A Fast Algorithm Combining FP-Tree and TID-List for Frequent Pattern Mining

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 - *Finding frequent patterns plays an essential role in mining associations, correlations, and many other interesting relationships among variables in transactional databases. The performance of a frequent pattern mining algorithm depends on many factors. One important factor is the characteristics of databases being analyzed. In this paper we propose FEM (FP-growth & Eclat Mining), a new algorithm that utilizes both FP-tree (frequent-pattern tree) and TID-list (transaction ID list) data structures to discover frequent patterns. FEM can adapt its behavior to the dataset properties to efficiently mine short and long patterns from both sparse and dense datasets. We also suggest a combination of several optimization techniques for effectively implementing FEM to speed up the mining process. The experimental results show that a significant improvement in performance is achieved.*

Keywords: knowledge mining, data mining, frequent pattern mining, association rule mining, frequent itemset.

1 Introduction

Frequent pattern mining is one of the fundamental problems in data mining. It plays an important role in finding many types of relationships among data such as associations [1], correlations [4], causality [5], sequential patterns [6], episodes [7] and partial periodicity [8]. Moreover, it helps in data indexing, classification, clustering, and other data mining tasks as well [9].

The frequent pattern mining problem aims to search for groups of itemsets, subsequences, or substructures that co-occur frequently in a dataset. 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 itemsets generated 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 [22]. Among numerous proposed methods, Apriori, FP-growth and Eclat are most popular and widely used.

The Apriori algorithm [1] utilizes the property, that a k-itemset is frequent only if all of its sub-itemsets are frequent, to reduce the search space of frequent itemsets. It is built on a recurrence relation where to find frequent k-itemsets, Apriori uses frequent (k-1)-itemsets found in the previous step. Many variants of Apriori have been proposed to improve the mining efficiency, e.g. direct hashing and pruning (DHP) [10], sampling technique [23], dynamic itemset counting (DIC) [24]. FP-growth [3] works in a divide-and-conquer fashion. It compresses the database into a FP-tree, constructs its conditional FP-trees and recursively mines on these trees to find the frequent itemsets. Some extensions of FP-growth include an array technique to reduce the FP-tree traversal time [13], the usage of FP-array data structure [16], H-mine [17] and nonordfp [18]. While Apriori and FP-growth explore the horizontal data format, Eclat [2] uses the vertical TID-lists of frequent (k-1)-itemsets to find the frequent k-itemsets by intersecting these TID-lists and computing their resulting supports. Mafia [11], AIM [14], mining using diffsets [21] are similar approaches in using the vertical data format. However, all these methods have advantages and disadvantages that make one suitable for specific databases and computing platforms. Hence, hybrid method is another approach to exploit the benefits of many mining methods.

In this paper we propose FEM (FP-growth & Eclat Mining), a new algorithm for frequent pattern mining that combines the techniques used in the FP-growth and Eclat algorithms. Our approach uses FP-tree to store the compact database in memory and recursively mine the frequent patterns from this data structure similar to the FP-growth approach. In addition, FEM will automatically switch from mining FP-trees using FP-growth to mining TID-lists using Eclat depending on the structure of the currently processed data. In order to enable the mining task using Eclat method, during the pattern growth process, the conditional pattern base [3] of a frequent item will be transformed into TID-lists [2] if its size is small enough for better mining on the vertical data structure. FEM can adapt its behavior to the database characteristics for efficiently mining both short and long patterns from sparse and dense datasets. We also suggest a combination of several optimization techniques for implementing FEM to speed up the frequent pattern mining process. Our experimental results show that a significant improvement of performance is achieved using our proposed approach.

This paper is organized as follows. Section 2 provides the essential background knowledge. Our FEM algorithm is described in section 3. Section 4 presents optimization techniques for FEM. Experiments and performance study are presented in section 5. The final section summarizes our study and points out some future research directions.

2 Background

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 transactional database D. The *support* of an *itemset* α (a set of items) is the number 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 δ, where δ is a user-specified minimum support threshold. Given a database D and minimum support threshold δ, the problem statement is to find the complete set of frequent itemsets in D.

For example, given the dataset in table I and minimum support threshold δ=3, the frequent 1-itemsets include *a, b, c, d and e* while *f* and *g* are infrequent because *f* and *g* occur only 2 times. Similarly, *ab, ac, ad, ae, bc, bd* are frequent 2-itemsets and *abc* is the only frequent 3-itemset found.

TABLE I
A DATASET WITH MINIMUM SUPPORT THRESHOLD = 3

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

2.2 The FP-growth algorithm and FP-tree structure

FP-growth is a well-known algorithm proposed by Han *et al.* [3] for frequent pattern mining. It utilizes the FP-tree (frequent pattern tree) to efficiently discover the frequent patterns. FP-tree is an extended prefix-tree structure that uses the horizontal database format and stores compressed information about patterns. It consists of one root node, a set of item prefix subtrees as the children of the root, and a frequent-item header table. Each node of the FP-tree includes an *item name,* a *link* to the next node in the linked list of its appropriate frequent item and a *count* indicating the number of transactions that contains all items in the path from the root node to the current node. The header table stores the frequent items in frequency descending order. Each entry of this table includes *item name*, item *support* and a *link* to the head node in the linked list of its frequent item. The FP-tree is organized so that if two transactions share a common prefix, the shared part can be merged as long as the count properly reflects the occurrence of each itemset in the transactions.

FP-growth requires only two scans on the dataset. In the first scan, the frequency items are found to generate the header table. The dataset is re-scanned to achieve and sort the frequent items in each transaction as illustrated in the third column in Table 1. These items are inserted into FP-tree in frequency descending order. If the appropriate node of an item exists, its count is increased by one. Otherwise, a new node is inserted to the FP-tree. Figure 1 illustrates the FP-tree constructed from the dataset in Table 1.

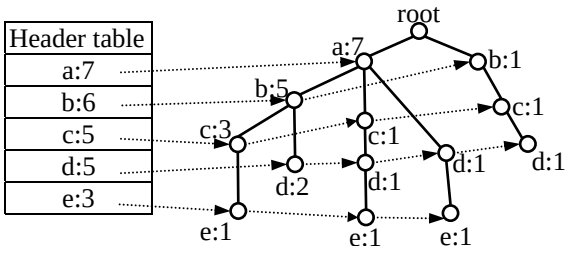


Fig. 1 FP-tree constructed from the dataset in Table I

The next step is to mine the FP-tree by constructing the conditional pattern base and then constructing the conditional FP-tree of each frequent item as described by an example in Section 3.2, and performing mining recursively on such a tree [3]. The frequent items of a resulting conditional FP-tree are combined with the suffix-pattern to generate the new frequent patterns.

2.3 The Eclat algorithm and TID-list structure

Eclat is another efficient algorithm for frequent pattern mining developed by Zaki *et al.* [2]. This algorithm utilizes the TID-list data structure (transaction ID list), a vertical format of database, for the mining task. A TID-list of an item or itemset is a list that stores the IDs of transactions containing that item or itemset. Eclat, similar to FP-growth, applies the depth-first approach to search for frequent patterns and needs only two database scans. It first scans the database to find all frequent items. In the second database scan, it generates the TID-lists of the frequent items. This algorithm organizes frequent k-itemsets into disjoint equivalence classes by common (k-1)-prefixes. The candidate (k+1)-itemsets can be generated by joining pairs of frequent k-itemsets from the same classes. The main advantage of using TID-list is that the support of a candidate itemset can be computed simply by intersecting the TID-lists of the two component subsets. A simple check on the resulting TID-list tells whether the new itemset is frequent or not. Figure 2 demonstrates the TID-lists and the Eclat mining process for the dataset in the Table I.

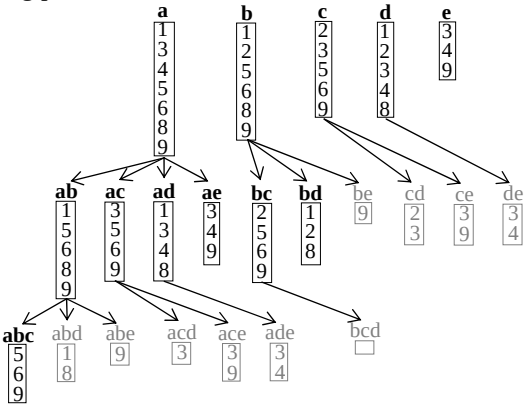


Fig. 2 Mining TID-lists using Eclat

3 The FEM algorithm

3.1 Overview of the FEM algorithm

In the FP-growth algorithm, the frequent patterns are discovered from the conditional FP-trees which are recursively constructed from the original FP-tree. The shape of FP-trees is usually wide for sparse datasets and more compact for the dense ones. In either case, the size of newly generated trees is much smaller than their original one. We found that this size will reduce to a level where mining with an alternative data structure performs better. Additionally, the conditional pattern base of a given item can be easily converted into TID-lists which are more cache-friendly than the trees with linked lists and pointers are. We therefore propose FEM, an algorithm combining mining techniques of FP-growth and Eclat, to discover frequent patterns. FEM flexibly uses both FP-tree and TID-list for its mining task. In the pattern growth process, it switches between FP-growth strategy and Eclat strategy depending on whether FP-tree or TID-list provides better performance. FEM consists of the following three main tasks:

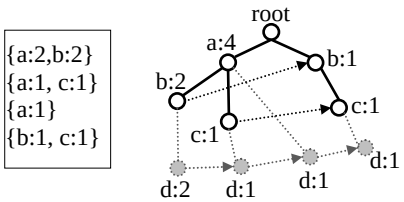
FP-tree construction: Database is scanned for the first time to find the frequent items and create the header table. A second database scan is conducted to get and sort frequent items of each transaction in the support-descending order and then build the FP-tree.

FP-tree mining: This task uses the mining solution of FP-growth. It constructs the conditional FP-trees and recursively mines these trees to find the frequent itemsets. However, before a conditional FP-tree is constructed, FEM will check the size of its conditional pattern base. If it is considerably small, FEM will transform it into TID-lists and switch to mining task using Eclat approach, described next. We represent the TID-lists in bit vector form for its efficiency in computation and memory consumption.

TID-list mining: In this task, we obtain the TID-lists using our bit-vector representation and continue searching for the 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 strategy in using vertical format of database.

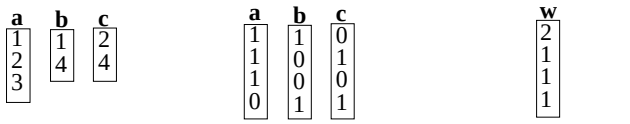
3.2 Transforming a conditional pattern base into TID-lists

A conditional pattern base is a "sub-database" which consists of the sets of frequent items co-occurring with the suffix pattern [3]. Each frequent item of a FP-tree has an equivalent conditional pattern base derived from that FP-tree. For example, the conditional pattern base of item d in the FP-tree (Fig. 1) has four sets {a:2, b:2}, {a:1, c:1}, {a:1}, {b:1, c:1} (Fig. 3-a). This conditional pattern base can be used to construct the conditional FP-tree (Fig. 3-b). In the FEM algorithm, we consider these sets as transactions (Fig. 3-c) and transform them into TID-lists for mining using Eclat approach. The transformation is executed by assigning each set with an ID and grouping IDs into lists. In our example, three TID-lists (Fig. 3-d) can be generated including {1, 2, 3} of item a, {1, 4} of item b and {2, 3} of item c. To save memory and benefit bitwise operation efficiency, we represent TID-lists in bit-vector form. The advantage of this approach was shown in [11]. Figure 3-e illustrates the TID bit vectors transformed from the conditional pattern base of item d. In other side, each set in the conditional pattern base has a frequency value indicating the number of its occurrence. We combine all the frequency values into a weight vector which will be used to compute the support of the TID-list. The weight vector in the given example is {2, 1, 1, 1} (Fig. 3-f).



TID	Items	Weight
1	a,b	2
2	a,c	1
3	a	1
4	b,c	1

(a) Conditional pattern base of item d (b) Conditional FP-tree of item d (c) Dataset equivalent to (a)



(d) TID-lists (e) TID bit vectors (f) Weight vector w

Fig. 3 TID-List and Bit vectors transformed from conditional pattern base of item d

During the *FP-tree mining* stage, thousands or even millions of conditional pattern bases are processed. However, only those whose size is considerably small are transformed into TID bit vectors. We use the number of nodes in the linked list of item α in the original FP-tree to decide whether to switch from *FP-tree mining* to *TID-list mining*. This criteria is used because a small number of nodes in the linked list of α usually indicates a small size of α's conditional pattern base. If this number of nodes is less than or equal to a threshold K, FEM will use Eclat strategy. The value of K depends on the properties of the dataset. In this study, we chose a value of 128 for K. Using this value, the maximum size of a TID bit vector is 16 bytes which is usually smaller than or equal to the size of just one node of FP-tree. The memory size of all TID bit vectors is therefore not greater than the number of items in the frequent pattern base multiplied by 16. This data structure requires much less memory space than an equivalent conditional FP-tree does. Furthermore, the bitwise operations on TID bit vectors will perform faster than creating and manipulating FP-trees. In the given example, the number of node in the linked list of item d is 4 (Fig. 3-b) which is less than 128, so its conditional pattern base will be transformed into TID-lists as described above. Finding an optimal value of K for each specific database will be a subject of our future work.

3.3 The FEM algorithm

The FEM algorithm consists of three components:

a. *FEM-mining:* the mining process first constructs the FP-tree from the original database and it then calls *FP-tree-mining* as represented below.

FEM-mining algorithm
Input: Transactional database D and min. support δ *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: Call FP-tree-mining($T, \varnothing, \delta$)

b. *FP-tree-mining*: this component is equivalent to the *FP-tree mining* task described in Section 3.1. Lines 7-12 in the following algorithm show the switching between the two mining tasks. The value of K in our current experiments is 128.

FP-tree-mining algorithm
Input: Conditional FP-Tree T, *suffix*, min. support δ *Output*: Set of frequent patterns 1: If FP-tree T contains a single path P 2: Then For each combination x of the nodes in P 3: Output $\beta = x \cup suffix$ 4: Else For each item α in the header table of FP-tree T 5: { Output $\beta = \alpha \cup suffix$ 6: Construct α's conditional pattern base C 7: If item α has more than K nodes in its linked list 8: Then { Construct α's conditional FP-tree T' 9: Call FP-tree-mining (T', β, δ) } 10: Else { Transform C into TID bit vectors V 11: and weight vector w 12: Call TID-list-mining (V, w, β, δ) } }

c. *TID-list-mining*: this component is equivalent to the *TID-list mining* task described in Section 3.1which the TID-lists are represented in the bit-vector form. This algorithm is called by FP-tree-mining and recursively mines until no new frequent pattern is found.

TID-list-mining algorithm
Input: Bit vectors V, weight vector w, suffix, min. support δ *Output*: Set of frequent patterns 1: Sort V in descending other of its item support 2: For each vector v_i in V 3: { Output β = item of $v_i \cup suffix$ 4: For each vector v_k in V, $k < i$ 5: { $u_k = v_i$ AND v_k 6: sup_k = support of u_k based on w 7: If $sup_k \geq \delta$ Then add u_k into U } 8: If all u_k in U are identical to v_i 9: Then For each combination x of the items in U 10: Output $\beta' = x \cup \beta$ 11: Else If U is not empty 12: Then Call TID-list-mining (U, w, β, δ) }

4 Optimization techniques for implementing FEM

The performance of a frequent pattern mining algorithm depends on many factors: data structure, database properties, CPU speed, I/O speed, memory size, minimum support threshold, etc. Table III shows the two implementations of the FP-growth algorithm (i.e. FP-growth_B and FP-growth_GZ. As the Table shows these two implementations result in varying performance on different datasets due to using different optimization approaches. We have incorporated a set of optimization techniques for implementing FEM that has effectively improved the runtime performance of our algorithm for variety of datasets. Following are the details:

FP-tree construction: In the second database scan, we pre-load the filtered transactions into a lexicographically sorted list as suggested in [12]; one copy of similar transactions is kept with its count. The transaction list size can be changed to fit the available memory. We organize this list in a binary tree and maintain its order while the list grows in size. This technique reduces the traversal and construction time of FP-tree. It also keeps the nodes most visited together to be allocated closely in the memory. Thus, it speeds up the mining stage as well.

Mining task using FP-growth: We exploit the technique proposed by [13]. An array-based implementation of prefix-tree-structure is used to improve the efficiency of the *FP-tree-mining* by reducing the need of traversal on FP-trees when constructing the conditional FP-tree.

Memory management: A chunk of memory is allocated for each FP-tree when FEM creates a new one and is discarded after all frequent itemsets from this FP-tree are generated. The chunk size is variable. This technique reduces overhead of allocating and freeing nodes [13].

Output processing: We preprocess the most frequent output values and store them in indexed tables as proposed in [15]. In addition, the similar part of two frequent itemsets outputted consecutively is processed only once. Hence, this technique improves considerably computation time on output reporting, especially when output size is huge.

I/O optimization: Data are read into a buffer before being processed into transactions. Similarly, the outputs are buffered and only written when the buffer is full. This technique reduces much I/O overhead.

5 Experiments and Performance study

5.1 Experiments

The experiments were performed on the Altus 2701 machine with dual AMD Opteron 2427, 2.2GHz, 32GB memory, running Linux OS. We used g++ for compilation. Five datasets used in our tests Connect, Mushroom, Accident, Retail and Webdocs are publicly available at the Frequent Itemset Mining Implementations Repository [19] and are reported in the Table II.

TABLE II
DATASETS AND THEIR PROPERTIES

Datasets	Type	Transactions	Items	Average length	Size
Connect	Dense	67557	129	43	8.82 MB
Mushroom	Dense	8124	119	23	557 KB
Accidents	Moderate	340183	468	33.8	33.8 MB
Retail	Sparse	88126	16470	10.3	3.79 MB
Webdocs	Sparse	1623346	52676657	177.23	1.37 GB

FEM was implemented using the optimization techniques in Section 4. For comparison, we benchmarked FEM and three state-of-the-art frequent pattern mining implementations: FP-growth_B by Borgelt [12], FP-growth_GZ by Grahne & Zhu [13] and AIM2 by Fiat & Shporer [14], [15] (i.e. an Eclat based approach) which are available at [19], [20]. The runtime with the considerably low supports for all datasets is reported in the Table III. The detailed performance comparison on Connect, Accident and Webdocs datasets with various supports are presented in Fig. 4 and Fig. 5 and Fig. 6. Table IV presents the runtime distribution between *FP-tree-mining* and *TID-list-mining* of FEM in both absolute runtime and percentage runtime.

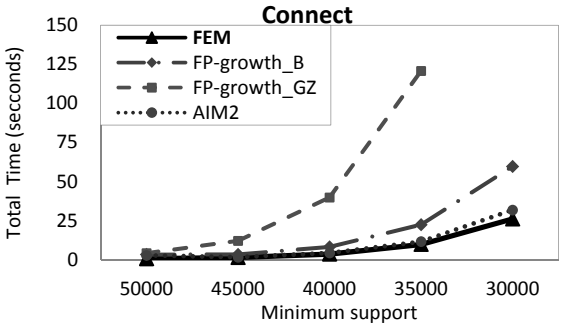


Fig. 4 Runtime on the connect dataset

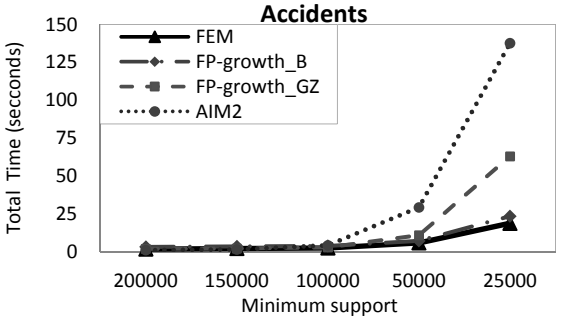


Fig. 5 Runtime on the accident dataset

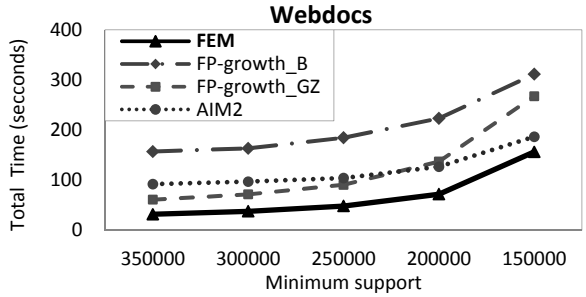


Fig. 6 Runtime on the webdocs dataset

5.2 Performance study

5.2.1 Performance comparison

All the four implementations tested on the same machine behaved differently on the dense and sparse datasets (Table III). FP-growth_B and AIM2 performed better than FP-growth_GZ on the dense datasets Connect and Mushroom. For the sparse datasets, AIM2 ran faster than both FP-growth_B and FP-growth_GZ on the Webdocs but slower on the Retail. For the Accident dataset, AIM2 did not perform as well as either the FP-growth_B or FP-growth_GZ. FEM performed quite well in every case. The performance of FEM reported in Table III and Fig. 4, 5, 6 shows that FEM outperforms the other algorithms on all test databases. Hence, FEM works better than both FP-growth and Eclat on variety of datasets.

TABLE III
RUNTIME (SECONDS) FOR FIVE DATASETS WITH SELECTED MIN SUPPORTS

Datasets	Minimum support	**FEM**	FP-Growth_B	FP-Growth_GZ	AIM2
Connect	30000	**25.55**	59.05	342.03	31.78
Mushroom	50	**21.64**	56.31	306.61	28.47
Accidents	25000	**18.61**	25.36	63.41	137.48
Retail	5	**1.62**	4.88	9.84	51.07
Webdocs	150000	**156.65**	341.56	267.09	186.49

5.2.2 The runtime distribution between two mining tasks

The runtime distribution between *FP-tree-mining* and *TID-list-mining* in Table IV and Fig. 7 shows that FEM distributes the mining workload dynamically depending on the dataset characteristics.

TABLE IV
TIME DISTRIBUTION BETWEEN FP-TREE-MINING & TID-LIST-MINING OF FEM

Datasets	Minimum support	Total time (seconds)	Mining time (seconds)	FP-tree-mining (seconds & %)	TID-list-mining (seconds & %)
Connect	30000	25.55	25.24	0.02 **0.07**%	25.22 **99.93**%
Mushroom	50	21.64	21.60	0.10 **0.5**%	21.50 **99.5**%
Accidents	25000	18.61	15.72	7.72 **49**%	8.00 **51**%
Retail	5	1.62	0.84	0.39 **47**%	0.45 **53**%
Webdocs	150000	156.65	112.51	112.25 **99.7**%	0.26 **0.3**%

Fig. 7 Time distribution between FP-tree-mining & TID-list-mining

For the dense datasets Connect and Mushroom, *TID-list-mining* was responsible for over 99% of the mining time. The shape of FP-tree of dense datasets is usually compact, so most of the conditional pattern bases satisfy the condition to switch from *FP-tree-mining* to *TID-list-mining*. In contrast, for the very large and sparse dataset Webdocs,

FP-tree-mining was responsible for 99.7% of the mining time because the large number of big FP-trees were generated and processed. For Accidents and Retail datasets, mining time was distributed equally for the two mining tasks which were 49% vs. 51% on Accidents and 47% vs. 53% on Retail. The percentage time was computed using the runtime of mining stage and the runtime of each mining task. It must be noted that the runtime distribution does not indicate the amount of work. In fact, *TID-list-mining* using faster bitwise operations and better memory layout will process larger amount of data than *FP-Tree-mining* does in the same amount of time. In addition, the runtime distribution will change when the minimum support varies.

6 Conclusion and future work

In this paper, we presented FEM, a new frequent pattern mining algorithm that combines the mining techniques of two famous algorithms FP-growth and Eclat. The performance merit of FEM is achieved by adapting the mining process to match the characteristics of the datasets. The combination of the optimization techniques for implementing FEM contributes to the improvement of performance as well. In future work, we plan to improve FEM further by integrating several other optimization techniques. We will investigate how to find the optimal value of K for specific databases based on their characteristics. In addition, we will study parallel approaches for implementing FEM on parallel and distributed systems as memory limitation is the largest barrier to deploy any sequential frequent pattern mining algorithm on large scale databases.

7 References

[1] R. Agrawal, R. Srikant, "Fast algorithms for mining association rules", in *Proc. of the Int. Conf. on Very large databases*, pp. 487-499, 1994.

[2] M. Zaki, S. Parthasarathy, M. Ogihara, W. Li, "New algorithms for fast discovery of association rules", in *Proc. of the 3rd Int. Conf. on Knowledge Discovery and Data Mining*, pp. 283-286, 1997.

[3] J. Han, J. Pei, Y. Yin, "Mining frequent patterns without candidate generation", in *Proc. of the Int. Conf. on Management of Data*, 2000.

[4] S. Brin, R. Motwani, C. Silverstein, "Beyond market basket: generalizing association rules to correlations", in *Proc. of the Int. Conf. on Management of Data*, 1997.

[5] C. Silverstein, S. Brin, R. Motwani, and J. Ullman, "Scalable techniques for mining causal structures". in *Proc. of the Int. Conf. on Very Large Data Bases*, pp. 594–605, 1998.

[6] R. Agrawal and R. Srikant, "Mining sequential patterns", in *Proc. of the Int. Conf. on Data Engineering*, pp. 3–14, 1995.

[7] H. Mannila, H. Toivonen, and A. I. Verkamo, "Discovery of frequent episodes in event sequences", *Data Mining and Knowledge Discovery*, Vol. 1 Issue 3, pp. 259-289, Sep. 1997.

[8] J. Han, G. Dong, Y. Yin, "Efficient mining of partial periodic patterns in time series dataset", in *Proc. of the Int. Conf. on Data Engineering*, pp. 106-115, 1999.

[9] J. Han, H. Cheng, D. Xin, X. Yan, "Frequent pattern mining: current status and future directions", *Journal Data Mining and Knowledge Discovery*, Vol. 15 Issue 1, pp. 55-86, August 2007.

[10] JS. Park, MS. Chen, P. Yu, "An effective hash-based algorithm for mining association rules", in *Proc. of the Int. Conf. on Management of Data*, pp. 175–186, 1995.

[11] D. Burdick, M. Calimlim, J. Gehrke, "MAFIA: a maximal frequent itemset mining algorithm for transactional databases", in *Proc. of the Int. Conf. on Data Engineering*, pp. 443–452, 2001.

[12] C. Borgelt, "An implementation of the FP-growth algorithm", in *the 1st Int. Workshop on OSDM: Frequent Pattern Mining Implementations*, 2005.

[13] G. Grahne, J. Zhu, "Efficiently using prefix-trees in mining frequent itemsets", in *Proc. of Workshop on FIMI*, pp 123–132, 2003

[14] A. Fiat, S. Shporer, "AIM: another itemset miner", in *Proc. of Workshop on FIMI*, 2003.

[15] S. Shporer, "AIM2: improved implementation of AIM", in *Proc. of Workshop on FIMI*, 2004.

[16] L. Liu, E. Li , Y. Zhang, Z. Tang, "Optimization of frequent itemset mining on multiple-core processor", in *Proc. of the 33rd Int. Conf. on VLDB*, 2007.

[17] J. Pei, J. Han, H. Lu, S. Nishio, S. Tang, D. Yang, "Hmine : Hyper-structure mining of frequent patterns in Large Databases", In *Proc. IEEE of Int. Conf. On Data Mining*, pp. 441–448, 2001.

[18] B. Racz. "nonordfp: An FP-growth variation without rebuilding the FP-tree", in *Proc. of ICDM Workshop on FIMI*, 2004.

[19] Frequent Itemset Mining Implementations Repository, *Workshop on FIMI*, 2003-2004 Available: http://fimi.ua.ac.be/

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

[21] M. J. Zaki, K. Gouda, "Fast vertical mining using diffsets", *Technical Report* 01-1, RPI, 2001.

[22] L. Zhou, Z. Zhong, J. Chang, J. Li, J.Z. Huang, S. Feng, "Balanced parallel FP-Growth with MapReduce", in *Conference on Information Computing and Telecommunications, IEEE*, pp. 243 – 246, 2010.

[23] H. Toivonen, "Sampling large databases for association rules", in *Proc. of the 1996 Int. Conf. on VLDB*, pp. 134–145, 1996.

[24] S. Brin, R. Motwani, JD. Ullman, S. Tsur, "Dynamic itemset counting and implication rules for market basket analysis", in *Proc. of the 1997 Int. Conf. on Management of Data*, pp. 255–264, 1997.