A comparative study of the performance of Hill Climbing, Simulated Annealing and Genetic Algorithms in finding global optima.

Chirvasa Matei & Rotariu George

December 6, 2023

1 Abstract

This paper analyses the differences between trajectory methods and evolution strategy by comparing how good Hill Climbing, Simulated Annealing and Genetic Algorithms are at estimating the global minima of a given function. Trajectory methods consider one candidate solution to the problem that they modify until the stopping condition is met. Evolution Strategy is an optimization technique that is based on the ideas of real life evolution. Being inspired by the process of natural selection, Genetic Algorithms consider a set of possible solutions to be a pupulation of individuals on which four basic operations are performed: mutation, crossover, selection and evaluation. We will study different approaches to the algorithms, namely for Hill Climbing we will consider three neighbour selection methods: Best Improvement, First Improvement and Worst Improvement and for the Genetic Algorithm we will observe how the parent selection and natural selection affect its performance. Key words: Genetic Algorithm, Simulated Annealing, Hill Climbing

2 Introduction

Trajectory methods consider one candidate solution to the problem that they modify until the stopping condition is met. They are named like this because the candidate solutions picked at every iteration of the algorithm can be plotted in the space of possible solutions, thus forming a trajectory. The Trajectory methods we will observe are Hill Climbing and Simulated Annealing.
Hill Climbing was 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[6] 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.
The third algorithm used in this study is the Genetic Algorithm which, as opposed to the first two mentioned algorithms, uses the Evolution Strategy optimization technique. The algorithm considers a set of possible solutions to be a pupulation of individuals on which four basic operations are performed: selection, mutation, crossover, and evaluation. An individual is characterized by a set of parameters known as Genes. Genes are joined into a string to form a Chromosome which represents the genetic interpretation of the solution(individual). The process of selection consists of picking a fixed number of individuals from the existing population based on how fit they are to survive into the next generation. The mutation step is characterized by giving a chance to each individual to alter some of their genes, namely to switch some genes from 0 to 1 and vice versa. This step introduces new data to into the population, thus exploring the fitness landscape layed out by the fitness function we want to maximize. The crossover step consists of a pair of individuals, which we will call parents, that mix their genetic information to create offspring that get added to the population. The last operation consists in the evaluation of each individual's fitness.

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. By studying the differences in their behaviour we can learn when and how to use them in order to make the most use out of the tools we are equipped with.

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[8].

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 $[\text{infimum, supremum}]^n$, 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[7] be 1.

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

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

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

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[10]:

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

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

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

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

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

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

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

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

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

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. The timing of all functions was done using a steady clock of the ⟨chrono⟩ module. The functions were ran at least 30 times in total on each dimension. 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.
The Genetic Algorithm was run with a population size of 200 individuals for 2000 generations and was implemented incrementally. For the fitness interpretations of the function we chose $f_{Negative}(x) = -f(x), f \in \{Michalewicz, Schwefel\}$ and $f_{Division}(x) = \frac{1}{f(x)}, f \in \{Dejong, Rastrigin\}$.At each step in the algorithm's development we added a new functionality and analysed its impact on the data. In its early stages, the algorithm recieved as parameters P_m the probability of a gene mutating and P_x the probability of an individual to be selected as a parent. It consisted of the method in which the parents are selected and the four basic operations: selection, mutation, crossover and evaluation. The parent selection was made bt attributing to each inividual a random value p_x between 0 and 1 and then sort the population based on the random value of each individual. Then, theindividuals are selected in pairs of two until th $P_x > p_x$. If there is an odd number of individuals for which the condition holds, then we give the last individual to make the list a $\frac{1}{2}$ chance to crossover with the first individual who didn't. The selection was made using the concept of the "Wheel of fortune", meaning each individual has a corresponding share on the wheel which we then spin for it to randomly land on the share of an individual. The shares are calculated as follows

$$individual.wheelShare = \frac{fitness(individual)}{\sum_{x \in population} fitness(x)}$$

. The bigger the share you have, the more likely it is to survive and make it to the next generation. The mutation was done by taking each individual and iterating through its genes, giving them the opportunity to mutate with a probability P_m. The crossover was a classic two point corssover, meaning we randomly choose two points in the chromosome of the parents where we cut them, splitting them into 3 strands of genetic information. The children were obtained by interchanging the middle strand between the parents.
The first improvement we made to the algorithm was adding elitism. Elitism is a concept which states that an elite set of individuals will get to survive onto the next generation without having to go through the selection. To put it simply, if you are in the top of the fitness ranking you are guaranteed a seat in the next generation. This is transmitted as the parameter P_{Elite}, representing the percent of the population to be considered elite.
The next improvement we made was to change the selection and crossover strategies. For the select, we tried a greedy approach, having the first 200 fittest individuals make it to the next generation. As for the crossover, instead of having a fixed number of points in which we cut the parents, we decided to parameterize it, so now instead of being 2 cut points and 3 strands there are $nrPoints$ cut

points and $nrPoints + 1$ strands of genetic materialfor the children. We also added mutated copies of the children to the population.
For the final improvement we added two changes in the implementation. The first change is inside the mutation step where we now calculate the average fitness of the population. If an individual's fitness is under the average, then the probability P_m with which he mutates increases. By doing this we allow the algorithm to explore more of the search space instead of converging to a local minima, while also exploiting the above average individuals. The second change was made in the crossover step where we changed the way we distribute the genetic strands to the children. Before this, we distributed them as follows:

$$parent_1 = P_{11}P_{12}...P_{1nrPoint}; parent_2 = P_{21}P_{22}...P_{2nrPoint}$$

$$child_1 = P_{11}P_{22}P_{13}...P_{1nrPoint}; child_2 = P_{21}P_{12}P_{23}...P_{2nrPoint}$$

Now we offer each strand an equal chance of being in $child_1$ and $child_2$. By doing this we now unlock combinations of genetic information that was inaccessible before. After tuning the parameters of the algorithm for each function, we settled on these parameters:

Function name	P_m	P_x	P_{Elite}	$nrpoints$
Dejong	1/100	1	0.1	5
Michalewicz	1/95	1	0.1	30
Rastrigin	1/100	1	0.1	5
Schwefel	1/135	1	0.1	50

Figure 1: List of parameters used in the Genetic Algorithm for th 5-dimensional version of the functions

Function name	P_m	P_x	P_{Elite}	$nrpoints$
Dejong	1/200	1	0.1	5
Michalewicz	1/190	1	0.1	30
Rastrigin	1/200	1	0.1	5
Schwefel	1/270	1	0.1	50

Figure 2: List of parameters used in the Genetic Algorithm for th 10-dimensional version of the functions

Function name	P_m	P_x	P_{Elite}	$nrpoints$
Dejong	1/600	1	0.1	5
Michalewicz	1/570	1	0.1	30
Rastrigin	1/600	1	0.1	5
Schwefel	1/810	1	0.1	50

Figure 3: List of parameters used in the Genetic Algorithm for th 30-dimensional version of the functions

In order to get a better understanding of the Genetic Algorithm we decided to improve our current version of it, thus we introduced Gray encoding in our algorithm. A common problem in Genetic Algorithms is that of premature convergence, meaning getting stuck in a local optima and not being able to ascape from it. This is where Gray encoding comes in handy. In order to understand why Gray encoding is helpful we must first understand what a Hamming cliff is. An example of a Hamming cliff is the transition from 7 to 8 in binary coding. Since 7 is interpreted as 0111 and 8 as 1000, the two numbers are at hamming distance 4 from each other even though their real interpretations are really close to each other. That means we would need to mutate 4 genes inside the chromosome in order to visit a neighbouring value which is improbable since we chose the mutation probability such that, on average, one bit of information changes per chromosome. In the Gray code, the Hamming distance is always one for any two strings (chromosomes) that are adjacent (differing by ε) in the real representation of the coordinates. That is not the case in the standard binary code where a single bit flip at the most significant position dramatically changes the value. By incorporating Gray encoding we get rid of the Hamming cliffs in our search space. All we need is to do is to decode the genetic information from Gray to binary in the evaluation part of the algorithm.
Besides Gray encoding, we made two other changes to the code. First, instead of returning the fittest individual from the last generation, we return the fittest individual among all generations. This is done by checking if the fittest individual in each generation is fitter than the one we memorize as the solution. Second, after we compute this point, we apply a steepest ascent Hill Climber algorithm in order to make sure we are in a local minima.

5 Results

For simplicity, we will now establish some abreviations: Genetic Algorithm(GA), Improved Genetic Algorithm (IGA), Hill Climbing (HC), Simulated Annealing with Geometric Arithmetic cooling (SAGA), Simulated Annealing with Geometric cooling (SAG), Best Improvement(BI), Worst Improvement(WI), FIst Improvement (FI) and FIst Improvement Non-randomized (FINR).

Link to spreadsheet containing all results

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 FI	0	0	0.37
HC FINR	0	0	0.0936
SAGA	0	0	0.826
SAG	0	0	0.1543
GA	0	0	3.5759
IGA	0	0	3.5826

Figure 4: 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 FI	0	0	1.51
HC FINR	0	0	0.4060
SAGA	0	0	1.37
SAG	0	0	0.1604
GA	0	0	5.2156
IGA	0	0	5.5231

Figure 5: 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 FI	0	0	14.7
HC FINR	0	0	6.7262
SAGA	0.0005	0	3.34
SAG	0.0001	0	0.2041
GA	0	0	12.0054
IGA	0	0	13.5386

Figure 6: 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 FI	-4.6856	0.0027	0.53
HC FINR	-3.6987	0.0002	1.1049
SAGA	-4.6671	0.0178	1.248
SAG	-3.6934	0.0058	1.2351
GA	-3.6988	0	3.9537
IGA	-3.6988	0	4.1728

Figure 7: 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 FI	-9.2214	0.1218	2.23
HC FINR	-8.4160	0.0950	8.5899
SAGA	-9.3482	0.1285	2.15
SAG	-8.4426	0.1092	2.4652
GA	-8.6252	0.0388	6.4053
IGA	-8.6271	0.0479	8.3929

Figure 8: 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 FI	-25.5217	0.366	23.29
HC FINR	-25.5630	0.3005	221.4815
SAGA	-28.1297	0.392	5.78
SAG	-27.1673	0.3446	9.3138
GA	-28.0786	0.1702	15.8929
IGA	-28.1342	0.1251	16.8350

Figure 9: 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 FI	0.9077	0.5756	0.44
HC FINR	0.9923	0.5286	0.1795
SAGA	1.2143	0.6362	0.891
SAG	0.8721	0.4933	0.2403
GA	0	0	3.5276
IGA	0	0	3.5316

Figure 10: 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 FI	5.9613	1.2364	1.79
HC FINR	5.9521	1.0533	1.0602
SAGA	5.2009	1.5669	1.48
SAG	5.1388	1.6275	0.3182
GA	0.1235	0.3707	5.4722
IGA	0	0	5.5300

Figure 11: 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 FI	40.3441	5.0542	18.38
HC FINR	36.4184	2.4564	22.2274
SAGA	25.2189	7.1704	3.66
SAG	22.9466	7.3625	0.6185
GA	6.9480	3.9435	12.3174
IGA	0.9956	1.3829	13.2598

Figure 12: 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 FI	-2070.7447	19.5121	0.88
HC FINR	-2070.35	28.9391	0.4296
SAGA	-2094.3337	1.352	1.113
SAG	-2094.6860	0.4072	0.2928
GA	-2094.8242	0.0791	4.3349
IGA	-2094.9144	0	4.8101

Figure 13: 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 FI	-3947.4555	67.7702	3.67
HC FINR	-3911.4676	66.1946	2.6621
SAGA	-4169.2265	33.2826	1.85
SAG	-4184.2282	12.2314	0.4040
GA	-4189.6098	0.1543	6.6127
IGA	4189.8288	0	7.5114

Figure 14: 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 FI	-10859.6574	154.9062	36.66
HC FINR	-10754.7054	134.9628	60.1260
SAGA	-12242.4767	156.8341	4.85
SAG	-12360.1771	110.2043	0.8707
GA	-12556.4507	27.3964	16.2548
IGA	-12561.5907	29.5437	18.1942

Figure 15: All algotithms applied on 30-dimensional Schwefel function

5.5 Interpretation

As can be seen from the 30-dimensional versions of Dejong, Rastrigin and Schwefel, the Genetic Algorithm outperforms Simulated annealing and HillClimbing. Te GA offers a much more stable solution, having a small standard deviation

compared to Simulated Annealing. However, this improvement comes at a cost. We can see the average time of the GA is larger than that of the Simulated Annealing. This is due to choosing a higher population size and number of generations, while also keeping the $P_x = 1$, which means that in every generation we add $\frac{populationsize}{2} \cdot 4$ children to the population, resulting in higher computational demand. Also, on the 30-dimensional version of the Michalewicz function, the GA performed worse than the SA both in results and in computation time. This indicates that either Simulated Annealing is more suitable to find the global minimum for the Michalewicz function, or that there is room for improvement inside the Genetic Algorithm, as it has the potential to surpass the performance of the Simulated Annealing. Judging based off of the other mentioned results, we suspect it is the latter.
We can see the Improved Genetic Algorithm is slightly better than the simple Genetic Algorithm at estimating the global minima while the computation time remains relatively close to that of the simple version. The difference is seen best on the 30-dimensional versions of the Rastrigin and Schwefel function, where the algorithm actually found the global minima. This tells us that the removal of the Hamming walls helped a lot in finding the solution to the two functions. We can also conclude that estimating the global minima of the Michalewicz function is a much harder problem than estimating it for the Dejong, Schwefel or Rastrigin functions.

6 Conclusions

As can be seen from the results, the Genetic Algorithm has the potential to outperform both Simulated Annealing and Hill Climbing. However, in order to do so, we must tweak the parameter values and find what methods of select, mutate, crossover and evaluate best fit our problem in order to overcome premature convergence inside the Genetic Algorithm. Nonetheless, we can see the power of the Evolution Strategy compared to Trajectory Methods.

References

[1] "Genetic algorithm", Wikipedia, Wikimedia Foundation, 20 November 2023, `https://en.wikipedia.org/wiki/Genetic_algorithm`

[2] "A Visual Guide to Evolution Strategies", Ha, David, blog.otoro.net, 2017, `https://towardsdatascience.com/genetic-algorithm-a-simple-and-intuitive-guide-51c04cc1f9ed`

[3] "Genetic Algorithm (GA): A Simple and Intuitive Guide", Dana Bani-Hani, 2020, `https://blog.otoro.net/2017/10/29/visual-evolution-strategies/`

[4] "An analysis of Gray versus binary encoding in genetic search", Uday K. Chakraborty and Cezary Z. Janikow, Information Sci-

ences, volume 156, number 3, pages 253-269, 2003, https://www.sciencedirect.com/science/article/abs/pii/S0020025503001786#preview-section-cited-by

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

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

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

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

[9] 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

[10] 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/