Programming Assignment 1

Due October 24, 2020 at 11:59 PM

1 General Description

Models of how a virus spreads through a community play an important role in deciding on public health measures. Graph algorithms are often used to simulate the spread. In this assignment, you will practice on aspect of that type of simulation.

Suppose we have a community modeled by a graph G, where vertices represent people and edges represent social connections. We will use a very simple model of virus transmission: there is a set of initially infected people ("sources"), and people get the virus with some probability through social contact with a person that is infected. Given the initial sources, and the social connections among the individuals, you have to estimate who will get infected in the community.

In this assignment, you will implement an algorithm that emulates the spread of the virus. With the help of this algorithm, you will experiment on various types of input to infer characteristics of the virus spread. You will submit working code through Gradescope, as well as summarize your findings in a report. Supported languages are Python and Java. (Unfortunately, we won't be able to support C++.)

Collaboration policy. For this assignment only very limited collaboration is allowed. You can discuss high level ideas with your classmates. However, you **cannot** share code or experimental results with each other. The findings in your report have to be your own. We will use plagiarism detection tools on the BFS portion of your assignment.

2 Deliverables

You need to submit the following.

- A working implementation of BFS as described in section 5.1. This code will be run on various test files in the autograder.
- A 2-page pdf document summarizing your experimental findings as described in section 4. The page limit is strict! Make sure that your writing is concise and easy to follow. While sometimes it makes sense to include certain counts (e.g. number of nodes infected) or numerical observations (e.g. results change significantly when $p = 0.5$), usually including raw data is not helpful. You can instead for example add figures (see examples in section 4) or verbal explanations.

3 Technical Description

Model setup. We model the community with help of a graph $G(V, E)$. Nodes correspond to people, and an edge between two nodes represent a social connection (e.g. friendship) between them. The input to this model is G, the set S of source nodes (may be multiple nodes) that are initially infected by the disease ($S \subset V$) , and the infection probability p.

We measure the disease propagation in days. Each day some people get infected. Initially on day 1 only the sources are sick. Once a person gets sick, the next day they infect each of their contacts with probability p. (After that first day they quarantine and don't infect any additional people on subsequent days.) Infections are independent from each other.

Incorporating the random spread. Given the random process for node infection it is hard to give an exact formula on how likely each node is going to be infected. However, we can get a reasonable estimate by measuring it empirically. We play out the infection process multiple times on G (using the same sources) and keep track of the infection of each individual node. We then take the average outcome for each node over the multiple process instances.

We call an edge e *active* if one node infected its neighbor through this edge. Observe, that the nodes in V that eventually get infected are the ones that are connected to a source through active edges. The day that a node v is infected is the distance of v to a source.

Hence, we can generate an instance G_i of the infection propagation by removing all edges from G that are inactive. Formally, we generate a graph $G_i(V, E_i)$, such that the nodes in G and G_i are the same. We obtain the set E_i by adding each edge in E independently with probability p. The edges in E_i are the active edges. (Thus, E_i is a subset of E.)

We can use BFS on this random subgraph G_i to find out whether—in this particular random instance—a node was infected and, if so, on what day.

Algorithm 1: GenRndInstance($G(V, E), p$)

```
1 E_i = ∅ ;
2 for e in E do
3 |   active = rnd([0,1]) /* generate random number                */
  |   /* edge is active with probability p                          */
4 |   if active ≤ p then
5 |   |   E_i.add(e);
6 return E_i
```

To find the average outcome of the process for each node, we repeat the following 100 times: first generate a graph instance G_i and then run BFS on G_i.

For each node v we can compute how likely it is to get infected by computing the fraction of instances that its distance from the source is finite. For the instances where v did get infected we can also compute the average time to infection by taking its average distance from the source.

Multiple sources. In this infection model it is possible that there are multiple sources $s_1, s_2, \ldots s_k$, i.e. several people are initially infected. We can deal with this by adding an additional node s. Add an edge (s, s_i) connecting it to each source, and add these edges to E_i with probability 1. (This ensures that all the sources get marked as infected on the first day.)

4 Experiments

As the second part of your assignment you will apply your algorithm to different types of graphs (the graphs are provided by us). We ask you to observe and discuss the virus propagation in the context of the structure of each graph. All of the graphs are synthetic generated so that they exhibit the desired properties for sure.

Run all of your experiments for each four of the infection probabilities $p_1 = 0.1$, $p_2 = 0.3$, $p_3 = 0.5$ and $p_7 = 0.7$ and report your findings on all.

4.1 Input.

We give you three graphs and source nodes for each. The graphs are artificially generated, each one based on a different principle.

Grid graph. (We refer to this graph as G_{gr}) In this graph nodes and edges are as if they are laid out on a square grid. It has 1024 nodes. Each node has degree 4 and is part of exactly 4 cycles with 4 edges. (e.g. edges are laid out along the edges of the squares in the grid.) While this graph is quite artificial, some real life networks are quite close to emulating this structure. Roughly, it models a setting where people can only infect the people who live very nearby. (The grid structure shows up in lots of real-world settings. One well known dataset with similar properties is the US Electrical Power Grid [1]. This data set does contain a few high degree nodes (which G_{gr} doesn't) that correspond to large relay stations. Road networks, such as the one for Pennsylvania [2], exhibit similar structure.)

sources. This data set comes with 4 source nodes that form a length-4 cycle. (i.e. they are the vertices of one of the squares.)

Grid graph with shortcuts. (denoted by G_{gs}) This graph is very similar to the vanilla grid graph. In fact, it is the same graph – consisting of 1024 nodes that are laid out along the edges of a square grid. However, in addition to the grid edges, each node v also connects to one additional node u chosen uniformly at random. (note, that it is possible that u is the additional neighbor to multiple nodes).

This graph is an example of a so called "small world" graph. Due to the shortcut edges the diameter (= maximum lengths of the shortest paths between pairs of nodes) of this graph is very low.

sources. The source nodes used are the same as for the vanilla grid.

Scale free network. (denoted by G_{sf}) These graphs follow a so-called Power-law degree distribution. In such graphs, the fraction of nodes of degree d is about $d^{-\gamma}$. Here γ is a parameter that

[1] `http://konect.cc/networks/opsahl-powergrid/`

[2] `http://snap.stanford.edu/data/roadNet-PA.html`

is characteristic of a graph. Typical values for γ are between 2 and 3. What this means is that there are a small number of nodes in this graph that have very high degrees, while the majority of the nodes have low degrees. The term *scale-free* refers to the fact that the degree distribution of the graph is independent of the size of n, it does not "scale" with the size. Check out this excellent set of slides by the SNAP group at Stanford [3]

We used a built-in graph generator for power-law graphs [4] from the well-known Python graph package NetworkX. The generator is an enhanced version of the Barabasi-Albert preferential attachment model [5].

sources. We give you two sets of source nodes for this data set. The first consists of two high degree nodes (One has degree 87, the other 135). The second consists of four nodes, each of degree 4.

Some background information: Power-law graphs are of interest since many real-life networks have been found empirically to follow such degree distributions (or other distributions with similar shapes). Some examples are the friendship connections in social networks[6] *and the hyper-link connections on the World Wide Web*[7]*. There are ongoing debates as to how well real life networks are modeled by these distributions*[8]*.*

4.2 Data Analysis.

You will analyze the infection spread on the various types of graphs. We give you multiple tasks. We provide you with very precise instructions for some of them and then there are a few more open ended questions.

Workflow. You will first compute the outcome of the virus propagation on the various inputs. Then you will analyze those results as specified below. For the open ended questions you may decide to run additional computations.

Input. You will run your initial experiments on the following data set + source combinations.

- `grid_1.txt` Contains the graph G_{gr} with one source (specified in the second line of the graph) and 1024 nodes. (Denote this graph with one source by G^1_{gr}.)
- `grid_4.txt` Contains the graph G_{gr} with 4 sources (specified in the second line of the graph) and 1024 nodes. (Denote by G^4_{gr})
- `grid_shortcut_1.txt` Contains the graph G_{gs} with one source (specified in the second line of the graph) and 1024 nodes. (Denote by G^1_{gs})

[3] `http://snap.stanford.edu/class/cs224w-2015/slides/04-powerlaws.pdf`.

[4] `networkx.generators.random_graphs.powerlaw_cluster_graph`

[5] It takes as input the number of nodes n an integer m and a probability p. We used parameters $n = 1000$, $m = 4$, $p = 0.3$. It is initiated with a complete graph of m nodes. Then nodes are added one at a time. Each new node connects to m others at random with probabilities proportional to the other nodes' degrees. Finally if node u is recently added and connected to v and w, then an extra edge is added between v and w with probability p.

[6] For example, Mislove et. al.`https://mislove.org/publications/SocialNetworks-IMC.pdf` find that the networks they investigate are in fact close to power-law, though they don't show a perfect fit.

[7] `http://snap.stanford.edu/class/cs224w-readings/faloutsos99powerlaw.pdf`

[8] https://www.quantamagazine.org/scant-evidence-of-power-laws-found-in-real-world-networks-20180215/

- `grid_shortcut_4.txt` Contains the graph G_{gs} with 4 sources (specified in the second line of the graph) and 1024 nodes. (Denote by G^4_{gs})
- `scalefree_high.txt` Contains G_{sf} with 2 high degree sources (specified in the second line of the graph) and 1000 nodes. (Denote by G^h_{sf})
- `scalefree_low.txt` Contains G_{sf} with 2 low degree sources (specified in the second line of the graph) and 1000 nodes. (Denote by G^l_{sf})

Running instructions. For each of the above data sets you run your model on all **four** values of p: $p = 0.1$, $p = 0.3$, $p = 0.5$, $p = 0.7$. For each combination of data – source – probability you run your model **100** times and take the average over those outcomes. Ideally, the many runs will be incorporated in your code and you only need to specify the input each time.

Output. For each run of your experiment you will retrieve two lists of length n. Your analysis will use these lists as input. The lists are (1.) `prob_infect` contains at index i the probability that node i is infected. (2.) `avg_day` contains at index i the average day (distance from a source) that node i is infected – among the days that it's infected at all.

Analysis of grid graphs. Describe your findings of your experiments. Often it is useful to create a figure with your data and include that in your report too. (Note that the example figures in this section are for results on a different input than are given to you.)

1. Run the experiments on both G^1_{gr} and G^4_{gr}. For each four value of p report the number of nodes infected in the two graphs. Is the number of infected nodes in G^4_{gr} significantly more than G^1_{gr}, for what values of p? Plot the *infection probability distribution* for both graphs. Figure 1 depicts such a plot. The X-axis corresponds to nodes, the Y-axis to the probability that each node is infected. The nodes are ordered in descending order of infection probability. The results for all four p are depicted in the same plot. Write your observations whether you think the probability distributions for G^1_{gr} and G^4_{gr} and the ps follow the same pattern and why. If it helps with your comparison, you may take the ratio of probabilities. E.g. for $p = 0.5$ take the probability distribution of G^1_{gr} and divide by the value for G^4_{gr}. The closer the results are to 1, the more similar the two results are.
2. Compare the average day that nodes get infected in G^1_{gr} and G^4_{gr}. Report your observations for all values of p. Round the average infection days for each node to an integer and then represent the data in a histogram similar to Figure 2. In our figure, to make the data easy visible we created two separate histograms, the first containing results for $p = 0.1$ and $p = 0.3$. The second for $p = 0.5$ and $p = 0.7$. Feel free to show your results in as few or as many histograms as you think is best.
3. Perform the same analysis as above on the probability distribution on inputs G^1_{gs} and G^4_{gs}.
4. Perform the same analysis as above on the infection day on inputs G^1_{gs} and G^4_{gs}.
5. Compare the outcome of your experiments on the grid without and with the shortcuts. For the graphs G^4_{gr} and G^4_{gs} compare the number of infected nodes and the infection probability distributions for all four p values. Further, compare the data on their average infection days.

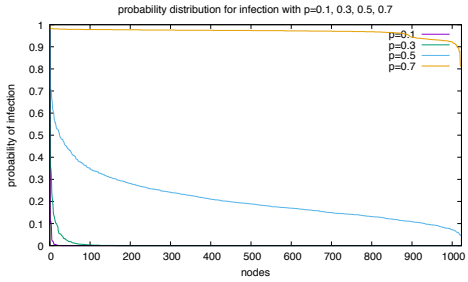


Figure 1: Probability of nodes in the grid being infected in descending order. Note that for $p = 0.1$ and $p = 0.3$ many nodes have 0 probability.

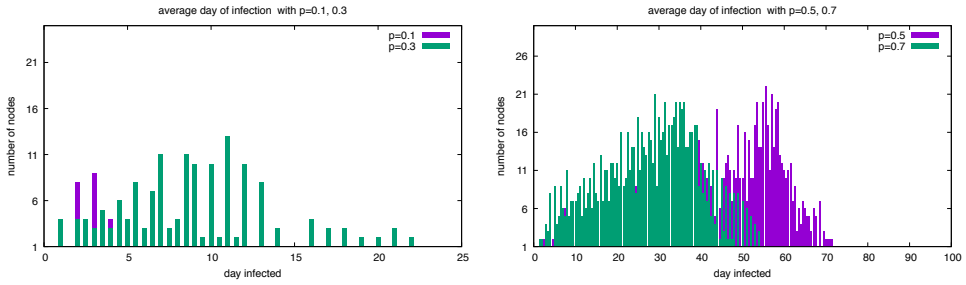


Figure 2: Average day of infection on the grid with one source for $p = 0.1$ and $p = 0.3$ (left) and $p = 0.5$, $p = 0.7$ (right).

In your explanation you may refer to figures included in the previous points. Or, if you think a new figure would help with your clear explanation , feel free to include it.

Comment whether you observed the same trends when comparing G^1_{gr} and G^1_{gs}. If they are very similar, then there is no need to share your figures for the latter comparison. If there a difference, then show the data you think best aides your argument.

6. *(open ended)* Formulate your opinion, how do the presence of the extra "shortcut" edges in G_{gs} influence the outcome when comparing G_{gr} and G_{gs}? (For both 1 and 4 sources.)

7. *(open ended)* For both G_{gr} and G_{gs} there seems to be a difference in terms of infected population between the infection probability being $p < 0.5$ or $p \geq 0.5$. What is your intuition for this and why? (Argue based on your understanding of the edge-layout of the graphs and their active edges.)

Analysis of scale free graphs.

1. Analyze the outcome on the scale free networks G^h_{sf} and G^l_{sf}. Again, run your experiments for all four values of p. Compare the infection probability distributions. This should be similar to the analysis you did to compare G^1_{gr} and G^4_{gr}.

2. Compare the average days of infection similar to your comparison earlier.

3. *(open ended)* Formulate your intuition, which nodes are most likely to get infected in both experiments G^h_{sf} and G^l_{sf}? Do you see a correlation between the distance of the infected node to the sources? Is there a correlation between the degree of a node and its infection probability? For this question we don't expect you to run a full analysis. Instead, look up some of the high probability nodes manually and base your explanation on those.

5 Implementation Details

Input files. The input files are given as .txt files. For a graph with N nodes, the file consists of $N + 2$ lines. Nodes are indexed $0 \dots N - 1$.

- line 1: number of nodes N
- line 2: comma separated integers corresponding to the source ids.
- lines $3 \dots N + 2$: line i consists of the comma separated list of neighbor ids of node $i - 3$.

5.1 Part 1: BFS implementation

We ask you to write your own BFS function with multiple source nodes. This is the basic BFS, without any of the random factors from the infection simulation. Your function will return the distance/level of each node to the nearest source. Your function will be tested for correctness on various input graphs through Gradescope.

starter code. We provide you with starter code in both Python and Java:

- `breadth_first_search.py`
- `breadth_first_search.java`

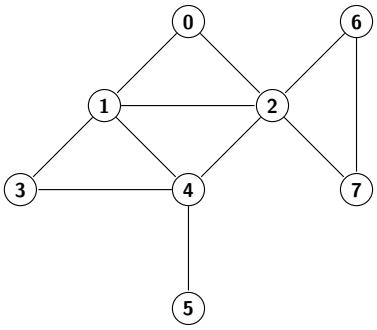
These files contain the code to read the content of the .txt files and load it in to predefined variables. It also contains the code to write the output of your BFS algorithm to a txt file into a format the autograder will accept. **Please do not alter the code already in the files, and do not change the name of the files, or the autograder will not accept them.**

The `readGraph(inputfile)` function returns the following three variables (with these exact names):

1. `N`: An integer representing the number of vertices in the graph.
2. `s`: List of integers representing the source nodes.
3. adj_list: A list of lists in which the 0th element is a list of vertices adjacent to vertex 0, the 1st element is a list of vertices adjacent to vertex 1, the 2nd element is a list of vertices adjacent to vertex 2, etc.

Write your own BFS. You will implement the body of the function `BFS(N,s, adj_list)`. Within the function the variable `level` is initialized, which is an n-sized list (populated with 'x's to serve as placeholders). Your code should perform a BFS search, and it should store the level of node i in the i^{th} entry of the list `level`. The function will return the variable `level` which is then used to write the output. You do **not** need to keep track of any information, other then `level`, about the execution of BFS , e.g. parents. **Please do not change the name of the variable `level`, as the starter code will pass its contents to the autograder.**

To better illustrate how this works, let's look at an example. Consider the following graph, which is based on Figure 3.2 in the book:



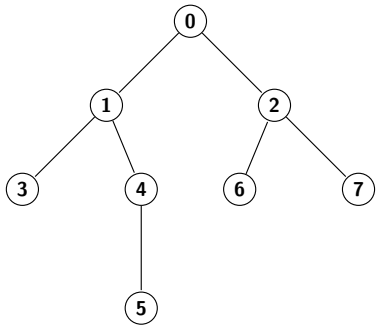
Say we would like to perform BFS starting from node 0. In this case, the variables in the code will be set as follows:

1. `N`: 8
2. `s`: 0

3. adj_list: [[1,2], [0,2,3,4], [0,1,4,6,7], [1,4], [1,2,3,5], [4], [2,7], [2,6]

So your algorithm might set the variable level to the following value, corresponding to the BFS tree below:

level = [0, 1, 1, 2, 2, 3, 2, 2]



We will provide you with a file you can use to test your code called input. This will set the variables N and adj_list to represent the same graph as in this example, but the start node s will be set to 6.

When you have finished writing your algorithm, you may submit your code to the autograder on Gradescope called **Programming Assignment 1, Part 1**. After you submit, the autograder will test your algorithm on several different graphs and indicate how many points you scored on each test. You are allowed to edit and resubmit your code as many times as you like before the due date. **You are not allowed to use non-standard Python or Java libraries in your code. Please do not submit code with "print" statements, as it will cause the autograder to malfunction. Also, note that the autograder will be checking for plagiarism, so please do not copy from each other or online resources.**

5.2 Part 2: Outbreak simulation

You will write the code to run your experiments based on the workflow described in section 4.2.

We provide you with the following (optional) starter code to use:

- outbreak.py

This code will accept a graph file name and an output file name in the command line, like this:

```
>> python outbreak.py grid.txt myfile.txt
```

In this example, the code will then load the data for the grid graph (grid.txt) and write the output to a file called myfile.txt. If you do not name your file at the command prompt, it will be named outbreak_output.txt.

First, you must copy and paste your BFS function solution into that you wrote for part 1 into outbreak.py. Then you may write your solution code for part 2 in the function model_outbreak (but feel free to write and use more functions if you would like). The model_outbreak function returns two lists of size N:

- prob_infect: the probability that each node ever gets infected

- avg_day: the average day of infection for each node (which can be the string `'inf'`, for infinity, if it is never infected)

Once you have populated these lists with the solution, the code will print your output file with the `writeOutput` function. Each line will represent a node, and the first N lines will have the probability of infection. **Then there will be a single blank line**, followed by N lines containing the average day of infection.

You may use the resulting output file to perform your analysis in section 4.2.