Homework 1

Stamate Valentin 2B4

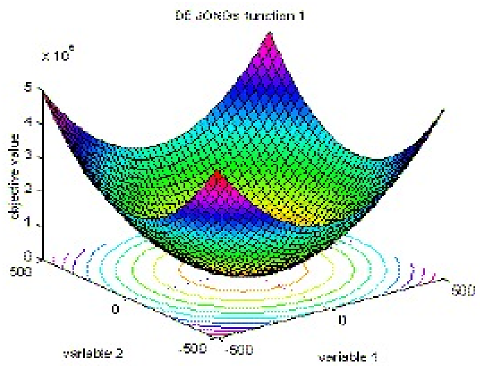
October 27, 2020

Abstract

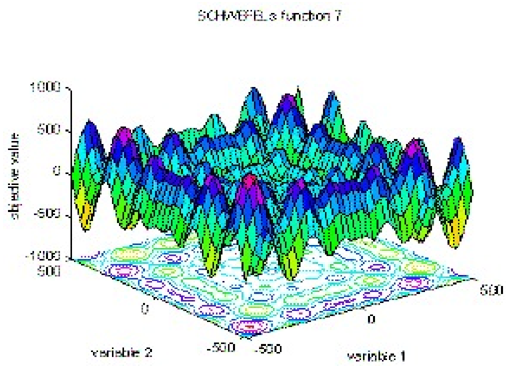
While the problems became more complex, finding an efficient algorithm to solve that problem became harder. This is why heuristic algorithms were a good alternative. The results given are pretty close and they have a better time complexity.

1 Introduction

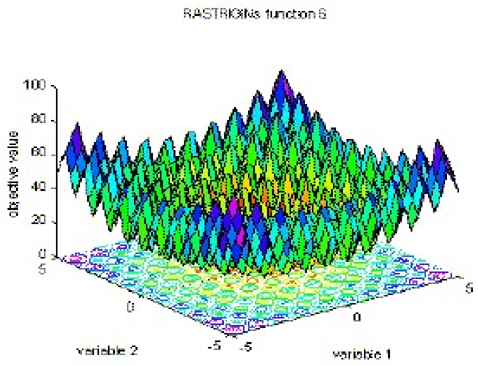
The raport contain contain the results, comparisons and a conclusion of my algorithms. The problem is finding the global minimum of a function with various dimensions. The motivation is to see if the methods I used(hich will be discussed in the next section) gives a good result for small or big imputs. To test the algorithms I selectd 4 functions: **DeJong's function**, **Schwefel's function**, **Rastrigin's function** and **Michalewicz's function**. As you can see each one has a different number of local minimum points.



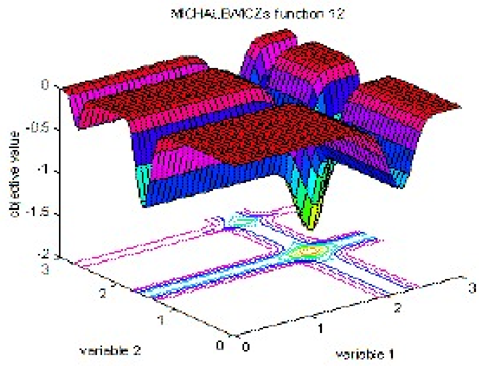
(a) DeJong's function



(b) Schwefel's function



(a) Rastrigin's function



(b) Michalewicz's function

2 Methods

I used three methods: **HillClimbing - First Improvement**, **HillClimbing - Best Improvement** and **Simulated Annealing** for each function. I combined HCFI and HCBI by adding a simple condition after finding the first neighbour.

For representing the numbers I used a bit map that contain all components of a point. This way I can easily generate a random neighbour. In my algorithm one neighbour in HC is one that has the most unsignificant bit changed the same way in numbers I add and subtract 1, but manipulating the bits. To generate a neighbour in SA I only negate one bit from the bit map. The main difference between SA and HC is that in SA I can choose a bad neighbour with a small probability. This was SA has a larger domain exploration than HC.

The initialization is the same for all methods. I generate a bit map with 0 or 1 for each position.

The HCFI algorithm stops when it cannot find any better neighbours. SA algorithm stops when is reaching a temperature eqal with $10^{-}8$.

3 Experiment Description

Each method I combined it with it's iterated version. The number of repetitions are equal with 10^5, enought to get a good result. To see how the methods are behaving over different inputs I choose 2, 5, 10, 15 and 30 as point dimension. The precision for each component is 5 meaning $\epsilon = 10^{-}5$. And the number global repetition is 30.

4 Results

Below are the results, a table for every input. Every cell has multiple values in this order: minimum value returned, running time, mean and standard distribution.

Algorithm Result (2)			
function	HCFI	HCBI	SA
De Jong	$1.81899e^{-}10$ 17s $1.81899e^{-}10$ 0	$1.81899e^{-}10$ 19s $1.81899e^{-}10$ 0	$4.73165e^{-}10$ 9s $8.665234e^{-}6$ $2.359996e^{-}5$
Schwefel	-837.964 27s -837.9059 0.04788257	-837.964 30s -837.903 0.05601569	-837.966 12s -837.8847 0.08802122
Rastrigin	$1.80426e-08$ 21s $1.80426e-08$ 0	$1.80426e-08$ 23s $1.80426e-08$ 0	0 9s 0.005112576 0.007384009
Michalewicz	-0.801323 22s -0.8013227 $4.794633e-07$	-0.801323 23s -0.8013223 -0.8013223	-0.801323 9s -0.8013221 $2.537081e-07$

Algorithm Result (5)			
function	HCFI	HCBI	SA
De Jong	$4.54747e-10$ 1m 56s $4.54747e-10$ 0	$4.54747e-10$ 1m 37s $4.54747e-10$ 0	$2.23713e-07$ 24s 0.04696127 0.1665665
Schwefel	-2088.19 3m 12s -1991.65 46.19519	-2057.46 2m 54s -1995.07 38.03182	-2053.71 37s -1895.261 79.70068
Rastrigin	$1.80426e-08$ 2m 44s $1.80426e-08$ 0	$1.80426e-08$ 2m 18s $1.80426e-08$ 0	5.95225 28s 9.125848 2.059847
Michalewicz	-3.69195 3m 13s -3.656661 0.02152056	-3.68325 2m 55s -3.646231 0.01848281	-3.6938 30s -3.460571 0.131784

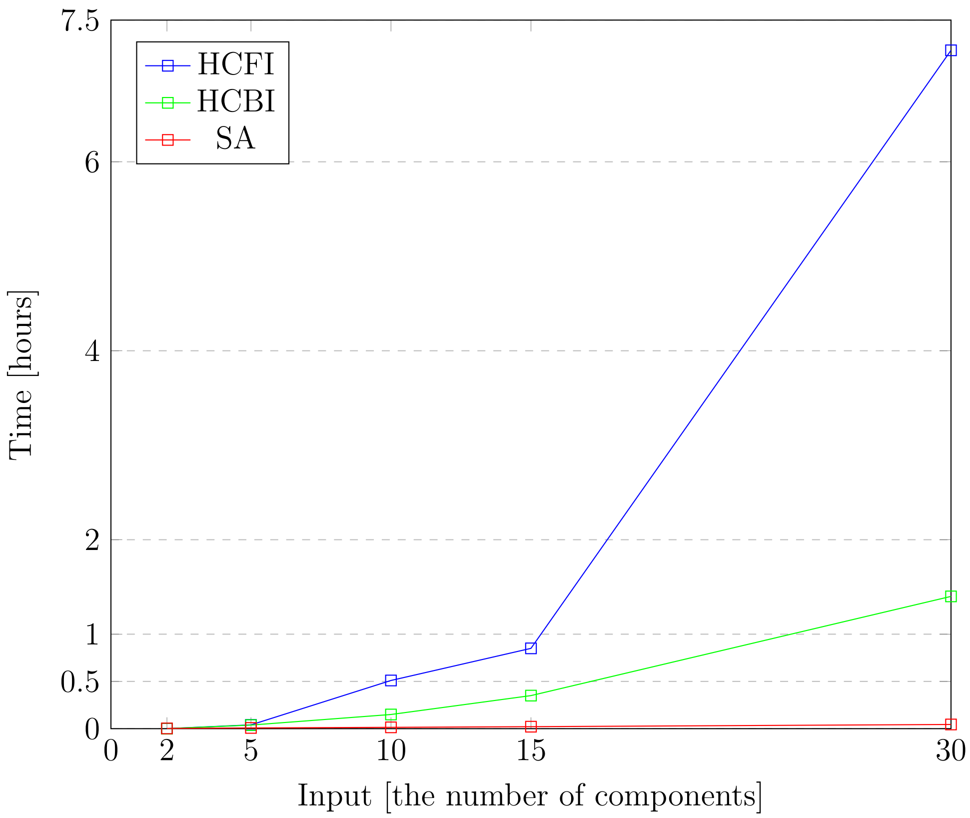
Algorithm Result (10)			
function	HCFI	HCBI	SA
De Jong	$9.09495e^{-}10$ 14m $9.09495e^{-}10$ 0	$9.09495e^{-}10$ 8m $9.09495e^{-}10$ 0	$1.05876e^{-}6$ 59s 0.0001858588 0.0004888328
Schwefel	-3634.03 23m -3394.599 97.74172	-3742.84 11m -3440.418 115.7556	-3630.58 59s -3211.317 247.4637
Rastrigin	$1.80426e^{-}8$ 18m $1.80426e^{-}8$ 0	$1.80426e^{-}8$ 9m $1.80426e^{-}8$ 0	8.68414 53s 25.81046 9.338441
Michalewicz	-7.84583 19m -7.05834 0.2676364	-7.60421 10m -7.131895 0.2289028	-8.1224 45s -6.258278 1.08582

Algorithm Result (15)			
function	HCFI	HCBI	SA
De Jong	$1.36424e^{-}9$ 40m $1.36424e^{-}9$ 0	$1.36424e^{-}9$ 17m $1.36424e^{-}9$ 0	$1.29213e^{-}6$ 1m 25s 0.3072009 1.222033
Schwefel	-4996.19 1h 4m -4675.259 143.6435	-4932.18 25m -4646.454 131.4299	-5035.86 1m 31s -4216.178 479.3601
Rastrigin	$1.80426e^{-}8$ 51m $1.80426e^{-}8$ 0	$1.80426e^{-}8$ 21m $1.80426e^{-}8$ 0	20.3447 1m 18s 36.68338 10.20887
Michalewicz	-10.938 44m -9.793424 0.3922462	-10.6536 20m -9.765478 0.2919562	-11.9283 1m 6s -8.65901 2.086646

Algorithm Result (30)			
function	HCFI	HCBI	SA
De Jong	$2.72848e^{-}09$ 5h 48m $2.72848e^{-}9$ 0	$2.72848e^{-}9$ 1h 11m $2.72848e^{-}9$ 0	0.0883307 2m 56s 2.972082 2.592697
Schwefel	−8481.25 8h 43m −8062.989 145.3006	−8437.5 1h 36m −8069.389 164.2671	−10557.7 2m 57s −9186.45 601.2258
Rastrigin	$1.80426e^{-}8$ 7h 11m $1.80426e^{-}8$ 0	$1.80426e^{-}8$ 1h 25m $1.80426e^{-}8$ 0	57.882 2m 40s 93.20304 19.97291
Michalewicz	−17.7899 7h 40m −16.48712 0.5326	−17.5032 1h 37m −16.40962 0.3950859	−23.5971 2m 18s −20.7609 1.685605

This graph represents the evolution of time when increasing the input size. The input is the number of components the function receives. I choose the Rastrigin's function because it has many local minimum points.

The input-time variation for Rastrigin's function



So, as shown in this graph the input has a signifficant time impact for HC algorithms. SA time doesn't change that much.

5 Comparisons

Next, I will discuss about the differences between these algorithms. The HC algorithm pick always a good neighbour: the best or the first. On the other side, SA algorithms can choose a bad neighbour but this chance descreases together with temperature. This way SA algorithm can explore more while HC algorithm can ramain stuck in a local minimum. Also, can be seen that SA algorithm is much more time efficient while giving a good result. But, increasing the input size, SA algorithm gives a more general result than a close one. Between the two HC methods, best improvement seems to be better and have a better time.

6 Conclusions

Finding global minimum can be a challenging goal. Deterministic algorithms can find it with backtracking but the time complexity makes it impossible to run in real time with big inputs. So, I tested three heuristic algorithms Hill Climbing first and best improvement and Simulated Annealing.

My conclusion is that SA can be a very good candidate for use cases bacause it is very fast comparing with HC algorithms witch gives a better result but they are much slower.

References

[1] https://stackoverflow.com/questions/26282709/heuristic-algorithm-for-finding-minimal-value-from-function/26283364

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

[3] https://en.wikipedia.org/wiki/Simulated_annealing

[4] https://www.youtube.com/watch?v=S9vs05eAGN0&ab_channel=EricSchirtzinger

[5] Second seminar notes

[6] http://www.geatbx.com/docu/fcnindex-01.html#P89_3085

[7] http://www.geatbx.com/docu/fcnindex-01.html#P150_6749

[8] http://www.geatbx.com/docu/fcnindex-01.html#P140_6155

[9] http://www.geatbx.com/docu/fcnindex-01.html#P204_10395

[10] https://www.geeksforgeeks.org/function-pointer-in-c/

[11] https://stackoverflow.com/questions/59565481/create-a-vector-from-a-txt-data-file-in-r