algorithms

MDPI

Article

Adaptive Mutation Dynamic Search Fireworks Algorithm

Xi-Guang Li, Shou-Fei Han *, Liang Zhao, Chang-Qing Gong and Xiao-Jing Liu

School of Computer, Shenyang Aerospace University, Shenyang 110136, China; lixiguang@sau.edu.cn (X.-G.L.); lzhao@sau.edu.cn (L.Z.); gongchangqing@sau.edu.cn (C.-Q.G.); xjydynl@126.com (X.-J.L.)

* Correspondence: hanshoufei@gmail.com; Tel.: +86-158-0405-7359

Academic Editor: Pierre Leone

Received: 23 February 2017; Accepted: 25 April 2017; Published: 28 April 2017

Abstract: The Dynamic Search Fireworks Algorithm (dynFWA) is an effective algorithm for solving optimization problems. However, dynFWA easily falls into local optimal solutions prematurely and it also has a slow convergence rate. In order to improve these problems, an adaptive mutation dynamic search fireworks algorithm (AMdynFWA) is introduced in this paper. The proposed algorithm applies the Gaussian mutation or the Levy mutation for the core firework (CF) with mutation probability. Our simulation compares the proposed algorithm with the FWA-Based algorithms and other swarm intelligence algorithms. The results show that the proposed algorithm achieves better overall performance on the standard test functions.

Keywords: dynamic search fireworks algorithm; Gaussian mutation; Levy mutation; mutation probability; standard test functions

1. Introduction

Fireworks Algorithm (FWA) [1] is a new group of intelligent algorithms developed in recent years based on the natural phenomenon of simulating fireworks sparking, and can solve some optimization problems effectively. Compared with other intelligent algorithms such as particle swarm optimization and genetic algorithms, the FWA adopts a new type of explosive search mechanism, to calculate the explosion amplitude and the number of explosive sparks through the interaction mechanism between fireworks.

However, many researchers quickly find that traditional FWA has some disadvantages in solving optimization problems; the main disadvantages include slow convergence speed and low accuracy, thus, many improved algorithms have been proposed. So far, research on the FWA has concentrated on improving the operators. One of the most important improvements of the FWA is the enhanced fireworks algorithm (EFWA) [2], where the operators of the conventional FWA were thoroughly analyzed and revised. Based on the EFWA, an adaptive fireworks algorithm (AFWA) [3] was proposed, which was the first attempt to control the explosion amplitude without preset parameters by detecting the results of the search process. In [4], a dynamic search fireworks algorithm (dynFWA) was proposed which divided the fireworks into core firework and non-core fireworks according to the fitness value and adaptive adjustment of the explosion amplitude for the core firework. Based on the analysis of each operator of the fireworks algorithm, an improvement of fireworks algorithm (IFWA) [5] was proposed. Since the FWA was proposed, it has been applied to many areas [6], including digital filter design [7], nonnegative matrix factorization [8], spam detection [9], image identification [10], mass minimization of trusses with dynamic constraints [11], clustering [12], power loss minimization and voltage profile enhancement [13], etc.

The aforementioned dynFWA variants can improve the performance of FWA to some extent. However, the inhibition of premature convergence and solution accuracy improvement are still challenging issues that require further research on dynFWA.

In this paper, an adaptive mutation dynamic search fireworks algorithm (AMdynFWA) is presented. In AMdynFWA, the core firework chooses either Gaussian mutation or Levy mutation based on the mutation probability. When it chooses the Gaussian mutation, the local search ability of the algorithm will be enhanced, and by choosing Levy mutation, the ability of the algorithm to jump out of local optimization will be enhanced.

The paper is organized as follows. In Section 2, the dynamic search fireworks algorithm is introduced. The AMdynFWA is presented in Section 3. The simulation experiments and analysis of the results are given in detail in Section 4. Finally, the conclusion is summarized in Section 5.

2. Dynamic Search Fireworks Algorithm

The AMdynFWA is based on the dynFWA because it is very simple and it works stably. In this section, we will briefly introduce the framework and the operators of the dynFWA for further discussion.

Without the loss of generality, consider the following minimization problem:

$$\min f(x) \tag{1}$$

The object is to find an optimal x with a minimal evaluation (fitness) value.

In dynFWA, there are two important components: the explosion operator (the sparks generated by the explosion) and the selection strategy.

2.1. Explosion Operator

Each firework explodes and generates a certain number of explosion sparks within a certain range (explosion amplitude). The numbers of explosion sparks (Equation (2)) are calculated according to the qualities of the fireworks.

For each firework X_i, its explosion sparks' number is calculated as follows:

$$S_i = m \times \frac{y_{\max} - f(X_i) + \varepsilon}{\sum_{i=1}^{N} (y_{\max} - f(X_i)) + \varepsilon} \tag{2}$$

where $y_{\max} = \max(f(X_i))$, m is a constant to control the number of explosion sparks, and ε is the machine epsilon to avoid S_i equal to 0.

In order to limit the good fireworks that do not produce too many explosive sparks, while the poor fireworks do not produce enough sparks, its scope S_i is defined as.

$$S_i = \begin{cases} round(a \times m), & S_i < a \times m \\ round(b \times m), & S_i > b \times m \\ round(S_i), & otherwise \end{cases} \tag{3}$$

where a and b are fixed constant parameters that confine the range of the population size.

In dynFWA, fireworks are divided into two types: non-core fireworks and core firework, and the core firework (CF) is the firework with the best fitness, and is calculated by Equation (4).

$$X_{CF} = \min f(x_i) \tag{4}$$

The calculations of the amplitude of the non-core fireworks and the core firework are different. The non-core fireworks' explosion amplitudes (except for CF) are calculated just as in the previous versions of FWA:

$$A_i = A \times \frac{f(X_i) - y_{\min} + \varepsilon}{\sum_{i=1}^{N} (f(X_i) - y_{\min}) + \varepsilon} \tag{5}$$

where $y_{\min} = \min f(X_i)$, A is a constant to control the explosion amplitude, and ε is the machine epsilon to avoid A_i equal to 0.

However, for the CF, its explosion amplitude is adjusted according to the search results in the last generation:

$$A_{CF}(t) = \begin{cases} A_{CF}(1) & t = 1 \\ C_r A_{CF}(t-1) & f(X_{CF}(t)) = f(X_{CF}(t-1)) \\ C_a A_{CF}(t-1) & f(X_{CF}(t)) < f(X_{CF}(t-1)) \end{cases} \tag{6}$$

where $A_{CF}(t)$ is the explosion amplitude of the CF in generation t. In the first generation, the CF is the best among all the randomly initialized fireworks, and its amplitude is preset to a constant number which is usually the diameter of the search space.

Algorithm 1 describes the process of the explosion operator in dynFWA.

Algorithm 1. Generating Explosion Sparks

```
Calculate the number of explosion sparks S_i
Calculate the non-core fireworks of explosion amplitude A_i
Calculate the core firework of explosion amplitude A_CF
Set z = rand (1, d)
For k = 1:d do
  If k ∈ z then
    If X_j^k is core firework then
      X_j^k = X_j^k + rand (0, A_CF)
    Else
      X_j^k = X_j^k + rand (0, A_i)
    If X_j^k out of bounds
      X_j^k = X_min^k + | X_j^k | % (X_max^k − X_min^k)
    End if
  End if
End for
```

Where the operator % refers to the modulo operation, and $X_{min}{}^k$ and $X_{max}{}^k$ refer to the lower and upper bounds of the search space in dimension k.

2.2. Selection Strategy

In dynFWA, a selection method is applied, which is referred to as the Elitism-Random Selection method. In this selection process, the optima of the set will be selected firstly. Then, the other individuals are selected randomly.

3. Adaptive Mutation Dynamic Search Fireworks Algorithm

The mutation operation is an important step in the swarm intelligence algorithm. Different mutation schemes have different search characteristics. Zhou pointed out that the Gaussian mutation has a strong local development ability [14]. Fei illustrated that the Levy mutation not only improves the global optimization ability of the algorithm, but also helps the algorithm jump out of the local optimal solution and keeps the diversity of the population [15]. Thus, combining the Gaussian mutation with the Levy mutation is an effective way to improve the exploitation and exploration of dynFWA.

For the core firework, for each iteration, two mutation schemes are alternatives to be conducted based on a probability p. The new mutation strategy is defined as:

$$X'_{CF} = \begin{cases} X_{CF} + X_{CF} \otimes Gaussian(), & if\ E < p \\ X_{CF} + X_{CF} \otimes Levy(), & Otherwise \end{cases} \quad (7)$$

where p is a probability parameter, X_{CF} is the core firework in the current population, and the symbol $\otimes$ represents the dot product. *Gaussian*() is a random number generated by the normal distribution with mean parameter $mu = 0$ and standard deviation parameter $sigma = 1$, and *Levy*() is a random number generated by the Levy distribution, and it can be calculated with the parameter $\beta = 1.5$ [16]. The value of E varies dynamically with the evolution of the population, with reference to the annealing function of the simulated annealing algorithm, and the value of E is expected to change exponentially, and it is calculated as follows:

$$E = e^{-(2t/T_{\max})^2} \quad (8)$$

where t is the current function evaluations, and T_{max} is the maximum number of function evaluations.

To sum up, another type of sparks, the mutation sparks, are generated based on an adaptive mutation process (Algorithm 2). This algorithm is performed N_m times, each time with the core firework X_{CF} (N_m is a constant to control the number of mutation sparks).

Algorithm 2. Generating Mutation Sparks

```
Set the value of mutation probability p
Find out the core firework X_CF in current population
Calculate the value of E by Equation (8)
Set z = rand (1, d)
For k = 1:d do
  If k ∈ z then
    Produce mutation spark X_CF' by Equation (7)
    If X_CF' out of bounds
      X_CF' = X_min + rand * (X_max − X_min)
    End if
  End if
End for
```

Where d is the number of dimensions, $X_{\min}$ is the lower bound, and $X_{\max}$ is the upper bound.

As Figure 1 shows, the Levy mutation has a stronger perturbation effect than the Gaussian mutation. In the Levy mutation, the occasional larger values can effectively help jump out of the local optimum and keep the diversity of the population. On the contrary, the Gaussian mutation has better stability, which improves the local search ability.

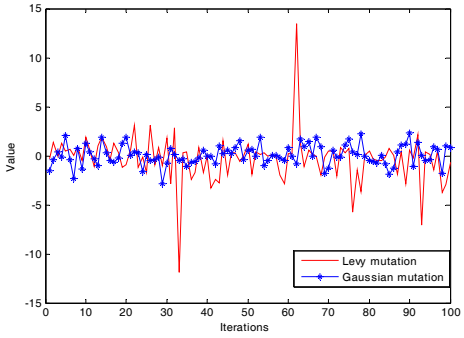


Figure 1. The value produced by the Levy mutation and Gaussian mutation.

The flowchart of the adaptive mutation dynamic search fireworks algorithm (AMdynFWA) is shown in Figure 2.

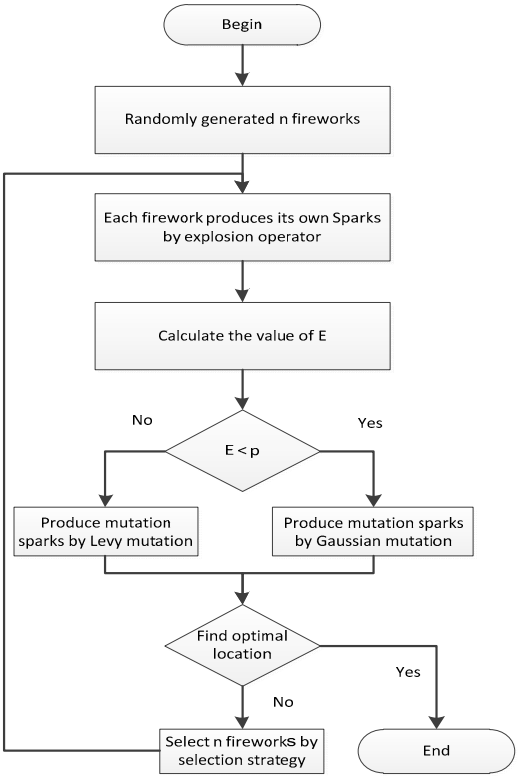


Figure 2. The flowchart of AMdynFWA.

Algorithm 3 demonstrates the complete version of the AMdynFWA.

Algorithm 3. Pseudo-Code of AMdynFWA

Randomly choosing m fireworks
Assess their fitness
Repeat
  Obtain A_i (except for A_{CF})
  Obtain A_{CF} by Equation (6)
  Obtain S_i
  Produce explosion sparks
  Produce mutation sparks
  Assess all sparks' fitness
  Retain the best spark as a firework
  Select other $m-1$ fireworks randomly
  Until termination condition is satisfied
Return the best fitness and a firework location

4. Simulation Results and Analysis

4.1. Simulation Settings

Similar to dynFWA, the number of fireworks in AMdynFWA is set to five, the number of mutation sparks is also set to five, and the maximum number of sparks in each generation is set to 150.

In the experiment, the function of each algorithm is repeated 51 times, and the final results after 300,000 function evaluations are presented. In order to verify the performance of the algorithm proposed in this paper, we use the CEC2013 test set [16], including 28 different types of test functions, which are listed in Table 1. All experimental test function dimensions are set to 30, $d = 30$.

Table 1. CEC2013 test set.

Function Type	Function Number	Function Name	Optimal Value
Unimodal Functions	1	Sphere function	−1400
	2	Rotated high conditioned elliptic function	−1300
	3	Rotated bent cigar function	−1200
	4	Rotated discus function	−1100
	5	Different powers function	−1000
Basic Multimodal Functions	6	Rotated rosenbrock's function	−900
	7	Rotated schaffers F7 function	−800
	8	Rotated Ackley's function	−700
	9	Rotated weierstrass function	−600
	10	Rotated griewank's function	−500
	11	Rastrigin's function	−400
	12	Rotated rastrigin's function	−300
	13	Non-continuous rotated rastrigin's function	−200
	14	Schewefel's function	−100
	15	Rotated schewefel's function	100
	16	Rotated katsuura function	200
	17	Lunacek Bi_Rastrigin function	300
	18	Rotated Lunacek Bi_Rastrigin function	400
	19	Expanded griewank's plus rosenbrock's function	500
	20	Expanded scaffer's F6 function	600
Composition Functions	21	Composition function 1 (N = 5)	700
	22	Composition function 2 (N = 3)	800
	23	Composition function 3 (N = 3)	900
	24	Composition function 4 (N = 3)	1000
	25	Composition function 5 (N = 3)	1100
	26	Composition function 6 (N = 5)	1200
	27	Composition function 7 (N = 5)	1300
	28	Composition function 8 (N = 5)	1400

Finally, we use the Matlab R2014a software on a PC with a 3.2 GHz CPU (Intel Core i5-3470), 4 GB RAM, and Windows 7 (64 bit).

4.2. Simulation Results and Analysis

4.2.1. Study on the Mutation Probability p

In AMdynFWA, the mutation probability p is introduced to control the probability of selecting the Gaussian and Levy mutations. To investigate the effects of the parameter, we compare the performance of AMdynFWA with different values of p. In this experiment, p is set to 0.1, 0.3, 0.5, 0.7, and 0.9, respectively.

Table 2 gives the computational results of AMdynFWA with different values of p, where 'Mean' is the mean best fitness value. The best results among the comparisons are shown in bold. It can be seen that $p = 0.5$ is suitable for unimodal problems $f1 - f5$. For $f6 - f20$, $p = 0.3$ has a better performance than the others. When p is set as 0.1 or 0.9, the algorithm obtains better performance on $f21 - f28$.

The above results demonstrate that the parameter p is problem-oriented. For different problems, different p may be required. In this paper, taking into account the average ranking, $p = 0.3$ is regarded as the relatively suitable value.

Table 2. Mean value and average rankings achieved by AMdynFWA with different p, where the 'mean' indicates the mean best fitness value.

Functions	p = 0.1	p = 0.3	p = 0.5	p = 0.7	p = 0.9
	Mean	**Mean**	**Mean**	**Mean**	**Mean**
f1	**−1400**	**−1400**	**−1400**	**−1400**	**−1400**
f2	$\mathbf{3.76 \times 10^5}$	3.84×10^5	4.56×10^5	3.96×10^5	4.13×10^5
f3	1.01×10^8	8.32×10^7	$\mathbf{5.56 \times 10^7}$	7.16×10^7	6.69×10^7
f4	−1099.9872	−1099.98	**−1099.988**	−1099.9870	−1099.984
f5	−1000	−1000	−1000	−1000	**−1000**
f6	−870.38	**−876.05**	−875.5	−875.29	−874.71
f7	−713.59	−711.45	**−713.69**	−712.66	−702.99
f8	**−679.069**	−679.057	−679.052	−679.063	−679.067
f9	**−578.503**	−577.189	−577.75	−577.436	−576.518
f10	**−499.976**	−499.968	−499.968	−499.972	−499.974
f11	−305.44	−302.436	−307.02	**−311.215**	−309.596
f12	−164.688	**−174.843**	−163.865	−173.722	−154.561
f13	−31.9988	**−36.4318**	−35.7453	−30.6652	−32.3421
f14	2616.647	**2543.716**	2676.641	2586.064	2704.535
f15	**3664.113**	3974.245	3888.197	3946.214	3723.16
f16	200.3942	200.3496	200.3884	**200.3441**	300.3698
f17	437.5601	425.8707	426.4633	**424.32**	428.1304
f18	583.18	577.8134	578.672	576.0805	**573.5208**
f19	506.931	506.5545	506.6363	507.0156	**506.3289**
f20	613.1458	613.154	**613.113**	613.594	613.423
f21	1047.089	1051.01	**1016.475**	1035.483	1049.556
f22	**3871.804**	3928.667	4109.614	4059.632	4032.769
f23	5402.42	5574.529	5524.135	5597.751	**5338.983**
f24	1264.25	1265.845	1265.61	1268.231	**1264.214**
f25	1390.105	**1387.764**	1387.808	1390.035	1391.654
f26	**1408.752**	1412.901	1424.752	1414.98	1412.238
f27	2203.579	2187.724	2192.054	2191.372	**2181.232**
f28	1812.154	1762.647	**1707.262**	1771.612	1830.575
	Average Ranking				
	2.93	2.82	2.86	3.07	2.93

4.2.2. Comparison of AMdynFWA with FWA-Based Algorithms

To assess the performance of AMdynFWA, AMdynFWA is compared with enhanced fireworks algorithm (EFWA), dynamic search fireworks algorithms (dynFWA), and adaptive fireworks algorithm (AFWA), and the EFWA parameters are set in accordance with [2], the AFWA parameters are set in accordance with [3], and the dynFWA parameters are set in accordance with [4].

The probability p used in AMdynFWA is set to 0.3. For each test problem, each algorithm runs 51 times, all experimental test function dimensions are set as 30, and their mean errors and total number of rank 1 are reported in Table 3.

The results from Table 3 indicate that the total number of rank 1 of AMdynFWA (23) is the best of the four algorithms.

Table 3. Mean errors and total number of rank 1 achieved by EFWA, AFWA, dynFWA, and AMdynFWA.

Functions	EFWA	AFWA	dynFWA	AMdynFWA
	Mean Error	Mean Error	Mean Error	Mean Error
f1	7.82×10^{-2}	**0**	**0**	**0**
f2	5.43×10^{5}	8.93×10^{5}	7.87×10^{5}	$\mathbf{3.84 \times 10^{5}}$
f3	1.26×10^{8}	1.26×10^{8}	1.57×10^{8}	$\mathbf{8.32 \times 10^{7}}$
f4	1.09	11.5	12.8	$\mathbf{2.02 \times 10^{-2}}$
f5	7.9×10^{-2}	6.04×10^{-4}	5.42×10^{-4}	$\mathbf{1.86 \times 10^{-4}}$
f6	34.9	29.9	31.5	**23.9**
f7	1.33×10^{2}	9.19×10^{1}	1.03×10^{2}	$\mathbf{8.85 \times 10^{1}}$
f8	2.10×10^{1}	$\mathbf{2.09 \times 10^{1}}$	$\mathbf{2.09 \times 10^{1}}$	$\mathbf{2.09 \times 10^{1}}$
f9	3.19×10^{1}	2.48×10^{1}	2.56×10^{1}	$\mathbf{2.28 \times 10^{1}}$
f10	8.29×10^{-1}	4.73×10^{-2}	4.20×10^{-2}	$\mathbf{3.18 \times 10^{-2}}$
f11	4.22×10^{2}	1.05×10^{2}	1.07×10^{2}	$\mathbf{9.75 \times 10^{1}}$
f12	6.33×10^{2}	1.52×10^{2}	1.56×10^{2}	$\mathbf{1.25 \times 10^{2}}$
f13	4.51×10^{2}	2.36×10^{2}	2.44×10^{2}	$\mathbf{1.63 \times 10^{2}}$
f14	4.16×10^{3}	2.97×10^{3}	2.95×10^{3}	$\mathbf{2.64 \times 10^{3}}$
f15	4.13×10^{3}	3.81×10^{3}	$\mathbf{3.71 \times 10^{3}}$	3.87×10^{3}
f16	5.92×10^{-1}	4.97×10^{-1}	4.77×10^{-1}	$\mathbf{3.4 \times 10^{-1}}$
f17	3.10×10^{2}	1.45×10^{2}	1.48×10^{2}	$\mathbf{1.25 \times 10^{2}}$
f18	$\mathbf{1.75 \times 10^{2}}$	$\mathbf{1.75 \times 10^{2}}$	1.89×10^{2}	1.77×10^{2}
f19	12.3	6.92	6.87	**6.55**
f20	14.6	**13**	**13**	**13**
f21	3.24×10^{2}	3.16×10^{2}	$\mathbf{2.92 \times 10^{2}}$	3.51×10^{2}
f22	5.75×10^{3}	3.45×10^{3}	3.41×10^{3}	$\mathbf{3.12 \times 10^{3}}$
f23	5.74×10^{3}	4.70×10^{3}	$\mathbf{4.55 \times 10^{3}}$	4.67×10^{3}
f24	3.37×10^{2}	2.70×10^{2}	2.72×10^{2}	$\mathbf{2.65 \times 10^{2}}$
f25	3.56×10^{2}	2.99×10^{2}	2.97×10^{2}	$\mathbf{2.87 \times 10^{2}}$
f26	3.21×10^{2}	2.73×10^{2}	2.62×10^{2}	$\mathbf{2.12 \times 10^{2}}$
f27	1.28×10^{3}	9.72×10^{2}	9.92×10^{2}	$\mathbf{8.87 \times 10^{2}}$
f28	4.34×10^{2}	4.37×10^{2}	$\mathbf{3.40 \times 10^{2}}$	3.62×10^{2}
total number of rank 1				
	1	4	**7**	23

Figure 3 shows a comparison of the average run-time cost in the 28 functions for AFWA, EFWA, dynFWA, and AMdynFWA.

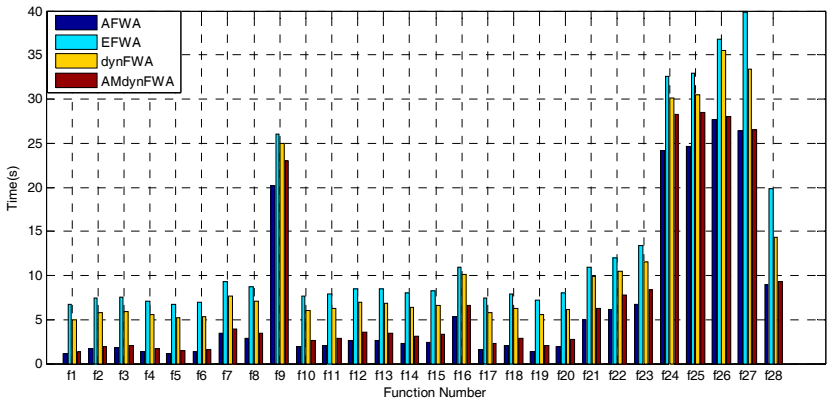


Figure 3. The EFWA, AFWA, dynFWA, and AMdynFWA run-time cost.

The results from Figure 3 indicate that the average run-time cost of EFWA is the most expensive among the four algorithms. The time cost of AFWA is the least, but the run-time cost of AMdynFWA is almost the same compared with AFWA. The run-time cost of AMdynFWA is less than that of dynFWA. Taking into account the results from Table 3, AMdynFWA performs significantly better than the other three algorithms.

To evaluate whether the AMdynFWA results were significantly different from those of the EFWA, AFWA, and dynFWA, the AMdynFWA mean results during the iteration for each test function were compared with those of the EFWA, AFWA, and dynFWA. The *T* test [17], which is safe and robust, was utilized at the 5% level to detect significant differences between these pairwise samples for each test function.

The *ttest2* function in Matlab R2014a was used to run the *T* test, as shown in Table 4. The null hypothesis is that the results of EFWA, AFWA, and dynFWA are derived from distributions of equal mean, and in order to avoid increases of type I errors, we correct the *p*-values using the Holm's method, and order the *p*-values for the three hypotheses being tested from smallest to largest, and we then have three *T* tests. Thus, the *p*-value 0.05 is changed to 0.0167, 0.025, and 0.05, and then the corrected *p*-values were used to compare with the calculated *p*-values, respectively.

Table 4. T test results of AMdynFWA compared with EFWA, AFWA and dynFWA.

Functions	*p*/Significance	EFWA	AFWA	dynFWA
f1	*p*-value	0	NaN	NaN
	significance	+	-	-
f2	*p*-value	1.5080×10^{-32}	5.1525×10^{-50}	2.6725×10^{-49}
	significance	+	+	+
f3	*p*-value	0.8004	0.4302	0.0778
	significance	-	-	-
f4	*p*-value	1.5546×10^{-136}	1.8922×10^{-246}	8.8572×10^{-235}
	significance	+	+	+
f5	*p*-value	0	NaN	NaN
	significance	+	-	-
f6	*p*-value	1.5957×10^{-14}	0.7108	0.0139
	significance	+	-	+
f7	*p*-value	1.8067×10^{-36}	0.5665	0.0084
	significance	+	-	+
f8	*p*-value	0.1562	0.0137	9.2522×10^{-6}
	significance	-	+	+
f9	*p*-value	7.0132×10^{-27}	0.0278	6.6090×10^{-8}
	significance	+	+	+
f10	*p*-value	2.7171×10^{-134}	7.3507×10^{-6}	0.0364
	significance	+	+	+
f11	*p*-value	2.2083×10^{-100}	3.0290×10^{-10}	0.0437
	significance	+	+	+
f12	*p*-value	1.7319×10^{-101}	1.3158×10^{-11}	1.8212×10^{-7}
	significance	+	+	+
f13	*p*-value	2.3914×10^{-89}	4.1645×10^{-36}	8.6284×10^{-37}
	significance	+	+	+
f14	*p*-value	0.0424	0.0117	4.4964×10^{-5}
	significance	+	+	+
f15	*p*-value	1.1749×10^{-6}	0.9976	0.6064
	significance	+	-	-
f16	*p*-value	2.2725×10^{-17}	8.9230×10^{-12}	2.3427×10^{-13}
	significance	+	+	+

Table 4. *Cont.*

Functions	*p*/Significance	EFWA	AFWA	dynFWA
f17	*p*-value	1.5713×10^{-81}	7.3257×10^{-10}	1.0099×10^{-6}
	significance	+	+	+
f18	*p*-value	0.8510	0.2430	0.1204
	significance	-	-	-
f19	*p*-value	3.6331×10^{-25}	5.3309×10^{-6}	0.0086
	significance	+	+	+
f20	*p*-value	3.5246×10^{-14}	0.2830	0.4615
	significance	+	-	-
f21	*p*-value	2.2455×10^{-6}	0.0120	0.0028
	significance	+	+	+
f22	*p*-value	3.2719×10^{-46}	0.0634	0.0344
	significance	+	-	-
f23	*p*-value	2.1191×10^{-33}	0.1225	0.4819
	significance	+	-	-
f24	*p*-value	8.9612×10^{-69}	9.0342×10^{-5}	6.0855×10^{-4}
	significance	+	+	+
f25	*p*-value	1.2812×10^{-59}	1.0745×10^{-6}	1.6123×10^{-8}
	significance	+	+	+
f26	*p*-value	4.6864×10^{-39}	2.5440×10^{-16}	1.1739×10^{-11}
	significance	+	+	+
f27	*p*-value	2.3540×10^{-46}	4.8488×10^{-6}	2.1456×10^{-7}
	significance	+	+	+
f28	*p*-value	6.4307×10^{-92}	0.4414	0.0831
	significance	+	-	-

Where the *p*-value is the result of the *T* test. The '+' indicates the rejection of the null hypothesis at the 5% significance level, and the '-' indicates the acceptance of the null hypothesis at the 5% significance level.

Table 5 indicates that AMdynFWA showed a large improvement over EFWA in most functions. However, in Unimodal Functions, AMdynFWA is not significant when compared with AFWA and dynFWA. In Basic Multimodal Functions and Composition Functions, the AMdynFWA also showed a large improvement over AFWA and dynFWA.

Table 5. Total number of significance of AMdynFWA compared with EFWA, AFWA and dynFWA.

Functions Type	EFWA	AFWA	dynFWA
Unimodal Functions (f1 − f5)	4	2	2
Basic Multimodal Functions (f6 − f20)	13	10	12
Composition Functions (f21 − f28)	8	5	5
Total number of significance in EFWA, AFWA and dynFWA			
	25	17	19

Figure 4 shows the mean fitness searching curves of the 28 functions for EFWA, AFWA, dynFWA, and AMdynFWA.

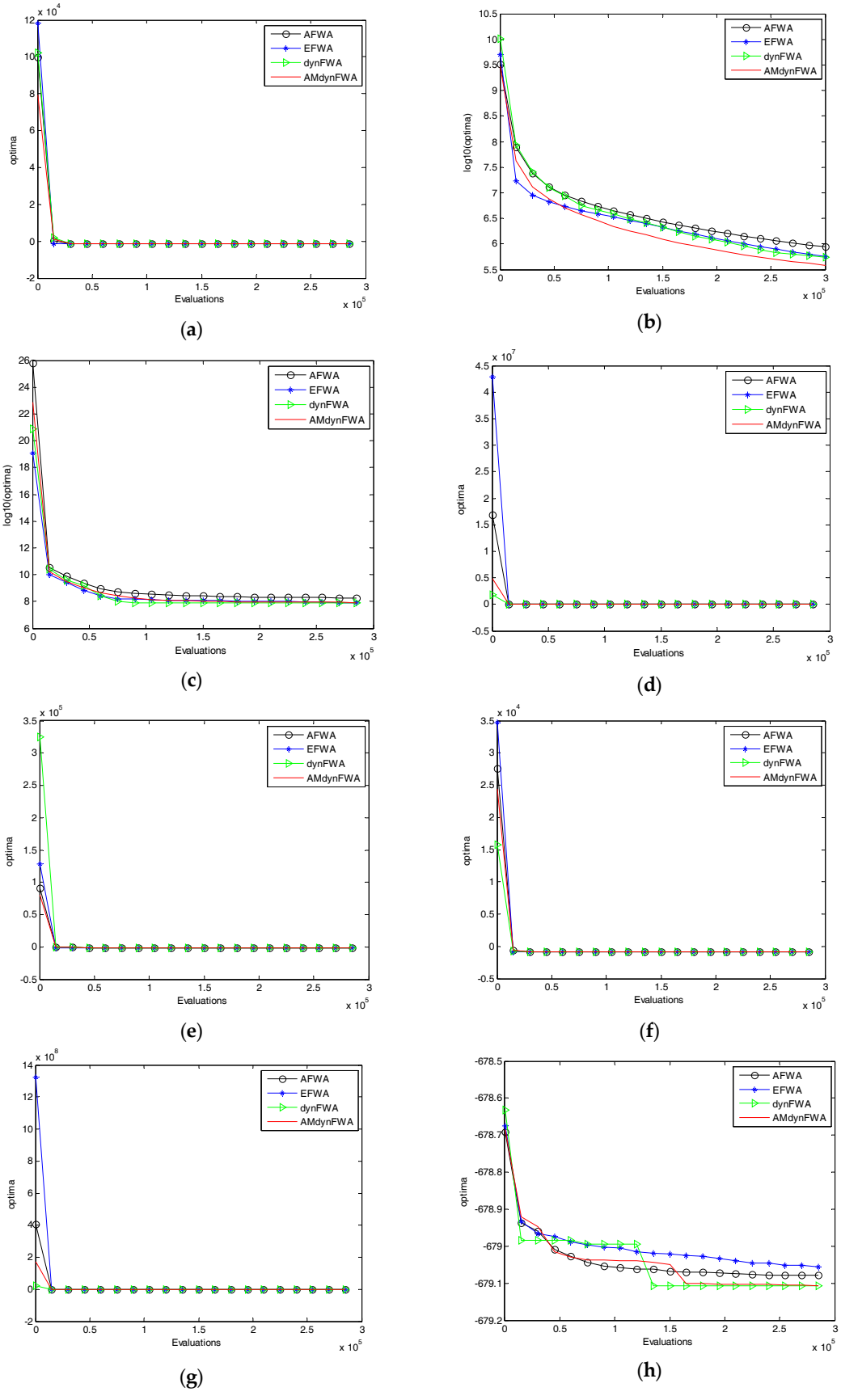


Figure 4. *Cont.*

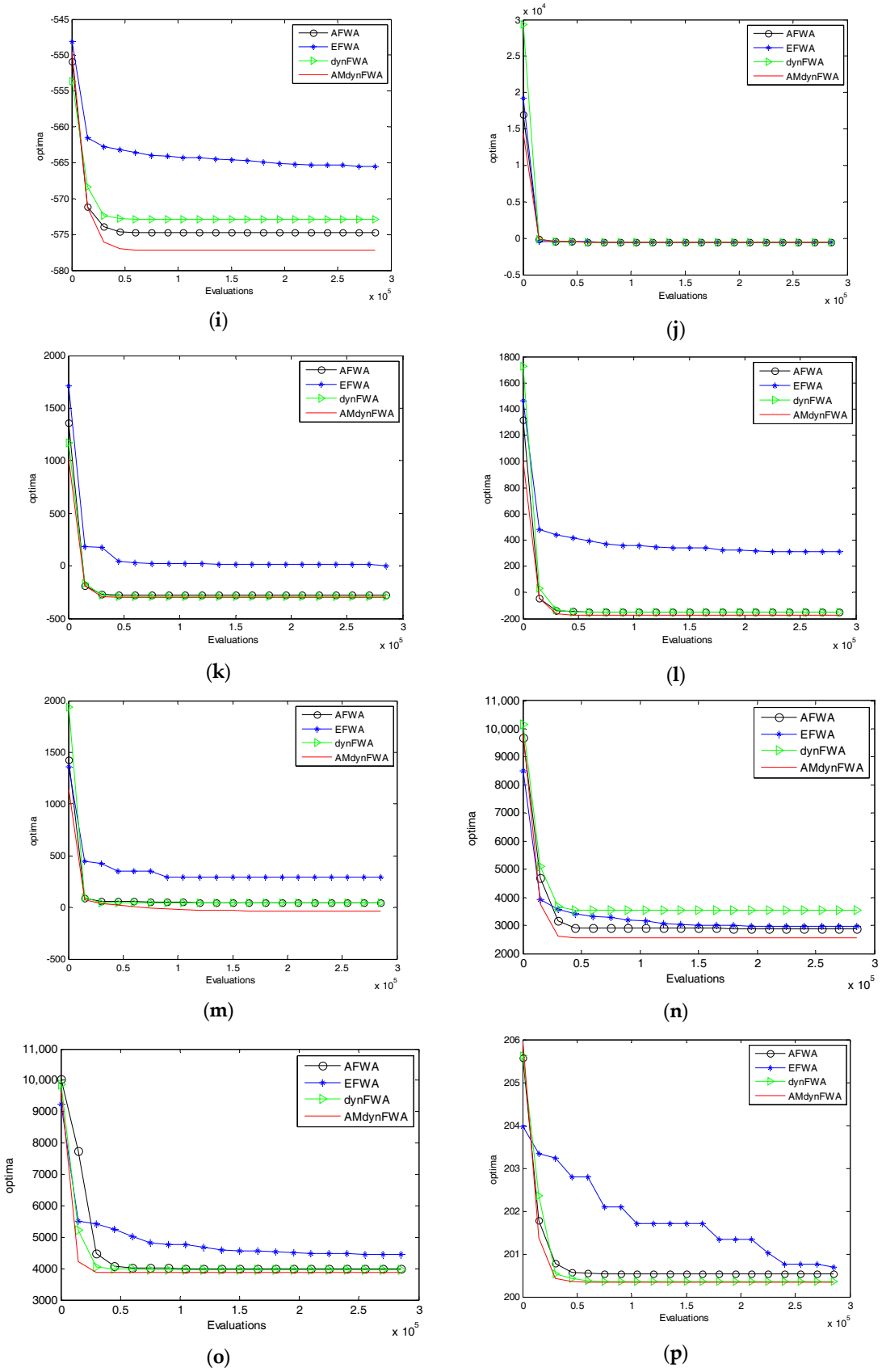
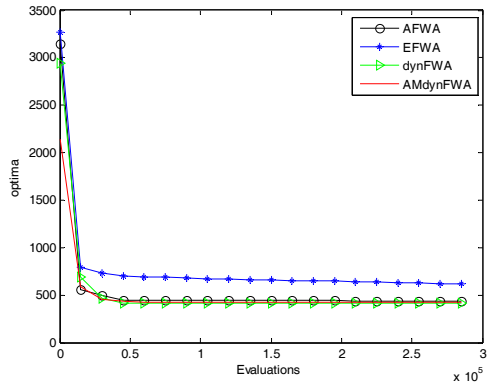
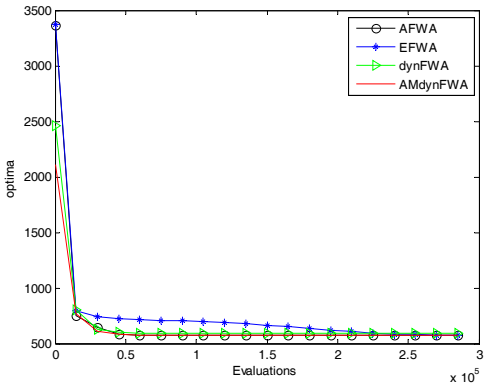


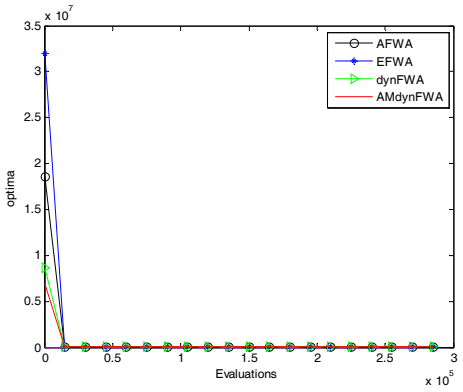
Figure 4. *Cont.*



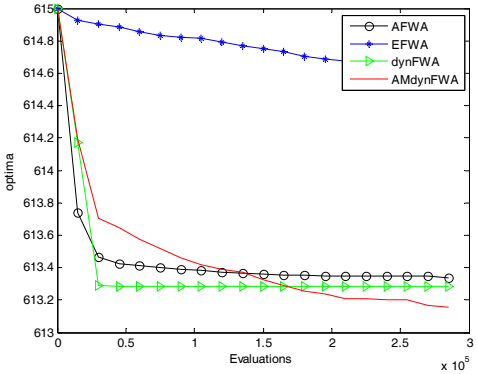
(**q**)



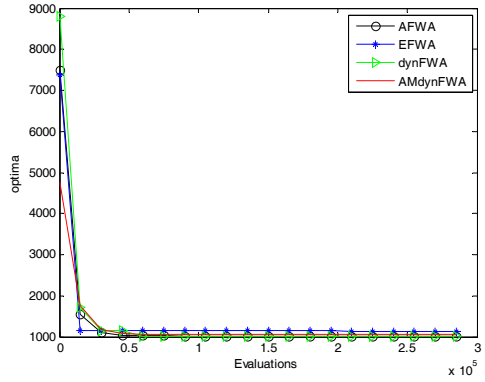
(**r**)



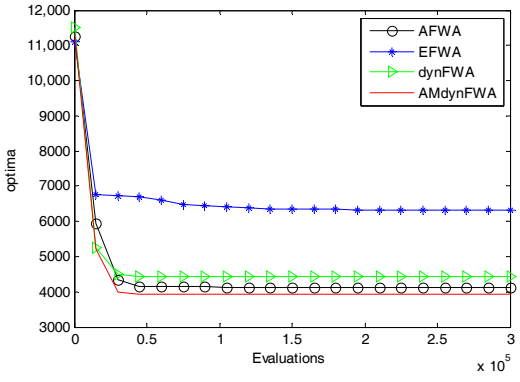
(**s**)



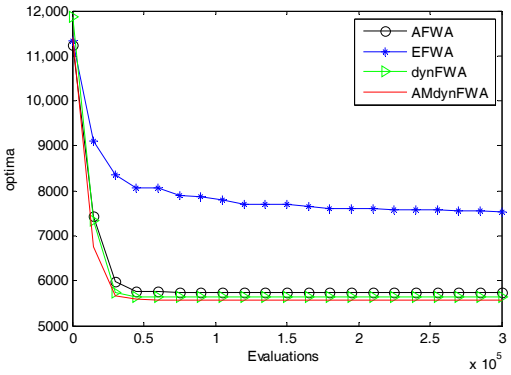
(**t**)



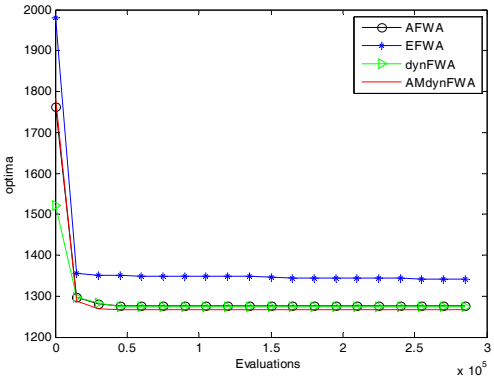
(**u**)



(**v**)



(**w**)



(**x**)

Figure 4. *Cont.*

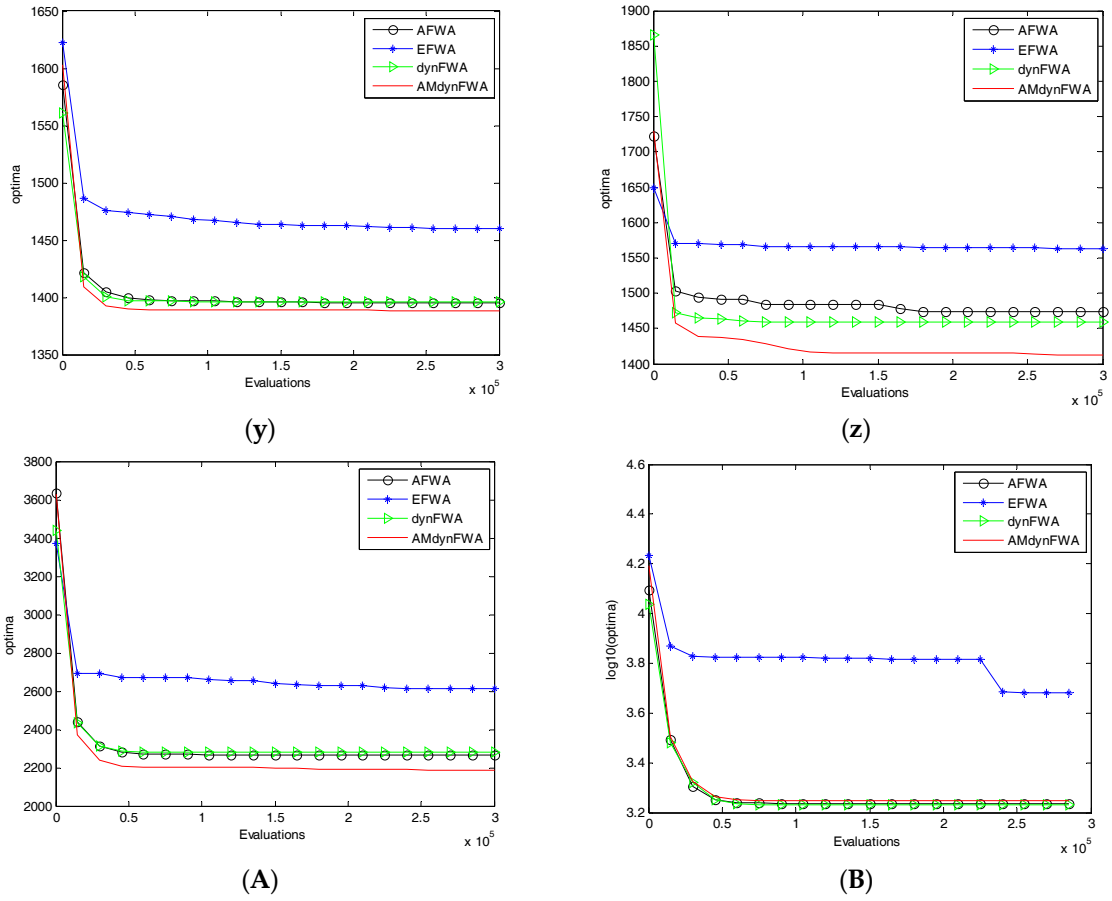


Figure 4. The EFWA, AFWA, dynFWA, and AMdynFWA searching curves. (**a**) f1 function; (**b**) f2 function; (**c**) f3 function; (**d**) f4 function; (**e**) f5 function; (**f**) f6 function; (**g**) f7 function; (**h**) f8 function; (**i**) f9 function; (**j**) f10 function; (**k**) f11 function; (**l**) f12 function; (**m**) f13 function; (**n**) f14 function; (**o**) f15 function; (**p**) f16 function; (**q**) f17 function; (**r**) f18 function; (**s**) f19 function; (**t**) f20 function; (**u**) f21 function; (**v**) f22 function; (**w**) f23 function; (**x**) f24 function; (**y**) f25 function; (**z**) f26 function; (**A**) f27 function; (**B**) f28 function.

4.2.3. Comparison of AMdynFWA with Other Swarm Intelligence Algorithms

In order to measure the relative performance of the AMdynFWA, a comparison among the AMdynFWA and the other swarm intelligence algorithms is conducted on the CEC2013 single objective benchmark suite. The algorithms compared here are described as follows.

(1) Artificial bee colony (ABC) [18]: A powerful swarm intelligence algorithm.
(2) Standard particle swarm optimization (SPSO2011) [19]: The most recent standard version of the famous swarm intelligence algorithm PSO.
(3) Differential evolution (DE) [20]: One of the best evolutionary algorithms for optimization.
(4) Covariance matrix adaptation evolution strategy (CMA-ES) [21]: A developed evolutionary algorithm.

The above four algorithms use the default settings. The comparison results of ABC, DE, CMS-ES, SPSO2011, and AMdynFWA are presented in Table 6, where the 'Mean error' is the mean error of the best fitness value. The best results among the comparisons are shown in bold. ABC beats the other algorithms on 12 functions (some differences are not significant), which is the most, but performs poorly on the other functions. CMA-ES performs extremely well on unimodal functions, but suffers from premature convergence on some complex functions. From Table 7, the AMdynFWA ranked the top three (22/28), which is better than the other algorithms (except the DE), and in terms of average

ranking, the AMdynFWA performs the best among these five algorithms on this benchmark suite due to its stability. DE and ABC take the second place and the third place, respectively. The performances of CMS-ES and the SPSO2011 are comparable.

Table 6. Mean errors and ranking achieved by ABC, DE, CMS-ES, SPSO2011, and AMdynFWA.

Functions	Mean Error/Rank	ABC	DE	CMS-ES	SPSO2011	AMdynFWA
f1	Mean error	**0**	1.89×10^{-3}	**0**	**0**	**0**
	Rank	1	2	1	1	1
f2	Mean error	6.20×10^{6}	5.52×10^{4}	**0**	3.38×10^{5}	3.84×10^{5}
	Rank	5	2	1	3	4
f3	Mean error	5.74×10^{8}	2.16×10^{6}	$\mathbf{1.41 \times 10^{1}}$	2.88×10^{8}	8.32×10^{7}
	Rank	5	2	1	4	3
f4	Mean error	8.75×10^{4}	1.32×10^{-1}	**0**	3.86×10^{4}	2.02×10^{-2}
	Rank	5	3	1	4	2
f5	Mean error	**0**	2.48×10^{-3}	**0**	5.42×10^{-4}	1.86×10^{-4}
	Rank	1	4	1	3	2
f6	Mean error	1.46×10^{1}	7.82	$\mathbf{7.82 \times 10^{-2}}$	3.79×10^{1}	2.39×10^{1}
	Rank	3	2	1	5	4
f7	Mean error	1.25×10^{2}	4.89×10^{1}	$\mathbf{1.91 \times 10^{1}}$	8.79×10^{1}	8.85×10^{1}
	Rank	5	2	1	3	4
f8	Mean error	$\mathbf{2.09 \times 10^{1}}$	$\mathbf{2.09 \times 10^{1}}$	2.14×10^{1}	$\mathbf{2.09 \times 10^{1}}$	$\mathbf{2.09 \times 10^{1}}$
	Rank	1	1	2	1	1
f9	Mean error	3.01×10^{1}	$\mathbf{1.59 \times 10^{1}}$	4.81×10^{1}	2.88×10^{1}	2.28×10^{1}
	Rank	4	1	5	3	2
f10	Mean error	2.27×10^{-1}	3.42×10^{-2}	$\mathbf{1.78 \times 10^{-2}}$	3.40×10^{-1}	3.18×10^{-2}
	Rank	4	3	1	5	2
f11	Mean error	**0**	7.88×10^{1}	4.00×10^{2}	1.05×10^{2}	9.75×10^{1}
	Rank	1	2	5	4	3
f12	Mean error	3.19×10^{2}	$\mathbf{8.14 \times 10^{1}}$	9.42×10^{2}	1.04×10^{2}	1.25×10^{2}
	Rank	4	1	5	2	3
f13	Mean error	3.29×10^{2}	$\mathbf{1.61 \times 10^{2}}$	1.08×10^{3}	1.94×10^{2}	1.63×10^{2}
	Rank	4	1	5	3	2
f14	Mean error	$\mathbf{3.58 \times 10^{-1}}$	2.38×10^{3}	4.94×10^{3}	3.99×10^{3}	2.64×10^{3}
	Rank	1	2	5	4	3
f15	Mean error	3.88×10^{3}	5.19×10^{3}	5.02×10^{3}	$\mathbf{3.81 \times 10^{3}}$	3.87×10^{3}
	Rank	3	5	4	1	2
f16	Mean error	1.07	1.97	$\mathbf{5.42 \times 10^{-2}}$	1.31	3.4×10^{-1}
	Rank	3	5	1	4	2
f17	Mean error	$\mathbf{3.04 \times 10^{1}}$	9.29×10^{1}	7.44×10^{2}	1.16×10^{2}	1.25×10^{2}
	Rank	1	2	5	3	4
f18	Mean error	3.04×10^{2}	2.34×10^{2}	5.17×10^{2}	$\mathbf{1.21 \times 10^{2}}$	1.77×10^{2}
	Rank	4	3	5	1	2
f19	Mean error	$\mathbf{2.62 \times 10^{-1}}$	4.51	3.54	9.51	6.55
	Rank	1	3	2	5	4
f20	Mean error	1.44×10^{1}	1.43×10^{1}	1.49×10^{1}	1.35×10^{1}	$\mathbf{1.30 \times 10^{1}}$
	Rank	4	3	5	2	1
f21	Mean error	$\mathbf{1.65 \times 10^{2}}$	3.20×10^{2}	3.44×10^{2}	3.09×10^{2}	3.51×10^{2}
	Rank	1	3	4	2	5
f22	Mean error	$\mathbf{2.41 \times 10^{1}}$	1.72×10^{3}	7.97×10^{3}	4.30×10^{3}	3.12×10^{3}
	Rank	1	2	5	4	3
f23	Mean error	4.95×10^{3}	5.28×10^{3}	6.95×10^{3}	4.83×10^{3}	$\mathbf{4.67 \times 10^{3}}$
	Rank	3	4	5	2	1
f24	Mean error	2.90×10^{2}	$\mathbf{2.47 \times 10^{2}}$	6.62×10^{2}	2.67×10^{2}	2.65×10^{2}
	Rank	4	1	5	3	2
f25	Mean error	3.06×10^{2}	2.89×10^{2}	4.41×10^{2}	2.99×10^{2}	$\mathbf{2.87 \times 10^{2}}$
	Rank	4	2	5	3	1

Table 6. *Cont.*

Functions	Mean Error/Rank	ABC	DE	CMS-ES	SPSO2011	AMdynFWA
f26	Mean error	$\mathbf{2.01 \times 10^2}$	2.52×10^2	3.29×10^2	2.86×10^2	2.12×10^2
	Rank	1	3	5	4	2
f27	Mean error	$\mathbf{4.16 \times 10^2}$	7.64×10^2	5.39×10^2	1.00×10^3	8.87×10^2
	Rank	1	4	2	5	3
f28	Mean error	$\mathbf{2.58 \times 10^2}$	4.02×10^2	4.78×10^3	4.01×10^2	3.62×10^2
	Rank	1	4	5	3	2

Table 7. Statistics of rank (SR) and average rankings (AR).

SR/AR	ABC	DE	CMS-ES	SPSO2011	AMdynFWA
Total number of rank 1	12	5	9	4	5
Total number of rank 2	0	10	3	4	11
Total number of rank 3	4	7	0	9	6
Total number of rank 4	8	4	2	7	5
Total number of rank 5	4	2	14	4	1
Total number of rank	76	72	93	87	70
Average ranking	2.71	2.57	3.32	3.11	2.5

5. Conclusions

AMdynFWA was developed by applying two mutation methods to dynFWA. It selects the Gaussian mutation or Levy mutation according to the mutation probability. We apply the CEC2013 standard functions to examine and compare the proposed algorithm AMdynFWA with ABC, DE, SPSO2011, CMS-ES, AFWA, EFWA, and dynFWA. The results clearly indicate that AMdynFWA can perform significantly better than the other seven algorithms in terms of solution accuracy and stability. Overall, the research demonstrates that AMdynFWA performed the best for solution accuracies.

The study on the mutation probability p demonstrates that there is no constant p for all the test problems, while $p = 0.3$ is regarded as the relatively suitable value for the current test suite. A dynamic p may be a good choice. This will be investigated in future work.

Acknowledgments: The authors are thankful to the anonymous reviewers for their valuable comments to improve the technical content and the presentation of the paper. This paper is supported by the Liaoning Provincial Department of Education Science Foundation (Grant No. L2013064), AVIC Technology Innovation Fund (basic research) (Grant No. 2013S60109R), and the Research Project of Education Department of Liaoning Province (Grant No. L201630).

Author Contributions: Xi-Guang Li participated in the draft writing. Shou-Fei Han participated in the concept, design, and performed the experiments and commented on the manuscript. Liang Zhao, Chang-Qing Gong, and Xiao-Jing Liu participated in the data collection, and analyzed the data.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Tan, Y.; Zhu, Y. Fireworks Algorithm for Optimization. In *Advances in Swarm Intelligence, Proceedings of the 2010 International Conference in Swarm Intelligence, Beijing, China, 12–15 June 2010*; Springer: Berlin/Heidelberg, Germany, 2010.
2. Zheng, S.; Janecek, A.; Tan, Y. Enhanced fireworks algorithm. In Proceedings of the 2013 IEEE Congress on Evolutionary Computation, Cancun, Mexico, 20–23 June 2013; pp. 2069–2077.
3. Zheng, S.; Li, J.; Tan, Y. Adaptive fireworks algorithm. In Proceedings of the 2014 IEEE Congress on Evolutionary Computation, Beijing, China, 6–11 July 2014; pp. 3214–3221.
4. Zheng, S.; Tan, Y. Dynamic search in fireworks algorithm. In Proceedings of the 2014 IEEE Congress on Evolutionary Computation, Beijing, China, 6–11 July 2014; pp. 3222–3229.

5. Li, X.-G.; Han, S.-F.; Gong, C.-Q. Analysis and Improvement of Fireworks Algorithm. *Algorithms* **2017**, *10*, 26. [CrossRef]
6. Tan, Y. *Fireworks Algorithm Introduction*, 1st ed.; Science Press: Beijing, China, 2015; pp. 13–136. (In Chinese)
7. Gao, H.Y.; Diao, M. Cultural firework algorithm and its application for digital filters design. *Int. J. Model. Identif. Control* **2011**, *4*, 324–331. [CrossRef]
8. Andreas, J.; Tan, Y. Using population based algorithms for initializing nonnegative matrix factorization. In *Advances in Swarm Intelligence, Proceedings of the 2010 International Conference in Swarm Intelligence, Chongqing, China, 12–15 June 2011*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 307–316.
9. Wen, R.; Mi, G.Y.; Tan, Y. Parameter optimization of local-concentration model for spam detection by using fireworks algorithm. In Proceedings of the 4th International Conference on Swarm Intelligence, Harbin, China, 12–15 June 2013; pp. 439–450.
10. Zheng, S.; Tan, Y. A unified distance measure scheme for orientation coding in identification. In Proceedings of the 2013 IEEE Congress on Information Science and Technology, Yangzhou, China, 23–25 March 2013; pp. 979–985.
11. Pholdee, N.; Bureerat, S. Comparative performance of meta-heuristic algorithms for mass minimisation of trusses with dynamic constraints. *Adv. Eng. Softw.* **2014**, *75*, 1–13. [CrossRef]
12. Yang, X.; Tan, Y. Sample index based encoding for clustering using evolutionary computation. In *Advances in Swarm Intelligence, Proceedings of the 2014 International Conference on Swarm Intelligence, Hefei, China, 17–20 October 2014*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 489–498.
13. Mohamed Imran, A.; Kowsalya, M. A new power system reconfiguration scheme for power loss minimization and voltage profile enhancement using fireworks algorithm. *Int. J. Electr. Power Energy Syst.* **2014**, *62*, 312–322. [CrossRef]
14. Zhou, F.J.; Wang, X.J.; Zhang, M. Evolutionary Programming Using Mutations Based on the t Probability Distribution. *Acta Electron. Sin.* **2008**, *36*, 121–123.
15. Fei, T.; Zhang, L.Y.; Chen, L. Improved Artificial Fish Swarm Algorithm Mixing Levy Mutation and Chaotic Mutation. *Comput. Eng.* **2016**, *42*, 146–158.
16. Liang, J.; Qu, B.; Suganthan, P.; Hernandez-Diaz, A.G. *Problem Definitions and Evaluation Criteria for the CEC 2013 Special Session on Real-Parameter Optimization*; Technical Report 201212; Zhengzhou University: Zhengzhou, China, January 2013.
17. Teng, S.Z.; Feng, J.H. *Mathematical Statistics*, 4th ed.; Dalian University of Technology Press: Dalian, China, 2005; pp. 34–35. (In Chinese)
18. Karaboga, D.; Basturk, B. A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm. *J. Glob. Optim.* **2007**, *39*, 459–471. [CrossRef]
19. Zambrano-Bigiarini, M.; Clerc, M.; Rojas, R. Standard particle swarm optimization 2011 at CEC2013: A baseline for future PSO improvements. In Proceedings of the 2013 IEEE Congress on Evolutionary Computation, Cancun, Mexico, 20–23 June 2013; pp. 2337–2344.
20. Storn, R.; Price, K. Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces. *J. Glob. Optim.* **1997**, *11*, 341–359. [CrossRef]
21. Hansen, N.; Ostermeier, A. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In Proceedings of the 1996 IEEE International Conference on Evolutionary Computation, Nagoya, Japan, 20–22 May 1996; pp. 312–317.