NETWORK SCIENCE ANALYTICS

CENTRALESUPÉLEC

Lab 1: Centrality Criteria

Instructor: Fragkiskos Malliaros
TA:Alexandre Duval

31 janvier 2022

Description

The aim of this first lab is to help you familiarize with the fundamental elements of network analysis and the processing of graphs via the *NetworkX*[1] Python package. You will also be asked to implement various centrality measures seen in class and to check the robustness of a graph.

Part I: Analysis of a real-world network

Exercise 1: Basic properties of the network

In this exercice, you will analyze the different structural properties of the `NetScience` collaboration network. This co-authorship network has been compiled from the bibliographies of two review articles and covers the network of researchers working on the field of network science. If an author i has co-authored a paper with an author j, the graph contains an undirected edge from i to j. Here, the graph is unweighted.

The graph is stored in the `NetScience.edgelist` file[2], as an edge list:

```
# Undirected graph: NetScience.edgelist
# Collaboration network of researchers working on network theory  (there
is an edge if authors coauthored)
# Nodes: 1461 Edges: 2742
# FromNodeId ToNodeId
344 342
344 343
...
```

For the following questions, it is recommended to use the *NetworkX* package of Python. You can install it using `pip install networkx` and access its documentation at `https://networkx.org/documentation/stable/reference/index.html`.

1. Load the network data into an undirected graph G, using the `read_edgelist()` function. Note that the delimeter used to separate values is the tab character $\backslash t$ and that the text following the `#` character correspond to comments. The general function syntax is detailed in the online docu.

[1] https://networkx.github.io/

[2] The data can be also downloaded from the following link: `http://vlado.fmf.uni-lj.si/pub/networks/data/collab/netscience.htm`.

```
read_edgelist(path, comments='#', delimiter=None, create_using=None,
    nodetype=None, data=True, edgetype=None, encoding='utf-8')
```

2. Complete the required fields in the `compute_network_characteristics()` method in order to find the following network characteristics: (1) number of nodes, (2) number of edges, (3) minimum degree, (4) maximum degree, (5) mean degree, (6) median degree of nodes in the graph and (7) density of the graph. For this task, you can use the built-in functions `min, max, median, mean` of the NumPy library, and the following method provides an iterator for $(node, degree)$ pairs:

```
graph.degree()
```

   Note that, the density of an undirected graph is defined as $2m/n(n-1)$, where n is the number of nodes and m is the number of edges.

Exercise 2: Connected components of the graph

A *connected component* of an undirected graph is a subgraph where every pair of nodes is connected and there is no any connection between the subgraph and the rest of the graph. In this exercise, we will consider the largest connected component of the network (also called Giant Connected Component – GCC).

1. A graph can be composed by many connected components, so we will firstly check whether the network is connected or not using the following method:

```
is_connected(G)
```

2. If it contains more than one connected component, we will extract the largest connected component (GCC). A generator for the sets of nodes of connected components of G can be obtained using the following function:

```
connected_components(G)
```

   You can use `G.subgraph(nodes)` to obtain the subgraph containing given the list of nodes.

3. Now, we can find the number of nodes and edges of the largest connected component (GCC) and examine in what fraction of the whole graph they correspond to.

Part II: Centrality Measures

Exercise 3: Implementation of Centrality Measures and Visualization of the Network

In this exercise, we will implement various centrality measures and visualize the `NetScience` network with respect to the centrality values of the nodes.

1. One of the most intuitive ways on detecting important nodes on the network is to look at their number of neighbours. That way, the *degree centrality* is defined as the degree of a node in an undirected graph. Fill in the following function to find degree the centrality of each node:

```
compute_degree_centrality(graph)
    degree_centrality = {}
```

```
    ...
    ...
    return degree_centrality
```

2. The *closeness centrality* of a node v in a connected graph is defined as the reciprocal[3] of the mean of shortest path distances between v and all other $n-1$ nodes. More formally, it can be written as follows:

$$C(v) := \frac{n-1}{\sum_{u \neq v} dist(v,u)} \tag{1}$$

In other words, we can say that a node is more central if it is closer to all the remaining nodes. Fill in the missing part of the following function to compute the closeness centrality of every node. You can use the `single_source_shortest_path_length(G,source,cutoff=None)` method in order to compute the shortest path lengths from a *source* node to all the other nodes.

```
def compute_closeness_centrality(graph):
    closeness_centrality = {}
    ...
    ...
    return closeness_centrality
```

3. The *harmonic centrality* of a node v is the sum of the reciprocal of the shortest path distances from v to all other nodes.

$$C(v) := \sum_{u \neq v} \frac{1}{dist(v,u)} \tag{2}$$

You can compute the *harmonic centrality* for each node by completing the following function:

```
def compute_harmonic_centrality(graph):
    harmonic_centrality = {}
    ...
    ...
    return harmonic_centrality
```

4. Figure 1 visualizes the graph using the `visualize(graph, values, node_size)` function (defined in the `helper.py` file) assigning different scores to each node. For this example, the network is visualized according to randomly assigned values to nodes. You can similarly visualize the graph based on the various centrality criteria defined above. That way, we will be able to examine how the structural position of a node in the graph is also related to its importance indicated by different centrality criteria.

5. (Optional) The k-core of a graph is a maximal subgraph where every node has degree at least k. In other words, it is a subgraph that can be obtained by recursively removing nodes having degree less than k. The *coreness* or *core number* of a node is the largest value k of the k-core subgraph that includes the node. Fill in the missing parts of the following function to find the core number of each node:

[3] reciprocal of x is 1/x

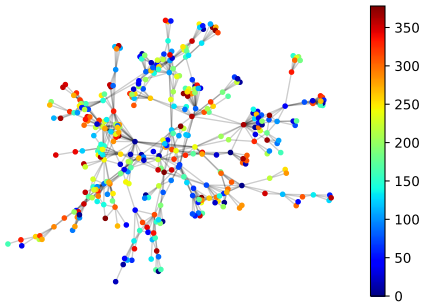

Figure 1: A visualization of the largest connected component of the `NetScience` network.

```
def compute_core_number(graph):
    core_number = {}
    ...
    ...
    return core_number
```

6. (Optional) As we have seen in the lecture, typically many nodes of the graph will have the same core number. Nevertheless, a node having neighbours with high coreness can be considered important. This way, we define a new centrality criterion for a node v, called *neighborhood coreness*, as follows:

$$C(v) = \sum_{u \in N(v)} cn(u) \tag{3}$$

where $cn(u)$ denotes the core number of u, and $N(v)$ is the neighbors of the node v. Please fill in the missing parts of the function below to compute the neighborhood coreness of the nodes of the graph:

```
def compute_neighborhood_coreness(graph):
    nb_coreness = {}
    ...
    ...
    return nb_coreness
```

Part III: Robustness of the Network

In the last part of the lab, we will work on the `CA-GrQc` collaboration network and examine its robustness. The arXiv GR-QC (General Relativity and Quantum Cosmology) collaboration network has been extracted from the e-print arXiv (arxiv.org) and covers scientific collaborations between authors for papers submitted to General Relativity and Quantum Cosmology category. If an author i co-authored a paper with author j, the graph contains an undirected edge from i to j.

Here we consider that the graph is unweighted; it is stored in the `ca-GrQc.edgelist` file.

Exercise 4: Analysis of the robustness

The robustness of a network is related to the capability of the network to retain its structure and connectivity properties, after losing a portion of its nodes and edges. Let us focus our attention to the case where the nodes of a network are deleted based on two strategies:

- *Random deletion*: delete a randomly selected node.
- *Targeted deletion*: delete a node chosen among the ones with the highest centrality measure in the network.

Depending on the type of network, the above two strategies can simulate various scenarios. For example, in the case of the Internet graph (nodes correspond to routers and edges capture physical connections between routers), a random deletion can be interpreted as an error that occurred in the network, where a router switched off due to technical problems (e.g., electricity problems). On the other hand, the targeted removal of nodes, for instance having high degree centrality, can simulate the case of an attack to the network, where the removal of nodes aims to cause a big damage to the network.

How the structure of the network is affected after random and targeted deletions of nodes? It has been observed that, due to the existence of the heavy-tailed degree distribution, real-world networks tend to be robust under random removal of nodes (e.g., errors) and vulnerable under the attacks to high degree nodes. The notion of robustness can be quantified by structural characteristics of the network, such as the fragmentation of the network into disconnected components. For instance, we can examine how the fraction of nodes belonging to the largest connected component (GCC) is affected by the random/-targeted removal of nodes. Here, your goal will be to examine the robustness of the `CA-GrQc` network, with respect to the above strategies.

For this question, you will work with the GCC of the graph (i.e., your initial graph is the GCC of the `CA-GrQc` network). To assess the robustness, you should examine how the size of GCC / other connected components is affected, after removing a certain fraction of nodes chosen either randomly or based on some centrality criterion.

1. Choose a centrality measure for the targeted attack scenario. Alternatively, you can use the following function to assign random importance values to the nodes for the random deletion strategy.

```
assign_random_values(graph, min_value=0, max_value=None,
replace=False)
```

2. Plot the number of connected components vs. the number of removed nodes.
3. Plot the ratio of the GCC size (against the full graph) with respect to the number of deleted nodes. On the same graph, also plot the ratio of the number of nodes in the rest of the graph (i.e. all other connected components), still wrt the number of nodes removed.

```
def plot_robustness_analysis(graph, node2values, k):
    num_connected_components = []
    gcc_sizes = []
    rest_sizes = []
    ...
    ...
```

4. Discuss briefly your observations depending on chosen strategies.

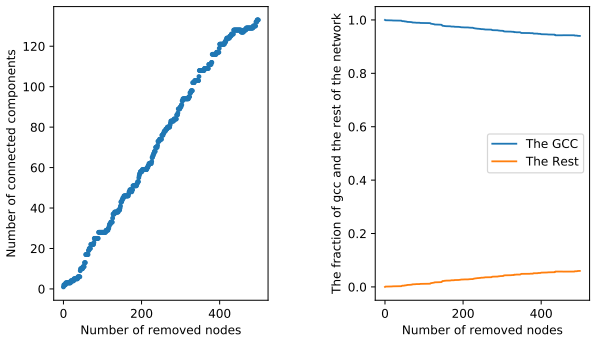

Figure 2: The robustness of the network for random deletion strategy.

For this question, `plot_robustness_analysis()` function was partially implemented for you, so you can use it to draw the figures. An example of plots is provided with Figure 2, where we randomly removed 500 nodes of the network.