Statistical Techniques for Data Science and Robotics - Assignment3

Hussein Younes
h.younes@innopolis.university

1 Introduction

In this report, I shall summarize the steps taken to implement the simulated annealing algorithm for solving the travelling salesman problem. SA is a stochastic global optimization algorithm. As many other randomized algorithms, SA appears to be surprisingly simple, yet efficient at achieving its task (given conditions). The name and inspiration for the algorithm come from metallurgy, where controlled heating and cooling of the material allows it to form larger crystalline structure and improve properties. For this project, I am going to implement the SA for TSP using python, and the following sections explain the algorithm and assumptions, and reports the results at the accordingly.

2 Methodology

The algorithm basically aims to optimize the route passing through the most populated Russian cities minimizing the total distance. The algorithm mainly consists of the following steps

1. Before implementing the algorithm, we need to do data preparation and build the suitable data structures for the used methods.
2. We start by sampling the initial solution x_0 and resetting the time step $t = 0$.
3. Generate the next candidate state x'.
4. We check the acceptance criteria and calculate the acceptance ratio $\alpha = \frac{p^*(x')}{p^*(x_t)}$.
5. Generate $u \sim U(0, 1)$. If $u < a$, accept the new state. Otherwise, propagate to the next state.
6. Cooling: Reduce the temperature by a cooling factor.
7. Increment t.
8. Repeat the previous steps until cooled down.

2.1 data preparation

for data preparation, we need to filter the most populated cities, calculate the distances between them using the geolocation (longitude and latitude), and represent the cities as a graph to set-up the data structure for the necessary methods of the algorithm.

2.2 Initial state sampling

For generating an initial solution, I used a greedy approach. Starting from a random node in the graph, choose the next nearest neighbor. Continue that until visiting all cities. The total cost of this greedy trip is the initial cost.

2.3 Temperature

The temperature is initially set to a value that's comparable to the standard deviation of the samples' costs, to avoid issues with the exponential function. The temperature is then cooled down by a factor. The algorithm stops when the temperature reaches a very small number (where it can no longer accept worse states). It is worth noting that the temperature doesn't cool down exactly after each step, instead, for more stability, the temperature is changed after a certain number of iterations.

2.4 Proposal distribution

For the proposal distribution, two cities in the current path simply get exchanged, and the new path is considered as the new candidate solution.

2.5 Acceptance criteria

The following acceptance criteria is used for the candidate solution; If the new solution results in a better total sum of distances, then the probability for accepting this solution is set to 1. Otherwise, the probability is set to $\exp(-\frac{\Delta}{T_k})$, where Δ is the difference between the cost of the new state and the cost of the old state. After that, if the acceptance probability is greater than a value sampled from a uniform distribution from 0 to 1, the new solution is accepted, otherwise, it is not.

$$P(\text{accept new}) = \begin{cases} \exp(-\frac{\Delta}{T_k}), & \text{if } \Delta \geq 0 \\ 1, & \text{if } \Delta < 0 \end{cases},$$

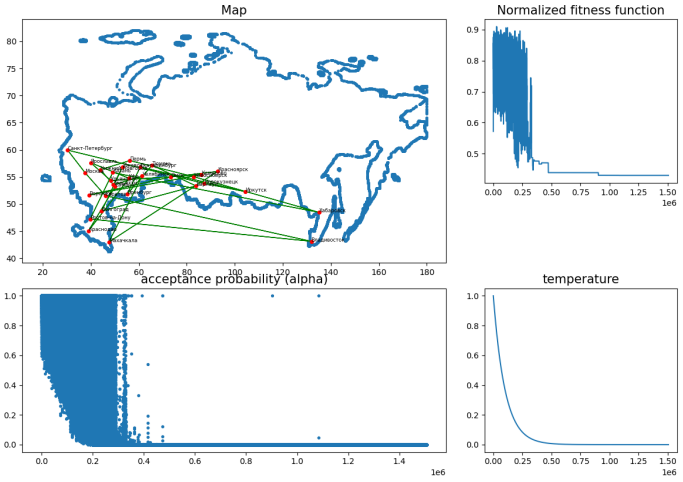

Figure 2. Slow cooling

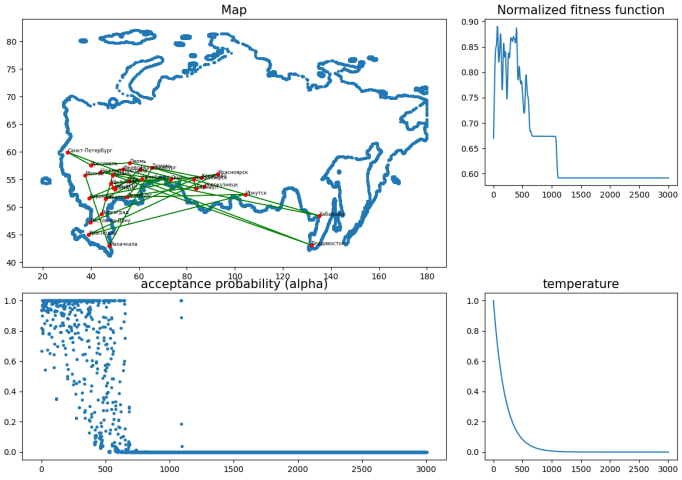

Figure 3. moderate cooling

3 Results

The above described algorithm was implemented and tested using python programming language, here's the link to the corresponding github repository. For comparison, 3 different cooling strategies were tested. For fast cooling, an initial temperature of 30000 was used with a cooling rate of 0.99 (meaning that the temperature gets multiplied by 0.99 every m iterations). The algorithm converged to a minimum cost of 19688.87680449212km and the results are illustrated in (Fig 1.). For slow cooling, (cooling rate of 0.999999), the algorithm takes considerably longer time to finish as opposed to fast cooling, and the optimal solution obtained has the cost of 18532.64651km, the results illustrated in (Fig 2.). For moderate cooling, with cooling ratio 16752.827988211982 (the average of the last two), the optimal solution has 16752.827988211982km total distance. The results of that are illustrated in (Fig, 3.). In all of the above experiments, we can see that at the beginning (when the temperature is high), there's a high probability of accepting solutions with higher total distance than the current one, and as the temperature starts decreasing the, the system gets more greedy, and only accepts solutions with a lower total distance.

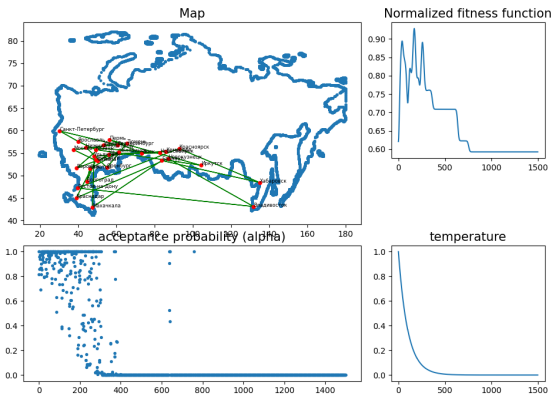

Figure 1. Fast cooling