Exercise set #7 (22 pts)

- The deadline for handing in your solutions is November 6th 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. Network thresholding and spanning trees: the case of US air traffic (6 pts)

In this exercise, we will get familiar with different approaches for thresholding networks, and will also learn how they can be used for efficiently visualizing networks. You are given an undirected network describing the US Air Traffic between 14th and 23rd December 2008[1]. In the network, each node corresponds to an airport and link weights describe the number of flights between the airports during the time period.

The network is given in the file `aggregated_US_air_traffic_network_undir.edg`, and `us_airport_id_info.csv` contains information about names and locations of the airports. The file `US_air_bg.png` contains a map of the US; **you can use the functions provided in the code template to visualize the network on the map**. In this exercise, you may also freely use all available `NetworkX` functions.

Return your results through a MyCourses quiz.

a) (1 pt) Let us first check the basic network properties to get some ideas on what the network is like. **Compute** the following quantities:

   - Number of nodes N, number of links L, and density D
   - Network diameter d
   - Average clustering coefficient C

   Hints:

   - For the clustering coefficient, consider the unweighted version of the network, where two airports are linked if there is a flight between them.
   - Check MyCourses for instructions on the accuracy of the results. Typically there is no need to report long decimal values with insignificant digits that the code returns.

b) (1 pt) **Visualize** the full network with all links on top of the map of the US. The resulting figure is somewhat messy due to the large number of visible links.

c) (2 pts) In order to reduce the number of plotted links, **compute** both the *maximal* and *minimal spanning trees* of the network and visualize them. Then, **answer** the following question:

[1]Data from the Bureau of Transportation Statistics.

- If you would like to understand the overall organization of the air traffic in the US, would you use the minimal or maximal spanning tree? **Why?**

Hint: For computing minimum spanning trees, use `nx.minimum_spanning_tree`. For computing maximum spanning trees, use `nx.maximum_spanning_tree`.

d) (2 pts) **Threshold** and **visualize** the original network by taking only the strongest M links into account, where $M = N - 1$ is the number of links in the maximal spanning tree. Then, **answer** the following questions:

- **How many** of the links in the thresholded network are the same as those in the maximal spanning tree?
- Given this number and the visualizations, does simple thresholding yield a similar network as the maximal spanning tree?

Hint: Note that the network is undirected, which means that edge (i, j) is the same as edge (j, i). When comparing the set of edges, however, the order of the nodes in the edge tuple matters. Make sure that each edge is represented in a consistent order across the two networks when comparing them.

2. Network sampling (5 pts)

Many network datasets are actually samples of the underlying graphs we are interested in. That is, nodes and edges in these empirical networks have been sampled such that we only observe certain parts of the real network. This can severely bias even the most simple network measures so that the sampled graph has quantitatively different properties when compared to the underlying graph. In this exercise we will see the effect of three different sampling schemes on the network transitivity C (also known as the global clustering coefficient) and derive estimators that can be used to correct for these biases.

Submit your solutions for this exercise through MyCourses as a single pdf file.

Let us first recall the definition of transitivity C:

$$C = \frac{\tau_\triangle}{\tau_\angle} = \frac{\sum_i E_i}{\sum_i \binom{k_i}{2}}\,, \tag{1}$$

where $\tau_\triangle$ is three times the number of triangles [2] (three nodes that are fully connected), $\tau_\angle$ is the number of two-stars (a node and two of its neighbors, regardless of whether they are triangles), i is a node, k_i is the degree of i, and E_i is the number of triangles centered on i (in other words, how many triangles pass by i).

Now let's get to work. Start by generating a network from a random model. Use the command `nx.relaxed_caveman_graph` to generate a graph with 55 communities of 12 people each, where each link is then rewired with probability 0.1. This is the "true" underlying graph from which we obtain samples.

[2] This is equal to three times the number of triangles since we are going through all nodes, and thus counting each triangle three times. We could define an alternative estimator that corrects for this overestimation, but for transitivity $\tau_\triangle = \sum_i E_i$ suffices.

a) (3 pts) First, let's implement the three sampling schemes. **Program** three functions that perform:

1. **Bernoulli sampling of nodes**: iterate over nodes, and sample each one with probability p. We observe an edge if and only if we have sampled its two constituting nodes.
2. **Bernoulli sampling of edges**: iterate over edges, and sample each one with probability p. We observe a node if and only if we have sampled at least one of its edges.
3. **Star sampling**: iterate over nodes, and sample each one with probability p. If you have directly sampled a node, you also observe all of its neighbors (a real-life example would be a dataset obtained by crawling through friendship lists of randomly selected users in a social networking website). Note: here we will have nodes that are sampled a) directly with probability p and b) indirectly via sampling a neighbor. It is useful to keep a list of nodes you sampled directly.

Obtain samples of the network using the three sampling schemes with probability $p = 0.28$. Then, use either the template code or your own to obtain empirical estimates of the number of triangles, the number of two-stars, and transitivity. **Report** your results on a table where the columns represent:

– sampled number of triangles,
– sampled number of two-stars,
– transitivity in sampled network,
– fraction of sampled triangles over triangles in original network,
– fraction of sampled two-stars over two-stars in original network,

and the rows represent the different sampling schemes (plus an extra row with the values of the original network). **Answer** the following questions:

– How do sampling schemes compare in the fraction of triangles/two-stars they preseve?
– Do sampling schemes affect two-stars/triangles in the same way?
– Transitivity via node sampling should be similar to the real value; what could be the reason for this?

Hint: For star sampling, there are at least two ways of counting the sampled two-stars: we can either focus on the directly sampled nodes and obtain their full degrees, or we can count all the two-stars regardless of whether the center node itself was selected (so if two nodes with the same neighbor are selected, we observe a two-star centered on a node not directly sampled). Both answers are correct, but using the directly-sampled nodes will make calculations easier on the next excercise.

b) (2 pts, pen and paper) Different sampling schemes alter the observed number of structures on the sampled networks. Luckily, knowing the sampling probability p, we can estimate how much these numbers vary. As an example, for Bernoulli sampling of **edges**, the probability of sampling a two-star from the original network is $p^e_\angle = p^2$ (we need to observe two edges), while a triangle is sampled with probability $p^e_\triangle = p^3$ (we need to observe three edges). **Derive and explain how to obtain**:

i) the probabilities of sampling a two-star and a triangle using Bernoulli sampling of **nodes** ($p^n_\angle$ and $p^n_\triangle$); and

ii) the probabilities of sampling a two-star and a triangle using **star sampling** ($p^s_\angle$ and $p^s_\triangle$).

Hint: You can check the validity of your calculations by comparing the fraction of two-stars and triangles sampled from the totals in the original network obtained in a), and your answers when $p = 0.28$.

Challenge exercises (5 pts)

c) (2 pt, pen and paper) The Horvitz-Thompson (HT) estimator is a simple way of correcting for the bias induced by sampling. Let $\hat{\tau}$ be the empirical count of a structure found in a sampled network, such as your results from a). If p_τ is the probability of observing these structures, then the HT estimator for the total counts is simply

$$\hat{\tau}^{HT} = \frac{1}{p_\tau}\hat{\tau}. \tag{2}$$

Explain why the HT estimator corrects for sampling bias. **Write and simplify** the HT estimators for the number of two-stars, triangles and transitivity for the three sampling schemes we explored before.

Hint: For transitivity, we may simply use a "plug-in" estimator by substituting the empirical estimators for the HT estimators; that is, use $\hat{\tau}^{HT}_\triangle$ and $\hat{\tau}^{HT}_\angle$ instead of $\hat{\tau}_\triangle$ and $\hat{\tau}_\angle$ in the transitivity formula.

d) (3 pts) **Implement** the HT estimator for the three sampling schemes. Given two selection probabilities p, we will sample at least $n = 150$ times to obtain distributions of some of our HT estimators, and compare it with the measurements from the sampled networks. For each $p = 0.35, 0.5$, and for each sampling scheme, obtain $n = 150$ samples from the original network, and **calculate** the HT estimator for the number of triangles and for transitivity. For $p = 0.5$ include also the empirical estimator (the counts without the HT correction), and for all plots include a vertical line depicting the value of the original network. As a summary, you will **report your results in six plots**:

- Histograms of estimator $\hat{\tau}^{HT,n}_\triangle$ for $p = 0.35, 0.5$ and $\hat{\tau}_\triangle$ (empirical counts) for $p = 0.5$ and true value of $\tau_\triangle$.
- Histograms of estimator $\hat{\tau}^{HT,n}_C$ for $p = 0.35, 0.5$ and $\hat{\tau}_C$ (empirical value) for $p = 0.5$ and true value of C.
- Histograms of estimator $\hat{\tau}^{HT,e}_\triangle$ for $p = 0.35, 0.5$ and $\hat{\tau}_\triangle$ (empirical counts) for $p = 0.5$ and true value of $\tau_\triangle$.
- Histograms of estimator $\hat{\tau}^{HT,e}_C$ for $p = 0.35, 0.5$ and $\hat{\tau}_C$ (empirical value) for $p = 0.5$ and true value of C.
- Histograms of estimator $\hat{\tau}^{HT,s}_\triangle$ for $p = 0.35, 0.5$ and $\hat{\tau}_\triangle$ (empirical counts) for $p = 0.5$ and true value of $\tau_\triangle$.
- Histograms of estimator $\hat{\tau}^{HT,s}_C$ for $p = 0.35, 0.5$ and $\hat{\tau}^C$ (empirical value) for $p = 0.5$ and true value of C.

In your plots, the distribution of HT estimators should lie around the real value. **Answer** the following questions:

- What is the effect of the sampling probability p on your estimators?
- How do the HT distributions differ between sampling schemes?
- In the last plot, the empirical estimators (without the HT correction) should be centered around the true value; why is this the case?

Hints:

- Since we want to observe how different samples may arise from the same network, we need to take a large number of samples ($n = 150$, for instance). However, while you are coding and testing it may be wise to use a smaller n. Keep in mind that if your code takes too much time to run on your computer, you can report results for a smaller n or for a smaller network.
- In case you were not able to obtain the theoretical probabilitites in excercise b), you can substitute the theoretical probabilities ($p^s_\triangle$, for example) with the fraction of sampled structures over the totals in the original network. If this is the case, please state so in your report and explain why this substitution is possible.

3. Thresholding of similarity networks (6 pt)

Similarity networks are weighted networks where each node is associated with a time series and link weights represent the temporal similarity (typically Pearson correlation) between these time series. Similarity networks are used commonly in the study of e.g. financial systems, brain function, and climate. Before further analysing the network, one needs to threshold it, that is, to decide which correlations are strong enough to be considered as network links. In this exercise, we'll construct a similarity network from neuroimaging data and explore how thresholding the network affects some of its properties.

The data used in this exercise is functional magnetic resonance imaging (fMRI) data from the ABIDE II database[3] [1]. fMRI is an indirect measure of brain activity: it measures the blood oxygen level in the brain. The pickle file `roi_time_series.pkl` contains the fMRI time series of 246 brain areas (known as Regions of Interest or ROIs) from the Brainnetome parcellation [2].

You can use the Python template available at JupyterHub to get started. Submit your solutions by taking a quiz in MyCourses.

a) (1 pt) **Download** the data, **construct** the full adjacency matrix by calculating the Pearson correlation coefficient between each pair of time series, and **visualize** it with `plt.imshow()`.

   Hints: Diagonal of the correlation matrix should contain ones, corresponding to the Pearson correlation of a time series with itself. Set the diagonal to zero to prevent self-links.

[3] `http://fcon_1000.projects.nitrc.org/indi/abide/abide_II.html`. The ABIDE II data collection was funded by National Institute of Mental Health (NIMH 5R21MH107045 and NIMH 5R21MH107045) and by gifts from Joseph P. Healey, Phyllis Green and Randolph Cowen to the Child Mind Institute.

b) (2 pts) **Write** a function for thresholding the adjacency matrix to a given density. While thresholding, set the links weaker than the threshold to 0 but keep the weights of the links stronger than the threshold. **Visualize** the adjacency matrix at densities 1%, 5%, 10%, 20% and 50%.

c) (2 pts) Small-worldness is often considered as an important property of brain networks: the balance between high clustering of regular networks and short path lengths of random networks supposedly optimally combines local processing and global efficiency[4]. The small-worldness coefficient σ is defined as

$$\sigma = \frac{C/C_r}{L/L_r}, \tag{3}$$

where C is the average clustering coefficient, L is the average shortest path length of the network, and C_r and L_r are the corresponding properties of a random network with the same density. A network is typically considered as small-world if $\sigma > 1$.

Threshold the adjacency matrix to 10 densities ranging from 1% to 100%. At each density, **calculate** C, C_r, L, L_r, and σ. **Visualize** your results as 3 plots: i) C and C_r as a function of density, ii) L and L_r as a function of density, iii) σ as a function of density.

Use the non-weighted versions of the functions for calculating C and L. Alternatively, you can set the weight of all super-threshold links to 1.

Hints:

- `nx.from_numpy_array()` creates network objects from adjacency matrices.
- Average shortest path length is not defined for non-connected networks. So, if the network is not connected, calculate the average shortest path length for the largest connected component. Check `nx.connected_components()`.
- NetworkX has a function called `sigma()` for calculating the small-worldness coefficient. This function creates the random reference network by rewiring the network links. While this ensures that the reference network has the same density as the original network, it is also very slow. So, we will calculate C_r and L_r by creating n_{random} ER graphs, calculating the average clustering and shortest path length for each of them, and averaging over the random graphs. You can use $n_{random} = 10$.
- Don't worry if running these calculations takes up to 1 minute.

d) (1 pt) Select the correct statement(s) to describe your results from parts a-c):

A: "Densities below 10% are too low for reasonable analysis since network structure is not visible."

B: "Average clustering coefficient of both similarity and ER networks increases linearly with increasing density."

C: "The network is small-world at all densities below 50% but the actual value of σ varies with density."

D: "Brain networks are small-world mostly because they have higher average clustering coefficient than random networks."

E: "In this network, average shortest path length affects the small-worldness more than average clustering coefficient."

F: "Optimal similarity network analysis should use a range of densities instead of a single one."

[4]However, interpretation of small-worldness in brain networks is subject to discussion. See e.g. [3] for a critical opinion.

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] A. Di Martino, D. O'Connor, B. Chen, K. Alaerts, J. S. Anderson, M. Assaf, J. H. Balsters, L. Baxter, A. Beggiato, S. Bernaerts *et al.*, "Enhancing studies of the connectome in autism using the Autism Brain Imaging Data Exchange II," *Scientific Data*, vol. 4, no. 1, pp. 1–15, 2017.

[2] L. Fan, H. Li, J. Zhuo, Y. Zhang, J. Wang, L. Chen, Z. Yang, C. Chu, S. Xie, A. R. Laird *et al.*, "The human Brainnetome atlas: A new brain atlas based on connectional architecture," *Cerebral Cortex*, vol. 26, no. 8, pp. 3508–3526, 2016.

[3] D. Papo, M. Zanin, J. H. Martínez, and J. M. Buldú, "Beware of the small-world neuroscientist!" *Frontiers in Human Neuroscience*, vol. 10, p. 96, 2016.