ST2: Mathematical Modeling of Propagation Phenomena
Propagation on Graphs

Instructor: Fragkiskos Malliaros
TA: Abdulkadir Çelikkanat

Wednesday, December 12, 2018

Description

In this lab, our goal is to find the most important nodes of a network. Those nodes can vary depending on how we define importance – so we will examine some commonly used centrality measures and observe their differences.

Part I: Centrality Measures

Exercise 1: Implementation of Centrality Measures and Visualization of the Network

In the first 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 neighbors. 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 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. 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:

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

5. As we have seen in the lecture, typically many nodes of the graph will have the same core number. Nevertheless, in many practical applications our goal is to obtain a ranked list of the nodes based on their coreness score. The main observation here is that a node that has neighbors with high coreness, it can be more important than other nodes. That way, we can 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
```

6. 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.

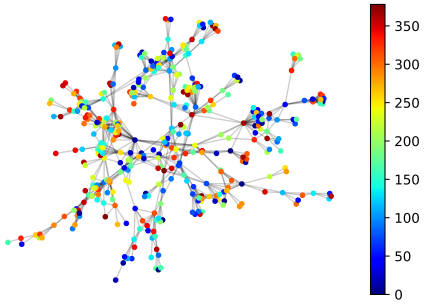

Figure 1: A visualization of the largest connected component of the `NetScience` network.

Part II: Comparison of Centrality Measures

Exercise 2: Vizualization and Comparison of Centrality Criteria

In this exercise, we will examine the correlation between different centrality measures on the largest connected component of the `NetScience` network.

1. Extract the centrality values of each node for two different measures of your choice and compute their Pearson correlation coefficient[1]. You can use the built-in `pearsonr` function[2] to compute the correlation.
2. Additionally, plot the values of two centrality measures (i.e., the first centrality measure vs. the second one) in order to visually observe potential correlation.

For this exercise, you are encouraged to use built-in functions[3] of `NetworkX`, such as `eigenvector_centrality`, `betweenness_centrality` and `katz_centrality` in addition to the measures implemented in the previous part. As an example, the relationship between the closeness and betweenness centrality measures is illustrated in Figure 2 and their correlation value is equal to 0.43023.

Part III: Robustness of the Network

In the last part of the lab, we will work on the `CA-GrQc` collaboration network to examine its robustness. The arXiv GR-QC (General Relativity and Quantum Cosmology) collaboration network has been

[1] https://en.wikipedia.org/wiki/Pearson_correlation_coefficient
[2] https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.pearsonr.html
[3] https://networkx.github.io/documentation/stable/reference/algorithms/centrality.html

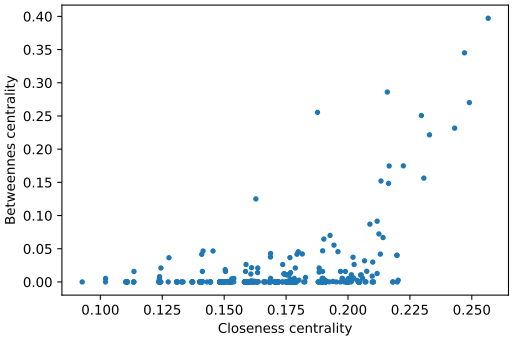

Figure 2: Relationship between *closeness* and *betweenness* centrality measures.

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 3: 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 scenaria. 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 that belong to the largest connected component (GCC) and the rest isolated components 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, we 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 and the rest components is affected, after removing a certain fraction of nodes chosen either randomly or based on

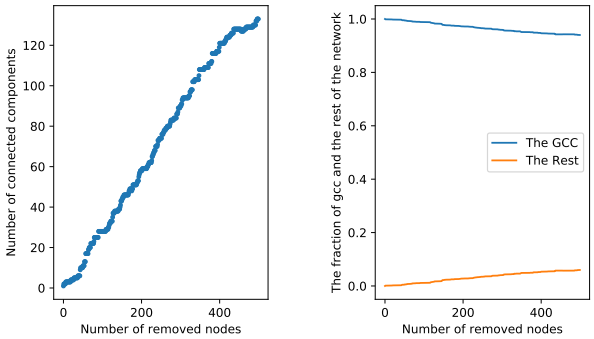

Figure 3: The robustness of the network for random deletion strategy.

some centrality criterion.

1. Choose a centrality measure for the targeted attack scenario. Moreover, you can use the following function to assign random 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 size of GCC and the size of the rest components (sum of their sizes) to the size of the whole graph vs. the number of deleted nodes, for both strategies.

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

4. Discuss briefly your observations depending on chosen strategies.

For this question, `plot_robustness_analysis()` function was partially implemented for you, so you can use it to draw the figures. An example is illustrated in Figure 3 by randomly removing 500 nodes of the network.