2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

Dynamic Search in Fireworks Algorithm

Shaoqiu Zheng, Andreas Janecek, Junzhi Li and Ying Tan

***Abstract*— We propose an improved version of the recently developed Enhanced Fireworks Algorithm (EFWA) based on an adaptive dynamic local search mechanism. In EFWA, the explosion amplitude (i.e., search area around the current location) of each firework is computed based on the quality of the firework's current location. This explosion amplitude is limited by a lower bound which decreases with the number of iterations in order to avoid the explosion amplitude to be [close to] zero, and in order to enhance global search abilities at the beginning and local search abilities towards the later phase of the algorithm. As the explosion amplitude in EFWA depends solely on the fireworks' fitness and the current number of iterations, this procedure does not allow for an adaptive optimization process. To deal with these limitations, we propose the Dynamic Search Fireworks Algorithm (dynFWA) which uses a dynamic explosion amplitude for the firework at the currently best position. If the fitness of the best firework could be improved, the explosion amplitude will increase in order to speed up convergence. On the contrary, if the current position of the best firework could not be improved, the explosion amplitude will decrease in order to narrow the search area. In addition, we show that one of the EFWA operators can be removed in dynFWA without a loss in accuracy — this makes dynFWA computationally more efficient than EFWA. Experiments on 28 benchmark functions indicate that dynFWA is able to significantly outperform EFWA, and achieves better performance than the latest SPSO version SPSO2011.**

I. Introduction

OPTIMIZATION problems can be found in many applications, ranging from the academic field to industrial world problems. The characteristics and requirements of these problems determine whether the overall best solution can be found within a limited time period [1]. Recently, various stochastic, population-based optimization algorithms based on Swarm Intelligence (SI) have been proposed with great success. The problem-solving ability of SI emerges from the interaction of simple information-processing units (either living creatures or lifeless bodies) that collectively work together as a swarm [1]. Inspired by the collective behaviors of swarms in nature, several SI algorithms have been proposed, such as Particle Swarm Optimization [2], Ant Colony Optimization [3], and many more.

The Fireworks Algorithm (FWA) [4] is a recently developed SI algorithm based on simulating the explosion process of real fireworks exploding and illuminating the night sky. In FWA, the fireworks (i.e, individuals) are let off to the potential search space and an explosion process is initiated for each firework. This stochastic *explosion* process is one of the key features of FWA. After the explosion, a shower of sparks fills the local space around the firework. Both fireworks as well as the newly generated sparks represent potential solutions in the search 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. The algorithm continues the search until a termination criterion (time, maximum number of iteration or fitness evaluation, or convergence) is reached.

FWA uses a so called explosion amplitude in order to balance the global and local search. Fireworks located at good positions can generate a large population of explosion sparks within a smaller range, *i.e.,* with a small explosion amplitude. Contrary, fireworks located at positions with lower fitness values can only generate a smaller population within a larger range, *i.e.,* with higher explosion amplitude. 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 use the same notations as in [5] to differentiate between the two distinct types of sparks: "explosion sparks" are generated by the explosion process, and "Gaussian sparks" are generated by Gaussian mutation.

Related work. Since its introduction in [4], FWA has proven its efficiency in dealing with optimization problems. The works based on FWA can be grouped into two categories, algorithm developments and applications.

Algorithm developments include single-objective FWA [5]–[8], multi-objective FWA [9] and parallel FWA implementations [10]. The majority of studies based on FWA have focused on the development of single-objective FWA. Zheng Y. *et al.* [6] proposed a hybrid algorithm FWA-DE between FWA and differential evolution (DE), by including mutation and crossover operators into FWA. In FWA-DE, in each iteration, the selected fireworks perform mutation, crossover and selection in order to create a new set of fireworks which will be used to generate *explosion* and *Gaussian* sparks. Experimental results on six benchmark functions indicate that this hybrid version of FWA and DE outperforms the standard versions of both, FWA and DE. In [7], Pei *et*

Shaoqiu Zheng, Junzhi Li and Ying Tan (corresponding author) are with the 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, ljz, ytan}@pku.edu.cn).

Andreas Janecek is with University of Vienna, Research Group Theory and Applications of Algorithms, 1090 Vienna, Austria (email: andreas.janecek@univie.ac.at)

This work was supported by the National Natural Science Foundation of China under grants number 61375119, 61170057 and 60875080.

978-1-4799-1488-3/14/$31.00 ©2014 IEEE

al. investigated the influence of approximation approaches on accelerating FWA with elite strategies by comparing approximation models, sampling methods, and sampling size. Results indicate that the random sampling method with a two degree polynomial model gains the fastest convergence speed. Additionally, Liu *et al.* [8] invested the influence on FWA when using different numbers of sparks and explosion amplitude strategies. The most significant improvement of FWA can be found in [5], where Zheng S. *et al.* proposed the Enhanced Fireworks Algorithm (EFWA) which incorporates five modifications compared to conventional FWA in order to eliminate the drawbacks of the original algorithm: (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. Zheng Y. *et al.* proposed a framework for multi-objective FWA (MOFWA) in [9], which based on their previous work [6]. In terms of parallel implementation, Ding *et al.* proposed GPU-FWA [10], a GPU implementation with several modification compared to conventional FWA in order reduce the interaction among fireworks.

FWA and applications. FWA has been used for FIR and IIR digital filters design in combination with cultural algorithms [11], the initialization of Non-negative Matrix Factorization (NMF) [12]–[14], parameter optimization in the process of pattern extraction for finger-vein identification [15]. Experimental results in these studies indicate that FWA is a promising swarm SI algorithm which is well suited for optimizing FIR and IIR digital filters [11], allows for fast convergence and low approximation error for NMF [12]–[14], and achieves low equal error rate for pattern extraction tasks [15].

Contributions. In FWA and EFWA, the explosion amplitude value is one of the key parameters which is used to balance between the local and global search capabilities of the algorithm. The fitness of the current location of each firework is used to calculate the explosion amplitude. The main idea is that a firework with *better* fitness (i.e., smaller fitness value for minimization problems [1]) can generate a *larger population* of explosion sparks within a *smaller range*, *i.e.,* with a small explosion amplitude, while fireworks with poorer fitness (i.e., higher fitness value) can only generate a smaller population within a larger range, *i.e.,* with higher explosion amplitude. As a result, fireworks at good locations will perform local search in a narrow range around the current location, while fireworks with higher fitness will perform global search in a wider range.

The minimal explosion amplitude check strategy (MEACS) in EFWA [5] introduced a lower-bound for this explosion amplitude in order to avoid that the explosion amplitude of the best firework found so far is set to zero. This strategy decreases the explosion amplitude solely with the current number of function evaluations, which heavily depends on the predefined number of iterations for the algorithm. Experimental results indicate that this strategy does not allow for efficient local search around the currently best solution. Therefore, in this paper, we present an appropriate strategy for varying the explosion amplitude dynamically based on the current success of the optimization process. Additionally, we show that it is possible to remove the rather time-consuming Gaussian sparks operator of EFWA in dynFWA without loss in optimization accuracy. The proposed dynFWA algorithm significantly improves the optimization results of EFWA and also significantly reduces the computational cost.

Synopsis. Section II briefly introduces the framework of EFWA, and properties of the minimal explosion amplitude check strategy of EFWA are discussed in detail in Section III. Section IV describes the dynamic explosion amplitude strategy of the proposed algorithm. Experimental results based on the CEC 2013 benchmark suite are presented in Section V, and concluding remarks are drawn in Section VI.

II. The Framework of EFWA

In this paper, EFWA [5] is used as baseline algorithm for dynFWA. As already mentioned, EFWA was proposed as an extension of conventional FWA [4] to overcome some of the limitations inherent in the original algorithm. EFWA consists of five different operators which are briefly summarized in the following (for details see [5]).

A. Minimal Explosion Amplitude Check

In order to automatically balance between exploration and exploitation capabilities of the algorithm, 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 number of explosion sparks s (Eq. 1) and the explosion amplitude A (Eq. 2) for each firework X_i are calculated as follows:

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

$$A_i = \hat{A} \cdot \frac{f(X_i) - y_{min} + \varepsilon}{\sum_{i=1}^{N}(f(X_i) - y_{min}) + \varepsilon} \tag{2}$$

where $y_{max} = max(f(X_i))$, $y_{min} = min(f(X_i))$, $f(X_i)$ denotes the fitness of firework i, and M_e and $\hat{A}$ are two constants to control the explosion amplitude and the number of explosion sparks, respectively, and ε is the machine epsilon. Additionally, the number of sparks s_i that can be generated by each firework is limited by an upper bound. Eq. 2 reveals that the explosion amplitude of the firework at the best location (i.e., smallest fitness value for minimization problems) will usually be very small (close to 0). If the explosion amplitude of the firework at the best location X_b is zero, the explosion sparks will be located at the same location as X_b. As a result, these sparks cannot improve the location of X_b. To overcome this problem, EFWA uses

[1] In this paper, without loss of generality, the optimization problem f is assumed to be a minimization problem.

a *minimal explosion amplitude check strategy* (MEACS) in order to bound the explosion amplitude A_i^k of each firework i in each dimension k as follows:

$$A_i^k = \begin{cases} A_{\min}^k & \text{if } A_i^k < A_{\min}^k, \\ A_i^k & otherwise, \end{cases} \tag{3}$$

where $A_{\min}^k$ decreases non-linearly with increasing number of function evaluations such that

$$A_{\min}^k = A_{init} - \frac{A_{init} - A_{final}}{e_{max}} \sqrt{(2e_{max} - t\,)\,t}, \tag{4}$$

where t refers to the number of function evaluation at the beginning of the current iteration, and e_{max} is the maximum number of evaluations. A_{init} and A_{final} are the initial and final minimum explosion amplitude, respectively.

B. Explosion Sparks Operator

Now, each firework explodes and creates *explosion sparks* within a given range around its current location. A firework with better fitness can generate a larger population of explosion sparks within a smaller range, *i.e.,*, with a small explosion amplitude. For each of the s_i explosion sparks of each firework X_i, Algorithm 1 is performed once.

Algorithm 1 – Generating "explosion sparks" in EFWA

1: Initialize 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
8: **end if**
9: **end for**

C. Mapping Operator

When the location of a new spark exceeds the search range in dimension k, this spark will be mapped to another location within the search space (in dimension k) with uniform distribution according to $\bar{X}_i^k = X_{min}^k + rand * (X_{max}^k - X_{min}^k)$.

D. Gaussian Sparks Operator

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. Algorithm 2 describes how Gaussian sparks are calculated. 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).

Algorithm 2 – Generating "Gaussian sparks" in EFWA

1: Initialize the location of the "Gaussian sparks": $\hat{X}_i = X_i$
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** $\hat{X}_i^k$ out of bounds **then**
7: map $\hat{X}_i^k$ to the potential space
8: **end if**
9: **end for**

E. Selection Operator

EFWA applies the computationally efficient *Elitism-Random Selection* (ERP, [1]). Although this method is rather simple compared to distance based selection operators that aim at selecting very divers individuals (cf. [4]), analysis in [5] has shown that there is almost no difference in terms of convergence, final fitness and standard deviation.

III. Properties of Minimal Explosion Amplitude Check Strategy (MEACS) in EFWA

For simplicity we use the following definitions:

Core Firework(CF): In each iteration, the firework at the currently best location is marked as core firework (CF). Thus, for minimization problems, among the set C of all fireworks the firework X_{CF} is selected as CF *iff*

$$\forall X_i \in C: f(X_{CF}) \leq f(X_i) \tag{5}$$

Local Minimum Space and Local Minimum Point: Given an objective function f, in a continuous space $\Psi \subseteq \Omega$, there $\exists$ only one point $\mathbf{x}$, $\exists\varepsilon$, and $f(x_i) - f(\mathbf{x}) \geqslant 0$, for $\forall x_i$, $|x_i - \mathbf{x}| \leqslant \varepsilon$, then $\mathbf{x}$ is a local minimum point. For region S, if these is only one local minimal point in it, then S is a local minimum space.

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 poorer fitness can only generate a smaller population within a larger range, *i.e.,* with higher explosion amplitude. This allows to balance between exploration and exploitation capabilities of the algorithm. *Exploration* refers to the ability of the algorithm to explore various regions of the search space in order to locate promising good solutions, while *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. [16]). Exploration is achieved by those fireworks which have a large explosion amplitude (*i.e.,* poorer 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, better fitness), since they reinforce the local search ability in promising areas.

In EFWA, the MEACS (cf. Section II-A) enforces the exploration capabilities at the early phase of the algorithm

(larger $A^k_{\min} \Rightarrow$ global search), while at the final phase of the algorithm the exploitation capabilities are enforced (smaller $A^k_{\min} \Rightarrow$ local search). This is further enforced by the non-linear decrease of $A^k_{\min}$ (cf. Eq. 4). Obvisouly, this procedure decreases the explosion amplitude solely with the current number of function evaluations which heavily depends on the pre-defined number of iterations for the algorithm. The explosion amplitude strategy should consider the optimization process information rather than solely the information about the current iteration (or evaluation) count. In order to tackle this problem we propose a dynamic explosion strategy for the CF in order to enhance the local search ability.

IV. The dynFWA

In dynFWA, fireworks are separated into two groups. The first group consists of the CF, while the second group consists of all remaining fireworks. The responsibility of the CF is to perform a local search around the best location found so far, while the responsibility of the second group is to maintain the global search ability. For both groups, the explosion amplitude is a key feature in order to efficiently and effectively improve the current locations of the fireworks. However, for the CF the selection of the explosion amplitude is even more important due to its high influence the convergence speed towards the local minimum point within the local minimum space and the property that it is always selected in the optimization process. Contrary to EFWA, in dynFWA the explosion amplitude of the CF is not calculated by Eq. 2. Instead, the local information of the optimization process (i.e., the information if the algorithm has improved its best location during the last iterations) is used for the calculation of the explosion amplitude for the CF. For all fireworks in the second group (non-CFs), the explosion amplitude is calculated similarly to EFWA, *i.e.,* by using Eq. 2, however, without using the minimum explosion amplitude check from Eq. 3. Recall that the explosion amplitude influences the calculation of the *explosion sparks*, while it has no impact on the computation of *Gaussian sparks*. In Section IV-D we further discuss that in dynFWA it is possible to completely remove the Gaussian sparks operator without a loss in accuracy.

A. Dynamic Explosion Amplitude for the First Group (CF)

The CF stores the best solution found so far. Let us define $\hat{X}_b$ as the "best" newly created explosion spark of all fireworks in the swarm, and $\Delta_f = f(\hat{X}_b) - f(X_{CF})$. Based on the value of Δ_f, there are two situations:

1) One or several explosion sparks have found a better position, *i.e.*, $\Delta_f < 0$ (for minimization problems).

It is possible that (i) an explosion spark generated by the CF has found the best position, or that (ii) an explosion spark generated by a *different* firework than the CF has found the best position. Both cases indicate that the swarm has found a new promising position and that $\hat{X}_b$ will be the CF for the next iteration.

(i) In most cases, $\hat{X}_b$ has been created by the CF. In such cases, in order to speed up the convergence of the algorithm, the explosion amplitude of the CF for the next iteration will be increased compared to the current iteration. Figures 1(a) and 1(b) illustrate this situation.

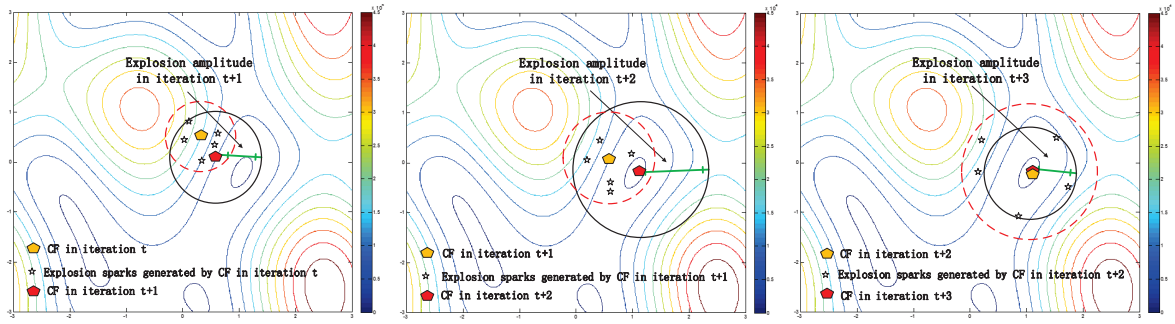
(ii) In other cases – though with a lower probability – a firework different from the CF will create $\hat{X}_b$. This situation happens more frequently during the earlier phase of the optimization process than during later iterations. In such cases, $\hat{X}_b$ will become the new CF for the next iteration (recall that the best location among all individuals is always selected for the next iteration). Since the position of the CF is changed, the current explosion amplitude which considers the optimization information of the position of the current CF will not be effective to the newly selected CF ($\hat{X}_b$). However, it is possible that $\hat{X}_b$ is located in rather close proximity to the previous CF: since the CF creates the large number of sparks among all fireworks, the random selection method may select several sparks created by the CF, which are initially located in close proximity to the CF. If so, the same consideration as in (i) applies, and the explosion amplitude of the CF will be increased. If $\hat{X}_b$ is created by a firework which is not in close proximity to the CF, the explosion amplitude can be re-initialized to the pre-defined value. However, since it is difficult to define "close" proximity, we do not compute the distance between $\hat{X}_b$ and X_{CF} but rely on the dynamic explosion amplitude update ability. Similarly to (i), the explosion amplitude is increased. If the new CF cannot improve its location in the next iteration, the new CF is able to adjust the explosion amplitude itself dynamically in the following iterations.

We underline the idea why an increasing explosion amplitude may accelerate the convergence speed: Assume that the current position of the CF is far away from the global/local minimum. Increasing the explosion amplitude is a direct and effective approach in order to increase the step-size towards the global/local optimum in each iteration, *i.e.*, it allows for faster movements towards the optimum. However, we note that usually the probability to find a position with better fitness decreases with increasing explosion amplitude due to the increased search space (obviously, this depends to a large extent on the optimization function).

2) None of the explosion sparks of the CF nor of all other fireworks has found a position with better fitness compared to the CF, *i.e.*, $\Delta_f >= 0$. In this situation, the explosion amplitude of the CF is reduced in order to narrow down the search to a smaller region around the current location and to enhance the exploitation capability of the local search of the CF. The probability to find a position with better fitness usually increases with decreasing explosion amplitude. Figure 1(c) illustrates the situation where the location of CF could not be improved.

B. Discussion

Algorithm 3 summarizes the dynamic update strategy as discussed in Section IV-A. Figure 2 shows the process of the amplification/reduction during the optimization of the sphere function for 1 000 iterations (algorithm dynFWA). As can be



(a) Iteration $t+1$ (b) Iteration $t+2$ (c) Iteration $t+3$

Fig. 1. Illustration of the amplification/reduction of the CF's explosion amplitude. In Figure 1(a), the radius of the circle with dashed red line indicates the explosion amplitude of the CF in iteration t, while the cirlce with solid black line indicates the explosion amplitude in iteration $t+1$; the increased explosoin amplitude indicates that in this situation, a better position has been found by the explosion sparks. In iteration $t+2$ (cf. Figure 1(b)), the CF is able to further improve its location, and, as a result, the explosion amplitude of the CF is further increased. Figure 1(c) shows an example when the fitness of the CF could not be improved. In this case, the CF's explosion amplitude is decreased in iteration $t+3$.

Algorithm 3 Dynamic explosion amplitude update for CF

Initialization: Define:

X_{CF} is the current location of the CF;
$\hat{X}_b$ is the best location among all explosion sparks;
A_{CF} is the current explosion amplitude of the CF;
C_a is the amplification coefficient;
C_r is the reduction coefficient;

Iteration:

1: **if** $f(\hat{X}_b) - f(X_{CF}) < 0$ **then**
2: $A_{CF} \leftarrow A_{CF} * C_a$;
3: **else**
4: $A_{CF} \leftarrow A_{CF} * C_r$;
5: **end if**

seen from Figure 2, the reduction and amplification explosion amplitudes happen in an alternating manner. Obviously, there are more reduction than amplification phases, which is partly caused by the values of C_a and C_r, which are set to 1.2 and 0.9, respectively, and also by the fact that the explosion amplitude is initially set to the size of the search space, which is a rather large number in the first iteration (cf. Section V-A).

In the following, we discuss why a reduction of the explosion amplitude increases the probability to find a better location. A Taylor series is used to represent the properties of the local region around the CF. Assume a continuously differentiable second-order optimization function g with k dimensions: if the position of the CF is *not* a local/global minimal point, and A_{CF} is the current explosion amplitude, then $g(\mathbf{x}) - g(X_{CF}) = \nabla g(X_{CF})^T(\mathbf{x} - X_{CF}) + \frac{1}{2}(\mathbf{x} - X_{CF})H(\mathbf{x})(\mathbf{x} - X_{CF})$, where $H(\mathbf{x}) = [\frac{\partial^2 g}{\partial x_i \partial x_j}]_{k\times k}$. According to the definition of "local/global minimal point" in Section III (i.e. X_{CF} is not a local/global minimum point), there $\exists\varepsilon$, $\forall\mathbf{x}$ in $S = \{\mathbf{x}|\,|\mathbf{x} - X_{CF}| \leqslant \varepsilon\}$ and $g(\mathbf{x}) - g(X_{CF}) = \nabla g(X_{CF})^T(\mathbf{x} - X_{CF}) + o(\nabla g(X_{CF})^T(\mathbf{x} - X_{CF}))$, where o means low order. From the Taylor series, if $\varepsilon \to 0$, then in region S, if there exists a point $\mathbf{x}_1$ and $\mathbf{x}_1 - X_{CF} = \Delta\mathbf{x}$, then there exists a point $\mathbf{x}_2$ and $\mathbf{x}_2 - X_{CF} = -\Delta\mathbf{x}$. Under this circumstance, the probability of generating a spark with smaller fitness than the CF is very high (i.e. $(g(\mathbf{x}_1) - g(X_{CF})) * (g(\mathbf{x}_2) - g(X_{CF})) < 0$). In case the CF does not find a better position while generating a number of sparks, it is likely that $A_{CF} \geqslant \varepsilon$. We cannot expect the property of region $T = \{\mathbf{x}|\varepsilon \leqslant |\mathbf{x} - X_{CF}| \leqslant A_{CF}\}$ that in region T whether there exists a position with better fitness compared to the CF, thus, if the CF generates sparks with uniform distribution in each dimension, the probability p' that a spark is located in S is $p' = \frac{\|S\|}{\|S\|+\|T\|}$, where $\|\|$ denotes the hypervolume of this region. If the CF does not find a better position, the explosion amplitude A_{CF} is reduced in order to increase the probability p' that the CF can generate a spark in region S thus to increase the probability that finding a point with smaller fitness than CF.

C. Explosion Amplitude for the Second Group (non-CF)

The explosion amplitudes for non-CF fireworks are calculated based on Eq. 2, *i.e.*, similar to EFWA but without the minimum explosion amplitude check strategy. Compared to the CF, these non-CF fireworks can only create a smaller number of explosion sparks within a larger explosion amplitude in order to perform the global search for the swarm. In situations where the CF gets stuck in local minima, this group of fireworks may be able to allow the algorithm to escape from premature convergence, since these fireworks continue the search in different areas of the search space.

D. Elimination of the Gaussian Sparks Operator

The motivation behind the Gaussian sparks (cf. [4]) is to further increase the diversity of the swarm. In EFWA, the Gaussian sparks are calculated by $X_i^k = X_i^k + (X_b^k - X_i^k) \times e$, where X_b is the location of the currently best firework(denoted), and $e = Gaussian(0, 1)$ (cf. Section II-D). From Figure 3, it can be seen that the newly generated sparks will be located along the direction between a selected firework i and the CF. Any newly generated Gaussian sparks will be either 1) close to the CF, 2) close to firework i, or 3) located along the direction between the CF and firework

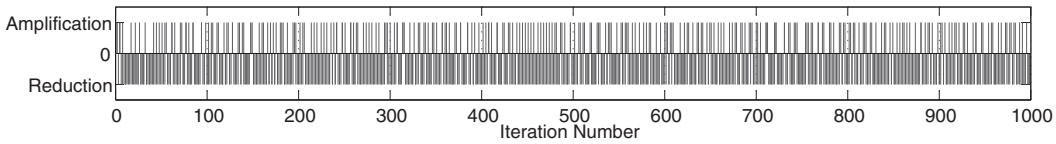


Fig. 2. The reduction and amplification of CF's explosion amplitude (using function f_1 (sphere function) from [17], cf. Section V)

i, however, with some distance to both, the CF and firework i. In the first two situations, the operator will have similar performance as the explosion sparks generated by the CF and firework i, respectively. In the third situation, the Gaussian spark can be interpreted as an explosion spark generated by a firework with large explosion amplitude. Thus, based on the above analysis it can be stated that in many situations Gaussian sparks will not be able to effectively increase the diversity of the swarm.

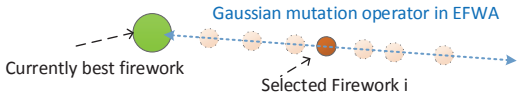


Fig. 3. Gaussian sparks operator in EFWA

E. Framework of dynFWA

Based on the operators discussed in the previous subsections, the final algorithm called Dynamic Search Fireworks Algorithm (dynFWA) is presented in Algorithm 4.

Algorithm 4 Framework of dynFWA

```
Initialize N fireworks and evaluate the quality
Initialize the explosion amplitude for CF
while termination criteria are not met do
    Calculate number of explosion sparks (cf. Eq 1)
    Calculate explosion amplitude for non-CF (cf. Eq. 2)
    for each firework do
        Generate explosion sparks (cf. Section II-B)
        Map sparks at invalid locations back to search space
        (cf. Section II-C)
        Evaluate quality of explosion sparks
    end for
    Update explosion amplitude of CF (cf. Alg.3)
    Select N fireworks for next iteration (cf. Section II-E)
end while
```

V. Experiments

To investigate the performance of the proposed dynFWA algorithm as well as the performance of the Gaussian mutation operator (*i.e.*, the performance after removing this operator), we compare two EFWA variants and two dynFWA variants. Each algorithm is tested with and without Gaussian mutation operator, respectively. Besides comparing dynFWA and EFWA, the most recent version of SPSO (SPSO2011, [18]) is used for performance comparison. In the following we briefly describe the five algorithms used for experimental evaluation:

- EFWA – the baseline algorithm as presented in [5] [2];
- EFWA-NG – in this algorithm, the Gaussian sparks operator has been removed from EFWA;
- dynFWA-G – this algorithm implements the dynFWA algorithm as described in Section IV *including* the Gaussian mutation operator.
- dynFWA – similar as dynFWA-G but *without* Gaussian mutation operator.
- SPSO2011 – the most recent SPSO variant. Compared to earlier versions of SPSO it features an improved velocity update by exploiting the idea of *rotational invariance* for the velocity update instead of sequential dimension-by-dimension update of older versions of SPSO (cf. [18]). [3]

A. Experimental Setup

Similar to EFWA, the number of fireworks in dynFWA is set to 5, but in dynFWA, the maximum number of explosion sparks (M_e in Eq. 1) in each iteration is set to 150. The reduction and amplification factors C_r and C_a of dynFWA are empirically set to 0.9 and 1.2, respectively, and A_{CF} is initially set to the size of the search space in order to maintain a high exploration capability at the beginning. All other parameters for dynFWA and all EFWA parameters are identical to [5], SPSO2011 parameters are listed in [18]. For each algorithm we performed 51 runs on each optimization function; the final mean results after 300 000 function evaluations are presented. As experimental platform we used MATLAB 2011b (Windows 7; Intel Core i7-2600 CPU @ 3.7 GHZ; 8 GB RAM). To validate the performance of the proposed algorithms we used the recent CEC 2013 benchmark suite that includes 28 different benchmark functions as listed in [17].

B. Experimental Results

In this section, we first evaluate the influence of removing the Gaussian sparks operator as discussed in Section IV-D. After that we evaluate the performance of dynFWA and compare its performance to EFWA and also SPSO2011.

[2] We note that experimental results in the EFWA paper [5] are unintentionally based on the dimension selection method of conventional FWA [4], which varies the number of adapted dimensions uniformly among all dimension. More details can be found at `http://www.cil.pku.edu.cn/research/FWA/index.html`. All results and pseudo-codes in this paper are based on the published pseudo-code in [5].

[3] In addition to the results (median, maximum, minimal) in [18], detailed results of SPSO2011 which include the results of each single run were also submitted to the *CEC2013 competition* held by P. N. Suganthan. These results are available at `http://goo.gl/pXB1WH`. The mean fitness error results in our paper are computed based on the results in the folder "1534" of file "Results-of-22-papers.zip"

TABLE I

WILCOXON SIGNED-RANK TEST RESULTS FOR EFWA vs EFWA-NG AND DYNFWA-G vs DYNFWA (BOLD VALUES INDICATE THE PERFORMANCE DIFFERENCE IS SIGNIFICANT).

F.	EFWA vs EFWA-NG			dynFWA-G vs dynFWA		
	EFWA	EFWA-NG	p-value	dynFWA-G	dynFWA	p-value
f_1	-1.3999E+03	-1.3999E+03	**2.316E-03**	-1.4000E+03	-1.4000E+03	1.000E+00
f_2	6.8926E+05	6.5258E+05	4.256E-01	7.6981E+05	8.6937E+05	1.801E-01
f_3	7.7586E+07	6.4974E+07	8.956E-01	1.2007E+08	1.2317E+08	6.393E-01
f_4	-1.0989E+03	-1.0989E+03	7.858E-01	-1.0863E+03	-1.0896E+03	**3.183E-02**
f_5	-9.9992E+02	-9.9992E+02	**4.290E-02**	-1.0000E+03	-1.0000E+03	1.463E-01
f_6	-8.5073E+02	-8.4462E+02	1.654E-01	-8.6524E+02	-8.6995E+02	9.156E-02
f_7	-6.2634E+02	-6.2991E+02	9.552E-01	-6.9946E+02	-7.0010E+02	6.663E-01
f_8	-6.7907E+02	-6.7906E+02	9.776E-01	-6.7909E+02	-6.7910E+02	4.997E-01
f_9	-5.6846E+02	-5.6889E+02	5.178E-01	-5.7435E+02	-5.7587E+02	1.711E-01
f_{10}	-4.9916E+02	-4.9918E+02	3.732E-01	-4.9994E+02	-4.9995E+02	3.591E-01
f_{11}	5.8198E+00	3.5430E+01	5.830E-02	-2.9978E+02	-2.9589E+02	6.127E-01
f_{12}	3.9944E+02	4.1107E+02	6.193E-01	-1.4993E+02	-1.4222E+02	4.762E-01
f_{13}	2.9857E+02	2.8909E+02	8.220E-01	5.4523E+01	5.3830E+01	8.513E-01
f_{14}	2.7240E+03	2.9344E+03	**4.101E-02**	2.8909E+03	2.9180E+03	8.147E-01
f_{15}	4.4595E+03	4.5515E+03	6.869E-01	3.9186E+03	4.0227E+03	4.879E-01
f_{16}	2.0063E+02	2.0056E+02	2.811E-01	2.0056E+02	2.0058E+02	7.358E-01
f_{17}	6.2461E+02	6.3152E+02	9.179E-01	4.5397E+02	4.4261E+02	1.197E-01
f_{18}	5.7361E+02	5.6953E+02	6.938E-01	5.8801E+02	5.8782E+02	8.660E-01
f_{19}	5.1022E+02	5.1012E+02	9.402E-01	5.0750E+02	5.0726E+02	6.193E-01
f_{20}	6.1466E+02	6.1457E+02	**1.559E-02**	6.1309E+02	6.1328E+02	3.632E-01
f_{21}	1.1178E+03	1.1362E+03	**6.910E-04**	9.9532E+02	1.0102E+03	6.431E-01
f_{22}	6.3181E+03	6.3674E+03	9.776E-01	4.1463E+03	4.1262E+03	9.402E-01
f_{23}	7.5809E+03	7.5707E+03	7.217E-01	5.6661E+03	5.6526E+03	9.402E-01
f_{24}	1.3452E+03	1.3611E+03	**1.079E-02**	1.2738E+03	1.2729E+03	8.586E-01
f_{25}	1.4426E+03	1.4435E+03	8.734E-01	1.3964E+03	1.3970E+03	8.882E-01
f_{26}	1.5461E+03	1.5400E+03	2.687E-01	1.4744E+03	1.4607E+03	1.337E-01
f_{27}	2.6210E+03	2.5780E+03	3.534E-01	2.2721E+03	2.2804E+03	8.147E-01
f_{28}	4.7651E+03	4.9949E+03	6.460E-01	1.7686E+03	1.6961E+03	3.555E-01

TABLE II

MEAN FITNESS ON THE BENCHMARK FUNCTIONS AND MEAN FITNESS RANK OF SPSO2011, EFWA AND DYNFWA.

F.	SPSO2011	Rank	EFWA	Rank	dynFWA	Rank
f_1	**-1.4000E+03**	**1**	-1.3999E+03	3	**-1.4000E+03**	**1**
f_2	**3.3719E+05**	**1**	6.8926E+05	2	8.6937E+05	3
f_3	2.8841E+08	3	**7.7586E+07**	**1**	1.2317E+08	2
f_4	3.7543E+04	3	**-1.0989E+03**	**1**	-1.0896E+03	2
f_5	**-1.0000E+03**	**1**	-9.9992E+02	3	-1.0000E+03	2
f_6	-8.6210E+02	2	-8.5073E+02	3	**-8.6995E+02**	**1**
f_7	**-7.1208E+02**	**1**	-6.2634E+02	3	-7.0010E+02	2
f_8	-6.7908E+02	2	-6.7907E+02	3	**-6.7910E+02**	**1**
f_9	-5.7123E+02	2	-5.6846E+02	3	**-5.7587E+02**	**1**
f_{10}	-4.9966E+02	2	-4.9916E+02	3	**-4.9995E+02**	**1**
f_{11}	-2.9504E+02	2	5.8198E+00	3	**-2.9589E+02**	**1**
f_{12}	**-1.9604E+02**	**1**	3.9944E+02	3	-1.4222E+02	2
f_{13}	**-6.1406E+00**	**1**	2.9857E+02	3	5.3830E+01	2
f_{14}	3.8910E+03	3	**2.7240E+03**	**1**	2.9180E+03	2
f_{15}	**3.9093E+03**	**1**	4.4595E+03	3	4.0227E+03	2
f_{16}	2.0131E+02	3	2.0063E+02	2	**2.0058E+02**	**1**
f_{17}	**4.1626E+02**	**1**	6.2461E+02	3	4.4261E+02	2
f_{18}	**5.2063E+02**	**1**	5.7361E+02	2	5.8782E+02	3
f_{19}	5.0951E+02	2	5.1022E+02	3	**5.0726E+02**	**1**
f_{20}	6.1346E+02	2	6.1466E+02	3	**6.1328E+02**	**1**
f_{21}	**1.0088E+03**	**1**	1.1178E+03	3	1.0102E+03	2
f_{22}	5.0988E+03	2	6.3181E+03	3	**4.1262E+03**	**1**
f_{23}	5.7313E+03	2	7.5809E+03	3	**5.6526E+03**	**1**
f_{24}	**1.2667E+03**	**1**	1.3452E+03	3	1.2729E+03	2
f_{25}	1.3993E+03	2	1.4426E+03	3	**1.3970E+03**	**1**
f_{26}	1.4861E+03	2	1.5461E+03	3	**1.4607E+03**	**1**
f_{27}	2.3046E+03	2	2.6210E+03	3	**2.2804E+03**	**1**
f_{28}	1.8013E+03	2	4.7651E+03	3	**1.6961E+03**	**1**
Mean Rank						
	SPSO2011	1.75	EFWA	2.68	dynFWA	**1.54**

Evaluation of Gaussian sparks operator. To evaluate whether the results of EFWA and dynFWA improve or deteriorate after removing the Gaussian sparks operator we compare the results of EFWA and EFWA-NG, and the results of dynFWA-G and dynFWA, respectively. In order to validate the improvement between any two algorithms, the Wilcoxon signed-rank test is conducted (cf. Section V-B). Assume that data X, Y are single fitness results for a given number of runs of two different algorithms. If the mean value of X is smaller than the mean value of Y and the Wilcoxon signed-rank test under 5% significance level is true, then it is believed that the results of X are significant better than Y. A comparison between EFWA $vs.$ EFWA-NG and dynFWA-G $vs.$ dynFWA are given in Table I:

- EFWA-NG performs slighty better than EFWA on 16 functions, while it performs slightly worse than EFWA on 12 functions. However, for 5 functions the Wilcoxon signed-rank test indicates that EFWA is significantly better than EFWA-NG, while for 1 function EFWA-NG is significant better than EFWA. Hence, for EFWA these results suggest that the Gaussian sparks operator should not be removed, although EFWA-NG is faster in terms of runtime (see next section).
- In general, the performance of dynFWA and dynFWA-G is very similar. Only for function f_4 dynFWA performs significantly better than dynFWA-G. This indicates that dynFWA without the Gaussian sparks operator achieves slightly better results than dynFWA-G, and is also faster in terms of runtime (see next section).

In the rest of this paper we use the best EFWA variant (EFWA *with* Gaussian sparks operator) and the best dynFWA variant (dynFWA *without* Gaussian sparks operator) for further comparison.

Comparison of dynFWA and EFWA. Table II shows the mean fitness value over 51 runs for each function for the three algorithms SPSO2011, EFWA and dynFWA, and the corresponding rank of each algorithm. Moreover, at the bottom of Table II we present the mean fitness rank for each algorithm. Table IV shows the runtime of each algorithm – the runtime of the fastest algorithm (dynFWA) is set to 1, for all other algorithms, the proportional runtimes compared to dynFWA are presented.

A comparison between the proposed dynFWA and EFWA reveals that dynFWA outperforms EFWA in terms of mean fitness, mean fitness rank and runtime. In can be seen that dynFWA achieves better mean fitness results than EFWA on 23 functions except function f_2, f_3, f_4, f_{14}, f_{18}. The mean fitness rank results suggest that dynFWA gains great advantages over EFWA. To test whether the improvement of dynFWA over EFWA is significant or not, a number of Wilcoxon signed-rank tests were conducted and the corresponding p-values are presented in Table III. The results indicate that the improvement of dynFWA is significant compared to EFWA for 22 benchmark functions. Moreover, in terms of computational complexity it can be seen that dynFWA significantly reduces the runtime of EFWA for the same number of function evaluations. This is mostly caused by the removal of the Gaussian sparks operator, which is computationally rather expensive (this is further indicated by a comparison of the runtimes of EFWA *vs.* EFWA-NG).

Comparison of dynFWA and SPSO2011. A comparison between the proposed dynFWA and the most recent SPSO

TABLE III

WILCOXON SIGNED-RANK TEST RESULTS FOR DYNFWA *vs.* EFWA (BOLD VALUES INDICATE THE SIGNIFICANT IMPROVEMENT).

F.	f_1	f_2	f_3	f_4	f_5	f_6	f_7
p-value	**0.00E+00**	6.94E-03	9.90E-02	0.00E+00	**0.00E+00**	**1.58E-03**	**0.00E+00**
F.	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}
p-value	**1.73E-02**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	1.41E-01
F.	f_{15}	f_{16}	f_{17}	f_{18}	f_{19}	f_{20}	f_{21}
p-value	**5.10E-05**	3.20E-01	**0.00E+00**	6.35E-02	**1.41E-04**	**0.00E+00**	**0.00E+00**
F.	f_{22}	f_{23}	f_{24}	f_{25}	f_{26}	f_{27}	f_{28}
p-value	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**

TABLE IV

RUNTIME COMPARISON.

SPSO2007	1.01	EFWA	1.30	dynFWA-G	1.17
SPSO2011	–	EFWA-NG	1.03	dynFWA	**1**

variant indicates dynFWA is able to achieve a better mean rank compared to SPSO2011 (cf. Table II). In total, dynFWA achieves better results (smaller mean fitness) than SPSO2011 on 17 functions, while SPSO2011 is better than dynFWA on 10 functions. For one function the results are identical. In terms of computational complexity we measured the runtime of SPSO2007, however, since the SPSO2011 results are adopted from the literature (see beginning of Section V) we cannot compare the runtimes as they depend on the implementation and the infrastructure. Table IV shows that compared to SPSO2007, dynFWA has almost the same runtime. Since the SPSO2011 operators (new velocity update strategies) appear to be more complex and therefore probably at least as time-consuming as the SPSO2007 operators (cf. [18], [19]), we expect SPSO2011 to have a similar or slightly higher computationally complexity compared to initial version of SPSO2007.

VI. CONCLUSION

In this paper, we have presented the *dynamic search Firework Algorithm (dynFWA)*, an improvement of the recently developed Enhanced Fireworks Algorithm (EFWA). dynFWA uses a dynamic explosion amplitude for the core firework (CF), *i.e.*, the firework at the currently best position. This dynamic explosion amplitude depends on the quality of the current local search around the CF. The main task for the CF is to perform a local search, while the responsibility for all other fireworks are to maintain the global search ability. Additionally, we have analyzed the possibility to remove the rather time-consuming Gaussian sparks operator of EFWA. From result of our experimental evaluation we conclude the following observations:

1) The proposed dynFWA algorithm significantly improves the results of EFWA and also reduces the runtime by more than 20%.
2) Compared with SPSO2011, dynFWA achieves a better mean rank among 28 benchmark functions with similar computational cost.
3) The Gaussian sparks operator of EFWA should not be removed in EFWA. However, removing this operator in dynFWA significantly reduces the runtime of dynFWA without loss in optimization accuracy.

In future work, we plan to further extend this work by incorporating different strategies to balance between the global and local search abilities of dynFWA. Currently, only the fitness is considered to calculate the number of explosion sparks and the explosion amplitude. Moreover, we will focus on designing new interaction strategies between fireworks in the swarm in order to further accelerate the convergence speed of dynFWA.

REFERENCES

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

[2] 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.

[3] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," Computational Intelligence Magazine, IEEE, vol. 1, no. 4, pp. 28–39, 2006.

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

[5] S. Zheng, A. Janecek, and Y. Tan, "Enhanced fireworks algorithm," in Evolutionary Computation (CEC), 2013 IEEE Congress on. IEEE, 2013, pp. 2069–2077.

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

[7] 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.

[8] J. Liu, S. Zheng, and Y. Tan, "The improvement on controlling exploration and exploitation of firework algorithm," in Advances in Swarm Intelligence. Springer, 2013, pp. 11–23.

[9] Y.-J. Zheng, Q. Song, and S.-Y. Chen, "Multiobjective fireworks optimization for variable-rate fertilization in oil crop production," Applied Soft Computing, vol. 13, no. 11, pp. 4253–4263, 2013.

[10] K. Ding, S. Zheng, and Y. Tan, "A gpu-based parallel fireworks algorithm for optimization," in Proceeding of the Fifteenth Annual Conference on Genetic and Evolutionary Computation Conference, ser. GECCO 2013. New York, NY, USA: ACM, 2013, pp. 9–16. [Online]. Available: http://doi.acm.org/10.1145/2463372.2463377

[11] 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.

[12] A. Janecek and Y. Tan, "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.

[13] ——, "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.

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

[15] S. Zheng and Y. Tan, "A unified distance measure scheme for orientation coding in identification." in Information Science and Technology, 2013 IEEE Congress on. IEEE, 2013, pp. 979–985.

[16] 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.

[17] J. Liang, B. Qu, P. Suganthan, and A. G. Hernández-Díaz, "Problem definitions and evaluation criteria for the cec 2013 special session on real-parameter optimization," 2013.

[18] M. Zambrano-Bigiarini, M. Clerc, and R. Rojas, "Standard particle swarm optimisation 2011 at cec-2013: A baseline for future pso improvements," in Evolutionary Computation (CEC), 2013 IEEE Congress on, 2013, pp. 2337–2344.

[19] M. Clerc, "Standard particle swarm optimization, from 2006 to 2011," Particle Swarm Central, 2011.