

## Exercise set #4 (14 pts)

- The deadline for handing in your solutions is October 11th 2022 20:00.
- Return your solutions (one .pdf file and one .zip file containing Python code) in MyCourses (Assignments tab). Additionally, submit your pdf file also to the Turnitin plagiarism checker in MyCourses.
- Check also the course practicalities page in MyCourses for more details on writing your report.

### 1. Percolation in Erdős-Rényi networks (8 pts)

Erdős-Rényi (ER) networks are random networks where  $N$  nodes are randomly connected such that the probability that a pair of nodes is linked is  $p$ . In network science, the ER random graphs are important because they provide the simplest reference to which one can compare real-world networks. Many interesting real networks are very large (in number of nodes) and sparse (in a sense that single nodes have very few connections as compared to the network size).

We will analyse large and sparse ER graphs, where the average degree  $\langle k \rangle$  is some fixed (and small) number, and the size of the network  $N$  is very large. Theoretically we will be thinking of networks that are infinitely large, but where  $\langle k \rangle$  is still fixed (i.e.,  $N \rightarrow \infty$  and  $p \rightarrow 0$  such that  $p(N-1) = \langle k \rangle$  stays constant). In terms of simulations we will use as large networks as is convenient from the computational point of view, with the idea that larger network sizes will give us better results.

In this exercise, we will analyze the percolation properties of ER graphs. We will especially focus on the *percolation threshold* which is the value of  $\langle k \rangle$  where the giant connected component appears (when  $N$  is increasing, based on definition the giant component exists if the size of largest connected component,  $S_{\max}$ , also grows in proportion to  $N$  or in other words  $S_{\max}/N \rightarrow s_{\max}$  where  $s_{\max} > 0$ ).

- a) (2 pts) It can be shown that large and sparse ER graphs are locally tree-like; meaning that there are few loops of any size and especially very few small loops. For the purposes of this exercises we will assume that the locally tree-like large and sparse ER graphs behave the same as trees. With this assumption, use the idea of branching processes and the concept of excess degree (presented in the lectures) to calculate the expected number of nodes at  $d$  steps away,  $n_d$ , from a randomly selected node in an ER network as a function of  $\langle k \rangle$  and  $d$ . Using this result, justify that in large and sparse ER networks, the giant component appears when  $\langle k \rangle > 1$ .

*Hints:*

- What is the relationship between  $n_d$  and  $n_{d-1}$ ? Can this relationship be re-written as a closed-form expression, that is, a non-recursive relationship that does not involve  $n_{d-1}$ ?
- How does this closed-form expression simplify for the special case of the ER network, given the properties of the degree distribution? Remember that the degree distribution of an ER network is a Poisson distribution when  $N \rightarrow \infty$  such that  $\langle k \rangle$  is

constant. A property of Poisson distribution is that the mean and the variance are equal, in other words, that  $\langle k \rangle = \langle k^2 \rangle - \langle k \rangle^2$ .

- b) (2 pts) Using numerical simulations calculate the  $n_d$  value for  $d \in \{0 \dots 15\}$ ,  $\langle k \rangle \in \{0.5, 1, 2\}$ , and starting from enough randomly selected nodes to get a good estimate for the expected value. Try out two network sizes:  $N = 10^4$  and  $N = 10^5$  to see how the size affects the calculations. Present a plot of mean  $n_d$  as a function of  $d$  for different values of  $k$ .

For this and the following tasks, you can use the Jupyter notebook available at Jupyterhub. Note that the code in the notebook reports the plots for sections b) and c) as one set for easier comparison. A reasonable number for starting nodes can be anywhere from hundreds to thousands based on how fast your computer and your simulation code is. With a moderately recent computers and an efficient implementation of the algorithm, you should be able to simply use the parameters provided in the the notebook.

- c) (1 pts) In this exercises we will explore the limits to the assumption of local tree-likeness of large, sparse Erdős-Rényi networks. More specifically, we will explore how far from any node you would have to look to see non-tree-like behavior. We will do this by calculating the number of edges that nodes at depth  $d$  have that go back to some earlier level in addition to the single edge that connects each node to the level  $d - 1$ , and reporting the average fraction of such edges to all edges that go from depth  $d$  to earlier levels/depths. In a perfect tree this fraction is exactly 0. Plot this fraction as a function of  $d$ .
- d) (2 pts) Calculate the component sizes of simulated ER networks, and use this data to (loosely) verify that the percolation threshold of ER networks is at the average degree of  $\langle k \rangle = 1$ . That is, for  $\langle k \rangle < 1$  the largest connected component is small (size being measured as number of nodes), and for  $\langle k \rangle > 1$  it quickly reaches the network size.

Do this by generating ER networks of size  $N = 10^5$  (or  $N = 10^4$  if you run into memory or processing power problems) with different average degrees:  $\langle k \rangle = [0.00, 0.05, \dots, 2.45, 2.50]$ . For each of the ER networks, **compute** the size of the *largest* component and **plot** it against  $\langle k \rangle$ . Note that the code in the notebook reports the plots for sections d) and e) as one set for easier comparison.

- e) (1 pt) Another, a more elegant, way to find out when the percolation transition happens is to try to find the point at which the possibility for the largest component size growth is the largest when the control parameter (here  $\langle k \rangle$  or  $p$ ) is changed very little. Think about the situation where  $\langle k \rangle$  is changed so slightly that a single link is added between the largest component and a randomly selected node that is not in the largest component. The expected change in the largest component size in this situation is some times called *susceptibility*, and it should get very large values at the percolation transition point. The susceptibility depends on the size distribution of all the other components, and it can be calculated with the following formula:

$$\chi = \frac{(\sum_i i^2 C(i)) - S_{max}^2}{(\sum_i i C(i)) - S_{max}}, \quad (1)$$

where  $C(i)$  is the number of components with  $i$  nodes. Calculate the susceptibility  $\chi$  for each network generated in exercise d), and again plot  $\chi$  as a function of  $\langle k \rangle$ . Explain the shape of the curve, and its implications.

## 2. Error and attack tolerance of networks (6 pts)

Error and attack tolerance of networks are often characterized using percolation analysis, where links are removed from the network according to different rules. Typically this kind of analyses are performed on infrastructure networks, such as power-grids or road networks. In this exercise, we will apply this idea to a Facebook-like web-page<sup>1</sup>, and focus on the role of strong and weak links in the network. In this network, each node corresponds to a user of the website and a link between two nodes represents contact, in form of messages, between the two users. Additionally, an integer “weight” value is assigned to each link that corresponds to the number of messages exchanged between the users. As opposed to a non-weighted network where nodes are either connected or disconnected, here a higher or lower weight on a link shows stronger or weaker relation (as expressed by number of messages) between each two users.

In the file `OClinks_w_undir.edg`, the three entries of each row describe one link:

`(node_i node_j w_ij)`,

where the last entry `w_ij` is the weight of the link between nodes `node_i` and `node_j`.

Your task is now to remove links (one by one) from the network in the order of

- (i) descending link weight (i.e. remove strong links first),
- (ii) ascending link weight (i.e. remove weak links first),
- (iii) random order

While removing the links, monitor the *size of the largest component*  $S$  as a function of the fraction of removed links  $f \in [0, 1]$ .

**Visualize**  $S$  as a function of  $f$  in all three cases **in one plot**. There should be clear differences between all three curves. To which of the three approaches is the network most and least vulnerable? In other words, in which case does the giant component shrink fastest / slowest?

*Hints:*

- For this task, you can use the Jupyter notebook available at Jupyterhub.
- In the exercise, `networkx.connected_components(G)` may turn out handy. It returns a list of the components of the network, each of them presented as a list of nodes belonging to the component.
- Let `components` be the outcome from `networkx.connected_components(G)`. For getting the largest component, you can use `max(components, key=len)`.
- Edges of the present network are tuples of three values. For sorting them based on their weight, `sorted` function with `key` parameter can be useful. For more information, check <https://wiki.python.org/moin/HowTo/Sorting>.
- The overall running time of this simulation can take up to a couple of minutes but not orders of magnitudes more.

---

<sup>1</sup>Data originally from <http://toreopsahl.com/datasets/>

## **Feedback (1 pt)**

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

## **References**