ASSIGNMENT 2

SOLVING TSP USING LOCAL SEARCH

The Travelling Salesman Problem (TSP): Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city? It is an NP-hard problem in combinatorial optimization, important in operations research and theoretical computer science. Often instead of providing the distances between the pairs directly, the Euclidean co-ordinates of the cities are given.

In this assignment you will be solving TSP using local search.

First Choice Hill Climbing

You can find the details about First Choice Hill Climbing in your text book. In this assignment you will be implementing it in the following manner.

```
current = Randomly generated initial state
count = 0
while (true)
    for( i=0; i < σ; i++)
        ran = random number between [0,1]
        if (ran <0.5)
               neighbor <- a successor of current generated
                            by “heuristic1”
        else
               neighbor <- a successor of current generated
                            by “heuristic2”

        if neighbor better than current
               break;
     if(i == σ)
         return current
     current = neighbor
     count++
```

Simulated Annealing

You can find the details about Simulated Annealing in your text book. However, the provided pseudocode in the book is for maximization problems, while TSP is a minimization problem. In this assignment you will be implementing it in the following manner.

```
current = Randomly generated initial state
for (t = 1; ; t++)

    T = schedule (t)
    if (T == 0) return current

    ran = random number between [0,1]
    if (ran <0.5)
          next <- a successor of current generated
                  by “heuristic1”
    else
          next <- a successor of current generated
                  by “heuristic2”
     delE = cost(next)-cost(current)

     if next better than current i.e., delE < 0
               current = next
     else
        current = next only with probability exp(-delE/kT)
```

Scheduling function

The scheduling function (**schedule**) should provide larger *T* at the beginning and gradually decrease it to 0. If *T* is lowered slowly enough, Simulated Annealing will find a global optimal solution with probability approaching 1. However, finding such function is very hard in practice. You can find many scheduling function in the literature. Some are given below:

Linear cooling scheme: schedule (t) = $T_0 - \eta t$

Exponential cooling scheme: schedule (t) = $T_0 \alpha^t$, here $0<\alpha<1$

Logarithmic cooling scheme: schedule (t) = c / log (1+t)

In this assignment you will be implementing the linear cooling function.

Heuristics for generating neighbors

In literature one can find many well known heuristics to generate neighbors of TSP. Some of them are greedy and some are random. You will implement the following heuristics in this assignment.

0-1 Exchange

Randomly pick a city and remove it from the route. Then insert it in a different random position. A pseudocode is given below. However, this pseudocode fails to produce different routes in a special case.

```
oneZeroExchange(route)
{
     n = length of route;
     ran = random integer between [1,n]
     p = the city at position ran
     remove p from route
     ran2 = random integer between [1,n-1] and not equal to ran
     insert p back into the route at position ran2
     return route
}
```

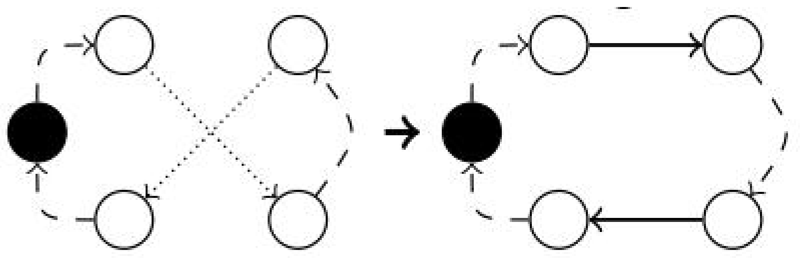
1-1 Exchange

Randomly pick two cities from the route and exchange them. A pseudocode is given below.

```
oneOneExchange(route)
{
     n = length  of route;
     ran = random integer between [1,n]
     ran2 = random integer between [1,n] and not equal to ran
     swap the cities at ran and ran2
     return route
}
```

2 - Opt

A 2-opt move removes two edges from the route and reconstructs it such that it does not cross over itself. Try to loop over all pair of edges, until one 2-opt move gives you a better solution. This strategy is called First Improvement Strategy.



```
twoOpt(route)
{
     for each possible pair of edges
         remove the edges and reconstruct route with
                        2-opt move  to get new_route
         if new_route is better than route
                     return new_route

     return route
}
```

Or-Opt

It attempts to improve the current tour by first moving a chain of three consecutive vertices in a different location (and possibly reversing it) to find an improved route. All possible chain is checked. If no improvement can be obtained, the process is then repeated with chains of two consecutive vertices, and then with single vertices. A pseudocode is given below, however it does not try reversing the chains.

```
or-opt(route)
{
     n = length of route
     for (chain_size=3;chain_size>=1;chain_size--)
     {
          for(i=1; i + chain_size-1<=n; i++)
          {
               chain = the subroute of route starting at
                position i and having a length of chain_size
               tmp_route = remove chain from route
               nn = length of tmp_route
               for(j=1;j<=nn;j++)
               {
                         new_route = insert chain to tmp_route
                                      at position j
                         if new_route is better than route
                              return new_route
               }
          }
     }
     return route
}
```

THE TASKS (A1+B1)

1) Implement First Choice Hill Climbing with the heuristic specified for your subgroup as per the following table. At the time of generating neighbor, select one of the two heuristics uniformly, i.e, with 50% chance Heuristic 1 will be used to generate a neighbor and the rest of the time Heuristic 2 will be used.

Subgroup	Heuristic 1	Heuristic 2
A1	0-1 Exchange	2 Opt (with First Improvement Strategy)
B1	1-1 Exchange	Or Opt (with First Improvement Strategy)

2) Run your algorithm with different values of the parameter σ. Each time you will run the algorithm 10 times, compute some statistics and present them as shown in the table. You must prepare the result before submission and show it in hard/soft copy at the time of evaluation.

Problem Instance	σ	Average Number of Iteration (*count*)	Average Cost	Minimum Cost	Gap = (AverageCost - BKS) / BKS × 100 %
att48	val1				
att48	val2				
att48	val3				
burma14	val4				
burma14	val5				
burma14	val6				
burma14	..				

The symbols bear the meaning as discussed in lecture.

THE TASKS (A2+B2)

1) Implement Simulated Annealing using with the heuristic specified for your subgroup as per the following table.

Subgroup	Heuristic 1	Heuristic 2
A2	1-1 Exchange	Or Opt (with First Improvement Strategy)
B2	0-1 Exchange	2 Opt (with First Improvement Strategy)

2) Run your algorithm with different values of the parameter T_0,η, and k. Each time you will run the algorithm 10 times, compute some statistics and present them as shown in the table. You must prepare the result before submission and show it in hard/soft copy at the time of evaluation.

Problem Instance	T_0	η	k	Average Cost	Minimum Cost	Gap = (AverageCost - BKS) / BKS × 100 %
att48	val1					
att48	val2					
att48	val3					
burma14	val4					
burma14	val5					
burma14	val6					
burma14	..					

The symbols bear the meaning as discussed in lecture.

Prepared By- Abdus Salam Azad

Input Format

The input files are simple text file with a ".tsp" extension. The first six lines contain some information about the problem. You need to read the 4th line to get the number of cities *n*. Starting from 7th line, the co-ordinates of the cities are given. The input file ends with a line with the string "EOF"

Output Format

Print the cost (length of the route) of your solution in the first line. The second line will print the solution as a sequence of integers. **Show the resultant route with GUI for bonus marks.**

Reference

1) TSP: https://en.wikipedia.org/wiki/Travelling_salesman_problem

2) Local Search, Stochastic Hill Climbing and Simulated Annealing :

Staring from chapter 4 upto 4.1.2(Including) of Russell and Norvig Book

3) 2-opt: https://en.wikipedia.org/wiki/2-opt

Prepared By- Abdus Salam Azad