Assignment 01

Author : Cristopher McIntyre Garcia
Email : cmcin019@uottawa.ca
S-N : 300025114

Introduction

The aim of this assignment is to implement a local search algorithm that will solve the Travelling Salesman Problem (TSP) with the shortest path possible. The meta heuristic search algorithm chosen is the Hill Climbing search algorithm and its many variations, which are First Ascent Hill Climbing search, Steepest Ascent Hill Climbing search, and Random Ascent Hill Climbing search. Also, and random search algorithm is implemented to be compared with the Hill Climbing search algorithms, which will allow us to evaluate their performance benefits. The algorithms will be compared with each other using the number of steps it took to find a best solution and the solution itself.

Search space details

Let's begin by explaining what a representation of the problem would entail. In TSP, we are given a set of cities, and in our case, their coordinates in a two dimensional space. The distance between two cities is therefore the euclidean distance between them. From a given city, one can travel to any other city except those already visited, and thus, our goal is to find a permutation of all the cities that will yield a minimal cost. As a result, a solution representation of the problem is a list of all the cities.

Next we explain the fitness function of the problem. As stated above, we are interested in finding a permutation of all the cities that yields a minimal path distance cost. We are therefore working on an optimization problem to find the smallest path possible between the first and last city while passing by every intermediate city in our permutation. Because we have the coordinates of every city, we can make a K x K distance matrix, where K is the number of cities we need to visit. The ith row in the matrix will represent the ith city provided to us, and the jth column will represent the jth city provided to us initially. The euclidean distance between the ith and jth city is stored in the matrix in the ith row and jth column. To find the distance between the ith and jth cities, we perform a matrix look up. Note that this matrix is only used for evaluating the fitness of a solution and will not be used to find a solution. The fitness function will therefore be the summed distance between every city in the permutation by lookup.

Next we explain the operation. Because we are performing local search, we are interested in the neighbours of our current solution. The operation is thus the swapping of a pair of cities in the permutation. In cases where we want to find all the neighbours of a solution, an operation could be performed multiple times, and the cost of each solution stored for evaluation later on. That said, we limit this to only the neighbours of a current solution, meaning only a single swap may be performed to find a different solution at a time.

Algorithms

In every algorithm, we start by choosing a random solution, and from there, locally search for a more fit solution.

The first algorithm is the First Ascent Hill Climbing search (or just First Ascent), in which we look through the neighbours of a solution, and if a better solution is found, it is chosen to be the new current solution. We proceed like this until no new solution is found. The advantage of this algorithm is the limited number of neighbours that need to be computed, and therefore might be a faster running algorithm. The disadvantage is that with every swap, we may not be as close to a global solution as we could be if we were to look through all the neighbours and choose a more optimal solution.

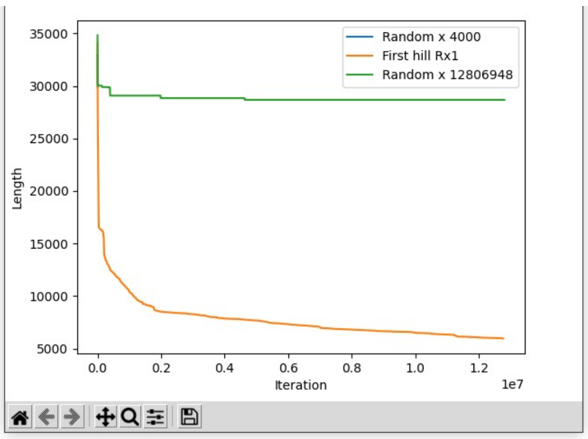
The second algorithm is the Steepest Ascent Hill Climbing search (or just Steepest Ascent), in which we look through the all neighbours of a solution. If any better solutions are found, their cost is saved and we proceed to generate the rest of the neighbours of the current solution. Once all the neighbours are found, if no better solutions were found, we end the search. If one or multiple better solutions are found, the best solution is chosen to be the new current solution. The advantage of this algorithm is that in choosing the best solution out of all the neighbours, with every swap we will be closer to the global solution than if we were to pick any other solution. This might mean we will find an overall better solution if the landscape of the solutions have steeper hills for more optimal solutions. However, generating all the neighbours of every solution may come at the price of a larger iteration count and so may lead to a slower running algorithm.

The third algorithm is the Random Ascent Hill Climbing search (or just Random Ascent), in which we look through the all neighbours of a solution. If any better solutions are found, they are saved and we proceed to generate the rest of the neighbours of the current solution. Once all the neighbours are found, if no better solutions were found, we end the search. If one or multiple better solutions are found, we choose a random solution from those found to be our new current solution. The advantage of this method is that we are able to choose a current solution that may not be optimal, but might lead to a better global solution in the long run. The disadvantage is that generating all the neighbours of every solution found, and not choosing the most optimal solutions will likely make this algorithm the slowest of them all.

Finally for the random search algorithm, we just generate a random permutation of all the cities and keep its total path distance. We do this for a certain number of iterations and keep the path with the shortest length. If it took x number of iterations to find a final solution with the methods above, we find x random paths and keep the best result. Note that we compare the paths of the random search one after the other, and so it is possible to find the same solution twice.

Results

First Ascent



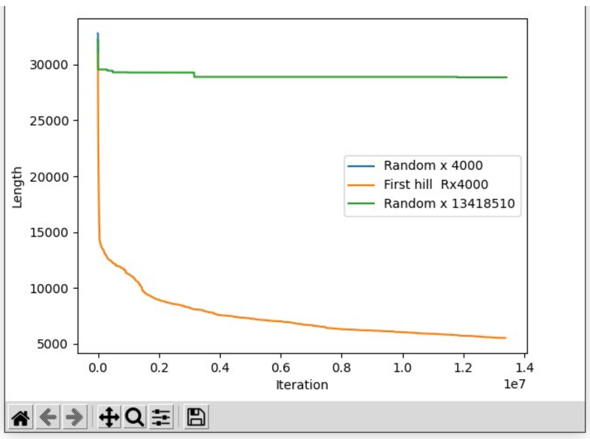


Figure 1. (left) First Ascent after 1 random iteration vs Random - (right) First Ascent after 4000 random iteration vs Random – File used a280.tsp

In the above, we can see that First Ascent out performs random search. At first, it may perform about the same, however after a small number of iterations, First Ascent finds paths smaller than random search. There does not seem to be any advantage in starting at a better solution found with 4000 random searches. In this case, the algorithm performed better when starting from a single random search.

Steepest Ascent

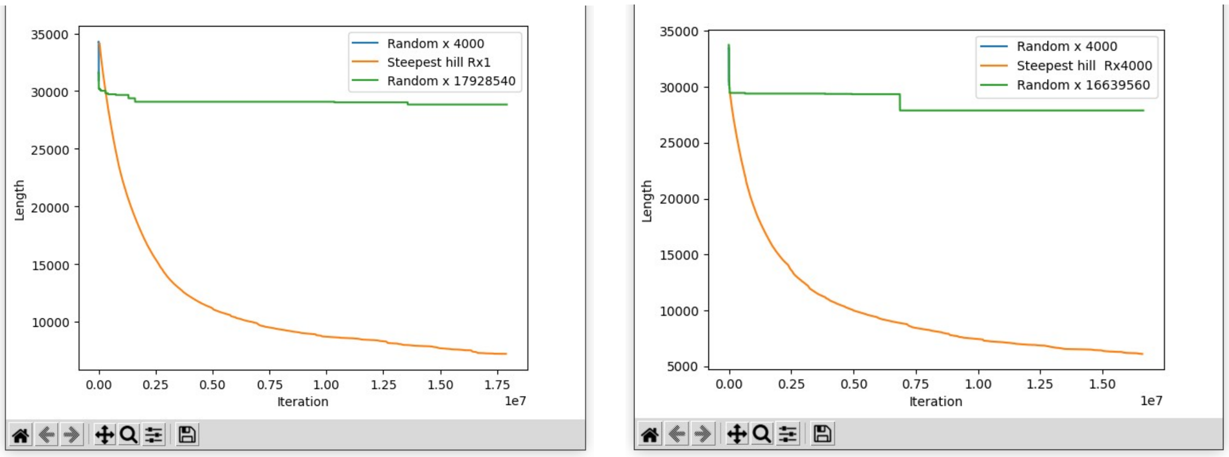


Figure 2. (left) Steepest Ascent after 1 random iteration vs Random - (right) Steepest Ascent after 4000 random iteration vs Random – File used a280.tsp

In the above, we see that Steepest Ascent outperforms random search by a large margin. When starting from a single random search, the random search algorithm seems to perform better for the first few iterations, however soon thereafter, plateaus and is unable to find any solutions that perform much better. Steepest Hill is able to maintain a good pace when finding new solutions. That said, it is not as fast at finding solutions than First Ascent. This is because before choosing a new solution, it must generate every neighbour of the current solution. Also, the final solution found does not seem to be better than the one found using First Ascent, and so it may not be a more desirable algorithm.

Random Ascent

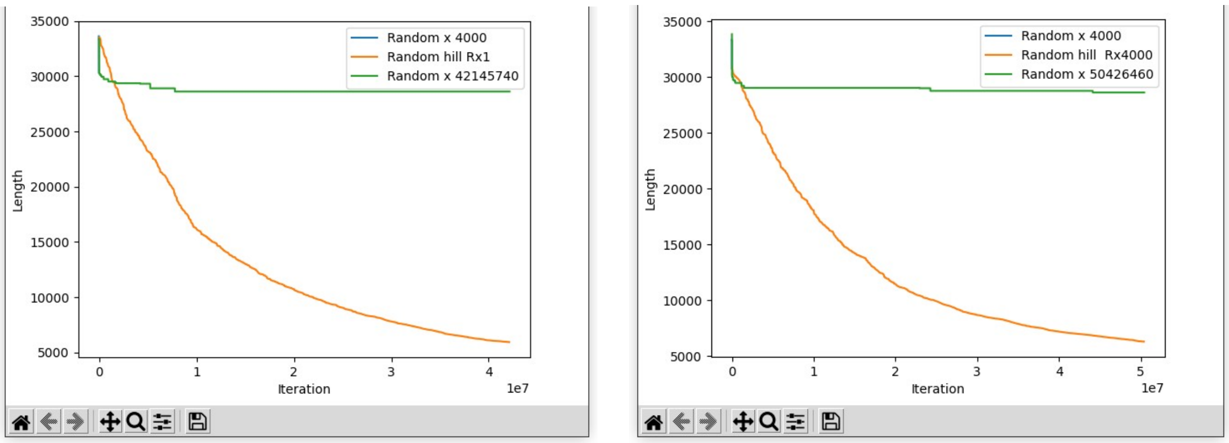


Figure 3. (left) Random Ascent after 1 random iteration vs Random - (right) Random Ascent after 4000 random iteration vs Random – File used a280.tsp

In the above, we see that Random Ascent outperforms random search by a large margin. It does benefit from starting from a more optimal solution after 4000 random searches. At the beginning, it seems to lag behind random search a little, however, as random search plateaus, Random Ascent keeps finding solutions with shorter paths. That said, it takes a longer amount of time to find a similar solution to the algorithm seen previously.

Random

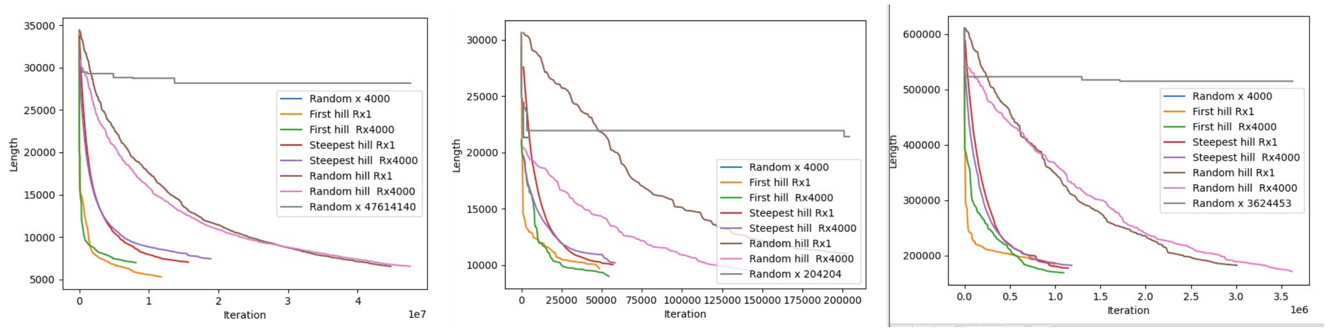


Figure 3. Results of each algorithm for a280, berlin52, and bier127 respectively (larger versions can be found at in the *images/All* folder)

The best performing algorithm is the First Ascent search, in terms of path length and iterations. It usually takes as many iterations or less than Steepest Ascent search, and finds shorter paths. Random Ascent search takes a very long time to perform, however it does find paths that are similar in length to the other algorithms. After a certain number of iterations, every algorithm out performs random search.

Conclusion

The best Hill Climbing algorithm for the TSP seems to be the First Ascent Hill Climbing search algorithm. It is able to find the best results out of all the other variations and does so in less iterations. Every algorithm is able to perform better than random search in the same number of iterations. Random search tends to hit a plateau early on, and seems to struggle when looking for better solutions. To see the generated paths, see the “paths” folder in the designated algorithm folder, in the “images” folder.