# TKINI_CSC591VirusP5

***Option 1: Virus Propagation on Static Networks***

1. For the SIS (susceptible, infected, susceptible) Virus Propagation Model (VPM), with transmission probability β = β 1 , and healing probability δ = δ 1 , calculate the effective strength ( s) of the virus on the static contact network provided ( static.network). See supplementary material provided for details on the SIS VPM and on how to calculate the effective strength of a virus. Answer the following questions:

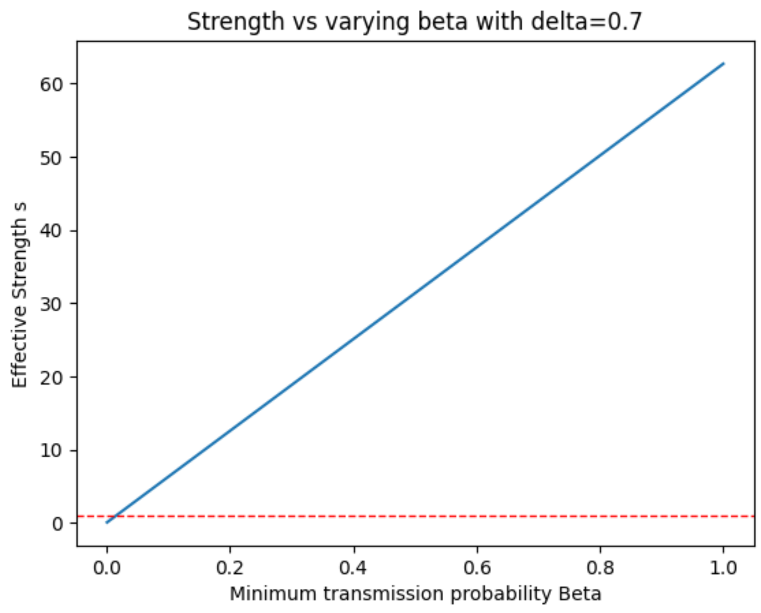
a. Will the infection spread across the network (i.e., result in an epidemic), or will it die quickly?

**1. For β = β1 (0.2) and δ = δ1 (0.70), the effective strength is 12.53.**
**Since effective strength is greater than 1, the infection will result in an epidemic.**
**2. For β = β2 (0.01) and δ = δ2 (0.60), the effective strength: 0.73.**
**Since effective strength is less than 1, the virus will not result in an epidemic.**

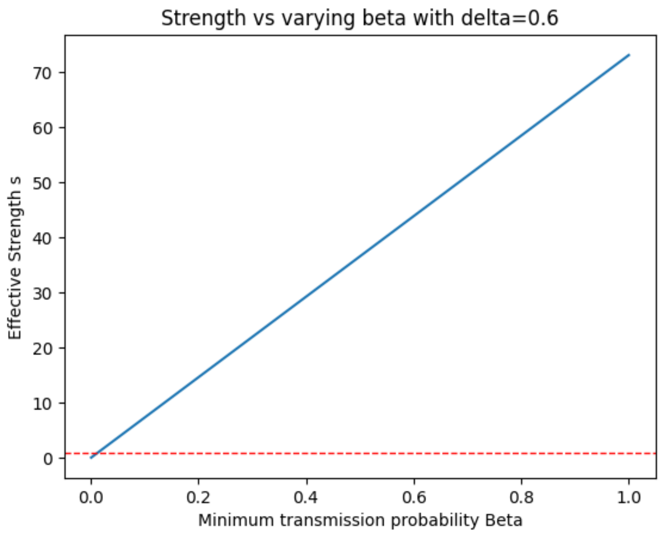
b. Keeping δ fixed, analyze how the value of β affects the effective strength of the virus ( suggestion: plot your results). What is the minimum transmission probability (β) that results in a network wide epidemic?

For this question, β ranges from 0.001 to 1 in 1000 equally spaced values.

1. **Fixed δ = δ1 (0.7), the minimum transmission probability (β) is 0.016.**



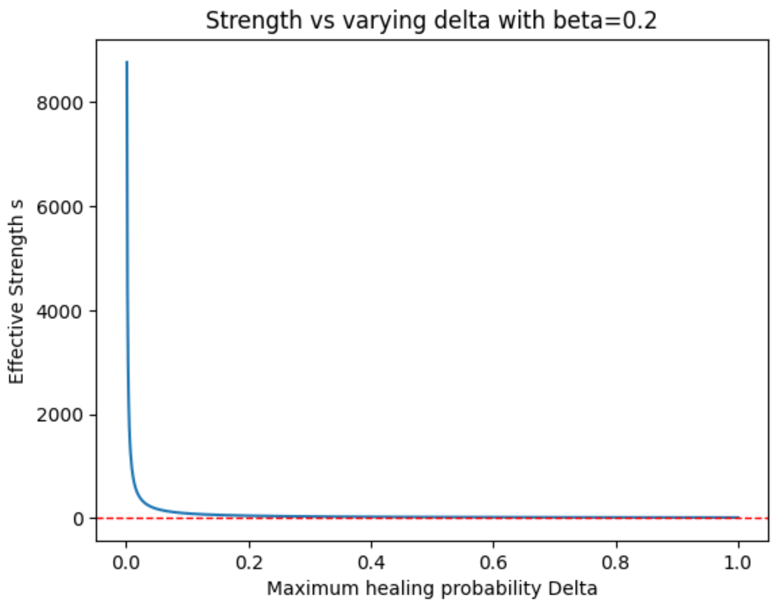
2.**Fixed δ = δ1 (0.6), the minimum transmission probability (β) is 0.0137.**



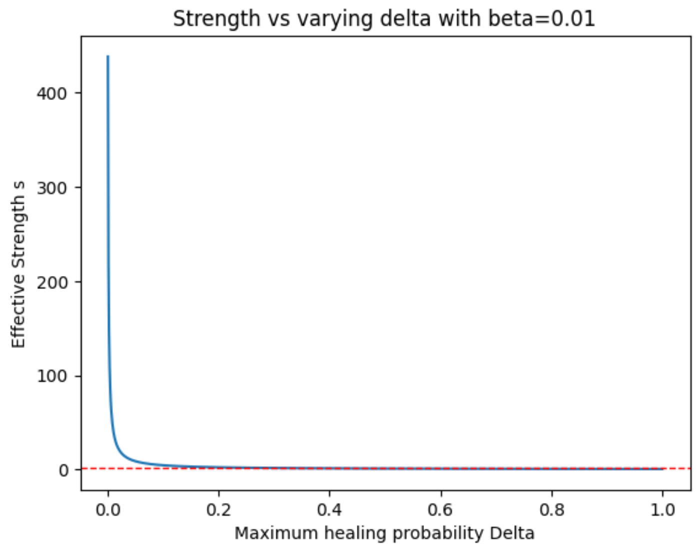
c. Keeping β fixed, analyze how the value of δ affects the effective strength of the virus ( suggestion: plot your results). What is the maximum healing probability (δ) that results in a Network wide Epidemic?

For this question, δ ranges from 0.001 to 1 in 1000 equally spaced values.

1. **Fixed β = 0.2, the maximum healing probability (δ) is 1.**



2. **Fixed β = 0.01, the maximum healing probability (δ) is 0.439**



2. Write a program that simulates the propagation of a virus with the SIS VPM, given a static contact network, a transmission probability (β), a healing probability (δ), a number of initially infected nodes ( c), and a number of time steps to run the simulation ( t). The initially infected nodes should be chosen from a random uniform probability distribution. At each time step, every susceptible (i.e., non infected) nodes have a β probability of being infected by neighboring infected nodes, and every infected node has a δ probability of healing and becoming susceptible again. Your program should also calculate the fraction of infected nodes at each time step.

a. Run the simulation program 10 times for the static contact network provided ( static.network), with β = β 1 , δ = δ 1 , c = n/10 ( n is the number of nodes in the network), and t = 100.

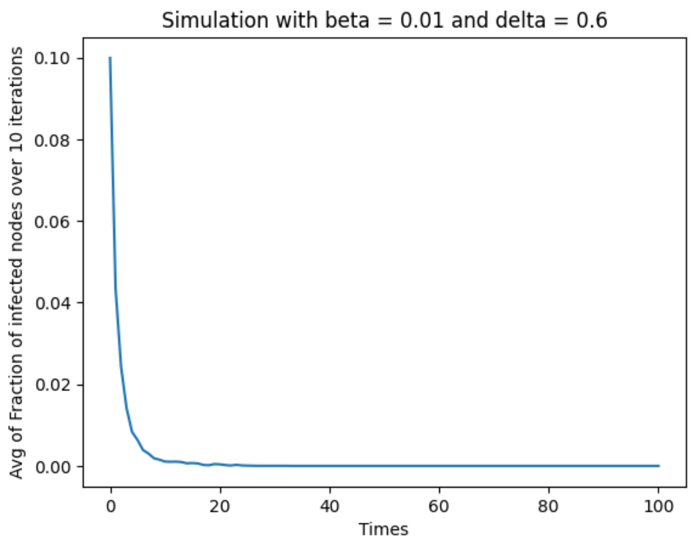
b. Plot the average (over the 10 simulations) fraction of infected nodes at each time step. Did the infection spread across the network, or did it die quickly? Do the results of the simulation agree with your conclusions in (1a)?

c. Repeat (2a) and (2b) with β = β 2 , and δ = δ 2 .

Simulation with β = β1 , and δ = δ1 gives the following result:

It is taking a very long time to compute the results. Have run in for more than 12 hrs on a 16GB RAM laptop. Maybe there is some optimization required in the code.

Simulation with β = β 2 , and δ = δ 2 gives the following result:



***As we see the fraction of nodes infected approaches 0 at t=100, we can say that there is no epidemic and the result is consistent with the results in Q1a as strength s < 1.***

3. Write a program that implements an immunization policy to prevent the virus from spreading across the network. Given a number of available vaccines ( k) and a contact network, your program should select k nodes to immunize. The immunized nodes (and their incident edges) are then removed from the contact network.
a. What do you think would be the optimal immunization policy? What would be its time complexity? Would it be reasonable to implement this policy? Justify.
For your program, use the following heuristic immunization policies:
Policy
A: Select k random nodes for immunization.
Policy
B: Select the k nodes with highest degree for immunization.
Policy
C: Select the node with the highest degree for immunization. Remove this node (and its incident edges) from the contact network. Repeat until all vaccines are administered.
Policy
D: Find the eigenvector corresponding to the largest eigenvalue of the contact network's adjacency matrix. Find the k largest (absolute) values in the eigenvector. Select the k nodes at the corresponding positions in the eigenvector.
For each heuristic immunization policy (A, B, C, and D) and for the static contact network provided
( static.network), answer the following questions:

b. What do you think is the intuition behind this heuristic?

- **For policy A, we immunize any k nodes that are randomly selected. This can be the case when the vaccines are given on a first come first serve basis or in a very**

**haphazard random manner. No fixed rule is followed for selecting a node. This policy can only be effective if k is comparable to the number of nodes.**

- **For policy B, we try to immunize the top k nodes with the highest degree. These nodes have a high number of edges so they can infect other nodes. Immunizing them can delay the spread of the virus. This is better than policy A as the nodes are selected using some strategy.**
- **For policy C we use a greedy approach at each step. We find out the node with the highest degree and then vaccinate it. All incident edges are removed and then the process is repeated. This policy is better than policy B as we take a greedy approach. In policy B it might be possible we are immunizing a dense subgraph but leaving the large sparse graph susceptible to the virus.**
- **For policy D we can say that high eigen vector values give us nodes that are connected to many nodes that in turn are connected to other nodes. This shows that the information is depicted with the highest extent possible. Immunizing such nodes would make the network less prone to epidemics.**

c. Write a pseudocode for this heuristic immunization policy. What is its time complexity?
Policy A:
Input: g: networkx graph, k: number of vaccines
Output adj: Updated adjacency matrix with immunized nodes removed from the graph.
def policyA(g, k):

```
def policy_A(g, k):
    immunized = set()
    total_node = nx.number_of_nodes(g)
    # get k random nodes
    immunized = list(np.random.choice(total_node,k,replace=False))
    for i in immunized:
        g.remove_node(i)
    # vaccinate the chosed nodes, no edge so cannot infect anyone
    # create empty adjacency matrix
    new_adjacency_matrix = [[0 for _ in range(total_node)] for _ in
range(total_node)]
    # populate the adjacency matrix 1 edge at a time
    for i in nx.edges(g):
        new_adjacency_matrix[i[0]][i[1]] = 1
        new_adjacency_matrix[i[1]][i[0]] = 1
    return new_adjacency_matrix
```

Time complexity analysis:
Select k random numbers: O(k)
Update adjacency matrix: O(2E )
Thus, the total time complexity of policy A is O(k+2E) = O(E) assuming k<<E

Policy B:
Input: g: networkx graph, k: number of vaccines
Output adj: Updated adjacency matrix with immunized nodes removed from the graph.

```python
def policy_B(g, k):
    immunized = set()
    total_node = nx.number_of_nodes(g)
    # sort the nodes based on degree of node
    degree = sorted(list(g.degree()), key=lambda x: x[1], reverse=1)
    for i in range(k):
        immunized.add(degree[i][0])

    for node in immunized:
        # vaccinate the chosen nodes, no edge so cannot infect anyone
        g.remove_node(node)
    # create empty adjacency matrix
    new_adjacency_matrix = [[0 for _ in range(total_node)] for _ in
range(total_node)]
    # populate the adjacency matrix 1 edge at a time
    for i in nx.edges(g):
        new_adjacency_matrix[i[0]][i[1]] = 1
        new_adjacency_matrix[i[1]][i[0]] = 1
    return new_adjacency_matrix
```

Time complexity analysis:
Sort degree list: O(NlogN )
Select k largest degree nodes : O(k)
Remove the k nodes : O(k)
Update adjacency matrix: O(2E )
Thus, the total time complexity of policy B is O(NlogN +2k + 2E) = O(Nlog N+ E)

Policy C:
Input: g: networkx graph k: number of vaccines
Output adj: adjacency matrix with specific node with immunized nodes removed from the graph.

```python
def policy_C(g, k):
    immunized = set()
    total_node = nx.number_of_nodes(g)
    while len(immunized) < k:
        # get the node with the highest degree
        degree = sorted(list(g.degree()), key=lambda x: x[1], reverse=1)
        immunized.add(degree[0][0])
        #remove the node from the graph and iterate again
        g.remove_node(degree[0][0])

    # create empty adjacency matrix
    new_adjacency_matrix = [[0 for _ in range(total_node)] for _ in
range(total_node)]
    # populate the adjacency matrix 1 edge at a time
    for i in nx.edges(g):
        new_adjacency_matrix[i[0]][i[1]] = 1
        new_adjacency_matrix[i[1]][i[0]] = 1
    return new_adjacency_matrix
```

Time complexity analysis:
Sort degree list: O(k* Nlog N )
Remove node from graph : O(k)
Adjacency matrix generation: O(2E )
Thus, the total time complexity of policy C is O(k*NlogN + k + 2E) = O(k*NlogN+E)

Policy D:
Input: adjacency_matrix: adjacency matrix of graph,
g: networkx graph   k: number of vaccines
Output adj: adjacency matrix with specific node with immunized nodes removed from the graph.

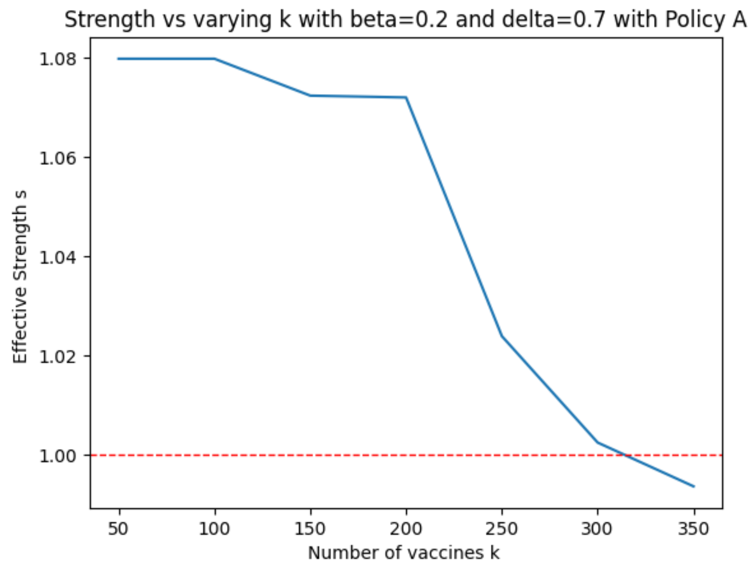
```
def policy_D(adjacency_matrix, g, k):
    total_node = nx.number_of_nodes(g)
    value, vector = np.linalg.eig(adjacency_matrix)
    eigen_set = [(value[i], vector[i]) for i in range(len(value))]
    eigen_set = sorted(eigen_set, key=lambda x: x[0], reverse=1)
    #get the eigen vetcor for corresponding largest eigen value
    largest_vector = eigen_set[0][1]
    largest_vector = np.absolute(largest_vector)
    # get the first k largest value in the eigen vector and get the
corresponding node
    target = [u[0] for u in sorted(enumerate(largest_vector),
reverse=True, key=itemgetter(1))[:k]]
    for i in target:
        g.remove_node(i)
    # create empty adjacency matrix
    new_adjacency_matrix = []
    #total_node = nx.number_of_nodes(g)
    new_adjacency_matrix = [[0 for _ in range(total_node)] for _ in
range(total_node)]
    # populate the adjacency matrix
    for i in nx.edges(g):
        new_adjacency_matrix[i[0]][i[1]] = 1
        new_adjacency_matrix[i[1]][i[0]] = 1
    return new_adjacency_matrix
```

Time complexity analysis:
Computing eigenvalues: O(N^3 )
Sort eigenvalues: O(NlogN)
Remove node from graph : O(k)
Adjacency matrix generation: O(2E )
Thus, the total time complexity of policy C is O(N^3 + NlogN + k +2E) = O(N^3)

d. Given k = k1 , β = β 1 , and δ = δ 1 , calculate the effective strength ( s) of the virus on the immunized contact network (i.e., contact network without immunized nodes). Did the immunization policy prevent a network wide epidemic?
***Immunization with Policy A***
***Effective strength of Policy A with beta = 0.2, delta = 0.7 is: 12.368142803635365***
***Since 12.37 >> 1 we can say that this will not prevent a epidemic***
***Immunization with Policy B***
***Effective strength of Policy B with beta = 0.2, delta = 0.7 is: 1.0802754802352394***
***Since 1.080 is near to 1 we can say this policy may prevent an epidemic***
***Immunization with Policy C***
***Effective strength of Policy C with beta = 0.2, delta = 0.7 is: 1.0845205395194377***
***Since 1.085 is near to 1 we can say this policy may prevent an epidemic***
***Immunization with Policy D***
***Effective strength of Policy D with beta = 0.2, delta = 0.7 is: 5.10062188256173***
***Since 5.1 >> 1 we can say that this will not prevent a epidemic***

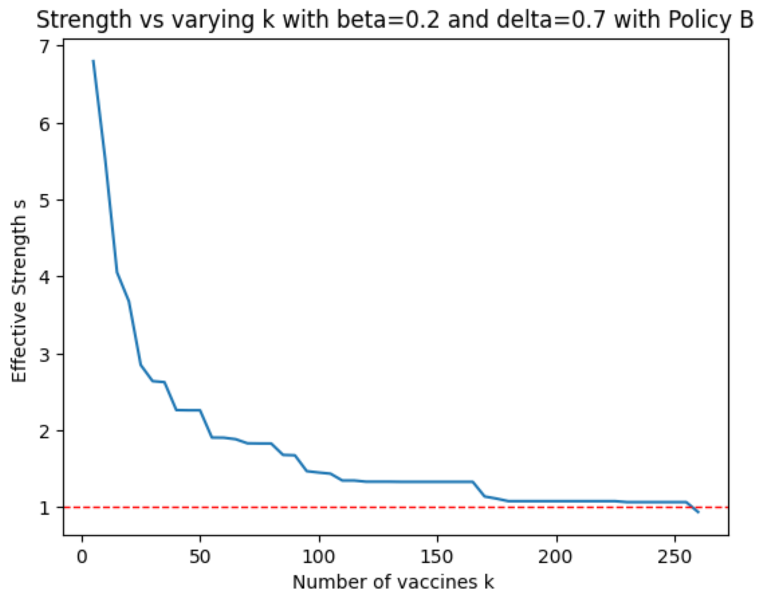
e. Keeping β and δ fixed, analyze how the value of k affects the effective strength of the virus on the immunized contact network (suggestion: plot your results). Estimate the minimum number of vaccines necessary to prevent a network wide epidemic.

Policy A:



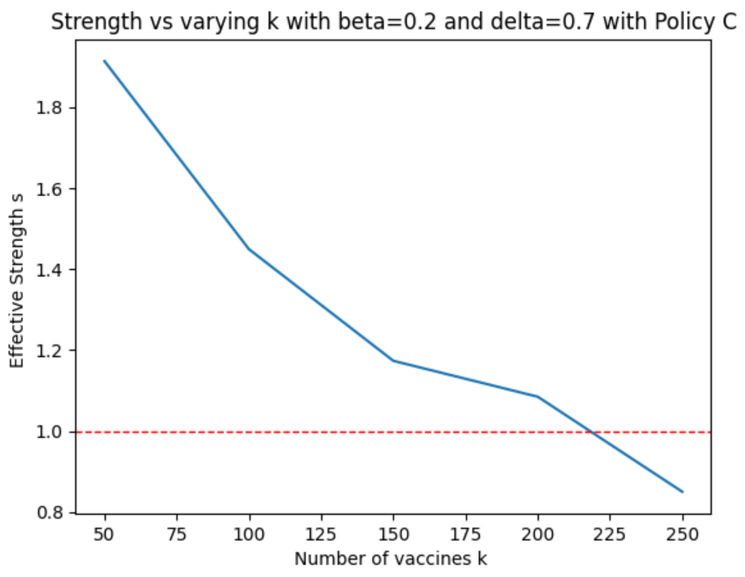
***The minimum number of vaccines required is about 315***

Policy B:



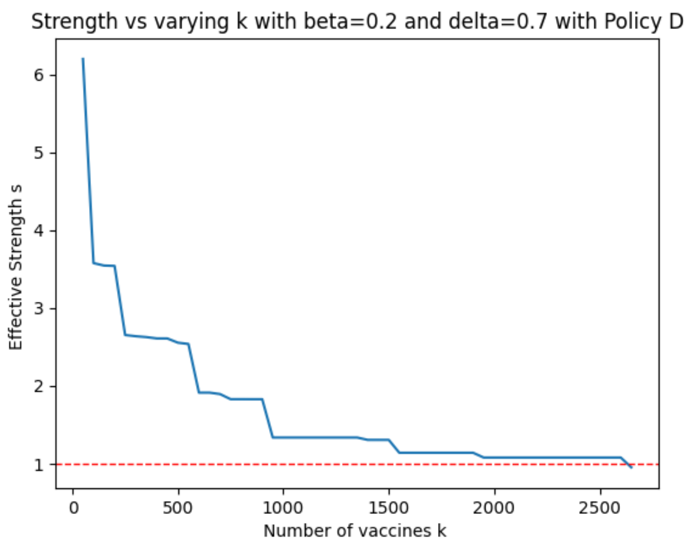
***The minimum number of vaccines required is about 260***

Policy C:



***The minimum number of vaccines required is about 220***

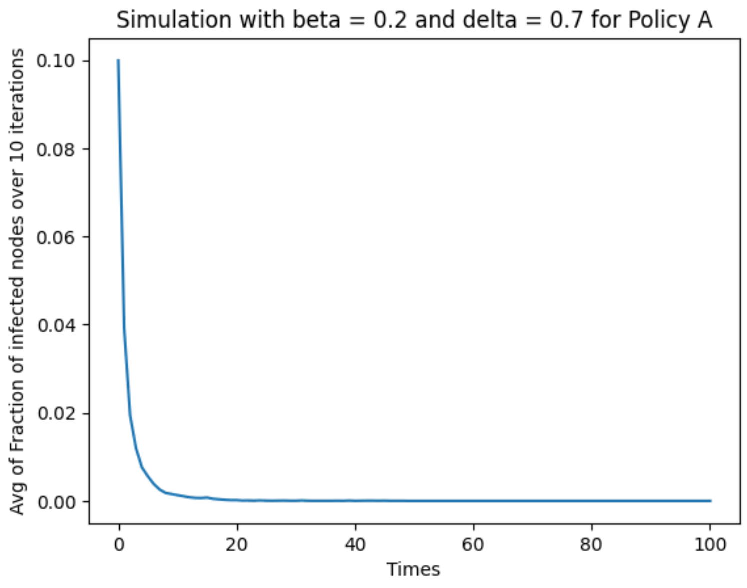
Policy D:



***The minimum number of vaccines required is about 2650***

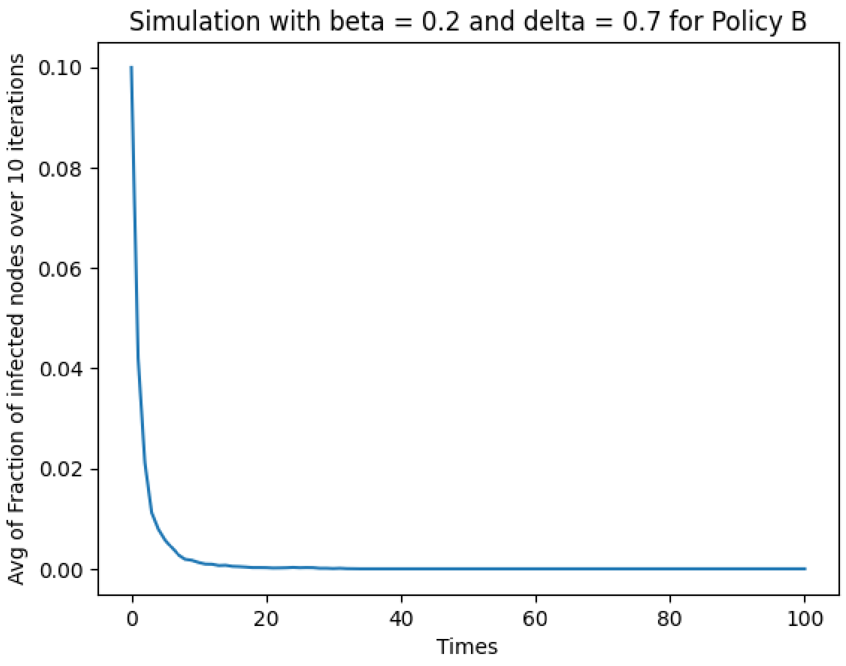
f. Given k = k1 , β = β 1 , δ = δ 1 , c = n/10, and t = 100, run the simulation from problem (2) for the immunized contact network 10 times. Plot the average fraction of infected nodes at each time step. Do the results of the simulation agree with your conclusions in (3d)?

Policy A:



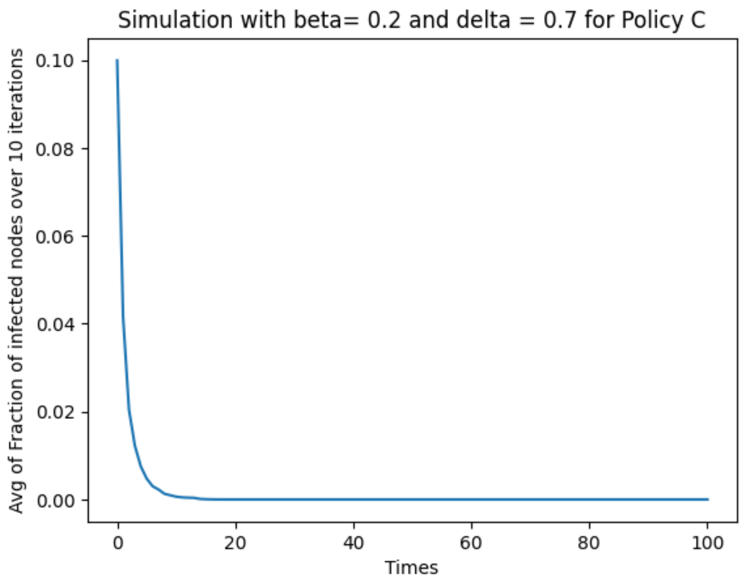
***As we see the fraction of nodes infected approaches 0 at t=100, we can say that there is no epidemic and the result is not consistent with the results in Q3d. This might have happened as the random nodes selected would have been those with high degree and their immunization would have prevented spread of virus.***

Policy B:



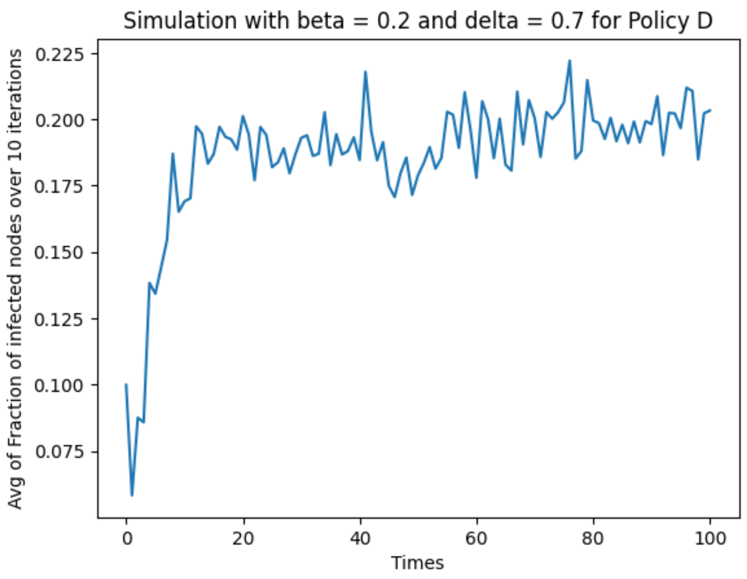
***As we see the fraction of nodes infected approaches 0 at t=100, we can say that there is no epidemic and result is consistent with the results in Q3d***

Policy C:



***As we see the fraction of nodes infected approaches 0 at t=100, we can say that there is no epidemic and result is consistent with the results in Q3d***

Policy  D:



***As we see the fraction of nodes infected does not approach 0 at t=100, we can say that there is an epidemic and result is consistent with the results in Q3d***