Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2017) 25: 2096 – 2107
© TÜBİTAK
doi:10.3906/elk-1602-113

Smart frequent itemsets mining algorithm based on FP-tree and DIFFset data structures

George GATUHA[*], **Tao JIANG**
College of Information & Communication Engineering, Harbin Engineering University, Harbin, Heilongjiang, P.R. China

Received: 09.02.2016 • **Accepted/Published Online:** 18.08.2016 • **Final Version:** 29.05.2017

Abstract: Association rule data mining is an important technique for finding important relationships in large datasets. Several frequent itemsets mining techniques have been proposed using a prefix-tree structure, FP-tree, a compressed data structure for database representation. The DIFFset data structure has also been shown to significantly reduce the run time and memory utilization of some data mining algorithms. Experimental results have demonstrated the efficiency of the two data structures in frequent itemsets mining. This work proposes FDM, a new algorithm based on FP-tree and DIFFset data structures for efficiently discovering frequent patterns in data. FDM can adapt its characteristics to efficiently mine long and short patterns from both dense and sparse datasets. Several optimization techniques are also outlined to increase the efficiency of FDM. An evaluation of FDM against three frequent itemset data mining algorithms, dEclat, FP-growth, and FDM* (FDM without optimization), was performed using datasets having both long and short frequent patterns. The experimental results show significant improvement in performance compared to the FP-growth, dEclat, and FDM* algorithms.

Key words: Association rule data mining, FP-tree, Eclat, FP-growth, frequent itemsets

1. Introduction

The introduction of the frequent itemset concept by Agrawal et al. [1] in 1993 made mining association rules a very popular data mining technique for finding unique relationships in datasets. The AIS algorithm was among the first data mining techniques proposed for association rules mining and the algorithm was later improved, giving rise to the a priori algorithm [2].

The a priori algorithm uses a bottom-up, breadth-first search and a harsh tree structure to generate candidate $k+1$-itemsets obtained from frequent k-itemsets. The algorithm first scans the database and computes all frequent items at the bottom. From the frequent itemsets obtained, a set of candidate 2-itemsets is formed. A second database scan is performed to obtain the support of the candidate itemsets. The process is repeated several times until all frequent itemsets are realized. The algorithm uses the downward closure property (all frequent itemsets subsets must be frequent). Therefore, only frequent k-itemsets are utilized for constructing candidate $(k+1)$-itemsets.

Several variations of the a priori algorithm were proposed in [3–6] and they show fairly good performance, especially on sparse datasets having short frequent itemsets such as market basket data. However, it has been found that their performance degrades considerably on dense-long frequent patterns datasets such as census

[*]Correspondence: ggrpb@yahoo.com

and telecommunication data. This degradation is due to several database scans performed by the algorithms to generate candidate itemsets. This in turn incurs high I/O overhead costs for scanning large databases repeatedly. Secondly, checking large candidate itemsets by pattern matching and especially on long patterns is computationally expensive since a frequent pattern of length n implies examining $2^n \times 2$ frequent items as well. When n is large the frequent itemsets mining methods are CPU-dependent as opposed to I/O. A comprehensive survey and analysis of association rule mining algorithms was done in [7].

FP-growth [8] has a unique compressed database representation, i.e. the FP-tree structure. It constructs a conditional FP-tree and mines this structure recursively to obtain frequent itemsets in a divide-and-conquer method. The H-mine algorithm is a variant of FP-growth that was proposed in [9]. It is essentially a derivative of FP-growth because it partitions the search space using the divide-and-conquer method but does not construct the FP-tree structure or physically projected databases. The algorithm employs trie-based and array-based data structures when dealing with dense and sparse datasets, respectively.

Eclat [10] uses depth-first transversal on a vertical database representation and counts the support of itemsets by using transaction identifier (Tid) intersections. It has been found to be very efficient especially on dense datasets. Several algorithms such as those presented in [3,11] also use the vertical database representation and Tid intersection for efficiency; however, they use compressed bitmaps as a representation of every itemset appearing on the transaction list. The compression bitmaps scheme has some shortcomings, especially if the transaction Tids are evenly distributed. To mitigate this problem, dEclat [12] was developed. The algorithm stores the difference of Tids referred to as the DIFFset between its prefix k-1 frequent itemset and candidate k-itemset instead of the Tid intersection sets (Tidsets). The support of the itemset is computed by subtracting the cardinality of the DIFFset from the support of the k-1 frequent itemset's prefix. The performance of the dEclat algorithm has experimentally been shown to be better than that of Eclat [12].

We propose a new algorithm that is based on the unique features of FP-tree and DIFFset data structures. Our algorithm is smart in the sense that it is able to switch between FP-tree and DIFFset-list mining techniques depending on the database under analysis. It uses the FP-tree structure, similar to FP-growth, for storing the compressed data structure and recursively mines from this FP-tree. If the conditional pattern base of the frequent itemset is small in size it is automatically transformed into the DIFFset-list mining technique. FDM efficiently mines both short and long patterns from dense and sparse datasets. Optimization techniques have also been suggested to improve the efficiency of our algorithm. Experimental results indicate a significant improvement in performance compared to FP-growth and dEclat data mining techniques on dense and sparse datasets, respectively.

The rest of this paper is arranged as follows: Section 2 gives the background of the study, Section 3 presents materials and methods, Section 4 outlines the experimental results and analysis, and Section 5 concludes the paper.

2. Background

2.1. Formal statement of the problem

The formal statement of describing association rule mining is as follows:

Let $I = \{i_1, i_2, ..., i_d\}$ be a representation of a set of distinct items in a database and $T = \{t_1, t_2, ..., t_m\}$ be the set of all transactions. Generally an itemset is a set of items where a k-itemset is an itemset having k items. For example {Diaper, Milk, Beer, Bread} is a 4-item itemset. An empty (null) itemset contains no items. D represents a database containing a set of transactions; Ti represents a set of items such that $Ti \subseteq I$.

$| Ti |$ is a representation of the number of items in a transaction Ti while $| D |$ represents total transactions in a database D. A unique identifier, a TID, is assigned to each transaction. An association rule is therefore an implication expression of $X \rightarrow Y$, where X and Y are disjoint itemsets, i.e. $X \cap Y = \emptyset$. Association rule strength is determined in terms of support count and confidence. Support count is an important property of itemsets; it is a representation of the number of transactions in a database containing a certain itemset. The support count, $\sigma(X)$, for an itemset X can be mathematically stated as:

$$\sigma(X) = | \{ti | X \subseteq ti,\, ti \in T\} | \tag{1}$$

$\{ti | X \subseteq ti,\, ti \in T\}$ denotes number of elements in a given set.

Confidence $X \rightarrow Y$ is the ratio, usually in percentage, of the total number of transactions having $X \cup Y$ to the number of transactions containing X in D.

Support count is an illustration of how often a certain rule is applicable to a certain dataset; on the other hand, confidence illustrates the frequency at which items in Y appear in transactions containing X.

Formal illustrations of these metrics are as follows:

$$\text{Support, } s(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{N} \tag{2}$$

$$\text{Confidence, } c(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)} \tag{3}$$

A rule with very low support may simply occur by chance and may also be uninteresting from a business point of view since it may be unprofitable to promote items that customers rarely buy.

Confidence is a measure of how reliable an inference made by a rule is. For a rule $X \rightarrow Y$, the higher the confidence, the higher the likelihood of Y being present in transactions containing X. Confidence can also provide the basis for estimating the conditional probability of Y given X.

2.2. FP-growth algorithm and the FP-tree structure

FP-growth was proposed by Han et al. [13] and is one of the most popular algorithms after the a priori and Eclat algorithms. It performs a depth-first search just like Eclat on all candidates and generates recursively all i-conditional databases. However, it uses the FP-tree structure for counting the support of candidates instead of the intersection-based technique used in Eclat. It stores all the transactions in a trie-based structure, and each item has a linked list on all transactions that occur together instead of storing every frequent item cover. The trie structure ensures that a prefix that is shared in several transactions is only stored once. Compared to Eclat it has been shown to be more efficient in terms of memory utilization.

The main advantage of the FP-tree technique is that it can greatly exploit the single prefix path case, i.e. when all the transactions seem to be sharing the same prefix in the observed database, the prefix can then be removed and all its subsets can be added to all frequent itemsets that can be found, therefore increasing efficiency in performance.

Essentially the FP-growth algorithm requires two database scans. In the first scan it computes a set of frequent itemsets sorted in descending order. The second scan yields an FP-tree structure that is a compressed database representation. The algorithm employs an FP-tree-based frequent pattern growth mining technique that starts from an initial suffix pattern, examining a conditional pattern base only; constructing a conditional FP-tree and frequent itemsets mining is done recursively. The pattern growth is achieved by joining the suffix pattern with the ones generated from the conditional pattern. Finally, the search method is a partition-based divide-and-conquer technique significantly reducing the size of the conditional pattern base of the FP-tree.

For example, the database in Table 1 has six transactions $T = \{1; 2; 3; 4; 5; 6\}$ and five items $I = \{$A; C; T; D; W$\}$. If we set the minimum support to 3 (50%), we have AT, TW, DW, ACT, CDW, ATW, CTW, and ACTW frequent itemsets meeting this threshold, while if we set the minimum support to 100% we only have C satisfying this condition. Figure 1 represents the FP-tree structure of the database.

Table 1. Database.

Tid	Items
1	A, C, T, W
2	C, D, W
3	A, C, T, W
4	A, C, D, W
5	A, C, D, T, W
6	C, D, T

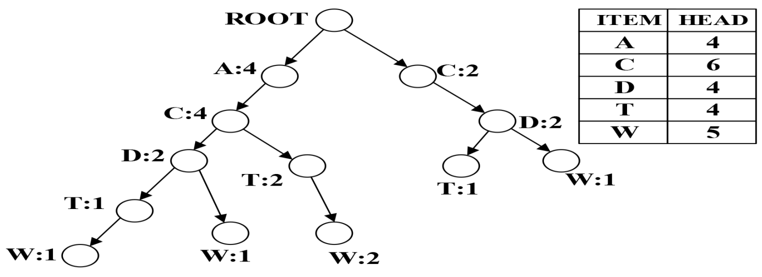


Figure 1. FP-tree of dataset.

2.3. DIFFset data structure

Zaki et al. [14] proposed an improvement to the Eclat algorithm, significantly reducing the memory requirements and computation of support even faster by using a vertical layout. They called this technique the DIFFset data structure or the dEclat algorithm. They proposed a method of storing only the differences in Tids between the class member and its prefix. The differences are stored in what they referred to as DIFFsets, which are basically the difference of two Tidsets. These differences run from the root to the node and finally to the children. Using this technique there is a significant reduction in the cardinality of sets, resulting in faster intersection and less memory utilization.

Formally, let us consider an equivalence class having prefix P and containing itemsets V,W. Let $t(V)$ represent the Tidset of V and $d(V)$ represent the DIFFset of V. When we use the Tidset method, we shall have *t(PV)* and *t(PW)* in the equivalence class, and to obtain *t(PVW)*, we check the cardinality of $t(PV) \cap t(PW) = t(PVW)$. If we use the DIFFset format, we have *d(PV)* instead of *t(PV)* and $d(PV) = t(P) - t(V)$, the set of Tids in $t(P)$ but not in $t(V)$. Similarly, we have $d(PW) = t(P) - t(W)$. So the support of PV is not equivalent to the size of its DIFFset. By the definition of *d(PV)*, it can be seen that $X|t(PV)| = |t(P)| - |t(P) - t(V)| = |t(P)| - |d(PV)|$. In other words, $sup(PV) = sup(P) - |d(PV)|$.

The DIFFset data structure compresses the database exponentially as longer itemsets are found. This allows the DIFFsets method to be extremely scalable compared to other methods. In Figure 2 it can be seen that TIDset database requires more memory resources to store 23 Tids compared to the DIFFset database that requires only 7 Tids.

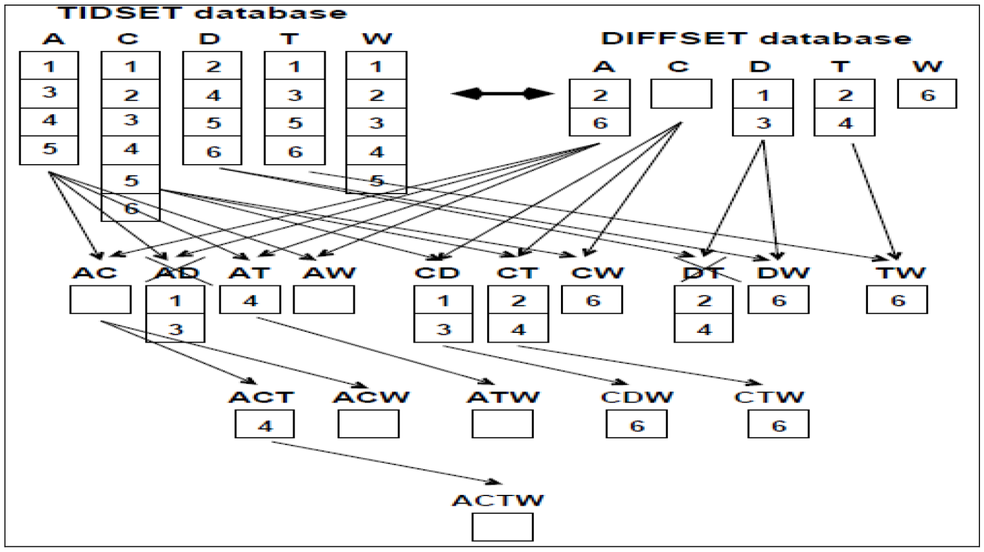


Figure 2. Comparison of TIDSET and DIFFSETS tree projection.

3. Materials and methods

3.1. Overview of the FDM algorithm

Frequent patterns are obtained recursively from the conditional FP-trees as in the FP-growth algorithm. The FP-tree shape is determined by the nature of the datasets; sparse datasets have large FP-trees and in dense datasets they are usually compact. However, in both cases the size of the FP-tree is usually compressed in comparison to the initial FP-tree. In our experiments we found that the size of the FP-tree is reduced significantly whereby mining using other data structures improves performance. It is also easier to convert the conditional pattern base of a particular itemset into DIFFsets that are more cache-friendly compared to pointers and linked lists.

The FDM algorithm combines the unique qualities of FP-growth and the dEclat algorithm. It uses an FP-tree structure and DIFFset list for performing the mining tasks. The algorithm switches between FP-growth and the dEclat mining strategy depending on the nature of the data to be analyzed. FDM comprises three main parts:

Construction of the FP-tree structure: Similar to FP-growth, the database is scanned first to obtain frequent itemsets and a header table is developed. The second database scan sorts the frequent itemsets in descending order of support and the construction of the FP-tree structure is done.

FP-tree mining: The task is similar to FP-growth mining. The conditional FP-tree structure is made and recursive mining is done to obtain the frequent itemsets. This is where the switching is done between dEclat and FP-growth depending on the size of the conditional pattern base. If the size is relatively small the FDM algorithm changes to the dEclat mining task. To improve the efficiency of the algorithm we represent the DIFFsets in bit vector format.

DIFFset mining: This process entails obtaining DIFFset-Tids using a bit vector and recursively searching for frequent patterns by ANDing the bit vectors. The patterns are obtained by joining the previous step's DIFFsets with the newly created frequent patterns. This technique is realized using the unique strategy of the dEclat algorithm. Using the DIFFset, the cardinality of sets representing itemsets is reduced significantly, resulting in faster intersection and less memory utilization [13].

3.2. Generating the DIFFset-list from a conditional pattern base

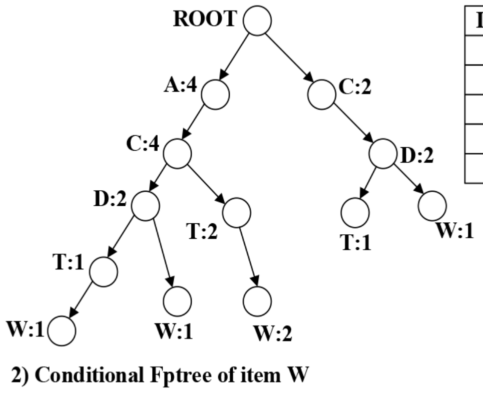
The authors in [14] defined a conditional pattern base as a subdatabase that comprised sets of frequent itemsets that occur together with the suffix patterns. The frequent items of an FP-tree have an equivalent conditional pattern base that is realized from the FP-tree.

During FP-tree mining several thousands of conditional patterns are processed. If the size is considerably small we switch to DIFFset bit vectors; otherwise, we perform FP-tree mining. We introduce a switching criterion where we use the number of nodes appearing in the linked list of y to decide whether to use the FP-tree or DIFFset mining. This criterion requires that we define a threshold X that acts as a logical boundary to assist in switching, and this criterion is used since a small number of nodes usually indicates a small size of y's conditional pattern base. If the number of nodes is more than X, FDM will switch to FP-tree mining, or else it will use the dEclat DIFFset mining strategy. The switching threshold X is explained in detail in Section 3.3 (component 4).

For example, in Figure 3, the conditional pattern base of W of the FP-tree structure from Figure 1 has 4 itemsets, i.e. {A:1,C:1,D:1,T:1} , {A:1,C:1,D:1} , {A:2,C:2,T:2} , {C:2,D:2}. If the FP-tree size is more than threshold X the conditional pattern will be used to construct the FP-tree structure as shown in Figure 3 (inset 2). Otherwise, these 4 itemsets will be transformed into a DIFFset-list for mining using the DIFFset mining technique. We assign each of the sets into a DIFFset-list and generate 4 lists, i.e. {4} for item A, {} for item C, {3} for item D, and {2,4} for item T. To benefit from memory and bitwise operation, the DIFFset-lists are transformed into bit-vector format [11] as shown in Figure 3 (inset 5). The weight of the frequencies is computed into a weight vector, which shall be used to compute the DIFFset-list support {1, 1, 2, 2} (Figure 3, inset 6).

{A:1,C:1,D:1,T:1}
{A:1,C:1,D:1,}
{A:2,C:2,T:2,}
{C:2,D:2}

1) Conditional pattern base of W



2) Conditional Fptree of item W

ITEM	HEAD
A	4
C	6
D	4
T	4
W	5

TID	Items	Weight
1	A,C,D,T	1
2	A,C,D	1
3	A,C,T	2
4	C,D	2

3)Database equivalent of W

A	C	D	T
4		3	2
			4

4)Diffset-list

A	C	D	T
0	0	0	0
0	0	0	1
0	0	1	0
1	0	0	1

5)Diffset-bit vectors

Weight
1
1
2
2

6)W weight vector

Figure 3. Conditional pattern base, DIFFset-list, and bit vectors of W.

3.3. The FDM algorithm

The FDM algorithm consists of four components:

1. FDM-mining: The initial mining process begins with construction of the FP-tree from the initial database. Thereafter, FP-tree-mining/DIFFset mining proceeds as shown in 2 and 3 depending on the nature of the dataset under study:

```
//$T$= tree, $\Phi$= itemsets, $\delta$= minimum support
*Input*: Transactional database $D$and $\delta$
*Output*: Complete set of frequent patterns
Step 1: Find all frequent itemsets by scanning $D$once
Step 2: Construct the FP-tree $T$ by scanning $D$the second time
Step 3: *itemsets* = total frequent itemsets in $D$
Step 4: *transactions* = total number of transactions in $D$
Step 5: Call switching (*itemsets*, *transactions*)
Step 6: Call FP-tree-mining process ($T$, $\Phi$, $\delta$)
```

2. FP-tree-mining: It uses the FP-growth algorithm strategy to recursively generate all the frequent patterns from the conditional tree. The size of the conditional tree is checked against the X threshold. If the size is less than X then a bit vector will be generated; otherwise, the FP-tree will be created.

```
Procedure FP-tree-mining (Conditional FP-tree $T$, suffix, $\delta$)
Output: Set of frequent itemsets
Step 1: If $T$ contains a single prefix path $P$ *//mining single prefix path FP-tree*
Step 2: Then for every combination $y$of the nodes in $P$
Step 3: Output $= y \bigcup$ *suffix*
Step 4: Else for each item q in the header table of FP-tree $T$
Step 5: Output $\beta = q \bigcup$ *suffix*
Step 6: Construct q conditional pattern base $C$
Step 7: $a$= Number of nodes q
Step 8: If $a > X$
Step 9: Then construct q's conditional FP-tree $T'$and call FP-tree-mining ($T'$, $\delta$, $\beta$)
Step 10: Else transform $C$into DIFFset-list bit vectors $Z$and the weight vector $w$and call bit vector mining ($Z$, $\beta$,$w$, $\delta$)
```

3. DIFFset-mining: This procedure collects all the DIFFset bit vectors from the database and recursively generates a frequent pattern by logically ANDing them. New patterns are created by joining the suffix patterns from the previous steps.

```
*Input*: Bit vectors $Z$, *suffix*, weight vector $w$, $\delta$
*Output*: A set of frequent itemsets
Step 1: Sort $Z$in descending order of its support frequency
Step 2: For each vector $v_i$in $Z$
Step 3: Output $\beta$ = item of $z_i \bigcup$ *suffix*
Step 4: For each vector $z_k$in $Z, k < i$
Step 5: $u_k = z_i$ AND $z_k$
Step 6: $sup_k$= support of $u_k$based on $w$
Step 7: If $sup_k \geq \delta$
Step 8: Then add $u_k$ into $U$
Step 9: If all $u_k$ in $U$ are identical to $v_i$
Step 10: Then for every combination $x$ of the items in $U$
Step 11: Output $\beta' = x \bigcup \beta$
Step 12: Else if $U$ is not null
Step 13: Then call DIFFset Mining ($U$, $\beta$, $w$, $\delta$)
```

4. Switching threshold: The value of X is determined based on database density estimation using the switching algorithm adopted from [15]. In the algorithm *NewPattern* and *Size* indicate both new frequent patterns obtained and the conditional pattern base size consecutively. The total number of frequent itemsets obtained for different values of X is stored in an array v that is used to determine the threshold for switching. The algorithm keeps track of the number of frequent patterns for several X-values that are multiples of 32 [16].

```
Input: NewPattern and Size
Output: Updated value of threshold $X$
Step 1: If switching is called for the first time then
Step 2: Create an array $P$ with $N$ elements
Step 3: Initialize all $Pi$ to zero
Step 4: For $i = 0$ to $N - 1$
Step 5: If Size $>$ i*Step then Pi = Pi + NewPatterns
Step 6: Else exit loop
Step 7: $X = 0$
Step 8: For $i = N - -1$ to 1
Step 9: If $Ri \geq 2$ then $X =$ (i+1)*Step and exit loop
```

3.4. Optimization techniques

Several factors determine the performance of frequent pattern mining algorithms such as CPU speed, data structure, memory size, database properties, I/O speed, and minimum support threshold.

Substantial work done in the FP-growth algorithm includes FP-tree traversal and the construction of new conditional FP-trees after the first tree is constructed from the first database scan. About 80% of CPU time is spent on traversing the FP-tree structure. The authors of [17] proposed an array-based prefix-tree structure that greatly improved FP-tree mining by limiting traversal when constructing the FP-tree. We adopted this approach in our algorithm to increase efficiency.

To further improve FP-tree transversal we have introduced, in the second database scan, a lexicographic list as suggested in [18]. The lexicographic list is organized into a binary tree and the order is maintained as the tree grows. The binary tree also maintains the nodes visited simultaneously next to each other in the memory. This improves the speed of mining. To improve the processing time of our algorithm, we utilize an indexed table as proposed in [19] for storing frequent output values. This greatly improves computation time, especially when we have big output size.

4. Results and discussion

To evaluate the FDM algorithm, eight experiments were performed using a Toshiba L635 Core i3 2.27 GHz, 2 GB RAM and running Windows 8.1 32-bit. The code was implemented in Java using the Eclipse platform. The FP-growth and dEclat algorithms were downloaded from [20] and used for benchmarking FDM. To evaluate the contribution of optimization, FDM was stripped of optimization giving rise to FDM*.

The experiments were performed using 6 datasets from FIMI's 03 repository [21]. The datasets are Mushroom, Pumb_star, Chess, Kosarak, Retail, and T10I4D100K. Table 2 shows the characteristics of the datasets.

Table 2. Experimental dataset characteristics.

Dataset	Transactions	Items	Average length	Nature
Mushroom	8124	119	23	Dense
Pumb_star	49,046	7117	50	Dense
Chess	3196	76	37	Dense
Kosarak	990,002	41,271	8.1	Sparse
Retail	88,126	16,470	10.3	Sparse
T10I4D100K	100,000	1000	40	Sparse

The experimental results show that FDM outperforms FP-growth (Figures 4–6) and dEclat for all tested sparse datasets (Figures 7–9) and dense datasets, respectively. FP-growth runs slowest on datasets Mushroom, Chess, and Pumb_star. This is attributed to the fact that the datasets are dense. dEclat runs slowest on datasets Kosarak, Retail, and T1014D100K since they are sparse.

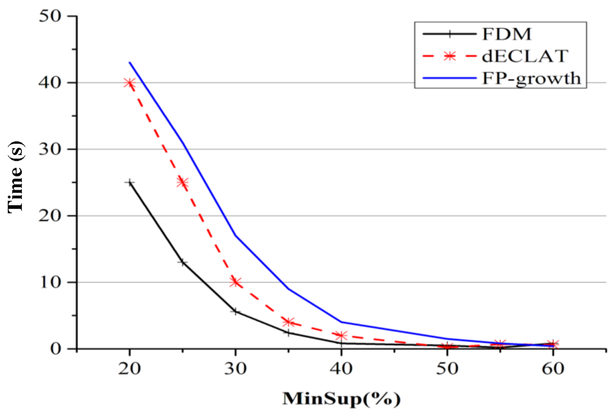


Figure 4. Chess dataset.

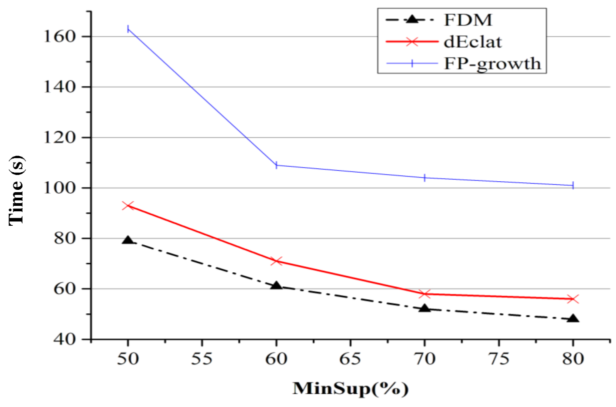


Figure 5. Mushroom dataset.

DIFFset mining is responsible for mining dense datasets since the shape of the FP-tree of dense datasets is usually compact and the conditional pattern bases meet the condition to switch from FP-tree mining to DIFFset mining strategy. In contrast to the sparse datasets, the FP-tree mining strategy is responsible for the mining task since many large FP-trees are generated and most of them fail to satisfy the switching criteria.

It is worthwhile to note that the mining distribution between the two strategies does not indicate the amount of work done. As a matter of fact, the DIFFset mining strategy using a more cache-friendly data layout and faster bitwise operations will process larger amounts of data compared to FP-tree mining under the same unit of time.

In addition, the distribution of mining varies when the minimum support changes. When the minimum support is set to low levels, more conditional FP-trees are generated and satisfy the condition to switch from FP-

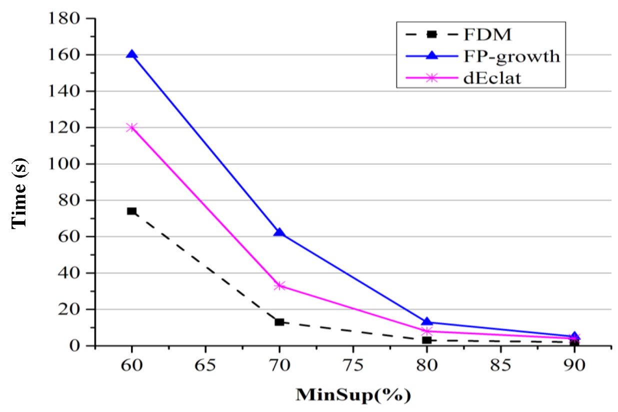


Figure 6. Pumb_star dataset.

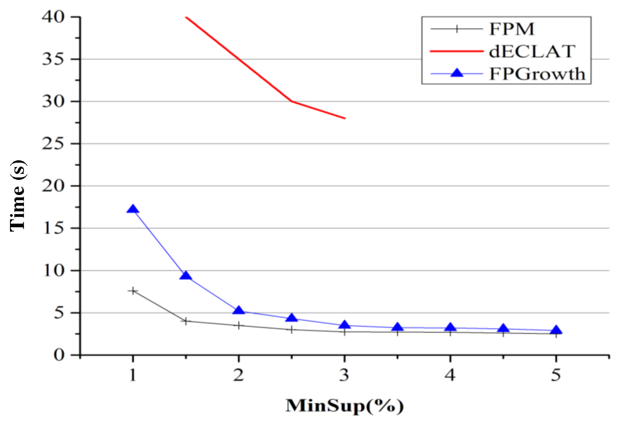


Figure 7. Retail dataset.

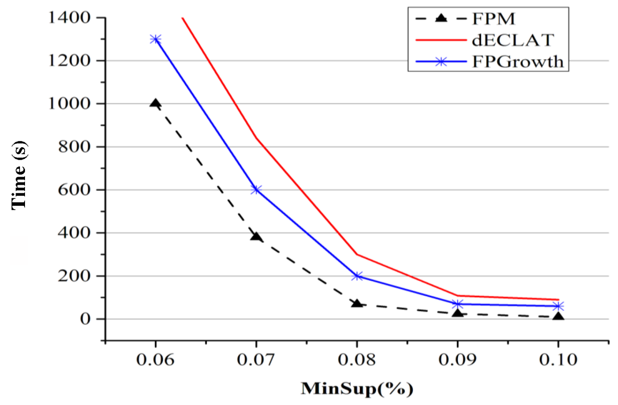


Figure 8. Kosarak dataset.

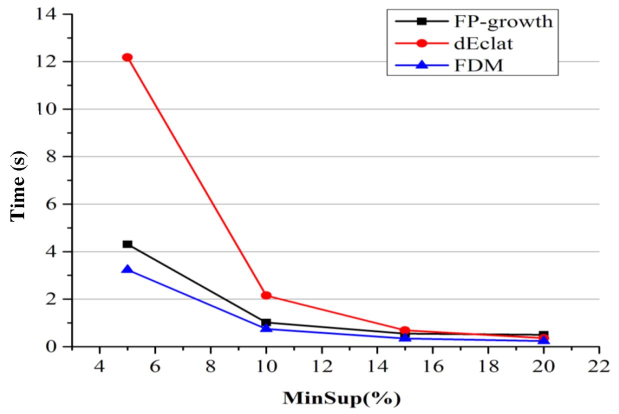


Figure 9. T1014D100K.

tree mining to DIFFset mining, making the percentage of mining time of DIFFset mining increase as minimum support reduces.

To provide insight into the performance merits of FDM and to determine the contribution of the optimization FDM was stripped of the optimization techniques. Experiments were performed on both dense datasets (Figure 10) using the Pumb_star dataset and sparse datasets (Figure 11) using Kosarak. A comparison

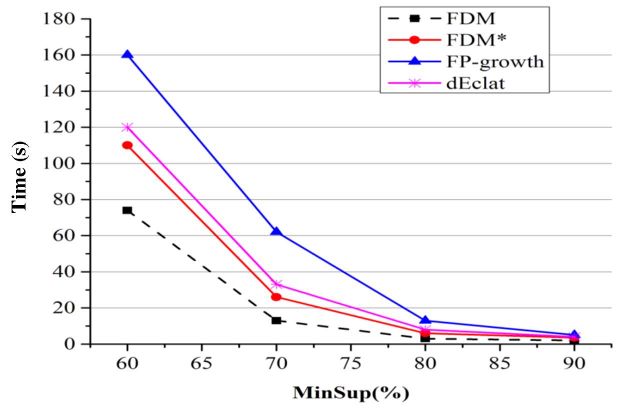


Figure 10. FDM without optimization on Pumb_star dataset.

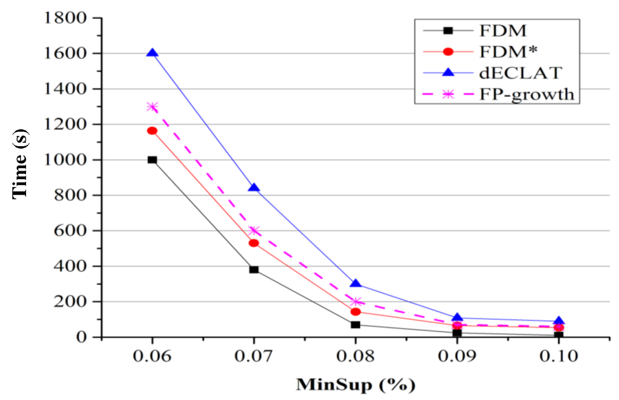


Figure 11. FDM without optimization on Kosarak dataset.

between FDM and FDM* indicates that there was a 32% improvement in runtime for sparse datasets and about 58% for dense datasets; it can therefore be concluded that optimization has contributed significantly for FDM.

In summary, FDM has the capability of switching between the two strategies at runtime by efficiently distributing the mining workload depending on the nature of the datasets under study. The optimization techniques in Section 3.4 were used in the implementation of FDM.

The datasets used in our experiments are publicly available and have been extensively cited and used in data mining experiments. Because dEclat is faster than Eclat [12] on dense datasets, our algorithm also outperforms Eclat. Since it has been proved that FP-growth [8] outperforms the a priori algorithm, it is only fair to conclude that our algorithm is much faster than the a priori algorithm.

5. Conclusion

In this paper, we have proposed FDM, a new frequent itemset mining algorithm that combines the power of both FP-tree and DIFFset data mining techniques for pattern recognition in data. The algorithm adapts the mining procedure depending on the nature of the dataset under analysis.

Experiments carried out have proved that the FDM algorithm outperforms both FP-growth and dEclat on datasets having short and long frequent patterns. It is also worthwhile to note that the optimization techniques have also contributed significantly to FDM performance.

In the future we plan to improve FDM implementation by introducing new and efficient optimization methods as well as to study the comparison of memory usage with other algorithms. We shall also study parallel approaches to implementing FDM in distributed systems, since limitations of memory and runtime have been identified as the major obstacles in the deployment of frequent pattern algorithms on large data warehouses.

Acknowledgment

This work was funded by the International Exchange Program of Harbin Engineering University for Innovation-Oriented Talents Cultivation.

References

[1] Agrawal R, Imielinski T, Swami AN. Mining association rules between sets of items in large databases. In: ACM SIGMOD International Conference on Management of Data; May 1993; Washington, DC, USA. New York, NY, USA: ACM. pp. 207-216.

[2] Agrawal R, Srikant R. Fast algorithms for mining association rules. In: 20th International Conference on Very Large Data Bases; 1994; Santiago, Chile. San Francisco, CA, USA: Morgan Kaufmann, pp. 487-499.

[3] Savasere A, Omiecinski E, Navathe S. An efficient algorithm for mining association rules in large databases. In: Proceedings of the 21st International Conference on Very Large Data Bases; 1995; Zurich, Switzerland. San Francisco, CA, USA: Morgan Kaufmann, pp. 432-444.

[4] Gunopulos D, Khardoni R, Mannila H, Saluja S, Toivonen H, Sewark R. Discovering all the most speci?c sentences. ACM T Database Syst 2003;28: 140-174.

[5] Xiang C, Sen S, Shengzhi X, Zhengyi L. DP-Apriori: A differentially private frequent itemset mining algorithm based on transaction splitting. Comput Secur 2015; 50: 74-90.

[6] Akshita B, Ashutosh G, Debasis D. Improvised apriori algorithm using frequent pattern tree for real time applications in data mining. Procedia Computer Science 2015; 46: 644-651.

[7] Wu X, Kumar V, Ross Quinlan J, Ghosh J, Yang Q, Motoda H, McLachlan GJ, Ng A, Liu B, Yu PS et al. Top 10 algorithms in data mining. Knowl Inf Syst 2007; 14: 1-37.

[8] Han J, Pei J, Yin Y. Mining frequent patterns without candidate generation. In: SIGMOD Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data; 2000; Dallas, TX, USA. pp. 1-12.

[9] Pei J, Han J, Lu H, Nishio S, Tang S, Yang D. Hmine: Hyper-structure mining of frequent patterns in large databases. In: Proceedings of the International Conference on Data Mining; November 2001; San Jose, CA, USA. pp. 441-448.

[10] Zaki M, Parthasarathy S, Ogihara M, Li W. New algorithms for fast discovery of association rules. In: Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining; 1997; Menlo Park, CA, USA. pp. 283-286.

[11] Burdick D, Calimlim M, Gehrke J. MAFIA: A maximal frequent itemset algorithm for transactional databases. In: Proceedings of International Conference on Data Engineering; April 2001; Heidelberg, Germany. pp. 443-452.

[12] Zaki MJ, Gouda K. Fast vertical mining using DIFFsets. In: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining; 2003; Washington, DC, USA. New York, NY, USA: ACM Press. pp. 326-335.

[13] Han J, Pei J, Yin Y, Mao R. Mining frequent patterns without candidate generation: a frequent-pattern tree approach. Data Min Knowl Disc 2004; 8: 53-87.

[14] Zaki MJ, Parthasarathy S, Ogihara M, Li W. New Algorithms for Fast Discovery of Association Rules. Technical Report URCS TR 651. Rochester, NY, USA: University of Rochester, 1997.

[15] Lan V, Gita A. Novel parallel method for mining frequent patterns on multi-core shared memory systems. In: Proceedings of the 2013 International Workshop on Data-Intensive Scalable Computing Systems; 2013; New York, NY, USA. pp. 49-54.

[16] Lan V, Gita A. Mining frequent patterns based on data characteristics. In: International Conference on Information and Knowledge Engineering; 16–19 July 2012; Las Vegas, NV, USA. pp. 369-375.

[17] Lan V, Gita A. Novel parallel method for association rule mining on multi-core shared memory systems. Parallel Comput 2014; 40: 768-785.

[18] Borgelt C. An implementation of the FP-growth algorithm. In: Workshop on Open Source Data Mining Software; 2005; New York, NY, USA. pp. 1-5.

[19] Shporer S. AIM2: Improved implementation of AIM. In: Proceedings of Workshop on FIMI; November 2004; Brighton, UK. pp. 23-32.

[20] Fournier-Viger PG, Gueniche T, Soltani A, Wu C, Tseng VS. SPMF: A Java open-source pattern mining library. J Mach Learn Res 2014; 15: 3389-3393.

[21] Lin JCW, Gan W, Fournier-Viger P, Hong TP. RWFIM: Recent weighted-frequent itemsets mining. Eng Appl Artif Intel 2015; 45: 18-32.