Network Analysis Lab

Social Data Management

December 3rd, 2018

The goal of this lab session is to analyse the basic properties of a social graph, using Python and the `networkx` package, by using a Jupyter notebook.

Documentation to support this lab can be found at:

- NetworkX: `https://networkx.github.io/documentation/stable/`,
- Matplotlib: `https://matplotlib.org/`,
- Tutorial on how to use NetworkX and Matplotlib in Jupyter: `https://github.com/networkx/notebooks`.

1 Installation and First Steps

1. If not present, install Jupyter and the NetworkX and Matplotlib Python packages.
2. Download the Jupyter notebook called `network_analysis_lab.ipynb` and the graph `karate`. The files must be in the same folder.
3. Run the Jupyter notebook, and check that all cells are executed correctly.
4. Using the examples already present in the notebook and the Networkx documentation, compute the diameter of the graph and the average distance in the graph. How do they compare?
5. Plot the distribution of the PageRank values in the graph. How does it compare to the distribution of the degrees (already plotted)?

2 Comparing `karate` to Random Networks

In this exercise, we aim to compare the results of the average distance and assortativity values in the graph with those of a random network having the same characteristics.

1. From the average degree of `karate` and its number of nodes, compute the corresponding values of p in a random network having the same characteristics.
2. Generate a random graph with the same p and N, using the function `networkx.gnp_random_graph` and plot it.
3. Compare the values of the average distance, assortativity, and clustering coefficient with the ones predicted *in expectation* in random graphs with the above p and N parameters.

3 Counting Triangles in Graphs

An important measure in graph analysis is the number of triangles it contains. A *triangle* of a graph G is a connected subgraph of G having 3 nodes, or a 3-clique.

Your task is to count the number of triangles in the graph, *without using the NetworkX implementation.* Proceed in two steps:

1. Compute the number of triangles in which each node is involved. The output should be a dictionary containing the node as a key and the number of triangles as a value.
2. For testing, compare your output with that of the `networkx.triangles` method.
3. Using the dictionary from step 1, output the total number of triangles in the graph.
4. Compute the expected number of triangles in a random graph having the same average degree. How does it compare to the one found in reality?

4 Community Detection

Implement the divisive community detection (Ravasz) as presented in the course slides. In short, the following steps should be followed, starting from a single community, i.e., the connected graph:

1. **Centrality computation**: in this lab we will use the *betweenness centrality* of each edge; use the NetworkX implementation as exemplified in the notebook.
2. **Clustering**: remove the edge having the highest centrality, and recompute the betweenness.
3. **Communities**: the communities correspond to the connected components of the graph; these should be output at every step.
4. Repeat steps 2 and 3 until every node is in its a different community.

At every step, compute the resulting *modularity.* Output only the partition corresponding to the maximal modularity.

On the original graph, draw the communities corresponding to the maximum modularity, by using per-community colors on each node.