CS 4/5740 Networks, Crowds, and Markets

Project #1: Random Graph Models

Submission requirements:

- A .zip file containing your source code. You may use any language you would like.
- A PDF (**submitted separately** to the Canvas assignment) containing each item below that is listed as a **Deliverable.** For each item contained in your PDF, clearly mark which deliverable it is associated with. Plots should be clearly labeled and have descriptive captions.

5740 Students:

- You will modify the baseline model in a way of your choosing; see Deliverable 4.

Tips:

- The technical purpose of this project is to gain familiarity with writing code that describes networks/graphs, and to learn a simple model of random graph formation.
- The extra-technical purpose of this project is to start you on a path to curiosity: as you generate these graphs and these plots, you owe it to yourself to take the opportunity to *describe what is happening in your own words.* Any time you come up with a result of some kind, try to explain what the result means in non-technical language.

Random Graphs. One popular topic of study in graph theory applied to social networks is that of processes for generating *random graphs,* where the edges in the graph are added according to some kind of stochastic (randomized) algorithm. One of the reasons researchers are interested in random graph models is that real social networks can be pretty hard to obtain, and even if you could get your hands on something like Facebook's friend graph (hint: Facebook doesn't let most people look at that), it would be so full of real human details that actually studying it would be problematic from an ethics and privacy standpoint. So researchers would like to come up with ways to generate synthetic networks which "look like" real human networks; they want these synthetic networks to be randomized so that they can make many of them and simulate various social processes on them.

Unfortunately, generating random networks that "look like" real social networks turns out to be surprisingly hard to do. In this project, you'll experiment with a simple random graph model. This will give you hands-on experience in thinking about how to ask this course's style of questions using code.

The world's simplest random graph model: There are n vertices, and each edge exists with probability p. In this model, to generate a random graph, you first need to choose 2 parameters:

- n: the number of nodes, and
- p: the probability that each edge exists.

Once you have selected n and p, the process is very simple: you iterate through all the *possible* edges. For each possible edge (i.e., each pair of vertices), add an edge with probability p, and don't add an edge with probability $1 - p$. You can think of this as flipping a coin that comes up "heads" a p fraction of the time, and comes up "tails" a $1 - p$ fraction of the time. For each possible edge, you flip the coin — if it comes up heads, you draw the edge. Otherwise you move on to the next one.

What kinds of graphs does this model create? That's your job to explore! On the extreme ends, it's hopefully obvious that if $p = 0$, you get an empty graph (no edges at all); if $p = 1$, you get a complete graph (all possible edges exist). In between those endpoints, bigger values of p should create a more connected graph.

Assignment:

1. Implement this random graph model in code. You may use any system you like as long as the above idea is implemented faithfully. If you want to provide examples visually to show that your code works (using a visualization tool like NetworkX), clearly state what values of n and p your examples are using.

   Deliverable 1: The code for your random graph model. If you give visual examples of graphs created by your algorithm, please annotate them clearly with values of n and p.

2. Thinking about global connectivity as a function of p. If $p = 0$, the graph is always empty (no edges). If $p = 1$, the graph is always complete (all edges). This means that in between, *in some sense* the graph goes from being empty to having many components, to having a giant component, to eventually being a connected graph. But each graph is random, so in what sense can we think about this transition? Here, we need to ask our questions in terms of statistical ideas like average and percentile. A very simple way to measure global connectivity is by asking *how many components does the graph have?* Your assignment is to explore the relationship between p and the number of components in a graph. Follow the directions in Deliverable 2 precisely.

   Deliverable 2: Choose a fixed value of n (ideally larger than 100). Create a plot which has # of components on the vertical axis, and p on the horizontal axis. For each value of p that you study (you should check *at least* the values $p = 0.1,\ p = 0.2,\ p = 0.3,\ \ldots,\ p = 0.9$), generate 10 (or more) random graphs and plot the # of components in each of those generated graphs. In addition, for each value of p, compute the *average* # of components and plot that using a distinct color.

   You may use any technique you want to compute the # of components. We've already seen an algorithm for answering this question (based on BFS), or you could use a built-in method in a package like NetworkX.

Is there a threshold on p above which all your generated graphs are connected?

3. Thinking about local connectivity as a function of p. If $p = 0$, every node has 0 neighbors (the *degree* of every node is 0). If $p = 1$, every node has $n - 1$ neighbors (the *degree* of every node is $n - 1$). In between, the average degree of nodes increases. Your assignment is to explore the relationship between p and the average node degree on the network. Follow the directions in Deliverable 3 precisely.

   Deliverable 3: Choose a fixed value of n (ideally larger than 100). Create a plot which has average node degree on the vertical axis, and p on the horizontal axis. For each value of p that you study (you should check *at least* the values $p = 0.1,\ p = 0.2,\ p = 0.3,\ \ldots,\ p = 0.9$), generate 10 (or more) random graphs, compute the average node degree (the average number of neighbors over all nodes), and plot the average degree from each of those generated graphs. In addition, for each value of p, compute the *average* (over all generated graphs) average node degree and plot that using a distinct color.

   You may use any technique you want to compute the average node degree.

4. For the graduate section: like I said earlier, creating random networks that look like real social networks is a weirdly challenging problem. For instance, one of the issues with the above random model is that it produces graphs with lower triadic closure than we'd expect to see in a real human network. Your goal is to come up with a modification to this random graph model and then repeat the experiments from Deliverables 2 and 3 using your modification. One idea to get you started is this: What if the probability of an edge appearing is a function of the edges which already exist? Would you expect highly-connected nodes to be *more* likely or *less* likely to form new connections? You could take this in any number of directions; for instance, you could make edges more likely if they complete a triad (lead to triadic closure).

   Deliverable 4 (CS 5740 only): Modify the random graph algorithm in some way of your choosing. Justify the modification using concepts from your daily life, your intuition about networks, or using concepts from the course. Clearly explain why you made the modification you did. Then repeat Deliverables 2 and 3 using your new model. Write a paragraph discussing why your modification did or did not lead to substantial changes in the structure of the networks. **Creativity here will be rewarded!**