Meta-heuristic methods in global optimization

Bita Alexandru

October 29, 2019

1 Abstract

Global optimization is a field of applied mathematics that deals with finding the extremal value of a function in a domain of definition, subject to various constraints on the variable values.

A global optimization problem with continuous variables may contain several local optima or stationary points. The problem of designing algorithms that obtain global solutions is very difficult when there is no overriding structure that indicates whether a local solution is indeed the global solution.

The ultimate goal of global optimization techniques is to develop a single method that:

- works for a large class of problems,
- finds the global optima with an absolute guarantee,
- uses very little computation.

2 Introduction

In general, classical optimization techniques struggle when dealing with global optimization problems. One of the main reasons why this happens is because they can easily get stuck in local minima.

The interaction between computer science and optimization has yielded new practical solvers for global optimization problems, called meta-heuristics. These are mainly based on simulating nature and artificial intelligence tools. Meta-heuristics mainly invoke exploration and exploitation search procedures in order to diversify the search all over the search space and intensify the search in some promising areas. Therefore, metaheuristics can't easily get stuck in local minima. However, metaheuristics are computationally costly due to their slow convergence. One of the main reasons for their slow convergence is that they may fail to detect promising search directions especially in the vicinity of local minima due to their random constructions.

Combining meta-heuristics with local search methods is a practical remedy to overcome the drawbacks of slow convergence and random constructions of meta-heuristics.

In these hybrid methods, local search strategies are inlaid inside metaheuristics in order to guide them especially in the vicinity of local minima, and overcome their slow convergence especially in the final stage of the search. As a result, the purpose of the present study is therefore to ascertain the effectiveness of using such kind of method as compared to a rather classical technique.

3 Methodology

This study compares the results given by a *Simulated Annealing* implementation and a *Hill Climbing* implementation while attempting to minimize a number of functions.

3.1 Simulated annealing

Simulated annealing is a probabilistic technique for approximating the global optimum of a given function. Specifically, it is a metaheuristic to approximate global optimization in a large search space for an optimization problem.

The name and inspiration come from annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects.

This notion of slow cooling implemented in the simulated annealing algorithm is interpreted as a slow decrease in the probability of accepting worse solutions as the solution space is explored. Accepting worse solutions is a fundamental property of metaheuristics because it allows for a more extensive search for the global optimal solution. [**?**]

3.2 Hill Climbing

In numerical analysis, *hill climbing* is a mathematical optimization technique which belongs to the family of local search. It is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by making an incremental change to the solution. If the change produces a better solution, another incremental change is made to the new solution, and so on until no further improvements can be found. [**?**]

There are more types of *Hill climbing* but for this study I have used the following two:

1. **Next-Ascent Hill climbing**: It examines the neighboring nodes one by one and selects the first neighboring node which optimizes the current cost as next node.

2. **Steepest-Ascent Hill climbing**: It first examines all the neighboring nodes and then selects the node closest to the solution state as of next node environment.

3.3 Implementation

The pseudocode for *Simulated Annealing* as follows:

```
begin
  randomize(x[])
  initialize(T)
  while T > CUT_OFF
    foreach x in x[]
      n=random(neighbourhood(x))
      if eval(n) > eval(x) then x=n
      else if random[0,1) < exp(-|eval(n)-eval(x)|/T) then x=n
    T *= COOLING_RATIO
  best_result=eval(x[])
end
```

The pseudocode for *Hill Climbing* as follows:

```
begin
  for i=0 to MAX_ITTR
  randomize(x[])
    foreach x in x[]
      local=false
      do
        n=better(neighbour(x)) // for NACH
        //  n=better(neighbourhood(x)) for SAHC
        if eval(n) > eval(x) then x=n
        else local=true
      until local
    if eval(x[]) > best_result then best_result=eval(x[])
end
```

More on notations and implementation:

1. all values are represented in binary; the number of bytes used is given by the inequation $2^n - 1 \geq (b-a)/\varepsilon$, where **n** is the number we need, ε is the precision we use and **[a,b]** is the interval we use;

2. neighbours for each value are generated when the program is first started;

3. at the beginning of every iteration, a random array of solutions is being generated;

4. operations on each solution and its neighbours are made according to the algorithm;

5. the algorithm ends either when the limit of iterations is reached(HC) or the temperature gets below a threshhold(SA);

- **randomize(array)** is a function that takes an array as input and generates for each of its elements a random value according to the problem's constraints;
- the **k**th **neighbour** of a number is a value at distance **+-k**ε;
- **neighbour(real)** is a function that returns the **neighbour** that is better evaluated than the value specified in the input;
- **neighbourhood(real)** is a function that returns the **neighbour** that has the best evaluation among all of the input's neighbours;

4 Results

Simulated Annealing has been configured with the following parameters:

```
Epsilon = 10^-4
Iterations = 100
Neighbours = 100
Temperature = 1000
Cutoff = Epsilon
Cooling ratio = 0.9
```

Hill Climbing has been configured with the following parameters:

```
Epsilon = 10^-4
Iterations = 100
Neighbours = 50
```

Both algorithms have been ran 30 times in order to draw a more realistic conclusion.

Below are being shown the results gathered upon executing both algorithms on 4 different functions.

	bays29 (2020)				**brazil58 (25395)**				**brg180 (1950)**				**pa56**
	Best	*Avg*	*Err*	*Time*	*Best*	*Avg*	*Err*	*Time*	*Best*	*Avg*	*Err*	*Time*	*Best*
SA	2020	2032.63	0.63	0.43	25395	25703.63	1.22	1.12	2140	2300.33	17.97	6.10	3049
GA													

4.1 Sphere

$f(\mathbf{x}) = f(x_1, x_2, ..., x_n) = \sum_{i=1}^{n} x_i^2, x_i \in [-5.12, 5.12]$

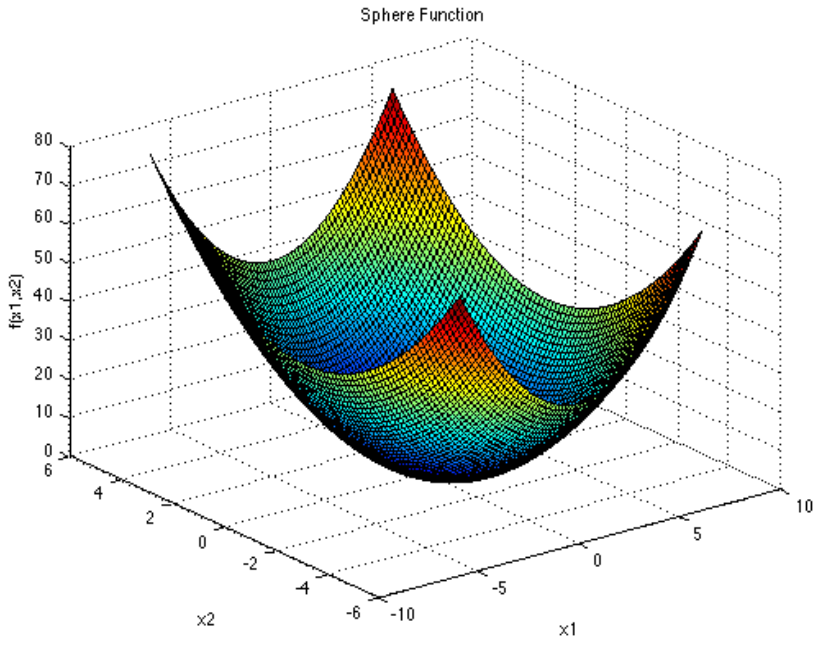

Figure 1: Sphere function [**?**] for 2 dimensions (x_1 and x_2).

4.2 Sum squares

$f(\mathbf{x}) = f(x_1,...,x_n) = \sum_{i=1}^{n} ix_i^2, x_i \in [-5.12, 5.12]$

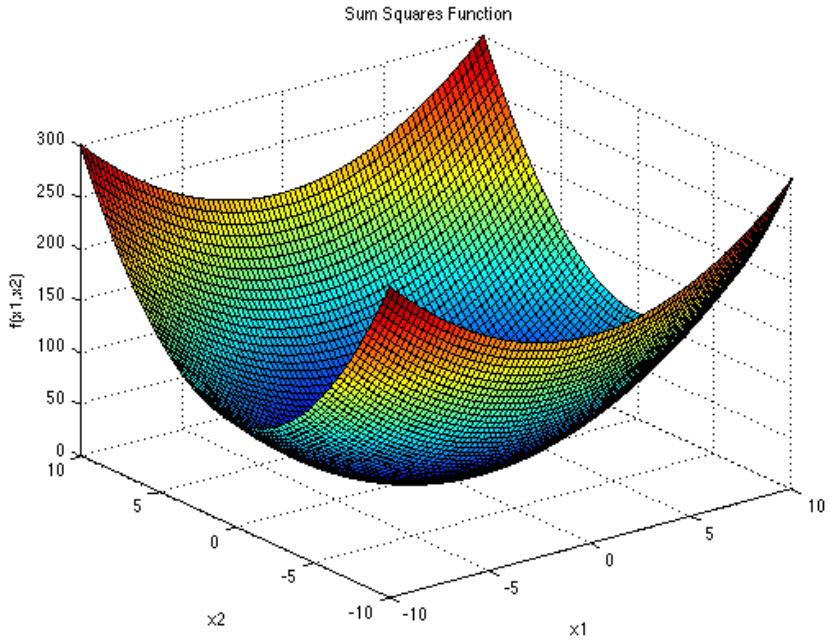

Figure 2: Sphere function [**?**] for 2 dimensions (x_1 and x_2).

Dimensions	**Next-Ascent Hill Climbing**				
	Best Val.	*Avg. Val.*	*Worst Val.*	*Std. Dev.*	*Avg. Time*
2	0.0000	0.0000	0.0000	0.0000	26.6827
5	0.0000	0.0000	0.0000	0.0000	56.1370
20	0.0000	0.0000	0.0000	0.0000	194.3200
	Steepest-Ascent Hill Climbing				
	Best Val.	*Avg. Val.*	*Worst Val.*	*Std. Dev.*	*Avg. Time*
2	0.0000	0.0000	0.0000	0.0000	3.4444
5	0.0000	0.0000	0.0000	0.0000	8.2531
20	0.0000	0.0000	0.0000	0.0000	33.3749
	Simulated Annealing				
	Best Val.	Avg. Val.	Worst Val.	Std. Dev.	Avg. Time
2	0.0000	0.0001	0.0005	0.0001	0.1594
5	0.0000	0.0003	0.0006	0.0002	0.5078
20	0.0006	0.0011	0.0021	0.0004	6.7616

Table 1: Comparing results obtained by NAHC, SAHC and SA on Sum Squares function.

4.3 Moved axis parallel hyper-ellipsoid

$f(\mathbf{x}) = f(x_1, ..., x_n) = \sum_{i=1}^{n} 5ix_i^2, x_i \in [-5.12, 5.12]$

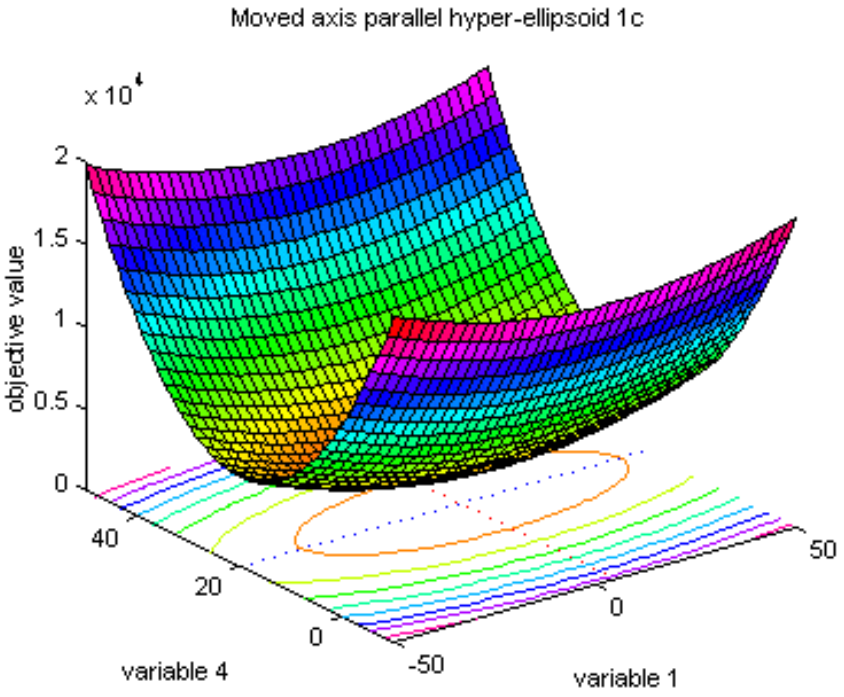

Figure 3: Sphere function [**?**] for 2 dimensions (x_1 and x_2).

Dimensions	**Next-Ascent Hill Climbing**				
	Best Val.	*Avg. Val.*	*Worst Val.*	*Std. Dev.*	*Avg. Time*
2	0.0000	0.0000	0.0000	0.0000	21.5001
5	0.0000	0.0000	0.0000	0.0000	77.4399
20	0.0000	0.0000	0.0000	0.0000	192.2441
	Steepest-Ascent Hill Climbing				
	Best Val.	*Avg. Val.*	*Worst Val.*	*Std. Dev.*	*Avg. Time*
2	0.0000	0.0000	0.0000	0.0000	3.4699
5	0.0000	0.0000	0.0000	0.0000	8.5806
20	0.0000	0.0000	0.0000	0.0000	33.0174
	Simulated Annealing				
	Best Val.	Avg. Val.	Worst Val.	Std. Dev.	Avg. Time
2	0.0000	0.0001	0.0004	0.0001	0.2491
5	0.0001	0.0003	0.0008	0.0002	0.7300
20	0.0004	0.0011	0.0020	0.0004	7.6733

Table 2: Comparing results obtained by NAHC, SAHC and SA on Moved axis parallel hyper-ellipsoid function.

4.4 Rastrigin

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

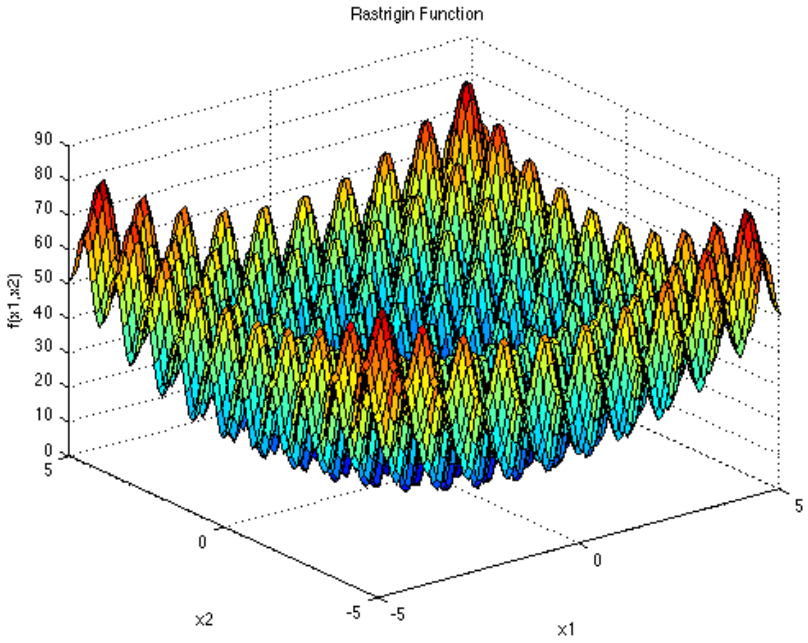

Figure 4: Sphere function [**?**] for 2 dimensions (x_1 and x_2).

Dimensions	**Next-Ascent Hill Climbing**				
	Best Val.	*Avg. Val.*	*Worst Val.*	*Std. Dev.*	*Avg. Time*
2	0.0000	0.3648	1.9900	0.5440	2.2493
5	0.9950	6.9651	10.9452	3.1149	7.8206
20	66.6660	91.6739	115.4214	12.1123	22.1756
	Steepest-Ascent Hill Climbing				
	Best Val.	*Avg. Val.*	*Worst Val.*	*Std. Dev.*	*Avg. Time*
2	0.0000	0.4312	0.9950	0.4931	0.4818
5	1.9900	6.7993	15.9202	2.7561	1.0484
20	74.6259	90.7452	106.4665	9.8715	4.0848
	Simulated Annealing				
	Best Val.	Avg. Val.	Worst Val.	Std. Dev.	Avg. Time
2	0.0000	16.4509	40.7954	12.4561	0.2295
5	20.8953	43.9132	103.4808	21.4173	0.7120
20	80.5965	162.7843	241.7878	33.4907	6.1156

Table 3: Comparing results obtained by NAHC, SAHC and SA on Rastrigin's function.

5 Discussion

First of all, it should be stated that all three algorithms give pretty good results on each function (knowing that the global minima for each is 0).

While *SAHC* is only slightly faster than *NAHC*, they both have quite the same results which are actually the right ones for all the functions excluding Rastrigin's.

The error on the latter is caused by the existence of multiple local minimas as opposed to the first three functions which only have one (and it is global).

Compared to both implementations of *Hill Climbing*, *Simulated Annealing* is observed to be a little off on the results but in exchange for that it is **crazy fast**. And most of the time, this is what we want: good enough results that we can get in a short period of time.

6 Conclusion

Combining meta-heuristics with local search methods is a practical remedy to overcome the drawbacks of slow convergence and random constructions of meta-heuristics. In these hybrid methods, local search strategies are inlaid inside metaheuristics in order to guide them especially in the vicinity of local minima, and overcome their slow convergence especially in the final stage of the search.

The purpose of the present study was therefore to ascertain the effectiveness of using such kind of method as compared to a rather classical technique and according to the tables shown earlier, we can clearly state that while the latter tend to give more precise results, they are exponentially slower than the former.

References

[1] Simulated annealing
`https://en.wikipedia.org/wiki/Simulated_annealing`

[2] Hill climbing
`https://en.wikipedia.org/wiki/Hill_climbing`

[3] Sphere function
`https://www.sfu.ca/~ssurjano/spheref.html`

[4] Sum squares function
`https://www.sfu.ca/~ssurjano/sumsqu.html`

[5] Moved axis parallel hyper-ellipsoid function
`http://www.geatbx.com/docu/fcnindex-01.html#P119_4694`

[6] Rastrigin's function `https://www.sfu.ca/~ssurjano/rastr.html`