Report

uOttawa

AI-Enabled Software Verification and Testing - CSI 5137[B]
Assignment 1

Introduction

In this assignment we aim to find the shortest path to travel all cities (call it Best Solution from now on) using a Metaheuristic algorithm.

I employed Hill Climbing as my algorithm and used Python to implement it on several datasets.

We will see more information about one of datasets and its result consequently, but at first let us look at the adopted algorithm and some concepts mainly from the second lecture.

Algorithm

As discussed in the second lecture, it wouldn't be a Metaheuristic view if we explore or consider all possible states to solve a problem. Needless to say, that for a TSP with 150 cities, it is extremely costly to calculate all states and compare results to find the best solution let alone for datasets with a thousand or two cities.

I adopted Hill Climbing (named as HC in this report) as one of famous Heuristic methods to solve given Travelling Salesperson Problem, but there were two major problems if I wanted to apply the basic HC algorithm:

1- In essence, HC algorithm starts with an arbitrary (random) solution and iteratively attempts to find a better solution by making incremental change to arguments of the algorithm, compares achieved total distance (which is our Fitness Function here in this problem) and does that till reaches to the one which is better compared to the previous one. We do not have sufficient time and computational power to use this solution on relatively small numbers of cities like 50 cities let alone datasets with more than 2000 cities to apply HC on all possible states and compare total distance of all states to find the shortest total distance eventually. I applied my optimized solution that takes only a mutation of all possible states on a dataset with 2103 cities (d2103.tsp) and it

took almost 10 times more than a dataset with 150 cities (ch150.tsp) to execute and it confirms that how costly it could be to apply HC on all paths both timely and computationally wise.
A dataset with only 5 cities will generate 120 unique paths, so obviously it is not worth for large dataset considering cost-benefit analysis.

2- If I were to use the basic HC algorithm, it was highly likely to fall into local optimum trap as it is a local search algorithm. It adopts an initial point randomly and construct the path based on that point. Possibility of finding an optimum or the global optimum point is strongly dependent on the initial point. So, I did feel to obtain an optimization solution to reduce possibility of being trapped in local optimum.

Considering above, I needed to optimize the adopted algorithm so that it minimizes described concerns.

In the other hand, I was supposed to use a Metaheuristic algorithm and I believe using a simple HC algorithm is not considered a Metaheuristic one.

Therefore, optimizing my algorithm not only made me able to benefit from a Metaheuristic view to solve the TSP, but also led to reduce the likelihood of presenting a local optimum as the best solution instead of the global optimum.

It is worth to mention that distance between cities is Fitness Function, changing arguments that effect the achieved best total distance like mutation rate and maximum iterations are our operators, and all possible paths or state space is our representation.

Optimization

I adopted Random Restart on top of HC algorithm to mitigate mentioned considerations specially to avoid being stuck in local optimum, shoulder, and flat local optimum (all are considered as local optimum in this report) in a way that Random Restart uses a mutation of state space.

That said, I came up with two different approaches of Random Restarts: state1 and state2, and I applied HC algorithm using both these Random Restart approaches and compared their results.

I also applied two different mutation rates, start city, and max iterations in the algorithm to compare results that we will observe in the Result section.

Details of my state1 and state 2 Random Restart approaches are as below:

- state1: its function *get_random_solution()* selects specific number of states randomly (here I chose 200 paths or states) and sorts them by total distance. It returns the one with shortest total distance as the initial state to *random_restart()*.
- state2: its function *get_best_solution_by_distance()* gets the start city and nearest city to the start city and does that repeatedly till creating a state. It returns this state as the initial state to *random_restart()*.

Random Restart function gets calculated matrix of cities' distances, home as start city, state1 or state2 as initial state, maximum number of iterations, and mutation rate which is the number (actually percentage) of cities to be replaced with each other for creating a new state. Eventually, it returns a path with shortest total distance among those states that the initial point of which was either the one from state1 or state2.

Experiment

To establish a platform for testing the proposed algorithm, I implemented it on two different datasets: att48.tsp and ch150.tsp which the first one consists of 48 cities and the latter propose 150 nodes to explore the best solution.

In addition, I changed 2 arguments of the algorithm beside changing *Home* as start city.

These are all due to the fact that getting stuck in local optimum is highly dependent on the initial point. As shown in Figure 1, if *a* is chosen as initial point we definitely end up in local optimum and if *b* is chosen as initial point, we will find global optimum. Therefore, I applied Random Restart on top of HC to optimize the algorithm and reduce possibility of local optimum, however, it will never eliminate this possibility.

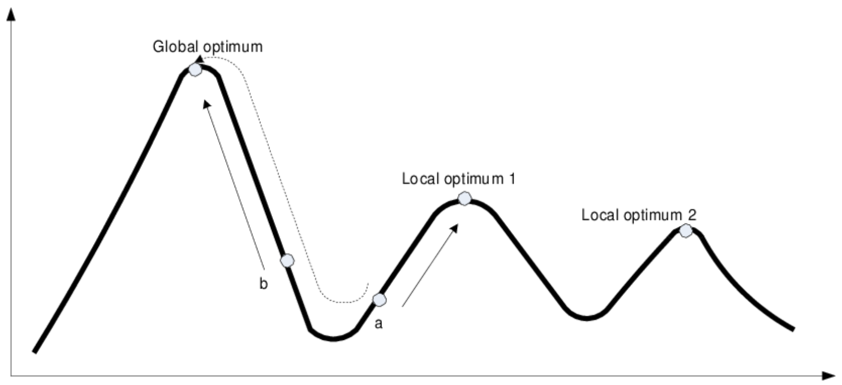

Figure 1

As discussed, I applied HC algorithm with two different Random Restart approaches, and I chose 2 and 20 for start cities (*Home*) with maximum iterations 1000 and 500 and mutation rates 0.01 and 0.06.

Home is first city, *max_iterations* is number of interactions of algorithm and *mutation_rate* indicates percentage of cities that will be replaced with each other. I chose relatively low number of *mutation_rate* as HC is a local search algorithm so we ought to find states in the neighborhood.

Numbers below each state express total distance of the best solution achieved by the algorithm with specifications showed in each table. Although my algorithm calculates total distances up to accuracy of eleven decimal places, all decimal places are removed from the table to enhance visualization of table. In fact, assuming integer numbers as miles, it has no significant impact on accuracy to have difference of eleven decimal places.

I double checked same numbers in Table 4 ensured that they are absolutely equal with different mutation rates though.

We will see further information about total distances or best solutions achieved by different parameters shown in the tables in Results section.

Dataset: aat48.tsp

Home: 2

Max Iteration	Mutation Rate	state1	state2
1000	0.01	48,649	37,931
1000	0.06	50,052	37,787
500	0.01	38,144	39,335
500	0.06	46,778	38,331

Table 1

Dataset: aat48.tsp

Home: 20

Max Iteration	Mutation Rate	state1	state2
1000	0.01	47,522	38,399
1000	0.06	39,681	38,633
500	0.01	48,677	38,422
500	0.06	45,110	38,861

Table 2

Dataset: ch150.tsp

Home: 2

Max Iteration	Mutation Rate	state1	state2
1000	0.01	11,950	7,991
1000	0.06	20,176	8,048
500	0.01	13,289	7,991
500	0.06	23,148	8,061

Table 3

Dataset: ch150.tsp

Home: 20

Max Iteration	Mutation Rate	state1	state2
1000	0.01	12,355	7,308
1000	0.06	18,919	7,480
500	0.01	12,254	7,308
500	0.06	24,094	7,480

Table 4

Results

As shown in tables, increasing iterations from 500 to 1000 led to a better or equal solutions with an exception of Home = 2 in aat48.tsp dataset. Decreasing mutation rates has positive effect in ch150.tsp with Home = 2 and 20, however, it has negative effect in att48.tsp with an exception with max iteration 500.

Solutions achieved by state2 are significantly better than state1 in all cases but in att48.tsp with maximum iterations 500 and mutation rate 0.01 which state1 is slightly better than state2.

In all, adopting state2 approach for Random Restart with any other given arguments on all datasets outperforms state1.

Reference:

Figure 1 adopted from https://www.researchgate.net/figure/Local-vrs-Global-optimum_fig1_277240109