A Performance Test for Hill Climbing & Simulated Annealing using Artificial Landscapes

Florin Eugen Rotaru

October 2022

Abstract

We implemented two mathematical optimization algorithms - Hill Climbing and Simulated Annealing - which attempt to find the extremum of a given function. The first one is designed for local search of the optimum, while the second one approximates the global optimum of the function. The implementations have been tested on 4 different functions, on 5, 10 and 30 dimensions. The results obtained, which are discussed below, show that generally H.C performs with a better accuracy than Simulated Annealing. Even so, in case the dimension is very large (such as 30), the runtime increases and the accuracy decreases, making the use of S.A preferable, especially considering its fast runtime.

1 Introduction

Hill Climbing Algorithm starts with an arbitrary initial solution and follows the strategy of comparing a neighbor with the current state. If the neighbor is heuristically better, it will accept the neighbor-state. The general structure of the algorithm is the following:

HillClimbing()
solution $\leftarrow$ *randomSolution*()
local $\leftarrow$ *false*
repeat
  neighbor $\leftarrow$ *improvement*()
  if *neighbor* is better than *solution* **then**
    solution $\leftarrow$ *neighbor*
  else
    local $\leftarrow$ *true*
  end if
until *local*

All the solutions are internally represented as bitstrings that store the function arguments. Deciding which solution is better implies converting the bits to Real $\mathbb{R}$ numbers and evaluating the function value. The neighbors are sets of arguments in the immediate proximity of our solution. (we will see later how they are generated).

improvement() may work in the following ways:

- Returns ***The Best Improvement***, i.e., the optimal neighbor.
- Returns ***The First Improvement***, i.e., the first neighbor that is better than the current solution, chosen from a sequence of randomly chosen neighbors.
- Returns ***The Worst Improvement***, i.e., a neighbor N_k better than the solution, such as $\forall i, N_i \in NEIGHBORS(solution) \Rightarrow$ if N_i is better than the solution, then N_i is better or equal to N_k

In case such improvements are not found, the initial solution is returned.

As we have previously mentioned, Hill Climbing is used for local search, hence we will iterate (10000 times) the above procedure in order to get the best local result. A single iteratedHillClimbing() execution represents **10000 of HillClimbing()** as defined above.

1.1 Simulated Annealing

To find a good solution we move from a solution to one of its neighbors :
If the cost decreases then the solution is changed and the move is accepted.
Otherwise, the move is accepted only with a probability depending on the cost increase and a control parameter called temperature.

SimulatedAnnealing()[1]
$T \leftarrow initialTemperature()$
$solution \leftarrow randomSolution()$
repeat
  repeat
    $neighbor \leftarrow randomNeighbor()$
    if *neighbor* is better than *solution* **then**
      $solution \leftarrow neighbor$
    else
      if $random[0,1) < e^{\frac{|val(neighbor)-val(solution)|}{T}}$ **then**
        $solution \leftarrow neighbor$
      end if
    end if
  until $local - condition$
  $T \leftarrow lowerTemperature(args)$
until $(T < \varepsilon)$

The $local - condition$ is true when the internal loop has iterated for a *number* of time, such as 1000.

The $T < \epsilon$ condition ends the outer loop when the temperature reaches a pre-defined minimum (in our case, this will be 0.0005)

There are many ways to calculate the *initialTemperature()*. The initial temperature value must be defined in such a way that almost any perturbation must be accepted during the first iteration of the cooling loop. [3]
The method used by us is the following ([2]):

Temperature is obtained using the formula:

$$T = -\frac{\Delta\overline{E}}{\log(\chi_0)}$$

where $\Delta\overline{E}$ is an estimation of the cost increase of positive transitions. This estimation is again obtained by randomly generating some transitions. Note that ΔE is the cost increase induced by a transition t, and χ_0 is the probability that the transition be accepted. The probability to accept bad moves is high at the beginning (around 0.8-0.9) to allow the algorithm to escape from local minimum. In our implementation **we used**

$$\chi_0 = 0.8,$$

for $\Delta\overline{E}$ we generated 5 consecutive transitions using *First Improvement* technique.

The cooling schedule *lowerTemperature(args)* also differs from implementation to implementation. Here we used the *geometric cooling schedule* [3]:

$$T_k = T \cdot \alpha^k$$

with α usually being 0.8-0.99 (**we used** 0.8).
The programs have been tested using the functions Rastrigin's, De Jong's 1, Michalewicz's, Schwefel's on 5-10-30 dimensions. The results are presented in the form of tables, followed by further discussions.

2 Our Method & Implementation

In order to implement the algorithms, we used C++ and some OOP features. Our implementations rely on two main classes: class *Problem* and class *Solution*
Here we give the most significant class members, we ignore setters, getters or functions that deal with the internal representation of solutions.

class Problem;

```
class Problem
{
 //...etc
        Solution current_solution;
        Solution final_solution;
        float (*function_used)();
        void GenerateSolution();  //initializes current_solution
        void GenerateAllNeighbors(); //sets the neighbors of current_solution
        Solution BestImprovement(); //return best improv neighbor
        Solution FirstImprovement(); //return first improv neighbor
        Solution WorstImprovement(); //return worst improv neighbor
        void HillClimbing(short improv); //modifies the current_solution
        void IteratedHillClimbing(short improv); //modifies the final_solution

        void SimulatedAnnealing();
 //etc...
};
```

GenerateAllNeighbors() computes all of the neighbors by sequentially negating every bit, that is, a solution with the bitset of length l will have l neighbors. [1]

class Solution:

An instance of *Solution* designates a point, its internal and external representations. It also stores all of its neighbors. Note that in other implementations the step of calculating the function values for all the neighbors is not necessary.

```
class Solution
{
typedef std::vector< std::vector<float> > float_matrix;
class Solution
{
public:
        boost::dynamic_bitset<> bits_representation;
        std::vector<float> solution_args;
        float solution_value;
        std::vector<Solution> all_neighbors; //stores all neighbors
        Solution operator=(const Solution rvalue);

};
```

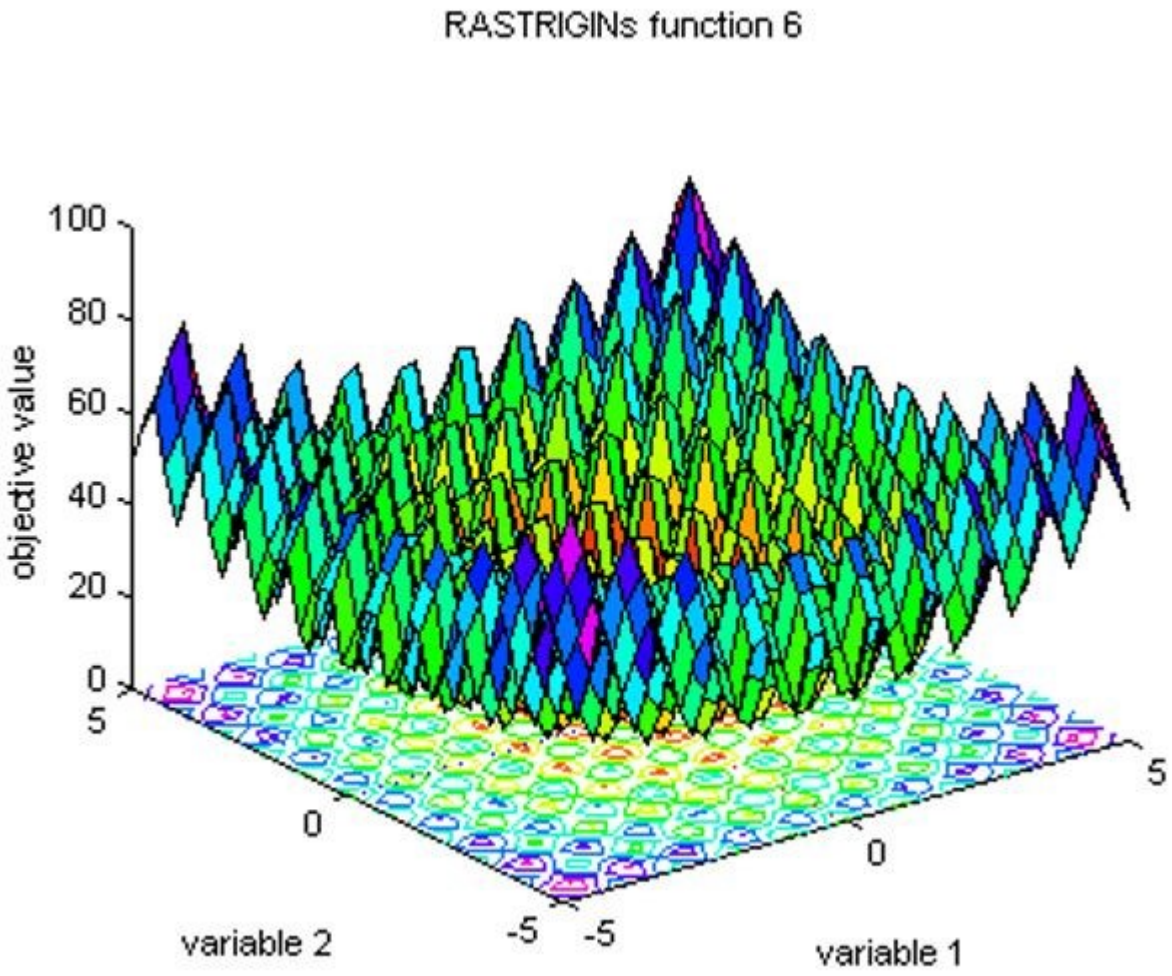
IteratedHillClimber() iterates 10000 times and executes $HillClimbing(improv)$, each time comparing the $final\ solution$ with the *current solution*, saving the better one.

3 Experimental Sample & Results

For the dimensions $n = 5$ and $n = 10$, the sample size we used is $s = 30$, meaning that for each variation of the algorithm, on those dimensions, 30 tests have been made.
For the dimension $n = 30$, we set a runtime upper bound of around $200min$.
For Simmulated Annealing the sample size is $s = 30$;

3.1 Rastrigin's

[6]



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

global minimum $f(\overline{x}) = 0, \overline{x} = 0$

$\mu(min), \sigma, \mu(t)$ *or* t on Dimension	n=5	n=10	n=30
First Improvement (H.C)	$\mu = 0.11014$ $\sigma = 0.30035$ $\mu(t) = 2.29417$	$\mu = 3.57407$ $\sigma = 0.30035$ $\mu(t) = 13.61$	$min = 31.396$ $t = 218.3$
Best Improvement (H.C)	$\mu = 3.8147 \cdot 10^{-7}$ $\sigma = 0.114441 \cdot 10^{-5}$ $\mu(t) = 1.875$	$\mu = 2.25937$ $\sigma = 0.66687$ $\mu(t) = 8.227$	$min = 21.0924$ $t = 228.3$
Worst Improvement (H.C)	$\mu = 1.94699$ $\sigma = 1.06031$ $\mu(t) = 9.033$	$\mu = 17.76$ $\sigma = 1.7279$ $\mu(t) = 14.690$	$min = 45.3924$ $t = 200$
Simmulated Annealing	$\mu = 5.8838$ $\sigma = 3.9918$ $\mu(t) = 0.2$	$\mu = 13.02$ $\sigma = 5.0639$ $\mu(t) = 0.45$	$\mu = 39.6733$ $\sigma = 4.871$ $\mu(t) = 3.03$

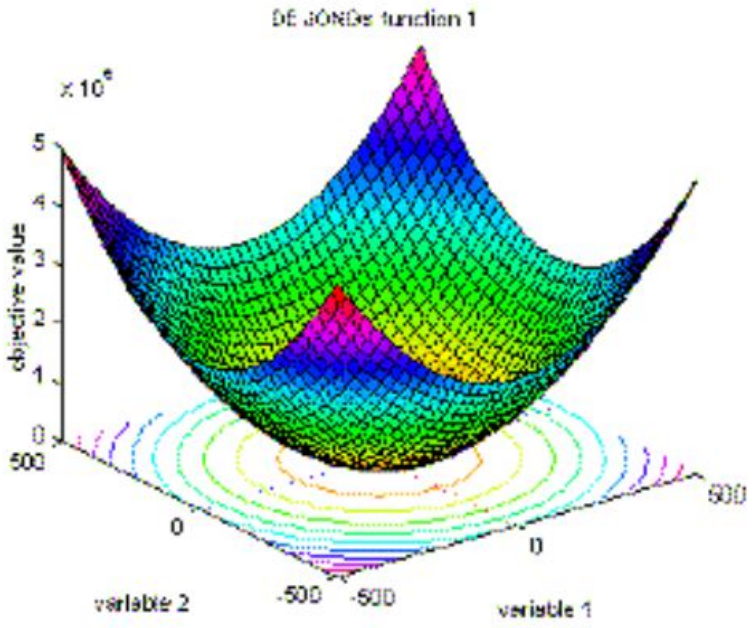
3.1.1 Best & Worst

Best & Worst on Dimension	n=5	n=10	n=30
First Improvement (H.C)	$Best = 0$ $Worst = 0.99496$	$Best = 1.98$ $Worst = 5.220$	
Best Improvement (H.C)	$Best = 0$ $Worst = 3.8147 \cdot 10^{-6}$	$Best = 0.9949$ $Worst = 3.984$	
Worst Improvement (H.C)	$Best = 0.9949$ $Worst = 3.99987$	$Best = 12.471$ $Worst = 21.3944$	
Simulated Annealing	$Best = 1.99292$ $Worst = 13.0801$	$Best = 5.46662$ $Worst = 22.424$	$Best = 33.54$ $Worst = 45.085$

Out of all H.C variations, the *Best Improvement* performs best, giving the best results in the shortest time. The second comes *First Improvement*, which was able to find the minimum, but fewer times and with a longer runtime (this is because $F.I$ randomly selects neighbors until it finds the first improvement.
The accuracy of *Worst Improvement* is far worse than the accuracy of the other two H.C's. It also goes through a lot more states and iterations, thus the longer runtime. Speaking of accuracy, Simulated Annealing performs overall similarly to W.I, but has a significat shorter runtime. The st.dev is also larger, meaning that the results are far away from each other(see *Best vs Worst*)

3.2 De Jong's 1

[6]



$f : [-5.12; 5.12] \to \mathbb{R}, f(\overline{x}) = \sum_{i=1}^{n} x_i^2$

global minimum $f(\overline{x}) = 0, \overline{x} = 0$

$\mu(min), \sigma, \mu(t)$ *or* t on Dim.	n=5	n=10	n=30
First Improvement (H.C)	$\mu = 1.08295 \cdot 10^{-10}$ $\sigma = 0$ $\mu(t) = 3.1966$	$\mu = 2.166 \cdot 10^{-10}$ $\sigma = 0$ $\mu(t) = 9.5$	min = $6.59{\cdot}10^{-10}$ $t = 221.067$
Best Improvement (H.C)	$\mu = 1.08295 \cdot 10^{-10}$ $\sigma = 0$ $\mu(t) = 2.641$	$\mu = 2.166 \cdot 10^{-10}$ $\sigma = 0$ $\mu(t) = 14.197$	min = $6.606 \cdot 10^{-10}$ $t = 164.45$
Worst Improvement (H.C)	$\mu = 1.08295 \cdot 10^{-10}$ $\sigma = 0$ $\mu(t) = 5.66$	$\mu = 2.16591 \cdot 10^{-10}$ $\sigma = 0$ $\mu(t) = 21.22$	min = $6.628{\cdot}10^{-10}$ $t = 200.017$
Simmulated Annealing	$\mu = 0.001695$ $\sigma = 0.000836$ $\mu(t) = 0.11$	$\mu = 0.00275697$ $\sigma = 0.0013362$ $\mu(t) = 0.305$	$\mu = 0.0071$ $\sigma = 0.00202$ $\mu(t) = 2.12$

3.2.1 Best & Worst

Best & Worst on Dim	n=5	n=10	n=30
First Improvement (H.C)	$Best = 1.08295 \cdot 10^{-10}$ $Worst = 1.08295 \cdot 10^{-10}$	$Best = 2.166 \cdot 10^{-10}$ $Worst = 2.166 \cdot 10^{-10}$	
Best Improvement (H.C)	$Best = 1.08295 \cdot 10^{-10}$ $Worst = 1.08295 \cdot 10^{-10}$	$Best = 2.166 \cdot 10^{-10}$ $Worst = 2.166 \cdot 10^{-10}$	
Worst Improvement (H.C)	$Best = 1.08295 \cdot 10^{-10}$ $Worst = 1.08295 \cdot 10^{-10}$	$Best = 2.16591 \cdot 10^{-10}$ $Worst = 2.16591 \cdot 10^{-10}$	
Simulated Annealing	$Best = 0.000458547$ $Worst = 0.0028639$	$Best = 0.001144$ $Worst = 0.004661$	$B = 0.00532$ $W = 0.00965$

Since De Jong's 1 function is unimodal, on the same dimension, all of the H.C variations return the same result (the st. dev. is zero), for the entire sample. We can only comment on the runtime of each of them:

for $n = 5$, *Best Improvement* seems to perform best

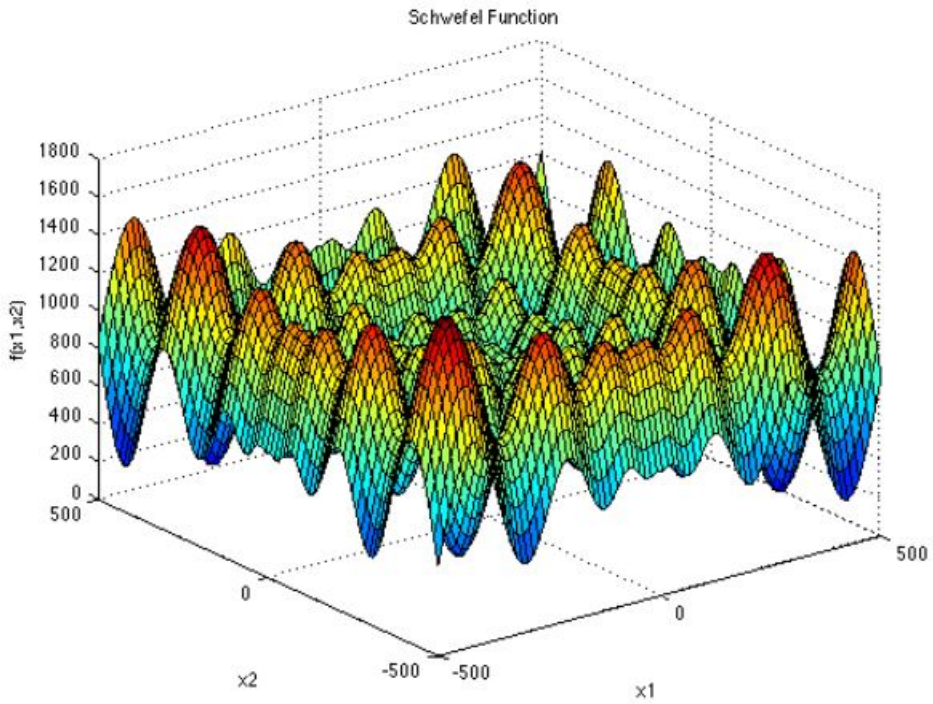
for $n = 10$, *First Improvement* seems to perform best, this is because as the function converges to the minimum, all of the neighbors are "better", so *First Improvement* is found in almost $O(1)$

for $n = 30$, again *Best Improvement* is better, as it travels through fewer transitions.

On the other hand, Simulated Annealing does not deliver the same results everytime, because it is still a heuristic algorithm. The runtime is considerably smaller, but the results are not that precise. Nevertheless, they are still close to the global minimum.

3.3 Schwefel's

[6]



$f : [-500; 500] \to \mathbb{R}, f(\overline{x}) = 418.9829n - \sum_{i=1}^{n} x_i \sin(\sqrt{|x_i|})$

global minimum $f(\overline{x}) = -n * 418.9829, \overline{x} = 418.9829$

$\{-2094.9145; -4189.829; -12569\}$

$\mu(min), \sigma, \mu(t)$ *or* t on Dimension	n=5	n=10	n=30
First Improvement (H.C)	$\mu = -2094.765$ $\sigma = 0.0496$ $\mu(t) = 4.213$	$\mu = -4030.648$ $\sigma = 27.6616$ $\mu(t) = 28.23$	min = -11153.1 $t = 200.067$
Best Improvement (H.C)	$\mu = -2094.91(*)$ $\sigma = 0$ $\mu(t) = 2.223$	$\mu = -4161.98$ $\sigma = 39.877$ $\mu(t) = 21.94$	min = -11476.8 $t = 183.7$
Worst Improvement (H.C)	$\mu = -2030.312$ $\sigma = 13.098$ $\mu(t) = 14.92998$	$\mu = -3974.765$ $\sigma = 26.885$ $\mu(t) = 91.14$	min = -10626.2 $t = 200.033$
Simmulated Annealing	$\mu = -1971.602$ $\sigma = 137$ $\mu(t) = 0.32$	$\mu = -3775.443$ $\sigma = 166.22$ $\mu(t) = 1$	$\mu = -11103.9$ $\sigma = 290.73$ $\mu(t) = 3.1366$

3.3.1 Best & Worst

Best & Worst on Dimension	n=5	n=10	n=30
First Improvement (H.C)	$Best = -2094.81$ $Worst = -2094.69$	$Best = -4067.38$ $Worst = -3983.07$	
Best Improvement (H.C)	$Best = -2094.91$ $Worst = -2094.91$	$Best = -4189.31$ $Worst = -4155.28$	
Worst Improvement (H.C)	$Best = -2041.04$ $Worst = -2014.27$	$Best = -4001.65$ $Worst = -3947.88$	
Simulated Annealing	$Best = -2094.81$ $Worst = -1695.39$	$Best = -3968.37$ $Worst = -3422.51$	$Best = -11470.7$ $Worst = -10580$

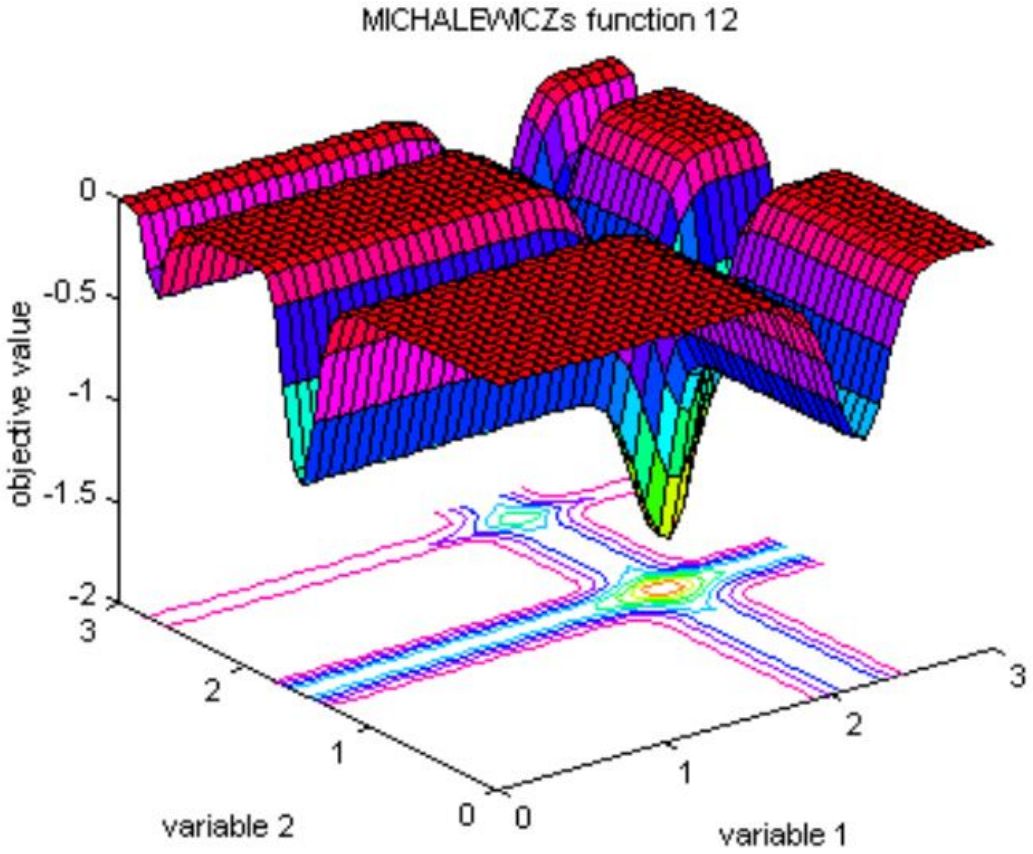
(*)One important thing to notice is that for $n = 5$, *Best Improvement* managed to calculate the global minimum with "almost" maximum precision for the entire sample (st. dev = 0), with a runtime that is also better than the other two.

For $n = 10$ and $n = 30$, *Best Imporvement* is still the closest the the minimum, but for the first one it has a worse runtime than *First Improvement*

Interestingly, on $n = 30$, Simulated Annealing's accuracy is closest to the one of *First Improvement*, in other cases, Simulated Annealing is worse than Hill Climber.

3.4 Michalewicz's

[6]



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

global minima
$f(\overline{x}) = -4.687, n = 5;$
$f(\overline{x}) = -9.66, n = 10;$

$\mu(min), \sigma, \mu(t)$ *or* t on Dimension	n=5	n=10	n=30
First Improvement (H.C)	$\mu = -3.698858$ $\sigma = 3.99 \cdot 10^{-11}$ $\mu(t) = 2.14$	$\mu = -8.45075$ $\sigma = 0.0365$ $\mu(t) = 9.526$	min = -25.7958 $t = 224.217$
Best Improvement (H.C)	$\mu = -3.69886$ $\sigma = 0$ $\mu(t) = 1.0722$	$\mu = -8.567$ $\sigma = 0.0331$ $\mu(t) = 6.75$	min = -26.4234 $t = 134.97$
Worst Improvement (H.C)	$\mu = -3.697$ $\sigma = 1.2132 \cdot 10^{-6}$ $\mu(t) = 3.793$	$\mu = -8.281$ $\sigma = 0.1083$ $\mu(t) = 29.53$	min = -24.0089 $t = 200.017$
Simmulated Annealing	$\mu = -3.62532$ $\sigma = 0.0886$ $\mu(t) = 0.09$	$\mu = -8.052692$ $\sigma = 0.2323$ $\mu(t) = 0.2$	$\mu = -25.788$ $\sigma = 0.3277$ $\mu(t) = 1.26$

3.4.1 Best & Worst

Best & Worst on Dimension	n=5	n=10	n=30
First Improvement (H.C)	$Best = -3.69886$ $Worst = -3.69885$	$Best = -8.47277$ $Worst = -8.37876$	
Best Improvement (H.C)	$Best = -3.69886$ $Worst = -3.69886$	$Best = -8.61445$ $Worst = -8.51105$	
Worst Improvement (H.C)	$Best = -3.69846$ $Worst = -3.69704$	$Best = -8.44006$ $Worst = -8.17259$	
Simulated Annealing	$Best = -3.69827$ $Worst = -3.49315$	$Best = -8.4651$ $Worst = -7.669$	$B = -26.0021$ $W = -25.1762$

As we have not calculated the actual minimum for Michalewicz's function for dimension $n = 30$, we give the Best and Worst solution with respect to the one computed by *Best Improvement H.C.*

For $n = 5$ we get similar results from all 4 algorithms, on Hill Climber, throughout the sample, the result obtained are very, very close to each other.

For $n = 10$ *Best Improvement* has the best runtime and is the most accurate

For $n = 30$ *Best Improvement* has the best runtime and probably is the most accurate too, Simulated Annealing being somewhat similar and, of course, a lot faster.

4 Conclusion

The main advantage of Hill Climbing Algorithms is precision and sample predictability (a small st. dev). It may be useful when the goal is to find a very accurate extremum, even if it means a lot of computations, the main disadvantage of H.C is the high runtime. Out of all the H.C variations, *Best Improvement* seems to perform best, being the most accurate, although it is sometimes slower than *First Improvement*. The only exception is the case when the evaluated function is unimodal, then, all variations are equally accurate, so we can choose the one with the best runtime(*First Improvement* or *Best Imporvement*).

Even so, when the dimension is large, such as 30, Hill Climber's performance is close to the one of Simulated Annealing, so it may make more sense to use S.A, given that the runtime of S.A may be a hundred times better. This aslo due to the fact that H.C loses precision when dealing with large dimensions.

On other dimensions, S.A may calculate a good extremum, but such good results would be rare throughout the sample, which has a large st.dev. A solution would be to always select the best result out of the sample, there may be instances, such as Schwefel's function evaluation, where this result is very close to the actual extremum.

References

[1] Eugen Croitoru UAIC-FII
Notiuni: Metode traiectorie
Hill Climbing
Simulated Annealing https://profs.info.uaic.ro/ eugennc/teaching/ga/

[2] Walid Ben-Ameur, Article in Computational Optimization and Applications · December 2004;
"Computing the Initial Temperature of Simulated Annealing"

[3] Rafael E. Banchs, 25 November 2019; "Simulated Annealing"

[4] Wikipedia, "Hill Climbing"

[5] Wikipedia, "Simulated Annealing"

[6] GEATbx.com, http://www.geatbx.com/ver$_{33}$/$fcnindex.html$