Course 67562 Spring 2025

Dynamics, Networks and Computation

Homework Exercise #1

Prof. Mor Nitzan *TA: Hagai Rappeport*

1 Submission Guidelines

Submission deadline is Thursday, the 24th of April 2025, at 23:59

1.1 Submission

Submit a single .ipynb file (Jupyter notebook) with your answers to the questions. (Python code cells for code questions, MarkDown cells for text/math) If there are any other files required to run the code in the notebook, supply them as well.

If you wish to submit in any other way (e.g. want to use another language), Okay it with me (Hagai) first.

1.2 External Libraries

You can use any functions and libraries you wish to e.g. represent graphs or sample from specific distributions. Do not use any function which obviates a question (e.g. a function uses MCMC to sample from a configuration model when the question asks you to implement such a function). If you have any doubt, ask in the dedicated ex. forum on Moodle.

1.3 Working in Groups

Your'e allowed to discuss questions and code implementations with other students, but the final answers should be yours alone. In particular, you should write all code yourself.

2 Network Robustness and Phase Transitions

In class we discussed network *robustness* - a network's ability to stay connected despite removal of certain nodes (either at random or ones chosen specifically via some criteria).

For a given network or random graph model, we defined p_∞ as the probability of a randomly chosen node to belong to the largest connected component in the graph[1]

2.1 Robustness in Grids

2.1.1 Two Dimensional Grids

Analyze the behavior of p_∞ as a function of the fraction of random nodes removed, f, in a 2D grid. Specifically, start with a 100×100 grid, and for each value of f in the range $[0, 0.01, 0.02, \ldots, 0.99, 1]$ remove a random fraction f of the nodes in the grid and compute p_∞. Plot the results (which from here on we'll call a graph's *robustness curve*). What do you observe?

[1] For a fixed network, this is simply the fraction of nodes belonging to the largest connected component in that network. For a random graph, this takes into account the distribution over graphs

2.1.2 Higher Dimensional Grids

What you expect would change in the results from the previous question if we instead used a three-dimensional grid? Why?

Repeat the analysis with a $20 \times 20 \times 20$ grid and discuss your results - Was your prediction confirmed?

2.2 Robustness in Erdős–Rényi Graphs

2.2.1 Random Node Removal

Compute and plot the robustness curve of a random graph sampled from the Erdős–Rényi distribution with $< k >= 4$ and $n = 2,000$ (either of the two variants we discussed in class).

How do you expect your results to change if you change either n or $< k >$? Why? Repeat the analysis changing these parameters. Was your prediction confirmed?

2.2.2 Adversarial Node Removal

Compute the *adversarial robustness curve* of the $< k >= 4, n = 2,000$ ER graph, where now for each fraction f we remove not a random fraction, but the fraction f of nodes with the highest degrees. Plot both robustness curves and compare.

2.3 Robustness of Real World Networks

Choose a network from this list and download it.

Compute both the robustness and the adversarial robustness curves and plot them. How do these compare to the ER case?

2.3.1 Adversarial Strategies

Suggest and implement a different attack strategy than removing the nodes by order of highest degree to lowest. Explain when (i.e. for networks with which characteristics) you think this may be a more efficient strategy to sabotage (disconnect) the network.

Compute adversarial robustness curves using both the degree-order strategy and yours to any network (real-world or random) in which you think your strategy should be more efficient. Was it?

3 Stub Matching

In recitation we described the stub matching procedure to sample from the configuration model parametrized by a given degree sequence $(k_1, \ldots, k_n)$ (Note that this refers to the *undirected* configuration model). We showed that it correctly samples uniformly from the space of pseudographs. The solution we discussed for restricting our sample space, e.g. to exclude multiple edges, is to use rejection sampling, i.e. to discard any sample which does not comply with the constraint.

3.1 Sample Pseudographs

Implement the function sample_CM_stub_matching() which takes as an argument a degree sequence (a vector of non-negative integers) and samples an *undirected* pseudograph with said degree sequence.

3.2 Rejection Sampling

Implement the function sample_CM_stub_matching_no_multiple() which samples an undirected graph (with no multiple edges) with the given degree sequence by using rejection sampling. Specifically, the function should call sample_CM_stub_matching() repeatedly until a graph with no multiple edges is returned, and record the number of failed attempts.

3.3 Power Law Degree Sequence

Implement the function power_law_deg_seq() which given a graph size and exponent $\alpha > 1$, samples a graphical [2] degree sequence where each degree is sampled independently from the power law distribution ($p(k) \propto k^{-\alpha}$). To ensure the sequence is graphical, you'll need to use rejection sampling here too (this can actually also be done with MCMC, but that's a bit of overkill). Note that we are *not* constraining the number of edges (as opposed to the case we discussed in recitation).

Note on sampling from a power-law distribution

Given a graph with n nodes, you need to sample each degree from the [integer] power-law distribution between 1 and $n-1$.

The easiest way to do this is to create a vector $\hat{p} \in \mathbb{R}^{n-1}$ where $\hat{p}_k = k^{-\alpha}$, normalize to obtain a probability vector $p_k = \frac{\hat{p}_k}{\sum \hat{p}_i}$ and sample from the induced distribution over $\{1, \ldots, n-1\}$.

You can also sample from a regular (unbounded) continuous power law and round to the closest integer, discarding samples larger then $n-1$, or else use Inverse Transform Sampling, to get a closed analytic form of the required distribution.

3.4 Effect of Degree Sequence on Number of Rejections

Sample 50 degree sequences for a graph with $n = 20$ nodes from the power law distribution with $\alpha = 2$. For each one of these, call sample_CM_stub_matching_no_multiple() and record how many rejections you encountered until sampling a graph with no multiple edges.[3]

Analyze the connection between the degree sequences and number of rejections. Can you find a property of the sequences which is correlated to this number? What causes this connection? Provide plots if necessary to support your answer.

4 Edge Swapping

In recitation, we discussed the edge swapping Markov Chain, in which at each iteration two edges, (u_1, v_1) and (u_2, v_2) were sampled, and swapped (as long as the swap did not result in a disallowed state of multiple edges between nodes) to give (u_1, v_2) and (u_2, v_1).

4.1 Stationary Distribution in the Naive Edge-Swapping Algorithm

In this question we'll analyze the stationary distribution of the following, seemingly very similar, edge-swapping Markov chains (differences highlighted in red):

Algorithm 1

```
for i ← 1 to N do
    edgePair ← getRandomEdgePair(G)
    if switchIsAllowed(G, edgePair) then
        G ← switch(G, edgePair)
        i ← i + 1
    end if
end for
return G
```

Algorithm 2

```
for i ← 1 to N do
    edgePair ← getRandomEdgePair(G)
    if switchIsAllowed(G, edgePair) then
        G ← switch(G, edgePair)
    end if
    i ← i + 1
end for
return G
```

[2] A graphical degree sequence is one which has a simple graph realization. The Erdos-Gallai theorem gives necessary and sufficient conditions for a sequence to be graphical in the undirected case

[3] Some sequences can take a *very* long time. If it seems like you've encountered one of these (say over a million rejections), you can stop and start over.

Assume that getRandomEdgePair(G) returns two different edges in G sampled at random, that switchIsAllowed($G, edgePair$) returns True iff swapping the edge pair results in a double edge, and that T is some number sufficiently large to ensure that both chains converge to their respective stationray distributions.

For each of these two algorithms, characterize the distributions which result from applying them. Is the distribution uniform? If so, prove it. If not, explain why not and write down the induced distribution on the minimal example discussed in recitation, the configuration model with $k^{\text{in}} = (1, 1, 2)$ and $k^{\text{out}} = (1, 1, 2)$.

4.2 Edge Swapping With No Self Edges

There are many applications in which we may wish to sample networks from the configuration model defined over the space of *simple* graphs, i.e. those with no multiple edges nor self edges.

Using the conditions we developed in recitation for convergence of a Markov chain to a stationary distribution, show that if we disallow graphs with self edges, the edge swapping algorithm is no longer guaranteed to converge to a uniform distribution over the restricted space. Suggest a fix to this problem (no need for a proof).

4.3 Implementing MCMC in Code

Implement a function CM_MCMC() which accepts a *directed* graph which will serve as the MCMC's first state and then performs edge swapping (the variant which samples uniformly over the space of all graphs including those with self edges, not your solution to the previous question) for T iterations[4]. Use your function to sample from the configuration model we used in recitation with $k^{\text{in}} = (1, 1, 2)$ and $k^{\text{out}} = (1, 1, 2)$ in which you can easily enumerate all possible graphs by hand and test correctness. Does your function sample uniformly from the required space?

Good Luck!

[4]There are no known bounds in the literature on convergence rates of the edge swapping Markov chain. We didn't discuss in class methods to detect convergence, so you may either implement any heuristic which seems reasonable to you or else simply fix T at some large number.