Lab Exercise 12: Testing the small world hypothesis

Duration: 3 sessions. **This is an examinable lab: an examination question will be based on the material of this laboratory exercise.**

For this assignment, you can only get full credit if you do not use code from outside sources. You can get partial credit by using properly attributed code from outside sources (see marking scheme for details).

Aims

To explore algorithms for finding short paths in graphs. To implement a shortest path algorithm. To investigate heuristic path finding algorithms.

Learning outcomes

On successful completion of this exercise, a student will:

- Have become acquainted with two methods for solving the all-pairs shortest path problem in graphs.
- Used complexity arguments to show which of these two is more efficient for a particular set of data.
- Have implemented Dijkstra's algorithm, and used it to test the small world hypothesis on social network data.
- Have implemented an approximate short path finding algorithm using local information, applied it to the same data and compared the results.

Summary

This lab is about finding shortest paths in a particular set of graphs representing network of friends on Facebook.

- You will understand what the "small world" hypothesis says about shortest paths on these graphs.
- You will use complexity arguments to show which is the most efficient algorithm to find short paths on these graphs.
- You will implement that algorithm.
- You will use that algorithm to test properties of path lengths on these graphs.
- You will also implement and test an algorithm for finding paths based on simple heuristics, and compare the results.

Deadlines: The unextended deadline is the end of your third scheduled lab for this exercise. If you need it, if you attend the lab you will get an automatic extension to your next session of COMP26120 labs (you must use submit to prove you finished in time and get it marked at the start of your next scheduled lab, which may be a marking session). You can also get an extension for good reason e.g. medical problems.

Description

Introduction

We are all part of social networks. These are the networks of people we interact with socially: acquaintances, friends, colleagues, teachers, family, etc. We can think of these networks as graphs. Each individual is a node, and is connected to the people that individual personally knows. One property these social networks are thought to have is what is sometimes called the "*small-world property*". That any pair of nodes in this very large network is connected by a short path. It is often said that any two humans are connected by a path involving no more than five other people, so the small-world property is also referred to as "six degrees of separation".

The six degrees of separation notion comes from a set of experiments carried out by Stanley Milgram in the late 1960s, which attempted to investigate chains of acquaintances in the United States. In the experiment, Milgram sent several packages to 160 random people living in Omaha, Nebraska, asking them to give the package to a friend or acquaintance whom they thought would bring the package closer to a set final individual, a stockbroker living in Boston, Massachusetts, 1450 miles away. Many were not delivered, but those that were passed through on average about six hands from sender to receiver. The experiment itself has since been criticised, but the idea that social networks possess this special "small-world" property has persisted.

If Milgram's experiments are to be believed, social networks graphs must possess two properties. First, between any two nodes there must be at least one short path (the small-world property). Second, assuming that not all paths are short, there must be some information available locally at each node which suggests a "good" next step, i.e one that is part of a short path. (This is the information used by the participants in the experiment who to deliver the package to next in the chain.)

This lab comes in three parts. Parts 1 and 2 test the first property: are the paths short in particular social networks. Part 3 tests the second property: is there a simple way to find the next node in a short path.

The social network data

The social network data is found in `/opt/info/courses/COMP26120/problems/ex12/` (use this path). These are the networks of Facebook friends from two US universities, Caltech and University of Oklahoma. These are represented as graphs in the same format as the graphs from lab exercise 11, and are in the files `Caltech.gx` and `Oklahoma.gx`. The Caltech data contains 769 nodes and 33,312 edges (e.g. approximately 43 friends in the same university each). The Oklahoma data contains 17,425 nodes and 1,785,056 edges (approximately 102 friends each). In both cases, only the friends from within the same university are included. The data was collected on a particular date in September 2005.

In addition to the graph data, there are two other files: `Caltech.at` and `Oklahoma.at`. These contain information about each of the "nodes": their gender, their major (i.e. degree subject) and so forth. This will be useful in Part 3 of the lab.

It should be noted that a Facebook friends network may be very different from a real social network. Unfortunately, it is very difficult to get data on real social networks, because this information is not recorded in any one place.

Part 1: Background Investigation

This part is worth 5 marks in total

First, make sure you understand the small-world hypothesis. Copy into your `ex12` directory the file `LabReport.tex` from the usual `COMP26120/problems/ex13` directory (use this path). (Don't copy the data files. They are about 50M and may crunch your quota.) Write in the first section of the `LabReport.tex` file your statement of the small-world hypothesis and how you are going to test it (without describing the algorithms).

We need a way to find the shortest path between the nodes of a graph, and we are interested the shortest paths between all of the nodes. This is called the *all-pairs shortest path problem*. There are (at least) two algorithms to solve this: Dijkstra's algorithm and Floyd's algorithm (also called Floyd-Warshall's algorithm). For these graphs, Dijkstra's algorithm is more efficient. We would like you to learn about the complexity of these two algorithms and find out why Dijkstra's algorithm is more efficient for these graphs. Write the complexities of the two algorithms in the `LabReport.tex` document and the argument showing that Dijkstra's algorithm is more efficient for these graphs.

Part 2: Finding the the shortest paths

This part is worth 15 marks in total

Part 2a: Implementation of the shortest-path algorithm

You will need to implement the shortest path algorithm. This should be Dijkstra's algorithm. You will need to learn about the algorithm and the implement it. We are not giving you any code for this; you must implement it from scratch. However, the graphs should be represented in the same format as in Lab exercise 11, so you can use the same code to read in the data files.

Dijkstra's algorithm requires a priority-first search. It contains a graph traversal algorithm which is similar to those you may have implemented in lab exercise 11, except you don't pull from your search list the most recently added (depth-first search) or the one which has been longest in the list (breadth-first search), but the one which is closest in distance to the starting node. Thus, you need to implement a priority queue.

You can implement a priority queue inefficiently, by using a linear search to find the best item each time you pull from the queue. You will get more points if you implement this in one of the efficient ways. For example, it can be implemented as a heap. The relevant section in the textbook, is section 2.4, and particularly 2.4.3. Finding shortest paths in graphs is discussed in Chapter 7 of the textbook. (If you use code from outside sources for this, you can only get partial credit for this part. See marking scheme for details.)

After implementing the priority queue, you can complete the implementation of Dijkstra's algorithm. **Note:** If you name your executable `dijkstra`, you may get a conflict with `/usr/bin/dijkstra`. However you organize your code, to submit it, put it all in a file called `part1.c`.

Part 2b: Test for the small-world property

Use your algorithm to measure properties of the paths in the two social networks. The graphs are represented in the same format as in Lab exercise 11, so you can use the code from there to read in the graphs. Write the results of these experiments and your conclusions in the `LabReport.tex` file. Are these small-world networks?

Part 3: Approximate path finding and heuristics

This part is worth 10 marks in total

Once an all-pairs shortest path algorithm is run, it is possible to give each node a look-up table of the next node in the shortest path to any target node (a routing table). However, sometimes it is not feasible to run an algorithm which explores the entire graph to generate this table. Certainly the participants in Milgram's experiment could not have implemented Dijkstra's algorithm, which requires an investigation of the entire network before they decide who to give the parcel to. These participants could only know about their friends and acquaintances, and had to choose which of those to give it to based on some quantity used to determine which of those was most likely to be the next step on the shortest path, or at least a short path. We will refer to such a quantity as a heuristic. The heuristic they might have used may have been geographical (give it to someone who has some connection to somewhere close to Boston), or social (give it to someone who knows a lot of people).

Using heuristics to find approximations to shortest paths may be useful when the graph is too big to implement Dijkstra or Floyd, or when it is changing too rapidly for there to be up-to-date information about the entire network at any node. In this part, you will investigate whether such heuristics find short paths in these networks.

We will call this approach "heuristic path following". It works like this. You choose some heuristic. To find a path to a target, each node chooses the next node from its outlist based on this heuristic. For example, for the Facebook data, the heuristic could be, choose the node among its friends with the most friends (the friendship relation is mutual, so out-degree is the same as in-degree in these graphs). Using this heuristic, the algorithm works as follows, to find the path from SOURCE to TARGET:

- CURRENT ← SOURCE;
- while (TARGET not in CURRENT.OUTLIST) and (CURRENT.OUTLIST not empty)
    - add CURRENT to PATH
    - Let MAXOUT be the unvisited node in CURRENT.OUTLIST with largest out-degree
    - CURRENT ← MAXOUT;
- endwhile
- add CURRENT to PATH

Here CURRENT is the current node, and PATH stores the entire path. The choose of heuristic determines the how current is updated. Other possible heuristics could be pagerank score, or some measure of how similar the node is to the target node. This approach is clearly not guaranteed to find the shortest paths (why not?). Can a good heuristic be found so that it finds short paths?

Your task is to implement this algorithm and test to what extent it finds shortest paths. Implement the general scheme above and run it using ONE heuristic. The heuristic you choose may be (1) the maximum degree heuristic mentioned above, (2) a heuristic of your own devising, OR (3) a heuristic based on trying to find "people like the target" as explained in the next paragraph.

A similarity heuristic: A heuristic you might try is to choose the node which is most like the target node based on one or more attributes. There are files of attributes about the nodes in the student graphs. These contain the following information: whether undergraduate or graduate student, or faculty; gender, major subject, joint or minor subject (if applicable), dorm, year, and high school. (Missing data is coded 0.) These are in the files `Caltech.at` and `Oklahoma.at`. The format is

```
1 1 2 205 0 169 2006 15903
2 2 2 207 0 0 2005 3029
3 2 1 208 0 0 0 3699
4 1 2 228 0 169 2006 17763
5 1 2 204 206 0 2006 2790
6 2 2 228 0 169 2005 50029
```

The first number is the node number. They are listed in order. So, the first node is a student, female, major subject 205, no second subject, lives in dorm number 169, will graduate in 2006, and came from high school 15903. The second node is a graduate student, also a female, major subject 207, no minor, will graduate in 2005, and came from high school 3029. You add these attributes to graph and read these in. If the attributes of the target are known, can a short path be found simply by moving to the node which is most like the target?

Implement code to generate paths testing the heuristic of your choice. Put the code in a file called `part2.c`.

Compare the results of your heuristic method with the results of Part 2 (actual shortest paths) on the two data sets. Write the results and conclusions in the `LabReport.tex` document.

Marking Process

You must use `labprint` and `submit` as normal.

`labprint` and `submit` will look for: `LabReport.tex`, `part1.c` and `part2.c`

The marks are awarded as follows:

```
Part 1
   1 mark: Stating the small-world hypothesis.
   4 marks: Giving the argument why Dijkstra's is more efficient than
            Floyd's on these graphs.
Part 2
   7 marks: Implementing priority queue. 4 marks if you implement
            inefficient linear search. You can get at most 4 marks if you
            use code from an outside source.
   5 marks: Implementing the rest of Dijkstra's algorithm. 2 marks if
            you use code from an outside source.
   3 marks: Using the algorithm on the data to answer the question
            about the small-world property.
Part 3
   8 marks: Implementing a heuristic path-finder
   2 marks: Comparing the heuristic method with Dijkstra's. Evaluating the results on the two test sets.


Total 30 marks
```