HW1 Report

Roșu Cristian-Mihai

November 3, 2019

Abstract

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. This is achieved by trading optimality, completeness, accuracy, or precision for speed. In a way, it can be considered a shortcut.

In this work, I explore this definition thoroughly by using two heuristic search algorithms to calculate the global minima of four distinct functions.

1 Introduction

This work aims to explore what a heuristic actually represents by addressing the problem of finding the global minimum of a function.

Here, I use *Rastrigin*'s, *De Jong*'s, *Schwefel*'s and *Michalewicz*'s functions as benchmarks to test two different heuristic search algorithms, **Iterated Hill-climbing** and **Simulated Annealing**. Below are the functions [1], in order:

$$f(x) = A \cdot n + \sum_{i=1}^{n} \left[x_i^2 - A \cdot cos(2\pi x_i)\right], A = 10, x_i \in [-5.12, 5.15]$$

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

$$f(x) = \sum_{i=1}^{n} -x_i \cdot \sin(\sqrt{|x_i|}), x_i \in [-500, 500]$$

$$f(x) = -\sum_{i=1}^{n} \left[\sin(x_i^2) \cdot \sin(\frac{i{x_i}^2}{\pi})^{2m}\right], m = 10, x_i \in [0, \pi]$$

Being multidimensional, each of the functions will be tested on 5, 10 and 30 dimensions, and for every test I will be recording the *minimum*, *maximum* and *average* **values** found, as well as the *minimum*, *maximum* and *average* **time** it took to obtain those results.

1.1 Motivation

The purpose of this work is to explore the capabilities of a heuristic.

In that sense, the problem of finding a function's global minimum has been chosen due to its ease of implementation and of understanding on a theoretical level. And as for the algorithms chosen, Iterated Hillclimbing and Simulated Annealing are the cornerstones of what genetics represents in computer science, on top of being equally easy to understand and implement.

The benchmark functions have been chosen such that the experiment may include as much variety as possible, since the domain and global minimum of each of them are very distinct.

2 Method

The algorithms used to calculate the global minima of the benchmark functions are **Iterated Hillclimbing** and **Simulated Annealing**.

They have been implemented as functions which take as parameters the function they are testing along with its domain and dimension. The solutions they produce are represented using bitstrings whose size are calculated in relation to the parameters using the following formula:

```
size=ceil(log2((upper-lower)*pow(10,PRECISION)))*dim
```

where `lower` and `upper` are the domain's ends and `dim` is the dimension on which the function is tested. `PRECISION` is a global variable used to decide how accurate the representation should be. In this work, I decided to change its value depending on the dimension that is being tested to achieve satisfactory run times: a precision of 10^3 for 5 dimensions, 10^2 for 10 dimensions and 10^1 for 30 dimensions.

Both algorithms also use the notion of *neighbour*. A neighbourhood is composed of every alternate bitstring resulted from negating one bit of the original.

To reach their solution, the algorithms *evaluate* an original bitstring and its neighbours and then decide which is *better* in order to progress, meaning whose function value is smallest.

Evaluating the bitstring is done by scaling each of its components (delimitated by size and dimension) to the function's domain and converting it to a real number, according to the following formula:

```
realnr=lower+decimal(component)*(upper-lower)/(pow(2, size)-1)
```

where `decimal()` is a function that converts a bitstring into an integer.

Below are the details for each of the algorithms [2]:

```
ITERATED HILLCLIMBING
begin
t := 0
initialize best
repeat
    local := FALSE
    select a candidate solution (bitstring) vc at random
    evaluate vc
    repeat
        vn := Improve(Neighborhood(vc))
        if eval(vn) is better than eval(vc)
            then vc := vn
        else local := TRUE
    until local
    t := t + 1
    if vc is better than best
        then best := vc
until t = MAX
end
```

Apart from the usual ones, the function this algorithm has been implemented into also takes an improvement method as one of the parameters.

A random bistring is generated as candidate solution. After that, another candidate solution is selected among the first's neighborhood according to the improvement method.

There are two ways of choosing which is the next candidate:

1. The first neighbour encountered whose evaluation is better than the original's - *First Improvement*

2. The neighbour whose evaluation is the best out of all the other ones - *Best Improvement*

If this next candidate's evaluation is better than the original's, the variable containing the original takes the candidate's value and the process repeats itself until that is no longer true. In the end, we are left with the *global minimum*.

Simulated Annealing

```
begin
t := 0
initialize the temperature T
select a current candidate solution (bitstring) vc at random
evaluate vc
repeat
    repeat
        select at random vn - a neighbor of vc
        if eval(vn) is better than eval(vc)
            then vc := vn
        else if random[0,1) < exp(-|eval(vn)-eval(vc)|/T)
            then vc := vn
    until (termination-condition)
    T := g(T; t)
    t := t + 1
until t = MAX
end
```

Similarly to Hillclimbing, first we generate a random bitstring as a candidate solution. After that, however, the next candidate is randomly chosen from the original's neighbourhood, instead of following a certain rule.

If this next candidate's evaluation is better than the original's, the variable containing the original takes the candidate's value, and if that is not true then there is a very small chance that the swap happens anyway. That probability is calculated as shown in the pseudocode above.

The `termination-condition` that I chose depends on the `temperature`. Being initialized at 1000, it is slowly reduced after every iteration using the function `g(T; t)`, which I opted to simply be `T=T*0.9`. Once it reaches a value lower than 1, the algorithm is terminated and we are left with the *global minimum*.

As observed, each algorithm is run until a certain number `MAX` is reached. Due to their probabilistic nature, both of them need to be run several times in order to obtain consistent results which can be compared to our expectations. In this work, I opted for the minimum number of **30** tests.

3 Experiment

The experiment consists in running both Iterated Hillclimbing and Simulated Annealing through each of the benchmark functions on 5, 10 and 30 dimensions, over 30 iterations, in order to get a sample big enough to compare them by looking at the minimum, maximum and average values produced, and the minimum, maximum and average times it took to reach those values.

In the case of Hillclimbing, I consider two variations of the algorithm determined by the improvement method (first and best improvement).

The code for this can be found on Github [3].

4 Results

Below are 4 tables corresponding to each of the 4 functions that hold the results to the experiment: Rastrigin, De Jong, Schwefel and Michalewicz, in that order.

Method	Dimension n	5	10	30
Iterated Hillclimbing Best improvement	Min	0.995071	8.28659	48.0815
	Max	19.9291	29.7096	101.606
	Average	9.234	19.7144	75.7797
	Min time	2554ms	6221ms	48168ms
	Max time	4666ms	11067ms	72440ms
	Average time	3623ms	8725ms	59518ms
Iterated Hillclimbing First improvement	Min	5.21921	12.8861	66.9285
	Max	25.1373	39.4802	137.74
	Average	12.2246	23.7794	93.6836
	Min time	1400ms	4010ms	41126ms
	Max time	3532ms	7724ms	72195ms
	Average time	2313ms	6045ms	52207ms
Simulated Annealing	Min	112.013	188.201	554.406
	Max	172.538	311.735	792.406
	Average	142.2785	262.306	693.281
	Min time	105ms	145ms	310ms
	Max time	133ms	187ms	387ms
	Average time	120ms	174ms	350ms

Figure 1: Ratrigin's function results

Method	Dimension n	5	10	30
Iterated Hillclimbing Best improvement	Min	0	0.000250489	0.0487589
	Max	0	0.000250489	0.0487589
	Average	0	0.000250489	0.0487589
	Min time	4069ms	12846ms	109112ms
	Max time	6241ms	18467ms	144269ms
	Average time	5187ms	15210ms	125895ms
Iterated Hillclimbing First improvement	Min	0	0.000250489	0.0487589
	Max	0	0.000250489	0.0487589
	Average	0	0.000250489	0.0487589
	Min time	1549ms	6868ms	53938ms
	Max time	4038ms	10086ms	89449ms
	Average time	28662ms	8411ms	70592ms
Simulated Annealing	Min	41.0977	96.2099	253.647
	Max	105.117	198.859	427.112
	Average	80.5765	136.814	350.287
	Min time	108ms	152ms	314ms
	Max time	128ms	189ms	430ms
	Average time	112ms	172ms	370ms

Figure 2: De Jong's function results

Method	Dimension n	5	10	30
Iterated Hillclimbing Best improvement	Min	-2026.33	-3917.88	-
	Max	0	0	-
	Average	-1691.67	-3340.8	(aprox.) -10000
	Min time	10776ms	51035ms	-
	Max time	15711ms	75839ms	-
	Average time	13532ms	63567ms	(aprox.) 15min
Iterated Hillclimbing First improvement	Min	-1792.17	-3741.59	-
	Max	0	0	-
	Average	-1536.16	-3186.17	(aprox.) -10000
	Min time	5786ms	29066ms	-
	Max time	11261ms	50673ms	-
	Average time	8434ms	38260ms	(aprox.) 15min
Simulated Annealing	Min	-136.383	-162.843	-138.451
	Max	1352.56	1892.73	2886.78
	Average	616.212	991.266	1430.33
	Min time	167ms	261ms	621ms
	Max time	205ms	347ms	780ms
	Average time	182ms	304ms	689ms

Figure 3: Schwefel's function results

Method	Dimension n	5	10	30
Iterated Hillclimbing Best improvement	Min	-4.49515	-8.70385	-19.9845
	Max	0	0	0
	Average	-3.86747	-7.79965	-17.7059
	Min time	1897ms	4589ms	21336ms
	Max time	3301ms	10151ms	36197ms
	Average time	2648ms	7392ms	28380ms
Iterated Hillclimbing First improvement	Min	-4.63334	-8.26517	-18.3539
	Max	0	0	0
	Average	-3.68547	-7.07535	-15.0016
	Min time	1116ms	2870ms	15072ms
	Max time	1956ms	6286ms	28352ms
	Average time	1577ms	4297ms	20305ms
Simulated Annealing	Min	-0.908522	-1.13222	-4.60035
	Max	0	0	0
	Average	-0.282703	-0.313583	-1.80264
	Min time	88ms	145ms	234ms
	Max time	121ms	187ms	295ms
	Average time	109ms	163ms	263ms

Figure 4: Michalewicz's function results

5 Conclusions

Iterated Hillclimbing using the best improvement approach is by far the most efficient method of all. The values it produces are the closest to the actual global minima of the functions, and the time it needs to accomplish that are relatively good.

The first improvement approach comes very close to being as precise and its times are faster, however the gap becomes too large as we scale the dimension upwards.

Simulate Annealing is, as observed, the fastest, but the values it produces are too distant from our expectation. Although it comes relatively close considering the time it needs to reach those values.

However, one thing they all share is the struggle of searching through a wide function domain, as seen with Schwefel's function. Hillclimbing takes a lot of time to produce even one solution while Simulated Annealing is way off mark with its result.

Heuristics show high capabilities of tackling with problems that classical methods are not fit to undertake. A deterministic approach to the task at hand would have a hard time even producing a result in a reasonable amount of time, while heuristics allow us to explore the realm of possible solutions using approximations.

References

[1] Site with details of the functions used
`http://www.geatbx.com/docu/fcnindex-01.html#P150_6749`

[2] Course site with the algorithms' details
`https://profs.info.uaic.ro/~pmihaela/GA/laborator2.html`

[3] Github repository for the project
`https://github.com/Nenma/ga-hw1`

[4] Wikipedia page for heuristic
`https://en.wikipedia.org/wiki/Heuristic_(computer_science)`

[5] Simple Latex tutorial I used
`http://www.docs.is.ed.ac.uk/skills/documents/3722/3722-2014.pdf`