Vertical Mining of Frequent Patterns Using Diffset Groups

Laila A. Abd El-Megid[1], Mohamed E. El-Sharkawi[2], Laila M. El-Fangary[3], Yehia K. Helmy[4]

[1, 3, 4] Dept. of Information Systems, Faculty of Computers & Information, Helwan University, Egypt,
[2] Dept. of Information Systems, Faculty of Computers & Information, Cairo University, Egypt
[1] eng.lole@yahoo.com, [2] mel_sharkawi@gmail.com, [3] lailaelfangary@hotmail.com,
[4]yehiahelmy@yahoo.com

Abstract

Frequent patterns discovery is a core functionality used in many mining tasks and large broad application. In this paper, we present a new algorithm, VMUDG, for vertical mining of frequent itemsets. The proposed algorithm adapts a new efficient approach that classifies all frequent 2-itemsets into separate groups according to their diffsets. Using these groups, the proposed algorithm offers three new distinct features; First, it allows calculating the support of N itemsets (N is>0) using one calculation process rather than N calculation processes. Second, it offers a chance to reduce the time needed for the manipulation of the itemsets diffsets. Third, it minimizes the need for checking the frequency condition for every itemset. A performance study of the proposed algorithm has been conducted. Several experiments show that the algorithm outperforms the well known dEclat algorithm.

Keyword*: Frequent patterns, verrtical mining, diffset, asscication rules, data mining*

1. INTRODUCTION

Frequent pattern mining has been a focused theme in data mining research for over a decade. It is a core technique used in many mining tasks like sequential pattern mining [1], structured pattern mining, correlation mining, associative classification, and frequent pattern-based clustering [2], as well as their broad applications [3] [4]. So, a great effort has been dedicated to this research and tremendous progress has been made to develop efficient and scalable algorithms for frequent pattern mining [5] [6]. The problem of mining frequent itemsets can be formulated as follows. Let I be a set of items and T a database of transactions, where each transaction has a unique transaction identifier (*Tid*) and contains a set of items. A set $X \subseteq I$ is called an *itemset*, and a set $Y \subseteq T$ is called a *tidset*. An itemset that contains *k* items is called a *k-itemset*. The *support* of an itemset X, denoted $\sigma(X)$, is the number of transactions in which X occurs. An itemset is *frequent* if its support is greater than or equal to a user-specified *minimum support* (*min_ sup*) value [6] [7].

In this paper, we present a new algorithm VMUDG (Vertical Mining using Diffset Groups) for vertical mining of frequent itemsets. The proposed algorithm uses the diffset data representation [6]. It depends on the classification of frequent 2-itemsets into groups according to their diffsets. It also structures the frequent 2-itemsets while they are processed in the groups compactly to achieve efficient memory usage.

Using these groups of frequent 2-itemsets, the algorithm efficiently generates frequent *k*-itemsets (where $k > 2$). This is done by a simple and direct extraction process from the groups, which consequently has a good effect on the level of performance. In addition, the new algorithm has three unique features like; first, in related work, on any level *L* each itemset is checked with other itemsets at this level to determine whether they can form frequent L+1-itemsets. So at any level, if we have *N* itemsets, we make $\sum i$ *(where i >=1 and i <= N-1)* number of checks between these itemsets. However, in our algorithm the itemsets are grouped into a set of groups. Each group contains multiple itemsets. The checks are done at the group level rather than the itemset level. If we have *N* itemsets distributed on *G* groups we will make $\sum i$ *(where i >=1 and i <= G-1)* number of checks between the groups. For any frequent item class has *N* itemsets; in the best case we have only one group and in the worst case we have *N* groups, and then we have an average $((N+1)/2)$ groups. This offers a chance for reducing the number of checks, done by the manipulation of the itemsets diffsets, and consequently results in minimizing the processing time. Second, the proposed algorithm offers the availability of calculating the support of *N* itemsets with one calculation process rather than *N* processes (one process for each itemset) as in related work. Third, in all related work, every itemset has to be checked for its frequency condition. However, the proposed VMUDG algorithm minimizes the need of checking the frequency of the itemsets through the validity condition it follows while developing the groups. Many experiments have been conducted to compare between the proposed VMUDG algorithm and the well known dEclat algorithm. All experiments show that the features of the proposed algorithm make it outperforms the dEclat algorithm. The rest of the paper is organized as follows: In section two we list and discuss the related work. Section three explains in details the proposed VMUDG algorithm and specifies its features. Also, in section three an illustrative example is given to show how the proposed algorithm carries out mining of all frequent itemsets. A performance study of the proposed VMUDG algorithm is given in section four. Finally, a conclusion is given in section five.

2.Related work

In this section, we will review the algorithms that follow the vertical mining approach for mining frequent itemsets. Eclat [7] is the first algorithm proposed to generate all the frequent itemsets using the depth-first strategy. It uses a tidset representation, where each item is associated with a set of transaction identifiers (Tids) where the item was shown in. The strength of the algorithm stems from conducting the mining process through a simple intersection operation on the transaction ids of the items. To perform this fast counting mechanism it did not use any complex data structure like the hash tree as some Apriori-like algorithms use. The algorithm, however, starts to suffer when being used with dense data sets that are characterized by high item frequency. In such case, the processing time required to perform the intersection process between the tidsets starts to be larger. Also, the required memory space required to store the tidsets incrementally grows.

The problems raised with the Eclat algorithm that follows the traditional tidset vertical data layout motivate to search for another vertical representation of data that can avoid these problems. Zaki [9] proposed a novel vertical data representation called Diffsets and implemented on the DEclat algorithm. At the level of frequent items (1-itemset), the diffset keeps for each frequent item the tids where this item has not been shown. However, for other frequent *k*-itemsets (where *k*>1) the diffset for a k+1-itemsets keeps only track of the differences of the diffset of its generating frequent k_itemsets. The dEclat algorithm has experimentally shown to result in significant performance enhancements and more memory reduction. And also although it reduced the memory required but it still needs to store each itemset with its diffset for processing further itemsets.

A new algorithm based on the diffset structure, FVDM [8] has been proposed to enhance the performance of the vertical mining algorithms. The algorithm follows the pattern growth methodology and utilizes it over the diffset structure. The algorithm consists of two main phases. The first phase is concerned with the mining of all long patterns for each item, while the next phase is concerned with generating all the sub itemsets from the previous discovered long patterns. Experiments have shown that the FDVM algorithm drastically outperforms the dEclat algorithm. However, the algorithm still needs the existence of the database into the memory to be scanned to check the frequency condition at the phase of generating hidden itemsets that are itemsets that have not been generated or considered before. It needs to recheck the dataset *N* times where *N* is the number of hidden itemsets which is a critical disadvantage when being used in large databases with real time requirements.

As shown, we can observe that all the previous algorithms require the existence of the database into the main memory in order to be able to conduct the mining process completely. If the database is too large to fit into the main memory, the algorithms will not be applicable or they will be enforced to use some other techniques that make the algorithms pay in performance. Goethals [9] has proposed the Medic algorithm to solve the problem of insufficient available memory space. It applies a simple technique on the basic form of the Eclat algorithm. The idea behind the Medic algorithm is to manipulate the frequent items with their corresponding Tids of the currently read transactions instead of all the database transactions. Also, as soon as there are items that could no longer exist in further coming transactions, the algorithm will remove these items with their corresponding tidset from the memory. By this mechanism, the whole database may never be entirely loaded into the main memory giving the algorithm a new strength. The advantages of this algorithm can be gained when being used in spare data sets. But the case will differ when being used in dense data sets, as we will find a lot of high frequency items that require a major part of the database to be retained in the main memory until they have been processed, leading the algorithm to lose its advantages.

Recently, Song and Rajasekaran [10], present a novel algorithm for mining frequent itemsets. This algorithm is referred to as the TM (Transaction Mapping) algorithm. In this algorithm, the tids of each itemset are mapped and compressed to continuous transaction intervals in a different space and the counting of itemsets is performed by intersecting these interval lists in a depth-first order along the lexicographic tree. When the compression coefficient becomes smaller than the average number of comparisons for intervals intersection at a certain level, the algorithm switches to transaction id intersection as in Eclat. Though the gained advantages of the TM algorithm, it still suffers from some problems; first, we may face the situation where the numbers of continuous intervals of tids become too close to the number of tids itself and accordingly the algorithm will lose its main feature. Second, there is an additional overhead required for calculating the intervals of the tids and monitoring the mining process to decide to continue using the algorithm or to switch to normal tidset approach. Also, in the same paper they have combined the dEclat algorithm with the TM algorithm and producing the dTM algorithm. However, experiments have shown that the performance of the dTM algorithm is worse than that of TM. It also shows that it performs almost like the dEclat in some experiments and in other experiments it is worse than the dEclat. This is due to the complex mechanisms it uses in calculating the support of the itemsets.

3. The proposed VMUDG algorithm

In this section we show how the proposed algorithm carries out mining of all frequent itemsets. The algorithm consists of two phases. Initially, the algorithm scans the source data to find frequent items. These items are represented using the diffset vertical representation. In the first phase; the frequent 2-itemsets are generated and classified into a set of groups. From the generated groups, the second phase extracts other *k*-itemsets, where $k \geq 3$. These phases will be described in the following sections. Before we proceed, we need the following definitions. For convenience we write an itemset {A, C, W} as ACW, and a diffset {2, 4, 5} as 245.

Definition 1: Given an itemset I ($i_1, i_2, \ldots, i_n$), the *prefix* of I, is the first item i_1.

Definition 2: Given a frequent item *i*, we associate with *i* a set of groups G = {g_{i1}, g_{i2},…,g_{ik}}. There are two types of groups; basic groups and class groups. The class groups are developed by specific manipulation of the basic groups. Once all the class groups of a frequent item *i* are generated there is no need for the basic groups and they are removed from the memory. The next section explains the process of constructing these groups. The overall process of the VMUDG algorithm is illustrated in Fig. 1

A. Group Construction

The following are the steps to construct the groups for each frequent item *i*:

Step 1: constructing basic groups: Each basic group has a header that contains the group's diffset, and a set of frequent 2-itemsets, each has item i as a prefix and has the same diffset as the group's diffset. The construction of the basic groups is done as follows: Generate all frequent 2-itemsets that have item i as a prefix. Once a frequent 2-itemset is generated we search for a group that has a diffset equal to the diffset of the currently processed frequent 2-itemset. If such a group is found, we union the processed frequent 2-itemset with the only itemset already existing in the group. For example, by taking the union of itemset AB and itemset AC, we have the itemset ABC. However, if such group is not found, create a new group for the processed frequent 2-itemset. The diffset of this group is the diffset of the frequent 2-itemset. The union of itemsets in the basic groups is done for efficient memory usage. Note that, as a result of continuous itemset union, any basic group always has one frequent itemset (not necessarily of length 2). Note that, basic groups have different diffsets. Fig. 2

Step 2: constructing class groups: once all basic groups are developed we start constructing the class groups. Given two groups (either basic or class) that belong to the groups of item i, if the two groups satisfy the following validity condition, we generate a class group by taking the union of the two groups. Where the validity condition is:

$$\mathbf{T - C - TI >= min\_sup}$$

Where,

- T= Total number of transactions
- C= │tids in the diffset of the item class│
- I = │tids in the diffset of the itemset│
- For any itemsets (M_1 ,M_2, …, M_n) :
  TI= │tids (M_1.diffset U M_2.diffset U… U M_n.diffset │

Here, TI represents the number of *Tids in the diffsets of all itemsets exist in g_{ia} and g_{ib}*

The union process aims to put in one group all frequent itemsets that exist together in a number of transactions that is greater than or equal to the minimum support. So, the previous rule ensures that all itemsets in the newly generated group are frequent. Fig. 3 shows the pseudo code of generating class groups. The itemsets in the groups with their corresponding diffsets are put together in one group. For example, if we have group G1 with itemset ABC and diffset 3 and another group G2 with itemset ADE and diffset 2, by union these two groups we have one group that has two records (tuples), each record is associated with one of the previous itemsets and its corresponding diffset.

```
Algorithm VMUDG
------------------------------------------------------------------------------------------
Input:   Min_Sup
         D  // The set of frequent items
         T  // Total number of transaction in the data set

Output: All frequent k-itemsets (where k>=1)

1   For each item $x_i \in$ D
2   A set of groups GS is empty
3           For each item $x_j \in$ D, having j> i
4                       R= $x_i \mu x_j$
5                       $\sigma$ (R) = $\sigma$ ($x_i$) - │d(R)│     // d (R) is the diffset of R
6                       If ($\sigma$ (R)) >= Min_sup then
7                       Basis_groups_construction(R, d(R))
8                       End if
9            Next $x_j$
10   Class_groups_construcion()
11   Next $x_i$
12   For each group $g_i$ in GS
13          For each itemset i in g
14                      Extract all sub itemsets and calculate the support as
                        in Rule 1
15          Next i
16   For each group $g_i \in$ GS
17          For each group $g_j \in$ GS, having j> i
18                      Union $g_i$ and $gj$
19                      Extract all sub itemsets and calculate the support as
                        in Rule 2
20          Next j
21    Next i
```

Figure 1. VMUDG algorithm

```
Basis_groups_construction (new_item, new_diffset))

    // Each group G has a header and an itemset

1     For each group g ∈ GS
2            If (new_diffset = g.header) then
3             g.itemset=g.itemset ∪ new_item
4             Exit
5            End if
6     Next g
7    GS.add_new_group(new_diffset, new_item)
```

Figure 2. Basis_groups_construction procedure

The union process between the groups is repeated respectively until there are no groups that can be joined together.

The union of basic groups to generate class groups provides a performance optimization for the algorithm on both levels of memory and processing time. For processing time, unlike previous algorithms that check each frequent 2-itemset intersection with other frequent 2-itemsets, we only check the intersection between the diffsets group where each group contains a set of frequent 2-itemsets. For memory usage, all frequent 2-itemsets that have the same diffset are merged together and the diffset is stored only once as one unit and also processed as one unit.

After all itemsets of class Ci are processed and assigned to the class groups, the next phase is applied on these groups. That is the two steps of the algorithm are pipelined and there is no need to wait for generating groups of other classes (frequent items).

B. Generating frequent k-itemsets

After generating and grouping all the frequent 2-itemsets of a class, we start the phase of generating the frequent k-itemsets.

```
Procedure Class_groups_construcion()
--------------------------------------------------------------------
1  For each group $g_i \in$ GS
2          For each group $g_j \in$ GS, having j> i
3              tno= $|g_i.header \mu g_j.header|$
4              If (Total –tno)  >= Min_sup then
5                      new_group= $g_i \cup g_j$
6                      GS.add_new_group()
7                      new_group.items.add($g_i$.item, $g_i$.header)
8                      new_group.items.add($g_j$.item, $g_j$header )
9                      End if
10         Next j
11   Next i
12   Delete all pre-processed basic_groups from GS
```

Figure 3. Class_groups_construction procedure

The mining process is conducted over each group in the class. Since the existing groups failed to be combined together according to the previous validity condition, that is they cannot form together further frequent k-itemsets, each group can be mined independently with no interaction with other groups. In the mining process, we first extract the frequent sub itemsets from each itemset in the group starting with the itemset's prefix. For example, when extracting sub itemsets of the ABCD itemset, we will have: AB, AC, AD, ABC, ABD, and ACD. The supports of all these itemsets are calculated once by Rule 1.

Rule 1

$$\textbf{Support (S) = T – C – I}$$

Then we join the itemsets in the group respectively and extract all sub itemsets from the new joined itemset to generate other frequent itemsets. For all extracted itemsets from any itemsets (M_1 ,M_2, …, M_n) the support of all extracted itemsets are calculated once according to Rule 2.

Rule 2

$$\textbf{Support (S) = T – C – TI}$$

Note that we can skip the process of calculating the support and produce only the frequent itemsets through the direct extraction from the itemsets in the group and thus offer a chance for reducing the processing time. Here we have not to check the frequency condition of the extracted itemsets because that they are frequent (because they exist in the same group that has been checked previously according to the validity condition).

After applying the previous two steps on all the groups existing in all the classes, we will have a complete set of frequent itemsets of all available lengths with their corresponding support.

C. Illustrative example

Here we explain how the proposed VMUDG algorithm works through all its phases with an illustrative example. The objective is to mine all frequent itemsets in the transactional data given on Table 1 having a minimum support value equals three.

Table 1: Transactional data

Tid	Items
1	ABCDFG
2	CDEFH
3	ABCDFG
4	ACEF
5	ABCDEFG
6	BDEGH

Table 2: Diffset of all items

Item	Diffset
A	2 6
B	2 4
C	6
D	4
F	6
G	2 4
E	1 3
H	1 3 4 5

Table 3: Diffset of frequent items

Frequent Item	Diffset
A	2 6
B	2 4
C	6
D	4
F	6
G	2 4
E	1 3

1. Scan the source data and convert it to its diffset vertical representation Table 2. Only the representation of the frequent items will be stored in the memory. Here in the example, item H is infrequent so it is removed Table 3.
2. Generate all frequent 2-itemsets for all frequent items and construct the basic groups Fig. 4. For each frequent item, when a frequent 2-itemset is generated we search for a group that has a header as its diffset if such group is found then the frequent 2-itemsets is merged with the itemset in the group (ex. AD, AG) otherwise we build a new group for this itemset (ex. AC).

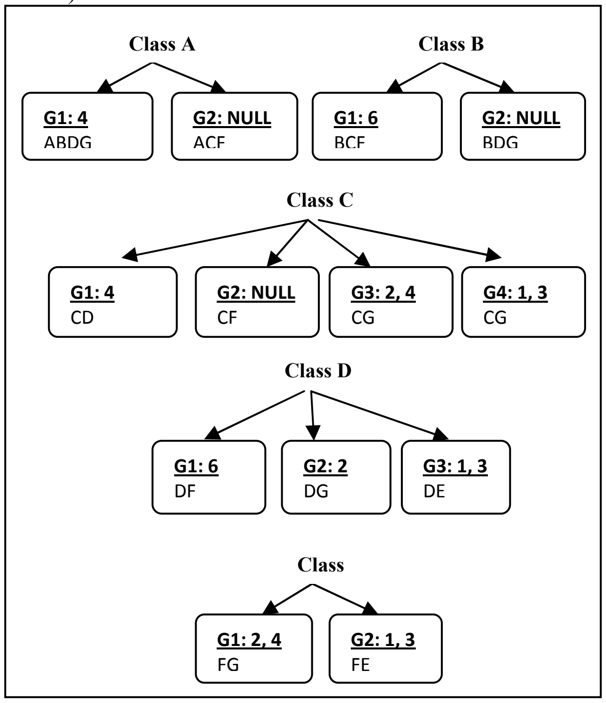

Figure 4: Basic groups of all frequent items

3. After generating all the basic groups, start the union process between them to get the class groups according to the validity condition. For example when trying to union the two basic groups of class A we will have T – C – TI = 6- 2- 1=3 which

is a valid value (as the min_sup is 3). So, the two groups can form one class group. However, by applying the validity condition on the two groups of class F it will result in a value 6 – 1- 4=1 which is less than the min_sup. Thus the two basic groups cannot be union in one group. Fig. 5 shows all the class groups of all classes.

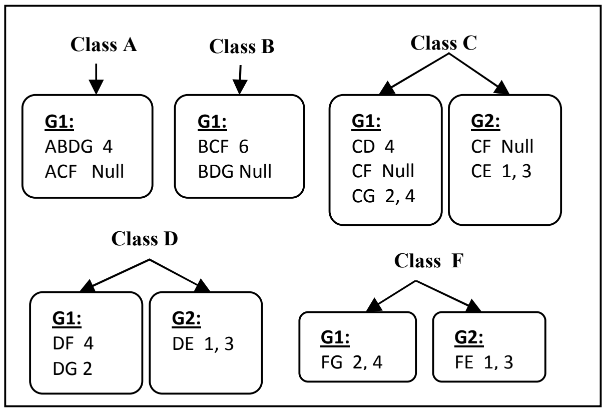

Figure 5: Class groups of all frequent items

4. After generating all the class groups for each frequent item, we start the phase of extracting frequent k-itemsets. In such phase, we don't need to refer to the original data set. The calculation of the itemsets support is done according to Rule 1 and Rule 2 as previously described. Figure 6 (a, b, c, d, e) shows the generated frequent k-itemsets of each class with their corresponding support and shows at which step of the mining process this itemset was generated.

Itemset	Sup	Phase	Itemset	Sup	Phase
AB	3	1	ABC	3	2
AD	3	1	ADC	3	2
AG	3	1	AGC	3	2
ABD	3	1	ABDC	3	2
ABG	3	1	ABGC	3	2
ADG	3	1	ADGC	3	2
ABDG	3	1	ABDGC	3	2
AC	4	1	ABF	4	2
AF	4	1	ADF	4	2
ACF	4	1	AGF	4	2
ABDF	3	2	ABGF	3	2
AGC	3	2	ADGF	3	2
ABDGF	3	2	ABDGCF	3	2

(a)

Itemset	Sup	Phase	Itemset	Sup	Phase
BC	3	1	BCFD	3	2
BF	3	1	BCFG	3	2
BCF	3	1	BCFDG	3	2
BD	4	1			
BG	4	1			
BDG	4	1			
BCD	3	1			
BCG	3	1			
BFD	3	1			
BFG	3	1			

(b)

4. Experimental results

In this section we measure the performance of the proposed VMUDG algorithm and also compare its performance of with the well-known vertical mining algorithm dEclat [12].

Itemset	Sup	Phase
CD	4	1
CF	5	1
CG	3	1
CDF	4	2
CDG	3	2
CFG	3	2
CDFG	3	2
CE	2	1

(c)

Itemset	Sup	Phase
DF	4	1
DG	4	1
DFG	3	2
DE	3	1

(d)

Itemset	Sup	Phase
FG	3	1
FE	3	1

(e)

Figure 6. Class_groups_construction procedure

The choice of the dEclat algorithm for the comparison purpose is done for the following reason. As mentioned before, the diffset data representation is more efficient and scalable than the tidset data representation. In the related work there are three algorithms that use the diffset. First, FDVM algorithm that requires the existence of the data set until finishing the mining process for checking the frequency of hidden pattern which is impractical when being used in real data sets. Second, the DTm algorithm which is a variation of the TM algorithm that uses the diffset. Experiments of this algorithm show that it is worse than the TM algorithm itself and almost has a performance like the dEclat algorithm due to the complex methods used in calculating the support of frequent itemsets. Third, the dEclat algorithm which has been proved experimentally outperforms other algorithms like FP-growth [12]. For that we found that comparison with the dEclat algorithm is most suitable. Datasets used in the experiments are downloaded from [14]. These data sets are often used in the study of frequent patterns mining. The characteristics of these datasets are shown in Table 4. All programs are implemented and compiled with Microsoft Visual C# Net 2005. All experiments are performed on a DELL 2GHz Core 2 Due laptop with 2G of memory, running Windows Vista. The accumulated time is measured from the beginning of reading the data set and converting it to its diffset representation to the end of the frequent pattern mining process.

Table 4: Data sets characteristics

Data	No. items	No. transactions
Mushroom	120	8124
Chess	75	3196
T25I10D10K	1000	9219
Pumsb	2113	49046

In Figure 7 and Figure 8, one can see that the execution time of our algorithm is better than dEclat in general. However, when the minimum support is lower, the performance of our method gets better than dEclat. This is because the number of frequent itemsets is very big when the minimum support is very low, dEclat needs more time to compare a lot of itemsets for frequency checking. Since VMUDG does not need to check every itemset's frequency, it can get better performance in lower minimum supports.

Figure 9 shows execution times used by the two algorithms for varying minimal support thresholds in mining T25I10D10K data set. As can be seen, VMUDG uses around half of the time that is used by dEclat. In mining Pumsb data set shown in

Figure 10, there are a lot of frequent patterns have the same diffset. The algorithm gets benefit of such nature when constructing the groups and the process was then only a simple extraction of the frequent k-itemsets from the groups. Utilizing such feature reduce the processing time of the VMUDG rather than the dEclat significantly.

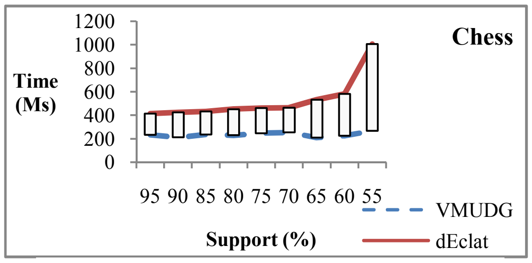

Figure 7. Run time for Chess data

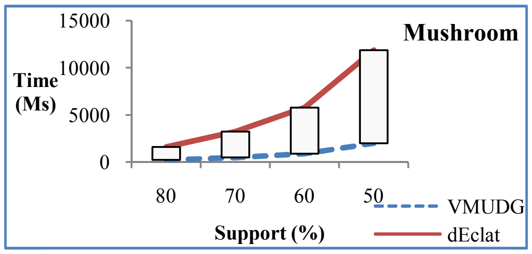

Figure 8. Run time for Mushroom data

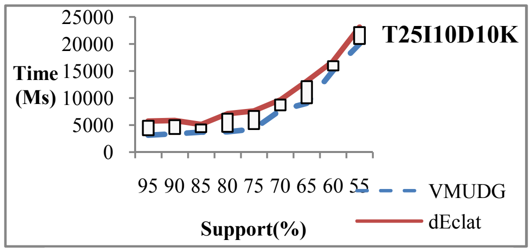

Figure 9. Run time for T25I10D10K data

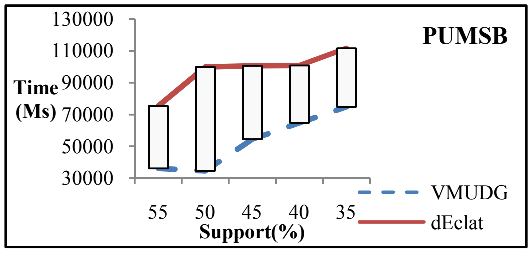

Figure 10. Run time for Pumsb data

5. Conclusion

In this paper, a new algorithm VMUDG (Vertical Mining Using Diffset Groups) for vertical mining of frequent itemsets is proposed. The proposed algorithm uses the diffset in classifying the frequent 2-itemsets into separate groups. By using these groups we can localize the generation of frequent k-itemsets achieving efficient processing and ensure bounded memory usage in the mining process. The proposed algorithm uses the diffset vertical data representation to generate only the frequent 2-itemsets. Itemsets of other lengths are extracted directly from the developed groups. Also, as a result of the independency between the groups in any class, we only need to keep in memory the currently processed group and thus support parallel mining. In addition, the algorithm does not require checking the itemsets frequency conditions for all itemsets, so, we only need to calculate its support through simple calculation processes. A comparison study between the proposed VMUDG algorithm and the dEclat algorithm has been conducted and shown that it in addition to its distinct features the VMUDG algorithm outperforms the dEclat algorithm.

References

[1] C. Zhu, X. Zhang, J. Sun, and B. Huang ,"Algorithm for Mining Sequential Pattern in Time Series Data", International Conference on Communications and Mobile Computing, pp. 258-262, January 2009

[2] H. Kriegel, P. Kroger and A. Zimek, "Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering", ACM Transactions on Knowledge Discovery from Data (TKDD), Vol.3, No.1, march 2009

[3] A.Ceglar and J. Roddick, "Association Mining", In ACM Computing Surveys, Vol.38, No.2, Article no. 5, July 2006.

[4] Y. Koh, N. Rountree, R. O'Keefe, "Mining Interesting Imperfectly Sporadic Rules", Knowledge and Information Systems, Vol. 14, No. 2, pp: 179-196, January 2008

[5] Z. Zheng, R. Kohavi, and L. Mason, "Real world performance of association rule algorithms", In ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), pp: 401-406, 2001.

[6] J. Han, H. Cheng, D. Xin, and X. Yan, "Frequent Pattern Mining: Current Status and Future Directions", Data Mining and Knowledge Discovery, Vol.15, No.1, pp. 55-86, 2007.

[7] J. Han, J. Pei, Y. Yin and R. Mao, "Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach", Data Mining and Knowledge Discovery, Vol.8, No.1, pp. 53-87, 2004.

[8] M. Zaki, "Scalable Algorithms for Association Mining", IEEE Transactions on Knowledge and Data Engineering, Vol.12, No.3, pp. 372-390, May-June 2000.

[9] P. Shenoy, J. Haritsa, S. Sudarshan, G. Bhalotia, M. Bawa and D. Shah, "Turbo-Charging Vertical Mining of Large Databases", In ACM Special Interest Group on Management of Data (SIGMOD), Vol.29, No.2, June 2000

[10] M. Zaki and K. Gouda, "Fast Vertical Mining Using Diffsets", In Knowledge Discovery and Data Mining (KDD), pp. 326-335, 2003.

[11] W. Consue, and W. Kurutach, "Novel Vertical Mining on Diffsets Structure", In Proceedings of the IEEE/WIC International Conference on Intelligent Agent Technology (IAT), pp. 343-349, October 2003.

[12] B. Goethals, "Memory Issues in Frequent Itemset Mining", In Proceedings of the ACM Symposium on Applied Computing (SAC), pp.530 - 534, March 2004.

[13] M. Song, S. Rajasekaran, "A Transaction Mapping Algorithm for Frequent Itemsets Mining" , IEEE Transactions on Knowledge and Data Engineering , Vol.18, No.4, pp. 472-481, April 2006.

[14] http://fimi.cs.helsinki.fi/testdata.html