Using Genetic Algorithms to Find Most Efficient Route in Estimating Fish Population

By

Luke Peterson 220394-4339

26^{th} November, 2015

VEL113F Design Optimization

Faculty of Industrial Engineering, Mechanical Engineering and Computer Science

University of Iceland

Table of Contents

Introduction

The Icelandic fishery service must calculate the population of fish in the ocean north of Iceland. In order to do this, there are 132 separate vectors that must be sailed through, each dragging a net in order to catch fish. The fish are then counted from the catch and thrown back. After having counts from each of the 132 locations (Figure 1), the overall populations can be accurately estimated.

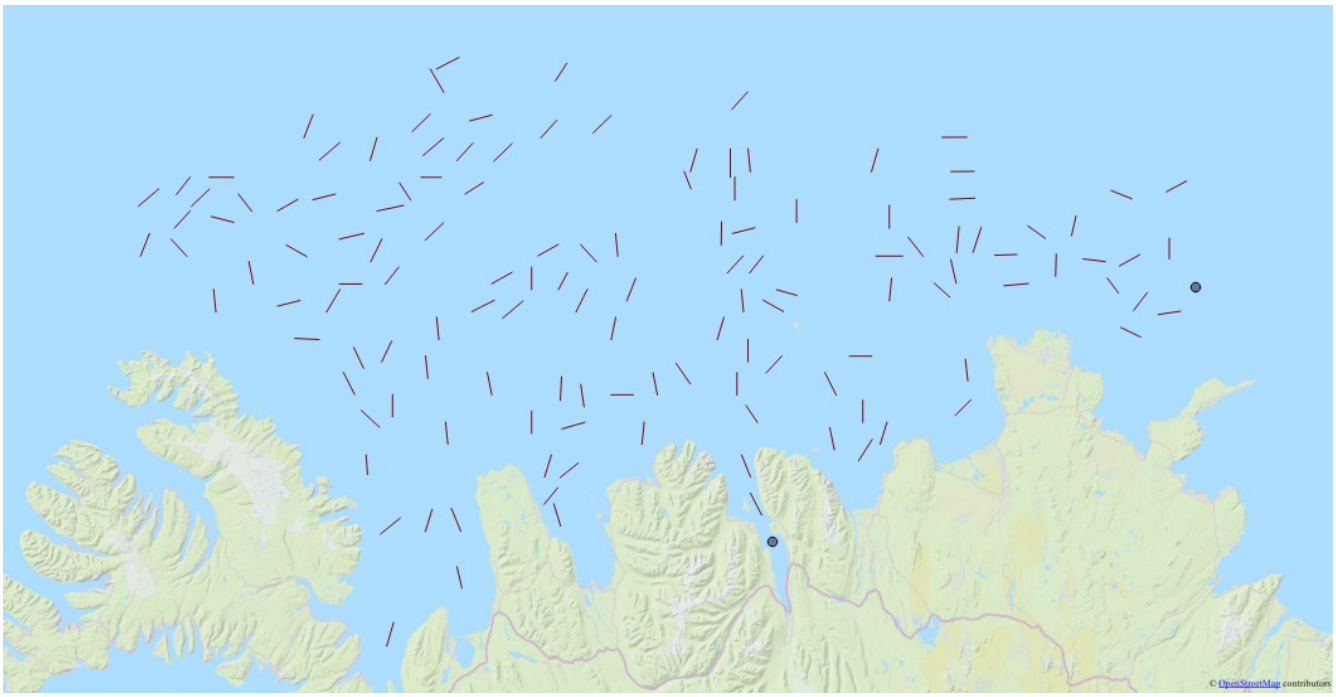

Figure 1 – the 132 vectors that must be sailed start and end points in the south and east, respectively.

Therefore, in order to estimate population of fish in the most efficient way possible, the shortest path encompassing all the vectors must be found.

Procedures

In order to solve this problem, the genetic algorithm approach was used. This comprised of using edge recombination as a crossover scheme, reciprocal mutation, and the Lin-Kernighan heuristic, coupled with elitist strategy in order to retain the minimum value between generations. The Lin-Kernighan implementation in this case is one that will not start the new array with the index randomly chosen, it will retain its old order until the first random index. This is because in this case, the route has a constant start and end point. It is not a perfect loop as assumed by the standard Lin-Kernighan algorithm. Crossover rate is .80 and mutation rate is .05. Typical runs comprised of a population size of 50 with 10000 generations.

Design

The performance index (PI) calculated for each individual is a modified distance formula. It is calculated from the end point of the previous vector to the beginning point of the next vector in each vector pair. The lengths of the vectors themselves are also counted, even though this is unnecessary as it will be a constant for every individual. Where it differs from the typical distance formula is in several environmental features. Water currents in the north of Iceland typically flow from west to east (Figure 3), and so sailing west has a 5% penalty in order to account for this (Figure 2).

$$\sqrt{((x1-x2)*1.05)^2+(y1-y2)^2}$$

Figure 2 – The modified distance formula used if sailing in a westward direction.

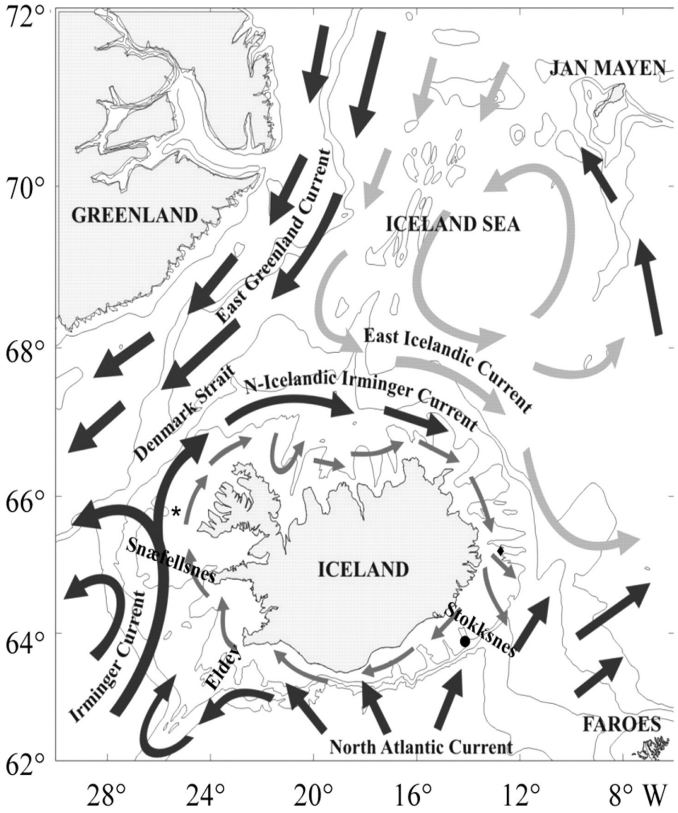

Figure 3 – Ocean currents north of Iceland. Image credit Marine Research Institute

Sailing around land is another consideration taken into account. Since the Euclidean distance formula will calculate straight lines between points, these lines must be checked if they

intercept land mass or not. This was done by drawing lines to outline the landmasses (Figure 4), and checking every line against these for intersection. If a line intersects (Figure 5), the distance of that line is recalculated as the sum of the sum of the two lines from the first vector, to the end point of the line of intersection, to the second vector (Figure 6). This is applied recursively, so in the case of lines intersecting for than one landmass, it will be calculated correctly. To check for intersection of the lines efficiently, orientation of the lines involved was found and cross products were compared, resulting in only two conditions to be checked in order determine an intersection. For a better understanding of how this works, it is implemented in the `dist()` and `linesIntersect()` functions in the code. Using this method, lines between points must be checked against only 9 lines for intersection to avoid all landmass.

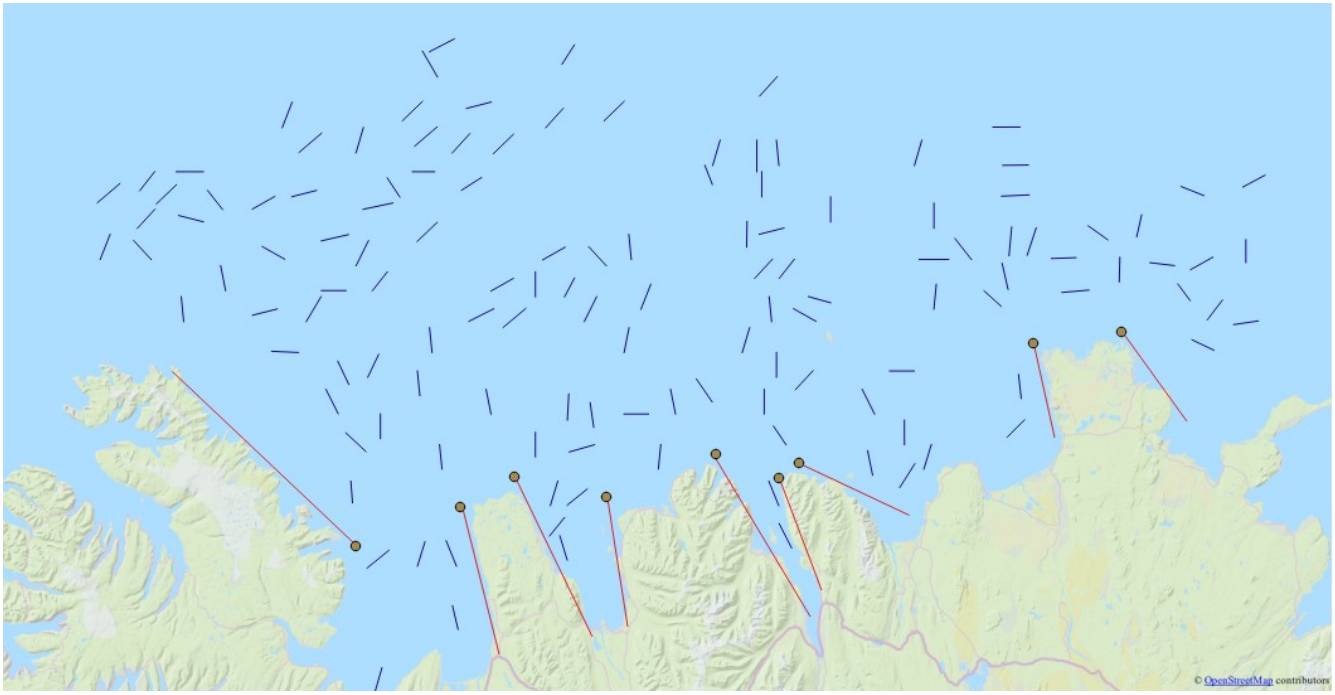

Figure 4 – The 9 lines that are checked for intersection

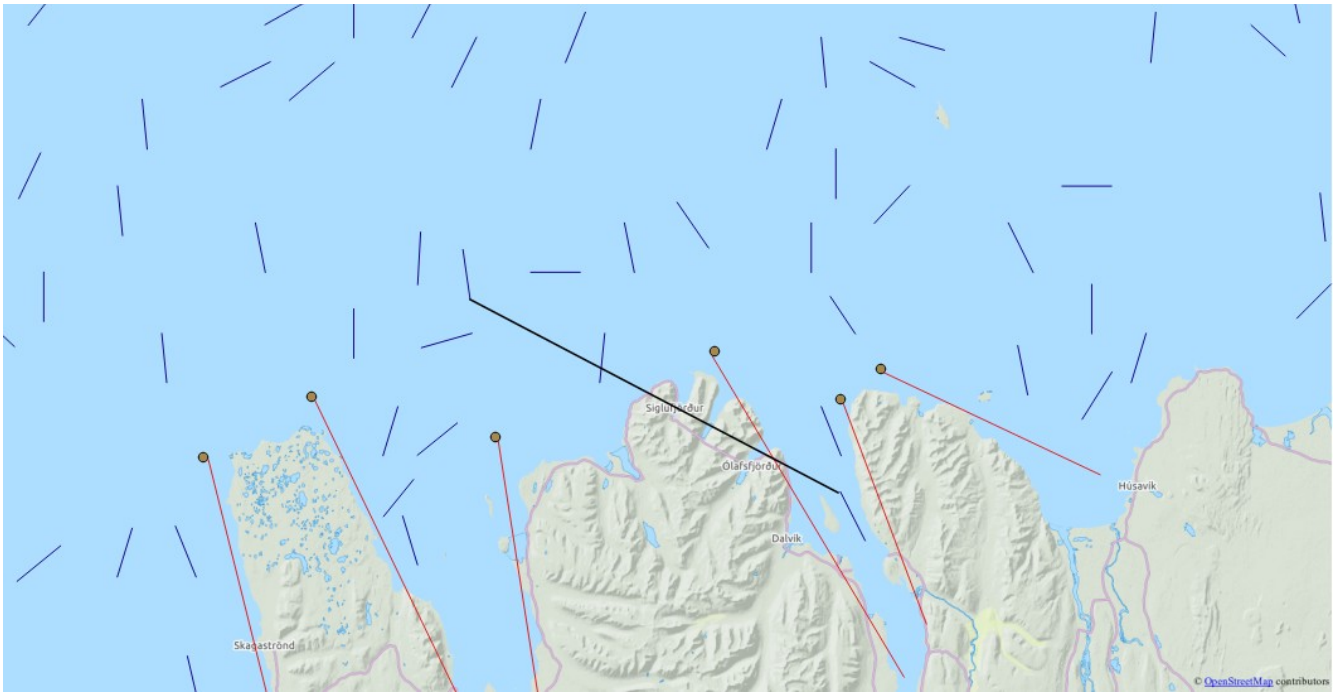

Figure 5 – An example of a line crossing landmass

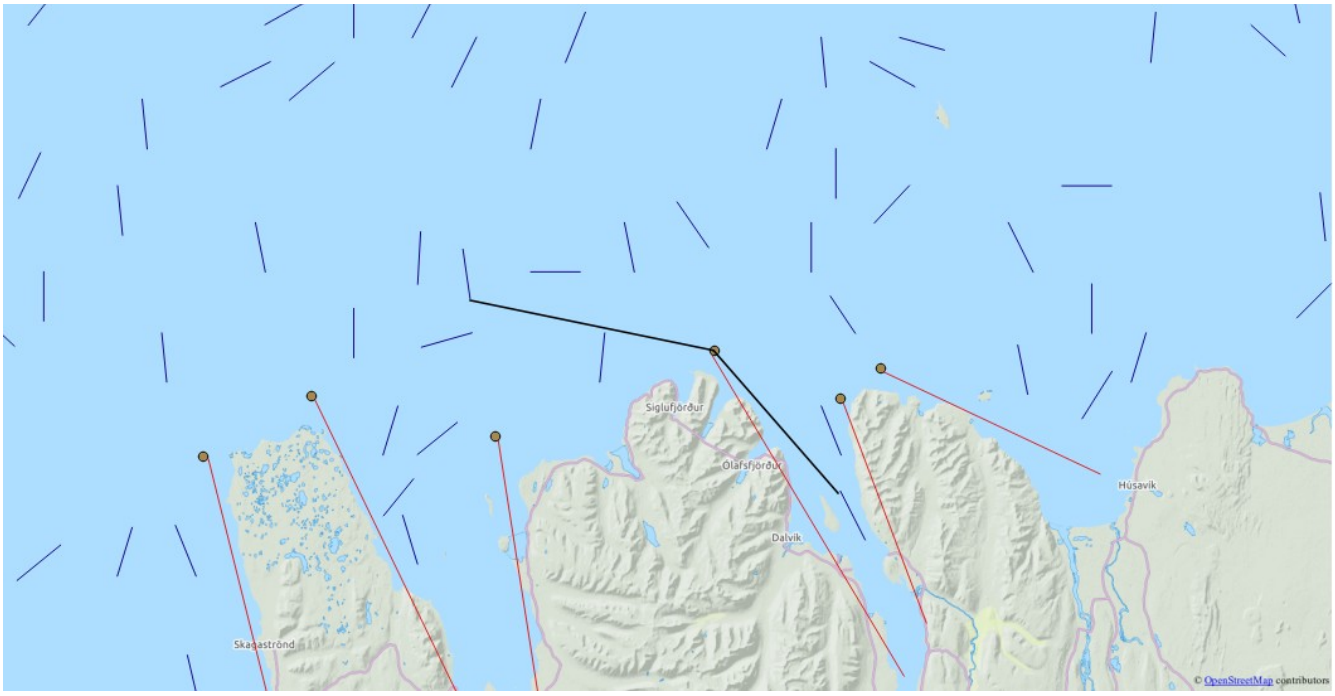

Figure 6 – How the distance between vectors will be calculated

Wind patterns are not accounted for similar to ocean currents, simply because wind is highly unpredictable in the region.

The largest caveat to note is that the PI for individuals does not have a reliable unit. It is something similar to degrees traveled, though is not entirely accurate because the euclidean distance formula does not account for the curvature of the earth, and the westbound penalty is applied directly to the distance. Because of this, the PI of an individual is only useful to compare against other individuals, and cannot be used to extrapolate cost, distance, or time.

Models

The model used is the typical model used in shortest path algorithms, simply an array the size of the amount of points to stop at, in this case it will be an array comprising of integers 0-131. The order of the integers in the array corresponds to the order of the points that the individual represents. The start and end points never change, and so are not included in the individual.

Several assumptions are inherent in this model type. We assume that each vector is visited exactly once and because of this there should be no cycles. We assume that all vectors are reachable from every other vector. And finally we assume that the path is continuous, that the Icelandic fishery service will complete the entire route without any delays or interruptions.

Optimization

In order to optimize the algorithm, several methods were tried. An ordered crossover was written and evaluated, and while it was faster than the edge recombination, it consistently gave results slightly poorer.

The current edge recombination algorithm has a small fix in place that runs directly after the ER function runs. This essentially checks individuals to see if any integers are missing, since in the implementation used there is a bug where in rare cases, the integer at the last index will be the integer od-1, despite the integer od-1 appearing in the individual already. The fix finds the missing integer and places it into the last spot in the individual. This should not change the effectiveness of the edge recombination, as the integer should only be missing if it is the last one not used and there are no edges pointing to it.

In order to further optimize the algorithm, a rewriting of the edge recombination algorithm

should be done with an emphasis on exploiting the vectorization properties of arrays in Numpy. In this case it was not done due to time constraints.

A large area of optimization needed in this algorithm is its many loops that are present. This limits the algorithm in the amount of threads that may be used at a time. Rewriting parts of the inner loops to utilize multithreaded architecture would vastly improve runtime.

Results

When the program is run, the results are given in the form of a result.txt file as well as a result.png image. The txt file will have the format of the PI after all generations, followed by the most fit individual, followed by a linestring of the points in the correct order that may be copy and pasted into GIS software such as ArcGIS or QGIS. The result.png file is a a graph of the progression of the algorithm.

The results that have been given are incomplete, yet promising. Due to time constraints, we were unable to run the algorithm for as long as needed to produce an optimal solution. The solution that is presented has a performance index of 75.6, whereas a randomly generated path may have a performance index around 250. Remember that this PI does not carry any significant meaning outside of comparisons to other indexes. The solution is presented, along with the progression of the minimum PI over all generations (Figure 8). This shows that the algorithm has a fairly predictable progression to the optimal solution. Looking at the map of the path produced from the algorithm, it is clear that a very efficient solution is forming (Figure 9).

```
[ 48.  43.  84. 125. 128. 127. 113.  59. 102.  15.  78.  75.
 117.  91. 101.  94.  57.  60.  53.  87.  65.  55.  54.   8.
   9.  10.  12.  89.  98.  97.  23.  11.  28.  77.  20. 116.
 115. 118.  66.   3.  25.  67.   5. 123.  74.  62.  18.  44.
 111. 130.  22.  40.  68.   4. 108.  45.  64.  63.   7.  14.
  16.  24. 104.  21.  83.  85.  99.  47.  49.  42.  41.  95.
  93.  90. 106. 112.  36.  30.  72.  70.  39.  80.  38. 100.
 131. 107. 114.  33.  27.  37.   0.  51.  56. 110.  58.  76.
   2. 109.  19.  88.  32.  31.  34.  35.   6. 119.  86. 124.
 122. 121.  52.  61. 105. 120.  26.  73.   1.  13.  46.  79.
  29.  17.  71.  69.  50.  81.  82. 126. 103.  96. 129.  92.]
```

Figure 7 – Given solution

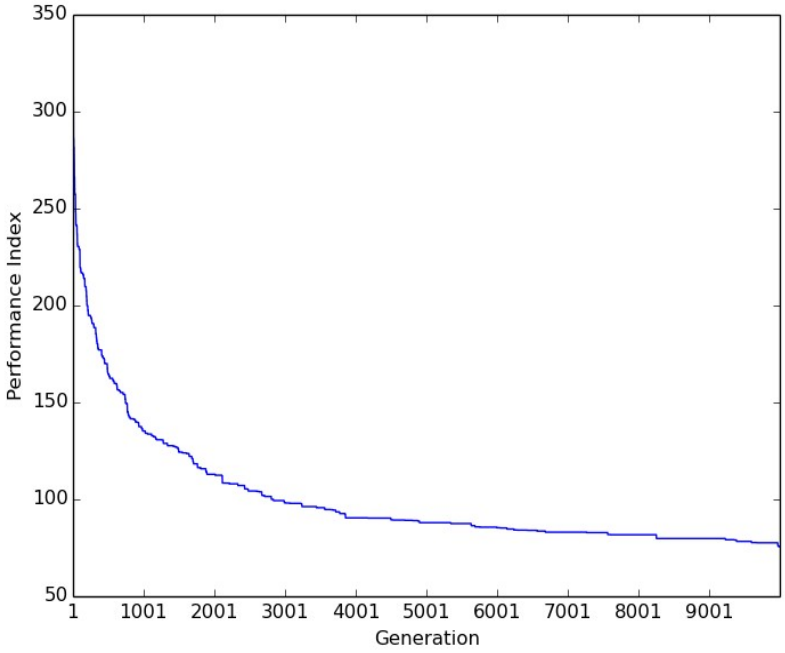

Figure 8 – The progression of the given solution

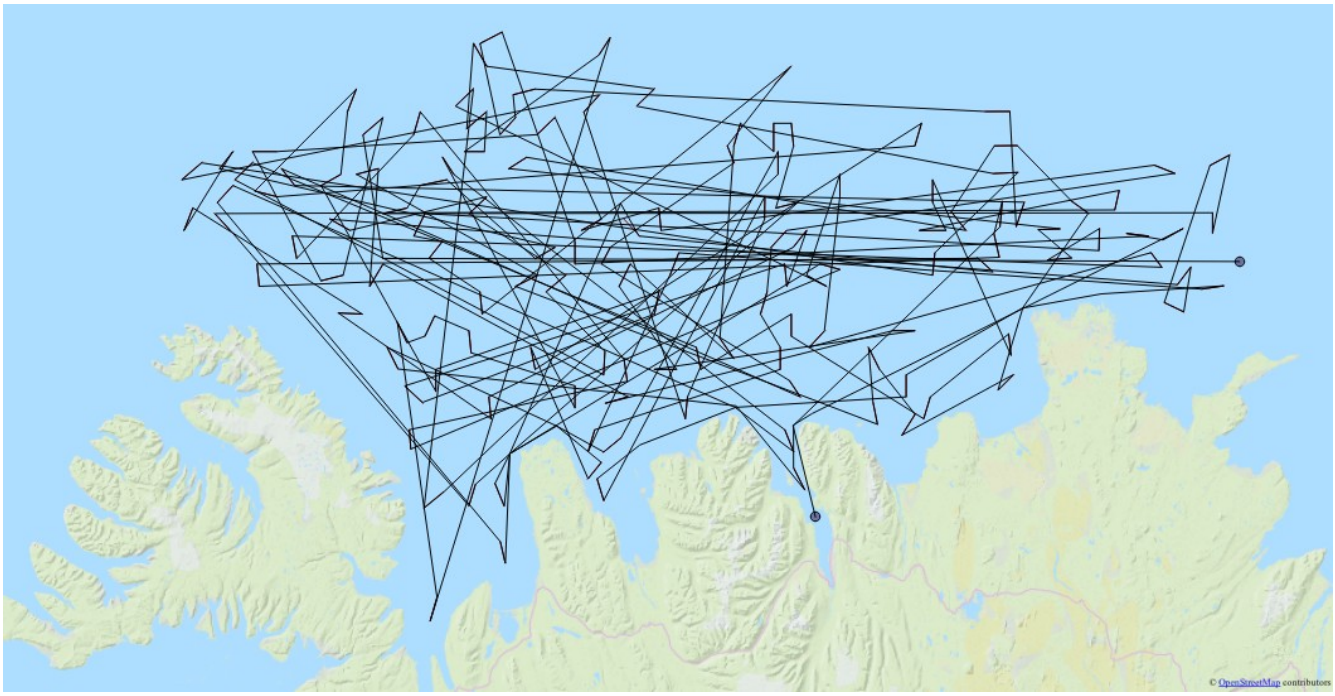

Figure 9 – The map of the given solution

Conclusion

While not given the optimal solution from running this algorithm, it is apparent that an optimal or near-optimal solution to this modified traveling salesman problem can be found using genetic algorithms. The edge recombination scheme and Lin-Kernighan heuristic have been shown to effectively work towards the optimal solution in a genetic algorithm. Given more time, an efficient path to visit every of the 132 vectors in the ocean north of Iceland would likely have been found. With the algorithm in place now, several modifications may be made in order to improve the runtime of the program and find a better solution. These include things such as taking advantage of vectorization inside Numpy as well as rewriting unnecessary loops in order to utilize multithreaded architecture.