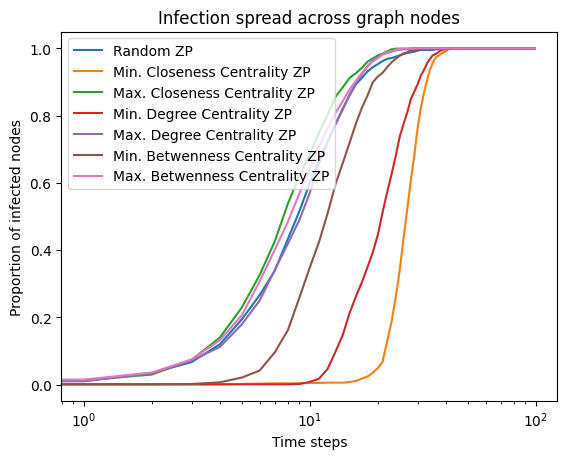
**CAI Lab 9**

Simulation of SI model of contagion over contact networks

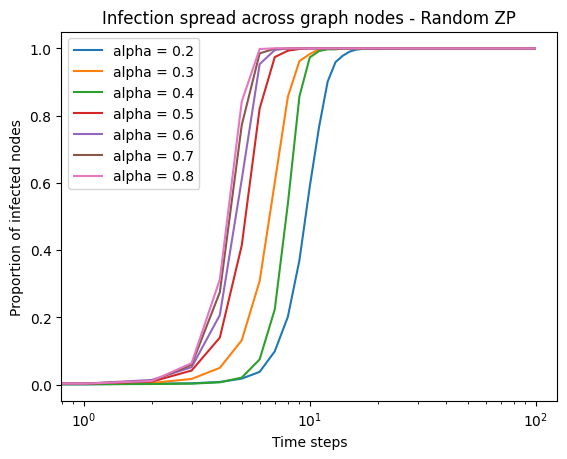
Gerard Comas & Laura García

As presented in class, the SI model classifies the nodes into two different groups: the susceptible class (S) and the infected class (I). The main goal of this lab session is to experiment with different network topologies and the spread of an “infection”. At the same time, we’ve decided to also observe the effect that different infection rates or changes in this parameter have on the evolution of the dynamical system. We’ll start by looking at each graph topology by itself, and we will then try to extract some general conclusions. We wrote the code in Python, using the *igraph* library. You can find the code in the section labeled *CODES* at the end.

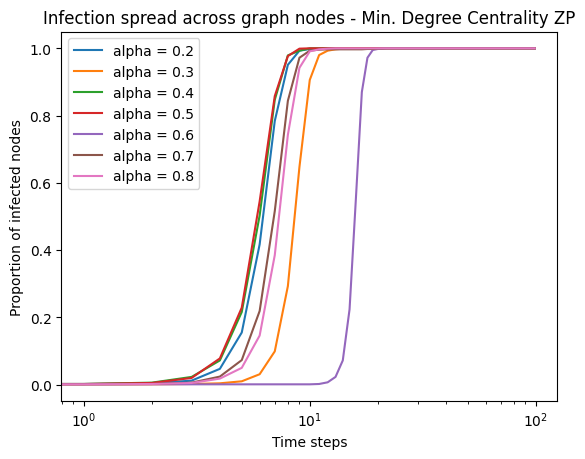
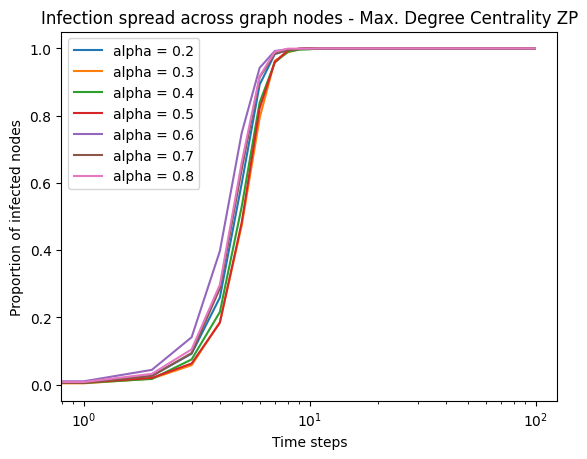
**Erdös-Rényi**

To see how the infection spreads through the network, we need a “zero patient” (z.p.) (the first infected node). The first question we asked ourselves was whether choosing it randomly or using some criteria (the three types of centrality we worked on during the lectures) has any effect on the spread of the infection. Since each node infects its nodes (those that are not yet infected) with a probability independently, we’ll expect for the disease to spread faster if we start at a highly central node. 

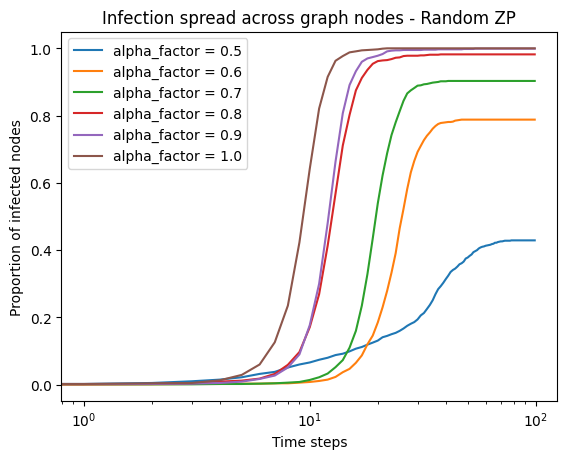
To test this hypothesis, we tried several z.p. choosing methods and plotting the proportion of infected nodes at each step. This test has been done with [[1]](#footnote-0), We can see that the shape of the curve is more or less the same for all the cases. However, when we choose the least central nodes it takes more time for the infection to start spreading. At the same time, when we start the infection at the most central node of the network, the spread happens way faster. Choosing the z.p. at random shows a behavior in the middle of the two previous cases: this is to be expected, for choosing the z.p. at random could mean we are choosing any of them (the least/most central or any other node in between).

Our next question is related to the infection rate . By the definition of , we’d expect that the higher the value of , the faster the infection will spread through the network. We confirmed this hypothesis: when choosing the z.p. at random, we do observe that infections with lower need more time for the network to be fully infected.

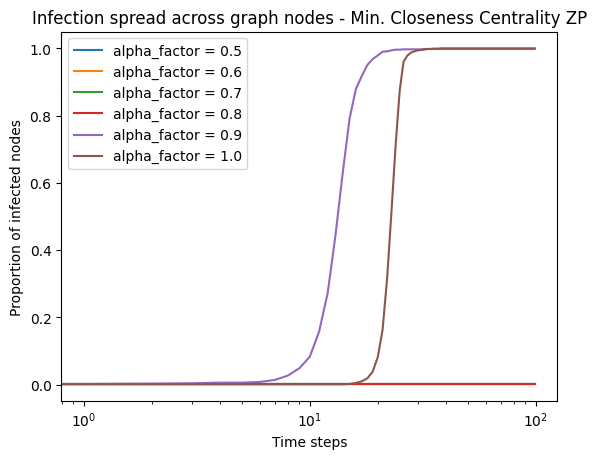
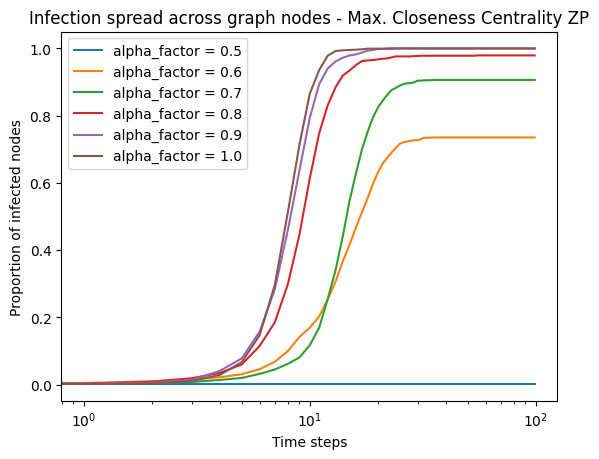
If we choose the z.p. with any of the other implemented methods, we observe some other behavior. When we set z.p. to the node with maximum centrality (for instance, degree centrality), we observe that does not seem as relevant, while when we choose for the minimum centrality node, it does make a difference. This is interesting because it basically means that, in an actual infectious disease, if a person is highly connected to the rest of the people in a community, it will not matter whether the virus is highly contagious or not: it will spread easily.



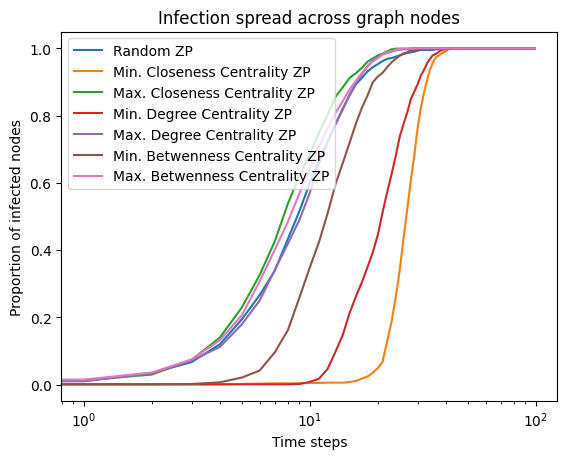
Finally, we saw that in all the previous cases the final proportion of infected nodes in the whole network is : every node gets infected. We wondered whether there was any way in which we could avoid that. We then thought of changing the infection rate throughout the spread process. Until now, the infection rate has been working as the probability that an infected node spreads the infection to its susceptible neighbors. We have assumed that, for each timestep, the infectious “capability” of a node decreases by some factor. Lower values of this factor should help reduce the infection spread through the network. In this case, for a more realistic scenario, we have used (still with ).

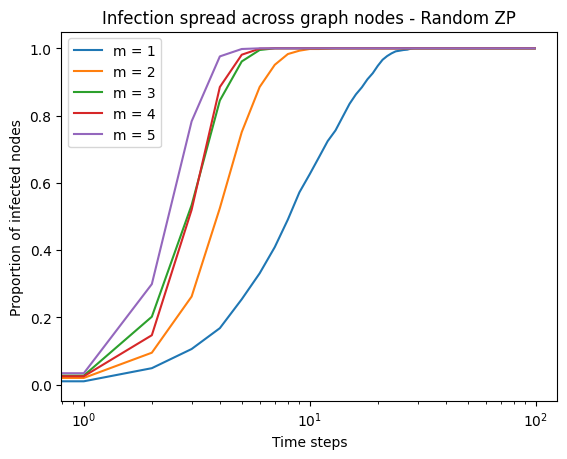
This is more or less what we saw in our experiments. Setting this factor should yield the same results seen before. When randomly choosing the z.p. we can see that decreasing does, in fact, help reduce the spread. This is because the probability of infecting other nodes decreases with each timestep. In real-life situations, this would be equivalent to saying that the infectious “power” of an infected person weakens as more time passes by (probably because this person will be medically treated and recover – though we’re not taking recovery into account in the SI model). 

At the same time, if we choose the z.p. by other methods (in this case, closeness centrality), we also see other results. When the z.p. is the most central node, we already see that the final proportion of infected nodes is only for values of near (which is something we expected). However, we get to stop the spread of the infection for smaller values of . We even see that for , the spread almost does not even start (probably due to the random component of the simulation, it is not infecting any neighboring node in the first steps and then its updated is too low to infect other nodes). When the z.p. is the least central node, we only observe a spread of the infection for high values of : if we quickly decrease the probability that a node infects and it is not really central to begin with, it is difficult for the infection to start spreading.



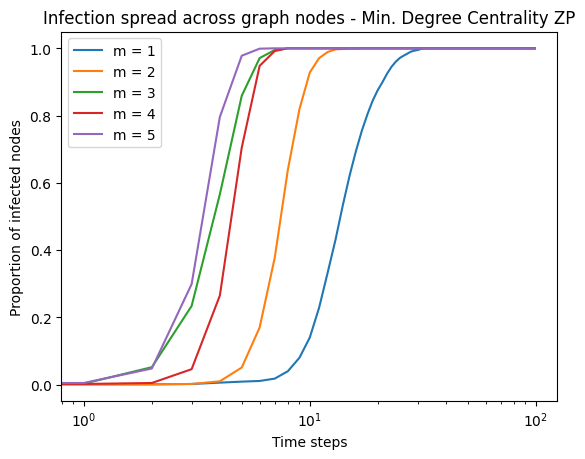
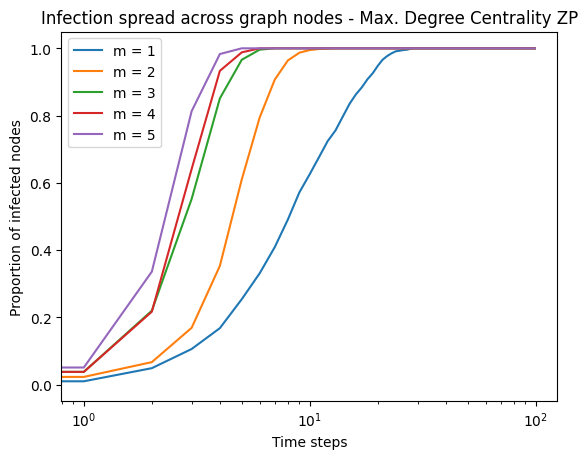
**Barabasi-Albert**

Again, our first question for this second kind of topology had to do with the method used to choose the z.p.. We know that the Barabasi-Albert (BA) graphs are built with preferential attachment, this is, it will result in graphs with very different degree centralities (there should be lots of nodes with low degree and a few with a considerably high degree). Therefore, we’d expect that choosing a more or less central node as the starting point of the infection (specially if we choose to go with degree centrality) should give different results.

For this first test, we have generated a BA graph with nodes and each node is added with edges. Just as before, we do see that when using the maximum centrality nodes as z.p. the infection spreads faster; and when using the minimum centrality nodes, we get slower spreads. This difference is especially noticeable at the start of the spread: for the minimum centrality, the contagious process needs way more time to “take off”; whereas for the maximum centrality the spread starts almost immediately. The random z.p. case almost overlaps with the maximum centrality cases (probably due to the randomness of the selection method). However, this has been done only for . A good next question to ask ourselves was to see whether the behavior is the same for all values of .

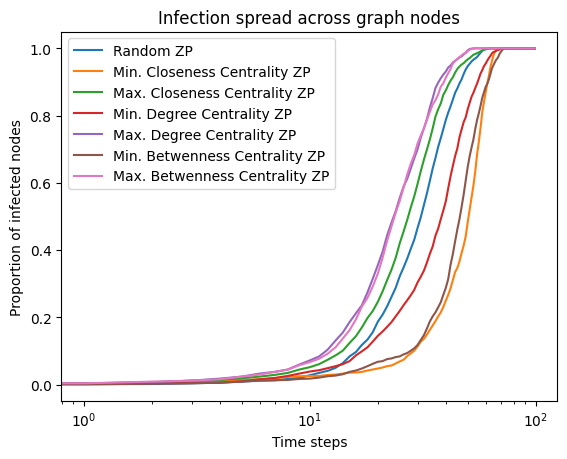
For a first trial, we wanted to see what would happen for the random z.p. method. We’ve seen that, for and (by default) the spread for is the slowest (as expected, for most of the nodes will only have 1 neighbor to infect and the process will evolve slowly); and the spreading happens faster the more we increase . Basically, when increases, we are asking for the minimum degree in the graph to be higher (any node will have at least edges). The higher the minimum degree, the higher the expected number of nodes that will be infected (even by the less central or less connected nodes). This, again, shows how important it is to isolate yourself when affected by a contagious disease.

If we do the same starting by minimum or maximum degree centrality (in this case, we’ve only focused on this type of centrality, given the main trait of BA graphs), we get other results. For maximum degree centrality, changing affects the slope of the infection curve and hence accelerates/decelerates the infectious process. For minimum degree centrality, we see that does not change the slope of the curve as much as it “shifts” the “take off” of the process.

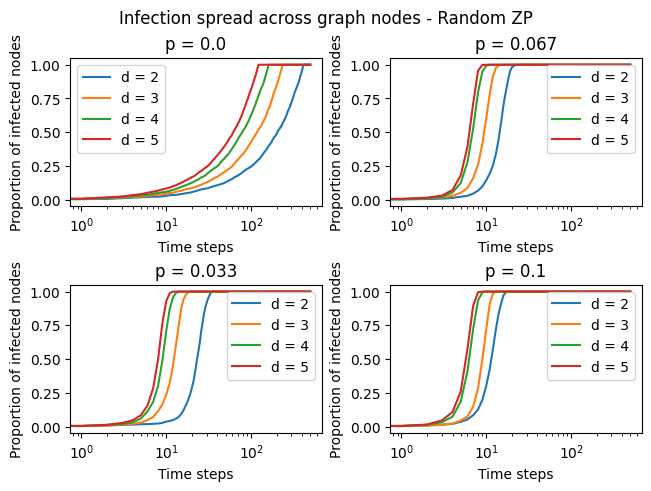


These results and observations make sense, if we think about the actual graph. For smaller values of there are fewer ways in which we can reach the nodes with low degree. This means that, if we start at the highest degree node, it will take the spread a while to reach such low degree nodes. On the other side, starting at those nodes, it will be difficult to find a node from which to really infect a higher number of other nodes.

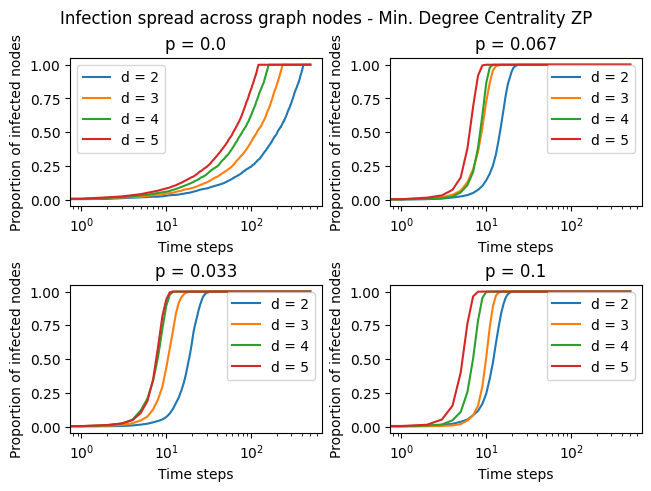
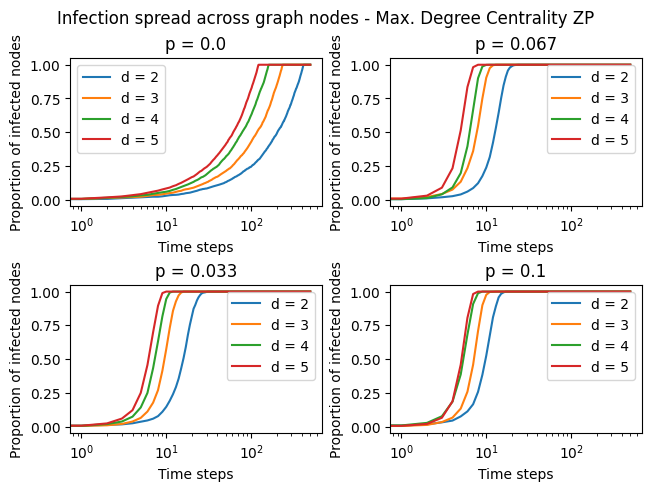
We can overall observe a very similar shape as for the ER topology. Therefore, we’d assume that the effect or would have on the network are pretty similar. We’ve been able to confirm this through our experiments[[2]](#footnote-1).

**Watts-Strogatz**

The last topology we’ll be using is the Watts-Strogatz (WS) model, which mainly aims to generate graphs with high clustering coefficients (ER graphs have low clustering coefficient due to the way in which edges are added). For a general WS graph with nodes, (connected neighbors on each side of each node in the initial lattice) and (chosen by the guidelines given during the lectures), we obtain the results shown to the right. As it happened for the BA model, the general shape of the infection process is pretty much the same as for ER networks. Choosing maximum centrality nodes as the z.p. also quickens the process, and choosing those of minimum centrality, slows the evolution of the infection (in terms of proportion of infected nodes). It must also be noted that changing and has the same effect explained in the ER section.

For this topology, though, we had another interesting question rise: how do the parameters and affect the infection spread. Given the way the WS graph is generated, directly affects the clustering coefficient of the graph (for will be the starting degree of each node and a node will have at least degree after possibly rewiring some edges): we would expect that higher clustering coefficient will help the infection spread faster. At the same time, is the probability that edges are rewired, which will directly affect, not only the final degree of each node, but also the diameter of the graph. We’d also expect for infections to spread faster in networks with low diameter.

To test these hypotheses, we run some more experiments. First, we chose a random z.p. We can see that the previously explained hypothesis seem to verify with these results: lower values of slow the infection down (in each timestep, a node will have a smaller number of expected nodes to infect). At the same time, though it is not as noticeable, the slope of the curve is more pronounced for greater values of (the spread of the infection is also faster). These results coincide with what we expected. We would expect the qualitative traits of these effects to remain the same when choosing the z.p. by centrality. The results we obtained for maximum and minimum degree centrality are shown next.



From these graphs we draw the same conclusions as for the ER and BA models: starting the infection at the least central node will delay its spread through the network and doing it at the most central node will make it spread sooner. We observe the same using the other centrality criteria presented in the lectures. If we compare this behavior with the random z.p., we see that there is no significant difference regarding the shape of the infection curve.

**CODES**

In the following section, we present the code implementations that were used to obtain the results discussed earlier.

Creates an Erdos-Renyi graph with the specified parameters.

def create\_ER\_graph(n, eps, a, b, random\_seed, plot=False):

"""

Parameters:

- n (int): The number of nodes in the graph.

- eps (float): The epsilon value for adjusting the probability range.

- a (float): The lower bound weight for the probability range.

- b (float): The upper bound weight for the probability range.

- random\_seed (int): The random seed for reproducibility.

- plot (bool): Whether to plot the graph or not. Default is False.

Returns:

- g (igraph.Graph): The created Erdos-Renyi graph.

"""

random.seed(random\_seed)

p\_lb, p\_ub = (1-eps)\*np.log(n)/n, (1+eps)\*np.log(n)/n

p = a\*p\_lb + b\*p\_ub

g = ig.Graph.Erdos\_Renyi(n, p)

g.vs['color'] = 'lightgrey'

g.vs['alpha'] = None

while 0 in g.degree():

g = ig.Graph.Erdos\_Renyi(n, p)

g.vs['color'] = 'lightgrey'

if plot:

fig, ax = plt.subplots()

ig.plot(

g,

target=ax,

vertex\_size=10

)

return g



Creates a Barabasi-Albert graph with the specified parameters.

def create\_BA\_graph(n, m, random\_seed, plot=False):

"""

Parameters:

- n (int): Number of nodes in the graph.

- m (int): Number of edges to attach from a new node to existing nodes.

- random\_seed (int): Seed for the random number generator.

- plot (bool): Whether to plot the graph or not. Default is False.

Returns:

- g (igraph.Graph): The generated Barabasi-Albert graph.

"""

random.seed(random\_seed)

g = ig.Graph.Barabasi(n, m)

g.vs['color'] = 'lightgrey'

g.vs['alpha'] = None

while 0 in g.degree():

g = ig.Graph.Barabasi(n, m)

g.vs['color'] = 'lightgrey'

if plot:

fig, ax = plt.subplots()

ig.plot(

g,

target=ax,

vertex\_size=10

)

return g



Create a Watts-Strogatz graph.

def create\_WS\_graph(n, d, p, random\_seed, plot=False):

"""

Parameters:

- n (int): Number of nodes.

- d (int): Number of connected neighbors in each side. 2d is the initial degree of each node (and the expected degree of a node).

- p (float): Probability with which an edge will change one of its nodes.

- random\_seed (int): Seed for the random number generator.

- plot (bool): Whether to plot the graph or not. Default is False.

Returns:

- g (igraph.Graph): The generated Watts-Strogatz graph.

"""

random.seed(random\_seed)

g = ig.Graph.Watts\_Strogatz(1, n, d, p)

g.vs['color'] = 'lightgrey'

g.vs['alpha'] = None

while 0 in g.degree():

g = ig.Graph.Watts\_Strogatz(1, n, d, p)

g.vs['color'] = 'lightgrey'

if plot:

fig, ax = plt.subplots()

ig.plot(

g,

target=ax,

vertex\_size=10

)

return g



Selects the zero patient node based on the specified mode and rank.

def choose\_zero\_patient(g, zp\_mode, zp\_rk = None):

"""

Parameters:

- g: The graph object.

- zp\_mode: The mode for selecting the zero patient. Possible values are

'random', 'closeness\_centrality', 'degree\_centrality', and 'betweenness\_centrality'.

- zp\_rk: The rank for selecting the zero patient. Possible values are 'max' and 'min'. Default is None.

Returns:

- The index of the selected zero patient node.

Raises:

- Exception: If the specified mode or rank is not valid.

"""

if zp\_mode == 'random':

return 0

elif zp\_mode == 'closeness\_centrality':

cl = g.closeness()

if zp\_rk == 'max':

return cl.index(max(cl))

elif zp\_rk == 'min':

return cl.index(min(cl))

else:

raise Exception('No valid rank for zero patient.')

elif zp\_mode == 'degree\_centrality':

deg = g.degree()

if zp\_rk == 'max':

return deg.index(max(deg))

elif zp\_rk == 'min':

return deg.index(min(deg))

else:

raise Exception('No valid rank for zero patient.')

elif zp\_mode == 'betweenness\_centrality':

bet = g.betweenness()

if zp\_rk == 'max':

return bet.index(max(bet))

elif zp\_rk == 'min':

return bet.index(min(bet))

else:

raise Exception('No valid rank for zero patient.')

else:

raise Exception('No valid mode for zero patient.')



Simulates the spread of infection in a graph.

def spread(g, alpha, zp\_mode ='random', zp\_rk=None, alpha\_factor = 1, total\_time = 100, plot=False):

"""

s Parameters:

- g: igraph.Graph object representing the graph.

- alpha: float, the infection probability.

- zp\_mode: str, the zero patient selection mode. Default is 'random'.

- zp\_rk: int, the random seed for zero patient selection. Default is None.

- alpha\_factor: float, the factor by which the infection probability is multiplied after each iteration. Default is 1.

- total\_time: int, the total number of iterations. Default is 100.

- plot: bool, whether to plot the graph at each iteration. Default is False.

Returns:

- Time: numpy.ndarray, array of time steps.

- Infected: numpy.ndarray, array of the proportion of infected nodes at each time step.

"""

g.vs['Infected'] = False

zp = choose\_zero\_patient(g, zp\_mode, zp\_rk)

g.vs[zp]['color'] = 'red'

g.vs[zp]['Infected'] = True

g.vs[zp]['alpha'] = alpha

Infected = np.array([1/g.vcount()])

count = 1

for i in range(total\_time-1):

infected\_g = g.vs.select(Infected=True)

for v in infected\_g:

neighbours = g.vs[v.index].neighbors()

g.vs[v.index]['alpha'] \*= alpha\_factor

for u in neighbours:

if (not g.vs[u.index]['Infected']) and (random.random() < g.vs[v.index]['alpha']):

g.vs[u.index]['Infected'] = True

g.vs[u.index]['color'] = 'green'

g.vs[u.index]['alpha'] = alpha

count += 1

Infected = np.append(Infected, [count/g.vcount()])

if plot:

fig, ax = plt.subplots()

ig.plot(

g,

target=ax,

vertex\_size=10

)

plt.show()

Time = np.arange(len(Infected))

return Time, Infected



1. Using higher values of n yielded the same results and required a lot more time to execute, so we decided to go with this value of n. [↑](#footnote-ref-0)
2. Since the plots resulting from it are pretty much the same as before, we decided not to put them again. [↑](#footnote-ref-1)