T0 - Numerical optimization using deterministic and nondeterministic algorithms

Oloieri Alexandru

October 21, 2019

Abstract

This paper analyzes and compares the behaviour and performance of two algorithms when dealing with a complex problem: finding the global minima/maxima of a mathematical function.

1 Introduction

In computer science, there are some problems which require a lot of resources (time, memory) in order to be solved (usually these are called hard problems), and, unfortunately, classical algorithms are too weak for them. This made scientists to research other types of algorithms, and that's how heuristic algorithms were invented. In computer science, artificial intelligence, and mathematical optimization, a heuristic is a technique designed for solving a problem more quickly when classic methods are too slow, or for finding an approximate solution when classic methods fail to find any exact solution.

A well known hard problem is finding the global minimum/maximum of a mathematical function which is defined on an interval. That's because a computer cannot execute an algorithm that evaluates the function in each point from the function domain, and then chooses the best point (because there are an infinity of points).

In this experiment we'll see how good are two different algorithms (a nondeterministic one and a heuristic algorithm) behaving in practice, when dealing with this problem, more exactly, when trying to find the global minimum of some functions with more than one dimension (which have arguments of type $x = (x_1, x_2, ..., x_n)$).

2 Algorithms

2.1 Brute force search

The deterministic algorithm we'll use for this experiment is a brute force search algorithm: for each function dimension we will choose k points in the

function domain (let s_i be a set with k elements corresponding to i^{th} dimension), and evaluate the function for each $x = (x_1, x_2, ..., x_n), x_i \in s_i$ (we generate the cartesian product between the obtained sets). If a function has n dimensions, we'll get k^n results (not necessarily different), and we will return the best of them.

2.2 Hill climbing

The nondeterministic algorithm we'll use for this experiment is a naive hill climbing: we will generate a random point in the function domain, choose a step (let's call it ϵ) and then repeat the following steps: let $x = (x_1, x_2, ..., x_n)$ be the current point and $best = f(x)$, then evaluate the function in the following points: $x_1 = (x_1+\epsilon, x_2, ..., x_n), x_2 = (x_1-\epsilon, x_2, ..., x_n), x_3 = (x_1, x_2+\epsilon, ..., x_n), ..., x_{n\cdot 2} = (x_1, x_2, ..., x_n - \epsilon)$, let $currentIterationBest = f(x_k), x_k \in 1, 2, ..., 2 \cdot n$ the best value when evaluating the function for all those points; if $currentIterationBest$ is worse than $best$, or is equal to $best$, then stop the algorithm, otherwise $x = x_k$.

3 Functions

Rastrigin's function:

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

Schwefel function:

$$f(x) = 418.9829 \cdot n - \sum_{i=1}^{n} \left[x_i \cdot sin(\sqrt{|x_i|})\right], x_i \in [-500.0, 500.0]$$

Sphere function:

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

Zakharov function:

$$f(x) = \sum_{i=1}^{n} x_i^2 + (\sum_{i=1}^{n} 0.5 \cdot i \cdot x_i)^2 + (\sum_{i=1}^{n} 0.5 \cdot i \cdot x_i)^4, x_i \in [-5.0, 10.0]$$

These functions were chosen with a specific reason: they have different types of graphs.

The Rastrigin and the Schwefel functions are non-convex, multimodal functions that have a lot of local minima, which should cause the Hill Climbing algorithm to get stuck in such a local minima, and so it won't always find the global maxima. The other two functions, on the other hand, are convex and unimodal functions, and they have no local minimum, except the global one, and this should be the perfect situation for the Hill Climbing algorithm.

We cannot really predict the behaviour of the brute force algorithm: if we do too little iterations (this implies that the step is too big), there exists a chance we may miss all the points where the function reaches a value close to global maxima.

4 Experiment

We'll run both algorithms for all 4 functions, and for each function, we'll set, at a time, 2, 5, 10 and 20 dimensions.

4.1 Brute force search

Since the brute force algorithm described at 2.1 is deterministic, for fixed n and k we will always get the same result, so it doesn't make sense to run the algorithm more than once for a combination of n and k.

4.2 Hill climbing

For hill climbing we will proceed different: let's call an "iteration" running the algorithm described at 2.2 once. A run will have 100 iterations, and for a function with fixed number of dimensions we will have 30 runs (so we will get 3000 results). Most of the times we'll use $\epsilon = 0.01$, but when the function is very complex and has a large number of dimensions, we'll use $\epsilon = 0.1$.

5 Results

5.1 Brute force search

For larger number of dimensions we had to choose a small value for k, as the algorithm is exponential and it takes a lot of time to run it.

dim	k	$f(x)$	time(s)
2	100	0.0	0.0
5	50	0.0	27.14
5	55	5.81	36.85
10	5	11.62	7.88
20	3	312.03	1631.19

Figure 1: Rastrigin's function analysis (brute force search)

dim	k	$f(x)$	time(s)
2	100	0.24	0.01
5	40	10.26	9.23
5	45	60.77	37.09
10	6	2329.37	36.32
20	2	4767.87	891.55

Figure 2: Schwefel function analysis (brute force search)

dim	k	$f(x)$	time(s)
2	100	0.0	0.01
5	45	0.25	3.60
5	50	0.0	4.18
10	7	20.40	6.44
20	2	0.0	97.76

Figure 3: Sphere function analysis (brute force search)

dim	k	$f(x)$	time(s)
2	100	0.01	0.01
5	30	0.0	0.60
5	50	0.08	5.60
10	6	0.0	7.42
20	2	200.0	161.06

Figure 4: Zakharov function analysis (brute force search)

5.2 Hill Climbing

dim	ϵ	best	worst	mean	stDev	time(s)
2	0.01	0.0	0.99	0.23	0.42	0.24
5	0.01	1.99	13.93	7.86	3.26	0.57
10	0.01	18.92	47.77	33.74	6.47	2.13
20	0.01	62.71	116.44	93.29	13.61	15.24

Figure 5: Rastrigin's function analysis (hill climbing)

dim	ϵ	best	worst	mean	stDev	time(s)
2	0.01	0.0	0.0	0.0	0.0	0.24
5	0.01	0.0	434.27	219.11	105.24	53.21
10	0.01	572.45	1185.92	886.36	168.22	433.67
20	0.1	1679.41	2983.97	2455.66	265.73	327.50

Figure 6: Schwefel function analysis (hill climbing)

dim	ϵ	best	worst	mean	stDev	time(s)
2	0.01	0.0	0.0	0.0	0.0	0.35
5	0.01	0.0	0.0	0.0	0.0	1.40
10	0.01	0.0	0.0	0.0	0.0	2.98
20	0.01	0.0	0.0	0.0	0.0	19.47

Figure 7: Sphere function analysis (hill climbing)

dim	ϵ	best	worst	mean	stDev	time(s)
2	0.01	0.0	0.0	0.0	0.0	0.40
5	0.01	0.0	0.0	0.0	0.0	1.46
10	0.01	0.0	0.0	0.01	0.0	7.42
20	0.01	0.16	0.23	0.21	0.01	52.58

Figure 8: Zakharov function analysis (hill climbing)

5.3 Interpretation

We can observe that the brute force algorithm found the global minima for some functions. This happened when k is even, and let's analyze why that happened. Let's take Rastrigin's function: it is defined on interval $[-5.12, 5.12]$, and if we divide it in k parts (and k is even), one of the elements from the cartesian product will be $x_{best} = (0, 0, ..., 0)$, and $f(x_{best}) = 0$. So even if we

reached the global minimum in some cases, this doesn't mean that the algorithm is good, it means only that way we chose to iterate through function domain was favorable for this function, as for other functions this doesn't happen very often: Zakharov function is defined on interval $[-5.0, 10.0]$, and we don't always reach the global minima when k is even.

The hill climbing algorithm performed very good for all functions and all number of dimensions. For the two convex functions, it found the global minima almost at every run, with the exception of Zakharov function with 20 dimensions, and than's only because ϵ was too large. For the other two functions, for 10 and 20 dimensions it didn't reach the global minimum (this means it got stuck in a local minima), but even so it had better results than the brute force algorithm for almost any case, while also having a very good running time.

For better results, for the brute force algorithm we can increase the value of k (but the time will grow exponentially), and for hill climbing algorithm we can use a lower value for ϵ, or we can increase the number of runs.

6 Conclusions

Even though it had good results for some functions, the brute force algorithm is not a reliable algorithm for finding the global minima/maxima of a function, as the programs take a very long time to run, and the algorithm behaviour cannot be anticipated: if the parameters are optimal, we can get a value very close to the global minimum/maximum, but if not, we just waste a lot of time and resources.

The hill climbing algorithm, on the other hand, produced optimal results for convex functions (as expected), and good results for the other two functions, while also having a low running time, compared to the brute force algorithm (especially the number of dimensions was 20), so we can afirm that in practice it's a good idea to use it, as it has a big advantage over the brute force algorithm: we try to reach global minima/maxima using a strategy.

References

[1] More about heuristics in computer science: `https://en.wikipedia.org/wiki/Heuristic_(computer_science)`

[2] Information about functions used in this experiment: `https://www.sfu.ca/~ssurjano/optimization.html`

[3] The github repository of the application used in this experiment: `https://github.com/OloieriAlexandru/Numerical-Optimization-Platform/tree/T0`