Comparison of Hill Climbing and Simulated Annealing Methods vs. Genetic Algorithms for Benchmark Function Minimization

Vlad Andrei Butnaru

November 30, 2023

Abstract

This study embarks on a comparative analysis of two distinct strategies for finding minima of benchmark functions: hill climbing and simulating annealing vs. genetic algorithm. 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 genetic algorithm outperforms simulating annealing in most cases, in accuracy and time-efficiency, although sometimes having the disadvantage that the consistency of producing good solutions and the time required for lower dimensions is lower than simulating annealing . As discussed in the previous paper, hill climbing is also outperformed by simulating annealing in almost all situations, so this paper will focus on compering the simulating annealing and genetic algorithm methods.

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 focusing on two main methods to find the best solutions: Genetic Algorithms and Simulated Annealing, Hill Climbing having not so competitive results compared to simulating annealing. However, the genetic algorithm is hybridised in the end with a simple run of best improvement hill climbing algoirthm and simulated annealing will be also combined with best-improvement hill climbing algorithms.

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.

Throughout this paper, when we refer to Hill Climbing we consider the best-improvement variant of it.

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 focusing on two approaches: genetic algorithm and simulated annealing. Both methods will be described in the next subsections. The function definitions are this ones:
Rastrigin's : $f(\mathbf{x}) = A \cdot D + \sum_{i=1}^{D} \left[x_i^2 - A \cdot \cos(2\pi x_i)\right], A = 10,$
$-5.12 \leq x_i \leq 5.12$
Michalewicz : $f(\mathbf{x}) = -\sum_{i=1}^{D} \sin(x_i) \cdot \left[\sin\left(\frac{i \cdot x_i^2}{\pi}\right)\right]^{2m}$, $m = 10$, $0 \leq x_i \leq \pi$
Schewefel : $f(x) = \sum_{i=1}^{D} x_i \cdot \sin\left(\sqrt{|x_i|}\right)$, $-500 \leq x_i \leq 500$
De Jong 1: $f(x) = \sum_{i=1}^{D} x_i^2$, $-5.12 \leq x_i \leq 5.12$
We will first discuss about the implementation principles between the 3 algorithms.

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:

$n = log_2((B - A) \cdot \epsilon)$, 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 as such:
   (a) we get the best neighbour of them all, for us the one with the lowest value
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

2.3 Genetic Algorithm

For the genetic algorithm we first set up our constants for the runs to come:
Population_size = 200
Generation_count = 2000
For each solution we generate a random population. We generate Population_size of individuals of the required bit string length for maintaining the accuracy wanted (as explained in the previous section). Then for each iteration until we reach Generation_count we first evaluate each individual and assign a fitness to it (the fitness is the negative evaluated value of the function + 30000 in order to reach positive values always). When we evaluate them we also calculate the best global value of a individual.
After that we create the new population that we are going to evaluate in the next generation run as follows:
15% - elitism (selecting the best individuals from the previous generation)
25% - generated by tournament selection (we choose a number of 10 random individuals and we keep the best one)
60% - generated by crossover of the already selected individuals (for the crossover of 2 individuals we generate 2 random cut points and select the second individual after the crossover)
Next we mutate the non-elites with a chance of 1.2% per bit of each individual.
Finally after all the generations are done we run a instance of hill climbing best improvement over all individuals in the last population in order reach the value of the local minimum.
One more implementation change from SA is that in GA for Rastrigin's function we use Gray Code encoding in order to get better results (in testing it made a significant difference of about 90% better results)

3 Experimental Setup

Using the programming language C++, we ran 50 times each code for all functions and for cases with n = 5, n = 10 and n = 30, while also timing all the

executions. We also introduces multi-threading in order to improve the computation time and to utilize the resources better. 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 -std=c++17 -o main

4 Experimental Results

4.1 Genetic Algorithm Results

Function	Dimension	Min Value	Average Solution	Std. Dev.	Time
Schewefel	5	-2094.91443	-2094.77218	0.08236	7.955536
Schewefel	10	-4189.61966	-4189.37144	0.13507	8.595545
Schewefel	30	-12533.48442	-12316.98115	132.95929	11.673897
Michalewicz	5	-4.68766	-4.67195	0.04479	7.780692
Michalewicz	10	-9.65513	-9.48987	0.13362	8.273262
Michalewicz	30	-28.99957	-28.01992	0.36562	10.643685
Rastrigin	5	0.00000	0.00000	0.00000	21.699372
Rastrigin	10	0.00000	0.00000	0.00000	24.427159
Rastrigin	30	0.00000	1.30670	1.39124	33.909106
De Jong	5	0.00000	0.00000	0.00000	11.271456
De Jong	10	0.00000	0.00000	0.00000	12.143502
De Jong	30	0.00000	0.00000	0.00000	14.750912

4.2 Simulated Annealing Results

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

4.3 Hill Climbing - Best Improvement Results

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

5 Discussion

In the comparative analysis of Simulated Annealing (SA) and Genetic Algorithms (GA) for function minimization, the results clearly indicate distinct performance characteristics between the two methods.

Firstly, SA consistently provided superior solutions compared to Hill Climbing Best Improvement across all functions and dimensions, reinforcing the previously established notion of its effectiveness.

Performance in Lower vs. Higher Dimensions: A critical observation from the study is the dimensionality's impact on each algorithm's efficacy. In low-dimensional spaces (5 and 10 dimensions), SA demonstrated superior precision and consistency (taking in consideration the smaller amount of time). This can be attributed to its ability to effectively navigate the search space and its proficiency in escaping local minima, which is particularly advantageous in less complex landscapes. In contrast, while GA also performed admirably in these dimensions, it was slightly outshined by the precision of SA.

However, as the dimensionality increased to 30, the trend shifted in favor of GA. This shift can be linked to GA's inherent mechanism of exploring the search space through a diverse population and genetic operators. The higher the dimensionality, the more complex the landscape becomes, and GA's ability to maintain diversity and explore various regions concurrently becomes a significant advantage.

Even if the values are sometimes lower than SA in the GA it can be attributed to the hard stop of 2000 generations (an approach chat stops after a certain time of non increasing the value could yield better results compared to SA).

Computational Time and Efficiency: Another critical aspect is the computational time. SA was generally faster in the early stages of optimization, particularly in lower dimensions. However, this advantage diminished as the dimensionality increased, with the algorithm requiring more iterations to adequately explore the space. GA's initial setup time, involving population generation and evaluation, was offset by its more efficient search in later generations, especially in higher dimensions where the complexity of the search space increased.

Implications for Optimization Strategies: These observations have significant implications for selecting optimization strategies. For problems characterized

by lower dimensions and less complex landscapes, SA offers a fast and effective solution. However, for high-dimensional problems or those with complex landscapes, GA's population-based approach provides a more robust mechanism for exploration and optimization.

6 Conclusion

This study presented a comprehensive comparison of Hill Climbing, Simulated Annealing, and Genetic Algorithms in minimizing benchmark functions. The results demonstrated that while Hill Climbing Best Improvement generally lagged in performance, both SA and GA have their unique strengths and applicability depending on the problem context.

Simulated Annealing emerged as a choice for lower-dimensional problems, offering precision and efficiency. Its ability to navigate complex landscapes and avoid the pitfalls of local minima makes it a reliable option for such scenarios.

Genetic Algorithms showcased their power in handling high-dimensional spaces, benefiting from the genetic diversity and evolutionary mechanisms inherent to this approach. While GA may require more computational time initially, its effectiveness in exploring vast search spaces and converging towards optimal solutions justifies its use in more complex and large-scale optimization problems.

In conclusion, the choice between SA and GA is not one-size-fits-all but should be guided by the specific characteristics of the problem at hand. The dimensionality, complexity of the search landscape, and the required computational efficiency are all critical factors to consider when choosing the most appropriate optimization method.

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/`.

[6] Noraini Mohd Razali and John Geraghty. *Genetic Algorithm Performance with Different Selection Strategies in Solving TSP*. In Proceedings of the World Congress on Engineering, 2011. `https://www.iaeng.org/publication/WCE2011/WCE2011_pp1134-1139.pdf`