Progress Report 2

Leah A. Chrestien

July 26, 2017

0.1 Algorithm Summary

This report outlines the steps involved in determining the nodes forming the range boundary/polygon .

- All the coordinates forming nodes are assigned a unique key. This key acts as an identifier.
- A starting location is chosen, based on which the range will be determined.
- A 'Haversine' formula[1] is introduced which determines the geographic distance between any two coordinates. At any step, this Haversine function is called with four input parameters (latitude 1, longitude 1, latitude 2, longitude 2) and this function returns the geographic distance between the coordinates in miles.
- The value of range (in miles) is set. This enables the algorithm to select nodes at that particular distance from the chosen centre. The algorithm uses the Haversine function to determine these edge nodes. These edge nodes are the filtered nodes.
- A Double dimension array is introduced at this stage to store the distances between all the filtered nodes.
- To achieve a polygon having the minimum number of intersecting edges, it is important to output the coordinates in an ordered manner. The Travelling Salesman problem(using nearest neighbour approach) is implemented to ensure minimum distances between all the filtered nodes. This problem is NP-hard and there is still a finite possibility of the intersection of edges. One can decrease this possibility of intersection by increasing the filtered nodes (and therefore increasing the original number of nodes) and improving the range precision. The output from this step is an ordered set of filtered nodes. Section 0.2 shows one such evolution of the Travelling Salesman algorithm.
- The ordered set of filtered nodes from the last step is shown an a geoJSON output. This can be readily used in any map editor. Section 0.3 shows two such plots of the geoJSON output on a map.

0.2 A Routing algorithm: The Travelling Salesman

Given a set of cities and the distance between any two cities, the travelling salesman has to visit each one of the cities, starting from a chosen city and returning back to the same city. Given an additional constraint that

he cannot visit a city more than once, the challenge of the problem is to minimise the total length of his trip. In order to get a convex polygon with minimum or no intersections, this arrangement of coordinates becomes necessary for forming the boundary. Figure 1 shows a set of labels. The distance between any two labels is given by the Euclidean distance. Starting from label 1, the evolution of the Travelling Salesman problem is seen, as it passes each label only once and returns back to the original label.

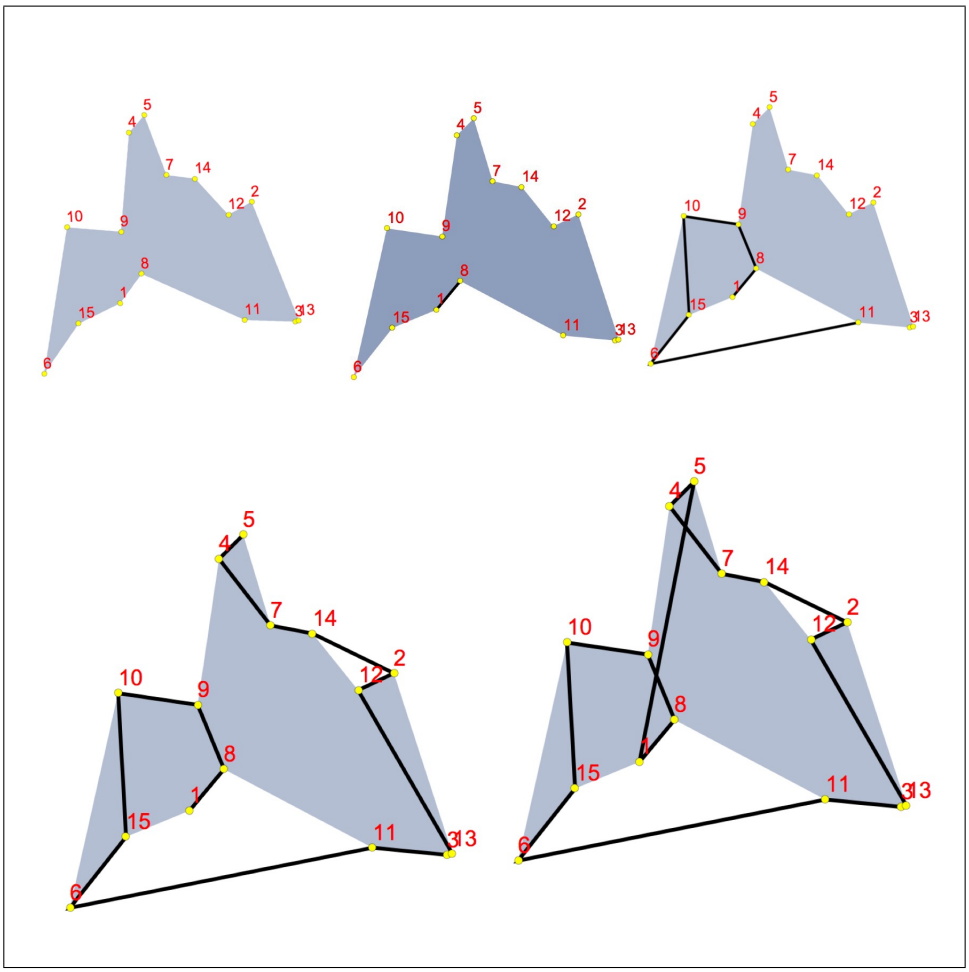
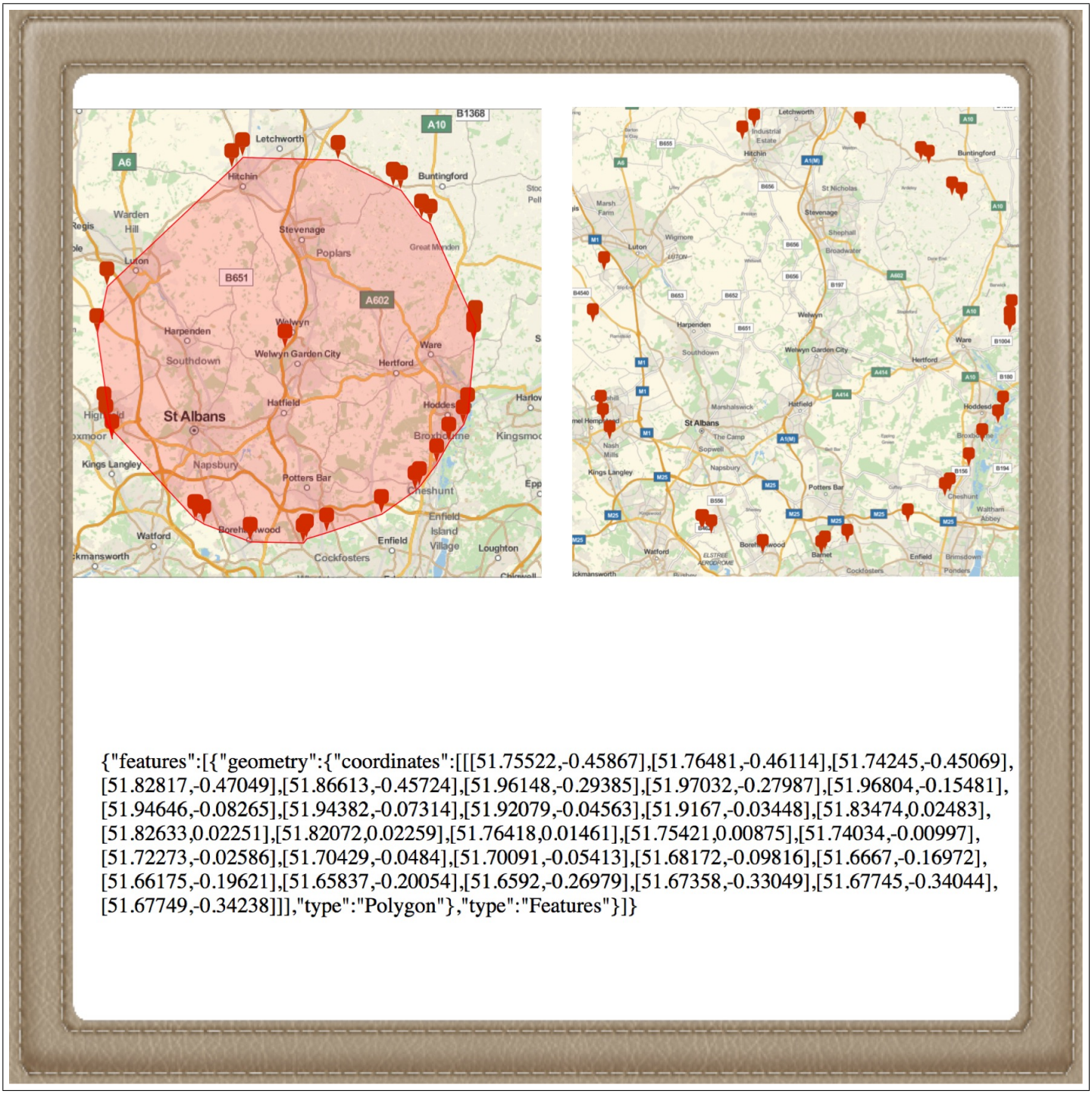


Figure 1: Evolution of the Travelling Salesman Algorithm.

0.3 Output of geoJSON

On executing the java project, the program outputs a set of ordered coordinates in the geoJSON format. The geoJSON output along with the plots showing the markers and the range are demonstrated in Figure 2 and Figure 3. No noticeable intersections are seen at the boundaries.



{"features":[{"geometry":{"coordinates":[[[51.75522,-0.45867],[51.76481,-0.46114],[51.74245,-0.45069],
[51.82817,-0.47049],[51.86613,-0.45724],[51.96148,-0.29385],[51.97032,-0.27987],[51.96804,-0.15481],
[51.94646,-0.08265],[51.94382,-0.07314],[51.92079,-0.04563],[51.9167,-0.03448],[51.83474,0.02483],
[51.82633,0.02251],[51.82072,0.02259],[51.76418,0.01461],[51.75421,0.00875],[51.74034,-0.00997],
[51.72273,-0.02586],[51.70429,-0.0484],[51.70091,-0.05413],[51.68172,-0.09816],[51.6667,-0.16972],
[51.66175,-0.19621],[51.65837,-0.20054],[51.6592,-0.26979],[51.67358,-0.33049],[51.67745,-0.34044],
[51.67749,-0.34238]]],"type":"Polygon"},"type":"Features"}]}

Figure 2: The nodes at a distance of 10.5-11 miles from centre coordinate {51.81571,-0.22428}.

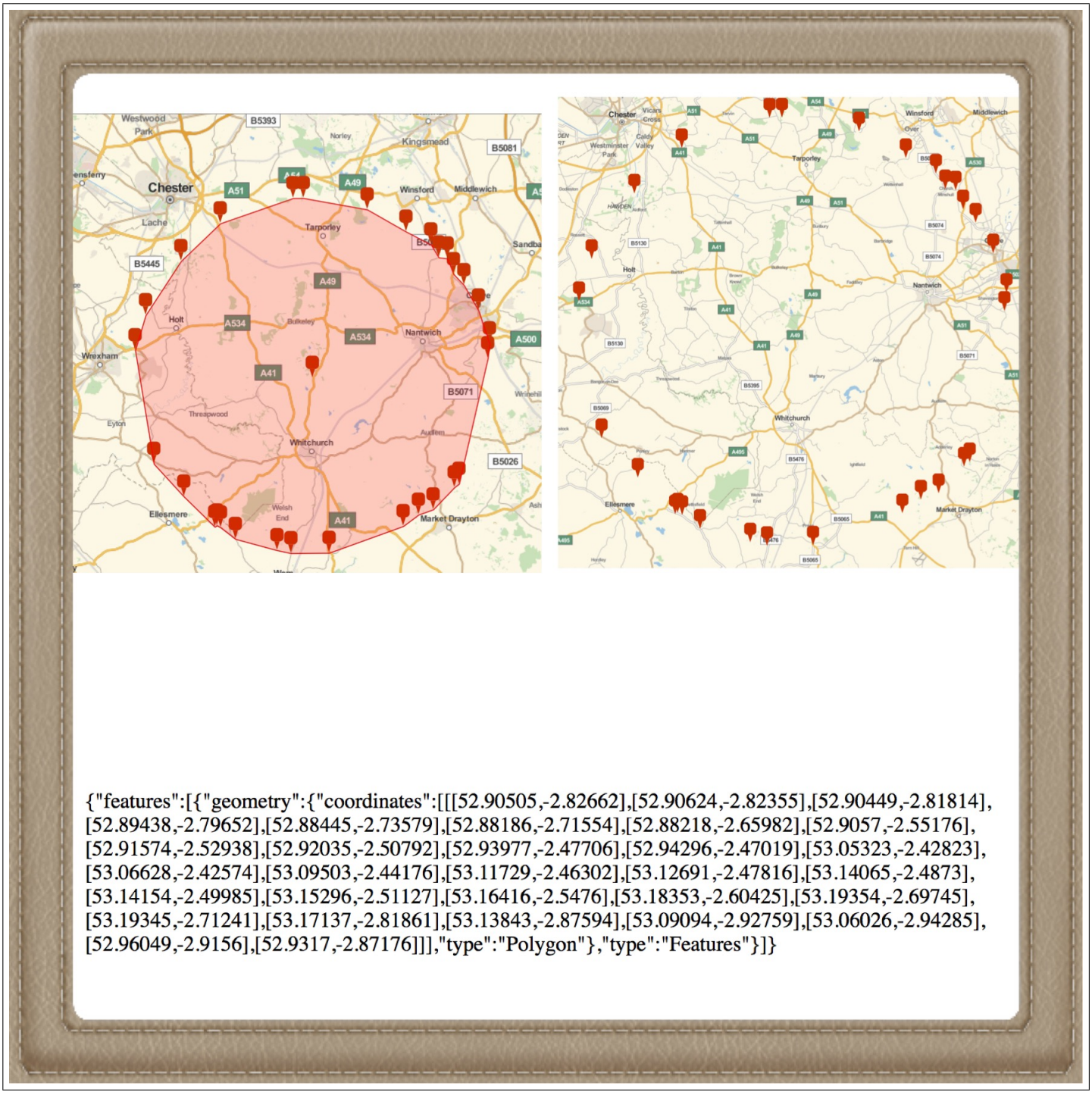


Figure 3: The nodes at a distance of 10.5-11 miles from centre coordinate {53.03599,-2.68422}.

0.4 geoJSON output for makers.

As of now, my java project outputs two different formats of geoJSON. As seen in Section 0.3, one such format is the format for drawing polygons. It also outputs the geoJSON format for drawing markers. One such output of ordered coordinates is shown is Figure 4. This can be used directly in any map editor.


```
{"features":[{"geometry":{"coordinates":[55.89871,-3.70006],"type":"Point"},"type":"Feature"},{"geometry":
{"coordinates":[55.90248,-3.70913],"type":"Point"},"type":"Feature"},{"geometry":{"coordinates":
[55.89793,-3.66415],"type":"Point"},"type":"Feature"},{"geometry":{"coordinates":
[55.89837,-3.6619],"type":"Point"},"type":"Feature"},{"geometry":{"coordinates":
[55.89771,-3.64642],"type":"Point"},"type":"Feature"},{"geometry":{"coordinates":
[55.89391,-3.62256],"type":"Point"},"type":"Feature"},{"geometry":{"coordinates":
[55.89957,-3.60992],"type":"Point"},"type":"Feature"},{"geometry":{"coordinates":
[55.90472,-3.55877],"type":"Point"},"type":"Feature"},{"geometry":{"coordinates":
[55.90697,-3.53673],"type":"Point"},"type":"Feature"},{"geometry":{"coordinates":
[55.91847,-3.49762],"type":"Point"},"type":"Feature"},{"geometry":{"coordinates":
[55.93217,-3.45366],"type":"Point"},"type":"Feature"},{"geometry":{"coordinates":
[56.03836,-3.35727],"type":"Point"},"type":"Feature"},{"geometry":{"coordinates":
[56.0824,-3.35877],"type":"Point"},"type":"Feature"},{"geometry":{"coordinates":
[56.08346,-3.36061],"type":"Point"},"type":"Feature"},{"geometry":{"coordinates":
[56.09514,-3.3683],"type":"Point"},"type":"Feature"},{"geometry":{"coordinates":
[56.16448,-3.44698],"type":"Point"},"type":"Feature"},{"geometry":{"coordinates":
[56.18661,-3.49142],"type":"Point"},"type":"Feature"},{"geometry":{"coordinates":
[56.15146,-3.85241],"type":"Point"},"type":"Feature"},{"geometry":{"coordinates":
[56.02284,-3.90751],"type":"Point"},"type":"Feature"},{"geometry":{"coordinates":
[56.00069,-3.8972],"type":"Point"},"type":"Feature"},{"geometry":{"coordinates":
[56.0019,-3.89076],"type":"Point"},"type":"Feature"},{"geometry":{"coordinates":
[55.99089,-3.89136],"type":"Point"},"type":"Feature"},{"geometry":{"coordinates":
[55.93212,-3.8183],"type":"Point"},"type":"Feature"}]}
```


Figure 4: The geoJSON output for location markers. The coordinates are at a distance of 10.5-11 miles from centre coordinate {56.05156,-3.63123}.

0.5 Algorithm comparison.

This section compares the working of my Travelling Salesman Algorithm with that of Wolfram Language, *Mathematica*. Wolfram Language, *Mathematica* uses the 'FindShortestTour' function[2] to determine the minimum path distances between a given set of coordinates. The ordered set of coordinates shown in Figure 5 is the output from my algorithm. It uses the Nearest neighbour approach as mentioned earlier in Section 0.1. Figure 6 shows the output of *Mathematica*. It is to be noted that the ordering of coordinates in

Figure 6 is different from that of Figure 5. The 'FindShortestTour' function of *Mathematica* finds a solution to the Travelling Salesman Problem using other approaches. [3]

```
{{52.6139, -2.69428}, {52.6081, -2.71984}, {52.5992, -2.73623}, {52.5907, -2.79116},
 {52.5832, -2.83436}, {52.5962, -2.94465}, {52.608, -2.97521}, {52.6095, -2.97913}, {52.6108, -2.98478},
 {52.6067, -2.98542}, {52.6311, -3.03545}, {52.709, -3.09613}, {52.7346, -3.09938}, {52.781, -3.08871},
 {52.7861, -3.09126}, {52.8105, -3.07321}, {52.8413, -3.04546}, {52.8534, -3.01342},
 {52.864, -2.99216}, {52.8842, -2.94203}, {52.8884, -2.90789}, {52.8958, -2.87103}, {52.8937, -2.82521},
 {52.8944, -2.79652}, {52.8732, -2.7244}, {52.8266, -2.63743}, {52.8253, -2.63217}, {52.7854, -2.60468},
 {52.7653, -2.59647}, {52.7531, -2.59081}, {52.7305, -2.58824}, {52.6972, -2.59881},
 {52.6907, -2.60063}, {52.6643, -2.62493}, {52.6519, -2.63173}, {52.6404, -2.65439}}
```

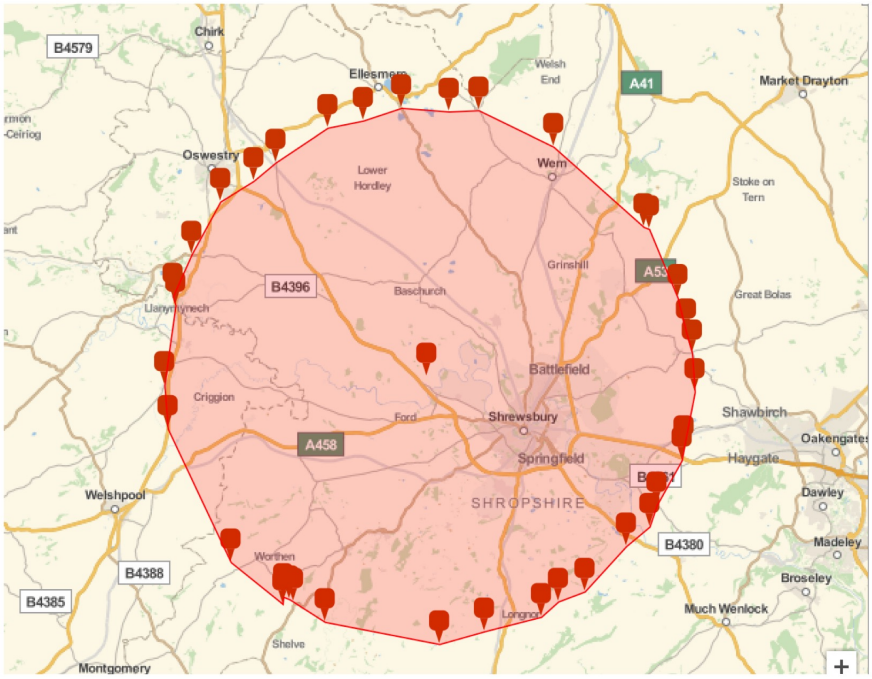


Figure 5: The shortest route according to my program output at a distance of 10.5-11 miles from centre coordinate {52.73954,-2.8467}.

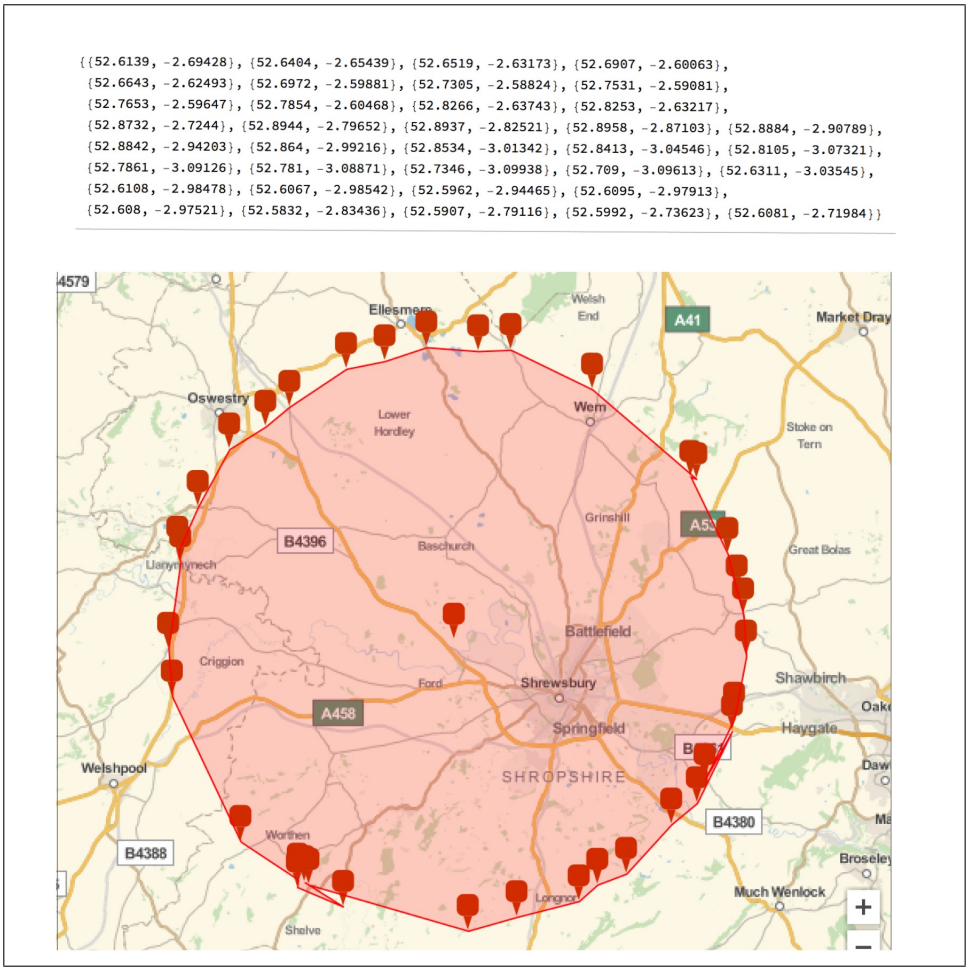


Figure 6: The shortest route found by Wolfram Mathematica at a distance of 10.5-11 miles from centre coordinate {52.73954,-2.8467}.

0.6 Road Map

Listed below are a few of the possible changes that can be implemented in the upcoming versions of this project.

- Making use of the BoundingBox filter from osmosis to extract our database of choice.
- Improve the API to include the battery level.
- Work on a mathematical function that decides on a range, taking into account the car model, battery level etc.

0.7 References

[1] http://www.movable-type.co.uk/scripts/latlong.html

[2] http://reference.wolfram.com/language/ref/FindShortestTour.html.en

[3] https://www.math.ku.edu/ jmartin/courses/math105-F11/Lectures/chapter6-part4.pdf