A hybrid method for solving optimization problems using Hill Climbing and a genetic algorithm

Anda Buinoschi MOC2, Luncașu Bogdan MOC1

Abstract

Methods inspired from nature gain their place more and more in optimization problems due to their good performances and their nature to approximate the global optima in a timely manner. This paper showcase a hybrid method for minimizing a set of functions based on Hill Climbing algorithm and a genetic algorithm. It achieves good results, better than the standalone algorithms. Our experiments on different simulations can prove this.

1 Introduction

Algorithms based on nature evolution are a great tool to solve optimization problems. Genetic algorithms and Hill Climbing are two methods that prove this. Although, a genetic algorithm can converge to a global minimum, it is a mundane work to find the right number of generations in which you can reach that point. Hill Climbing is an iterative local search method based on a binary encoding for points drawn from the interval of definition and searching between their neighbours a better value function; if a neighbour is better than the current point, then we update the point as being the neighbour and search again through the neighbours. A genetic algorithm is inspired by biology and genetics. Each point has a binary representation in the role of chromosome and there have been defined some operations (selection, cross-over, mutation) based on evolutionary systems in order to get better candidate solutions within each generation of chromosomes. Our work consists in running Hill Climbing, a genetic algorithm and a hybrid method between these two, on a set of testing functions and an analysis of their performance.

2 Methods

2.1 Number representation

Considering that a function f has the following definition: $f : [a, b] \to \mathbb{R}$, then the interval is split in $N = (b - a) \cdot 10^d$ sub-intervals, where d denotes the

precision that we want. The binary number will have $n = ceil(log_2 N)$ bits, where *ceil* represents superior integer part function, and its decoding will be based on formula

$$x_{real} = a + \frac{decimal(x_{binary}) \cdot (b - a)}{2^n - 1},$$

where *decimal* is the ordinary decoding from binary into integer function.

2.2 Algorithms

2.2.1 Hill Climbing

Hill Climbing is a iterative local search method for optimization. It is in the family of local search methods because they are based on evaluation of the neighbour points around a candidate solution. Each candidate solution has a binary representation and it lays in the function definition interval. The pseudocode of Hill Climbing algorithm is the following:

Algorithm 1: Hill Climbing algorithm

```
t = 0;
initialize best, MAX;
while t ≤ MAX do
    local = FALSE;
    select a candidate solution (bitstring) v_c at random;
    evaluate v_c;
    repeat
        v_n = Improve(Neighbourhood(v_c));
        if eval(v_n) is better than eval(v_c) then
            v_c = v_n;
        else
            local = TRUE;
        end
    until local;
    t = t + 1;
    if v_c is better than best then
        best = v_c;
end
```

Function $Neighbourhood$ generates the neighbours of the candidate solution v_c, by switching a bit in the candidate solution (according to Hemming 1 neighbourhood) and function $Improve$ returns the best neighbour so far v_n according to method first improvement or best improvement. The method first improvement returns the first best neighbour found so far, while best improvement returns the best neighbour out of the entire neighbourhood. Also, the evaluation inside the if statements of the algorithm have to be changed accordingly: if the problem is to find the minimum of the function then it is needed to check if $eval(v_n) < eval(v_c)$ and $v_c < best$ and vice-versa if you need to find the maximum.

2.2.2 Genetic algorithm

Inspired by genetics, this algorithm is based on operators such as mutation and crossover which can be found in many biological structures. Its idea is based on evolution of candidate solutions from a generation to another, maintaining a population of solutions and it takes place under the control of a fitness function which represent how good is a candidate for the problem. A candidate solution is called chromosome which is formed from genes (in our case, the bits that are in a representation) and these genes have some possible values called alleles (0 or 1 in our paper). The algorithm in pseudocode is the following:

Algorithm 2: Genetic Algorithm

```
t = 0;
generate P(t);
evaluate P(t);
while not stopping condition do
    t = t + 1;
    select P(t) from P(t - 1);
    cross-over P(t);
    mutate P(t);
    evaluate P(t);
end
```

Function $P(t)$ returns the population of candidate solutions at generation t. Selection operator is built for creating more and more fitted populations from a generation to another. It is a factor that induces either exploring of better solution or exploiting the best known so far from the population. This happens because it takes into account the fitness function of the population that I will describe below. We defined the fitness function as:

$$fitness(candidate) = 1.1 \cdot max(population) - evaluate(candidate).$$

The constant 1.1 is defined here as a possibility for worse candidate to reach a better solution; function max returns the maximum value of the optimization function across the population and *evaluate* returns the value of the function of the current candidate solution.
The selection that we chose for our work is roulette wheel selection. For this type of selection, it is associated a probability to each chromosome in the population to be chosen for the next generation, the better the fitness, the more probable to be in the next generation. This is computed as the following:

$$probability(i) = \frac{fitness(i)}{\sum_{j=1}^{N} fitness(j)},$$

where i denotes an individual in the population and N is the number of chromosomes in the population.
Other types of selection are: rank selection, tournament selection, elitism selection. For rank selection, it is prevented a premature convergence of the algorithm. The chromosomes are ordered by their fitness value and the probability

of selection is proportional to their rank. The pressure of selection is low if the fitness is a high value and it is high if the fitness is a low value. Tournament selection chooses randomly a set of k chromosomes and out of these only the top j are selected for survival. This repeats until we get the desired numbers of individuals. The elitism selection is based on retaining the most k fitted chromosomes for each generation.
In a more formal sense, the roulette wheel selection is the following:

Algorithm 3: Roulette wheel selection

EVALUATE P **for** $i = 1$ *to* $popSize$
  $eval[i] = fitness(P[i])$;
TOTAL FITNESS **for** $i = 1$ *to* $popSize$
  $T+ = eval[i]$;
INDIVIDUAL SEL. PROB. **for** $i = 1$ *to* $popSize$
  $p[i] = eval[i]/T$;
CUMULATIVE SEL. PROB. $q[0] = 0$;
for $i = 1$ *to* $popSize$
  $q[i] = q[i-1] + p[i]$;
SELECTION **for** $i = 1$ *to* $popSize$
  uniformly generate r in $(0, 1]$;
  select for survival chromosome j for which $q[j] \leq r < q[j+1]$;

In order to keep the population size across generations, we also have to do a cross-over over the selected candidates. Each chromosome has a probability to be a parent for the next generation chosen apriori. And for each parents pair, we randomly select a cutting point for cross-over. Let's say the the cutting point of two parents is c which can be an integer value in the interval $[1, n-2]$, where n is the length of the bitstring representation. Then the two offsprings are: $offspring_1 = append(parent1[0 : c], parent2[c : n])$ and $offspring_2 = append(parent2[0 : c], parent1[c : n])$, where the operator $[a : b]$ is an indexing operator from index a to b-1 inclusive and *append* appends the elements from the lists given as parameters.
Mutation is an operator which drives the current candidate to other possible better chromosomes. It is an operator which drives the solution to an explorative way. The mutation in a genetic algorithm consists of changing a gene into the chromosome. In our paper, each chromosome has its own probability to suffer a mutation and also there is 1% chance for a gene to be mutated.

2.2.3 Hybrid Hill Climbing

Our research is based on an algorithm which combines Hill Climbing and a genetic algorithm. In this sense, we run a single iteration in Hill Climbing but instead of having a randomly created candidate solution, we use as starting point the result of the genetic algorithm after a number of generations. In this way, the algorithm will search locally in the genetic algorithm's result neighbourhood.

2.3 Testing function

2.3.1 Griewangk function

The Griewangk function has the following form:

$$f(x) = \sum_{i=1}^{n} \frac{{x_i}^2}{4000} - \prod_{i=1}^{n} cos\left(\frac{x_i}{\sqrt{i}}\right) + 1, where - 600 \leq x_i \leq 600.$$

It's global minimum optima is: $f(x) = 0, x(i) = 0, i = 1 : n$. When we look into the entire interval of the function, we can suspect that it is easy for an algorithm to find the global optima, but if we would zoom in on a smaller interval, we assume that the algorithm can reach into a local optima.

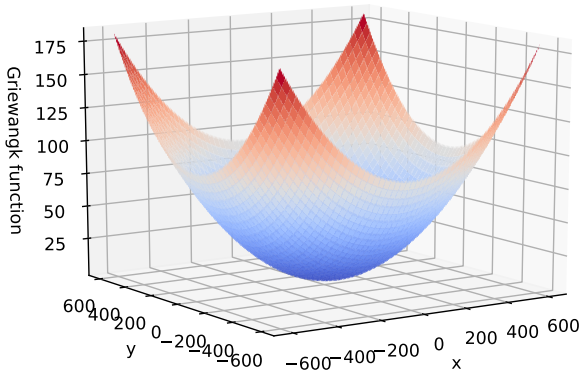

Figure 1: Griewangk function on its entire definition interval

2.3.2 Rastrigin function

Rastrigin function has the following definition:

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

having the global optima of: $f(x) = 0, x(i) = 0, i = 1 : n$.

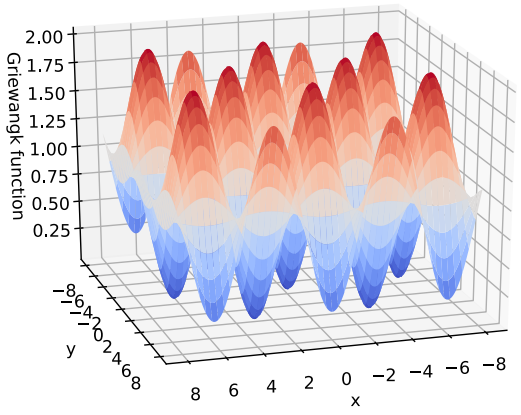

Figure 2: Griewangk function only in the interval [-8,8]

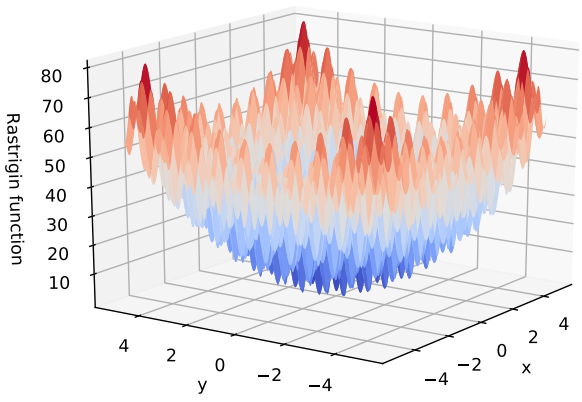

Figure 3: Rastrigin function on its entire definition interval

2.3.3 Rosenbrock function

The Rosenbrock function is the following:

$$f(x) = \sum_{i=1}^{n-1} 100 \cdot \left(x_{i+1} - x_i^2\right)^2 + (1 - x_i)^2, -2.048 \leq x_i \leq 2.048$$

with the global optima $f(x) = 0, x(i) = 1, i = 1 : n$.

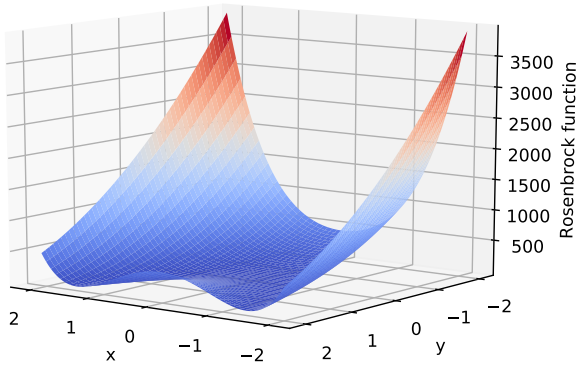

Figure 4: Rosenbrock function on its entire definition interval

2.3.4 Six-hump Camelback

The Six-hump Camelback has the following definition:

$$f(x) = \left(4 - 2.1 \cdot x_1^2 + \frac{x_1^4}{3}\right) \cdot x_1^2 + x_1 \cdot x_2 + \left(-4 + 4 \cdot x_2^2\right) \cdot x_2^2,$$
$$-3 \leq x_1 \leq 3, -2 \leq x_2 \leq 2$$

and its global optima is

$$f(x_1, x_2) = -1.0316, (x_1, x_2) = (-0.0898, 0.7126), (0.0898, -0.7126).$$

3 Conclusion

...

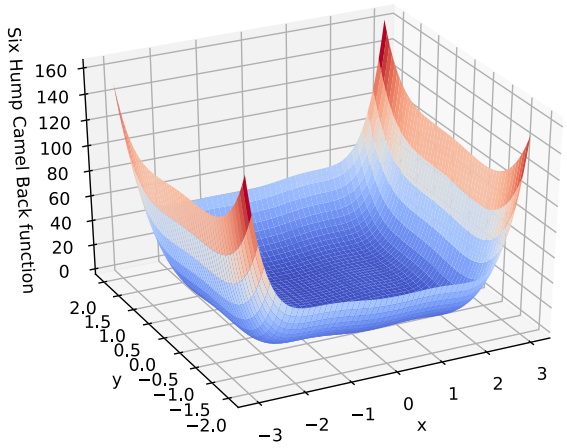

Figure 5: Six-hump Camelback function on its entire definition interval

References

[1] Melanie Mitchell, "An introduction to genetic algorithms", 1998
`https://www.academia.edu/12824545/An_Introduction_to_Genetic_Algorithms_-_Melanie_Mitchell`

[2] Breaban Mihaela, Nature Inspired Methods class course, 2020
`https://profs.info.uaic.ro/~pmihaela/MOC/trajectory.html`

[3] Wikipedia site for Selection, Mutation and Crossover
`https://en.wikipedia.org/wiki/Selection_(genetic_algorithm)`
`https://en.wikipedia.org/wiki/Mutation_(genetic_algorithm)`
`https://en.wikipedia.org/wiki/Crossover_(genetic_algorithm`

[4] MIT Open Course, "A Basic Introduction to Genetic Algorithms", Prof. Olivier de Weck, 2010
`https://ocw.mit.edu/courses/institute-for-data-systems-and-society/ids-338j-multidisciplinary-system-design-optimization-spring-2010/lecture-notes/MITESD_77S10_lec11.pdf`