University of Córdoba

Computer Science Engineering Degree
3rd course

Metaheuristics

Hill Climbing and Simulated Annealing

Valentín Gabriel Avram Aenachioei
Alexandra-Maria Borsan
Aicha Bracken-Soliman
Radu Ciurca
Barbara Kereselidze
Victor Rojas Muñoz
Davit Tchanturia

Academic year 202-2023
Córdoba, June 9, 2023

Contents

Tables index **ii**

Images Index **iii**

1 Hill Climbing **1**

1.1 Analyze the algorithm's performance when we increase the problem complexity (search space). Take the number of cities to do that 1

1.2 Does the algorithm always get the best solution? Why? What does it depend on? 3

1.2.1 Initial Solution 3

1.2.2 Local maximum/minimum: 3

1.2.3 Plateau: 3

1.3 Iterated Local Search. Does the algorithm always gets the best solution? Why? What does it depend on? 4

1.3.1 Solution Quality 4

1.3.2 Runtime 4

1.3.3 Parameter Sensitivity 4

1.4 Compare both algorithms to demonstrate which one performs better 5

2 Simulated Annealing **6**

2.1 Analyze the algorithm's performance when we increase the problem complexity (search space). Take the number of cities to do that 6

2.2 Does the algorithm always get the best solution? Why? What does it depend on? 8

2.3 Modify the code to consider different temperatures, stopping criteria and cooling functions. Does the algorithm always gets the best solution? Why? What does it depend on? What is the role of the cooling function on the final performance? . . . 9

2.4 Compare all the algorithms (Hill Climbing and Simulated Annealing) to demonstrate which one performs better. 14

List of Tables

1 Hill Climbing performance 1
2 Algorithm performance comparison 6
3 Simulated Annealing perfomance 6
4 Simulated Annealing perfomance 10

List of Figures

1 Hill Climbing performance 2
2 Algorithms performance comparison 5
3 Simulated Annealing performance 7
4 First cooling function performance 11
5 Second cooling function performance 12
6 Third cooling function performance 13
7 Algorithms performance comparison 15

In this paper, we will set out how we have done the first parctical assignment, *Hill Climbing and Simulated Annealing*, answering the questions asked in the instructions of the lab assignment.

1 Hill Climbing

1.1 Analyze the algorithm's performance when we increase the problem complexity (search space). Take the number of cities to do that

When the problem's complexity increases, the number of possible solutions grows considerably, i.e. the search space increases. Therefore, the execution time is going to be longer and the solution is not going to be as precise as it would have been if the number of stops has been lower. Therefore, we would need more iterations/repetions to increase the probability of obtaining the optimal solution, if the number of possible solutions is bigger.

The table 1 shows the average performance of the algorithm over all datasets, in this case, trying a 100 iterations.

Dataset	**Average runtime**	**Accuracy**	**Accuracy / Runtime**
FIVE	0.557124805	83.2 %	0.006696212
P01	0.851062727	17.4 %	0.048911651
GR17	1.078659344	0 %	0
FRI26	3.457708168	0 %	0
DANTZIG42	22.83137789	0 %	0
ATT48	45.15664773	0 %	0

Table 1: Hill Climbing performance

The image 1 shows the average performance of the Hill Climbing algorithm over all datasets, in this case, trying a 100 iterations, it is the graphical representation of the previous data.

It can be observed that the accuracy of the algorithm drops to zero as the dataset size and the search space increases. Likewise, the computation time increases as the dataset size and the search space increases.

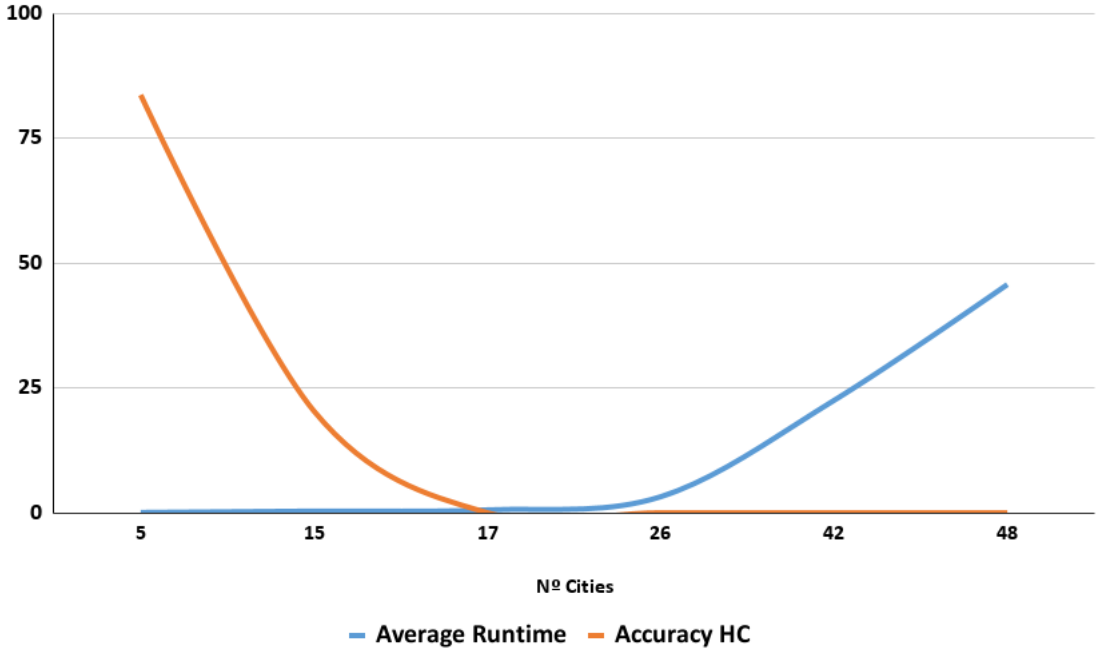

Figure 1: Hill Climbing performance

1.2 Does the algorithm always get the best solution? Why? What does it depend on?

Finding the best solution depends on several factors, such as the size of the search space, the initial solution from which we start, and the number of iterations or attempts of the algorithm.

The algorithm does not always provide the best solution due to the different regions in Hill Climbing. In smaller datasets, which means smaller search spaces, it is less likely to get stuck in a local minima. In bigger search spaces, the algorithm will only find the optimal solution if the solution at any given moment does not approach the basin of attraction of any local minimum.

If the algorithm reaches any of the following, it's not going to be optimal:

1.2.1 Initial Solution

It is crucial that the quality of the initial solution is as optimal as possible since it will have a huge impact on the final solution. If the final and the initial solution are quite far away the algorithm might not even be able to find the solution even after going through the entire search space, where it can be trapped in a local minima.

1.2.2 Local maximum/minimum:

At a local maximum/minimum all neighboring states have a value that is worse than the current state. Since hill-climbing uses a greedy approach, it will not move to the worse state and terminate itself. The process will end even though a better solution may exist.
To overcome the local maximum problem: Utilize the backtracking technique. Maintain a list of visited states. If the search reaches an undesirable state, it can backtrack to the previous configuration and explore a new path. Theoretically this is a good solution, but in practice, it is unusual to have that much memory available.

1.2.3 Plateau:

On the plateau, all neighbors have the same value. Hence, it is not possible to select the best direction.

To overcome plateaus: Make a big jump. Randomly select a state far away from the current state. Chances are that we will land in a non-plateau region. This process is a separate metaheuristic, named Multi Start Local Search.

1.3 Iterated Local Search. Does the algorithm always gets the best solution? Why? What does it depend on?

As happens when applying Hill Climbing, the algorithm does not always provide the best solution, due to the same problems as Hill Climbing, and also, the perturbation done to the solution may not be enough to escape the basin of attraction of a local minimum.

1.3.1 Solution Quality

The possibility of both algorithms getting stuck, for instance in the local optimum is always present, this can lead to suboptimal solutions, however, if we compare the two, iterated local search is less likely to get stuck in the local optimum due to it's random perturbation steps.

1.3.2 Runtime

Both algorithms runtime depends on the size of the input (the number of cities) but local search algorithm might be faster since it performs simple search, while the ILS applies perturbations to random solutions which leads to more iterations and bigger a runtime. On the other hand, if the initial solution is not good enough and perturbations steps are smaller, ILS might have a faster performance.

1.3.3 Parameter Sensitivity

Both algorithms are significantly dependent on the parameters provided to them, number of cities, perturbation steps, initial solution, number of iterations. We will consider the Datasets, Average Runtime and Accuracy in order to analize if the algorithm provides the best solutions and how it varies.

1.4 Compare both algorithms to demonstrate which one performs better

The image 2 shows the average performance of the Iterated Local Search algorithm over all datasets, in this case, trying a 100 iterations. It is a graphical representation of the average performance.

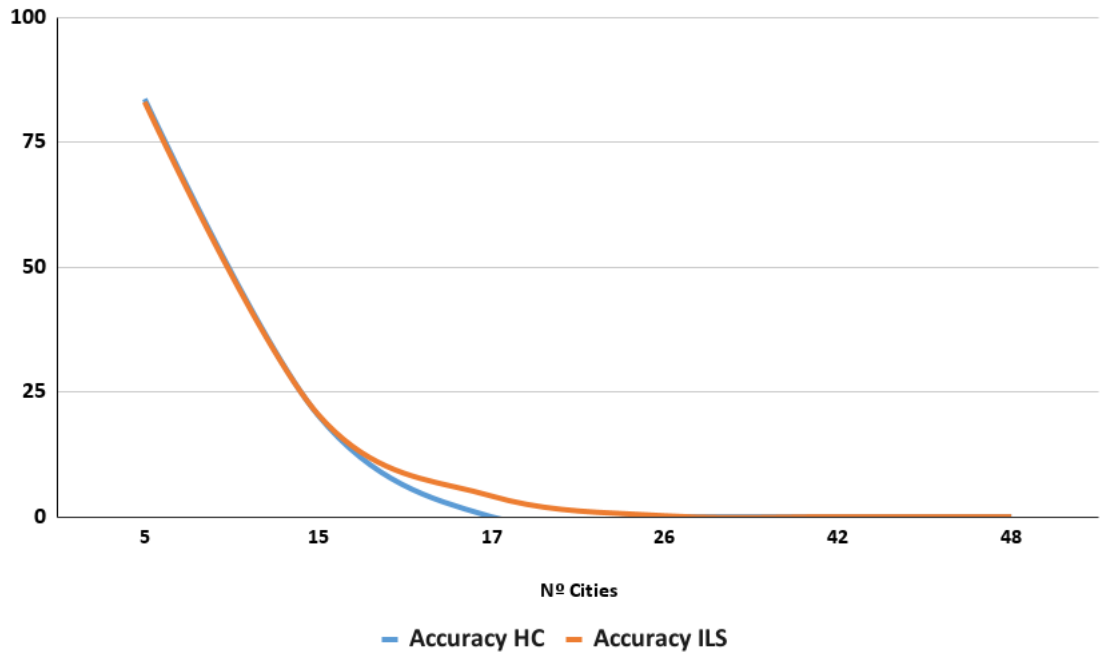

Figure 2: Algorithms performance comparison

As a comparison, he table 2 shows the average performance of the algorithms over all datasets, in this case, trying a 100 iterations.

We can observe that using the Iterated Local Search algorithm slightly increases the accuracy, especially on larger datasets, at the cost of increased runtime.

Dataset	Hill Climbing Runtime	Hill Climbing Accuracy	ILS Runtime	ILS Accuracy
FIVE	0.557124805	83.2%	0.654697	83.4%
P01	0.851062727	17.4%	0.92799	17%
GR17	1.078659344	0%	1.198201	4.6%
FRI26	3.457708168	0%	3.596329	0.4%
DANTZIG42	22.83137789	0%	23.24956	0%
ATT48	45.15664773	0%	49.28776	0%

Table 2: Algorithm performance comparison

2 Simulated Annealing

2.1 Analyze the algorithm's performance when we increase the problem complexity (search space). Take the number of cities to do that

In the same way as in Hill Climbing or ILS, when the search space increases, the execution time is going to be longer and the solution is not going to be as precise as it would have been in a smaller dataset. This is caused by the same reasons as in previous algorithms, such as the initial random solution. Even if we manage to escape the basin of attraction of some of the local minimums., this slight improvement proves a better accuraccy in smaller search spaces, but in bigger problems where the algorithm does not get close to the optimal solution, the accuracy will be zero.

The table 3 shows the average performance of the Simulated Annealing algorithm over all datasets, in this case, trying a 100 iterations.

Dataset	Average Runtime	Accuracy	Accuracy / Runtime
FIVE	0,453733059	100%	0,004537331
P01	1,966794605	11,46%	0,171556033
GR17	2,487586801	1,02%	2,438810589
FRI26	5,117269764	0%	0
DANTZIG42	24,78515877	0%	0
ATT48	33,7302733	0%	0

Table 3: Simulated Annealing perfomance

The image 3 shows the average performance of the Simulated Annealing algorithm over all datasets, in this case, trying a 100 iterations. It is a graphical representation of the average performance, using the average obtained results using all possible combinations of initial temperature and cooling function.

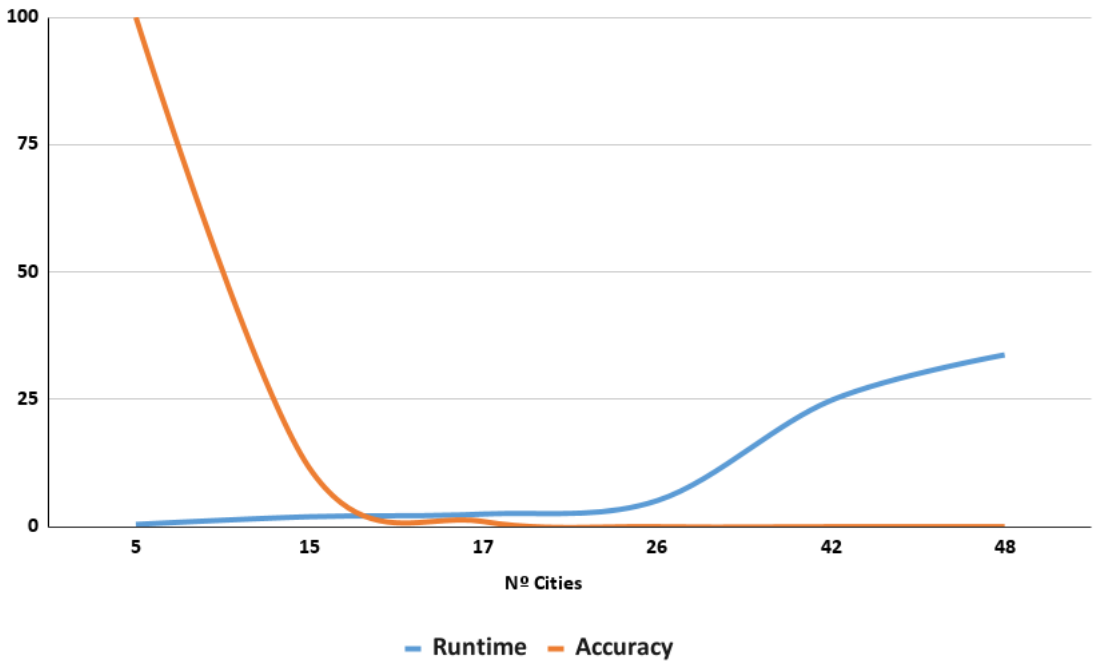

Figure 3: Simulated Annealing performance

2.2 Does the algorithm always get the best solution? Why? What does it depend on?

As in the previous algorithms, it cannot always obtain the optimal solution, even if it can improve the accuracy.
Due to the way the probability is calculated, when the temperature is higher, is it more likely that the algorithm accepts a worse solution. This promotes **Exploration / Diversification** of the search space and allows the algorithm to more likely travel down a sub-optimal path to potentially find a global minimum.
When the temperature is lower, the algorithm is less likely or will not accept a worse solution. This promotes **Exploitation / Intensification**, which means that once the algorithm is in the right search space, there is no need to search other sections of the search space and should instead try to converge and find the global maximum. This effect of the temperature on the performace is also modified by the initial temperature used and the cooling function used to decrease the temperature per iteration.
To sum it up, due to the way we calculate the probability, the temperature we chose is going to have a significant impact on the algorithm. The advantages of this algorithm are that it is easy to implement and it provides optimal solutions to a wide range of problems. The disadvantages are that it can take long to run if the annealing schedule is very long and there are a lot of tunable parameters in this algorithm.

2.3 Modify the code to consider different temperatures, stopping criteria and cooling functions. Does the algorithm always gets the best solution? Why? What does it depend on? What is the role of the cooling function on the final performance?

The table 4 shows the average performance of the Simulated Annealing algorithm over the **P01** dataset, in this case, trying a 100 iterations. As experiments, we used different initial temperatue values, such as 5, 10 and 50.
As different cooling functions, we used:

1. $t = 0.99 \cdot t$
2. $t = \frac{t0}{1+(0.99 \cdot t)}$
3. $t = t0 \cdot (0.99 \cdot t)$

We did not tried any different stopping criterias, because with these parameters we wont reach a stopping criteria different than the number of iterations, or even reaching a new stopping criteria, the performance will not improve. Also, a good way to experiment the effect of the parameters in the performance could have been changing the acceptance function.

Cooling Function = $0.99 \cdot t$		
Initial Tº	**Runtime**	**Accuracy**
5	1,712038002	9,44%
10	2,090519886	13,28%
50	2,553160372	18,26%
Cooling Function = $\frac{t_0}{1+(0.99 \cdot t)}$		
Initial Tº	**Runtime**	**Accuracy**
5	0,4638439178	0,04%
10	0,8841618299	1,16%
50	3,747741013	19,82%
Cooling Function = $t_0 \cdot 0.99^t$		
Initial Tº	**Runtime**	**Accuracy**
5	1,706358733	10,48%
10	2,095985961	12,6%
50	2,447341728	18,1%

Table 4: Simulated Annealing perfomance

The image 4 shows the average performance of the Simulated Annealing algorithm with different values for initial temperature while usinge the first cooling function tested: $t = 0.99 \cdot t$.
The image 5 shows the average performance of the Simulated Annealing algorithm with different values for initial temperature while usinge the second cooling function tested: $t = \frac{t0}{1+(0.99 \cdot t)}$, and the image 6 does the same with the third cooling function: $t = t0 \cdot (0.99 \cdot t)$.
All three graphs are based on the average performance over all the datasets.

As a brief conclusion, we can see that, using all three cooling functions, the most influential factor is the initial temperature. With a lower initial temperature value, the stoppimg condition is reached earlier, which means lower runtimes, but also a lower accuracy, since less iterations are done. In the same way, a higher initial temperature value means a higher accuracy at the cost of increased runtime, since more iterations are done.

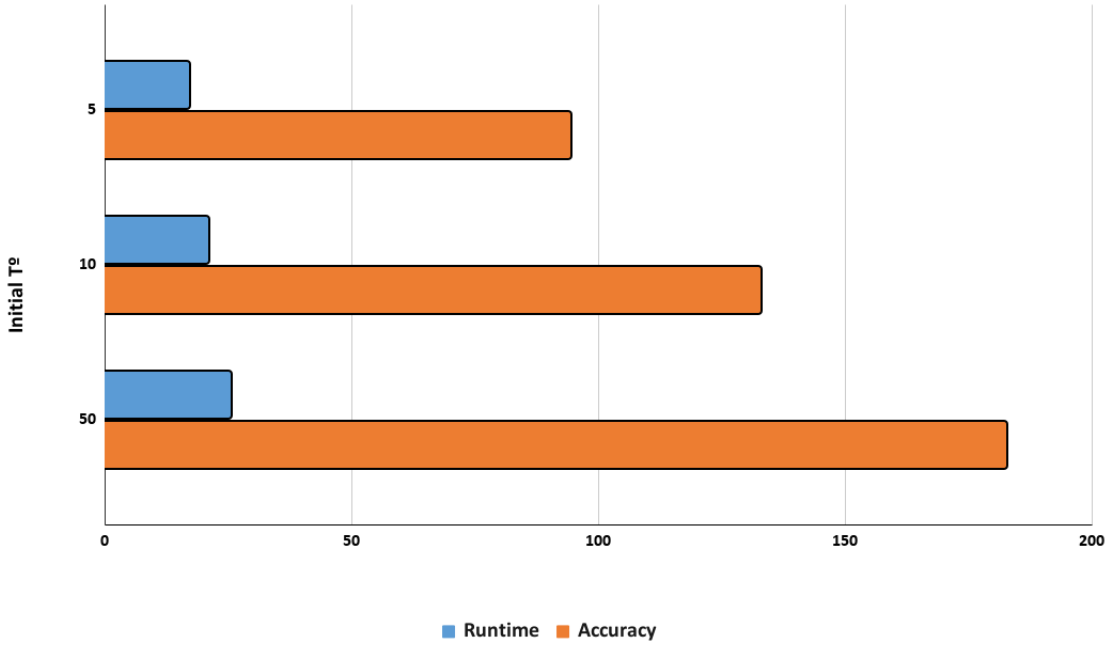

Figure 4: First cooling function performance

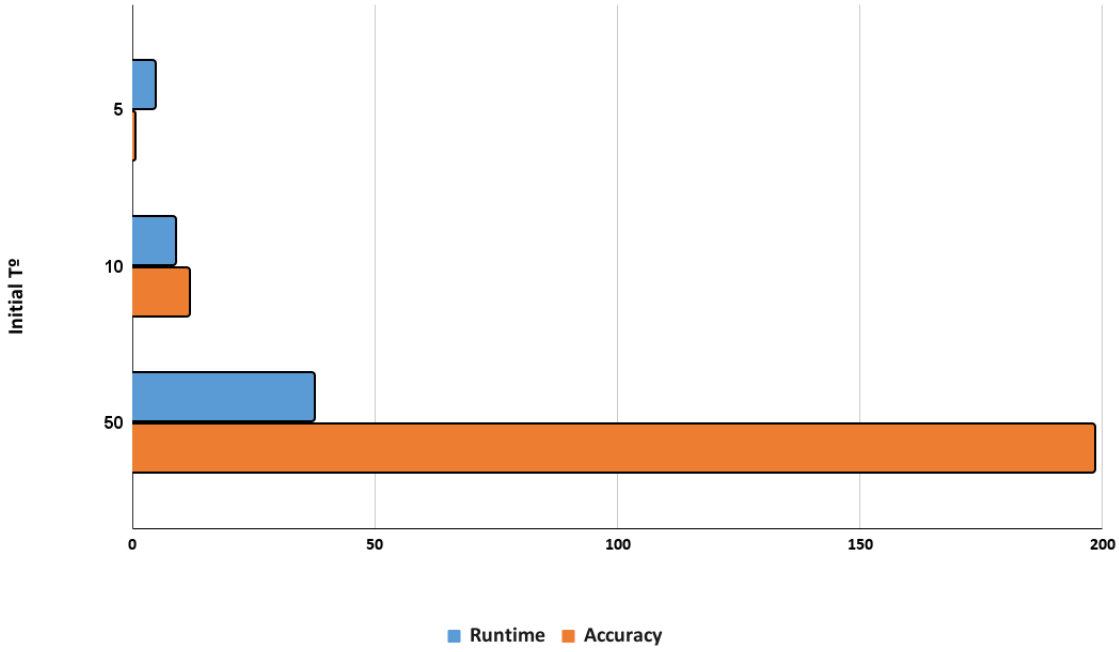

Figure 5: Second cooling function performance

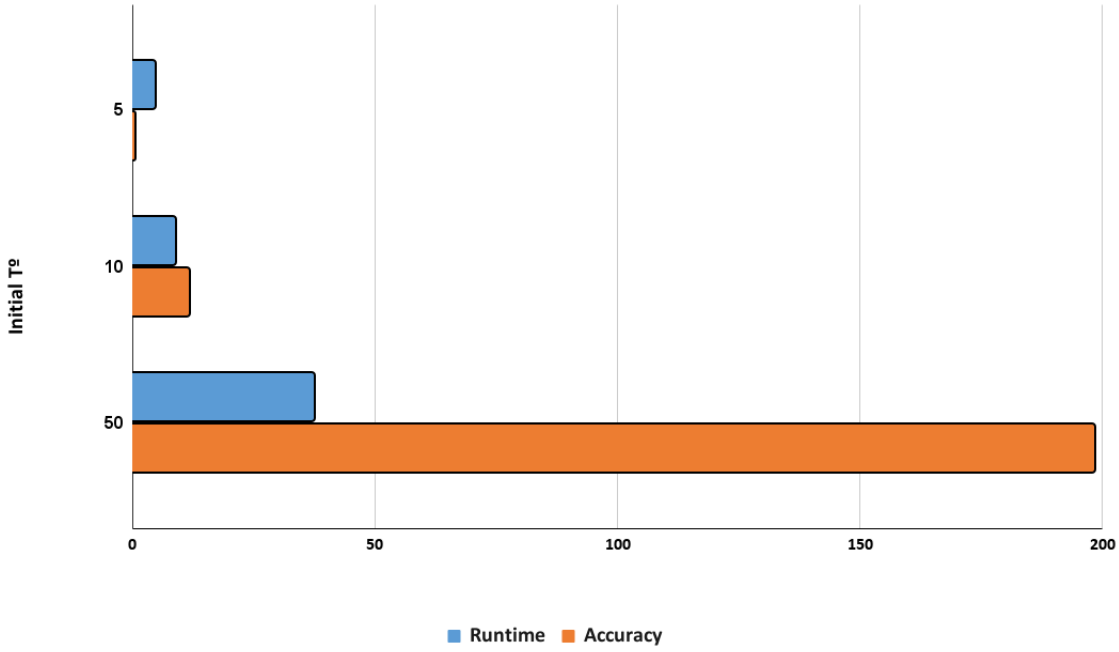

Figure 6: Third cooling function performance

2.4 Compare all the algorithms (Hill Climbing and Simulated Annealing) to demonstrate which one performs better.

Both Iterared Local Search and Simulated Annealing can be considered an improvement based on the Hill Climbing metaheuristic. Hill Climbing attempts to reach an optimal solution by checking if its current state has the best cost/score in its neighborhood, makign it prone to getting stuck in local optima.
Iterated Local Search tries to improve this problem by adding a random diversification component in each obtained solution. Even so, it may not be able to escape the basin of attraction of the local minimum.
Simulated Annealing manages to escape that basin of attraction, but even so, it does not manages to find the optimal solution in bigger search spaces.
We can conclude that, in smaller search spaces, all algorithms perform well, with few differences in performance, with Hill Climbing standing out for its shorter computation time. For bigger search spaces, all three algorithms have a worst perfomance, none of the algorithms prove to be worthy. The only point to note would be that Simulated Annealing is able to achieve 100% accuracy when working in small search spaces.

The image 7 shows a graphical comparison between the average performance of the 3 algorithms over all datasets.

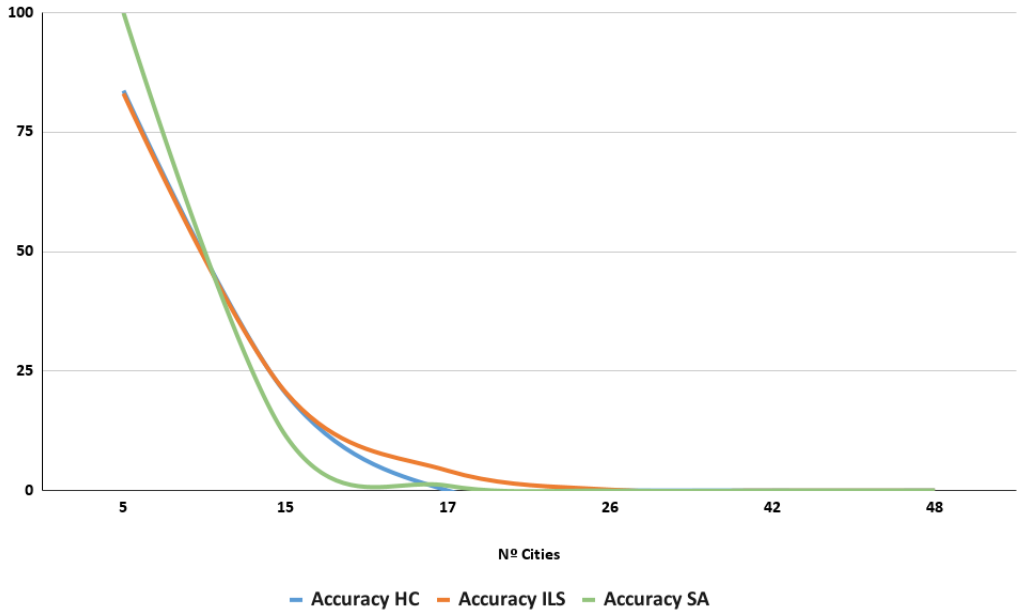

Figure 7: Algorithms performance comparison