Homework 2 Problem 2

Due Feb 20 by 11pm **Points** 26 **Submitting** a file upload
File Types py, pdf, and doc **Available** Jan 27 at 4:10pm - Feb 20 at 11pm 24 days

This assignment was locked Feb 20 at 11pm.

Shortest-path betweenness of edges: BFS revisited [26 pts]

The *betweenness centrality* of an edge e, denoted by x_e, is defined as
$x_e = \sum_{\{s,t\} \subseteq V} \frac{n_{st}^e}{n_{st}}$,
where n_{st}^e is the total number of shortest paths between nodes s and t that pass through edge e, n_{st} is the total number of shortest paths between nodes s and t, and the summation is taken over every two distinct nodes that are connected by a path. In this problem, we will revisit Algorithm **BFS** and its implementation so that we can use it to efficiently compute the betweenness of all edges in a graph g.

Section 3.6B in the book **Networks, Crowds, and Markets** (https://www.cs.cornell.edu/home/kleinber/networks-book/) shows how to compute the edge betweenness values. We first must compute the *flow* through the graph from each source node, i, to all other nodes in the graph. As the book describes, the flow of an edge is a measure of the number of shortest paths from node i that flow across that edge. The book provides an English description of the algorithm to compute the flow from a node and an example in Figure 3.20 that shows the flow values from node A in the graph.

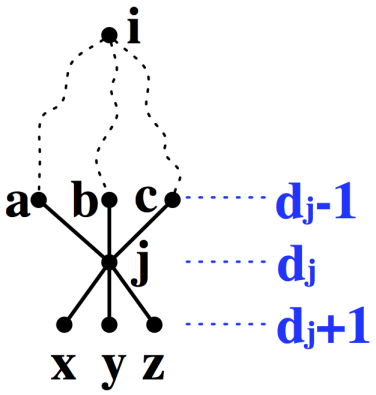

Figure 1. Computing flow from the perspective of node j, assuming nodes a, b, and c are closer than j to the initial node i, and nodes x, y, and z are farther than j from the initial node i.

In addition to reading the book description (which you should definitely do), Figure 1 will help illustrate how the **ComputeFlow** algorithm works. In the figure, node j is distance d_j from the source node i. Node j is representative of any node in the graph. In general, node j will have 0 or more neighbors that are one step closer to i (nodes a, b, and c in the figure) and 0 or more neighbors that are one step further from i (nodes x, y, and z in the figure).

For each node, j, in the graph, the total amount of flow that arrives at the node, f_j is the sum of the flow along all of the edges between j and nodes with distance $d_j - 1$ from the source node, i. If f_{mn} denotes the flow over edge $\{m, n\}$ in the graph, then in Figure 1, the total amount of flow arriving at node j is $f_j = f_{aj} + f_{bj} + f_{cj}$.

Furthermore, the total amount of flow that arrives at any node, j, must be equal to the total amount of flow that leaves node j (over edges to neighbors further from i) plus one. The additional unit of flow represents the paths that terminate at j. Therefore, for node j in the figure, the following equation must hold:

$$f_j = f_{aj} + f_{bj} + f_{cj} = 1 + f_{jx} + f_{jy} + f_{jz}$$

It is easiest to calculate this flow *backwards*. In other words, if you know f_{jx}, f_{jy}, and f_{jz} you can compute the flow that must leave node j by summing these three flows. The flow that arrives at node j is this sum plus 1, which is the right side of the equation above. You then divide this flow up among the incoming edges weighted by the number of paths that go through that node. In general, the flow along an incoming edge $\{k, j\}$ is $f_j \times n_k / n_j$. So, for instance, $f_{aj} = f_j \times n_a / n_j$.

If you start with the set of nodes that are the maximum distance, $maxd$ (*e.g.*, $\{K\}$ in Figure 3.20 in the book), away from the source node, i (*e.g.*, node A in Figure 3.20 in the book), there is no outgoing flow. So, f_j for those nodes is simply 1. You divide this flow among the incoming edges according to the number of shortest paths through the connected nodes. Then you can consider the set of nodes that are $maxd - 1$ away from the source, then $maxd - 2$ away, and so on. Finally, you will arrive at the source node and every reachable edge will have been allocated the appropriate flow value.

1. In algorithm **BFS** on Homework 1, we made use of the variable d_j, to denote the distance of node j from the initial node i. Now, we will define a new variable n_j for every node j in the graph, whose values will be computed in a modified algorithm **BFS**, starting from node i, as follows (we will show also the computation of d at the same time):
   - If $d_h = \infty$, then
     - $d_h \leftarrow d_j + 1$;
     - $n_h \leftarrow n_j$;
   - ElseIf $d_h = d_j + 1$, then
     - $n_h \leftarrow n_h + n_j$;

   For the initial node i, we set n_i to 1. Notice that the computation of d has not changed, but we repeat it here since its value is used in determining the value of n. Write a revised function `bfs(g, startnode)` that takes the same input and does the same computations as the original **BFS** algorithm, the added steps to compute n, as given by the steps above. The function should return a tuple that contains d and n, where the first element was returned by **BFS** (the version for computing distances) from Homework 1, and n is the new variable.
2. Running bfs from Part 1 on graph g_1 in Figure 2, starting from node 0, report the values d_j and n_j for every node j in g_1. What quantity does the value of n_j, for a node j, reflect?

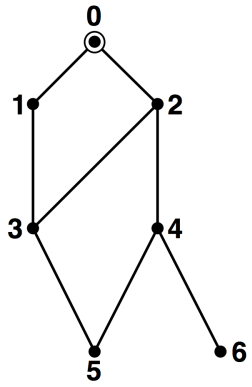

Figure 2. Graph g_1 for Part 2.

3. Write a function `compute_flow(g, dist, paths)` that computes the flow across all of the edges using d and n from **BFS**. Specifically, to compute the flow from node i in g, you would use your `compute_flow` function as follows:

   ```
   dist, npaths = bfs(g, i)
   flow = compute_flow(g, dist, npaths)
   ```

   This function returns a dictionary in which the keys are frozensets with two elements that represent an edge in the graph and the values are the flow value for the corresponding edge. You should consider testing your function on the graph in Figure 3.20 of the book using source node A. (The graphs in the book are provided in `bookgraphs.py`.) You should get the same results as in the figure. Failing to test this function is a bad idea, both in terms of your grade and in terms of your likely success on later problems.

4. Write a function, `shortest_path_edge_betweenness(g)`, that computes the shortest-path-based betweenness of all edges of graph g by summing, for each edge, the scores that the edge receives from all runs of `compute_flow` (for every run of `bfs` from a different starting point, `compute_flow` is run, thus producing for each edge n scores, if there are n nodes in the graph g).

   This function should return a dictionary in which the keys are frozensets with two elements that represent an edge in the graph and the values are the betweenness value for the corresponding edge.

   This function should utilize your `bfs` and `compute_flow` functions. Run it on the example in the **appendix** and ensure that your functions are working properly.

   Report the results for the graph in Figure 3.20. Which edges have the highest betweenness? What does this imply about the graph?

Homework 2 Problem 3

Due Feb 20 by 11pm **Points** 29 **Submitting** a file upload
File Types py, pdf, and doc **Available** Jan 27 at 4:10pm - Feb 20 at 11pm 24 days

This assignment was locked Feb 20 at 11pm.

Community Structure in Social Networks [29 pts]

In many social settings, tightly-knit subgroups are embedded into the larger network. One method for detecting tightly-knit regions in large networks is to partition the graph. Section 3.6 in the book **Networks, Crowds, and Markets** (https://www.cs.cornell.edu/home/kleinber/networks-book/) describes the simple Girvan-Newman method for graph partitioning, by successively removing edges of high betweenness. Based on the book description, the graph partitioning continues until no edges remain the graph. However, if we hope to find true communities in a social network, an ideal partitioning needs to stop when those "true" communities have been found. Given that we often do not know the communities *a priori*, we need a *stopping criterion* to tell the partitioning procedure to terminate and return the communities identified. Once such criterion is given by the *modularity measure*, which we now describe.

Consider a certain stage of the partitioning algorithm at which the graph $g = (V, E)$ is broken into k connected components $C_0, C_1, \ldots, C_{k-1}$, where $C_i \subseteq V$ for each $0 \leq i \leq k-1$. Define the symmetric $k \times k$ matrix D, where $D[i, j]$ is the fraction of all edges in the graph that connect nodes in component i to nodes in component j; see Figure 3.

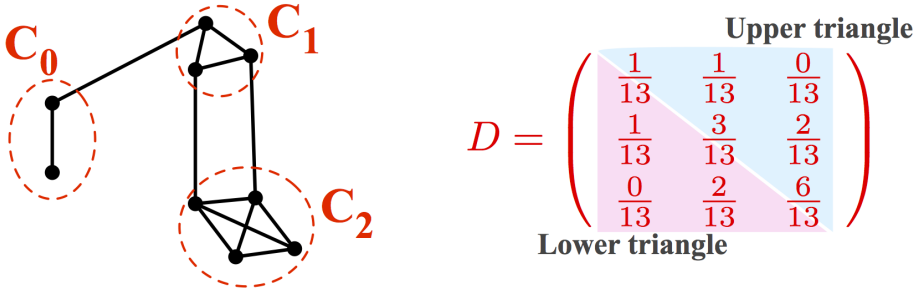

Figure 3. A graph g (left) and its D matrix (right). There are 13 edges in g. Entry $D[i, j]$ denotes the fraction of those 13 edges that have one endpoint in cluster C_i and the other endpoint in cluster C_j. Notice that the sum of the entries in the upper triangle of D must add up to 1 (and so does the sum of the entries in the lower triangle). For this matrix, we have $Tr(D) = D[0, 0] + D[1, 1] + D[2, 2] = 10/13$

The modularity measure is defined by
$Q = Tr(D) - \| D^2 \|$,
where $Tr(D) = \sum_{i=0}^{k-1} D[i, i]$ and $\|D^2\|$ is the sum of the elements of the matrix D^2. This quantity, Q, measures the fraction of the edges in the network that connect nodes of the same type (*i.e.*, within communities edges) minus the expected value of the same quantity in a graph with the same community

divisions but random connections between the nodes. In general, the higher the value of Q, the better the community structure is.

Typically, we will calculate Q for each split of a network into communities as graph partitioning proceeds, and look for local peaks in its value, which indicate satisfactory splits. The height of a peak is a measure of the strength of the community division.

1. Write function `compute_q(g,c)` that computes the value of Q, as given by the equation above, from the set c of communities of graph g. That is, c is a list of sets of nodes that form a partition of the nodes of graph g (*i.e.*, the output from `connected_components`, which we are providing). This function should return a single floating point number (which is the value of Q). We have also provided the function `gn_graph_partition(g)` which uses your `compute_q`, `connected_components`, and `shortest_path_edge_betweenness` functions to partition the graph g using the Girvan-Newman method. Test your code on the graphs of Figures 3.14 and 3.15 in the book. Full results are given in the Appendix of this homework.
2. In one of the classical studies in social network analysis, Wayne Zachary observed social interactions between the members of a karate club over two years. He constructed networks of ties between members of the club based on their social interactions both within and outside the club; see Fig. 3.13 in the book. During the course of the study, a dispute arose between the club's administrator and its principal teacher, and a result the club eventually split in two, forming two smaller clubs, centered around the administrator and the teacher. The question is: does the network structure itself contain enough information to predict the fault line, along which the club split? To answer this question, run `gn_graph_partition` on the karate club graph, and plot the values of Q you find. How many peaks do you observe? Report the community structure that achieves the highest modularity value Q. Does this community structure correspond to the community structure (*i.e.*, two small clubs) observed by Zachary?
3. In this problem, we will analyze a network of Rice University Facebook users (all of whom are students) and investigate whether the students form communities based on certain attributes. We have provided you with a subset of the Rice University Facebook graph in `rice-facebook.repr` that includes a sample of undergraduate students. The file `rice-facebook-undergrads.txt` provides data mapping the anonymized students with their college, major, and year. **Remember that even though it has been anonymized, this data is not to be shared or made public**.
    1. For the provided subgraph of Rice undergraduates, plot the Q values when using the Girvan-Newman partitioning method on the graph.
    2. Identify the community structure that corresponds to the highest value of Q. What is the highest value of Q? How many communities are there?
    3. What attribute(s) have the strongest influence on the community structure?