Genetic Algorithms
Homework 1

Ciobotaru Mihai , 2B3

November 9, 2021

Abstract

This research paper covers the definition , implementation and comparison based on experiments between Hill Climbing algorithms and Simulated Annealing , both being genetic algorithms.

Contents

1 Introduction

This report aims to present two important algorithms used in genetic programming : Hill Climbing and Simulated Annealing . In the following pages I will explain what a genetic algorithm is , how H.C. and S.A. work and show their power of optimization through an experiment . This experiment involves search for the global minima of four different functions with the algorithms mentioned earlier . With the help of the results , we will make a comparison between them and draw conclusions about them .

2 Algorithms

2.1 Genetic Algorithms

In computer science and operations research, a genetic algorithm (GA) is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). 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. Some examples of GA applications include optimizing decision trees for better performance, automatically solve sudoku puzzles, hyperparameter optimization, etc.

2.2 Hill Climbing

A hill-climbing algorithm is a local search algorithm that moves continuously upward (increasing) until the best solution is attained. This algorithm comes to an end when the peak is reached.

This algorithm has a node that comprises two parts: state and value. It begins with a non-optimal state (the hill's base) and upgrades this state until a certain precondition is met. The heuristic function is used as the basis for this precondition. The process of continuous improvement of the current state of iteration can be termed as climbing. This explains why the algorithm is termed as a hill-climbing algorithm.

A hill-climbing algorithm's objective is to attain an optimal state that is an upgrade of the existing state. When the current state is improved, the algorithm will perform further incremental changes to the improved state. This process will continue until a peak solution is achieved. The peak state cannot undergo further improvements.

Algorithm 1 Hill Climbing

```
bestSolution = $generateRandomSolution()$
bestScore = $evaluateSolution($bestSolution$)$
$for(i = 0; i < iterations; ++i)$
    $generateNeighbours($bestSolution$)$
    $for(auto\ candidate : neighbours)$
        newScore = $evaluateSolution($candidate$)$
        $if($newScore < bestScore$)$
            bestScore = newScore
            bestSolution = newSolution
```

2.3 Simulated annealing

Simulated Annealing (SA) is an effective and general form of optimization. It is useful in finding global optima in the presence of large numbers of local optima. "Annealing" refers to an analogy with thermodynamics, specifically with the way that metals cool and anneal. Simulated annealing uses the objective function of an optimization problem instead of the energy of a material.

Implementation of SA is surprisingly simple. The algorithm is basically hill-climbing except instead of picking the best move, it picks a random move. If the selected move improves the solution, then it is always accepted. Otherwise, the algorithm makes the move anyway with some probability less than 1. The probability decreases exponentially with the "badness" of the move, which is the amount deltaE by which the solution is worsened (i.e., energy is increased.)

The probability of accepting a worse solution :
Prob = 1- Prob(accepting uphill move) = 1 - $e^{\frac{-\Delta difference}{temperature}}$ $\quad\Delta$ difference=$|bestValue - newValue|$

A parameter T is also used to determine this probability. It is analogous to temperature in an annealing system. At higher values of T, uphill moves are more likely to occur. As T tends to zero, they become more and more unlikely, until the algorithm behaves more or less like hill-climbing. In a typical SA optimization, T starts high and is gradually decreased according to an "annealing schedule". The parameter k is some constant that relates temperature to energy (in nature it is Boltzmann's constant.)

Simulated annealing is typically used in discrete, but very large, configuration spaces, such as the set of possible orders of cities in the Traveling Salesman problem and in VLSI routing. It has a broad range of application that is still being explored.

Algorithm 2 Simulated Annealing

```
bestSolution = $generateRandomSolution()$
bestScore = $evaluateSolution($bestSolution$)$
$for(i = 0; i < iterations; ++i)$
    $generateNeighbours($bestSolution$)$
    $for(auto\ candidate : neighbours)$
        newScore = $evaluateSolution($candidate$)$
        $if($newScore < bestScore$)$
            bestScore = newScore
            bestSolution = newSolution
        $else\ if(exp(($bestScore − newScore$)\ /\ $temperature$))$
            bestScore = newScore
            bestSolution = newSolution
            coolTemperature()
```

For making this report , the code I used started with a temperature of 100 and for every time the algorithm enters the branch of choosing a worse solution the temperature cools down by 0.995 of it's original value .

2.4 Generating solutions

Both Hill Climbing Algorithm and Simulated Annealing purpose is to produce better solutions over more iterations . This is accomplished by choosing one chromosome (solution) every iteration and mutating its genes in order to get a generation of new chromosome which we will use in the next iteration to pick a new chromosome . This process represent the essence of these genetic algorithms .

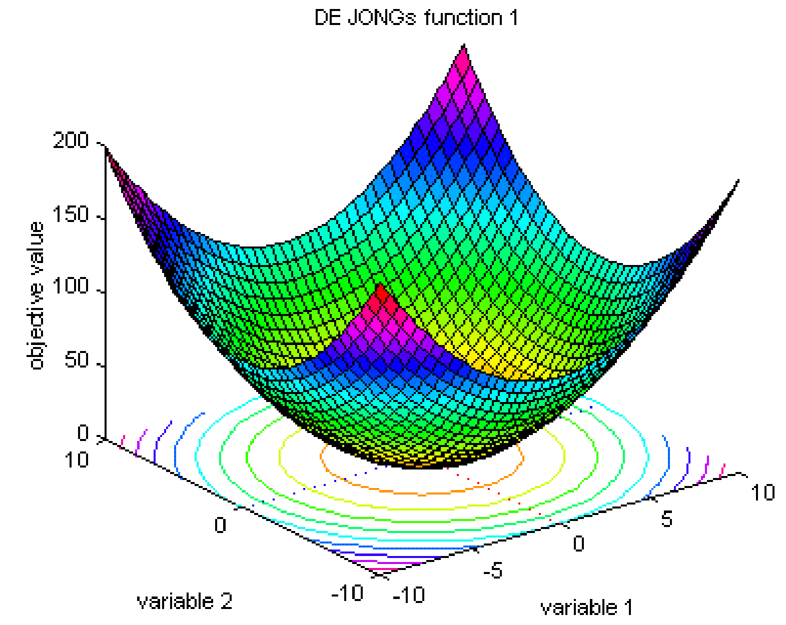
For the code used in this experiment I generated the neighbours in the same way for both algorithms . For every two bits from the end to the start of the chromosome I I inverted their value thus making a generations of n - 1 chromosomes starting from one chromosome, where n is the size of starting chromosome .

3 Functions to be tested

3.1 De Jong‘s function .

The simplest test function is De Jong’s function 1. It is also known as sphere model. It is continuos, convex and unimodal.

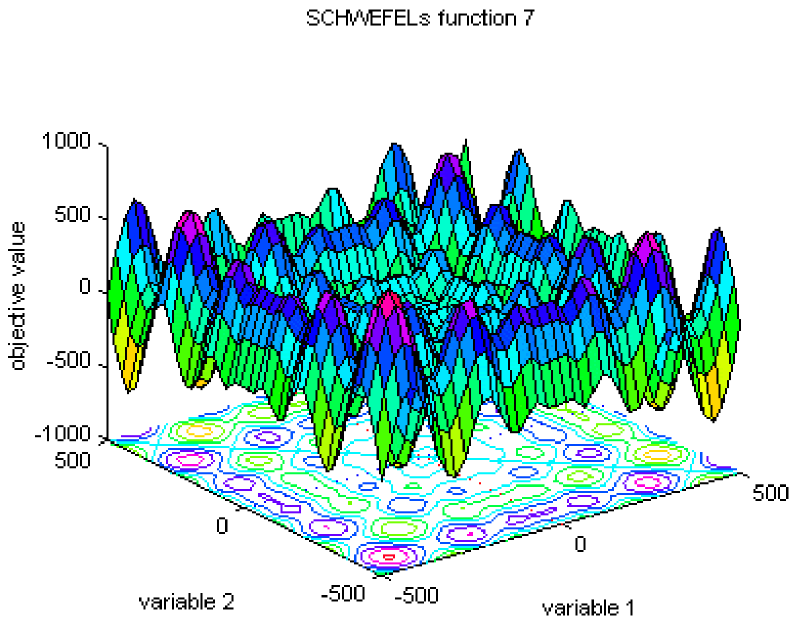
f : $[-5,12,5,12]^n \Rightarrow R, f(x) = \sum_{i=1}^{n} x_i^2$



3.2 Schwefel’s function .

Schwefel’s function [Sch81] is deceptive in that the global minimum is geometrically distant, over the parameter space, from the next best local minima. Therefore, the search algorithms are potentially prone to convergence in the wrong direction.

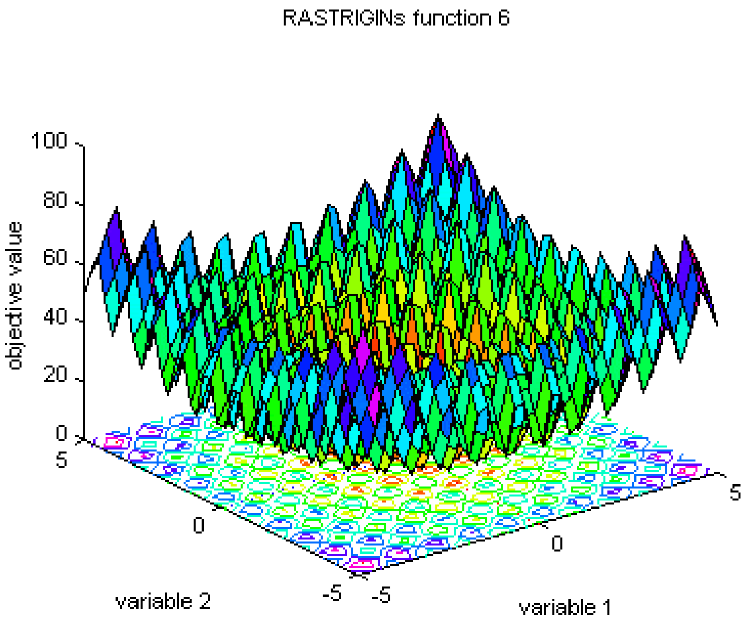
f : $[-500,500]^n \Rightarrow R, f(x) = \sum_{i=1}^{n} -x_i \cdot sin(\sqrt{|x_i|})$



3.3 Rastrigin's function .

Rastrigin's function is based on function 1 with the addition of cosine modulation to produce many local minima. Thus, the test function is highly multimodal. However, the location of the minima are regularly distributed.

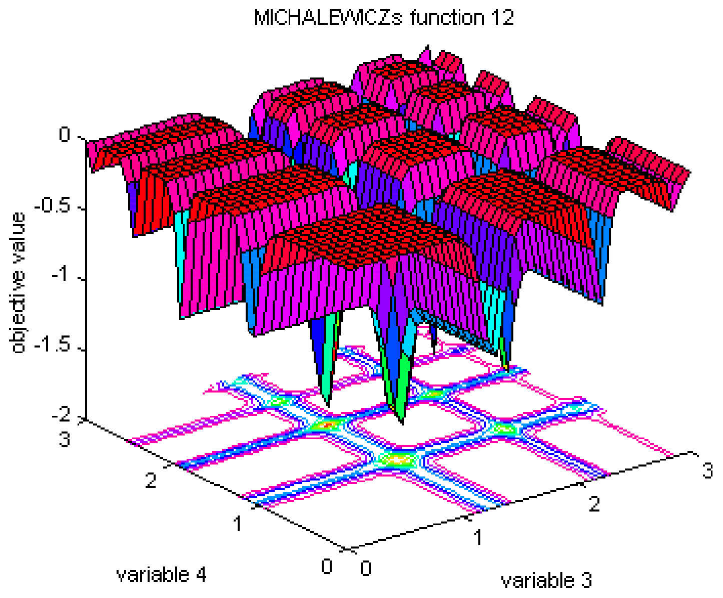
f : $[-5,12,5,12]^n \Rightarrow R, f(x) = 10 \cdot n + \sum_{i=1}^{n} (x_i^2 - 10 \cdot (2 \cdot \pi \cdot x_i))$



3.4 Michalewicz's function .

The Michalewicz function [Mic92] is a multimodal test function (n! local optima). The parameter m defines the "steepness" of the valleys or edges. Larger m leads to more difficult search. For very large m the function behaves like a needle in the haystack (the function values for points in the space outside the narrow peaks give very little information on the location of the global optimum).

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



4 Experiment

For this experiment we will be comparing the time and value result‘s of Hill Climbing algorithms and Simulated Annealing in relation with the functions presented above . Each algorithm goes through 500 generations and tries to find the global minimum . For each algorithm we will run it for 400 times in order to have precise results . We will test each function each for the 5th ,10th and 30th dimension with a precision of 5 decimals. We are interested in finding the best,worst and average time and values and inserting them in tables to have a good visual representation of the results . The algorithms depicted will be implemented in C++ ,compiled with g++ and ran on Ubuntu 18.04.6 LTS OS with Intel® Core™ i5-8265U CPU @ 1.60GHz × 8 processor .

4.1 De Jong‘s 1 function .

Dimensions	Algorithm	Average	Worst	Best	Avg. Time(s)
5	FIHC	0.00000	0.00001	0.00000	0.50772
	BIHC	0.00000	0.00001	0.00000	0.52688
	SA	0.00001	0.00003	0.00000	0.53893
10	FIHC	0.00001	0.00001	0.00000	1.76225
	BIHC	0.00001	0.00001	0.00000	1.85105
	SA	0.00001	0.00001	0.00000	1.88168
30	FIHC	0.00002	0.00002	0.00001	12.41868
	BIHC	0.00002	0.00002	0.00001	15.14539
	SA	0.00002	0.00002	0.00001	15.23265

4.2 Schwefel’s function .

Dimensions	Algorithm	Average	Worst	Best	Avg. Time(s)
5	FIHC	-2090.77954	-1701.52869	-2094.91444	1.04017
	BIHC	-2091.86359	1701.61822	-2094.91444	1.08996
	SA	-2084.94706	-1701.49871	-2094.91443	1.10650
10	FIHC	-4178.62142	-3796.01101	-4189.82880	3.61197
	BIHC	-4182.73306	-3796.08427	-4189.82887	4.00180
	SA	-4167.90092	-3796.08427	-4189.82633	4.02178
30	FIHC	-12242.16245	-10432.01064	-12568.95746	8.72211
	BIHC	-12551.84651	-12175.45655	-12569.48148	17.91231
	SA	-11957.89946	-12491.76075	-12569.37348	16.52397

4.3 Rastrigin’s function .

Dimensions	Algorithm	Average	Worst	Best	Avg. Time(s)
5	FIHC	6.49286	21.01186	0.00025	0.52538
	BIHC	1.43719	11.13935	0.00010	0.53120
	SA	3.04881	16.08704	0.00010	0.54977
10	FIHC	12.45414	27.20517	1.24021	1.82583
	BIHC	2.79471	22.23488	0.00035	1.89475
	SA	6.47576	29.62160	0.00081	1.95420
30	FIHC	70.49311	21.01064	1.214796	9.10564
	BIHC	31.18246	9.90016	0.00244	9.19241
	SA	28.43003	10.28954	0.00306	9.18831

4.4 Michalewicz's function .

Dimensions	Algorithm	Average	Worst	Best	Avg. Time(s)
5	FIHC	-4.14842	-3.00788	-4.68752	0.53119
	BIHC	-4.12068	-2.53429	-4.68764	0.54422
	SA	-4.11096	-2.03949	-4.68757	0.54804
10	FIHC	-8.17759	-6.38718	-9.51387	1.88050
	BIHC	-8.29387	-6.01194	-9.59792	1.98624
	SA	-8.29976	-5.66804	-9.55632	1.99319
30	FIHC	-24.20723	-20.87499	-26.85518	7.67801
	BIHC	-25.01129	-22.67337	-27.47162	9.19241
	SA	-24.93673	-22.32254	-27.20926	9.18831

5 Comparison

The results shown above depict the efficiency of each algorithm in finding the global minima . Looking at the results , we can deduce that First Improvement Hill Climbing (nearest ascent) takes the least time to run but got the worst values . Best Improvement Hill Climbing (steepest ascent) returns the best values but takes a long time to run . Simulated Annealing takes the most time to run and returns values close to the Best Improvement Hill Climbing Algorithm .

6 Conclusion

This study proved the power of optimization granted by evolutionary algorithms . This branch of programming represents a great asset for optimization . The report above succeeded to show that by presenting these algorithms solving difficult computing problems .

References

[1] Wikipedia - Hill Climbing | *https://en.wikipedia.org/wiki/Hill_climbing*

[2] Wikipedia - Simulated annealing | *https://en.wikipedia.org/wiki/Simulated_annealing*

[3] Wikipedia - Genetic algorithm | *https://en.wikipedia.org/wiki/Genetic_algorithm*

[4] Eugen Croitoru: Teaching: Genetic Algorithms | *https://profs.info.uaic.ro/∼eugennc/teaching/ga/*

[5] GEATbx - Example Functions | *http://www.geatbx.com/docu/fcnindex-01.html*