M345SC 2019 Homework 2

Due date/time: 1/3/2019, 23:59 GMT

Clarifications (Edited 25/2/19)

25/2/18:

- Part 1.2 i) If there are no feasible paths, the function should return an empty list.
- Part 2: There is a fourth state of "immune cells" so the initial conditions don't need to add to 1. Additionally, the fractions are with respect to the initial number of cells at a node, so an individual "fraction" can be larger than 1 for t>0. The expression for F_{ij} simplifies to 0/0 if node j is isolated ($q_j = 0$). You may assume such nodes are not present in the graph (it is also fine if you set $F_{ij} = 0$ for such cases).

24/2/19:

- Note that the sums in the equations in part 2 should run from 0 to N-1 as the nodes are numbered starting from 0.

23/2/19:

- There is a mistake in the function modelN provided in the template code, p2_template.py. The parameters g and k are in the wrong order. They should be in the same order as in model1, so please use:

```
a,theta0,theta1,g,k,tau=params
```

A corrected template code is here: | p2_template_v2.py

22/2/19:

- Please do not modify input from what is specified. If you would like to add additional optional input, check with the instructor. You may add additional functions as needed as long as they do not use prohibited external modules.
- Part 1.1: You may assume that it is possible to complete all tasks in a finite number of days. Note that the dependency list for a task contains **all** other tasks that must be completed before it can be started. So if task A depends on B, and B depends on C, then C will be in the dependency list for A.
- Part 1.2: You may assume that the adjacency list is "symmetric" in the sense that if A[i] contains (j,Lij), then A[j] will contain (i,Lij). The signal does not "split" or "recombine" at a junction. You can assume that it is routed to move along only one individual link between junctions.
- Part 2: The template codes incorrectly assign parameter values for "theta1" and "theta2" These should be changed to "theta0" and "theta1", respectively, to match the problem statement:

```
a,theta0,theta1,g,k,tau=params
```

This project consists of two parts – in the first, you will be developing/implementing algorithms on graphs, and in the second you will be simulating the propagation of an infection through an organism's body.

Getting Started: Template files have been added to the course repo in the hw2/ directory. You can also download the files directly from here: | p1_template.py | p2_template.py

Place these files in a folder, and create copies of the template files named *p1_dev.py* and *p2_dev.py* in this new directory. Ultimately, the final files that you submit should be titled *p1.py* and *p2.py*. Browse through the *dev* file;

there are a few function headers, and you will have to add code for each of these functions as described below. First, however, you should *add your name and 8-digit college id to the docstring at the top of each file.*

Part 1. (10 pts)

For each question in this part, you should develop a correct, efficient algorithm, implement the algorithm in Python, and discuss the running time and efficiency of your implementation. Your discussion should include an estimate of the asymptotic running time and a concise description of how you constructed this estimate. You may use *only* numpy and the collections module in your final submission for part 1. No other external modules (networkx, scipy, etc…) are allowed in your codes for this part.

1) You are designing an automated assembly process for a revolutionary new smartphone. The process consists of *N* tasks. *M* of these *N* tasks require at least one other task to have been completed before the task itself can be started. Each task requires one day to complete (and must be started and finished within a day). You have an unlimited number of multitalented robots at your disposal which can each complete one task per day. What is the shortest number of days needed to assemble the phone?

You are provided a *dependency list*, *L*, as input. This is a list containing *N* sublists. The sublist, $L[i]$ contains the indices of tasks that must be completed before task *i* can be started ($N - M$ of these sublists will be empty). Complete the function, *scheduler* in *p1_dev.py* so that it efficiently assigns a day to each task so that the total number of days to complete all tasks is minimized. Tasks are numbered from *0* to *N-1* and the function should return an N-element list whose *ith* element is an integer corresponding to the day on which task *i* is completed. For example, if *M=0*, then you should return a list with *N* zeros. You may assume that there are no pairs of tasks where each requires the other to have been previously completed and that $N > M$. Add a discussion of your implementation to the docstring of your function.

2) Consider the propagation of a signal with initial amplitude, a_0, through a telecommunication network with *N* junctions (junctions are numbered from *0* to *N-1*). As a signal propagates between two connected junctions, *i* and *j*, it experiences a loss characterized by L_{ij}. If the amplitude at junction *i* is *a*, the amplitude at junction *j* is $L_{ij}a$. Note that $0 \leq L_{ij} < 1$ and $a_0 > 0$. The signal at a junction can be boosted back to a_0 provided that its amplitude when it reaches the junction exceeds a threshold: $a \geq a_{min}$ with $a_{min} > 0$ a specified threshold that is the same for each junction. If the signal amplitude falls below this threshold when it reaches a junction, it is removed from the network and is **not** considered to have successfully reached the junction.

Network data is provided via an n-element adjacency list, *A*. The *ith* element of *A* is a list containing two-element tuples of the form (j, L_{ij}). if A[3] = [(4,0.5),(0,0.25)] this indicates that junction 3 has connections to two other nodes, junctions 4 and 0, and that $L_{3,4} = 0.5$ and $L_{3,0} = 0.25$. The network is undirected, so $L_{ij} = L_{ji}$.

i) Develop and implement an efficient algorithm to determine if a signal with initial amplitude, a_0, can successfully propagate from junction J_1 to junction J_2. Here, $a_0, J_1, J_2, A, a_{min}$ are all provided as input. Complete the function *findPath* in *p1_dev.py* so that it finds one feasible path (if such a path exists) and returns the path in a list containing a sequence of integers corresponding to the sequence of junctions that that the signal passes through. So, if there is a feasible path between $J_1 = 5$ and $J_2 = 12$ via junctions 3 and 7 (in that order), the function should return [5,3,7,12]. Add a discussion of your implementation to the docstring of your function.

ii) Now, develop an efficient algorithm to determine the minimum a_0 needed for a signal to successfully propagate from junction J_1 to J_2. A, J_1, J_2, a_{min} are all provided as input. Complete the function *a0min* in *p1_dev.py* so that it finds both $a_{0,min}$ and a feasible path from J_1 to J_2 with $a_0 = a_{0,min}$ (if a path exists). The function should return both $a_{0,min}$ and the computed path (see the function documentation). If no feasible path exists for any a_0, return $a_{0,min} = -1$ and an empty list for the path. Add a discussion of your implementation to the docstring of your function.

Notes for part 1: 1) It is generally up to you to decide if the running time for your algorithm is sufficiently small, however for question 2 ii), you are not expected to construct a linear time algorithm.

2) You are not allowed to use heapq, however, if you determine that the best approach to a problem requires a binary heap, you should choose this approach, implement it without a heap, and then discuss the benefits of the heap (and its influence on the asymptotic running time) in your analysis.

Part 2. (10 pts)

In Part 2, you will develop code to simulate the the spread of an infectious agent through an organism. The organism is modeled as an anatomical network with *N* nodes, and each node corresponds to a very large number of cells. Cells can be transported between some nodes, and we assume that there is a tendency for moving cells to flow towards highly-connected nodes. Three variables characterize the state of a node: S_j, I_j, and V_j which correspond to the fraction of cells at node *j* which are: Spreaders, Infected (but not spreaders), and healthy but Vulnerable to the agent. Our (crude) model for this problem is:

$$\frac{dS_i}{dt} = \alpha I_i - (\gamma + \kappa) S_i + \sum_{j=0}^{N-1} (F_{ij} S_j - F_{ji} S_i)$$

$$\frac{dI_i}{dt} = \theta S_i V_i - (\kappa + \alpha) I_i + \sum_{j=0}^{N-1} (F_{ij} I_j - F_{ji} I_i)$$

$$\frac{dV_i}{dt} = \kappa(1 - V_i) - \theta S_i V_i + \sum_{j=0}^{N-1} (F_{ij} V_j - F_{ji} V_i)$$

$$\theta = \theta_0 + \theta_1 (1 - sin(2\pi t))$$

The summation terms on the RHS control the transport between nodes while the other terms on the RHS dictate the dynamics within a node. The *flux matrix*, F_{ij} is defined as $F_{ij} = \tau \frac{q_i A_{ij}}{\sum_{k=0}^{N-1} q_k A_{kj}}$ where A_{ij} is the adjacency matrix of the network (which is unweighted and undirected), q_i is the degree of node *i*, τ is a model parameter, and F_{ij}/τ corresponds to the fraction of cells moving from *j* which are going to adjacent node, *i*. Note that $\sum_{i=0}^{N-1} F_{ij} = \tau$.

The parameter α controls conversion of infected cells to spreaders, θ controls conversion of vulnerable cells into infected cells, γ controls the recovery rate of spreaders, κ controls the conversion of all non-vulnerable cells to vulnerable. For questions 1 and 2 below, you can simply fix these parameters as: $\alpha = 50$, $\theta_0 = 80$, $\theta_1 = 105$, $\gamma = 71$, $\kappa = 1.0$.

The initial condition should correspond to only one specified node being infected. All other nodes should have *V=1*, *I=0*, *S=0*.

1. First, consider the case where $\tau = 0$. Complete the function, *model1*, so that it computes a numerical solution for this model on a network provided as input in the form of a networkx graph. The timespan of the simulation and other model parameters are also provided as input – see the documentation in the template file. The node which should be initially infected is also specified as input, and the initial condition for this node should be, *(V,I,S) = (0.1,0.05,0.05)*. The function should return an array containing *S* for the initially-infected node at each time step (including the initial condition). When input parameter *display* is True, the function should also create a figure of this *S* vs time.

2. Now, complete *modelN* so that it simulates the model with finite positive τ. The input is the same as in *model1* with the exception that when *display=True*, two figures should be generated: 1) a figure of $< S(t) >$ where <f> represents an average over the *N* nodes of the network and a figure of $< (S(t) - < S(t) >)^2 >$.

i) Complete the function, *RHS*, so that it computes and returns the right-hand side of the model equations (see the function docstring). You should assume that RHS will be called a very large number of times within one call to modelN and construct your code accordingly. You should also design your code for very large networks, say, *N=1e4* or larger, though it is recommended that you develop and test your code with much smaller graphs. Construct an estimate of the number of operations required to compute $dS_i/dt, i = 0, \ldots, N-1$ in one call to RHS. Add this estimate to the docstring of *RHS* along with a concise explanation of how it was constructed.

ii) Add the needed code below *RHS* to 1) simulate the model for *Nt* time steps from *t=0* to *t=tf* with the initial condition the same as in *model1*, 2) display the mean and variance of *S* when required, and 3) return the mean and variance of S.

3. Consider the infection model with $\kappa = \alpha = \gamma = \theta_1 = 0$. Investigate the similarities and differences between the resulting dynamics and linear diffusion on Barabasi-Albert graphs. You should design your own approach to the problem, and identify 1-3 key points, carefully discuss them, and produce a few figures illustrating numerical results related to the key points. Save and submit the figures with your codes (with names fig1.png, fig2.png, ...) Add code for generating your figures along with your discussion to the function, *diffusion*. It is sufficient to consider B-A graphs with *n=100* and *m=5* (see networkX documentation). Add code in the *name==main* portion of the code to call *diffusion* and generate the figures you are submitting with your assignment (the figures do not have to be identical due to the randomness of the B-A model).

Note for Part 2: You may use numpy, scipy, matplotlib, and networkX for part2. Do not use any other modules without the instructor's permission.

Further Notes:

1. Marking will consider both the correctness and efficiency of your code as well as the soundness of your analysis.

2. All figures created by your code should be well-made and properly labeled. The title of each figure should include your name and the name of the function which created it.

3. In order to assign partial credit, markers must be able to understand how your code is organized and what you are trying to do.

4. You are allowed to discuss general aspects of Python and the problem statement with others, however you should not discuss your particular implementations with other students or show your code to other students.

5. Please be sensible with the time you allocate to Q2.3, an exhaustive investigation of the problem is not expected.

Submitting the assignment

To submit the assignment for assessment, go to the course blackboard page, and find the link for "Homework 2" Click on this link and then click on "Write Submission" and add the statement: "This is all my own unaided work unless stated otherwise."

Click on "Browse My Computer" and upload your final files, *p1.py*, *p2.py*, *fig1.png*, *fig2.png*, etc... . Finally, click "Submit".