Computational methods assignment 4 - Traveling salesman problem

Name:Benjamin Stott, ID:18336161

January 2021

1 Simulated annealing

1.1 modifications to code

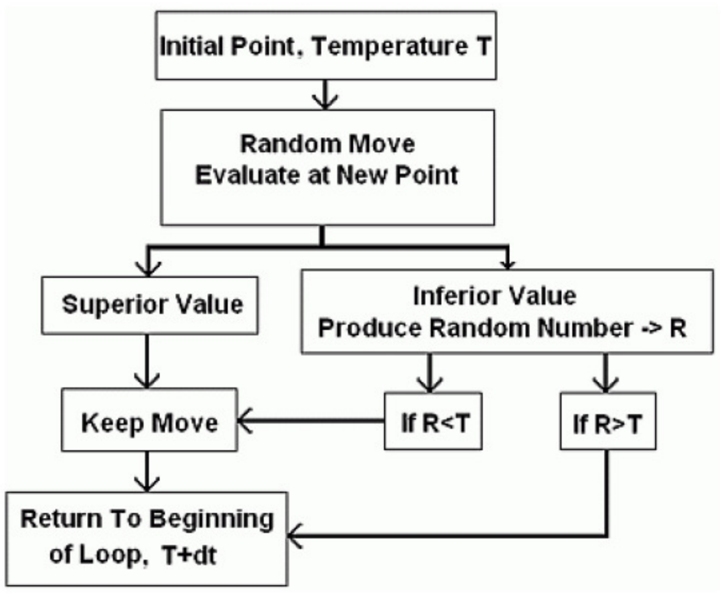
The first step towards making a simulated annealing algorith is to define a fitness. This is the total path length to the destination and back to the starting point(i.e the 'fitness'). This can be done with the following code

```
    path = 0
for i in range(self.N - 1):
path += self.gcd(route[i],route[i+1])

path += self.gcd(route[-1],route[0])

return path
```

In the annealing step we can take two random numbers from the list of indices of cities and swap 2 such cities for each retry of an iteration(similar to the sudoku notebook in which block positions were swapped). This new 'candidate' route is accepted for candidates whose fitness is lower than the previous iteration, otherwise they are accepted according to a boltzmann like term.
I have included an image that sums up annealing logic obtained from researchgate.net on 'The operating logic of Simulated Annealing method'.



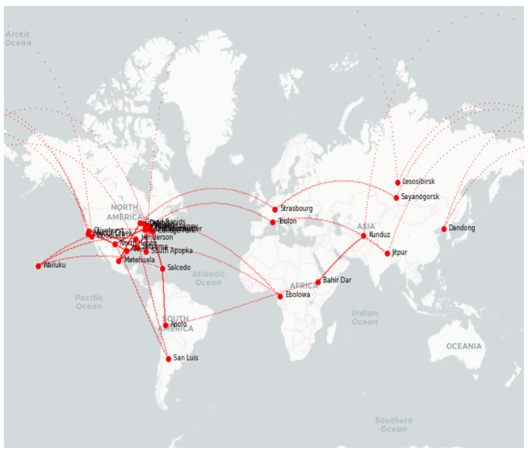
It was also necessary to define a new self value called 'self.solutionhistory' which would store each iteration of current routes in order to create an animation composed of these later.

1.2 Annealing schedule

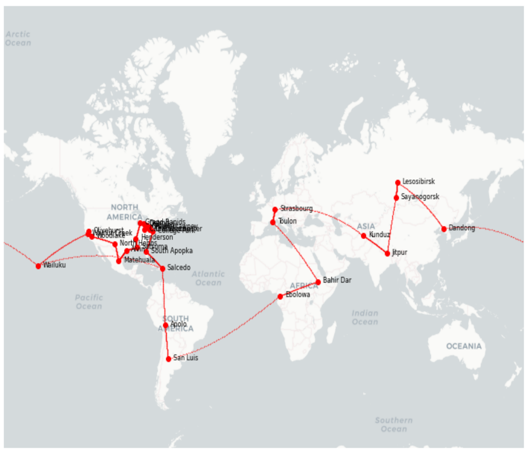
The annealing schedule or rather the manner in which each optimal solution was reached was found by calling the function with
irishsoln = SalesmanAnneal(irishlats, irishlons, irishcities, Irelandmap,stoppingiter=-1)

Using the irish cities list as an example. This loops over the annealing algorithm until an optimal solution is reached. It is also possible to put a predefined limit on the number of iterations but allowing it to run by itself requires less tweaking. This was done for all solutions obtained. The class function has the values of temperature and whatnot contained within it so these will automatically be called regardless.

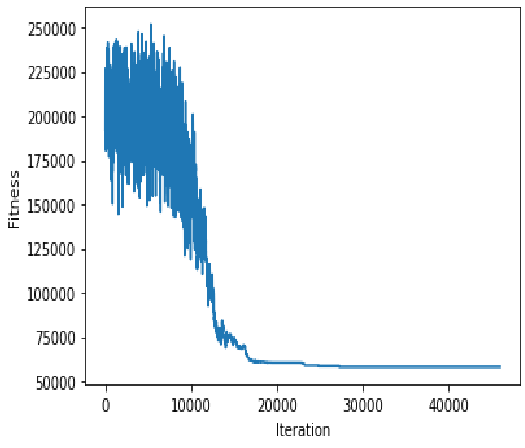
1.3 30 random cities



(a) Initial route between 30 random cities



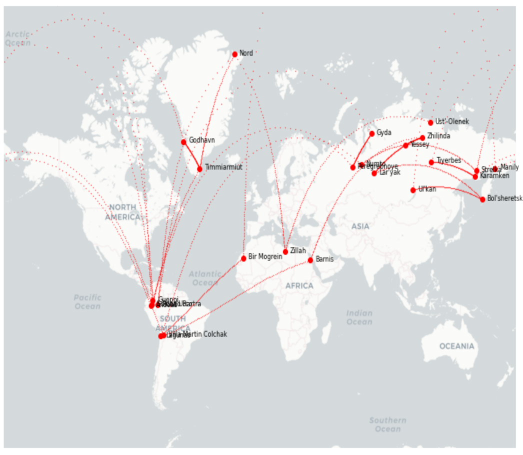
(b) Final 'best' route between 30 random cities



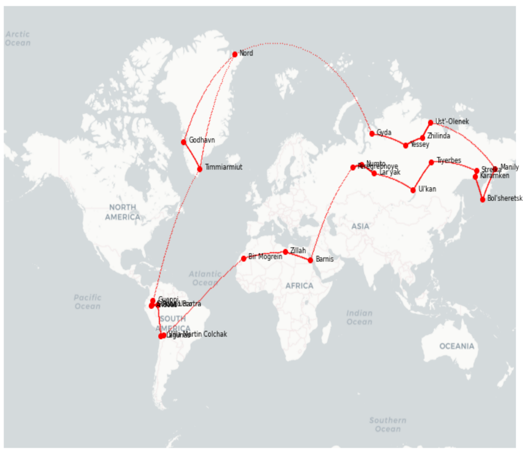
(c) Iteration vs fitness for annealing solutions of 30 random cities

Figure 1: Relevant annealing data for 30 random cities

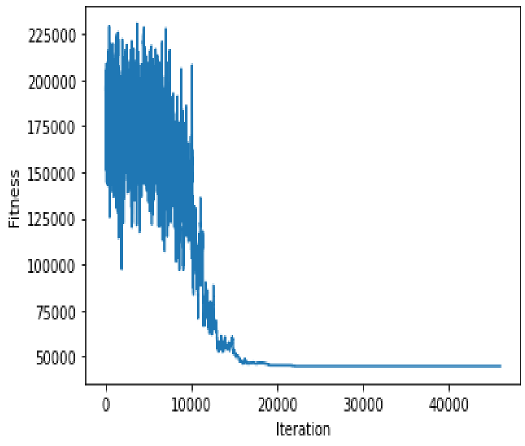
1.4 25 most populous cities



(a) Initial route between 25 most populous cities



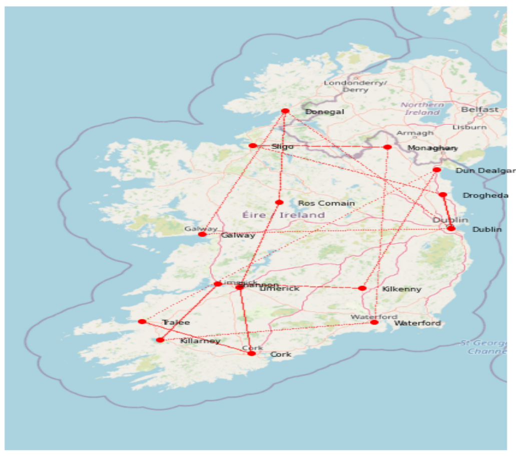
(b) Final 'best' route between 25 most populous cities



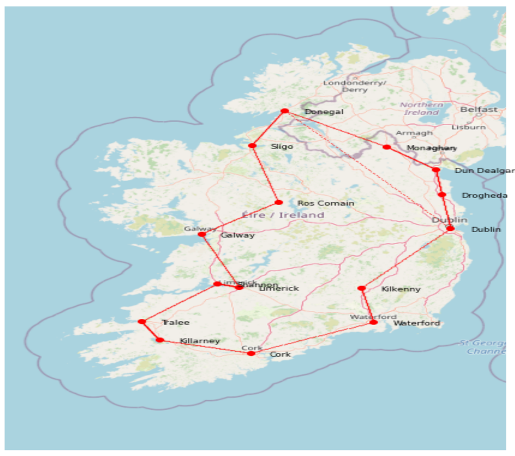
(c) Iteration vs fitness for annealing solutions of 25 most populous cities

Figure 2: Relevant annealing data for 25 most populous cities

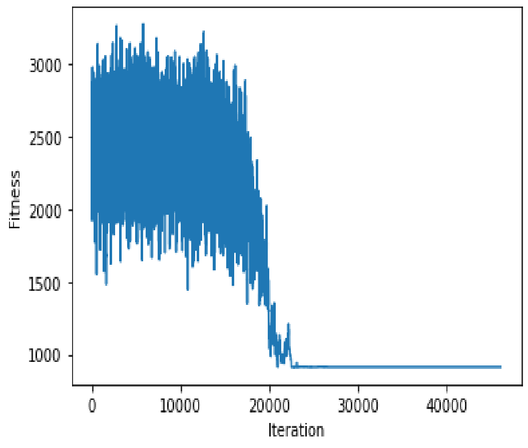
1.5 all Irish cities



(a) Initial route between all Irish cities in list



(b) Final 'best' route between all Irish cities in list



(c) Iteration vs fitness for annealing solutions of all irish cities in list

Figure 3: Relevant annealing data for all irish cities in list

1.6 animation code logic

My code for the animation more or less takes the self.initial route values for the initial frame of the animation with the update step changing route for each value of the solution history corresponding to a certain frame. By cycling through different values of history for each different frame of the animation, I can then produce an animation looping over these. I have also added code that allows the user to convert the file into a gif at 30fps(therefore being somewhat more sped up than when viewing the animation in the notebook). The animation is attached and the code for such an animation is in the notebook attached. I animated the convergence of the irish cities route to the optimal solution since it was the most zoomed in and allowed a better idea of what's going on.