Introduction to Network Analysis
Homework 2

Nace Gorenc

April 2021

1 Where is SN100?

The importance of the dolphin SN100 is in this case measured by betweenness centrality since it highlights nodes that are bridges between other nodes. Table 1 represents 5 most important nodes in this network. As it can be observed, there are 2 dolphins with the importance higher than 0.2. Note that the betweenness centrality measure is calculated by the algorithm in `networkx` library, which has slightly different normalization factor $(2/((n-1)(n-2)))$ compared to the normalization factor which we have seen in the lectures $(1/n^2)$.

The printout of the code used for this problem can be seen below.

```
def e1():
    dolphins = nx.read_adjlist('data/dolphins')
    bc = nx.betweenness_centrality(dolphins, normalized=True)
    print('Betweenness centrality:')
    print({key: value for key, value in sorted(bc.items(),
    ↪  key=lambda item: item[1], reverse=True)[:5]})
```

Dolphin's name	Betweenness centrality measure
SN100	0.248
Beescratch	0.213
SN9	0.143
SN4	0.139
DN63	0.118

Table 1: Most important dolphins according to betweenness centrality measure.

2 Is software scale-free?

Figure 1 and 2 represent degree distributions p_k vs. node degree k graphs. In both network the the degree distributions and out-degree distributions have

similar shape until $k < 20$ and after that we can see the similarity between degree and in-degree distributions. The major difference between this two network is that Java language has fewer root libraries that do not depend on any other libraries, however there are a lot of libraries that depends on a small number of other so called root libraries, and Lucene search engine has a lot of root libraries and libraries that depend on fewer than 10 other libraries. Additionally, both networks contain contain a great number of higher level libraries that no other library depend on.

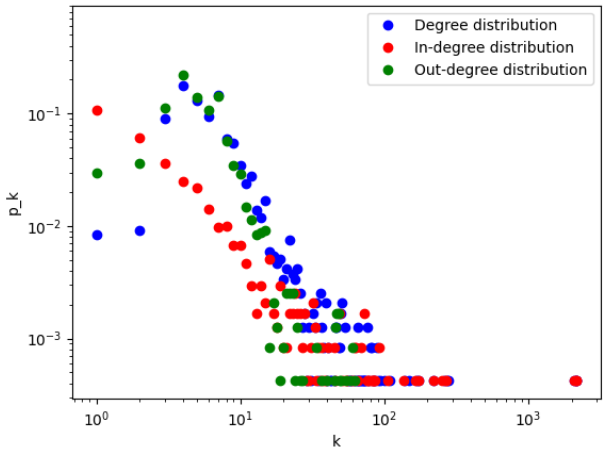

Figure 1: Degree distributions for Java language.

It appears that in-degree distributions of both networks follow a power-law due to linearity of the dots on the logarithmic graph. On the other hand, degree and out-degree distributions of both graph seem to follow Poisson distribution since the shape of the graphs resemble the Poisson distribution.

Using the likelihood formula, we can plot a graph γ vs. K_{min} of both in-degree distributions. As we can observe, both plots in figures 3 and 4 flatten when k_{min} is greater than 25 thus this is our k_{min}. Furthermore, the power-law exponents γ of Java and Lucene network for $k_{min} = 25$ are 2.14 and 2.12, respectively.

The printout of the code used for this problem can be seen below.

```
def generate_plot(s):
    g = read_undirected('data/' + s)
    tmp = [x for _, x in g.degree()]
    p_k = [tmp.count(i) / g.number_of_nodes() for i in
    ↪  range(max(tmp) + 1)]
    plt.plot(p_k, 'bo')
    tmp = [x for _, x in g.in_degree()]
```

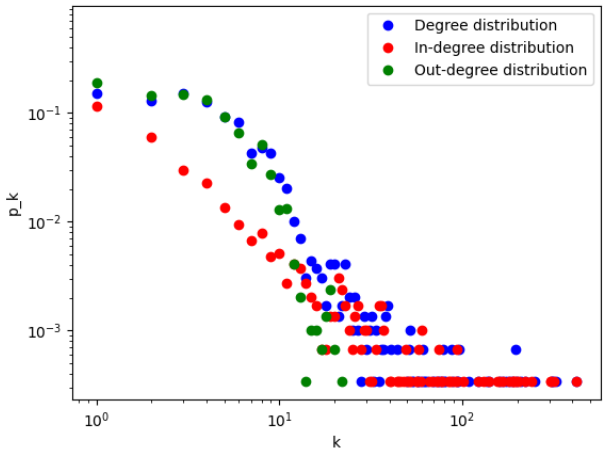

Figure 2: Degree distributions for Lucene search engine.

```
    p_k = [tmp.count(i) / g.number_of_nodes() for i in range(max(tmp) + 1)]
    plt.plot(p_k, 'ro')
    tmp = [x for _, x in g.out_degree()]
    p_k = [tmp.count(i) / g.number_of_nodes() for i in range(max(tmp) + 1)]
    plt.plot(p_k, 'go')
    plt.yscale('log')
    plt.xscale('log')
    plt.legend(['Degree distribution', 'In-degree distribution', 'Out-degree distribution'],
               loc='upper right')
    plt.xlabel('k')
    plt.ylabel('p_k')
    plt.savefig(s + '.png')
    plt.show()


def gamma(s):
    k = [i for i in range(1, 101)]
    gamma = []
    g = read_undirected('data/' + s)
    tmp = [x for _, x in g.in_degree()]
    for k_min in k:
        n = sum([i >= k_min for i in tmp])
        gamma.append(1 + n / (sum([np.log(i /(k_min - 0.5)) if i >= k_min else 0 for i in tmp])))
```

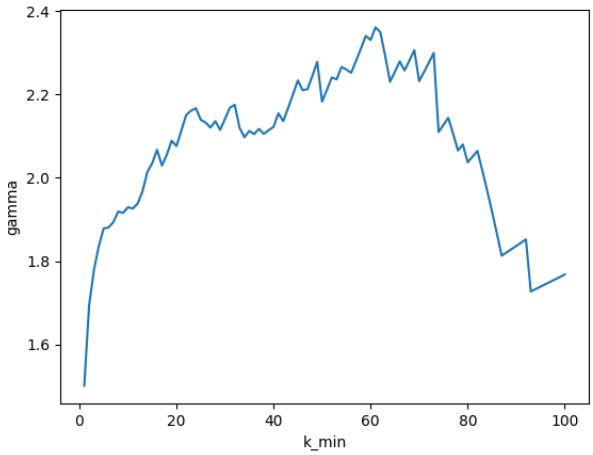

Figure 3: Java language.

```
    plt.plot(k, gamma)
    plt.xlabel('k_min')
    plt.ylabel('gamma')
    plt.savefig(s + '_gamma.png')
    plt.show()
    n = sum([i >= 25 for i in tmp])
    print(1 + n / (sum([np.log(i / (25 - 0.5)) if i >= 25 else 0
    ↪  for i in tmp])))


def e2():
    software = ['java', 'lucene']
    for s in software:
        generate_plot(s)
        gamma(s)
```

3 Errors and attacks on the Internet

Figures 5 and 6 represent plots of the errors and attacks on the Internet and random graph, respectively.

As we can observe, random failures of nodes in the internet does not effect the connectivity of the network. However, when we disconnect certain nodes with the highest node degrees the internet quickly fails to deliver information across the network since this nodes are likely routers that connect large parts of the internet. On the other hand, random graph is robust to random failures and malicious attacks due to the fact that each node has a node degree similar

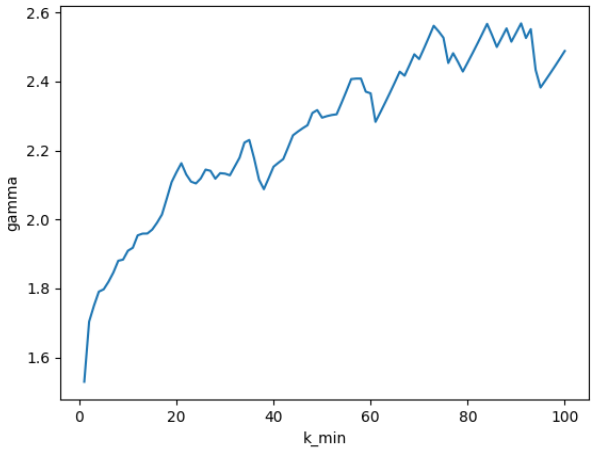

Figure 4: Lucene search engine.

to the average node degree.

The printout of the code used for this problem can be seen below.

```
def node_removal_list(n, p):
    l = []
    for i in n:
        if np.random.choice(np.arange(0,2), p=[1-p, p]):
            l.append(i)
    return l


def e3():
    fractions = [0, 0.1, 0.1, 0.1, 0.1, 0.1]
    f = [0, .1, .2, .3, .4, .5]
    pl = []
    g = nx.read_adjlist('data/nec')
    for p in fractions:
        rem = node_removal_list(list(g.nodes), p)
        g.remove_nodes_from(rem)
        largest_cc = max(nx.connected_components(g), key=len)
        pl.append(len(largest_cc) / g.number_of_nodes())
    plt.plot(f, pl)
    pl = []
    g = nx.read_adjlist('data/nec')
    for p in fractions:
        rem = int(p*g.number_of_nodes())
        l = list(g.degree())
        l.sort(key=lambda x: x[1], reverse=True)
```

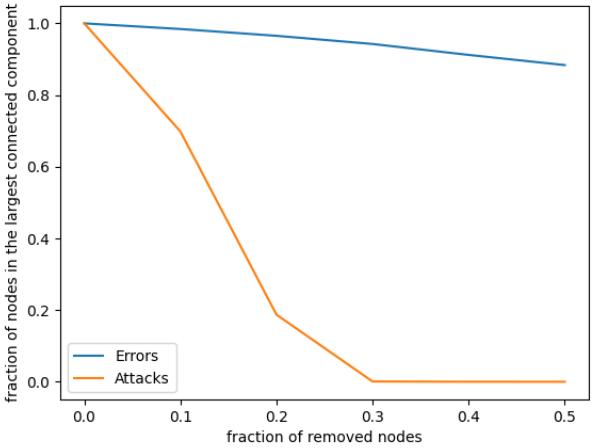

Figure 5: Internet.

```
        tmp = [x for x, _ in l[:rem]]
        g.remove_nodes_from(tmp)
        largest_cc = max(nx.connected_components(g), key=len)
        pl.append(len(largest_cc) / g.number_of_nodes())
    plt.plot(f, pl)
    plt.legend(['Errors', 'Attacks'], loc='lower left')
    plt.xlabel('fraction of removed nodes')
    plt.ylabel('fraction of nodes in the largest connected
    ↪  component')
    plt.savefig('internet.png')
    plt.show()

    pl = []
    g = nx.read_adjlist('data/nec')
    n = g.number_of_nodes()
    m = g.number_of_edges()
    g = nx.gnm_random_graph(n, m)
    for p in fractions:
        rem = node_removal_list(list(g.nodes), p)
        g.remove_nodes_from(rem)
        largest_cc = max(nx.connected_components(g), key=len)
        pl.append(len(largest_cc) / g.number_of_nodes())
    plt.plot(f, pl)
    pl = []
    g = nx.gnm_random_graph(n, m)
    for p in fractions:
        rem = int(p * g.number_of_nodes())
```

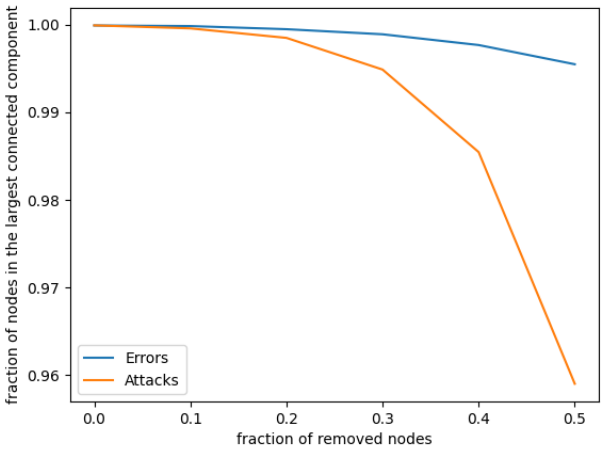

Figure 6: Random graph.

```
        l = list(g.degree())
        l.sort(key=lambda x: x[1], reverse=True)
        tmp = [x for x, _ in l[:rem]]
        if tmp:
            g.remove_nodes_from(tmp)
        largest_cc = max(nx.connected_components(g), key=len)
        pl.append(len(largest_cc) / g.number_of_nodes())
    plt.plot(f, pl)
    plt.legend(['Errors', 'Attacks'], loc='lower left')
    plt.xlabel('fraction of removed nodes')
    plt.ylabel('fraction of nodes in the largest connected
    ↪  component')
    plt.savefig('er.png')
    plt.show()
```

4 HIV and network sampling

Firstly, scale-free networks have a power-law degree distribution. We can observe from the shape of the plots in the figure 7 that the original social network has a straight line shape which typical for power-law distribution and the sampled network has a bit curvy line thus it is not scale-free.

Secondly, small-world networks have high average clustering coefficient $\langle C \rangle$ and short average distance $\langle d \rangle \simeq \frac{\log n}{\log \langle k \rangle}$. Let us observe the average clustering coefficients and average distances of both network in the table 2.

As we can see, both network have a relatively short average distances. How-

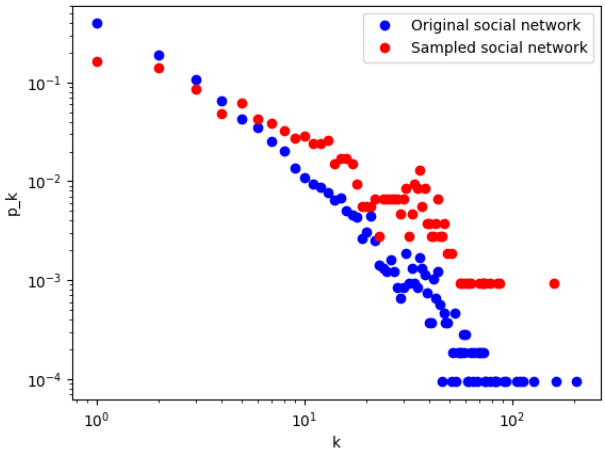

Figure 7: Node degree distributions of the original and sampled social network.

Network	$\langle C \rangle$	$\langle d \rangle$	$\frac{\log n}{\log \langle k \rangle}$
Original	0.27	7.49	6.12
Sampled	0.46	5.75	3.47

Table 2: Measures required for small-world network.

ever, the original network has a low clustering coefficient thus is not small-world network. On the other hand, the sampled network has high clustering therefore is a small-world network.

The printout of the code used for this problem can be seen below.

```
def distances(graph, i):
    d = {i: np.infty for i in list(graph.nodes)}
    q = deque()
    q.append(i)
    d[i] = 0
    while q:
        i = q.popleft()
        for j in graph[i]:
            if d[j] == np.infty:
                d[j] = d[i] + 1
                q.append(j)
    return d


def average_distance(g):
    d = list()
```

```
    for i in list(g.nodes):
        d.append(distances(g, i))
    n = g.number_of_nodes()
    return sum([v for tmp in d for v in tmp.values()]) / (n *
    ↪  (n-1))


def e4():
    g = nx.read_adjlist('data/social')
    rand_node =
    ↪  list(g.nodes)[np.random.randint(g.number_of_nodes())]

    walk = {rand_node}
    while True:
        if len(walk) * 10 >= g.number_of_nodes():
            break
        adj = list(g.adj[rand_node])
        rand_node = adj[np.random.randint(len(adj))]
        walk.add(rand_node)
    g_ind = g.subgraph(walk)
    print('AIDS <C> = ' + str(nx.average_clustering(g_ind)))
    print('AIDS <d> = ' + str(average_distance(g_ind)))
    print('AIDS : ' + str(np.log(g_ind.number_of_nodes()) /
    ↪  np.log(sum([x for _, x in list(g_ind.degree())]) /
    ↪  g_ind.number_of_nodes())))
    print('Social <C> = ' + str(nx.average_clustering(g)))
    print('Social <d> = ' + str(average_distance(g)))
    print('S : ' + str(np.log(g.number_of_nodes()) / np.log(sum([x
    ↪  for _, x in list(g.degree())]) / g.number_of_nodes())))

    tmp = [x for _, x in g.degree()]
    p_k = [tmp.count(i) / g.number_of_nodes() for i in
    ↪  range(max(tmp) + 1)]
    plt.plot(p_k, 'bo')
    tmp = [x for _, x in g_ind.degree()]
    p_k = [tmp.count(i) / g_ind.number_of_nodes() for i in
    ↪  range(max(tmp) + 1)]
    plt.plot(p_k, 'ro')
    plt.yscale('log')
    plt.xscale('log')
    plt.legend(['Original social network', 'Sampled social
    ↪  network', 'Out-degree distribution'],
               loc='upper right')
    plt.xlabel('k')
    plt.ylabel('p_k')
    plt.savefig('aids.png')
    plt.show()
```

5 Who to vaccinate?

Considering the friendship paradox which state that on average most people have fewer friend than their friends, the scheme where we select a number of individuals and than vaccinate a random acquaintance of theirs provide a better immunization.