Applied Computer Science, vol. 13, no. 1, pp. 61–74
doi: 10.23743/acs-2017-06

Submitted: 2016-12-10
Revised: 2017-01-05
Accepted: 2017-03-01

Fireworks algorithm, Function optimization, Swarm intelligence, Mathematical programming, Natural computing

Evans BAIDOO[*]

FIREWORKS ALGORITHM FOR UNCONSTRAINED FUNCTION OPTIMIZATION PROBLEMS

Abstract

Modern real world science and engineering problems can be classified as multi-objective optimisation problems which demand for expedient and efficient stochastic algorithms to respond to the optimization needs. This paper presents an object-oriented software application that implements a firework optimization algorithm for function optimization problems. The algorithm, a kind of parallel diffuse optimization algorithm is based on the explosive phenomenon of fireworks. The algorithm presented promising results when compared to other population or iterative based meta-heuristic algorithm after it was experimented on five standard benchmark problems. The software application was implemented in Java with interactive interface which allow for easy modification and extended experimentation. Additionally, this paper validates the effect of runtime on the algorithm performance.

1. INTRODUCTION

Function optimisation problems have been solved by many different techniques. Optimization in operation research explains a process of finding the high value of a function in a domain definition which is subject to numerous constraints on the variable values. As a rule of thumb, function optimization problem finds the optimal solution of an objective function definition by means of iteration (Ren & Wu, 2013). Characters of these optimisation problems are mostly identified as linear or non-linear, continuous or discrete, concave or convex functions etc.

[*] Kwame Nkrumah University of Science and Technology, Department of Computer Science, PMB, KNUST, Ghana, Email: ebaidoo2.cos@st.knust.edu.gh

In such instances conversion from constraint function optimization problem into an unconstrained problem is implemented by focusing on the designed special operators and penalty functions so as to enable the feasibility of the solution at all times.

There are a wide range of mathematical programming algorithms which offer various techniques to control various optimization problems as numerical, discrete or combinatorial optimization problems, but most of the methods at most times fail to return satisfactory results. In operations research, as an alternative to the mathematical programming methods, bio-inspired optimization algorithms have become very popular. Over the last decade, researchers are paying attention to nature-inspired heuristics. These algorithms centre on the cooperative intellectual behaviours of animal groups, insects, bee or ant colonies etc and its problem solution abilities. Swarm system as mostly referred have lots of advantages in finding solutions to many optimization problems (Bonabeau, Dorigo & Theraulaz, 1999). As applied in many technical fields, such as data mining, signal processing, network routing, pattern recognition and others, this system is a kind of random search algorithm that simulates the biological population evolution and hence solves complex stochastic optimization problems through cooperation of individuals and species competition (Yuan, de Oca, Birattari & Stutzle, 2012). Typical among the swarm intelligence algorithms are the ant colony optimization (ACO) algorithm (Chandra et al., 2012), the bee colony (ABC) algorithm (Karaboga & Basturk, 2007) particle swarm optimization (PSO) algorithm (Kennedy & Eberhart, 1995), and the genetic algorithm (GA) (Tang, Man, Kwong & He, 1996) enthused by the Darwinian law. Among the listed SI algorithms, PSO is one of the largely accepted algorithms for probing optimal locations in an undefined dimensional space.

A new swarm intelligence algorithm which aroused worldwide concern in 2010 with an excellent optimisation performance was one proposed by Ying Tan. This algorithm which inspired by the emergent swarm behaviour of fireworks is referred to as Fireworks algorithms. This algorithm follows in the tradition of swarm intelligence (Tan & Zhu, 2010).

In this paper an object-oriented implementation of fireworks algorithm is tested on unconstrained function optimization problems. A software program was developed in Java to solve the function optimisation problem, test the results of benchmark functions and also to test the system robustness and performances. The remaining of the paper is organised as follows: In section 2 Fireworks algorithm is introduced followed by section 3 with a brief explanation of five standard unconstrained Benchmark functions. Section 4 details the software implementation of the fireworks algorithm with Section 5 presenting the simulation experiment and results. Finally the paper concludes with Section 6.

2. FIREWORKS ALGORITHM

2.1. Background

A novel swarm intelligence algorithm, Fireworks Algorithm (FA), over the past decade has been implored to solve universal optimisation problems although research of various works according to survey, have shown few implementation. Proposed by Tan and Zhu, this algorithm mimics fireworks explosion activity and behaviour. Its first implementation was to identify its superior performance over Standard PSO and Clonal PSO (Tan & Zhu, 2010). Zheng et al. (2012) put forward a hybrid fireworks-differential evolution (FWA-DE) algorithm. They use the crossover, mutation and selection operators in the Differential evolution algorithm. An enhanced fireworks algorithm (EFWA) presented by Zheng et al. (2013) was used to improve the least radius detection rate, the rules of mapping and spark selection approach of the explode sparks. In 2014, a Hybrid approach which put together FWA and differential mutation (FWA-DM), was formulated. Having successfully experimented on CEC 2014 benchmark functions, it proves to be a good solution. Another proposed solution is the dynamic search firework algorithm (DFWA) (Zheng et al., 2014). This algorithm works by dividing the fireworks population into basic fireworks with optimal fitness value and non-core fireworks. This strategy enables the algorithm to undertake local search and global search efficiently and effectively. As put forward by Ding et al. (2013), the parallelized GPU-based Fireworks Algorithm (GPU-FWA) proficiently exploit graphical processing unit (GPU), to solve large-scale problems. In a research paper by Li et al. (2014), they propose the adaptive firework algorithm (AFWA) to carry out self-tuning of blast radius. They measure the distance involving the best individual and the discussed individual by setting the distance as the next blast radius of the best individual. From this kind of technique, the adaptive step size adjustment shows good optimization performance on the improved FWA.

The focal source of motivation for the FA is the process of starting out a firework. Any time a firework is start out, shower of sparks consume the local space in the region of it. Tan disclose that, the explosion progression of a firework can be analyse as an exploration in the local space about an exact point where the firework left through the sparks created in the explosion (Tan & Zhu, 2010). The explosion of Firework reveals two specific behaviours. A healthy created fireworks, presents numerous sparks and the sparks tends to be engulf in the centre of the explosion. The fireworks is mostly found in areas in the search space which tends to promising and may possibly be close to optimal solution. Therefore it is ideal to produce enough sparks to locate the best spot in the region of the firework. In contrast, poor quality fireworks demonstrate divergent behaviour. Mostly in such instances, sparks generated are relatively few and scattered in the local space. This activity indicates that, the best solution

to the problem is distant away from the spot of the firework and for that matter, the radius of the search could be larger.

2.2. Algorithm attitude

The fireworks algorithm (FA) is a novel iterative-based disperse optimization algorithm approach that imitates the behaviour of fireworks explosion that identifies the equilibrium between global search and local exploration by means of regulating the explosion method of fireworks bombs. The overall algorithm is stochastic in nature. When there is an explosion of fireworks, the sparks produced spread in the air filling their immediate region. For this matter to locate a point x in the specified function f(x) = y, fire explosion can be created in the area which is possible to search continually until sparks locates the point close to x or find the x space. In every generation of explosion, N positions of the fireworks are chosen. Then after N other positions are chosen from the recent sparks and fireworks positions for the next firework explosion. From the results obtained after the explosion, a rough estimation of the position is recorded until the finest position is recorded else fireworks explosions are continued with explosion locations of N selected again. A graphical clarification is expressed in figure 1.

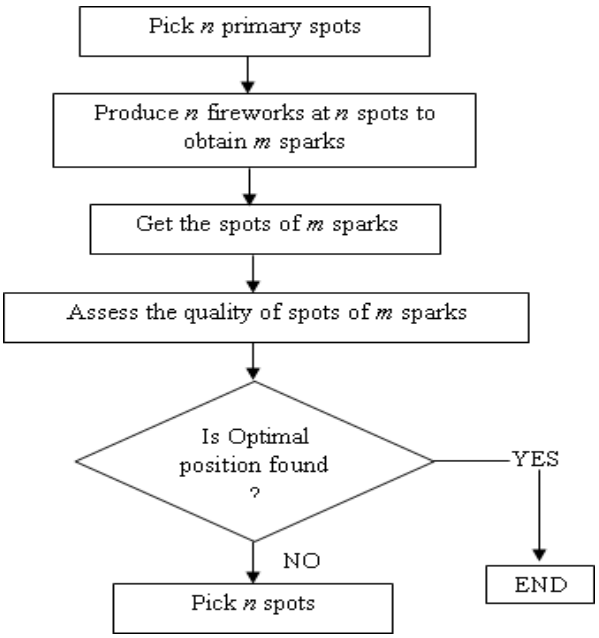

Fig. 1. Flowchart of Fireworks Algorithm

2.3. Algorithm Structure

Two important elements of the FA are the number of sparks and the amplitude of explosion. To illustrate the first element of the algorithm, assuming that FA is intended for some kind of optimisation problem:

$$Minimize\ f(x)\ \in\ \mathbb{R}, x_{min}\ \leq x\ \leq\ x_{max}\ , \tag{1}$$

in this case$x = \ \mathrm{x}_1, \mathrm{x}_2 \ldots, \mathrm{x}_\mathrm{n}$ represent a probable solution, with *f*(*x*) being the objective function and $\mathrm{x}_{\min}$ and $\mathrm{x}_{\max}$ denoting extent of the solution search space. So therefore the number of sparks that can be produced by each single firework x_i can be stated as:

$$S_i = m\ \frac{y_{max} - f(x_i) + \xi}{\sum_{i=1}^{n}(y_{max} - f(x_i)) + \zeta}, \tag{2}$$

Where m symbolize a parameter which take care of the general number of sparks generated by n fireworks, with $\mathrm{y}_{\max} = \max(\ \mathrm{f}(\mathrm{x}_\mathrm{i}))\ (\ \mathrm{i} = 1, 2, \ldots, \mathrm{n})$ being the worst value of the target function amongst the n fireworks, and ξ, is the least computer constant exploited to evade zero-division-error. From experimentations by (Tan & Zhu, 2010), S_i needs to be smaller so as to avoid devastating outcomes under terrible firework explosions, therefore bounds on S_i are created as expressed in 3.

$$\widehat{S_\iota} = \begin{cases} round(a\ \cdot m)\ \ if\ S_i < a \cdot m \\ round(b\ \cdot m)\ \ if\ S_i > b \cdot m \\ round(S_i)\ \ otherwise\ a\ <\ b\ <\ 1, \end{cases} \tag{3}$$

where *a* and *b* represent preset constant parameters. The second important element of FA defines the amplitude of explosion. A well created firework amplitude usually is smaller compared to that of a terrible one. The expression defined in 4 illustrates the amplitude of each explosion.

$$A_i\ =\ \hat{A}\ *\ \frac{f(x_i) - y_{min} + \xi}{\sum_{i=1}^{n}(\ f(x_i)) - y_{min} + \zeta}, \tag{4}$$

where $\hat{A}$ refers to the value of the highest explosion amplitude with $y_{min} = min\ \big(f(x_i)\big)(\ i = 1, 2, \ldots, n)$ representing best firework of the target function in *n* fireworks. In the course of an explosion, the z direction (dimension) of sparks is affected. The number of randomly affected directions is obtained by

$$z = round(d \cdot \chi), \tag{5}$$

where d denotes the optimisation problem number dimension of location x, with χ being a uniform distribution of a random number ranging from 0 and 1. With the aim of determining the x_i firework location of a spark, a spark location x_j is first generated. The entire process is made known in pseudo-code 1.

Pseudo-code 1

Find the initial spark's location: $x_j = x_i$
Choose random z dimensions of x_j by adopting Eq. (5)
Compute the displacement: h $= A_i \cdot \sigma$;
for each chosen dimension x_k^j of x_j **do**
  $x_k^j = x_k^j + h$
  If $x_k^j < x_k^{min}$ *or* $x_k^j > x_k^{max}$
    map x_k^j to the promising space:
    $x_k^j = x_k^{min} + |x_k^j|\%(x_k^{max} - x_k^{min})$;
  end if
end for

From the pseudo-code 1, σ represent a random number of intervals (-1, 1). To maintain the goal of diversity of sparks, a second approach to establishing sparks generation is implemented - Gaussian explosion as illustrated in Pseudo-code 2. A Gaussian (1, 1) function which refers to the Gaussian distribution with a single standard deviation and mean is adapted to describe the explosion coefficient. With this approach mˆ sparks are generated in every Gauss explosion.

Pseudo-code 2

Establish the initial spark's location: $\hat{x}_j = x_i$
Choose arbitrary z dimensions of $\hat{x}_j$ by adopting Eq. (5)
Compute the coefficient of Gauss explosion: g = *Gaussian(1, 1)*
for each chosen dimension $\hat{x}_k^j$ of $\hat{x}_j$ **do**
  $\hat{x}_k^j = \hat{x}_k^j \cdot g$
  If $\hat{x}_k^j < x_k^{min}$ *or* $\hat{x}_k^j > x_k^{max}$ in that case
    map x_k^j to the promising space:
    $\hat{x}_k^j = x_k^{min} + |\hat{x}_k^j|\%(x_k^{max} - x_k^{min})$;
  end if
end for

At the start of every iteration, n fireworks locations are chosen. The finest position x^* according to the best target function $f(x^*)$ is always kept and reassigned for the next iteration. At this point, the selection of *n*–1 locations are

chosen rooted on their distance with other locations to maintain sparks diversity. The measure of distance between a location or spot and other spots/locations is generally determined in FA as:

$$R(x_i) = \sum_{j \in k} d(x_i, x_j) - \sum_{j \in k} \| x_i - x_j \|, \quad (6)$$

where K is the set of all present locations of both fireworks and sparks. At this point the probability of selecting location x_i can be expressed in (7) as:

$$P(x_i) = \frac{R(x_i)}{\sum_{j \in k} R(x_j)}, \quad (7)$$

where $P(x_i)$ indicate the probability that the location x_i will be chosen. Putting all together Pseudo-code 3 illustrates the build up of Fireworks algorithm in high level language.

Pseudo-code 3 in High level language

```
Arbitrarily establish n location of fireworks
iterate until complete
   for every individual firework
    determine the firework amplitude
    determine the total of regular sparks
    Create the regular sparks
   end for
   create special Gauss sparks
   calculate every individual spark from sparks list,
     choose n to serve as new fireworks locations
   generate n new fireworks
end iteration
present the finest spark location found
```

The function estimations of the FA are such that there are about $n + m + \hat{m}$ carried out by each generation. Assuming the optimum of a target function can be identified in generation T, the FA complexity will be $o(n + m + \hat{m})$. A further explanation of the behaviour of the algorithm process is demonstrated in the graph in Figure 2 taken from James McCaffrey (2016).

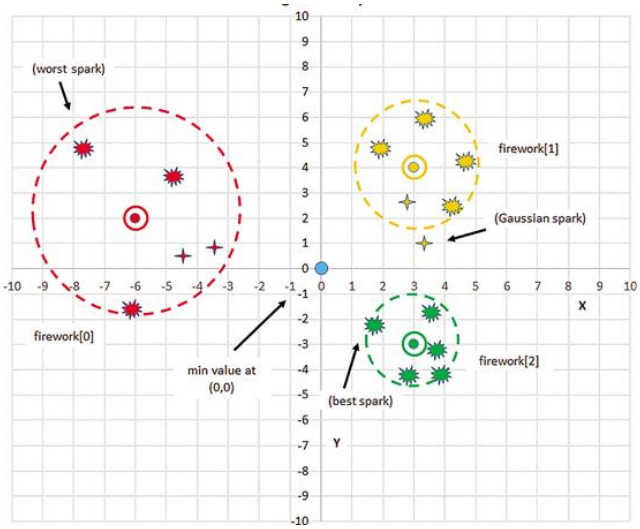

Fig. 1. Fireworks optimization algorithm

In summing up, pseudo-code 1 and 2, presents two sorts of sparks which are respectively generated in each of the iteration. In the first spark kind, the sparks number and explosion amplitude depend to a larger extent the worth of the fireworks. This is sharply in contrast with other sparks which are created using the Gaussian explosion process for firework searching in the local Gauss space. Subsequently, the *n* positions of the succeeding explosion are chosen once the two kinds of spark positions are obtained. Pseudo-code 3 put together the overall structure of FA.

3. BENCHMARK FUNCTIONS

Numerous benchmark functions have been reported in literature over the years yet; there have not been an accepted standard list. This paper experimented five rich set of popular benchmark functions with varied qualities in terms of valley landscape, separability, and modality to assess the character traits of the solution algorithm adopted – in this case check its correctness, toughness and general performance of the implementation program as adopted in Virtual Library of Simulation Experiments: "Test Functions and Datasets" (2016) and Bacanin et al (2014). In this paper the unconstrained function optimization problems used is of the form:

$$\underset{x}{minimize}\ f(x)$$

Subject to: $x \in \omega \leq 0, i = 1 \ldots m$

$x_j^l \leq x_j \leq x_j^u$, j = 1...n

Where, $f(x)$ represent the objective function to be minimized, with x being the continuous vector variable of domain $\omega \subset R^n$ and $f(x): \omega \rightarrow R$ representing a continuous real-valued function. The lower and upper bounds defined within each function dimension is represented by ω.

The first is the Dixon-Price test function, f_1. This function is a continuous, non-separable and multimodal minimization test operation. It search domain lies in $-10 \leq x_i \leq 10, i = 1, 2, \ldots, n$ with its global minimum, f(x) at 0.

Griewank function, f_2 which has its global minimum value at 0 with the function initialization range from [−600,600], is second. It is a continuous and differentiable function which has a corresponding universal optimum solution as $x_{opt} = (x_1, x_2, \ldots, x_n) = (100, 100, \ldots, 100)$. This function although multimodal, its multimodality diminishes with high dimensionalities (n > 30) and therefore appears uni-modal.

Rosenbrock, f_3 makes the list as third. It is a well-known traditional optimization problem with a 2 dimensional function which describe a deep valley with a parabola form of the shape $x_1^2 = x_2$ that results to the global minimum. Owing to the non-linearity of the valley, lots of algorithms converge slowly since they vary the direction of the search constantly and for this reason this problem has been repetitively used in assessing gradient-based optimization algorithms performance. Valley function is unimodal with the initialization interval of X [−30, 30].

The fourth test function is the Schwefel function, f_4. This function is complex, with many local minima. Initialization range for the function is [−500, 500]. The surface of Schwefel function is made up of a large amount of peaks and valleys. It is a deceptive function which possesses two global minimum with its global minimum over the parameter space from the succeeding best local minima geometrically far-off. Thus its search algorithm is ably susceptible to converging in a wrong direction. It has its global minima $x^* at = = \pm[\pi(0.5 + k)]^2, f_4(x^*) = -418.983$. The difficulty with this test function is that its gradient cannot bend along their axis owing to the epitasis with their variables. For this reason, most algorithms that make use of the gradient leisurely converge.

The Sphere concludes the benchmark functions. This function has the properties of being separable, scalable, continuous and multimodal. Its interval range lies in the region of $0 \leq x_i \leq 10$ with its universal minimal value at 0 and optimum solution being $x_{opt} = (x_1, x_2, \ldots, x_n) = (0, 0, \ldots, 0)$.

The functions expression, its initialization and intervals are further expressed in Table 1.

Tab. 1. Benchmark functions expression and initialisation

Function	Expression	Initialisation
Dixon-Price	$f_1(x) = (x_1 - 1)^2 + \sum_{i=2}^{n} i(2x_i^2 - x_{i-1})^2$	[–10, 10]
Griewank	$f_2(x) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i-1}^{n} cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	[–100, 100]
Rosenbrock	$f_3(x) = \sum_{i=1}^{n-1} [100.(x_{i+1} - x^2{}_i)^2 + (x_i - 1)^2]$	[–30, 30]
Schwefel	$f_4(x) = \sum_{i=1}^{n} -x_i \sin(\sqrt{\lvert x_i \rvert}$	[–500, 500]
Sphere	$f_5(x) = \sum_{i=1}^{n} x_i^2$	[0, 10]

4. eFIREWORKS IMPLEMENTATION

This paper implements a software program which uses an object oriented approach to its execution of the fireworks algorithm. The developed software is named eFireworks (explosive Fireworks). When an objected oriented approach is adopted there is improvement in the software scalability and therefore there is lesser time execution even when there is an implementation of different optimization problems of new programming concept. Object oriented capabilities allow for addition of varying optimization functions and lots more benchmark problems with little or no effort. Additionally, object oriented implementation allows for easy modification of the behaviour of algorithm functions. Similar works have been suggested for artificial fish swarm algorithm in James McCaffrey (2016) and ABC in Virtual Library of Simulation Experiments: "Test Functions and Datasets" (2016). The developed software with its graphical user interface is created in Java programming language. Java was chosen partly due to the fact that its paradigm entirely imitates object-oriented and the ability that it is based on classes, concurrent and is general purposed. Additionally, its implementation dependencies can be as few as possible. An added advantage that java has is its ability to allow application developers to "write once, run anywhere." This consequently makes most Java application convenient and portable thus allowing software – eFireworks – to be executed on varying computer system. The software adopts a graphical user interface. The graphical user interface among other qualities enables easier control of parameters and test function to optimize.

The software application is deeply adopts the theory of tightly connected abstract classes and inheritance. This concept enables easy adaptation to new functional problems and the future extendibility of the program. eFireworks is developed using java 1.7.0.25, NetBeans 7.1. 2 with Windows 8 operating system of x64 bit. Figure 2 illustrate a Screenshot of the vital Graphical user interface (GUI) of eFireworks.

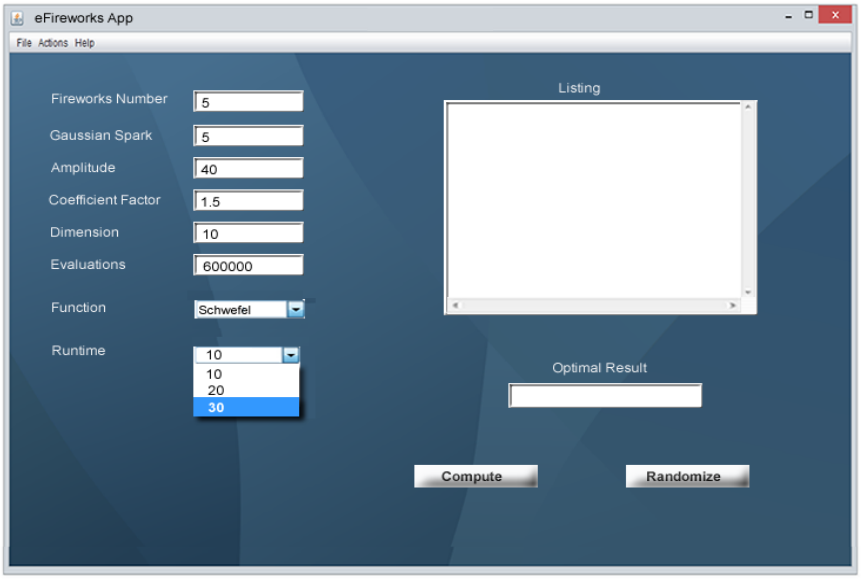

Fig. 2. Screenshot of eFireworks User interface

5. OPTIMIZATION TEST AND RESULTS

All tests were executed on intel(R) Core (TM_i3-3110M CPU@ 2.40GHz laptop equipped with 4GB RAM on Windows 8 x64 Operating System in NetBeans 7.1.2 IDE. The NetBeans IDE and operating system were the only processes running during test execution. To validate the algorithm and examine runtime effects on the algorithm performance, two sets of test was conducted on each of the five benchmark functions. 10 runs was set for the first execution and 30 runs for the second, each using different random seeds. The best, statistical mean and standard deviation results were the indicators that were observed from the problem solution. The bounds used for the upper and lower parameters were the default values from Table 1.The benchmark dimensionality for all functions is set to 10 (D = 10) with the maximum number of function evaluation perked at 600,000 for all functions. The parameter settings of FA used are same as in Tan and Zhu (2010): number of fireworks, n = 5,

total number of regular sparks, m = 50, special Gaussian spark, $\widehat{m}$ = 5, amplitude, A = 40, maximum spark of firework, b = 0.8 and minimum spark of firework, a = 0.04. Results of test values are shown in Table 2 for runtime of 10 and Table 3 for runtime of 30.

Tab. 2. Experimental results of 10 runs

Function	Best cost function	Statistical Mean	Standard Deviation
Dixon-Price	**4.7E-10**	2.01E-01	0.321355
Griewank	**0.00172**	0.0273431	0.017826912
Rosenbrock	**0.02364**	0.19397877	0.087896961
Schwefel	**-4489.82887**	-4309.82887	103.2795559
Sphere	**2.048E-09**	0.025760842	0.081249567

As can be identified from Table 2 and 3, the optimization obtained with Fireworks algorithm presents satisfactory results for all under listed benchmark problems. Comparison can be drawn with other famous heuristic or bio-inspired algorithms and software systems such as Karaboga and Basturk (2007) & Bacanin (2014). The algorithm proves to be robust in its operations and presents optimal results.

Tab. 3. Experimental results of 10 runs

Function	Best cost function	Statistical Mean	Standard Deviation
Dixon-Price	**3.66E-10**	0.135119376	0.27035624
Griewank	**0.0001125**	0.019550378	0.022130294
Rosenbrock	**0.010225**	0.103153984	0.107639116
Schwefel	**-5389.82887**	-4547.919763	369.4342795
Sphere	**0**	0.008030625	0.043813971

Drawing comparison from Table 2 and Table 3 it can be deduce that as the number of runs are increased, results obtained tends to be slightly better. It can therefore be concluded that the performance of the optimization solution, Fireworks algorithm is marginally affected by alteration of number of runs although it may be disregarded since the results presented is of tiny deviation.

6. CONCLUSIONS

In this paper, an implementation of a population based swarm intelligence algorithm, Fireworks algorithm was adopted to solve unconstrained function optimization problems using a developed object-oriented graphical user interface application. The application presents a much more expedient way to work and allows for easy modification for other optimization problems as compared to the original version. The algorithm was tested on five benchmark functions and performance recorded. The optimization solution algorithm demonstrated its ability to contain several benchmark problems but there is some margin of improvement. In future, the study can be extended to investigate and study other global and real life problems such as urban traffic regulation, job scheduling, parcel delivery etc.

REFERENCES

Bacanin, N., Tuba, M., & Stanarevic, N. (2012). Artificial Fish Swarm Algorithm for Unconstrained Optimization Problems. *Applied Mathematics in Electrical and Computer Engineering*, 405–410.

Bonabeau, E., Dorigo, M., & Theraulaz, G. (1999). *Swarm Intelligence: From Natural to Artificial Systems*. New York: Oxford University Press Inc.

Ding, K., Zheng, S. Q., & Tan, Y. (2013). A GPU-based Parallel Fireworks Algorithm for Optimization. *Gecco'13: Proceedings of the 2013 Genetic and Evolutionary Computation Conference*, 9–16.

Karaboga, D., & Basturk, B. (2007). A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *Journal of Global Optimization, 39*(3), 459-471. doi:10.1007/s10898-007-9149-x

Kennedy, J., Eberhart, R. C. (1995). Particle swarm optimization. *Proceedings of IEEE International Conference on Neural Networks, 4*, 1942–1948.

Li, J., Zheng, S., & Tan, Y. (2014). Adaptive Fireworks Algorithm. *2014 IEEE Congress on Evolutionary Computation (CEC)*, 3214–3221. doi:10.1109/CEC.2014.6900418

McCaffrey, J. (2016, September). *Fireworks Algorithm Optimization*. Retrieved from https://msdn.microsoft.com/en-us/magazine/dn857364.aspx

Mohan, B. C., & Baskaran, R. (2012). A survey: Ant Colony Optimization based recent research and implementation on several engineering domain. *Expert Systems with Applications, 39*(4), 4618-4627. doi:10.1016/j.eswa.2011.09.076

Ren, Y., & Wu, Y. (2013). An efficient algorithm for high-dimensional function optimization. *Soft Computing, 17*, 995-1004. doi:10.1007/s00500-013-0984-z

Tan, Y., & Zhu, Y. (2010). Fireworks Algorithm for Optimization. In: Y. Tan, Y. Shi, & K.C. Tan (Eds.), *Advances in Swarm Intelligence. ICSI 2010. Lecture Notes in Computer Science* (vol. 6145, pp. 355–364). Springer.

Tang, K. S., Man, K. F., Kwong, S., & He, Q. (1996). Genetic algorithms and their applications. *IEEE Signal Processing Magazine, 13*(6), 22-37. doi:10.1109/79.543973

Virtual Library of Simulation Experiments: Test Functions and Datasets (n.d.). Retrieved August, 2016, from https://www.sfu.ca/~ssurjano/optimization.html

Yuan, Z., de Oca, M. A. M., Birattari, M., & Stutzle, T. (2012). Continuous optimization algorithms for tuning real and integer parameters of swarm intelligence algorithms. *Swarm Intelligence, 6*(1), 49–75. doi:10.1007/s11721-011-0065-9

Zheng, S. Q., Janecek, A., Li, J. Z., & Tan, Y. (2014). Dynamic Search in Fireworks Algorithm. *2014 IEEE Congress on Evolutionary Computation (Cec)*, 3222–3229.

Zheng, S., Janecek, A., & Tan, Y. (2013). Enhanced Fireworks Algorithm. *2013 IEEE Congress on Evolutionary Computation*, 2069-2077. doi:10.1109/CEC.2013.6557813

Zheng, Y. J., Xu, X. L., & Ling, H. F. (2012). A hybrid fireworks optimization method with differential evolution operators. *Neurocomputing*, *148*, 75–80. doi:10.1016/j.neucom.2012.08.075