Assignment 1

Ventura Lucena Martínez

March, 2021

Contents

List of Figures

Chapter 1

Introduction

The aim of this report is to focus on the development of both local and global search techniques. Specifically, we focused on Hill Climbing and Simulated Annealing techniques in which we analyze different parameters such as their behavior when the size of the problem increases, or its complexity.

Chapter 2

Hill Climbing

On the first hand, we have worked with a local search using the Python code "HillClimbing.py" provided in Moodle, which corresponds to a generic Steepest-Ascent Hill Climbing algorithm to solve the "Travelling Salesman Problem". Hill Climbing tries to find the best solution to this problem by starting out with a random solution. Once this solution is obtained, then neighbours are generated. This means the following solutions slightly differ from the previous one obtained. If one of these neighbours provides a better solution, then it replaces the current solution. In case there is no better solution, it returns the current solution as the best one.

It is also important to note that it does not always provides the best solution to the problem at hand, as it can sometimes "get stuck" in a local maximum; that is, a point where the current solution is not the best solution to the problem, but where none of the direct neighbors of the current solution is better than the same. Therefore, a non-optimal solution will be returned.

2.1 Default size

A total of 5 executions have been carried out on the source code example, which corresponds to the following distance matrix:

$$\begin{bmatrix} 0 & 400 & 500 & 300 \\ 400 & 0 & 300 & 500 \\ 500 & 300 & 0 & 400 \\ 300 & 500 & 400 & 0 \end{bmatrix} \tag{2.1}$$

The results obtained were the following:

Iteration 1	
Route 1 length:	1400
Time:	8.368492126464844e-05 seconds
Final solution:	Final solution: [3, 2, 1, 0]
Final route length:	1400

Iteration 2	
Route 1 length:	1800
Route 2 length:	1400
Time:	0.00010895729064941406 seconds
Final solution:	[3, 0, 1, 2]
Final route length:	1400

Iteration 3	
Route 1 length:	1600
Route 2 length:	1400
Time:	0.000118255615234375 seconds
Final solution:	[0, 3, 2, 1]
Final route length:	1400

Iteration 4	
Route 1 length:	1800
Route 2 length:	1400
Time:	0.00011372566223144531 seconds
Final solution:	[1, 2, 3, 0]
Final route length:	1400

Iteration 5	
Route 1 length:	1800
Route 2 length:	1400
Time:	0.00011038780212402344 seconds
Final solution:	[0, 3, 2, 1]
Final route length:	1400

2.2 N size

The next step has been to check how the algorithm behaves as we increase the size of the problem considerably, expanding the code with the matrix generator of the "TSPGenerator.py" file. 5 algorithm executions have been carried out, with an increase in the size of the problem from 10 to 200 cities out of 10 in 10. The results were stored in 5 text files, named "data_x.txt" [1], with "x" the corresponding execution of the algorithm. Finally, the results obtained have been

[1] See "Random_HillClimbing.py" to check or modify the source code.

shown in a graph, which can be seen in figure 2.1, which clearly shows how the evolution satisfies an exponential curve:

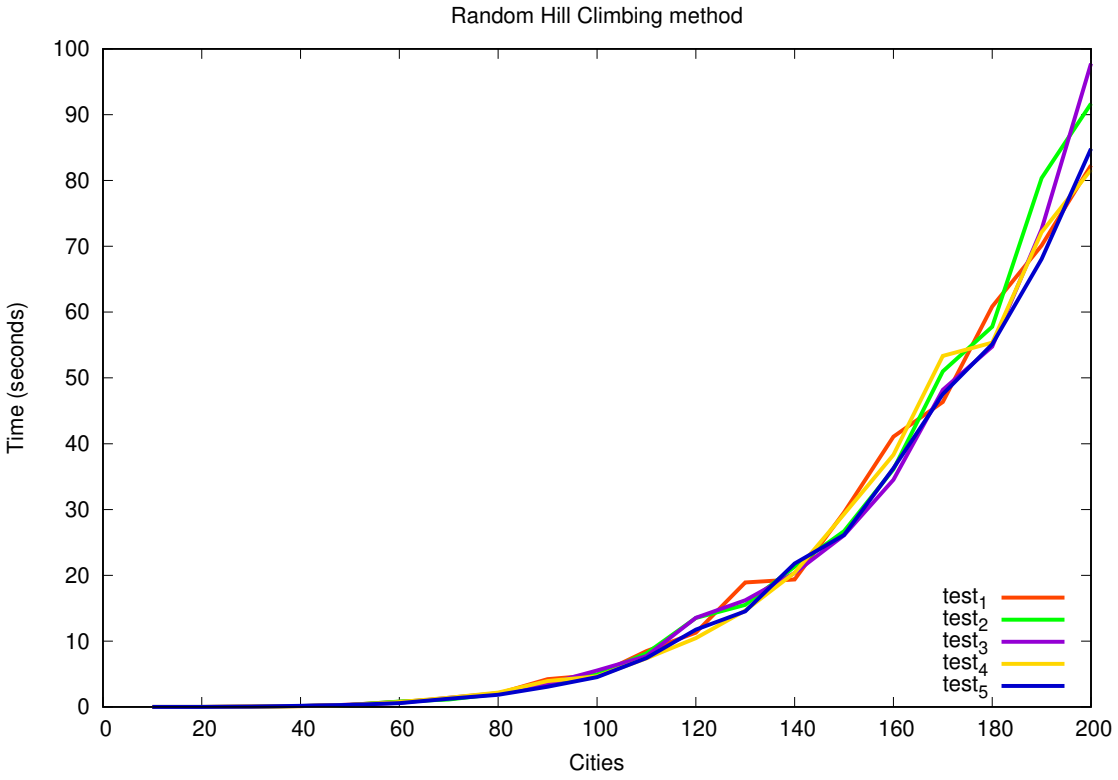

Figure 2.1: Random Hill Climbing method test.

The optimal solution to the problem is not always obtained: the algorithm can get stuck in a local maximum, in such a way that this situation prevents it from reaching the optimal solution.

2.3 Iterated local search

In this case, the algorithm has been executed iteratively and with the same dataset, in such a way that it can be observed if, in any case, a better result is obtained than the respective one in section 2.2.[2] The results for samples of 10, 50, 100, 150, and 200 are as follows:

10 cities			
Iterated		**Random**	
Best path	**Time (seconds)**	**Best path**	**Time (seconds)**
1430	0.001196146011352539	1769	0.0005662441253662109
1642	0.0009441375732421875		
1430	0.0009124279022216797		
1585	0.0012199878692626953		
1585	0.0006325244903564453		

[2]The results obtained for the same study range as in the previous section (10-200) have been stored in the directory "iterated_hill_climbing_data".

50 cities			
Iterated		**Random**	
Best path	**Time (seconds)**	**Best path**	**Time (seconds)**
6913	0.32426953315734863	6829	0.25955700874328613
6726	0.26101112365722656		
6306	0.327211856842041		
6442	0.3137209415435791		
6097	0.2676558494567871		

100 cities			
Iterated		**Random**	
Best path	**Time (seconds)**	**Best path**	**Time (seconds)**
10309	5.4097795486450195	9471	6.133375644683838
9578	5.346598148345947		
9032	6.175368547439575		
10336	4.696484565734863		
10077	5.701991319656372		

150 cities			
Iterated		**Random**	
Best path	**Time (seconds)**	**Best path**	**Time (seconds)**
13734	30.043731212615967	13285	25.468738794326782
13505	26.251903533935547		
12940	31.102091550827026		
12085	34.80124354362488		
12869	31.5150146484375		

200 cities			
Iterated		**Random**	
Best path	**Time (seconds)**	**Best path**	**Time (seconds)**
15620	94.37819290161133	15278	90.07622241973877
15853	101.46241903305054		
16012	88.86783027648926		
15270	106.33706521987915		
15717	96.85107588768005		

Observing the data, we see that the differences are not very high, although a better result is obtained in the Iterated Local Search. This may be mainly due to the fact that several passes are made through the graph, in such a way that there is a high probability of improving the result, although, a priori, this does not mean that one method is better than another.

Chapter 3

Simulated Annealing

On the second hand, we have worked with a global search using the Python code "SimAnnealing.py". Simulated Annealing is a metaheuristic search algorithm for global optimization problems; the general objective of this type of algorithm is to find a good approximation to the optimal value of a function in a large search space. This optimum value is called the "global optimum".

3.1 N size

In this case, we have directly studied how the algorithm behaves as the size of the problem increases. In the same way as in the previous method, with 5 different samples the behavior of the algorithm[1] has been evaluated for a range of (10-200) cities, with a temperature with a decreasing range of (10-0.5). The result is as follows:

[1] See "Random_SimAnnealing.py" to check or modify the source code.

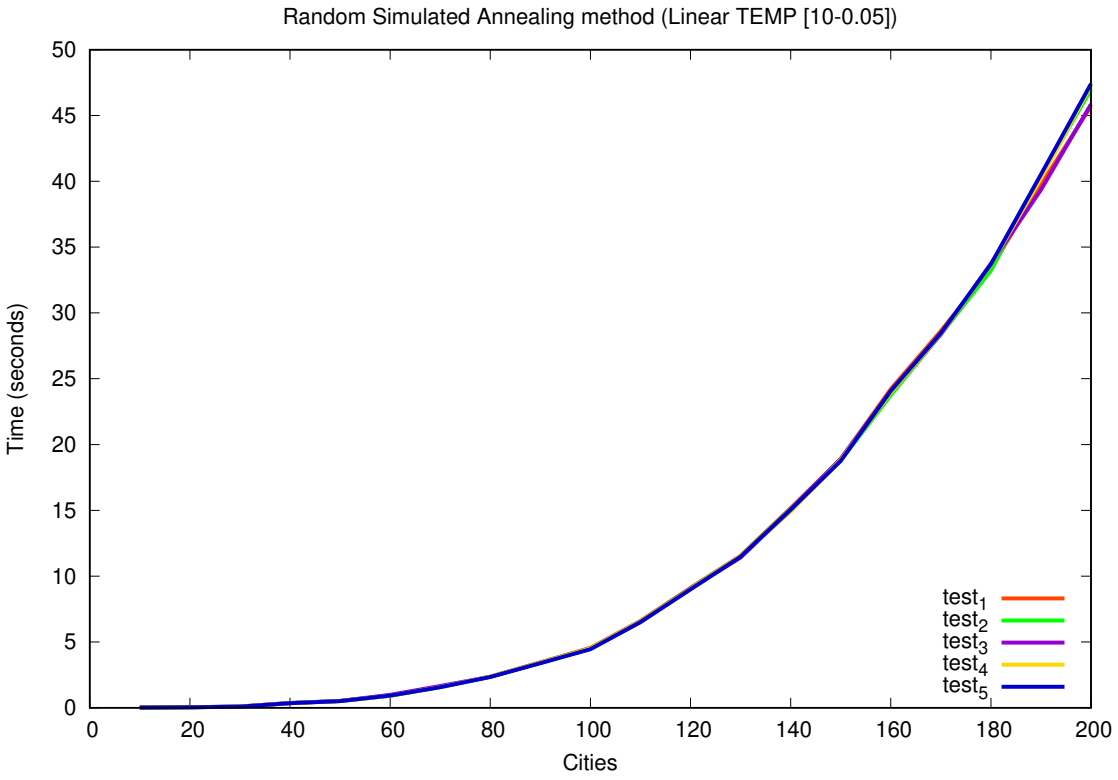

Figure 3.1: Random Simulated Annealing method test (Linear TEMP [10 - 0.5]).

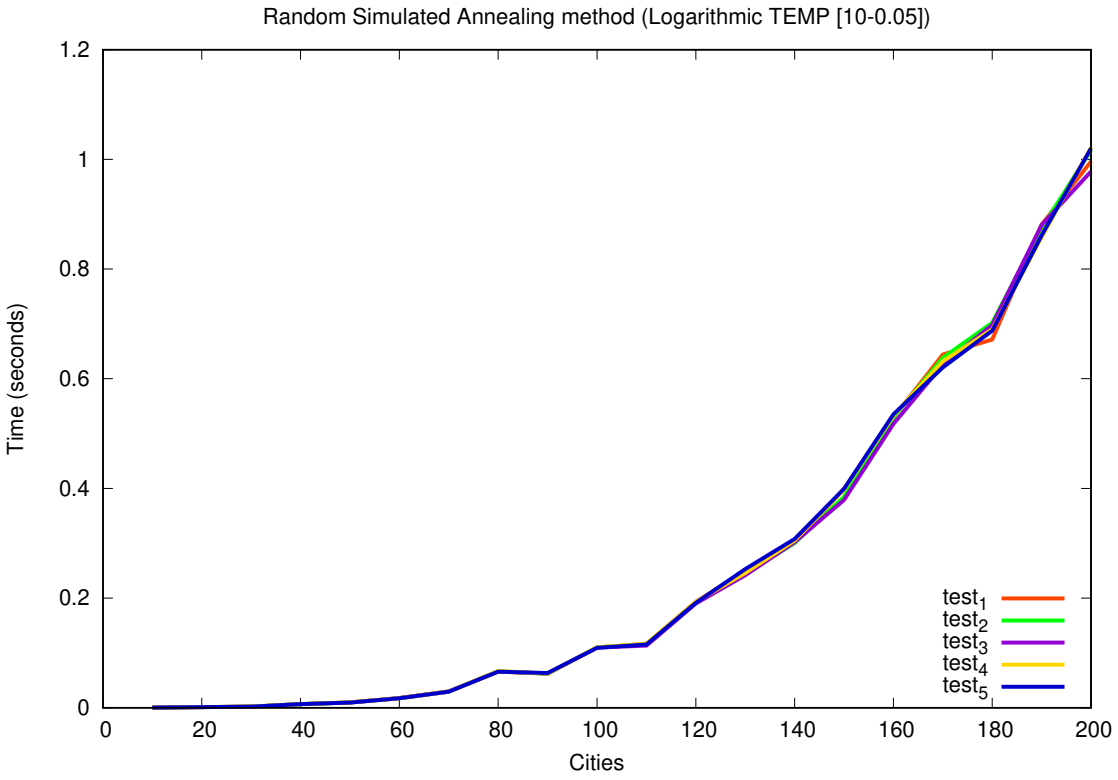

Figure 3.2: Random Simulated Annealing method test (Logarithmic TEMP [10 - 0.5]).

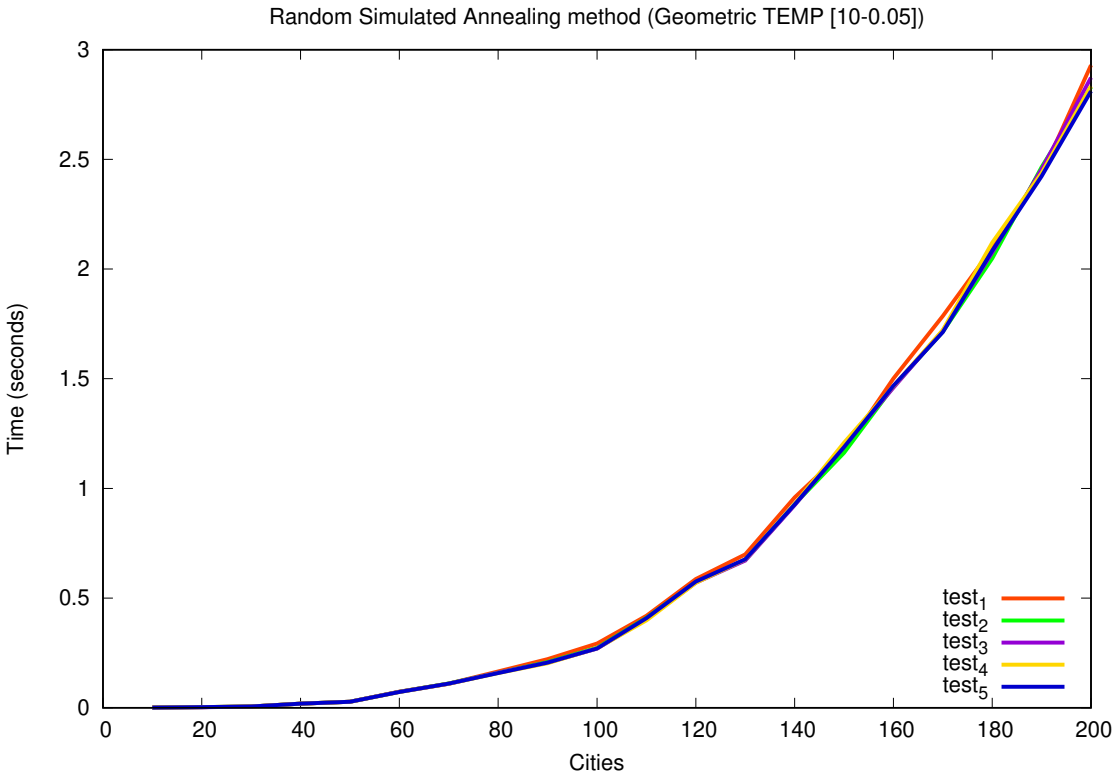

Figure 3.3: Random Simulated Annealing method test (Geometric TEMP [10 - 0.5]).

It can be seen that this method is much faster (with the default settings) than the previous one at section 2.2, but at the cost of much less efficiency. In sections 3.2.1, 3.2.2 and 3.2.3 we will study in more detail:

3.2 Iterated global search

As the previous section 2.3, the algorithm has been executed iteratively and with the same dataset, in such a way that it can be observed if, in any case, a better result is obtained than the respective one in section 3.1.[2] The results for samples of 10, 50, 100, 150, and 200 are as follows:

[2]The results obtained have been stored in "simulated_annealing_data" directory. Each subdirectory contains its own "README.txt"

3.2.1 Linear

10 cities	
Iterated	
Best path	**Time (seconds)**
2926	0.0015952587127685547
3113	0.0014204978942871094
2071	0.0013260841369628906
2509	0.001435995101928711
3084	0.0013041496276855469

50 cities	
Iterated	
Best path	**Time (seconds)**
7983	0.7355279922485352
8554	0.6342146396636963
8755	0.632047176361084
8212	0.636631965637207
8213	0.6155929565429688

100 cities	
Iterated	
Best path	**Time (seconds)**
23236	5.902791976928711
21949	5.725757837295532
21015	5.827342987060547
22593	5.748596668243408
21725	5.807369232177734

150 cities	
Iterated	
Best path	**Time (seconds)**
32586	24.413987398147583
33569	24.96064281463623
34547	25.138503313064575
31883	25.318193197250366
34498	25.210740566253662

200 cities	
Iterated	
Best path	**Time (seconds)**
48254	66.03151535987854
46499	65.72531199455261
48709	65.89467072486877
49417	65.1207184791565
50379	64.82171130180359

3.2.2 Logarithmic

10 cities	
Iterated	
Best path	**Time (seconds)**
4474	0.0006887912750244141
2951	0.0006072521209716797
3152	0.0006051063537597656
2995	0.0006165504455566406
3286	0.0005900859832763672

150 cities	
Iterated	
Best path	**Time (seconds)**
66597	0.5170221328735352
66895	0.48668837547302246
70207	0.42569875717163086
74449	0.4956526756286621
68726	0.4281330108642578

50 cities	
Iterated	
Best path	**Time (seconds)**
21312	0.017411231994628906
20314	0.014244794845581055
23291	0.014613151550292969
21130	0.013487100601196289
22962	0.013506650924682617

200 cities	
Iterated	
Best path	**Time (seconds)**
97190	1.2930617332458496
100814	1.1365227699279785
99250	1.139885663986206
101414	1.2001094818115234
94724	1.1812944412231445

100 cities	
Iterated	
Best path	**Time (seconds)**
43503	0.11750435829162598
50777	0.14738082885742188
44248	0.14747381210327148
42562	0.11710190773010254
47004	0.13296842575073242

3.2.3 Geometric

10 cities	
Iterated	
Best path	**Time (seconds)**
2926	0.0015952587127685547
3113	0.0014204978942871094
2071	0.0013260841369628906
2509	0.001435995101928711
3084	0.0013041496276855469

50 cities	
Iterated	
Best path	**Time (seconds)**
19834	0.04161262512207031
17201	0.03303194046020508
22035	0.05698966979980469
18327	0.034420013427734375
20594	0.03327059745788574

100 cities	
Iterated	
Best path	**Time (seconds)**
41557	0.3235898017883301
40242	0.3383448123931885
40911	0.26884889602661133
44240	0.24174833297729492
39408	0.2564363479614258

150 cities	
Iterated	
Best path	**Time (seconds)**
61784	1.2619507312774658
68597	1.2039363384246826
67193	1.1346631050109863
64160	1.1485040187835693
70517	1.1908643245697021

200 cities	
Iterated	
Best path	**Time (seconds)**
94369	3.0945987701416016
90121	2.941612482070923
104320	2.8436856269836426
95850	2.987704277038574
100847	2.93882417678833

Chapter 4

Conclusion

After numerous tests, changing the corresponding parameters of each method, it has been concluded that the hill climbing method has had a better performance and efficiency in finding optimal solutions to the TSP problem with the datasets created. There are large differences in execution times and solutions provided between the two methods, although this may be improved by changing parameter values. None of the changes therein have caused Simulated Annealing to provide better data than Hill Climbing.

The conclusive data are as follows:

NOTE:

- HC = Hill Climbing.
- SA - L = Linear Simulated Annealing.
- SA - Log = Logarithmic Simulated Annealing.
- SA - G = Geometric Simulated Annealing.

n = 10	**Best Path**	**Time (seconds)**
HC	1430	0.001196146011352539
SA - L	2071	0.0013260841369628906
SA - Log	2951	0.0006072521209716797
SA - G	2071	0.0013260841369628906

n = 50	**Best Path**	**Time (seconds)**
HC	6097	0.2676558494567871
SA - L	7983	0.7355279922485352
SA - Log	20314	0.014244794845581055
SA - G	17201	0.03303194046020508

n = 100	**Best Path**	**Time (seconds)**
HC	9032	6.175368547439575
SA - L	21015	5.827342987060547
SA - Log	42562	0.11710190773010254
SA - G	39408	0.2564363479614258

n = 150	Best Path	Time (seconds)
HC	12085	34.80124354362488
SA - L	31883	25.318193197250366
SA - Log	66597	0.5170221328735352
SA - G	61784	1.2619507312774658

n = 200	Best Path	Time (seconds)
HC	15270	106.33706521987915
SA - L	46499	65.72531199455261
SA - Log	94724	1.1812944412231445
SA - G	90121	2.941612482070923

Appendix A

Source Files

The source files used in each section have been the following:

- **Random_HillClimbing.py:** section 2.1.
- **Iterated_HillClimbing.py:** section 2.2.
- **Random_SimAnnealing.py:** section 3.1.
- **SimAnnealing.py:** section 3.2.

The following scripts have been used to create the charts:

- **random_hill_climbing_chart_generator.sh**.
- **simulated_annealing_chart_generator.sh**.

Appendix B

Simulated Annealing parameters

Throughout the study, different configurations of the parameters of each of the variations of this method have been tested, none of them being better than those that were used. The values can be observed in their respective test.

Bibliography

[1] Practice 1 - Introduction to the lab sessions.

[2] Practice 1 - Source code.

[3] Practice 1 - Statement.

[4] How to Implement the Hill Climbing Algorithm in Python.