Project 3: Giant component of a random graph

Due: 11:59pm, Sunday, May 19

Collaboration Policy You can discuss topics related to programming or graphs with your classmates or others. You may also do research on the internet or from books (this is certainly not required). But you must design and write your own algorithms and write your own report. In particular, you should understand how your code works and be able to justify your design choices.

Introduction

In this project you will explore "giant component" of a random graph. Recall that the vertices of an (undirected) graph G can be partitioned into sets called *connected components*, such that every pair of vertices in each set can be joined by a path consisting of edges of the graph, and no pair of vertices from different sets can be joined by such a path. The number, size, and structure of the connected components are important properties of graphs that arise when modeling networks.

You will investigate the relationship between the connected components (particularly the size of the largest one - the *giant component*) in a random graph and the number of edges in the graph. With few edges, all the connected components are small and there are many of them. As the number of edges increases, eventually it is very likely that the graph will be connected (i.e. have a single connected component). But between these two extremes there is a transition point when the graph starts to have one large component, and many smaller ones. You will empirically explore this transition point.

Programming tasks [70 pts]

(1) `[15 pts]` Write code that generates a random graph G with n vertices, where each edge appears with probability p. This means that two distinct vertices are connected to each other with an edge with probability p.

(2) `[5 pts]` Plot visual representations of random graphs for various values of n up to 50, and various values of p. You can use the package `networkx` or some other way of plotting.

(3) `[20 pts]` Write code that takes in a graph G and returns a list of the connected components. You can choose how you represent the graph; you want to be able efficiently compute the connected components. Test out the code on some of the graphs you generated in the previous part.

(4) `[10 pts]` Optimize your code from the previous part (if necessary) so that you can compute the connected components of random graph G with $n = 1000$, and $p = 1/1000$ in a few seconds

or less. Write a function that takes in a graph and outputs the size of the giant component (measured in number of vertices). Test this for a random graph with $n = 1000$, and $p = 1/1000$.

(5) `[10 pts]` You will now see what happens to the size of the giant component of random graphs for p near $1/n$. If $p = 1/n$, then the number of edges hitting a vertex v (the degree of v) is around 1. Compute the size of the giant component for random graphs with $n = 10, 50, 100, 250, 500, 1000$, and $p = t/n$ where t ranges from 0 to 4 in steps of size 0.05. Do 20 trials for each of the possible (n, p) values and take the average of the size of the giant component over those 20 trials. Generate a plot of your results, with one curve for each of the possible values of n. Make the x-axis the value of pn (which is the average degree of a vertex), and the y-axis the value of the size of giant component divided by n.

From this plot, you should see the emergence of a giant component as p increases from below $1/n$ to above this threshold.

(6) `[10 pts]` Do experiments and create a plot as in the previous part, but this time instead of computing the size of giant component, compute the ratio of the size of the giant component to the size of the second largest component.

This quantity should also experience a transition around $p = 1/n$; below this threshold the giant component isn't much bigger than the second largest component, while above the threshold, the giant component gets much bigger than the second largest component.

Analysis and report [30 pts]

- Describe the algorithm that you used to compute connected components in (3). How do you think the running time would scale with the number of vertices n and the number of edges in the input graph?
- Describe any optimizations you had to make in (4) to meet the run-time condition.
- Interpret the results about the size of the giant component you computed in (5). What kind of transition occurs around $p = 1/n$? What do you think would happen if you were able to test random graphs with very large n?
- Interpret the results from (6).
- What are some other questions about properties of random graphs that you could explore in a similar fashion?

What to turn in

You will submit everything on Blackboard. Include the following:

- A `.ipynb` notebook file named `code_firstname_lastname.ipynb` with all your code from the **Programming tasks** section. Make sure to clearly separate the various parts. Also, test this notebook by restarting it and running from the beginning; this should produce no errors.
- A `pdf` file named `report_firstname_lastname.pdf` with your work from the **Analysis and report** section. The exact format of this report is up to you. It can be hand-written (legibly) and then scanned (legibly) into a `pdf` document. Or it can be be made on a computer. It must be in `pdf` form (it is generally easy to convert other forms such as .doc to `pdf`).