

# GRAPH ANALYTICS AND APPLICATIONS

## ÉCOLE CENTRALE CASABLANCA

### Lab 1: Centrality Criteria

Instructor: Fragkiskos Malliaros

January 6, 2026

#### Description

The aim of this first lab is to help you familiarize with the fundamental elements of network analysis. In particular, you will learn how to manipulate graphs using the widely used *NetworkX*<sup>1</sup> Python package. Then, you will be asked to implement various centrality measures seen in class, as well as to evaluate the robustness of a graph.

*Networkx* documentation:

<https://networkx.org/documentation/stable/reference/index.html>

#### Recommended Set up

To perform this lab and all future occurrences, make sure that you have a clean python environment (conda, virtualenv) and access to Jupyter Notebooks. Download the lab session on Edunao. Open the *lab1-jupyter.ipynb* notebook and install the required packages. An example is provided below.

```
# Create env .
conda create -n sdd-centrale-casa
conda activate sdd-centrale-casa
# Install ipykernel
conda install ipykernel
# Check pip is installed
pip -V
# Check python version
python --version
# Install matplotlib
pip install matplotlib
# Install networkx
pip install networkx
# Install scipy
pip install scipy
# Load the downloaded lab1-jupyter.ipynb notebook and select the sdd-centrale-casa kernel.
```

---

<sup>1</sup><https://networkx.github.io/>

## Part I: Analysis of a real-world network

### Exercise 1: Basic properties of the network

In this exercise, you will analyze the different structural properties of the NetScience collaboration network. This co-authorship network 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 of your folder<sup>2</sup>, 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.

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

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

2. Complete the `compute_network_characteristics()` method to print 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 are no connections between the subgraph and the rest of the graph. In this exercise, we will consider the largest connected component of the network.

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

```
is_connected(G)
```

<sup>2</sup>It can be downloaded from: <http://vlado.fmf.uni-lj.si/pub/networks/data/collab/netscience.htm>.

2. If it contains more than one connected component, extract the largest connected component (gcc). The node composition of connected components can be obtained using:

```
connected_components(G)
```

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

3. Find the number of nodes and edges of the largest connected component 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 to detect important nodes in 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<sup>3</sup> of the mean of shortest path distances between  $v$  and all other  $n - 1$  nodes. More formally, it is written as:

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

In other words, we say that a node is central if it is close to many other nodes. Fill in the corresponding 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 distance 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.

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

---

<sup>3</sup>reciprocal of  $x$  is  $1/x$

Fill in the following function:

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

4. You can visualize the graph using the `visualize(graph, values, node_size)` function (defined in the `helper.py` file), which assigns different scores to each node. In this example, the network is visualized according to randomly assigned values to nodes. Change this piece of code to visualize the graph based on the different centrality criteria. Observe which nodes are found important according to the chosen method.
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:

```
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:

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

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 size of GCC with respect to the number of deleted nodes. On the same graph, also plot the ratio of the sum of the size of other connected components.

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

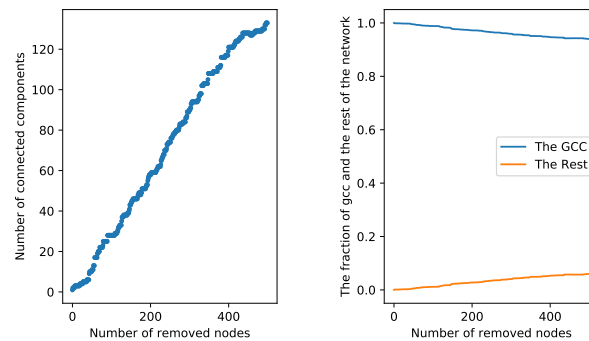


Figure 1: The robustness of the network for random deletion strategy.

...

...

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 of plots is provided with Figure 1, where we randomly removed 500 nodes of the network.