Computational Optimization
Homework 5

Nedialko B. Dimitrov

By completing this homework assignment you will learn about:

- Working with network data, and network data structures
- Calling a solver within python
- Processing string data, plotting complex geographic results

In this homework, you will implement a poor-man's version of Google Maps. Many real-world problems can be represented as network optimization problems. This homework is just one example of such a problem, based on data from Austin. In addition to the above learning objectives, in this homework I will purposefully give you much less guidance on how to organize your code! This means, that you should spend a little time on how you want things organized, so that your code is easy to write, and your functions have the data that they need.

1 Build The Austin Road Network

Download the file **hw5_files.zip**. It contains data on Austin's streets. The data comes from the city of Austin website ftp://ftp.ci.austin.tx.us/GIS-Data/Regional/coa_gis.html. To make things easier for you, I combined the shapes lat/lon data with the metadata into a single file called **austin.csv**. In addition, I automatically looked up the addresses of several locations in Austin, and stored that data in **addresses.csv**.

Create a class that takes as input street data, such as **austin.csv**, and creates a network data structure for the city. The class should basically do the following:

- Each street should be an edge. Ends of streets, specified by lat/lon locations, should be nodes. Two way streets should give edges in both directions. One way streets should give an edge in only one direction. The column **ONE_WAY** in the data specifies the street direction, with a value of **'FT'** specifying that the street has the same direction as the shape, and **'TF'** specifying that the street has the reverse direction from the shape. Any other value is a two-way street. Hint: You will need to pull out the lat/lon data for the start and end points of every street using a regular expression and **df.kmlgeometry.str.extract**.
- We will use the class to find routes from one address in **addresses.csv** to another. To do this, the class needs to do the following:
  - Given a lat/lon pair, find the closest node to that point. This is because we need to map an address to some node on the network.

– Given a start node and a destination node, we need to produce the shortest path, in terms of driving time, between those nodes. I want you to implement this functionality with two methods **def getSPNetworkx(self, startnode, destnode)** that finds the shortest path using Dijkstra's algorithm implemented in networkx, and a second method **def getSPLP (self, startnode, destnode)** that finds the shortest path using an LP and Pyomo coupled with a solver. The two methods should be interchangeable, meaning they should function exactly the same in terms of their inputs and outputs.

2 Visualize, Solve Some Shortest Paths

Using Geoplotter, or any code you want, plot the street network and the addresses. Plot ETC as a green dot, and all the others as a red dot. You should produce a plot that looks something like this:

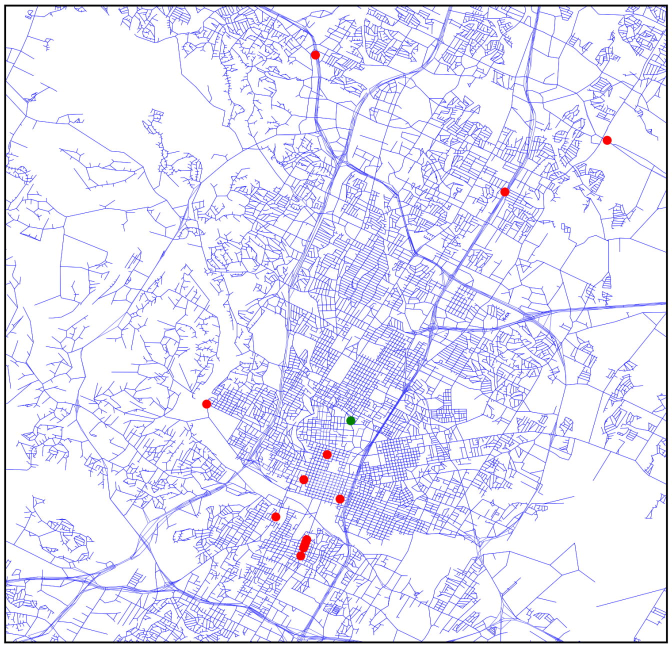
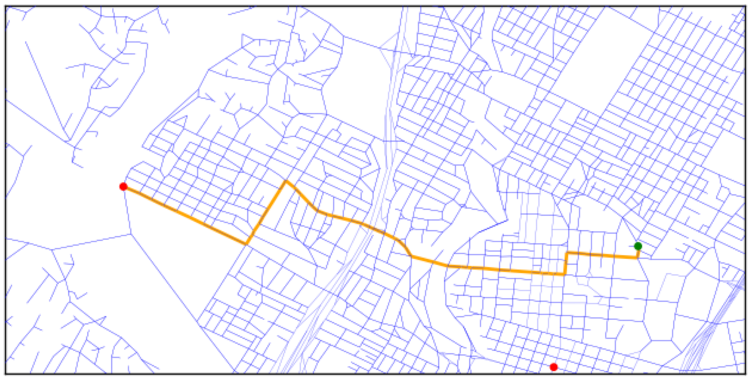


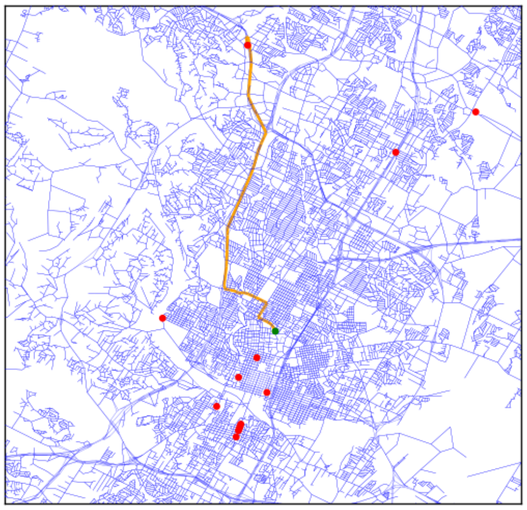
Figure 1: A zoomed-in version of the Austin network, with addresses. The zoom-in bounding box here is (-97.8526, 30.2147, -97.6264, 30.4323).

A key part of programming is reading other people's code to try to understand what it is doing. I have not taught you how to use Geoplotter to plot network data, however, it does have code to do this! But, you have to read it, to figure out what kind of data it expects as input. . . the data input specifies the locations of all the nodes, and the styles of all the nodes and edges.

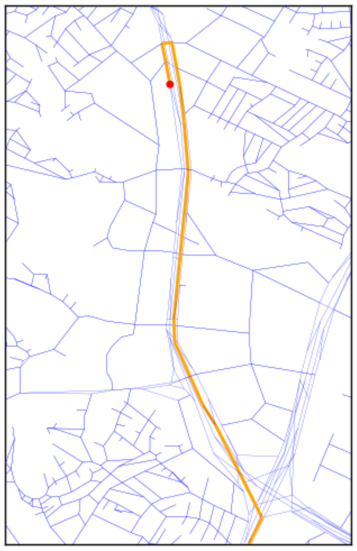
In addition to plotting the overall network, plot a couple of routes. Calculate one of these using your getSPNetworkx method, and calculate the other using your getSPCplex method. Have the first route be from ETC to Hula Hut, and the second one be from ETC to Rudy's. The plots you get out should look something like this:



(a) Route to Hula Hut



(b) Route to Rudy's



(c) Route to Rudy's, zoom-in

Figure 2: Routes calculated and displayed for the Austin network. We get real-world behavior, like the highway access road U-turn in 2(c) because of the one-way streets we created in the network.

3 Final Thoughts

What to turn in: For this assignment, turn in three things:

1. Your Python code in a file called **hw5.py**.
2. The plots:
   - The network overall

- Route to Hula Hut
- Route to Rudy's

3. A short (less than one minute) video explanation of your code called **hw5−explanation.avi**.

Key take-aways: From this assignment, you should take-away these things:

1. I gave you much less direction on this homework! I only said **what needs to be done** and not much on how to do it. Of course, in the process, you ran into a lot of bugs and you had to figure out organization and function on your own. But, that's really how a real-world programming situation works. The result should give you confidence in being able to get things done.

2. Now you know how to work with, and display network data. In my experience, even if the optimization we are doing is not inherently a network optimization like shortest path, the data can often be visualized as a network. Network problems are both extremely common and useful.

3. Finally, now you know how to do the basics of 1) Reading real-world data 2) Creating and solving an optimization 3) Interpreting the results in the real-world. These three steps are the core of essentially all operations research problems. Often we loop these three, refining the model in each loop, making the optimization problem and solution more meaningful. For example, if we had traffic data, we could re-execute the loop in this assignment, incorporating the traffic data into the mathematical model, giving better routes.