Social Networks and Online Markets

Homework 1

Due: 16/6/2023, 23:59

Instructions

You must hand in the homeworks electronically and before the due date and time.

The first homework has to be done by each **person individually**.

Handing in: You must hand in the homeworks by the due date and time by an email to `aris@diag.uniroma1.it` that will contain as attachment **(not links to some file-uploading server!)** a .zip or .pdf file with your answers.
After you submit, you will receive an acknowledgement email that your homework has been received and at what date and time. If you have not received an acknowledgement email within 2 days after you submit then contact Aris.

The solutions for the theoretical exercises must contain your answers either typed up or hand written clearly and scanned.

If you need any technical help, for Problem 5, you can email Gianluca De Carlo: `decarlo@diag.uniroma1.it`. For other questions, email Aris.

For information about collaboration, and about being late check the web page.

Problem 1. In this homework you need to implement some of the models that we have seen, and measure experimentally some of the graph properties. Of course, each model has its own parameters.

1. The Erdös–Rènyi G_{np} random graph model. Parameters:
   - n: number of nodes
   - p: probability of an edge to exist
2. The Watts–Strogatz small-world model. Parameters:
   - n: number of nodes
   - k: number of initial edges adjacent to each node
   - β: probability of rewiring. In particular, for each node, we consider its $k/2$ neighbors to the right, and each of them is replaced with a random node in the graph.
3. The Barabási–Albert preferential attachment model. Parameters:
   - n: number of nodes
   - ℓ: number of neighbors that a newly arrived node comes with.

   Assume that the inistial graph is a single node. If it makes your life easier, if multiple edges fall on the same node you can ignore the multiple edges.

The goal is to understand these models, for different parameter combinations. Therefore, for each of these models, you should experiment for different values of the parameters. However, the parameter n should always be high (at least $10K$ but it can be up to the order of millions depending on the power of your computer).

For each of the parameter combinations, compute and report in an organized way:

- Degree distribution (you should display it with a plot)
- Diameter
- Clustering coefficient
- Other graph parameters that you may think are interesting

For each set of parameters create different graphs and check if the behavior and the values you obtain are the same for each of these graphs.

You are allowed to use a library (such as NetworkX in case you use Python) to handle the graph or compute the graph functions. However, you should implement yourselves the graphs and not use library or other code for that. If you have any questions about what is allowed, feel free to ask Aris.

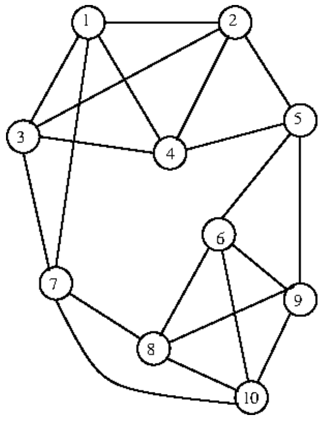
You should hand in a zip file containing the code of your program results and a report, in pdf format, which will contain the results of your findings. In the report, for each model, you should display a table with the different combinations and the values that you obtain, and for each parameter combination a plot depicting the degree distribution.

As an advice for when you try your programs, first try with smaller values of n to make sure that your program works (e.g., 500 or 1000), then you should try the higher ones.

Problem 2. We create a small-world graph G according to the following model. We start with a *directed* cycle with n nodes. That is, we have nodes $v_1, v_2, \dots v_n$ and we have a directed edge from each v_i to v_{i+1} (v_n is connected to v_1). Each of these edges has length 1. In addition there exists another "central" node v, which is connected to each of the nodes v_i with probability p, each choice being mutually independent from all the other choices. Each edge (v, v_i) that exists is undirected and has length 1/2. In other words, we create some shortcuts of length 1 between some pairs of nodes (those for which an edge exists).

1. Consider two nodes v_i, v_j on the cycle, such that the distance from v_i to v_j on the cycle is k. Let $P(\ell, k)$ be the probability that the shortest path between v_i and v_j is exactly ℓ. Calculate $P(\ell, k)$.
2. Compute the distribution $P(\ell)$ of the shortest path between the nodes on the cycle.
3. **Optional, bonus:** Compute the average distance between the nodes on the graph. Assume that n is sufficiently large.

Problem 3. We are given the following graph, $G = (V,E)$.



1. Find the densest subgraph using the greedy algorithm we saw in class.
2. Find a minimum cut.
3. Demonstrate (by calculating λ_2, $\phi(G)$, etc.) that Cheeger's inequalities hold for this graph.
4. Find the cut that satisfies Part 3 (and show that it does).

You may use a computer to compute eigenvalues and eigenvectors, but you must specify in your solution how (what program and what code/commands you used).

Problem 4. Consider a graph modeling a social network, where for each node v we have a score $s(v)$ of its importance (e.g., its PageRank score), which is known. We have a *directed* edge from a node u to a node v if node u is spying node v. If we "approach" a person u, we can corrupt him and then we can find information about all the people he is spying, that is, about all the nodes v such that the edge (u,v) is in the graph. Assume that have enough resources to corrupt k people. Our goal is to find what nodes we should approach, such as to maximize the sum of the importances of all the people that we can spy to. (Note that if we have corrupted more than one person with links to a node u, the score of u is counted only once.) Provide an approximation algorithm for solving this problem, and prove the approximation ratio.

Problem 5. Exercise: Building a Graph Neural Network for Node Classification

In this exercise, you will implement a graph neural network (GNN) model using Python and a popular graph library, such as NetworkX or igraph. Your task is to train the GNN to perform node classification on a social network dataset.

1. Choose some social network datasets: You can use a publicly available dataset, such as the Facebook social circles dataset (`https://snap.stanford.edu/data/ego-Facebook.html`), or any other dataset of your choice. Ensure that the dataset contains node features and class labels for node classification.
2. Preprocess the dataset: Load the dataset into your Python environment and preprocess it as necessary. You may need to handle missing values, perform feature scaling, and split the data into training and testing sets.
3. Build the GNN model: Implement a GNN model using a Python library that supports GNN architectures, such as PyTorch Geometric (PyG) or Deep Graph Library (DGL). Define the layers, activation functions, and any other necessary components of your GNN model.
4. Train the GNN model: Train your GNN model on the training set. Use an appropriate optimization algorithm (e.g., Adam) and a suitable loss function (e.g., cross-entropy loss) for node classification. Monitor the training process and evaluate the model's performance on the validation set at regular intervals.
5. Evaluate the GNN model: Once training is complete, evaluate the performance of your GNN model on the testing set. Calculate relevant evaluation metrics, such as accuracy, precision, recall, and F1-score, to assess the model's effectiveness in classifying the nodes in the social network.
6. Experiment and analyze: Experiment with different configurations of your GNN model, such as varying the number of layers, adjusting hyperparameters, or trying different GNN architectures (e.g., Graph Convolutional Networks or GraphSAGE). Analyze the results and draw conclusions about the impact of these changes on the model's performance.

Optional (bonus): Implement additional techniques, such as graph pooling or graph attention mechanisms, to enhance the performance of your GNN model.

Remember to document your code clearly and provide explanations for each step of the exercise. Good luck with your implementation and analysis!