2021-03-09

Comparing Gravitational Search with and without Annealing

David Kleingeld

Email
dskleingeld@gmail.com

Contents

1 Introduction

Many fields depend on good optimization techniques. Metaheuristics commonly used are Particle Swarm Optimization and Simulated Annealing.

There is no best meta heuristic for all existing problems. In 2009 a new algorithm inspired by the physics of gravity: the Gravitational Search Algorithm (GSA) was introduced[1].

In GSA there are multiple particles spread over the parameter space of the optimization problem. They each have a fitness given the optimization problem with their individual parameters. We can see that fitness as their mass. Newtons second law and his law of gravitation is then used to determine how these particles move at the next time step. For GSA to work best the law of gravitation is modified and some randomness is introduced.

GSA is weak in its local search ability to improve this Gravitational Algorithm Based Simulated Annealing (GABSA) was created. It uses simulated annealing to improve the local search ability [2]. Here I attempt to reproduce some of the results from the GABSA paper. However as there is no publicly available source for GABSA I will first implement GSA before extending it with GABSA.

This report discusses my implementation of GSA and GABSA, shows how my implementations perform compared to the cited papers before discussing the performance and finally concluding if this replicates the results of the papers.

2 Theory

Here I will give a concise explanation of both search algorithms starting with GSA as the other is a slight modification upon this. For a more detailed description refer to the GSA[1] and GABSA[2] papers.

2.1 GSA

The interaction between the particles are governed by Newtons laws, given the mass (fitness[1]) and position (parameters) of each particle (solution) we can derive how they move. In eq. (1a) m is the mass of our particle for whom we want to know its next position, M_i and R_i the mass and distance to one of the other particles, G[2] a decreasing function, $\boldsymbol{a}$ how our particle will accelerate finally k starts out as the total number of particles and linearly decreasing to one during the search. GSA uses a modification of Newtons law of gravitation

[1] normalized between 0 and 1
[2] a constant in physics

where R^2 is replaced with R.

$$\boldsymbol{F} = m\boldsymbol{a} \qquad \boldsymbol{F} = \sum_{i=0}^{i=k} \frac{GmM_i}{R_i} \tag{1a}$$

$$m\boldsymbol{a} = \sum_{i=0}^{i=k} \frac{GmM_i}{R_i} \tag{1b}$$

$$\boldsymbol{a} = \sum_{i=0}^{i=k} \frac{GM_i}{R_i} \tag{1c}$$

Note $\boldsymbol{a}$ does not depend on m, without this result any solution with a fitness of zero would ruin the search turning all values into `Nan`. To find the parameters for the next iteration of the search we update all particles velocities and then their positions using eq. (2a). The amount the velocity change is randomized (this improves GSA exploration), here r is a random number between 0 and 1.

$$\boldsymbol{v}_{t_2} = \boldsymbol{v}_{t_1} + r\boldsymbol{a}\Delta_t \tag{2a}$$

$$\boldsymbol{s}_{t_2} = \boldsymbol{v}_{t_2}\Delta_t \qquad \Delta_t = 1 \tag{2b}$$

When each particle ends up at the same location the search result converged and we can not search further as the particles will no longer accelerate anywhere.

2.2 GABSA

Adding annealing to the parameter update in eq. (2a) gives us GABSA. To do so we do the following after eq. (2b)

$$m_{t_1} = f(\boldsymbol{s}_{t_1}) \tag{2c}$$

$$m_{t_2} = f(\boldsymbol{s}_{t_2}) \tag{2d}$$

$$exp = e^{-\frac{m_{t_2} - m_{t_1}}{T}} \tag{2e}$$

$$\tag{2f}$$

$$\boldsymbol{s}_{t_2} = \begin{cases} \boldsymbol{s}_{t_2} & m_{t_2} > m_{t_1} \vee r_a \leq exp \\ \boldsymbol{s}_{t_1} & m_{t_2} \leq m_{t_1} \end{cases} \tag{2g}$$

Here T, the temperature, is a decreasing *cooling* function and r_a new random number between 0 and 1. As the temperature decreases the chance lowers that we accept new parameters when they lead to a worse solution.

3 Implementation

For GSA eq. (3) is used to calculate G at each timestep. With g_0 and α paramaters of the search and N the total number of search iterations.

$$G_i = g_0 \cdot e^{-\alpha * i/N} \tag{3}$$

The number of particles that still attract other particles (k in eq. (1a)) drops as in eq. (4), Here P is the number of particles, N the number of search steps and i the current step, starting at 1.

$$k_i = P - \frac{P}{i/N} + 1 \tag{4}$$

As cooling function for GABSA we used simple linear cooling starting at temperature T_0 and cooling to 0 see eq. (5).

$$t_i = T_0 \cdot \frac{N - i + 1}{N} \tag{5}$$

3.1 Code

I chose to write the implementation of GSA from scratch in *Rust*[3]. Rust was chosen as I am interested in seeing how suitable Rust is for such work, additionally the performance is welcome during testing. The result is a GSA object keeping track of the problem and search parameters and optional methods to set a seed for the random generator, chose weather to use annealing (GABSA) and finally to search with or without gathering statistics.

To create a GSA object we need:

1. g_0 and α
2. the maximum number of search steps N
3. the evaluation function that takes an array of input values and returns a single number
4. the end criterion, a function that given the current iteration and best solution decides if the seach should continue

[3]https://en.wikipedia.org/wiki/Rust_(programming_language)

To start a search call either *search* with the search range and population size to use or call *seach_w_stats* with an additional *stats* argument that implements the *Stats* trait. One such object that is provided is *TrackFitness*, it keeps track of the average and best fitness during the search.

4 Results

Here I present my GSA implementation evaluated using test functions from both papers[4]: $f_{1_{gsa}}$ and $f_{2_{gsa}}$ from the GSA paper and $f_{1_{gasba}}$ $f_{2_{gabsa}}$ and $f_{3_{gabsa}}$ from the GABSA paper. Finally I will compare my GSA and GABSA implementation using for $f_{1_{gsa}}$, $f_{2_{gsa}}$ and $f_{3_{gabsa}}$.

In practise the only difference between the papers f_1 and f_2 is their dimensionallity. See table 1 for the definition of these functions and table 2 for the paramaters used. For each function 100 searches where performed.

Table 1: *Test functions used*

	Function	Range	Optimal value
$f_{1_{gsa}}$	$\sum_{i=1}^{d} x_i^2$	$[-100, 100]^d$	$[0,0]^d = 0$
$f_{2_{gsa}}$	$\sum_{i=1}^{d} \lvert x_i \rvert + \prod_{i=1}^{d} \lvert x_i \rvert$	$[-10, 10]^d$	$[0,0]^n = 0$
$f_{3_{gabsa}}$	$0.5 + \frac{\sin^2\sqrt{x_1^2 + x_2^2} - 0.5}{\left[1 + 0.001\left(x_1^2 + x_2^2\right)\right]^2}$	$(-100, 100)$	$(0,0) = 0$

Table 2: *Paramaters used for the gsa and gasba runs*

Function	Population (N)	Dimension (D)	# Iterations	g_0	α	t_0
f_{gsa}	50	30	1000	100	20	-
$f_{i_{gasba}}$	50	2	50	100	20	20
$f_{3_{gasba}}$	50	2	100	100	20	20

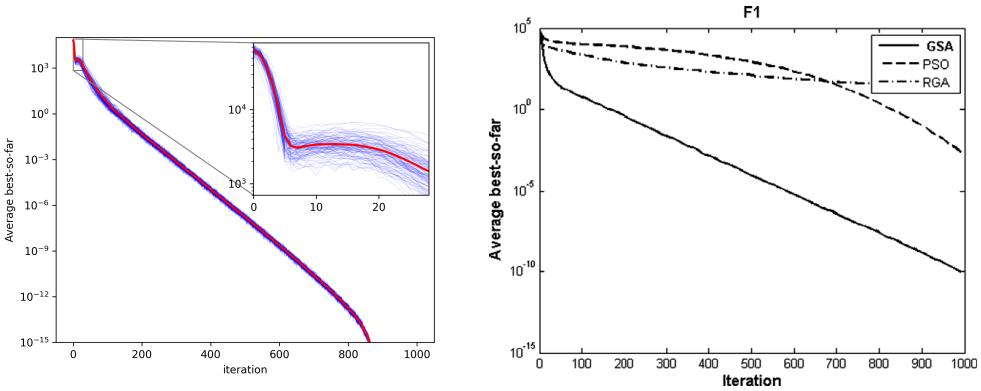
[4]The suffix to a function $f_{i_{\text{suffix}}}$ refers to the paper not the method used

Table 3: *Numerical results found here compared to those in the GSA and GABSA papers*

Function	Annealing	Average best so far		Optimal Solution	
		Here	Paper[1]	Here	Paper[2]
$f_{1_{gsa}}$	no	$[5.7 \pm 7.9] \cdot 10^{-19}$	$7.3 \cdot 10^{-11}$	$2.0 \cdot 10^{-20}$	-
	yes	$[5.7 \pm 7.4] \cdot 10^{-19}$	-	$2.6 \cdot 10^{-20}$	-
$f_{2_{gsa}}$	no	$[2.8 \pm 1.5] \cdot 10^{+01}$	$4.0 \cdot 10^{-5}$	6.4	-
	yes	$[8.5 \pm 2.9] \cdot 10^{+01}$	-	26	-
$f_{3_{gabsa}}$	no	$[2.8 \pm 2.7] \cdot 10^{-02}$	-	$5.6 \cdot 10^{-17}$	-
	yes	$[1.8 \pm 1.0] \cdot 10^{-01}$	-	$5.6 \cdot 10^{-17}$	$2.6 \cdot 10^{-6}$
$f_{1_{gabsa}}$	yes	$[4.7 \pm 7.8] \cdot 10^{-2}$	-	$8.0 \cdot 10^{-7}$	$1.6 \cdot 10^{-6}$
$f_{2_{gabsa}}$	yes	$[2.5 \pm 1.4] \cdot 10^{3}$	-	$1.7 \cdot 10^{-2}$	$1.2 \cdot 10^{-4}$

The GSA implementation

In fig. 1 we see the comparision between my implementation and that of the GSA paper. Note the slight hump at the start enlarged in the top right and the steeper decent of my implementation.



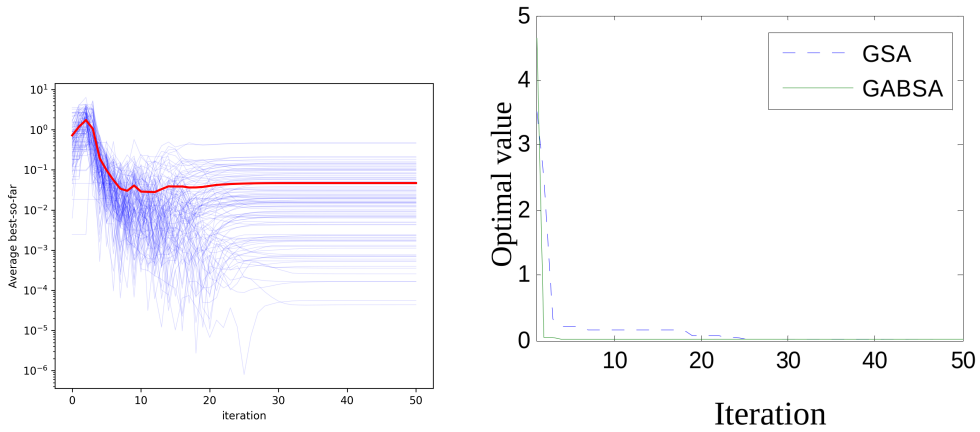
(a) *My implementation, in blue 100 individual runs in red the average of these runs*

(b) *Implementation in GSA paper, in black the performance of GSA*

Figure 1: *Average best-so-far fitness for function* $f_{1_{gsa}}$ *compared.*

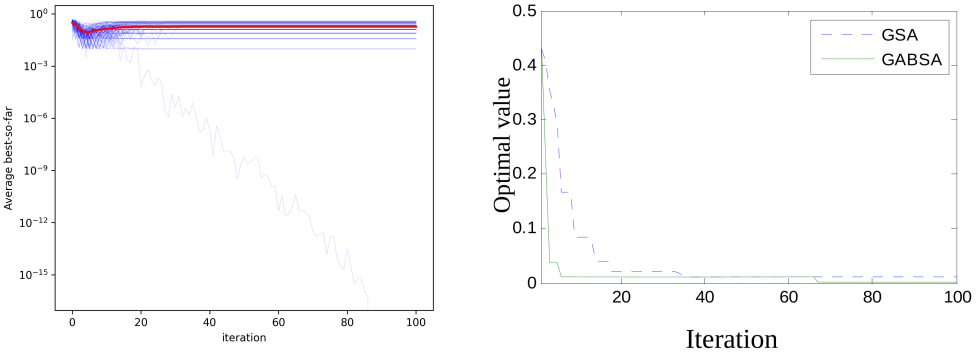
The GABSA implementation

In figs. 2 and 3 the comperision between my implementation and that in the GABSA paper. Note in fig. 2a again the bump at the start and the conversion to discrete values. In fig. 3a we see only one out of a hundred runs that follows the *Optimal solution* figure for $f_{2_{gabsa}}$ from table 3. The plots from the GABSA paper (figs. 2b and 3b) are not usable for comparison as they have a linear axis and little detail visible.



(a) *My GABSA implementation* **(b)** *The original GABSA implementation*

Figure 2: *Problem $f_{2_{gbsa}}$ from the GABSA paper[2] compared*

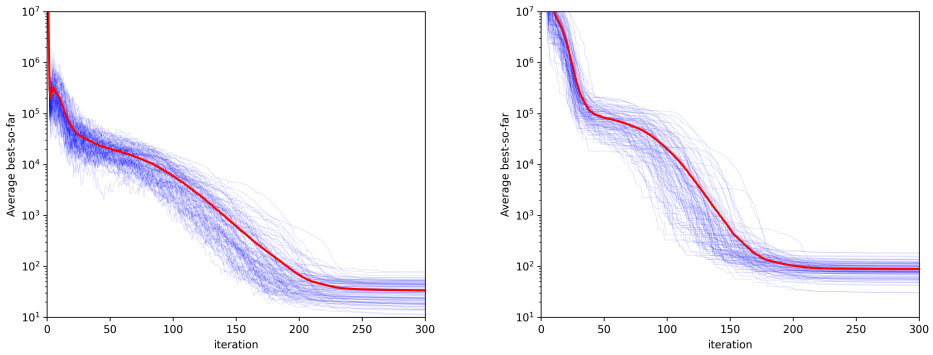


(a) *My GABSA implementation* **(b)** *The original GABSA implementation*

Figure 3: *Problem $f_{3_{gbsa}}$ from the GABSA paper[2] compared*

GSA vs GABSA

A full comparison between GSA with and without annealing is out of the scope of this report. In fig. 4 we compare my implementation of GSA and GABSA.



(a) *GSA* **(b)** *GABSA*

Figure 4: *Problem $f_{2_{gsa}}$ from the GSA paper[1] using my GSA and GABSA implementation*

5 Discussion & Conclusion

I implemented both GSA and GABSA and attempted to replicate parts of the GSA[1] and GABSA[2] papers. Let us see what I can conclude from the results, starting with GSA.

In table 3 there are two comparisons between my and the authors implementation of GSA. Function f_1 optimized to a better value, the comparison using f_1 in fig. 1 shows that my implementation also converged faster. The hump at the start with my implementation is not present in the authors work. Clearly my implementation can not be their GSA. Looking at f_2 my work converges 6 orders of magnitude worse then it should. I can only conclude that I did not successfully replicate GSA.

The GABSA paper [2] did not specify the cooling function used for annealing and its plots have little to no detail. The latter due to the strange choice for a linear scale[5]. Additionally instead of reporting an average and variance the paper only mentions the best value reached among many runs. The paper does not mention the number of tries to reach this value.

Since my GABSA work is based on my GSA implementation which I just concluded was faulty we expect to perform weakly. From the plots we see my implementation has a huge variance between runs. We see (table 3) that our results are better for 3 of the 2 problems tried. Given the way the results are presented[6] in the GABSA paper I can not compare to it. I do not know whether my GSA work comes close to a correct GABSA implementation. To conclude, neither GSA and GABSA where correctly replicated here, GABSA can never be replicated exactly without access to the original data or source.

Given the state of the GABSA paper[2] it should be recreated and the results there presented verified. Future work should include building GABSA on top of a verified GSA implementation then trying multiple cooling functions. Alternatively time could be spend fixing the here presented GSA implementation.

[5] this hides almost every detail of the plot

[6] frankly speaking I am disappointed this paper got through peer review

References

[1] Esmat Rashedi, Hossein Nezamabadi-pour and Saeid Saryazdi. "GSA: A Gravitational Search Algorithm". In: *Information Sciences* 179.13 (2009). Special Section on High Order Fuzzy Sets, pp. 2232–2248. ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2009.03.004. URL: https://www.sciencedirect.com/science/article/pii/S0020025509001200.

[2] Esmat Rashedi, Hossein Nezamabadi-pour and Saeid Saryazdi. "GSA: a Gravitational Search Algorithm". In: *Information Sciences* 179 (June 2009), pp. 2232–2248. DOI: 10.1016/j.ins.2009.03.004.

A Run Instructions

A.1 Installation

Install the rust version management tool[7]. Unzip this project then navigate to the directory containing `Cargo.toml`. Alternatively this work can be cloned from github[8]. Set the rust version for the current directory to nightly:

```
rustup override set nightly-2021-3-09
```

A.2 Running

Then build from source and run the experiments for GSA run in the root dir of the repository[9]:

```
cargo r --release --gsa
```

To generate the data for GABSA use:

```
cargo r --release --gabsa
```

Then the plots can be generated by running[10]:

```
python3 plot.py
```

[7] https://www.rust-lang.org/learn/get-started
[8] https://github.com/dskleingeld/gravitational_search
[9] the same dir as Cargo.toml
[10] or 'python plot.py' if your python version 3 install is called python