An Analysis of the Efficiency of Hill-Climbing and Simulated-Annealing algorithms in Calculating the Local Minimum of a Numerical Benchmark Function

Harabagiu Ștefan-Alexandru

November 3, 2022

1 Abstract

To be able to choose an optimum local-search algorithm for a particular problem, we need to know the advantages and disadvantages of each, as well as the differences between each other. Since the efficiency of an algorithm depends on multiple factors (programming language, processing power of the system, implementation details, the problem which has to be solved etc.) we cannot solely rely on theory. Testing each algorithm on different numerical benchmarking functions allows us to acquire a solid understanding of the way in which each algorithm works depending on the input. This article shows that in the majority of cases, Best-Improvement Hill-Climbing outperforms its two counterparts, Worst-Improvement Hill-Climbing and First-Improvement Hill-Climbing and that Simulated-Annealing can have varying results depending on its calibration, ranging from getting better results in a faster time from its original algorithm to being the worst algorithm out of all.

2 Introduction

2.1 Motivation

The purpose of this study is to test the differences between four different local-search algorithms: Best-Improvement Hill-Climbing, Worst-Improvement Hill-Climbing, First-Improvement Hill-Climbing and Simulated-Annealing to aid in choosing one for solving a certain optimization problems.

2.2 Problem Description

The method in which these algorithms were tested is trying to find the minimum of 4 different numerical benchmark functions: Dejong's, Rastrigin's, Schwefel's and Michalewicz's each testing different proprieties of the algorithms. Running these tests on different dimensions of these functions shows the exponentially growing difficulty of finding the local minimum that comes with these types of problems.

3 Methods

3.1 Search-Space

Each minimum is search is limited within a space $[a, b]$, predefined for each function. The precision of our tests is the same for all functions, that being 10^{-5} because 5 decimals after 0 is good compromise between precison and running speed since increasing the precision greatly increases the running time of the algorithms. The given space can then be divided into $(b-a)\cdot 10^5$ subintervals. The parameters of a function are represented as an array of bits, each parameter being $\lceil log_2\big((b-a)\cdot 10^5\big)\rceil$ bits long.

3.2 Hill-Climbing

Hill-Climbing is a local-search algorithm that takes a heuristic approach to finding the local minimum. Hill-climbing starts by generating a random solution in the form of an array of bits, and calculates the value of the function with a given solution. Depending on the variant of Hill-Climbing, all neighbors are generated at the same time or incrementally. A neighbor of a solution is the solution with one of its bits flipped.

solution	neighbours
0101	1101
	0001
	0111
	0100

$\underbrace{1101000101111010 0}_{neighbourhood}$

The value of each neighbor is then evaluated and compared with the current solution. Best-Improvement Hill-Climbing selects the neighbor which evaluates the lowest and is lower than our solution, while Worst-Improvement selects the neighbor which evaluates the highest but is still lower than the current solution. First-Improvement Hill-Climbing selects the first neighbor which is lower than our solution. The algorithm repeats itself until it cannot find another improvement. The algorithms ran were iterative Hill-Climbers, which keep the lowest minimum found after finishing and compare it with a previous minimum. If it is lower, the lowest minimum is updated. The algorithm then restarts itself a fixed number of times to try and find a better solution.

3.3 Simulated-Annealing

Simulated-Annealing is a meta-heuristic algorithm which helps Hill-Climbing overcome one of its main downsides: getting stuck in a local optima. Due to Hill-Climbings nature of only selecting better solutions, once it finds an alley, it will continue descending and never climb back up again, confining the search space to that zone. To overcome this, whenever a neighbor does not improve our current solution, we have a chance to select it either way. The formula used to calculate the probability of choosing another state is called *Annealing-Schedule*, and for this experiment has been chosen as

$$\textit{uniform([0, 1)} < e^{\frac{-|eval(neighbor) - eval(solution)|}{temperature}}.$$

Temperature is a constant chosen before runtime, and it is decreased every time we select a new neighbor by another constant called *Cooling schedule*, which is smaller than 1 making the temperature gradually decrease. The higher the temperature, the higher the probability of choosing a worse neighbor, and the lower the temperature, the lower the probability. When the temperature is approximately 0, the chance to choose a worse neighbor is also 0, reducing the algorithm to the normal Hill-Climbing algorithm. For our tests, the temperature chosen was 10, with a cooling constant of 0.70. This was selected after manually testing different combinations of temperatures with different cooling constants on Rastrigin's function. The pool of tested temperatures is $\{1, 10, 100\}$ and the pool of cooling constants $\{0.60, 0.65, 0.70, 0.75, 0.80, 0.85, 0.90, 0.95\}$. The results showed that some of the best results are yielded by 0.95 cooling constant and temperature of 100/10/1 or lower temperatures like 0.65/0.70 and temperatures of 10/1. However, selecting a temperature of 100 greatly increased the run-time of the algorithm due to the frequent change of temperature done, so the $10, 0.70$ pair was chosen, which gives us a pretty rapid Annealing-Schedule. This version of Simulated-Annealing is based on the First-Improvement Hill-Climbing

3.4 Specifications

All the following calculations will be run on an 11th Generation Intel Core i5-1135G7 with 2.40 GHz with 4 cores and 8 virtual cores and 16 GB of RAM. The algorithms are written in C++ and make use of STL, which reduces their performance as an array of bools would have been faster, but harder to implement and more error-prone. The process is run on Windows 10, from Visual Studio with all other applications closed and in a multithreaded environment.

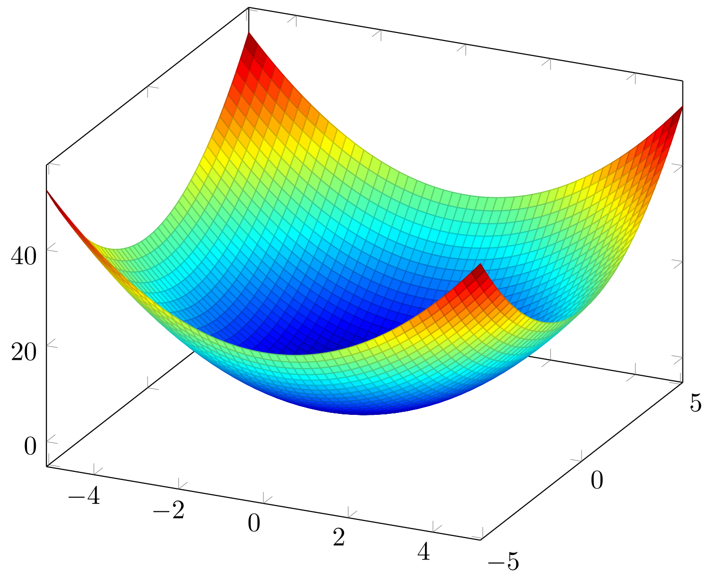
3.5 Algorithm Settings

All functions were run in their 5,10 and 30 dimension variations. Each Hill-Climbing was restarted 250 times before concluding a minimum. For each function and each dimension, the algorithm was run in a multithreaded way, launching 8 threads concluding 2000 iterations on each repetition. This process was repeated 30 times for all functions resulting in 60000 iterations for each dimension. Dejong's function was not run this many times since it would always find the global minimum after a single iteration, so it was run for time-testing purposes only.

4 Functions

4.1 Dejong

$$f(x) = \sum_{i=1}^{n} x_i^2,\ -5.12 \le x \le 5.12$$



Global minimum: $f(x) = 0, x_i = 0\ i = 1 : n$

Method		Dimesions	Mean	StdDev	Min	Max	Total Time
Hill-Climbing	Best-Improvement	5	0	0	0	0	4 s
		10	0	0	0	0	25 s
		30	0	0	0	0	175 s
	First-Improvement	5	0	0	0	0	3 s
		10	0	0	0	0	18 s
		30	0	0	0	0	50 s
	Worst-Improvement	5	0	0	0	0	8 s
		10	0	0	0	0	710 s
		30	0	0	0	0	305 s
Simulated-Annealing		5	0	0	0	0	0 s
		10	0	0	0	0	2 s
		30	0	0	0	0	92 s

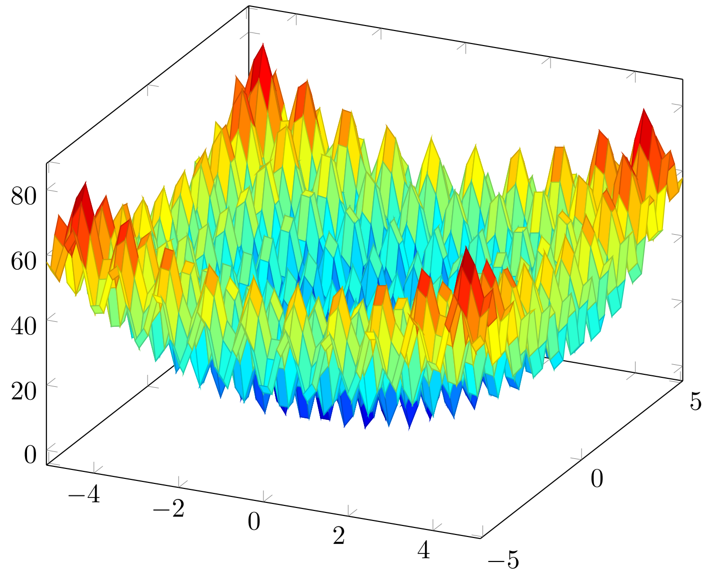
Dejong's function is unimodal, so it does not pose any challenges for our local search algorithms. They always find the global minimum in one iteration.

(note that the 30 dimension variant has been run for only 16000 iterations instead of the usual 60000)

4.2 Schwefel

$$f(x) = \sum_{i=1}^{n} -x_i \cdot \sin\sqrt{|x_i|}, \; -500 \leq x \leq 500$$

Global minimum: $f(x) = -n \cdot 418.9829, x_i = 420.9687 \; i = 1 : n$



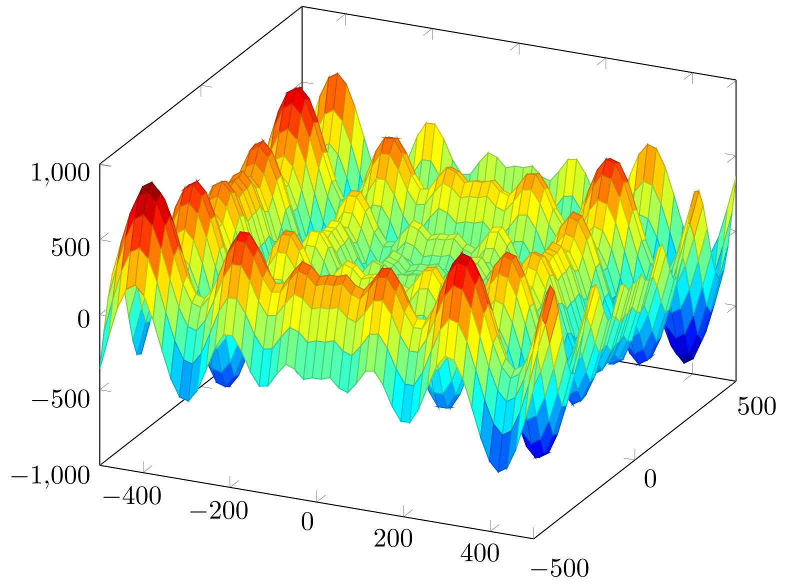
Dimensions	Global Minimum
5	-2094.9145
10	-4189.8290
30	-12569.4870

Method		Dimesions	Mean	StdDev	Min	Max	TotalTime
Hill-Climbing	Best-Improvement	5	-2094.84	0	-2094.81	-2094.71	7 s
		10	-4108.8	0	-4189.31	-4002.6	49 s
		30	-11415.2	0	-11806.8	-11237.1	1358 s
	First-Improvement	5	-2059.16	0	-2094.5	-2033.59	21 s
		10	-3997.04	0	-4074.32	-3928.62	108 s
		30	-11175.3	0	-11552.5	-10905.1	1907 s
	Worst-Improvement	5	-1548.34	0	-1964.59	-1328.37	9 s
		10	-2270.83	0	-2891.91	-1987.96	45 s
		30	-4079.6	0	-5199.01	-3046.6	1467 s
Simulated-Annealing		5	-2078.55	0	-2094.41	-2060.47	2 s
		10	-3924.36	64.3492	-4086.71	-3800.06	21 s
		30	-10876.3	120.957	-11148.9	-10595.7	715 s

Schwefel's function is complex, with many local minima which might fool the algorithm into converging into the wrong direction. Even though Worst-Improvement has worse performance than First-Improvement here, it seems to take a shorter time to finish due to possibly getting stuck in a plateau and not being able to find a better solution. Simulated-Annealing performs similarly to its Hill-Climbing counterpart, albeit faster but with higher standard deviation.

4.3 Rastrigin

$$f(x) = 10 \cdot n + \sum_{i=1}^{n} \left(x_i^2 - 10 \cdot \cos\left(2 \cdot \pi \cdot x_i\right)\right),\ -5.12 \leq x \leq 5.12$$



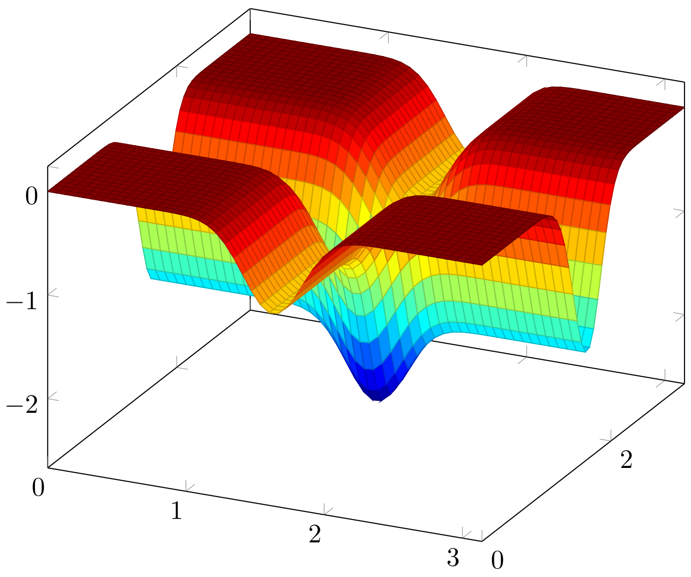
Global minimum: $f(x) = 0, x_i = 0\ i = 1 : n$

Method		Dimesions	Mean	StdDev	Min	Max	TotalTime
Hill-Climbing	Best-Improvement	5	0.279832	0.454501	0	0.99496	3 s
		10	3.26945	0.810473	1.00001	4.46156	22s
		30	25.9969	1.84469	20.7949	30.6881	599 s
	First-Improvement	5	1.04691	0.436438	0	2.17165	13 s
		10	5.43476	1.19292	4.0027	8.47648	66 s
		30	32.3711	2.653211	28.9575	35.6863	567 s
	Worst-Improvement	5	1.62499	0.553581	0	2.00002	32 s
		10	7.00116	1.30273	2.9004	9.5329	126 s
		30	41.8256	2.78729	35.8712	49.1126	3493 s
Simulated-Annealing		5	0.745906	0.48198	0	1.23582	1 s
		10	5.02352	0.843582	3.02574	6.91064	10 s
		30	29.6957	2.54571	27.3343	37.1712	307 s

Rastrigin's function is highly multimodal, but with local minimum well spread. This tests our algorithm's power to search the whole search space. We can see Simulated-Annealing greatly improving on its counterpart in both accuracy and run-time due to it being calibrated on Rastrigin's function.

4.4 Michalewicz

$$f(x) = -\sum_{i=1}^{n} \sin(x_i) \cdot \left(\sin\left(\frac{i \cdot x_i^2}{\pi}\right)\right)^{2 \cdot m},\ 0 \le x \le \pi$$



Dimensions	Global Minimum
5	-4.687
10	-9.660
30	-29.630

Method		Dimesions	Mean	StdDev	Min	Max	TotalTime
Hill-Climbing	Best-Improvement	5	-4.68726	0.000581	-4.68766	-4.68534	4 s
		10	-9.41339	0.066140	-9.55991	-9.26444	25 s
		30	-27.1584	0.30089	-28.0452	-26.7434	3171 s
	First-Improvement	5	-4.68253	0.010386	-4.68623	-4.62451	9 s
		10	-9.02653	0.17235	-9.37354	-8.6561	52 s
		30	-24.3589	0.33643	-25.1864	-23.8984	3833 s
	Worst-Improvement	5	-3.68091	0.22716	-4.08932	-3.21856	5 s
		10	-6.98312	0.39347	-7.20084	-5.55414	46 s
		30	-9.83502	0.63785	-11.3977	-8.05316	3241 s
Simulated-Annealing		5	-2.92091	0.327169	-3.79963	-2.46697	2 s
		10	-4.2615	0.393477	-5.20894	-3.55414	5 s
		30	-8.35302	0.637856	-10.3977	-7.05316	39 s

Michalewicz's function is also highly multimodal, however between each local minimum the search space is flat, making it hard for our algorithm to find which way to go. We can see that this causes the Worst-Improvement Hill-Climbing and Simulated-Annealing to get stuck and are not be able to find any good result.

5 Conclusions

Not surprisingly, Best-Improvement Hill-Climbing seems to outperform all the other algorithms in its ability to find the lowest minimum with the greatest consistency. This is probably due to the fact that it tend to greedily select its neighbors from which to move. First-Improvement Hill-Climbing tends to run at about the same speed, but with slightly worse accuracy in terms of minimum found. Its standard deviation seems to also be a bit higher. Worst-Improvement Hill-Climbing seems to be the slowest, and provides the worst results. It can also get stuck in plateaus, making the results provided be very bad compared to the others. Simulated-Annealing is a very interesting algorithm, in the sense that it is not as good generally at finding the lowest minimum, but always runs the fastest, even with the frequent change in temperature chosen for this experiment. Its standard deviation is the highest, meaning it is not very consistent at finding good results, however when calibrated properly (as this was calibrated on Rastrigin's function) it outperforms its Hill-Climbing counterpart greatly in both accuracy and speed, making it a better algorithm if used properly. The rapid cooling technique used seems to perform poorly on functions with more plateaus such as Michalewicz, for which a slower cooling time would have allowed it to find some of the local minimums.

6 Bibliography

References

[1] Wikipedia
https://en.wikipedia.org/wiki/Hill_climbing
https://en.wikipedia.org/wiki/Simulated_annealing
https://en.wikipedia.org/wiki/Local_search_(optimization)

[2] Eugen Nicolae Croitoru
https://profs.info.uaic.ro/~eugennc/teaching/ga/
https://gitlab.com/eugennc/teaching/-/tree/master/GA

[3] Hartmut Pohlheim
http://www.geatbx.com/docu/fcnindex-01.html#P86_3059

[4] Hein de Haan
https://towardsdatascience.com/
how-to-implement-the-hill-climbing-algorithm-in-python-1c65c29469de

[5] cppreference
https://en.cppreference.com/w/

[6] Martin Ankerl
https://martin.ankerl.com/2018/12/08/fast-random-bool/