ECE 592 – Topics in Data Science

Project 2: Dijkstra's Algorithm Plans a North Carolina Trip

Due: October 9, 2019

1 Administrative Instructions

1. For any clarification or doubts, the TA Hangjin Liu (hliu25 AT ncsu DOT edu) is in charge of homeworks and projects. She should be your first point of contact on homework- and project-related issues.

2. The project can be submitted individually, in pairs, or triples.

3. You should submit electronically through Moodle by midnight the day that the homework is due.

4. Your report should describe any mathematical derivations, responses to questions, and results including any plots. Please justify your answers carefully, including providing interesting insights when relevant.

2 Data Retrieval and Pre-processing

(a) Download a North Carolina (NC) mileage chart at the following URL, `http://www.mileage-charts.com/chart.php?p=chart&a=NA&b=US&c=NC`. This chart will give you a 433×433 matrix, where each matrix element represents the distance between 433 cities in NC.

(b) Import this matrix into Matlab or Python. You can try importing it as a cell first, in order to keep track of city names corresponding to the rows and columns.

(c) Zero-out all diagonal entries of the imported matrix, because self-distances should be zero.

(d) In the matrix, replace all distances above some threshold, perhaps 20 miles, with ∞. This step constrains the driver to go between nearby cities directly. (Assuming your threshold is 20 miles, you will be left with 3,780 real-valued matrix entries.) This matrix represents your road map for NC.

3 Dijkstra's Algorithm

(a) From your NC road map, pick any two cities on approximately opposite sides of NC. For example, you could pick *Murphy* in the western part of the state and *Elizabeth City* in the eastern part; you may choose another pair of cities if you prefer. Please provide a Google Maps screen-shot for the Google-recommended path between these cities.

(b) Write code for Dijkstra's Algorithm in Matlab or Python. You can first test it on a small example to make sure that your algorithm works correctly. Feel free to (briefly) show these results as well.

(c) Your job now is to determine the shortest route between the two cities you selected in (a) above. Run your Dijkstra implementation, and show results.

(d) Select a few cities from your computed path, and overlay it with the Google-recommended path you showed in (a). Hopefully you will see that your algorithm provides a result similar to Google's.

(e) Repeat Steps (a)-(d) for another pair of cities. Again, the cities should be relatively far from each other.

Note: Your algorithm might get stuck due to the 20 mile threshold, which may significantly reduce the number of finite-length edges in the graph. To resolve this problem, feel free to either:

- Choose a different pair of cities (make sure that they are far enough apart).
- Re-compute the NC road map in Step (d) with a maximum hop distance of more than 20 miles (you can try 30 or 40 miles and re-run your code).

4 Analysis

(a) Verify the time complexity for Dijkstra's algorithm empirically. To do so, plot the execution times for your algorithm as a function of the number of nodes traversed; compare the results with the theoretical time complexity on the same plot.

(b) Read about the A^* search algorithm (Wikipedia page is a good reference) and implement it in Matlab or Python. Again, make sure to verify your code on a small example as done for Dijkstra's.

(c) The A^* algorithm is meant to be faster than Dijkstra's, but uses more memory. Please analyze and verify this time-memory trade-off by running A^* for calculating the shortest distance between a pair of cities you have already considered using Dijkstra's algorithm. Discuss your findings.