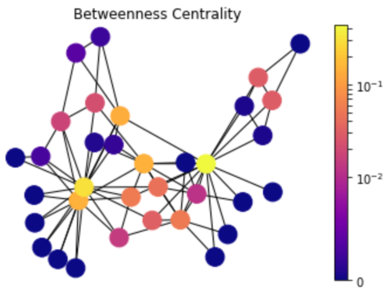
Note on Betweenness Centrality

Shambhavi Suryanarayanan, Elizaveta Rebrova

October 2022

Betweenness centrality is a widely used measure that captures **an importance of a vertex in a network in allowing information to pass from one part of the network to the other**. It is defined based on the amount of shortest paths within the network that has to pass through the given vertex. Informally, network "bottlenecks" (including bridges and local bridges) have higher betweenness values. The picture below is due to Can Güney Aksakalli.



The goal of this note is to review the formal definition and ways to calculate the betweenness centrality.

1 Definition of betweenness Centrality

Definition 1.1 (Betweenness Centrality)**.** Consider a graph $G = (V, E)$. For $s, t, v \in V$ let $\sigma_{s,t}$ denote the number of shortest paths from s to t and $\sigma_{s,t}(v)$ denote the number of shortest paths from s to t that pass through v. Then, the betweenness of node v, denoted by $h(v)$ is given by:

$$h(v) := \frac{1}{2} \sum_{s,t \in V\, s,t \neq v} \frac{\sigma_{s,t}(v)}{\sigma_{s,t}}.$$

- For the betweenness of a node v, we do not consider paths that originate or end at v.
- The factor of $\frac{1}{2}$ appears because we if we are considering a path p from s to t, we would be accounting for it twice, once in $\sigma_{s,t}$ and again in $\sigma_{t,s}$. Instead, one can as well only consider unordered pairs (s, t) from V. *Overall, in this definition, each pair of vertices s and t (so that both s and t are distinct from v) should be considered once.*
- The definition above is consistent with the examples we calculated in class, see also the example below.

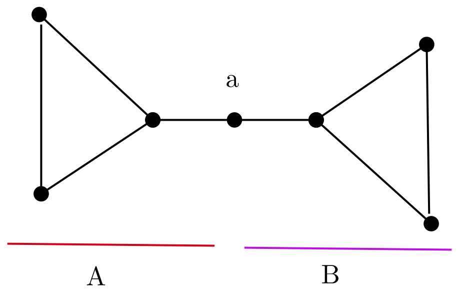
  However, in literature, it is typical to normalize the definition of betweenness in other ways accounting for the total size of the network (note that the values of $h(v)$ tend to be larger not only in reflection of the comparatively important role of a vertex, but also in reflection of the total network size). Two alternative popular ways to rescale the betweenness value are

1. to define $h_1(v)$ by dividing $h(v)$ through by the number of pairs of nodes inn the network not including v (so, by $(n-1)(n-2)/2$ for undirected networks wih n vertices). This way, $h_1(v) \in [0,1]$ for all v. But the values of $h_1(v)$ can be very small in large sparse netwoks (and real-life networks tend to be very sparse!)
2. to normalize based on the maximal and minimal values of $h(v)$, that is
$$h_2(v) := \frac{h(v) - \min_v h(v)}{\max_v h(v) - \min_v h(v)}$$
Check that in this case we also ensure that $h_2(v) \in [0,1]$ for any v and we do not shrink the range of values of the betweenness function as much as in the defnition of $h_1(v)$.

Finally, note that rescalings do not matter if you only need to compare the betweennesses of different vertices within one network (like in HW2 Problem 1). For the exams, you only need to know the Definition 1.1 of $h(v)$, not other possible ways to rescale it.

- The function for computing betweenness in the NetworkX package in python uses a slightly different definition of betweeness than what we consider. Pre-defined functions for computing centrality measures cannot be used in the exams.

2 Example



Let's consider the betweenness of the node a. Let the nodes on the left of a, be denoted by A and those on the right by B.

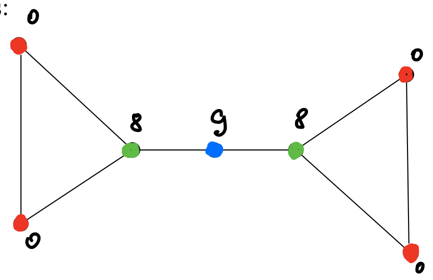
For any pairs of nodes s,t in A, none of the shortest paths would pass through a. Hence, $\sigma_{s,t}(a) = 0$. This similarly holds for pairs of node in B.

Now consider a node $s \in A$ and $t \in B$. All of the shortest paths from s,t would have to pass through v. As removing a would make A and B disconnected. We can also note that here, all the shortest paths will be unique. Thus $\sigma_{s,t}(a) = \sigma_{s,t}$. Similar argument holds for $s \in B$ and $t \in A$.

Hence,

$$h(a) = \frac{1}{2}\left(\sum_{s\in A.t\in B} \frac{\sigma_{s,t}(a)}{\sigma_{s,t}} + \sum_{s\in A,t\in B} \frac{\sigma_{s,t}(a)}{\sigma_{s,t}}\right) = \frac{1}{2}\left(\sum_{s\in A.t\in B} 1 + \sum_{s\in A,t\in B} 1\right) = 3 \times 3 = 9$$

Similarly, all node betwennesses:



Note that this is only a minimal example to illustrate the normalization factor that we use. For less trivial examples see lecture notes and the textbook.

3 Calculating betweenness centrality (for edges)

For small networks, it is enough to use Definition 1.1 to compute betweenness centrality. If you deal with a large network, one needs to find *all* shortest paths between *all* vertices, and this can be too slow. A modification of the BFS (breadth first search) algorithm can address this issue. Its version computing *edge betweenness* can be found in the textbook in Section 3.6 B (do not worry about it for the midterm).

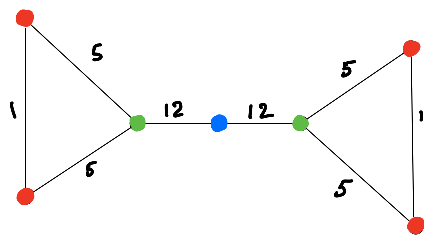
3.1 Edge betweennes by definition

The betweeness centrality measure for the edge e can be defined as

$$h_e(v) := \frac{1}{2} \sum_{s,t \in V} \frac{\sigma_{s,t}(e)}{\sigma_{s,t}}$$

with the same agreement on normalization (each shortest path in the graph is computed once). Note, that the endpoints of an edge e are also the part of the count. By the way, it implies that the edge betweennes os strictly positive for every edge (while the node betweenness can be zero).

Here are the edge betweennesses for the same graph as above:



Indeed, a red-red edge is only included in a short path between the two closest red vertices; red-green edges are in all shortest paths from their red endpoint to 5 other vertices; and the computation for blue-green edges is similar to the node betweenness above: we count all the paths between three nodes on one side and 4 nodes on the other side of this edge ($3 \cdot 4 = 12$).

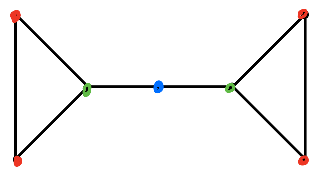
3.2 Edge betweennes by a modificaion of the BFS algorithm

The following algorithm is described on the pages 80-81 of the Easley&Kleinberg textbook.

1. Startinng from every vertex,
   (a) Construct the BFS (breadth first search) tree,
   (b) In the tree, compute *potentials of all vertices* as numbers of (shortest) paths going strictly down from the root to a given vertex. This is worked out top to bottom, for example, the vertices in the first layer of the BFS tree have potentials 1.
   (c) Then, compute *edge flows* as the number of units needed to bring 1 unit of flow to each vertex on the bottom layer of the BFS tree while leaving 1 unit of flow at every vertex along the downward paths. This is worked out bottom to top, the flow splits between the incoming (from the top) edges proportionally to the previous vertex potentials. For the horizontal edges, the flow is zero.
2. Add up lows found on Step 1 for every edge (over all BFS trees) to almost get betweenness measure for each edge.
3. Divide all the numbers by 2 to remove double-counting paths from s to t and from t to s.

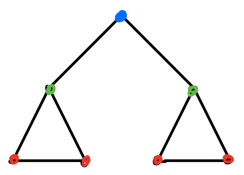
Scroll to the next page for the same example as above computed via the BFS approach.

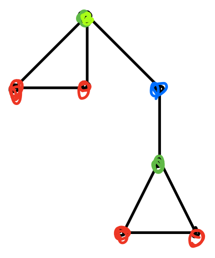
The example from the previous page via BFS flows:

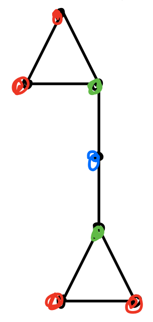


BFS trees:

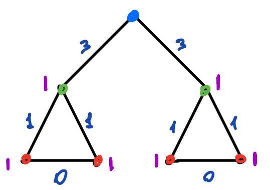
1 tree ↴ 2 trees ↴ 4 trees ↴



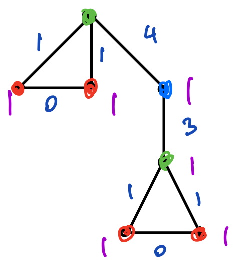




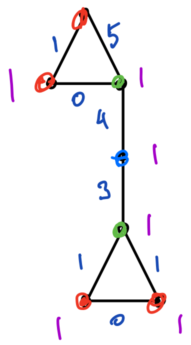
with potentials and flows:



case 1



case 2 (2 trees)



case 3 (4 trees)

Final computations:

For 🔵—🟢 : $\frac{1}{2}\Big[\overset{\text{case 1}}{3} + \overset{\text{case 2}}{(4+3)} + \overset{\text{case 3}}{(4+4+3+3)}\Big] = \frac{1}{2}\big[12+12\big] = 12$

For 🔴—🟢 : $\frac{1}{2}\Big[\overset{\text{case 1}}{1} + \overset{\text{case 2}}{(1+1)} + \overset{\text{case 3}}{(1+5+1+0)}\Big] = \frac{1}{2}\big[5+5\big] = 5$

For 🔴—🔴 : $\frac{1}{2}\Big[\overset{\text{case 1}}{0} + \overset{\text{case 2}}{0} + \overset{\text{case 3}}{(1+1+0+0)}\Big] = \frac{1}{2}\big[2\big] = 1$