Analysis of different heuristic methods of determining the global minimum of a function

Chirvasa Matei & Rotariu George

November 5, 2023

1 Abstract

In this article we will assess the effectiveness of different probabilistic methods in determining the global minimum of a function. The studied methods were Hill Climbing, with different improvement variants, namely best, first and worst improvements, and Simulated Annealing using a Geometric Arithmetic cooling method. We will observe a higher degree of variance in the results generated by the Simulated Annealing method, but with much faster runtime and similar, if not better, results compared to the Hill Climbing methods.
Key words: Simulated Annealing, Hill Climbing, Global Optimization

2 Introduction

We will study two commonly used heuristic methods, with the first being Hill Climbing[1], tested using three improvement methods. The best improvement method, also known as steepest ascent hill climbing, compares all neighbouring solutions to a problem and picks the one that makes most progress towards the optimization goal, whereas the first improvement method will choose the the first value that makes progress towards the optimization goal. The worst improvement method is counter-intuitive, as it will select the neighbouring value that makes least progress towards the optimization goal, but isn't worse than the previously selected value. The hill climbing algorithm will find local optima to a given function, and as such, the functions containing only the global optima will be solved with a probability $p = 1$.
The second algorithm to analyse in this paper is the Simulated annealing[2] algorithm, named after the heat treatment used on metals in order to make them more workable, and then slowly letting the metal cool at room temperature. The main difference between the two is that simulated annealing is capable of exploring suboptimal neighbours of the current solution in order to get past the local optima limitations that the hill climbing algorithm faced. The cooling method can greatly modify the efficacy of the algorithm, but will not affect the actual implementation. A random neighbouring value will be chosen, and if the value

makes progress towards the optimization goal it is selected, otherwise, given an implementation defined probability that relies on the current temperature, the value may or may not be chosen, in order to find a better optima.

2.1 Motivation

The problem of finding the global optimum of a function appears in many other problems in our field, including AI and NP-Optimization problems, and as such developing efficient and effective algorithms to solve these problems became a necessity in our field. It became readily apparent that the tried and tested deterministic methods of solving such problems were either too slow to serve any purpose, or the solutions rendered were too imprecise to have any merit. This lead to the development of heuristic methods in order to, at the very least, approximate a solution to the problem.

3 Method

3.1 Functions

The heuristic algorithms were applied on the following multivariate functions: De Jong's Sphere function, Schwefel's function, Rastrigin's function and Michalewicz's function[4].

De Jong's Sphere function:

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

Schwefel's function:

$$f(x) = \sum_{i=1}^{n} -x_i \cdot sin\left(\sqrt{|x_i|}\right), x_i \in [-500, 500]$$

Rastrigin's Function:

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

Michalewicz's Function:

$$f(x) = -\sum_{i=1}^{n} sin(x_i) \cdot sin^{2m}\left(\frac{i \cdot x_i^2}{\pi}\right), m = 10, x_i \in [0, \pi]$$

In order to analyse the previously mentioned algorithms, we will consider the functions in their 5, 10 and 30-dimensional forms.

3.2 Algorithms

In order to obtain any meaningful information about our results, we have to establish certain parameters. We shall set the explorable values of a domain [infimum, supremum]^n, to be the set S_n, where:

$$S_n = \{x | x \in [\text{infimum}, \text{supremum}]^n, x = (\text{infimum})^n + a \cdot \varepsilon, a \in \mathbb{N}^n\}, \varepsilon = 10^{-5}$$

Simply put, the minimal step between two values of a domain must be at least ε for at least one dimension of the value, and the value $x_0 = (\text{infimum})^n$ must be included in the set of possible values. This limitation is imposed in order to avoid excessively long computation times, as this study is meant to show the power of the heuristic approaches, and finer detail may be used when the results convey meaningful information other than a proof of concept. The implementation of both algorithms will interpret the domain values using their bit representation, and as such we have chosen the neighbours of any sequence of bits to be any other sequence of bits such that their Hamming distance[3] be 1.

$$\text{Let } a \in \{0,1\}^n, \text{Neighbours}: \mathbb{B}^n \to 2^{\mathbb{B}^n}, \text{HD} : \mathbb{B}^n \times \mathbb{B}^n \to \mathbb{N},$$

$$\text{where HD is the Hamming Distance function.}$$

$$\text{Neighbours}(x_0) = \{x | x \in \mathbb{B}^n, \text{HD}(x, x_0) = 1\}$$

This representation leads to some consequences, particularly when dealing with neighbouring values. When tackling real numbers, the neighbouring values are similar, whereas bit strings at a Hamming Distance of 1 may have a difference of as much as $\frac{1}{2} \cdot (\text{supremum} - \text{infimum})$ in their real number representation. This will lead to different local optima compared to those we may expect when using real number representations, and as such the local optima in which the Hill Climbing algorithm may get 'stuck' are different from the local optima of the function in most cases.

The hill climbing algorithm, irrespective of improvement method, will run in linear time per value, in order to determine it's neighbours. We can assume that the evaluation and bit conversion functions run in constant time, as the amount of bits needed to represent a value are known at compile-time. In theory, runtime complexity of the hill climbing algorithm rivals that of a deterministic, exhaustive search, but in practice, the nature of the multivariate functions with multiple local optima prevents this. It is much harder to estimate the runtime of the simulated annealing algorithm, as it is entirely reliant on chance in order to even finish running. In practice however, this algorithm is significantly faster than any of the hill climbing methods. In the worst case, neither algorithm surpasses the complexity class $O\left(\left(\frac{\text{range}}{\varepsilon}\right)^d\right)$, where d is the Number of dimensions, and the range is the difference between the supremum and the infimum.

Converting bit strings to the real form is simple. We establish how many bits are necessary to encode the possible values inside of a function's domain, and thus know how many bits are required to build one dimension of the domain, assuming a multi-dimensional domain, as previously mentioned. We generate

the integers representing the 'index' of the real number, and then build it using the following formula[6]:

$$\text{let f be the function to analyse, } f\colon [a,b]^n \to \mathbb{R}$$

$$\text{let } N = (b-a)\cdot 10^{-\varepsilon}, N = |S_1|, \text{The proof of this fact is trivial}$$

$$\text{Additionally, but not necessarily important, we have } N^n = |S_n|$$

$$\text{We need } k = \lceil \log_2(N) \rceil \text{ bits in order to encode one dimension}$$

$$\text{Bit string length for n dimensions will be } |L| = k \cdot n, \text{where } L = \oplus_{i=1}^{n} x_{\text{bits}_i}$$

$$\text{Assuming } \oplus \text{ to be the concatenation operator}$$

$$\text{We will define int as the conversion function from bit string to integer}$$

$$\text{int}\colon \mathbb{B}^n \to \mathbb{N},\ \text{int}(x_{\text{bits}}) = \sum_{i=1}^{n} x_{\text{bits}_i} \cdot 2^{i-1}$$

$$\text{Then, } x_{\text{real}_i} = a + \text{int}\left(x_{\text{bits}_i}\right) \cdot \frac{b-a}{2^k - 1}$$

4 Experiment

The implementation for the two algorithms was written in C++, compiled using MSVC C++20 in Release mode. All floating-point values used were represented on 64 bits. The RNG method that will be used in all of the algorithms relies on the Mersenne Twister of the ⟨random⟩ module, that shall be initialised using a seed sequence of 16 bytes generated by a random device, after which the first 5000 values will be discarded in order to 'warm-up' the generator. The timing of all functions was done using a steady clock of the ⟨chrono⟩ module. The functions were ran at least 40 times in total on each dimension, using each type of improvement, as well as the simulated annealing, on 4 different threads, using asynchronous calls from the ⟨future⟩. In order to maintain a faster runtime, when referring to a bit string throughout the paper, we will actually be using an array of booleans, as there is no practical difference between the two methods of bit representation.
Each call to the Hill Climbing algorithm executed the improvement function on a randomly generated bit string a number of times $k = 10^3$ and returned the best result out of every run. It is important to note that each of the calls were independent of one another. The implementation of the best and worst improvement methods are similar, exploring the entire neighbouring space of each variable. The first improvement version of the implementation has a slightly different implementation, as simply iterating through the bit string until an improvement is found would lead to a bias towards local optima generated by modifying the first half of the bit string. Instead, we must also randomize the order in which each index is being visited each time the neighbouring space is being searched for an improvement. For the sake of showcasing the difference

between a randomized selection and a non-randomized one, we implemented both methods.
The Simulated Annealing was implemented using both a Geometric-Arithmetic and a Geometric cooling method. The first one works as follows:

$$\text{let } T \text{ be the current temperature}, T = a \cdot T + b, a = 0.99, b = 10^{-7}, T_0 = 10^4$$

The stopping condition used was:

$$\text{if } T > 10^{-4}\text{: continue, else stop}$$

The second cooling method used was:

$$\text{let } T \text{ be the current temperature}, T = a \cdot T, a = 0.999, T_0 = 10^4$$

The stopping condition for the second method was:

$$\text{if } T > 10^{-5}\text{: continue, else stop}$$

The probability that a worse neighbouring value will be chosen is:

$$p = e^{-|\frac{x_{\text{new}} - x_{\text{old}}}{T}|}$$

The Geometric Arithmetic implementation will reach a cooled down state in $k_1 = 1844$ steps, while the Geometric one will reach a cooled state in $k_2 = 18412$ steps. Additionally, for the first cooling method we have chosen to repeat the number of steps i at each new temperature value, where i represents the amount of times the temperature had already been changed. Thus, the algorithm did a total number of steps of $\sum_{i=1}^{k_1} i$, meaning $\approx 17 \times 10^5$. For the second cooling method, we have chosen to repeat the number of steps for a constant 100 iterations every time the temperature cooled. Thus, for the second cooling method the algorithm did a total number of steps of $k_2 \cdot 100$, meaning ≈ 1841200 steps.

5 Results

For simplicity, we will now establish some abreviations: Hill Climbing (HC), Simulated Annealing with Geometric Arithmetic cooling (SAGA), Simulated Annealing with Geometric cooling (SAG), Best Improvement(BI), Worst Improvement(WI), First Improvement Randomized (FIR) and First Improvement Non-randomized (FINR).

5.1 Dejong function

Algorithm used	Average f(x)	σ of f(x)	Average runtime(s)
HC BI	0	0	0.146
HC WI	0	0	0.302
HC FIR	11.4351	2.9219	0.21
HC FINR	0	0	0.0936
SAGA	0	0	0.826
SAG	0	0	0.1543

Figure 1: All algotithms applied on 5-dimensional Dejong function

Algorithm used	Average f(x)	σ of f(x)	Average runtime(s)
HC BI	0	0	1
HC WI	0	0	2.01
HC FIR	1.0392	0.4349	0.067
HC FINR	0	0	0.4060
SAGA	0	0	1.37
SAG	0	0	0.1604

Figure 2: All algotithms applied on 10-dimensional Dejong function

Algorithm used	Average f(x)	σ of f(x)	Average runtime(s)
HC BI	0	0	23.74
HC WI	0	0	47.83
HC FIR	114.0378	10.1385	1.62
HC FINR	0	0	6.7262
SAGA	0.0005	0	3.34
SAG	0.0001	0	0.2041

Figure 3: All algotithms applied on 30-dimensional Dejong function

5.2 Michalewicz function

Algorithm used	Average f(x)	σ of f(x)	Average runtime(s)
HC BI	-4.6869	0.0010	0.668
HC WI	-2.7815	0.3264	0.017
HC FIR	-3.4019	0.2426	0.092
HC FINR	-3.6987	0.0002	1.1049
SAGA	-4.6671	0.0178	1.248
SAG	-3.6934	0.0058	1.2351

Figure 4: All algotithms applied on 5-dimensional Michalewicz function

Algorithm used	Average f(x)	σ of f(x)	Average runtime(s)
HC BI	-9.3588	0.078	5.33
HC WI	-4.129	0.3885	0.07
HC FIR	-4.9379	0.3745	0.33
HC FINR	-8.4160	0.0950	8.5899
SAGA	-9.3482	0.1285	2.15
SAG	-8.4426	0.1092	2.4652

Figure 5: All algotithms applied on 10-dimensional Michalewicz function

Algorithm used	Average f(x)	σ of f(x)	Average runtime(s)
HC BI	-27.0491	0.296	128.11
HC WI	-8.1131	0.6712	0.67
HC FIR	-8.9402	0.5629	2.7
HC FINR	-25.5630	0.3005	221.4815
SAGA	-28.1297	0.392	5.78
SAG	-27.1673	0.3446	9.3138

Figure 6: All algotithms applied on 30-dimensional Michalewicz function

5.3 Rastrigin function

Algorithm used	Average f(x)	σ of f(x)	Average runtime(s)
HC BI	0.4975	0.5038	0.198
HC WI	1.7925	0.6293	1.285
HC FIR	14.1032	3.7678	0.070
HC FINR	0.9923	0.5286	0.1795
SAGA	1.2143	0.6362	0.891
SAG	0.8721	0.4933	0.2403

Figure 7: All algotithms applied on 5-dimensional Rastrigin function

Algorithm used	Average f(x)	σ of f(x)	Average runtime(s)
HC BI	3.8668	0.7346	1.64
HC WI	7.6206	1.3159	10.84
HC FIR	65.3611	9.3334	0.24
HC FINR	5.9521	1.0533	1.0602
SAGA	5.2009	1.5669	1.48
SAG	5.1388	1.6275	0.3182

Figure 8: All algotithms applied on 10-dimensional Rastrigin function

Algorithm used	Average f(x)	σ of f(x)	Average runtime(s)
HC BI	28.1095	2.6267	48.18
HC WI	45.6166	3.6626	309.83
HC FIR	346.8527	19.4845	1.84
HC FINR	36.4184	2.4564	22.2274
SAGA	25.2189	7.1704	3.66
SAG	22.9466	7.3625	0.6185

Figure 9: All algotithms applied on 30-dimensional Rastrigin function

5.4 Schwefel function

Algorithm used	Average f(x)	σ of f(x)	Average runtime(s)
HC BI	-2094.7919	0.0871	0.545
HC WI	-1394.9864	166.1433	0.663
HC FIR	-1642.8964	101.9519	0.120
HC FINR	-2070.35	28.9391	0.4296
SAGA	-2094.3337	1.352	1.113
SAG	-2094.6860	0.4072	0.2928

Figure 10: All algotithms applied on 5-dimensional Schwefel function

Algorithm used	Average f(x)	σ of f(x)	Average runtime(s)
HC BI	-4067.7727	48.6378	4.65
HC WI	-1944.4423	186.2629	4.96
HC FIR	-2684.2246	634.6331	5.14
HC FINR	-3911.4676	66.1946	2.6621
SAGA	-4169.2265	33.2826	1.85
SAG	-4184.2282	12.2314	0.4040

Figure 11: All algotithms applied on 10-dimensional Schwefel function

Algorithm used	Average f(x)	σ of f(x)	Average runtime(s)
HC BI	-11409.7169	155.4901	135.54
HC WI	-3442.8535	426.841	137.07
HC FIR	-3934.0969	341.7946	3.41
HC FINR	-10754.7054	134.9628	60.1260
SAGA	-12242.4767	156.8341	4.85
SAG	-12360.1771	110.2043	0.8707

Figure 12: All algotithms applied on 30-dimensional Schwefel function

5.5 Interpretation

On the 5 and 10 dimensional versions of the results, we may notice that the Best Improvement and the Simulated Annealing methods performed similarly, with Simulated Annealing being slightly slower on the 5-dimensional functions and slightly faster on the 10-dimensional functions. When working with the 30 dimensional functions however, their differences become apparent. Simulated Annealing greatly outperforms the hill climbing algorithm, both in runtime and accuracy. Additionally, both first and worst improvement methods were greatly outclassed by the best improvement algorithm, with only an advantage to runtime for the first improvement method, and worst improvement having no redeeming qualities.

6 Conclusions

Given the experiment's results, we may conclude the power of both heuristic algorithms when compared to the deterministic exhaustive search alternative that wouldn't have finished running in the following century, with approximative results that rival the real and expected results. This alternative to computing can be expanded as needed, adding or removing precision easily, depending on the time constraints and quality of outputs required. We have noticed the power of the Best Improvement algorithm, but at the cost of a decently high runtime, which we can assume grows with the number of dimensions added to a target function's domain, although it isn't unreasonable in most use-cases. First improvement gave incredibly fast results, but at a huge cost to accuracy. Worst improvement didn't show any advantages, and it may only apply to very specific functions, with further testing required before a conclusion may be drawn. The strongest algorithm, that seemed to improve in both accuracy and speed while increasing the function's domain was Simulated Annealing, which gave better results on the 30-dimensional functions, and approximately equal results to those of Best improvement on the smaller dimensions tested.

References

[1] "Hill climbing", Wikipedia, Wikimedia Foundation, 23 March 2023, https://en.wikipedia.org/wiki/Hill_climbing

[2] "Simulated annealing", Wikipedia, Wikimedia Foundation, 26 August 2023, https://en.wikipedia.org/wiki/Simulated_annealing

[3] "Hamming distance", Wikipedia, Wikimedia Foundation, 23 September 2023, https://en.wikipedia.org/wiki/Hamming_distance

[4] Hartmut Pohlheim, "GEATbx: Example Functions (single and multi-objective functions) 2 Parametric Optimization", GEATbx, December 2006, http://www.geatbx.com/docu/fcnindex-01.html

[5] Mahdi, Walid, Seyyid Ahmed Medjahed, and Mohammed Ouali. "Performance analysis of simulated annealing cooling schedules in the context of dense image matching." Computación y Sistemas 21.3 (2017): 493-501. https://www.scielo.org.mx/scielo.php?pid=S1405-55462017000300493&script=sci_arttext&tlng=en

[6] Eugen Nicolae Croitoru, Department of Computer Science "Alexandru Ioan Cuza" University of Iasi, Romania, "Teaching: Genetic Algorithms", retrieved October 31, 2023, from https://profs.info.uaic.ro/~eugennc/teaching/ga/