Assignment 1. Hill climbing and simulated annealing

Metaheuristics

José Manuel Izquierdo Ramírez

Contents

List of Figures

1 Exercise 1.

1.1 How does this algorithm behave as we increase the complexity of the problem (number of cities in the TSP)?

For this question I have evaluated the algorithm measuring the time and the legnth of the final path, to do this I have done the mean of five iterations for each city using eleven cities between 5 and 150.

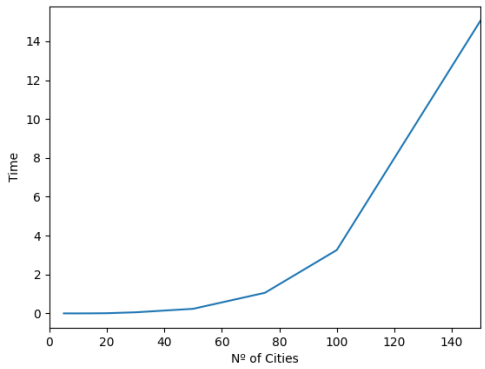

Figure 1: Time complexity of Hill Climbing

From this figure we can say that the complexity of the algorithm increases exponentially as the number of cities increases.

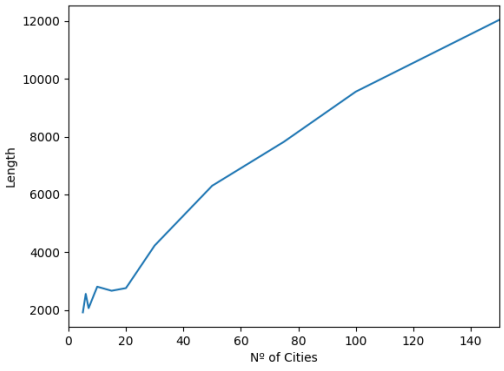

Figure 2: Length of final path TSP problem

From this figure we can say that obviously the more cities there are the longer the path will be, however it seems that the heuristic are finfing good solutions because the slope does not increase too much.

1.2 Do you always get the best solution? Why? What does it depend on?

We do not obtain always the best solution, cause of the first randomly generated solution. Sometimes the algorithm get trapped on a local maximum and it never reachs the global maximum.

1.3 Modify the code to start the search again from another initial solution (Iterated local search). Have you managed to improve? Why?

It obteins better results but needs more computation time. On my case I do it for 1,10 and 100 iterations.
Doing that It does not matter if the first random solution is not good, because along the execution of the algorithm it can reach a better neighbour. For each iteration it compares if the solution given is better than the best solution ever and if it is better it store the value of it.

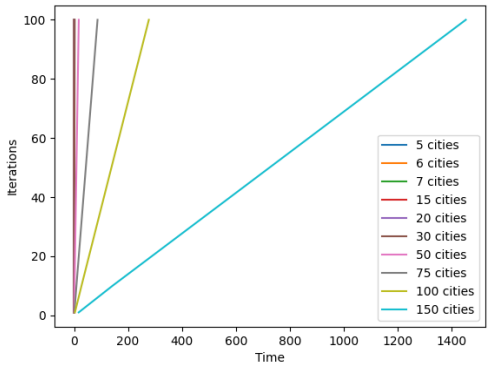

Figure 3: Computation time changing the iterations

2 Exercise 2.

2.1 How does this algorithm behave as we increase the problem (number of cities in the TSP)?

As I have done with the Hill Climbing algorithm I am going to evaluate the algorithm measuring the time and the legnth of the final path, to do this I have done the mean of five iterations for each city using the same cities as the previous exercise.

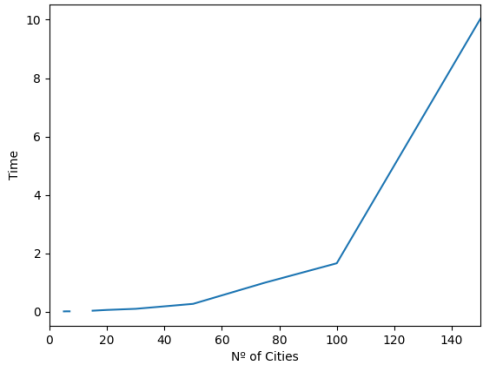

Figure 4: Time complexity of Simmulated Annealing

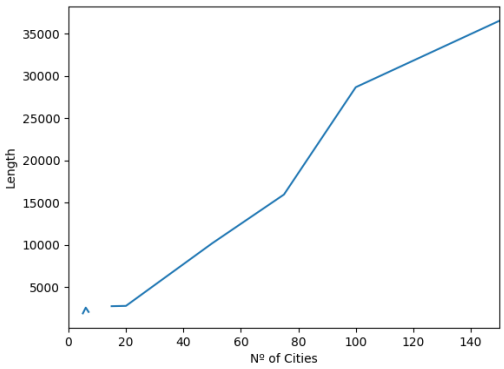

Figure 5: Length of final path Simmulated Annealing

2.2 Do you always get the best solution? Why? What does it depend on?

This algorithm not always obtains the best solution, it depends on the initial temperature and the stop condition.
When the temperature is too high we have more probabilities to found the best solution, also we can start near the best solution and get far away and do not find it.

2.3 Analyze how the behavior of the algorithm varies as we change the stop criteria and the initial temperature.

I have evaluated the algorithm with 75 cities and 5 iterations for each value of initial temperature and stop condition.

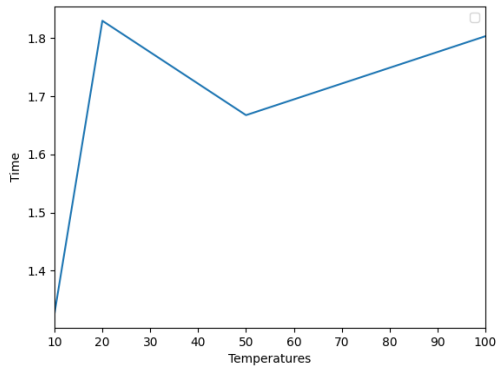

Figure 6: Execution time of Simmulated Annealing changing the initial temperature

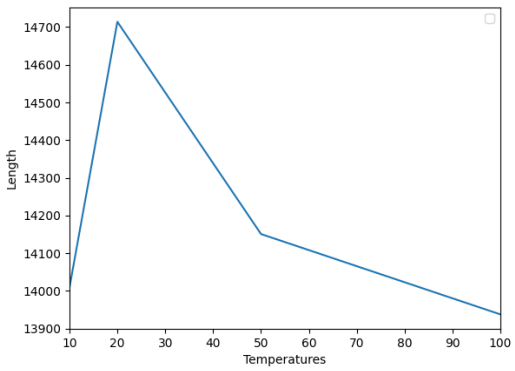

Figure 7: Length of the solution of Simmulated Annealing changing the initial temperature

As I had thought when you increase the initial temperature you are very close to the best solution, however the computational time increases.

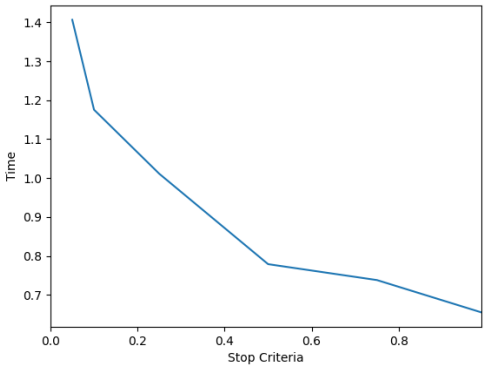

Figure 8: Execution time of Simmulated Annealing changing the stop condition

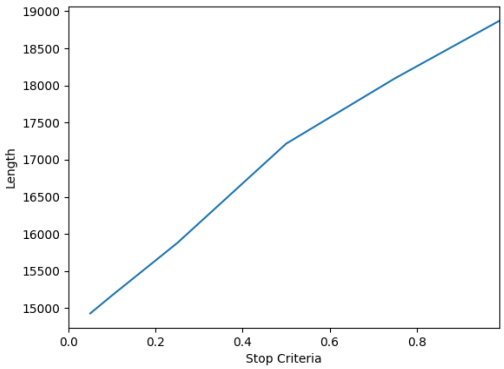

Figure 9: Length of the solution of Simmulated Annealing changing the stop condition

As we can see when the stop condition is too high the algorithm stop to early.

2.4 Modify the code to use different cooling functions

2.4.1 Logarithmic.

```
t = (alpha * t)/math.log(1+it)
```

2.4.2 Geometric.

```
t = math.pow(alpha,it)*t
```

2.5 How do these functions affect the final results? Why? Help yourself by representing the values of these functions. Look for new features and compare them to the old ones.

Changing the cooling function I have evaluate the algorithm using a range between 5 and 150 cities and 5 iterations for each one.

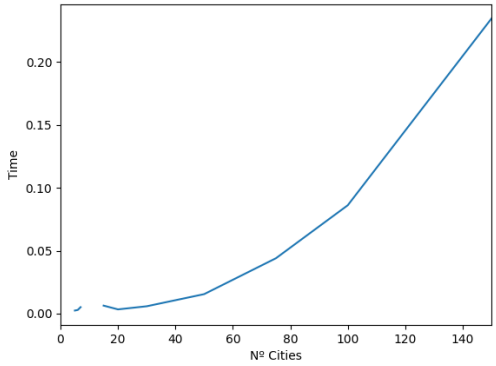

Figure 10: Time complexity using Geometric cooling function

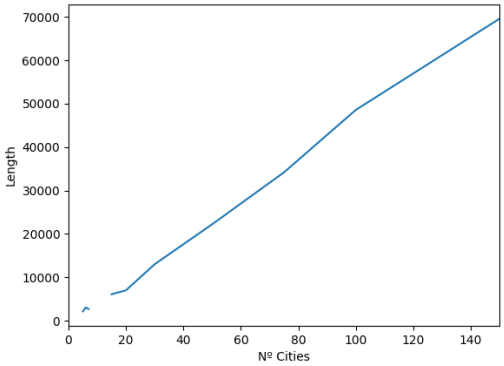

Figure 11: Length of the solution using Geometric cooling function

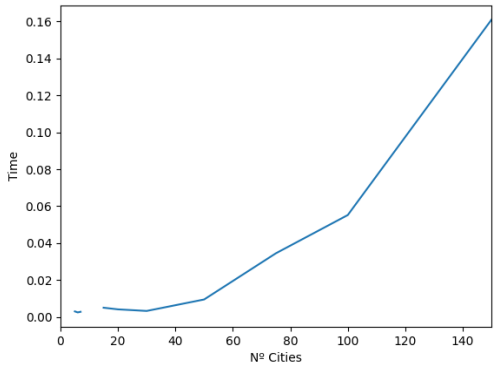

Figure 12: Time complexity using Logarithmic cooling function

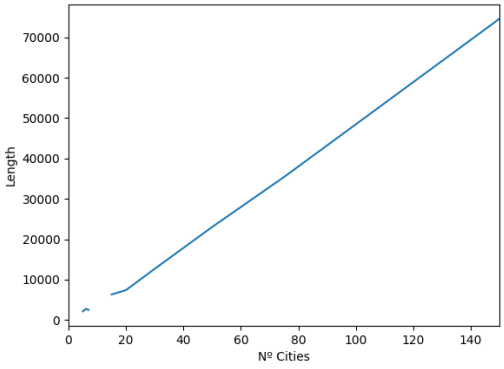

Figure 13: Length of the solution using Logarithmic cooling function

2.6 How would you improve the algorithm? For example, reheating so often. Modify the code with this and any other improvement you guess is appropriate. Analyze the results.

To improve that algorithm I would take some time evaluating all the possibles combinations of parameters to found the best one.