Travel Salesman Problem

The travel salesman problem (TSM) is a famous problem in computer science. The problem is summarized as follows: imagine you are a salesperson who needs to visit some number of cities. Because you want to minimize costs spent on traveling, you want to find out the most efficient route, one that will require the least amount of traveling. You are given the distances (or cost) between cities.

Use hill climbing with restart like the one used in queens_hill_climbing.py program used in class (you can download the program from Canvas) to solve this problem.

Problem Setup:

As a test data define the following four cities as a matrix (list of list):

```
# define the distance between cities
tsp = [[0, 400, 500, 300],
    [400, 0, 300, 500],
    [500, 300, 0, 400],
    [300, 500, 400, 0]
    ]
cities = len(tsp) # number of cities
```

State representation:

A state is represented as a list of cities. For example, a state can be [0, 2, 1, 3]. This means that the route is $0 \rightarrow 2 \rightarrow 1 \rightarrow 3$.

A start state is a list of random order of cities. Let us fix the start state to be 0, and the rest of the start state list is a random order of cities. You can do that by the following code:

```
# randomly generate cities from 1 to 3
state = list(range(cities))   # state is [0, 1, 2, 3]
random.shuffle(state) # generate random order of cities.
```

Value function:

The value function takes a state as a parameter and computes the distance sum between the cities in a route. For example, given the state [1, 0, 2, 3], then the value function return the total distance going through the route $1 \rightarrow 0 \rightarrow 2 \rightarrow 3 \rightarrow 1$ (at the end, we have to go

back to the start state; in this example it is 1). You can use the tsp matrix to get the distance: tsp[1][0] + tsp[0][2] + tsp[2][3] + tsp[3][1] = 400 + 500 + 400 + 500 = 1800. So, the value function returns 1800. Remember, that after city 3 we must go back to the first city that we started with.

Get-neighbor function:

The get neighbor function takes a state as a parameter and return a new state by swapping randomly any two cities. For example, state [1, 0, 2, 3], we can randomly select two cities and swap them generating a new state, for example [0, 1, 2, 3]; here we swapped the cities 0 and 1.

Use random restart hill climbing to complete the code and find the best (smallest) route. Use the template attached with this assignment.

What to submit:

- Create a GitHub repository and save your program.
- Submit to Canvas a GitHub link to your repository.