From HTML to PostGIS
(Project documentation)

Arnau Gàmez Montolio
Zhu Jiyi

21st January, 2019

0 Task

The problem we are asked and aim to solve is to create an application for visualizing a Vehicle Routing Problem solutions. More specifically, a Web UI that will let us introduce the data and will be able to send it to a server, that will handle the data and manage to solve the problem and send back the solutions for the Web UI to be able to represent it properly.

1 Description of the VRP problem

The Vehicle Routing Problem (VRP) is a generalization of the Traveling Salesman Problem (TSP). In a VRP, the goal is to find the optimal set of routes for a fleet of vehicles delivering goods or services to various locations.

For the problem, we are given a finite number of vehicles available and a central node (the "factory" or "depot") from where all vehicles will go out and come back after their route is completed.

What we want to accomplish here is to minimize the length of the longest single route for all vehicles.

2 General approach

As a user interface, we will use a WebUI made with usual web technologies. We will use JS alongside the HTML and CSS (with special use of canvas and JQuery) focusing on AJAX methods to make the HTTP requests to send and retrieve data to/from server.

We will use a python3 server that will handle the HTTP requests and deal with the data. It will also call the actual solving algorithm implementation (done in python too) that will get the data received, compute the solutions and return it back in a proper way so the server can return it as a JSON well-formatted response.

3 Implementation details

3.1 WebUI

The general structure of the WebUI is divided in a couple of headings followed by the canvas region where the data will be visualized and all the necessary input fields to add data on its left. The first node added (with ID = 0) will be the factory and will be of different color from all the following ones (red instead of black). Numbers indicating the ID of each node added will also be plotted into the canvas. We can see the information of each node in a nice dialog above each node when we click on them.

We can also load the nodes directly from a file on the server named *info_load.dat* and properly formatted. When we load a file, the nodes in there will be automatically drawn into the canvas, clearing out previous data. We can still add nodes on top of the loaded data without any problem and those will be taken into account for the solutions' computation.

When we press the "Solve" button, the required information (number of vehicles and all nodes' information as well) will be POSTed to the server that will properly store all that information in the file *info.dat* and then a GET API call will be made asking for the VRP solutions for the given data. As a response to the call, we will receive a solutions objects that will contain all necessary information to properly draw the edges and distances of the actual solutions that will be immediately drawn into the canvas.

The solutions will be drawn as edges of different color for each different vehicle (route) and showing the distance between each edge. It will appear a floating windows with the information of each vehicle route and the total route too.

The UI functionality can be found in the following files:

- index.html: basic structure.
- style.css: styling of the WebUI
- script.js: manages the logic and the API calls.

3.2 Server

As said before, the server is just a simple python3 server built on top of http.server module. It properly handles the API calls, takes the appropriate actions according to the type of the API call received and the endpoint where it has been called and constructs the response accordingly.

It parses the data of nodes received as JSON and stores it in files to be used afterwards to compute solutions or to load back data to the UI. It is also responsible of calling the actual solver algorithm implementation, retrieve the response and send it back.

The server implementation can be found in the following file:

- server.py

and has to be called as "python3 server.py" opening the server at localhost:1234

3.3 Solving

The solution's implementation is based on the ortools library by Google that gives us the ability to compute solutions to different routing problems. The ortools library is imported as a python module. First of all the data model is created. In particular, we need to create all the distances between nodes to be able to compute best solution. The distances are computed as euclidean distances and stored in a distance matrix.

Alongside the other needed data, we use the ortools capabilities in order to derive as much optimal solutions as possible with few constrains (using max distance for a single vehicle, for instance).

When the solutions are computed, they are printed in the server log and a response is constructed and properly returned (as the return value of the server called that called the solver) so the server can pack it as a JSON data response.

The algorithm implementation to compute the solutions can be found in the following file:

- vrp.py

As said before, the solutions sent back will let the UI to display the actual edges (routes) with distances of each vehicle of the VRP solutions we were looking for.

4 Appendices

4.1 File format

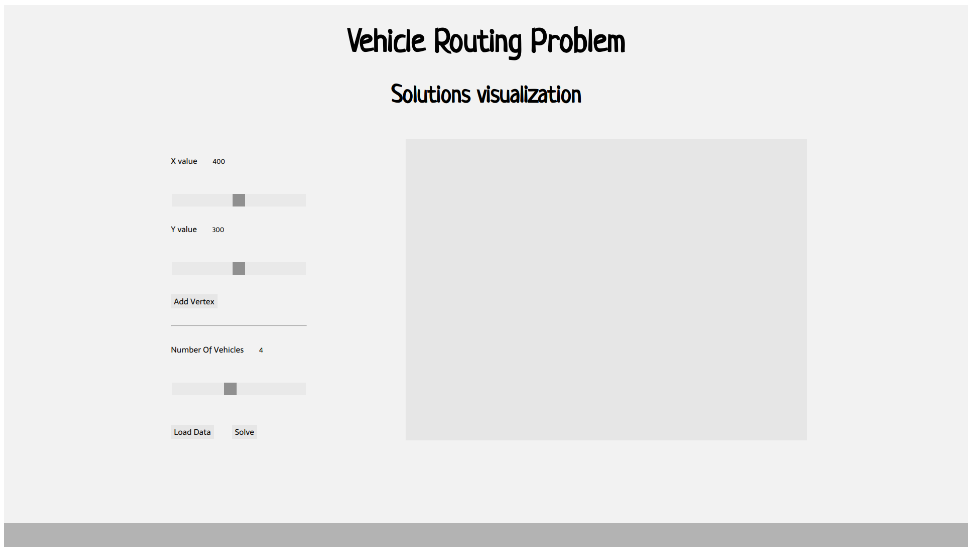
The file “info.dat” where information is saved in order to solve the problem as well as the file “info_load.dat” follow the same simple format that allows anyone to easily save and restore particular situations for the problem to be solved.

It is a plain-text file with the following structure:

```
N
{”id”:0,”X”:x,”Y”:y}
{”id”:m,”X”:x,”Y”:y}
...
```

That is, the first line is an integer number indicating the number of vehicles to use. The second line will be the depot node and it is required for it to be the first line and with id = 0. The following lines are each one for each remaining node, specifying the three elements that constitute a node (a number of the id different from 0, and the X, Y coordinates within their ranges).

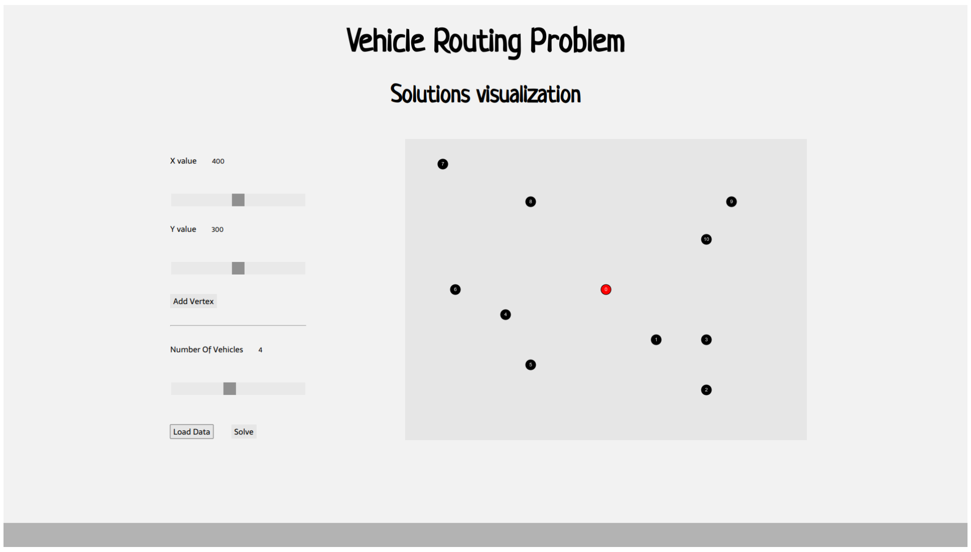
4.2 Screenshots



X value 400

Y value 300

Number Of Vehicles 4


Vehicle Routing Problem

Solutions visualization

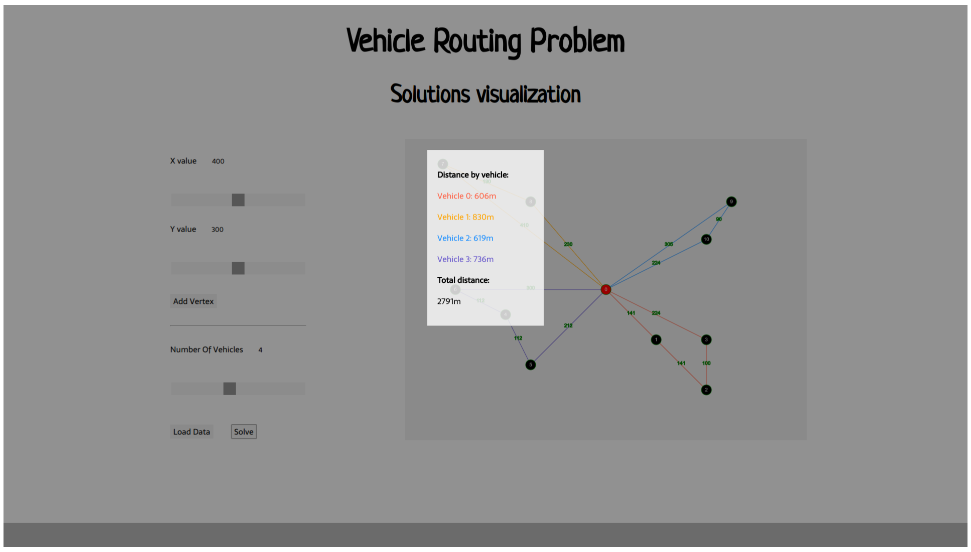
X value 400

Y value 300

Add Vertex

Number Of Vehicles 4

Load Data Solve


Vehicle Routing Problem

Solutions visualization

X value 400

Y value 300

Add Vertex

Number Of Vehicles 4

Load Data Solve

Distance by vehicle:

Vehicle 0: 606m

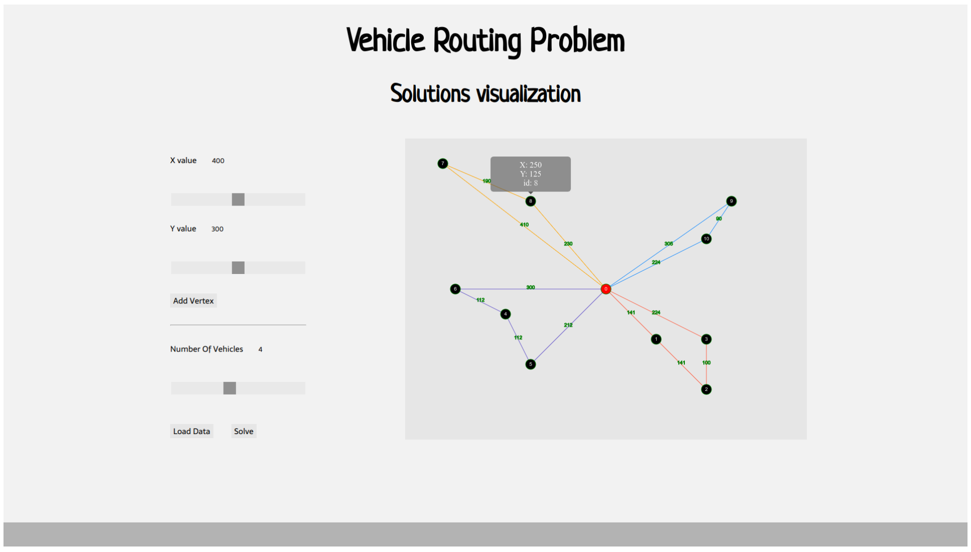
Vehicle 1: 830m

Vehicle 2: 619m

Vehicle 3: 736m

Total distance:

2791m


Vehicle Routing Problem

Solutions visualization

X value 400

Y value 300

Add Vertex

Number Of Vehicles 4

Load Data Solve

X: 250
Y: 125
id: 8