FUNCTION MINIMA USING HILL CLIMBING, SIMULATED ANNEALING AND GENETIC ALGORITHMS

Encut Octavian, grupa 2A1

December 17, 2021

Abstract

Solving optimization problems with deterministic algorithms can be very time consuming, that is why non-deterministic algorithms were invented. These algorithms, such as hill-climbing, simulated-annealing and genetic algorithms can offer a pretty good estimation of the result in a fraction of the time it would take a deterministic algorithm to find a solution. This paper compares the results of finding the global minimum of a function with non-deterministic algorithms to the actual results for the problem while also comparing them against each other to see which one is better.

1 Introduction

In this paper we will observe the results of non-deterministic algorithms when trying to find the global minimum of De Jong 1, Schwefel, Rastrigin and Michalewicz functions and seeing how close they get to the real minimum while also comparing the results from each function to see which one is a better fit for that problem.

2 Methods

We are going to be using three algorithms, the Hill-Climbing algorithm, which has two variations, the Simulated-Annealing algorithm and a Genetic Algorithm.

2.1 Hill-Climbing

Hill-Climbing is a mathematical optimization technique which belongs to the family of local search. The Hill-Climbing algorithm searches for a solution by arbitrarily selecting a solution and then iteratively searching for a better solution from a set of candidates. The candidates are the hamming neighbours of the current solution. Based on the type of improvement method that we choose, the algorithm is looking for either the first candidate that is better or the best one. The algorithm starts with selecting a random solution and evaluating it. After that, depending on which method we choose, first improvement or best improvement we search for either the first hamming neighbour that is better or the best one. After performing the improvement of the current solution, a check is performed to see if the improved solution is better. If it is better, the same procedure starts again,if it is a worse solution it looks for another candidate and if it is equal to the last solution, the search stops concluding that it found the best candidate that it could find. The problem with using hill-climbing is that it can easily get stuck in one basin and concluding that it found the global minimum when in fact it only found a local minimum. To get around this problem we run the algorithm a lot of times (at least 1000) to get a more accurate reading.

2.2 Simulated-Annealing

Simulated annealing is a probabilistic technique for approximating the global optimum of a given function.

The name of the algorithm comes from annealing in metallurgy, a technique involving heating and controlled cooling of a material to alter its molecular structure and improve it. The idea behind this implementation is that a high temperature allows jumps out of a local basins trading a better solution now for the chance to get a better one later in the "annealing" process.

The algorithm starts by arbitrarily selecting a candidate solution and an initial temperature. Then we must have two conditions: the termination condition(the number of steps per temperature) and halting criterion(the steps it takes for the temperature to reach equilibrium). Inside the inner loop a random hamming neighbour of the arbitrarily selected solution is selected as a canditate and evaluated. If it is better, it becomes the new solution and if it's not better we select a random number from 0 to 1 and using a forumula we calculate the probability of the process to go uphill. If the random number we selected is smaller, the solution becomes the candidate even if it's a worse fit for now. The probability of accepting a worse solution decrease with the temperature.

2.3 Genetic Algorithm

In computer science and operations research, a genetic algorithm (GA) is a metaheuristic inspired by the process of natural selection. Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on biologically inspired operators such as mutation, crossover and selection. In a genetic algorithm, a population of candidate solutions (called individuals) to an optimization problem is evolved toward better solutions. Each candidate solution has a set of properties (its chromosomes or genotype) which can be mutated and altered; traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible.

The algorithm starts by arbitrarily selecting a population , usually the population has a size of several hundred individuals. After the initial selection, a rank is given to each individual, this rank is called "fitness" and it represents how fit an individual is to live to the next generation or create offspring. After that a selection is made in which a couple individuals are chosen, either completely randomly or based on how "fit" they are. Those individuals will create offspring using a system called crossover in which genes from each parent is selected, but like in real life mistakes can happen, and those mistakes are represented my mutation. Each children has a low probability to mutate, meaning that one of the chromosomes will change to create a more diverse population. Those children will replace the current population, creating a new generation. Then we give a fitness rank to each individual and the cycle continues until a termination criteria is reached.

2.4 Implementation details

Software and hardware used:

- All algorithms were written in Python
- All algorithms were ran using either a personal computer or Google Colaboratory.

Hill-Climbing:

- The solution is represented by a binary vector from which we can calculate a base 10 solution
- The precision chosen for representing the solution and candidate solutions is 10e-5
- The number of iterations for each hill-climb is 1000

Simulated-Annealing

- The solution is represented by a binary vector
- The initial temperature is set to 50
- Temperature decreases by 1%
- The halting criterion is temperature <10e-5
- The termination condition is i <1000

- The precision chosen for representing the solution and candidate solution is 10e-5

Genetic Algorithm

- The population is made out of 200 individuals
- The fitness is created by taking the multiplicative inverse of the solution
- The selection is made by choosing the best 50 individuals in a generation based on fitness
- The crossover is done by combining both parents, each chromosome having a 50% chance to be chosen from either the first parent or the second parent
- The probability to mutate is 1%
- The termination condition is reaching 1000 generations
- The precision chosen for representing the solution and candidate solution is 10e-5

3 Studied Functions

3.1 De Jong1 Function (Sphere Function)

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

where n is the number of parameters accepted by the function. $-5.12 \leq x_i \leq 5.12$, where i = 1, ..., n. Global minimum $x_i = 0$ i = 1,..., n with $f(x) = 0$.

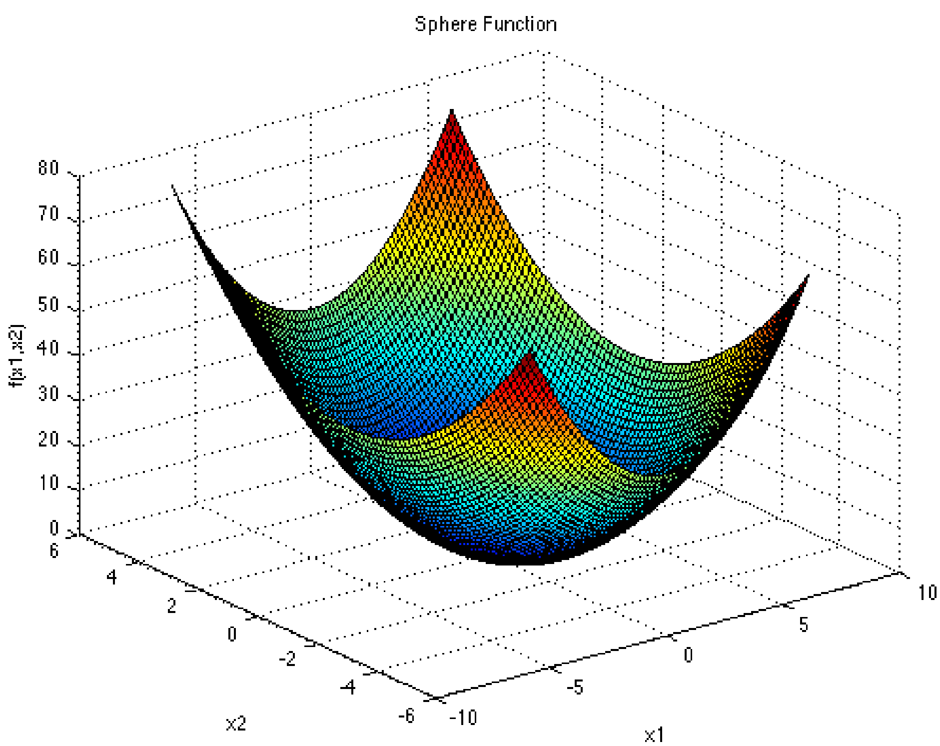

Figure 1: De Jong1 function with n=2

Size	Mean	Sample St.Dev.	Min	Max	Avg Time(s)	Min Time(s)	Max Time(s)
5	0.000000	0.000000	0.000000	0.000000	68.5473	59.2135	74.8802
10	0.000000	0.000000	0.000000	0.000000	500.3573	453.8468	554.5646
30	0.000000	0.000000	0.000000	0.000000	14173.5399	13126.4075	15053.9845

Table 1: Hill-Climbing First-Improvement on De Jong function

Size	Mean	Sample St.Dev.	Min	Max	Avg Time(s)	Min Time(s)	Max Time(s)
5	0.000000	0.000000	0.000000	0.000000	123.8004	109.1239	141.8235
10	0.000000	0.000000	0.000000	0.000000	938.0467	834.3735	1056.0325
30	0.000000	0.000000	0.000000	0.000000	19888.9602	19000.1243	20855.5847

Table 2: Hill-Climbing Best-Improvement on De Jong function

Size	Mean	Sample St.Dev.	Min	Max	Avg Time(s)	Min Time(s)	Max Time(s)
5	0.000033	0.000015	0.000001	0.000054	66.2173	61.0691	70.1234
10	0.000041	0.000008	0.000029	0.000056	118.8072	117.0685	120.1234
30	0.001696	0.000369	0.00108	0.00218	365.2198	361.1083	370.1234

Table 3: Simulated-Annealing on De Jong function

Size	Mean	Sample St.Dev.	Min	Max	Avg Time(s)	Min Time(s)	Max Time(s)
5	0.000000	0.000000	0.000000	0.000000	17.418103	14.560814	26.100237
10	0.000000	0.000000	0.000000	0.000000	31.691707	26.670947	36.265024
30	0.000005	0.000001	0.000002	0.000008	98.620819	83.670668	194.444578

Table 4: Genetic Algorithm on De Jong function

Observations

From these tables we can deduce that both hill-climbing methods produce very accurate results but they take a very long time to run. The simulated-annealing process is a lot faster than hill-climbing, but produces less accurate results. The genetic algorithm is even faster than simulated-annealing and produces results almost as good as the hill-climbing in a fraction of the time, so using this type of method for finding the minimum of De Jong's function is the best choice.

3.2 Schwefel's Function

$$f(x) = 418.9829n - \sum_{i=1}^{n} x_i sin\left(\sqrt{|x_i|}\right)$$

where n is the number of parameters accepted by the function. This function's domain is: $-5.12 \leq x_i \leq 5.12$, where i = 1,..., n. Global minima $x^* = (420.9687, ..., 420.9687)$ with $f(x) = 0$.

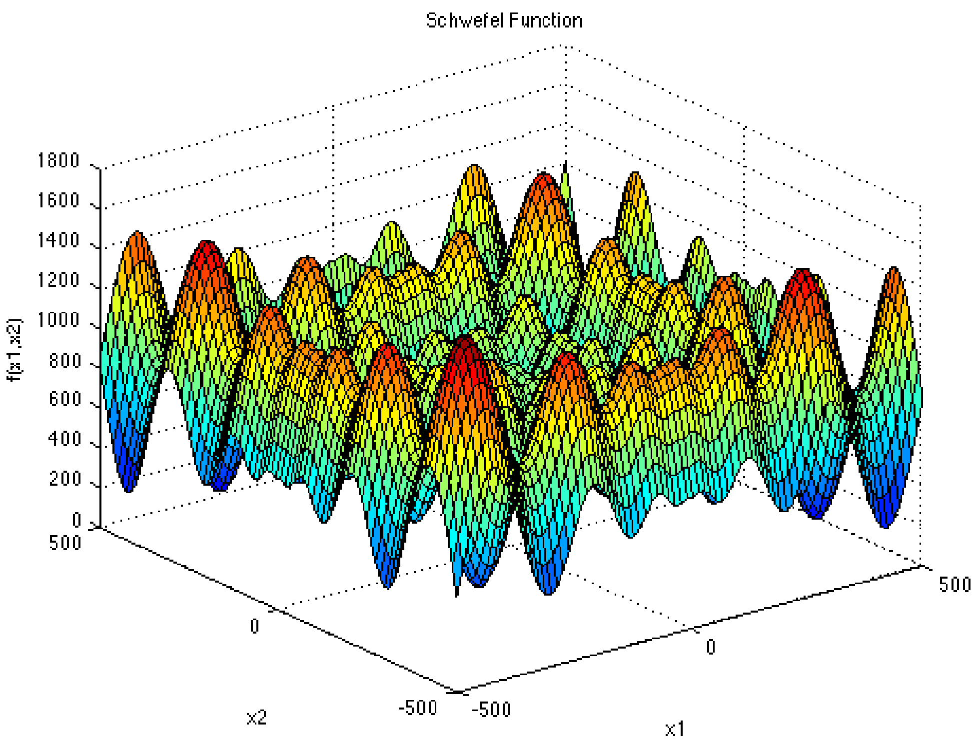

Figure 2: Schwefel's function with 2 parameters

Size	Mean	Sample St.Dev.	Min	Max	Avg Time(s)	Min Time(s)	Max Time(s)
5	0.274816	0.105933	0.10132	0.425120	133.3520	109.9116	156.5571
10	279.18907	19.88445	248.34585	314.45110	985.3349	830.2621	1175.8302
30	2058.00285	178.88529	1764.83707	2368.4368	28786.6038	22855.1898	33751.4288

Table 5: Hill-Climbing First-Improvement on Schwefel's function

Size	Mean	Sample St.Dev.	Min	Max	Avg Time(s)	Min Time(s)	Max Time(s)
5	0.132660	0.016748	0.104989	0.164848	229.1922	189.1036	274.4016
10	156.014249	20.168827	118.54435	187.403500	1827.6264	1471.1080	2107.0800
30	2100.96066	98.59600	1925.42023	2243.53634	38994.2552	37234.1234	41575.1379

Table 6: Hill-Climbing Best-Improvement on Schwefel's function

Size	Mean	Sample St.Dev.	Min	Max	Avg Time(s)	Min Time(s)	Max Time(s)
5	0.261241	0.032336	0.208033	0.312369	82.5830	79.4086	85.1234
10	8.545116	31.351386	0.622834	152.542143	157.1101	153.9898	162.1234
30	446.79320	6.344279	437.41126	459.06086	493.1323	483.9060	532.1234

Table 7: Simulated-Annealing on Schwefel's function

Size	Mean	Sample St.Dev.	Min	Max	Avg Time(s)	Min Time(s)	Max Time(s)
5	0.277037	0.109828	0.103753	0.518401	22.332010	20.003457	25.327791
10	0.557794	0.116602	0.314260	0.830714	42.533993	40.296923	47.311355
30	14.227845	18.987292	1.384587	69.843015	127.846150	114.08566	145.116164

Table 8: Genetic Algorithm on Schwefel's function

Observations

When studying Schwefel's function we can deduce that using hill-climbing is not efficient and not accurate when going past 5 parameters. Going to 30 parameters is a really bad ideea, the algorithm being very inefficient and really inaccurate. The simulated annealing is a lot more accurate than either of the hill climbing methods and runs in a fraction of the time, but it is a lot worse than the genetic algorithm, which produces very good results in an even lower amount of time. The high standard deviation in the case of the genetic algorithm for 30 parameters is caused by a few outliers, most of the results being very good (between 1 and 2), but some results were around 30 or even 60, meaning that we may need to run this method a few times to get a good result because this function has a lot of local minima and it is possible for the algorithm to get stuck in a local minimum, even if it is unlikely. Having more generations can fix that problem aswell.

3.3 Rastrigin's Function

$$f(x) = 10n + \sum_{i=1}^{n} x_i^2 - 10cos(2\pi x_i)$$

where n is the number of parameters accepted by the function. $-5.12 \leq x_i \leq 5.12$, where i = 1, ..., d. Global minima t $x^* = (0, ..., 0)$ with $f(x) = 0$.

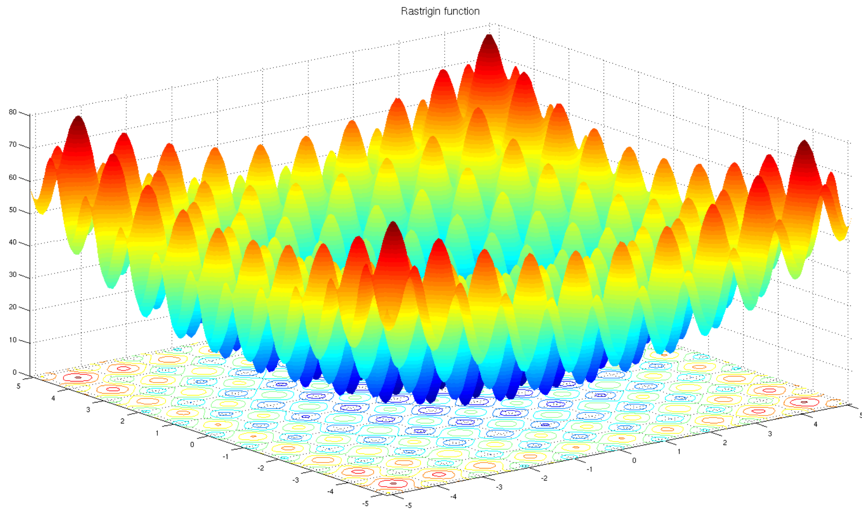

Figure 3: Rastrigin's function with 2 parameters

Size	Mean	Sample St.Dev.	Min	Max	Avg Time(s)	Min Time(s)	Max Time(s)
5	0.742064	0.116895	0.543124	0.99496	53.5747	44.8966	63.6960
10	5.247774	0.624267	4.231234	6.179114	404.6980	331.9070	465.6980
30	29.156883	2.343697	25.423143	34.277929	10918.7390	9030.0356	12736.3827

Table 9: Hill-Climbing First-Improvement on Rastrigin's function

Size	Mean	Sample St.Dev.	Min	Max	Avg Time(s)	Min Time(s)	Max Time(s)
5	0.585060	0.218539	0.243123	0.99496	90.1067	76.0874	107.8406
10	3.959848	0.305734	3.432144	4.461564	694.5498	578.9362	806.2987
30	23.509789	5.234313	14.234123	32.683032	14498.2556	13960.38	15321.6548

Table 10: Hill-Climbing Best-Improvement on Rastrigin's function

Size	Mean	Sample St.Dev.	Min	Max	Avg Time(s)	Min Time(s)	Max Time(s)
5	2.745102	0.288241	2.214234	3.235909	88.8165	87.4047	90.9421
10	7.095952	0.677361	5.984986	8.234124	178.2118	175.9785	180.1765
30	23.532717	1.789885	20.359655	26.2421234	546.7618	542.5505	520.626

Table 11: Simulated-Annealing on Rastrigin's function

Size	Mean	Sample St.Dev.	Min	Max	Avg Time(s)	Min Time(s)	Max Time(s)
5	0.018495	0.011811	0.007292	0.037648	17.366951	14.993816	20.010142
10	2.141430	2.099488	0.000000	8.630882	32.457615	29.429127	37.086191
30	18.712908	6.761577	9.871049	34.280611	96.72490	89.73556	111.20472

Table 12: Genetic Algorithm on Rastrigin's function

Observations Looking at the tables we can deduce that hill-climbing generates better results than simulated annealing for a small number of parameters. Going to a bigger number simulated-annealing generates almost the same results, if not better in less than 10% of the time it takes any hill-climbing algorithm to run. The genetic algorihm generates results that are even more accurate than simulated annealing while also being faster, so for this function genetic algorithms are the best way to aproximate a minimum.

3.4 Michalewecz's Function

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

where n is the number of parameters accepted by the function. $-5.12 \leq x_i \leq 5.12$, where i = 1, ..., n. Global minimum $f(x) = -4.687658$ for n=5, $f(x) = -9.66015$ for n=10 and $f(x) = -29.630883$ for n=30.

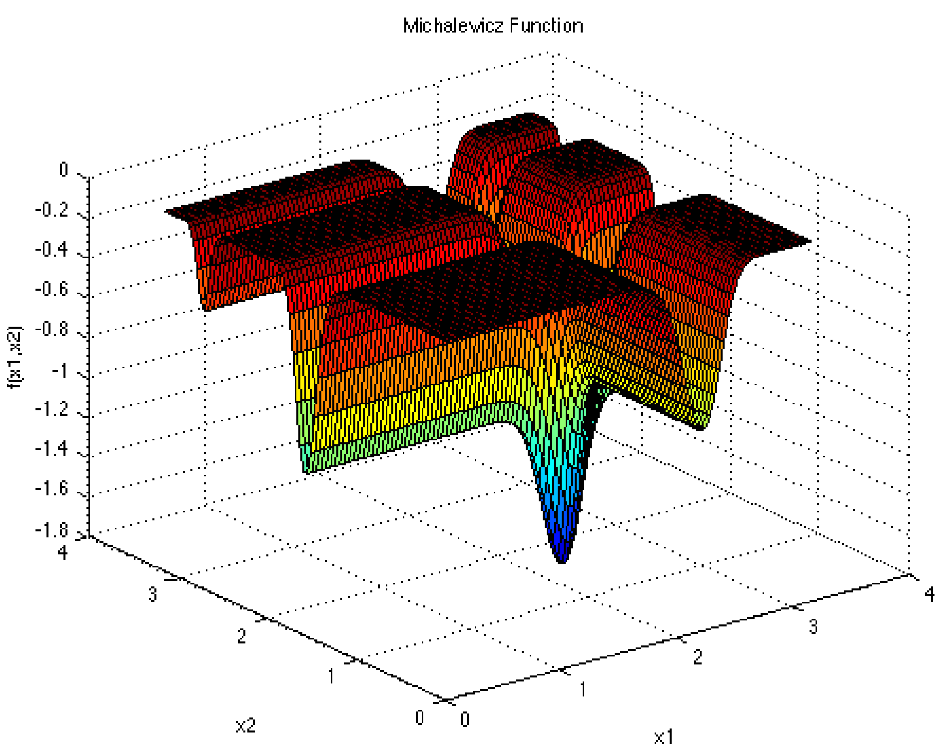

Figure 4: Michalewecz's function with 2 parameters

Size	Mean	Sample St.Dev.	Min	Max	Avg Time(s)	Min Time(s)	Max Time(s)
5	-3.698848	0.000006	-3.698857	-3.698839	64.9485	60.2566	70.2341
10	-8.457517	0.052155	-8.543533	-8.377716	495.9911	439.8548	556.8234
30	-27.08001	0.648812	-28.32424	-25.8904	12281.7440	11065.3205	13234.5326

Table 13: Hill-Climbing First-Improvement on Michalewecz's function

Size	Mean	Sample St.Dev.	Min	Max	Avg Time(s)	Min Time(s)	Max Time(s)
5	-3.691741	0.004468	-3.698857	-3.682342	95.3952	90.4493	102.2134
10	-8.637782	0.009643	-8.65456	-8.62259	758.8783	715.4169	813.8432
30	-26.28852	0.111116	-26.47421	-26.0960	20439.5029	18757.2536	22023.6552

Table 14: Hill-Climbing Best-Improvement on Michalewecz's function

Size	Mean	Sample St.Dev.	Min	Max	Avg Time(s)	Min Time(s)	Max Time(s)
5	-3.513630	0.012856	-3.535497	-3.494317	109.4717	97.1697	120.1234
10	-8.149670	0.049045	-8.222273	-8.07726	192.7311	184.5886	221.1214
30	-27.73979	0.193680	-28.02385	-27.37092	582.9556	567.0276	627.6422

Table 15: Simulated-Annealing on Michalewecz's function

Size	Mean	Sample St.Dev.	Min	Max	Avg Time(s)	Min Time(s)	Max Time(s)
5	-3.638990	0.075892	-3.698857	-3.382724	16.576502	14.475406	19.822184
10	-8.217577	0.286733	-8.653518	-7.235098	31.255236	26.694520	35.771760
30	-26.51297	0.424171	-27.30908	-25.66841	93.173776	77.798829	106.693177

Table 16: Genetic Algorithm on Michalewecz's function

Observations

Looking at the tables we can deduce that simulated annealing generates better results than hill climbing even if they are pretty close. The annealing algorithm runs much faster than any of the hill climbing algorithms. Best improvement hill climbing is almost twice as slow compared to first improvement. The genetic algorithm while being a lot faster than even simulated annealing generates decent results, but the standard deviation is fairly high so we need a high number of runs to find a very good result.

4 Conclusion

This paper compared Hill-Climbing, Simulated Annealing and Genetic Algorithms in a multitude of situations and showed both the advantages and disadvantages of implementing a non-deterministic algorithm. On one hand these algorithms are relatively easy to implement, are faster is most situations and can produce accurate results. On the other hand the results they generate aren't always accurate, even if they do get pretty close to the actual result. The genetic algorithm is better in almost every way than both, hill-climbing and simulated annealing, because it runs in a very short amount of time and generates really good results in most situations, even if the results usually vary a bit from run to run, they are on average better than either algorithm used in this paper. Better results are possible, especially for the genetic algorithm because there are a lot of things that we can change, but this paper proved that, even without a lot of tuning, in most situations this type of algorithm is a better fit than any of the other algorithms tested.

Biblography

1. Jon Fincher - Reading and Writing CSV Files in Python https://realpython.com/python-csv
2. Wikipedia - Simulated annealing https://en.wikipedia.org/wiki/Simulated_annealing
3. Wikipedia - Hill Climbing https://en.wikipedia.org/wiki/Hill_climbing
4. Wikipedia - Rastrigin Function https://en.wikipedia.org/wiki/Rastrigin_function
5. Derek Bingham - Virtual Library of Simulation Experiments: Test Functions and Datasets For Michalevicz Function https://www.sfu.ca/~ssurjano/michal.html
6. GEATbx: Genetic and Evolutionary Algorithm Toolbox for use with Matlab http://www.geatbx.com/docu/fcnindex-01.html
7. Derek Bingham - Virtual Library of Simulation Experiments: Test Functions and Datasets For Schwefel Function https://www.sfu.ca/~ssurjano/schwef.html
8. Eugen Croitoru - Genetic Algorithms https://profs.info.uaic.ro/~eugennc/teaching/ga/
9. https://www.overleaf.com
10. edureka! - Hill Climbing algorithm https://www.youtube.com/watch?v=_ThdIOA9Lbk&t=764s
11. Computerphile - Hill Climbing Algorithm & Artificial Intelligence https://www.youtube.com/watch?v=oSdPmxRCWws
12. Dr. Trefor Bazett - Intro to LaTeX : Learn to write beautiful math equations https://www.youtube.com/watch?v=Jp0lPj2-DQA
13. Neal Holtschulte,Melanie Moses - Should Every Man be an Island? https://www.cs.unm.edu/~neal.holts/dga/paper/holtschulte2013island.pdf
14. https://latex-tutorial.com/tutorials/
15. Towards Data Science - Introduction to Genetic Algorithms Including Example Code https://bit.ly/32bGIRU
16. Wikipedia - Genetic Algorithms https://en.wikipedia.org/wiki/Genetic_algorithm
17. GeeksforGeeks-Genetic Algorithms https://www.geeksforgeeks.org/genetic-algorithms/
18. Jason Brownlee - Simple Genetic Algorithm From Scratch in Python https://machinelearningmastery.com/simple-genetic-algorithm-from-scratch-in-python/
19. PyGAD - Python Genetic Algorithm https://pygad.readthedocs.io/en/latest/