Simulated Annealing vs Hill Climbing

Travelling Salesman Problem

Artificial Intelligence course project

Nikola Bulat

Computer Science Master’s Degree, University of Padua, Italy

nikola.bulat@studenti.unipd.it

September 2023

https://github.com/Bulat27/hill-climbing-vs-simulated-annealing

1 Introduction
This project deals with the comparison of the Simulated Annealing (SA) and Hill Climbing (HC) algorithms on the Travelling Salesman Problem (TSP). The goal is to test:

- How do different parameter values affect these algorithms?
- How effective these algorithms are on the TSP?
- Which one performs better?

It is important to emphasize that the idea was to test the simple versions of the algorithms in order to understand how they perform. Potential optimizations in further work are reported in the last section.

Sections 2, 3, and 4 provide a brief introduction to TSP, HC, and SA respectively. They can be skipped by anyone who has already read about these topics. Section 5 explains my choices regarding algorithmic and implementation specifics. In section 6, I have described the performed experiments and discussed the results. Section 7 states some of the conclusions and further considerations for this project.

2 Travelling Salesman Problem

2.1 Description
The **travelling salesman problem** (**TSP**) asks the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?". More formally, it is modeled as a graph problem. It is modelled as an undirected weighted graph, such that cities are the graph's vertices, paths are the graph's edges, and a path's distance is the edge's weight. It is usually modelled as a complete graph and the goal is to find the minimum weight cycle that traverses each vertex exactly once (except the origin vertex).

There are two versions of this problem: **symmetric TSP** and **asymmetric TSP**. In the symmetric TSP, the distance between two cities is the same in each opposite direction, forming an undirected graph. This symmetry halves the number of possible solutions. In the asymmetric TSP, paths may not exist in both directions or the distances might be different, forming a directed graph. In this project, I focus on the symmetric TSP problem.

2.2 Why TSP?
Why is TSP interesting for Local Search and Optimization algorithms? TSP belongs to the class of **NP-complete** problems which means (I will not be formal here to keep the report concise) that we must resort to heuristics, optimizations, and similar methods instead of hoping to solve it exactly. Moreover, the following theorem has been proven: For any function *f(n)* that can be computed in time polynomial in *n*, there is no polynomial-time *f(n)*-approximation algorithm for TSP, unless P = NP. Not only is it hard to solve TSP exactly, but it is also hard to approximate it with a tight bound. For these reasons, I decided to test simple local search algorithms on the TSP.

3 Hill Climbing

Hill Climbing is a heuristic optimization process that iteratively advances towards a better solution at each step in order to find the best solution in a given search space. It is a straightforward and quick technique that iteratively improves the initial solution by making little changes to it. Hill Climbing only accepts solutions that are better than the current solution and employs a greedy technique to iteratively move towards the best solution at each stage. Hill Climbing may not locate the global optimum because it is susceptible to becoming caught in local optima.

There are many variants of hill climbing algorithm. In the most basic version, the search considers all the neighbors and chooses the best one as the next candidate. There is also **Stochastic hill climbing** that chooses at random form among the uphill moves.

One implementation of this approach is **First-choice hill climbing** which generates successors randomly until one is generated that is better than the current solution. Another version is **random-restart hill climbing** that conducts many hill-climbing searches from randomly generated initial states.

4 Simulated Annealing

One of the main downfalls of the hill climbing algorithm is that it tends to get stuck **in the local optimum** as it never makes downhill moves. Moreover, it can get stuck because of **ridges** and **plateaus**. Adding some random walk into hill climbing search to escape these regions can help in finding a better local optimum or even the **global optimum**. Simulated annealing is an algorithm that combines uphill and downhill moves. Inspiration is drawn from the process of annealing in metallurgy.

The algorithm begins with a randomly generated initial solution and incrementally improves it by accepting the less desirable solutions with a certain probability while the better solution is always accepted. The probability of accepting a worse solution decreases as the temperature T goes down. Bad moves are more likely to be allowed at the start when T is high and less likely when it is low. This allows for exploration at the beginning of the algorithm and exploitation towards the end. The idea is that the algorithm will converge to a good local or even global optimum. If the schedule decreases the temperature to 0 slowly enough, then all the probability is concentrated on the global optima which the algorithm will find with probability approaching 1.

In practice, the performance of the SA algorithm heavily depends on the parameters such as initial temperature, cooling schedule, end temperature, acceptance probability function, etc. Proper tuning of these parameters should allow for slow (but not too slow to be efficient) convergence towards a good solution. Adequate cooling schedules are often problem-specific, and researchers have derived a multitude of parameter sets that allow for adaptive cooling schedules.

5 Methodology
Algorithms, data loading, and analysis are implemented in Python programming language. I haven't used any libraries that provide optimized implementations, but I have implemented simple versions of the algorithms from scratch using the pseudocode from the course book and referring to a few blogs from the internet. I report some decisions I've made during the implementation that are important for understanding the results of the experiments.

5.1 Problem representation
First of all, implementing the algorithms on the TSP required defining the objective function and the neighboring solution. The objective function is defined as the length of the cycle. For this reason, the goal is to find the **global minimum**. Regarding the neighboring solutions, there are a multitude of ways to define them. I have chosen a very simple one: the neighbor of the current cycle is obtained by randomly swapping two vertices in the path.

5.1.2 Algorithms choice
As there are many variants of simulated annealing (depending on the cooling schedule) and hill climbing algorithms, I had to choose the ones suitable for my problem. In this section, I only describe the functioning of the algorithms while the specific parameter values are reported in the experiments section.

The initial idea was to use the "plain" hill climbing version where all the neighboring solutions are considered at each step and the best one is chosen as the next candidate. However, initial testing showed that this approach is not feasible. Number of neighboring solutions defined in the way I've described is $O(n^2)$. Considering that this calculation must be done at each iteration of the hill climbing algorithm, it quickly becomes infeasible for bigger graphs. For this reason, I have decided to use the first-choice hill climbing algorithm where the random neighbor is generated until one that is better than the current solution is found. Of course, the maximum number of attempts is specified, and it is varied throughout the experiments. Moreover, I have used random restart with the hope of starting at the solution that converges to a good local optimum.

For the simulated annealing, the choice of the cooling schedule must be defined. I have tested a few of the most popular variants and decided that I will use the **geometric cooling schedule**. I start with the initial temperature $T(0)$ and update it as $T(i+1) = T(i) * \alpha$ after each iteration, where i denotes the current iteration and α denotes the cooling rate between 0 and 1. The acceptance rate I've used is $e^{-\delta/T}$, where δ is the difference between a random neighbor and the current cycle length.

Literature suggests some more complicated cooling schedules that are adaptive to the search space. However, the goal of this project is to compare the simple versions of the algorithms against each other, thus I have decided to stick with the very common geometric schedule and test different sets of parameters.

6 Experiments

6.1 Experiments setup

I have performed experiments on 10 benchmark instances available in the **TSPLIB** (http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/). The graphs I've used have the following sizes: 14, 22, 29, 42, 52, 70, 100, 198, 442, 1002.

I report the tests with five sets of parameters for the Simulated Annealing algorithm and four sets of parameters for the Hill Climbing algorithm. I have devised these sets of parameters experimentally and I've decided to perform thorough analysis on these sets that I've found most suitable for my project. Parameter sets are recorded in the tables below.

Initial temperature	**Cooling rate**
1000	0.9995
1000	0.99995
100	0.999997
variable[1]	0.999999
1000	0.999997

Table 1 - Simulated Annealing parameter sets

Num restarts	**Max attempts**
10	100
10	1000
10	2500
10	10000

Table 2 - Hill Climbing parameter sets

Every set of parameters was tested on every graph for 10 independent runs and the results are averaged. Percentage differences from the optimal solution are reported for the average of the 10 runs for a certain set of parameters. This makes the analysis more stable, and the obtained result is less likely to be due to the stochastic nature of the algorithms.

6.2 Analysis

Given that the algorithms' performance highly depends on the parameters, I've decided that the analysis should be split into two major parts:

- Understanding how varying parameters influence the algorithms
- Comparing algorithms against each other

[1] variable initial temperature = 0.9 * average distance between neighbors

6.2.1 Simulated annealing varying parameters

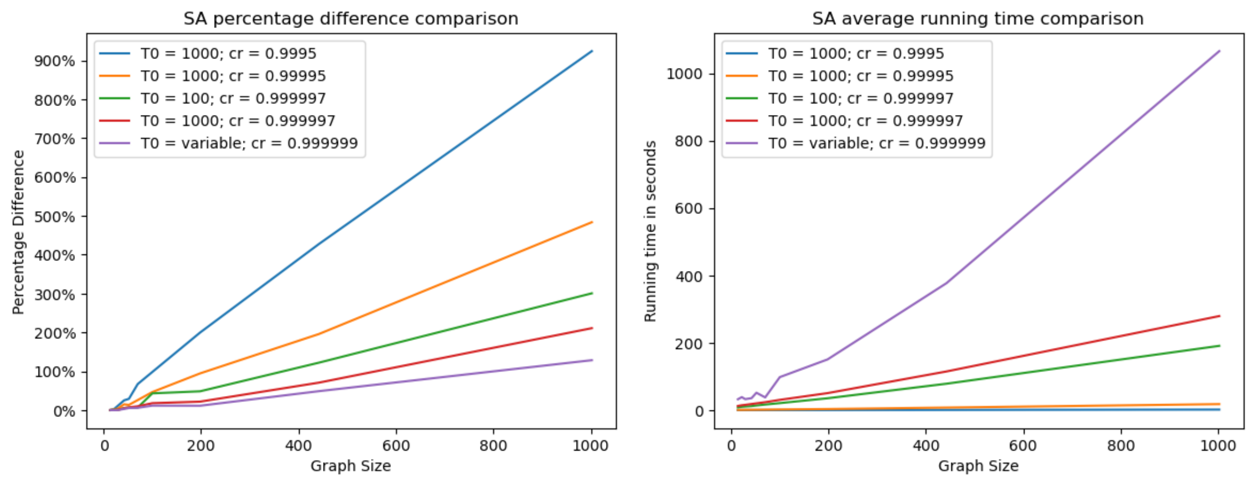

Figure 1 - SA solution quality and running time comparison

In Figure 1 we can see the effect of different parameter sets on the SA algorithm. As expected, keeping the temperature high for longer leads to more exploration of the search space resulting in better solutions, but takes more time to converge. An important thing to notice is that as the solution approaches the optimal, it takes more and more time to improve it. This behavior is very intuitive, as it is fairly easy to make progress towards the optimal solution in the beginning, but it becomes much harder to refine it as we approach the optimum. Deciding between the set of parameters depends on the particular problem and the desired time-quality tradeoff.

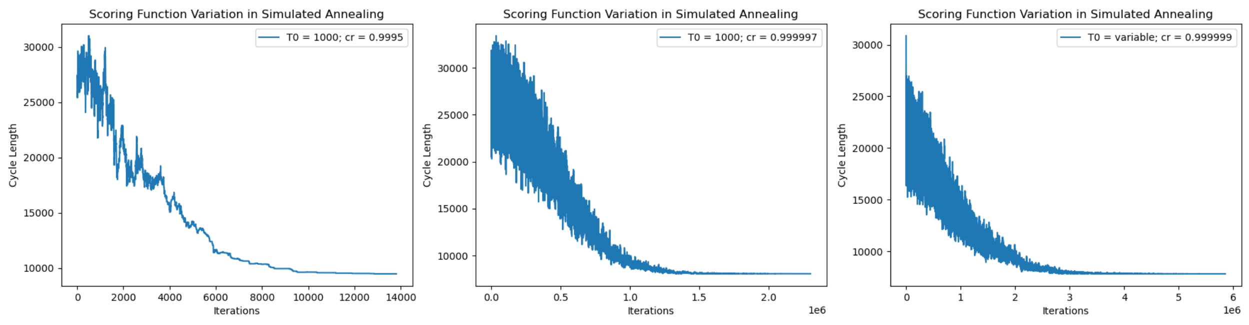

Figure 2 - Scoring function variation from faster to slower cooling schedule

In Figure 2, I have plotted the scoring function during the iterations of the SA algorithm on the *berlin52* graph instance. We can get an idea of the convergence behavior as the cooling schedule slows down. The number of iterations is around 14000, 2 million and 6 million.

I have also found that increasing the cooling rate has a much better impact on the results than increasing the initial temperature as the graphs grow in size. Slow exploration is a good way to get near-optimal results, but it becomes extremely slow for large graphs.

6.2.2 First choice hill climbing varying parameters

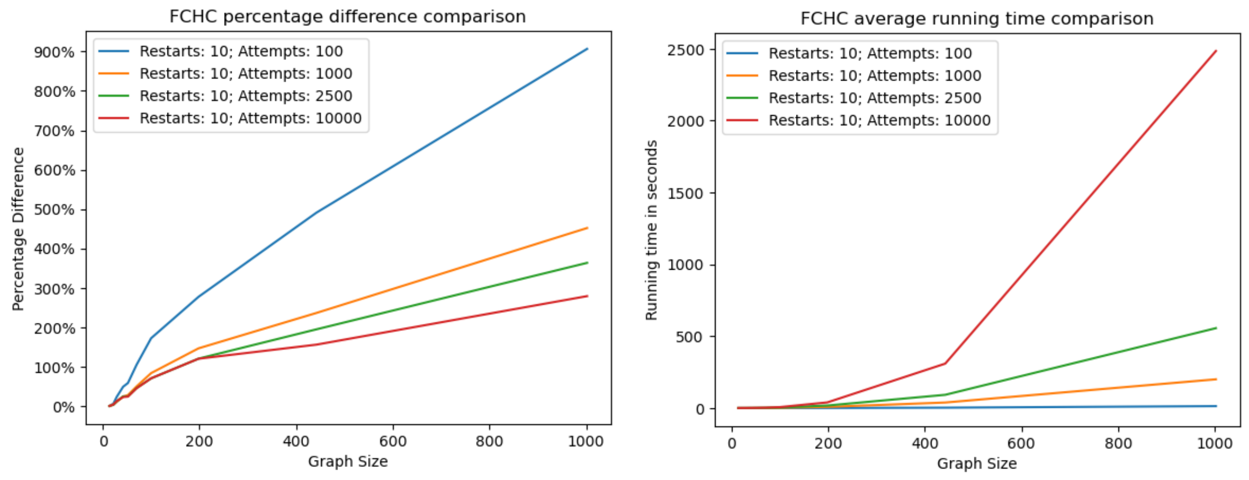

Figure 3 - First choice hill climbing solution quality and running time comparison

In Figure 3 we can see the percentage difference and average running time comparison for the First Choice Hill Climbing with Restart. Changing the number of maximum attempts at each algorithm iteration becomes more important as the graphs grow in size. This seemed to be a major drawback of the first-choice hill climbing approach. I will discuss it in more detail after showing the results from the next plot.

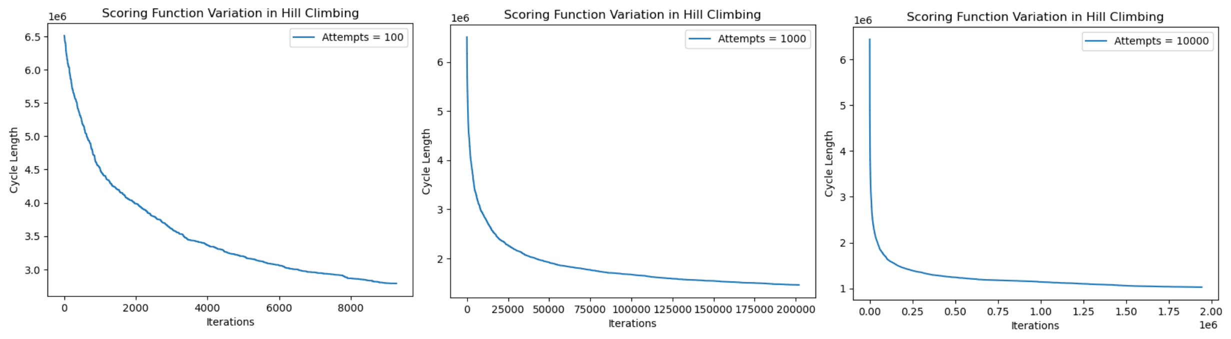

Figure 4 - Scoring function variation changing the number of max attempts

In Figure 4 we can see the scoring function variation of the first-choice hill climbing algorithm on graph *pr1002*. I have chosen the largest graphs among the ones I've tested to emphasize the aforementioned problem. We can see that the number of iterations explodes as we increase the number of maximum attempts allowed for finding a better neighbor. Moreover, these iterations are necessary because the algorithm needs more time to "climb" the local hill it is on.

So, the bigger the graph the harder it is to find the neighbor better than the current solution. On the other hand, the search space is larger, and the algorithm needs more restarts. Hence, we need both parameters to have high values in order to find the good local optimum and "climb" towards it. Looking at Figure 3, we can see that increasing these parameters results in extremely long running times.

6.2.3 Simulated Annealing vs First Choice Hill Climbing

I conclude that simulated annealing performs significantly better than hill climbing in my parameter setting. Reporting the results for all the combinations of parameters wouldn't be clear due to the number of different parameter sets. Hence, in order to be concise, I will report the results that I have found to be the most descriptive of the comparison between the algorithms.

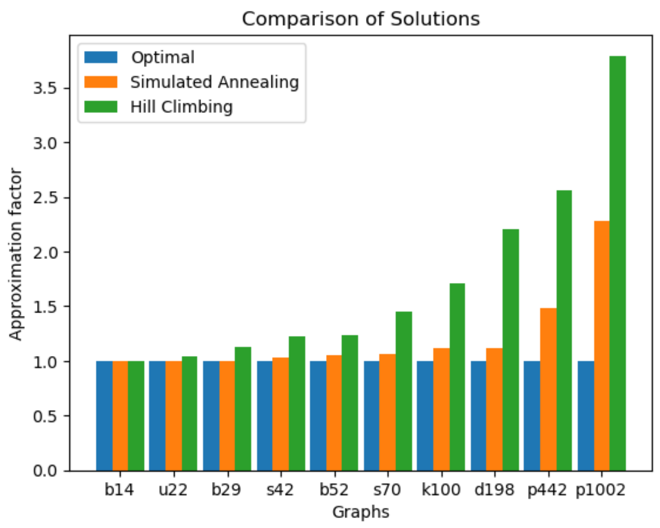

Figure 5 - Simulated annealing vs First Choice Hill Climbing with Restart

In Figure 5 we can see the depiction of the approximation factor of both algorithms for all the tested graphs. These results are obtained from the slowest-converging parameter sets for both algorithms, i.e., T0 = variable, cr = 0.999999 for simulated annealing and Restarts = 10, Max attempts = 10000 for the first-choice hill climbing. It is clear that simulated annealing outperforms hill climbing, and the conjecture is that the difference would become more prominent on the bigger graphs. Moreover, simulated annealing reaches these solutions way faster than hill climbing which can be seen from the previous plots.

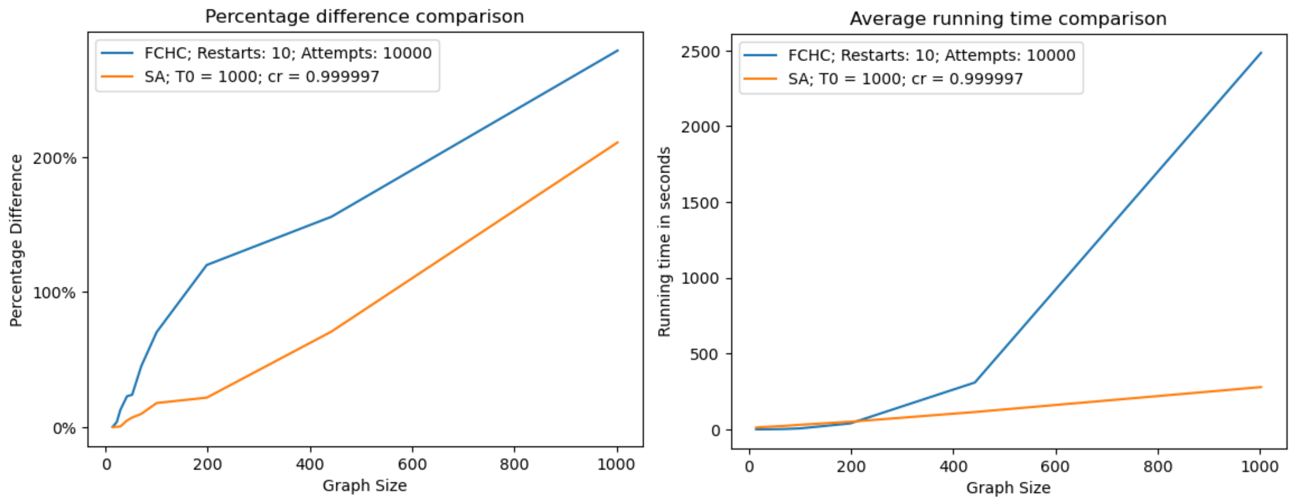

Figure 6 - Simulated annealing vs First Choice Hill Climbing with Restart 2

I’ve used Figure 6 to further emphasize the superior performance of simulated annealing. This parameter set for simulated annealing that converges significantly faster than the previous one still performs much better than hill climbing as graphs grow. The conclusion is that hill climbing with restart might work for small graphs, but as the number of nodes starts increasing the search space becomes too complicated for only upward moves.

I want to note that increasing the number of restarts and decreasing the number of attempts also wouldn’t work (from my results). The aforementioned problem of trying to “climb” up the hill in graphs with many vertices doesn’t allow the algorithm to converge to a good local optimum even if one of the starting points ends up near it.

7 Conclusion and further considerations

In this section, I want to get back to the initial questions expressed in the introduction and provide answers based on the obtained results. Moreover, I will state some of the future development paths of this project.

Results indicate that simulated annealing outperforms first-choice hill climbing with restart. While hill climbing may be effective in small search spaces, the big problems seem to present a major challenge for it, and more exploration provided by simulated annealing is necessary.

The effectiveness of both simulated annealing and hill climbing greatly depends on the set parameters. While it might be easy to solve smaller TSP instances, the bigger problems present quite a challenge. However, the results that I have obtained are nowhere near state-of-the-art results in terms of solution quality and time efficiency. The travelling salesman problem seems to be too hard to solve optimally with simple heuristics and elementary parameter tuning.

Reading the literature, I have found that the most important points for further work would be:

- Optimized implementation of algorithms
- More sophisticated parameter tuning, e.g., Adaptive cooling schedules for Simulated Annealing
- Additional heuristics
- More complicated algorithms using SA or HC just as one of the steps

Of course, in addition to these considerations, further work could be focused on trying out more parameter sets and applying them to more benchmark graph instances. In this way, some of the conjectures I made could be confirmed or disputed depending on the additional results.