Identifying Dominating Graphs over Vertex Connectivity and Order Constraint in a Social Network

Abhay A. Bhamaikar and Pralhad Ramchandra Rao

***Abstract*—Identifying highly robust graphs (i.e. graph having high vertex connectivity value) is a well known problem in the field of social network analysis. Robustness of a social network implicitly assumes that social links are dynamic and should be allowed to be changed with and without restrictions or constrains. In social network finding robust subgraphs is hard problem i.e. NP Complete Problem. The paper focuses on identifying whether the original graph is robust or not. It takes into consideration the Maximum degree, Minimum degree and Vertex Connectivity values of given Graph (G), based on which the algorithm determines above Property.**

The algorithm uses vertex connectivity and order constraints to find robust subgraphs i.e. Dominating (H) from set of all induced subgraphs. The social network G is decomposed using minimum cut decomposition algorithm over vertex connectivity parameter. The skyline approach is used to determine the solution set Dominating (H). The computational time of algorithm is improved by using preprocessing techniques like identifying and deleting cut set vertices and also by using pruning strategies like minimum degree criteria.

***Index Terms*—Skyline, order, vertex connectivity, robustness.**

I. Introduction

In the last years, there has been an ever-increasing research activity in the study of real-world complex networks (the world-wide web, the Internet autonomous-systems graph, co authorship graphs, phone call graphs, email graphs and biological networks, to cite a few). These networks, typically generated directly or indirectly by human activity and interaction, appear in a large variety of contexts and often exhibit a surprisingly similar structure. One of the most important notions that researchers have been trying to capture is node centrality": ideally, every node (often representing an individual) has some degree of influence or importance within the social domain under consideration, and one expects such importance to be reacted in the structure of the social network; centrality is a quantitative measure that aims at revealing the importance of a node.

In this paper, by a graph we mean an undirected graph with no loops and no multiple edges. Graphs are widely used to represent complex structures that are difficult to model. Using labeled graphs or unlabeled graphs depends on the application need. The vertex set of a graph G is denoted by V (G) and the edge set by E (G). A graph G is a subgraph of H if G has all its vertices and edges of H and it is denoted by H ⊆ G. G is called a super graph of H.

Existing pattern discovery approaches operate by using simple constraints on the mined patterns. For example, given a database of graphs, a typical graph mining task is to report all subgraphs that appear in at least *s* graphs, where *s* is the frequency support threshold. In other cases, we are interested in the discovery of dense or highly-connected subgraphs. In such a case, a threshold is defined for the density or the connectivity of the returned patterns. Other constraints may be defined as well, towards restricting the number of mined patterns. However, the methods proposed so far apply the constraints by posing simple thresholds in the mining process (e.g., give me all subgraphs which contain at least *n* vertices and their frequency is at least *f*).

There are three important limitations with this approach:

1) There is an on-off decision regarding the eligibility of patterns, i.e., a pattern either satisfies the constraints or not.
2) In the case where the constraints are very strict, we risk an empty answer or an answer with only a few patterns, and
3) In the case where the constraints are too weak the number of patterns may be huge.

In skyline approach, the records returned to the user are the ones that are not dominated by any other record, where domination is based on the values of each record. Let *p* and *q* be two records, each composed of *d* attributes. We denote by *pi* (*qi*) the value of the *i*th attribute of *p* (*q*). Record *p* dominates record *q* if *p* is "as good as" *q* in all attributes and is "better" than *q* in at least one attribute. Assuming a preference in large values, *p* is better than *q* in the *i*th attribute if $pi > qi$.

Skyline processing is scale invariant, it does not require a Ranking function, it does not require any threshold and can be used as long as the data dimensionality is low (e.g., below 10). For high dimensional spaces the probability that a record dominates another is very small and this may lead to an increased answer size towards applying preferences in subgraph discovery, each subgraph can be seen as a record containing two attributes: 1) the order (number of vertices) and 2) the vertex connectivity. The importance of a discovered subgraph increases as both the order and the vertex connectivity increase. Therefore, the best possible subgraphs (termed skyline subgraphs) are the ones that are maximized both in order and vertex connectivity.

In top-*k ranking* algorithm, a ranking function is applied to records and the *k* records with the highest score are returned to the user. Skyline approach provides large number of

Manuscript received April 15, 2013; revised June 26, 2013.

Abhay A. Bhamaikar is with the Department of Information technology, Shree Rayeshwar Institute of Engineering and Information technology, Shivshail, Karai, Shiroda – Goa, India (e-mail: abhay_bhamaikar@rediffmail.com).

Pralhad Ramchandra Rao is with the Department of Computer Science and Technology, Goa University, India.

dominating but important subgraphs in the solution sets. Top k ranking algorithm reduces the solution set and assists in obtaining only the highly robust subgroups.

A *labeled graph* can be represented by a 4-tuple, $G = (V, E, L, l)$, where V is a set of vertices and E is a set of edges, L is a set of labels, $l: V \cup E \rightarrow L$ in which l is a function assigning labels to the vertices and edges. Given an undirected graph G, if a path exists between any two vertices, G is called a *connected graph*.

Definition 1 Graphs: A graph $G = (V, E)$ is a pair in which V is a (non-empty) set of vertices or nodes and E is either a set of edges $E \subseteq \{\{v, w\} \mid v, w \in v, v \in w\}$ or a set of arcs $E \subseteq \{(v, w) \mid v, w \in v, v \in w\}$. In the latter case we call the graph directed.

Definition 2 The vertex connectivity $\kappa(G)$ of a connected graph G (other than a complete graph) is the minimum number of vertices whose removal disconnects G. When $\kappa(G) \geq \kappa$, the graph is said to be κ-connected (or κ-vertex connected). When we remove a vertex, we must also remove the edges incident to it.

The significance of a graph G is determined by two attributes, the *graph order* (number of vertices) denoted as N and the *vertex connectivity* of G, denoted as $\kappa(G)$.

The vertex connectivity, $\kappa(G)$, of a graph G is a measure of its *robustness*. If $\kappa(G)$ is large, then the graph is considered more robust. A small $\kappa(G)$ is a sign that the graph can be easily decomposed in two subgraphs, and therefore it is considered less robust.

Definition 3 Maximum Degree (Δ) of a given Graph (G) is defined as the largest degree over all the vertices.

Definition 4 Minimum degree (δ) of a given Graph (G) is defined as the smallest degree over all the vertices.

Definition 5 Monotonic Constraint: A monotonic Constraint is a constraint C_m such that for all Subgraphs H derived from a Graph G satisfies C_m if H satisfies it.

Monotonic Graph over Vertex Connectivity and Order Constraint implies that the set of subgraphs (H) obtained by decomposing the graph (G) will always contain the value of the Vertex connectivity and Order less than that of Graph (G).

Let $H(G)$ be the set of all induced subgraphs of graph G. Among all subgraphs in $H(G)$ we are interested in determining the most important ones regarding order and vertex connectivity.

Each subgraph g is represented as a pair $Ng, \kappa(g)$, where Ng is the number of vertices of g and $\kappa(g)$ the vertex connectivity. Between two subgraphs $gi \in H(G)$ and $gj \in H(G)$, gi is considered more significant than gj if one of the following holds:

– $N(gi) > N(g\,j)$ and $\kappa(gi) > \kappa(gj)$: in this case, gi has more vertices that gj and also the edge connectivity of gi is higher than that of gj.

– $N(gi) = N(gj)$ and $\kappa(gi) > \kappa(gj)$: in this case, gi and gj have the same number of vertices, and gi has higher connectivity than gj .

– $N(gi) > N(gj)$ and $\kappa(gi) = \kappa(gj)$: in this case, gi contains more vertices than gj, but the edge connectivity of the two graphs is the same.

More formally:

Dominating (H) = $\{gx \in H(G): gz \in H(G), gz <: gx\}$

II. Related Works

There is an on-going interest in the research community regarding knowledge discovery from graph data [1], [2]. In this section, we briefly present some fundamental contributions related to our work. Density has been used as a measure of subgraph importance. In [2] an algorithm has been studied to determine the densest subgraph of a given input graph, by using $O(\log N)$ min-cut computations, where N is the number of vertices of graph G. However, the densest subgraph may not be adequate to draw conclusions regarding the properties of the initial graph. The basic limitation of the algorithm is that it is not able to discover more than one dense sub-graph. The algorithm proposed in [3], [4] is able to determine many dense bipartite subgraphs of large graphs.

The density concept has been also used in [5] where the authors study the problem of dense subgraph discovery across a set of input graphs. Instead of applying a dense discovery algorithm in each graph, a summary graph is first constructed and then processed to determine the important subgraphs. A similar method has been used in [6], [7] for reconstruction of human transcriptional regulatory modules. The main limitation of this technique is that the answer is composed by all subgraphs satisfying the constraints. This set may be too small or too large, according to the constraints applied. Since density cannot describe adequately the coherency of a graph, the concept of edge connectivity has been used, which is a well-known concept in Graph Theory [8]-[10]. Edge connectivity has been applied as a clustering tool, where clusters are formed by the vertices of a graph G that show a high degree of connectivity [11], [12].

In [5] the authors describe two algorithms (CLOSECUT and SPLAT) for mining frequent subgraphs in relational graphs, by using connectivity constraints. The algorithms report all closed subgraphs that satisfy the connectivity constraints and the support threshold. A general framework for incorporating constraints into the discovery process has been proposed in [6]. The gPrune method is proposed which uses the concept of pattern-inseparable data-antimonotonicity. This framework fails to consider cases where only the most important patterns (regarding some preference criteria) are required. All the aforementioned research efforts are characterized by the application of specific constraints that must be met by the discovered subgraphs. These constraints may involve the number of vertices, the density, the edge connectivity, the vertex connectivity [13], [14] or the frequency. We go beyond this approach, proposing a technique for the discovery of subgraphs that are "as good as possible" with respect to some important characteristics such as the number of vertices and the edge connectivity. This way, only the most significant subgraphs (regarding the preference criteria) are exposed, whereas the rest are not contained in the final result. Our work is inspired by the plethora of research proposals towards supporting skyline query processing in database systems [15], [16]. The recent contribution is with regard to SkyGraph algorithm [17] where in the edge connectivity and number of vertices has been taken as constraints.

The work in the paper is inspired from [18]–[20]

III. Contribution

In the paper we determine set of dominating sub-graphs from the set of all the induced sub-graphs (H) of the given graph (G) which dominates over the values of order and vertex connectivity. The induced sub-graphs are obtained using minimum cut decomposition over vertex connectivity parameter. The skyline approach is used to determine the solution set (Dominating (H). The computational time of algorithm is improved using preprocessing techniques and pruning strategies.

The problem we study in this work is formally stated as follows: Given a Relational Graph G, we determine the set of induced connected Sub-graph of G, which are maximal with respect to the number of vertices and vertex Connectivity Constraint.

IV. Monotonic Graphs

Given a relational graph G, a general method *is* to generate all induced connected subgraphs, and then compute the vertex connectivity and order of subgraphs. However, the number of subgraphs in G is exponential in relation to the number of vertices, and therefore the performance of the method degrades rapidly for large graphs. For this reason, we are interested in a polynomial time algorithm to solve the problem. Such an algorithm, termed Algorithm Vertex Monotone Graph (G), is proposed in the sequel.

Theorem 1

Let δ min (g) denotes the minimum degree among all vertices of g and Ng the number of vertices of g. If $\delta \min(g) \geq Ng/2$, then $\lambda(g) = \delta\min(g)$. As we know $K(g) \leq \lambda(g) \leq \delta(g)$,Hence $K(G) \leq \delta\min(g)$.

Lemma 1

The worst case for the Algorithm Mining Important Sub-Graph appears when in each min-cut computation one of the subgraphs contains a single vertex.

Proof*:* Let N and M denote the number of vertices and the number of edges of the input graph. The complexity of the min-cut algorithm is $O(M \cdot N + N2 \cdot \log N)$. Since we are interested on the number of vertices, we express M as a function of N by considering three different cases:

1) $M = O(N)$,
2) $M = O(N \cdot \log N)$, and
3) $M = O(N2)$.

For the first two cases the worst-case complexity of min-cut is $O(N2 \cdot \log N)$, whereas in the third case the complexity becomes $O(N3)$.We will show that the worst-case complexity appears when the application of each min-cut computation results in a completely unbalanced cut, where one of the subgraphs contains a single vertex (i.e., each time only one vertex is removed from the graph).

Let $F(N) = C \cdot N^2 \cdot \log N$ (the $O(N^3)$ case is handled in a similar manner), where C is a positive real constant. To prove the statement of the lemma it is sufficient to prove that for any integer

$$x \in [0, N-2]: F(1) + F(N-1) \geq F(1+x) + F(N-1-x)$$

Essentially, this inequality states that we cannot find a worse cut than that produced by isolating a vertex each time we apply the min-cut algorithm. Thus, for any integer number $x \in [0, N-2]$, a more balanced cut cannot have worse performance.

The above inequality is equivalent to the following one:

$$F(N-1) - F(N-1-x) \geq F(1+x) - F(1)$$

Since $F(N)$ is increasingly monotone, both parts of inequality are positive. Moreover, since the growth rate of $F(N)$ (as determined by its first derivative $F(N) = 2 \cdot C \cdot N \cdot \log N + (C/\ln 2) \cdot N$) increases in a log linear fashion by increasing N, it follows that inequality is true.

Lemma 2

Let g be a subgraph produced during the generation of the MCD-tree. It is not necessary to continue the decomposition process if g is one of the following: 1) a tree, 2) a cycle or 3) a clique.

Proof *let g* be the subgraph corresponding to the node of interest. 1) If g is a tree, then $K(g) = 1$ and any induced connected subgraph of g has an vertex connectivity of 0 (single vertex) or 1 (a tree) and contains at most $Ng - 1$ vertices. This means that any induced subgraph of g is dominated by g and therefore cannot be part of the Solution set. 2) If g is a cycle, then $K(g) = 2$. Any induced connected subgraph of g has an vertex connectivity of 1 and contains at most $Ng - 1$ vertices. Again, these facts show that g dominates any connected subgraph induced by g. 3) If g is a clique, then $K(g) = Ng - 1$. Any induced connected subgraph of g has an Vertex connectivity at most $K(g) - 1$ and contains at most $Ng - 1$ vertices. Again, it is observed that g dominates all induced connected subgraphs of g. In conclusion, it is safe to declare the node associated to g as a leaf node and terminate the decomposition process since the continuation does not have an impact on the final result set.

Lemma 3

If the vertex connectivity of the given graph (G) is one and the given graph is tree then the given graph (G) satisfies monotonic property over Vertex Connectivity and Degree Constraint.

Proof: Consider a Graph G, Let the vertex connectivity of Graph (G) be 1 i.e. $K(G) = 1$.

Since the given graph is tree its vertex connectivity is always 1 and the vertex connectivity of its induced subgraph is either 1 or 0.

Hence a Graph (G) satisfies monotonic property over vertex connectivity constraint.

Lemma 4

If Δ is the Maximum Degree and δ is the minimum Degree of a given graph (G) then the Vertex connectivity of the subgraph (H) obtained from the given graph (G) is always less than or equal to Maximum Degree.

Proof:

1) The *Subgraph* (H) obtained from given Graph (G) can have Minimum Degree at most Δ.
2) The Vertex connectivity of (H) cannot be greater than Minimum Degree (δ).

3) The *vertex*-connectivity of a graph is less than or equal to its edge-connectivity. That is, $\kappa(G) \leq \lambda(G)$. Both are less than or equal to the minimum degree of the graph, since deleting all neighbors of a vertex of minimum degree will disconnect that vertex from the rest of the graph.

Hence Vertex Connectivity of H is at most Δ.

Lemma 5

Consider a Graph *G* Having maximum degree = Δ, Minimum Degree = δ and Vertex Connectivity = K_G .Let *H* be set of all induced subgraphs obtained from Given Graph *G* having Vertex Connectivity $K_{H.}$

If maximum degree is equal to Minimum Degree ($\Delta = \delta$) then the Vertex connectivity of all the induced subgraph ($K_{H.}$) is always less than the edge connectivity K_G of Given Graph. Under the condition that Given Graph *G* does not contain any Cut Vertex.

Proof: Since the Maximum Degree (Δ) = Minimum Degree (δ).Therefore Edge Connectivity of given graph λ_G is less than equal to Maximum Degree (Δ) or Minimum Degree (δ).

The Edge Connectivity λ_H of Subgraph (*H*) is always less than equal to Maximum Degree (Δ) from Lemma One.

Hence the given graph will always satisfy monotone constraint over edge connectivity.

Lemma 6

Consider a Graph *G* Having maximum degree = Δ, Minimum Degree = δ and Vertex Connectivity = K_G .Let *H* be set of all induced subgraphs obtained from Given Graph *G* having Vertex Connectivity $K_{H.}$

If the Vertex Connectivity K_G of Graph (*G*) is less than Minimum Degree (δ) and there exists clique with Maximum degree equal to δ then there exists Subgraph whose Vertex connectivity is greater then Vertex connectivity of Graph (*G*), Hence Graph Does not Satisfy Monotonic Property.

Proof: If K_G is Vertex connectivity of Graph *G* and δ is minimum degree of Graph G and Vertex Connectivity K_G is less than Minimum Degree δ of graph *G* and there exist a clique whose minimum degree is equal to δ, then Vertex connectivity (K_H) of clique is equal to δ.

This Implies K_H is greater than $K_{G.}$

Hence a given Graph (*G*) satisfies anti-monotonic property over Vertex connectivity constraint.

Lemma 7

Consider a Graph *G* Having maximum degree = Δ, Minimum Degree = δ and Vertex Connectivity = K_G .Let *H* be set of all induced subgraphs obtained from Given Graph *G* having Vertex Connectivity $K_{H.}$

If the Vertex Connectivity K_G of Graph (*G*) is equal to Minimum Degree (δ) and minimum degree is less than equal to n where n is less than Maximum Degree (Δ) and if there exists clique with Maximum degree equal to n+1 then there exists Subgraph whose Vertex connectivity is greater then Vertex connectivity of Graph (*G*), Hence Graph Does not Satisfy Monotonic Property.

Proof: Let κ_G be Vertex connectivity of Graph *G* and δ be minimum degree of Graph *G*.

If $K_G = \delta$ and δ is less than equal to n and Δ is greater than n and there exist a clique with minimum degree equal to $n + 1$, This implies Vertex Connectivity (K_H) of clique is greater than δ.

Hence a given Graph (*G*) satisfies anti-monotonic property over Vertex connectivity constraint.

V. Algorithm and Explanation

In this section we propose Algorithm Rob-sky (H) for detecting important sub- graphs over vertex connectivity and order constraint. The later part of the section explains the algorithm in brief.

Algorithm for Mining dominating sub-graphs

Algorithm Dominating (H)
Input: G, Initial Input Graph
Output: Dominating Sub-Graphs (H)

1) Initialize queue Q; solution (g) = Null;
2) Insert connected components of G into Q;
3) While (Q not empty)
4) If Graph (G) contains a cut set && Graph (G) is a Tree
5) Then G is dominating.
6) Else
7) Calculate Max. Degree (Δ) and Min. Degree (δ).
8) End if
9) If Maximum Degree = = Minimum Degree
10) Then G is dominating.
11) Else Calculate Vertex Connectivity (K)
12) End if
13) If Maximum Degree ≠ Minimum Degree
14) If Minimum Degree = = Vertex Connectivity
15) && Minimum Degree = = n, n < Maximum Degree
16) If There Does Not Exist Clique with
17) Degree (n+1)
18) Then G is dominating
19) Endif
20) Else If Minimum Degree > Edge Connectivity
21) If there exist no Clique with degree (δ).
22) Then G is dominating
23) Endif
24) Endif
25) If above conditions does not hold go to step 26
26) If Graph (G) contains cut vertex
27) Update solution (g);
28) Disconnect G by removal of Vertex
29) Insert Sub-Graph H into Q
30) Endif
31) While ($\delta_{min}(g) \geq [N_g / 2]$) /* pre-processing */
32) Compute Vertex Connectivity κ (G);
33) Update solution (g);
34) Remove a vertex with degree $\delta_{min}(g)$;
35) End if;
36) If (H is a terminal graph) /* pruning */.
37) Calculate Vertex Connectivity κ (H);
38) Update solution (g);
39) Else
40) Calculate Vertex Connectivity κ (H);
41) Update solution (g);
42) Run min-cut algorithm on H;
43) Compute vertex connectivity κ (H)
44) Update solution (g);
45) End if
46) Return results to step 4

Explanation

Line 4 to Line 7: Here the given Graph (G) is checked for the cut vertex. If it contains a cut vertex then it is checked if the given graph is tree. If it satisfies above condition then the given graph satisfies monotonic property. Else calculate maximum degree and Minimum Degree of the Graph (G).

Line 8 to Line 12: Here if the Maximum Degree of the Graph (G) equals Minimum Degree of Graph (G) then the given Graph satisfies the monotonic property. Else calculate the Vertex Connectivity of the given graph.

Line 13 to Line 18: Here if the Maximum degree of given graph is not equal to Minimum Degree then the algorithm checks for the following condition:

If minimum degree equal to Vertex Connectivity and assigns variable n for the value of minimum degree.

Determine if there exist a clique with degree n+1, If There exist no clique with degree n+1 then the given graph satisfies monotonic property.

Line 20 top Line 23: If Minimum degree of a given graph is greater than Vertex Connectivity then check for the following condition:

Line 26 to Line 30: The Graph (G) is checked for cut vertex. If cut vertex is detected, then it is added to solution set and the graph G is further disconnected by removing cut vertex.

Line 31 to Line 35: This is pre-processing module. The Condition ($\delta_{min}(g) \geq [N_g / 2]$) is checked. If the sub-graph satisfies the condition, then the vertex connectivity of sub-graph is calculated and added to the solution set. Then the Sub-Graph is further disconnected by removing vertex of Minimum degree.

Line 36 to Line 38: This is pruning module. This is based on mechanism of detecting Tree, Cycle or Clique. If either of above is detected then the sub-graph is added to solution set by calculating its vertex connectivity.

Line 40 to Line 45: If none of the sub-graphs are detected for tree, cycle or Clique then the Min Cut Decomposition algorithm is carried out on the Sub-Graph. After every iteration the vertex connectivity of the sub-graph is calculated and added to solution set.

VI. Experimental Results

Algorithm Dominating (H) has been implemented in Java and all experiments have been performed on an Intel Core Duo at 2.2GHz, with 2GB RAM running Windows Vista. The performance evaluation study is based on real-life graph data sets.

A. Real Life Graph Data Set 1

TABLE I: The Solution Set (Dominating (*H*)) for Real Life Graph Data Set 1

Total Number of Nodes	5242
Total Number of Edges	14478
Disconnected Graphs	69
Minimum Degree	1
Maximum Degree	81
Trees Detected	82
Cycles Detected	0
Cliques Deetcted	56
Dominating (H)	138
Time Taken	26.08 Secs

B. Real Life Graph Data Set 2

TABLE II: The Solution Set (Dominating (*H*)) for Real Life Graph Data Set 2

Total Number of Nodes	1379917
Total Number of Edges	3843320
Disconnected Graphs	80
Minimum Degree	1
Maximum Degree	56
Trees Detected	198
Cycles Detected	0
Cliques Deetcted	50
Dominating (H)	44
Time Taken	46.03 Seconds

C. Real Life Graph Data Set 3

TABLE III: The Solution Set (Dominating (*H*)) for Real Life Graph Data Set 3

Total Number of Nodes	7115
Total Number of Edges	100762
Disconnected Graphs	13
Minimum Degree	1
Maximum Degree	165
Trees Detected	57
Cycles Detected	1
Cliques Deetcted	0
Dominating (H)	57
Time Taken	38.07 Seconds

VII. Performance Evaluation

The software "U-GIGA" project aims at providing Graphical User Interface for various graph mining algorithms. The software has been implemented in JAVA (jdk1.5.0 or jdk1.6.0_12) and the graphs and their corresponding attributes are displayed to the user using the JAVA applet-viewer. All the graph algorithms have been implemented in Java and all experiments have been performed on an Intel Core Duo at 2.2GHz, with 2GB RAM running Windows Vista. The software basically checks for graph parameters such as order, size, etc. that are stored or saved in a foreign source (i.e. in a text file) and using this information about the graph, various graph attributes such as vertex/edge connectivity, clique detection, time complexity, etc. are calculated and displayed on the user interface. The software aims at investigating performance by varying the graph parameters and also keeps a track on the time taken to evaluate a graph (i.e. the time complexity of the software).The various phases of the software for evaluating and displaying a particular graph scanned are as follows: Scan the text file, which the user chooses on the interface (by clicking the corresponding file link) and store the graph data to the corresponding variables. Using these variables, perform the following task and calculate the following graph attributes:

Define and store a particular path in variables, so as to draw and display the graph, using the information from these variables, on the user interface.

1) Calculate the number of bridges present in the graph
2) Calculate the edge connectivity/vertex connectivity.
3) Calculate the cliques detected.
4) Check if the given graph is a tree or not.

5) Calculate the minimum and maximum degree of the graph.
6) Check if the graph is monotonic, using information of vertex connectivity, edge connectivity, cliques, tree information.

These graph attributes are then displayed to the user.

VIII. Conclusion

In this paper we proposed a novel way to determine dominating subgraph over vertex connectivity and order constraint. We have developed an efficient algorithm Dominating (*H*) which determines whether the given graph is dominating over vertex connectivity and order constraint. Software using Java Programming Language has been designed, which assists in determining important sub-graph .It also helps in determining the value of Maximum Degree, Minimum Degree, vertex connectivity of given graph (*G*) and Determine if there exist a clique in given graph (*G*).

References

[1] D. J. Cook and L. B. Holder, *Mining graph data*, London: Wiley, 2006.
[2] J. Chuzhoy and S. Khanna, "Algorithms for single-source vertex-connectivity," in *Proc. IEEE FOCS*, October 2008, pp. 105-114.
[3] A. Hanneman and M. Riddle. Introduction to social network methods. [Online]. Available: http://www.faculty.ucr.edu/ hanneman/nettext/.
[4] D. Gibson, R. Kumar, and A. Tomkins, Discovering large dense subgraphs in massive graphs, in *Proc. the 31st VLDB conference*, 2005, pp. 721–732.
[5] H. Hu, X. Yan, Y. Huang, J. Han, and X.-J. Zhou, "Mining coherent dense subgraphs across massive biological networks for functional discovery," *Bioinformatics*, vol. 21, no. 1, pp. i213–i221, 2005.
[6] F. Zhu, X. Yan, J. Han, and P. S. Yu, "gPrune: a constraint pushing framework for graph pattern mining," in *Proc. PAKDD conference*, 2007, pp. 388–400.
[7] X. Yan, X. J. Zhou, and J. Han, "Mining closed relational graphs with connectivity constraints," in *Proc. ACM KDD conference*, 2005, pp. 324–333.
[8] Y. Mansour and B. Schieber, "Finding the edge connectivity of directed graphs," *Journal of Algorithms*, vol. 10, no. 1, pp. 76-85, 1989.
[9] S. Even, "An algorithm for determining whether the connectivity of a graph is at least k," *SIAM Journal of Computing*, vol. 4, no. 3, pp. 393-396, 1975.
[10] D. J. Kleitman, "Methods for investigating connectivity of large graphs," *IEEE Trans. Circuit Theory*, vol. 16, no. 2, pp. 232-233, 1969.
[11] E. Hartuv and R. Shamir, "A Clustering algorithm based on graph connectivity," *Inform Process Letters*, vol. 76, no. 4-6, pp. 175-181, 2000.
[12] Z. Wu and R. Leahy, "An optimal graph theoretic approach to data clustering: theory and its application to image segmentation," *IEEE Trans Pattern Anal Machine Intell*, vol. 15, no. 11, pp. 1101–1113, 1993.
[13] Z. Galil, "Finding the vertex connectivity of graphs," *SIAM Journal of Computing*, vol. 9, no. 1, pp. 197-199, 1980.
[14] Z. Galil and G. F. Italiano, "Reducing edge connectivity to vertex connectivity," *ACM SIGACT News*, vol. 22, no. 1, pp. 57-61, 1991.
[15] S. Borz sonyi, D. Kossmann, and K. Stocker, "The Skyline operator," in *Proc. the 17th international conference on data engineering*, 2001, pp. 421–430.
[16] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive skyline computation in database systems," *ACM Trans Database System*, vol. 30, no. 1, pp. 41–82, 2005.
[17] N. Papadopoulos, A. Lyritsis, and Y Manalopoulos, "SkyGraph: an algorithm for important subgraph discovery in relational graphs," *DMKD*, Springer, vol. 17, no. 1, pp. 57-76, 23 June 2008.
[18] A. A. Bhamaikar and P. R. Rao, "Detecting monotonic graph over edge connectivity constraint," *International Journal of Innovative Technology and Creative Engineering*, vol. 1, no. 12, pp. 6-12, 2011.
[19] A. A. Bhamaikar and P. R. Rao, "Mining important sub-graphs using skyline approach over vertex connectivity constraint," *CiiT International Journal of Data Mining Knowledge Engineering*, May, 2012.
[20] A. A. Bhamaikar and P. R. Rao, "Detecting cliques using degree and connectivity constraints," *International Journal of Data Mining & Knowledge Management Process*, vol. 2, no. 2, pp. 39-45, 2012.

Abhay A. Bhamaikar was born in Panaji, the Capital city of State of Goa, India. He has done his bachelors of engineering in computers and masters of engineering in internet technology from Padre Conceicao College of Engineering, Verna – Goa, India. He is pursuing his Ph.D from Department of Computer Science and Technology, Goa University, India. His major field of study is Data Mining.

He is working as assistant professor, Department of Information technology, Shree Rayeshwar Institute of Engineering and Information technology, Shivshail, Karai, Shiroda – Goa, India .My Areas of Interest are Data Mining, Graph Mining, Design and Analysis of Algorithm.

Mr. Bhamaikar is Life member of Computer Society of India.

Pralhad Ramchandra Rao is a professor and head, Department of Computer Science and Technology, Goa University, Taleigao Plateau, Goa. His areas of Interest is Data Mining, Graph Mining, Graph theory and Databases.