Comparison of Hill Climbing and Simulated Annealing Methods for Multi-variable Function Minimization

Vlad Andrei Butnaru

November 2, 2023

Contents

Abstract

For function optimization, determining accurate and efficient methods to find minimums in multivariable functions is of the highest importance. This study embarks on a comparative analysis of two distinct strategies: a hill-climbing and simulated annealing algorithms. Using Rastrigin's, Schwefel's, Michalewicz's and De Jong's functions as the experimental backdrop, evaluated in 5-dimensional, 10-dimensional and 30-dimensional spaces.
Initial results indicate that the simulated annealing is usually the better choice, although having the disadvantage that the consistency of producing good solutions is lower than hill climbing. Another thing to note is that even if the best improvement outputed better results, the difference between that and the first improvement is not very big.

1 Introduction

Optimization is a key concept in many fields of science and engineering. It's all about finding the best solution to a problem. To test how well optimization methods work, we often use benchmark functions. In this report, we'll look at four such functions: De Jong 1, Schwefel's, Rastrigin's, and Michalewicz's. These functions are tricky to work with (some having a lot of local minimums or other having hard to find global ones) and give us a good idea of how effective our methods are.

We'll be using two main methods to find the best solutions: Hill Climbing and Simulated Annealing. Hill Climbing has three different types: first improvement, best improvement, and worst improvement. Simulated Annealing will be combined with best-improvement.

The analyze will try and provide an overall comparison between the performance (expressed by the time it takes to determine the values) and the accuracy (expressed by the difference between the real and obtain results).

To make our study thorough, we'll test these methods on different versions of our functions: with 5, 10 and 30 dimensions. We want our answers to be accurate, so we'll aim for a precision of 5 decimal places.

In the next parts of this report, we'll explain our methods, describe our tests, and share our findings.

2 Methods

We are going to determine the values of the functions using two approaches: hill-climbing (first, best and worst improvement) and simulated annealing. Both methods will be described in the next subsections. The function definitions are this ones:
Rastrigin's : $f(\mathbf{x}) = A \cdot D + \sum_{i=1}^{D} \left[x_i^2 - A \cdot \cos(2\pi x_i)\right], A = 10,$
$-5.12 \le x_i \le 5.12$
Michalewicz : $f(\mathbf{x}) = -\sum_{i=1}^{D} \sin(x_i) \cdot \left[\sin\left(\frac{i \cdot x_i^2}{\pi}\right)\right]^{2m}$, $m = 10$, $0 \le x_i \le \pi$
Schewefel : $f(x) = \sum_{i=1}^{D} x_i \cdot \sin\left(\sqrt{|x_i|}\right)$, $-500 \le x_i \le 500$
De Jong 1: $f(x) = \sum_{i=1}^{D} x_i^2$, $-5.12 \le x_i \le 5.12$

2.1 Hill Climbing method

When calculating our results we want to achieve a precision of at least 10^{-5}. Hence we first need to calculate the number of bits for the representation of numbers having this accuracy. This can be achieved using the formula:

$$n = log_2((B - A) \cdot \epsilon), \text{ where [A, B] is the domain of the function and } = 10^{-5}$$

After finding this number we generate a bitstring of length n, that we then decode into a number (we consider that a binary representation), let's call it

N. To get a value in the domain of the function from this one, we just have to calculate:

$$M = N/(2^n - 1),\ M \in [0,\ 1)$$

$$R = A + M \cdot \text{ (B - A)}$$

R is now the decoded value, but this has to be done for all dimensions D. This can be done using one single bitstring. After calculating D values we can decode them into one single result by computing the function value. This is our starting best, let's call it S (and also global best, lets call it B). Now we will do the following algorithm for C $\leq$ T (in testing T = 1000 was enough to yield good results), where C = 0 in the begging:

1. Calculate all neighbours of S (to calculate all neighbours from an encoded value, we just flip bits one by one), let's call their array V.
2. For each neighbour, decode it and calculate its value, we obtain a array with candidate solutins
3. For each candidate solution we select one of them based on the method chosen as such:
   (a) Best Improvement: we get the best neighbour of them all, for us the one with the lowest value
   (b) First Improvement: we random shuffle V and choose the first neighbour with a value better then S, even if is not the best among all other
   (c) Worst Impreovemnt: We select the neighbour which improves the least our solution
4. If for the methods above we don't end up finding a better number we check if S is better than B (the best global solution). If true, then B will become S. We generate a new S. Add 1 to C.
   Else: we save v (the result from V) in S and start over with step 1.

2.2 Simulated Annealing Method

For simulated annealing we are using a improved version of the best-improvement hill climbing algorithm, having the following changes:

1. The stopping condition for the local improvement is now constrained by the number of times the solution did not improve (in testing we used the constant 25 for the number of times) not by just the first time we did not improve as before
2. In case we did not find a better solution we add 1 to the number of times we did not improve and with a chance depending on temperature T we flip a bit in each of the dimensions of the current solution (but we do not update the best value).

3. To calculate the chance for changing the bits of a number we calculate $\Delta = e^{-|v-B|/T}$ and generate a random number in the interval [0, 1). If the number is less than Δ, we flip the bits, else we do nothing more

4. After a solution did not improve for the chosen number of times, we decrease T using $T = T * 0.995$, ($T_{initial} = 100$).

5. We stop after we decrease the temperature 2500 times

3 Experimental Setup

Using the programming language C++, we ran 40 times each code for all functions and for cases with n = 5, n = 10 and n = 30, while also timing all the executions. The code used for gathering the data can be found on this github repository:

https://github.com/vladsuper5555/geneticAlgorithms,

The compilation options are the following:

```
g++ main.cpp -O3 -ffast-math -std=c++17 -o main
```

4 Experimental Results

4.1 Simulated Annealing Results

Function	Dimension	Best Solution	Worst Solution	Average Solution	Standard Dev.	Average Time
De_Jong	5	0.00000	0.00000	0.00000	0.00000	0.37610
De_Jong	10	0.00000	0.00000	0.00000	0.00000	0.67855
De_Jong	30	0.00000	0.00000	0.00000	0.00000	2.24824
Michalewicz	5	-4.68766	-4.68151	-4.68667	0.00139	0.75381
Michalewicz	10	-9.65447	-9.37903	-9.55896	0.07199	2.15544
Michalewicz	30	-29.36671	-28.18284	-28.97573	0.24827	15.46783
Rastrigin	5	0.00000	0.00000	0.00000	0.00000	0.56537
Rastrigin	10	0.00000	4.92333	1.37384	1.47056	1.41138
Rastrigin	30	0.00000	24.05740	9.07665	6.61739	7.78595
Schewefel	5	-2094.91443	-2094.91132	-2094.91316	0.00075	1.82512
Schewefel	10	-4189.82601	-4189.20624	-4189.52408	0.15441	3.15535
Schewefel	30	-12568.86023	-12567.82416	-12568.34701	0.23608	10.72501

4.2 Hill Climbing - Best Improvement Results

Function	Dimension	Best Solution	Worst Solution	Average Solution	Standard Dev.	Average Time
De_Jong	5	0.00000	0.00000	0.00000	0.00000	0.08962
De_Jong	10	0.00000	0.00000	0.00000	0.00000	0.34352
De_Jong	30	0.00000	0.00000	0.00000	0.00000	3.28725
Michalewicz	5	-3.69886	-3.69820	-3.69877	0.00020	0.49034
Michalewicz	10	-8.55617	-8.21488	-8.36965	0.09015	3.60038
Michalewicz	30	-26.53327	-25.30700	-25.82972	0.34375	88.52737
Rastrigin	5	0.00000	1.99498	1.07853	0.54469	0.32041
Rastrigin	10	2.23073	7.69740	5.12468	1.34449	2.27042
Rastrigin	30	23.83127	31.86695	28.55553	2.13587	55.20664
Schewefel	5	-2094.91380	-2060.57462	-2091.44690	9.90072	0.44251
Schewefel	10	-4189.30927	-3834.39706	-4023.95676	78.87228	1.73259
Schewefel	30	-11683.64427	-10952.95817	-11226.38748	182.94094	29.78132

4.3 Hill Climbing - First Improvement Results

Function	Dimension	Best Solution	Worst Solution	Average Solution	Standard Dev.	Average Time
De_Jong	5	0.00000	0.00000	0.00000	0.00000	0.05024
De_Jong	10	0.00000	0.00000	0.00000	0.00000	0.18555
De_Jong	30	0.00000	0.00000	0.00000	0.00000	1.74959
Michalewicz	5	-3.69886	-3.69438	-3.69756	0.00171	0.33052
Michalewicz	10	-8.49697	-8.07167	-8.22727	0.12652	2.32017
Michalewicz	30	-25.89731	-24.36228	-24.96948	0.32374	52.78540
Rastrigin	5	0.00000	2.47167	1.49780	0.67794	0.19162
Rastrigin	10	3.47167	9.16918	6.78932	1.36543	1.30528
Rastrigin	30	34.33458	45.44246	39.91860	3.03720	30.92851
Schewefel	5	-2094.80981	-1907.41315	-2030.37111	51.31307	0.24472
Schewefel	10	-4036.94707	-3702.48633	-3843.53865	89.48270	0.91178
Schewefel	30	-10933.74818	-10477.85642	-10644.10269	144.30698	13.82723

4.4 Hill Climbing - Worst Improvement Results

Function	Dimension	Best Solution	Worst Solution	Average Solution	Standard Dev.	Average Time
De_Jong	5	0.00000	0.00000	0.00000	0.00000	0.01634
De_Jong	10	0.00000	0.00000	0.00000	0.00000	0.06480
De_Jong	30	0.00000	0.00000	0.00000	0.00000	0.61407
Michalewicz	5	-4.67046	-3.77759	-4.26168	0.20743	0.21434
Michalewicz	10	-8.92978	-7.54835	-8.18192	0.30581	1.72528
Michalewicz	30	-24.50010	-21.06464	-22.40172	0.91221	52.34335
Rastrigin	5	2.00000	6.99929	3.96316	1.23328	0.20321
Rastrigin	10	6.47173	19.15829	13.98730	2.87791	1.40897
Rastrigin	30	45.84600	68.16010	58.22150	5.17600	33.97395
Schewefel	5	-1987.27869	-1780.10039	-1914.67826	62.44475	0.21726
Schewefel	10	-3871.62807	-3453.86925	-3651.15587	120.01003	1.08139
Schewefel	30	-10899.84348	-9728.18776	-10266.64649	248.26775	19.56971

5 Discussion

The results presented in the previous section show the performance of the Hill Climbing and Simulated Annealing methods across different benchmark functions and dimensions. A comparative analysis of the results yields several key observations.

Firstly, the Simulated Annealing method consistently found solutions close to the optimal for all functions across all tested dimensions, while the Hill Climbing method, showed weaker results.

For more complex functions like Michalewicz's, the Simulated Annealing method performed noticeably better in terms of finding solutions closer to the known optimum, especially in higher dimensions. The Best Improvement version of Hill Climbing, did not fare as well since it got usually stuck in local minimums.

Rastrigin's function, known for its numerous local optima, proved to be a challenging landscape for all methods. However, Simulated Annealing, with its ability to escape local optima due to its probabilistic nature, outperformed the Hill Climbing variants.

Although Simulated Annealing and Hill Climbing (Best Improvement) both approached the global optima for Schewefel's function, there were significant variations in performance among the Hill Climbing variants, especially in higher dimensions. This again shows the effectiveness of the ability of simulated annealing in "forgetting" the local best in the search for the global one;

In the case of De Jong's 1 function all the functions found the global best due to the nature of the function allowing both methods to always find easily the next best.

Another aspect to point out is the fact that the standard deviation was always under ≤ 10 for simulated annealing while not so common among the other cases.

The average time taken by each method offers insights into their computational efficiency. Hill Climbing generally took longer due to its iterative approach (having somewhat a fixed number of steps). For Simulated annealing on the other hand, even if the stopping condition for local minims is based on the number of times we did not improve, had faster computation time. This is because of the calibration of the number of times we try to improve and the decrease in temperature.

6 Conclusion

The comparative analysis of Hill Climbing and Simulated Annealing methods in this study offers valuable insights into their optimization capabilities across different functions and dimensions. While both methods have their strengths, the choice of method should be influenced by the specific nature of the problem at hand.

Simulated Annealing, with its inherent ability to escape local optima, proves advantageous in complex landscapes with multiple local optima. On the other hand, Hill Climbing, especially the Best Improvement variant, is more computationally efficient and can be effective for simpler functions or when used as a preliminary search method.

In conclusion, no single method can be universally deemed superior. Researchers and practitioners should make informed choices based on the nature of their optimization problems, the precision required, and available computational resources.

Bibliography

[1] The Matplotlib Development Team. *Matplotlib: Visualization with Python.* [Online]. Available: `https://matplotlib.org/`.

[2] Kyoto University. *Operations Research as Mathematical Taikonautics.* [Online]. Available: `http://www-optima.amp.i.kyoto-u.ac.jp/`.

[3] Simon Fraser University. *Simon Fraser University: Engaging the World*. [Online]. Available: `https://www.sfu.ca/`.

[4] N. Eugen. *Genetic Algorithms Course*. [Online]. Available: `https://profs.info.uaic.ro/~eugennc/teaching/ga/`.

[5] Wikipedia contributors. *Wikipedia, The Free Encyclopedia*. [Online]. Available: `https://en.wikipedia.org/wiki/`.