Find out the vertices contributing to vertex connectivity

Shiv Pratap Singh, Nazish Tabassum, Arqum Ahmad

Abstract—The vertex connectivity K(G) of a graph G is the minimum number of nodes whose deletion disconnects it. Vertex connectivity is sometimes called ”point connectivity” or simply ”connectivity.”

A graph with $k > 0$ is said to be connected, a graph with k=2 is said to be biconnected and in general, a graph with vertex connectivity k is said to be k-connected.

Index Terms—

I. Motivation

This problem is used in telephone lines or railway networks to analyse which particular node will cause how much damage if the network flow from that node has been forced to stop.For e.g In the context of telephone networks, the vertex connectivity is the smallest number of switching stations that a terrorist must bomb in order to separate the network.Thus, priority to save that particular stations are more than any other stations.

II. Algorithm

[utf8]inputenc

A. Brute Force Approach:

Input : A graph G with cardinality equal to n.
Output : A set of vertices S whose size is equal to vertex connectivity of graph G and it contains all vertices enlisted in vertex connectivity.

```
FindVerticesInVertexConnectivity(G)
1    RemoveSelfLoop
2    RemoveParallelEdges
3    S = emptySet
4    If graph is disconnected or n = 1
5      return S
6    If graph is complete
7      Select any n-1 vertices and add it to S
8      Return S
9    For i = 1 to (n-2)
10     Generate every combination of vertices of size i
11     remove those vertices and add it to set S
12     If graph is disconnected
13        return S
14     empty set S
```

B. Max-Flow Approach:

Input : A graph G with cardinality equal to n.
Output : A set of vertices S whose size is equal to vertex connectivity of graph G and it contains all vertices enlisted in vertex connectivity.

```
FindVerticesInVertexConnectivity(G)
1    RemoveSelfLoop
2    RemoveParallelEdges
3    S = emptySet
4    If graph is disconnected or n = 1
5      return S
6    If graph is complete
7      Select any n-1 vertices and add it to S
8      Return S
9    Replace each edge (x, y) ∈ E with arcs (x, y) and
     (y, x), and call the resulting digraph D.
10   For each pair v, w in G which are non-adjacent
11      For each vertex u other than v and w in G,
        replace u with two new vertices u1 and u2, and
        then add the new arc (u1, u2). Connect all the
        arcs that were coming to u in G to u1, and
        similarly, connect allthe arcs that were going
        out of u in G to u2 in D.
12      Assign v as the source vertex and w as the sink
        vertex.
13      Assign the capacity of each arc to 1, and call
        the resulting network H.
14      T = ComputeMinCutUsingMaxFlow(H, v, w)
15      if (|T|< |S|)
16        S = T
17      Restore network H back to D.
18   return S
```

```
ComputeMinCutUsingMaxFlow(H, v, w)
1  Run Ford-Fulkerson algorithm and consider the final
   residual graph
2  Find the set of vertices that are reachable from
   source in the residual graph.
3  All edges which are from a reachable vertex to non-
   reachable vertex are minimum cut edges.
4  Find the corresponding vertices to edges all the edges
   that form minimum cut edges.
5  Add all those vertices to Set T
6  Return T
```

III. Description

Algorithm 1 is simple brute force algorithm with tries all the combinations.

For algorithm 2 approach is as follow :-

The vertex-connectivity K (G) of a graph G = (V,E) is the least cardinality |S|of a vertex set $S \subset V$ such that G −S is either disconnected or trivial. Such a set S is called a minimum vertex-cut. We will first explain how the computation of K(G) reduces to solving a number of max-flow problems. Let v and w be two vertices in graph G = (V,E). If vw $\notin$ E, we define k(v, w) as the least number of vertices, chosen from V −{v, w}, whose deletion from G would destroy every path between v and w, and if vw $\in$ E then let k(v, w) = n −1, where n is the order of the graph. Clearly (G) can be expressed in terms of k(v, w) as follows:

K(G) = min{ k(v, w) unordered pair v , w in G }.

So when we find K(G) it gives us the list of vertex in vertex connectivity of Graph(G).

IV. Implementation and Results

The algorithm has been implemented in c++ language. Test case are generated manually using c++ library. For particular input size running time of input is calculated. The tool used for plotting graph is gnuplot.

Time complexity of first algorithm is exponential

Time complexity of second algorithm is O(Nˆ3*E)

A. Plot of time taken vs input size

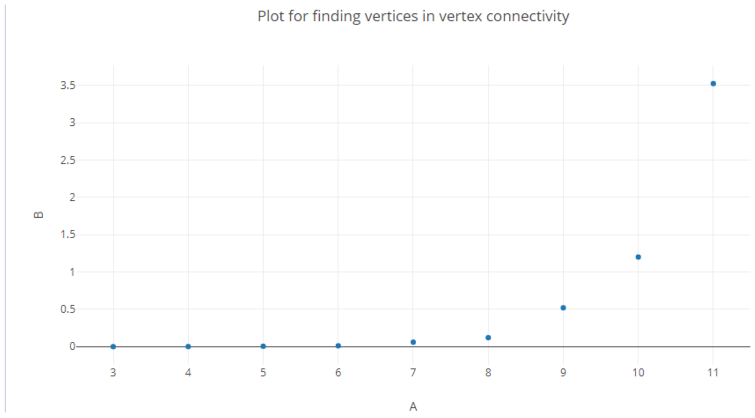

Fig. 1. Algorithm1 graph

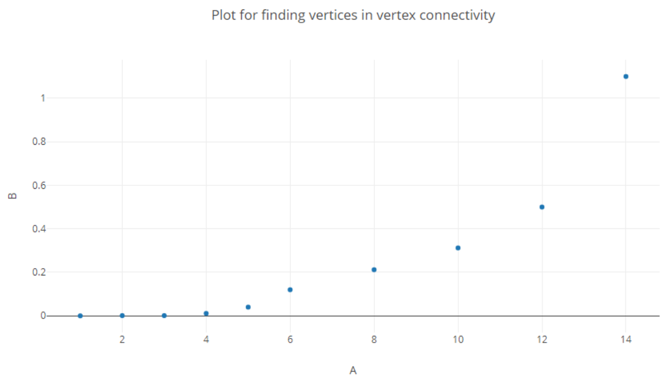

Fig. 2. Algorithm2 graph

V. Conclusion

Time complexity of first algorithm is exponential because this algorithm runs for every combination of size i where i $\in$ (1, n) Time complexity of second algorithm is O(Nˆ3*E) where N is cardinality of G and E is no. of edges in G.

VI. Refrences

[1]$shttp : //mathworld.wolfram.com/VertexConnectivity.html(for$

[2]$https : //www.cse.msu.edu/\ cse835/Papers/Graph_connectivity_re$