Assignment 2
School of Mathematics
The University of Edinburgh
Year: 2022/2023

THE UNIVERSITY of EDINBURGH

In this exercise we apply simulated annealing to the travelling salesman problem (TSP). Question 1 2, 3 and 4 use the Python script hw2.py, which is available on Learn. Submit 1) a Python script and a PDF file, or 2) a Jupyter Notebook.

We denote the ith element of vector x by x_i and use the 0-based index. That is, if $x \in \mathbb{R}^p$,

$$x = (x_0, x_1, \ldots, x_{p-1}).$$

Similarly, we denote the (i, j)th element of matrix A by $A_{i,j}$. If $A \in \mathbb{R}^{p \times q}$,

$$A = \begin{pmatrix} A_{0,0} & A_{0,1} & \cdots & A_{0,q-1} \\ A_{1,0} & A_{1,1} & \cdots & A_{1,q-1} \\ \vdots & \vdots & \ddots & \vdots \\ A_{p-1,0} & A_{p-1,1} & \cdots & A_{p-1,q-1} \end{pmatrix}.$$

Let $C = \{0, 1, \ldots, n-1\}$ be a set of cities, where n is the number of cities, and $d \in \mathbb{R}^{n \times n}$ be the matrix whose (i, j)th element is the cost to travel from city i to j. We refer to d as the distance matrix. The cost is assumed to be symmetric: for each i and j, $d_{i,j} = d_{j,i}$.

Recall that a solution is a tour (i.e. a closed loop) which visits each city exactly once. We encode a solution as a permutation. For example, $x = (0, 1, \ldots, n-1)$ corresponds to the route starting at city 0, moving to city $1, 2, \ldots, n-1$ and then go back to city 0.

Exercise 1 *Simulated Annealing for TSP*

1. Implement `compute_total_cost` in hw2.py which takes a solution x and a distance matrix and computes the total cost of the route corresponding to x. (1 point)

2. Implement `run_greedy_heuristic` in hw2.py which takes a distance matrix and runs Algorithm 1. (1 point)

Algorithm 1: Greedy heuristic for TSP

Input : The number of cities n, the distance matrix d.
Let $c \leftarrow 0$ (i.e. set the starting city to 0).
Let $x \leftarrow [c]$, i.e. an ordered list containing c.
Let $S \leftarrow \{1, 2, \ldots, n-1\}$, i.e. a set of cities containing $1, 2, \ldots, n-1$.
for $i = 1, 2, \ldots, n-1$ **do**
    Find city $c' \in S$ closest to c (i.e., $c' = \arg\min_{\tilde{c} \in S} d_{c,\tilde{c}}$).
    Remove c' from S and append c' at the end of the list x.
    Let $c \leftarrow c'$.
Return x as a solution.

3. Implement `sample_two_opt` in hw2.py which takes a solution x and samples a solution uniformly from $N(x)$, where $N(x)$ is the set of neighbouring solutions obtained by applying two-opt to x (see Algorithm 2).[1] (1 point)

Algorithm 2: Solution sampling based on two-opt for TSP

Input : A solution x

Sample $i, j \in \{0, 1, \ldots, n-1\}$ s.t. $i < j$ and $j - i \notin \{0, 1, n-1\}$.

Return $[x_0, x_1, \ldots, x_i, x_j, x_{j-1}, \ldots, x_{i+1}, x_{j+1}, x_{j+2}, \ldots, x_{n-1}]$.

4. Implement `run_simulated_annealing` in hw2.py and run simulated annealing for 20 times using the distance matrix provided in distances.npy. Plot the objective values of the iterates (see Figure 1 for the format of the plot). Make sure the plot is readable (e.g. put axis labels). (3 points)

5. Alice, Bob and Carlota used `run_greedy_heuristic` and `sample_two_opt` functions and ran simulated annealing on the same TSP instance. The temperatures they used are the following:

$$T_1(k) = \frac{100}{0.1k+1}, \quad T_2(k) = \frac{1}{0.1k+1}, \quad T_3(k) = \frac{0.01}{0.1k+1}.$$

We know that they used different temperatures, but we do not know which one they used. Figure 1 shows the result of 20 runs of simulated annealing they got. Explain which are the most likely temperatures they used. (2 points)

6. Dulcinea and Emily used `run_greedy_heuristic` and `sample_two_opt` functions and ran simulated annealing on the same TSP instance. The temperatures they used are the following:

$$T_4(k) = 10 \cdot 0.99^{k/2}, \quad T_5(k) = \frac{10}{\log(k+1)+1}.$$

We know that they used different temperatures, but we do not know which one they used. Figure 2 shows the result of 20 runs of simulated annealing they got. Explain which are the most likely temperatures they used. (2 points)

[1] We define $N(x)$ as a set of solutions which can be obtained by deleting two edges in x and re-connecting the cities. We assume that $N(x)$ does not include x itself.

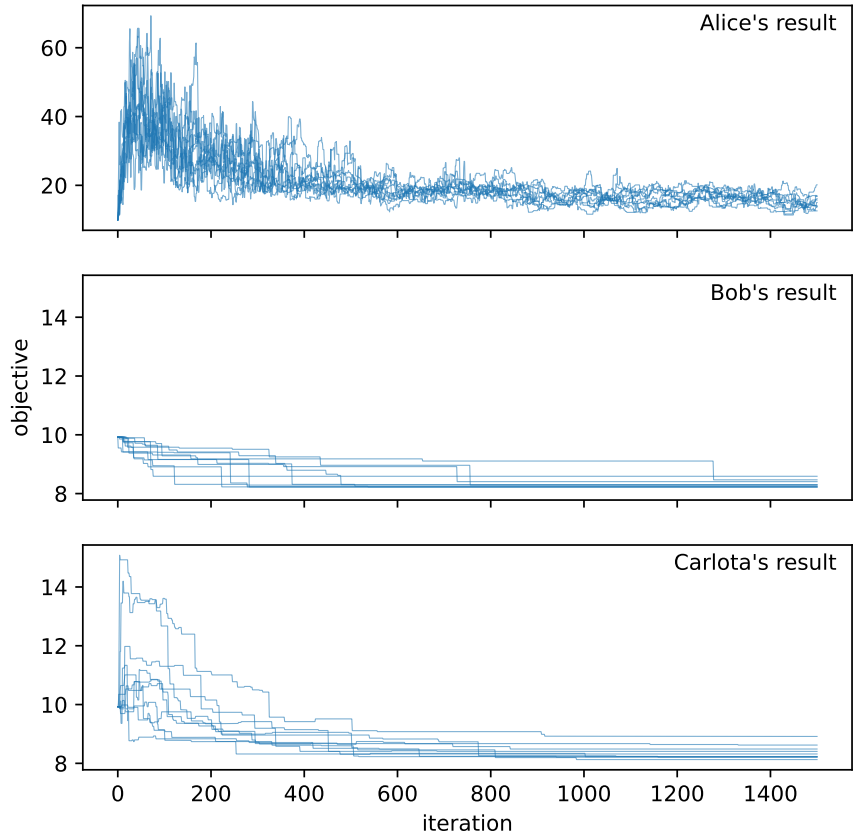

Figure 1: Results of simulated annealing.

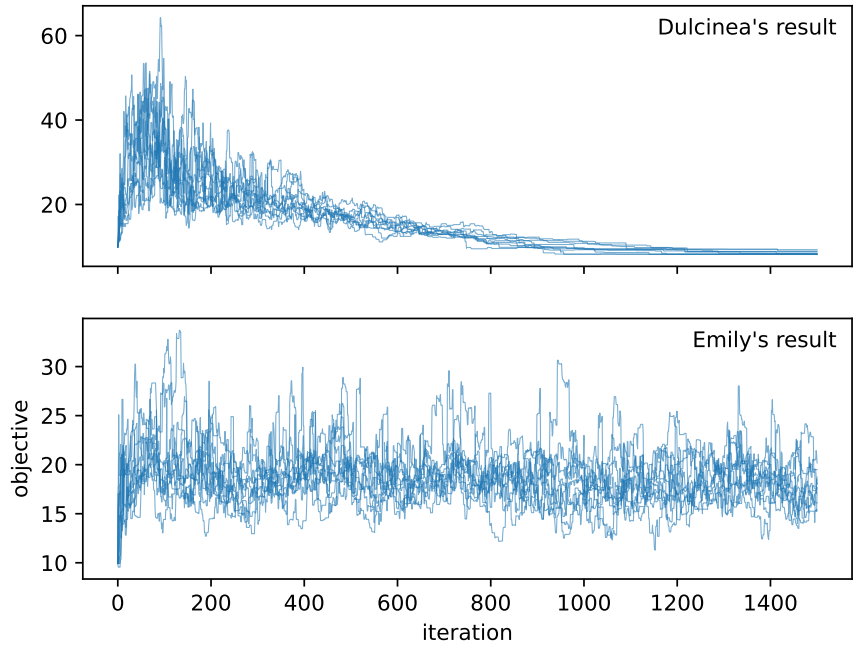

Figure 2: Results of simulated annealing.