# Assignment 6

For the given problem of the travelling salesman (referred to commonly as the `tsp` problem), I have used the `simulated annealing` algorithm.

### Algorithm

The implementation of the `simulated annealing` algorithm used must be understood. An array of randomly ordered numbers from *0* to *N-1* is created, which indicates the order in which the cities are traversed. Thus, it is this array which must be optimized. A *move* is taken to be a random swapping of two elements of this array. A move is considered successful if the resulting distance is lower than the original distance. Unsucessful swaps are still updated as per the probability:

$$P(\Delta E) = e^{-\frac{\Delta E}{T}}$$

Where $T$ is the temperature at that point, which slowly decays out, and $\Delta E$ is the change in path length.
Initially, due to the high value of temperature, there is a higher chance of even an unsuccessful swap, and thus the function searches more. As the temperature decays out, the updates become more and more favorable, albeit still accompanied with a certain amount of random searching. This algorithm thus is very useful for optimization.

### Code Description

The code must be run from the terminal using the the simple python command `python3 <file_name>.py`.
The code requests from the user the relative path of the text file containing the problem. Upon providing this, the code runs successfully.
After the cities are passed into the fucntion, a progress bar displays the progress. After the code is finally executed, the optimal path array as well as the corresponding path length is returned. An image of the final path is saved. The percentage improvement in path from a random initial guess is also printed out, which varies from one iteration to another.

### Hyperparameters

In the attatched code, the initial temperature is taken to be `2000`, the decay rate is taken to be `0.99` and the number of epochs is taken to be `50,000`

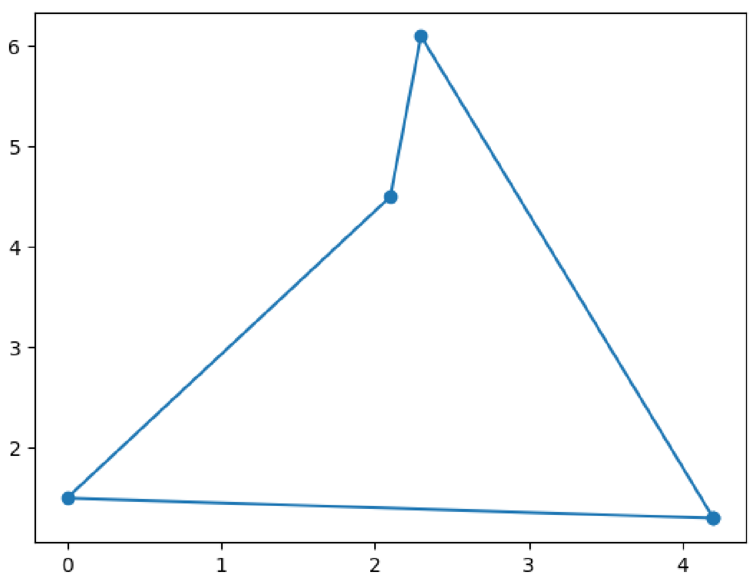
### 4 Cities

Given the following input:

```
4
0.0 1.5
2.3 6.1
```

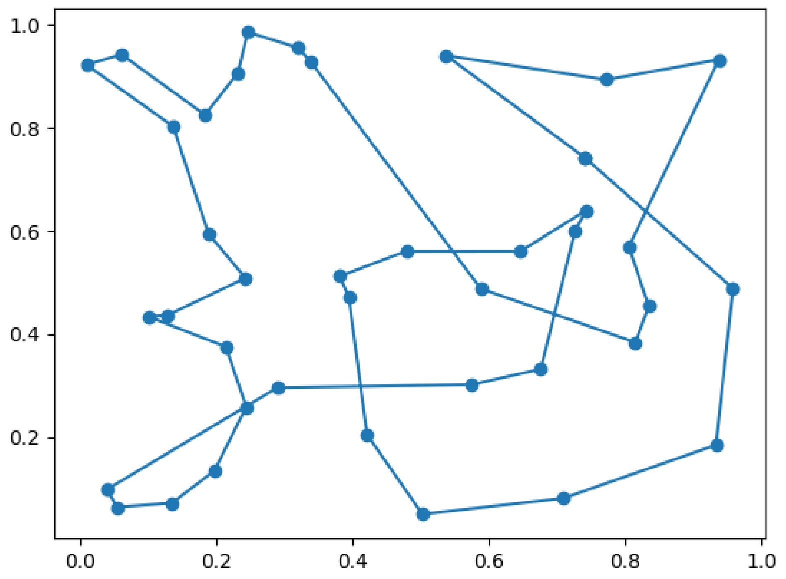
```
4.2 1.3
2.1 4.5
```

We see that the optimal path length is `14.64154124236167`. The percentage improvement in path varies per iteration depending on the initial random guess. The optimal path is as follows:



### 40 Cities

We see that with so many cities, we do not get an exact answer unless we increase the number of iterations to a significantly high value. Upon running the code once, we get the optimized distance to be `6.787098691823884`. This varies per run due to the randomness of initialization as well as the randomness of each swap, and not enough number of iterations. The optimized path looks as follows:


1.0
0.8
0.6
0.4
0.2
0.0
0.2
0.4
0.6
0.8
1.0