COMPARISON OF SWARM INTELLIGENCE ALGORITHMS

CS2IS2 Final Project (2017/2018)

Marcus Sung, Gustavo Panez Velazco, Saad Tariq Malik

sungm@tcd.ie, panezveg@tcd.ie, maliksa@tcd.ie

Abstract. In this paper we analyze 3 prominent Swarm intelligence algorithms, namely Particle Swarm Optimization, the Gravitational Search algorithm and the Firefly algorithm. Open source implementation of the algorithms is used to benchmark their performance against notable optimization problems. Past work is discussed and related to the analysis performed. A comparison between the algorithms is discussed.

1 Introduction

Optimisation has always presented itself in everyday life. In order to deal with restrictions. Individuals have to produce a solution that serve the source limits and lack of knowledge of the full problem. They are common problems in disciplines such as computer science, data analytics engineering and many more.

A class of solution methods that address these problems is metaheuristic algorithms. These algorithms are a non-deterministic, which search the solution space based on an educated guess and trial and error approaches. Swarm based algorithms are part of this solution class, and are mostly inspired by the social behaviour of animals. The algorithms analysed in this paper are swarm based algorithms.

2 Problem Definition and Algorithm

2.1 Optimization Problem

An optimization problem is the problem of finding the best solution from the solution space. Typically, this involves either minimizing (minimization problem) or maximizing (maximization problem) a function (called the objective function) given certain constraints.

Optimization problems can be further divided into two categories: Discrete or continuous. Discrete optimization problems are ones where the solution space's variables are discrete. Similarly, for continuous optimization problems the solution variables are continuous.

2.2 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a population-based algorithm that is used to solve optimization problems. It was inspired by flocking of birds, and in PSO a 'particle' represents a potential solution. Particles move around the solution space in a certain fashion and a solution is declared when all particles converge at a point.

2.2.1 Pseudocode

Below is the pseudocode for PSO [4]:

Initialize individuals

```
Find cur_best
Set gbest = cur_best
FOR i = 1 to N
  Calculate individual's velocity
  Change individual's velocity
  Update individual's positions
  Select new agents as per the selection method
  IF cur_best better than gbest
    SET gbest to cur_best
  END IF
END FOR
save gbest
```

Where: cur_best: An individual's local best solution; gbest: The global best solution found so far; N: Number of iterations

2.2.2 Mathematical Model and formal definition

Formally, the position vectors can be defined as follows [1]:

$$\overrightarrow{X_i^d} = [x_i^d, y_i^d, z_i^d, \dots]$$

where X_i^d= Position vector of the ith particle on the dth iteration. X can have any number of dimensions.

The PSO search strategy involves updating the position vectors using a velocity vector, defined as follows [1]:

$$\overrightarrow{V_i^{t+1}} = w\overrightarrow{V_i^t} + c_1 r_1 \left(\overrightarrow{P_i^t} - \overrightarrow{X_i^t}\right) + c_2 r_2 \left(\overrightarrow{G^t} - \overrightarrow{X_i^t}\right)$$

Where:
V_i^t, V_i^{t+1}:Velocity vector of the ith particle on (t)th and (t+1)th iteration,
V_i^t= Velocity vector of the ith particle on the (t)th iteration,
P_i^t= Position vector of the personal best solution after (t)th iteration,
G^t= Position vector of the global best solution after (t)th iteration,
X_i^t= Position vector of the particle after (t)th iteration.
r_1, r_2, r_3: Magnitudes of the vectors that can be tuned.
w, c_1, c_2 : **Input parameters** (details to follow later)

The updated position is the vector addition of the position vector at (d)th iteration and the velocity vector at (d+1)th iteration [1].

$$\overrightarrow{X_i^{d+1}} = \overrightarrow{X_i^d} + \overrightarrow{V_i^{d+1}}$$

2.2.3 Explanation

The general idea is for individuals to move around the solution space, while keeping track of the best solution that the particle itself has encountered (personal best), the best solution that the swarm as a whole has encountered (global best), and the current velocity of the particle

After each iteration, the PSO search strategy updates the velocity vector using the 3 rules, i.e. each particle needs to move a certain amount [1]: 1) In its current direction, 2) In the direction of its personal best solution, and 3) In the direction of the global best solution. The vector sum of the above 3 vectors results in the updated velocity vector.

2.2.4 Input parameters and their effect

There are 3 input parameters: w (inertia), c1 (cognitive component) and c2 (social component), and all these parameters impact the **exploration** and **exploitation** of the algorithm. *Exploration* involves searching for a better solutions far away in the solution space, whereas *exploitation* involves searching nearby the current position and obtaining a better solution.

When the inertia is 0, the particles do not move far away in their current direction. Therefore, the exploration is minimum (wrt to inertia) and exploitation is maximum. In contrast, when inertia is high, the exploration is maximum whereas the exploitation is minimum [1]. When c1 is 0 and c2 is maximum, the exploration is maximum and exploitation minimum. When c1 is maximum and c2 is 0 the exploration is highest and the exploitation is minimum.

2.2.5 Critique on research paper on PSO

Eberthart et al introduced a new form of PSO [2]. This new implementation (dubbed the 'lbest' paradigm), modifies the social component of the PSO search strategy and introduces a new 'neighborhood' parameter. Rather than keeping track of and tending towards the global best solution of the entire swarm, in the 'lbest' version of PSO each particle only tracks the nearest neighbors' best known solution.
The paper tested both the original PSO algorithm and the 'lbest' version against Schaffer f6 function, which has many local optima. The final conclusion of the paper was that, while the original PSO performs better in terms of number of iterations, the lbest PSO seems to be more resistant to local minima and performs better.

The paper controlled for the following parameters in comparing the algorithms: neighborhood, number of particles and Vmax (maximum magnitude of velocity). It measured the median number of iterations needed to reach the global optimum.

One drawback in the study was that for all trials, the population size was set at 20 and it did not study the effects of population size on the performance of the algorithms.

2.3 Gravitational Search Algorithm [6]

The Gravitational Search Algorithm (GSA) is a relatively new population-based optimization algorithm introduced in 2009 by Rashedi, et al. It represents search

agents as objects with mass, and applies basic Newton physics laws, such as the law of gravity and the law of motion, to coalesce the agents among each other. Search agents change by gaining additional mass as they achieve better solutions, given by a fit function. As the best agents get mass they apply a greater attraction force on other agents. Agents with large masses, i.e., good solutions, are influenced to a lesser extent by agents with smaller masses, because of the inertia effect.

2.3.1 Pseudocode

Generate initial population randomly
REPEAT
  Evaluate fitness for each particle
  Update constant G and factor k, and calculate best and worst fit among population
  Calculate mass M and acceleration a for every particle
  Update velocity and position of each particle
UNTIL end criteria is TRUE
Return best fit

2.3.2 Explanation

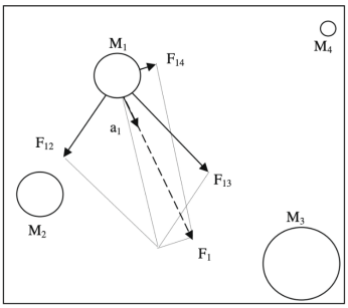
Gravity is defined by the following equation, where Fij is the force applied on the particle i by the particle j, G is a gravitational constant that decreases as time elapses, Mj is the mass of the particle j, Mi is the mass of the particle i, and R is the distance between both particles.

$$F_{ij} = G\frac{M_j \times M_i}{R^2}$$

The law of motion indicates that when a force is applied to a particle, it experiences acceleration proportional to the force, and inversely proportional to its mass, as described in the following equation.

$$a_i = \frac{F_i}{M_i}$$

The following figure depicts an example of how forces from multiple particles result in movement in a single direction by the resulting F1 force:



In GSA, the population represents search agents that are considered particles. Their masses represent how good of a solution they have found at the moment. Each particle is defined by 2 variables: its position, and its mass. At the beginning of the algorithm execution, particles are placed in random positions. A particle's position can be thought of as a vectors in a d-dimensional space. See below for example of particle ith position, which is defined across n dimensions:

$$X_i = (x_i^1, \ldots, x_i^d, \ldots, x_i^n) \quad \text{for } i = 1, 2, \ldots, N,$$

GSA takes the law of gravity and calculates the force that particle j applies on particle i at time t, e.g., iteration t, along dimension d (see below). In this equation, R is the Euclidean distance between both particles.

$$F_{ij}^d(t) = G(t)\frac{M_i(t) \times M_j(t)}{R_{ij}(t) + \varepsilon}\left(x_j^d(t) - x_i^d(t)\right)$$

Thus, the total force applied on particle i along the dimension d would be the sum of forces applied by all the other particles. However, GSA introduces a probabilistic component to allow the algorithm to explore solutions, which is especially important at the beginning to avoid focusing on a local optima. For that purpose, it computes a random weight for each component force when calculating the sum of forces along dimension d. Moreover, to balance the algorithm, GSA builds its exploitation capability by only taking into consideration the force from the k particles with largest mass. This parameter decreases linearly as time passes. At the beginning, all agents apply force, but eventually only the agent with the greatest mass will apply force on the others. The resulting equation includes a random number with range from 0 to 1 generated for each i,j pair:

$$F_i^d(t) = \sum_{j \in Kbest, j \neq i} rand_j F_{ij}^d(t)$$

The gravitational constant decreases with time to adjust the accuracy of the search. The following formula defines how to calculate the value of G at time t:

$$G(t) = G(t_0) \times \left(\frac{t_0}{t}\right)^{\beta}, \quad \beta < 1,$$

The mass of each particle depends on the fit function's output with respect to the particle. For example, in a function optimization scenario, the position of the particle serves as input to obtain the fit function output. To calculate the mass of particle i, Mi, a normalization step is performed over the resulting value, followed by transformation from a scalar to a percentage value, as described by the two following equations:

$$m_i(t) = \frac{fit_i(t) - worst(t)}{best(t) - worst(t)}$$

$$M_i(t) = \frac{m_i(t)}{\sum_{j=1}^{N} m_j(t)}$$

The normalization step is with respect to the best and worst fit value achieved among the population during the current iteration. The calculation of the masses allows the algorithm to compute the forces applied by each particle over others, as explained before. Finally, once the acceleration is calculated, it is possible to compute the velocity and the new position of each particle after it has experienced all the combined forces. GSA considers that the new velocity of a particle is a fraction of its

previous velocity plus the effect given by its acceleration. The velocity calculation also uses a random factor with range 0 to 1, as a way to give a stochastic characteristic to the search. Having calculated the new velocity, GSA updates the particle's position easily:

$$v_i^d(t+1) = rand_i \times v_i^d(t) + a_i^d(t)$$

$$x_i^d(t+1) = x_i^d(t) + v_i^d(t+1)$$

Once all the masses, constant G, factor k, forces, and accelerations have been computed and the particles have been updated to their new positions, GSA considers the current iteration completed, and the process is ready to be repeated. Because the mass is heavily influenced by the fit value, which indicates good solutions, then as the algorithm iterates, particles should move closer and closer to the best solution, and eventually find the global optima.

2.2.5 Critique on research paper on GSA

GSA is not the first algorithm based on law of kinematics. For example, Central Force Optimization (CFO) algorithm already used such laws to solve optimization problem. But compared to CFO, GSA has stochastic capabilities in the way forces and velocities are calculated which allow it to explore options and avoid local optima. Moreover, GSA is more robust and its results tend to not be affected dramatically by the initial positions of the agents, given that they are placed at random instead of deterministically as CFO.

It's been argued that the algorithm's name is misleading as it is not really based in the law of gravity, because in the formula to compute $F_{ij}^d(t)$ it only uses the distance (R) between particles to normalize the direction of the force along dimension d. Experts reason that a true gravitational algorithm would use R^3 [8].

Finally, from our own analysis, we have observed that GSA requires more time per iteration compared to other algorithms such as PSO. We believe that is due to not only the many complex calculations required to determine the force and next position but also that these calculations are required for many particles, even when considering that GSA reduces the hyper parameter k as time passes. Many researchers have observed this problem and introduced improvements that allow GSA to scale more efficiently in terms of agents, and converge faster [9].

2.4 Firefly Algorithm (FFA)

Fireflies produce luminescent flashes as a signal system to communicate with other fireflies. In most cases this is done to attract the opposite sex or prey. The Firefly Algorithm is inspired by the firefly's biochemical and social aspects. The flashing light is produced by a process called Bioluminescence.
The rhythmic flash, rate of flashing and the amount of time constitutes the part of the signal system that attract the opposite sexes to each other. Light intensity obeys the inverse square law at a particular distance r from the light source. This means that

light intensity decreases with the distance which makes most fireflies visual to a limited distance.

2.4.1 Pseudocode

Below is the pseduocode for Firefly Algorithm:

```
Objective function f(x), x=(x1, x2, ... , xd)T
 Initialize a population of fireflies xi(i = 1, 2, ... , n)
 Define light absorption coefficient gamma
 WHILE count < MaximumGeneratons
  FOR i = 1 : n (all n fireflies)
    FOR j = 1 : i
    Light intensity Ii at xi is determined by f(xi)
    IF Ii > Ij
       Move firefly i towards j in all d dimensions
     ELSE
       Move firefly i randomly
     END IF
     Attractiveness changes with distance r via exp[-γ r2]
    Determine new solutions and revise light intensity
    END FOR j
  END FOR i
  Rank the fireflies according to light intensity and find the current best
END WHILE
```

2.4.2 Explanation

Some of the following assumptions are made in the FFA:

- All fireflies are attracted to each other regardless of sex
- The attractiveness and brightness decrease as the distance increases and are proportional to each other. The less bright firely will move to the brighter one and at random if there is not a brighter one.
- Brightness is determined or affected by the shape of the objective function.

The important issues in FFA are the formulation of the attractiveness and the variation of light intensity. The fireflies brightness reflects the location superiority and decides the moving direction. The degree of attractiveness determines the firefly's moving distance. The brightness and attractiveness are always updated, with the aim to achieve the goal of optimisation.
Relative brightness is defined as:

$$I = I_0 \times e^{-\gamma r_{ij}}$$

Where: I_0 is the original light intensity, γ is the light absorption coefficient and r_{ij} is the distance between firefly i and j. Light intensity will decrease as distance increases and along with media absorption. Coefficient γ is set to reflect this.
Attraction is defined by:

$$\beta = \beta_0 \times e^{-\gamma r_{ij}}$$

Where: β_0 is the original attraction value.
Location is updated by

$$x_i = x_i + \beta \times (x_j - x_i) + \alpha \times (rand - 1/2)$$

Where: x_i and x_j is the position of firefly i and j in the space; α is the step size factor, a constant on [0, 1]; rand is a random factor which is uniformly distributed on [0, 1] .
Two parameters of the algorithm are attractiveness coefficient and randomization coefficient play an important role in determining the optimal solution in the search space. The values are crucially important in calculating the speed of the convergence and the behaviour of FA algorithm. Firefly algorithm parameters may not change by the time during iterations.

2.4.3 Paper discussion

The shortcomings of FFA are that it can easily fall into local extreme points and are easy to premature. A paper by Liangyu Ma and Pengrui Cao [5] suggest two different algorithms that improve on the base FFA. These two improvements are based on inertia weight and chaotic sequence. It was found that as mentioned standard FFA was open to premature convergence. As the dimension of the functions tested increased there would be some failures to converge. The inertia weight based firefly algorithm performed better in comparison to the standard FFA but was still open to premature convergence and cannot optimise multi-peak functions.

The results of the paper showed that *Chaotic Sequence Based Variable-Scale FA* offered more accurate and stable results over the standard and inertia based algorithm. When caught in the local extreme point, the CSFA adopts variable-scale chaos optimization and the Gauss perturbation to fine search, which avoids the problem of local optimum and repeated oscillation around the global optimum. [5]

In multiple papers FFA can be seen to be compared to PSO. A paper [10] describes PSO as easy to implement with few parameters and overall less computing needed. They did not provide clear benchmarks or quantifiable results for this however, the results in this paper (section 4) prove this to be plausible. It describes FFA as flexible with any problem and results show this to be fairly plausible however its shortcomings may have led it to fall short on one function. A paper [10] describes FFA as the best out of two other algorithms which included PSO however failed to include computation time into the results. From the results in this paper we can see that the computation time is considerably less using PSO which is a factor worth looking at due to the significant difference.

4 Experimental Results

4.1 Methodology

The evaluation of the models was performed by testing the algorithms on a set of 5 common optimisation functions that have a single global optima [7]. These functions were Ackley, Easom, Sphere, Bohachevsky (Bohac), and Sum of Squares (Sos) functions. The test consisted in executing 25 runs of each algorithm over each of the functions. Each algorithm execution terminates after 100 iterations. At each run the mean error and execution time in microseconds was computed. Then, the results were averaged to get a final numbers of the performance results of each algorithm on each function.

4.2 Results

	GSA		FFA		PSO	
Function	**Mean Error**	**Time (us)**	**Mean Error**	**Time (us)**	**Mean Error**	**Time (us)**
Ackley	0.4163481936	635765.96	6.258057561	820914.52	5.953342924	14235.08
Easom	0.3654278763	629116.72	0.8314660661	679041.24	5.12E-09	8604.48
Sphere	0.2006613117	617482.56	0.2778053268	699571.48	3.01E-11	9647.4
Bohac.	0.1827698551	616901.48	0.2955551445	699378.64	1.65E-09	9470.56
SoS	0.1806994948	621159.04	0.2738230523	718920.52	2.88E-11	10171.84

The table above shows the results measured on each algorithm over the discussed functions.

4.3 Discussion

The results in section 4.2 shows the differences in the algorithms mean error and execution time with respect to each function.
A clear difference can be seen regarding execution time. PSO clearly outperforms the other two algorithms by approximately 2 orders of magnitude. This is due to its low computational complexity in comparison to GSA and FFA algorithms, in which PSO requires much less than the others. Furthermore, PSO consistently achieved the lowest mean error among the 3 algorithms, by 8 or more orders of magnitude for 4 out of the 5 functions tested. However, for the Ackley function, PSO's accuracy was significantly worse compared to its accuracy on the other functions. The Ackley function contains multiple attractive local optima close to the global optima. Thus, PSO, and FFA didn't explore the space enough and settled for local optima which caused a high mean error. FFA has similar execution time to GSA, and frequently achieves similar accuracy.

5 Conclusions

In this report three swarm intelligence algorithms were analysed: particle swarm optimisation, gravitational search algorithm and firefly algorithm. An in depth explanation of each algorithm was provided and also some of the past papers relating to each algorithm was discussed. An experimentation was conducted using open source packages to put some of the findings discussed in the papers to test. It was found that each algorithm on its own performed well however PSO performed significantly better in terms of overall error and execution time. Both GSA and FFA provide good results however can be tied down by their complexity as discussed. PSO allows much more flexibility to be built on top of it as it is already simple to implement compared to the others. Finally, GSA might be better option than FFA and PSO when one needs to solve optimization problems with many attractive local optima, while PSO might be the best option when short execution time is a priority.

References

1. *Learn Particle Swarm Optimization (PSO) in 20 minutes* (2018) YouTube video, added by Ali Mirjalili [Online]. Available at https://www.youtube.com/watch?v=JhgDMAm-imI [Accessed 16 April 2019].
2. Eberhart, R. and Kennedy, J. (n.d.).: A new optimizer using particle swarm theory. MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science.
3. Jing Wang, A Novel Firefly Algorithm for Portfolio Optimization Problem,IAENG International Journal of Applied Mathematics, 49:1 2019
4. SISDevelop, SwarmPackagePy, (2017), GitHub repository, https://github.com/SISDevelop/SwarmPackagePy
5. Liangyu Ma, Pengrui Cao: Comparative Study of Several Improved Firefly Algorithms∗, IEEE, 978-1, 2016
6. Rashedi, Esmat, Hossein Nezamabadi-pour and Saeid Saryazdi: GSA: A Gravitational Search Algorithm, Information Sciences Journal 179, 2009
7. Panez, G.: Analysis of Swarm Algorithms (2019), GitHub repository, https://github.com/panezg/AI_Assignment2
8. Gauci, M., Dodd, T.J. & Groß, R.: Why 'GSA: a gravitational search algorithm' is not genuinely based on the law of gravity, Natural Computing, 11-4, 2012
9. Mohd Sabri N., Puteh M., and Rusop M.: A review of gravitational search algorithm, International Journal of Advances in Soft Computing and its Applications, 2013
10. Ahmad Sanmorino: A Brief Comparison of Particle Swarm Optimization Algorithm and Firefly Algorithm, JURNAL INFORMATIKA UPGRIS Vol. 4, No. 1, (2018)
11. Siddharth Agarwal, M-tech, Amrit Pal Singh, Nitin Anand, M-Tech : Evaluation Performance Study of Firefly Algorithm, Particle Swarm Optimization and Artificial Bee colony algorithm for Non-Linear mathematical Optimization functions, 2012