AN APPROACH TO DETECT COMMON COMMUNITY SUB-GRAPH BETWEEN TWO COMMUNITY GRAPHS USING GRAPH MINING TECHNIQUES

BAPUJI RAO AND ANIRBAN MITRA
Department of CSE & IT
V.I.T.A.M. Berhampur, Odisha, INDIA
{rao.bapuji, mitra.anirban}@gmail.com

PRASANTA PADHI
Sysnetglobal Tech.(Pvt) Ltd, DRM Office,
E.Co. Railway, Sambalpur, Odisha, INDIA
prasanta.pdhi@gmail.com

ABSTRACT: Among various available graph patterns, one can discover the frequent sub-structures from a set of graphs. They are useful for characterizing graph sets, finding difference among groups of graphs, classifying and clustering graphs, and building graph indices leading to knowledge extraction. The Apriori-based approach, one such technique uses the breadth-first search (BFS) strategy because of its level-wise candidate generation. In this paper, we have started our work with representing the graphs, studying pattern growth and analysing the existing algorithm. Further, we have proposed our derived technique in the same direction. Our proposed technique is observed to be slightly efficient as compared to the existing algorithm. The paper concludes with analysis on the proposed technique with supportive examples, output, results and screen shot.

KEYWORDS: apriori, breadth-first search, community graph, community adjacency matrix, graph mining.

INTRODUCTION

Among various kinds of graph patterns, it is possible to discover the frequent sub-structures from a set of graphs. They are useful for finding difference among groups of graphs, classifying and clustering graphs, and building graph indices. The discovery of frequent sub-structures usually consists of two steps. In the first step, it generates frequent sub-structure candidates while the frequency of each candidate is checked in the second step. Most studies are trying to discover frequent sub-structures and the second step involvement of graph isomorphism which is NP-complete.

The initial frequent sub-structure mining algorithm, called AGM, was proposed by Inokuchi et al. [1], which shares similar characteristics with the Apriori-based item set mining [9]. The Apriori property is also used by other frequent sub-structure discovery algorithms such as FSG [6] and the path-join algorithm [8]. All of them require a join operation to merge two (or more) frequent sub-structures into one larger sub-structure candidate. They distinguish themselves by using different building blocks: vertices, edges, and edge-disjoint paths. In frequent sub-structure mining, Apriori-based algorithms have two kinds of considerable overheads: (i) joining two size-k frequent graphs (or other structures like paths [8]) to generate size-$(k + 1)$ graph candidates, and (ii) checking the

frequency of these candidates separately. These overheads constitute the performance bottleneck of Apriori-based algorithms.

The Apriori-based approach has to use the breadth-first search (BFS) strategy because of its level-wise candidate generation. To determine whether a size-*(k* + 1*)* graph is frequent; it has to check all of its corresponding size-*k* sub-graphs to obtain an upper bound of its frequency. Before mining any size-*(k* + 1*)* sub-graph, the Apriori-based approach usually has to complete the mining of size-*k* sub-graphs. Therefore, BFS is necessary in the Apriori-like approach. In contrast, the pattern growth approach is more flexible on the search method. Both breadth-first search and depth-first search (DFS) can work.

We have applied Apriori-based and pattern growth algorithms on our proposed social graph to detect the frequent sub-structure.

GRAPH REPRESENTATION TECHNIQUES

A graph can be represented in memory broadly in two different ways [5, 7, 10, 11].

Sequential Representation

This representation of technique is further classified into two.

Adjacency Matrix

Let ***G*** be a graph with n nodes or vertices $V_1, V_2, ..., V_n$ having one row and one column for each node or vertex. Then the adjacency matrix $A = [a_{ij}]$ of the graph ***G*** is the ***nXn square matrix***, which is defined as:

$$a_{ij} = \begin{cases} 1 \text{ if there is an edge from } V_i \text{ to } V_j \\ 0 \text{ Otherwise} \end{cases}$$

This kind of matrix contains only 0 and 1, is called bit matrix or Boolean matrix. In undirected graph, the adjacency matrix will be a symmetric. For example in the given digraph ***G*** in Figure 1(i) has vertices ***V = {A, B, C, D, E}*** and the set of edges ***E = {(A, B), (A, C), (B, D), (A, E)}***. Then the adjacency matrix of the graph ***G*** is shown in Figure 1(ii).

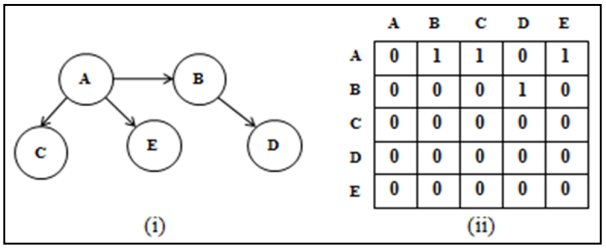


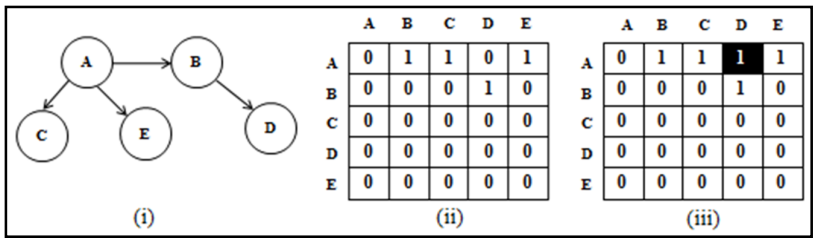
Figure 1. (i) Digraph G, (ii) Adjacency matrix of G

Path Matrix

The path matrix of graph ***G*** having ***n***-vertices is the ***nXn square matrix***, which is defined as:

$$P_{ij} = \begin{cases} 1 \text{ if there is a path from } V_i \text{ to } V_j \text{ via } V_k \\ 0 \text{ Otherwise} \end{cases}$$

The path matrix only shows the presence or absence of a path between a pair of vertices and also about the presence or absence of a cycle at any vertex. It never says the total number of paths in a graph. Let us consider a graph $G = \{A, B, C, D, E\}$. Its adjacency matrix and the final path matrix P is shown in Figure 2. An edge shows from A to D in Figure 2(iii) which is the indication of presence of path in the graph.



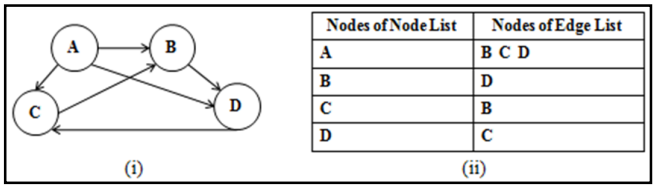
	A	B	C	D	E
A	0	1	1	0	1
B	0	0	0	1	0
C	0	0	0	0	0
D	0	0	0	0	0
E	0	0	0	0	0

	A	B	C	D	E
A	0	1	1	1	1
B	0	0	0	1	0
C	0	0	0	0	0
D	0	0	0	0	0
E	0	0	0	0	0

Figure 2. (i) Graph G, (ii) Adjacency matrix of G, (iii) Path matrix P of G

Linked List Representation

It consists of two types of lists, a *node list* and an *edge list*. A node list is a double linked list whose node consists of three parts: *Info*, *Next*, and *Adj*. *Info* is the information part of a node or vertex, *Next* is a pointer which holds address of next node of node list, and *Adj* is a pointer which holds address of node of edge list where the actual adjacent is present. An edge list is a single linked kind whose node consists of two parts: *Node* and *Edge*. *Node* is a pointer which holds address of node of node list where the adjacent is present and *Edge* is also a pointer which holds address of next node of edge list.



Nodes of Node List	Nodes of Edge List
A	B C D
B	D
C	B
D	C

Figure 3. (i) Digraph G, (ii) Adjacency list of G

Let us consider a graph $G = \{A, B, C, D\}$. The adjacency list for the graph G is shown in Figure 3(ii). From this we can represent its linked list representation in the memory.

FREQUENT GRAPH

Given a labelled graph dataset, $D = \{G_1, G_2, \ldots, G_n\}$, where frequency ($g$) is the number sub-graphs in D. A graph is frequent if its support is not less than a minimum support of nodes or vertices.

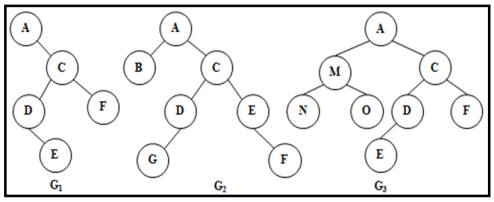


Figure 4. Graph dataset $D = \{ G_1, G_2, G_3\}$

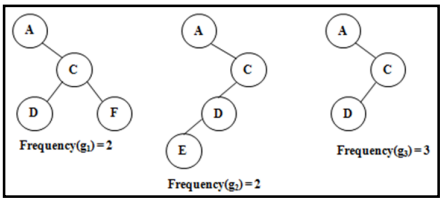


Figure 5. Frequency graphs

Let us consider a sample graph which is shown in Figure 4, has three datasets $\boldsymbol{D = \{G_1, G_2, G_3\}}$. We depict three frequent sub-graphs from the datasets $\boldsymbol{D}$, which is shown in Figure 5. So the three frequent sub-graphs $\boldsymbol{\{g_1, g_2, g_3\}}$ frequencies are $\boldsymbol{\{2, 2, 3\}}$.

APRIORI-BASED APPROACH

Apriori-based frequent item set mining algorithms developed by Agrawal and Srikant [9]. The frequent graphs having larger sizes are searched in a bottom-up manner by generating candidates having an extra vertex, edge, or path.

The process of Apriori-based methods is as follows. S_k is the frequent substructure set of size k. It adopts a level-wise mining methodology. At each of the iteration, the sizes of newly discovered frequent sub-structures are increased by one. New sub-structures are generated by joining two similar but slightly different frequent sub-graphs. Then the newly formed graphs are checked for their frequency. The detected frequent sub-graphs are used to generate larger candidates in the next round.

The candidate generation for frequent item set is as follows. Let us consider two frequent item sets of size 3 each :*(pqr)* and *(qrs)*. It generates a candidate frequent item set of size 4 which is *(pqrs)*. Here two item sets are *(pqr)* and *(qrs)* are frequent. Then we check the frequency of *(pqrs)*. So, the candidate generation problem in frequent sub-structure mining becomes much harder than frequent item set mining since there are different ways of joining two sub-structures.

Another kind of candidate generation strategies AGM [1] proposed a vertex-based candidate generation method that increases the sub-structure size by one vertex at each iteration is also follow the below algorithm logic. Two size-k frequent graphs are joined only when the two graphs have same size-$(k - 1)$ sub-graph. Here the size of a graph means the number of vertices in a graph. The newly formed candidate includes the common size-$(k - 1)$ sub-graph and the additional two vertices from the two size-k patterns. Since it is undetermined whether there is an edge connecting the additional two vertices, we actually can form two candidates. FSG proposed by Kuramochi and Karypis [6] adopts an edge-based method that increases the substructure size by one edge in each call of the below given Algorithm code. In this, two size-k patterns are merged if and only if they share the same sub-graph that has $k - 1$ edges, which is called the *core*. Here the size of a graph means the number of edges in a graph. The newly formed candidate includes the core and the additional two edges from the size-k patterns.

Other Apriori-based methods such as the disjoint-path method proposed by Vanetik [8] use more complicated candidate generation procedures. A sub-structure pattern with $k + 1$ disjoint path is generated by joining sub-structures with k disjoint paths. Apriori-based algorithms have more overheads while joining two size-k frequent sub-structures to generate size-$(k + 1)$ graph

candidates. To avoid such overheads, we need to follow non-Apriori-based algorithms which have been developed recently.

PATTERN GROWTH APPROACH

A graph can be extended by adding a new edge. The edge may or may not introduce a new vertex to the newly formed graph. The extension of new graph may extend is in a ***forward*** or ***backward*** direction. The algorithm discovers the same graph more than ones. This repeated discovery has to be avoided. If there exist n different $(n-1)$-edge graphs can be extended to the same n-edge graph. It lacks the repeated discovery of the same graph. So the generation and detection of duplicate graphs may cause additional workloads. To reduce the generation of duplicate graphs, each frequent graph should be preserved as possible. This principle leads to the design of several new algorithms such as gSpan [13], MoFa [2], FFSM [13], SPIN [4], and Gaston [12].

OUR PROPOSED APPROACH

We have studied the scenario of a social graph [10, 11], which consists of various villages in a panchayat (panchayat is an Indian term for administration of villages). In a village different types of communities live together and have connectivity. Taking this scenario into mind, one can compare two community graphs for finding a similar sub-graph from it. For such case we have proposed an algorithm for detecting frequent sub-graph between two community graphs. A simple technique using graph theory is employed to detect the frequent sub-community graph between two community graphs. The proposed algorithm has been given below.

Algorithm Frequent_Graph_Inbetween_2_Graphs (V_1, S_1, V_2, S_2) (Algorithm conventions [5])
V_1 [1:S_1, 1:S_1]: Adjacency matrix of Community Graph-1.
V_2 [1:S_2, 1:S_2]: Adjacency matrix of Community Graph-2.
Result [1: Size, 1: Size]: Global 2D-Array for representation of frequent graph's adjacency matrix.
Size: The unique communities from villages V_1 and V_2.

1. [Assign 0s to Result[ ][ ] Adjacency Matrix of Frequent Graph]
Repeat For i:=1, 2, ..., Size:
Repeat For j:=1, 2, ..., Size:
Set Result[i][j] := 0.
End For
End For
2. Set i := 2.
Set j := 2.
3. Repeat While i<=S_1
Do
(a) Repeat While j<=S_2
Do
If V_1[i][1] = V_2[j][1],
Then
(i) Call Column_Compare (V_1, S_1, i, V_2, S_2, j).
(ii) i:=i+1.
(iii) j:=j+1.
Else

(iv) j:=j+1.
End If
End While
(b) Set j := i.
(c) i:=i+1.
End While
4. Exit.

Procedure Column_Compare (V_1, S_1, r_1, V_2, S_2, r_2)
1. Set c_1 := 2.
Set c_2 := 2.
2. Repeat While c_1<=S_1
Do
(a) Repeat While c_2<=S_2
Do
(a.1) If $V_1[1][c_1] = V_2[1][c_2]$,
Then
[Edge between same pair of communities in both the graphs]
If $V_1[r_1][c_1] = V_2[r_2][c_2] = 1$,
Then
[Finding actual row position of village in Result [ ][ ] Matrix]
(i) Set row := 2.
(ii) Repeat While TRUE
Do
If V_1[r1][1]=Result[row][1],
Then
Break.
Else
row := row+1.
End If
End While
[Show edge of same pair of communities in the Result [ ][ ] Matrix]
(iii) Set p:=1.
(iv) Repeat While TRUE
Do
(iv.a) If V_1[1][c_1]=Result[1][p],
Then
(I) Set Result[row][p] := 1.
(II) Break.
End If
(iv.b) p:=p+1.
End While
End If
(a.2) c_1:=c_1+1.
(a.3) c_2:=c_2+1.
Else
(a.4) c_2:=c_2+1.
End If [close of if(a.1)]

End While [close of while (a)]
(b) Set $c_2 := c_1$.
(c) $c_1 := c_1 + 1$.
End While [close of while (2)]
3. Exit.

Example
Let us consider a community graph for villages V_1, V_2, V_3, V_4, V_5, and V_6 which is shown in Figure 6 [10, 11]. For villages V_1, V_2, V_3, V_4, V_5, and V_6, the communities are $\{C_1, C_2, C_3, C_4\}$, $\{C_1, C_2, C_3, C_5\}$, $\{C_1, C_2, C_3, C_4, C_5\}$, $\{C_1, C_3, C_4, C_8\}$, $\{C_1, C_2, C_3, C_5, C_9, C_{10}\}$, and $\{C_2, C_3, C_4, C_5, C_8, C_9\}$ respectively.

We have considered two community graphs for villages V_3 and V_5 which are shown in Figure 7(i) and Figure 7(iii) for detection of a frequent sub-community graph. Its adjacency matrices are shown in Figure 7(ii) and Figure 7(iv).

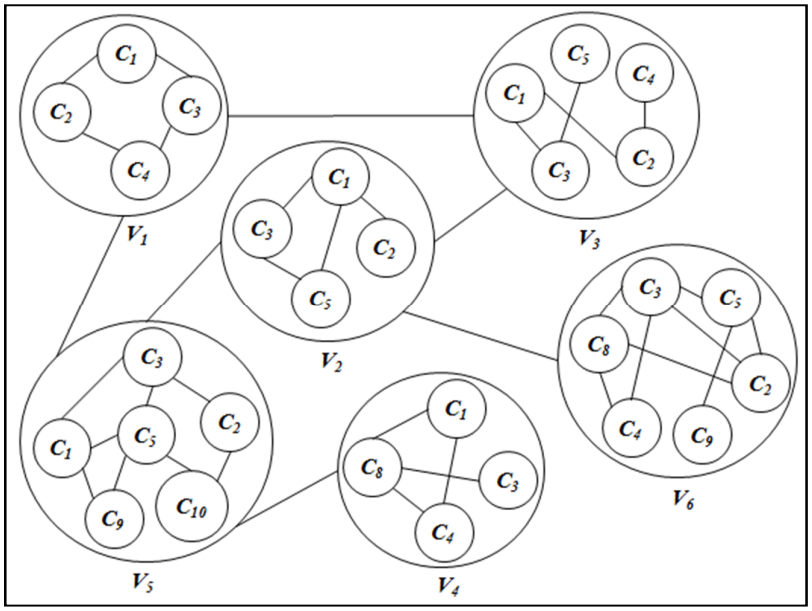
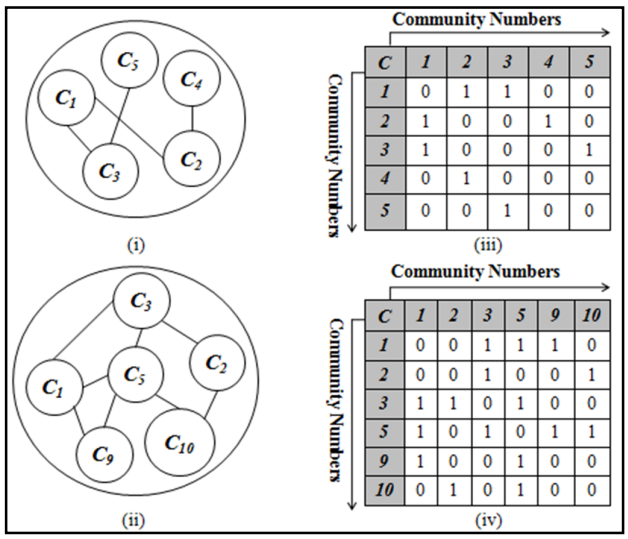


Figure 6. Community graph of villages $V = \{V_1, V_2, V_3, V_4, V_5, V_6\}$

The above algorithm has to pass four parameters such as V_3, 5, V_5, and 6. Then the algorithm combines both the communities V_3 and V_5. It able to find the order of resultant adjacency matrix by finding the total number of unique communities (here it is 7). Then the adjacency matrix is created of order 7×7 which is shown in Figure 8(iii). Then the final resultant matrix is formed by calling the procedure ***Column_Compare*** $(V_1, S_1, r_1, V_2, S_2, r_2)$ which is shown in Figure 9(i).

Finally by using the resultant matrix in Figure 9(i), we can draw the frequent sub-community graph which is shown in Figure 9(ii).



C	1	2	3	4	5
1	0	1	1	0	0
2	1	0	0	1	0
3	1	0	0	0	1
4	0	1	0	0	0
5	0	0	1	0	0

C	1	2	3	5	9	10
1	0	0	1	1	1	0
2	0	0	1	0	0	1
3	1	1	0	1	0	0
5	1	0	1	0	1	1
9	1	0	0	1	0	0
10	0	1	0	1	0	0

Figure 7. (i) Community graph of village V_3
(ii) Community graph of village V_5
(iii) Adjacency matrix of village V_3
(iv) Adjacency matrix of village V_5

$V_3 = \{ C_1, C_2, C_3, C_4, C_5 \}$, $C_{tot1} = 5$, $V_5 = \{ C_1, C_2, C_3, C_5, C_9, C_{10} \}$, $C_{tot2} = 6$
$Size = V_3 \cup V_5 = 7$. So the unique communities are $\{C_1, C_2, C_3, C_4, C_5, C_9, C_{10}\}$

C	1	2	3	4	5
1	0	1	1	0	0
2	1	0	0	1	0
3	1	0	0	0	1
4	0	1	0	0	0
5	0	0	1	0	0

(i)

C	1	2	3	5	9	10
1	0	0	1	1	1	0
2	0	0	1	0	0	1
3	1	1	0	1	0	0
5	1	0	1	0	1	1
9	1	0	0	1	0	0
10	0	1	0	1	0	0

(ii)

C	1	2	3	4	5	9	10
1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0

(iii)

Figure 8. (i) Adjacency matrix of village V_3
(ii) Adjacency matrix of village V_5
(iii) Initial adjacency matrix *Result [ ][ ]*
of common community sub-graph

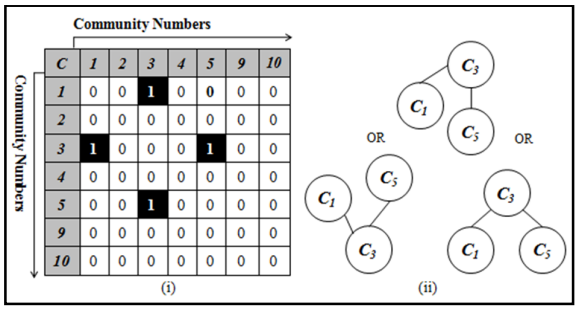
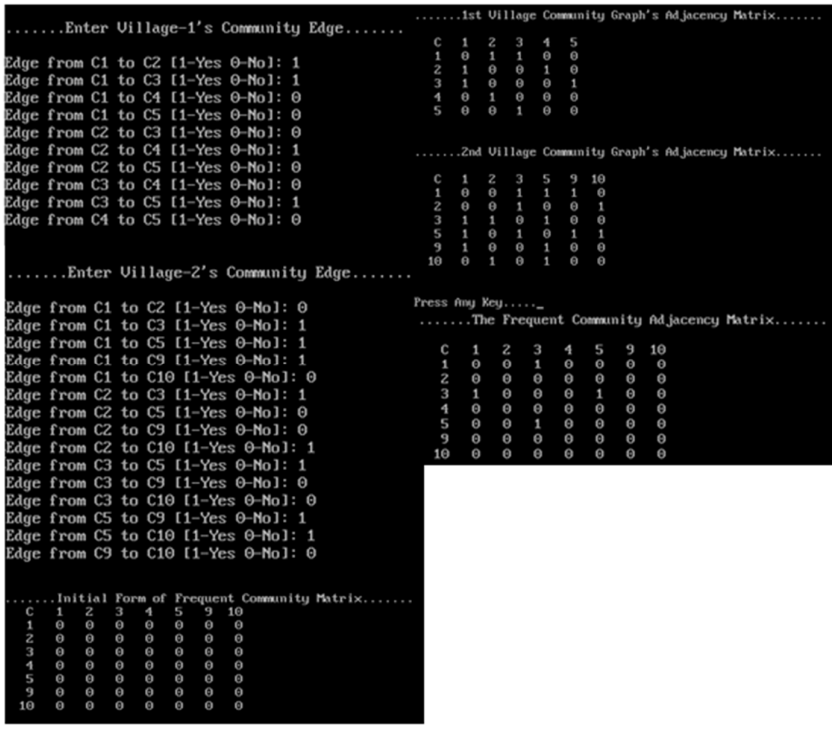


Figure 9. (i) Final adjacency matrix *Result* of common community sub-graph between villages V_3 and V_5
(ii) The drawn common community sub-graph from villages V_3 and V_5

Example's Output



CONCLUSIONS

Apriori-based approach uses the breadth-first search (BFS) strategy because of its level-wise candidate generation. This paper initiate with a thorough analysis on this existing algorithm.

Further the authors have proposed their derived technique based on the Apriori-based approach. It was observed that the proposed technique is slightly efficient as compared to the existing algorithm. Results and observations on suitable examples were computed using the proposed technique and its related sample output and have included to justify.

REFERENCES

A.Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In Proceedings of 2000 European Symposium Principle of Data Mining and Knowledge Discovery (PKDD'00), pp. 13–23, Lyon, France, Sept. 2000.

C. Borgelt and M. R. Berthold. Mining molecular fragments: Finding relevant substructures of molecules. In Proceedings of 2002 International Conference on Data Mining (ICDM'02), pp. 211–218, Maebashi, Japan, Dec. 2002.

J. Huan, W. Wang, and J. Prins. Efficient mining of frequent subgraph in the presence of isomorphism. In Proceedings of 2003 International Conference on Data Mining (ICDM'03), pp. 549–552, Melbourne, FL, Nov. 2003.

J. Prins, J. Yang, J. Huan, and W. Wang. Spin: Mining maximal frequent subgraphs from graph databases. In Proceedings of 2004 ACM SIGKDD International Conference on Knowledge Discovery in Databases (KDD'04), pp. 581–586, Seattle, WA, Aug. 2004.

Lipschutz, Seymour. Schaum‟s outline of Data Structures, Tata McGraw-Hill Publishing Company Limited, 7 West Patel Nagar, New Delhi 110 008.

M. Kuramochi and G. Karypis. Frequent subgraph discovery. In Proceedings of 2001 International Conference on Data Mining (ICDM'01), pp. 313–320, San Jose, CA, Nov. 2001.

Mitra A., Satpathy S. R. Paul S. : Clustering analysis in social network using Covering Based Rough Set, Advance Computing Conference (IACC), 2013 IEEE 3rd International, India, 2013/2/22, 476-481, 2013.

N. Vanetik, E. Gudes, and S. E. Shimony. Computing frequent graph patterns from semistructured data. In Proceedings of 2002 International Conference on Data Mining (ICDM'02), pp. 458–465, Maebashi, Japan, Dec. 2002.

R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In Proceedings of 1994 International Conference Very Large Data Bases (VLDB'94), pp. 487–499, Santiago, Chile, Sept. 1994.

Rao, Bapuji and Mitra, A. A New Approach for Detection of Common Communities in a Social Network using Graph Mining Techniques. 2014 IEEE International Conference on High Performance Computing & Application (ICHPCA-2014), Bhubaneswar, India, Dec 22-24, 2014.

Rao, Bapuji and Mitra, A. An Approach to Merging of two Community Sub-Graphs to form a Community Graph using Graph Mining Techniques. 2014 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC-2014), Coimbatore, India, pp. 460-466, Dec 18-20, 2014.

S. Nijssen and J. Kok. A quickstart in frequent structure mining can make a difference. In Proceedings of 2004 ACM SIGKDD International Conference on Knowledge Discovery in Databases (KDD'04), pp. 647–652, Seattle, WA, Aug. 2004.

X. Yan and J. Han. gSpan: Graph-based substructure pattern mining. In Proceedings of 2002 International Conference on Data Mining (ICDM'02), pp. 721–724, Maebashi, Japan, Dec. 2002.