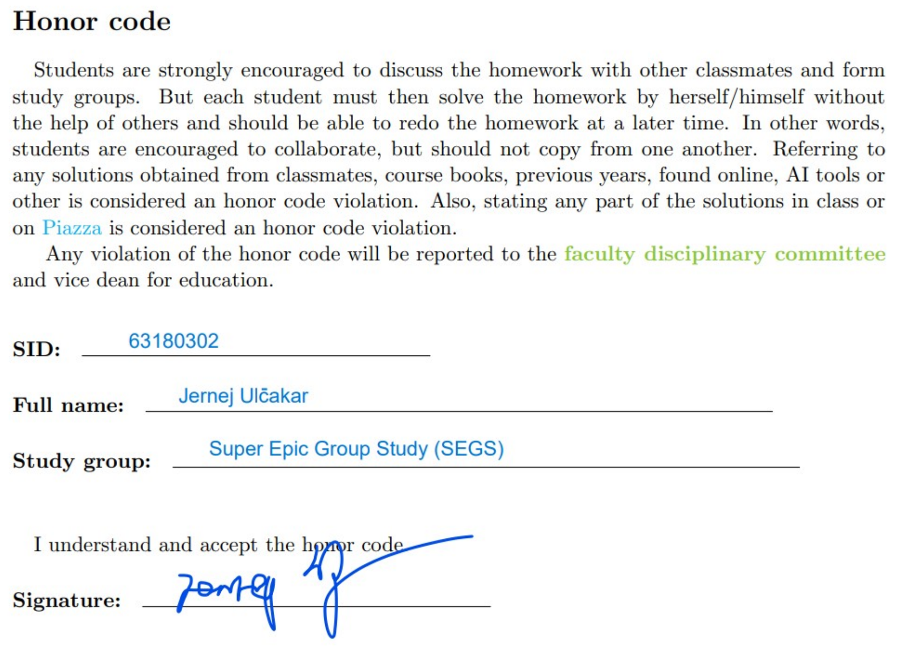
Introduction to Network Analysis (INA)

Homework 1

Avtor
Jernej Ulčakar
ju7857@student.uni-lj.si
Ljubljana, FRI
2023/24

Contents



Honor code

Students are strongly encouraged to discuss the homework with other classmates and form study groups. But each student must then solve the homework by herself/himself without the help of others and should be able to redo the homework at a later time. In other words, students are encouraged to collaborate, but should not copy from one another. Referring to any solutions obtained from classmates, course books, previous years, found online, AI tools or other is considered an honor code violation. Also, stating any part of the solutions in class or on Piazza is considered an honor code violation.

Any violation of the honor code will be reported to the **faculty disciplinary committee** and vice dean for education.

SID: 63180302

Full name: Jernej Ulčakar

Study group: Super Epic Group Study (SEGS)

I understand and accept the honor code.

Signature: [illegible]

1 Connected components

Assume a simple undirected graph with n nodes, m edges, and c connected components. Show that the following two inequalities hold. *(Use mathematical induction for the first inequality and simple reasoning for the second.)*

$$n - c \leq m \leq \binom{n-c+1}{2}$$

1. Proving the first inequality $n - c \leq m$
   Consider the basis: $n = 1$ which makes the connected components $c = 1$. Now the amount of edges can be minimally 0, which means that the node doesn't have a connection with itself. The inequality holds in this case.
   We use the induction step where we assume that the inequality holds for $n = k$ nodes and $n = k+1$ nodes. When we add 1 more node, we can either connect it with an edge to already existing component, which would only increase m by 1 (inequality holds), or we just create a new component (we don't connect it anywhere). This means that c and n increases by 1.
   This means that $n - c$ increases at most by 1 (if an edge is added) while m increases by 1 or if no new edges are added, $n - c$ stays the same as well as m.
   $(n+1) - c \leq (m+1)$ if adding a connection to an existing component
   $(n+1) - (c+1) \leq m$ if not connecting the new node to anything

2. Explaining the second inequality $m \leq \binom{n-c+1}{2}$
   The maximum amount of edges in a graph with n nodes and c connected components is when the graph is complete (each node connects to all other nodes). In this case each node has $n - 1$ connections, however we have to account that edges are counted twice. Therefore the maximum number of edges is $\binom{n}{2} = \frac{n(n-1)}{2}$, which means $c = 1$. In case of no edges ($m = 0$), the right part is also 0 (due to binomial being $\binom{1}{2} = 0$).

The criterion is useful for real networks because ensures that a graph is connected if the number of edges is $m \geq n - 1$. The minimal amount of edges to connect the whole graph is $n - 1$ edges (this would be a tree-like graph).

2 Node linking probability

Consider a graph model where the edges are placed independently between each pair of nodes i and j with the probability p_{ij} proportional to

$$p_{ij} \sim v_i v_j$$

where $v_i \geq 0$ is some number associated with node i. First show that the expected node degree $\langle k_i \rangle$ is proportional to v_i. (Show $\langle k_i \rangle = Cv_i$ for some constant C). Next, derive an exact expression of p_{ij} in terms of only the degree sequence $k = k_1, k_2, \ldots, k_n$ and recognize the result.

- In a graph with such probability for an edge, the expected node degree for a node i is the sum of all probabilities p_{ij}

$$\langle k_i \rangle = \sum_{\substack{j=1 \\ i \neq j}}^{n} v_i v_j = v_i \cdot \sum_{\substack{j=1 \\ i \neq j}}^{n} v_j = Cv_i$$

  In the equation we have shown that the constant C is equal to the sum of all other node values.

- We want to derive the expression p_{ij} in terms of only degree sequences, we can achieve this like so:

$$\langle k_i \rangle = Cv_i, \qquad v_i = \frac{\langle k_i \rangle}{C}$$
$$p_{ij} \sim v_i v_j = \frac{\langle k_i \rangle}{C} \frac{\langle k_j \rangle}{C} = \frac{\langle k_i \rangle \langle k_j \rangle}{C^2}$$

  we have to note that the constant C is different for v_i and v_j due to the sum of two different values; however we can ignore this since in large graphs this almost makes no difference.

$$k = \sum_{i=1}^{n} \langle k_i \rangle = \sum_{i=1}^{n} Cv_i = C \sum_{i=1}^{n} v_i = C^2$$

  Thus forming the previous equation into this form:

$$p_{ij} \sim v_i v_j = \frac{\langle k_i \rangle \langle k_j \rangle}{C^2} = \frac{\langle k_i \rangle \langle k_j \rangle}{k}$$

3 Random node selection

Erdős-Rényi graph model requires efficient implementation of a random selection of nodes, which can be easily achieved with most network representations. More realistic models require more sophisticated random selection procedures and associated network representations.

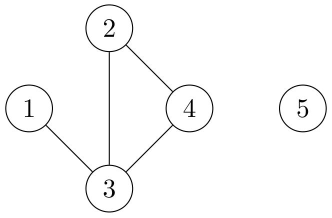
Design an algorithm that does not select the nodes uniformly at random, but proportional to their degree. More precisely, the node i should be selected with the probability $\frac{k_i}{2m}$, where k_i is its degree and m is the number of edges. The algorithm must run in constant time O(1), while you can assume any standard network representation. (Think more about the network representation than the algorithm.)

- We create an edge connectivity list (no duplicates) and then we can select an edge. This will result in selecting two nodes, so we can flip a coin which one should be selected. This is also an explanation to the formula of probability $\frac{k_i}{2m}$, where the node i appears k_i times in the edge list, the amount of edges is m and then flipping the coin for selection $\frac{1}{2}$.

- The probability of selecting a node is higher for the ones with higher degree (amount of edges on a node); therefore the graph would be mostly connected with the nodes that have a lot of connectivity.

- Special case if the graph is fully connected (the degree of all nodes is $n-1$), then the random node selection probability is $\frac{2(n-1)}{2n(n-1)} = \frac{1}{n}$ for each node (it is uniformly selected). Another special property of such formula is that we ignore all nodes that have no connectivity (their degree is 0).

- An example of using the proposed algorithm can be seen in the graph below:



We create an edge list: (1,3), (2,3), (2,4), (3,4). From this list we can now choose an edge (so out of 4 we choose 1 at uniformly random). At random we chose the edge (2,3) and now we flip a coin for the node selection. The random selected node in the end is 3.

The algorithm works in O(1) time. Each time we select a node we roll random numbers twice, which speeds up the selection.

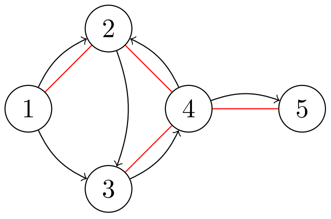
4 Weak & strong connectivity

Depth-first search is the most efficient algorithm for finding connected components of undirected networks. What would the same algorithm find in a directed network if one would follow the links in any direction? What would the algorithm find if one would follow the links only in the right direction? What would the algorithm find if one would follow the links only in the opposite direction?

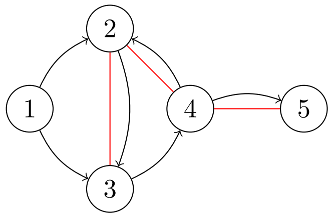
Based on your answers design an algorithm for finding strongly connected components in directed networks. Implement the algorithm and find strongly connected components of Enron e-mail communication network (You should not use the Kosaraju algorithm.) Compute the number of strongly connected components and the size of the largest one. Are the results expected?

- Using DFS on a directed graph by following the links in **any** direction would find all nodes that have a path from or to starting position (it would find the largest weakly component that includes the start node); however it wouldn't tell us anything about strong connectivity.
- Using DFS on a directed graph by following the links only in the **right** direction it would find us all nodes that the starting node has access to but not necessarily the nodes that lead to starting node.
- Using DFS on a directed graph by following the links only in the **opposite** direction is similar to previous point; however in this case it would find all nodes that eventually lead to the starting node.

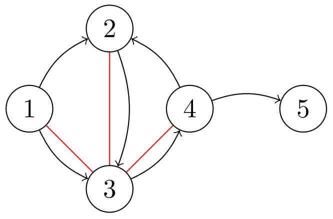
An example for each of the occasion where the starting node is 4. Red color represents the found nodes by following the rules per point (using DFS).



(a) Ignoring direction



(b) Following the direction



(c) Following opposite direction

By analyzing these examples we can implement a simple algorithm that goes over every node that hasn't been yet visited and then running DFS in the direction of the links from that node and saving it into a set (or list). Then we run the same DFS but this time in the opposite direction of the links and save into into a new set. We then find the intersection of nodes found in the set and this represents our strongly connected components.
In the upper example this would mean to find an intersection between nodes (2,3,4,5) and (1,2,3,4), which is (2,3,4) which makes a strongly connected component.

Here's the code to the proposed algorithm with answers provided for questions on Enron e-mail communications network. The whole algorithm took around 1.5s to execute on the whole network (most of the time is consumed on reversing the graph). The code can be found in my GitHub repository.

```python
import networkx as nx

network_name = "enron.net"
G = nx.convert_node_labels_to_integers(nx.DiGraph(nx.read_pajek(network_name)),first_label=1)
G.name = network_name

def DFS(G: nx.Graph, node,v):
    visited = set()
    stack = []
    stack.append(node)
    while stack:
        u = stack.pop()
        if u not in visited and u not in v:
            visited.add(u)
            for neighbor in G.neighbors(u):
                stack.append(neighbor)
    return visited

#Finding the largest connectivity
def connectivity_function(G):
    visited = set()
    connectivity = []
    G_rev = G.reverse()
    for i in G:
        if i not in visited:
            crawl = DFS(G,i,visited)
            crawl_reverse = DFS(G_rev,i,visited)

            #now we check the intersection of the node i
            intersection = crawl.intersection(crawl_reverse)
            visited.update(intersection) #we add the nodes that are intersected, so we don't look at them again
            connectivity.append(intersection)
    return connectivity

connectivity = connectivity_function(G)
largest_connectivity = max(connectivity,key=len)
```

The graph properties are as follows: 87.273 vertices, 1.148.072 edges; The results of our algorithm is:

- number of strongly connected components: 78058
- largest strongly connectivity length: 9164

The graph is an unusual one since it has a lot of strongly connected components and the largest one isn't even that long (10% of all nodes).

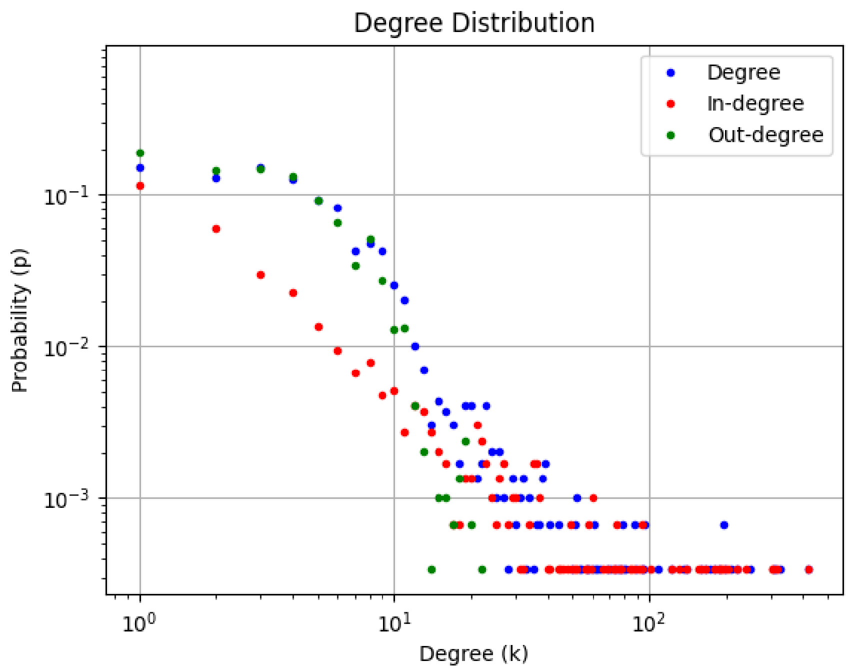
5 Is Java scale-free?

Consider a software class dependency network representing Lucene search engine. This is a directed network where node i links to node j if the software class represented by i depends on or uses the class represented by j. Compute the degree distribution pk and plot it on a log-log plot by representing each distinct (k, p_k) with a single dot, $k > 0$. Next, compute also the in-degree distribution $p_{k^{in}}$ and the out-degree distribution $p_{k^{out}}$, and superimpose them on the same plot using dots of different colors.

Compare all three degree distributions p_k and highlight the differences between them. Do the distributions appear to be scale-free following a power-law $k^{-\gamma_\cdot}$? For the distributions not to appear like a power-law, reason why. For the distributions that do seem to follow a power-law, reason why and compute their power-law exponent $\gamma_\cdot$ using the maximum likelihood formula

$$\gamma_\cdot = 1 + n_\cdot \left[\sum_{i=1}^{n} ln \frac{k_i^\cdot}{k_{min}^\cdot - \frac{1}{2}} \delta(k_i^\cdot \geq k_{min}) \right],$$

where k_i is the $\cdot$-degree of node i, $k_{min}^\cdot \geq 1$ is some reasonable choice for the minimum degree cutoff and $n_\cdot \geq n$ is the number of nodes thus considered.



The power-law exponent γ is in our case 1.25716. This is close to 1, meaning the graph behaves scale-free. Even from visual perspective (looking at the plot), the degree distribution exhibits a straight line (especially the in-degree), making it scale-free.

The code can be found on my GitHub repository.

```python
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
G = nx.convert_node_labels_to_integers(nx.DiGraph(nx.read_pajek("lucene.net")),
     label_attribute = 'label')
G.name = "lucene.net"

degree_sequence = [x[1] for x in G.degree()]
in_degree_sequence = [x[1] for x in G.in_degree()]
out_degree_sequence = [x[1] for x in G.out_degree()]

k_count = np.bincount(k)
in_degree_counts = np.bincount(in_degree_sequence)
out_degree_counts = np.bincount(out_degree_sequence)

num_nodes = len(in_degree_sequence)
pk = k_count/len(k)
pk_in = in_degree_counts/num_nodes
pk_out = out_degree_counts/num_nodes

nonzero_indices = np.nonzero(pk)
k_values = np.arange(len(pk))[nonzero_indices]
pk_nonzero = pk[nonzero_indices]

nonzero_indices_in = np.nonzero(pk_in)
k_values_in = np.arange(len(pk_in))[nonzero_indices_in]
pk_in_nonzero = pk_in[nonzero_indices_in]

nonzero_indices_out = np.nonzero(pk_out)
k_values_out = np.arange(len(pk_out))[nonzero_indices_out]
pk_out_nonzero = pk_out[nonzero_indices_out]

#here's code for plot... saving space for report
#...

def power_law(degree_sequence, k_min):
    n = len(degree_sequence)
    degree_sequence = np.array(degree_sequence)
    degree_sequence = degree_sequence[degree_sequence >= k_min]
    n_eff = len(degree_sequence)
    sum_ln_k = np.sum(np.log(degree_sequence / (k_min-0.5)))
    gamma = 1 + n_eff * (1 / sum_ln_k)
    return gamma

gamma = power_law(degree_sequence, 5)
print("Gamma:", gamma_degree)
```

6 Five networks problem

You are given five networks in edge list format.

- Flickr social affiliation network
- IMDb actors collaboration network
- Small part of a university Web graph
- Sample of computer science citation network
- Realization of the Erdös-Rényi random graph

All networks were reduced to their largest (weakly) connected component and represented with simple graphs. (Notice that two of the networks are directed.)

Your task is to figure out which network is which by examining its structure. Describe every step of your reasoning and/or provide necessary computations.

	num. nodes	num. Edges	Avg. Degree	Max. Degree	Avg Clustering
Network 1	291934	1037011	7.104	21442	0.253
Network 2	189207	703484	7.436	44	$3.437e^{-5}$
Network 3	94093	766258	16.287	10622	0.0
Network 4	202275	509097	5.0337	5606	0.079
Network 5	83159	5510251	132.523	6514	0.616

By analyzing the network statistics, we can assume that **network 3** is a Flickr social affiliation network due to average clustering coefficient being 0. This is because Flickr affiliation is a bipartite graph (can be colored with 2 colors; tree graph).

Network 2 has to be the realization of Erdös-Rényi graph due to maximum degree being 44, which is a property of such graphs that the maximum node degree is very low.

I would argue that **network 1** is a sample of computer science citation network since similar topics will cite same/similar papers making the max degree higher than the Web graph. This is because a very important computer science paper is usually cited many times as it is core to the topic, giving it a very large degree. This is also a directed graph.

Network 5 has to be IMDb actors collaboration network due to average clustering being high as well as average degree being huge. The explanation of this is that certain actors play in similar movies or their sequels but not necessarily all of the actors in the next movie of the same type. An example of such actors are Johnny Depp and Helena Bonham Carter. This is especially more noticeable for series (actors) or anime series (voice actors).

The Small part of a university Web graph should in the end be **network 4** due to web content being clustered in links such as headers and menus making it a directed graph; however the links to images and other content mostly aren't duplicated, making the average clustering lower.

7 Who to vaccinate?

The probability that an individual would spread some viral disease through her/his social network is proportional to k^2, where k is the degree of the corresponding node.

Consider two immunization schemes for preventing the spread of diseases. In the first scheme, you randomly select some number of individuals and vaccinate them. In the second scheme, you first select the same individuals, but then rather vaccinate a random acquaintance of theirs. Which of the two schemes will provide better immunization? Why?

- The better occasion is the second one, where we select the individuals acquaintances.
- This is due to Friendship-Paradox, which in this case states that the acquaintances of the selected individual have a higher degree (are more acquainted) than the selected individual.