Problem Set Two, CR15 ENS de Lyon. Course authorities : Karsai / Crespelle / Unicomb

Due date : 01/12/2016, by 11:59 p.m.

Complementary material : N/A

1 Introduction to the SNAP grap library

In the last problem set you developed your own tools in order to output key properties of simple graphs. In this tutorial, you have the option of either developing your own tools to solve the problems, or using the graph library SNAP, or Stanford Network Analysis Platform. This graph library was originally developed by a PhD student at Stanford, Jure Leskovec.

Snap.py is a python interface for the SNAP library, which is a high performance platform well suited to the analysis of large graphs. From the Stanford website, "the core SNAP library is written in C++ and optimized for maximum performance and compact graph representation. It easily scales to massive networks with hundreds of millions of nodes, and billions of edges".

You will find the download and installation instructions at https://snap.stanford.edu/snappy/index.html

The API for SNAP is straightforward, for example,

- Graph = snap.TUNGraph.New() initialises an unweighted and undirected graph,
- Graph.AddNode(1) and Graph.AddNode(2) add nodes 1 and 2 to the graph just generated, and
- Graph.AddEdge(1,2), you guessed it, adds an edge between nodes 1 and 2.

1.1 Simple applications of SNAP

The aim here is not to develop a complicated routine, but to give you a first exposure to SNAP, so that you feel comfortable calling upon it later, if you so decide.

- Create a random graph with 10 nodes and 15 edges, outputing the degree distribution to a file.
- Compute the sizes of the connected components and output these to an external file.
- Output the average of the shortest paths, adding more edges to the graph if it's not connected.
- Get the clustering coefficient *per node* and output this information to an external file, not confusing this metric to the slightly different but identically named quantity from the last problem set. Calculate the average of these clustering coefficients.

In the following sections you have the freedom to choose your own tools, be it SNAP, your "Graph" class from the first problem set, or python functions that you may develop from scratch here. Whatever you decide, present me something convincing that I can easily follow, run and mark, even at 2 o'clock in the morning.

2 The Erdös-Rényi random graph model

Here we calculate some properties of the ER random graph, a staple model in graph theory. There are two closely related variants of the ER model. First, the $G(n, p)$ graph, with n nodes and a constant probability p of an edge existing between any pair of nodes. Second, is the $G(n, m)$ graph, which has n nodes with m edges randomly distributed between node pairs. Both variants have the restriction of being simple graphs,

disallowing loops and multiple edges. This means that, as nodes are distinguishable, each graph is drawn from the sets $G_{n,p}$ and $G_{n,m}$ with probability

$$p^m(1-p)^{C_2^n - m}, \tag{1}$$

where C_2^n is the binomial coefficient. We can think of p as a parameter that interpolates between empty graphs, with $p = 0$, and complete graphs, with $p = 1$.

2.1 Generating ER graphs

Write two functions, *ERnp* and *ERnm*, that generate Erdös-Rényi graphs, each depending on two input paramaters, n and p or n and m.

2.2 Degree distributions of random graphs

Write a function *compare_edge_count* that takes in the parameters n and p, and outputs the ratio of the number of edges in the corresponding $G_{n,p}$ and $G_{n,m}$ graphs, where $m = p \times C_2^n$, but of course integer.

Show, using plots if necessary, that your graph generation methods obey the theoretical degree distribution

$$P(k) = C_k^{n-1} p^k (1-p)^{n-k-1}, \tag{2}$$

where $P(k)$ is the probability of a node having degree k.

3 The Erdös-Rényi model and connected components

In their 1960 paper, Paul Erdös and Alfréd Rényi provided a deep analysis of the class of graphs $G_{n,p}$, studying the characteristics of the graphs as a function of the parameter p. Their results demonstrated that

- if $np < 1$, a graph in $G_{n,p}$ will almost surely have no connected components of size larger than $\mathcal{O}(\log n)$,
- if $np = 1$, a graph in $G_{n,p}$ will almost surely have a largest component whose size is $\mathcal{O}(n^{2/3})$,
- if $np \to c > 1$, where c is a constant, then a graph in $G_{n,p}$ will almost surely have a unique giant component containing a positive fraction of the vertices. No other component will contain more than $\mathcal{O}(\log n)$ vertices. Furthermore,
- if $np < (1-\epsilon)\log n$, a graph in $G_{n,p}$ will almost surely contain isolated vertices, and
- if $np > (1+\epsilon)\log n$, a graph in $G_{n,p}$ will almost surely be connected.

From the final two points, we observe that $\log n/n$ is considered a sharp threshold with repect to p. Note however that we are normally working in the limit of large n, where $p > \log n/n$ is satisfied anyway.

In this section, the goal is to carry out your own analysis to convincingly illustrate that your random graph methods obey the above properties. Remember that Erdös and Rényi proposed a probabilistic model, so you are not expected you to provide a proof. Rather, design some experiments showing that under the right conditions, the above characteristics are exhibited with high probability.