiny based on memory usage is demonstrated using graphical analysis from Fig. 8a–e.

(a) *First Category:* The same outstanding performance of *FP-Growth* on dense data sets can be seen in case of *Mushroom* data set as well due to smaller size of the FP-Tree. *H-Mine* have been found to consume little bit more memory than *FP-Growth*. The highest memory consumption was by *Apriori TID* followed by *Apriori*.

(b) *Second Category:* Due to the same reason discussed for previous data sets, *CLOSET* managed to consume the lowest amount of memory among the second category of algorithms. *CHARM* and *LCM* have been consistent in their amount of memory consumption as the earlier data sets. *FP-Close* on the contrary have shown disappointing performance due to its tendency of retaining the FCI's at every pass. *Apriori Close* as expected has consumed the highest amount of memory among all.

(c) *Third Category:* The size of the FP-Tree generated by *FPMax* for *Mushroom* is very small due to the density of the data set. As, a result the memory consumption by this algorithm is quite less compared to other algorithms of the same category, the highest being that of *Max-Miner*. Due to the storage of MFI's in main memory, *MAFIA* and *GenMax* consumed higher memory than *FPMax*.

(d) *Fourth Category: RP-Tree* algorithm under this category consumed the lowest amount of memory which is obvious due to the retainment of only the rare-item itemsets. Followed by *RP-Tree* is the *Apriori Inverse* algorithm that consumed a reasonable amount of memory upon setting the *maxsup* value to 60%. Highest amount of memory is consumed by *ARIMA* which is the only algorithm generating the complete set of rare itemsets. *Apriori Rare* and *MS-Apriori* curtailed the amount of memory consumed to some extent compared to *ARIMA*.

Itemsets generated

The number of itemsets generated under the given threshold for *Mushroom* data set are quite high in number, the highest being that of the rare itemsets. The frequent itemsets generated are slightly less in number than the rare itemsets. The number of closed frequent and maximal frequent items generated are very less, almost negligible in comparison to the rare and frequent itemsets. Figure 9 graphically shows the number of different types of itemsets generated from *Mushroom* data set.

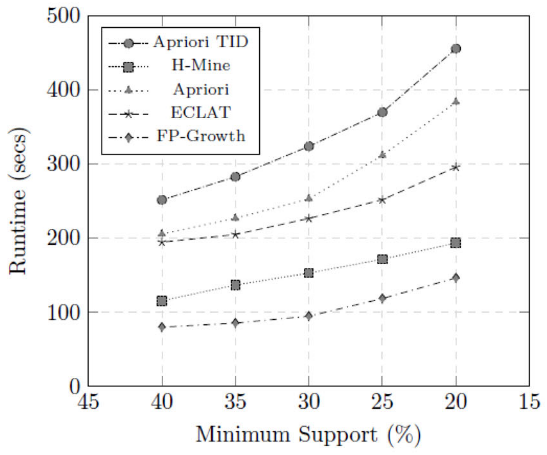
Retail

Detailed analysis based on performance of the algorithms on *Retail* data set is provided in this section.

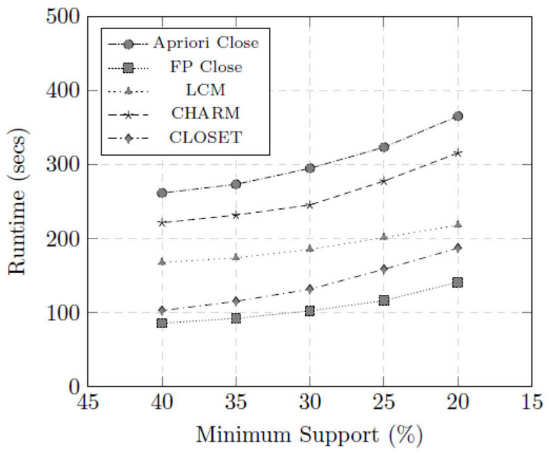
Execution time

The execution time analysis of the algorithms is depicted in this section using graphical analysis from Fig. 10a–e.

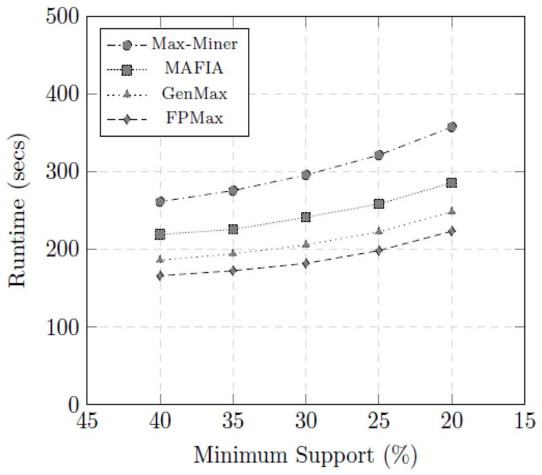
(a) *First Category: FP-Growth* expend the highest amount of execution time for the *Retail* data set. This is obvious as it is a sparse data set containing lesser number of frequent items. A higher amount of execution time



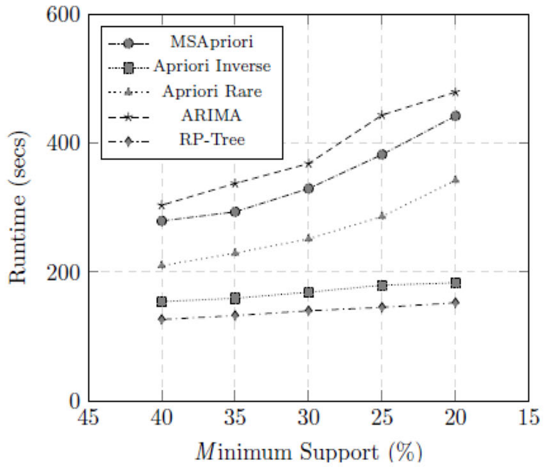
(a) Frequent Itemset Generation



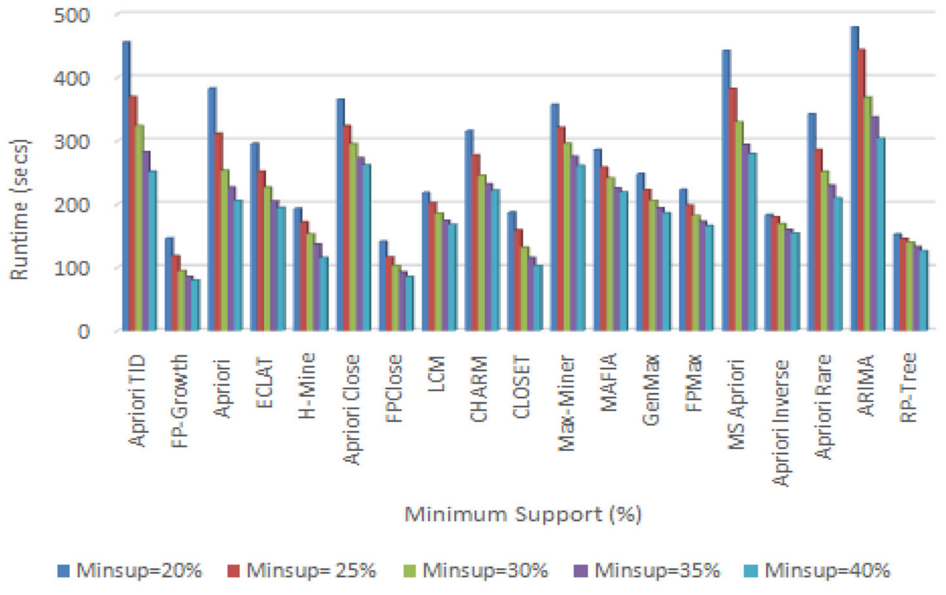
(b) Closed Frequent Itemset Generation



(c) Maximal Frequent Itemset Generation

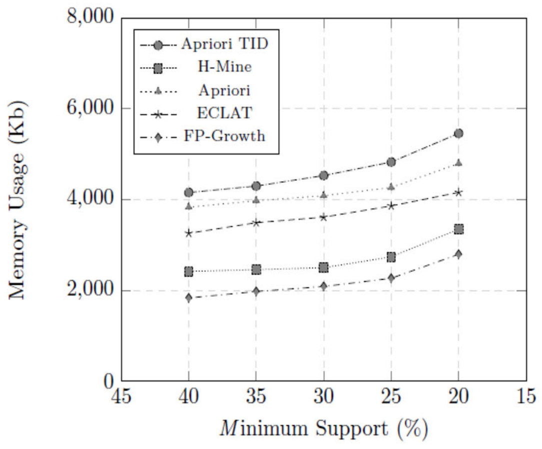


(d) Rare Itemset Generation

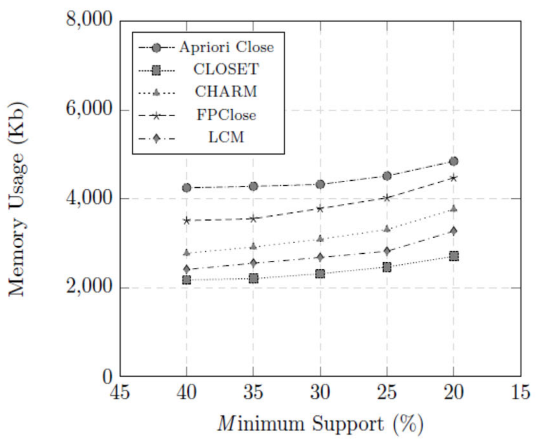


(e) Execution Time of Algorithms

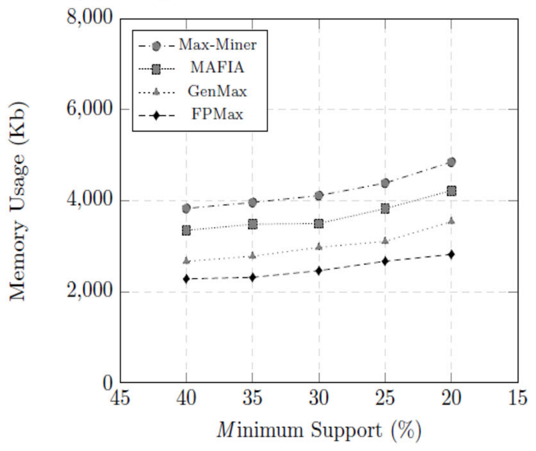
Fig. 7 Execution time on mushroom data set. **a** Frequent itemset generation. **b** Closed frequent itemset generation. **c** Maximal frequent itemset generation. **d** Rare itemset generation. **e** Execution time of algorithms



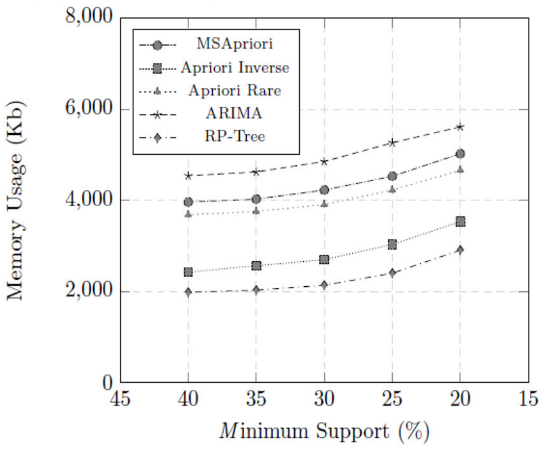
(a) Frequent Itemset Generation



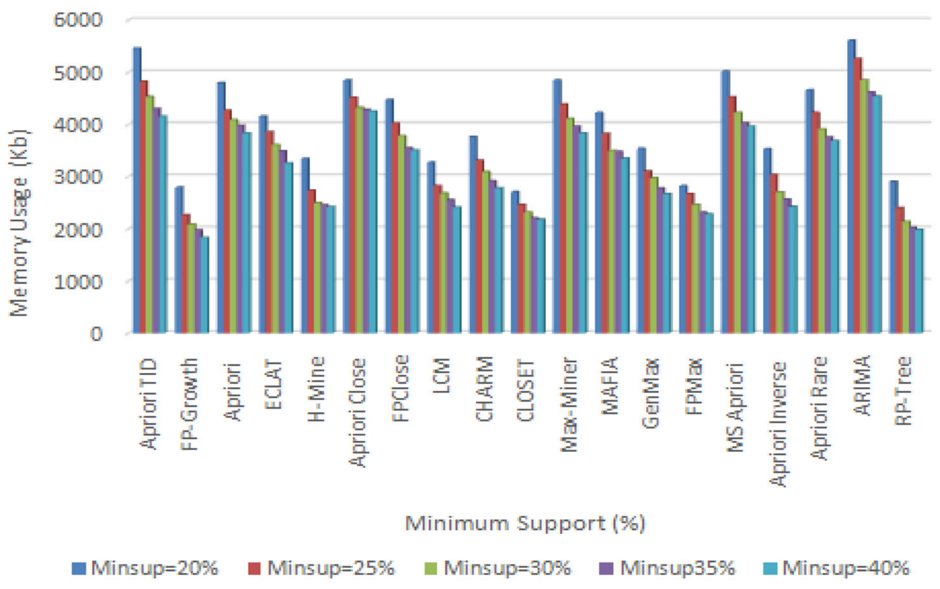
(b) Closed Frequent Itemset Generation



(c) Maximal Frequent Itemset Generation



(d) Rare Itemset Generation



(e) Memory Usage of Algorithms

Fig. 8 Memory usage on mushroom data set. **a** Frequent itemset generation. **b** Closed frequent itemset generation. **c** Maximal frequent itemset generation. **d** Rare itemset generation. **e** Memory usage of algorithms

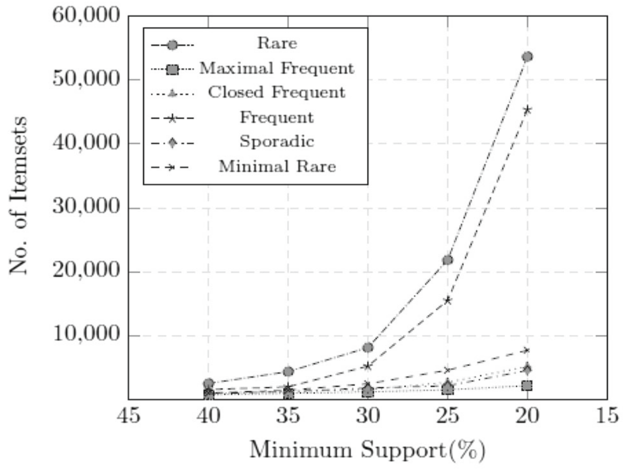


Fig. 9 Itemsets generated in mushroom data set

is invested in the FP-Tree construction and the resulting tree is big and bushy as the amount of node sharing is minimal. *H-Mine* outperformed all the algorithms due to its efficient use of tree data structure. Level-wise approaches like *Apriori*, *Apriori TID* and *ECLAT* performed better than *FP-Growth* as lesser number of frequent patterns and candidates are generated.

(b) *Second Category:* Array based technique employed by *FP-Close* proved to be costlier at lower supports as huge number of itemsets are generated. *CLOSET* performed better than *FP-Close* due to the usage of top-down projection tree that requires the traversal of only the global FP-Tree. The large number of candidates generated increases the closure computation cost and the execution time of *Apriori Close*. This made *Apriori Close* the most expensive algorithm in this category. *LCM* outperforms all the algorithms under this category and the performance gap becomes more evident as support value decreases.

(c) *Third Category: Retail* is a data set having short ATL and the MFI's generated are also very short. The FP-Tree generated by *FP-Max* in this case, is very big and bushy. At high support values, there are only few MFI's. *FP-Max* however, expends a lot of time in the FP-Tree construction and despite that only few MFI's are generated from the FP-Tree. Short ATL of *Retail* however did not affect the bit vector operations performed by *MAFIA* and *GenMax* due to which they performed better than *FP-Max*.

(d) *Fourth Category: ARIMA* proved to be costliest among all for this data set as well. *MS-Apriori* managed to perform better than *ARIMA* like previous cases. *RP-Tree* spent higher execution time for this data set as compared to other data sets due to the construction of costly FP-Trees. *Apriori Inverse* and *Apriori Rare* outperformed others in this category.

Memory usage

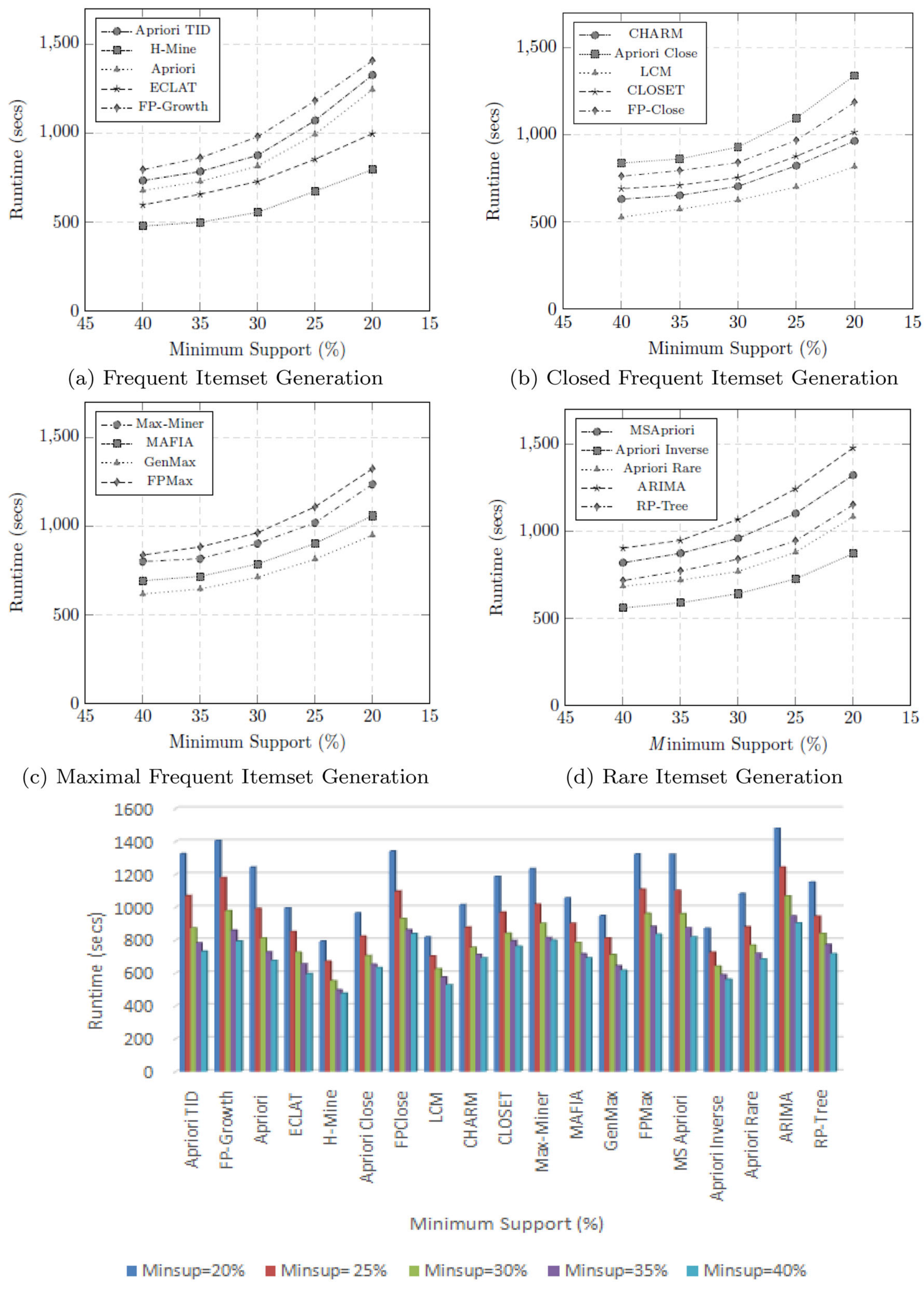
Performance evaluation of the algorithms in terms of memory usage is provided in this section. Figure 11a–e provides a comparative analysis of the algorithms based on their memory efficiency.

(a) *First Category:* The highest amount of memory under this category is consumed by the *FP-Growth* algorithm. As *Retail* is a sparse data set, the amount of node sharing is less due to which large number of nodes are generated in the FP-Tree. Storage of these nodes in main memory proved to be costly for *FP-Growth* in terms of memory usage. However, the number of frequent patterns and hence the candidates generated are less that befitted *AprioriTID* and *Apriori*. *H-Mine* is the clear winner in this case as queue data structures consume less amount of memory.

(b) *Second Category:* The performance of *FPClose* algorithm has been found to be worse among the algorithms under this category. The retainment of recursively built FP-Trees and CFI trees resulted in a high amount of memory consumption. *CHARM* unexpectedly consumed the lowest amount of memory. The memory consumption of *AprioriClose* is due to the retainment of FMG's in main memory obtained using off-line closure computation. Memory consumption remained constant for *LCM* while *CLOSET* consumed slightly higher memory than *Apriori Close*.

(c) *Third Category:* The storage of big and bushy FP-Trees generated by *FPMax* resulted in high memory consumption by the algorithm. *Max-Miner* consumed slightly less memory than *FP-Max*. *GenMax* maintained its consistency in memory consumption and proved to be the best algorithms under this category. The superiority of *GenMax* comes from the fact that it maintains only the local MFI's for comparison rather than the entire set of MFI's. *MAFIA* is the next best algorithm in terms of memory consumption under this category.

(d) *Fourth Category: ARIMA* still consumed the highest amount of memory among the rare itemset generation algorithms due to the retainment of both frequent and rare itemsets followed by *MSApriori*. *RP-Tree* proved to be a little costlier this time as compared to its memory consumption in dense data sets. The reason is same with that of other pattern mining algorithms. *Apriori Rare* consumed less amount of memory this time as compared to *RP-Tree* followed by *Apriori Inverse*.



(a) Frequent Itemset Generation

(b) Closed Frequent Itemset Generation

(c) Maximal Frequent Itemset Generation

(d) Rare Itemset Generation

(e) Execution Time of Algorithms

Fig. 10 Execution time on retail data set. **a** Frequent itemset generation. **b** Closed frequent itemset generation. **c** Maximal frequent itemset generation. **d** Rare itemset generation. **e** Execution time of algorithms

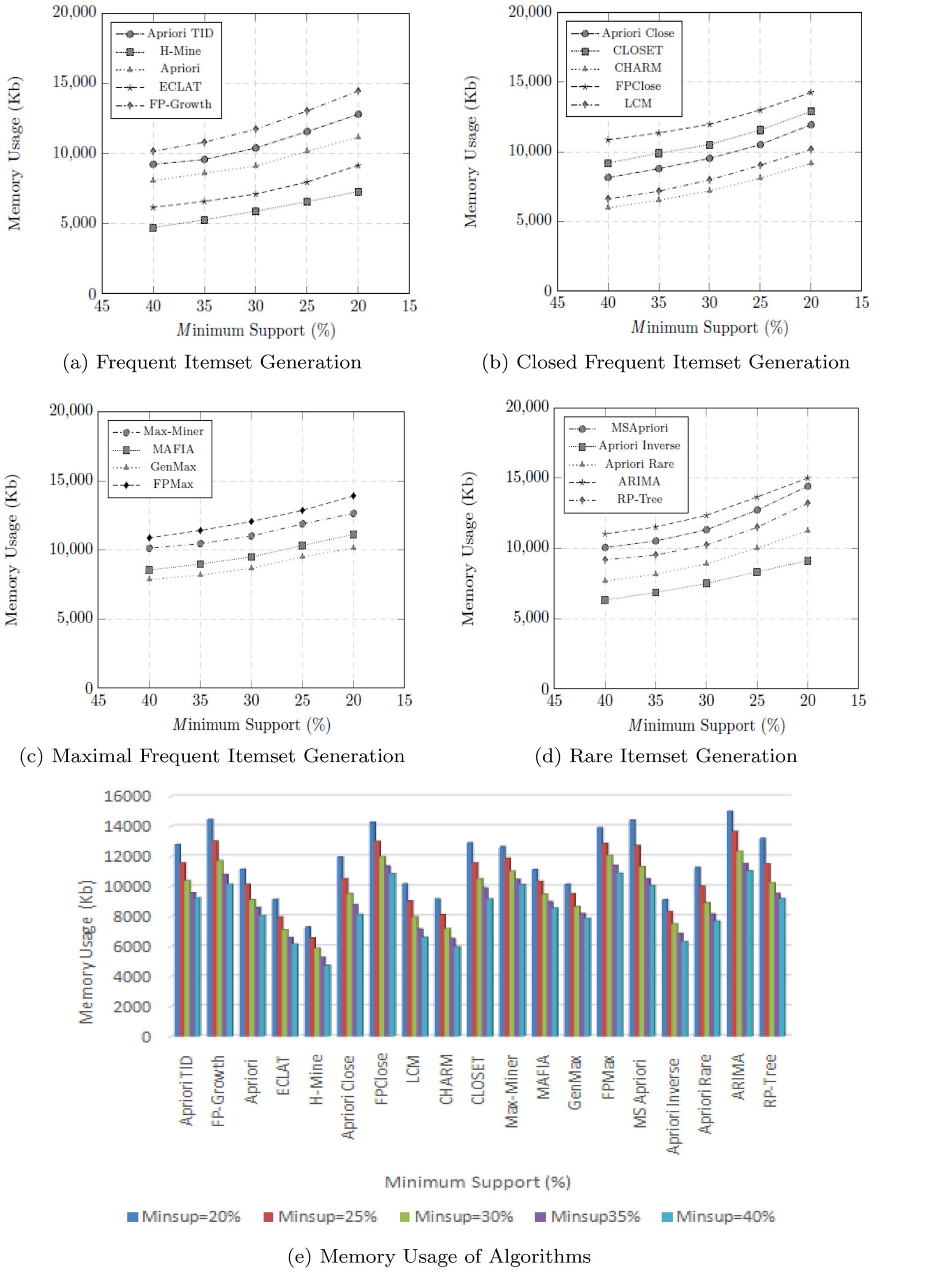


Fig. 11 Memory usage on retail data set. **a** Frequent itemset generation. **b** Closed frequent itemset generation. **c** Maximal frequent itemset generation. **d** Rare itemset generation. **e** Memory usage of algorithms

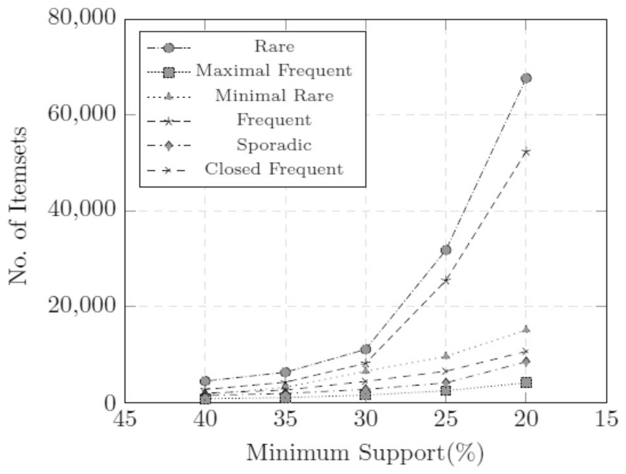


Fig. 12 Itemsets generated in retail data set

Itemsets generated

Retail being a sparse data set has lesser number of frequent itemsets. Thus, as expected the number of rare itemsets generated is highest for this data set. Minimal rare itemsets are the next in number followed by closed frequent itemsets. Only few maximal frequent itemsets and sporadic itemsets are generated for this data set.

T10I4D100K

This section contains performance evaluation of the algorithms on sparse data set, *T10I4D100K* based upon the execution time invested, the amount of memory used and the number of itemsets generated.

Execution time

Execution time spent by the algorithms and the respective comparative analysis is shown in this section with the help of graphical analysis from Fig. 13a–e.

(a) *First Category:* The same trend has been observed for this category of algorithms like the previous case. *H-Mine* maintained its effective performance in case of sparse data set. *FP-Growth* still proved to be the most expensive one among all. *Apriori TID* and *Apriori* were the next in row in terms of performance. *ECLAT* demonstrated consistent performance for this data set as well.

(b) *Second Category: Apriori Close* was the most expensive algorithm under this category of algorithms unlike the *Retail* data set. This is because *Apriori Close* compares the support of each k-candidate itemset generated with the k-1 subsets to find out whether it is an FMG which increase its computational overhead. *FPClose* surprisingly performed better than *CLOSET* due to the usage of an array based implementation avoiding repeated FP-Tree traversal during the construction of header tables for new entries. Among *CHARM* and *LCM*, the former highly outperforms *LCM* specifically at higher support values.

(c) *Third Category:* The performance of *MAFIA* and *GenMax* continued to be consistent for this data set as well. *FPClose* once again failed to impress in terms of performance with *Max-Miner* spending slightly lower execution time than *FPClose*.

(d) *Fourth Category: ARIMA* has been found to compensate its performance in terms of execution time due to the retainment of frequent as well as rare itemsets. *MS-Apriori* invested slightly lower execution time than *ARIMA*. *RP-Tree* spent higher execution time for this data set as compared to dense data sets due to the generation of FP-Trees but displayed feasible execution time as it generates only the rare-item itemsets. *Apriori Inverse* and *Apriori Rare* demonstrated similar performances with a slight advantage to the former one.

Memory usage

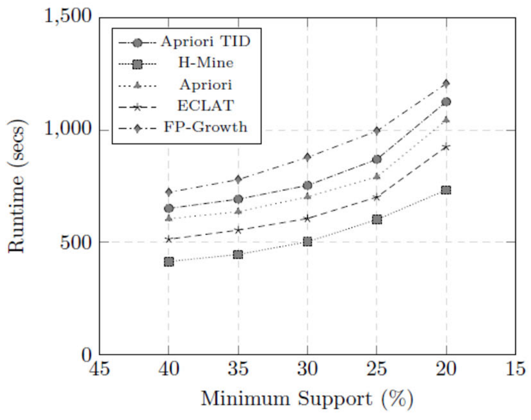
This section illustrates the memory analysis of the algorithms considered in the study from Fig. 14a to e.

(a) *First Category: H-Mine* maintained its appreciable performance for this data set as well. *ECLAT* was slightly expensive than *H-Mine*. *FP-Growth*, as expected proved to be extravagant for this sparse data set. *AprioriTID* and *Apriori* were the second and third expensive algorithms in terms of memory consumption under this category.

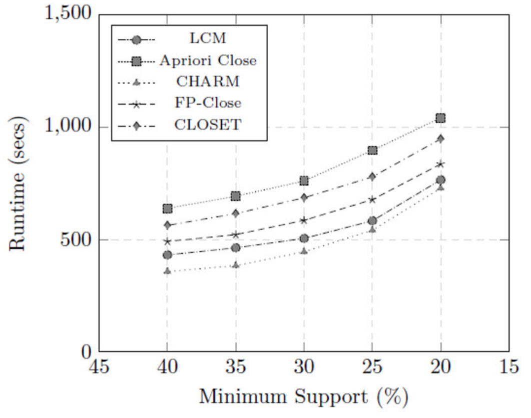
(b) *Second Category:* For this category of algorithms, the scenario is completely opposite to that in *Retail* data set. *AprioriClose* in this case consumed the highest amount of memory rather than *FPClose*. This is because *AprioriClose* attempts to store each extracted FMG in main memory prior generating the whole set of FMG's. On the other hand, the higher memory consumption of *FPClose* comes from the fact that it retains the recursively built FP-Trees and CFI trees in main memory that are quite large in case of sparse data sets. *CHARM* largely outperformed the rest of the algorithms under this category.

(c) *Third Category: FPMax* is the most extravagant algorithm under this category as it needs to store all the FP-Trees and MFI trees generated, in main memory. *Max-Miner* stores every candidate itemset generated due to which its memory consumption considerably increases. *GenMax* outperformed the rest of the algorithms due to retainment of only the set of local MFI"s.

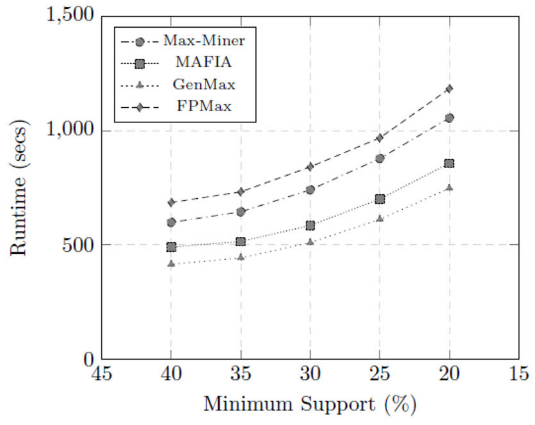
(d) *Fourth Category: ARIMA* stores the frequent itemsets and MRG's which increases its memory overhead.



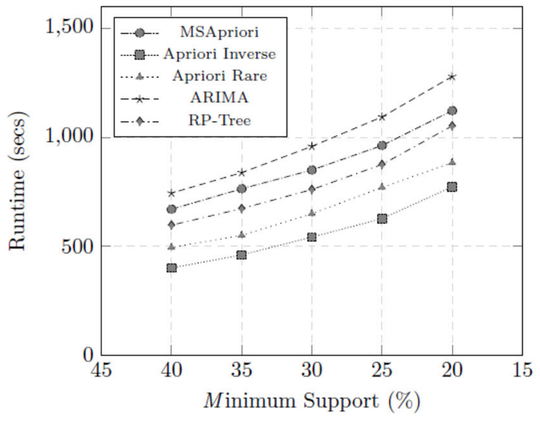
(a) Frequent Itemset Generation



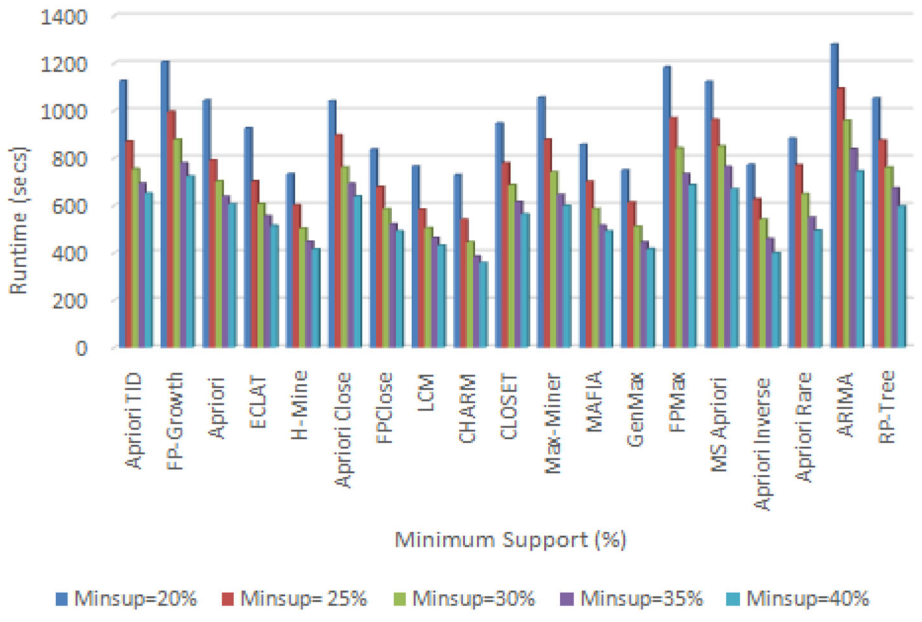
(b) Closed Frequent Itemset Generation



(c) Maximal Frequent Itemset Generation

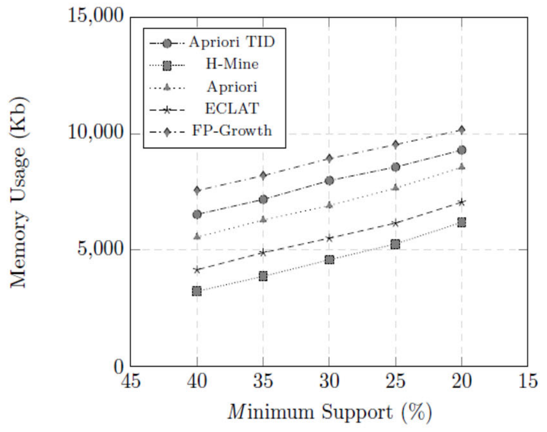


(d) Rare Itemset Generation

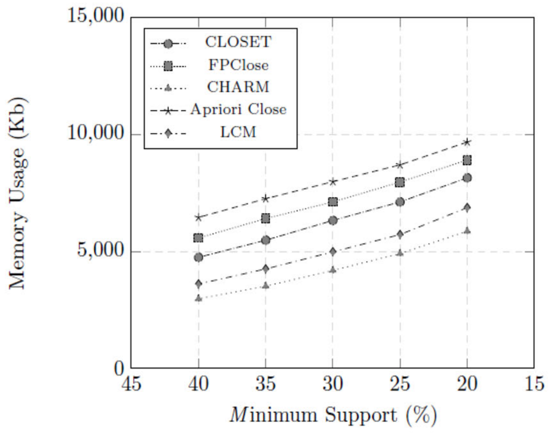


(e) Execution Time of Algorithms

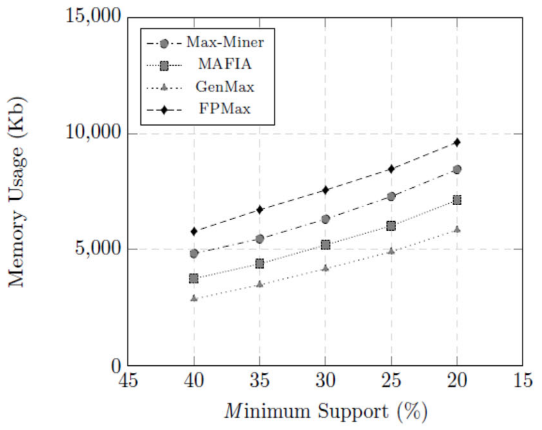
Fig. 13 Execution time on T10I4D100K data set. **a** Frequent itemset generation. **b** Closed frequent itemset generation. **c** Maximal frequent itemset generation. **d** Rare itemset generation. **e** Execution time of algorithms



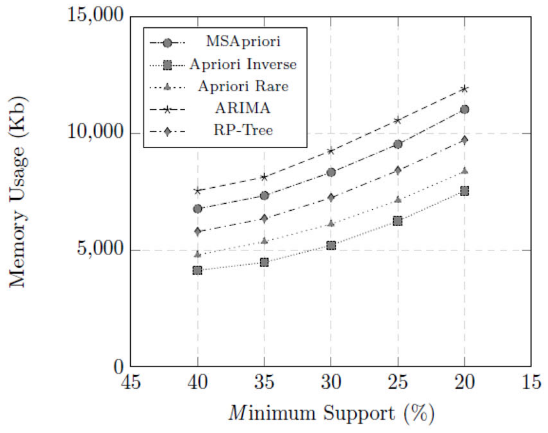
(a) Frequent Itemset Generation



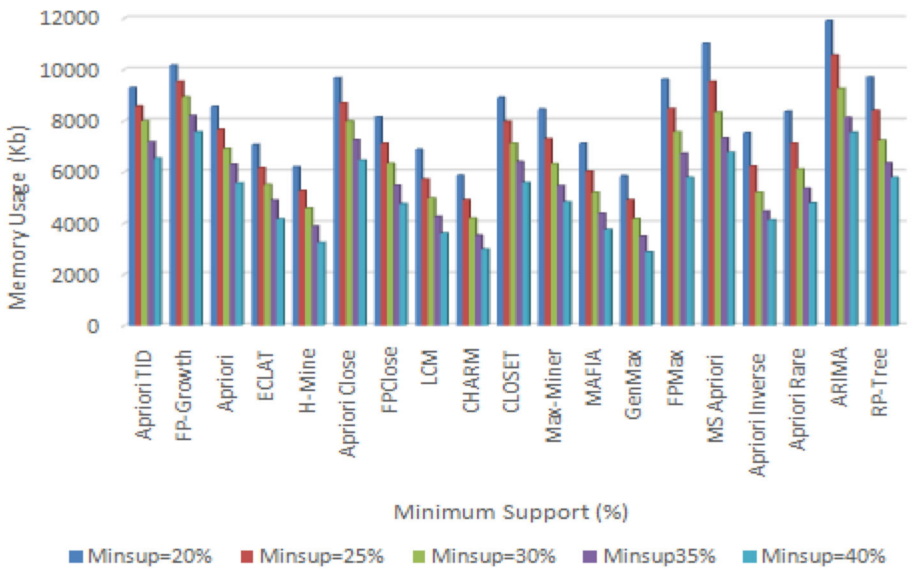
(b) Closed Frequent Itemset Generation



(c) Maximal Frequent Itemset Generation



(d) Rare Itemset Generation



(e) Memory Usage of Algorithms

Fig. 14 Memory Usage on T10I4D100K data set. **a** Frequent Itemset Generation. **b** Closed Frequent Itemset Generation. **c** Maximal Frequent Itemset Generation. **d** Rare Itemset Generation. **e** Memory Usage of Algorithms

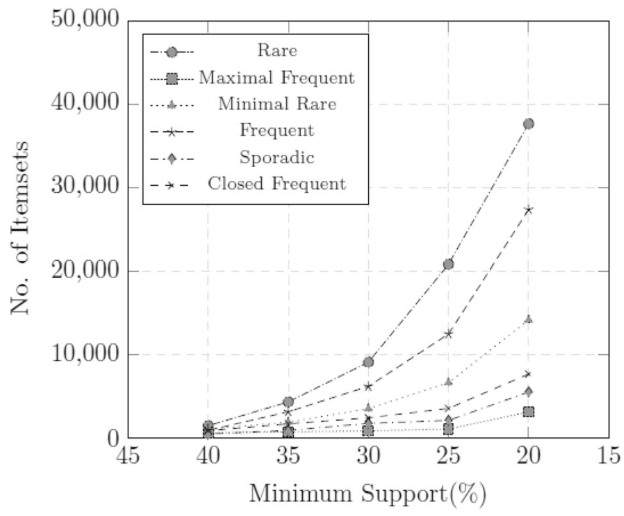


Fig. 15 Itemsets generated in T10I4D100K data set

Retaining the FP-Trees in memory proved to be expensive for *RP-Tree*. The performance of *Apriori Rare* and *Apriori Inverse* remained constant for this data set.

Itemsets generated

Figure 15 illustrates the different types of itemsets generated in T10I4D100K data set. As can be observed in the figure, the highest number of itemsets generated for this data set are the rare itemsets. The frequent itemsets generated are also very high in number but slightly than the number of rare itemsets. The next in number are the minimal rare itemsets followed by few sporadic, closed frequent and maximal frequent itemsets.

T40I10D100K

Analysis of the algorithms based on their performance in data set *T40I10D100K* is given in this section.

Execution time

The amount of time invested by the algorithms in their execution is illustrated in this section through a graphical analysis in Fig. 16a–e.

(a) *First Category:* For this data set, *AprioriTID* largely outperforms *FP-Growth* as the generation of FP-Trees by *FP-Growth* expends a lot of time specifically for sparse data sets. *Apriori* and *ECLAT* performed marginally better than *AprioriTID* with *H-Mine* being the most substantial algorithm under this category.

(b) *Second Category: AprioriClose* spent the highest amount of execution time for reasons similar to the previous data set. The execution time of *CLOSET* is affected by its tendency of preserving all the extracted FCI's in the same tree. *FPClose* under such circumstances performed marginally better than *CLOSET*. The performances of *LCM* and *CHARM* have been found to be similar with slight advantage to *CHARM*.

(c) *Third Category:* Among the third category of algorithms, the most efficient one is again the *GenMax* algorithm since it compares the newly extracted MFI with only the set of local MFI's unlike the other levelwise algorithms that performs the comparison with the previously extracted MFI's. *FPClose* in this case is highly affected by the recursive generation of FP-Trees and MFI trees. *Max-Miner* performed slightly better than *FPClose* followed by *MAFIA*.

(d) *Fourth Category: ARIMA* due to the generation of MRI's and MRG's proved to be expensive even for this data set. *MS-Apriori* too spent a considerable amount of time in determining and assigning MIS values to the items. *Apriori Rare* due to the generation of only MRI's and *Apriori Inverse* due to the generation of only sporadic itemsets for a *Maxsup* value of 60% performed consistently well than other rare itemset generation algorithms. *RP-Tree* however proved to be a little expensive due to generation of large and bushy FP-Trees.

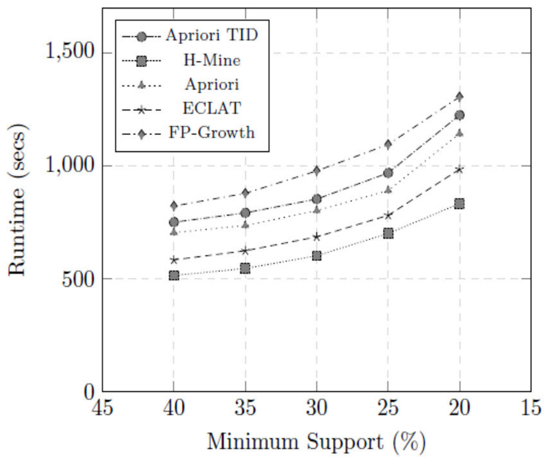
Memory usage

Efficiency of the algorithms in terms of memory usage is depicted in this section with the help of a graphical analysis from Fig. 17a to e.

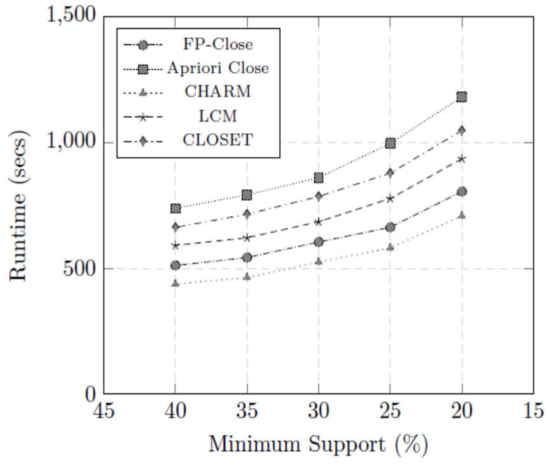
(a) *First Category:* The scenario of memory usage for this category is similar to the previous data set. *FP-Growth* has been again found to be affected by sparseness of the data set where it is obliged to generate large and bushy FP-Trees. Attempt to accommodate the huge number of candidate itemsets generated, proved to be costly for both *AprioriTID* and *Apriori*. However, their performance with respect to *FP-Growth* has been much better as compared to dense data sets. *ECLAT* and *H-Mine* performed exceptionally well with slight advantage to *H-Mine* that outperformed all the algorithms.

(b) *Second Category: CHARM* prove to be most memory efficient among the other algorithms. Memory consumption of *CLOSET* and *FPClose* tends to increase at lower *minsup* values when the number of FCI's increases. *AprioriClose* consumed the highest amount of memory in this category. Memory requirement of *LCM* however, remained mostly constant.

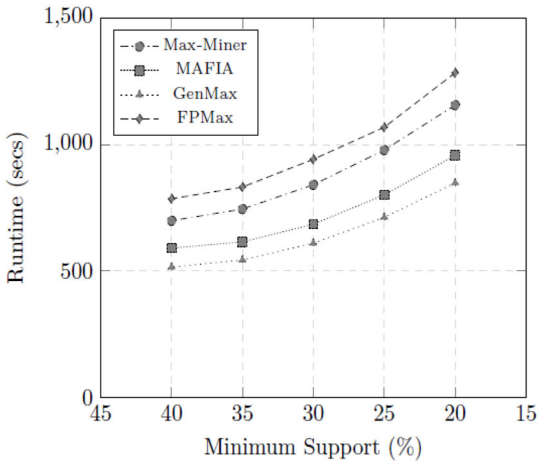
(c) *Third Category:* The amount of memory expended by *FPMax* was considerably higher for this data set. *Max-Miner* too consumed a higher amount of memory with only a minimal difference to that *FPMax*. *MAFIA* and



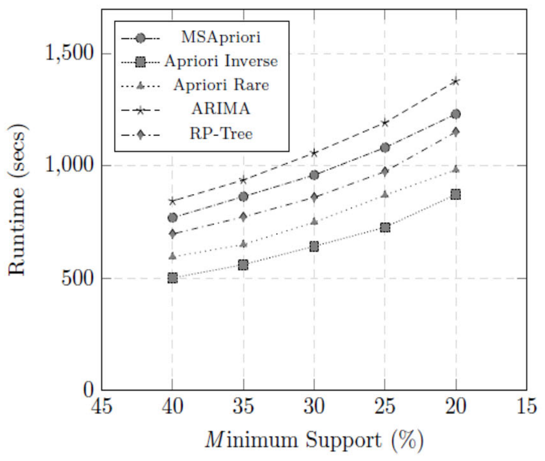
(a) Frequent Itemset Generation



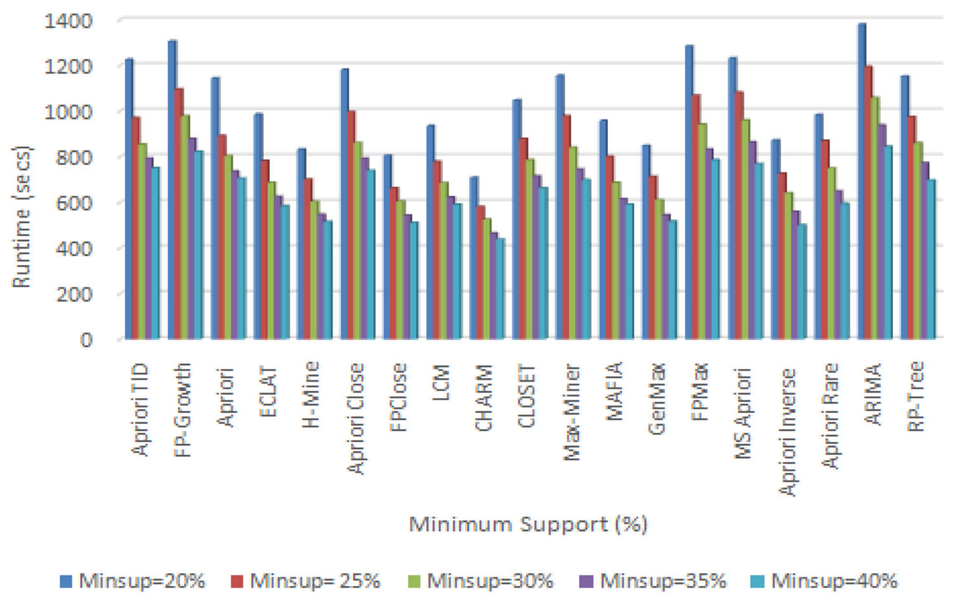
(b) Closed Frequent Itemset Generation



(c) Maximal Frequent Itemset Generation



(d) Rare Itemset Generation



(e) Execution Time of Algorithms

Fig. 16 Execution time on T40I10D100K data set. **a** Frequent Itemset Generation. **b** Closed Frequent Itemset Generation. **c** Maximal Frequent Itemset Generation. **d** Rare Itemset Generation. **e** Execution Time of Algorithms

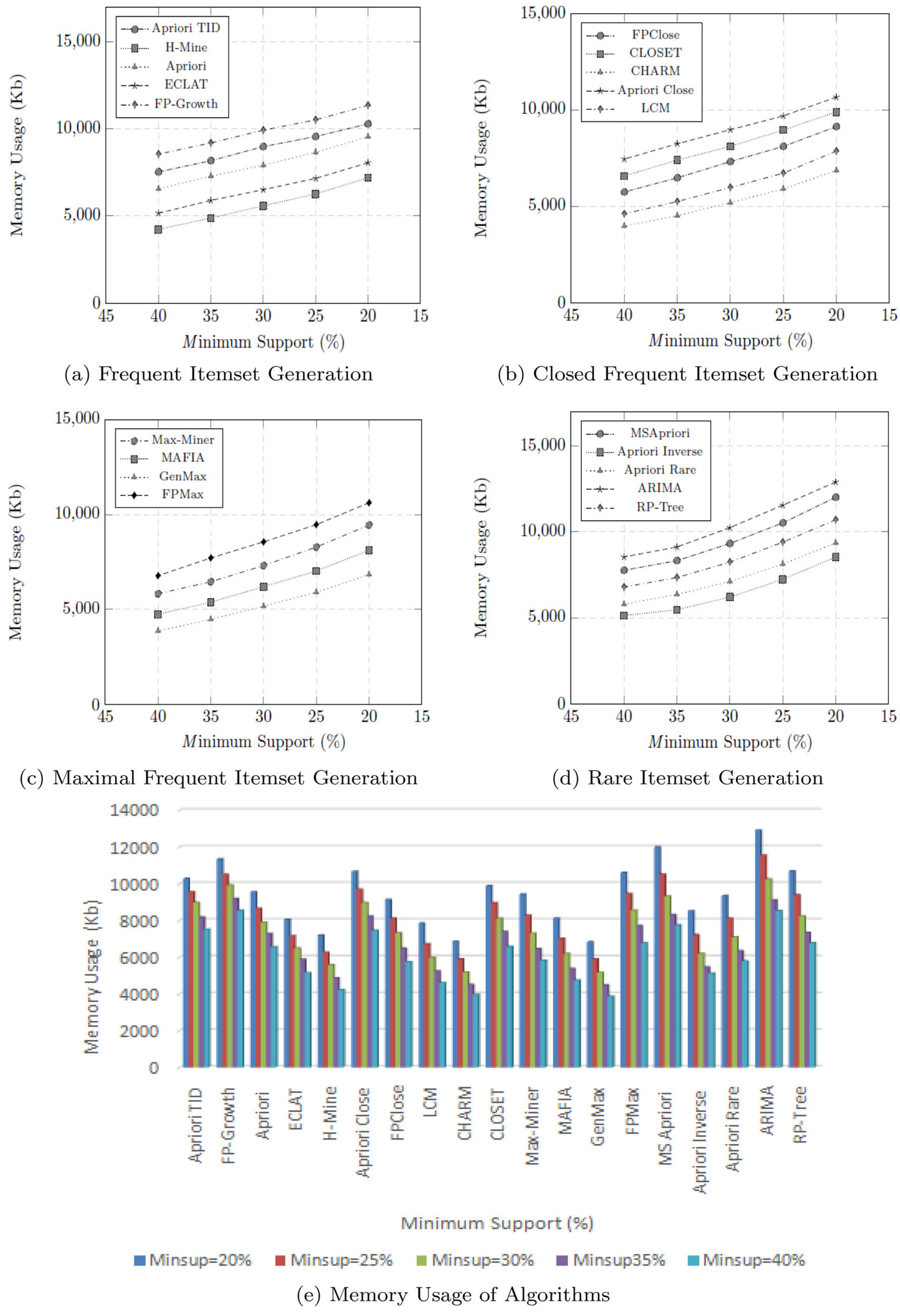


Fig. 17 Memory Usage on T40I10D100K data set. **a** Frequent itemset generation. **b** Closed frequent itemset generation. **c** Maximal frequent itemset generation. **d** Rare itemset generation. **e** Memory usage of algorithms

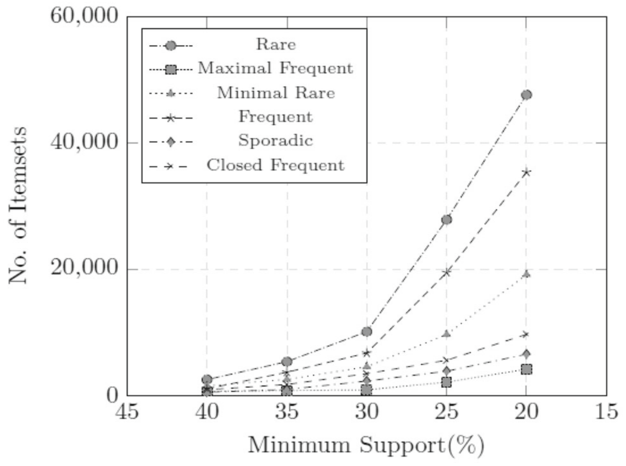


Fig. 18 Itemsets generated in T40I10D100K data set

GenMax continued to perform well for this data set as well.

(d) *Fourth Category: Apriori Rare* and *Apriori Inverse* were the most memory efficient in this category of algorithms. *ARIMA* consumed the highest amount of memory followed by *MSApriori*. *RP-Tree* due to the retainment of only rare-item itemsets in the FP-Tree, performed better than *ARIMA* and *MSApriori*.

Itemsets generated

An illustration of different types of itemsets generated in this data set is shown in Fig. 18. The scenario is similar to the previous data set where the number of rare itemsets is quite large compared to other types of itemsets. Marginal number of frequent itemsets have been generated followed by the minimal rare itemsets. On the contrary, only few sporadic, closed frequent and maximal frequent itemsets have been produced.

Discussion

We performed the first three set of experiments on dense real data sets: *Zoo, Lymph* and *Mushroom*. The results obtained were similar for all the three dense data sets. For the first category of algorithms generating frequent itemsets, the performance of *FP-Growth* has been found to be the best among all. Efficiency of the algorithm in terms of execution time and memory usage is quite impressive in case of dense data sets. However, for very large data sets, the algorithm may run out of memory. In such cases, an existing scalable variant of the algorithm can be employed. Thus for extracting frequent itemsets from data sets where there is a dense distribution of data, *FP-Growth* can be preferred. For the second category, exploiting *FPClose* for generating closed frequent itemsets can be a good option considering its exceptional performance on dense data sets. Even though *FPClose* outperformed *LCM* by a slight difference, it is worth mentioning that the execution time and memory usage of *LCM* remained constant throughout, which is quite appreciable. Among the algorithms in the third category, *FPMax* performed well due to long ATL and short APL of the three dense data sets. However, *FPMax* fail to show appreciable performance for data sets having long ATL and short APL as well as long ATL and long APL. This is because when length of the transaction is very high, then *FPMax* spends huge amount of time in the construction of MFI-Tree and the resulting tree will also be large and bushy. In such cases, *MAFIA* and *GenMax* are expected to outperform *FPMax* since the cost of bit-vector operations and set intersections will be less. For the fourth category of algorithms, *RP-Tree* is the best option if the users are interested only in the rare-item itemsets as it failed to generate the complete set of rare items. For generating the rare itemsets, *ARIMA* even though expensive is preferred as it is the only algorithm that is capable of generating frequent itemsets as well as the complete set of rare itemsets.

The next three set of experiments were conducted on one real and two synthetic sparse data sets: *Retail, T10I4D100K* and *T40I10D100K*. *H-Mine* performed exceptionally well among the algorithms in the first category for all the three sparse data sets. *H-Mine* overcomes the poor performance of *FP-Growth* on sparse data sets. For the second category of algorithms, performances of *LCM* and *CHARM* were very close to each other. *LCM* proved to be the most efficient one for *Retail* data set while *CHARM* was better than others in case of the two synthetic data sets. Among the algorithms in the third category, *FPMax* failed to perform like in case of dense data sets. This is because *Retail* and the other two synthetic data sets have short ATL and a relatively short APL. *FPMax* spends a lot of time in the FP-Tree construction even though only a small number of MFI's are generated from the sparse data sets. Bit-vector operations performed by *GenMax* however, remains unaffected by short ATL of these sparse data sets due to which it performed the best among all the algorithms. In case of the rare itemset generation algorithms, *Apriori Rare* generating only MRI's and *Apriori Inverse* generating only sporadic itemsets, invested the lowest amount of time. *RP-Tree* failed to impress in case of sparse data sets despite generating only a subset of rare itemsets due to the generation of big and bushy FP-Trees.

The results obtained for all the data sets in the four different categories have been summarized in Table 3. The table elicits the best performing algorithm in terms of execution time and memory usage in each category of pattern mining techniques.

From Table 3, it can be observed that among the frequent pattern mining techniques, FP-Growth proved to be the best

Table 3 Performance comparison of all categories of pattern mining techniques

Data set	Frequent itemset		Closed frequent itemset		Maximal frequent itemset		Rare itemset	
	Execution time	Memory Usage	Execution time	Memory usage	Execution time	Memory usage	Execution time	Memory usage
Zoo	FP-Growth	FP-Growth	FP-Close	CLOSET	FPMax	FPMax	RP-Tree	RP-Tree
Lymph	FP-Growth	FP-Growth	FP-Close	CLOSET	FPMax	FPMax	RP-Tree	RP-Tree
Mushroom	FP-Growth	FP-Growth	FP-Close	CLOSET	FPMax	FPMax	RP-Tree	RP-Tree
Retail	H-Mine	H-Mine	LCM	CHARM	GenMax	GenMax	Apriori Rare	Apriori Rare
T10I4D100K	H-Mine	H-Mine	CHARM	CHARM	GenMax	GenMax	Apriori Rare	Apriori Rare
T40I10D100K	H-Mine	H-Mine	CHARM	CHARM	GenMax	GenMax	Apriori Rare	Apriori Rare

algorithm in terms of execution time as well as memory usage for all the dense data sets: Zoo, Lymph and Mushroom. The performance of FP-Growth in case of sparse data sets is not very convincing. Its performance in case of sparse data sets can be enhanced by adopting some mechanism to make the FP-Tree structure more compact or using any other supporting data structure for storing the patterns. For the sparse data sets: Retail, T10I4D100K and T40I10D100K performance of H-Mine is the best among all the frequent pattern mining techniques. To improve its performance in case of dense data sets, it is beneficial to swap its queue data structure to FP-tree since FP-tree's compression by common prefix path sharing and then mining on the compressed structures will be more efficient compared to the queue data structure. Among the closed frequent pattern mining techniques, FP-Close emerged as the best algorithm in terms of execution time while the performance of CLOSET was identified to be the best in terms of memory usage for the dense data sets. Performance of FPClose in terms of memory usage can be improved if it avoids retaining the FCI's at every pass. For the sparse data sets, CHARM was the clear winner in terms of both execution time and memory usage. Only in case of Retail data set, performance of LCM was found to be slightly better than CHARM. Among the maximal frequent pattern mining techniques, FPMax was the best algorithm in terms of execution time and memory usage for all the dense data sets while GenMax emerged as the best algorithm for all the sparse data sets. RP-Tree proved to be the best algorithm among the rare pattern mining techniques for dense data sets and Apriori Rare demonstrated the best performance for sparse data sets in terms of both execution time and memory usage. Performance of RP-Tree can be improved for sparse data sets by employing additional data structures for pattern storage or by compressing the size of the tree structure used. Apriori Rare on the other hand, needs to employ data structures that can guarantee limited storage space for the patterns generated in order to handle the dense data sets efficiently.

Threats to validity

In order to gauge the quality of work, it is necessary to consider the threats to validity of the study [5]. This section discusses the threats to validity of this study. The following threats to validity of the study need to be considered:

(a) **Threshold range:** The results of study and the performance of algorithms are completely dependent on the value of support thresholds used. The range of values chosen for support thresholds are user defined and change in values might affect the final results of the study.
(b) **Data set reliability:** The data sets used in this study are the ones widely used by the techniques considered

for experimentation. The results obtained and the behavior of algorithms might vary from one data set to other depending on the data set characteristics. It might be possible that the findings may vary for some data sets not being used in the study.

(c) **Limited data sample size:** Due to hardware and software limitations, the data sets considered are not very large. Larger data sets could undermine the validity of the results obtained in this study.

(d) **Lack of good descriptive statistics:** The results shown in this study are basically the average of observed results or average of multiple runs. However, measure of variation of the observed results, such as their standard deviation, minmax range or a box-plot, are not presented as the study is primarily based on experimental evaluation.

(e) **Lack of discussion on code instrumentation:** The publicly available source code or implementations used in the experiments may hide specific tweaks or instrumentations to favor certain instances or algorithms, thus influencing the observed results.

(f) **Lack of evaluations for instances of growing size and complexity:** The algorithms considered for evaluation in this study may have been designed to handle data sets that may vary in size and complexity. The evaluation of the algorithms should have been done across a breadth of problem instances, both varying in size and complexity, to provide an assessment on their limits to handle different instances.

Conclusion

Through this paper, we attempted to provide a structural and analytical comparative scrutiny of some of the widely referenced pattern mining techniques for guiding the researchers in making the most appropriate selection for data analysis using pattern mining. The prime focus of this comparative study is to enable the researchers gain an insight into the performance and respective pros and cons of the fundamental pattern mining techniques through a detailed theoretical and empirical analysis.

Initially, we provide a structural classification of the pattern mining techniques in four different categories based upon the type of itemsets generated and a theoretical comparison of all these algorithms specifying their merits and demerits. To gauge the performance of the concerned algorithms, we presented an empirical analysis based on several decisive parameters. We performed experiments using three real and three synthetic benchmark data sets. Both dense and sparse data sets 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 number of itemsets generated by the algorithms.

None of the algorithm however, can be termed the best in its category for all data characteristics under different conditions. The selection of an algorithm in a particular category depends upon the characteristic of data and the requirement of user. There are certain challenging issues faced by the algorithms in each category that affects their performance. Among the frequent pattern mining techniques, Apriori, Apriori TID and ECLAT suffers from the drawback of huge execution time and memory consumption due to their approach of levelwise search in finding the frequent itemsets. H-Mine despite being less expensive in terms of execution time and memory consumption, is only suitable for sparse data sets. FP-Growth is the winner among all when it comes to dense data sets, but for large data sets the FP-Tree structure might not get accommodated in main memory. For the closed frequent pattern mining techniques, the variant of FP-Growth, FPClose suffers from the same drawback. Apriori Close on the other hand, proves to be costly while mining long patterns. LCM and CLOSET does not work well at lower support thresholds while CHARM fails to show good performance at higher support thresholds. Maximal frequent pattern mining technique Max-Miner suffers from the same drawback of high execution time and memory consumption like other levelwise search approaches. FPMax is not suitable for large data sets as the generated FP-Tree may not fit into main memory. MAFIA and GenMax perform expensive bit vector operations due to which they are not suitable for databases having short itemsets. Among the rare pattern mining techniques, MS Apriori expends huge execution time and memory and also requires the identification of an extra parameter β to identify the rare itemsets. Apriori Inverse and Apriori Rare can generate only the minimal rare itemsets. ARIMA despite generating the complete set of rare itemsets, proves to be costly in terms of execution time and memory usage. RP-Tree being a tree based approach performed better than all the algorithms, but cannot generate the complete set of rare itesmets.

The experimental study indicates that performance of the algorithms is greatly affected by the characteristics of data sets used. An in depth study of the data set characteristics can be an important aspect of research in the field of pattern mining. To better understand the behavior of algorithms, a structural characterization of the data sets need to be done beyond the number of transactions/items, the average transaction size or the density. It has been further observed that average transaction length and average pattern length are some other factors that have an important impact on algorithm performances. In this regard, an important research direction would be to perform a detailed analysis of how and why these factors affect the performances of algorithms. Finding effective and costless metrics to analyze the performance of algorithms can be another future perspective to work on.

Conflict of interest None of the authors have any competing interests.

References

1. Adda M, Wu L, Feng Y (2007) Rare itemset mining. In: Machine Learning and Applications, 2007. ICMLA 2007. Sixth International Conference on, IEEE, pp 73–80
2. Aggarwal A, Toshniwal D (2019) Frequent pattern mining on time and location aware air quality data. IEEE Access 7:98,921–98,933
3. Agrawal R, Srikant R (1995) Mining sequential patterns. In: Data Engineering, 1995. Proceedings of the Eleventh International Conference on, IEEE, pp 3–14
4. 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
5. Ali S, Briand LC, Hemmati H, Panesar-Walawege RK (2009) A systematic review of the application and empirical investigation of search-based test case generation. IEEE Trans Software Eng 36(6):742–762
6. Bayardo RJ Jr (1998) Efficiently mining long patterns from databases. ACM Sigmod Record 27(2):85–93
7. Bhatt U, Patel P (2015) A novel approach for finding rare items based on multiple minimum support framework. Proc Comput Sci 57:1088–1095
8. 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
9. 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
10. Borah A, Nath B (2017c) Rare association rule mining: a systematic review. Int J Knowl Eng Data Min 4(3–4):204–258
11. 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
12. Borah A, Nath B (2018b) Identifying risk factors for adverse diseases using dynamic rare association rule mining. Expert Syst Appl 113:233–263
13. Borah A, Nath B (2018c) Rare association rule mining from incremental databases. In: Pattern analysis and applications, pp 1–22
14. Borah A, Nath B (2019a) Incremental rare pattern based approach for identifying outliers in medical data. Appl Soft Comput 85(105):824
15. Borah A, Nath B (2019b) Performance analysis of tree-based approaches for pattern mining. In: Computational intelligence in data mining, Springer, pp 435–448
16. Borah A, Nath B (2019c) Rare pattern mining: challenges and future perspectives. Complex & Intelligent Systems 5(1):1–23
17. Borah A, Nath B (2019d) Tree based frequent and rare pattern mining techniques: a comprehensive structural and empirical analysis. SN Appl Sci 1(9):972
18. Burdick D, Calimlim M, Gehrke J (2001) Mafia: A maximal frequent itemset algorithm for transactional databases. In: Data Engineering, 2001. Proceedings. 17th International Conference on, IEEE, pp 443–452
19. Dua D, Graff C (2017) UCI machine learning repository. http://archive.ics.uci.edu/ml
20. Goethals B (2003) Survey on frequent pattern mining. Univ of Helsinki 19:840–852
21. Gouda K, Zaki MJ (2005) Genmax: an efficient algorithm for mining maximal frequent itemsets. Data Min Knowl Disc 11(3):223–242
22. Grahne G, Zhu J (2003a) Efficiently using prefix-trees in mining frequent itemsets. In: FIMI, vol 90
23. Grahne G, Zhu J (2003b) High performance mining of maximal frequent itemsets. In: 6th International workshop on high performance data mining
24. Han J, Pei J, Yin Y, Mao R (2004) Mining frequent patterns without candidate generation: a frequent-pattern tree approach. Data Min Knowl Disc 8(1):53–87
25. Han M, Ding J, Li J (2017) Tdmcs: an efficient method for mining closed frequent patterns over data streams based on time decay model. Int Arab J Inf Technol 14(6):851–860
26. Koh YS, Rountree N (2005) Finding sporadic rules using apriori-inverse. In: Advances in knowledge discovery and data mining, Springer, pp 97–106
27. Lee G, Yun U, Ryu KH (2014) Sliding window based weighted maximal frequent pattern mining over data streams. Expert Syst Appl 41(2):694–708
28. Li Y, Xu J, Yuan YH, Chen L (2017) A new closed frequent itemset mining algorithm based on gpu and improved vertical structure. Concurr Comput Pract Expe 29(6):e3904
29. Liu B, Hsu W, Ma Y (1999) Mining association rules with multiple minimum supports. In: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, pp 337–341
30. Min F, Zhang ZH, Zhai WJ, Shen RP (2020) Frequent pattern discovery with tri-partition alphabets. Inf Sci 507:715–732
31. Nam H, Yun U, Yoon E, Lin JCW (2020) Efficient approach for incremental weighted erasable pattern mining with list structure. Expert Syst Appl 143(113):087
32. 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
33. Pei J, Han J, Mao R et al (2000) Closet: an efficient algorithm for mining frequent closed itemsets. ACM SIGMOD workshop on research issues in data mining and knowledge discovery 4:21–30
34. 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, 2001. ICDM 2001, Proceedings IEEE International Conference on, IEEE, pp 441–448
35. Pyun G, Yun U, Ryu KH (2014) Efficient frequent pattern mining based on linear prefix tree. Knowl-Based Syst 55:125–139
36. Rodríguez-González AY, Lezama F, Iglesias-Alvarez CA, Martínez-Trinidad JF, Carrasco-Ochoa JA, de Cote EM (2018) Closed frequent similar pattern mining: reducing the number of frequent similar patterns without information loss. Expert Syst Appl 96:271–283

37. Shahbazi N, Soltani R, Gryz J, An A (2016) Building fp-tree on the fly: Single-pass frequent itemset mining. In: Machine learning and data mining in pattern recognition, Springer, pp 387–400
38. Sinthuja M, Puviarasan N, Aruna P (2019) Mining frequent itemsets using proposed top-down approach based on linear prefix tree (td-lp-growth). In: International conference on computer networks and communication technologies, Springer, pp 23–32
39. Szathmary L, Napoli A, Valtchev P (2007) Towards rare itemset mining. In: Tools with artificial intelligence, 2007. ICTAI 2007. 19th IEEE International Conference on, IEEE, vol 1, pp 305–312
40. Szathmary L, Valtchev P, Napoli A (2010) Generating rare association rules using the minimal rare itemsets family. Int J Softw Inf (IJSI) 4(3):219–238
41. Tanbeer SK, Ahmed CF, Jeong BS, Lee YK (2009) Efficient single-pass frequent pattern mining using a prefix-tree. Inf Sci 179(5):559–583
42. 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
43. Tsang S, Koh YS, Dobbie G (2011) Rp-tree: rare pattern tree mining. In: Data warehousing and knowledge discovery, Springer, pp 277–288
44. Uno T, Asai T, Uchida Y, Arimura H (2003) Lcm: An efficient algorithm for enumerating frequent closed item sets. In: FIMI, vol 90
45. Vo B, Pham S, Le T, Deng ZH (2017) A novel approach for mining maximal frequent patterns. Expert Syst Appl 73:178–186
46. 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
47. Yan D, Qu W, Guo G, Wang X (2020) Prefixfpm: a parallel framework for general-purpose frequent pattern mining. In: 2020 IEEE 36th international conference on data engineering (ICDE), IEEE, pp 1938–1941
48. Yun U, Lee G, Ryu KH (2014) Mining maximal frequent patterns by considering weight conditions over data streams. Knowl-Based Syst 55:49–65
49. Zaki MJ (1997) Fast mining of sequential patterns in very large databases. University of Rochester Computer Science Department, New York
50. Zaki MJ, Hsiao CJ (2002) Charm: an efficient algorithm for closed itemset mining. SDM SIAM 2:457–473
51. Zheng Z, Kohavi R, Mason L (2001) Real world performance of association rule algorithms. In: Proceedings of the 7th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, pp 401–406