COMS 6998-2 Social Networks Fall '14

Data Challenge 1: Finding your way in a Geo-Social Network

We have learned in class that structures play a very special role in social networks; they do not systematically arranged all connections that are formed, but they significantly influence which ones are more likely to hold statistically. The goal of this first contest is to give you a chance to leverage properties of a real social network to build an algorithm solving a graph exploration problem with minimum cost.

Let us take an example: If you live in a place for a long time, you are likely to have lots of friends who also live in same place. This fact is helpful when we want to predict how a social graph develops in the vicinity of a given individual. Research has shown that geographic distance does affect social structure. But does this inform the design of new algorithms? Or, to put it differently, are you able to find a model for this geo-social correlation and use it to find ways to explore social graph effectively?
Here your goal will be to find a feasible path to a destination using a minimum of local queries, a problem related to Milgram routing.

Motivating scenario

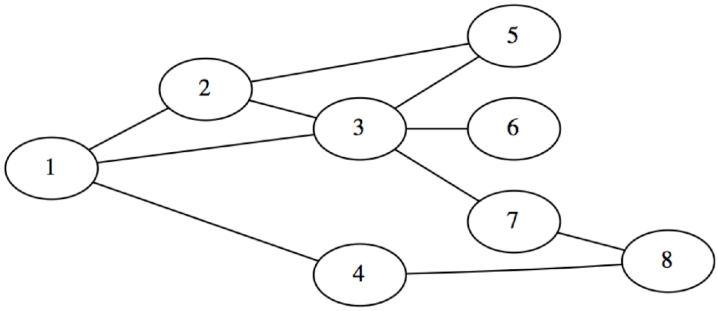
You, an active social network user, want to get connected to a target person through the network. Imagine this person is the CTO of a new start-up you heard about, and you have a particular idea to pitch to her. Most likely, a direct email from you (an unknown unsolicited stranger) would be ignored, even if you insist, so your goal is to find a chain of acquaintance through people who know each other, to convince each member of this chain and in the end reach that person.

At an abstract level, finding this chain appears relatively simple:

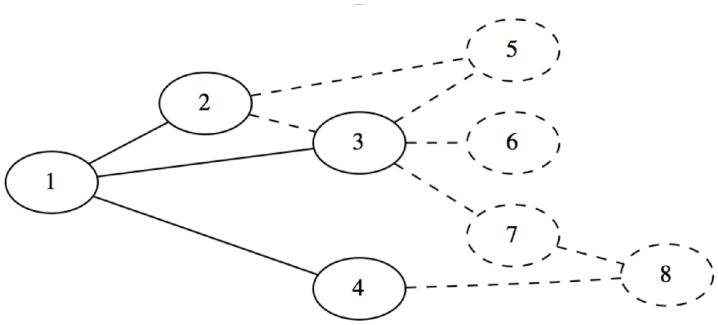
1. Begin with your friends. List them as known persons.
2. Pick the one who you think is the closest to the target from all known person, spend time convincing this person, asking him or her to disclose to you her or his list of friends. Then add this person's friends to your known persons list.
3. Repeat 2 until the target person is in the known persons list.

So, basically what you will do is that you choose one of your friends that you think can get you closer to the target, and then check his/her friend list. If you find someone in the list that you think is closer to the target, you go to check that person's friend list. You do this repeatedly until you find a path to the target person. Remember each query takes some resource (time to contact the person, convince him or her, perhaps even by providing some reward in case the project is successful), so you would like to minimize those.

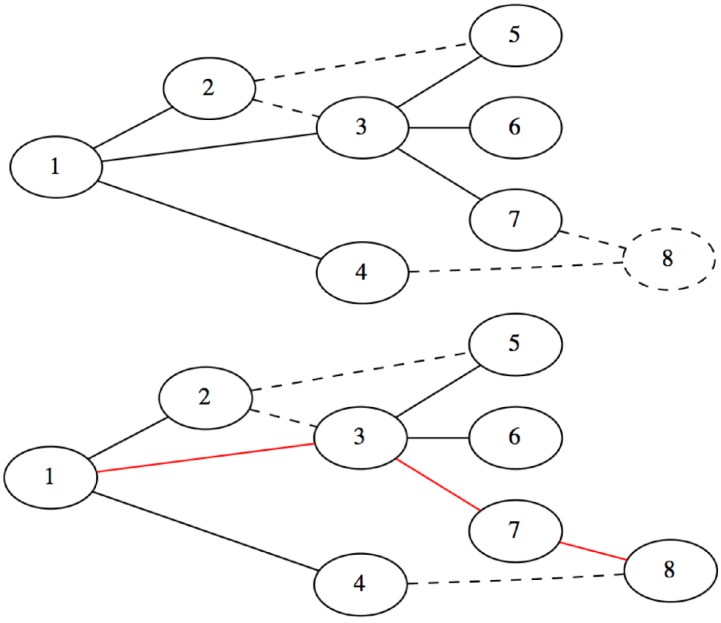
Let us illustrate this on an example. The following social graph contains you labeled as 1 and the target person in this case with label 8.



At the beginning, 2, 3 and 4 are your friends. You don't know the rest of the network.



You think 3 or 3's friend is more likely to know 8. So you go check 3's friend list and found 5,6 and 7



Now you think 7 or 7's friend is more likely to know 8, so you go check 7's friend list, and, bingo, you find 8. The path is 1->3->7->8

As a result of this exploration, you can get connected to 8 with the help from 3

and 7. Only two intermediaries were necessary.

Deciding whom to explore is the key aspect of this process. For the previous example, if you choose 4 instead of 3 or 7, you will get a shorter path 1->4->8.

This example seems puzzling as no information in advance differentiate 3 from 4 as one being *closer* to the target. This is where the most important aspect of this data challenge comes into play: you do have **additional information**. In this case, we assume that all users are members of a popular social network where you can publicly "check-ins" to venues (yes, what a strange idea to do that, but let's assume it for the sake of this challenge, and that everyone accepted this service's EULA). As an example, using the data you immediately found that 4 and 8 both checked-in at a coffee shop almost at the same time, while 3 and 8 seems to be far away from each other. Knowing this, you will think that 4 is closer to 8 than 3, an important hint to find a single intermediary to connect to the CTO.

Task Description and format

In this challenge, you will be given:

- Two numbers, standing for source ID and target ID.
- A file that contains lines of [time latitude longitude userID] tuples, representing check-in data.
- A query server that responses to friend list query. You will ask the server for a user's friend list.

There will be two main tasks (see organization of the contest below):

1. Find a path from the source user to the target user in the social graph with the minimum *number of queries* made to the server.
2. Find a path from the source user to the target user in the social graph with the shortest geographic movement. Here, we take as a convention that user's position is defined as his/her **last** check-in location. If a user has no check-in, he's forbidden for this task.

There are two data sets: a training data set and a testing data set. Both of them contain a social graph and corresponding check-in data, and the two sets have same distribution. The latter one will be used in a 24-hour competition, in which you will run your code to solve specific test cases.

The server is now running with the social graph of the training set, and you're given corresponding check-in data. To help you tune your model, the social graph of the training set is also given to you. However in the competition you will have only the check-in data of the testing set. The social graph will **NOT** be given and querying the server will be the only way to learn the structure of it.

Server Interface

The server accepts HTTP request and returns JSON strings. The available operations are,

- Start a test case
  Address: http://vienna.clic.cs.columbia.edu:8008/start
  Example Query: Request the address directly
  Example Return:
  {"token": "4605faf0432211e49754005056bb1bb1", "problem": ["7572", "4851"]}
  Where token will be used in other queries of this test case. Tokens expire in roughly one hour. The example test case is finding a route from 7572 to 4851. In the competition it will also return whether this test case is for task 1 or task 2. (There will be another document describing the rules and the details of the competition. The interface will be slightly different, but it will not need massive code change)
- Query a user's friend list
  Address:
  http://vienna.clic.cs.columbia.edu:8008/neighbors?token=[token]&node=[userID]
  Example Query:
  http://vienna.clic.cs.columbia.edu:8008/neighbors?token=4605faf0432211e49754005056bb1bb1&node=7572
  Example Return:
  {"neighbors": [{"214": 311}, {"34651": 9}, {"3618": 40}, {"9451": 13}, {"37015": 2}, {"6620": 75}, {"26902": 9}, {"37009": 1}, {"16686": 4}, {"8715": 76}, {"9405": 36}, {"6320": 21}, {"34689": 3}, {"23741": 6}, {"35612": 4}, {"773": 130}, {"9847": 17}, {"17666": 18}, {"7122": 33}, {"7169": 13}, {"7084": 108}, {"37010": 1}, {"37011": 1}, {"37012": 1}, {"37013": 1}, {"37014": 1}, {"12577": 86}, {"7583": 25}, {"6319": 21}, {"4851": 22}]}
  Where neighbors contains <userID: degree> pairs. Degrees of friends are given to you to help you make decision.
  The server will count how many queries you made. The number of queries for each test is limited to 100 queries. Once you reached this limit, the server will return {"error":"Step exceeds limit"}
- Commit a path
  Address: http://vienna.clic.cs.columbia.edu:8008/commit?token=[token], with POST parameter path=xxx,yyy,zzz,....
  Example Query:
  http://vienna.clic.cs.columbia.edu:8008/commit?token=4605faf0432211e49754005056bb1bb1, POST parameter path=7572,4851
  Example Return:
  {"success":1,"queries":1,"cost":123.45,"target_distance":0}
  If your path is valid, success will be 1, otherwise it will be 0. It returns you how many queries you have made and how long in geographic length is your path. The target_distance is the geo-distance from the final user of your path to the target user, it will be non-zero if and only if success=0. The token will be invalidated once you commit a path.

If you don't know how to request and parse json objects from a HTTP server, check out following libraries. Note, you don't have to learn them in depth; simply reading some examples and enabling yourself to call the server and parse the return is enough.

- C/C++: libcurl or Asio HTTP Client + rapidjson or PropertyTree JSON
- Java: Apache HttpClient + json-simple or org.json
- Python: requests
- Matlab: JSON Parser (For HTTP requests use Matlab built-in function urlread)
- R: RCurl + RJSONIO

Organization of the contest and grading

The grading of this challenge is organized in three phases:

- Phase I: Provide a correct answer to the challenge (i.e. write something that works at least for the simplest test case, and be able to commit some correct results in the competition)

- Phase II: Results in the competition for task 1: # of queries made if a path is found within query limit. Otherwise, length of shortest path from nodes that has returned by the server (known persons) to the target node.

- Phase III: Results in the competition for task 2: Geo-length of committed path (cost) if a valid path is found within query limit. Otherwise, target_distance.

In terms of dates,

- We will answer **all** questions from now until October 8th. After which we will **not** answer **any** question related to phase I (formatting, installation, data access etc.). So plan either to be early or, if you like to be last minute, you'll be on your own. Questions and clarifications specific to Phase II and Phase III and how to optimize your code will still be answered, but remember that TAs and office hours become very busy close to the end of the competition.
- All submissions for all Phase will have to be entered before October 15th 11:55pm EDT.