A general measure of Betweenness

Aravind Suresh
EE14B013
Dept. of Electrical Engineering
Indian Institute of Technology, Madras

***Abstract*—Betweenness centrality is a measure of centrality in a graph based on shortest paths. For every pair of vertices in a graph, there exists a shortest path such that either the number of edges between them (undirected graph) or the sum of the weights of the edges (directed graphs) is minimized. Betweenness centrality of a vertex depends on how many of these shortest paths pass through that particular "vertex". It is a measure of how important that node is, to the entire network.**

***Index Terms*—Complex Network Analysis, Centrality metrics, Betweenness centrality, Graphs**

I. Introduction

A graph is represented by a (V, E) where V denotes the set of vertices and E denotes the set of edges. An edge is a link between 2 vertices. Graphs can be weighted or unweighted depending on whether the edges have a weight associated with them or not. If the edges have a directional significance, such a graph is called a directed graph. Otherwise it is referred to as an undirected graph. Undirected graphs can be represented as a directed graph but with an extra edge (v, u) for every directed edge (u, v).

II. Algorithm

To compute betweenness centrality, we need to identify the shortest paths between all pairs of vertices. The implementation differs on whether the graph is weighted (Johnson's algorithm) or unweighted (Brande's algorithm). For an unweighted graph, shortest paths from a single source can be found using Breadth-first search algorithm in O(V + E) time. For a weighted graph, shortest paths from a single source can be found using Dijkstra's algorithm in O(E + VlogV) time. Since, we need to identify shortest paths between all pairs of vertices the overall complexity is $O(VE + V^2)$ for unweighted graphs and $O(VE + V^2 \log V)$ for weighted graphs.

III. Features

Type	Weighted	Unweighted
Directed	YES	YES
Undirected	YES	YES

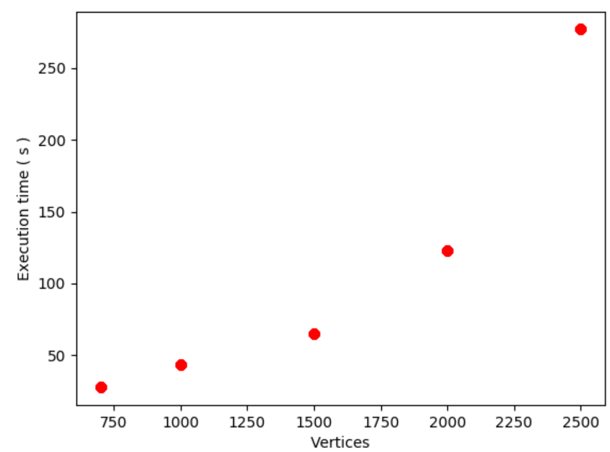
- The algorithm takes the graph as the input (number of vertices and edges) and the weights for the different vertex-pairs. It is implemented in C++ and the graph-generator used is implemented in Python. The implementation is available here.
- The algorithm considers all shortest paths of equal lengths. That is, if there are multiple paths of same total weight (edge count) all those paths are considered in calculating betweenness centrality.
- The shortest path contributions for a given node are weighted with the prior weight provided and the result is normalised with the sum of total weights excluding vertex pairs involving the given node.

Type	Memory	Time
Directed-Weighted	$O(V^2 + E)$	$O(VE + V^2 \log V)$
Directed-unweighted	$O(V^2 + E)$	$O(VE + V^2)$
Undirected-Weighted	$O(V^2 + E)$	$O(VE + V^2 \log V)$
Undirected-unweighted	$O(V^2 + E)$	$O(VE + V^2)$

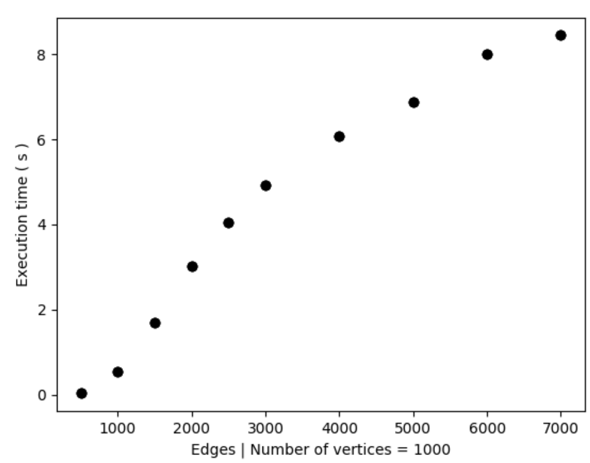
IV. Analysis

Using a graph generator, random graphs were generated with various numbers of vertices and edges. The algorithm was executed on those graphs and the results are summarised below.

- Firstly different graphs were generated with different number of vertices from 700 to 2500. The number of edges was proportional to V i.e E/V was nearly constant (= 40). The plot is shown below.
- We have a figure almost like a parabola thus showing the quadratic dependence of time complexity on the number of vertices in the graph.



- Different graphs were generated with different number of edges between 500 and 7000 for a fixed number of vertices (V = 1000). The plot is shown below.
- The points almost lie on a straight line and thus prove the linear dependence of execution time on the number of edges in the graph.



References

[1] Networks: An Introduction, M. E. J. Newman, Oxford University Press, 2010.
[2] A Faster Algorithm for Betweenness Centrality - Algorithmics
[3] Betweenness centrality - Wikipedia