Novel Row Enumeration Approach of Graph-Based Frequent Itemsets Mining

Mohammad-Arsyad Mohd-Yakop, Shuzlina Abdul-Rahman, and Sofianita Mutalib

***Abstract*—A lot of algorithms performing Frequent Itemsets Mining (FIM), however, some of the glitches in the algorithms still require attention, particularly when the mining process involves a high dimensional dataset. The Directed Acyclic Graph in High Dimensional Dataset Mining (DAGHDDM) is a graph-based mining algorithm that represents itemsets in the complete graph before FIM takes place. Nevertheless, the construction of complete graph creates unnecessary edges and makes the search space large and affects the overall algorithm performance. This research aims to speed up the searching process by creating relevant edges in the graph to reduce the search space by rearranging the items using the common prefix rowset. We proposed a novel frequent itemsets mining using row enumeration approach on graph based structure called Frequent Row Graph Closed (FRG-Closed). Designing the FRG-Closed involves new data structure creation known as Frequent Row Graph (FR-Graph). We performed the experiments to compare the performance of FRG-Closed with DAGHDDM algorithm. The result of the experiments revealed the FRG-Closed capability to mine the frequent closed itemsets faster than its counterpart, DAGHDDM algorithm. Moreover, the FRG-Closed is also able to handle lower minimum support compared to the DAGHDDM for a larger transaction.**

***Index Terms*—Data mining, graph theory, high dimensional, frequent itemset.**

I. Introduction

Frequent Itemset Mining (FIM) was first applied in customer transaction database. It is also known as market basket analysis, where the results can be used by the retailers for marketing and promotions [1]. FIM involves a critical process in searching for itemset of frequent itemsets, in which its frequency is not less than the minimum support value [2]. The discovered frequent itemsets are significant to represent strong associations between items in the database. However, the amount of data to be processed in memory would affect the searching process of FIM. Nowadays, recorded dataset has expanded dramatically since data are captured in various types and ways. This phenomenon introduced the 'Big Data' era involving an enormous data collection based on 5V's, which are Volume, Velocity, Variety, Veracity and Value [3]-[5]. Volume refers to the vast amount of data generated every second while Velocity refers to the speed at which new data is generated at a time. Giving example the number of transactions happened in each minute or the number of status updated on social media every hour. Variety refers to the different types of data that are available today such as video, sound, image, sensor, etc. while Veracity refers to the quality of the available data. Lastly, Value refers to how valuable the available data is. These massive volume of data, as a result, posed a problem in analyzing them for understanding. Furthermore, the analysis of large amount of data could be carried out for diverse purposes requiring a variety of tasks.

This paper presents an algorithm that can be applied in high dimensional large datasets; i.e. large number of items and large number of records (rows), reflecting the first "V" which is the data volume. Based on the previous study [6], the high dimensional data is related to the data that have a large number of dimension/item and a small number of records. Examples of the high dimensional data are microarrays, time series data in financial and neuroimaging. There are two ways of enumerating itemsets in searching for the frequent itemsets, namely column enumeration and row enumeration. The column enumeration computes the itemsets based on the variables or items contained in the dataset. As for the row enumeration, the searching process is done by enumerating the itemsets through the transactions or records contained in the dataset.

Past researchers have manipulated data representation structure, such as array, hash, tree or graph representation to optimize the efficiency of the mining which in turn may overcome the data volume problem. The way the dataset is represented is by implementing the selected data structure that is capable to optimize the mining process. Based on literatures, the compression of dataset using a graph representation has shown its success as it consumes less memory compared to the FP-tree [7]. However, the two main questions in pursuing this research are:

- How to improve the representation of transaction database based on existing directed acyclic graph structure?
- How to mine the frequent closed itemsets with the improved representation?

Our work is based on Dong *et al.*, [2] that introduced a graph-based algorithm called Directed Acyclic Graph on High Dimensional Dataset Mining (DAGHDDM). The DAGHDDM uses Directed Acyclic Graph (DAG) theory as a new dataset structure and applies the row enumeration strategy whereby each vertex is represented as a row instead of items. This makes the searching space smaller. However, DAGHDDM mines on a complete graph that is created before the mining process. In this research, we attempt to improve the existing graph representation and adopt row enumeration strategy by creating only the necessary edges using the graph representation so that the unnecessary edges path between two nodes can be removed. The rest of this paper is organized as follows: In Section II, we describe the preliminaries and the novel graph components. In Section III,

Manuscript received May 6, 2018; revised July 1, 2018.

Shuzlina Abdul-Rahman is with the Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA, 40450 Shah Alam, Selangor, Malaysia (email: shuzlina@tmsk.uitm.edu.my).

doi: 10.18178/ijmlc.2018.8.4.706

we present the FRG-closed algorithm followed by the experimental results and analysis in Section IV. Finally, we conclude this paper in Section V.

II. Preliminaries and Novel Graph Components

This section presents the basic definitions and the novel graph components required for further understanding of the concepts. The preliminaries include the general definition of transposed table, closed itemset and frequent itemset. The process of creating the vertical dataset is explained next and followed by the explanation on the novel graph data structure components.

A. Definition

Definition 1 (transposed table)

A dataset T consists of set of rows and set of items denoted as $R = \{r_1, r_2, r_3, \dots, r_k\}$ and $I=\{i_1, i_2, i_3, \dots, i_n\}$ respectively as shown in Table I. TT is denoted as the transposed table which is composed of n number of rows, where each of it is identified by an item, $i_m \in I$ as shown in Table II.

TABLE I: Dataset T

Row ID	Item
1	*a, c, f, m, p*
2	*a, b, c, f, l, m, o*
3	*b, f, o*
4	*b, c, p*
5	*a, c, f, l, m, p*

TABLE II: Transposed Table tt of T

Row ID	Item
a	*1, 2, 5*
b	*2, 3, 4*
c	*1, 2, 4, 5*
f	*1, 2, 3, 5*
l	*2, 5*
m	*1, 2, 5*
o	*2, 3*
p	*1, 4, 5*

Definition 2 (closed itemset and closed rowset)

Each closed itemset, *I*, has a related closed rowset, *RS*, where it contains all items in itemset *I* and vice versa. Itemset *S* is called "closed" when it has no superset *S'* that has the same support value (sup), $\sup(S) \neq \sup(S')$. Similarly, to the rows, where *RS* is called "closed" when it has no super row that has the same itemset, item (*RS*) ≠ item (*RS'*).

Definition 3 (frequent itemset and large rowset)

An itemset is called "frequent" if the support is equal or greater than the threshold (minimum_support), $\sup(S) \geq$ minimum_support. Rowset is called "large" if the number of rows in a rowset is greater than the threshold, size(*RS*) ≥ minimum_support. The frequent closed itemset is determined when the itemset is "frequent" and "closed". When mining frequent closed itemsets using row enumeration strategy, the closed itemset are found by searching the closed rowset based on Definition 2. The discovered closed itemset then is verified to determine whether it is a large rowset or not. Based on Definition 3, if the rowset is "large", then the itemset for the rowset is "frequent". Therefore, frequent closed itemsets are discovered by searching the rowset that is "closed" and "large".

B. Creating Vertical Dataset

In order to find frequent closed itemsets and large rowsets, the dataset firstly is being transposed and later the vertical table is created. The creation of the vertical dataset is vital because the item and its rowset would be used in the construction of the FR-Graph. Each row in the vertical dataset consists of item and rowset, that refer to the set of row with the item. Table III shows the input dataset D and given the minimum support equals to 2. The first step was counting the distinct items in dataset D. In Table IV, all items in dataset D were listed together with their frequency. After the items frequency was counted, the next step was to remove the infrequent items whenever the item frequency was lower than the given threshold as shown in Table V. By performing this process, the left itemset in the dataset was left with only the frequent 1-itemset.

TABLE III: Input Dataset T

Row ID	Item
1	*a, c, d, f, i, g, m, p*
2	*a, b, c, f, l, m, o*
3	*b, f, j, h, o*
4	*b, c, k, p, s*
5	*a, c, e, f, l, m, n, p*

TABLE IV: Items and Their Frequency from Dataset D

Item	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*	*k*	*l*	*m*	*n*	*o*	*p*
Frequency	3	3	4	1	1	4	1	1	1	1	1	2	3	1	2	3

TABLE V: Items and Their Frequency after Removing Infrequent Item

Item	*c*	*f*	*a*	*b*	*p*	*m*	*l*	*o*
Frequency	4	4	3	3	3	3	2	2

TABLE VI: Vertical Dataset from Dataset

Row ID	Item	Sorted Rowset
a	1, 2, 5	2, 5, 1
b	2, 3, 4	2, 3, 4
c	1, 2, 4, 5	2, 5, 1, 4
f	1, 2, 3, 5	2, 5, 1, 3
l	2, 5	2, 5
m	1, 2, 5	2, 5, 1
o	2, 3	2, 3
p	1, 4, 5	5, 1, 4

TABLE VII: RowID Header Table

Row	2	5	1	3	4
Number of item	7	6	5	3	3

After the removal process of the infrequent items was performed, the creation of the vertical dataset was initiated as shown in Table VI. The frequent 1-itemset was listed in the first column in a vertical dataset. In the second column, the rowset of the item was inserted. The rowset is the list of rows where the item is available. For example, if rowset of item a is {1,2,5}, item a exists at row 1, 2, and 5 in the dataset. The third column is the sorted rowset by refering to the number of items in each row, and these numbers are recorded as RowIDHeader in Table VII. By sorting the itemset according to the edges, the creation of the graph construction process can be minimized. This concept was inspired by the FP-Tree data structure which used the common prefix of the itemset to rearrange the item. Instead of using the itemset, the FR-Graph used the rowset as the common prefix.

C. Constructing the Frequent Row Graph

After the vertical dataset was created, the novel graph-based data structure called Frequent Row Graph (FR-Graph) was constructed. In the following sections, the components of the FR-Graph and its attributes are explained first, followed by how the FR-Graph was constructed from the vertical dataset and lastly, the property of the graph.

FR-Graph Components

The graph representation would organize the items in the dataset and benefit the searching process to be more efficient. The FR-Graph consists of three components as defined below:

i. Class FRGEdge
   - Integer *source*
   - Integer *destination*
   - Set<Integer> *items*

ii. Class FRGNode
   - Integer *row_id*
   - List<FRGEdge> *edges*

iii. List<Integer> *RowIDHeader*

Firstly, FRGEdge is a class that stores the information of the dataset. It consists of three elements: *source*, *destination*, and *items*. The first and second elements are the *source* and the *destination* which indicates the direction of the edge from what node to what node it is pointed to. The third element is the variable *items* that is used to store the item that is available in both nodes of source and destination.

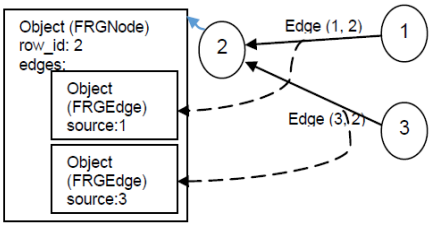
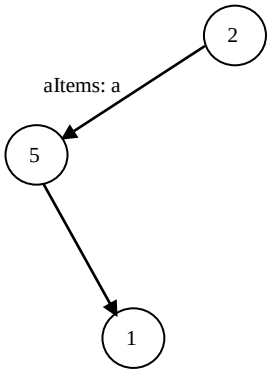
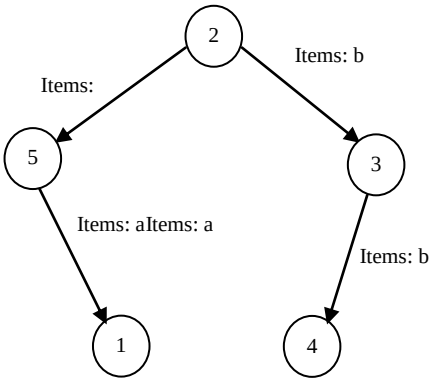


Fig. 1. The FRGNode illustration.

Next, the second component, FRGNode is a class that represents the node in the graph that consists of two elements: *row_id* and *edges*. The *row_id* is a variable that holds the value of *rowID*. The *edges* element is a list of FRGEdge object. Unlike the usual implementation of the directed graph, the edges held by the corresponding node is the edge that points to that node. For example, the list of edges in element *edge* for node 2 (*row_id* = 2) includes edge(1, 2) and edge(3, 2), as in Fig. 1. The third component in the FR-Graph is the *RowIDHeader* where it is a list of *rowID* and it is referred to to sort the rows for vertical table.



(a) After item a insertion



(b) After item b insertion

Fig. 2. Constructed FR-Graph after item a and b insertion.

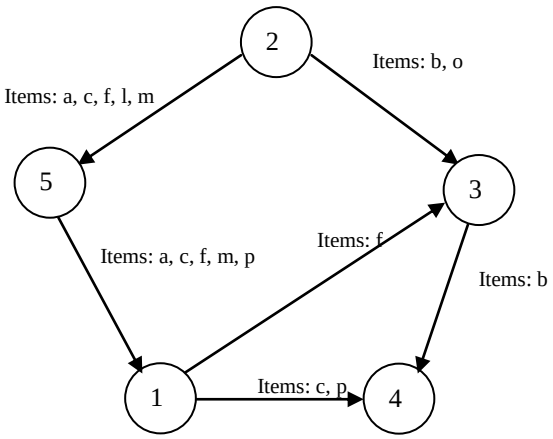


Fig. 3. Complete FR-Graph.

FR-Construction Process

Once the dataset has been transposed, the graph structure was used to represent the dataset. Firstly, the enumeration of all the items was done in the vertical dataset. To visualize the algorithm process, Fig. 2 shows the insertion of item *a* with rowset {2, 5, 1} and item *b* with rowset {2, 3, 4} in the FR-Graph. Three nodes are created and linked with the two edges as shown as an item *a* is inserted in the graph as illustrated in Fig. 2(a). Then, for item *b*, another two nodes and two edges are created as illustrated in Fig. 2(b). After all items in Table III are inserted, the graph construction is completed. Fig. 3 shows the illustrated FR-Graph.

Property of FR-Graph

Three *properties* of FR-Graph can be observed from the construction process:

Property 1: The number of nodes equals or less to the number of rows at most

Proof: In the FR-Graph, each node represents a row and the number of *nodes* equals to the number of rows in the dataset. Assuming that, the dataset has several infrequent items, if these infrequent items are deleted, there may exist rows that contain only these infrequent items, which will also be deleted. In this case, the number of the nodes is less than the number of the rows.

Property 2: Only necessary edges are created

Proof: The reason why rowset of each item is sorted based on RowIDHeader is to reduce the number of edges creation. By *sorting* the rowset, the common rowID is shared among all items. This will compress the graph by sharing the common prefix rowset.

Property 3: The FR-Graph contains the complete information about the dataset with fewer edges created

Proof: Based on the construction of FR-Graph, each item in a dataset is inserted into the edges. These items have met the condition as frequent items. The edge shows the existing items between the nodes. For example, edge(2, 5) contains items a, c, f, l, and m which mean all these items are available in rowID 2 and 5. Moreover, only the relevant edge is created during the construction process. This means, the pruning process has been done prior to the creation of FR-Graph.

III. Frequent Row Graph-Closed Algorithm

This section discusses a novel strategy to improve the searching process of frequent itemsets by adopting a row enumeration approach on graph-based representation. The proposed algorithm was based on the DAGHDDM algorithm [2] that adopts graph-based dataset representation with row enumeration strategy. We name the algorithm Frequent Row Graph-Closed or FRG-Closed. We begin by explaining the two theorems namely, the path retrieving and the item merging.

A. Theorem 1 (Path Retrieving)

The constructed graph (FR-Graph) allows the retrieval of the large rowset by traversing the nodes in the reverse direction.

Proof: To prove theorem 1, a new method called getPaths was created. Given an example based on Fig. 3, each node was named as *nX*, where n is the node and X indicates the rowID. All possible paths that can be travelled from each node is [*n*4] = {{*n*4, *n*1, *n*5, *n*2}, {*n*4, *n*3, *n*1, *n*5, *n*2}, {*n*4, *n*3, *n*2}}, [*n*3] = {{*n*3, *n*1, *n*5, *n*2}; {*n*3, *n*2}}, [*n*1] = {{*n*1, *n*5, *n*2}}, [*n*5] = {{*n*5, *n*2}}, [*n*2] = {{*n*2}}.

To get the possible largest path, depth first search manner was performed. For example, *n*4 was identified as the starting node. First, these paths were initialized as null. This *path_list* is meant to store the list of paths that were found while traversing the graph. This process was initiated at node *n*4 and then traversed to node n1 with itemset {*c*, *p*}.

As the first move, the itemset {*c*, *p*} with path {*n*4, *n*1} was inserted into path_list with the support of 2. Before it travels to the next node which is *n*5, the intersection between items on the edge(5, 1), {*c*, *f*, *a*, *m*, *p*} and current itemset, {*c*, *p*} was performed, and resulted in this itemset {*c*, *p*}. If the intersection itemset was not empty, then the process proceeded to the node *n*5. Next, at node *n*5, the itemset in the *path_list* was checked again. The resulting itemset {*c*, *p*} produced a new path, which was {*n*4, *n*1, *n*5} with the support value of 3. Since the current itemset was contained in the *path_list*, so the *path_list* was updated from {*n*4, *n*1} to {*n*4, *n*1, *n*5} and the support value from 2 to 3. After updating the path and the support value, the algorithm would make a check to see where the preceeding node traverse to. From node *n5*, edge(2, 5) was the only edge found in *n*5. Again the intersection was performed with the current itemset and edge(2, 5).items, similarly to {*c*, *f*, *a*, *m*, *l*} ∩ {*c*, *p*} = {*c*}. The searching process was terminated here since there was no edge that pointed to node *n*2 or the intersection item of the edge and the current itemset was empty. The *getPaths* function returns the pair of rowset and itemset list. In the end, the paths retrieved from node *n*4 is [*n*4] = {{*n*4, *n*1, *n*5, *n*2} : {*c*}, {*n*4, *n*1, *n5*} : {*c*, *p*}, {*n*4, *n*3, *n*2} : {*b*}}. Note that the path {*n*4, *n*3, *n*1, *n*5, *n*2} was not included in the paths list because there was no itemset produced by the intersection between items on edge(3, 4), {f} and items on edge(1, 3), {b}. The rowset returned from the *getPaths* method did not have any complete itemset.

Fig. 4 shows the algorithm of the *getPaths* method. First, *paths* list was created to store the list of pair itemset and path found (Fig. 4: *getPaths* function, line:1). At line 4, the *getPaths* function will enumerate all edges that link to the startNode. The empty list called *path* is created to store the traversed node and the first node (*startNode*) was inserted (Fig. 4: *getPaths* function, line:3-4). For *itemset*, the items in the enumerated edges is stored in the list just like *path* (Fig. 4: *getPaths* function, line:5). Then, the *getPathsSub* function is called with updated *itemset*, *path* and *sourceNode* as the input (Figure 4: *getPaths* function, line:6). In the *getPathsSub* function, the input *itemset* is checked whether it is exist in the *paths* list or not. If not exist, insert into *paths* list, else, only insert if the *rowset* size in the *paths* list is smaller than *path* size +1 (Fig. 4: *getPathsSub* function, line:1-6). Then for each edge in the *sourceNode*, if the intersection of the *item* in the *edge* and *itemset* is not empty, the *getPathsSub* is recursively called (Fig. 4: *getPathsSub* function, line:8-12). At the end, the paths list should contain the pair of itemset and path that start with specific node.

```
Input: FR-Graph G,startNode
Output: list of the largest rowset with its itemset
getPaths(G, 4)
Begin
 1.  Map<itemset, rowset> paths = null;
 2.  For each edge in startNode.edges
 3.          List path = null;
 4.          path.add(startNode);
 5.          itemset = edge.items;
 6.          getPathsSub(edge.source, itemset, path,
             paths);
 7.  //end foreach
 8.  return paths;
End

getPathsSub(currentNode,itemset, path, paths)
Begin
 1.  If itemset does not exist in paths.key
 2.          paths.put(itemset, path currentNode)
 3.  Else
 4.          If (path ∪ currentNode).size > rowset of
             the itemset in paths
 5.                  paths.put(itemset, path ∪
                     currentNode)
 6.          //end if
 7.  //end if
 8.  For each edge in currentNode.edges
 9.          If itemset ∩ edge.items ≠ ∅
10.                 getPathsSub(edge.source,
                    itemset ∩ edge.items, path,
                    paths);
11.         //end if
12. //end foreach
```

Fig. 4. The getpaths method.

B. Theorem 2 (Item-Merging)

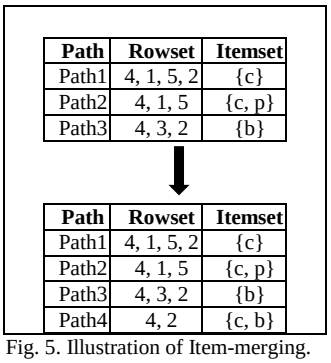
Since the *getPaths* method retrieved all the possible large rowsets, the closed sub-rowset (rowset.size-1) could be established by performing an intersection of two large rowsets. Since each node was represented as a row, the path in the graph was considered as a rowset.

Proof: Since the *getPaths* method would return the largest rowset with its intersection itemset, further checking was

required to ensure that the rowset had the complete itemset. Closed sub-rowset can be obtained by following these two rules:

Rule1: The size of the rowset must be compared to a larger or an equal size of rowset.

Rule 2: Performing an intersection between two rowsets and storing both *items from two rowsets as the new intersection rowset.*



Path	Rowset	Itemset
Path1	4, 1, 5, 2	{c}
Path2	4, 1, 5	{c, p}
Path3	4, 3, 2	{b}

Path	Rowset	Itemset
Path1	4, 1, 5, 2	{c}
Path2	4, 1, 5	{c, p}
Path3	4, 3, 2	{b}
Path4	4, 2	{c, b}

Fig. 5. Illustration of Item-merging.

A notation of <*RS* : *I*> is assumed as a set of path where *RS* is the rowset and *I* is the itemset for the *RS*. For example, <{4, 1, 5, 2} : {c}>, <{4, 1, 5} : {*c*, *p*}>, <{4, 3, 2} : {*b*}> is *path1*, *path2* and *path3* respectively. *path1* would not be checked since it has the largest rowset in the list. *path3* would be examined to have a better view on how the *item-merging* works. The checking process can be initialized by comparing *path3* with *path1*. Since *path3.rowset* $\not\subset$ *path1.rowset*, the AND operation has to be performed to get an intersection between these two paths. The intersection of *path3.rowset* and *path1.rowset* is {4, 3, 2} $\cap$ {4, 1, 5, 2} = {4, 2}. Now, the new path, namely *path4*, is created with the rowset {4, 2}. To get the itemset for *path4*, the UNION operation was performed. Since *path3* and *path1* itemse in the new *path4* with rowset {4, 2} and itemset {*b*, *c*}. Fig. 5 illustrates the paths before and after the item-merging process.

IV. Experimental Results and Analysis

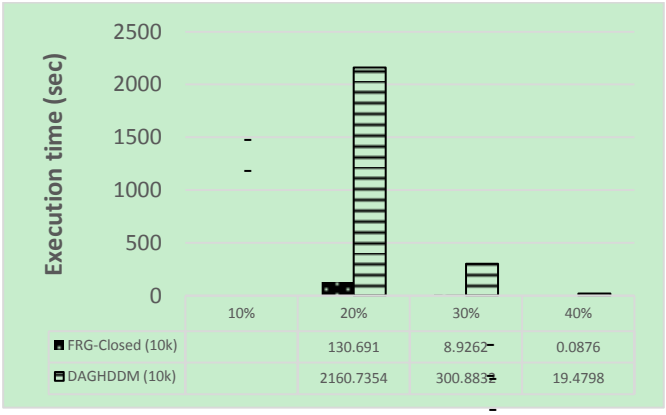
In this experiment, seven synthetic datasets were prepared and two experiments were conducted. The datasets were generated using the SPMF library [4]. To generate the dataset, function *generateDatabase* was called requiring three input parameters: number of transactions, number of dimensions and maximum number of item per transaction. As the result, transaction dataset with predetermined numerical value was generated. In the generated dataset, the numerical value represented an item. This generated dataset was named D#T# in which D# referred to the number of dimensions which is the number of distinct items in the dataset, and T# is the number of transactions. These synthetic datasets were generated based on the pre-defined parameter values as shown in Table IX.

The number of dimensions and transactions generated was based on the previous studies [8], [10], [11] as the benchmark to determine the range value of dimension and transaction as high dimensional dataset. The number of transactions in the high dimensional dataset is between 40 to 80 transactions while the dimension is starting from 4,000 to 100,000 dimensions. Two sets of experiment were conducted to examine the algorithm performance based on different dimensionality of the datasets and different number of transactions. The results are discussed based on the execution time at different values of minimum support which are 10%, 20%, 30%, and 40%. The dash "-" value in the result is when the algorithms were not able to finish the mining process in 3 hours or hit the not-enough-memory error.

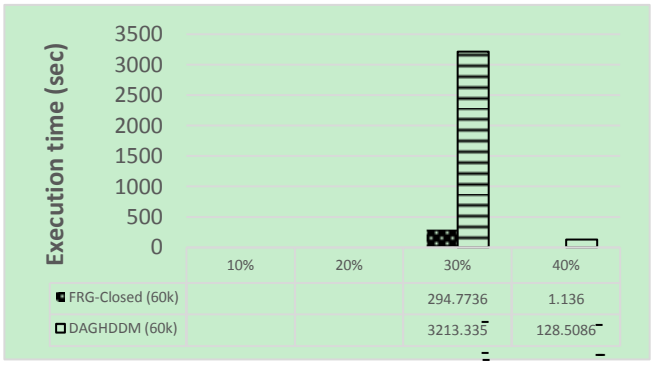
TABLE IX: Dataset Used in the Experiments

Dataset	Dimension	Transaction
D10kT50	10,000	50
D60kT50	60,000	50
D100kT50	100,000	50
D4000T40	4000	40
D4000T50	4000	50
D4000T60	4000	60
D4000T80	4000	80

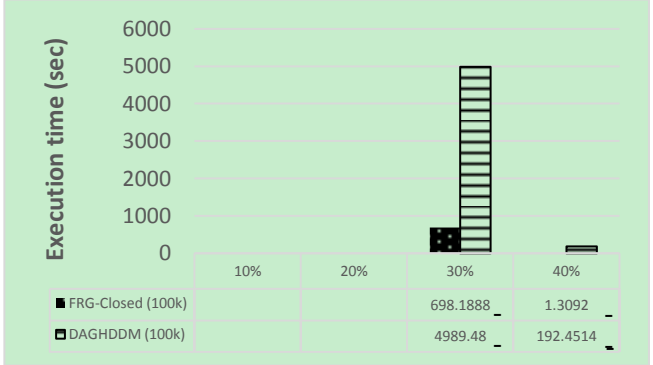
A. Experiment I (Dimensionality) Result



(a) D10kT50



(b) D60kT50



(c) D100kT50

Fig. 6. Result of experiment I at different number of dimensions.

Experiment I was conducted to evaluate the performance of FRG-Closed by comparing to the DAGHDDM, as the most similar existing algorithm with the respect to different values of dimension. There are three different dimensions: 10000, 60000, and 100000 dimensions. The execution time and the memory usage were observed throughout the experiment. The results of Experiment I showed that FRG-Closed was generally faster than DAGHDDM. This experiment further showed that both the FRG-Closed and DAGHDDM algorithms could not complete the mining process with the minimum support of 10% for the dataset D10kT50 and minimum support of 20% for datasets D60kT50 and D100kT50 after 3 hours of execution. Fig. 6 shows the result on execution time taken to complete the mining process.

Since both algorithms could complete the mining process at the minimum support of 30%, the graph was plotted to show the effect of dataset dimensions on both algorithms as shown in Fig. 7. This trend shows that the increment of dimension does affect the execution time. The DAGHDDM algorithm showed an increment in the execution time from 300 seconds at 10,000 dimensions to 4,989 seconds at 100,000 dimensions. There was a dramatic increment of the execution time for the DAGHDDM algorithm when the dataset dimensions became larger. For the FRG-Closed, execution time began with 9 seconds at 10,000 dimensions up to 698 seconds execution time at 100,000 dimensions. In contrary to the DAGHDDM, the FRG-Closed showed a slight execution time increment when the dataset dimensions grew.

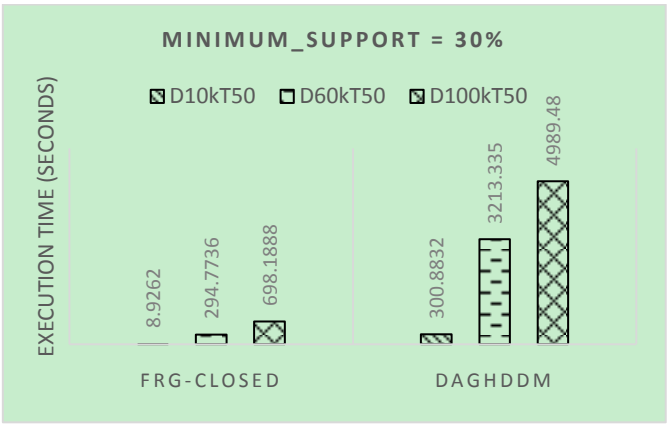
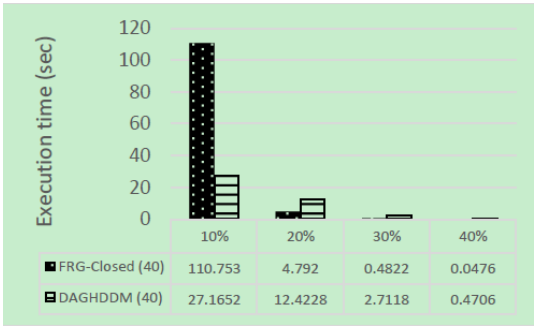


Fig. 7. Result on dimensionality effect at minimum support 30%.

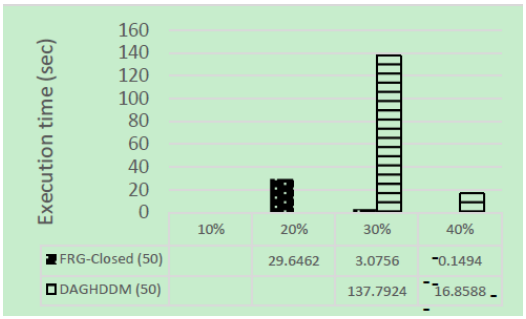
The recorded increment of time indicates that the FRG-Closed algorithm is capable of processing faster than the DAGHDDM algorithm. In general, FRG-Closed has recorded more than 90% reduction of execution time as compared to DAGHDDM. The dataset D100kT50, at the minimum support of 40%, recorded the highest difference in the percentage of 99.48%. It proved the proposed algorithm FRG-Closed has successfully reduced the execution time in the range of 95% average.

B. Experiment II (Transactions) Result

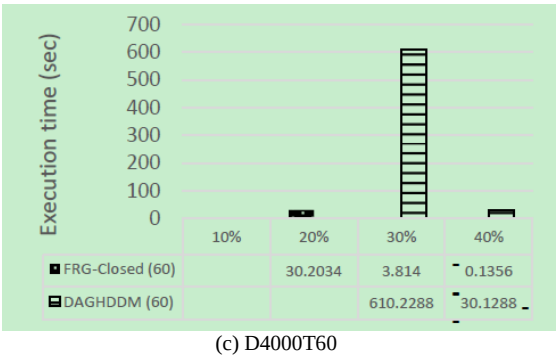
Fig. 8 exhibits the results of Experiment II for algorithm testing on four datasets with different number of transactions and similar number of dimensions. The dataset D4000T40 was the only dataset that can be mined at the minimum support of 10% for both algorithms. The result showed that the decrease of minimum support value caused a longer execution time. The FRG-Closed took a longer time compared to the DAGHDDM algorithm at the minimum support of 10% and this reduced dramatically at 20% of minimum support. For dataset D4000T50 and D4000T60, the results appeared for both algorithms. There was not much difference in execution time recorded for both datasets at any minimum support value. However, when the transactions increased, D4000T80, clearly showed that the gap in both algorithms can be identified, as DAGHDDM was only able to mine at 40% of minimum support. This experiment revealed that the FRG-Closed can process the dataset with the number of transactions at 50 onwards, at the lowest minimum support of 20%. Prediction of quality is lower than the prediction given by the arithmetic mean of the dependent variable of the sample.



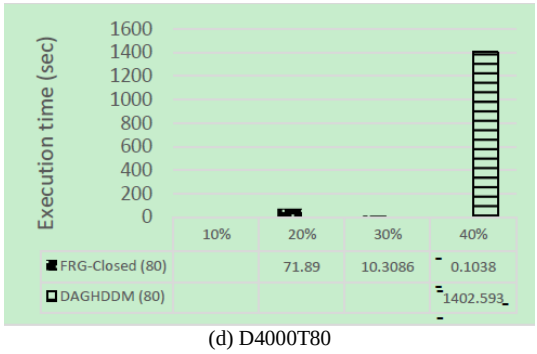
(a) D4000T40



(b) D4000T50



(c) D4000T60



(d) D4000T80

Fig. 8. Result of experiment II at different number of transactions.

As can be seen in this result, FRG-Closed has demonstrated the ability to reduce the execution time greatly. Beginning at 50 transactions onwards, FRG-Closed reached up to 99% reduction time. Meanwhile, for the lowest transaction number, D4000T40, at the minimum support of 10%, FRG-Closed recorded an increment of 300% in execution time compared to DAGHDDM algorithm. However, at the minimum support of 20% to 40%, it again showed a reduction in execution time up to 89%. Based on the presented results in the previous sections, several findings are highlighted as below:

Finding 1: For dataset D4000T40, the execution time for the FRG-Closed was much higher than the DAGHDDM at the minimum support of 10%. The high execution time was required in FRG-Closed as many item-merging processes were needed during the mining process. The item-merging process was performed after retrieving all paths that starts from each node. Since the item-merging process was done to compare each itemset with the rest of the paths list, it resulted in a burden when the retrieving paths were large. Generally, the more paths were retrieved for one node, the more item-merging processes were performed. Unlike the DAGHDDM algorithm, there was no item-merging process involved in the algorithm.

Finding 2: With lesser number of edges created in the FR-Graph data structure, it helps in mining efficiently even at a low minimum support, such as 10 – 20%, in larger transactions dataset. This was proven when FRG-Closed has shown its capability to mine the dataset that contained up to 80 transactions at the lowest minimum support of 20% even though it required more time. By having less edges, the traversing path in the graph was narrow and straightforward.

Finding 3: Even though the row enumeration strategy was applied in both graph-based algorithms (FRG-Closed and DAGHDDM), problem still occur to mine lower minimum support for many transactions. This is because many rows meant that there was more nodes creation in the graph. This leads to a high number of edges creation as well as the number of traversing paths.

V. Conclusion

Generally, this research contributes in constructing a compact graph representation for transaction database and proposes an efficient mining algorithm by adopting the depth first search method. The FRG-Closed algorithm has introduced a novel data structure called FR-Graph to arrange the dataset more efficiently. The evaluation in this research has shown that FRG-Closed has managed to mine faster than the DAGHDDM algorithm on high dimensional datasets. FRG-Closed employed a novel data structure representation by adopting a graph-based approach in which each node was represented in a row while edge represented the availability of the items between the two nodes. The order of rowset in the vertical dataset shared the common prefix rowset. This made the number of edges created in the graph less, but still having all the required information, itemsets and nodes.

Acknowledgment

The authors would like to thank to the Ministry of Higher Education, Malaysia for the research grant FRGS 156/2013 and also to Research Management Centre, Universiti Teknologi MARA, Selangor, Malaysia for the research support.

References

[1] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," *ACM Sigmod Record*, vol. 22, no. 2, pp. 207-216, 1993.
[2] J. Dong *et al*., "A DAG-based algorithm of mining frequent closed itemsets in high dimensional datasets," *International Journal of Advancements in Computing Technology*, vol. 4, no. 19, 2012.
[3] J. Fan, H. Fang, and L. Han, "Challenges of big data analysis," *National Science Review*, vol. 1, no. 2, pp. 293-314, 2014.
[4] P. Fournier-Viger, A. Gomariz, A. T. Soltani, C.-W. Wu, and V. S. Vincent, "SPMF: A Java open-source pattern mining library," *The Journal of Machine Learning Research*, vol. 15, no. 1, 3389-3393, 2014.
[5] D. Laney, "3D data management: Controlling data volume, velocity and variety," *META Group Research Note*, vol. 6, no. 70, 2001.
[6] H. D. K. Moonesinghe, *Graph Based Methods for Pattern Mining*, Michigan State University, 2007.
[7] M. J. Zaki, S, Parthasarathy, M. Ogihara, and W. Li, "New algorithms for fast discovery of association rules," *KDD*, vol. 97, pp. 283-286, 1997.
[8] P. F. Cong, G. Tung, A. K. Yang, and M. J. Zaki, "Finding closed patterns in long biological datasets," in *Proc. the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003, pp. 637-642.
[9] R. Vimieiro and P. Moscato, "Disclosed: An efficient depth-first, top-down algorithm for mining disjunctive closed itemsets in high-dimensional data," *Information Sciences*, vol. 280, pp. 171-187, 2014.
[10] S. Yin and O. Kaynak, "Big data for modern industry: Challenges and trends [point of view]," *IEEE*, vol. 103, no. 2, pp. 143-146, 2015.
[11] L. Yu and H. Liu, "Feature selection for high-dimensional data: A fast correlation-based filter solution," in *Proc. the 20th international Conference on Machine Learning*.

Mohammad Arsyad Mohd Yakop received his bachelor's degree in information technology (intelligent systems engineering) with honours from Universiti Teknologi MARA in 2013 and M.Sc in Computer Science from Universiti Teknologi MARA, Malaysia in 2017. He is currently work as an application engineer in banking industry. His research interest includes data analytics, semantic analysis and machine learning.

Shuzlina Abdul Rahman received her bachelor's degree in computer science from Universiti Sains Malaysia in 1996, master of science in information technology from Universiti Utara Malaysia in 2000 and PhD in science and system management from Universiti Kebangsaan Malaysia in 2012. She is currently working as an associate professor at the Faculty of Computer & Mathematical Sciences, Universiti Teknologi MARA, Malaysia. Her research interest includes computational intelligence, machine learning and data analytics & optimization.

Sofianita Mutalib is currently working as a senior lecturer in the Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA, Malaysia. She had received bachelor and master degrees from Universiti Kebangsaan Malaysia in 1998 and 1999. She is actively doing research in applied data mining and data analytics, for various area and different types of data. Her interest also has been shown through yearly publications in proceedings and journals.