Result Replication: "A Framework for Self-Tuning Optimization Algorithm"

Alex Gideon
Rochester Institute of Technology (RIT)
asg9582@g.rit.edu

Tyler Forman
RIT
tjf6126@g.rit.edu

Gerrit Krot
RIT
gnk1165@g.rit.edu

Abstract

This paper attempts to recreate the Self-Tuning Firefly Algorithm described in [2] by optimizing the hyper-parameters of the Firefly algorithm using the Firefly algorithm. The goal is to show drastic improvement in iterations to converge due to improved hyper-parameters across multiple functions. These functions include the original 5 tested in the paper: Ackley, Dejong Sphere, Rastrigin, Yang's Forest, Zakharov, and additionally, the Eggcrate function. Across every function, optimizing the hyper-parameters was able to drastically reduce the number of iterations needed to reach a "good" solution. By analyzing the best hyper-parameters found, we surmised that very small alpha values, $< \frac{1}{300}$ are necessary for convergence. β and γ are more subject to the specific function, with convex functions generally preferringβ within the range [.2,.5] and γ within the range (0,.2], and complicated functions preferring β values between [0.5,.8], and γ values between [.3,.8]. These results should help other people using the Firefly algorithm to pick starting parameters well suited for their individual problem, particularly if they know it's general shape.

1 Introduction

At its core, every problem in life is one that requires optimization. Every decision you make is at the cost of something else, and it's up to you to figure out which one yields the most benefit. In its most traditional form, optimization is a pure mathematical problem, generally involving calculus. However, calculus makes one key assumption, that is the function you are optimizing is differentiable. Reality however, is not so kind many problems in life are non-differentiable, meaning we cannot so easily mathematically find the optimal value, metaheuristic optimization comes from that realization. Algorithms that work on non-differentiable functions are incredibly valuable for these problems. Each of these algorithms generally have "hyper-parameters" that dictate the more specific operation. The hyper-parameters are chosen problem to problem, and as a result are not the same between applications. As will be shown, the choice of hyper-parameter can mean a speed increase of up to 30x faster convergence, so there is a real desire to get the "correct" parameters. However, the choices in parameters are extremely vague in most literature and it is often left to the user to choose alone. Remember, all problems are optimization problems and thus the choice of hyper-parameters can also be re-framed as an optimization problem. The main goal of this paper is to dive deeper into the idea of using metaheuristic algorithms, specifically the firefly algorithm, to optimize hyper-parameters of itself. The goal of this is to give the reader a better understanding of the choices behind the hyper-parameters of the firefly algorithm and allow the reader to choose parameters more effectively for specific problems in the future.

NeurIPS 2023 Generative AI and Biology Workshop.

1.1 Motivation

When exploring the many different available metaheuristics, one thing is clear between all of them: the choices in hyper-parameters are incredibly important. No matter how good your algorithm is at solving a particular problem, if the hyper-parameters you choose are bad, your algorithm will be bad. When learning about different types of algorithms and metaheuristics, we are generally given a range of "ideal values" into which each hyper-parameter should fall. Think about genetic algorithms where you have a probability of crossover and a probability of mutation. It's all fine and dandy if we are given these parameters, but we simply do not know if these parameters are best for our problem. A self-tuning algorithm is the idea that we can use an existing algorithm to tune itself. Furthermore, we can also use an algorithm to tune the hyper-parameters for any algorithm. This will hopefully make the choices of hyper-parameters clearer so that we can understand the reasoning behind how and why they are defined.

2 Related Work

2.1 Firefly Algorithm

At the core of this project is the Firefly algorithm. This algorithm is based on how real-life fireflies find mates using brightness of their light. The goal of this algorithm is to find a minimum of a function. We start by generating a number of fireflies within the search space of a given function, where each firefly has a position corresponding to a solution. Each firefly also has a brightness, which is based on its cost at the given position. At each iteration, each firefly attempts to move towards any firefly that is brighter than it. If there is a brighter firefly, it moves towards that firefly using the function:

$$x_i^{t+1} = x_i^t + \beta * e^{-\gamma r^2}(x_j^t - x_i^t) + \alpha\epsilon \tag{1}$$

Otherwise, the firefly moves randomly according to the function:

$$x_i^{t+1} = x_i^t + \alpha\epsilon \tag{2}$$

Here, a firefly i at position x_i moves towards a brighter firefly j at position x_j. r represents the distance between firefly i and firefly j. ϵ is a random value pulled from a normal distribution with a mean of 0, and α is a hyper-parameter that represents our step size for random movement. The hyper-parameter β represents firefly attractiveness, and hyper-parameter γ simulates light absorption through the medium, which ultimately determines how diminished the brightness of firefly j is to firefly i. γ can also be thought of as the degree to which fireflies are more attracted to closer fireflies than further away ones. In the event that a firefly i finds no other firefly brighter than it, it instead moves randomly using the function $x_i^{t+1} = x_i^t + \alpha\epsilon$. This process is run for a number of iterations or until an accepted tolerance is reached. After each iteration, the fireflies are checked and compared to the best position found by any firefly, which is determined by the cost at that position. If a better position is found such that it has a lower cost, the best position is updated. After a stopping criteria is reached, whether it be a maximum number of iterations or a desired tolerance, the best found position and cost are returned from the function.

2.2 Literature Review

We can look to the previous work of Ariyaratne, et al. [1] to see a practical application of the self-optimizing firefly algorithm [2], described in detail below in the Proposed Solution section of the paper. The goal of this paper was to demonstrate that the Ant colony system (ACS) method is viable for solving the Traveling Salesman Problem (TSP). To ensure that they were finding the best possible results, the hyper-parameters of ACS were tuned using the self-optimizing firefly algorithm. By tuning the hyper-parameters as the problem progressed, the authors were able to generally see an improvement to their best result in exchange for more time spent. This paper glosses over the exact manner in which they improved their hyper-parameters, merely saying that after the ants finish an iteration, they "update parameter values". Looking at the results for the paper, we can see that it takes about 50 trials before the hyper-parameters start to normalize. The way that this paper uses the self-optimizing firefly algorithm is different. The original paper describing the self-optimizing firefly algorithm [2], optimizes the hyper-parameters to reduce the number of iterations before the algorithm finds an acceptable solution. Instead, this paper optimizes the hyper-parameters to find

better solutions. The paper shows that the self-optimizing firefly algorithm can provide optimal hyper-parameters to suit a variety of goals, from improving the best result to improving the efficiency of the algorithm.

3 Problem Description

As it stands, there are recommended hyper-parameters when working with the Firefly algorithm for general use cases, but these aren't guaranteed to be the best. As such, the problem we wish to solve is finding optimal hyper-parameters for the firefly algorithm such that it converges in a minimal number of iterations for a given function. The functions we plan to find optimal hyper-parameters for are from the original paper.
The functions we will be referencing are the Ackley Function 3, Dejong's Sphere Function 4, the Egg Crate Function 5, Rastrigin's Function 6, Yang's forest function 7, and Zakharov's Function 8

$$-20 \times \exp\left(-0.2 \times \sqrt{\frac{1}{d} \times \sum_{i=1}^{d} x_i^2}\right) - \exp\left(\frac{1}{d} \times \sum_{i=1}^{d} \cos(2\pi x_i)\right) + e + 20 \tag{3}$$

Equation 3: Ackley Function, Optimal Value at [0,0..,0] = 0, Bounded from [-5,5]

$$f(\mathbf{x}) = \sum_{i=1}^{d} x_i^2 \tag{4}$$

Equation 4: Dejong Function, Optimal Value at [0,0..,0] = 0, Bounded from [-5.12,5.12]

$$f(x, y) = \sum_{i=1}^{d} (x_i^2) + 25 \times \sum_{i=1}^{d} (\sin^2(x_i)) \tag{5}$$

Equation 5: Egg Crate Function, Optimal Value at [0,0..,0] = 0, Bounded from $[-2\pi, 2\pi]$

$$f(\mathbf{x}) = 10 \cdot d + \sum_{i=1}^{d} \left[x_i^2 - 10 \cdot \cos(2\pi x_i)\right] \tag{6}$$

Equation 6: Rastrigin's Function, Optimal Value at [0,0..,0] = 0, Bounded from [-5.12,5.12]

$$f(\mathbf{x}) = \sum_{i=1}^{d} |x_i| \exp(-\sum_{i=1}^{d} \sin(x_i^2)) \tag{7}$$

Equation 7: Yang's Forest Function, Optimal Value at [0,0..,0] = 0, Bounded from $[-2\pi, 2\pi]$

$$f(\mathbf{x}) = \sum_{i=1}^{d} x_i^2 + \left(\sum_{i=1}^{d} 0.5 \cdot i \cdot x_i\right)^2 + \left(\sum_{i=1}^{d} 0.5 \cdot i \cdot x_i\right)^4 \tag{8}$$

Equation 8: Zakharov's Function, Optimal Value at [0,0..,0] = 0, Bounded from [-5,5]

4 Proposed Solution

Our proposed solution is to implement the Self-Tuning Firefly optimization algorithm outlined in the paper by Xin-She Yang [2]. This algorithm works by utilizing the firefly algorithm as a means to tune hyper-parameters for a given function to find the "best" set of hyper-parameters. We define "best" hyper-parameters as ones that allow a function to converge to an optimal value in the smallest number of iterations possible. To start, we define a three-dimensional search space that represents the set of all possible values for α, β, and γ. Then, we define 29 fireflies within this search space with random values bounded between 0 and 2 and an extra firefly with a set of recommend parameter values $\alpha = 0.25, \beta = 1, \gamma = 2$. The fireflies in this search space are referred to as parent fireflies. From here, we then run the firefly algorithm with the values of each parent firefly in the search space as the hyper-parameters for a given function and return the result in terms of an average number of iterations needed to converge. The fireflies created from the parent fireflies are known as child fireflies and are all run in parallel for the sake of runtime efficiency. We then use these returned values as our cost value for each parent firefly and move the parent fireflies in their search space using the outlined movement method in section 2.1. Initially, we use the recommended hyper parameters for the firefly movement, which again are $\alpha = 0.25, \beta = 1, \gamma = 2$. However, after the first iteration, we compare all of the fireflies to determine which one converged in the least number of iterations. Once we find it, we then set the hyper-parameters that are used for the parent firefly movement to the best value found. Due to how we set up the parent fireflies, on the first iteration, this value will either be the recommended set of hyper-parameters or one of the randomly determined hyper-parameters of one of the other parent fireflies. Regardless of which hyper-parameters are chosen, this process is repeated for 50 iterations. After each iteration, we check again to see if a better set of hyper-parameters has been found and update the parent firefly hyper-parameters accordingly. Once this process is finished, we return the best set of parent hyper-parameters found for this function.

5 Experimental Results

The parent firefly function was run with 20 fireflies; each mini-firefly had a population of 20, a maximum generations of 1000, and a convergence tolerance of 10^{-4}. The "cost" of the parent firefly was the average iteration time for a given firefly with a given set of hyper-parameters over ten trials. The parent firefly had a maximum generation of 50 and was run 30 separate times. The resulting best hyper-parameters and average iterations it took to converge are shown in figures 1(a), 1(b), **??**, 1(d), 1(e), and 1(f).

For ease of use, statistics for function convergence can be found in table 1.

Table 1: Hyper-parameter statistics by function.

Name	Alpha	Beta	Gamma	Iterations
Ackley	0.02973 ± 0.07596	0.75416 ± 0.56086	0.96338 ± 0.77917	549.25667 ± 409.67245
Dejong Sphere	0.03156 ± 0.01712	0.28311 ± 0.15380	0.06971 ± 0.07725	9.64667 ± 1.95778
Rastrigin	0.00217 ± 0.00115	0.79738 ± 0.40925	0.33396 ± 0.24639	110.84667 ± 224.96427
Yang Forest	0.00157 ± 0.00465	0.68054 ± 0.49521	0.38306 ± 0.51235	441.98333 ± 629.50887
Zakharov	0.02822 ± 0.01423	0.34539 ± 0.22984	0.12541 ± 0.15054	11.39667 ± 3.88565
Eggcrate	0.00760 ± 0.00394	0.47570 ± 0.37394	0.07566 ± 0.05031	23.58000 ± 13.13685

There are many outlier values for the iteration count for each function; table 2 removes outliers and iterations that did not converge to see what our data could have been; if we used a higher max generations or max iterations value.

Comparisons between the initial hyper-parameters convergence rates and the best found parameters convergence rates can be found in tables 3 and 4

Table 2: Non-convergence free Iterations for each function.

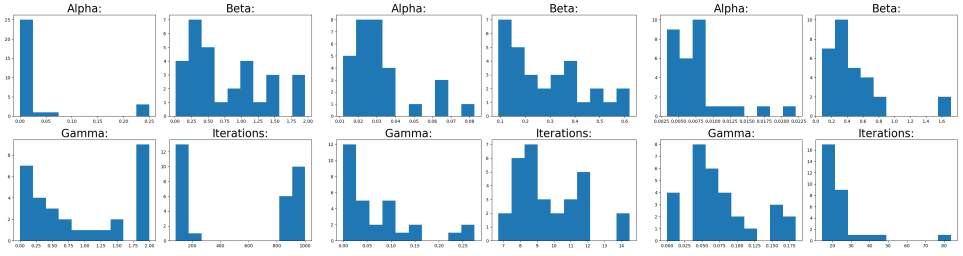
Name	Iterations
Ackley	121.20000 ± 27.65068
Dejong Sphere	9.64667 ± 1.95778
Rastrigin	51.97857 ± 10.49682
Yang Forest	115.40000 ± 46.36618
Zakharov	11.39667 ± 3.88565
Eggcrate	23.58000 ± 13.13685

Table 3: Best Parameters Convergence Rate

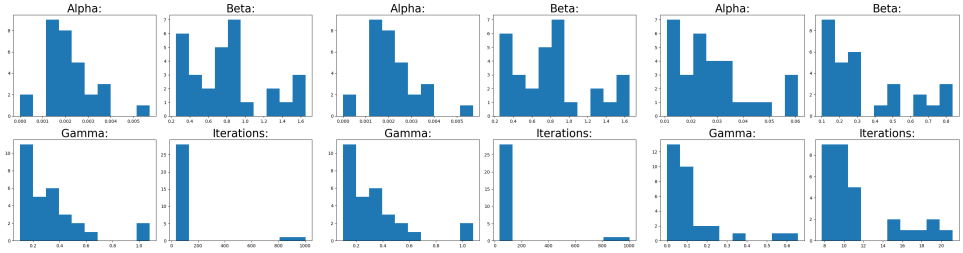
Name	Iterations to Converge	Cost	Average Time for 1 Trial (Seconds)
Ackley	130.36667 ± 63.81143	0.00007 ± 0.00002	0.04355
Zakharov	11.16667 ± 4.67677	0.00004 ± 0.00003	0.00386
Rastrigin	94.53333 ± 80.45526	0.00005 ± 0.00003	0.03136
Yang's Forest	127.93333 ± 99.80078	0.00007 ± 0.00002	0.04162
Eggcrate	25.03333 ± 9.72791	0.00004 ± 0.00003	0.00824
Dejong's Sphere	8.73333 ± 3.38559	0.00005 ± 0.00003	0.00282

Table 4: Initial Parameters Convergence Rate ($\alpha = .25 \quad \beta = 1 \quad \gamma = 2$)

Name	Iterations to Converge	Cost	Average Time for 1 Trial (Seconds)
Ackley	100000 ± 0	0.00130 ± 0.00073	33.55943
Zakharov	410.26667 ± 310.19767	0.00004 ± 0.00003	0.13988
Rastrigin	$63725.60000 \pm 38093.76514$	0.00014 ± 0.00014	21.11155
Yang's Forest	100000 ± 0	0.01616 ± 0.01165	32.57424
Eggcrate	$65843.43333 \pm 43100.34204$	0.00985 ± 0.01869	21.45430
Dejong's Sphere	242.13333 ± 221.38334	0.00005 ± 0.00003	0.07749



(a) Ackley function (b) Dejong's Sphere function (c) Eggcrate function



(d) Rastrigin function (e) Yang Forest function (f) Zakharov function

Figure 1: Hyperparameters distributions of the parent firefly algorithm optimizing parameters of a firefly algorithm over various functions

6 Discussion

For each function, the parent function was given an initial set of hyper-parameters, that being $\alpha = .25 \quad \beta = 1 \quad \gamma = 2$. With only 20 fireflies for the parent function and the child function, the only difference between the two runs was the cost function it was optimizing.

For the purposes of this paper and project, we set the tolerance that each child firefly was optimizing to as 10^{-4}. The parent function optimized the iterations it took to get to that value. Something we expected was for the hyperparameters of each function to become a normal distribution. However, an important thing to note is that due to time constraints, we were only able to run each parent optimization 30 times; while this does meet the criteria of the central limit theorem, the algorithm did not converge every time. This is seen in the charts in figure 1. For example, the Rastrigin function did converge to a very small iteration count most of the time, but two of the 30 runs did not converge in 50 parent tuning generations. This lack of convergence may be due to random chance, specific to the type of function, or the arbitrary limitation of how many generations the parent function and child function had set as their maximum. Regardless of its reason, we would need to run the algorithm significantly more times to get a better idea of the distribution of the hyper-parameters, something which was unfortunately out of the scope of this project due to runtime constraints.

Since the parent function optimizes the child function's average iterations to converge to a given tolerance over a series of runs, if the child function does not converge within that maximum number of iterations, the parent function will be left with each firefly having the exact same value as the starting parameters. Unless at least one child firefly algorithm, on average, can converge without hitting the maximum generations, all fireflies will randomly move as their "costs" are all identical. This is what results in the lack of convergence. When the child algorithm doesn't converge within its maximum iteration count, the algorithm becomes a swarm of random walks. Eventually, we would hope that the random walk would cause a firefly to stumble into better hyper-parameters, and for most functions and most runs, it did. However, there were some exceptions, which resulted in the wide gaps in results we see on some of the functions.

Despite the lack of convergence for some runs, when the algorithms did converge and find better hyper-parameters, the improvements were incredible, especially for functions that are not convex. To analyze the improvements, we can run each algorithm with an incredibly high number of maximum iterations with the initial hyper-parameters and the best-found hyper-parameters. The results for every function can be found in table 3 and table 4. As seen, the Ackley Function and Yang's forest function were unable to converge to the optimal value within a tolerance of 10^{-4} even over 100000 iterations. When we use the best parameters, however, both of these functions converge within 150 iterations on average. This reduces the average time for one trial from nearly 30 seconds down to 50 milliseconds, which is over a 99% decrease. Unlike the Ackley function and Yang's forest function, which have many minima the function can become stuck in, Dejong's Sphere function is convex, there is no local minima to get stuck in, so improvements were less extreme. Despite this, we were still able to decrease the average iterations from 242 iterations to an average of nearly nine iterations.

One result that we found particularly interesting was how small the α parameter was. The α parameter dictates how large a random walk should be; the sizes of these parameters were much lower than we expected. We believe that the optimal value for α is inherently tied with the size of a local minima for a given function as well as the range. For each function we tested, the optimal α value was at least smaller than the $\frac{1}{300}$ bounds we were optimizing within and potentially even as small as $\frac{1}{8000}$. Yang's Forest function, Rastrigin, and Egg crate all had ratios of $\frac{\text{bounds}}{\alpha} > 1000$, and the Ackley function, Zakharov's and Dejong's Sphere function had ratios of $\frac{\text{bounds}}{\alpha} \approx 300$.

The beta values and gamma values both seem to be higher for multi-model functions such as the Ackley function, Rastrigin function, and Yang's forest function. However, the egg-crate function is an exception, with an extremely low Gamma value similar to the Sphere function. The same could be said with the β values. The functions with many local optima also had higher beta values. We believe that the random walk is so small because the values for beta and gamma are so large. Rather than using random walks to get out of minima, beta, and gamma are used to leap out of minima completely and very quickly.

The largest limitation of performing this type of optimization for day-to-day problems is two-fold. The first is that the speed at which the parent functions optimize is extremely slow. One iteration of the parent function could take roughly 30 seconds per generation; for the purposes of this project, we

limited ourselves to 50 generations, 30 trials, and six functions. Overall, the time it took to run just one function ended up being over 5 hours in some cases. To remedy this, we propose transfer learning. We believe using hyper-parameters from similar functions or the real world as initial starting points will help with the speed.

The second problem comes with the problem's definition. Since we are defining the cost function of the parent firefly algorithm as the number of iterations to reach a certain tolerance, we must then also know when we have reached the tolerance to stop the child firefly algorithm. In real-world applications, we will not know when we have reached this value.

Despite those shortcomings, performing the analysis on different types of functions as we have done here allows us to recommend hyper-parameters generically based on potential traits of the function. First, if the function is nearly convex, with small local optima or not many local optima, use smaller values of beta and gamma. Keep β within the range of [.2,.5] and γ within the range of (0,.2]. If the optimization function is messy, with many local optima, the function can get stuck in, use larger β values, [0.5,.8], and use γ values of [.3,.8]. For all types of functions, use an α value that is at least as small as $\frac{1}{300}$ of the total range of the function you are optimizing. If there is bad convergence, the first thing we recommend is lowering the alpha value, as it seems to have the greatest effect on convergence.

Future work may include the classification of function types based on their local optima and the determination of whether the similarities observed here continue to occur. Implementing transfer learning may drastically speed up the process at which these functions converge, and in other papers, non-static parameters have been shown to increase the rate at which these algorithms converge; implementing decay rates for parameters or non-static may be an interesting angle to aim new self-optimizing research towards.

7 Conclusion

To conclude, we were able to successfully reproduce the method of self-tuning optimization using the firefly algorithm outlined in the paper by Xin-She Yang [2]. The biggest drawback after implementation came from the time to execute each function, which, even after applying parallelization where possible, still took much longer than expected. This leaves room for future work in terms of increasing runtime efficiency to make this method of self-optimization more worthwhile from a time perspective. Methods of improvement involve utilizing the GPU for parallel processes and implementing transfer learning to find ideal hyper-parameters faster.

Despite the runtime of the self-tuning taking an incredibly long time, the results speak for themselves. The hyper-parameters reported by the self-tuning are drastically different from the recommended hyper-parameters and can, in extreme cases, increase the runtime of optimization by over 30 times.

Overall, this project gave decent insight into which hyper-parameters are relevant from one function to another and acts as a good foundation if we choose to improve upon this project in the future.

References

[1] Anuradha Ariyaratne, T.G.I. Fernando, and Sunethra Weerakoon. A self-tuning firefly algorithm to tune the parameters of ant colony system (acsfa). *International Journal of Swarm Intelligence*, 3:309 – 331, 04 2018.

[2] Xin-She Yang, Suash Deb, Martin Loomes, and Mehmet Karamanoglu. A framework for self-tuning optimization algorithm. *Neural Computing and Applications*, 23, 12 2013.