SE Project Documentation

Contributors:

Name	UnityID
Rohan Pillai	rspillai
Bhoomi Shah	bshah2
Shruti Kangle	sskangle
Poorva Kulkarni	pnkulkar
Alisha Shahane	asshahan

Problem Description:

The objective of the project is to decide whether a given virus with parameters such as initial number of infected people, number of available vaccines, healing probability, transmission probability will eventually result into a pandemic or not.

Class Description:

The class “Pandemic” implements all methods required for the following:

- Check whether a virus results in a pandemic or not
- Run simulation
- Implement immunization policies

Member variables:

- Transmission probability (b)
- Healing probability (d)
- Number of vaccines (num_vaccine)

Function Descriptions:

1. getGraph(self, filename):

   Input:
   - filename: Name of the graph file

   Output:
   - g: Graph formed using the library igraph

   Description:

   For every line in the network file, it adds two nodes and an edge between them to the graph.

2. getStrength(self, g):

   Input:
   - g: The igraph

   Output:
   - strength: Strength of the graph

   Description:

   The strength of a graph can be calculated using this formula,

   Strength = maximum_eigen_value * (b / d)

3. predictPandemic(self, strength):

   Input:
   - strength: The strength of the graph

   Output:

   The function tells whether the virus has resulted into a pandemic or not.

   Description:

   If the strength of the virus is greater than 1, it will result in a pandemic.

4. runOneSimulation(self, g, time_step_per_simulation, initially_infected_nodes_perc)
   Input:
   - g: The graph
   - time_step_per_simulation: This indicates how many time_steps (for eg days) should the simulation run for.
   - initially_infected_nodes_perc: Percentage of people in the graph initially infected before the simulation starts.

   Output:
   - infected_count: It is the number of people infected after that simulation.

   Description:
   - Step1: Calculate the number of people infected initially.
   - Step2: For every infected person, we calculate the number of its neighbors that can be infected. This is done based on transmission probability (beta).
   - Step3: Similarly, based on healing probability (delta) how many of the infected people will recover.
   - Step4: Repeat 2 and 3 for time_step_per_simulation number of times.

5. runAllSimulation(self, g, simulation_count,time_step_per_simulation, initially_infected_nodes_perc)

   Input:
   - g: The graph
   - simulation_count: The number of times you wish to run the simulation.
   - time_step_per_simulation: This indicates how many time_steps (for eg days) should the simulation run for.
   - initially_infected_nodes_perc: Percentage of people in the graph initially infected before the simulation starts.

   Output:

   Based on the calculations, it decides if a virus results into a pandemic or does it die out. Also plots the results where x axis represents time step and y axis represents average infected people per time stamp.

   The average number of infected nodes for every time step, is an average of number of infected people for that particular time step across all simulations.

   Description:
   - Step1: Calculate the number of people infected initially.
   - Step2: For every infected person, we calculate the number of its neighbors that can be infected. This is done based on transmission probability (beta).
   - Step3: Similarly, based on healing probability (delta) how many of the infected people will recover.
   - Step4: Repeat 2 and 3 for time_step_per_simulation number of times.

6. runImmunization(self, g, policy)

   Input:
   - g: The graph
   - policy: Immunization policy (first or second or third or fourth)

   Output:

   Displays plot of strength as we go on increasing the number of vaccines.

   Description:

   It is the driver code for implementing one of the four immunization policies.

7. firstImmunizationPolicy(self, g, num_vaccines):

   Input:

   - g: The graph
   - num_vaccines: Vaccines available for immunization in that iteration

   Output:

   New immunized graph.

   Description:

   First policy randomly selects people who get a vaccine. The chosen number of people should be equal to vaccines available in that iteration (num_vaccines).

8. secondImmunizationPolicy(self, g, num_vaccines): #TO BE IMPLEMENTED

   Input:

   - g: The graph
   - num_vaccines: Vaccines available for immunization in that iteration

   Output:

   New immunized graph.

   Description:

   In this policy, it selects the people with highest degree to give vaccine. The chosen number of people should be equal to vaccines available in that iteration (num_vaccines).

9. thirdImmunizationPolicy(self, g, num_vaccines): #TO BE IMPLEMENTED

   Input:

   - g: The graph
   - num_vaccines: Vaccines available for immunization in that iteration

   Output:

   New immunized graph.

   Description:

   The policy iteratively gives vaccine to the node with highest degree. The number of iterations will be equal to the available vaccines.

10. fourthImmunizationPolicy(self, g, num_vaccines): #TO BE IMPLEMENTED

Input:

- g: The graph
- num_vaccines: Vaccines available for immunization in that iteration

Output:

New immunized graph.

Description:

The policy finds people corresponding to the maximum eigen value in the graph. Vaccine will be provided to these people which are equal to the number of available vaccines.

Instructions:

The code runs on the following global parameter combination,

1.
   - Transmission probability = beta_1 = 0.01
   - Healing probability = delta_1 = 0.6
   - Number of vaccines = [20, 21, 22,…30]

2.
   - Transmission probability = beta_2 = 0.2
   - Healing probability = delta_2 = 0.7
   - Number of vaccines = [15, 20, 25, …50]

General Steps to run:

1. Create class object with any of the above parameter combinations.
2. Create a graph using member function.
3. Predict whether it results to a pandemic.
4. Run simulations with desired parameter.
5. Run any of the 4 immunization policies.