Aalto University
CS-E5740 Complex Networks, Autumn 2023
Hiraoka, Kivelä, Korhonen
Exercise set #4

Exercise set #4 (22 pts)

- The deadline for handing in your solutions is October 9th 2023 23:59.
- There are multiple ways for submitting your solutions. Check each question for details.
- Return also one `.zip` file containing all your Python code of the round in MyCourses.
- Check also the course practicalities file in MyCourses for more details on submitting your solutions.

1. Centrality measures for undirected networks (6 pt)

In this exercise, we will get familiar with some common centrality measures by applying them to undirected networks (although these measures can all be generalized to directed networks).

Submit your solution as a pdf file in MyCourses.

Below, we list and define the measures used in this exercise:

1. degree $k(i)$:
   Number of neighbors of node i.
2. betweenness centrality $bc(i)$:
   Number of shortest paths between other nodes of the network that pass through node i. If there are several shortest paths between a given pair of nodes, then the contribution of that node pair to the betweenness of i is given by the fraction of those paths that contain i. Betweenness scores are normalized by $(N-1)(N-2)$, i.e. the number of node-pairs in the network, excluding pairs that contain i (because paths starting or ending in node i do not contribute to the betweenness of i), which is the maximum possible score. If σ_{st} is the number of shortest paths from s to t and σ_{sit} the number of such paths that contain i, then
   $$bc(i) = \frac{1}{(N-1)(N-2)} \sum_{s \neq i} \sum_{t \neq i} \frac{\sigma_{sit}}{\sigma_{st}}.$$
3. closeness centrality $C(i)$:
   The inverse of the average shortest path distance to all other nodes except i:
   $$C(i) = \frac{N-1}{\sum_{v \neq i} d(i,v)}.$$
4. k-shell $k_s(i)$:
   Node i belongs to the k-shell if it belongs to the k-core of the network but does not belong to the $k+1$-core. The k-core is the maximal subnetwork (i.e. the largest possible subset of nodes and links between them) where all nodes have at least degree k. Explicitly, the 1-core is formed by removing nodes of degree 0 (isolated nodes) from the network; the 2-core is formed by removing nodes of degree 1 and iteratively removing the nodes that get degree 1 or 0 because of the removal; the 3-core is formed by removing nodes of degree less than 3 and those nodes that get degree less than 3 because of the removal, and so on. The 1-shell is then the set of nodes that was removed from the 1-core to obtain the 2-core.

5. eigenvector centrality $e(i)$:
   Eigenvector centrality is a generalization of degree that takes into account the degrees of the node's neighbors, and recursively the degrees of the neighbors of neighbors, and so on. It is defined as the eigenvector of the adjacency matrix that corresponds to the largest eigenvalue.

a) (2 pt) Your first task is to compute the requested centralities by hand (without using a computer) for the selected nodes in the network shown in Fig. 1.

   i) Betweenness centrality of node **B** (include the intermediate calculation steps in your report).
   ii) Closeness centrality of node **B** (include the intermediate calculation steps in your report).
   iii) k-shell centrality of all nodes in the network shown in Fig. 1.

   Note that you are not asked to report the degree and eigenvector centralities, because the first one is trivial and the latter one might be too difficult to calculate by pen-and-paper (you would need to compute the eigenvalues of a 5×5 matrix!)

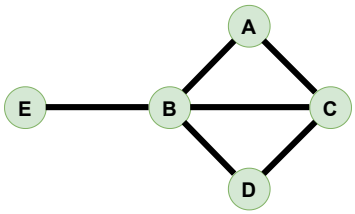

Figure 1: A small undirected network.

b) (2 pt) Use NetworkX to compute the degree, betweenness, closeness, and eigenvector centralities for the networks for the three networks (see also Fig. 2):

   – Cayley tree with $k = 3$ and $l = 3$ (`cayley_tree_edge_list.edg`). A Cayley tree is a tree in which every node except the leaves has k neighbors. The parameter l is the number of layers in the tree, i.e., the path length between the root node and the leaves.
   – Zachary karate club network (`karate_club_edge_list.edg`). The Zachary karate club network is a well-known social network studied by sociologist Zachary (1977). The nodes represent members of a karate club and the links represent interaction between members.
   – Dolphin social network (`dolphins_edge_list.edg`). A social network between bottlenose dolphins living off New Zealand compiled by Lusseau et al. (2003). The nodes represent dolphins and the links represent frequent associations between dolphins.

   Visualize the relationships between the four centralities as scatter plots in a pairwise manner. You can use the code template provided for this and the following tasks.

   Hint: For some of the networks, the power iteration algorithm used by NetworkX to

calculate eigenvector centrality may not converge. In this case, increase the value of the tolerance (`tol`) parameter of `eigenvector_centrality()` until the iteration converges.

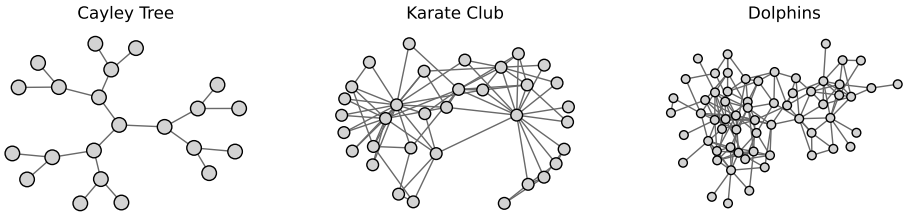

Figure 2: The networks used in part b

c) (1 pt) To highlight the differences between the centrality measures, visualize each network studied in part b) (the Cayley tree, karate club network, and dolphin social network), using each of the centrality measures to define the colors of the network nodes. To make the visualization easier, coordinates of the nodes are provided in `.pkl` files (`cayley_tree_coords.pkl`, `karate_club_coords.pkl`, `dolphines_coords.pkl`).

d) (1 pt) Explain, in 3-10 sentences, the relationship between the degree and eigenvector centrality in the three networks studied in parts b) and c). Describe the differences between the two centrality measures and explain why those differences occur.

2. PageRank for directed networks (12 pts)

PageRank, a generalization of eigenvector centrality for directed networks, was first developed for search engines like Google to determine the centrality of web pages. If we consider a random walker that with probability d moves to one of the successors of the current node (i.e., the nodes *to* which the current node is linked) and with probability $1 - d$ teleports to a random node, the PageRank of each node is equal to the fraction of time that the random walker has spent at that node.

In this exercise, we will investigate the behavior of PageRank in a simple directed network (see Fig. 3). To get started, you can use the provided code template.

Submit your solutions by taking a quiz in MyCourses.

a) (1 pt) Load the network given in file `pagerank_network.edg` and, as a sanity check, visualize it with `nx.draw`.

Hints:

- Use parameter `create_using=nx.DiGraph()` when reading the edge list.
- For a better visuazlization of the network, use `nx.draw` with parameters. For the details about parameters, check the documentation of `nx.draw_networkx`. Note that `nx.draw` is an alias for `nx.draw_networkx` (with small differences in default parameter settings).
- The NetworkX functions for visualization, including the spring layout algorithm used by `nx.draw` as its default, work only with undirected networks. Here, we apply the Kamada-Kawai layout algorithm `nx.kamada_kawai_layout` to the undirected version of the network for computing node positions.

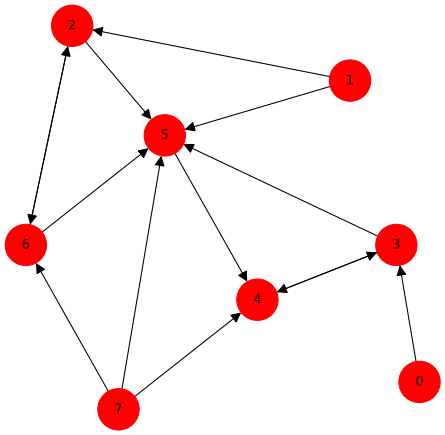

Figure 3: A simple directed network.

– For better visualization of the directed edges, use `connectionstyle='arc3, rad=0.1'` as an argument to `nx.draw` to have curved edges.

b) (4 pt) Write a function that computes the PageRank on a network by simulating a random walker. In more detail,

1. Initialize the PageRank of all nodes to 0.
2. Pick the current node (the starting point of the random walker) at random.
3. Increase the PageRank of the current node by 1.
4. Select the node to which the random walker will move next:
   * Draw a random number $p \in [0, 1]$.
   * If $p < d$, the next node is one of the successors of the current one. Choose it at random.
   * Else, the random walker will teleport. Pick the next node randomly from all nodes in the network.
5. Repeat steps 3-4 N_{steps} times.
6. Normalize the PageRank values by N_{steps}.

Use your function to compute PageRank in the example network. Visualize the result on the network by using the PageRank values as node color values. Compare your results with `nx.pagerank` by plotting both results as a function of node index.

Hint: The damping factor is normally set to $d = 0.85$. $N_{steps} = 10000$ is a reasonable choice.

c) (4 pt) The above algorithm is a naive way of computing PageRank. The actual algorithm behind the success of Google, introduced by its founders, Larry Page and Sergey Brin, is based on power iteration [1].

Power iteration finds the leading eigenvector for the "Google matrix" (or other matrices) very fast under certain conditions. An intuitive way of thinking about the power iteration algorithm is to think that at time $t-1$ you have a vector $x(t-1)$ where each element gives the probability of finding the walker. You use the rules of the random walk/teleportation process to find out what are the probabilities of finding the random walker at each node at time t. That is, you increase the time t and calculate $x(t)$ based on $x(t-1)$ until the vector x doesn't change anymore.

Write a function that computes the PageRank by using power iteration. In more detail,

1. Initialize the PageRank of all nodes to $\frac{1}{n}$, where n is the number of nodes in the network. That is, at time step $t = 0$ your PageRank vector contains the same value for all nodes, so it is equally likely to find the walker at any node. (Any other initialization strategy is possible as long as the sum of all entries is 1, and the closer the initial vector is to the final vector, the faster you will find the stationary PageRank values)
2. Increase the time step t by 1 and create a new empty PageRank vector $x(t)$.
3. Fill in each element of the new vector PageRank vector $x(t)$ using the old PageRank vector $x(t-1)$ and the formula:

$$x_i(t) = (1-d)\frac{1}{n} + d\sum_{j\in\nu_i}\frac{x_j(t-1)}{k_j^{\text{out}}},$$

   where ν_i is the set of nodes that have a directed link ending at i. For each node $j \in \nu_i$, k_j^{out} is j's out-degree. In summary, for each node i you need to calculate their entry in the new PageRank vector $x(t)$ as a sum of two parts:
   - the probability that the walker will randomly teleport into the node, given by

$$(1-d)\frac{1}{n},$$

     and
   - the probability that the walker will move from a neighbor j to node i. Iterate over each in-neighbor j of the node i (i.e., there is a link from j to i) and add the neighbor's contribution

$$d\frac{x_j(t-1)}{k_j^{\text{out}}}$$

     to the entry of node i in the new PageRank vector $x(t)$.
4. Repeat steps 2 and 3 $N_{\text{iterations}}$ times.

Use your function to compute PageRank in the sample network and visualize the result on top of the network as in b).

Hints:

- The damping factor is normally set to $d = 0.85$.
- You can monitor the progress of the power iteration by printing out the change in the PageRank vector $\Delta(t) = \sum_i |x_i(t) - x_i(t-1)|$ after each time step. The change $\Delta(t)$ should be a decreasing function of t. $N_{\text{iterations}} = 10$ should be more than enough in most cases.

- You can list the incoming edges to node i with the function `net.in_edges(i)`, where `net` is the network object. Alternatively, you can use the function `net.predecessors(i)`, which returns an iterator over predecessor nodes of node i.
- The sum of all elements in the PageRank vector should always equal one. There may be small deviations from this due to numerical errors, but much larger or smaller values are an indication that something is wrong with the code.

d) (3 pt) The aim of this task is to understand how the structure of a network relates to PageRank. Answer the following questions.

i) Choose all the correct statements about the relationship between the PageRank and the degree of a node:

A The PageRank of a node is always higher if its in-degree is higher.
B The PageRank of a node is always higher if its out-degree is lower.
C The PageRank of a node is affected by its in-degree, but not by its out-degree.
D The PageRank of a node is dependent on the PageRanks of its predecessors and their out-degrees.
E The PageRank of a node is dependent on the PageRanks of its successors, but not on their in-degrees.
F The PageRank of a node with zero in-degree is zero.

ii) If a node belongs to a strongly connected component, how does that affect the PageRank of the node? Explain in 2-5 sentences.

iii) Explain why the PageRank of nodes 3 and 4 in the example network is higher than that of node 5 in 2-5 sentences.

iv) How could the information about the network's structure be used in improving the power iteration algorithm given in part c)? Explain in 1-3 sentences.

Hint: Are there ways to incorporate this information in order to make the algorithm converge faster?

Challenge exercises (4 pts)

e) (2 pt) The Google search engine indexes billions of websites and the algorithm for calculating the PageRank needs to be extremely fast. In the original paper about PageRank [1], by Google founders Larry Page and Sergey Brin, they claim that their "iterative algorithm" is able to calculate the PageRank for 26 million webpages in a few hours using a normal desktop computer (in 1998).

Come up with a rough estimate of how long it would take for your power iteration algorithm (part c) and naive random walker algorithm (part b) to do the same. You can assume that the average degree of the 26 million node network is small and that the power iteration converges in the same number of time steps as it does for your smaller networks. For the random walk, you can assume you need to run enough time steps such that the walker visits each node 1000 times, on average. You can also omit any considerations of fitting the large network in memory, or the time it takes to read it from disk, etc. Under these assumptions, you can simply calculate the time it takes to run the algorithm in a reasonably-sized network, and then multiply the result by the number of times the 26 million node network is larger than your reasonably-sized network.

Report all calculations and parameters you use (such as size of the network, number of time steps, etc.). Simply reporting the result without telling how you obtained it will not get you any points.

Hints:

- There are several ways of timing your code. You can use for example IPython's `\%timeit` command or the Python `timeit` module.
- The small sample network is probably going to be too small to accurately test the speed of your function. You should aim for a network for which it takes several seconds to run the PageRank function (note that a network size that works for the power iteration algorithm may be far too large for the random algorithm to finish in a reasonable time). You might find it useful to use network models in NetworkX to run your code. For example,
  `nx.directed_configuration_model(10**4*[5],10**4*[5], create_using=nx.DiGraph())`
  will produce a network with 10000 nodes where each node has in- and out-degrees equal to 5 using the configuration model.
- Don't feel bad if you cannot beat Larry and Sergey in speed by using NetworkX and Python. These tools are not meant for speed of computation and even modern computers might not be enough to help. Also, your competitors invented Google.

f) (2 pt) Investigate the effect of the damping factor d on the PageRank values of the network used in parts a)-c). Repeat the PageRank calculation with, e.g., 5 different values of $d \in [0, 1]$ and plot the PageRank as a function of node index (plots of all values of d in the same figure).

Interpret the results. How and why does the change of d affect the rank of the nodes and the absolute PageRank values? Explain what happens when $d = 0$ and when $d = 1$.

In case you do not trust your implementations in b) and c), you can use PageRank values obtained with `nx.pagerank` in this last task.

Feedback (1 pt)

To earn one bonus point, give feedback on this exercise set and the corresponding lecture latest two days after the exercise round's submission deadline. You can find the feedback form in MyCourses.

References

[1] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Computer networks and ISDN systems*, vol. 30, no. 1-7, pp. 107–117, 1998.