International Journal of Advanced Computer Research (ISSN (print): 2249-7277 ISSN (online): 2277-7970)
Volume-3 Number-2 Issue-10 June-2013

An Efficient Algorithm for Calculating Maximum Bipartite matching in a Graph

Neha Bora[1], Swati Arora[2], Nitin Arora[3]

Abstract

In this paper we proposed a new approximation algorithm for calculating the min-cut tree of an undirected edge-weighted graph. Our algorithm runs in O (V^2.logV + V^2.d), where V is the number of vertices in the given graph and d is the degree of the graph. It is a significant improvement over time complexities of existing solutions. We implemented our algorithm in objected oriented programming language and checked for many input cases. However, because of an assumption it does not produce correct result for all sorts of graphs but for the dense graphs success rate is more than 90%. Moreover in the unsuccessful cases, the deviation from actual result is very less and for most of the pairs we obtain correct values of max-flow.

Keywords

Approximation Algorithm, Max-Flow, Min-Cut, Object Oriented Programming.

1. Introduction

Graph connectivity is one of the classical subjects in graph theory and has many applications like Reliability of communication networks, cluster analysis, transportation planning, chips and circuit design. In the maximum flow problem we are given a flow network G = (V, E) which is a graph in which each edge (u, v) $\in$ E has a non-negative capacity c(u, v) $\geq$ 0. If (u, v) $\notin$ E then it is assumed that c(u, v) = 0 [1][2]. We distinguish two vertices in a flow network a source s and a sink t. In this we wish to compute the greatest rate at which material can be shipped from the source s to the sink t without violating any capacity constraints.

Neha Bora, Computer Science & Engineering Department, Uttarakhand Technical University, G. B. Pant Engineering College, Pauri, India.
Swati Arora, Computer Science & Engineering Department, Uttarakhand Technical University, G. B. Pant Engineering College, Pauri, India.
Nitin Arora, Computer Science & Engineering Department, Uttarakhand Technical University, Women Institute of Technology, Dehradun, India.

Finding the minimum cut of an undirected edge-weighted graph is the fundamental algorithmic problem. Precisely it consists in finding a non-trivial partition of graph vertex set V into two parts such that the cut weight, the sum of weights of the edges connecting the two parts is minimum. Given a graph G = (V, E) with vertex set V, edge set E and weight function w: E $\rightarrow$ R, it can be shown that there are at most n-1 distinct min-cuts among the total n(n-1)/2 pairs of nodes. We represent these n-1 min-cuts by a tree, called Min-Cut Tree, which always exists and not need to be necessarily unique and has some properties. The nodes of the tree are the same as the nodes of the initial graph, (i.e. V). Each edge is assigned a value. For every pair s, t, we can find the min-cut value by following the (unique) path between s and t in the min-cut tree. Suppose that e is the edge with minimum value on that path. Then value (e) is also the min-cut value between s and t in the initial graph G. To actually find the cut between s and t, we simply cut off the edge e of minimum value on the s-t path. The two connected subsets of nodes in the tree, also define the min-cut between s and t in the initial graph G [4][5][6].

2. Literature Survey

Ford and Fulkerson [8] shown the duality of the maximum flow and minimum s-t cut. This theorem states that the value of maximum flow in a flow network G with source s and sink t is equal to the value of minimum s-t cut of Graph G.

In 1961, Gomory and Hu [7] shown that in a Graph having n nodes, there can be only n-1 numerically different flows. They proposed a method to compute min-cut tree by computing only n-1 minimum s-t cuts. In 1997, M. Stoer and F. Wagner [3] presented an algorithm for finding the min-cut tree of an undirected edge-weighted graph without using any flow techniques. This algorithm is one of a small number of papers treating questions of graph connectivity by non-flow-based methods. Time complexity of this algorithm is much better than those of flow based algorithms.

3. Novel Approximation Algorithm

We present a new approximation algorithm for constructing the minimum cut tree. We calculate an upper-bound value for each node in the graph. We define the upperbound value of each node as the value of cut which separates this node from rest of the graph. We used the Lemma: The value of minimum cut of a graph G separating N_i and N_j is less than or equal to minimum of the upperbound values of two nodes N_i and N_j. We proceed by finding an edge uv such that upon merging the two nodes N_u and N_v we are able to reduce the upperbound value of the new node, i.e.

$$\text{upperbound}(N_u) + \text{upperbound}(N_v) - 2*w(u,v) \leq \max(\text{upperbound}(N_u), \text{upperbound}(N_v))$$

We start from the node having the minimum upperbound value and check for all of the edges leaving it. If we are able to reduce the upperbound value by merging it with any of the nodes, we merge the nodes and repeat the same procedure. If we are not able to reduce the upperbound value of node, we check for rest of the nodes in the increasing order of upperbound value. If at any stage it is not possible to merge any node, then we merge that pair of nodes which results in minimum increment of the upperbound value. After all the nodes in the graph are merged and it has only one node left, we proceed to construct the min-cut tree by using the information from intermediate stages. We move from last to first stage and at each stage we see the two nodes that were merged during last stage and separate the node with smaller of the two upperbound values from the other by an arc bearing the value equal to the smaller of the two upperbound values. Since we are considering the nodes in the increasing order of upperbound values, checking for N_i itself implies that N_j has already been checked and it was not possible to reduce its upperbound value at all. So in this case upperbound(N_j) cannot be reduced.

Our algorithm is based on the assumption that if we are merging two nodes N_i and N_j and if upperbound(N_i) < upperbound(N_j) then it is not possible to merge N_j with any other node which will result in a node having upperbound value which is less than upperbound(Ni).

After running the procedure with more than 20000 randomly generated graphs we have figured out that for graphs having density >= 0.4, success rate of algorithm is more than 90%. Moreover in the unsuccessful cases, the deviation from actual result is very less (usually for less than 5% pairs) and for most of the pairs we obtain correct values of max-flow.

```
Procedure: Min-Cut Tree(G)
Input: Undirected edge-weighted graph G
Output: Min-Cut Tree
Calculate the upperbound values for each node.
while(number of vertices in the current graph > 1)
    loop(Consider the vertices in the increasing order of upperbound value)
        if(upperbound value can be reduced by merging a node with any adjacent node)
        then merge those two adjacent nodes
            break;
        End if
    End loop
 if (it is not possible to merge any pair of nodes)
 then merge the pair of nodes which results in minimum increment of the upperbound value.
End if
End While
Construct Min-Cut Tree T by using the information from intermediate stages as described:
Move from last to first stage.
At each stage check the two nodes that were merged during last stage.
Separate the node with lower upperbound value from the other by an arc bearing the value equal to the lower upperbound value.
return T
```

Time Complexity of our algorithms is $O(V^2.\log V + V^2.d)$, where V is the number of vertices in the given graph and d is the degree of the graph. This is an improvement over the best existing $O(V^4)$ solution for minimum cut tree problem.

4. Snapshots and Results

In the following figures we have shown the steps used in our efficient algorithm by taking some Undirected edge-weighted graph G. We start from the node having the minimum upperbound value and check for all of the edges leaving it. If we are able to reduce the upperbound value by merging it with any of the nodes, we merge the nodes and repeat the same procedure. If we are not able to reduce the upperbound value of node, we check for rest of the nodes in the increasing order of upperbound value.
If at any stage it is not possible to merge any node, then we merge that pair of nodes which results in minimum increment of the upperbound value.

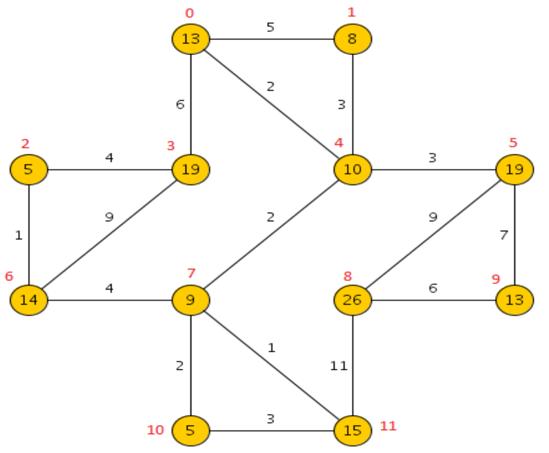

Figure 1.1: Input Undirected edge-weighted graph G

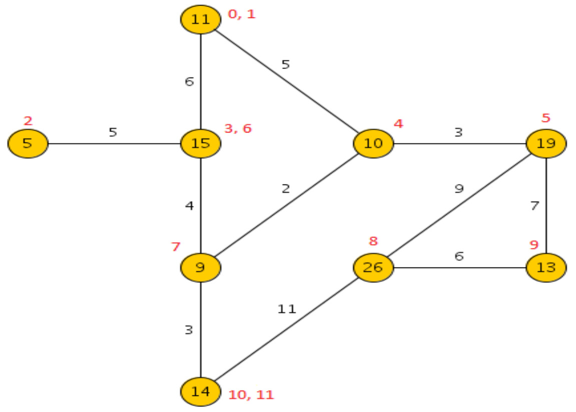

Figure 1.4: Graph G after merging the node 3 and node 6

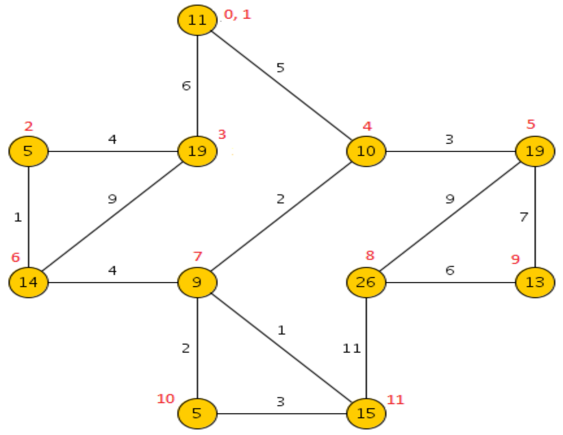

Figure 1.2: Graph G after merging the node 0 and node 1

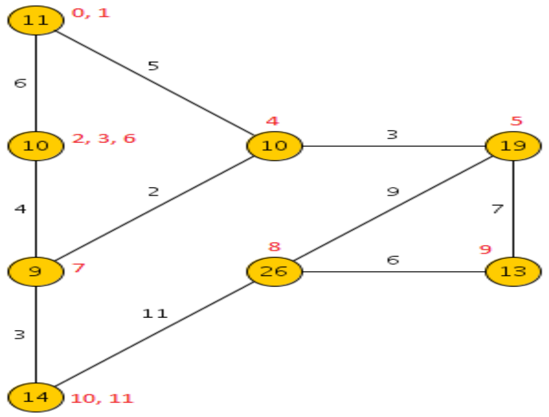

Figure 1.5: Graph G after merging the node 2 and node 3, 6

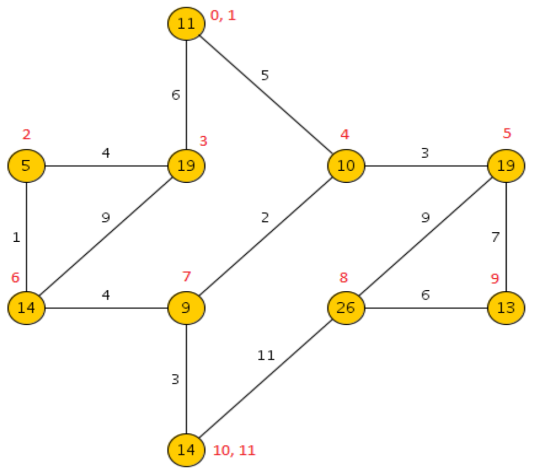

Figure 1.3: Graph G after merging the node 10 and node 11

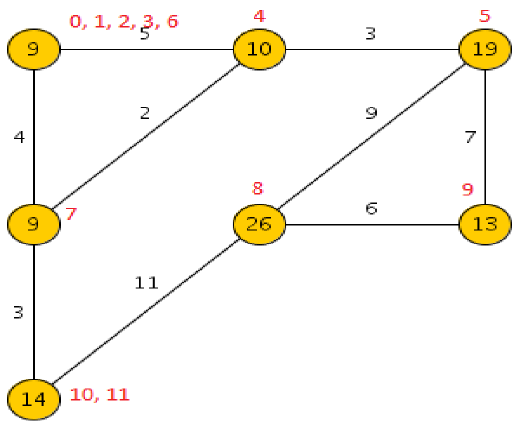

Figure 1.6: Graph G merging the node 0, 1 and node 2, 3, 6

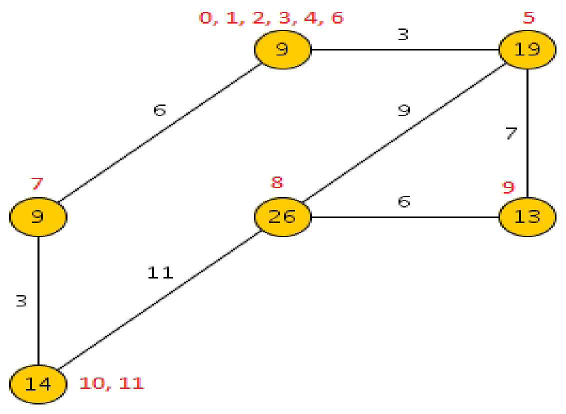

Figure 1.7: Graph G after merging the node 0, 1, 2, 3, 6 and node 4

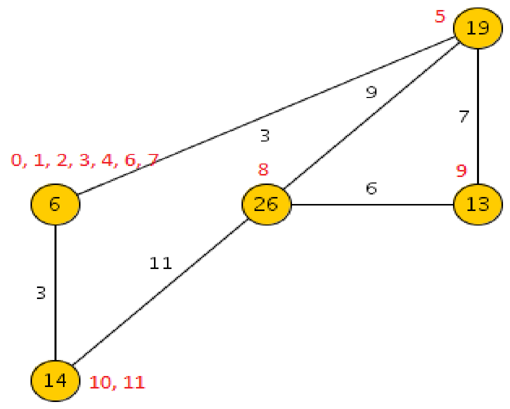

Figure 1.8: Graph G after merging the node 0, 1, 2, 3, 4, 6 and node 7

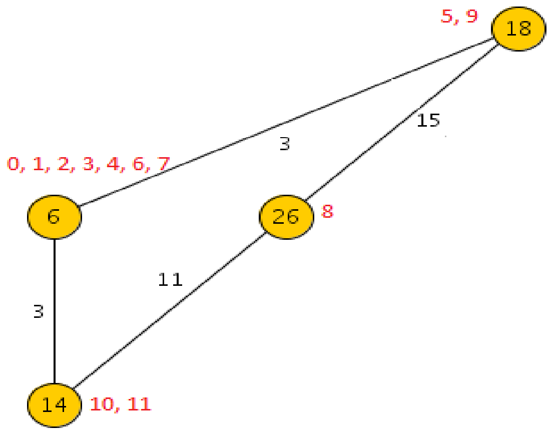

Figure 1.9: Graph G after merging the node 5 and node 9

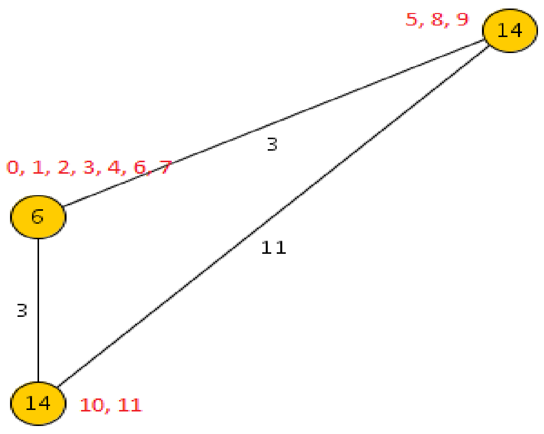

Figure 1.10: Graph G after merging the node 5, 9 and node 8

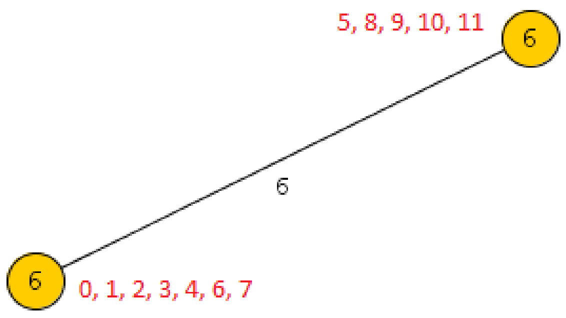

Figure 1.11: Graph G after merging the node 5, 8, 9 and node 10, 11

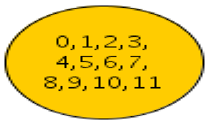

Figure 1.12: Graph G after merging all the nodes

After all the nodes in the graph are merged and it has only one node left, we proceed to construct the min-cut tree by using the information from intermediate stages.

We move from last to first stage and at each stage we see the two nodes that were merged during last stage and separate the node with smaller of the two upper bound values from the other by an arc bearing the value equal to the smaller of the two upper bound values. Since we are considering the nodes in the increasing order of upper bound values, checking for N_i itself implies that N_j has already been checked and it was not possible to reduce its upper bound value at all. So in this case upperbound(N_j) cannot be reduced.

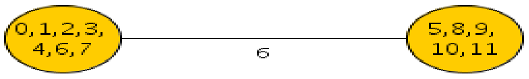

Figure 2.1: Partial Min-Cut tree after separating the nodes 0, 1, 2, 3, 4, 6, 7 and node 5, 8, 9, 10, 11

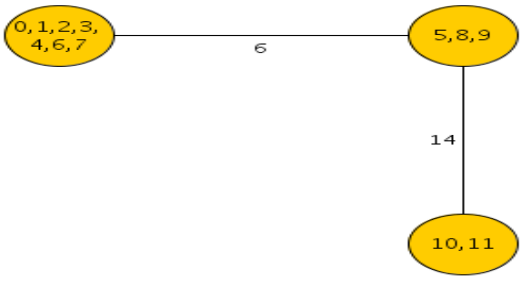

Figure 2.2: Partial Min-Cut tree after separating the node 5, 8, 9 and node 10, 11

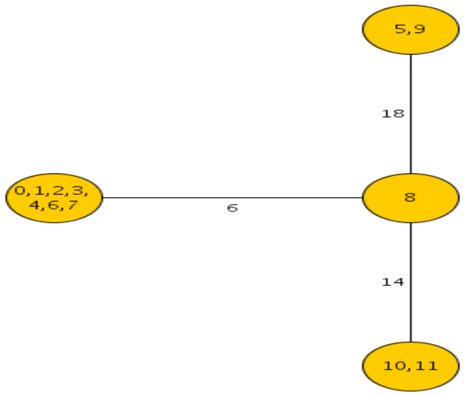

Figure 2.3: Partial Min-Cut tree after separating the node 5, 9 and node 8

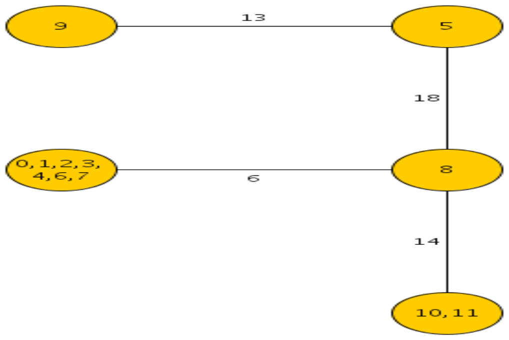

Figure 2.4: Partial Min-Cut tree after separating the node 0, 1, 2, 3, 4, 6 and node 7

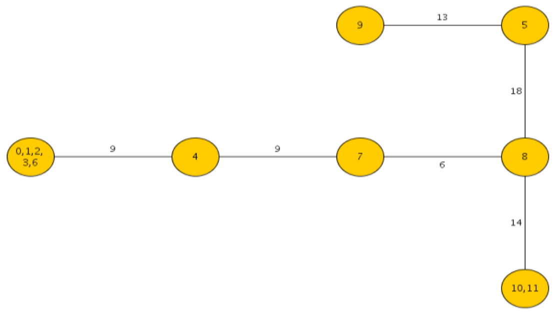

Figure 2.5: Partial Min-Cut tree after separating the node 0, 1, 2, 3, 6 and node 4

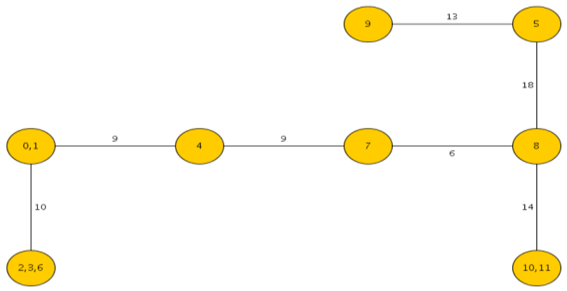

Figure 2.6: Partial Min-Cut tree after separating the node 0, 1 and node 2, 3, 6

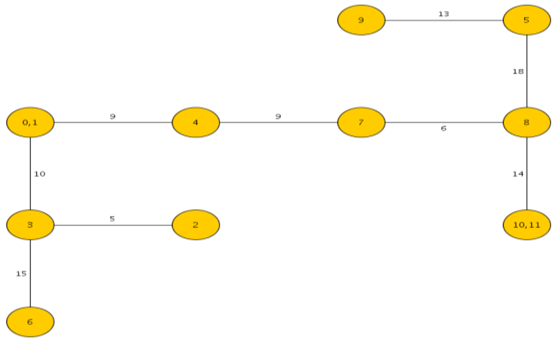

Figure 2.7: Partial Min-Cut tree after separating the node 2, node 3 and node 6

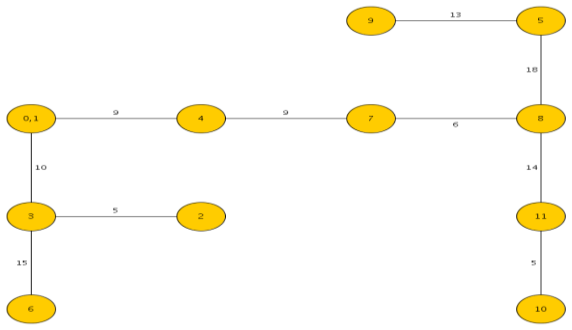

Figure 2.8: Partial Min-Cut tree after separating the node 10 and node 11

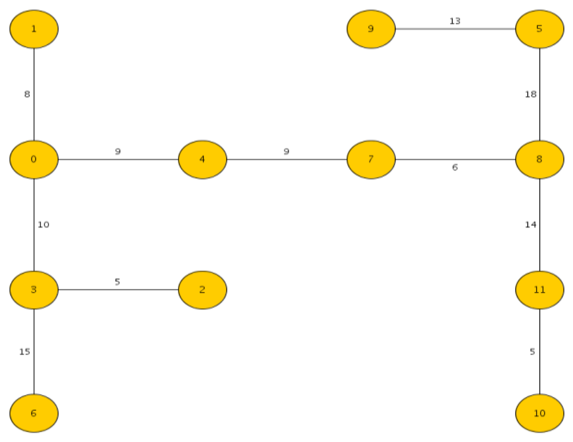

Figure 2.9: Min-Cut tree after separating the node 0 and node 1

We generated 7500 random graphs of different densities but having fixed number of nodes. Edge-weights were also random and were between 1-300. Results of running our algorithm with these graphs are summarized in following plots:

Random Graphs with fixed number of nodes

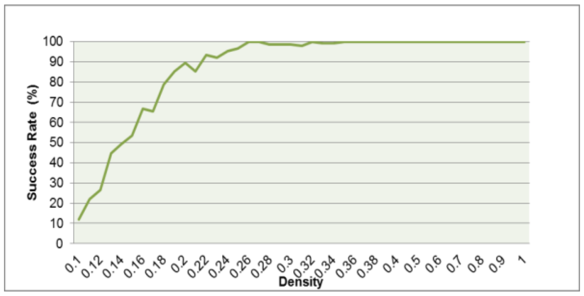

Figure 3: Plot of Success Rate Vs Density (Number of nodes were fixed to 50)

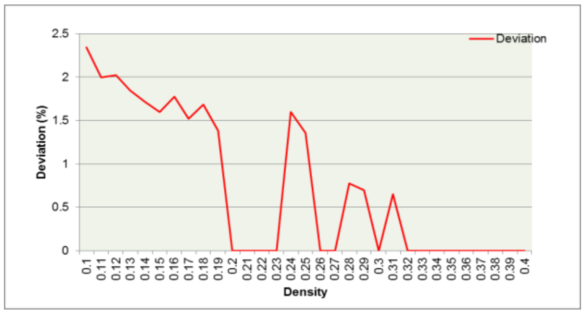

Figure 4: Plot of Deviation Vs Density (For unsuccessful test cases)

It is clear from figure 3 that for desity >= 0.4 success rate is about 100%. Figure 4 says that for the unsuccessful test cases deviation from the actual result is less than 3%. It means that even in the case of failure we get correct valus of max-flows or min-cuts for most of the pair of nodes.

Random Graphs with Random number of nodes

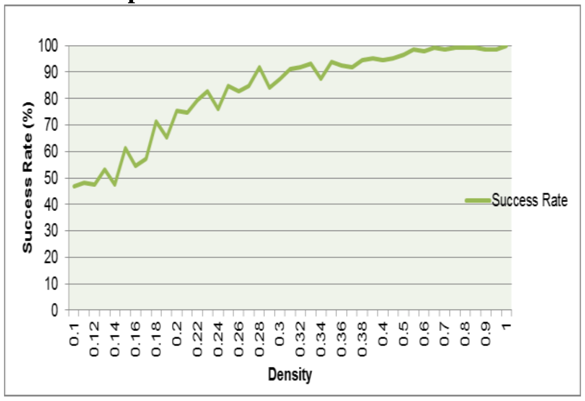

Figure 5: Plot of Success Rate Vs Density (Number of nodes were random 5-55)

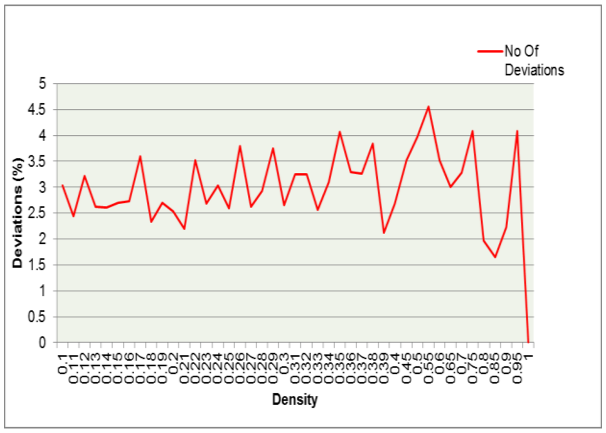

Figure 6: Plot of Deviation Vs Density (For unsuccessful test cases)

It is clear from figure 5 that for density >= 0.4 success rate is more than 92%. Figure 6 says that for the unsuccessful test cases deviation from the actual result is less than 5%. It means that even in the case of failure we get correct valus of max-flows or min-cuts for most of the pair of nodes.

5. Conclusion and Future Work

This algorithm runs in O (V^2.logV + V^2.d), where V

is the number of vertices in the given graph and d is the degree of the graph. It is a significant improvement over time complexities of existing solutions. However, because of an assumption it does not produce correct result for all sorts of graphs but for the dense graphs success rate is more than 90%. Moreover in the unsuccessful cases, the deviation from actual result is very less and for most of the pairs we obtain correct values of max-flow.
In future this algorithm can be further improved for giving the best result for all the input cases.

References

[1] Arora N., Kaushik P. K. and Singh S. P., "A Survey on Methods for finding Min-Cut Tree", International Journal of Computer Applications (IJCA), New York, Vol. 66, pp. 18-22, March 2013.
[2] Kumar A, Singh S. P. and Arora N., "A New Technique for Finding Min-Cut Tree", International Journal of Computer Applications (IJCA), New York, Vol. 69, pp. 1-7, May 2013.
[3] Stoer M. and Wagner F., "A Simple Min-Cut Algorithm", Journal of the ACM (JACM), Vol. 44, issue 4, pp. 585-591, 1997.
[4] Brinkrneier M. "A Simple and Fast Min-Cut Algorithm", Theory of Computing Systems, Vol. 41, issue 2, pp. 369-380, 2007.
[5] Hu T. C., "Optimum Communication Spanning Trees", SIAM J. Computing, Vol. 3, issue 3, 1974.
[6] Flake G. W., Tarjan R. E. and Tsioutsiouliklis K., "Graph Clustring and Minimum Cut Trees, Internet Mathematics, Vol. 1, issue 4, pp. 385-408, 2004.
[7] Gomory, Ralph E., and Tien Chung Hu. "Multi-terminal network flows." Journal of the Society for Industrial & Applied Mathematics 9, no. 4 (1961): 551-570.
[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, "Introduction to Algorithms", MIT Press, PHI, India, pp. 643-663, 2003.

Nitin Arora received B. Tech. (Computer Science & Engineering) from MNNIT Allahabad in 2008 and M. Tech. (Computer Science & Engineering) from Govind Ballabh Pant Engineering College, Pauri, Garhwal Uttarakhand in 2012. He is the member of IANEG (USA), ISOC (USA) and IACSIT (USA) and has published over several research papers in National and International journals/conferences in the field of Mobile Ad-Hoc Networks, Data Structures and Algorithms. He started his career as a Lecturer from Shobhit Institute of Engineering and Technology, Saharanpur and later on promoted as an Assistant Professor in the Department of Computer Science & Engineering at Women's Institute of Technology (WIT) Constituent college of Uttarakhand Technical University (UTU), Dehradun. He finished his M. Tech. from Govind Ballabh Pant Engineering College, Pauri, Garhwal, Uttarakhand in the field of Data Structures and Algorithms Design under Ministry of HRD, Government of India fellowship. Currently he is working as an Assistant Professor in the Department of Computer Science & Engineering at Women's Institute of Technology (WIT) Constituent College of Uttarakhand Technical University (UTU), Dehradun.