**CS 260 Final Project [100 points]**

Social networks have become prominent on peoples’ minds in no small part because of online social networking sites such as facebook and twitter. Of course, social networks have in fact been around as long as humans have been, well, social. And, yes, even engineers have social networks.

In marketing, social networks play a special role: everyone knows that when you convince someone to buy a product, you may well have also convinced some of their friends to buy it. This phenomenon is fundamentally connected to a well-known observation in social psychology: we often do stuff that our friends do. The general principle of friends influencing friends does not stop with buying things. We influence each other in our ideas and beliefs as well.

Back to the marketing example. If you are a marketing company, and you happen to know what a social network looks like (at least crudely), how can you take advantage of that? If you have a product to sell, and limited time, one idea is to go after people who can influence the most others through their direct social network contacts. In other words, you want to market your product to a K individuals who (if they buy it, of course) will jointly expose it to the most others. But how do you identify these K individuals? That, it turns out, is a challenging combinatorial optimization problem. And this is the problem you will be solving in your final project.

Actually, you will solve a “stylized” (simplified) version of this problem. It goes as follows. You have a graph (a social network; nodes on this graph correspond to people, links to personal friendships). **You wish to choose a set of K nodes in this graph so that the total number of graph neighbors of these nodes is maximized over all possible subsets of K nodes on the graph**. Don’t be fooled: the total number of neighbors of K nodes is not the same as the sum of the numbers of friends each node has (there will usually be overlap in friends among individuals).

In this project, you will be provided a python code base consisting of the network implementation. Your mission will be to implement an “agent” who computes a set of K nodes (specified as “budget”) that they wish to “seed” (that is, to target). The code base will then automatically compute both the time your implementation takes to run, and the utility (number of neighbors of your chosen nodes), all averaged over 10 seeded runs. Here’s what the code base contains:

Root directory:

**runSimulations.py**: the driver (to run your code, just type python runSimulations.py)

**util.py**: don’t worry about it; it’s just some utility functions that help us read the params.conf configuration file

network directory (package):

**node.py**: implementation of a node in the network

**network.py**: implementation of the network class, including the graph generator. Note that nodes in this network are going to be referred to by their index, which runs from 0 to numNodes – 1. You may find getNeighbors(nodeIndex), size(), and degree(nodeIndex) methods helpful. getNeighbors(nodeIndex) will return the list of neighbors of the specified node (index in the network class). Size() returns the number of nodes in the network. Degree(nodeIndex) returns the number of neighbors of the specified node.

cascade directory (package):

Don’t worry about it.

agents directory (package):

**agent.py**: a superclass for the agent that you will implement

**myagent.py**: your agent implementation. **This should be the only class you modify**. In particular, you will need to implement the method selectNodes(). You can write any auxiliary methods/classes inside this file that you wish.

Now, here are the tasks.

**Task 1: ALL STUDENTS. Due: 12/1/2014, NOON. Submit myagent.py ONLY on OAK.**

Unpack the provided “project.zip” file. Inside you will find a directory structure with the code base above. It will have the file params.conf. THIS IS THE CONFIGURATION THAT WILL BE USED TO EVALUATE YOUR CODE. If you wish to play with it, be sure to test your code using the provided conf file. In particular, you will be tested on a network consisting of 50 nodes, and your budget (K in the above narrative) will be 4; that is, you will be choosing 4 out of 50 nodes to target.

*Requirements*

Your implementation must satisfy two requirements, as evaluated on a computer in the FGH 201 lab:

1. Your average running time should not exceed 0.6 seconds. Yes, that’s under 1 second!
2. Your average utility must be 32.6.

Your final grade will be calculated as follows. Suppose that your code takes t seconds on average and has average utility of U. Your grade will be:

G = 100 \* e-0.02(max{t,0.6} – 0.6) – 0.2(32.6 - U)

So, if your code runs in 0.6 seconds and gets utility of 32.6, you get 100. If it runs in 1 second and gets optimal utility, you get 99. If it runs in 10 seconds, and returns an optimal solution, you get 83. If it runs in 1 second and returns a utility of 30, you get 59. You get the idea.

**Task 2: REQUIRED FOR GRADUATE STUDENTS. OPTIONAL FOR UNDERGRADUATES (2% bonus if you win the competition). DUE: 12/3/14, NOON. Submit myagent.py ONLY on OAK. *Late submission will forfeit their spot in the competition.***

In this task, you have to deal with a much larger network, and a larger set of seeded individuals. In particular, you will find paramsGrad.conf which has the following parameters: numNodes = 800, budget = 10. Your time constraint is now 11 seconds, and utility target is 630. Grade calculation will be slightly modified:

G = 100 \* e-0.02(max{t,11} – 11) – 0.02(630 - U)

*Competition*

There will be two separate competitions: one for graduate students, one for undergraduates. The scores for the competition will be computed as follows:

S = e-0.02t +0.01U

The winner of each competition will receive a 2% bonus for the course.