Webology (ISSN: 1735-188X)
Volume 18, Number 6, 2021

Recurrent Sub-Graph Drilling Of The Invisible Items

Dr. B. Senthil Kumaran

Assistant Professor, PG & Research Department of Computer Science,
Jairams Arts & Science College, Karur -03. (Affiliated to Bharathidasan University, Trichirappalli-24)

Abstract

A major role after the completion of research area is drilling the graph sector and extortion of the invisible pattern from the graph is a difficult task and it should also provide shovel considerable patterns. It is a competing task and the glimpse of this paper focuses on uncovering useful patterns from the data of the graph with the help of novel algorithm which is named as " Recurrent Sub-Graph Drilling of the Invisible items - RSDI algorithm". It provoke easy mechanism to conceal the devastation of prolong running time and memory portion. The graphs are probably transferred into textual transformation datum therefore except the invisible items others are taken into and this is passed into binary representations to explore the recurrent sub-graphs. The final outcome is practically valued by the process of the art prevailing algorithms to exemplify the acquisition of the proposed algorithm.

Keyword: Graph mining – sub graph mining – data mining – binary representations - drilling – uncovering

INTRODUCTION

The avant-garde in graph mining field is frequent sub-graph mining (FSM). The ultimate target of FSM, to identify the repeated sub-graphs in the provided graph datum where the event organized is exceeded beyond the limited esteem provided by the user to find out the sub graphs.

The basic format of FSM is to module candidates (Sub-Graph candidates) with the depth or breadth as former techniques and by implying it at the helping area as stated by the user [4]. The Extension of the FSM is sighed as two important views that should be faced with efficiency

(I) Identifying the overall candidates repeatedly in the Sub-Graph beyond the redundancy.

(II) Leave out the repeated check of the modules sub-graphs to avoid the time complexity.

Therefore, the safety should be measured to avoid the generation of copy or superfluous candidates. Getting back to tally checking might require a redundant relationship of candidate Sub-Graphs in the details of the data and FSM which is taken as the expansion of Frequent Item set Mining (FIM) that advances with regards to ASM[4]. Various scientists mentioned results for addressing the problems recognized with FSM and final coda property concludes with item set which is matched up with candidate sub-graphs age. The glimpse of this paper accompanies various best FSM related algorithms utilized by numerous strategies in accordance with time complexity, memory complexity and some space related problems too.

The recurrent sub-graph mining can be categorized into two fundamental categories, namely

1. Mining with graph collections and finding out the recurrent sub-graphs.
2. Mining with one large graph and to find out the sub-graphs.

Considering a graph datum set Gd = { G_1, G_2, G_3,…. G_N} I e..,. G1, G2, G3 where the collection of different graphs jotted in the datum set, the lessened support count threshold σ ($0 < \sigma \leq 1$). Therefore the support of M,

$$\text{Min Sup}(M) = |\delta(M)| / N$$

In which $|\delta(M)|$, cardinality of δ(M) and N, overall amount of graph that presents the graph dataset. Thus, M is recurrent, if $Sup(M) \geq \sigma$. The idea is simpler when the superset is recurrent, therefore all the subsets are also recurrent.

PROPOSED TASK:

The proposed algorithmic task, a unique large datum and the model datum is shown in the figure 1. Formerly the graph traversed from up to down and the vertices, nodes and edges are identified to transform the graphical datum into textual transformation data. The transformed transaction data is exclaimed with the invisible item alone in each and every row. The invisible datum indicates that converted binary presentation and therefore te specified candidates are joined using easy binary operation and the user as well provided with low support count threshold value.

Figure 1: Single graph dataset

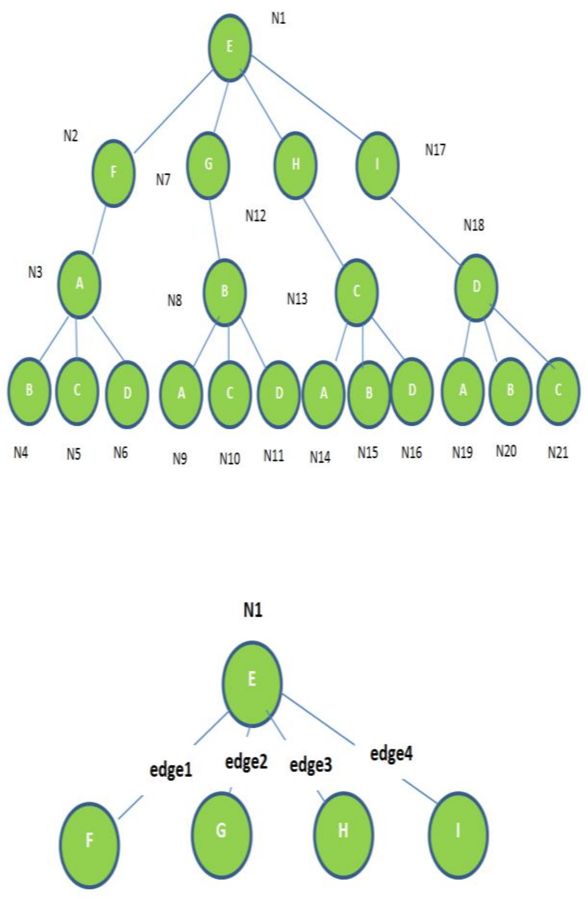
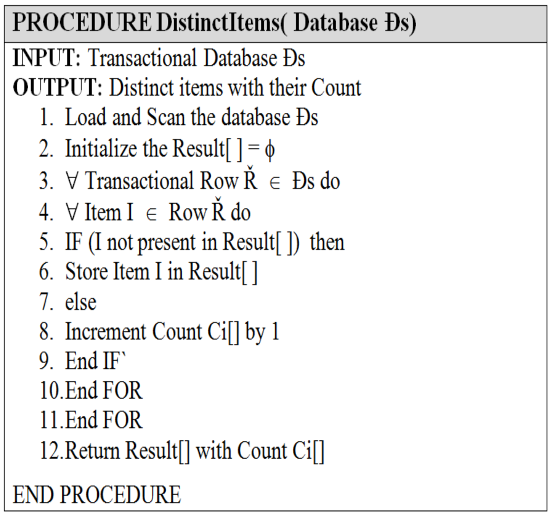


Figure 2: Finding the edges

Therefore, the numbers of edges are 4 as shown in the figure 2. Simultaneously for each and every node the edges are identified.

```
PROCEDURE Convert Graph To Text( Graph G)
Input: Single Graph data G
Output: Textual Labels with number of edges present in them
    1. Load the single Graph dataset
    2. Initially Scan the graph dataset to find the number of levels
    3. Detect the vertexes present in the graph
    4. Initialize edge =1 // to find the number of edges
    5. ∀ Vertex V in Graph G
          Discover the edges connected with the v
```



```
PROCEDURE DistinctItems( Database Ɖs)
INPUT: Transactional Database Ɖs
OUTPUT: Distinct items with their Count
    1. Load and Scan the database Ɖs
    2. Initialize the Result[ ] = ϕ
    3. ∀ Transactional Row Ř ∈ Ɖs do
    4. ∀ Item I ∈ Row Ř do
    5. IF (I not present in Result[ ]) then
    6. Store Item I in Result[ ]
    7. else
    8. Increment Count Ci[] by 1
    9. End IF`
    10.End FOR
    11.End FOR
    12.Return Result[] with Count Ci[]
END PROCEDURE
```

```
            Fetch the labels of the node
            Increment edge = edge + 1
            Store the labels  and edge  RES
    6.  End For
    7.  Return RES
```

Figure 3: Pseudo code to convert graph to text

The graph which is transformed into the textual data is figured out in the table 1 and therefore the table that has specific items are identified and once after they are customized the lessened support counts the invisible item represented which is processed below in the following section.

Table 1: Converted graph to text

Node	**items**	**Edges**	**Node**	**Items**	**edges**
N1	**E, F, G, H, I**	4	N11	**D, B**	1
N2	**F, E, A**	2	N12	**H, E, C**	2
N3	**A, F, B, C, D**	4	N13	**C, H, A, B, D**	4
N4	**B, A**	1	N14	**A, C**	1
N5	**C, A**	1	N15	**B, C**	1
N6	**D, A**	1	N16	**D, C**	1
N7	**G, C, B**	2	N17	**I, E, D**	2
N8	**B, G, A, C, D**	4	N18	**D, I, A, B, C**	4
N9	**A, B**	1	N19	**A, D**	1
N10	**C, B**	1	N20	**B, D**	1

The ultimate method to figure out the specific item is shown in the below figure 4 and then the pseudo code has been showed ,

Figure 4: Pseudo code to find the distinct items

The support count is provided by the user is 4 and the distinct items are found to be {A, B , C, D, E, F, G, H, I, } where the item count of F, G, H, I are found to be 3 and as it is lower than the user defined support, those items are pruned. The pruned transactional data representation is shown in the following table 2 and from this table the missing items are discovered.

Table 2: Pruned transactional data

Node	items	Node	Items
N1	**E**	N11	**D, B**
N2	**E, A**	N12	**E, C**
N3	**A, B, C, D**	N13	**C, A, B, D**
N4	**B, A**	N14	**A, C**
N5	**C, A**	N15	**B, C**
N6	**D, A**	N16	**D, C**
N7	**C, B**	N17	**E, D**
N8	**B, A, C, D**	N18	**D, A, B, C**
N9	**A, B**	N19	**A, D**
N10	**C, B**	N20	**B, D**

Thus, the items are invisible in each node are identified and represented as shown in the following table 3.

Table 3: Missing item transactional data

NODE	ITEMS	NODE	ITEMS
N1	**A,B,C, D**	N11	**A,C,E**
N2	**B,C,D**	N12	**A,B,D**
N3	**E**	N13	**E**
N4	**C,D,E**	N14	**B,D,E**
N5	**B,D,E**	N15	**A,D,E**
N6	**B,C,E**	N16	**A,B,E**
N7	**A,D,E**	N17	**A,B,C**
N8	**E**	N18	**E**
N9	**C,D,E**	N19	**B,C,E**
N10	**A,D,E**	N20	**A, C,E**

The table 3 is represented and when the item is presented in the node then it will be marked as one and elsewhere marked as zero. Therefore, the pseudo code to perform this task is proven in the figure 5.

```
PROCEDURE Binary Value(Missing Item M)
Input: Missing Item M
Output: Bit vector representation of data
    1. Load missing item Data set M and scan it.
    2. ∀ Row Ro∈M  begin
    3. ∀ Item It  ∈Ro  begin
    4. If [ It present in Ro] begin
    5. Mark as "1" in out
    6. Else
    7. Mark as "0" in out
    8. Close IF
    9. Close For
    10. Close For
    11. Return out
```

Figure 5: Pseudo code to represent the missing item dataset in binary format

The recurrent graphs are identified as computing the probable candidate and thus it start from the 2 item set value A and B.

A = 10000010011100111001

B = 11001100000101011010 |OR

AB =11 0 0 1 1 1 0 0 1 1 1 0 1 1 1 1 0 1 1

Then , count the number of zeroes and from the result {AB}=1100111001110111 1011, that the number of zero is 6 and the user defined support provided is 4. The count of AB is greater than the lessened support and therefore the graph is identified to be frequent. Then, the later level 3-itemset is found using this {AB} | {C}

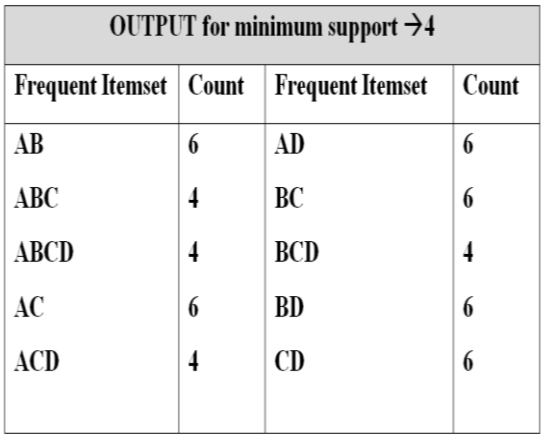
AB = 1 1 0 0 1 1 1 0 0 1 1 1 0 1 1 1 1 0 1 1

C = 1 1 0 1 1 1 0 0 1 0 1 0 0 0 0 0 1 0 1 1 | OR

ABC = 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 0 1 1

The number of zeroes is identified to be 3 and it is smaller in amount than the minimum support count and henceforth it is irrecurrent and the later level 4-itemset gets ignored. The final recurrent sub-graph is shown in the following table 5.

Table 5: Final result



OUTPUT for minimum support →4			
Frequent Itemset	Count	Frequent Itemset	Count
AB	6	AD	6
ABC	4	BC	6
ABCD	4	BCD	4
AC	6	BD	6
ACD	4	CD	6

PROPOSED ALGORITHM

```
ALGORITHM FSMM
Input: Graph Database G , min_sup
Output: Frequent Items
    1.  Load the graph dataset
    2.  Convert graph to test Text
        Table
    3.  Load the transaction table
        Text table
    4.  Find the missing Value  Tm
    5.  Bin=Binary Value(dataset
        Tm)
    6.  Find the level wise calculation
    7.  Count the number of zeroes
    8.  If [Count >= min_sup]
    9.  Store the item set in RES
    10. Calculate the next level item
        set
    11. Else
    12. Prune the Item set
    13. Return RES
```

Figure 6: Pseudo code of the proposed algorithm FSMM

EXPERIMENTAL EVALUATION

The proposed RSDI algorithm gets executed on the system comprising of 2.66 GHz I7 processor machine and a 4 GB memory running on Microsoft 10 ultimate operating system. Therefore, the algorithm is written in java based SPMF data mining toolkit. The RSDI algorithm compared with the existing algorithms like FSG [1], GSPAN [2], GASTON [3] and the results showcased proves the efficient working of the RSDI algorithm.

Kuramochi and Karypis [1] designed the FSG algorithms for mining all recurrent sub-graphs from graph datasets, with a level-wise approach as on the Apriori concepts and thus the algorithm had been the first one compared with the proposed RSDI.

The writer Yan and Han [2] designed GSPAN, that employed depth- first search, framed on a pattern growth principle same as the FP-growth algorithm and herewith the candidate has generated but it includes a heavy memory.

The writer Nijssen et al. Designed an efficient recurrent sub-graph mining tool, called Gaston, it identifies the recurrent substructure that are given in a number of phases of developing complexity [3].

The synthetic datum generator initially created a set of candidate graphs (the total number is controlled by L) and user specified size (I). Thus, the parameters produced in the synthetic graph generator are given in the table 6.

Table 6: Parameter used in synthetic dataset

Parameters	**Description of parameter**
D	Total number of graph
L	Total number of probable frequent sub-graph present
T	Number of Edges
V	Label count
I	Edge size
E	Edge label count

The datum set generated by the following table and three datum sets are generated and used for experimental comparison are in respect to the runtime and memory consumption.

Table 7: Dataset used

Dataset generated	**Number of Sequences**	**Avg. edges**	**Potential freq patterns**

D15kT30L200I11V4E4	10000	35	250
D25kT40L350I16V4E4	20000	45	375
D120kT50L500I20V4E4	100000	55	550

The formulated RSDI algorithm and some other prolonging algorithms are executed by the synthetic datum sets as given in the table 7. Thus the results formed after the existence in respect to the due time consumption is marked in the table 8.

Table 8: Run time comparison on D10kT30L200I11V4E4 (Small) dataset

RUNTIME (m SEC)					
Dataset - D15kT30L200I11V4E4					
Algorithm	**User defined Min_Sup values**				
	10	**100**	**1000**	**2000**	**2500**
FSG	1244	947	831	787	659
GSPAN	1219	894	726	678	646
GASTON	1079	859	717	626	538
FSMM	978	722	665	556	437

Thus, the experimental value brought out in the table 8 clearly defines that the proposed RSDI algorithms work on with small synthetic datum set. The GATSON algorithm relatively mirrors the performance of RSDI when a bigger support count is given but in case of the other two algorithms auctioned worsen and has taken a longer time to exit.

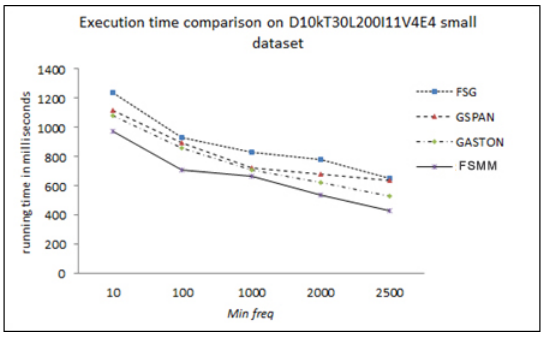


Figure 7: Graph related to runtime comparison

Table 9: Memory consumption comparison

MEMORY USAGE (MB)					
Dataset - D15kT30L200I11V4E4					
Algorithm	**User defined Min_Sup value**				
	10	**100**	**1000**	**2000**	**2500**
FSG	323	109	78	58	35
GSPAN	276	93	66`	42	29
GASTON	224	79	57	37	27
FSMM	168	66	45	29	21

The next comparison, forwarded with memory footprint and therefore the following table 9 proves the comparison of memory consumption.

The table 9 keenly figures out that the proposed RSDI algorithm which outscores the other three algorithms with a better margin in respect to the memory consumption. Therefore the minimum support value is diagnosed below 5, FSG becomes insufficient due to memory error and then the minimum support value lessened alive 3000 almost all the algorithms acted simultaneously.

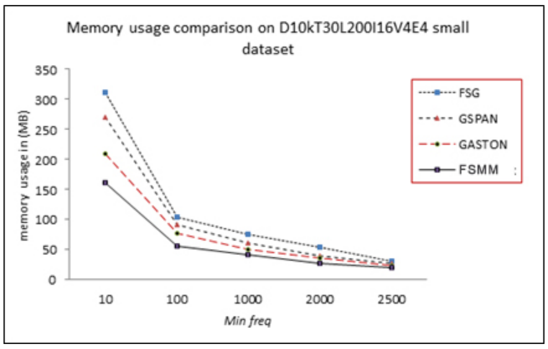


Figure 8: Graph related to memory consumption

CONCLUSION

The proposed algorithm RSDI proved that this papers glimpse and its experimental results are innovative keenly band therefore the RSDI overwhelmed the other algorithms by large amount of force in accordance with the runtime and memory consumption. This proposed algorithm keenly saves a huge amount of runtime and memory when it is executed on a large graph that datum sets are proved as asset to the research area.

REFERENCES

1. M. Kuramochi and G. Karypis. Frequent subgraph discovery. In Proc. International Conference on Data Mining'01, 2001.
2. Yan, X. and Han, J.W. 2002. gSpan: Graph-based Substructure pattern mining, In Proceedings of International Conference on Data Mining, 721–724.
3. S. Nijssen and J.N. Kok. The gaston tool for frequent subgraph mining. Electronic Notes in Theoretical Computer Science, 127:77-87, 2005.
4. Chen, M.S., Han,J.andYu,P.S. 1996 Data mining – An overview from database perspective, IEEE Transaction on knowledge and data engineering 8 , 866-883
5. http://cygnus.uta.edu/subdue/databases/index.html
6. http://citeseer.ist.psu.edu/oai.html
7. http://vlsicad.cs.ucla.edu/cheese/ispd98.html