2013 IEEE Congress on Evolutionary Computation
June 20-23, Cancún, México

# Enhanced Fireworks Algorithm

Shaoqiu Zheng†, Andreas Janecek‡ and Ying Tan†
† Department of Machine Intelligence, School of Electronics Engineering and Computer Science, Peking University
Key Laboratory of Machine Perception (Ministry of Education), Peking University, Beijing,100871, P.R. China
Email: zhengshaoqiu@pku.edu.cn, ytan@pku.edu.cn
‡ University of Vienna, Research Group Entertainment Computing, 1090 Vienna, Austria
Email: andreas.janecek@univie.ac.at

***Abstract*—In this paper, we present an improved version of the recently developed Fireworks Algorithm (FWA) based on several modifications. A comprehensive study on the operators of conventional FWA revealed that the algorithm works surprisingly well on benchmark functions which have their optimum at the origin of the search space. However, when being applied on shifted functions, the quality of the results of conventional FWA deteriorates severely and worsens with increasing shift values, *i.e.,* with increasing distance between function optimum and origin of the search space. Moreover, compared to other meta-heuristic optimization algorithms, FWA has high computational cost per iteration. In order to tackle these limitations, we present five major improvements of FWA: ($i$) a new minimal explosion amplitude check, ($ii$) a new operator for generating explosion sparks, ($iii$) a new mapping strategy for sparks which are out of the search space, ($iv$) a new operator for generating Gaussian sparks, and ($v$) a new operator for selecting the population for the next iteration. The resulting algorithm is called *Enhanced Fireworks Algorithm (EFWA)*. Experimental evaluation on twelve benchmark functions with different shift values shows that EFWA outperforms conventional FWA in terms of convergence capabilities, while reducing the runtime significantly.**

## I. Introduction

The Fireworks Algorithm (FWA) [1] is a recently developed swarm intelligence algorithm based on simulating the explosion process of fireworks. In analogy with real fireworks exploding and illuminating the night sky, the fireworks (i.e, individuals) in FWA are let off to the potential search space. For each firework, an explosion process is initiated and a shower of sparks fills the local space around it. Fireworks as well as the newly generated sparks represent potential solutions in the search space. Similar to other optimization algorithms, the goal is to find a "good" (ideally the global) solution of an optimization problem with bound constraints in the form $\min_{x\in\Omega} f(x)$, where $f : \mathbb{R}^N \rightarrow \mathbb{R}$ is a nonlinear function, and $\Omega$ is the feasible region. FWA presents a new search manner which searches the potential space by a stochastic explosion process within a local space. A principle FWA works as follows: At first, $N$ fireworks are initialized randomly, and their quality (*i.e.,* fitness) is evaluated in order to determine the explosion amplitude and the number of sparks for each firework. Subsequently, the fireworks explode and generate different types of sparks within their local space. Finally, $N$ candidate fireworks are selected among the set of candidates, which includes the newly generated sparks as well as the $N$ original fireworks.

In order to ensure diversity and balance the global and local search, the explosion amplitude and the population of the newly generated explosion sparks differ among the fireworks. A firework with *better* fitness can generate a *larger population* of explosion sparks within a *smaller range*, *i.e.,* with a small explosion amplitude. Contrary, fireworks with lower fitness can only generate a smaller population within a larger range, *i.e.,* with higher explosion amplitude. This technique allows to balance between exploration and exploitation capabilities of the algorithm. While *exploration* refers to the ability of the algorithm to explore various regions of the search space in order to locate promising good solutions, *exploitation* refers to the ability to conduct a thorough search within a smaller area recognized as promising in order to find the optimal solution (cf. [2]). Exploration is achieved by those fireworks which have a large explosion amplitude (*i.e.,* lower fitness), since they have the capability to escape from local minima. Exploitation is achieved by those fireworks which have a small explosion amplitude (i.e, high fitness), since they reinforce the local search ability in promising areas. After the explosion, another type of sparks are generated based on a Gaussian mutation of randomly selected fireworks. The idea behind this is to further ensure diversity of the swarm. In order to improve readability we assign new notations to the two distinct types of sparks: "explosion sparks" are generated by the explosion process, and "Gaussian sparks" are generated by Gaussian mutation.

**Contributions.** In the original FWA paper [1], it has been shown that FWA works very well on functions which have their optimum at the origin. For all functions presented in [1], FWA achieved better fitness than Standard PSO and Clonal PSO [3], respectively, with a significantly smaller number of function evaluations. However, we found that FWA has problems when being applied on shifted functions, *i.e.,* functions which do not have their optimum at the origin. In fact, with increasing distance between optimum and origin of the function, the results of FWA decline. Until now, it is unclear which operators of FWA are responsible for this loss in accuracy. Thus, it is important to improve the current operators of FWA in order to make the algorithm applicable on different functions, which may be shifted and/or rotated. Another problem of FWA is the high computational cost per iteration. This is mostly caused by the operator which selects the individuals for the next generation. The goal of this paper

978-1-4799-0454-9/13/$31.00 ©2013 IEEE 

is to improve the operators of FWA and to analyze to which extent each of these operators is responsible for its behavior. We present improvements and/or alternatives of these operators and propose a new algorithm called Enhanced FWA (EFWA). Compared to FWA, EFWA achieves stable results for shifted and non-shifted functions, and shows improvements over FWA in terms of convergence capabilities and computational cost.

**Related work.** So far, FWA has been applied for solving practical optimization problems [4]–[7], combined with other optimization algorithms [8], [9], and improved in another study [10]. Janecek and Tan [4]–[6] used FWA together with Particle Swarm Optimization (PSO), Genetic Algorithms (GA), Differential Evolution (DE), and Fish School Search for improving the initialization of Non-negative Matrix Factorization (NMF). Their results indicate that FWA could not compete with the other optimization algorithms when the number of dimensions (*i.e.,* the rank of NMF) was small, but achieved good results when the number of dimensions was increased. Bureerat [7] compared twelve different optimization algorithms on 35 benchmark functions with different dimensions ranging from 2 to 30. FWA was ranked as the 6-th best algorithm for optimizing these benchmark functions, which is better than GA and PSO. Gao and Diao [8] proposed the Cultural Firework Algorithm (CFA) which combines ideas from Cultural Algorithms (CAs) and FWA. CFA acquires problem-solving knowledge (beliefs) from the explosion of fireworks and in return makes use of that knowledge to better guide the search (cf. [8]). Results indicate that CFA is well suited for optimizing FIR and IIR digital filters, and outperforms various PSO variants for this type of optimization problem. Zheng *et al.* [9] proposed a hybrid algorithm between FWA and DE by including operators from DE into FWA. The results indicate that the hybrid algorithm outperforms both, the conventional FWA and the conventional DE. However, the experiments were only conducted on functions which have their optimum at the origin and hence show similar effects as conventional FWA.

So far, one study [10] has focused on improving the conventional FWA by investigating the influence of approximation approaches for accelerating the FWA search by elite strategy. The authors compared different approximation and sampling methods, and also different sampling numbers, and analyzed the acceleration performance of FWA based on ten benchmark functions. Their results indicate that the random sampling method with a two degree polynomial model gains the best performance. We point out that different sampling methods are able to improve and speed up the FWA. However, other and probably more important operators of FWA have not been analyzed nor improved in this study.

**Synopsis.** We present an overview of conventional FWA in Section II, and discuss its limitations in Section III. In Section IV we propose the new EFWA algorithm, which is based on several improvements and/or alternatives of/to the FWA operators. Our experiments are evaluated and discussed in Section V. Finally, we conclude the paper and give directions for future work in Section VI.

## II. The Conventional FWA Algorithm

As already mentioned, FWA incorporates an automatic procedure to balance exploration and exploitation capabilities. Fireworks with better fitness will have a smaller explosion amplitude and a larger number of explosion sparks than firework with lower fitness. Assume that the number of fireworks is $N$ and the number of dimensions is $d$, then the explosion amplitude $A$ (Eq. 1) and the number of explosion sparks $s$ (Eq. 2) for each firework $X_i$ are calculated as follows:

$$A_i = \hat{A} \cdot \frac{f(X_i) - y_{min} + \varepsilon}{\sum_{i=1}^{N} (f(X_i) - y_{min}) + \varepsilon} \tag{1}$$

$$s_i = M_e \cdot \frac{y_{max} - f(X_i) + \varepsilon}{\sum_{i=1}^{N} (y_{max} - f(X_i)) + \varepsilon} \tag{2}$$

where $y_{max} = max(f(X_i))$, $y_{min} = min(f(X_i))$, $\hat{A}$ and $M_e$ are two constants to control the explosion amplitude and the number of explosion sparks, respectively, and $\varepsilon$ is the machine epsilon. To avoid the overwhelming effects of fireworks at good locations, the number of sparks is bounded by

$$s_i = \begin{cases} round(aM_e) & \text{if } s_i < aM_e, \\ round(bM_e) & \text{if } s_i > bM_e, \\ round(s_i) & otherwise. \end{cases} \tag{3}$$

where $a$ and $b$ are constant parameters that confine the range of the population size. Based on $A_i$ and $s_i$, the explosion operator is performed (cf. Alg. 1). For each of the $s_i$ explosion sparks of each firework $X_i$, Algorithm 1 is performed once. In Line 7 of Algorithm 1, the operator % refers to the modulo operation (remainder of division), and $X^k_{min}$ and $X^k_{max}$ refer to the lower and upper bounds of the search space in dimension $k$.

**Algorithm 1** – Generating "explosion sparks" in FWA

1: Initialize the location of the "explosion sparks": $\hat{X}_i = X_i$
2: Calculate offset displacement: $\triangle X = A_i \times rand(-1, 1)$
3: Set $z^k = round(rand(0, 1))$, $k = 1, 2, ..., d$
4: **for** each dimension of $\hat{X}^k_i$, where $z^k == 1$ **do**
5: $\quad \hat{X}^k_i = \hat{X}^k_i + \triangle X$
6: $\quad$ **if** $\hat{X}^k_i$ out of bounds **then**
7: $\quad\quad \hat{X}^k_i = X^k_{min} + |\hat{X}^k_i| \ \% \ (X^k_{max} - X^k_{min})$
8: $\quad$ **end if**
9: **end for**

After the explosion, another type of sparks, the *Gaussian sparks*, are generated based on a Gaussian mutation process (cf. Algorithm 2). This algorithm is performed $M_g$ times, each time with a randomly selected firework $X_i$ ($M_g$ is a constant to control the number of Gaussian sparks).

In order to retain the information and pass it to the next generation, a new population of fireworks is selected at the end of each iteration. All original fireworks, as well as all explosion and Gaussian sparks can be selected for the next iteration (in total, $N$ fireworks/sparks are selected). The current best location is always kept for the next iterations. In order to improve the diversity, the remaining $N - 1$ locations are

**Algorithm 2** – Generating "Gaussian sparks" in FWA

Initialize the location of the "Gaussian sparks": $\tilde{X}_i = X_i$
Calculate offset displacement: $e = Gaussian(1, 1)$
Set $z^k = round(rand(0, 1))$, $k = 1, 2, ..., d$
**for** each dimension of $\hat{X}_i^k$, where $z^k == 1$ **do**
  $\tilde{X}_i^k = \tilde{X}_i^k \times e$
  **if** $\tilde{X}_i^k$ out of bounds **then**
    $\tilde{X}_i^k = X_{min}^k + |\tilde{X}_i^k| \ \% \ (X_{max}^k - X_{min}^k)$
  **end if**
**end for**

selected based on a distance based selection operator (cf. [11]). For location $X_i$, the selection probability $p_i$ is calculated by:

$$p(X_i) = \frac{R(X_i)}{\sum_{j \in K} R(X_j)} \tag{4}$$

$$R(X_i) = \sum_{j \in K} d(X_i, X_j) = \sum_{j \in K} ||X_i - X_j|| \tag{5}$$

where $K$ is the set of all current locations including original fireworks and both types of sparks (without the best location). As a result, fireworks or sparks in low crowded regions will have a higher probability to be selected for the next iteration than fireworks or sparks in crowded regions.

## III. Properties of Conventional FWA

As already mentioned in the original FWA paper [1], FWA outperformed SPSO and CPSO significantly and converged in most cases towards the function optimum already after a few iterations. However, when applying FWA on shifted functions the results worsen progressively with increasing distance between function optimum and origin of the search space. By investigating the operators of FWA, we found that some of them create [map] sparks at [to] locations which are close to the origin of the search space, independent of the function optimum. This behavior is mostly caused by the mapping operator and the Gaussian sparks operator. In terms of runtime we found that the cost per iteration of FWA is significantly higher than for most other optimization algorithms. In Section IV we analyze all operators of FWA in detail and point out which operators are responsible for its actual behavior and its high computational cost. In summary, conventional FWA has the following drawbacks:

- $(i)$ For functions which have their optimum at the origin, FWA will find the optimal solution very fast. However, not due to the intelligence of the algorithm but due to the specific mapping and Gaussian mutation operators which map/create sparks close to the origin.
- $(ii)$ For functions which have their optimum far away from the origin, FWA has to face the two drawbacks that the mapping operator rebounds most solutions which are out of the search space to locations which are far away from the function optimum, and that the mutation operator creates many sparks at locations close to the origin (*i.e.,* again far away from the optimum).
- $(iii)$ FWA has a high computational cost per iteration.

## IV. The Proposed EFWA

In this section we present the new operators of EFWA in detail. We point out the limitations of each operator in conventional FWA and discuss the applied changes and novelties.

### A. A new Minimal Explosion Amplitude Check

Eq. 1 shows how the explosion amplitude for each firework is calculated in conventional FWA. A firework with better fitness will have a smaller explosion amplitude while a firework with lower fitness has a larger explosion amplitude. Although this idea seems reasonable, the explosion amplitude of the fireworks having the best (or a very good) fitness will usually be very small, *i.e.,* close to 0. If the explosion amplitude is [close to] zero, the explosion sparks will be located at [almost] the same location as the firework itself. As a result, it may happen that the location of the best firework cannot be improved until another firework has found a better location. In order to avoid this problem, we introduce a lower bound $A_{\min}$ of the explosion amplitude, which is based on the progress of the algorithm. During the early phase of the search, $A_{\min}$ is set to a higher value in order to facilitate exploration, with increasing number of evaluations, $A_{\min}$ is decreased in order to allow for better exploitation capabilities around good locations. For each dimension $k$, the explosion amplitude $A_i^k$ is bound as follows:

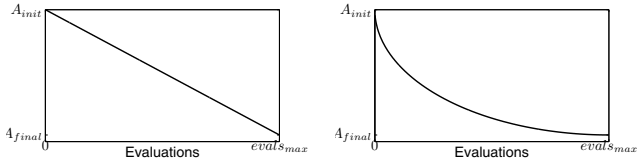
$$A_i^k = \begin{cases} A_{\min}^k & \text{if } A_i^k < A_{\min}^k, \\ A_i^k & otherwise. \end{cases} \tag{6}$$

A new value for $A_{\min}$ is calculated in each iteration. In this work, we use two different modes to calculate $A_{\min}$. The first approach is based on a linearly decreasing function (cf. Eq. 7), and the other approach is based on a non-linearly decreasing function (cf. Eq. 8).

$$A_{\min}^k(t) = A_{init} - \frac{A_{init} - A_{final}}{evals_{max}} * t \tag{7}$$

$$A_{\min}^k(t) = A_{init} - \frac{A_{init} - A_{final}}{evals_{max}} \sqrt{(2 * evals_{max} - t)t} \tag{8}$$

In both equations, $t$ refers the number of function evaluation at the beginning of the current iteration, and $evals_{max}$ is the maximum number of evaluations. $A_{init}$ and $A_{final}$ are the initial and final minimum explosion amplitude, respectively. Compared to the linear decrease of $A_{min}$, the non-linear decrease enhances exploitation already at an earlier stage of the algorithm (*i.e.,* after fewer iterations). Figure 1 shows a graphical representation of Eq. 7 and 8.




(a) Linear decrease (b) Non-linear decrease

Fig. 1. Linearly and non-linearly decreasing minimal explosion amplitude

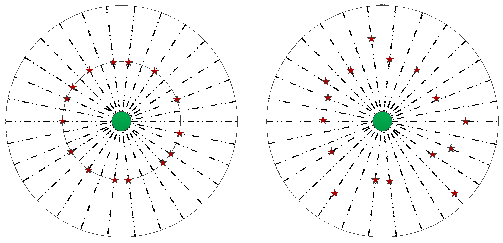
### B. A new Operator for Generating Explosion Sparks

Lines 2 and 5 of Algorithm 1 show how the offset displacement $\triangle X$ is added to the current location. As can be seen, the offset displacement is only calculated once (Line 2) and the same value is added to the location of selected dimensions. Obviously, adding the same value in each dimension leads to a bad local search ability. To avoid this problem, we calculate a different offset displacement for selected dimensions (those dimensions, where $z^k$ equals 1). Algorithm 3 shows the proposed process of generating explosion sparks in EFWA.

**Algorithm 3** – Generating "explosion sparks" in EFWA

1: Initialize the location of the "explosion sparks": $\hat{X}_i = X_i$
2: Set $z^k = round(rand(0,1))$, $k = 1, 2, ..., d$.
3: **for** each dimension of $\hat{X}_i^k$, where $z^k == 1$ **do**
4: Calculate offset displacement: $\triangle X^k = A_i \times rand(-1, 1)$
5: $\hat{X}_i^k = \hat{X}_i^k + \triangle X^k$
6: **if** $\hat{X}_i^k$ out of bounds **then**
7: map $\hat{X}_i^k$ to the potential space (see next subsection)
8: **end if**
9: **end for**

Figure 2 shows the difference between the generation of explosion sparks in FWA (cf. Algorithm 1), and EFWA (cf. Algorithm 3). As discussed before, the offset displacement in FWA is similar for all selected dimensions, in EFWA a different offset displacement is calculated in each dimension.



(a) FWA (b) EFWA

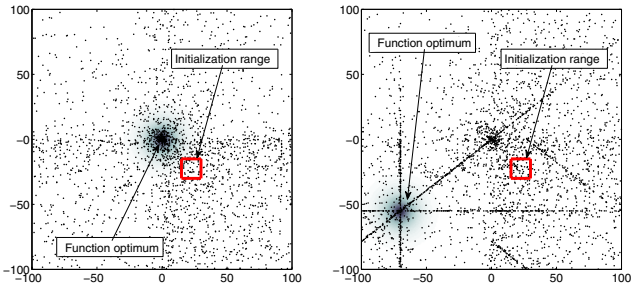
Fig. 2. Generation of explosion sparks in FWA and EFWA

### C. A new Mapping Operator

In conventional FWA, when the location of a new spark exceeds the search range in dimension $k$, the new spark will be mapped to another location according to $\bar{X}_i^k = X_{min}^k + |X_i^k| \,\%\, (X_{max}^k - X_{min}^k)$ (cf. Algorithms 1 and 2). In many cases, a spark will exceed the allowed search space only by a rather small value. Moreover, as the search space is often equally distributed ($X_{min}^k \equiv -X_{max}^k$), the adjusted position $\bar{X}_i^k$ will be very close to the origin in many cases. The following example is used to explain this comment: Consider an optimization problem within the search space $[-20, 20]$. If, in dimension $k$, a new spark is created at the point $X^k = 21$, it will be mapped to the location $\bar{X}_i^k = -20 + |21| \,\%\, (40)$. Since the result of the modulo operation $21 \,\%\, (40) = 21$, $X^k$ will be mapped to the location $\bar{X}_i^k = 1$, which is already very close to the origin. In cases where $X_{min} \equiv -X_{max}$, this mapping operator is partly responsible for drawbacks $(i)$ and $(ii)$ as mentioned in Section III. In order to avoid the problems caused by the conventional mapping operator we replace this method with a uniform random mapping operator $\bar{X}_i^k = X_{min}^k + rand * (X_{max}^k - X_{min}^k)$, which maps the sparks to any location in the search space with uniform distribution.

### D. A new Operator for Generating Gaussian Sparks

Together with the mapping operator, the Gaussian mutation operator is the main reason why conventional FWA works significantly better than other optimization algorithms for functions which have their optimum at the origin (cf. the results in [1]). Figures 3(a) and 3(b) show the location of the Gaussian sparks for a two-dimensional Ackley function with the optimum at [0, 0] and [-70, -55], respectively. In each iteration, the location of (only) the Gaussian sparks is plotted and not deleted. The location of the Gaussian sparks in Figure 3(a) indicates that most sparks are located at the origin, *i.e.,* close to the optimum. Moreover, it can be seen that the areas close to the coordinate axes are also more crowded than other parts of the search space. Figure 3(b) reveals an interesting fact about the location of the Gaussian sparks for the shifted Ackley function. Even though the optimum is now far away from the origin, many sparks are located close to the center. Obviously, many sparks in Figure 3(a) are not located close to the center because of the intelligence of the algorithm, but rather because many Gaussian sparks are created near to the origin of the search space, independent of the location of the function optimum. We point out that Figures 3(a) and 3(b) were created using the mapping operator of conventional FWA.



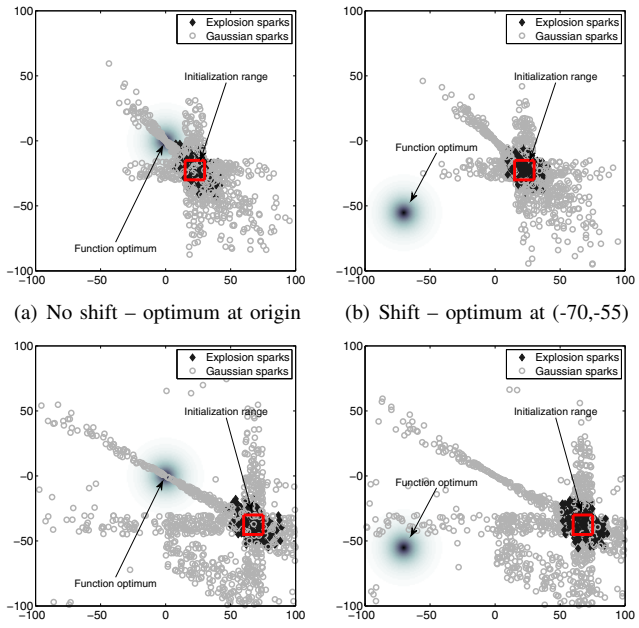

(a) No shift – optimum at origin (b) Shift – optimum at (-70,-55)

Fig. 3. The locations of the Gaussian sparks using the conventional FWA (Ackley function using 100 000 function evaluations)

The reason for this behavior is the calculation of the Gaussian sparks as shown in Lines 2 and 5 of Algorithm 2. In Line 2, $e$ is set to a random value from a normal distribution with expected value and variance both set to 1. In cases where $e$ is close to 0, $\hat{X}_i^k$ will be close to 0 as well (Line 5). As a result, many Gaussian sparks will be located close to the origin of the search space in dimension $k$. Moreover, for large $e$, many Gaussian sparks are created at locations which are out of bounds. In this case, the mapping operator of conventional

FWA (cf. Section IV-C) will map the newly created spark to a location which is in many cases close to the origin. Another problem of the conventional Gaussian sparks operator is the fact that fireworks which are already located close to the origin of the search space cannot escape from this location; if firework $X_i^k$ is close to zero, the location of spark $\tilde{X}_i^k$ be close to zero as well, since $\tilde{X}_i^k = X_i^k \times e$.

**Initialization.** Since conventional FWA is able to converge towards the optimum already after very few function evaluations (we again refer to the result in [1]), we also analyzed the behavior of FWA during the first iteration. Figures 4(a) to 4(d) show the distribution of the explosion and Gaussian sparks, respectively, directly after the initialization with different initialization ranges. We repeatedly initialized the fireworks, created the two types of sparks, and plotted their locations until we reached 5 000 function evaluations (around 100 repetitions). The distribution of the sparks is (expectedly) independent of the function optimum, and shows a similar behavior for different initialization ranges, which were set to dim 1: $[15, 30]$; dim 2: $[15, 30]$ for Figures 4(a) and 4(b) and dim 1: $[60, 75]$; dim 2: $[30, 45]$ for Figures 4(c) and 4(d). Obviously, some Gaussian sparks are located very close to the origin of the function, independent of the initialization range. This is another indication why conventional FWA is able to find the optimum of centered functions within a few iterations.




(a) No shift – optimum at origin (b) Shift – optimum at (-70,-55)

(c) No shift – optimum at origin (d) Shift – optimum at (-70,-55)

Fig. 4. Sparks after initialization in conventional FWA

**The new Gaussian spark operator.** In order to avoid the problems of the conventional Gaussian mutation operator, we propose a new Gaussian mutation operator which is computed by $X_i^k = X_i^k + (X_B^k - X_i^k) * e$, where $X_B$ is the location of the currently best firework/explosion spark found so far, and e = $\mathcal{N}(0, 1)$. Details are given in Algorithm 4.

**Algorithm 4** – Generating "Gaussian sparks" in EFWA

1: Initialize the location of the "Gaussian sparks": $\tilde{X}_i = X_i$
2: Set $z^k = round(rand(0, 1))$, $k = 1, 2, ..., d$
3: Calculate offset displacement: $e = Gaussian(0, 1)$
4: **for** each dimension $\hat{X}_i^k$, where $z^k == 1$ **do**
5: $\hat{X}_i^k = \hat{X}_i^k + (X_B^k - \hat{X}_i^k) * e$, where $X_B$ is the position of the best firework found so far.
6: **if** $\tilde{X}_i^k$ out of bounds **then**
7: $\tilde{X}_i^k = X_{min}^k + rand * (X_{max}^k - X_{min}^k)$
8: **end if**
9: **end for**

As shown in Figure 5, the new mutation operator will stretch out along the direction between the current location of the firework and the location of the best firework. This ensures diversity of the search but also involves some global movement towards the best location found so far. This new operator only involves a movement towards the origin of the search space if the currently best firework is located at the origin.

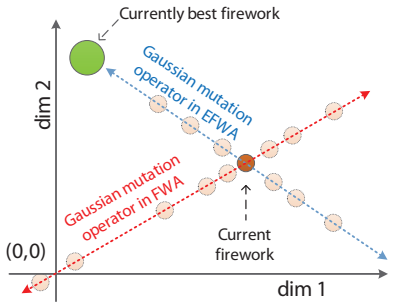
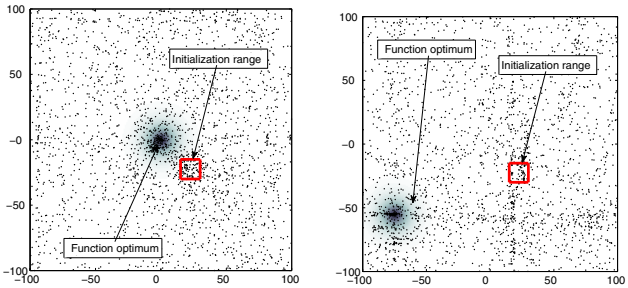



Fig. 5. Difference between the Gaussian sparks operator in FWA and EFWA

Figure 6 shows similar plots as Figure 3, however, for the newly proposed operator. The Gaussian sparks are uniformly distributed over the whole search space, with the largest amount of sparks being created close to the optimum. When comparing Figures 3(b) and 6(b) the differences between the old and the new operator are clearly visible. With the new operator, the sparks are not located around the origin of the search space if the optimum is far away from it. Figures 6(a) and 6(b) were created using the new mapping operator.




(a) No shift – optimum at origin (b) Shift – optimum at (-70,-55)

Fig. 6. The locations of the Gaussian sparks in EFWA (Ackely function using 100 000 function evaluations)

### E. A new Selection Operator

FWA involves a distance based selection strategy which favors to select fireworks/sparks in less crowded regions of the search space (cf. Eq. 4 and 5). Although selecting locations in low crowded regions with higher probability increases diversity, this selection operator has the drawback of being computational very expensive. A runtime profiling of the original FWA code revealed that the selection operator of conventional FWA is responsible for the majority of the runtime. In order to speed up the selection process of the population for the next generation, we apply another selection method, which is referred to as *Elitism-Random Selection* (ERP) method [12]. In this selection process, the optima of the set will be selected first. Then, the other individuals are selected randomly. Obviously, the computational complexity of ERP is only linear with respect to the number of fireworks, and therefore reduces the runtime of EFWA significantly.

## V. Experiments

In order to investigate the performances of the new operators we compare conventional FWA not only to the newly proposed Enhanced Fireworks Algorithm (EFWA), but also three variants of EFWA (denoted as eFWA-X) which can be regarded as a hybridization of FWA and EFWA. While EFWA uses all newly proposed operators, the eFWA-X variants use only some of them. Table I summarizes which of the new operators are used by the different algorithms. The abbreviations of the operators refer to EXP: new operator for generating explosion sparks; MAP: new mapping operator; GAU: new operator for generating Gaussian sparks; AMP 1: minimal explosion amplitude check linear decrease; AMP 2: minimal explosion amplitude check non-linear decrease; SEL: new selection operator. E.g., eFWA-I uses the new operators EXP, MAP and GAU, the new operator AMP is not included, and the selection operator is taken from conventional FWA. Besides comparing different FWA variants with each other, Standard PSO (SPSO) is used for performance comparison.

TABLE I
Algorithms and new operators

| | EXP | MAP | GAU | AMP 1 | AMP 2 | SEL |
|---|---|---|---|---|---|---|
| FWA | ○ | ○ | ○ | ○ | ○ | ○ |
| eFWA-I | ● | ● | ● | ○ | ○ | ○ |
| eFWA-II | ● | ● | ● | ● | ○ | ○ |
| eFWA-III | ● | ● | ● | ○ | ● | ○ |
| EFWA | ● | ● | ● | ○ | ● | ● |

### A. Experimental Setup

Twelve functions are selected as a test suite. Table II illustrates the names, numbers, search space (*Range*), optimal locations ($Opt\ \vec{x}$), fitness at the optimal location ($Opt\ f(\vec{x})$), and dimensions (*Dim.*). For each function, the initial range is set to $[X_k^{max}/2, X_k^{max}]$, where $X_k^{max}$ is the upper bound of the search space in the $k^{th}$ dimension. In the experiments, a number of shift values are added to these basic functions in order to shift the global optimum. We used seven different shift indexes in order to analyze the influence of different shift values on the performance of the algorithms (see Table III). For each shift value SI, the position of the function will be shifted (in each dimension) by the corresponding shift value SV. If SI is equal to zero, the function is not shifted. E.g., for function $f_1$ and an SI of 6, the function will be shifted in each dimension by $0.7 * ((100 - (-100))/2) = 70$, while the search range remains unaffected.

TABLE II
Benchmark functions used for evaluation

| Function name | # | Range | Opt. $\vec{x}$ | Opt. $f(\vec{x})$ | Dim. |
|---|---|---|---|---|---|
| Sphere | 1 | $[\pm 100]$ | $0.0^D$ | 0 | 30 |
| Schwefel 1.2 | 2 | $[\pm 100]$ | $0.0^D$ | 0 | 30 |
| General. Rosenbrock | 3 | $[\pm 30]$ | $1.0^D$ | 0 | 30 |
| Ackley | 4 | $[\pm 32]$ | $0.0^D$ | 0 | 30 |
| Generalized Griewank | 5 | $[\pm 600]$ | $0.0^D$ | 0 | 30 |
| Generalized Rastrigin | 6 | $[\pm 5.12]$ | $0.0^D$ | 0 | 30 |
| Penalized Func. P16 | 7 | $[\pm 50]$ | $1.0^D$ | 0 | 30 |
| Six-hump Camel-back | 8 | $[\pm 5]$ | $(-.09, .71)$ $(.09, -.71)$ | $-1.032$ | 2 |
| Goldstein-Price | 9 | $[\pm 2]$ | $(0, -1)$ | 3 | 2 |
| Schaffer | 10 | $[\pm 100]$ | $0.0^D$ | 0 | 2 |
| Axis Par. Hyp. Ell. | 11 | $[\pm 5.12]$ | $0.0^D$ | 0 | 30 |
| Rotated Hyp. Ell. | 12 | $[\pm 65.5]$ | $0.0^D$ | 0 | 30 |

TABLE III
Shift index (SI) and shift value (SV). A shift index of zero indicates that the function is not shifted.

| SI | SV | SI | SV | SI | SV |
|---|---|---|---|---|---|
| 1 | $0.05 * \frac{X_k^{max} - X_k^{min}}{2}$ | 2 | $0.1 * \frac{X_k^{max} - X_k^{min}}{2}$ | 3 | $0.2 * \frac{X_k^{max} - X_k^{min}}{2}$ |
| 4 | $0.3 * \frac{X_k^{max} - X_k^{min}}{2}$ | 5 | $0.5 * \frac{X_k^{max} - X_k^{min}}{2}$ | 6 | $0.7 * \frac{X_k^{max} - X_k^{min}}{2}$ |

For all experiments, $A_{init}$ and $A_{final}$ (Eq. 7 and 8) are set to $(X_{max}^k - X_{min}^k) \times 0.02$ and $(X_{max}^k - X_{min}^k) \times 0.001$, respectively. All other parameters of (E)FWA are taken from [1], and SPSO parameters are taken from [13]. As experimental platform we used MATLAB 2011b, running Win 7 on an Intel Core i7-2600 CPU; 3.7GHZ; 8GB RAM. Compared to conventional FWA, the only additional parameters of EFWA are $A_{init}$ and $A_{final}$, which can be fixed as fractions of the search space. The other parameters are ($\hat{A}$, $M_e$, $a$, $b$, $N$, $M_g$).

### B. Experimental Results

In this section, we first evaluate the influence of the newly proposed operators presented in Section IV. After that, we compare EFWA with conventional FWA and also with SPSO, our baseline reference. The final results after 300 000 function evaluations are presented for all twelve benchmark functions in Table V as mean fitness and standard deviation over 30 runs. Moreover, for selected functions we show the convergence plots in Figures 8 to 10.

**Evaluation of EXP, MAP and GAU.** Since these three [1] operators are responsible for the fact that conventional FWA

[1] We note that the questionable part of the explosion sparks operator of conventional FWA is the mapping operation, not the explosion operation itself.

performs better on function which have their optimum at the origin of the search space than on functions whose optimum is far away from the center (cf. Section III), all eFWA variants use the new operators EXP, MAP and GAU. As expected, when SI=0 conventional FWA reaches the optimum of all functions that have their optimum at $0.0^D$ (marked in red font in Table V). However, as already mentioned, this good performance is not due to the intelligence of the algorithm but rather because the Gaussian explosion operator and the mapping operator of conventional FWA create/map many sparks to locations which are very close to the search space. Indeed, Table V reveals that when SI=0, conventional FWA only fails to find the optimum of two function, $f_3$ and $f_7$ – both functions have their optimum at $1.0^D$, and thus not at the origin of the search space. With increasing SI, the results of FWA worsen significantly for most functions, especially so for $f_1$, $f_2$, $f_5$, $f_7$, $f_{11}$, $f_{12}$. When comparing the results of conventional FWA with eFWA-I (using the new operators EXP, MAP, GAU), it can be seen that changing these three operators alone does not improve the performance of the algorithm. Although the results are more stable with respect to different shift values, in most cases they are worse than the results of conventional FWA. In the next paragraph we will discuss how the AMP operator is able to improve eFWA-I.

**Evaluation of AMP.** Table V reveals that the minimal explosion amplitude check strategy (*AMP*) is crucial for improving the diversity of the fireworks. The difference between the results of eFWA-I, which does not use an explosion amplitude check, and eFWA-II (linear decrease of $A^k_{min}$) and eFWA-III (non-linear decrease of $A^k_{min}$) are obvious. Both, eFWA-II and eFWA-III, clearly outperform eFWA-I. Comparing the results of eFWA-II and III, it can be seen that eFWA-III achieves slightly better results. Hence, the non-linear decrease of the minumum explosion amplitude $A^k_{min}$ is preferred over the linear decrease of $A^k_{min}$, and will be used in EFWA.

**Evaluation of SEL.** Compared to eFWA-III, EFWA replaces the time consuming distance based selection operator (cf. Eq. 4) with the new selection operator (cf. Section IV-E). In terms of convergence, final fitness and standard deviation, there is almost no difference between the selection operators. The results of eFWA-III and EFWA are almost identical for all functions (cf. Table V). The only exceptions are function $f_4$, with advantages for the distance based selection operator (eFWA-III), and $f_6$, with advantages for the elitism-random selection operator (EFWA). In terms of computation cost the picture is different. The new selection operator decreases the runtime of EFWA drastically, as shown in Figure 7 for the function $f_{10}$. As can be seen, the runtime of EFWA is much shorter than the runtimes of conventional FWA or the eFWA-X variants, which all use the conventional distance based selection operator. We note that the fractions of runtimes for all other functions are very similar to Figure 7.

**EFWA vs. conventional FWA.** A comparison between all eFWA-X variants and EFWA reveals that EFWA is the best

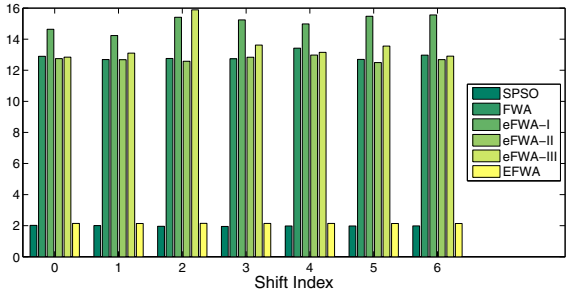



Fig. 7. Runtime

algorithm in terms of convergence, final result and runtime. Hence, in the remaining parts of this section we use EFWA for comparison with conventional FWA and with the baseline algorithm SPSO. To show whether the improvement of EFWA over conventional FWA is significant or not, a number of $t$-tests were conducted and the recorded $p$-values are given in Table IV. The null hypothesis that EFWA achieves similar mean results than FWA is tested against the alternative that EFWA achieves better mean results over FWA. If $p < 0.05$, and the mean results of EFWA are lower than the results of FWA, the null hypothesis will be rejected and the test will be reported as significant; otherwise it will be reported as not significant. Hence, bold values in Table IV indicate that the improvement of EFWA is significant for the corresponding function / SI combination.

TABLE IV
$t$-TEST RESULTS FOR EFWA VS. CONVENTIONAL FWA.

| SI | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| $f_1$ | 6.6E-17 | **2.3E-15** | **3.6E-16** | **3.7E-15** | **1.E-14** | **6.4E-16** | **1.E-17** |
| $f_2$ | 1.2E-15 | **7.1E-12** | **7.7E-11** | **2.9E-12** | **2.5E-12** | **9.1E-15** | **3.7E-21** |
| $f_3$ | 6.1E-04 | 8.5E-06 | 2.1E-01 | 2.7E-01 | **1.5E-02** | **5.5E-05** | **8.E-04** |
| $f_4$ | 2.5E-01 | 1.7E-01 | 8.0E-05 | 3.1E-63 | 2.7E-48 | 9.8E-46 | 1.2E-52 |
| $f_5$ | 1.9E-10 | 5.2E-02 | **4.E-12** | **2.6E-14** | **4.E-16** | **1.8E-18** | **4.7E-20** |
| $f_6$ | 8.5E-21 | 1.1E-20 | 4.0E-17 | 6.6E-19 | 1.3E-14 | 8.5E-15 | 6.1E-15 |
| $f_7$ | **6.7E-07** | **2.5E-08** | **3.7E-12** | **1.6E-15** | **1.6E-13** | **1.1E-13** | **8.7E-15** |
| $f_8$ | 1.0E+00 | 3.3E-01 | 1.8E-01 | 5.7E-02 | **6.E-03** | **3.6E-04** | **2.8E-04** |
| $f_9$ | **2.4E-02** | **6.7E-05** | **2.4E-03** | **2.8E-05** | **5.5E-05** | **1.2E-04** | **8.7E-04** |
| $f_{10}$ | NaN | **5.9E-03** | **5.8E-03** | **4.5E-09** | **8.5E-07** | **4.3E-09** | **8.5E-08** |
| $f_{11}$ | 1.4E-12 | **6.1E-05** | **3.2E-13** | **1.2E-13** | **1.2E-15** | **2.E-11** | **2.8E-14** |
| $f_{12}$ | 1.1E-16 | **9.3E-16** | **3.8E-15** | **2.4E-15** | **8.7E-17** | **1.9E-12** | **1.3E-15** |

We draw the following conclusions from Tables IV and V:

Functions are *not* shifted (i.e, SI = 0): FWA achieves better results than EFWA for all functions which have their optimum at the origin, and also for function $f_3$ which has the optimum at $1.0^D$ (cf. Table II). For the other functions which have their optimum at a different location than the origin ($f_7$, $f_8$, and $f_9$), we can see that EFWA can improve the results of FWA for functions $f_7$ and $f_9$. For function $f_8$, both algorithms achieved similar results, however, with smaller variance for EFWA.

Functions are shifted (i.e, SI != 0): With increasing SI, EFWA is able to improve the results of FWA progressively. The results of EFWA are much more stable with respect to increasing

SI for most functions. Figure 8 (function $f_2$)[2] is an example of a function where the results of FWA are worsening with increasing SI, while the results of EFWA remain unaffected. The results in Table IV indicate that the improvement of EFWA over conventional FWA is significant for almost all functions, when SI is increased. However, we also note that for the functions $f_4$ and $f_6$, FWA has very stable results even for increasing SI. Surprisingly, for these functions FWA outperforms EFWA and also SPSO. Figure 9 shows the convergence plot for function $f_4$. As can be seen, FWA is able to improve its results continuously over the full duration of the algorithm. Although the results worsen with increasing SI, the results are superior to all other algorithms, which often get stuck in local minima for larger SI.

**EFWA vs. SPSO.** Figure 9 reveals that not only EFWA has problems with this function, but also SPSO. Although this function is an extreme case, Table V shows that SPSO is rather sensitive to increasing SI values and fails to converge towards the optimum for several functions when the value of SI is large. For small SI, SPSO is often able to achieve better results than EFWA, and also outperforms EFWA for any SI for function $f_5$. However, for functions $f_1$, $f_3$, $f_7$, $f_{11}$ and $f_{12}$, the results of SPSO worsen significantly when SI is increased. An example of this behavior is shown in Figure 10. For small SI, SPSO converges quickly to the optimum and achieves better results than EFWA. However, for SI $\in \{3, 5, 6\}$, SPSO converges to results which are by several orders of magnitude larger (worse) than the results of EFWA. We also note that for the functions $f_2$ and $f_6$ the results of SPSO are generally worse than the results of EFWA.

### C. Discussion

From the results of our experimental evaluation, we conclude the following observations:

- In general, EFWA shows significant improvements over conventional FWA for most functions.
- With increasing shift values, EFWA achieves much better results than FWA.
- SPSO often achieves better results than EFWA for small SI.
- However, SPSO often fails to converge towards the optimum for large SI, which is not the case with EFWA.
- The results of EFWA remain almost unaffected even if the optimum of the function is shifted towards the edges of the search space.
- The new operator AMP, which limits the lower bound of the explosion amplitude, is an important tool to balance between exploration and exploitation abilities.
- EFWA reduces the runtime of FWA by a factor of six.

Although the results indicate the usefulness and benefits of EFWA, some open question remain in this context. At first, the behavior of FWA on the functions $f_4$ and $f_6$ is not fully understood at the moment. A second open research question refers to the improvement of EFWA for small SI. The results indicate that SPSO achieves good results for small SI, but worsens when SI is increased. EFWA, on the other hand, remains unaffected when SI is increased, but has a slower convergence that SPSO for some functions, and converges to a larger fitness for small SI. Improving the convergence speed and quality of EFWA would result in a very stable algorithm that is absolutely not influenced by the location of the function optimum within the search space.

[2] We omitted the convergence plots for SI=0, since the FWA results for SI=0 would increase the necessary scale on the y-axis and hamper the readability

## VI. Conclusion

In this paper, we have presented the *Enhanced Fireworks Algorithm (EFWA)*, a significant improvement of the recently developed Fireworks Algorithm (FWA). In order to eliminate the drawbacks of conventional FWA, we performed a comprehensive study on the basic FWA operators and presented several improvements: besides improving the operators for creating explosion and Gaussian sparks, we fixed some problems of the mapping operator, and introduced a new operator which limits the lower bound of the explosion amplitude, a parameter which allows to balance between exploration and exploitation. In order to speed up the runtime, we further introduced a new selection operator.

Experimental evaluation showed that, with the exception of one benchmark function, the results of EFWA are very stable and remain almost unaffected even if the optimum of the function is shifted towards the edges of the search space. In general, EFWA shows significant improvements over conventional FWA. Compared to SPSO, which turned out to be rather sensitive to increasing shift values, EFWA achieves very stable results, and has the advantage that its results do not deteriorate even for large shift values. In terms of computational cost, the new selection operator is faster by a factor of 6 compared to the distance based selection operator of conventional FWA, making EFWA as fast as SPSO.

In future work, we will focus on further improving EFWA by exploiting the full potential of exploration and exploitation capabilities of the algorithm. We are currently extending our work on improving the explosion sparks and Gaussian sparks operator, and on optimizing the newly introduced AMP operators, which seems to be crucial for EFWA. Besides that, we plan to include an in-depth analysis of the influence of different parameters for EFWA, and to compare our algorithms with other nature-inspired optimization techniques.

## Acknowledgment

This work was supported by the National Natural Science Foundation of China under grants number 61170057 and 60875080.

## References

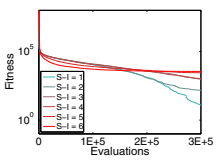
[1] Y. Tan and Y. Zhu, "Fireworks algorithm for optimization," Advances in Swarm Intelligence, pp. 355–364, 2010.

[2] M. Clerc and J. Kennedy, "The particle swarm - explosion, stability, and convergence in a multidimensional complex space," Trans. Evol. Comp, vol. 6, no. 1, pp. 58–73, 2002.

TABLE V

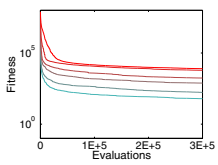
MEAN AND VARIANCE (IN PARENTHESIS) OF ALL TWELVE BENCHMARK FUNCTIONS USED IN THE EXPERIMENTS (SI=SHIFT INDEX).

| SI | Alg. | $f_1$ | $f_2$ | $f_3$ | $f_4$ | $f_5$ | $f_6$ | $f_7$ | $f_8$ | $f_9$ | $f_{10}$ | $f_{11}$ | $f_{12}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | SPSO | 0.000e+0(0.0e+0) | 6.515e+0(6.4e+0) | 1.740e+1(2.5e+1) | 1.982e+1(1.6e-1) | 5.751e-4(2.2e-3) | 1.508e+2(2.5e+1) | 0.000e+0(0.0e+0) | -1.032e+0(0.0e+0) | 3.000e+0(0.0e+0) | 0.000e+0(0.0e+0) | 0.000e+0(0.0e+0) | 0.000e+0(0.0e+0) |
| | FWA | 0.000e+0(0.0e+0) | 0.000e+0(0.0e+0) | 1.807e+1(1.1e+1) | 0.000e+0(0.0e+0) | 0.000e+0(0.0e+0) | 0.000e+0(0.0e+0) | 7.088e-3(6.1e-3) | -1.032e+0(0.0e+0) | 3.000e+0(1.2e-6) | 0.000e+0(0.0e+0) | 0.000e+0(0.0e+0) | 0.000e+0(0.0e+0) |
| | eFWA-I | 3.744e+1(1.7e+1) | 2.916e+3(1.1e+3) | 3.098e+3(1.9e+3) | 2.673e+0(3.1e-1) | 2.773e-1(1.8e-1) | 1.720e+1(3.5e+0) | 8.316e+0(4.2e+0) | -1.032e+0(1.2e-5) | 3.000e+0(7.5e-5) | 3.245e-3(4.7e-3) | 8.834e+0(3.5e+0) | 4.313e+2(1.7e+2) |
| | eFWA-II | 7.031e-3(2.5e-3) | 9.426e-1(2.5e-1) | 1.065e+2(1.2e+2) | 1.745e-2(2.7e-3) | 7.989e-2(4.4e-2) | 7.262e+1(1.9e+1) | 2.514e-4(1.2e-4) | -1.032e+0(0.0e+0) | 3.000e+0(0.0e+0) | 0.000e+0(0.0e+0) | 1.966e-4(8.9e-5) | 3.634e-2(1.6e-2) |
| | eFWA-II | 1.144e-3(4.1e-4) | 2.300e-1(6.8e-2) | 7.877e+1(9.9e+1) | 2.357e-1(9.6e-1) | 8.776e-2(5.3e-2) | 5.331e+1(1.3e+1) | 4.273e-5(1.7e-5) | -1.032e+0(0.0e+0) | 3.000e+0(0.0e+0) | 0.000e+0(0.0e+0) | 3.947e-5(1.4e-5) | 6.128e-3(2.2e-3) |
| | EFWA | 9.704e-4(3.0e-4) | 2.221e-1(7.8e-2) | 7.933e+1(8.7e+1) | 6.430e-2(3.0e-1) | 7.204e-2(4.1e-2) | 1.597e+1(3.6e+0) | 3.540e-5(1.5e-5) | -1.032e+0(0.0e+0) | 3.000e+0(0.0e+0) | 0.000e+0(0.0e+0) | 4.027e-5(1.9e-5) | 5.765e-3(1.8e-3) |
| 1 | SPSO | 0.000e+0(0.0e+0) | 1.244e+1(1.4e+1) | 1.687e+1(2.2e+1) | 1.979e+1(9.6e-2) | 7.396e-4(2.3e-3) | 1.334e+2(3.2e+1) | 0.000e+0(0.0e+0) | -1.032e+0(0.0e+0) | 3.000e+0(0.0e+0) | 3.239e-4(1.8e-3) | 0.000e+0(0.0e+0) | 0.000e+0(0.0e+0) |
| | FWA | 2.370e-1(8.5e-2) | 6.169e+1(3.1e+1) | 1.469e+0(8.7e-1) | 2.191e-1(2.6e-1) | 1.117e-1(4.0e-2) | 4.344e+0(2.6e+0) | 1.323e-2(9.6e-3) | -1.032e+0(1.8e-7) | 3.000e+0(4.4e-6) | 2.269e-3(4.2e-3) | 1.603e-2(1.9e-2) | 2.254e+0(7.8e-1) |
| | eFWA-I | 3.668e+1(1.0e+1) | 3.105e+3(6.8e+2) | 9.627e+3(4.2e+3) | 3.304e+0(3.7e-1) | 3.256e-1(2.1e-1) | 2.257e+1(5.0e+0) | 8.016e+0(3.5e+0) | -1.032e+0(1.0e-5) | 3.000e+0(4.4e-5) | 3.244e-3(4.7e-3) | 1.040e+1(3.5e+0) | 4.506e+2(1.3e+2) |
| | eFWA-II | 6.621e-3(2.9e-3) | 9.042e-1(2.2e-1) | 9.748e+1(9.6e+1) | 1.668e-2(3.3e-3) | 6.811e-2(3.9e-2) | 7.973e+1(2.0e+1) | 2.405e-4(1.4e-4) | -1.032e+0(0.0e+0) | 3.000e+0(0.0e+0) | 0.000e+0(0.0e+0) | 2.239e-4(9.2e-5) | 3.602e-2(1.4e-2) |
| | eFWA-II | 9.933e-4(2.9e-4) | 2.241e-1(7.2e-2) | 6.288e+1(6.8e+1) | 7.038e-2(3.5e-1) | 7.930e-2(5.6e-2) | 5.579e+1(9.6e+0) | 3.887e-5(1.6e-5) | -1.032e+0(0.0e+0) | 3.000e+0(0.0e+0) | 0.000e+0(0.0e+0) | 3.327e-5(1.5e-5) | 6.482e-3(2.1e-3) |
| | EFWA | 9.592e-4(2.5e-4) | 2.120e-1(7.7e-2) | 1.212e+2(1.2e+2) | 5.209e-1(1.1e+0) | 8.800e-2(5.2e-2) | 1.860e+1(4.2e+0) | 3.633e-5(1.4e-5) | -1.032e+0(0.0e+0) | 3.000e+0(0.0e+0) | 0.000e+0(0.0e+0) | 3.657e-5(1.5e-5) | 5.166e-3(1.7e-3) |
| 2 | SPSO | 0.000e+0(0.0e+0) | 1.495e+2(7.4e+2) | 3.059e+1(4.8e+1) | 1.997e+1(1.8e-1) | 8.212e-4(3.4e-3) | 1.160e+2(2.5e+1) | 0.000e+0(0.0e+0) | -1.032e+0(0.0e+0) | 3.000e+0(0.0e+0) | 0.000e+0(0.0e+0) | 0.000e+0(0.0e+0) | 0.000e+0(0.0e+0) |
| | FWA | 6.743e-1(2.3e-1) | 1.691e+2(9.3e+1) | 8.140e+1(4.2e+1) | 7.262e-1(4.1e-1) | 2.036e-1(6.0e-2) | 6.834e+0(2.1e+0) | 6.801e-2(3.3e-2) | -1.032e+0(4.0e-7) | 3.000e+0(1.7e-5) | 2.270e-3(4.2e-3) | 6.040e-2(2.6e-2) | 5.654e+0(2.1e+0) |
| | eFWA-I | 3.781e+1(1.5e+1) | 2.893e+3(9.9e+2) | 1.837e+4(8.3e+3) | 3.504e+0(4.9e-1) | 5.993e-1(2.3e-1) | 2.454e+1(4.3e+0) | 7.558e+0(2.8e+0) | -1.032e+0(7.0e-6) | 3.000e+0(5.3e-5) | 3.570e-3(4.8e-3) | 1.062e+1(3.7e+0) | 5.173e+2(1.6e+2) |
| | eFWA-II | 6.984e-3(2.5e-3) | 9.910e-1(4.0e-1) | 1.076e+2(9.6e+1) | 1.677e-2(3.1e-3) | 7.924e-2(5.0e-2) | 7.571e+1(1.9e+1) | 2.565e-4(1.3e-4) | -1.032e+0(0.0e+0) | 3.000e+0(0.0e+0) | 0.000e+0(0.0e+0) | 2.095e-4(8.6e-5) | 3.749e-2(1.5e-2) |
| | eFWA-II | 1.082e-3(3.4e-4) | 2.186e-1(6.5e-2) | 7.765e+1(1.1e+2) | 2.542e-1(7.8e-1) | 7.496e-2(5.1e-2) | 5.782e+1(1.2e+1) | 4.273e-5(1.9e-5) | -1.032e+0(0.0e+0) | 3.000e+0(0.0e+0) | 0.000e+0(0.0e+0) | 3.030e-5(1.1e-5) | 6.516e-3(3.3e-3) |
| | EFWA | 9.298e-4(3.3e-4) | 2.139e-1(6.4e-2) | 1.072e+2(1.0e+2) | 8.745e+0(9.6e+0) | 8.202e-2(4.5e-2) | 1.822e+1(4.1e+0) | 3.400e-5(1.2e-5) | -1.032e+0(0.0e+0) | 3.000e+0(0.0e+0) | 0.000e+0(0.0e+0) | 3.433e-5(1.1e-5) | 6.667e-3(2.1e-3) |
| 3 | SPSO | 0.000e+0(0.0e+0) | 9.758e+2(1.6e+3) | 2.533e+1(2.9e+1) | 1.993e+1(3.4e-2) | 4.105e-4(2.2e-3) | 1.125e+2(2.2e+1) | 3.662e-4(2.0e-3) | -1.032e+0(0.0e+0) | 3.000e+0(0.0e+0) | 0.000e+0(0.0e+0) | 0.000e+0(0.0e+0) | 4.123e+2(2.3e+3) |
| | FWA | 1.580e+0(5.8e-1) | 7.699e+2(3.7e+2) | 1.626e+2(1.5e+2) | 1.425e+0(5.6e-1) | 2.984e-1(9.1e-2) | 3.701e+0(2.2e+0) | 1.917e-1(6.8e-2) | -1.032e+0(4.6e-7) | 3.000e+0(1.3e-5) | 6.803e-3(4.5e-3) | 1.315e-1(5.5e-2) | 1.475e+1(5.3e+0) |
| | eFWA-I | 4.452e+1(1.6e+1) | 2.908e+3(7.1e+2) | 2.427e+4(1.2e+4) | 3.597e+0(4.9e-1) | 7.671e-1(1.5e-1) | 2.252e+1(3.9e+0) | 8.726e+0(3.7e+0) | -1.032e+0(1.1e-5) | 3.000e+0(7.3e-5) | 4.227e-3(4.9e-3) | 1.246e+1(3.6e+0) | 4.762e+2(1.7e+2) |
| | eFWA-II | 6.899e-3(2.6e-3) | 9.309e-1(3.7e-1) | 1.090e+2(1.1e+2) | 4.718e-1(1.2e+0) | 7.603e-2(6.0e-2) | 9.446e+1(2.0e+1) | 2.412e-4(1.2e-4) | -1.032e+0(0.0e+0) | 3.000e+0(0.0e+0) | 0.000e+0(0.0e+0) | 2.259e-4(7.8e-5) | 3.647e-2(1.3e-2) |
| | eFWA-II | 1.077e-3(3.5e-4) | 2.255e-1(9.0e-2) | 7.070e+1(9.0e+1) | 5.836e-1(1.5e+0) | 8.533e-2(5.1e-2) | 6.029e+1(1.3e+1) | 4.063e-5(1.9e-5) | -1.032e+0(0.0e+0) | 3.000e+0(0.0e+0) | 0.000e+0(0.0e+0) | 4.040e-5(1.7e-5) | 6.686e-3(3.3e-3) |
| | EFWA | 9.764e-4(4.4e-4) | 2.275e-1(7.9e-2) | 1.196e+2(1.5e+2) | 1.984e+1(3.0e-1) | 8.905e-2(6.0e-2) | 1.954e+1(5.0e+0) | 3.747e-5(2.5e-5) | -1.032e+0(0.0e+0) | 3.000e+0(0.0e+0) | 0.000e+0(0.0e+0) | 3.350e-5(1.4e-5) | 6.378e-3(2.2e-3) |
| 4 | SPSO | 0.000e+0(0.0e+0) | 1.088e+3(1.6e+3) | 1.887e+1(1.7e+1) | 1.998e+1(3.1e-2) | 0.000e+0(0.0e+0) | 8.588e+1(1.6e+1) | 0.000e+0(0.0e+0) | -1.032e+0(0.0e+0) | 3.000e+0(0.0e+0) | 0.000e+0(0.0e+0) | 0.000e+0(0.0e+0) | 0.000e+0(0.0e+0) |
| | FWA | 2.423e+0(9.2e-1) | 1.735e+3(8.3e+2) | 2.708e+2(1.3e+2) | 1.924e+0(4.5e-1) | 3.325e-1(9.6e-2) | 9.380e+0(2.8e+0) | 3.432e-1(1.5e-1) | -1.032e+0(2.2e-6) | 3.000e+0(2.0e-5) | 5.533e-3(4.9e-3) | 2.340e-1(8.2e-2) | 2.168e+1(6.9e+0) |
| | eFWA-I | 3.587e+1(1.2e+1) | 3.209e+3(9.4e+2) | 3.009e+4(1.8e+4) | 4.344e+0(5.1e-1) | 7.931e-1(1.5e-1) | 2.770e+1(4.9e+0) | 9.571e+0(3.2e+0) | -1.032e+0(1.8e-5) | 3.000e+0(1.2e-4) | 2.603e-3(4.4e-3) | 1.254e+1(5.0e+0) | 5.213e+2(1.7e+2) |
| | eFWA-II | 6.770e-3(2.4e-3) | 9.315e-1(4.4e-1) | 1.368e+2(1.4e+2) | 9.402e+0(9.1e+0) | 8.309e-2(4.7e-2) | 1.023e+2(2.2e+1) | 2.594e-4(1.4e-4) | -1.032e+0(0.0e+0) | 3.000e+0(0.0e+0) | 0.000e+0(0.0e+0) | 2.206e-4(1.0e-4) | 3.914e-2(1.7e-2) |
| | eFWA-II | 1.101e-3(3.7e-4) | 2.284e-1(7.1e-2) | 8.679e+1(1.3e+2) | 4.419e+0(6.3e+0) | 6.719e-2(5.2e-2) | 7.012e+1(1.1e+1) | 3.480e-5(1.6e-5) | -1.032e+0(0.0e+0) | 3.000e+0(0.0e+0) | 0.000e+0(0.0e+0) | 3.833e-5(1.6e-5) | 6.959e-3(2.7e-3) |
| | EFWA | 1.108e-3(4.5e-4) | 2.323e-1(7.8e-2) | 1.747e+2(1.6e+2) | 1.996e+1(2.1e-2) | 7.947e-2(4.4e-2) | 1.919e+1(4.2e+0) | 2.943e-5(1.1e-5) | -1.032e+0(0.0e+0) | 3.000e+0(0.0e+0) | 0.000e+0(0.0e+0) | 3.673e-5(1.7e-5) | 5.984e-3(2.1e-3) |
| 5 | SPSO | 0.000e+0(0.0e+0) | 2.794e+3(2.4e+3) | 2.370e+1(2.8e+1) | 1.999e+1(5.5e-3) | 0.000e+0(0.0e+0) | 6.956e+1(1.7e+1) | 0.000e+0(0.0e+0) | -1.032e+0(0.0e+0) | 3.000e+0(0.0e+0) | 0.000e+0(0.0e+0) | 1.966e+0(1.1e+1) | 2.147e+2(1.2e+3) |
| | FWA | 3.222e+0(1.1e+0) | 6.081e+3(2.3e+3) | 6.202e+2(5.6e+2) | 2.144e+0(5.4e-1) | 3.748-1(9.6e-2) | 9.403e+0(2.3e+0) | 6.438e-1(2.7e-1) | -1.032e+0(4.2e-6) | 3.000e+0(5.6e-5) | 6.807e-3(4.5e-3) | 4.358e-1(2.3e-1) | 4.245e+1(2.0e+1) |
| | eFWA-I | 3.997e+1(1.1e+1) | 2.886e+3(7.5e+2) | 2.620e+4(1.7e+4) | 5.788e+0(1.0e+0) | 8.246e-1(1.3e-1) | 2.503e+1(5.6e+0) | 9.795e+0(5.9e+0) | -1.032e+0(1.5e-5) | 3.000e+0(1.2e-4) | 3.898e-3(4.8e-3) | 1.142e+1(3.9e+0) | 4.953e+2(1.6e+2) |
| | eFWA-II | 6.710e-3(2.9e-3) | 9.733e-1(2.8e-1) | 1.988e+2(1.8e+2) | 1.993e+1(2.9e-1) | 7.305e-2(4.0e-2) | 1.112e+2(2.7e+1) | 2.530e-4(1.5e-4) | -1.032e+0(0.0e+0) | 3.000e+0(0.0e+0) | 0.000e+0(0.0e+0) | 2.195e-4(7.1e-5) | 3.622e-2(1.2e-2) |
| | eFWA-II | 1.088e-3(2.9e-4) | 2.330e-1(6.5e-2) | 1.335e+2(1.7e+2) | 1.824e+1(4.5e+0) | 7.295e-2(4.4e-2) | 7.583e+1(1.3e+1) | 4.390e-5(1.6e-5) | -1.032e+0(0.0e+0) | 3.000e+0(0.0e+0) | 0.000e+0(0.0e+0) | 3.603e-5(1.3e-5) | 5.990e-3(2.1e-3) |
| | EFWA | 1.080e-3(3.4e-4) | 2.299e-1(6.3e-2) | 1.328e+2(1.3e+2) | 1.999e+1(5.6e-3) | 8.474e-2(4.8e-2) | 1.830e+1(3.8e+0) | 3.360e-5(1.1e-5) | -1.032e+0(0.0e+0) | 3.000e+0(0.0e+0) | 0.000e+0(0.0e+0) | 3.950e-5(1.9e-5) | 6.432e-3(1.9e-3) |
| 6 | SPSO | 4.200e+2(6.6e+2) | 3.541e+3(2.9e+3) | 2.282e+5(5.3e+5) | 2.000e+1(6.5e-4) | 0.000e+0(0.0e+0) | 4.655e+1(2.0e+1) | 1.000e+5(3.1e+5) | -1.032e+0(0.0e+0) | 3.000e+1(0.0e+0) | 0.000e+0(0.0e+0) | 3.209e+1(3.8e+1) | 3.943e+3(5.0e+3) |
| | FWA | 3.596e+0(1.1e+0) | 7.910e+3(1.7e+3) | 7.382e+2(7.5e+2) | 2.488e+0(3.1e-1) | 4.050e-1(9.7e-2) | 9.641e+0(2.5e+0) | 7.653e-1(2.9e-1) | -1.032e+0(2.7e-6) | 3.000e+1(3.4e-3) | 6.156e-3(4.8e-3) | 4.951e-1(2.0e-1) | 4.277e+1(1.5e+1) |
| | eFWA-I | 3.923e+1(1.5e+1) | 2.509e+3(6.6e+2) | 2.662e+4(1.3e+4) | 1.109e+1(4.0e+0) | 8.873e-1(1.1e-1) | 2.444e+1(4.4e+0) | 7.906e+0(2.8e+0) | -1.032e+0(4.5e-6) | 3.000e+1(1.5e-3) | 4.866e-3(4.9e-3) | 9.823e+0(3.6e+0) | 4.403e+2(1.6e+2) |
| | eFWA-II | 6.704e-3(3.0e-3) | 9.404e-1(2.7e-1) | 2.339e+2(2.2e+2) | 2.000e+1(1.7e-3) | 7.339e-2(4.7e-2) | 1.323e+2(3.0e+1) | 2.425e-4(1.4e-4) | -1.032e+0(0.0e+0) | 3.000e+1(0.0e+0) | 0.000e+0(0.0e+0) | 2.464e-4(6.8e-5) | 3.659e-2(1.2e-2) |
| | eFWA-II | 1.045e-3(3.5e-4) | 2.170e-1(7.7e-2) | 1.374e+2(1.2e+2) | 1.986e+1(1.3e-1) | 8.347e-2(4.1e-2) | 9.067e+1(2.6e+1) | 4.267e-5(2.0e-5) | -1.032e+0(0.0e+0) | 3.000e+1(0.0e+0) | 0.000e+0(0.0e+0) | 3.603e-5(1.0e-5) | 6.369e-3(2.7e-3) |
| | EFWA | 1.086e-3(3.6e-4) | 2.115e-1(6.4e-2) | 2.129e+2(2.1e+2) | 2.000e+1(1.3e-3) | 7.208e-2(4.4e-2) | 1.992e+1(4.3e+0) | 3.687e-5(1.5e-5) | -1.032e+0(0.0e+0) | 3.000e+1(0.0e+0) | 0.000e+0(0.0e+0) | 3.443e-5(1.8e-5) | 6.195e-3(2.5e-3) |



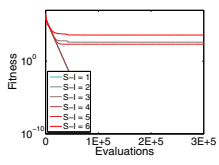

(a) SPSO (b) FWA (c) eFWA-III (d) EFWA

Fig. 8. Function $f_2$ – Schwefel 1.2.




(a) SPSO (b) FWA (c) EWA-III (d) EFWA

Fig. 9. Function $f_4$ – Ackley




(a) SPSO (b) FWA (c) EWA-III (d) EFWA

Fig. 10. Function $f_{12}$ – Rotated Hyper Ellipsoid


[3] Y. Tan and Z. Xiao, "Clonal particle swarm optimization and its applications," in Evolutionary Computation, 2007. CEC 2007. IEEE Congress on, 2007, pp. 2303 –2309.

[4] A. Janecek and Y. Tan, "Iterative improvement of the multiplicative update nmf algorithm using nature-inspired optimization," in Natural Computation (ICNC), 2011 Seventh International Conference on, vol. 3. IEEE, 2011, pp. 1668–1672.

[5] ——, "Swarm intelligence for non-negative matrix factorization," International Journal of Swarm Intelligence Research (IJSIR), vol. 2, no. 4, pp. 12–34, 2011.

[6] ——, "Using population based algorithms for initializing nonnegative matrix factorization," in Proceedings of the second international conference on Advances in swarm intelligence, ser. ICSI'11. Springer-Verlag, 2011, pp. 307–316.

[7] S. Bureerat, "Hybrid population-based incremental learning using real codes," Learning and Intelligent Optimization, pp. 379–391, 2011.

[8] H. Gao and M. Diao, "Cultural firework algorithm and its application for digital filters design," International Journal of Modelling, Identification and Control, vol. 14, no. 4, pp. 324–331, 2011.

[9] Y. Zheng, X. Xu, and H. Ling, "A hybrid fireworks optimization method with differential evolution," Neurocomputing, 2012.

[10] Y. Pei, S. Zheng, Y. Tan, and T. Hideyuki, "An empirical study on influence of approximation approaches on enhancing fireworks algorithm," in Proceedings of the 2012 IEEE Congress on System, Man and Cybernetics. IEEE, 2012, pp. 1322–1327.

[11] G. Lu, D. Tan, and H. Zhao, "Improvement on regulating definition of antibody density of immune algorithm," in Neural Information Processing, 2002. ICONIP'02. Proceedings of the 9th International Conference on, vol. 5. IEEE, 2002, pp. 2669–2672.

[12] A. P. Engelbrecht, Fundamentals of Computational Swarm Intelligence. John Wiley & Sons, 2006.

[13] D. Bratton and J. Kennedy, "Defining a standard for particle swarm optimization," in Swarm Intelligence Symposium, 2007. SIS 2007. IEEE. IEEE, 2007, pp. 120–127.