practical_2_part_1_network_robustness_solutions

October 2, 2024

Name: Write your name here

1 SYSTEMS AND NETWORK BIOLOGY - PRACTICAL 2 (PART 1)

2 Robustness of gene regulatory networks

To submit your report, answer the questions below and save the *notebook* clicking on `File > Download as > iPython Notebook` in the menu at the top of the page. **Rename the notebook file** to "`practicalN_name1_name2.ipynb`", where `N` is the number of the practical, and `name1` and `name2` are the first surnames of the two team members (only one name if the report is sent individually). Finally, **submit the resulting file through the *Aul@-ESCI***.

Remember to label the axes in all the plots.

*IMPORTANT REMINDER: Before the final submission, remember to **reset the kernel** and re-run the whole notebook again to check that it works.*

In this session we will study two publicly available biological networks. We will assess their robustness to attacks. To that end we will use the Python package `NetworkX`.

3 0. NetworkX instructions

These are the main `NetworkX` instructions that you will need to use in this practical:

- `<graph>.to_undirected()` returns an undirected graph representation of the `<graph>` directed graph object. This allows to use some of the Networkx methods that only work with undirected graphs.
- `<graph_copy> = <graph>.copy()` returns a copy of the `<graph>` graph object.
- `<graph>.number_of_nodes()` returns the number of nodes of the `<graph>` object.
- `<graph>.number_of_edges()` returns the number of edges of the `<graph>` object.
- `<graph>.nodes()` returns a list with all the nodes in `<graph>`.

- `networkx.adjacency_matrix(<graph>)` creates the adjacency matrix of a network. The matrix is produced in the `SciPy` sparse format. You should use the `todense` method of this data type to generate a `numpy` matrix.
- `networkx.number_connected_components(<graph>)` returns the number of connected components of the undirected graph `<graph>`.
- `networkx.connected_component_subgraphs(<graph>)` returns a list of the connected components in the undirected graph `<graph>`. The components are ordered from the largest to the smallest. ***Use the function below if you receive an error here***
- `<graph>.subgraph(c) for c in nx.connected_components(<graph>)` does the same as the function directly above. Use this if you receive an error when trying to use the function above.
- `<graph>.remove_node(<node>)` removes the node with name `<node>` from the `<graph>` graph.
- `networkx.draw_networkx` draws the network. It is convenient to use the `pos` argument with the `spring_layout` function of NetworkX to position the nodes such that the network is not too tangled.

First, load all necessary Python modules (including NetworkX).

4 1. Network analysis

There is a large number of databases of cellular regulatory networks (involving metabolic reactions, protein-protein interactions, gene regulation, …). In this practical session we will work with transcription factor networks, which can usually be treated as directed simple graphs. We will the **RegulonDB** database, which contains most known gene regulatory interactions in *Escherichia coli*. To download the database, access the RegulonDB website, then click **Downloads** and then **Experimental Datasets**. Finally, download the file labeled **TF - gene interactions**, and place it in the folder where you will be working.

To load the network you can use the following function:

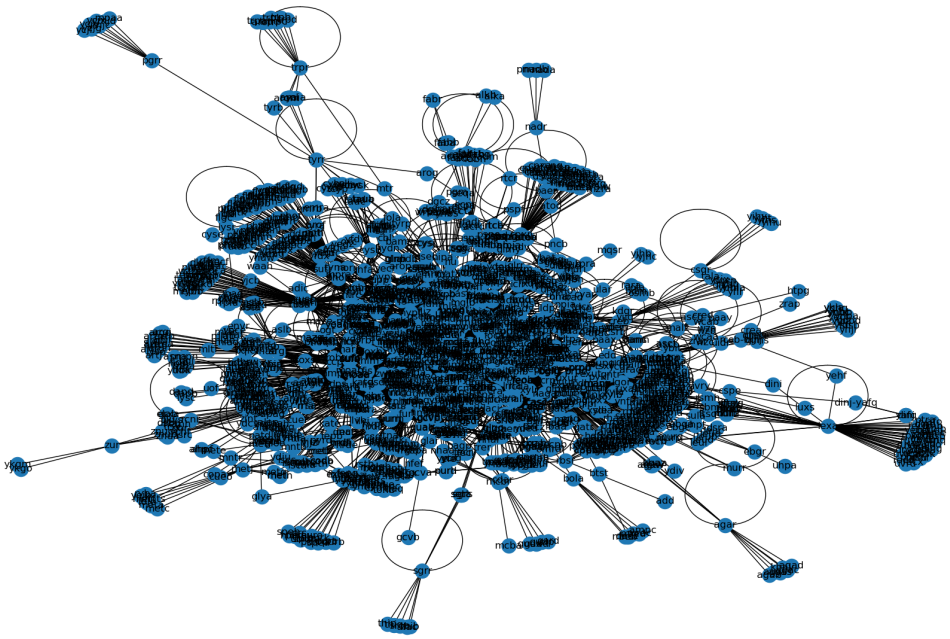
Load the database into a graph object and calculate the number of genes (nodes), interactions (edges) and the mean degree.

```
Number of genes: 1909
Number of edges: 4439
Mean degree: 4.650602409638554
```

Comment your result here

How many *connected components* (a set of vertices in a graph that are linked to each other by paths) does the network have? You will have to convert the network to an undirected graph first. Select the largest component of the network (save it to a new graph) and repeat the calculations of number of nodes, number of edjes and mean degree. Draw this new subgraph.

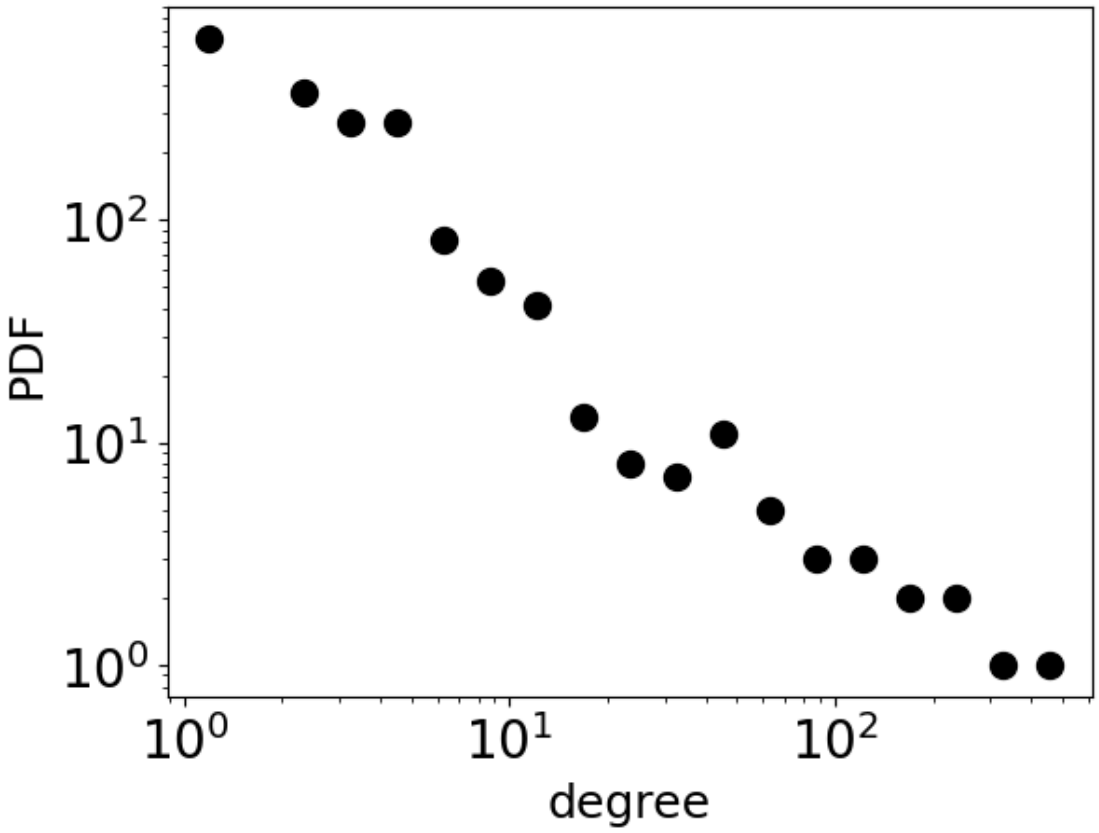
```
Number of connected components = 27
Number of genes: 1802
Number of edges: 4328
Mean degree: 4.803551609322975
```



Comment your result here

Next, create the adjacency matrix of the network, and use it to compute the degrees of all nodes and the average degree of the network. Print the average degree and plot the degree distribution of the network **(do NOT use specially dedicated functions of NetworkX for those tasks)**.

```
Average degree =  4.803551609322975
```



Comment your result here

---

5 2. Robustness to failures

Different classes of networks react in different ways when some of their nodes disappear. Here we will study the robustness to random failures **using only the largest connected component of the RegulonDB network**. To do so, copy the largest connected component of the network into a new graph. Next remove iteratively 20 arbitrary (i.e. randomly chosen) nodes from this graph and check if its connectivity has changed: does it remain as a single connected component or does the failure break it apart into multiple sub-graphs?

Note: to select a random node use the function `randint` from the module `numpy.random`.

```
Failure of node fdng breaks the largest connected component into 1 portion(s)
Failure of node hded breaks the largest connected component into 1 portion(s)
Failure of node phoe breaks the largest connected component into 1 portion(s)
Failure of node cysj breaks the largest connected component into 1 portion(s)
Failure of node fbab breaks the largest connected component into 1 portion(s)
```

```
Failure of node soxs breaks the largest connected component into 8 portion(s)
Failure of node aspc breaks the largest connected component into 8 portion(s)
Failure of node ilvn breaks the largest connected component into 8 portion(s)
Failure of node hofn breaks the largest connected component into 8 portion(s)
Failure of node hflc breaks the largest connected component into 8 portion(s)
Failure of node emry breaks the largest connected component into 8 portion(s)
Failure of node gspj breaks the largest connected component into 8 portion(s)
Failure of node sufd breaks the largest connected component into 8 portion(s)
Failure of node enth breaks the largest connected component into 8 portion(s)
Failure of node argu breaks the largest connected component into 8 portion(s)
Failure of node alka breaks the largest connected component into 8 portion(s)
Failure of node phob breaks the largest connected component into 36 portion(s)
Failure of node ykia breaks the largest connected component into 36 portion(s)
Failure of node rsta breaks the largest connected component into 36 portion(s)
Failure of node flge breaks the largest connected component into 36 portion(s)
```

Comment your result here

---

6 3. Sensitivity to attacks

We now study the effect of directed attacks, namely removal of the most important nodes of the network, and see how they affect its connectivity. Although there is a variety of ways to measure the importance of a node, for the sake of simplicity we will restrict ourselves to using the degree.

Using the `sorted` function in Python, list the 10 nodes with highest degree.

```
['crp', 'fnr', 'fis', 'ihf', 'h-ns', 'arca', 'fur', 'narl', 'lrp', 'nsrr']
```

Generate a new copy of the largest component of the network as you did above. Remove iteratively the 10 nodes with highest degree of the network and check how the connectivity changes.

```
Atacking node crp breaks the largest connected component into 77 portion(s)
Atacking node fnr breaks the largest connected component into 107 portion(s)
Atacking node fis breaks the largest connected component into 171 portion(s)
Atacking node ihf breaks the largest connected component into 197 portion(s)
Atacking node h-ns breaks the largest connected component into 217 portion(s)
Atacking node arca breaks the largest connected component into 242 portion(s)
Atacking node fur breaks the largest connected component into 290 portion(s)
Atacking node narl breaks the largest connected component into 332 portion(s)
Atacking node lrp breaks the largest connected component into 396 portion(s)
Atacking node nsrr breaks the largest connected component into 432 portion(s)
```

Comment your result here
