```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Nov  6 16:05:24 2023

@author: halaszveronika
"""

import math
import numpy as np
from scipy.sparse import csr_matrix
from scipy.sparse.linalg import expm
import scipy as sp
import matplotlib.pyplot as plt
import networkx as nx
from statistics import mean
from operator import itemgetter
from netneurotools import metrics

#Original
def Original(G):
    G1 = G
#    node_closeness = nx.closeness_centrality(G, distance='weight', wf_improved=False)
    return (G1)
"""
def plot(G):
    edge_betweenness_dict = nx.edge_betweenness_centrality(G)
    sorted_edge_betweenness = sorted(edge_betweenness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist_1 = [i[0] for i in sorted_edge_betweenness[ :(len(G.edges) * 20 // 100)-1]]
    edgelist_2 = [i[0] for i in sorted_edge_betweenness[(len(G.edges) * 20 // 100) :(len(G.edges) * 40 // 100)-1]]
    edgelist_3 = [i[0] for i in sorted_edge_betweenness[(len(G.edges) * 40 // 100) :(len(G.edges) * 60 // 100)-1]]
    edgelist_4 = [i[0] for i in sorted_edge_betweenness[(len(G.edges) * 60 // 100) :(len(G.edges) * 80 // 100)-1]]
    edgelist_5 = [i[0] for i in sorted_edge_betweenness[(len(G.edges) * 80 // 100) :(len(G.edges))-1]]
    pos = nx.spring_layout(G, seed=7)  # positions for all nodes - seed for reproducibility
    # nodes
    nx.draw_networkx_nodes(G, pos, node_size=50)
    # edges
    nx.draw_networkx_edges(G, pos, edgelist=edgelist_1, width=0.6)
    nx.draw_networkx_edges(G, pos, edgelist=edgelist_2, width=0.5)
    nx.draw_networkx_edges(G, pos, edgelist=edgelist_3, width=0.4)
    nx.draw_networkx_edges(G, pos, edgelist=edgelist_4, width=0.3)
    nx.draw_networkx_edges(G, pos, edgelist=edgelist_5, width=0.2)
    ax = plt.gca()
    ax.margins(0.2)
    plt.axis("off")
    plt.tight_layout()
    plt.show()



   # elarge = [(u, v) for (u, v, d) in G.edges(data=True) if d["weight"] > 0.5]
   # esmall = [(u, v) for (u, v, d) in G.edges(data=True) if d["weight"] <= 0.5]
"""

# Removing 20 percent of the edges at once according to their weight
def top_20_edge_weight(G):
    G2 = nx.Graph(G)
    weight = nx.get_edge_attributes(G2,"probabilities")
    sorted_edge_weight = sorted(weight.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_weight[ :(G2.number_of_edges() * 20 // 100)-1]]
    G2.remove_edges_from(edgelist)
#    node_closeness = nx.closeness_centrality(G2, distance='weight', wf_improved=False)
    return (G2)

# Removing 30 percent of the edges at once according to their weight
def top_30_edge_weight(G):
    G3 = nx.Graph(G)
    weight = nx.get_edge_attributes(G3,"probabilities")
```

```
    sorted_edge_weight = sorted(weight.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_weight[ :(G3.number_of_edges() * 30 // 100)-1]]
    G3.remove_edges_from(edgelist)
#    node_closeness = nx.closeness_centrality(G3, distance='weight', wf_improved=False)
    return (G3)

# Removing 40 percent of the edges at once according to their weight
def top_40_edge_weight(G):
    G4 = nx.Graph(G)
    weight = nx.get_edge_attributes(G4,"probabilities")
    sorted_edge_weight = sorted(weight.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_weight[ :(len(G4.edges) * 40 // 100)-1]]
    G4.remove_edges_from(edgelist)
#    node_closeness = nx.closeness_centrality(G4, distance='weight', wf_improved=False)
    return (G4)

# Removing 50 percent of the edges at once according to their weight
def top_50_edge_weight(G):
    G21 = nx.Graph(G)
    weight = nx.get_edge_attributes(G21,"probabilities")
    sorted_edge_weight = sorted(weight.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_weight[ :(len(G21.edges) * 50 // 100)-1]]
    G21.remove_edges_from(edgelist)
#    node_closeness = nx.closeness_centrality(G2, distance='weight', wf_improved=False)
    return (G21)

# Removing 80 percent of the edges at once according to their weight
def top_80_edge_weight(G):
    G22 = nx.Graph(G)
    weight = nx.get_edge_attributes(G22,"probabilities")
    sorted_edge_weight = sorted(weight.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_weight[ :(len(G22.edges) * 80 // 100)-1]]
    G22.remove_edges_from(edgelist)
#    node_closeness = nx.closeness_centrality(G3, distance='weight', wf_improved=False)
    return (G22)

# Removing 90 percent of the edges at once according to their weight
def top_90_edge_weight(G):
    G23 = nx.Graph(G)
    weight = nx.get_edge_attributes(G23,"probabilities")
    sorted_edge_weight = sorted(weight.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_weight[ :(len(G23.edges) * 90 // 100)-1]]
    G23.remove_edges_from(edgelist)
#    node_closeness = nx.closeness_centrality(G4, distance='weight', wf_improved=False)
    return (G23)

# Removing 20 percent of the edges at once according to their betweenness centrality
def top_20_edge_betweenness_centrality(G):
    G5 = nx.Graph(G)
    edge_betweenness_dict = nx.edge_betweenness_centrality(G5, weight = 'weight')
    sorted_edge_betweenness = sorted(edge_betweenness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_betweenness[ :(len(G5.edges) * 20 // 100)-1]]
    G5.remove_edges_from(edgelist)
#    node_closeness = nx.closeness_centrality(G5, distance='weight', wf_improved=False)
    return (G5)

# Removing 20 percent of the edges at once according to their betweenness centrality without considering weights
def top_20_edge_betweenness_centrality_unweighted(G):
    G5u = nx.Graph(G)
    edge_betweenness_dict = nx.edge_betweenness_centrality(G5u)
    sorted_edge_betweenness = sorted(edge_betweenness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_betweenness[ :(len(G5u.edges) * 20 // 100)-1]]
    G5u.remove_edges_from(edgelist)
#    node_closeness = nx.closeness_centrality(G5, distance='weight', wf_improved=False)
    return (G5u)

# Removing 20 percent of the edges at once according to the closeness centrality of the endpoints
def top_20_edge_closeness_centrality(G):
```

```
    G6 = nx.Graph(G)
    edge_closeness_dict = {(u, v):(nx.closeness_centrality(G6, u = u, distance='weight', wf_improved=False) +
nx.closeness_centrality(G6, u = v, distance='weight', wf_improved=False)) for (u,v) in G6.edges()}
    sorted_edge_closeness = sorted(edge_closeness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_closeness[ :(len(G6.edges) * 20 // 100)-1]]
    G6.remove_edges_from(edgelist)
#    node_closeness = nx.closeness_centrality(G6, distance='weight', wf_improved=False)
    return (G6)

# Removing 20 percent of the edges at once according to the closeness centrality of the endpoints without considering weights
def top_20_edge_closeness_centrality_unweighted(G):
    G6u = nx.Graph(G)
    edge_closeness_dict = {(u, v):(nx.closeness_centrality(G6u, u = u, wf_improved=False) + nx.closeness_centrality(G6u, u = v,
wf_improved=False)) for (u,v) in G6u.edges()}
    sorted_edge_closeness = sorted(edge_closeness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_closeness[ :(len(G6u.edges) * 20 // 100)-1]]
    G6u.remove_edges_from(edgelist)
#    node_closeness = nx.closeness_centrality(G6, distance='weight', wf_improved=False)
    return (G6u)

# Removing 30 percent of the edges at once according to their betweenness centrality
def top_30_edge_betweenness_centrality(G):
    G7 = nx.Graph(G)
    edge_betweenness_dict = nx.edge_betweenness_centrality(G7, weight = 'weight')
    sorted_edge_betweenness = sorted(edge_betweenness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_betweenness[ :(len(G7.edges) * 30 // 100)-1]]
    G7.remove_edges_from(edgelist)
#    node_closeness = nx.closeness_centrality(G7, distance='weight', wf_improved=False)
    return (G7)

# Removing 30 percent of the edges at once according to their betweenness centrality without considering the weights
def top_30_edge_betweenness_centrality_unweighted(G):
    G7u = nx.Graph(G)
    edge_betweenness_dict = nx.edge_betweenness_centrality(G7u)
    sorted_edge_betweenness = sorted(edge_betweenness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_betweenness[ :(len(G7u.edges) * 30 // 100)-1]]
    G7u.remove_edges_from(edgelist)
#    node_closeness = nx.closeness_centrality(G7, distance='weight', wf_improved=False)
    return (G7u)

# Removing 30 percent of the edges at once according to the closeness centrality of the endpoints
def top_30_edge_closeness_centrality(G):
    G8 = nx.Graph(G)
    edge_closeness_dict = {(u, v):(nx.closeness_centrality(G8, u = u, distance='weight', wf_improved=False) +
nx.closeness_centrality(G8, u = v, distance='weight', wf_improved=False)) for (u,v) in G8.edges()}
    sorted_edge_closeness = sorted(edge_closeness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_closeness[ :(len(G8.edges) * 30 // 100)-1]]
    G8.remove_edges_from(edgelist)
#    node_closeness = nx.closeness_centrality(G8, distance='weight', wf_improved=False)
    return (G8)

# Removing 30 percent of the edges at once according to the closeness centrality of the endpoints without considering the
weights
def top_30_edge_closeness_centrality_unweighted(G):
    G8u = nx.Graph(G)
    edge_closeness_dict = {(u, v):(nx.closeness_centrality(G8u, u = u, wf_improved=False) + nx.closeness_centrality(G8u, u = v,
wf_improved=False)) for (u,v) in G8u.edges()}
    sorted_edge_closeness = sorted(edge_closeness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_closeness[ :(len(G8u.edges) * 30 // 100)-1]]
    G8u.remove_edges_from(edgelist)
#    node_closeness = nx.closeness_centrality(G8, distance='weight', wf_improved=False)
    return (G8u)

#Removing 40 percent of the edges at once according to their betweenness centrality
def top_40_edge_betweenness_centrality(G):
    G9 = nx.Graph(G)
    edge_betweenness_dict = nx.edge_betweenness_centrality(G9, weight='weight')
    sorted_edge_betweenness = sorted(edge_betweenness_dict.items(), key = itemgetter(1), reverse = True)
```

```
    edgelist = [i[0] for i in sorted_edge_betweenness[ :(len(G9.edges) * 40 // 100)-1]]
    G9.remove_edges_from(edgelist)
#    node_closeness = nx.closeness_centrality(G9, distance='weight', wf_improved=False)
    return (G9)

#Removing 40 percent of the edges at once according to their betweenness centrality without considering the weights
def top_40_edge_betweenness_centrality_unweighted(G):
    G9u = nx.Graph(G)
    edge_betweenness_dict = nx.edge_betweenness_centrality(G9u)
    sorted_edge_betweenness = sorted(edge_betweenness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_betweenness[ :(len(G9u.edges) * 40 // 100)-1]]
    G9u.remove_edges_from(edgelist)
#    node_closeness = nx.closeness_centrality(G9, distance='weight', wf_improved=False)
    return (G9u)

# Removing 40 percent of the edges at once according to the closeness centrality of the endpoints
def top_40_edge_closeness_centrality(G):
    G10 = nx.Graph(G)
    edge_closeness_dict = {(u, v):(nx.closeness_centrality(G10, u = u, distance='weight', wf_improved=False) +
nx.closeness_centrality(G10, u = v, distance='weight', wf_improved=False)) for (u,v) in G10.edges()}
    sorted_edge_closeness = sorted(edge_closeness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_closeness[ :(len(G10.edges) * 40 // 100)-1]]
    G10.remove_edges_from(edgelist)
#    node_closeness = nx.closeness_centrality(G10, distance='weight', wf_improved=False)
    return (G10)

# Removing 40 percent of the edges at once according to the closeness centrality of the endpoints without considering the
weights
def top_40_edge_closeness_centrality_unweighted(G):
    G10u = nx.Graph(G)
    edge_closeness_dict = {(u, v):(nx.closeness_centrality(G10u, u = u, wf_improved=False) + nx.closeness_centrality(G10u, u =
v, wf_improved=False)) for (u,v) in G10u.edges()}
    sorted_edge_closeness = sorted(edge_closeness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_closeness[ :(len(G10u.edges) * 40 // 100)-1]]
    G10u.remove_edges_from(edgelist)
#    node_closeness = nx.closeness_centrality(G10, distance='weight', wf_improved=False)
    return (G10u)

#Removing 40 percent of the edges in two rounds according to their betweenness centrality
def top_20_20_edge_betweenness_centrality(G):
    G11 = nx.Graph(G)
    length = len(G11.edges)
    edge_betweenness_dict = nx.edge_betweenness_centrality(G11, weight='weight')
    sorted_edge_betweenness = sorted(edge_betweenness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_betweenness[ :(length * 20 // 100)-1]]
    G11.remove_edges_from(edgelist)
    edge_betweenness_dict = nx.edge_betweenness_centrality(G11, weight='weight')
    sorted_edge_betweenness = sorted(edge_betweenness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_betweenness[ :(length * 20 // 100)-1]]
    G11.remove_edges_from(edgelist)
#    node_closeness = nx.closeness_centrality(G11, distance='weight', wf_improved=False)
    return (G11)

#Removing 40 percent of the edges in two rounds according to their betweenness centrality without considering the weights
def top_20_20_edge_betweenness_centrality_unweighted(G):
    G11u = nx.Graph(G)
    length = len(G11u.edges)
    edge_betweenness_dict = nx.edge_betweenness_centrality(G11u)
    sorted_edge_betweenness = sorted(edge_betweenness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_betweenness[ :(length * 20 // 100)-1]]
    G11u.remove_edges_from(edgelist)
    edge_betweenness_dict = nx.edge_betweenness_centrality(G11u)
    sorted_edge_betweenness = sorted(edge_betweenness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_betweenness[ :(length * 20 // 100)-1]]
    G11u.remove_edges_from(edgelist)
#    node_closeness = nx.closeness_centrality(G11, distance='weight', wf_improved=False)
    return (G11u)
```

```
#Removing 40 percent of the edges in two rounds according to the closeness centrality of the endpoints
def top_20_20_edge_closeness_centrality(G):
    G12 = nx.Graph(G)
    length = len(G12.edges)
    edge_closeness_dict = {(u, v):(nx.closeness_centrality(G12, u = u, distance='weight', wf_improved=False) + nx.closeness_centrality(G12, u = v, distance='weight', wf_improved=False)) for (u,v) in G12.edges()}
    sorted_edge_closeness = sorted(edge_closeness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_closeness[ :(length * 20 // 100)-1]]
    G12.remove_edges_from(edgelist)
    edge_closeness_dict = {(u, v):(nx.closeness_centrality(G12, u = u, distance='weight', wf_improved=False) + nx.closeness_centrality(G12, u = v, distance='weight', wf_improved=False)) for (u,v) in G12.edges()}
    sorted_edge_closeness = sorted(edge_closeness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_closeness[ :(length * 20 // 100)-1]]
    G12.remove_edges_from(edgelist)
#    node_closeness = nx.closeness_centrality(G12, distance='weight', wf_improved=False)
    return (G12)

#Removing 40 percent of the edges in two rounds according to the closeness centrality of the endpoints without considering the weights
def top_20_20_edge_closeness_centrality_unweighted(G):
    G12u = nx.Graph(G)
    length = len(G12u.edges)
    edge_closeness_dict = {(u, v):(nx.closeness_centrality(G12u, u = u, wf_improved=False) + nx.closeness_centrality(G12u, u = v, wf_improved=False)) for (u,v) in G12u.edges()}
    sorted_edge_closeness = sorted(edge_closeness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_closeness[ :(length * 20 // 100)-1]]
    G12u.remove_edges_from(edgelist)
    edge_closeness_dict = {(u, v):(nx.closeness_centrality(G12u, u = u, wf_improved=False) + nx.closeness_centrality(G12u, u = v, wf_improved=False)) for (u,v) in G12u.edges()}
    sorted_edge_closeness = sorted(edge_closeness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_closeness[ :(length * 20 // 100)-1]]
    G12u.remove_edges_from(edgelist)
#    node_closeness = nx.closeness_centrality(G12, distance='weight', wf_improved=False)
    return (G12u)

#Removing 40 percent of the edges in four rounds according to their betweenness centrality
def top_10_10_10_10_edge_betweenness_centrality(G):
    G13 = nx.Graph(G)
    length = len(G13.edges)
    edge_betweenness_dict = nx.edge_betweenness_centrality(G13, weight='weight')
    sorted_edge_betweenness = sorted(edge_betweenness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_betweenness[ :(length * 10 // 100)-1]]
    G13.remove_edges_from(edgelist)
    edge_betweenness_dict = nx.edge_betweenness_centrality(G13, weight='weight')
    sorted_edge_betweenness = sorted(edge_betweenness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_betweenness[ :(length * 10 // 100)-1]]
    G13.remove_edges_from(edgelist)
    edge_betweenness_dict = nx.edge_betweenness_centrality(G13, weight='weight')
    sorted_edge_betweenness = sorted(edge_betweenness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_betweenness[ :(length * 10 // 100)-1]]
    G13.remove_edges_from(edgelist)
    edge_betweenness_dict = nx.edge_betweenness_centrality(G13, weight='weight')
    sorted_edge_betweenness = sorted(edge_betweenness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_betweenness[ :(length * 10 // 100)-1]]
    G13.remove_edges_from(edgelist)
#    node_closeness = nx.closeness_centrality(G13, distance='weight', wf_improved=False)
    return (G13)

#Removing 40 percent of the edges in four rounds according to their betweenness centrality without considering the weights
def top_10_10_10_10_edge_betweenness_centrality_unweighted(G):
    G13u = nx.Graph(G)
    length = len(G13u.edges)
    edge_betweenness_dict = nx.edge_betweenness_centrality(G13u)
    sorted_edge_betweenness = sorted(edge_betweenness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_betweenness[ :(length * 10 // 100)-1]]
    G13u.remove_edges_from(edgelist)
    edge_betweenness_dict = nx.edge_betweenness_centrality(G13u)
    sorted_edge_betweenness = sorted(edge_betweenness_dict.items(), key = itemgetter(1), reverse = True)
```

```
    edgelist = [i[0] for i in sorted_edge_betweenness[ :(length * 10 // 100)-1]]
    G13u.remove_edges_from(edgelist)
    edge_betweenness_dict = nx.edge_betweenness_centrality(G13u)
    sorted_edge_betweenness = sorted(edge_betweenness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_betweenness[ :(length * 10 // 100)-1]]
    G13u.remove_edges_from(edgelist)
    edge_betweenness_dict = nx.edge_betweenness_centrality(G13u)
    sorted_edge_betweenness = sorted(edge_betweenness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_betweenness[ :(length * 10 // 100)-1]]
    G13u.remove_edges_from(edgelist)
#    node_closeness = nx.closeness_centrality(G13, distance='weight', wf_improved=False)
    return (G13u)

#Removing 40 percent of the edges in four rounds according to the closeness centrality of the endpoints
def top_10_10_10_10_edge_closeness_centrality(G):
    G14 = nx.Graph(G)
    length = len(G14.edges)
    edge_closeness_dict = {(u, v):(nx.closeness_centrality(G14, u = u, distance='weight', wf_improved=False) + 
nx.closeness_centrality(G14, u = v, distance='weight', wf_improved=False)) for (u,v) in G14.edges()}
    sorted_edge_closeness = sorted(edge_closeness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_closeness[ :(length * 10 // 100)-1]]
    G14.remove_edges_from(edgelist)
    edge_closeness_dict = {(u, v):(nx.closeness_centrality(G14, u = u, distance='weight', wf_improved=False) + 
nx.closeness_centrality(G14, u = v, distance='weight', wf_improved=False)) for (u,v) in G14.edges()}
    sorted_edge_closeness = sorted(edge_closeness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_closeness[ :(length * 10 // 100)-1]]
    G14.remove_edges_from(edgelist)
    edge_closeness_dict = {(u, v):(nx.closeness_centrality(G14, u = u, distance='weight', wf_improved=False) + 
nx.closeness_centrality(G14, u = v, distance='weight', wf_improved=False)) for (u,v) in G14.edges()}
    sorted_edge_closeness = sorted(edge_closeness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_closeness[ :(length * 10 // 100)-1]]
    G14.remove_edges_from(edgelist)
    edge_closeness_dict = {(u, v):(nx.closeness_centrality(G14, u = u, distance='weight', wf_improved=False) + 
nx.closeness_centrality(G14, u = v, distance='weight', wf_improved=False)) for (u,v) in G14.edges()}
    sorted_edge_closeness = sorted(edge_closeness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_closeness[ :(length * 10 // 100)-1]]
    G14.remove_edges_from(edgelist)
#    node_closeness = nx.closeness_centrality(G14, distance='weight', wf_improved=False)
    return (G14)

#Removing 40 percent of the edges in four rounds according to the closeness centrality of the endpoints without considering the 
weights
def top_10_10_10_10_edge_closeness_centrality_unweighted(G):
    G14u = nx.Graph(G)
    length = len(G14u.edges)
    edge_closeness_dict = {(u, v):(nx.closeness_centrality(G14u, u = u, wf_improved=False) + nx.closeness_centrality(G14u, u = 
v, wf_improved=False)) for (u,v) in G14u.edges()}
    sorted_edge_closeness = sorted(edge_closeness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_closeness[ :(length * 10 // 100)-1]]
    G14u.remove_edges_from(edgelist)
    edge_closeness_dict = {(u, v):(nx.closeness_centrality(G14u, u = u, wf_improved=False) + nx.closeness_centrality(G14u, u = 
v, wf_improved=False)) for (u,v) in G14u.edges()}
    sorted_edge_closeness = sorted(edge_closeness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_closeness[ :(length * 10 // 100)-1]]
    G14u.remove_edges_from(edgelist)
    edge_closeness_dict = {(u, v):(nx.closeness_centrality(G14u, u = u, wf_improved=False) + nx.closeness_centrality(G14u, u = 
v, wf_improved=False)) for (u,v) in G14u.edges()}
    sorted_edge_closeness = sorted(edge_closeness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_closeness[ :(length * 10 // 100)-1]]
    G14u.remove_edges_from(edgelist)
    edge_closeness_dict = {(u, v):(nx.closeness_centrality(G14u, u = u, wf_improved=False) + nx.closeness_centrality(G14u, u = 
v, wf_improved=False)) for (u,v) in G14u.edges()}
    sorted_edge_closeness = sorted(edge_closeness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_closeness[ :(length * 10 // 100)-1]]
    G14u.remove_edges_from(edgelist)
#    node_closeness = nx.closeness_centrality(G14, distance='weight', wf_improved=False)
    return (G14u)
```

```
#Removing 50 percent of the edges at once according to their betweenness centrality
def top_50_edge_betweenness_centrality(G):
    G15 = nx.Graph(G)
    edge_betweenness_dict = nx.edge_betweenness_centrality(G15, weight='weight')
    sorted_edge_betweenness = sorted(edge_betweenness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_betweenness[ :(len(G15.edges) * 50 // 100)-1]]
    G15.remove_edges_from(edgelist)
#    node_closeness = nx.closeness_centrality(G9, distance='weight', wf_improved=False)
    return (G15)

#Removing 50 percent of the edges at once according to their betweenness centrality without considering the weights
def top_50_edge_betweenness_centrality_unweighted(G):
    G15u = nx.Graph(G)
    edge_betweenness_dict = nx.edge_betweenness_centrality(G15u)
    sorted_edge_betweenness = sorted(edge_betweenness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_betweenness[ :(len(G15u.edges) * 50 // 100)-1]]
    G15u.remove_edges_from(edgelist)
#    node_closeness = nx.closeness_centrality(G9, distance='weight', wf_improved=False)
    return (G15u)

# Removing 50 percent of the edges at once according to the closeness centrality of the endpoints
def top_50_edge_closeness_centrality(G):
    G16 = nx.Graph(G)
    edge_closeness_dict = {(u, v):(nx.closeness_centrality(G16, u = u, distance='weight', wf_improved=False) +
nx.closeness_centrality(G16, u = v, distance='weight', wf_improved=False)) for (u,v) in G16.edges()}
    sorted_edge_closeness = sorted(edge_closeness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_closeness[ :(len(G16.edges) * 50 // 100)-1]]
    G16.remove_edges_from(edgelist)
#    node_closeness = nx.closeness_centrality(G10, distance='weight', wf_improved=False)
    return (G16)

# Removing 50 percent of the edges at once according to the closeness centrality of the endpoints without considering the
weights
def top_50_edge_closeness_centrality_unweighted(G):
    G16u = nx.Graph(G)
    edge_closeness_dict = {(u, v):(nx.closeness_centrality(G16u, u = u, wf_improved=False) + nx.closeness_centrality(G16u, u =
v, wf_improved=False)) for (u,v) in G16u.edges()}
    sorted_edge_closeness = sorted(edge_closeness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_closeness[ :(len(G16u.edges) * 50 // 100)-1]]
    G16u.remove_edges_from(edgelist)
#    node_closeness = nx.closeness_centrality(G10, distance='weight', wf_improved=False)
    return (G16u)

#Removing 80 percent of the edges at once according to their betweenness centrality
def top_80_edge_betweenness_centrality(G):
    G17 = nx.Graph(G)
    edge_betweenness_dict = nx.edge_betweenness_centrality(G17, weight='weight')
    sorted_edge_betweenness = sorted(edge_betweenness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_betweenness[ :(len(G17.edges) * 80 // 100)-1]]
    G17.remove_edges_from(edgelist)
#    node_closeness = nx.closeness_centrality(G9, distance='weight', wf_improved=False)
    return (G17)

#Removing 80 percent of the edges at once according to their betweenness centrality withouit considering the weights
def top_80_edge_betweenness_centrality_unweighted(G):
    G17u = nx.Graph(G)
    edge_betweenness_dict = nx.edge_betweenness_centrality(G17u)
    sorted_edge_betweenness = sorted(edge_betweenness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_betweenness[ :(len(G17u.edges) * 80 // 100)-1]]
    G17u.remove_edges_from(edgelist)
#    node_closeness = nx.closeness_centrality(G9, distance='weight', wf_improved=False)
    return (G17u)

# Removing 80 percent of the edges at once according to the closeness centrality of the endpoints
def top_80_edge_closeness_centrality(G):
    G18 = nx.Graph(G)
    edge_closeness_dict = {(u, v):(nx.closeness_centrality(G18, u = u, distance='weight', wf_improved=False) +
nx.closeness_centrality(G18, u = v, distance='weight', wf_improved=False)) for (u,v) in G18.edges()}
```

```
    sorted_edge_closeness = sorted(edge_closeness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_closeness[ :(len(G18.edges) * 80 // 100)-1]]
    G18.remove_edges_from(edgelist)
#    node_closeness = nx.closeness_centrality(G10, distance='weight', wf_improved=False)
    return (G18)

# Removing 80 percent of the edges at once according to the closeness centrality of the endpoints without considering the
weights
def top_80_edge_closeness_centrality_unweighted(G):
    G18u = nx.Graph(G)
    edge_closeness_dict = {(u, v):(nx.closeness_centrality(G18u, u = u, wf_improved=False) + nx.closeness_centrality(G18u, u =
v, wf_improved=False)) for (u,v) in G18u.edges()}
    sorted_edge_closeness = sorted(edge_closeness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_closeness[ :(len(G18u.edges) * 80 // 100)-1]]
    G18u.remove_edges_from(edgelist)
#    node_closeness = nx.closeness_centrality(G10, distance='weight', wf_improved=False)
    return (G18u)

#Removing 90 percent of the edges at once according to their betweenness centrality
def top_90_edge_betweenness_centrality(G):
    G19 = nx.Graph(G)
    edge_betweenness_dict = nx.edge_betweenness_centrality(G19, weight='weight')
    sorted_edge_betweenness = sorted(edge_betweenness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_betweenness[ :(len(G19.edges) * 90 // 100)-1]]
    G19.remove_edges_from(edgelist)
#    node_closeness = nx.closeness_centrality(G9, distance='weight', wf_improved=False)
    return (G19)

#Removing 90 percent of the edges at once according to their betweenness centrality without considering the weigths
def top_90_edge_betweenness_centrality_unweighted(G):
    G19u = nx.Graph(G)
    edge_betweenness_dict = nx.edge_betweenness_centrality(G19u)
    sorted_edge_betweenness = sorted(edge_betweenness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_betweenness[ :(len(G19u.edges) * 90 // 100)-1]]
    G19u.remove_edges_from(edgelist)
#    node_closeness = nx.closeness_centrality(G9, distance='weight', wf_improved=False)
    return (G19u)

# Removing 90 percent of the edges at once according to the closeness centrality of the endpoints
def top_90_edge_closeness_centrality(G):
    G20 = nx.Graph(G)
    edge_closeness_dict = {(u, v):(nx.closeness_centrality(G20, u = u, distance='weight', wf_improved=False) +
nx.closeness_centrality(G20, u = v, distance='weight', wf_improved=False)) for (u,v) in G20.edges()}
    sorted_edge_closeness = sorted(edge_closeness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_closeness[ :(len(G20.edges) * 90 // 100)-1]]
    G20.remove_edges_from(edgelist)
#    node_closeness = nx.closeness_centrality(G10, distance='weight', wf_improved=False)
    return (G20)

# Removing 90 percent of the edges at once according to the closeness centrality of the endpoints without considering the
weights
def top_90_edge_closeness_centrality_unweighted(G):
    G20u = nx.Graph(G)
    edge_closeness_dict = {(u, v):(nx.closeness_centrality(G20u, u = u, wf_improved=False) + nx.closeness_centrality(G20u, u =
v, wf_improved=False)) for (u,v) in G20u.edges()}
    sorted_edge_closeness = sorted(edge_closeness_dict.items(), key = itemgetter(1), reverse = True)
    edgelist = [i[0] for i in sorted_edge_closeness[ :(len(G20u.edges) * 90 // 100)-1]]
    G20u.remove_edges_from(edgelist)
#    node_closeness = nx.closeness_centrality(G10, distance='weight', wf_improved=False)
    return (G20u)



def probabilities(G):
    probabilities = {}
    for u in G.nodes:
        for v in G[u]:
            if u < v:
```

```
            probabilities[(u,v)] = np.random.uniform(0.05, 0.95)
            probabilities[(v,u)] = probabilities[(u,v)]
    nx.set_edge_attributes(G, values=probabilities, name='probabilities')
    return(probabilities)

def weights_to_probabilities(G, probabilities):
    weight = {}
    for u in G.nodes:
        for v in G[u]:
            weight[(u,v)] = math.log((1 / probabilities[(u,v)]), math.e)
    nx.set_edge_attributes(G, values=weight, name='weight')

def probability_matrix(G, probabilities):
    W = []
    for u in G.nodes:
        M = []
        for v in G.nodes:
            if v in G[u]:
                p = probabilities[(u, v)]
            elif v == u:
                p = 0
            else:
                p = 0
            M.append(p)
        W.append(M)
    weights = np.array(W)
    return(weights)

def probability_matrix_simulation(G, probabilities):
    weights = csr_matrix((N, N))
    for u in G.nodes:
        for v in G.nodes:
            if v in G[u]:
                weights[(u, v)] = probabilities[(u, v)]
            elif v == u:
                weights[(u, v)] = 0
            else:
                weights[(u, v)] = 0
    return(weights)



def update(state, weights):
    """Iterative update of the network state. This is slow in python. If needed, it could be accelerated with dedicated C code or
JAX"""
    state_new = np.zeros_like(state)
    # iterate over nodes that are true
    for i,s in enumerate(state):
        if s:
            # iterate over connected nodes
            for j in weights[i].indices:
                if np.random.rand() < weights[i,j]:
                    state_new[j] = True
                weights[j,i] = weights[j,i] / 1.5
                    # for j in G[i]:
                  #    if np.random.rand() < weights[(i,j)]:
                  #        state_new[j] = True
    return state_new

def update_with_immunity(state, weights, immunity):
    state_new = np.zeros_like(state)
    minus_recovery_rate = 0.6
    for i,s in enumerate(state):
        if s == False:
            for j in weights[i].indices:
                if (np.random.rand() < (weights[(i,j)] / immunity[i])):
                    state_new[i] = True
                else:
```

```
                state_new[i] = False
        if s == True:
            if (np.random.rand() > (minus_recovery_rate / immunity[i])):
                immunity[i] = immunity[i] * 1.5
                for j in weights[i].indices:
                    if (np.random.rand() < (weights[(i,j)] / immunity[i])):
                        state_new[i] = True
                    else:
                        state_new[i] = False
            else:
                state_new[i] = True
    return state_new


def simulation(G, N):

    np.mean(np.array(weights.sum(axis=0)))
    plt.figure(figsize=(6,4))
    plt.imshow(weights.toarray(), cmap='gray_r')
    plt.colorbar(orientation='vertical', pad=0.2)
    plt.tight_layout()


    # create a state vector (bool)
    state = np.zeros(N, dtype=bool)
    # simulation that starts from a randomly active site
    state[np.random.randint(N)] = 1
    T = 100
    num_infected = np.zeros(T)
    num_infected[0] = state.sum()
    max_infected = 1
    immunity = []
    for i in range(N):
        immunity.append(1.0)
    trajectory = []
    for t in range(1, T):
        state = update_with_immunity(state, weights, immunity)
        num_infected[t] = state.sum()
        max_infected = max(max_infected, num_infected[t])
        trajectory.append(num_infected[t])
    print(trajectory)



    plt.figure(figsize=(6,4))
    plt.plot(num_infected)
    plt.xlabel('Time')
    plt.ylabel('Number of infected sites')
    plt.tight_layout()
    plt.show()



N = 300  # number of nodes
G = nx.barabasi_albert_graph(N, 50)   # specify the parameter
probabilities = probabilities(G)
weights_to_probabilities(G, probabilities)

G1 = Original(G)
G2 = top_20_edge_weight(G)
G3 = top_30_edge_weight(G)
G4 = top_40_edge_weight(G)
G21 = top_50_edge_weight(G)
G22 = top_80_edge_weight(G)
G23 = top_90_edge_weight(G)
G5 = top_20_edge_betweenness_centrality(G)
G6 = top_20_edge_closeness_centrality(G)
G7 = top_30_edge_betweenness_centrality(G)
```

```
G8 = top_30_edge_closeness_centrality(G)
G9 = top_40_edge_betweenness_centrality(G)
G10 = top_40_edge_closeness_centrality(G)
G11 = top_20_20_edge_betweenness_centrality(G)
G12 = top_20_20_edge_closeness_centrality(G)
G13 = top_10_10_10_10_edge_betweenness_centrality(G)
G14 = top_10_10_10_10_edge_closeness_centrality(G)
G15 = top_50_edge_betweenness_centrality(G)
G16 = top_50_edge_closeness_centrality(G)
G17 = top_80_edge_betweenness_centrality(G)
G18 = top_80_edge_closeness_centrality(G)
G19 = top_90_edge_betweenness_centrality(G)
G20 = top_90_edge_closeness_centrality(G)

G5u = top_20_edge_betweenness_centrality_unweighted(G)
G6u = top_20_edge_closeness_centrality_unweighted(G)
G7u = top_30_edge_betweenness_centrality_unweighted(G)
G8u = top_30_edge_closeness_centrality_unweighted(G)
G9u = top_40_edge_betweenness_centrality_unweighted(G)
G10u = top_40_edge_closeness_centrality_unweighted(G)
G11u = top_20_20_edge_betweenness_centrality_unweighted(G)
G12u = top_20_20_edge_closeness_centrality_unweighted(G)
G13u = top_10_10_10_10_edge_betweenness_centrality_unweighted(G)
G14u = top_10_10_10_10_edge_closeness_centrality_unweighted(G)
G15u = top_50_edge_betweenness_centrality_unweighted(G)
G16u = top_50_edge_closeness_centrality_unweighted(G)
G17u = top_80_edge_betweenness_centrality_unweighted(G)
G18u = top_80_edge_closeness_centrality_unweighted(G)
G19u = top_90_edge_betweenness_centrality_unweighted(G)
G20u = top_90_edge_closeness_centrality_unweighted(G)


print('G = nx.barabasi_albert_graph(300, 50)')   # according to the parameters



for graph in [G1, G2, G3, G4, G21, G22, G23, G5, G5u, G6, G6u, G7, G7u, G8, G8u, G9, G9u, G10, G10u,
#               G11, G11u, G12, G12u, G13, G13u, G14, G14u,
              G15, G15u, G16, G16u, G17, G17u, G18, G18u, G19, G19u, G20, G20u]:
    weights = probability_matrix_simulation(graph, probabilities)


    simulation(graph, N)
```