Mining Frequent Patterns Based on Data Characteristics

Lan Vu, Gita Alaghband, Senior *Member, IEEE*
Department of Computer Science and Engineering, University of Colorado Denver, Denver, CO, USA
{lan.vu, gita.alaghband}@ucdenver.edu

Abstract - *Frequent pattern mining is crucial part of association rule mining and other data mining tasks with many practical applications. Current popular algorithms for frequent pattern mining perform differently: some are good for dense databases while the others are ideal for sparse ones. In our previous research, we developed a new frequent pattern mining algorithm named FEM that runs fast on both sparse and dense databases. FEM combines the mining strategies of FP-growth and Eclat and given a user-specified threshold it adapts its mining behaviors to the data characteristics to efficiently find all short and long patterns from different database types. However, for best performance of FEM, an appropriate threshold value used to control the switching between its two mining tasks need to be selected by the user. In this paper, we present DFEM, an improved algorithm of FEM that automatically adopts a runtime dynamic threshold to better fit to the characteristics of the databases. The experimental results show that DFEM outperforms FEM and other popular frequent pattern mining algorithms including Apriori, Eclat, FP-growth on both sparse and dense databases.*

Keywords: data mining, frequent pattern mining, association rule mining, frequent itemset, transactional database.

1 Introduction

Frequent pattern mining is a fundamental task in data mining which is used to find many types of relationships among variables in large databases such as associations [1], correlations [2], causality [3], sequential patterns [4], episodes [5] and partial periodicity [6]. Moreover, it helps in data indexing, classification, clustering, and other data mining tasks as well [7]. Thus, frequent pattern mining has become a focused research with numerous practical applications including consumer market-basket analysis, web mining, similarity search of complex structured data, network intrusion detection and many others [7], [8].

The frequent pattern mining problem aims to search for groups of itemsets, subsequences, or substructures that co-occur in a database with their frequency no less than a user-specified minimum support threshold. For example, a set of items (itemset), such as milk and bread that appear frequently together in a database is a frequent itemset or frequent pattern. In a typical transactional database, the number of distinct single items and their combinations are usually very large. For a small minimum support threshold, the number of generated itemsets can be extremely large. Hence, it is a great challenge to design algorithms for mining frequent patterns that scale with memory size and run in reasonable time [9]. For this reason, many methods for mining frequent patterns have been developed in last two decades [7], e.g Apriori [1], Eclat [10], FP-growth [11], direct hashing and pruning (DHP) [12], sampling technique [13], dynamic itemset counting (DIC) [14], AIM [15], mining using diffsets [16], technique to reduce the FP-tree traversal time [18], H-mine [19] and nonordfp [20]. Among those, Apriori, Eclat and FP-growth are the most well-known and widely used. The efficiency of these three algorithms has been demonstrated by many researchers. However, performance of these algorithms is significantly different for different database types. For example, Eclat works well on dense databases [15], [21], [22] while FP-growth runs faster on sparse ones [11], [17], [18], [19], [20], [23]. Therefore, it is difficult to select a suitable algorithm for specific applications. In addition, for commercial database systems like Oracle RDBMS, MS. SQL Server and IBM DBS2 and statistical software like R, SAS and SPSS Clementine which support data mining tasks [24], [25], [26], the characteristics of databases vary depending on their real applications. For the efficiency of these systems and applications of frequent pattern mining, it is essential to have algorithms that work efficiently for most database types.

Contribution: In this paper, we present DFEM, a new frequent pattern mining algorithm that combines the techniques of FP-growth and Eclat and depends on the data characteristics to select an appropriate technique for each subparts of the database to efficiently discover all long and short frequent patterns from sparse and dense databases. DFEM is based on and a major improvement of FEM, our previously developed algorithm [27]. Unlike FEM, it does not need a pre-determined, user specified threshold on when to switch between the two mining strategies. It automatically finds a runtime dynamic threshold to adjust its search behavior based on the characteristics of databases to improve the mining performance, especially when minimum support threshold is low.

In a brief overview, DFEM uses FP-tree to compact the database in memory and recursively mine the frequent patterns from this data structure like the FP-growth approach. During the mining process, if a conditional pattern base [11] is small enough to be better mined using vertical data structure, it is converted to TID bit vectors and a weight vector. The algorithm then switches from mining FP-trees using FP-growth to mining TID bit vectors using an approach improved from Eclat. The switching decision is based on a threshold *K* whose value is measured at runtime from data being processed.

We benchmarked DFEM and six other algorithms including Apriori [1], Eclat [10], FP-growth [11], FP-growth* [18] AIM2 [21], FEM [27]. The experimental results show that DFEM outperforms all of these algorithms for many real sparse and dense datasets. The performance merit of DFEM is obtained by efficiently distributing some subparts of the database to be mined by its FP-tree mining task and the other ones by its TID bit vector mining task.

The rest of this paper is organized as follows. Section 2 provides the background knowledge. The DFEM algorithm is presented in Section 3. Section 4 introduces the method of finding the threshold *K*. Experiments and discussion are presented in Section 5. The final section summarizes our study and points out some future research directions.

2 Background

In this section, we describe the frequent pattern mining problem and revisit FP-growth and Eclat, the two mining methods from which our proposed algorithm is derived, to analyze their features, strengths and weaknesses.

2.1 Frequent pattern mining problem

The frequent pattern mining problem can be stated as follows: Let $I = \{i_1, i_2, \ldots, i_n\}$ be the set of all distinct items in the transactional database *D*. The *support* of an *itemset* α (a set of items) is the percentage of transactions containing α in D. A *k-itemset* α, which consists of *k* items from *I*, is frequent if *α's support* is no fewer than *minsup*, where *minsup* is a user-specified minimum support threshold. Given a database *D* and a *minsup*, the problem statement is to find the complete set of frequent itemsets in *D*. We use the two terms *pattern* and *itemset* as well as *database* and *dataset* interchangeably in this paper. For example, given the dataset in Table 1 and *minsup*=30%, the frequent 1-itemsets include *a, b, c, d* and *e* while *f* and *g* are infrequent because their support is only 22%. Similarly, *ab, ac, ad, ae, bc, bd* are frequent 2-itemsets and *abc* is the only frequent 3-itemset.

TABLE 1
A DATASET WITH MINIMUM SUPPORT THRESHOLD = 30%

TID	Items	Sorted frequent items
1	b,d,a	a,b,d
2	c,b,d	b,c,d
3	c,d,a,e	a,c,d,e
4	d,a,e	a,d,e
5	c,b,a	a,b,c
6	c,b,a	a,b,c
7	f,g	
8	b,d,a	a,b,d
9	c,b,a,e,f,g	a,b,c,e

The exponential search space of itemsets makes finding frequent patterns a nontrivial task. Hence, many scientists have been working on developing efficient and scalable algorithms for frequent pattern mining. Most proposed methods are heuristics to trim down the search space in order to make the solution of this problem feasible. Apriori, Eclat and FP-growth are the most popular and widely used algorithms for mining frequent patterns.

2.2 The Eclat algorithm

Eclat is a popular frequent pattern mining algorithm developed by Zaki *et al.* [10]. It deploys a depth-first search strategy and requires only one database scan. To find frequent k-itemsets, the TID-lists of frequent (k-1)-itemsets, the vertical data layout, are intersected and the frequencies of their resulting lists are computed. The support of an itemset can be easily computed without multiple database scans as needed in the Apriori approach. Hence, the I/O cost is reduced considerably. Eclat has been shown to be one of the best algorithms for long patterns and/or dense databases [15], [21], [22]. Although this method takes advantage of the candidate generation approach, its depth-first search order may require more infrequent itemsets generated and tested than Apriori does. As a result, Eclat's efficiency reduces for sparse databases with short patterns where most itemsets are infrequent.

2.3 The FP-growth algorithm

FP-growth another well-known algorithm proposed by Han *et al.* [11] for frequent pattern mining. It adopts a data structure called FP-tree to compress and store data in memory. It then constructs conditional FP-trees and recursively mines these trees to find all frequent patterns without the requirement of generating a large number of frequent pattern candidates. FP-growth was shown to outperform previously developed methods including Eclat and Apriori [11], [17], [18], [19], [20]. For some dense databases or mining with low minimum support, the number of frequent patterns is very large. For each frequent k-itemset, FP-growth creates a conditional FP-tree used to find the frequent (k+1)-itemsets. Thus, the cost of generating a large number of FP-trees results in the degradation of performance. In such cases, FP-growth does not work as well as Eclat [15], [21], [22].

3 The DFEM algorithm

3.1 Mining frequent patterns based on data characteristics

Studying many real databases and their characteristics [28], [30], we found that most consist of a group of items occurring much more frequently than the others. The ratio of items in this group is smaller for sparse databases and larger for dense ones. Because of their high *supports*, these items appear in most transactions as well as most frequent patterns discovered from a database. If we remove the less frequent items from all transactions of a database, the remaining database will have the characteristics of a dense database and the removed part will have the characteristics of a sparse one. Moreover, the FP-tree of the FP-growth algorithm is constructed with the nodes of the most frequent items on the top because items are added into FP-tree in the frequency descending order. During the mining process of FP-growth, conditional FP-trees [11] are recursively constructed from parent trees. Our study shows that the size of these trees will reduce to a level where the conditional FP-trees mostly consist of items with high frequencies and have the characteristics of small dense databases [27]. According to the review of Eclat and FP-growth in Section 2, we propose a mining method that applies Eclat's strategy for the dense part and FP-growth's strategy for the sparse part will be more efficient than either Eclat or FP-growth alone.

In our previous study [27], we developed FEM, a frequent pattern mining algorithm based on above observation. FEM combines the mining techniques of Eclat and FP-growth and applies suitable mining strategy for each subparts of database. It has been shown to work better than many popular algorithms including Eclat and FP-growth on both sparse and dense databases by automatically distributing the workload to its two mining tasks where the dense data parts are handled by the mining task inherited and improved from Eclat and the sparse ones are mined using another mining task similar to FP-growth. FEM does this by applying a predefined threshold *K* reflecting the data specific characteristics to decide when to switch between the two mining tasks.

In this paper, we present, DFEM, a dynamic version of FEM that adopts a dynamic threshold *K* whose value is calculated at runtime from data being processed to better fit to data characteristics. This approach relieves the user from finding the appropriate value of *K*. DFEM performs better than FEM and other popular algorithms, especially when minimum support threshold (*minsup*) is low.

3.2 Overview of the DFEM algorithm

DFEM combines the techniques used in the FP-growth and Eclat algorithms. It uses FP-tree to store the compact database in memory and recursively mines the frequent patterns from this data structure similar to FP-growth. During its mining process, DFEM automatically switches between mining FP-trees using FP-growth to mining TID bit vectors using an approach improved from Eclat. The switching decision is based on a runtime dynamic threshold K measured from data. DFEM consists of the three main tasks:

FP-tree construction: Database is scanned for the first time to find the frequent items and create the header table. A second database scan is conducted to get frequent items of each transaction and insert these items into the FP-tree in their frequency descending order. Figure 1 demonstrates the FP-tree generated from the dataset in Table 1.

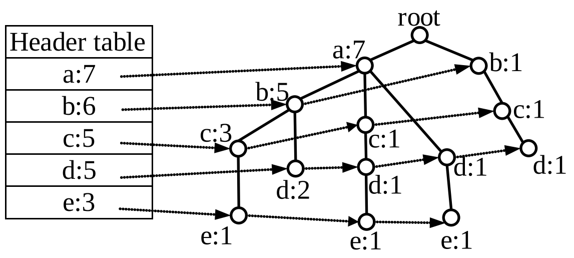

Fig. 1 FP-tree constructed from the dataset in Table 1

FP-Tree mining: This task uses the mining solution of FP-growth where frequent patterns are generated from the conditional FP-trees constructed recursively [11]. However, DFEM is different from FP-growth because only a subset of conditional pattern bases is used to create the conditional FP-trees. The conditional pattern bases whose size are smaller or equal to a threshold *K* will be transformed into TID bit vectors and a weight vector (see Section 3.4) and mining process switches to the *TID bit vector mining* task. K is computed at runtime using a measurement on data being processed as described in Section 4.

TID bit vector mining: This task obtains the TID bit vectors and continues searching for frequent patterns recursively by logical ANDing these bit vectors. The new patterns are constructed by concatenating the suffix pattern of previous steps with the newly generated frequent patterns. This mining task is inspired by Eclat's mining strategy [10]. However, TID bit vectors are used instead of the TID-lists due to their efficiency in computational time and memory consumption [8].

3.3 Algorithmic description

DFEM consists of three main sub algorithms: DFEM, MineFPTree and MineBitVector.

DFEM algorithm: This algorithm (Figure 2) builds the FP-tree, initializes the threshold K using *UpdateK* method (Figure 6) and invokes *MineFPTree* (Figure 3).

```
DFEM algorithm
Input: Transactional database D and minsup
Output: Complete set of frequent patterns
1:   Scan D once to find all frequent items
2:   Scan D a second time to construct the FP-tree T
3:   items = the number of frequent items in D
4:   trans = the number of transactions in D
5:   Call UpdateK(items,trans)
6:   Call MineFPTree(T,∅,minsup)
```

Fig. 2 DFEM algorithm

MineFPTree algorithm: This algorithm (Figure 3) executes the *FP-tree mining* task. If the condition at Line 13 is not satisfied, *MineBitVector* (Figure 4) will be invoked. Otherwise, *MineFPTree* will recursively mine work in recursive manner to generate frequent patterns from FP-tree data structure. Line 5 and 12 are used to update threshold K using the method described in Section 4.

```
MineFPTree algorithm
Input: Conditional FP-Tree T, suffix, minsup
Output: Set of frequent patterns
1:  If FP-tree T contains a single path P
2:  {   For each combination x of the items in P
3:        {  Output $\beta = x \cup suffix$  }
4:           n = the number of outputs $\beta$
5:           Call UpdateK(n,1)  }
6:  Else
7:  {   For each item $\alpha$ in the header table of FP-tree T
8:       { Output $\beta = \alpha \cup suffix$
9:         Construct $\alpha$'s conditional pattern base C
10:          n = the number of items in C
11:          size = the number of nodes in the linked list of $\alpha$
12:          Call UpdateK(n, size)
13:          If  size > K   Then
14:          {  Construct $\alpha$'s conditional FP-tree T'
15:             Call MineFPTree(T',$\beta$,minsup)   }
16:          Else
17:          {  Transform C into TID bit vectors V
18:                                and weight vector w
19:             Call MineBitVector(V,w,$\beta$,minsup) }
20:  }  }
```

Fig. 3 MineFPTree algorithm

MineBitVector algorithm: This algorithm (Figure 4) executes the *TID bit vector mining* task as described in Section 3.2.

```
MineBitVector algorithm
Input: Bit vectors V, weight vector w, suffix, minsup
Output: Set of frequent patterns
1:   Sort V in support-descending order of their items
2:   For each vector $v_i$ in V
3:   {    Output $\beta$ = item of $v_i \cup suffix$
4:        For each vector $v_j$ in V with $j < i$
5:        {   $u_j = v_i$ AND $v_j$
6:            $sup_j$ = support of $u_j$ computed using w
7:            If $sup_j \geq minsup$  Then add $u_j$ into U
8:        }
9:        If all $u_j$ in U are identical to $v_i$
10:       Then For each combination x of the items in U
11:                 Output $\beta' = x \cup \beta$
12:       Else If U is not empty
13:       Then Call MineBitVector(U,w,$\beta$,minsup)
14:  }
```

Fig. 4 MineBitVector algorithm

3.4 Transforming a conditional pattern base into TID bit bectors and a weight vector

This is an important step to enable the mining process using TID bit vectors. During the *FP-tree mining* stage, thousands or even millions of conditional pattern bases are processed. However, only those whose sizes are considerably small are transformed into TID bit vectors and weight vector. The size of a conditional pattern base is the number of sets in that base [11]. We use the number of nodes in the linked list of the item of the currently processed FP-tree to decide whether to switch from *FP-tree mining* to *TID bit vector mining* because the number of sets in a conditional pattern base is bounded by and equivalent to this number. The transformation is executed in following steps:

1. Given a conditional pattern base with sets of n items, create n bit vectors whose size are equal to the number of sets and initialized to zero.
2. For each item in the ith set, set the ith bit of its vector to one.
3. Repeat step 2 for every available sets in the conditional pattern base.

Furthermore, each set in a conditional pattern base has a frequency value indicating the number of its occurrence. We combine all frequency values into a weight vector which is used to compute the *support* of items or itemsets.

For example, Figure 5-a presents the conditional pattern base of item d of the FP-tree in Figure 1. If its size does not satisfy the condition for *TID bit vector mining,* this conditional pattern base is used to create the conditional FP-tree (Figure 5-b). Otherwise, it will be transformed into the TID bit vectors (Figure 5-c) and the weight vector (Figure 5-d) by applying above transformative steps.

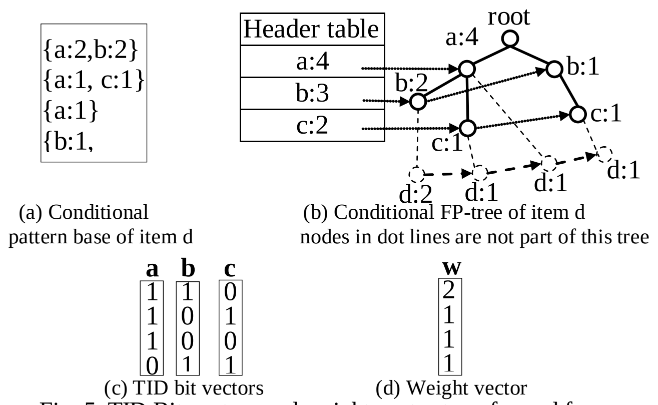

Fig. 5 TID Bit vectors and weight vector transformed from conditional pattern base of item d

4 Computing a dynamic K in DFEM

DFEM is developed to automatically determine a good threshold *K* that controls the switching between two mining tasks. This is the main difference between DFEM and FEM. In FEM, the value of *K* is predefined and users need to manually specify a fixed value of *K* for FEM to perform well on a certain database [27]. In contrast, DFEM estimates the near-optimal dynamic value of K at runtime.

4.1 Studying impact of varying K on FEM

We arrive with our approach in DFEM by studying and analyzing impact of varying *K* on FEM. When *K* increases starting from 0, the mining time of FEM and the number of frequent patterns found solely by the *FP-tree mining* task reduce because more conditional pattern bases satisfy the switching condition. FEM will perform best when *K* is equal to a value that presents the best cut off between sparse and dense portions of the database handled by the *FP-tree mining* and the *TID bit vector mining* tasks respectively. For K_i larger than this best *K*, a portion of sparse data is shifted to *TID bit vector mining* which reduces the benefits of *FP-tree mining* and results in unchanged or increasing the running time of FEM. Table 2 presents the impact of varying *K* on FEM for the Kosarak dataset [28]

TABLE 2
MEASUREMENTS OF FEM FOR KOSARAK (MINSUP=0.07%)

Thres. K_i	Mining time (second)	# frequent patterns by *MineFPTree* (P_i)	Ratio R_i
0	3341	2776266097	N/A
32	2939	1316339679	2.1
64	2146	206479285	6.4
96	1664	26795140	7.7
128	1206	2413815	11.1
160	1005	407051	5.9
192	934	86575	4.7
224	**871**	**63876**	**1.4**
256	870	58304	1.1

Let $\{K_0, K_1, \ldots K_N\}$ be the set of all values of *K* where $K_i = K_{i-1}+32$; P_i is the number of frequent patterns generated by the *FP-tree mining* task when K_i is applied; and R_i is the ratio indicating the difference between P_i and P_{i-1}. R_i is computed as $R_i = P_{i-1}/P_i$,$(i = 1 \ldots N)$. According to the above observation and our extensive tests on many datasets from a well-known repository [28], the best K_i is the one satisfying the condition $R_i < 2 \ni (\nexists\, R_j \geq 2,\ \forall j > i)$. In other words, FEM will perform best at the smallest K_i where increasing *K* does not result in a sharp drop in the number of frequent patterns found by the *FP-tree mining* task. In Table 2, this condition is satisfied for K_i=224.

4.2 Automatically computing K in DFEM

It is challenging to apply the above approach to automatically find the best *K* because this value of *K* can only be specified when the mining process completes and all P_i and R_i are computed. Hence, a practical method to predict a near-optimal value of K_i based on all P_i's estimated dynamically at runtime is developed in DFEM as described in the *UpdateK* algorithm in Figure 6.

```
UpdateK algorithm
Input: NewPatterns and Size
Output:  updated value of threshold K
1:  If UpdateK is called for the first time then
2:  {     Create an array P with N elements
3:        Initialize all $P_i$ to zero      }
4:  For $i = 0$ to $N – 1$
5:      If Size > i*Step then $P_i = P_i$ + NewPatterns
6:      Else Exit Loop
7:   $K = 0$
8:   For $i = N$-1 to 1
9:        If $R_i \geq 2$  then  $K = (i+1)$*Step and Exit Loop
```

Fig. 6 The UpdateK algorithm

In this algorithm, an array *P* with *N* elements is created and updated for every conditional pattern base processed in *MineTree* (Section 3.3) where *N* is the number of K_i being considered. Then, the best *K* is computed by finding the smallest K_i satisfying the condition in Section 4.1. We choose default values of *N*=9 and *Step*=32 where *Step* is the distance between K_i and K_{i-1} so that the best *K* is in the range 0-256 which is found the best range in our studies and allows only small conditional pattern base is transformed to in TID bit vectors (Section 3.4). For *Step*=32, the maximum size of TID bit vectors limited by *K* is a multiple of 32 (4 bytes) has better memory utilization. This distance also guarantees that applying the condition in Section 4.1 helps to specify a near-optimal of K. In Figure 6, *NewPatterns* indicates the number of new frequent patterns and is equal to the number of items in conditional pattern base C; *Size* is the size of *C*.

5 Experiments and performance study

We benchmark DFEM, FEM and five popular frequent pattern mining algorithms. DFEM is then studied in-depth to show how well it adapts to data characteristics.

5.1 Experimental setup

Software: Seven algorithms in our experiments includes DFEM, Apriori [1], Elcat [10], FP-growth [11], FP-growth* [18], AIM2 [21] and FEM [27]. DFEM and FEM are implemented using the optimizing techniques introduced in [27]. The state-of-the-art implementations of other five algorithms are obtained from [28] and [29].

Hardware: The seven algorithms tested on a machine with dual AMD Opteron 2427 processors, 2.2GHz, 24GB memory and 160 GB hard drive running CentOS 5.3, a Linux-based distribution. We used g++ for compilation.

Datasets: Five real datasets with various characteristics (Table 3) used in our benchmark were obtained from the Frequent Itemset Mining Implementations Repository [28], a well-known repository for frequent pattern mining.

TABLE 3
DATASETS AND THEIR PROPERTIES

Datasets	Type	# Items	Average Length	# Transactions	Size (MB)
Chess	Dense	76	37	3196	0.31
Mushroom	Dense	119	23	8124	0.56
Accidents	Moderate	468	33.8	340183	33.8
Retail	Sparse	16470	10.3	88126	3.79
Kosarak	Sparse	41271	8.1	990002	32.3

5.2 Performance comparison

The experimental results (Figure 7) show that DFEM runs stably and outperforms the popular algorithms for all tested sparse and dense dataset, while the other algorithms behave differently for different datasets. DFEM also performs better than FEM, our previous developed algorithm because DFEM adapts better to data characteristics. Apriori runs slowest on three datasets Chess, Accidents, Kosarak but it does better than FP-growth* for Mushroom as well as Eclat, FP-growth* and AIM2 for Retails dataset. Eclat works better than the others except AIM2, FEM and DFEM on the dense datasets. However, for the sparse datasets such as Retail and Kosarak, Eclat runs slower than most of the others. Compared to Eclat, two algorithms FP-growth, FP-growth* run faster for the dense datasets but slower for the sparse ones. AIM2, a variant of Eclat, performs well for some sparse and dense datasets but worse for the other ones. This experiment demonstrates the time efficiency of DFEM for both sparse and dense datasets.

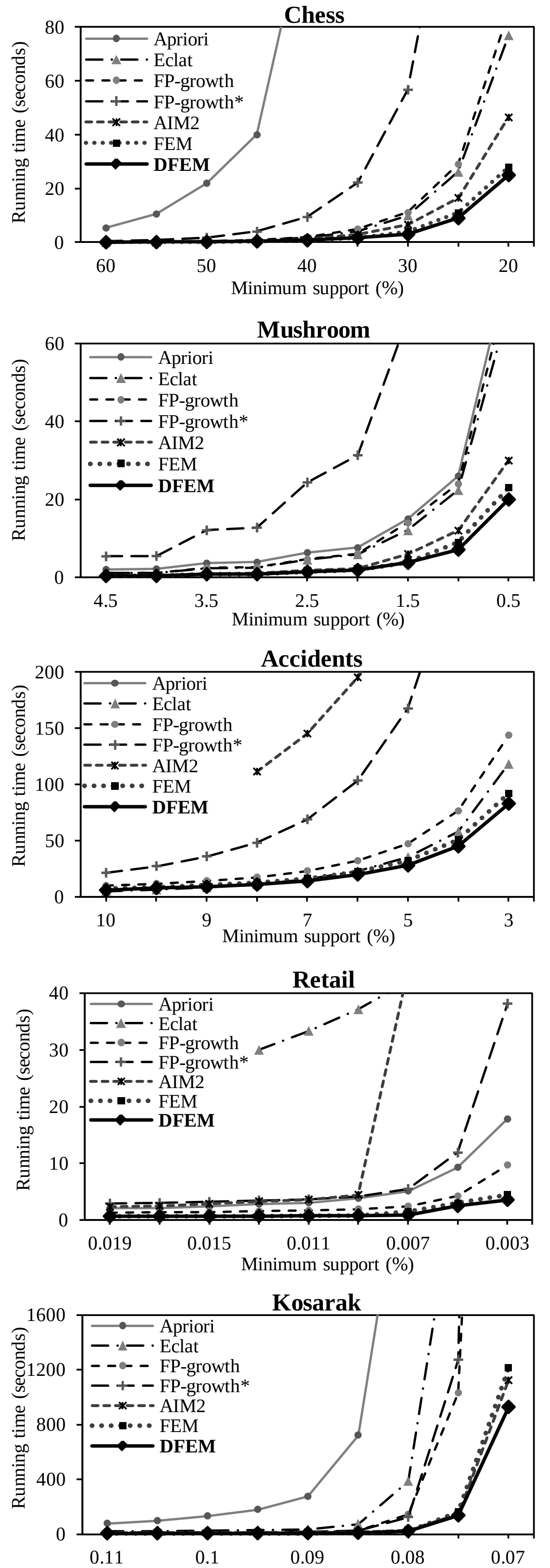

Fig. 7 Performance comparison of DFEM and other algorithms

5.3 Analyzing performance merits of DFEM

To provide insight into the performance merits of DFEM, the mining time of the *FP-Tree mining* task and the *TID bit vector mining* task were measured separately to observe the time distribution of each mining task in total mining time. The experimental results on five selected datasets are reported in Table 4. We found that DFEM distributes mining workload to its two mining tasks dynamically depending on data specific. From these results, *TID bit vector mining* is responsible for over 90% of the mining time for the dense datasets like Chess and Mushroom because the shape of the FP-tree of dense datasets is usually compact and most conditional pattern bases satisfy the condition to switch from *FP-Tree mining* to *TID bit vector mining*. In contrast, for the sparse datasets like Retail and Kosarak, *FP-Tree mining* is responsible for 90% - 99% of the mining time because many large FP-trees are generated and most of them do not satisfy the switching condition. For the Accidents dataset whose density is moderate, the mining time distribution of DFEM is balanced between the two mining tasks. It must be noted that the mining time distribution does not indicate the amount of work. In fact, the *TID bit vector mining* task using faster bitwise operations and more cache-friendly data layout will process larger amounts of data than *FP-Tree mining* does in a same unit of time.

TABLE 4
MINING TIME DISTRIBUTION BETWEEN TWO MINING TASKS OF DFEM

Datasets	Type	Minimum support (%)	FP-Tree mining time (%)	TID vit vector mining time (%)
Chess	Dense	40	4	96
Mushroom	Dense	2.5	3.5	96.5
Accidents	Moderate	5	55	45
Retail	Sparse	0.011	92	8
Kosarak	Sparse	0.09	95.5	4.5

In addition, the mining time distribution changes when the minimum support varies (Figure 8). As the minimum support is set to lower levels, more small conditional FP-trees are generated and hold the condition to switch from FP-tree mining to *TID bit vector mining* which makes the mining time percentage of *TID bit vector mining* increases as the minimum support is reduced.

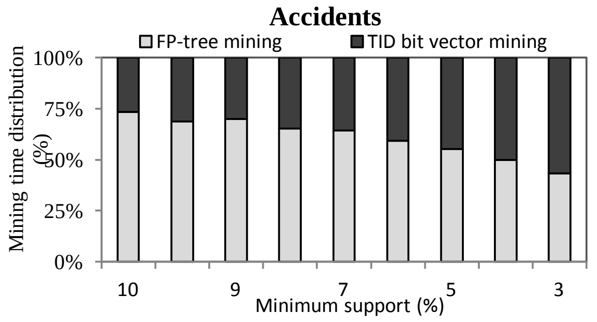

Fig. 8 Mining time distribution of DFEM for Accidents dataset

In conclusion, DFEM have the ability to switch between strategies at runtime by distributing the mining workload to the appropriate strategy that best fit the data characteristics.

6 Conclusion and future work

In this paper, we present DFEM, a new algorithm for frequent pattern mining that can adapt its mining behavior to data characteristics to efficiently find all frequent patterns from both sparse and dense databases. Compared to FEM, our previously developed algorithm for this mining task, DFEM can automatically specify a good runtime threshold K used to switch between the two mining tasks. The experimental results show that DFEM significantly improve the performance of mining frequent patterns and outperforms other well-known algorithms for different database types. In future work, we will study parallel approaches for implementing FEM and DFEM on parallel and distributed systems because memory limitation and computational time are the major obstacles to deploying any sequential frequent pattern mining algorithm on very large scale databases.

7 References

[1] R. Agrawal, R. Srikant, "Fast Algorithms for Mining Association Rules," *Proc. of the 20th Int. Conf. on Very Large Databases*, pp. 487-499, 1994.

[2] S. Brin, R. Motwani, C. Silverstein, "Beyond Market Basket: Generalizing Association Rules to Correlations," *Proc. ACM SIGMOD Management of Data*, vol. 26, issue 2, pp. 265-276, Jun. 1997.

[3] C. Silverstein, S. Brin, R. Motwani, J. Ullman, "Scalable Techniques for Mining Causal Structures," *J. Data Mining and Knowledge Discovery*, vol. 4, issue 2-3, pp. 163–192, July 2000.

[4] R. Agrawal, R. Srikant, "Mining Sequential Patterns," *Proc. Data Engineering*, pp. 3–14, 1995.

[5] H. Mannila, H. Toivonen, and A. I. Verkamo, "Discovery of Frequent Episodes in Event Sequences," *J. Data Mining and Knowledge Discovery*, vol. 1, issue 3, pp. 259-289, Sep. 1997.

[6] J. Han, G. Dong, Y. Yin, "Efficient Mining of Partial Periodic Patterns in Time Series Database," *Proc. IEEE Data Engineering*, pp. 106-115, Mar. 1999, doi: 10.1109/ICDE.1999.754913.

[7] J. Han, H. Cheng, D. Xin, X. Yan, "Frequent Pattern Mining: Current Status and Future Directions," *J. Data Mining and Knowledge Discovery*, vol. 15, issue 1, pp. 55-86, Aug. 2007.

[8] D. Burdick, M. Calimlim, J. Flannick, J. Gehrke, T. Yiu, "MAFIA: A Maximal Frequent Itemset Algorithm," *IEEE Trans. on Knowledge and Data Engineering*, vol. 17, no. 11, pp. 1490–1504, Nov. 2005, doi: 10.1109/TKDE.2005.183

[9] H. Li, Y. Wang, D. Zhang, M. Zhang, E. Chang, "PFP: Parallel FP-Growth for Query Recommendation," *Proc. 2008 ACM Recommender systems*, pp. 107-114, 2008.

[10] M. Zaki, S. Parthasarathy, M. Ogihara, W. Li, "New algorithms for fast discovery of association rules," *Proc. Knowledge Discovery and Data Mining*, pp. 283-286, 1997.

[11] J. Han, J. Pei, Y. Yin, "Mining Frequent Patterns without Candidate Generation," *Proc. Management of Data*, vol. 29, issue 2, pp. 1-12, Jun. 2000.

[12] JS. Park, MS. Chen, P. Yu, "An Effective Hash-based Algorithm for Mining Association Rules," *Proc. ACM SIGMOD Management of Data*, vol. 24, issue 2, pp. 175–186, May 1995.

[13] H. Toivonen, "Sampling Large Databases for Association Rules," *Proc. Very Large Databases*, pp. 134–145, 1996.

[14] S. Brin, R. Motwani, JD. Ullman, S. Tsur, "Dynamic Itemset Counting and Implication Rules for Market Basket Analysis," *Proc. ACM SIGMOD Management of Data*, vol. 26, issue 2, pp. 255–264, 1997.

[15] A. Fiat, S. Shporer, "AIM: Another Itemset Miner," *Proc. Frequent Itemset Mining Implementations*, 2003.

[16] M. J. Zaki, K. Gouda, "Fast Vertical Mining Using Diffsets," *Proc. ACM SIGKDD Knowledge Discovery and Data Mining*, pp. 326-335, 2003.

[17] C. Borgelt, "An Implementation of the FP-growth Algorithm," *Proc. OSDM Frequent Pattern Mining Implementations*, Aug. 2005.

[18] G. Grahne, J. Zhu, "Efficiently Using Prefix-trees in Mining Frequent Itemsets," *Proc. Frequent Pattern Mining Implementations*, pp 123–132, 2003.

[19] J. Pei, J. Han, H. Lu, S. Nishio, S. Tang, D. Yang, "Hmine : Hyper-structure Mining of Frequent Patterns in Large Databases," *Proc. IEEE Data Mining*, pp. 441–448, Nov. 2001, doi: 10.1109/ICDM.2001.989550

[20] B. Racz. "nonordfp: An FP-growth Variation Without Rebuilding the FP-tree," *Proc. IEEE ICDM Workshop on Frequent Itemset Mining Implementations*, Nov. 2004.

[21] S. Shporer, "AIM2: Improved Implementation of AIM," *Proc. IEEE Frequent Itemset Mining Implementations*, Nov. 2004.

[22] L. Schmidt-Thieme., "Algorithmic Features of Eclat," *Proc. IEEE Frequent Itemset Mining Implementations*, Nov. 2004.

[23] L. Liu, E. Li , Y. Zhang, Z. Tang, "Optimization of Frequent Itemset Mining on Multiple-core Processor," *Proc. of the 33rd Int. Conf. on Very Large Databases*, pp. 1275-1285, 2007.

[24] W. Li, A. Mozes, "Computing Frequent Itemsets Inside Oracle 10g," *Proc. of the 30th Int. conf. on Very Large Databases*, pp. 1253–1256, 2004.

[25] C. Utley, "Introduction to SQL Server 2005 Data Mining," *Microsoft SQL Server 9.0 technical articles*, available at: http://technet.microsoft.com/en-us/library/ms345131.aspx, Jun. 2005.

[26] T. Yoshizawa, I. Pramudiono, M. Kitsuregawa, "SQL Based Association Rule Mining Using Commercial RDBMS (IBM db2 UBD EEE)," *Proc. Data Warehousing and Knowledge Discovery*, pp. 301–306, 2000.

[27] L. Vu, G. Alaghband, "A Fast Algorithm Combining FP-Tree and TID-List for Frequent Pattern Mining," *Proc. IEEE of the 2011 Inf. Conf. on Information and Knowledge Engineering*, pp. 472-477, Jul. 2011, Las Vegas, NV, USA.

[28] "Frequent Itemset Mining Implementations Repository," *Workshop on Frequent Itemset Mining Implementation*, 2003-2004, available at http://fimi.ua.ac.be

[29] Christian Borgelt, "Frequent Pattern Mining Implementations," available: http://www.borgelt.net.

[30] B. Goethals, M. J. Zaki, "Advances in Frequent Itemset Mining Implementations: Report on FIMI'03," *ACM SIGKDD Explorations Newsletter - Special issue on learning from imbalanced datasets*, Vol. 6 Issue 1, pp. 109-117, June 2004, New York, NY, USA.