In [1]:
```
!pip install python-louvain
!pip install folium
```

```
Requirement already satisfied: python-louvain in c:\users\97254\anaconda3\lib\site
-packages (0.16)
Requirement already satisfied: numpy in c:\users\97254\anaconda3\lib\site-packages
(from python-louvain) (1.24.3)
Requirement already satisfied: networkx in c:\users\97254\anaconda3\lib\site-packa
ges (from python-louvain) (3.1)
Requirement already satisfied: folium in c:\users\97254\anaconda3\lib\site-package
s (0.14.0)
Requirement already satisfied: branca>=0.6.0 in c:\users\97254\anaconda3\lib\site-
packages (from folium) (0.6.0)
Requirement already satisfied: requests in c:\users\97254\anaconda3\lib\site-packa
ges (from folium) (2.27.1)
Requirement already satisfied: jinja2>=2.9 in c:\users\97254\anaconda3\lib\site-pa
ckages (from folium) (2.11.3)
Requirement already satisfied: numpy in c:\users\97254\anaconda3\lib\site-packages
(from folium) (1.24.3)
Requirement already satisfied: MarkupSafe>=0.23 in c:\users\97254\anaconda3\lib\si
te-packages (from jinja2>=2.9->folium) (2.0.1)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\97254\anaconda3\lib
\site-packages (from requests->folium) (2021.10.8)
Requirement already satisfied: charset-normalizer~=2.0.0 in c:\users\97254\anacond
a3\lib\site-packages (from requests->folium) (2.0.4)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\users\97254\anaconda3\l
ib\site-packages (from requests->folium) (1.26.9)
Requirement already satisfied: idna<4,>=2.5 in c:\users\97254\anaconda3\lib\site-p
ackages (from requests->folium) (3.3)
```

In [2]:
```
import community
import networkx as nx
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import random
import heapq
import os
import EoN
import pydot
import folium
```

Data preparation

In [3]:
```
# user_1, user_2
brightkite_edges_np =
np.loadtxt(r"D:\Desktop\cp_networks\project\data\Brightkite_edges.txt",
```

```
delimiter='\t', dtype=int)
# user 2, user 1
brightkite_edges_np_reverse = np.array([[r[1], r[0]] for r in
brightkite_edges_np])

# user, check-in time, latitude, longitude, location id
brightkite_totalCheckins_np =
np.loadtxt(r"D:\Desktop\cp_networks\project\data\Brightkite_totalCheckins
delimiter='\t', dtype=str)
```

In [4]:

```
# convert to pandas dataframe
brightkite_edges = pd.DataFrame(brightkite_edges_np, columns=['user_1',
'user_2'])
brightkite_totalCheckins = pd.DataFrame(brightkite_totalCheckins_np
                            , columns=['user', 'check-in time',
'latitude', 'longitude', 'location id'])

# fix data type
brightkite_totalCheckins = brightkite_totalCheckins.astype({'user':
'int'})

# remove rows with missing values
missing_time_filter = brightkite_totalCheckins['check-in time'] == ""
indexes_to_remove = [i for i in range(len(missing_time_filter)) if
missing_time_filter[i]]
print(f"{sum(missing_time_filter)} rows without check-in time")
brightkite_totalCheckins =
brightkite_totalCheckins.drop(indexes_to_remove)

# convert coordinates from str to float
brightkite_totalCheckins = brightkite_totalCheckins.astype({'latitude':
'float'})
brightkite_totalCheckins =
brightkite_totalCheckins.astype({'longitude': 'float'})
```

```
6 rows without check-in time
```

In [5]:

```
brightkite_edges
```

Out[5]:

	user_1	user_2
0	0	1
1	0	2
2	0	3
3	0	4
4	0	5
...	...	...
428151	58225	58226
428152	58225	58227
428153	58226	58220
428154	58226	58225
428155	58227	58225

428156 rows × 2 columns

In [6]:
```
brightkite_totalCheckins
```

Out[6]:

	user	check-in time	latitude	longitude	location id
0	0	2010-10-17T01:48:53Z	39.747652	-104.992510	88c46bf20db295831bd2d1718ad7e6f5
1	0	2010-10-16T06:02:04Z	39.891383	-105.070814	7a0f88982aa015062b95e3b4843f9ca2
2	0	2010-10-16T03:48:54Z	39.891077	-105.068532	dd7cd3d264c2d063832db506fba8bf79
3	0	2010-10-14T18:25:51Z	39.750469	-104.999073	9848afcc62e500a01cf6fbf24b797732f8963683
4	0	2010-10-14T00:21:47Z	39.752713	-104.996337	2ef143e12038c870038df53e0478cefc
...	...	...	...	...	...
4747282	58222	2009-01-23T02:30:34Z	33.833333	35.833333	9f6b83bca22411dd85460384f67fcdb0
4747283	58224	2009-01-03T15:06:54Z	33.833333	35.833333	9f6b83bca22411dd85460384f67fcdb0
4747284	58225	2009-01-20T13:58:14Z	33.833333	35.833333	9f6b83bca22411dd85460384f67fcdb0
4747285	58226	2009-01-20T13:30:09Z	33.833333	35.833333	9f6b83bca22411dd85460384f67fcdb0
4747286	58227	2009-01-21T00:24:35Z	33.833333	35.833333	9f6b83bca22411dd85460384f67fcdb0

4747281 rows × 5 columns

In [7]:
```
# find all unique users
users_from_edges =
set(brightkite_edges['user_1']).union(set(brightkite_edges['user_1']))
users_from_checkins = set(brightkite_totalCheckins['user'])
users = sorted(list(users_from_checkins.union(users_from_edges)))
number_of_users = len(users)

print(f"users_from_edges is contained in users_from_checkins:
{users_from_edges.issubset(users_from_checkins)}")
print(f"users_from_checkins is contained in users_from_edges:
{users_from_checkins.issubset(users_from_edges)}")
print("." * 50)
print(f"total number of users: {number_of_users:,}")
```

```
users_from_edges is contained in users_from_checkins: False
users_from_checkins is contained in users_from_edges: True
..................................................
total number of users: 58,228
```

In [8]:
```
# find data time range (in months)
recoreded_years = set([t[:4] for t in brightkite_totalCheckins['check-
in time']])
start_year, end_year = min(recoreded_years), max(recoreded_years)
start_month = min(set([t[5:7] for t in brightkite_totalCheckins['check-
in time'] if t[:4] == start_year]))
end_month = max(set([t[5:7] for t in brightkite_totalCheckins['check-in
time'] if t[:4] == end_year]))

print(f"Data is from {start_month}/{start_year}  until
{end_month}/{end_year}")
```

```
Data is from 03/2008  until  10/2010
```

In [9]:
```
# create graph time stemps (in months)
time_stemps = []

for year in ['2008', '2009', '2010']:
    for month in (['0' + str(i) for i in range(1,10)] + [str(i) for i
in range(10,13)]):
        time_stemps.append(year + "-" + month)

time_stemps.remove('2008-02')
time_stemps.remove('2008-01')
time_stemps.remove('2010-11')
```

```
time_stemps.remove('2010-12')

number_of_timestemps = len(time_stemps)
print(f"number of timestemps (month): {number_of_timestemps}\n")
print("timestemps:\n")
print(time_stemps)
```

```
number of timestemps (month): 32

timestemps:

['2008-03', '2008-04', '2008-05', '2008-06', '2008-07', '2008-08', '2008-09', '200
8-10', '2008-11', '2008-12', '2009-01', '2009-02', '2009-03', '2009-04', '2009-0
5', '2009-06', '2009-07', '2009-08', '2009-09', '2009-10', '2009-11', '2009-12',
'2010-01', '2010-02', '2010-03', '2010-04', '2010-05', '2010-06', '2010-07', '2010
-08', '2010-09', '2010-10']
```

Create Networks

In [10]:

```
# divide check-in data per month
check_ins_per_month = dict()
i =0
for ts in time_stemps:
    filt = brightkite_totalCheckins['check-in time'].str.startswith(ts)
    check_ins_per_month[ts] = brightkite_totalCheckins[filt]
```

In [11]:

```
# base graph (no weights, just nodes and edges)
G = nx.Graph()
G.add_nodes_from(users)
G.add_edges_from(brightkite_edges_np)
G.add_edges_from(brightkite_edges_np_reverse)

# graph for each month (with nodes weight)
Graphs_per_month = dict()
for l, ts in zip(check_ins_per_month.values(), time_stemps):
    cur_G = nx.Graph()
    cur_G.add_nodes_from(users)
    cur_G.add_edges_from(brightkite_edges_np)
    cur_G.add_edges_from(brightkite_edges_np_reverse)
    # add nodes weight = number of check-ins
    users_weights = [0 for _ in range(number_of_users)]
    for row in l.values:
        users_weights[row[0]] += 1
    for i, user in enumerate(users):
```

```
        cur_G.nodes[user]['weight'] = users_weights[i]
    Graphs_per_month[ts] = cur_G
```

__ Analysis __

In [12]:
```
# create subgraph of the 11 initials and their x neighberhood
def get_nodes_neighberhood(graph, nodes, depth):
    neighberhood = set()
    for node in nodes:
        new_neighbors = nx.single_source_shortest_path_length(graph,
node, cutoff=depth)
        for nn in new_neighbors:
            neighberhood.add(nn)
    return neighberhood
```

In [13]:
```
def get_random_strech_neighborhood(graph, nodes, depth, sample_rate):
    neighborhood = set(nodes)
    for node in nodes:
        new_neighbors = nx.single_source_shortest_path_length(graph,
node, cutoff=depth)
        # adding a sample from each distance
        for d in range(1, depth+1):
            dist_d = set([node for node, distance in
new_neighbors.items() if distance == d])
            dist_d = dist_d - neighborhood
            sr = min(sample_rate, len(dist_d))
            neighborhood =
neighborhood.union(random.sample(list(dist_d), sr))
    return neighborhood
```

General

In [14]:
```
# when users first start their activity
started_in = []
didnt_start = 100 # %
for i, year in enumerate((['2008', '2009', '2010'])):
    filt = brightkite_totalCheckins['check-in
time'].str.startswith(year)
    u = set(brightkite_totalCheckins['user'][filt])
```

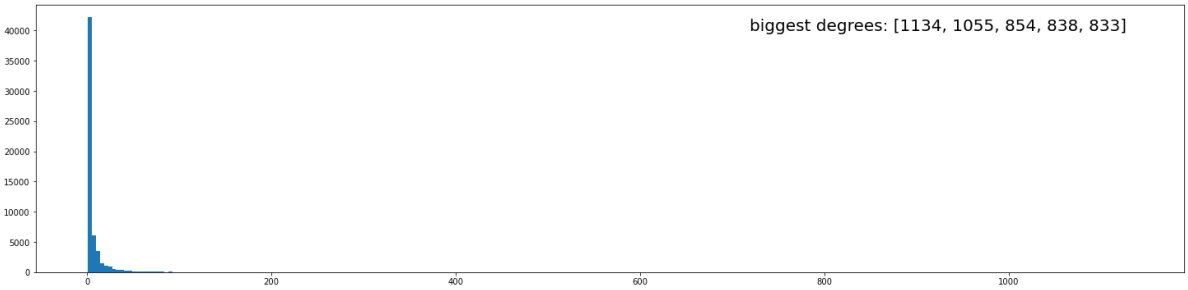
```
    for j in range(i):
        u -= started_in[j]
    started_in.append(u)
    p = (len(started_in[i])/number_of_users)*100
    didnt_start -= p
    print(f"{p:.1f}& of users did their first check-in in {year}")
print(f"\n{didnt_start:.1f}& of users didn't do any check-ins {year}")
```

```
47.9& of users did their first check-in in 2008
36.2& of users did their first check-in in 2009
4.2& of users did their first check-in in 2010

11.7& of users didn't do any check-ins 2010
```

Degree Distribution

In [15]:

```
degrees = [len(list(G.neighbors(user))) for user in users]
max_degrees = heapq.nlargest(5, degrees)
plt.figure(figsize=(25, 6))
plt.hist(degrees, bins=len(set(degrees)))
plt.text(0.95, 0.95, f'biggest degrees: {max_degrees}', fontsize=20,
horizontalalignment='right', verticalalignment='top',
transform=plt.gca().transAxes)
plt.show()
```

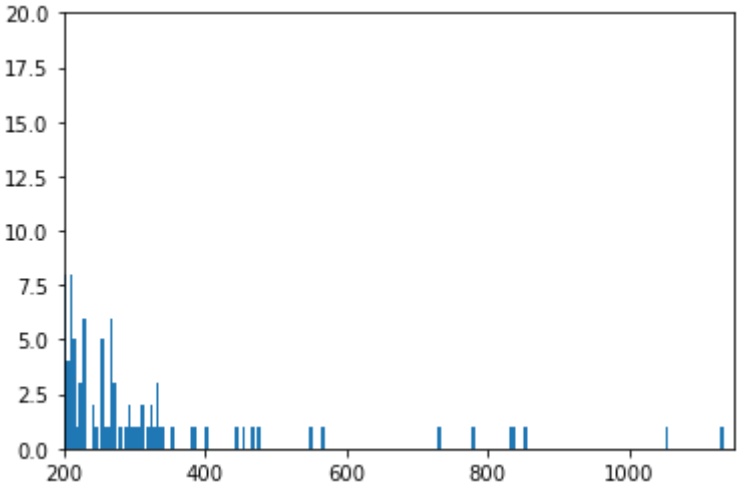


In [16]:

```
plt.hist(degrees, bins=len(set(degrees)))
plt.xlim(200, 1150)
plt.ylim(0, 20)
```

Out[16]:

```
(0.0, 20.0)
```



In [17]:
```
# degrees information
print(f"average degree in the network: {np.average(degrees):.1f}")
print(f"median degree in the network: {np.median(degrees)}")
ps = [np.percentile(degrees, 25), np.percentile(degrees, 50),
np.percentile(degrees, 75)]
print(f"percentiles of degrees (25, 50, 75): {ps[0], ps[1], ps[2]}")
print(f"top 10 higest degrees in the network: {heapq.nlargest(10,
degrees)}")
```

```
average degree in the network: 7.4
median degree in the network: 2.0
percentiles of degrees (25, 50, 75): (1.0, 2.0, 6.0)
top 10 higest degrees in the network: [1134, 1055, 854, 838, 833, 779, 732, 569, 5
50, 475]
```

Connected componets

In [18]:
```
# Find connected components
cc = nx.connected_components(G)
connected_components = []
for component in cc:
    connected_components.append(list(component))

print(f"number of connected componets: {len(connected_components)}")

print(f"number of connected componets with more then 10 nodes: "
                                        f"{len([c for c in
connected_components if len(c) > 10])}")

print(f"number of connected components with more than 20 nodes: "
                                       f"{len([c for c in
connected_components if len(c) > 50])}\n")
```

```
print(f"Giant connected component contains
{(len(connected_components[0])/number_of_users)*100:.1f}% "
                                            "of the total users in
the network")

# define subgraph induced by the GCC
gcc = connected_components[0]
gcc_graph = G.subgraph(gcc)
```

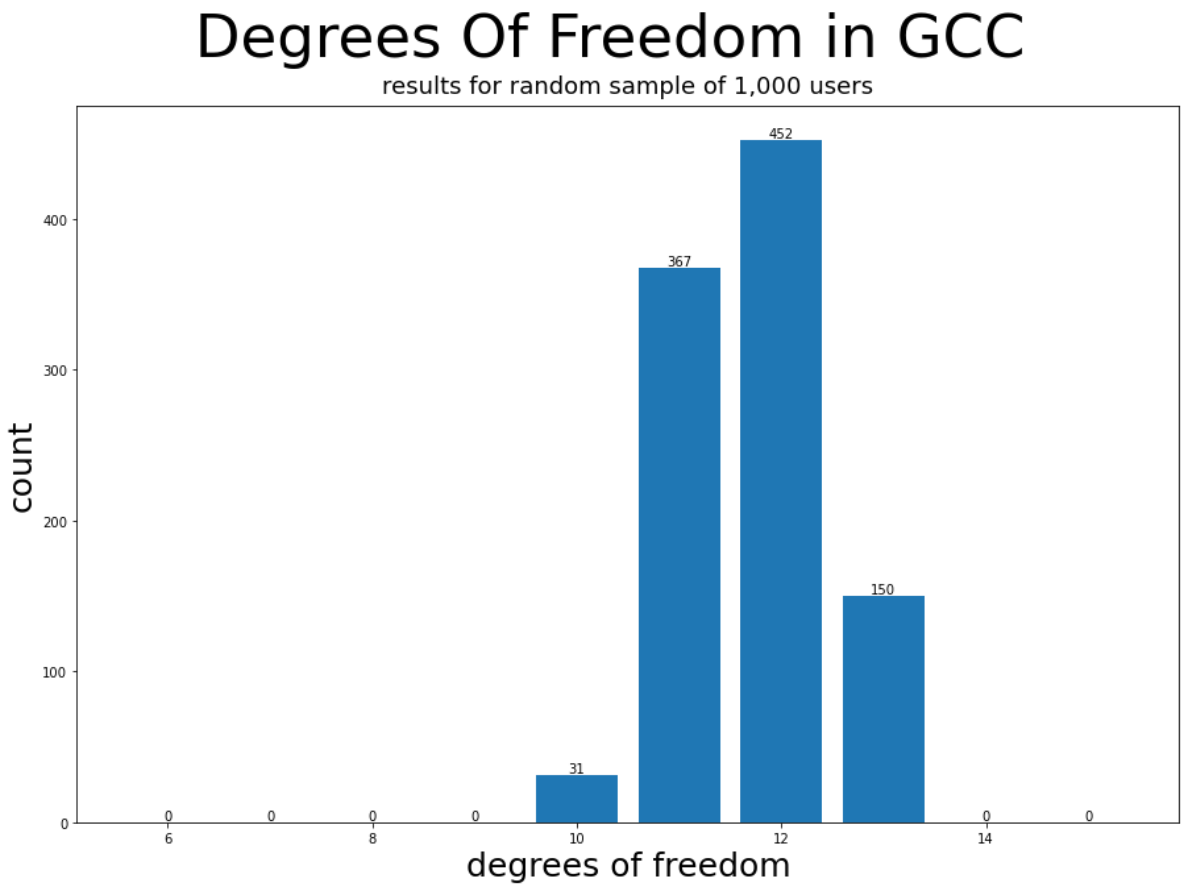
```
number of connected componets: 547
number of connected componets with more then 10 nodes: 4
number of connected components with more than 20 nodes: 1

Giant connected component contains 97.4% of the total users in the network
```

Small-World phenomena

In [19]:

```
# examine only gcc
SWP_G = G.subgraph(connected_components[0])

# sample 1000 random users and check how many users are in their x-
distanced neighberhood
sizes = {key: 0 for key in [6,7,8,9,10,11,12,13,14,15]}
used = []

for _ in range(1000):
    # sampling a user that didn't been sample before
    cur_user = None
    while cur_user is None:
        cur_user = random.sample(connected_components[0], 1)
        if cur_user in used:
            cur_user = None
        else:
            used.append(cur_user)

    # find distances
    dist = 5
    neighberhood = set()
    while(len(neighberhood) < len(connected_components[0]) and dist <
13):
        dist += 1
        neighberhood = get_nodes_neighberhood(G, cur_user, dist)
    sizes[dist] += 1
```

In [20]:
```
plt.figure(figsize=(15,10))
plt.bar(sizes.keys(), sizes.values())
for i, height in enumerate(list(sizes.values())):
    plt.text(i+6, height, str(height), ha='center', va='bottom')
plt.title("results for random sample of 1,000 users", fontsize=18, y=1,
va='bottom')
plt.suptitle("Degrees Of Freedom in GCC", fontsize=45)
plt.xlabel("degrees of freedom", fontsize=25)
plt.ylabel("count", fontsize=25)
plt.show()
```

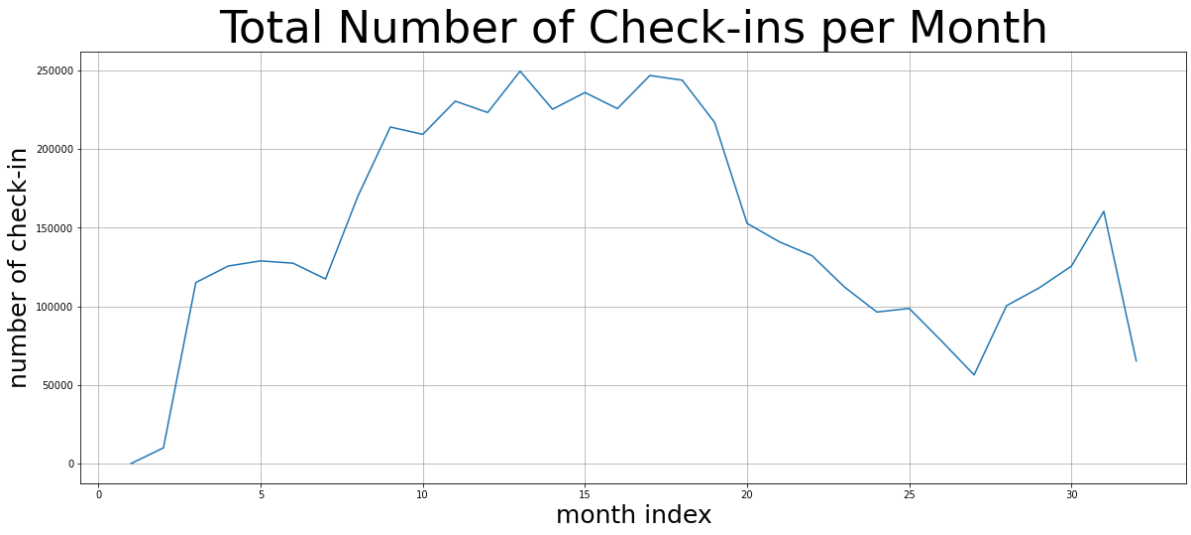


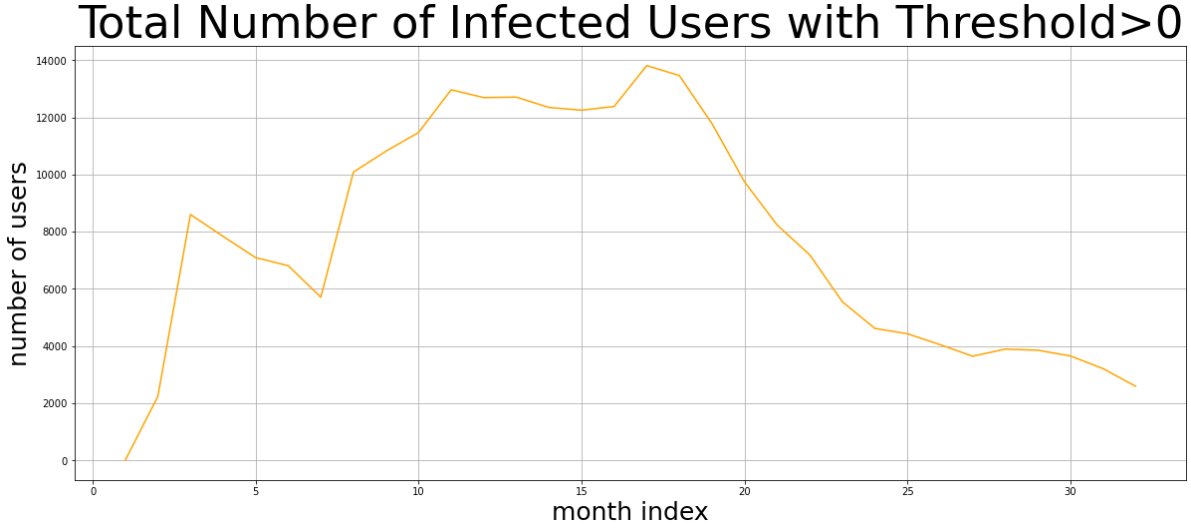
Brightkite Popularity

(users level of activity)

In [21]:
```
infected_users_per_month = []
th = 0
for t in time_stemps:
    cur_g = Graphs_per_month[t]
    tot = 0
    for user in cur_g:
        if(cur_g.nodes[user]['weight'] > th):
            tot += 1
    infected_users_per_month.append(tot)
```

In [22]:
```
plt.figure(figsize=(20,8))
plt.plot(np.arange(1,len(time_stemps)+1), [len(c) for c in
check_ins_per_month.values()])
plt.title("Total Number of Check-ins per Month", fontsize=45)
plt.xlabel("month index", fontsize=25)
plt.ylabel("number of check-in", fontsize=25)
plt.grid(True)
plt.show()

plt.figure(figsize=(20,8))
plt.plot(np.arange(1, len(time_stemps)+1), infected_users_per_month,
color='orange')
plt.title(f"Total Number of Infected Users with Threshold>{th}",
fontsize=45)
plt.xlabel("month index", fontsize=25)
plt.ylabel("number of users", fontsize=25)
plt.grid(True)
plt.show()
```





SIR Model

Year Based Communities

Comparison between communities based on year of first check-in Vs kernighan-lin bisection Algorithm

In [23]:
```
from networkx.algorithms.community import kernighan_lin_bisection

gcc_subgraph =
gcc_graph.subgraph(list(started_in[0].union(started_in[1])))

# kernighan_lin partitioning
partition = kernighan_lin_bisection(gcc_subgraph)
kernighan_lin_communities = {0: partition[0], 1: partition[1]}
kernighan_lin_score = nx.community.modularity(gcc_subgraph,
kernighan_lin_communities.values())
print(f'kernighan-lin algorithm community score:
{kernighan_lin_score}')



# started year based partitioning

    # remove nodes that aren't in GCC
nodes_to_remove = set()
started_in_gcc = started_in.copy()
for i in [0,1]:
    for node in started_in[i]:
        if node not in gcc:
            nodes_to_remove.add(node)
    started_in_gcc[i] -= nodes_to_remove

    # partition
year_communities = {'2008': started_in_gcc[0], '2009':
started_in_gcc[1]}
year_communities_score = nx.community.modularity(gcc_subgraph,
year_communities.values())
print(f'year based community score: {year_communities_score}')
```

```
kernighan-lin algorithm community score: 0.38338722358275756
year based community score: 0.1858984898709072
```

Initial Infected nodes Vs Random nodes

In [24]:
```
# Initial 'Infected' Users
cur_graph = Graphs_per_month['2008-03']

initial_infected_nodes = dict()
for node in cur_graph.nodes:
    if cur_graph.nodes[node]['weight'] > 0:
        w = cur_graph.nodes[node]['weight']
        print(f"user {node} with {w} check-ins and has
{len(list(cur_graph.neighbors(node)))} friends")
        initial_infected_nodes[node] = w
print(f"\ntotal of {len(initial_infected_nodes)} initial infected users
")
```

```
user 12 with 30 check-ins and has 113 friends
user 39 with 3 check-ins and has 4 friends
user 45 with 16 check-ins and has 159 friends
user 876 with 6 check-ins and has 39 friends
user 881 with 2 check-ins and has 24 friends
user 1108 with 2 check-ins and has 3 friends
user 1587 with 6 check-ins and has 33 friends
user 1777 with 3 check-ins and has 21 friends
user 4570 with 2 check-ins and has 17 friends
user 4867 with 16 check-ins and has 4 friends
user 5081 with 2 check-ins and has 1 friends

total of 11 initial infected users
```

Neighberhood Size - "initial_11" vs "random_11"

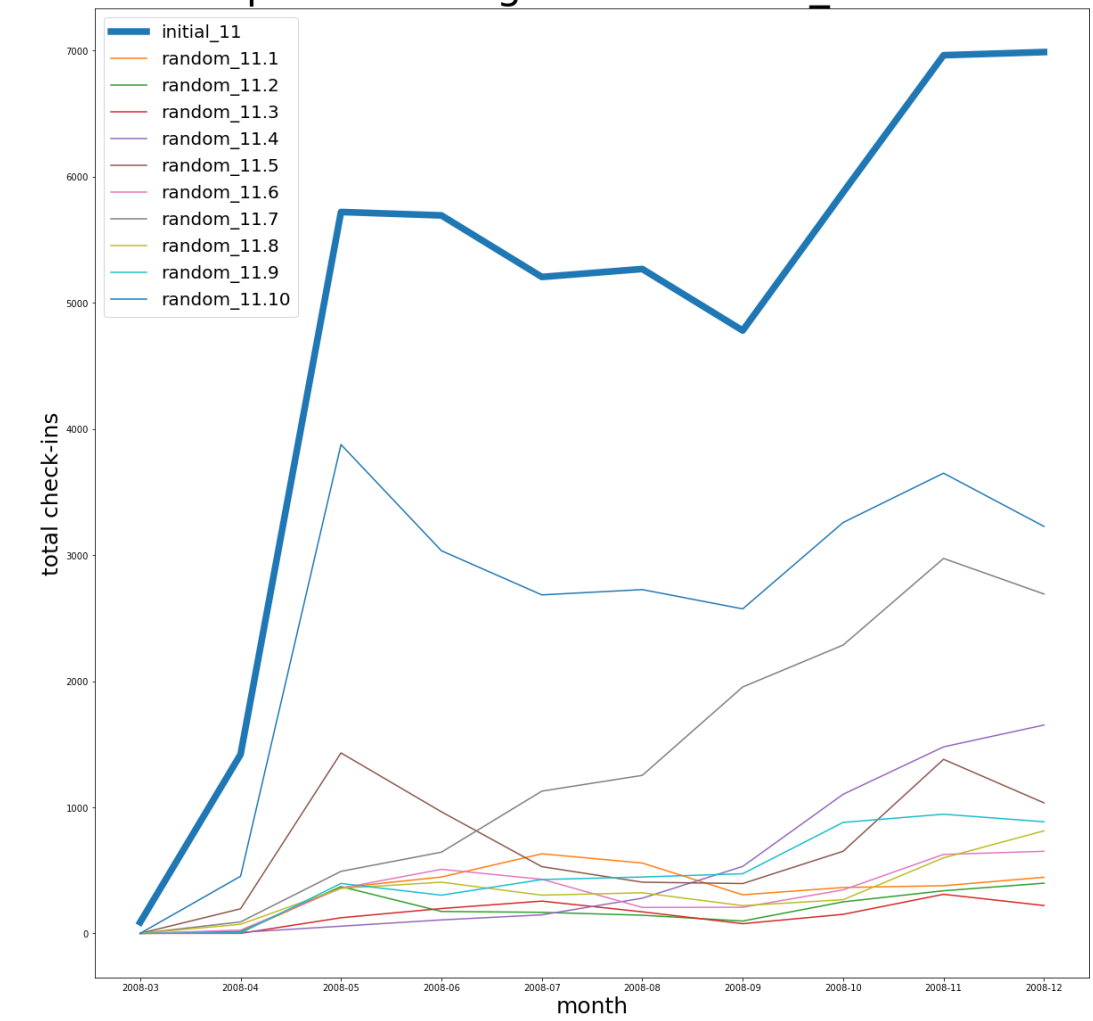
In [25]:
```
sizes_11 = []

number_of_samples = 3
sizes_random = [[] for _ in range(number_of_samples)]
depths = [i for i in range(1,11)]
for d in depths:
    sizes_11.append(len(get_nodes_neighberhood(cur_graph,
initial_infected_nodes, d)))

    for r in range(number_of_samples):
        sizes_random[r].append(len(get_nodes_neighberhood(cur_graph,
random.sample(users, 11), d)))

print(sizes_11)
```

```
[312, 7987, 36605, 52500, 55914, 56559, 56703, 56726, 56736, 56739]
```

growth of random isolated nodes Vs neighbors of infected_11

In [26]:
```
neighberhood_11 = get_nodes_neighberhood(cur_graph,
initial_infected_nodes, 1)
```

In [27]:
```
plt.figure(figsize=(20, 20))

vals = []
for t in time_stemps[0:10]:
    tot = 0
    for u in neighberhood_11:
        tot += Graphs_per_month[t].nodes[u]['weight']
    vals.append(tot)

plt.plot(time_stemps[0:10], vals, linewidth=7.5, label = "initial_11")
# plt.legend(, fontsize=20)

for i in range(10):
    random_11 = random.sample(list(set(users)-
set(initial_infected_nodes.keys())), 11)
    random11_neighberhood =
get_nodes_neighberhood(Graphs_per_month['2008-03'], random_11, 1)
    vals = []
    for t in time_stemps[0:10]:
        tot = 0
        for u in random11_neighberhood:
            tot += Graphs_per_month[t].nodes[u]['weight']
        vals.append(tot)
    plt.plot(time_stemps[0:10], vals, label = f"random_11.{str(i+1)}")
#     plt.legend(f"random_11.{i+1}",fontsize=20)

plt.title("Growth comparison - neighbors of initial_11 Vs random_11",
fontsize=45)
plt.ylabel("total check-ins", fontsize=25)
plt.xlabel("month", fontsize=25)
plt.legend(fontsize=20)
plt.show()
```



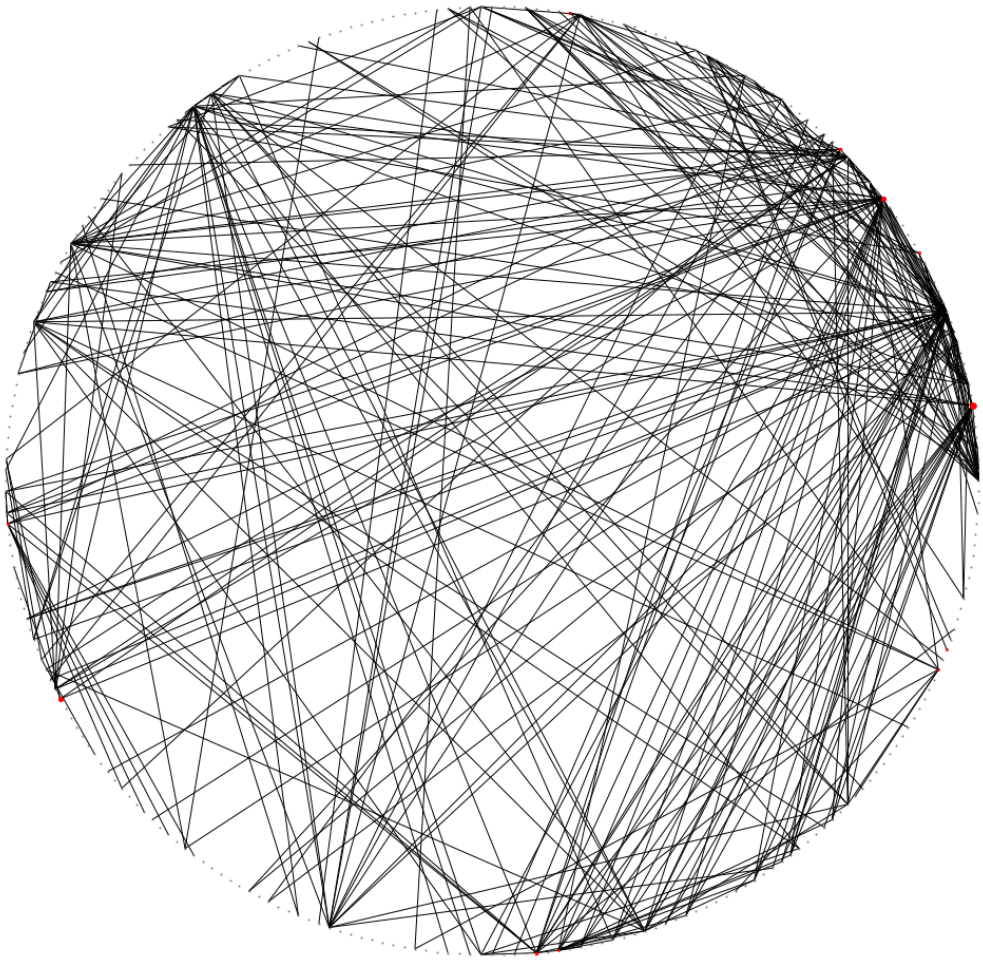
Disease Spreading

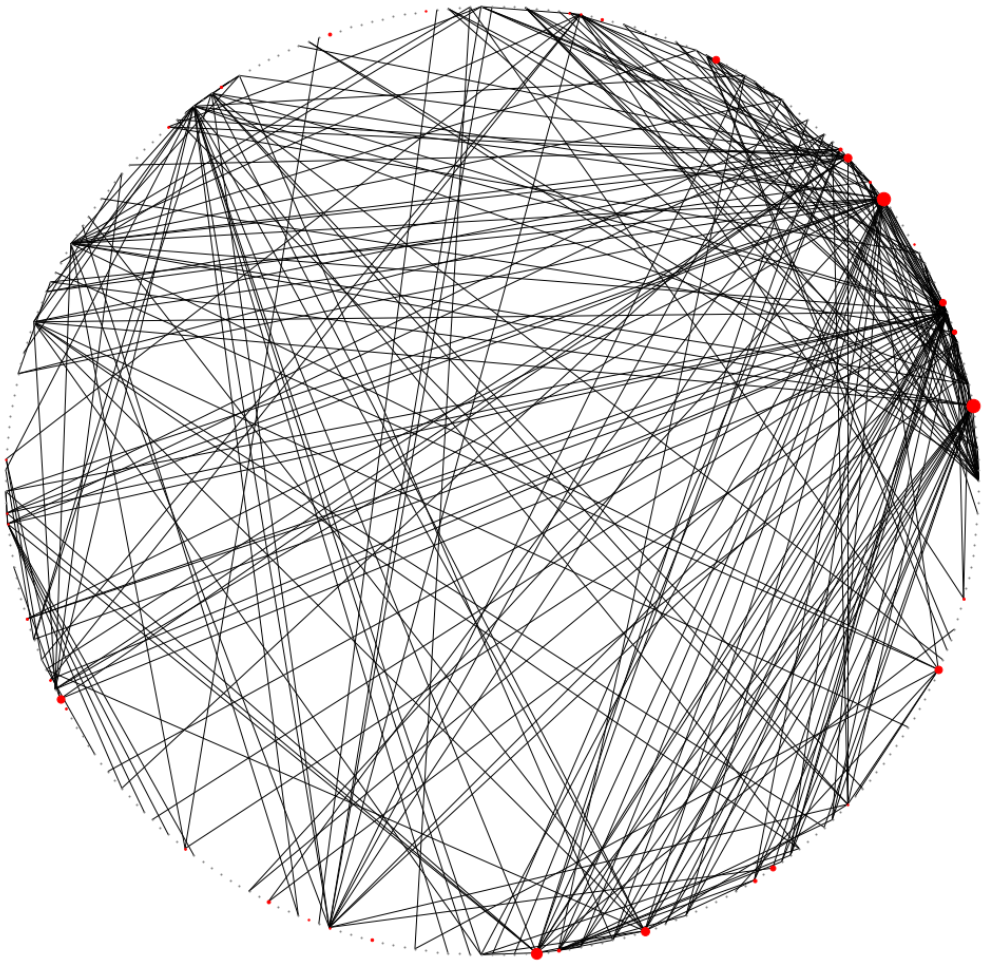
among random nodes at up to 5 hopes from initial_11

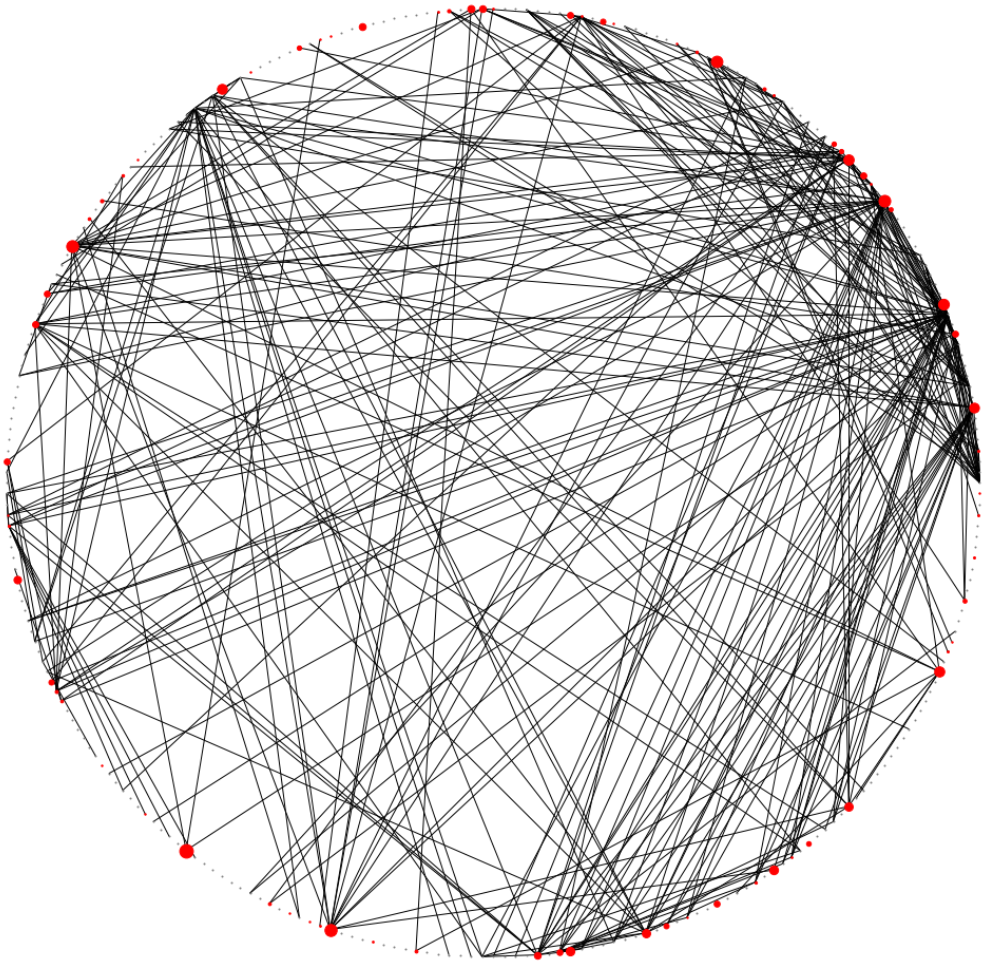
In [28]:

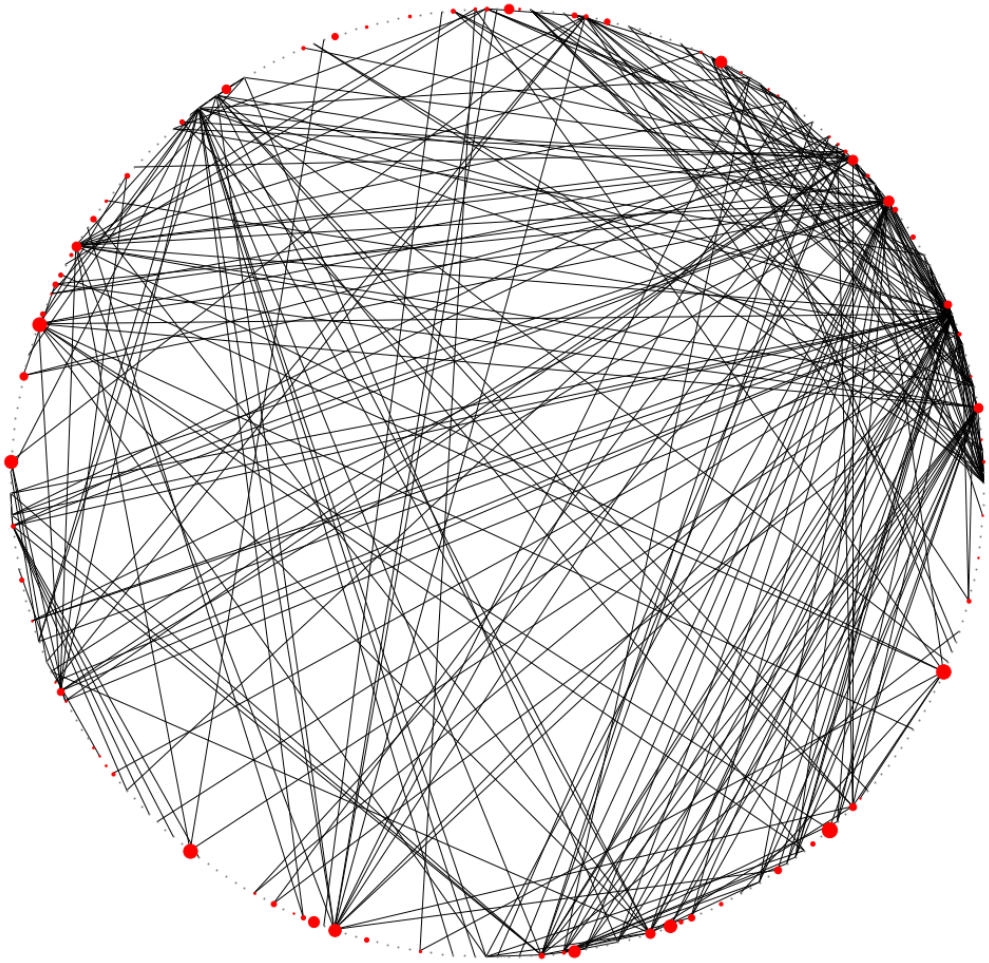
```
stretch_neighborhood = get_random_strech_neighborhood(G,
list(initial_infected_nodes.keys()), 5, 5)

for t in time_stemps[:5]:
    sg = Graphs_per_month[t].subgraph(stretch_neighborhood)

    plt.figure(figsize=(20, 20))
    pos = nx.circular_layout(sg, scale=2)

    for node in stretch_neighborhood:
        size = sg.nodes[node]['weight']
        if size>0:
            nc = 'red'
            ns = size + 1
        else:
            nc = 'grey'
```

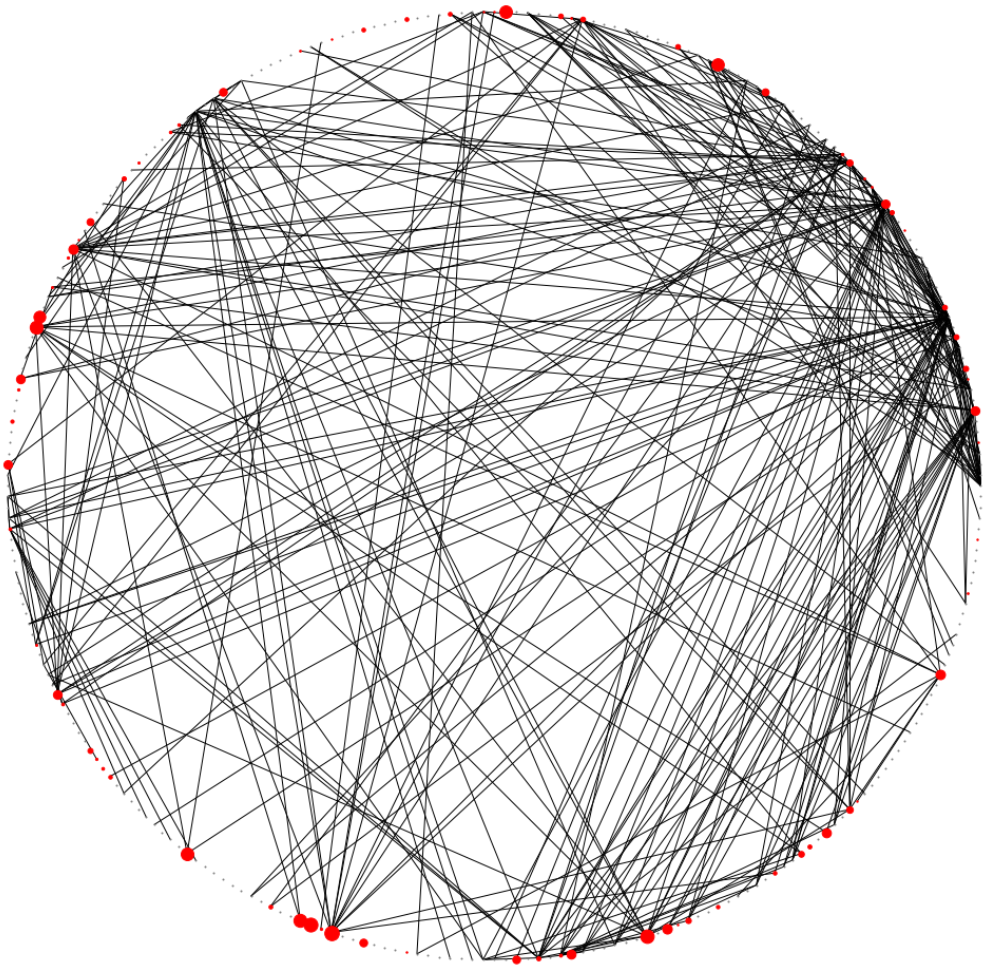
```
            ns = 1

        nx.draw_networkx_nodes(sg, pos, nodelist=[node], node_color=nc,
node_size=ns)

    nx.draw_networkx_edges(sg, pos)
    plt.axis('off')
    plt.show()
```











Disease Spreading

among random nodes on a shortest paths between initial_11 nodes to nodes at distance 5

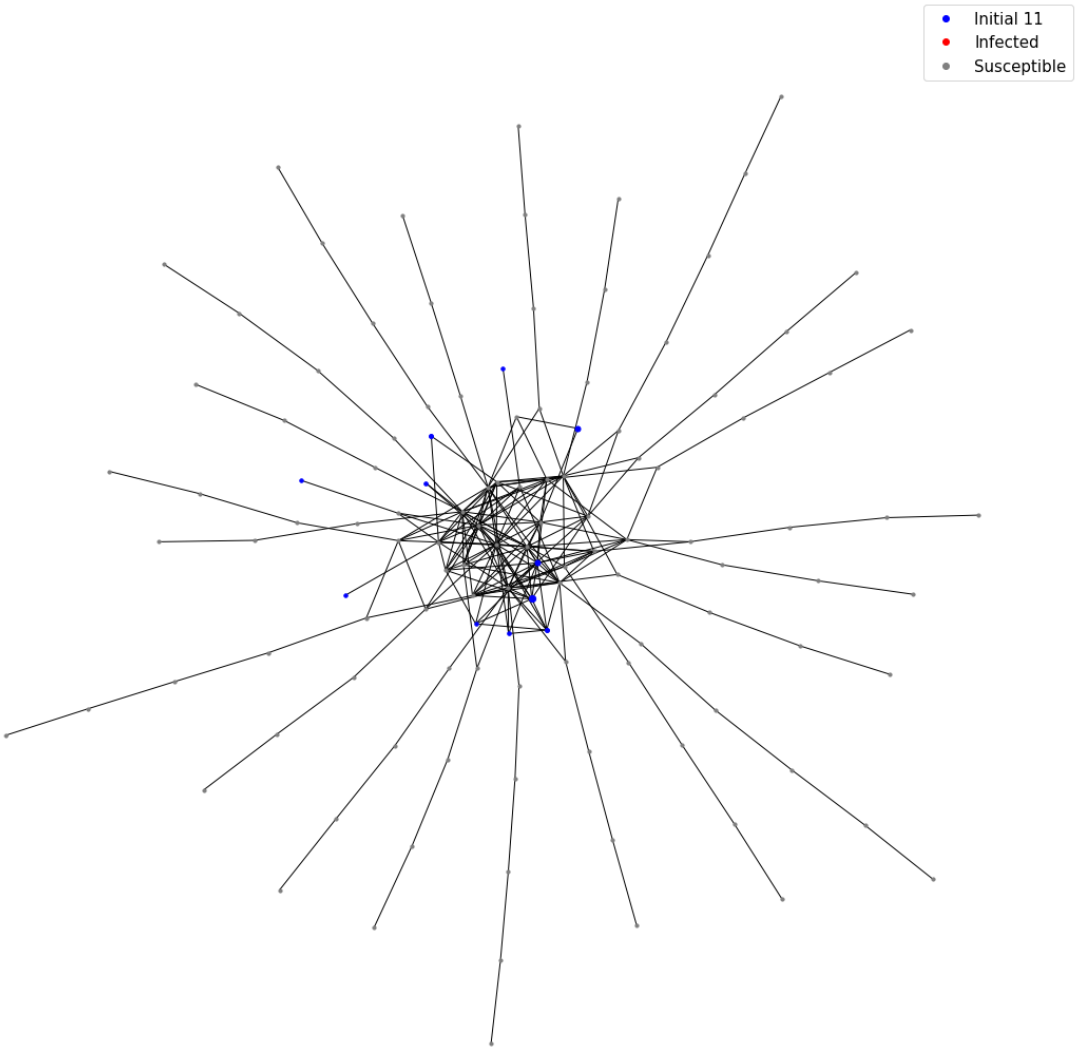
In [29]:
```
def get_paths(graph, nodes, depth):

    chosen_nodes = set(nodes)

    for node in initial_infected_nodes:
        neighborhood = nx.single_source_shortest_path_length(graph,
node, cutoff=depth)
        nodes_at_distance_depth = [n for n, distance in
neighborhood.items() if distance == depth]
        rn = None
        while (rn is None):
            rn = random.sample(nodes_at_distance_depth, 2)
            # verify nodes don't already used
```

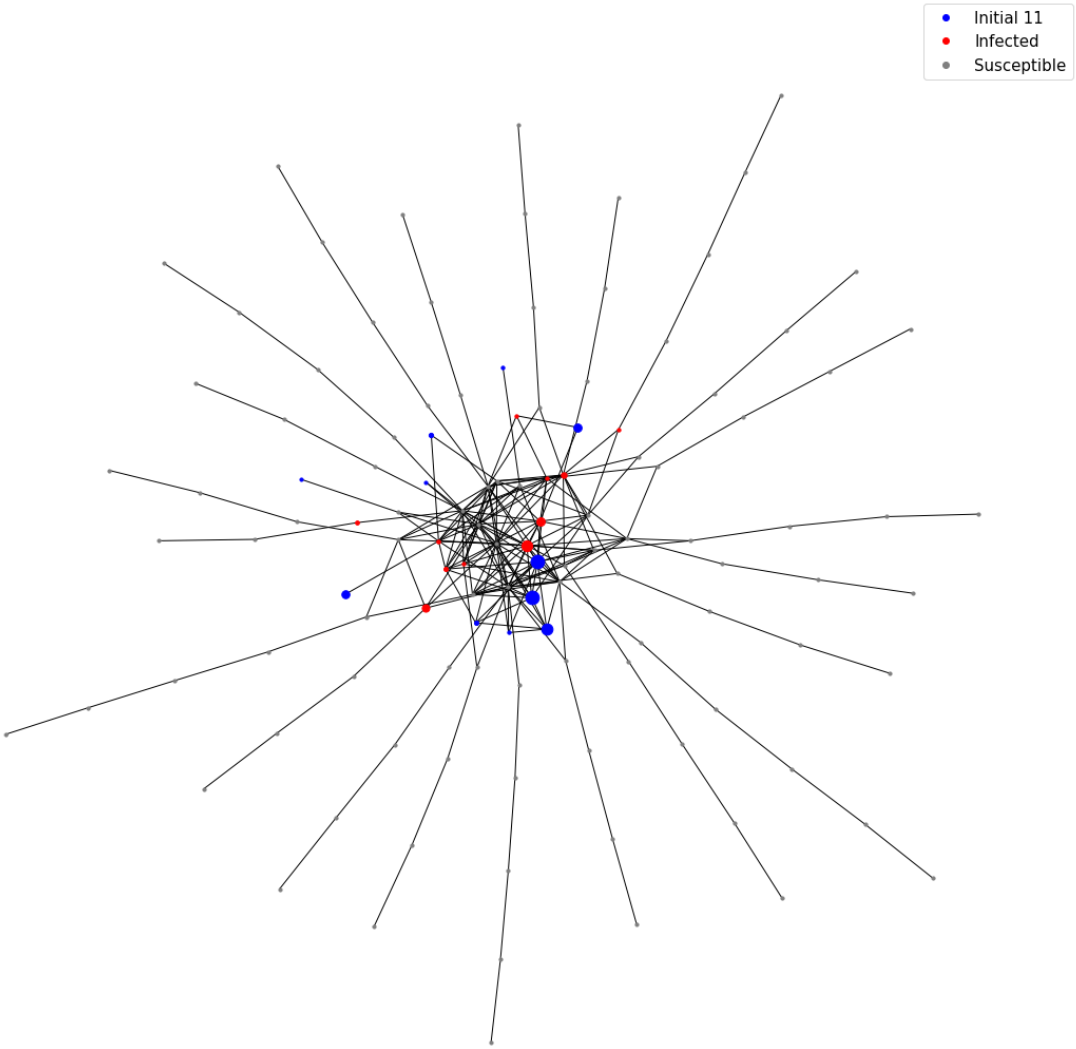
```
            if sum([1 if n in chosen_nodes else 0 for n in rn]) > 0:
                rn = None

        cur = [nx.shortest_path(graph, node, n) for n in rn]
        fin = []
        for sa in cur:
            fin += sa
        chosen_nodes = chosen_nodes.union(chosen_nodes, set(fin))
    return chosen_nodes
```

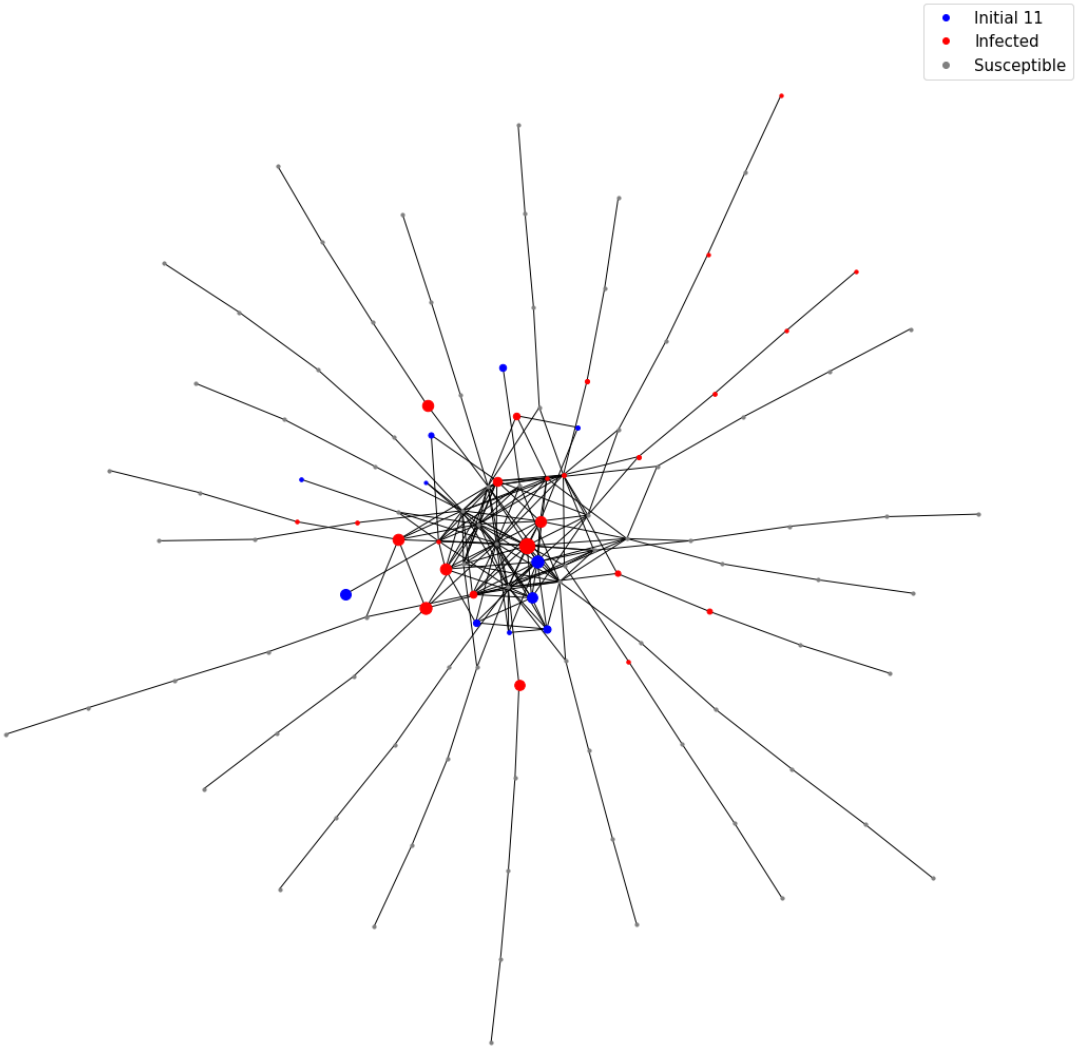
In [30]:
```
sample_near_11_nodes = get_paths(G,
list(initial_infected_nodes.keys()), 6)
```

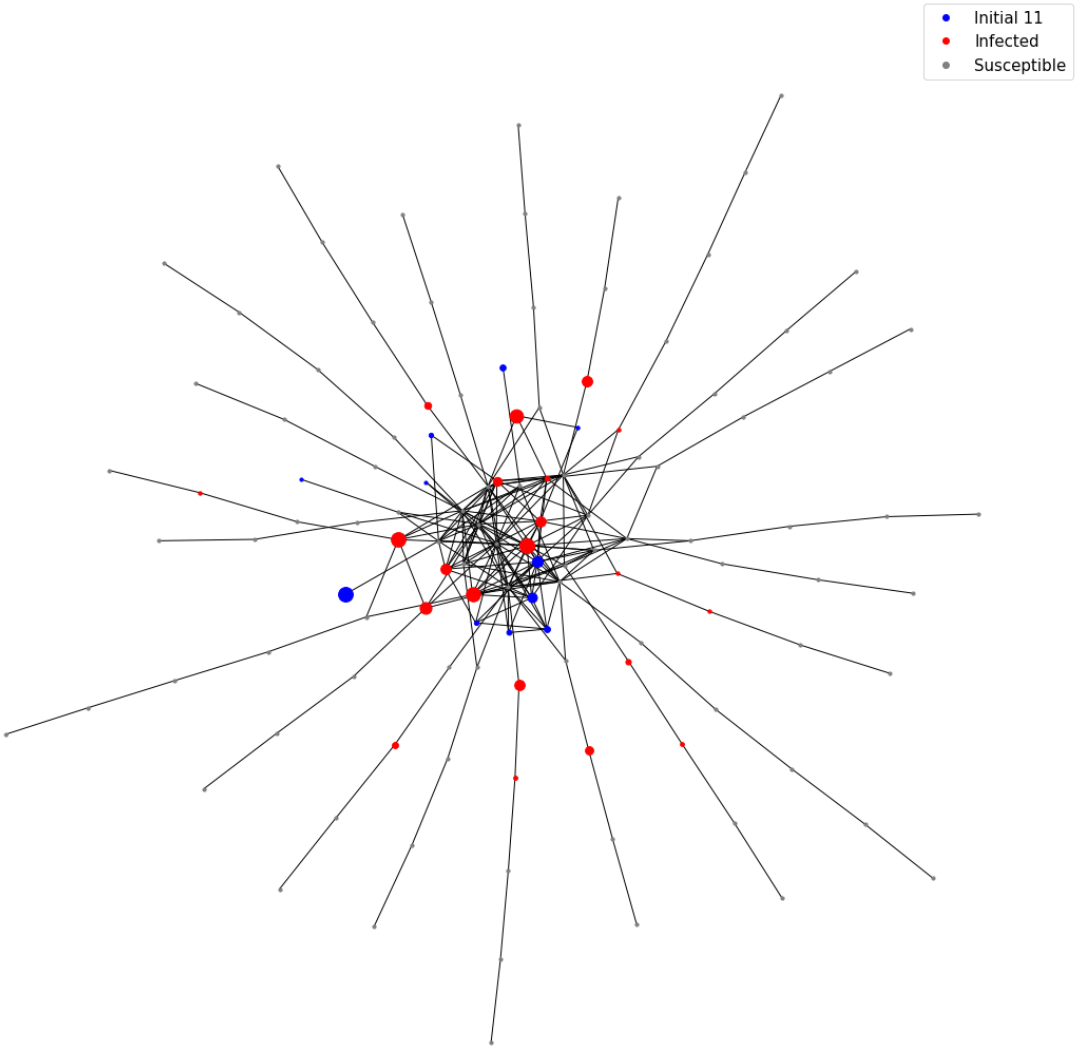
In [31]:
```
# folder path to save the plots
folder_path = r'D:\Desktop\cp_networks\project\results'
# Create the folder if it doesn't exist
os.makedirs(folder_path, exist_ok=True)

for t in time_stemps:
    sg = Graphs_per_month[t].subgraph(sample_near_11_nodes)
    plt.figure(figsize=(20, 20))
    pos = nx.kamada_kawai_layout(sg)

    for node in sample_near_11_nodes:
        size = sg.nodes[node]['weight']

        if node in initial_infected_nodes:
            nc = 'blue'
            ns = (size + 10)
        else:
            if size > 0:
                nc = 'red'
                ns = (size + 10)
            else:
                nc = 'grey'
                ns = (size + 10)

        nx.draw_networkx_nodes(sg, pos, nodelist=[node], node_color=nc,
node_size=ns)
    nx.draw_networkx_edges(sg, pos)
```

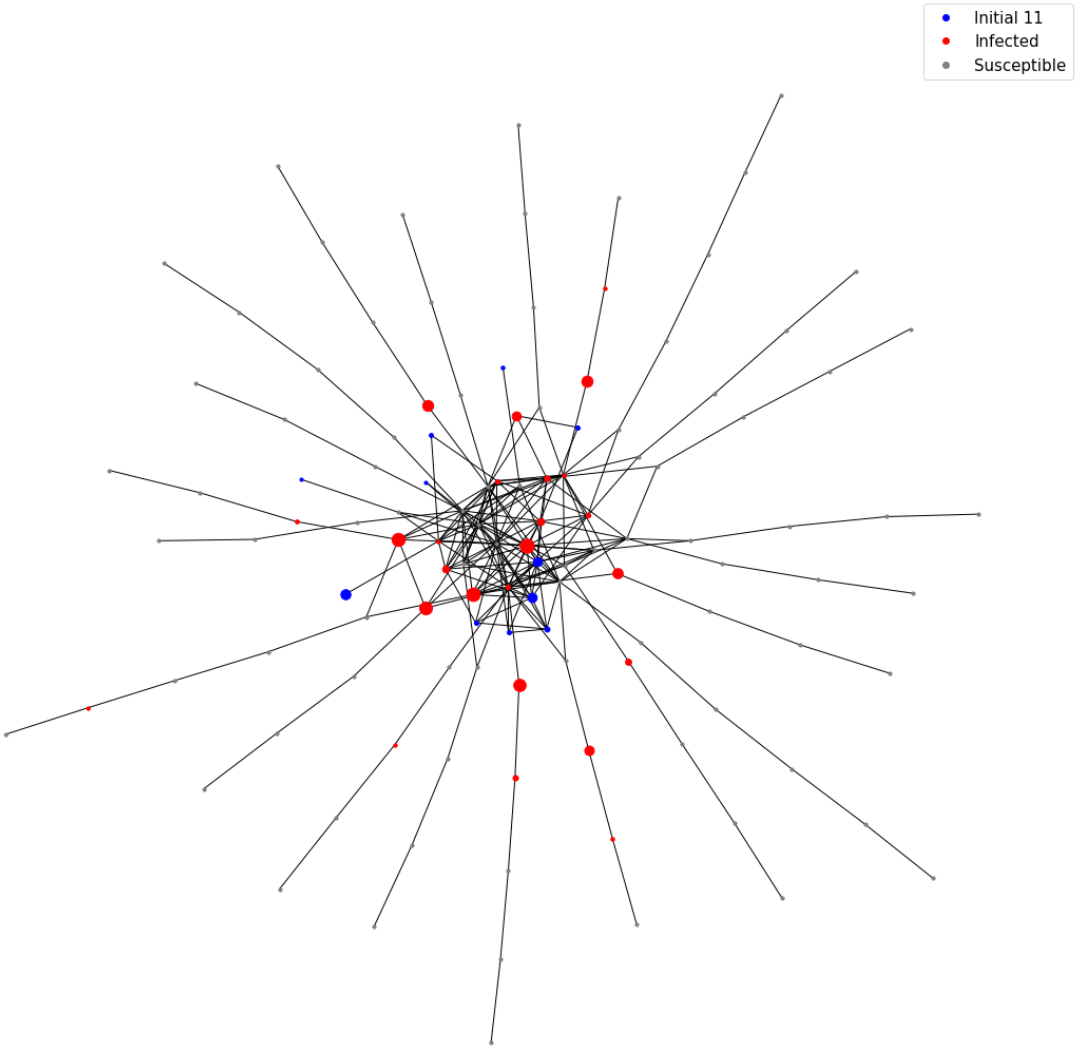
```
    legend_labels = ['Initial 11', 'Infected', 'Susceptible']
    legend_colors = ['blue', 'red', 'grey']
    legend_elements = [plt.Line2D([0], [0], marker='o', color=color,
label=label, linestyle='') for color, label in zip(legend_colors,
legend_labels)]

    plt.axis('off')
    plt.legend(handles=legend_elements, fontsize=15)
    plt.title(f"time {t}", fontsize=35).set_color("white")

    # Save the plot to the specified folder path
    filename = f"plot_{t}.png"
    plt.savefig(os.path.join(folder_path, filename))

    plt.show()
    plt.close()
```

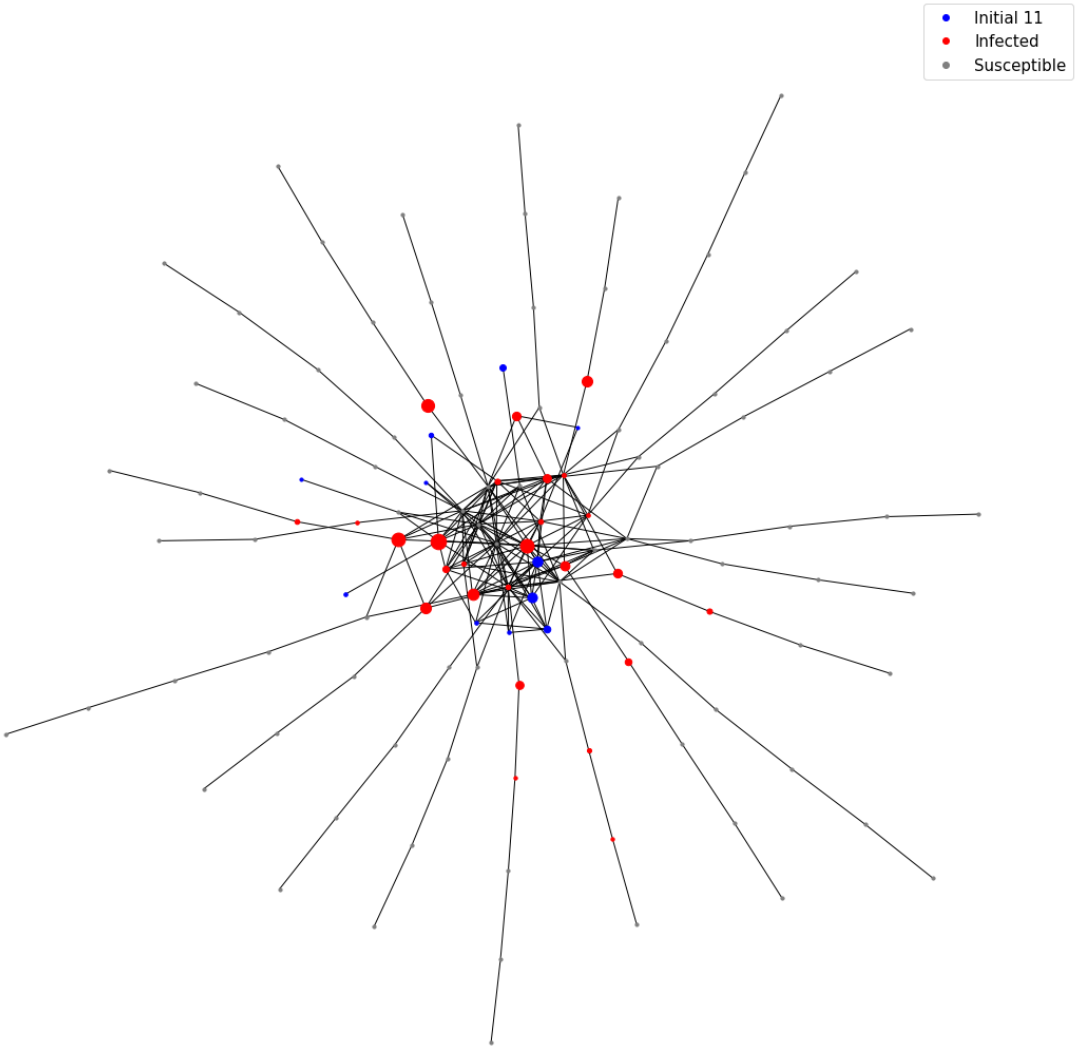


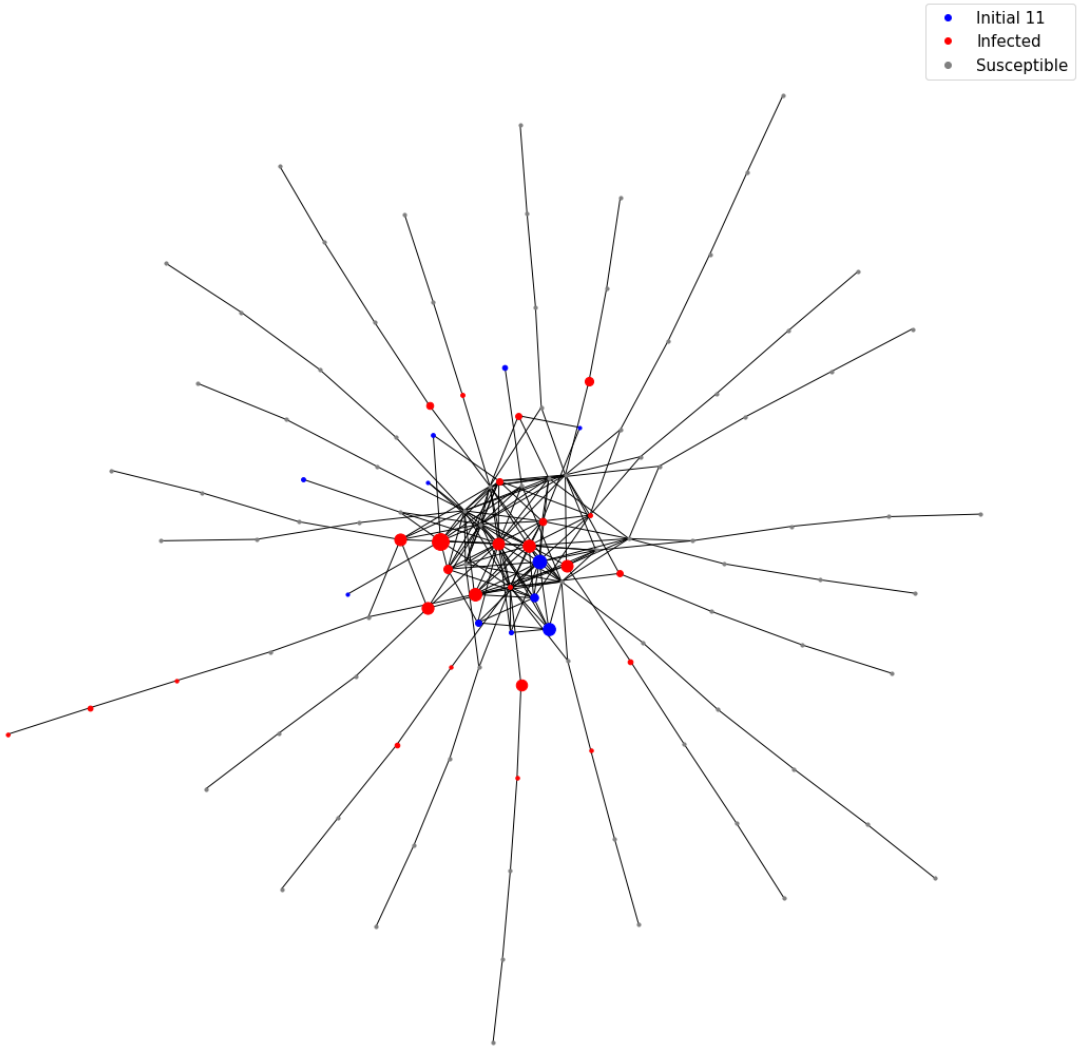

Initial 11
Infected
Susceptible

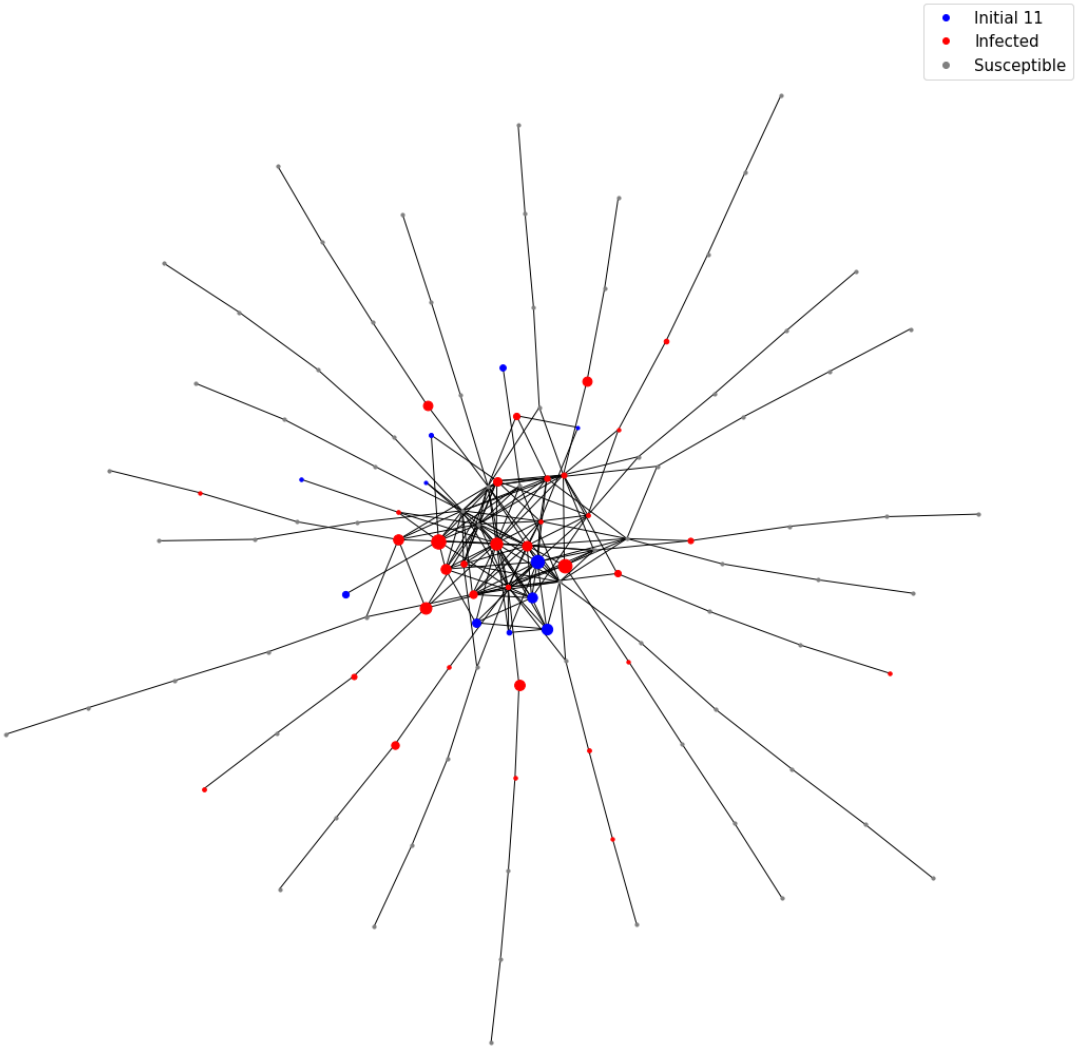

Initial 11
Infected
Susceptible

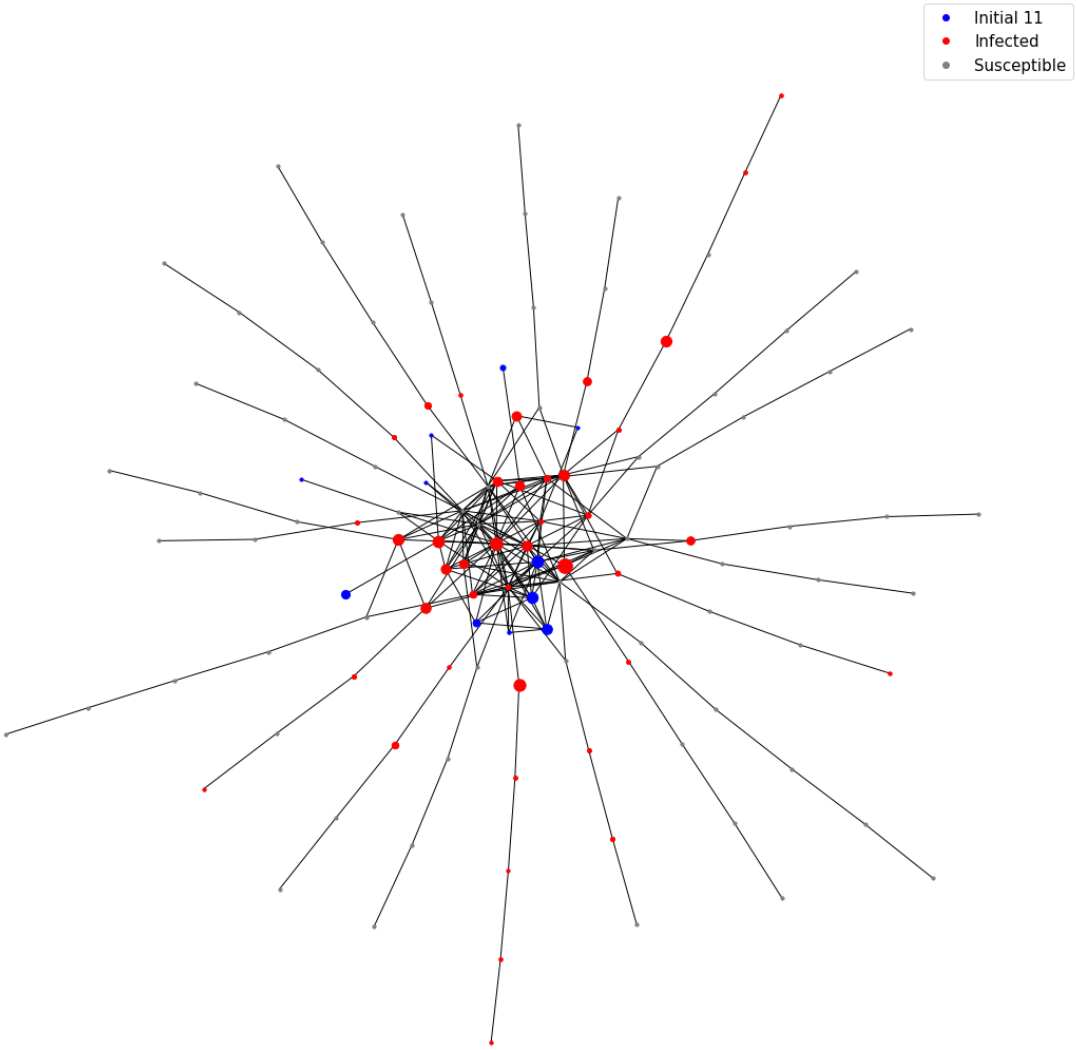

Initial 11
Infected
Susceptible

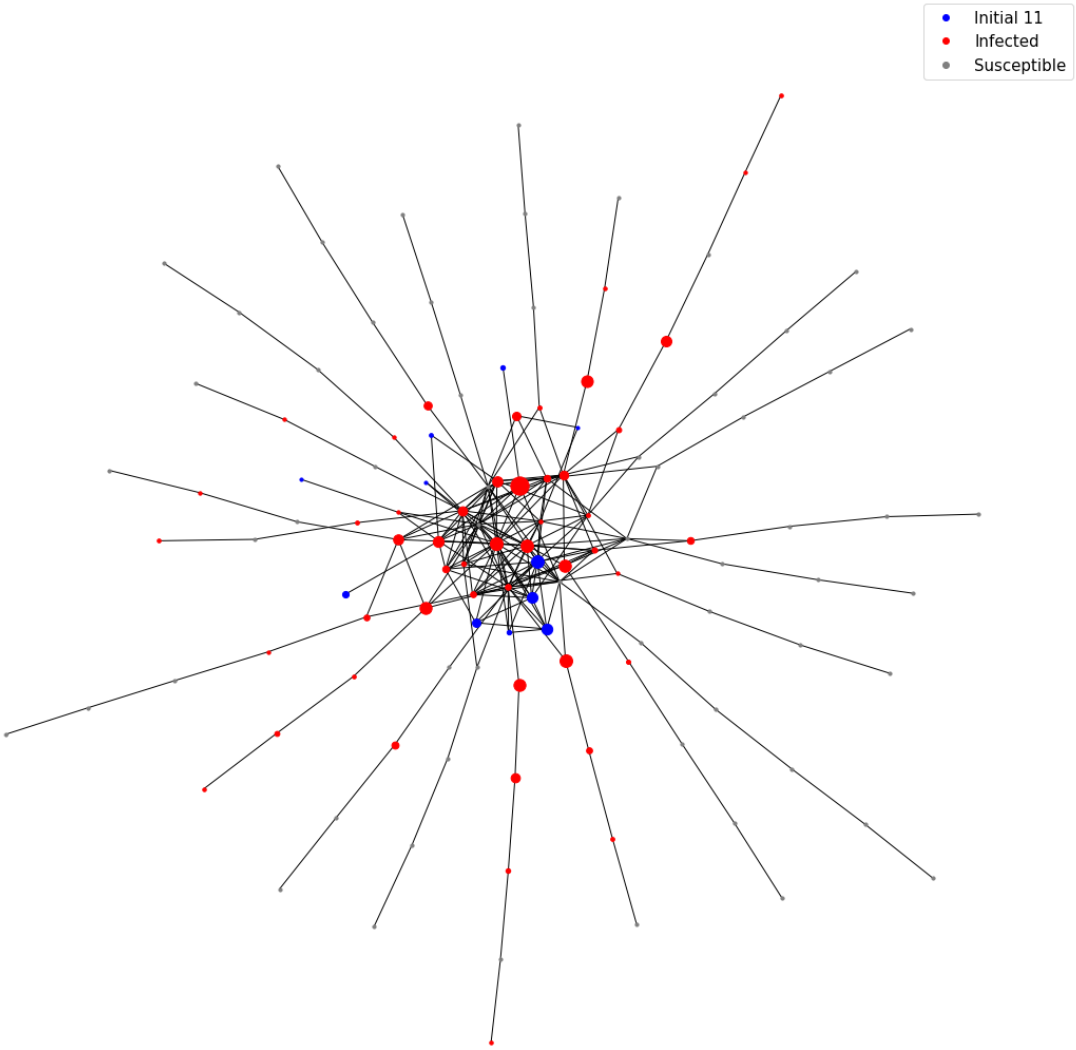

Initial 11
Infected
Susceptible

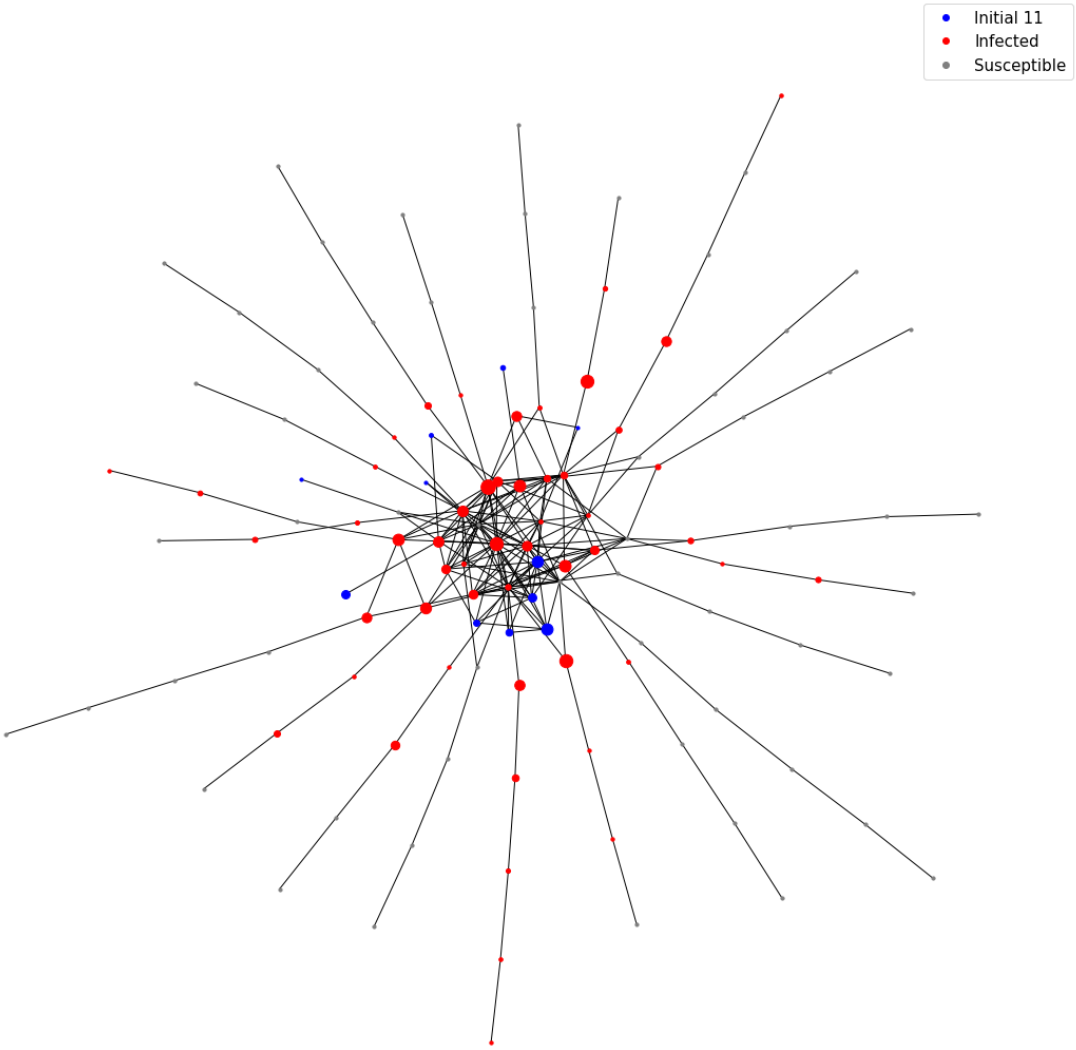

Initial 11
Infected
Susceptible

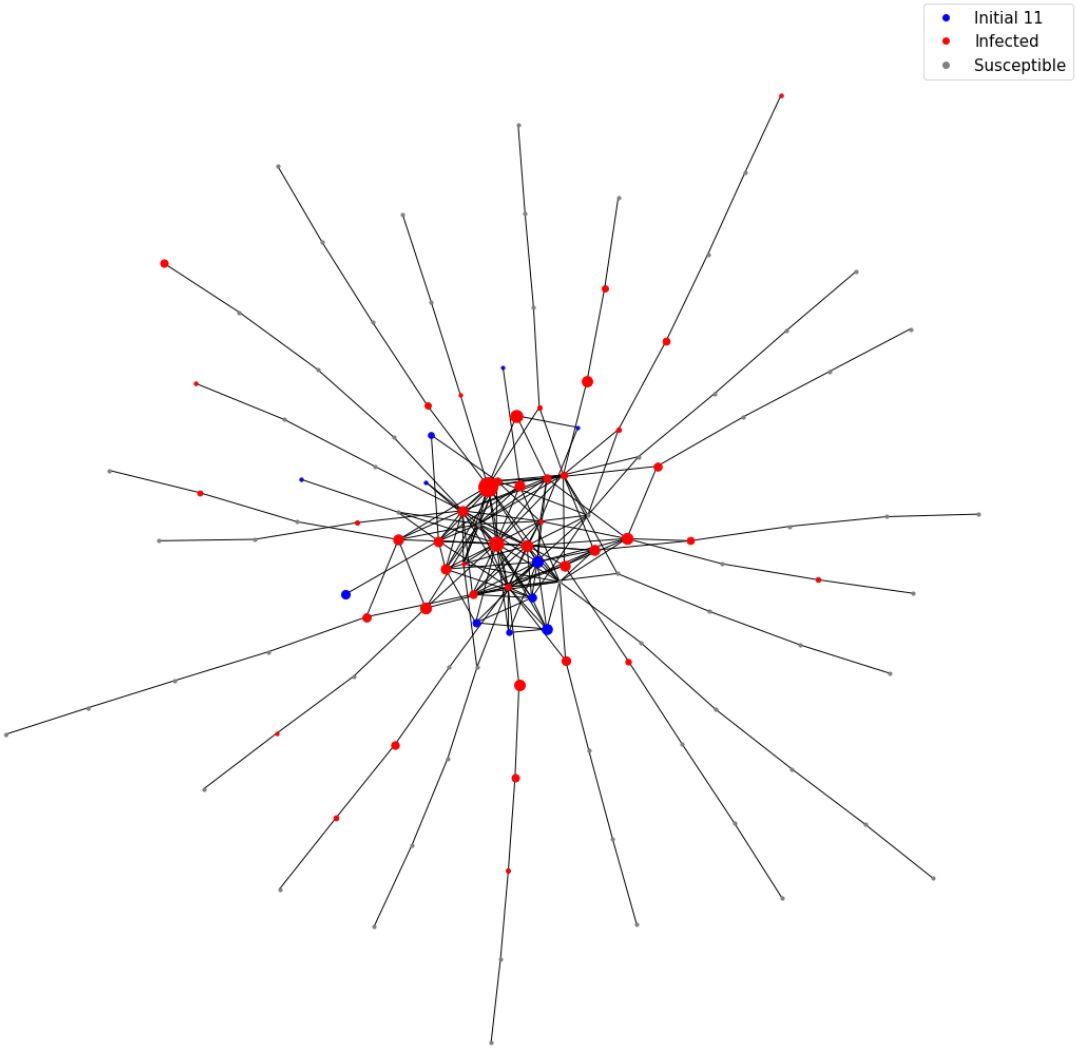

Initial 11
Infected
Susceptible

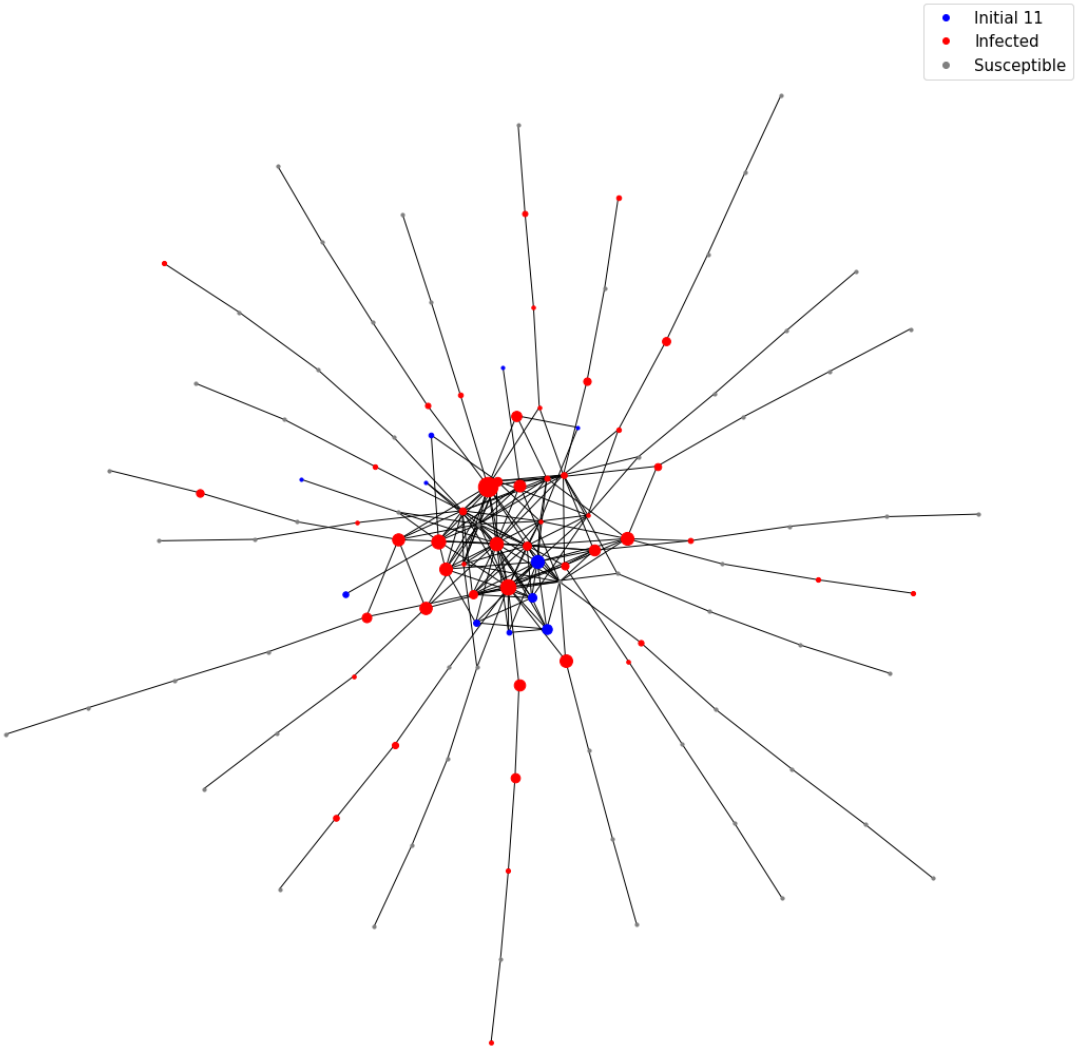

Initial 11
Infected
Susceptible

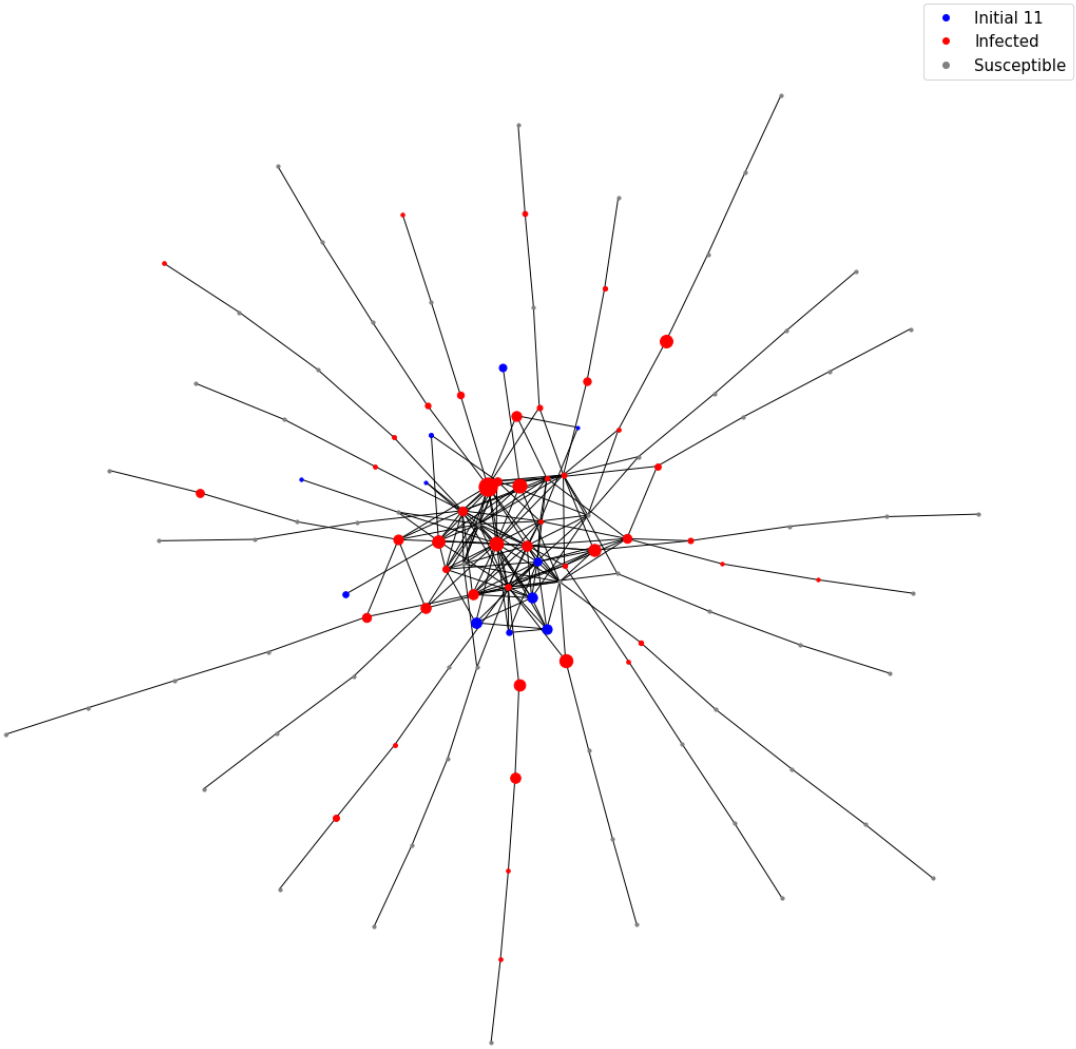

Initial 11
Infected
Susceptible

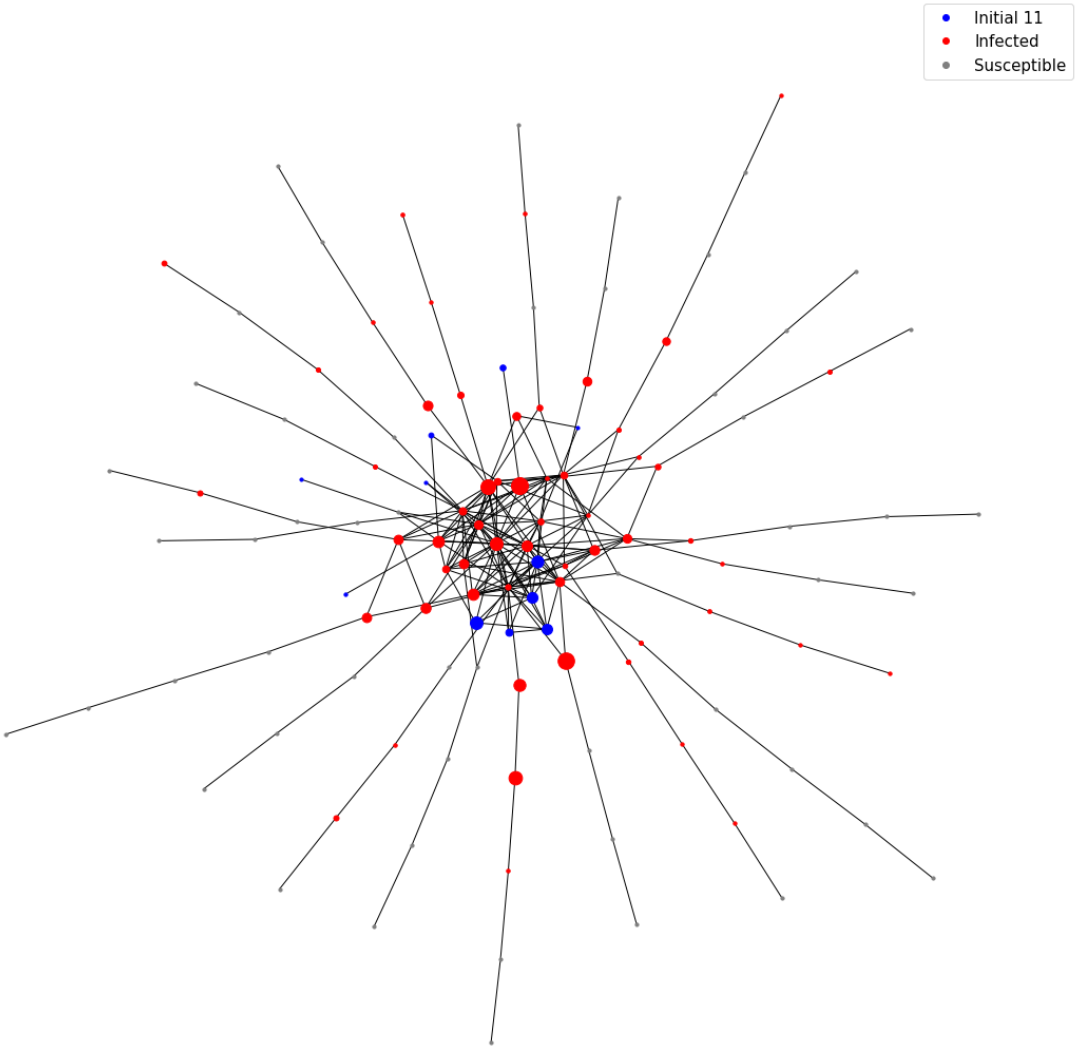

Initial 11
Infected
Susceptible

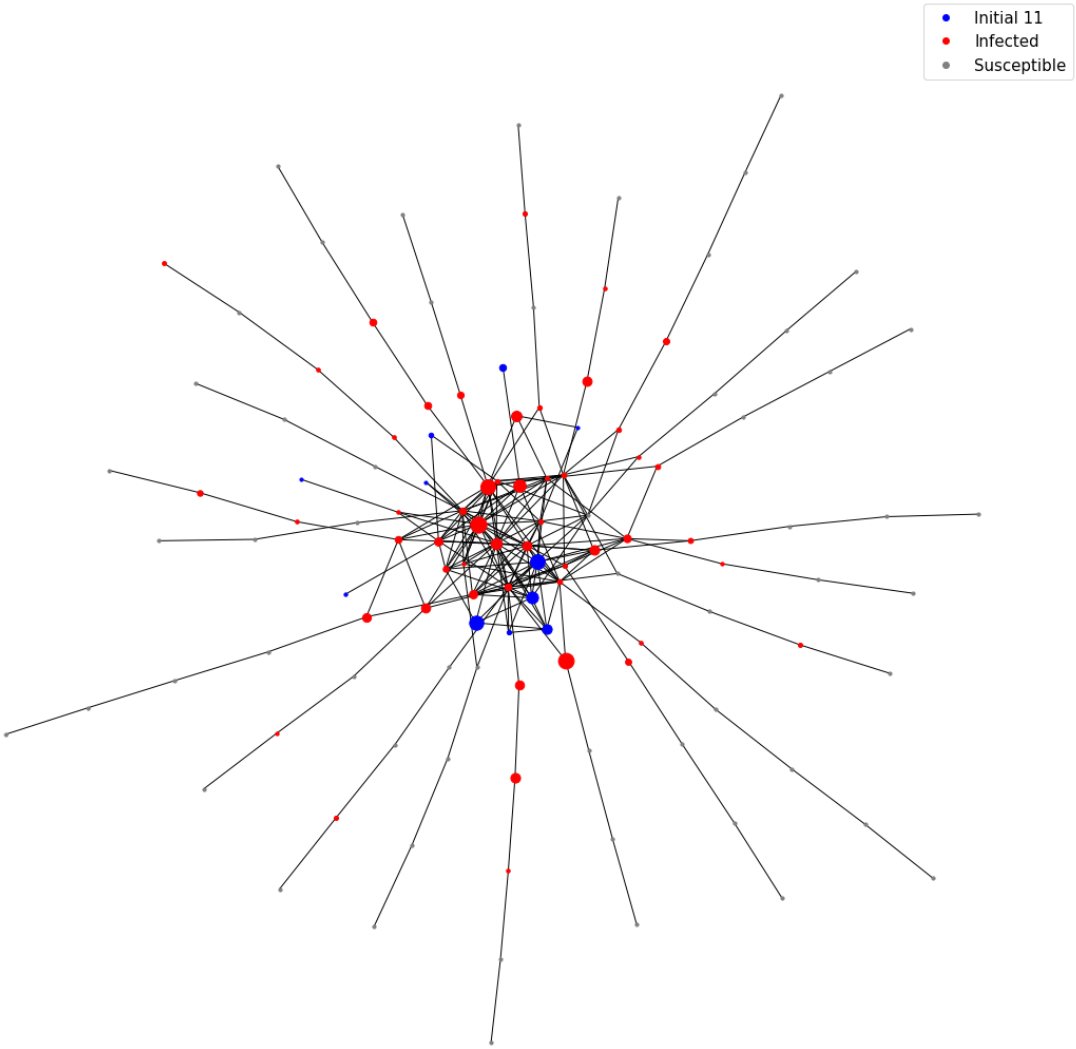

Initial 11
Infected
Susceptible

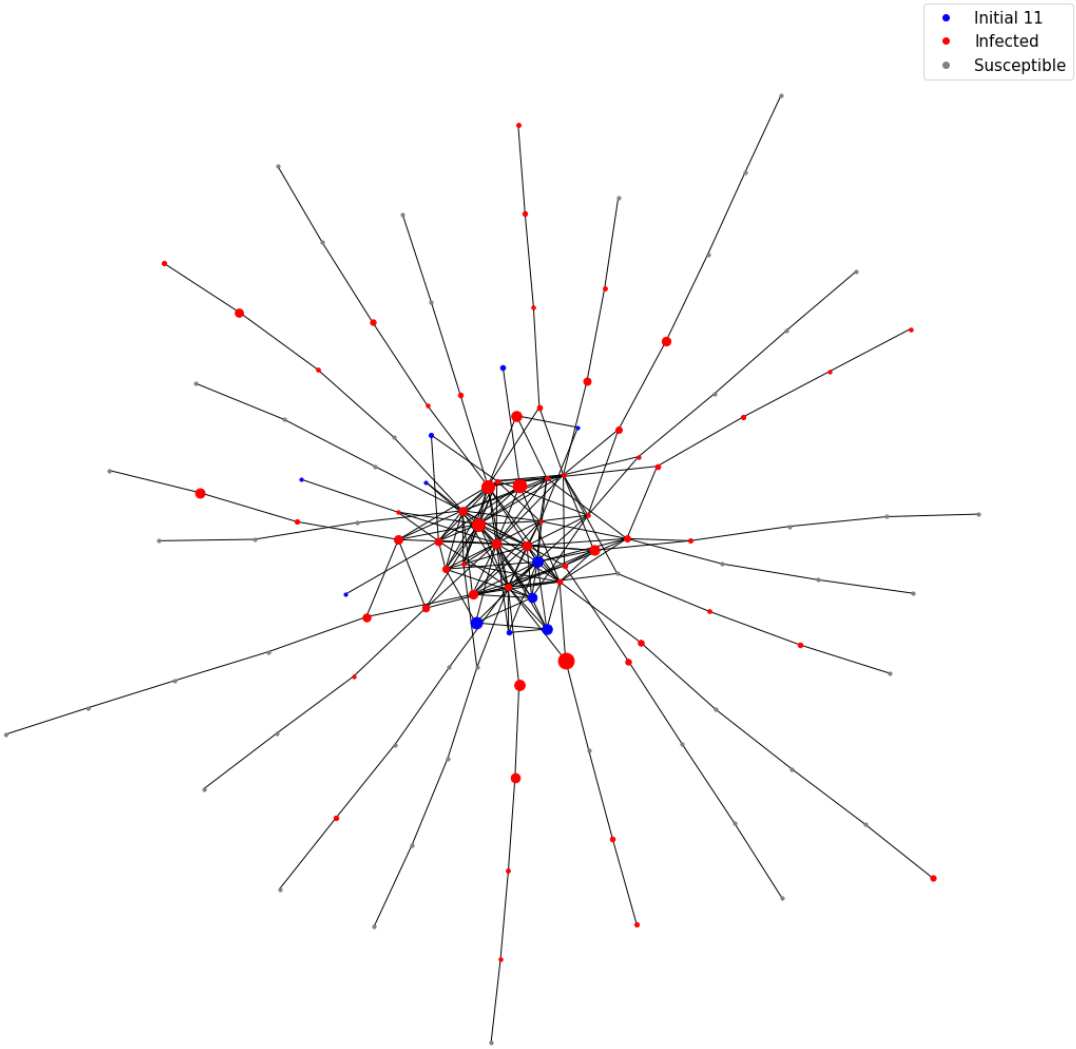

Initial 11
Infected
Susceptible

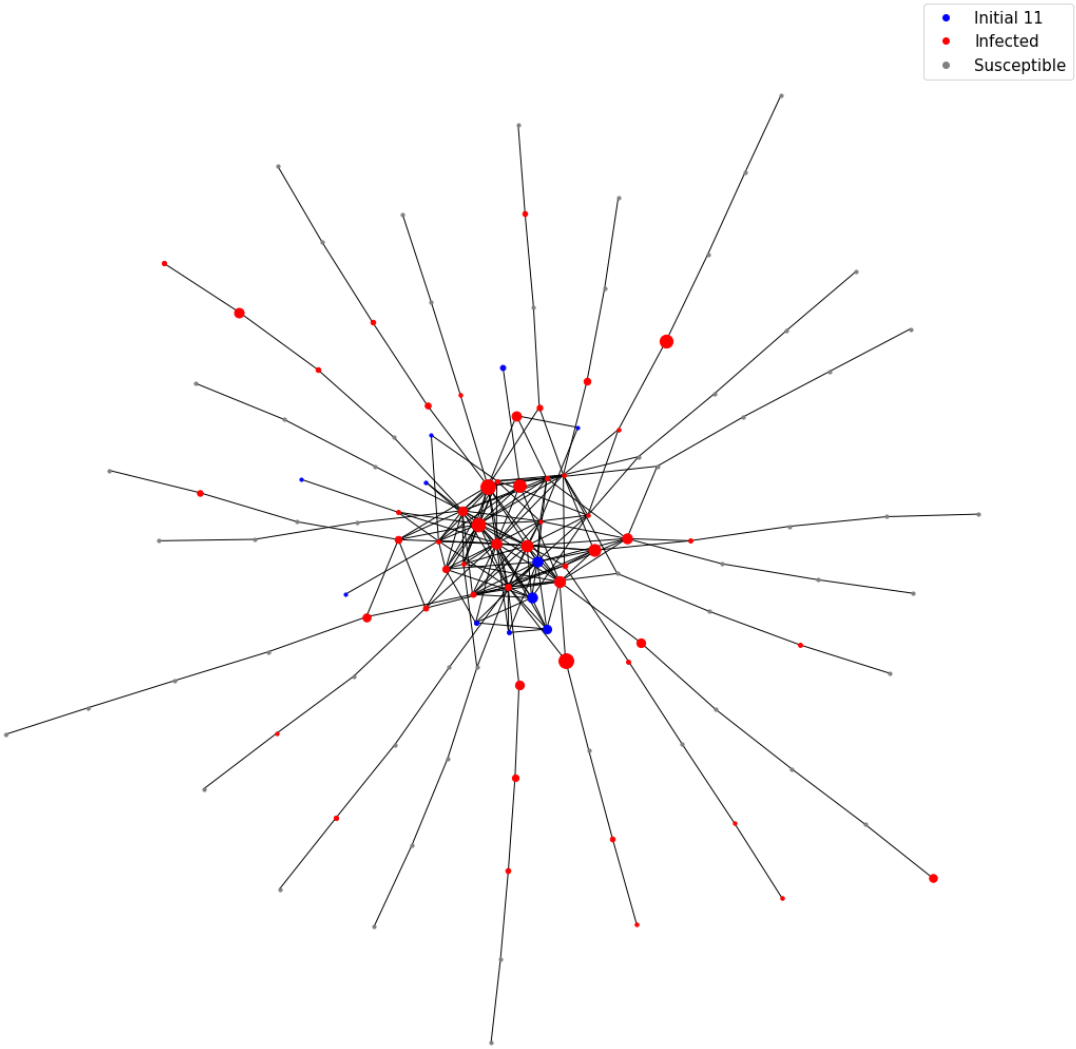

Initial 11
Infected
Susceptible

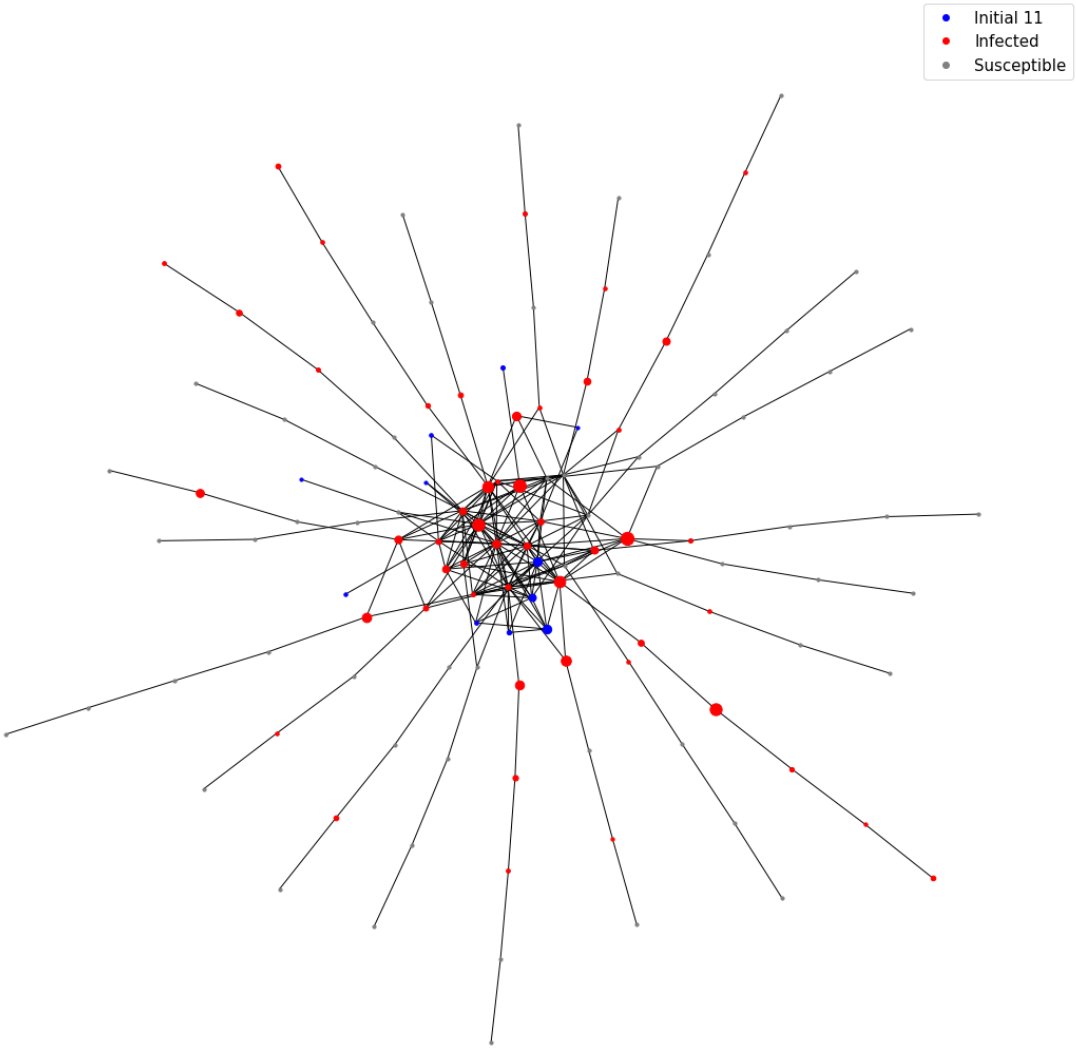

Initial 11
Infected
Susceptible

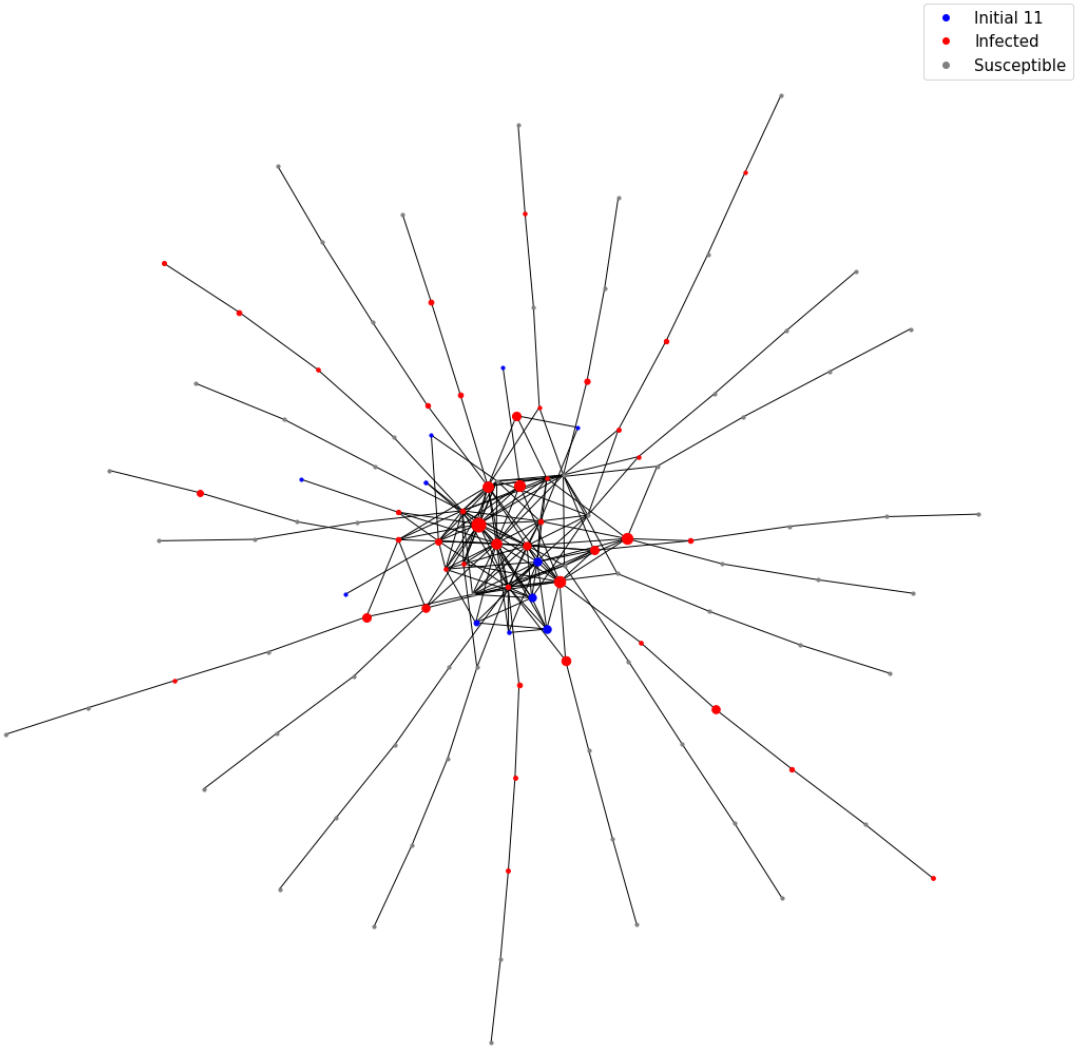

Initial 11
Infected
Susceptible

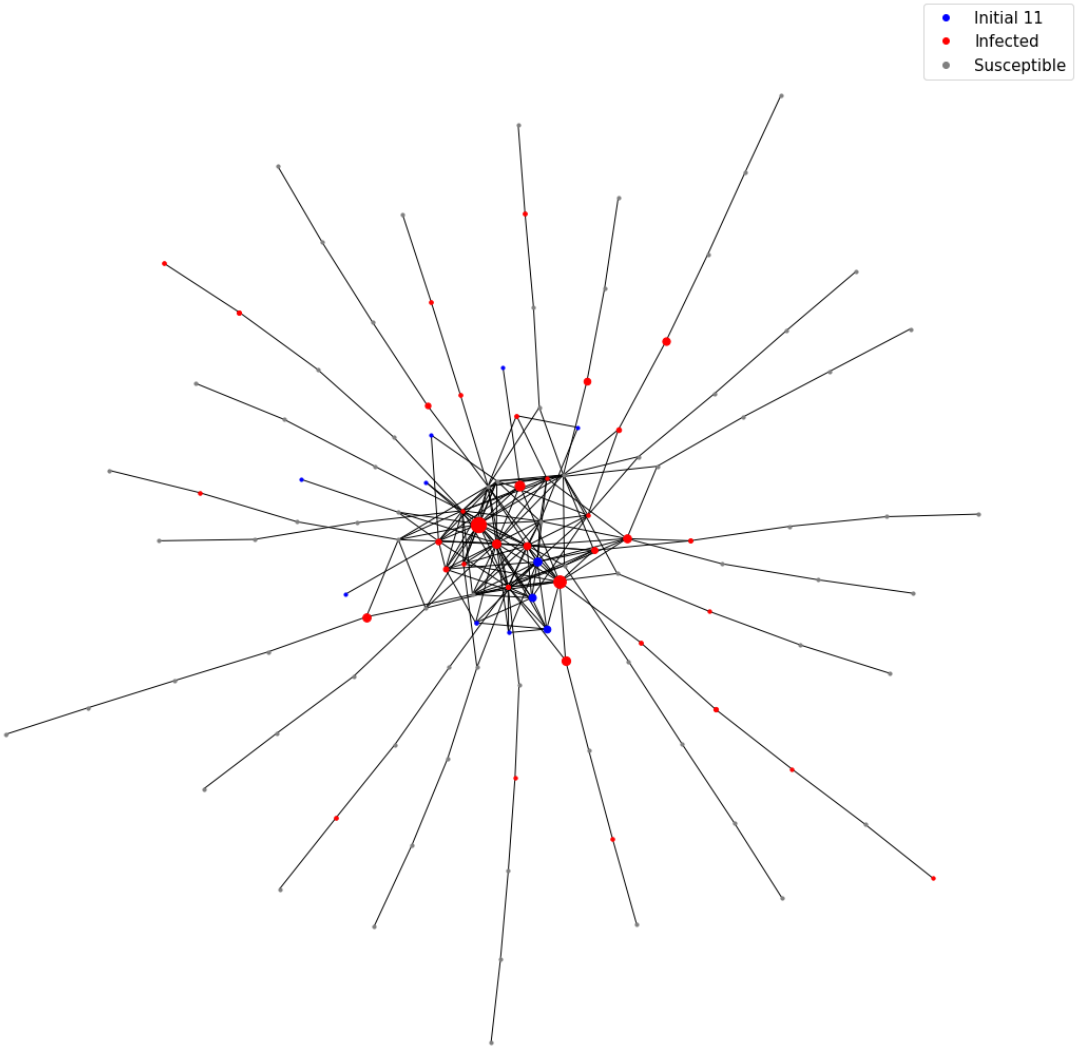

Initial 11
Infected
Susceptible

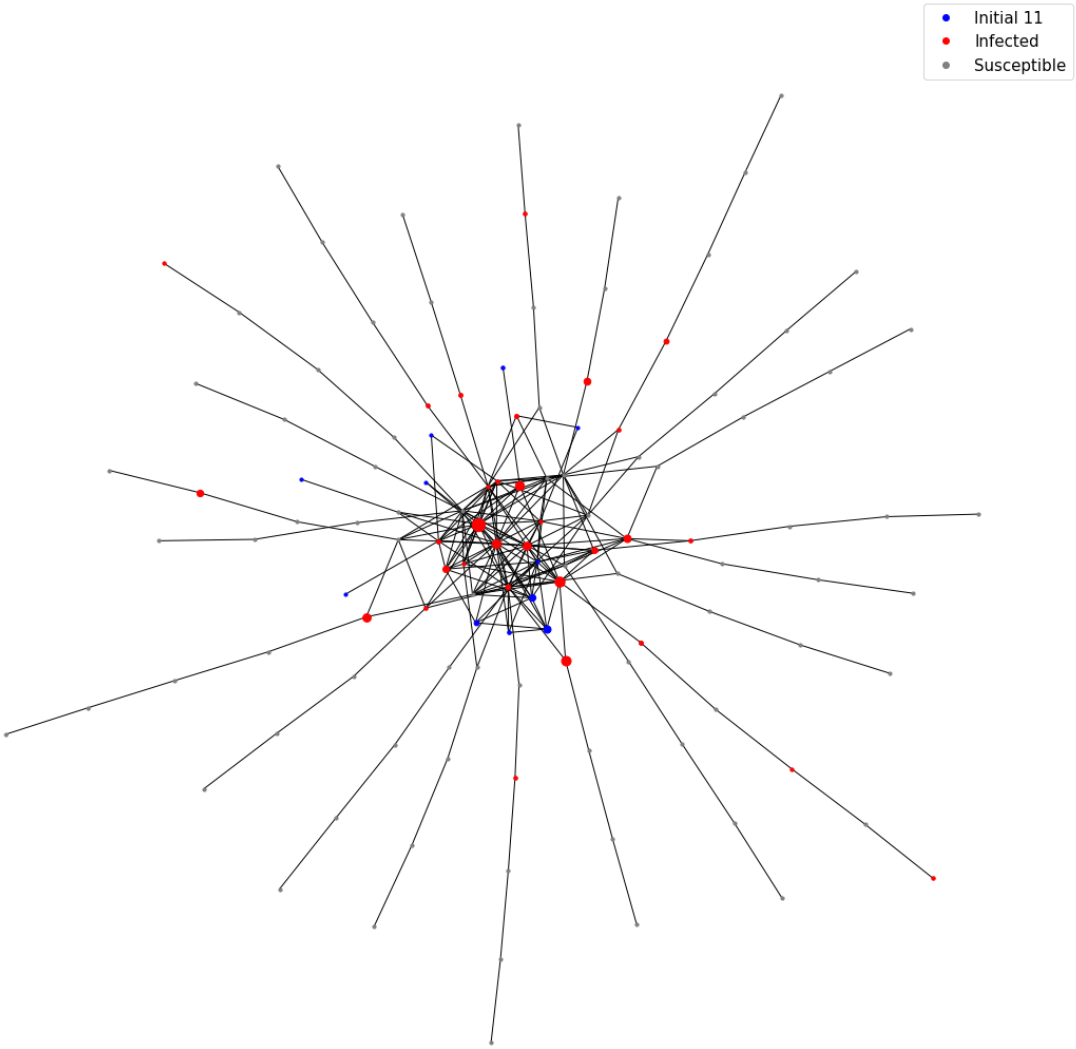

Initial 11
Infected
Susceptible

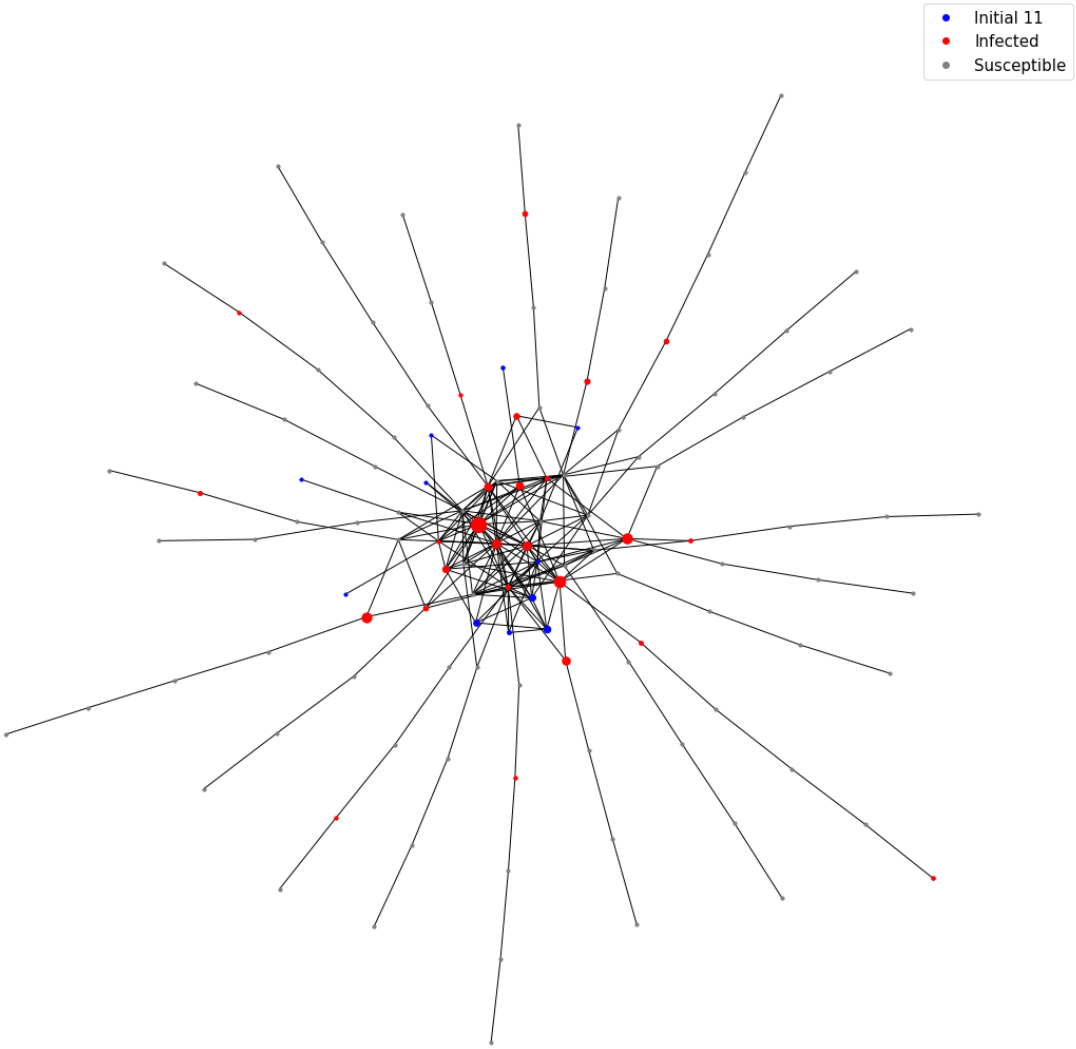

Initial 11
Infected
Susceptible

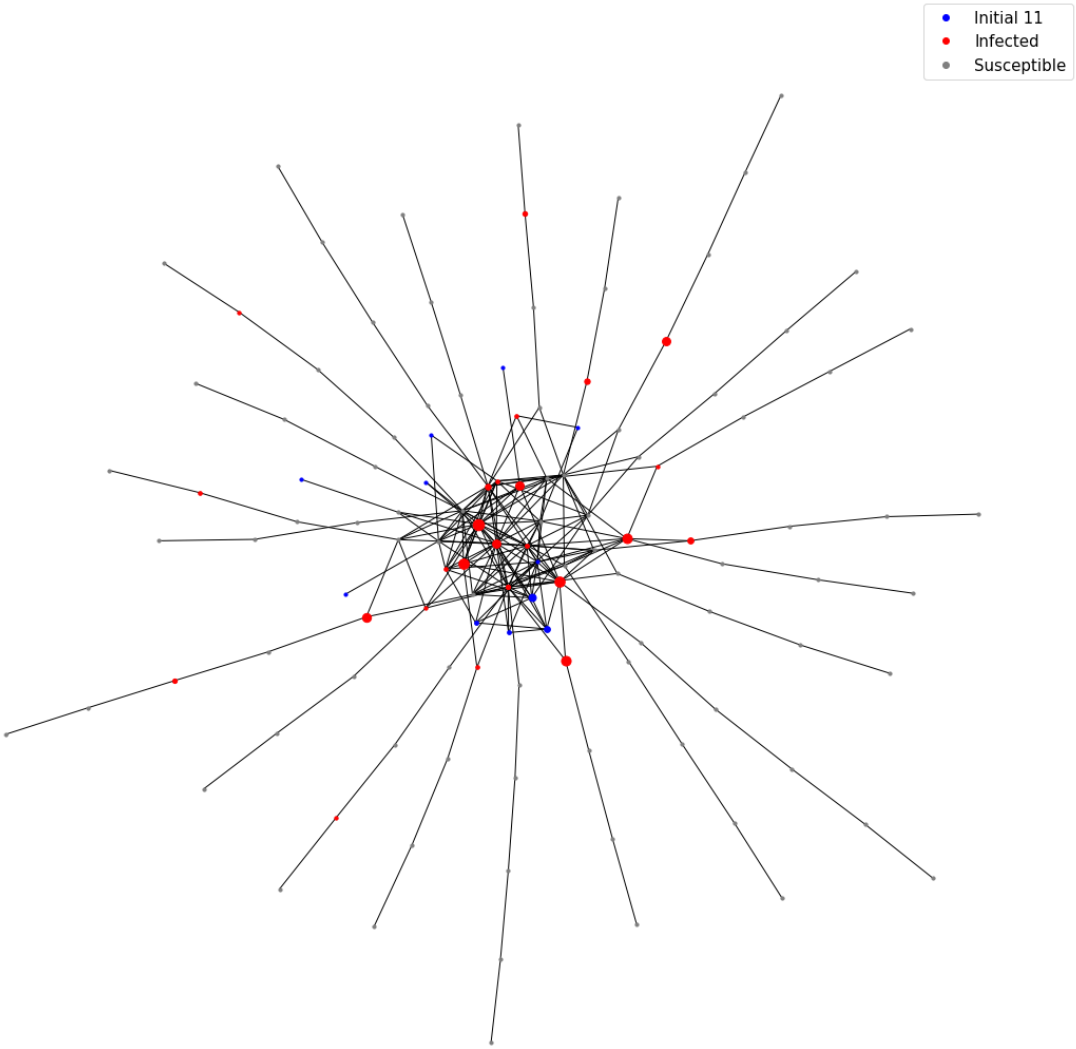

Initial 11
Infected
Susceptible

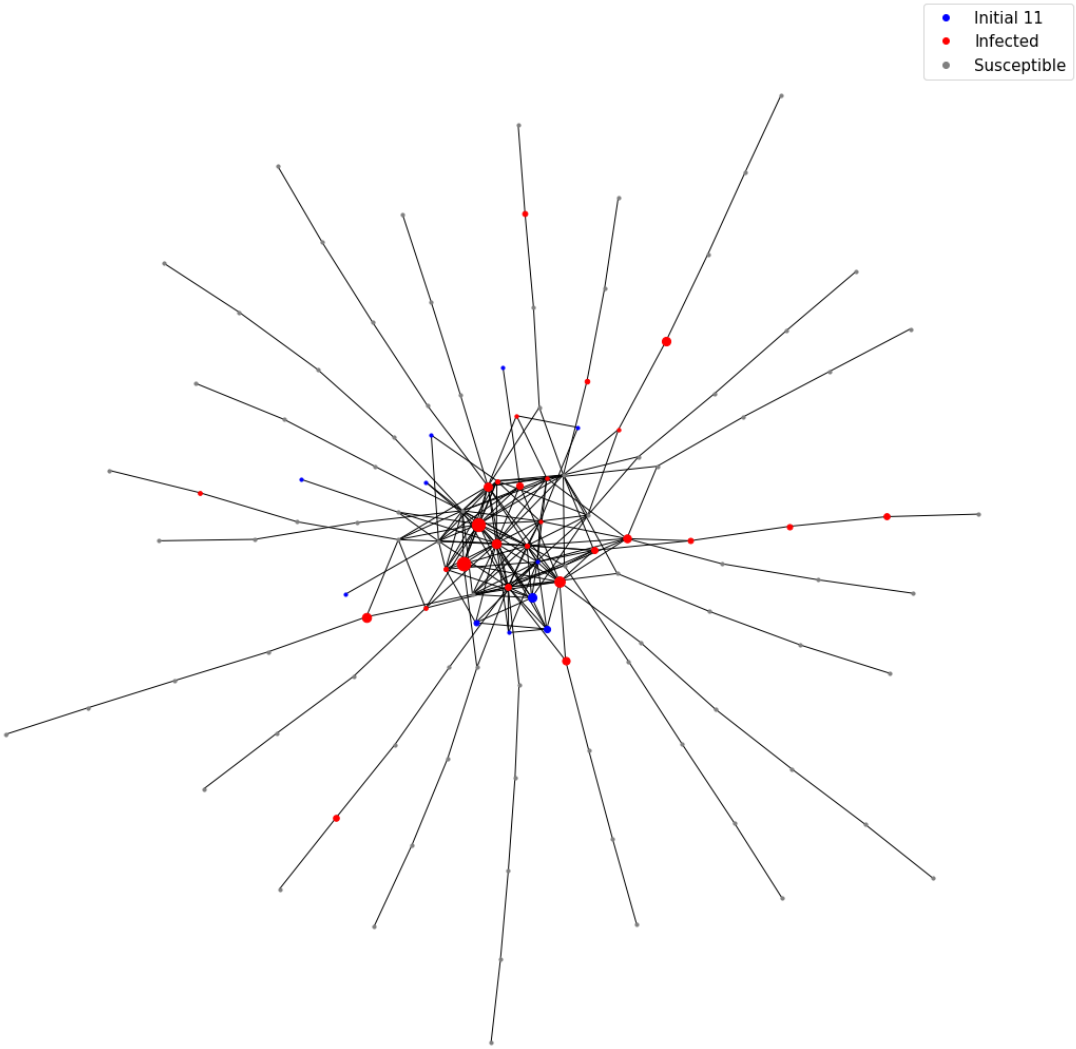

Initial 11
Infected
Susceptible

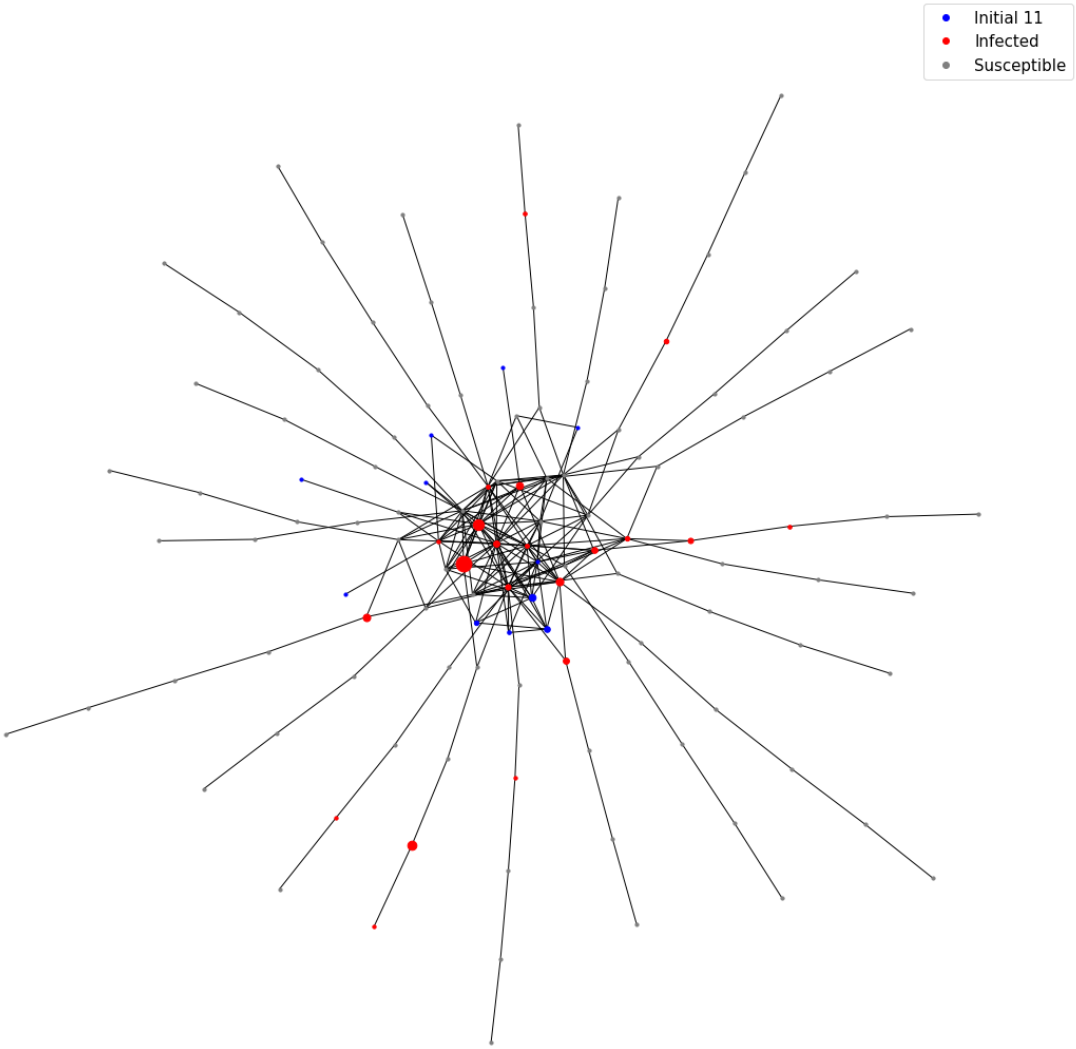

Initial 11
Infected
Susceptible

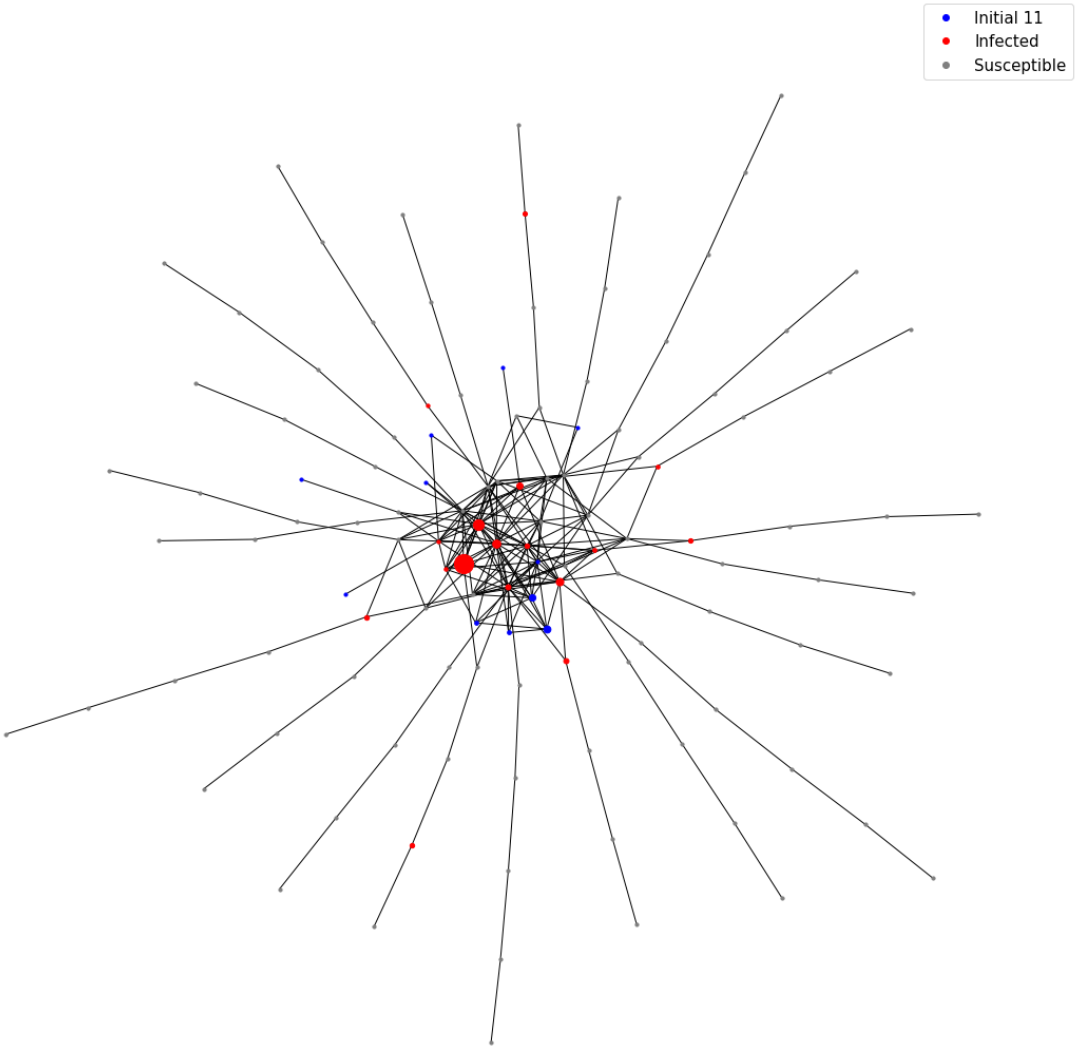

Initial 11
Infected
Susceptible

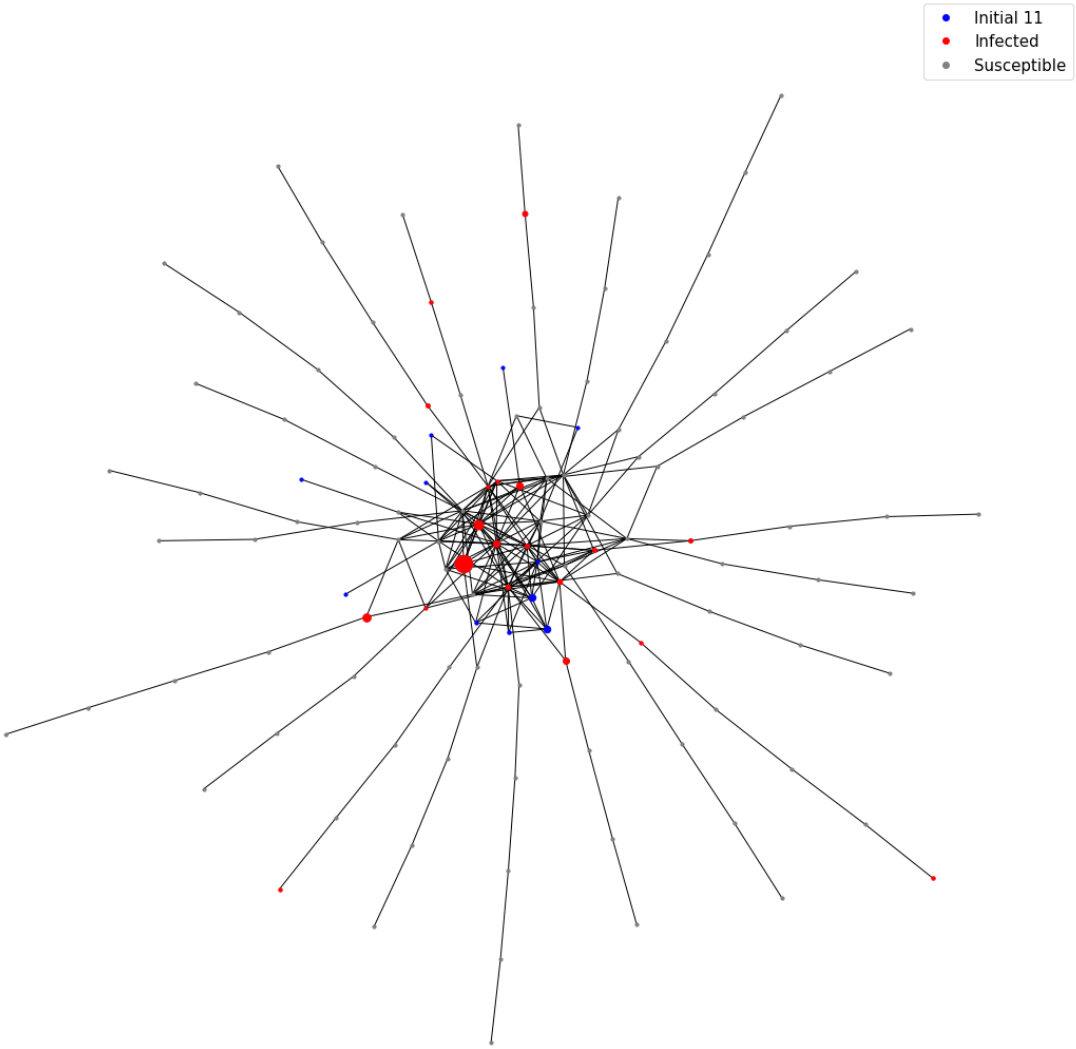

Initial 11
Infected
Susceptible

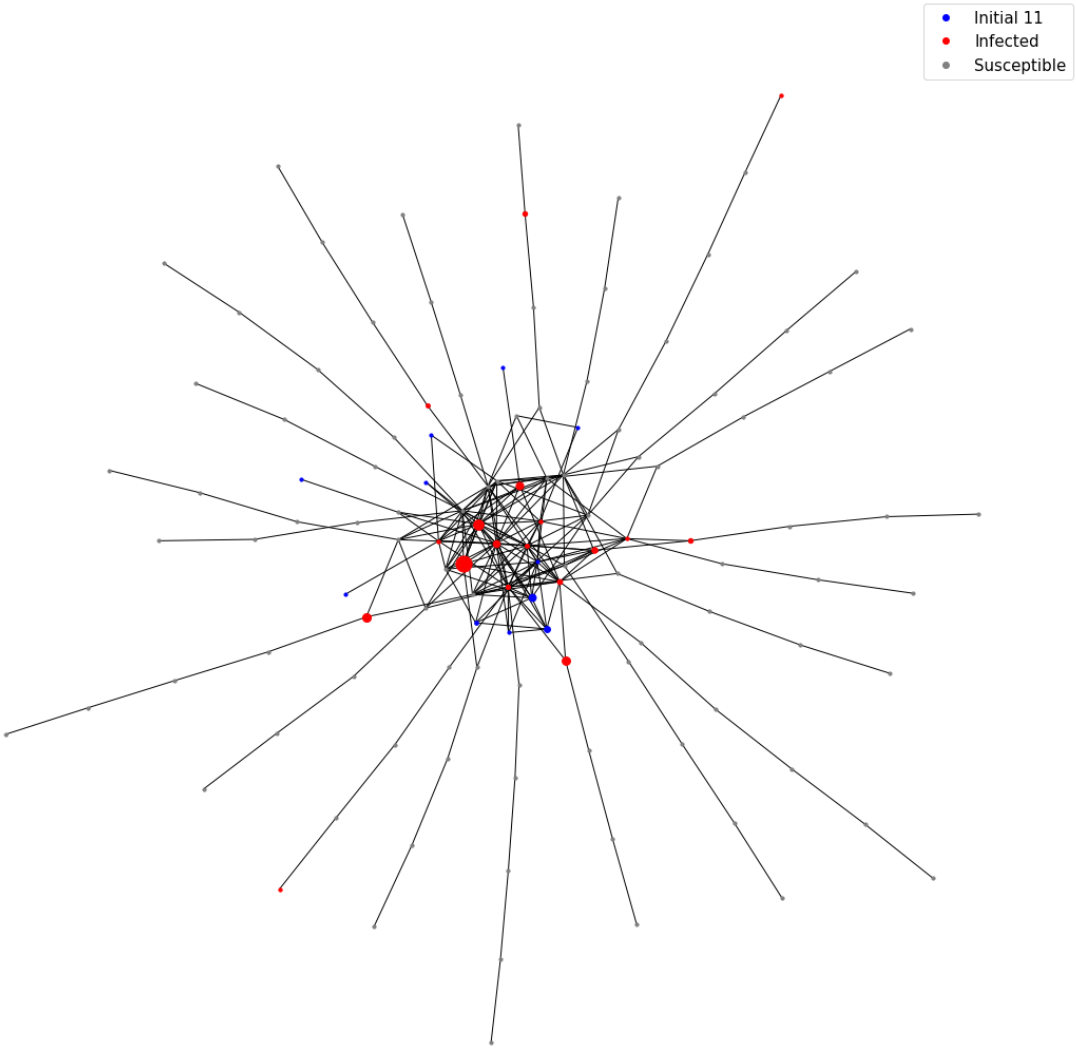

Initial 11
Infected
Susceptible

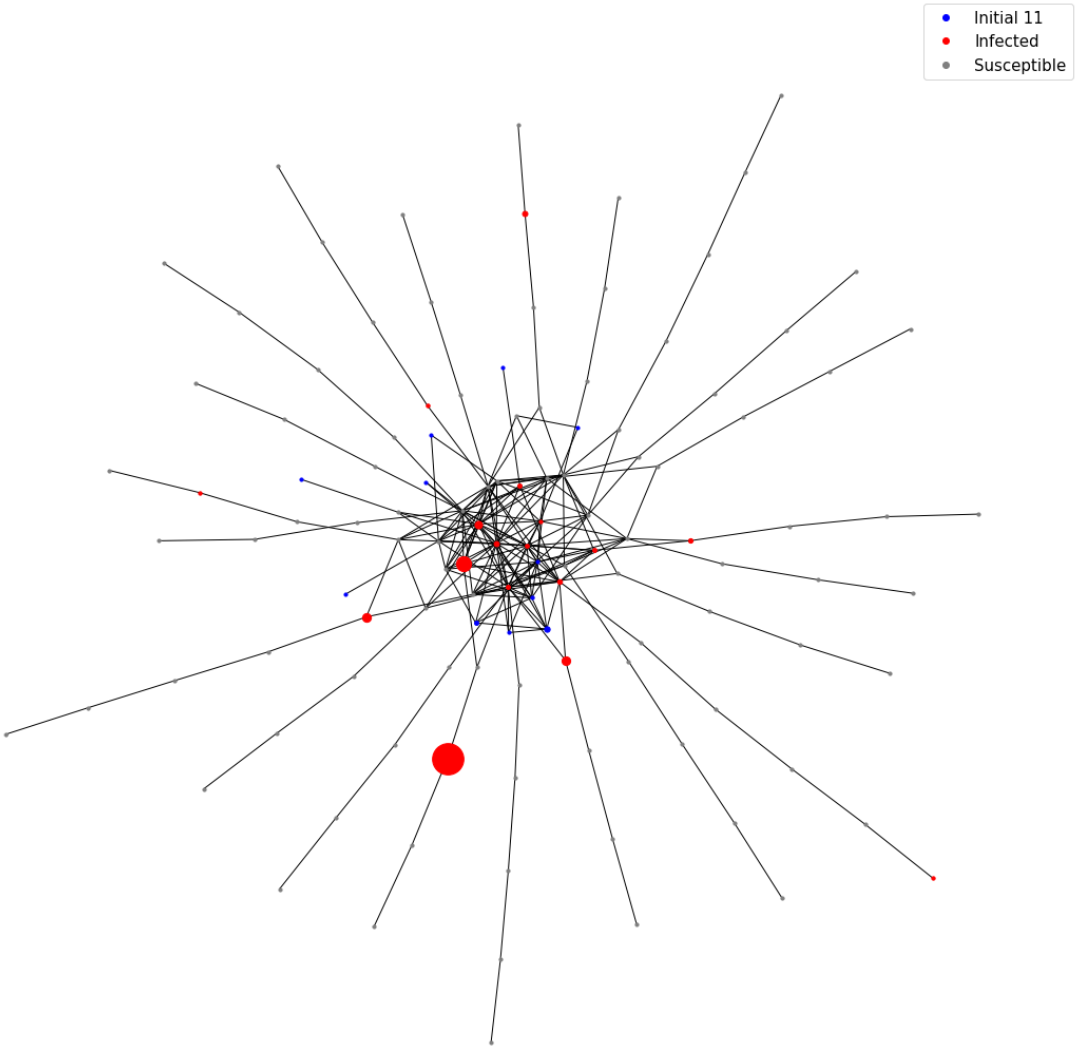

Initial 11
Infected
Susceptible

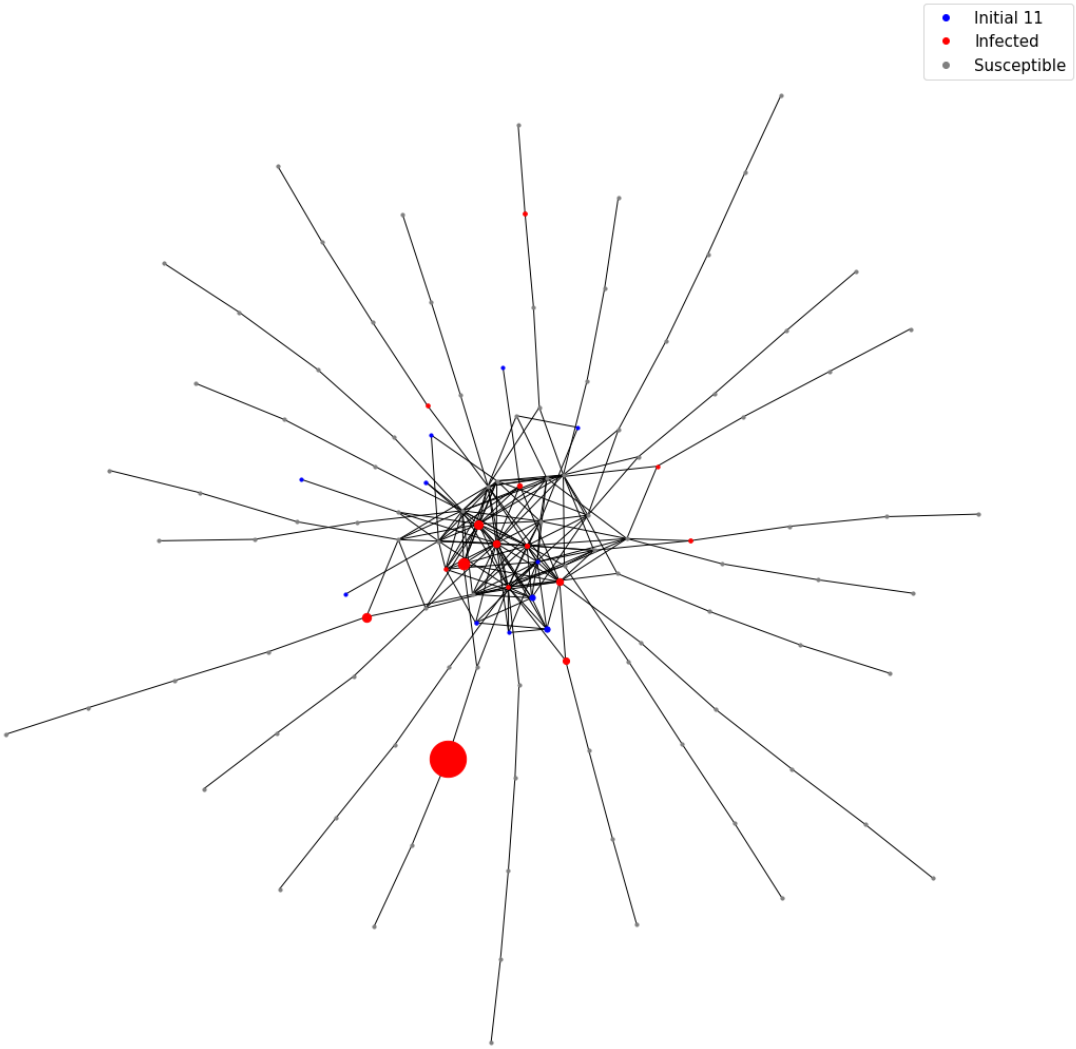

Initial 11
Infected
Susceptible

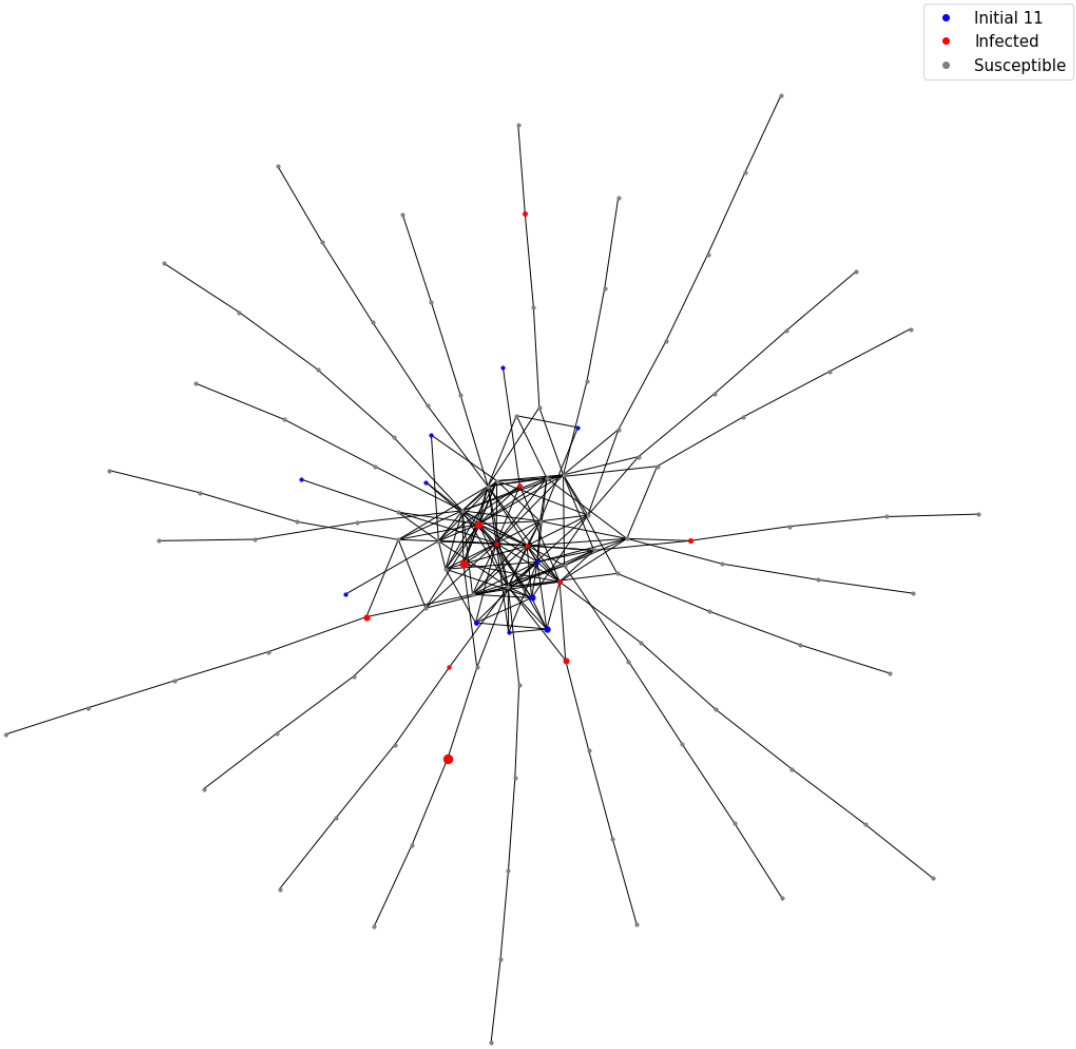

Initial 11
Infected
Susceptible


Initial 11
Infected
Susceptible


Initial 11
Infected
Susceptible


Initial 11
Infected
Susceptible


Initial 11
Infected
Susceptible



In [ ]:

Friends Influence on Check-ins

Comparison between new check-ins among user with/without friends with check-ins the the previous month

In [32]:
```
def get_all_infecteds(graph, th):
    infecteds = []
    for user in users:
        if graph.nodes[user]['weight'] > th:
            infecteds.append(user)
    return infecteds

def get_nodes_neighbors(graph, nodes):
    neighbors = set(nodes)
    for n in nodes:
```

```
        neighbors = neighbors.union(set(list(graph.neighbors(n))))
    return neighbors
```

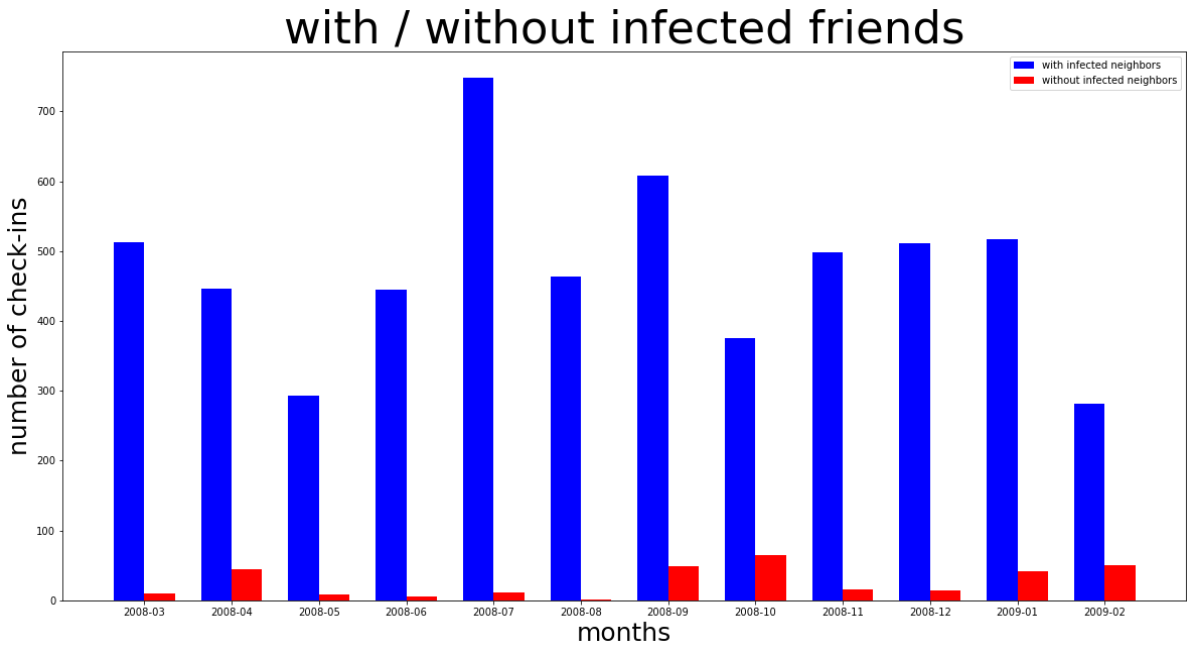
In [33]:

```
with_infected = []
without_infected = []
for i, ts in enumerate(time_stemps[:-20]):
    g = Graphs_per_month[ts]
    infected_users = get_all_infecteds(g, 0)

    users_with_infected_neighbors = get_nodes_neighbors(g,
infected_users)
    users_without_infected_neighbors = set(users)-
set(users_with_infected_neighbors)

    sample_with = random.sample(list(users_with_infected_neighbors),
100)
    tot_with = 0
    for n in sample_with:
        tot_with += Graphs_per_month[time_stemps[i+1]].nodes[n]
['weight']

    sample_without =
random.sample(list(users_without_infected_neighbors), 100)
    tot_without = 0
    for n in sample_without:
        tot_without += Graphs_per_month[time_stemps[i+1]].nodes[n]
['weight']

    with_infected.append(tot_with)
    without_infected.append(tot_without)
```

In [34]:

```
plt.figure(figsize=(20,10))
bar_width = 0.35
pos1 = np.arange(len(time_stemps[:-20]))
pos2 = pos1 + bar_width

plt.bar(pos1, with_infected, color='blue', width=bar_width, label='with
infected neighbors')
plt.bar(pos2, without_infected, color='red', width=bar_width,
label='without infected neighbors')
```

```
plt.xlabel('months', fontsize=25)
plt.ylabel('number of check-ins', fontsize=25)
plt.title('with / without infected friends', fontsize=45)

# Adding x-axis tick labels
plt.xticks(pos1 + bar_width / 2, time_stemps[:-20])

# Adding legend
plt.legend()

# Displaying the plot
plt.show()
```



for time x: both groups are represent random nodes without check-ins in month x, such that

- blue group: **has** neighbors with check-ins in nonth x
- red group: **dosn't have** neighbors with check-in in month x

Clustering Coefficient

In [35]:

```
# Calculate the local clustering coefficient of all nodes
nodes_CC = []
for user in users:
    nodes_CC.append(nx.clustering(G, user))

print(f"clustering coefficient mean (Global Clustering Coefficient):
{np.mean(nodes_CC):.1f}")
print(f"clustering coefficient median: {np.median(nodes_CC):.1f}")
ps = np.percentile(nodes_CC, [25,75])
```

```
print(f"clustering coefficient 25,50,75 percentiles: {ps[0]:.1f},
{ps[1]:.1f}")
```

```
clustering coefficient mean (Global Clustering Coefficient): 0.2
clustering coefficient median: 0.0
clustering coefficient 25,50,75 percentiles: 0.0, 0.2
```

Comminities detection

In [36]:
```
# find nodes in gcc that has check-ins
nodes_with_checkins = set(list(started_in[0]) + list(started_in[1]) +
list(started_in[2]))
nodes_in_gcc = set(gcc)
CD_nodes = set(users)
CD_nodes = CD_nodes.intersection(nodes_in_gcc, nodes_with_checkins)
```

In [37]:
```
# sub graph based on CD_nodes
CD_graph = G.subgraph(CD_nodes)
```

In [38]:
```
# adding each user his weighted average location
for user in CD_nodes:
    filt = brightkite_totalCheckins["user"] == user
    user_latitides = list(brightkite_totalCheckins["latitude"][filt])
    user_longitude = list(brightkite_totalCheckins["longitude"][filt])
    if(len(user_latitides) == 0 or len(user_latitides) == 0):
        continue
    user_location = [np.average(user_latitides),
np.average(user_longitude)]
    CD_graph.nodes[user]['user_location'] = user_location
```

In [39]:
```
# finding friendship based communities using louvain algorithm
partition = community.community_louvain.best_partition(CD_graph)
community_ids = list(set(partition.values()))
luvain_communities = {key: [] for key in community_ids} # key =
community ID, values = nodes in community
for node, community_id in partition.items():
    luvain_communities[community_id].append(node)

print("Louvain Algorithm Results:\n")
print(f"    - there are {len(luvain_communities)} communities")
print(f"    - each community has {np.average([len(l) for l in
luvain_communities.values()])} members on average")
```

```
louvain_community_score = nx.community.modularity(CD_graph,
luvain_communities.values())
print(f'\nLouvain algorithm community score:
{louvain_community_score:.1f}')
```

```
Louvain Algorithm Results:

    - there are 618 communities
    - each community has 81.36407766990291 members on average

Louvain algorithm community score: 0.7
```

locations handling

In [40]:
```
# find all unique locations
locations = set(list(brightkite_totalCheckins['location id']))
number_of_locations = len(locations)
print(f"there are {number_of_locations:,} unique locations")
```

```
there are 772,966 unique locations
```

In [41]:
```
# find popular locations
th = 10

location_hist = {key: 0 for key in locations}
for row in brightkite_totalCheckins.iterrows():
    cur_loc = row[1][4]
    location_hist[cur_loc] += 1

popular_locations_dict = {key: value for key, value in
location_hist.items() if value >= th}
print(f"there are {len(popular_locations_dict)} location that were
visited more then {th} times")
```

```
there are 48181 location that were visited more then 10 times
```

In [42]:
```
# dataframe of unique locations
kl = brightkite_totalCheckins[~brightkite_totalCheckins['location
id'].duplicated()]
kl = kl[['location id', 'latitude', 'longitude']]

# drop un-popular locations
filterd_kl = kl[kl['location id'].isin(popular_locations_dict)]

# create location id - coordinated  dictionary
```

```
kmeans_locations = dict()
for row in filterd_kl.iterrows():
    kmeans_locations[row[1][0]] = [row[1][1], row[1][2]]
```

In [43]:
```
filterd_kl
```

Out[43]:

	location id	latitude	longitude
1	7a0f88982aa015062b95e3b4843f9ca2	39.891383	-105.070814
2	dd7cd3d264c2d063832db506fba8bf79	39.891077	-105.068532
3	9848afcc62e500a01cf6fbf24b797732f8963683	39.750469	-104.999073
4	2ef143e12038c870038df53e0478cefc	39.752713	-104.996337
8	f6f52a75fd80e27e3770cd3a87054f27	39.827022	-105.143191
...	...	...	...
4743614	c65c3f4ee3b411dd98df003048c10834	52.073852	4.402297
4743663	1bc8ebb8dd7411dd95e2003048c0801e	51.980792	5.093807
4746696	c0a1807ca22411dd89796bd10ef17d6f	31.285000	120.677778
4746945	b2f89d8243c11de9226003048c0801e	51.480093	-0.171339
4747100	c7017a267a9c11dd99970030487eb504	57.749306	11.982254

48181 rows × 3 columns

In [44]:
```
# Kmeans
from sklearn.cluster import KMeans

k = 675
kmeans = KMeans(n_clusters=k)
kmeans.fit(list(kmeans_locations.values()))
labels = kmeans.labels_
centers = kmeans.cluster_centers_
```

In [45]:
```
unique_labels = np.unique(labels)

clusters = {label : [] for label in unique_labels}
for label, location in zip(labels, kmeans_locations.values()):
    clusters[label].append(location)
```

In [46]:
```
knn_network = nx.Graph()
knn_network.add_nodes_from(unique_labels)

for l in unique_labels:

```

```
    knn_network.nodes[l]['cor'] = centers[l]
    knn_network.nodes[l]['w'] = len(clusters[l])
```

In [47]:
```
max_size = 0
for node in knn_network.nodes:
    cur = knn_network.nodes[node]['w']
    if cur > max_size:
        max_size = cur
print(max_size)
```

```
1588
```

In [69]:
```
m = folium.Map([0,0], zoom_start=2)
```

In [70]:
```
luvain_communities[community_id].append(node)

colors = ['blue', 'red', 'green', 'orange', 'purple', 'yellow', 'cyan',
'magenta', 'lime', 'pink']



for i, c_id in enumerate(list(luvain_communities.keys())):
    for user in luvain_communities[c_id]:
        coors = CD_graph.nodes[user]['user_location']

        folium.CircleMarker(location=coors, radius=0.001,
color=colors[i],\
                            fill=True, fill_color='red').add_to(m)
    if i == 9:
        break
```

In [71]:
```
m.save('network_map.html')
```

In [72]:
```
display(m)
```

Make this Notebook Trusted to load map: File -> Trust Notebook

In [57]:

```
m = folium.Map([0,0], zoom_start=2)

# divide each community to based location clusters
for c, l in enumerate(luvain_communities.keys()):
    com = luvain_communities[l]

    com_locations = [CD_graph.nodes[user]['user_location'] for user in
com]

    k = 20

    if(len(com_locations) < k):
        c-=1
        l-=1
        continue


    kmeans = KMeans(n_clusters=k)
    kmeans.fit(com_locations)
    labels = kmeans.labels_
    centers = kmeans.cluster_centers_

#     print(len(labels))
#     print(len(centers))
#     break

    # add each user to his cluster
```

```
    unique_labels = np.unique(labels)

    com_clusters = {label : [] for label in unique_labels}
    for label, user in zip(labels, com):
        com_clusters[label].append(user)

    max_size = max( [len(com_clusters[label]) for label in
unique_labels]  )

    for i, label in enumerate(unique_labels):
        size = (len(com_clusters[label]) / max_size)*20

        folium.CircleMarker(location=centers[i], radius=size,
color="None", fill=True, fill_opacity=0.3,
fill_color=colors[c],).add_to(m)

    if c== 9:
        break

m.save('users_com_clusters.html')
display(m)
```

Make this Notebook Trusted to load map: File -> Trust Notebook

In [ ]:

divide users to cluster based on their average location

In [58]:
```
all_users_locations = []

for user in CD_nodes:
    all_users_locations.append(CD_graph.nodes[user]['user_location'])

k = 593
kmeans = KMeans(n_clusters=k)
kmeans.fit(all_users_locations)
labels = kmeans.labels_
centers = kmeans.cluster_centers_

# add each user to his cluster
unique_labels = np.unique(labels)

user_loc_clusters = {label : [] for label in unique_labels}
for label, user in zip(labels, CD_nodes):
    user_loc_clusters[label].append(user)
```

compare comunities vs Kmeans

In [59]:
```
from sklearn.metrics import confusion_matrix

users_labels_location = {user: -1 for user in CD_nodes}
for c in user_loc_clusters:
    for u in user_loc_clusters[c]:
        users_labels_location[u] = c

users_labels_friendships = {user: -1 for user in CD_nodes}

for l in luvain_communities.keys():
    for u in luvain_communities[l]:
        users_labels_friendships[u] = l


# friendships is ground truth
gt = list(users_labels_friendships.values())
predictions = list(users_labels_location.values())


contingency_table = confusion_matrix(gt, predictions)
```

```
majority_labels = contingency_table.argmax(axis=0)
purity = sum(contingency_table[majority_labels, majority_labels])
purity_1 = purity / len(predictions)

# locations is ground truth
gt = list(users_labels_location.values())
predictions = list(users_labels_friendships.values())



contingency_table = confusion_matrix(gt, predictions)
majority_labels = contingency_table.argmax(axis=0)
purity = sum(contingency_table[majority_labels, majority_labels])
purity_2 = purity/ len(predictions)

print(f"purity score: {(purity_1+purity_2)/2}")
```

```
purity score: 0.07778971024004136
```

the purity value of 0.074 indicates that the clusters generated by the K-means algorithm do not align well with the ground truth class labels

In [60]:

```
def get_community_locations(community):
    locations_dict = dict()
    for user in community:
        # get user unique locations
        filt = brightkite_totalCheckins['user'] == user
        user_locations = set(brightkite_totalCheckins['location id']
[filt])
        # add location to dict
        for loc in user_locations:
            if loc in locations_dict:
                locations_dict[loc] += 1
            else:
                locations_dict[loc] = 1
    return locations_dict
```

In [61]:

```
# sample 10 communities
sample_communities_ids = random.sample(community_ids, 10)



for sci in sample_communities_ids:

    cur_com= luvain_communities[sci]
```

```
    cur_com_locations = get_community_locations(cur_com)

    print(cur_com_locations)
    break
```

{'c9bdbfa61bad11deb207003048c10834': 1, 'ec43b9aee3411dd8894003048c10834': 1, '502
319ccf6fd11ddb103003048c10834': 1, '35014350f53e11dd86a8003048c0801e': 1, '4c0b5fb
8f53e11ddbbcb003048c0801e': 1, 'fb833a60e51711ddaa63003048c10834': 1, 'ed6d9bfea22
411ddb95fe7778d52da1f': 1, 'ed727b42a22411dd941befde87c23d9c': 1, '10a63e5ef53c11d
db8e8003048c0801e': 1, '6125aa04f84311dd9b01003048c10834': 1, '652ed26af53711ddb8e
8003048c0801e': 1, '65ee930ef55d11dd87d4003048c0801e': 1, '12bbbcd0a19111de81d4003
048c0801e': 1, '2a8bfeca18b11de9c41003048c0801e': 1, 'eee6bbe6a22411dd818fa39e0936
a5f6': 1, 'b9aeb556d7e2d9d98a94d72c9ee1f8f5f4d5080a': 2, '5abc4478aa0711dda5da0030
48c10834': 1, '3fb7252707811deae78003048c0801e': 1, 'ee8b1b88a22411dda44763941ed6e
aad': 1, '26fb7768c6ad97211777b67cfb039175cca46edf': 1, '53f4ad123d9e11dea37500304
8c0801e': 1, '65d0d916975d11de8bef003048c10834': 1, '7d1c9a4cda0c11ddae90003048c08
01e': 1, '865160887d7811de8321003048c0801e': 1, 'e022498ea57211dea416003048c0801
e': 1, '7631ffd8a39711de81d4003048c0801e': 1, '81282708de411deb60d003048c10834':
1, '6f75dc2a56fb11de87dc003048c10834': 1, 'd1140b907f9711de8fd3003048c0801e': 1,
'eee72e5aa22411ddb7f1bb1cd1bbdff6': 2, 'eee15af2a22411dda1ff57b30bcde9ad': 1, '22b
bdb96951863e24a07de1d45e9e372fec9dd07': 1, 'c06d8590eb8f37c76368f7b3767cd62677239c
7d': 1, '852a8d783d9111dea375003048c0801e': 1, 'b5c0396a300311dea60f003048c0801e':
1, 'eee86306a22411ddb34627726925f235': 2, 'f0659b0eb4bc731b422cc89a9d29b0d4d23f081
1': 1, 'ee8da380a22411ddacb64be8e10f7889': 1, 'ab91a3809c1d11dea3b4003048c10834':
1, 'eeeaeffea22411ddbfd6bf3c638cc10a': 2, '01fe6e339bb1ba4f71b9ebc7814b8080167bffe
a': 2, '7afe8dde95c211de8353003048c0801e': 1, 'eee65836a22411ddb166eb4ad3d6619c':
1, 'ee8f4c08a22411dda22cbf1f401c61a6': 1, 'eee3ab9aa22411dd8c512f637529ce81': 2,
'ee8fadeca22411ddba43df71e1364b99': 1, '9a97534c94f011deb1af003048c10834': 1, 'aeb
93962b4ae11dd87aa003048c10834': 1, 'f19fabfc48626b89786f34aa2b899a68756ac77a': 1,
'f84d895fbaf0891297cd38a5a42c91acfe6bfb34': 1, 'ee8e2bf2a22411dd8e904b7b9bf031fb':
1, '9b65a36cc7e07cf920bd7f71f44f73f0e8f9cff0': 1, 'f276f8bccfe111ddae44003048c0801
e': 1, 'ee8f00eaa22411dd8872bf9bf7b0ad34': 1, 'bcadf326950b11de8bef003048c10834':
1, '08470ff7dd3c562e825a88f7e70b991116ec5852': 1, '403b3d716bda65e2e0e99a08a158ec5
793f968ed': 1, '2e846c7e815111de9a04003048c10834': 1, 'ee8b3e74a22411ddb861079e640
ce098': 1, 'cec8fe7e86c0aa6623daa6ea8a721f301cc19a82': 1, 'eee4d074a22411ddae4bbb1
e21968850': 2, '29dc9520c65811dd8946003048c0801e': 1, 'ee8c106aa22411dd92d7bb6f148
5f74d': 1, 'eeb1d52aa22411dda762efa6cc34e945': 2, 'ee8d1c44a22411dd86f7271ec44d348
b': 1, 'ee8ebd60a22411ddbb532f697233f2fb': 1, '49fe9c66d3d5469dc3c153c279683e45bde
3dd73': 1, 'e50b37c9f5b11de81d4003048c0801e': 1, 'ee89b932a22411dd9a4cdb67f86ed02
1': 1, '8033c508dba511dd86ac003048c10834': 1, 'ee8db636a22411ddb435535d4622fd25':
1, 'eee3b478a22411ddb6960b7f13fe5d9b': 2, 'eee65174a22411dd83e16724c727187e': 1,
'eee6b1b4a22411dd9780ff005eaa720e': 1, '8a32396a437211de9bcd003048c10834': 1, 'ee8
ebbf8a22411dd8b9187f9e8864b3b': 1, 'eee94604a22411dd8a4d8380b9427db4': 2, '478cd61
8301411de9120003048c10834': 1, 'ee8bfeeaa22411dd9bb17b6a7da216bc': 1, 'f08ac195a28
85f84edd3a75c4aa56a312b9d184c': 1, '278aad6c728711de8ea5003048c0801e': 1, '65c5a95
815ce11dead3f003048c0801e': 1, 'eee9124ca22411dd93d7af04d530ed0b': 2, 'c1be731a864
011deb7ac003048c10834': 1, 'ee8e18b0a22411dd83933be732149b04': 1, '8ec169109c0f11d
eba66003048c0801e': 1, 'eeb32862a22411ddb80b239fdc065379': 1, '5ead7586a07d11ddb00
5003048c10834': 1, 'e2894b6ae64f11dda8cf003048c0801e': 1, 'ce8a9664f09511dd8f9e003
048c0801e': 1, '8d943afae58411dda75f003048c10834': 1, 'ef4eabf2a22411dd9aae6b92cce
fd77c': 1, 'eea862a6a22411ddbe72fbece05437c2': 1, 'eeff5e08a22411dda0c373750b46bad
6': 1, 'eeaa6cb8a22411dda2d5279cc453a89a': 1, 'ee8b1d0ea22411ddb074dbd65f1665cf':
1, 'eee9ba30a22411dd9a5cab1422c82a7d': 1, 'eee73b3ea22411dd903dc316899fdbf7': 1,
'3c26dffc8a2e8804dfe2c8a1195cfaa5ef6d0014': 1, 'eeeb5b60a22411dd954f079825aaed8b':
1, 'eee9500ea22411dd9f8cf34104e39240': 1, 'ee04ad6ea22411dd936b5be9ea306840': 1,
'eeb33852a22411ddbb4ceb623c91c680': 1, 'eee750baa22411dd984de747441b9f41': 1, 'eee
67564a22411ddbfd6539d6084c529': 1, 'eea8e2a8a22411ddbdcb436601530fb6': 1, 'eee54ff
ea22411dd89d9c7174084a076': 1, 'eee5b368a22411dd822c2f74bce10b7c': 1, 'eee53820a22
411ddab67632b4044673c': 1, 'eea933aca22411dd87b1c77564a379bc': 1, 'eea890dca22411d
da701174ac7bdc282': 1, 'eee3bbdaa22411dda6a60ba13ad65f81': 1, 'eee45478a22411ddaa4
edb301cb5010a': 1, 'ee9db086a22411dd906423e5f562c73d': 1, 'eeaa267ca22411dd803ca39

```
e7e1bd944': 1, 'a186a7fd26e0fa8ecde1a80bf24899e476dc2b65': 1, 'eeb1c6b6a22411dd894
c537d8ce2a025': 1}
```

In [ ]:

In [62]:

```
colors = [
    "AliceBlue",
    "AntiqueWhite",
    "Aqua",
    "Aquamarine",
    "Azure",
    "Beige",
    "Bisque",
    "Black",
    "BlanchedAlmond",
    "Blue",
    "BlueViolet",
    "Brown",
    "BurlyWood",
    "CadetBlue",
    "Chartreuse",
    "Chocolate",
    "Coral",
    "CornflowerBlue",
    "Cornsilk",
    "Crimson",
    "Cyan",
    "DarkBlue",
    "DarkCyan",
    "DarkGoldenRod",
    "DarkGray",
    "DarkGreen",
    "DarkKhaki",
    "DarkMagenta",
    "DarkOliveGreen",
    "DarkOrange",
    "DarkOrchid",
    "DarkRed",
    "DarkSalmon",
    "DarkSeaGreen",
    "DarkSlateBlue",
    "DarkSlateGray",
```

```
    "DarkTurquoise",
    "DarkViolet",
    "DeepPink",
    "DeepSkyBlue",
    "DimGray",
    "DodgerBlue",
    "FireBrick",
    "FloralWhite",
    "ForestGreen",
    "Fuchsia",
    "Gainsboro",
    "GhostWhite",
    "Gold",
    "GoldenRod",
    "Gray",
    "Green",
    "GreenYellow",
]
```

In [63]:
```
len(colors)
```

Out[63]:
```
53
```

In [64]:
```
m = folium.Map([0,0], zoom_start=2)

# divide each community to based location clusters
for c, l in enumerate(luvain_communities.keys()):
    com = luvain_communities[l]
    if len(com) < 50:
        c -= 1
        continue

    com_locations = [CD_graph.nodes[user]['user_location'] for user in
com]

    k = 20
    kmeans = KMeans(n_clusters=k)
    kmeans.fit(com_locations)
    labels = kmeans.labels_
    centers = kmeans.cluster_centers_

    # add each user to his cluster
```

```
    unique_labels = np.unique(labels)

    com_clusters = {label : [] for label in unique_labels}
    for label, user in zip(labels, com):
        com_clusters[label].append(user)

    max_size = max( [len(com_clusters[label]) for label in
unique_labels]  )

#     print(c)
    for i, label in enumerate(unique_labels):
        size = (len(com_clusters[label]) / max_size)*20

        folium.CircleMarker(location=centers[i], radius=size,
color="None", fill=True, fill_opacity=0.3,
fill_color=colors[c]).add_to(m)

    if c== 50:
        break

m.save('users_com_clusters_50.html')
display(m)
```

Make this Notebook Trusted to load map: File -> Trust Notebook

2 - Bipartite Graph for each timestemps

"left" side - locations, 'right' side - users. edge represent check in of user in a location

In [65]:
```
# create network*************
G2 = nx.Graph()

G2.add_nodes_from(locations)
G2.add_nodes_from(users)
for ci in brightkite_totalCheckins.values:
    user, location = ci[0], ci[-1]
    if(G2.has_edge(location, user)):
        G2[user][location]['weight'] += 1
    else:
        G2.add_edge(location, user)
        G2[user][location]['weight'] = 1
```

Network Properties

In [66]:
```
print(f"number of 'left side' nodes: {number_of_locations}")
print(f"number of 'right side' nodes: {number_of_users}")
print(f"total number of nodes: {number_of_locations +
number_of_users}")
print(f"total number of edges: {G2.number_of_edges()}")
```

```
number of 'left side' nodes: 772966
number of 'right side' nodes: 58228
total number of nodes: 831194
total number of edges: 1076169
```

In [67]:
```
# users average degree
avg_users_degree = sum([G2.degree[user] for user in users]) /
number_of_users
print(f"average number of places check-in by a user (user averege
degree): {avg_users_degree}")

# location average degree
avg_locations_degree = sum([G2.degree[location] for location in
locations]) / number_of_locations
print(f"average number of people check-in in a location (averege
location degree): {avg_locations_degree}")
```

```
average number of places check-in by a user (user averege degree): 18.481984612214
056
average number of people check-in in a location (averege location degree): 1.39225
91679323542
```

In [68]:
```
# number of location with no check-ins
th = 2
```

```
no_checkin_location = sum([1 if G2.degree[location] <= th else 0 for
location in locations])
perc = ( (no_checkin_location) / (number_of_locations) ) * 100
print(f"{no_checkin_location:,} out of {number_of_locations:,}
({perc:.1f}%) locations have {th} check-ins or less")
```

```
732,425 out of 772,966 (94.8%) locations have 2 check-ins or less
```

In [79]:

In [ ]: