ECS 253 / MAE 253, Network Theory and Applications
Spring 2016
Advanced Problem Set # 4, Due May 24
Topic: Configuration model random graphs

Note 1: Some of the problems herein require you to implement computer code. The purpose is to teach you generic skills about how to build a random graph and how to run simulations on a graph. For these problems, you can either use your favorite graph library (e.g., networkX, igraph, ...) or do everything from scratch (e.g., by implementing your own adjacency list as a vector of vector). Although we encourage you to use a graph library, you should only call "basic graph operations" from such libraries. Indeed, calling a "high level" function such as `nx.configuration_model` defeats the purpose of learning how to build your own random graph. The following operations are "basic graph operations": creating a network with N nodes and no edges; adding/deleting a node; adding/deleting an edge; requesting who are the neighbors of a node; storing information in a node/edge (i.e., node/edge properties). Use common sense.

Note 2: This document uses the first N natural numbers (i.e., $1, 2, 3, \cdots, N-1, N$) to refer to each node of a graph. Depending of the programming language you are using, indexing the elements of a vector may start at 0 or 1. Hence, you are free to use the first N non-negative integers (i.e., $0, 1, 2, \cdots, N-2, N-1$) in your computer code if you want to.

1 Building a configuration model random graph

Let $\mathbf{k} = (k_1, k_2, k_3, \cdots, k_N)$ be a vector of N non-negative integers such that $\sum_{i=1}^{N} k_i$ is an even number. An instance of the configuration model random graph with degree sequence $\mathbf{k}$ can be obtained as follows.

- Make sure that $\sum_{i=1}^{N} k_i$ is an even number (give an error if not).
- Create an undirected graph containing N nodes and no edges.
- Create a vector (or list or multiset...) such that, for each $1 \leq i \leq N$, it contains k_i copies of i. This object will hereafter be known as the "stub list". *Example: The vector* $(1, 2, 3, 4, 5, 5, 6, 6, 6, 7, 7, 7)$ *is a valid stub list in the case* $\mathbf{k} = (k_1, k_2, k_3, k_4, k_5, k_6, k_7) = (1, 1, 1, 1, 2, 3, 3)$.
- Sample uniformly at random one element from the stub list; call that element i and remove it from the stub list. Sample uniformly at random another element from the (now shorter) stub list; call that element j and remove it from the stub list. Add an edge between node i and node j in the network. Repeat this step as long as the stub list is not empty.

The resulting network will be a random graph with degree sequence $\mathbf{k}$. For the purpose of this problem, we do not worry about repeated edges (i.e., more than one links between two nodes) and self loops (i.e., a node with an edge to itself).

(a) Write a computer implementation of this algorithm.

(b) Using the degree sequence $\mathbf{k} = (1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3)$, generate a configuration model random graphs and display the result as a figure.

(c) Let $\mathbf{n}$ be a vector such that the kth component represents the number of nodes of degree k. For example, $\mathbf{n} = (0, 8, 4, 2)$ corresponds to the $\mathbf{k} = (1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3)$. Write a function that receives $\mathbf{n}$ as an input and returns $\mathbf{k}$.

(d) Let $\mathbf{p}$ be a vector such that the kth component denotes the probability of having a node of degree k. Write a function that receives $\mathbf{p}$ and returns $\mathbf{k}$.

2 Percolation and spreading

Consider the three functions described below.

`percolation`:

- Receive as input a graph G and a real number p such that $0 \leq p \leq 1$.
- Make a graph with no edges and as many nodes as G has; call this graph without edges G'.
- Iterate over all the edges of G. Suppose the current edge is between nodes u and v. Get a random number uniformly distributed in the interval $[0, 1)$. If that number is lower than p, add in G' an edge between nodes u and v.
- Return the graph G'.

Hence, each edge of G has probability p to be present in G'.

`spreading`:

- Receive as input a graph G, a node index v, and a real number T such that $0 \leq T \leq 1$.
- "Mark" all nodes as "unreached" by one of the following two methods:
  - Create a vector of bool called `is_reached` containing as many entries as there are nodes in G. Initialize all its entries to "False"; **OR**
  - Give to each node in G the bool "node property" `is_reached`. Initialize them all to "False".
- Create an empty vector (or other appropriate container) of indices; call it `unresolved`, and place v in it.

- Set an integer variable `number_reached` with value 1.
- Repeat the following as long as `unresolved` is not empty. Get in u the value of an element of `unresolved`, and remove said element from `unresolved`. If u is marked as unreached (i.e., if `is_reached` is "False" for that node), do the following:
  - Increment `number_reached` by one.
  - Mark u as reached (i.e., set `is_reached` to "True" for that node).
  - For each neighbor w of u, generate a random number uniformly distributed in the interval $[0, 1)$. If the number is lower than T, add w to `unresolved`.
- Return `number_reached`.

`component_size`:

- Receive as input a graph G and a node index v.
- Call your `spreading` function for the graph G, the node index v, and using $T = 1$.
- Return the `number_reached` returned by the aforementioned function.

(a) The fifth bullet of `spreading` does not specify *which* element of `unresolved` should be removed to become u. Does the outcome `number_reached` depend on this choice? Why?

(b) Explain why the value returned by `component_size` corresponds to the size of the component to which u belong.

(c) Let G' be a graph returned by `percolation` (with parameters G and p). Show (by hand) that calling `spreading` with the parameters G', v and T is statistically equivalent to calling `spreading` with the parameters G, v and pT. From this result, deduce a relationship between `spreading` and the size of the component to which v belong in a graph returned by `percolation`.

(d) Implement these three functions.

(e) We shall now use the code you have written up to this point to understand percolation/spreading on a random network with a given degree sequence. For a given graph, you will simulate spreading with a given probability of transmission (T) and compute the probability distribution for the total number of nodes reached starting from a node at random. The process is described below. Write code to do the following:

- Receive T, **n** as inputs.
- Initialize a vector **res** to store the result. If the number of nodes in the network is N this vector has size N. Since spreading process could reach at most N nodes (and always starts with one random node).
- Repeat the following steps `number_simulations` times:

  - Create a configuration model random graph with $\mathbf{n}$ as the input.
  - Do $\mathcal{A}$ or $\mathcal{B}$ (See below). Store the result in S
  - Increment the Sth entry of the vector **res** by 1.

- Normalize the vector **res** such that component i gives the probability of the process spreading to i nodes starting from a random node. This is the result we are interested in.

$\mathcal{A}$: Call `spreading`.
$\mathcal{B}$: Call `percolation` and then `component_size`.

(f) Run the above process for $\mathbf{n} = (0, 8, 4, 2)$ and $T = 0.4$. Report the result.