KINGS MAPS

“Real world” application



2022 A-Level Computer Science

By Oscar Pearce, 8196

Analysis:

A look into the travelling salesman problem

The travelling salesman problem (TSP) has been a classic problem for mathematicians and computer scientist since the 1930s. While there exists no efficient algorithm to solve real world networks, a heuristic approach leads to a solution in most cases. This problem involves finding a path that visits any finite number of locations at least once, before returning to the starting point. There exist two variations of the TSP, classical and practical. The classical approach does not allow for locations to be visited more than once, where the practical allows multiple visits to a single location.

Hailing from pathfinding and optimisation, the TSP has obvious real-world parallels. One such example would be, believe it or not, a travelling salesman. The example that I am using for my project is a postman working for amazon. Applying the TSP to this client will reduce the overall time spend driving, as the total distance travelled will be minimised. This will not only save time, but money for the client’s employer (I’m sure he needs it).

Interview:

* What software is currently in use for delivery?
* What additional feature would you find most useful?
* How do you view the routes you are taking?
* How often do you run the same, or very similar, route in a week?
* Are the routes produced by the existing software efficient?

An implementation of the TSP for this problem would ensure that the client takes only the most efficient route, reducing wasted fuel and saving the client time. My solution would also allow the client to save previously calculated routes as well as load them for repeated use. In order to abstract real world locations and roads to a model that can be computed, data needs to be extracted from the real world. This will be done through Google Maps distance matrix API, which takes an array of starting latitudes and longitude, returning the distance between each combination.

Locations in the real world will be abstracted to “nodes” with an associated latitude and longitude, so that they can be displayed on a map once calculation is complete. Roads in the real world are abstracted to an integer value, an “arc”, that can represent the time taken to travel along it or the distance to travel along it from start to end. Connections between nodes, arcs, can be represented through an adjacency matrix. This is a dictionary type structure that lists all a node’s connections, arcs, and the associated weight. An adjacency matrix defines a network, or graph. The TSP algorithm that I will implement involves performing many algorithms on a graph to find upper and lower bounds, and testing if they are optimal.

My aims for this project are:

* Produce an algorithm to solve the TSP for any adjacency matrix
  + Users must select a starting node
  + Must be fast
* Produce an adjacency matrix representative of a set of real-world points
  + Users will be able to click on an interactive map to add nodes
  + Users will be able to remove and change the names of nodes
  + Interacting with the map and nodes must be straight forward and feel smooth
* Produce an intuitive and responsive UI
  + Must be easy to use
  + Must be aesthetically pleasing
  + Animations are something to think about
* Save solutions for repeated use
  + These will be saved to a local JSON file that can be read by the program
  + Allow previous solutions to be shown on a map

A graph will be represented in my project as a class, with various methods that will perform an algorithm on the graph. Arcs will also be represented as a class, with attributes detailing the nodes connected by it, and an attribute defining its weight. A graph can have many arcs, however, is not dependent on them to be classed as a graph. The general process of creating a new solution will be as follows:

* User selects “new solution”
* User clicks on map a desired number of times, adding a node each time
* Latitude and Longitude of where the user clicked is stored
* User selects start node
* User selects “calculate”
* Google maps distance API queried with Lat/Lng of nodes
* Adjacency matrix created with information from distance API
* Adjacency matrix sent to back-end TSP solver
* Sequence of nodes to visit sent back to front-end
* Sequence displayed on map

Design:

A look into system design

The following diagram shows the data flow in the system, from user clicking to a finished solution:

## 

There will be three main areas in the application that the user can interact with: the landing page, the new solution page, and the previous solutions page. The following diagram shows the relationship between the three:

Graphical user interface, text, application, chat or text message

Description automatically generated

System flowchart indicating major stages in the flow of control:

