Which Mapping Rule in the Fireworks Algorithm is Better for Large Scale Optimization

Xuemei Ye†, Junzhi Li*, Bo Xu† and Ying Tan*
†Institute of Automation, Chinese Academy of Sciences
*Key Laboratory of Machine Perception (Ministry of Education), Department of Machine Intelligence,
School of Electronics Engineering and Computer Science, Peking University

***Abstract*—Fireworks algorithm(FWA), which is proposed for global optimization of complex function, becomes a hot spot in optimization field recently, caused by its competitive performance. Boundary handling for FWA, which maps the out-of-bound sparks into feasible space, is critical for its convergence efficiency. However, random mapping rule, which is widely used for boundary handling, always caused computing resource waste, especially for high-dimensional optimization. In this paper, we propose three novel mapping rules to speed up large scale optimization of FWA. Meanwhile, to evaluate the effectiveness of the new rules, we compare them by representative nine benchmark functions on different dimensionality scale. Experimental results indicate that the mirror rule which we proposed, achieve superior performance for most optimization functions.**

I. Introduction

The fireworks algorithm (FWA) [1] is a swarm intelligence algorithm inspired by the phenomenon of the fireworks explosion in the night sky. It has attracted much research interest since it was proposed in 2010.

The FWA is composed of two essential operators. In the explosion operator, numerous explosion sparks are generated around the fireworks within certain explosion amplitudes. In the selection operator, the fireworks of a new generation are selected from these sparks. There is also an optional mutation operator, in which the locations of the fireworks are mutated to generate mutation operators.

Besides the three operators, there is also a mapping rule in the fireworks algorithm. If the optimization problem is constrained, the mapping rule maps the out-of-bound sparks back into the feasible space. The mapping rule is very important when the FWA is applied in real-world problems.

The FWA has proven serviceable in many real-world applications [2]–[7]. The FWA with a random mutation operator has some global convergence properties [8], while the explosive search manner is typically locally convergent [9]. Variants of the FWA has also been successfully implemented on parallel platforms such as CUDA [10], [11] and MapReduce [12].

With the era of big data coming, more and more large scale (high dimensional) optimization problems spring up in both science and industry. Large scale optimization has brought great challenges to meta-heuristic algorithms because it is featured by many special difficulties, such as the curse of dimensionality. Therefore, large scale optimization has attracted more and more interest in this field. However, most of the previous research on the FWA is based on low or middle dimensional objective functions [13]–[18].

In this paper, we investigate the performances of different mapping rules in the FWA when facing large scale optimization problems. A previous empirical study [19] has implied that mapping to the boundary and mapping to a limited stochastic region are the best mapping rules on middle scale optimization problems. We extend that work for the following reasons.

1) That work was based on the conventional FWA [1], which is known for several drawbacks [13]. One of them is the performance of the conventional FWA may suffer dramatically when the optimal point of the objective function is shifted from the origin. These drawbacks may interfere the discussion on mapping rules.
2) The discussion of that work is based on the experimental results on middle dimensional test functions (up to 200 dimensions). While in this paper, we test these mapping rules on high dimensional test functions (up to 1000 dimensions). As we will see, the properties of high dimensional test functions are quite special.
3) Mapping to a limited stochastic region requires an extra preset parameter to control the limited region, which is not preferred because in real-world applications the actual search ranges vary greatly. In this paper, we investigate a natural and intuitive mirror mapping rule instead.

For the convenience of discussion, the following minimization problem is considered in this paper:

$$\min_{\mathbf{x}\in[\mathbf{lb},\mathbf{ub}]} f(\mathbf{x}), \tag{1}$$

where $\mathbf{x}$ is a vector within the Euclidean space, $\mathbf{ub}$ and $\mathbf{lb}$ represent the upper bound and the lower bound vectors of the search space respectively.

The rest of this paper is organized as follows. In Section II we provide a brief introduction of bare bones fireworks algorithm. In Section III we introduce five kinds of mapping rules, experiments results are present in Section IV. The analysis and discussion of experiments results are presented in Section V. Finally Section VI concludes this paper.

II. Bare Bone Fireworks Algorithm

In order to avoid the interference of other operators, the simplest bare bones fireworks algorithm (BBFWA) [9] is used

as a baseline throughout this paper. In the BBFWA, only one firework is adopted and only the essential explosion and selection operators are used. The pseudo code of the BBFWA is shown in Algo. 1.

Algorithm 1 Bare Bones Fireworks Algorithm

1: sample $\mathbf{x} \sim U(\mathbf{lb}, \mathbf{ub})$
2: evaluate $f(\mathbf{x})$
3: $\mathbf{A} \leftarrow \mathbf{ub} - \mathbf{lb}$
4: **repeat**
5: **for** $i = 1$ **to** n **do**
6: sample $\mathbf{s}_i \sim U(\mathbf{x} - \mathbf{A}, \mathbf{x} + \mathbf{A})$
7: evaluate $f(\mathbf{s}_i)$
8: **end for**
9: **if** $\min(f(\mathbf{s}_i)) < f(\mathbf{x})$ **then**
10: $\mathbf{x} \leftarrow \arg\min f(\mathbf{s}_i)$
11: $\mathbf{A} \leftarrow C_a\mathbf{A}$
12: **else**
13: $\mathbf{A} \leftarrow C_r\mathbf{A}$
14: **end if**
15: **until** termination criterion is met.
16: **return** $\mathbf{x}$.

$\mathbf{lb}$ and $\mathbf{ub}$ are the lower and upper boundaries of the search space. $\mathbf{x}$ is the location of the firework, n is the number of generated sparks, $\mathbf{s}_i$ are the locations of explosion sparks and $\mathbf{A}$ is the explosion amplitude.

In each generation, n sparks are generated uniformly within a hyperrectangle bounded by $\mathbf{x} - \mathbf{A}$ and $\mathbf{x} + \mathbf{A}$. After that, if the best spark is a better solution than the firework, it will take the place of the firework and the explosion amplitude will be multiplied by an amplification coefficient $C_a > 1$. Otherwise, the explosion amplitude will be multiplied by a reduction coefficient $C_r < 1$ and the current firework will be kept. Note that in step 7, if a spark is located outside the boundaries, it can be replaced by a new spark in the feasible space according to a certain mapping rule.

The way the explosion amplitude is controlled in the BBFWA (lines 11 and 13 in Alg. 1) is called dynamic explosion amplitude. The core idea of this dynamic explosion amplitude is described as follows: if in one generation no better solution is found, that means the explosion amplitude is too long (aggressive) and thus needs to be reduced to increase the probability of finding a better solution, and otherwise it may be too short (conservative) to make the largest progress and thus needs to be amplified. With the dynamic control, the algorithm can keep the amplitude appropriate for the search. That is, the dynamic explosion amplitude is long in the early phases to perform exploration, and is short in late phases to perform exploitation.

III. Mapping Rules

In this section, we present the mapping rules investigated in this paper. Figure 1 presents the distinction of five mapping rules, the red and green points represent the firework and explosion sparks respectively, the frame represents the search boundaries. From the figure, we can see the distinction of five kinds of mapping rules. In the Section V we will analyze these mapping rules in detail.

A. Deletion

The simplest way to deal with out-of-bound sparks is to delete them. If in a certain generation, no sparks is located in the feasible search space, then this generation will be passed. This is not likely a good mapping rule because computational resources are wasted.

B. Modularization

The modular mapping rule was introduced in the conventional fireworks algorithm [1]. It has been later considered as one of the several reasons why the conventional fireworks algorithm performs abnormally well when the optimal point of the objective function is located near the origin [13], because the result of the modular arithmetic here is typically small (close to zero).

Formally, in the modular mapping rule, for every dimension $k = 1, 2, ..., d$,

$$x^k = lb^k + |x^k| \mod (ub^k - lb^k) \text{ if } x^k < lb^k \text{ or } x^k > ub^k. \tag{2}$$

C. Randomization

The random mapping rule was firstly introduced in the enhanced fireworks algorithm [13] to fix the drawback of the modular mapping rule. If a dimension of a spark is located outside the boundaries, it will be replaced with an uniformly randomly chosen number in the feasible space.

Some experiments on low dimensional or middle dimensional test functions indicate that the performance of the random mapping rule will not suffer dramatically when the origin is shifted from the origin [13].

Formally, in the random mapping rule, for every dimension $k = 1, 2, ..., d$,

$$x^k \sim U(lb^k, ub^k) \text{ if } x^k < lb^k \text{ or } x^k > ub^k. \tag{3}$$

D. Mirror

In the mirror mapping rule, if a dimension of a spark is located outside the boundaries, it will be replaced with one which is symmetric to it about the boundary.

Formally, in the mirror mapping rule, for every dimension $k = 1, 2, ..., d$,

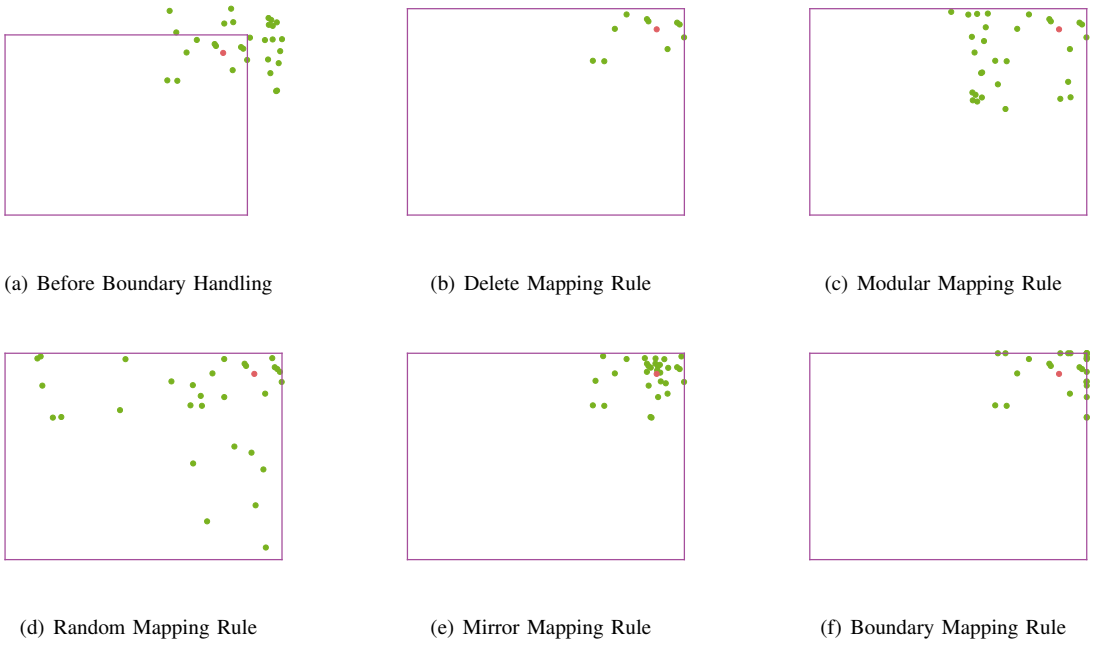
$$x^k = \begin{cases} lb^k + (lb^k - x^k) & \text{if } x^k < lb^k \\ ub^k - (x^k - ub^k) & \text{if } x^k > ub^k \end{cases} \tag{4}$$

E. Boundary

In the boundary mapping rule, if a dimension of a spark is located outside the boundaries, it will be replaced with one located on the boundary.

Formally, in the boundary mapping rule, for every dimension $k = 1, 2, ..., d$,

$$x^k = \begin{cases} lb^k & \text{if } x^k < lb^k \\ ub^k & \text{if } x^k > ub^k \end{cases} \tag{5}$$



(a) Before Boundary Handling (b) Delete Mapping Rule (c) Modular Mapping Rule

(d) Random Mapping Rule (e) Mirror Mapping Rule (f) Boundary Mapping Rule

Fig. 1: Spark Position in Five Kinds of Mapping Rules

IV. Experiments

Which mapping rule presented previously is better for large scale optimization? For investigating this question, several groups of experiments are designed. Experimental setup is first introduced, followed by the experimental results.

A. Experimental Setup

To compare five kinds of mapping rules present previously, we designed 9 groups experiments on the selected 9 benchmark functions, the dimensions of each experiment is 100, 400, 700, 1000 respectively. The mathematical form of the functions are given in Table I.

The mapping rules only affect the convergence speed and do not affect the convergence result, as the convergence result is determined by the algorithm itself. Our goal is to investigate the performance of mapping rules, so the experiment iteration number is set to 1000, which should be enough to show the convergence speed of different mapping rules. Beside that, computing resources are limited in real-world, so this can be consider as a simulation of real-world problems. Faster convergence rate will spend less resources. When handle big data problem, convergence rate sometimes play a decisive role, that's why we do the research to find whether have a better mapping rule.

In order to make the experimental results more credible, in the experiment, each dimension of each function is repeated 50 times. In real-world problems, we do not know where the optimal point is. In order to simulate real-world problems, we shift the optimal point of the objective function randomly each time. The parameters of each run time: n = 30, Cr = 0.9, Ca = 1.2, Evaluation times: 30000. The environment of experiments is python 2.7; Win 10; Intel(R) Core(TM) i7-7700 CPU; 3.60GHz; 16GB RAM.

B. Experimental Results

36 groups of experiments were conducted. In order to better see convergence speed of different mapping rules, we take the logarithm of the function value, which are shown in figures 2 to 10. Each line in the figure is the mean of 50 runs, the range of each line presents the confidence interval, which is to make sure that the experimental results reliable and obtained by adding and subtracting the standard deviation from the mean. As mentioned in the mapping rule section, Figure 1 presents the distinction of five mapping rules.

In order to make the conclusion more believable, we performed 36 significant tests on four dimensions and nine functions. The mean of 50 runs consists of the 1000 iterations of the data, and it does not meet the normal distribution and cannot be represented by a few parameters. Therefore, only nonparametric tests can be used. We need to test the difference of the five rules, the sample size is 1000, so we choose the Kruskal-Wallis test by ranks instead of Friedmans test. The p-value of Kruskal-Wallis test by ranks given in Table II. There are five mapping rules, so $df = 4$. All 36 p-values were lower than 0.005, indicating a very significant difference in the five mapping rules.

V. Discussion

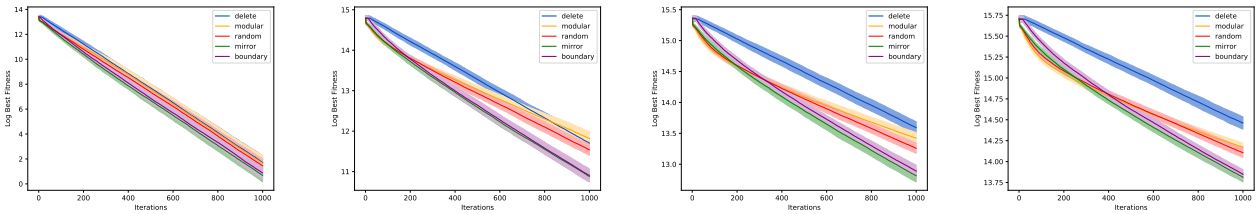
In this section, we first introduce the influence of dimension on the mapping rules, then divide them into two groups according to the characteristics and the performance of the mapping rules. The randomization and the modularization,

Attributes	Name	Expressions	Opt.$f(x)$
Unimodal	Sphere	$f(x) = \sum_{i=0}^{d} x_i^2$	0.0
	Cigar	$f(x) = x_0^2 + 10^6 \sum_{i=1}^{d} x_i^2$	0.0
	Discus	$f(x) = 10^6 x_0^2 + \sum_{i=1}^{d} x_i^2$	0.0
	Ellipse	$f(x) = \sum_{i=1}^{d} (10^{\frac{4(i-1)}{(d-1)}}) x_i^2$	0.0
Multi-modal	Step	$f(x) = \sum_{i=0}^{d} (\lfloor x_i + 0.5 \rfloor)^2$	0.0
	Tablet	$f(x) = 10^4 x_0^2 + \sum_{i=1}^{d} x_i^2$	0.0
	Rosenbrock	$f(x) = \sum_{i=0}^{d} (1 - x_i)^2 + 100(x_{i+1} - x_i^2)^2$	0.0
	Griewank	$f(x) = \frac{1}{4000} \sum_{i=1}^{d} x_i^2 - \prod_{i=1}^{d} \cos(\frac{x_i}{\sqrt{i}}) + 1$	0.0
	Bohachevsky	$f(x) = \sum_{i=0}^{d} (x_i^2 + 2x_{i+1}^2 - 0.3\cos(3\pi x_i) - 0.4\cos(4\pi x_{i+1}) + 0.7)$	0.0

TABLE I: Benchmark Functions used for Evaluation

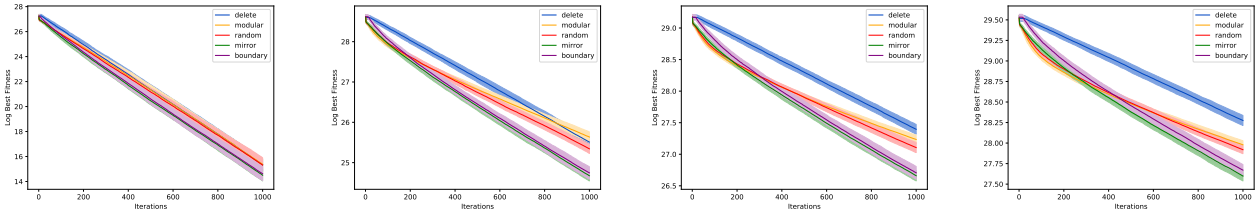
Dim \ Function	Sphere	Cigar	Discus	Ellipse	Rosenbrock	Bohachevsky	Griewank	Step	Tablet
100	5.2004e-10	2.0166e-08	0.0000	0.0000	3.8942e-25	1.9833e-07	2.0837e-31	3.9468e-18	1.5920e-83
400	6.8662e-56	1.6982e-55	0.0000	0.0000	2.1203e-50	5.7326e-58	2.0651e-298	2.3427e-59	0.0000
700	3.3005e-104	2.0809e-96	0.0000	0.0000	3.7699e-88	2.7724e-101	0.0000	4.7617e-99	0.0000
1000	4.5715e-142	3.0167e-142	0.0000	0.0000	2.1687e-123	7.5289e-145	0.0000	1.0750e-143	0.0000

TABLE II: P-value of Kruskal-Wallis Test by Ranks



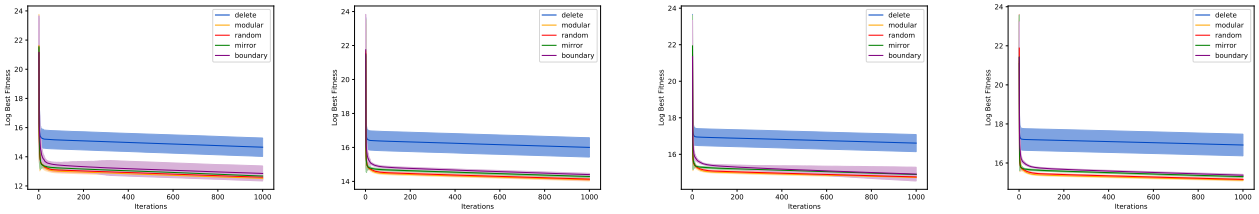
(a) dimensionality = 100 (b) dimensionality = 400 (c) dimensionality = 700 (d) dimensionality = 1000

Fig. 2: Sphere Function



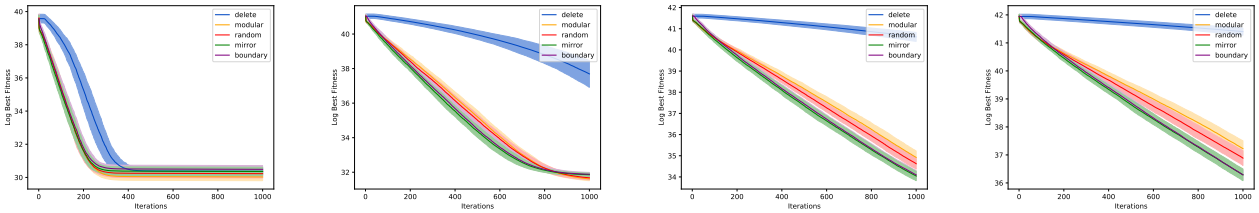
(a) dimensionality = 100 (b) dimensionality = 400 (c) dimensionality = 700 (d) dimensionality = 1000

Fig. 3: Cigar Function



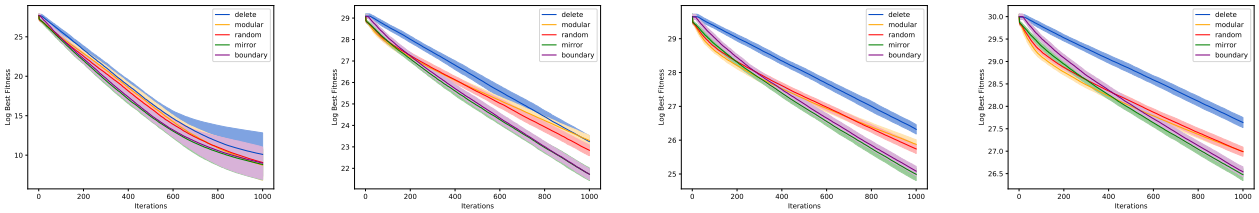
(a) dimensionality = 100 (b) dimensionality = 400 (c) dimensionality = 700 (d) dimensionality = 1000

Fig. 4: Discus Function



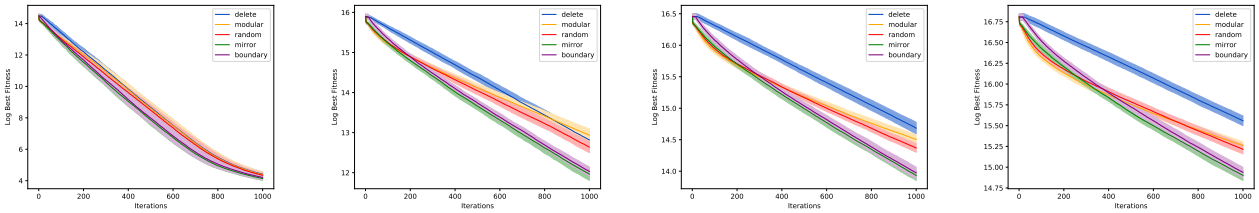
(a) dimensionality = 100 (b) dimensionality = 400 (c) dimensionality = 700 (d) dimensionality = 1000

Fig. 5: Ellipse Function



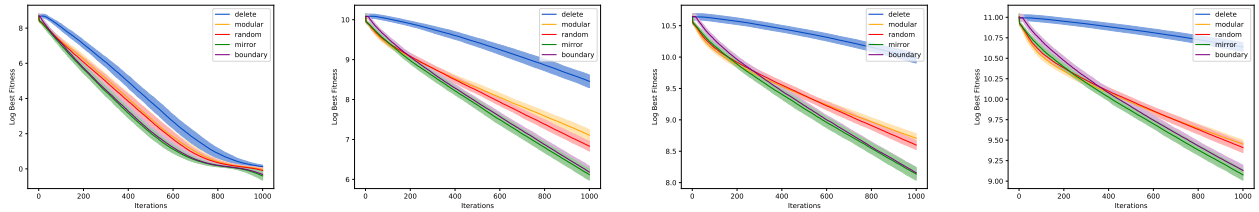
(a) dimensionality = 100 (b) dimensionality = 400 (c) dimensionality = 700 (d) dimensionality = 1000

Fig. 6: Rosenbrock Function



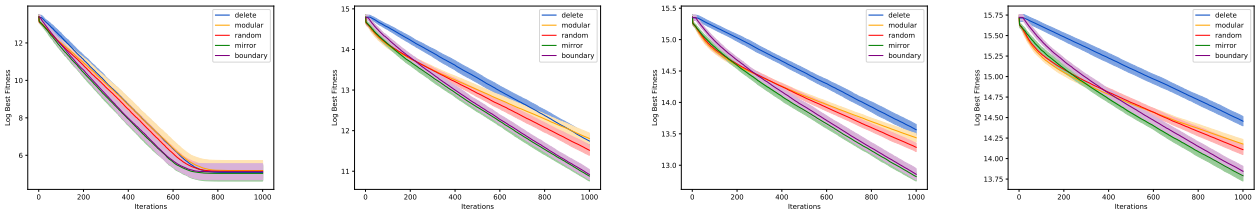
(a) dimensionality = 100 (b) dimensionality = 400 (c) dimensionality = 700 (d) dimensionality = 1000

Fig. 7: Bohachevsky Function



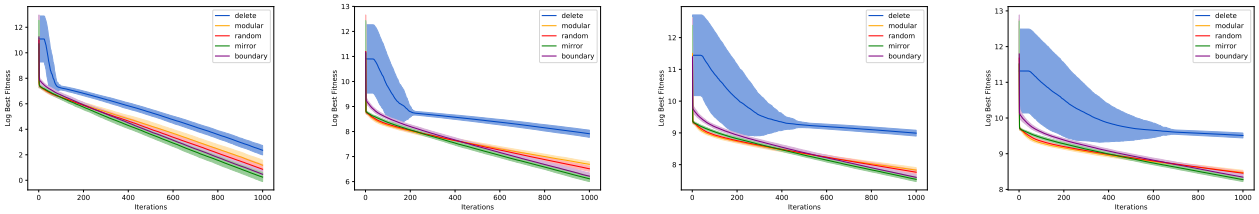
(a) dimensionality = 100 (b) dimensionality = 400 (c) dimensionality = 700 (d) dimensionality = 1000

Fig. 8: Griewank Function



(a) dimensionality = 100 (b) dimensionality = 400 (c) dimensionality = 700 (d) dimensionality = 1000

Fig. 9: Step Function



(a) dimensionality = 100 (b) dimensionality = 400 (c) dimensionality = 700 (d) dimensionality = 1000

Fig. 10: Tablet Function

the boundary rule and the mirror rule, they are respectively compared and analyzed. Finally, we analyze and discuss the performance of the five mapping rules in the entire iteration process.

A. Dimension

From figures 2 to 10, we can see that as the dimension increases, the influence of the mapping rules on the convergence speed increases. When the dimension is 100, the confidence intervals of the lines cover each other, and when the dimension is 1000, the distance between the lines increases. So in large-scale optimization problems, the effect of mapping rules on algorithm efficiency is great, which can be explained mathematically. The probability of sparks falling near the border is:

$$P = 1 - (1 - 2\epsilon)^d$$

d indicates the dimension, ϵ indicates boundary range, and P increases as d increases. When d rises to a certain value, P tends to 1. The closer the optimal point is to a certain boundary, the more points are out-of-bound. It is more important to handle the out-of-bound point. A good rule can utilize this high-dimensional mathematical property to improve the convergence rate.

B. Modularization vs. Randomization

Modular mapping rule were first proposed in conventional fireworks algorithm [1]. According to the formula (2): Assuming that the search range is uniform distributed as [-100,100], a dimension k of a newly generated spark x out-of-bounds, $x^k = 101$. It will be mapped to the location $\overline{x^k} = -100 + 101\%200$, $\overline{x^k}$ will eventually be mapped to $\overline{x^k} = 1$, This is very close to the origin. According to the BBFWA algorithm, the closer the firework is to the optimal point, the smaller the fireworks explosion amplitude. The sparks outside the boundary are also closer to the boundary. According to formula(2), modularization with a modulo operation, so that many of these sparks are mapped near the origin. With uniform sampling in the search space ($lb^k \equiv ub^k$), the modularization will have many sparks around the origin. As shown in Figure 1.

In high-dimensional cases, the probability of the optimal point falling on the boundary is far greater than the probability of falling on the original point. Therefore, this is a waste of computing resources. Obviously, this is not the most desirable rule. Even in low-dimensional cases, the modularization may makes the sparks away from the optimal point. So the enhanced fireworks algorithm [13] proposed a random mapping rule to fix the drawback of the modularization, in which sparks are uniformly mapped to the search space. Therefore, the randomization's global search capability is better than the modularization and its convergence speed will be higher. In low-dimensional cases, the randomization is the best mapping rule currently known, which is also widely used in the current fireworks algorithm.

Both of the two rules map sparks over the feasible area without using high-dimensional mathematical characteristics, moving many of the sparks away from the optimal point which is usually near the border. Unless some special functions need more search, (Such as the discus function, from the Table I, we can see its first dimension accounted for much larger than other dimensions. So it needs to explore between multiple dimensions to find out which the dimension is the best.) otherwise it is very inefficient undesirable.

The difference between the two rules is that the modularization maps some points near the origin so that reducing the search capability, which further weakens the competitiveness of the modularization. When some functions need to be explored more thoroughly, or when multi-modal functions have multiple local optima, the randomization will be useful.

C. Boundary vs. Mirror

The boundary mapping rule maps the spark to the boundary. If spark explosion happens on the boundary, half of the points outside the boundary and will be mapped on the boundary after the mapping. This reduces the sparks' diversity and exploration capability while wasting computing resources.

The mirror mapping rule makes the location of the spark a certain distance from the boundary, which increases diversity of the sparks.

Both of the two rules take advantage of the mathematical properties of high-dimensional cases, mapping sparks to the vicinity of the boundary. The two rules increase the search intensity near the boundary. At the expense of the global search capability, they achieve the local search capability near the boundary. This increases the probability of finding the

best point in high-dimensional cases. When it comes to low-dimensional situations, the optimal point is not necessarily located near the boundaries. In this situation, the mapped sparks may be far away from the optimal point. So both rules can be thought as tailor-made rule for large scale optimization.

D. Overall Analysis and Comparison

The behavior of the deletion is most undesirable. Because it does not use the high-dimensional math features, and does not increase search capacity. Removing sparks directly reduces the number and diversity of sparks, reducing the scope of the next generation of choices and requiring more time to find the optimum. It does not make full use of computing resources. Even in the case of low-dimensional, the deletion is not desirable.

Let us take a look at the other four rules. In the first 200 iterations, the randomization and the modularization outperformed the mirror rule and the boundary rule. That's because in the early phases, we don't have enough information about the optimal point, so global exploration capabilities are more important than local search capabilities. Global exploration capabilities ensured that the sparks landed throughout the feasible space.

When iteration increases to a certain value, less than 200 generally speaking, the mirror rule and the boundary rule outperforms the randomization and the modularization. At this time, the firework approaches the optimal point, which is known from the mathematical characteristics, close to the boundary. The boundary rule and the mirror rule increase the detection intensity at the boundary, accelerating the convergence process.

From the BBFWA algorithm, we know that when the next generation of fireworks is not better than itself, the explosion amplitude will decrease. So when the fireworks are close to the global or local optimum, the explosion amplitude becomes very small, sparks generated by the explosion barely go beyond the borders and the impact of mapping rules is diminished. Therefore, in the later stage of iteration, the performance of various mapping rules gradually converge.

In high-dimensional, the probability of random points falling near the boundary is very high. So why random mapping rule is less effective than the boundary rule and the mirror rule? That's because it is randomly mapped to any boundary, the mirror rule and the boundary rule map sparks to the specified boundaries, so their convergence are higher.

Although the randomization and the modularization outperformed the mirror rule in early iterations, the mirror rule and the boundary rule quickly catch up with them in a few iterations. The mirror rule and the boundary rule are more stable. According to the comparison of the rules, the mirror rule is better, so in high-dimensional general cases, the mirror rule is the most preferable rule known so far.

But it can be seen that the curve of the discus function is special, because the mathematical expression of the discus function is: $f(x) = 10^6 * x_0^2 + \sum_{i=0}^{d} x_i^2$, x_0 is more important than other dimensions, it is difficult to optimize the other $(d-1)$ dimensions. so the exploration is more important than exploitation capacity. This fact reminds us, we need to consider the characteristics of function itself in some cases.

VI. Conclusion

In this paper, we investigate whether there is a better mapping rule for large scale optimization. Experiments on the selected 9 functions with 4 different dimensions present high-dimensional cases have special mathematical properties. We analyze the underlying causes of the different performance of each mapping rule. The modular mapping rule was first proposed and was replaced by the random mapping rule. It gathered sparks to the origin, reduced exploration capabilities, and did not take advantage of high-dimensional math features. So it is not desirable in either high or low dimensional problem. The random mapping rule was proposed in order to fix the drawbacks of the modularization. It improves the search ability of the spark. The randomization is very suitable in the low-dimensional case and undesirable in the high-dimensional case. The deletion is the worst method, reducing the number and variety of sparks, and possibly even affecting the convergence of the algorithm. It is the least recommended rule. The boundary mapping rule and the mirror mapping rule both utilize the high-dimensional math features, but the boundary rule reduces the spark diversity. Yet the mirror rule is more flexible and good for exploration, so it works better and more consistently. The mirror mapping rule is the most desirable rule found so far.

The conclusion of this paper is based on a particular version of fireworks algorithms (BBFWA). To investigate whether the research conclusion is suitable for other algorithms, future work will extend the mapping rules proposed in this paper to other fireworks algorithms, such as enhanced fireworks algorithm (EFWA), dynamic fireworks algorithm (DynFWA). We aim to develop more efficient mapping rule to improve the performance of the fireworks algorithm.

Acknowledgment

This work was supported by the Natural Science Foundation of China (NSFC) under grant no. 61673025 and partially supported by National Key Basic Research Development Plan (973 Plan) Project of China under grant no. 2015CB352302.

References

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

[2] Y. Fu, J. Ding, H. Wang, and J. Wang, "Two-objective stochastic flow-shop scheduling with deteriorating and learning effect in industry 4.0-based manufacturing system," *Applied Soft Computing*, 2017.

[3] C. Manickam, G. P. Raman, G. R. Raman, S. I. Ganesan, and N. Chilakapati, "Fireworks enriched p&o algorithm for gmppt and detection of partial shading in pv systems," *IEEE Transactions on Power Electronics*, vol. 32, no. 6, pp. 4432–4443, 2017.

[4] S. Reddy, L. K. Panwar, B. Panigrahi, and R. Kumar, "Modeling of carbon capture technology attributes for unit commitment in emission-constrained environment," *IEEE Transactions on Power Systems*, vol. 32, no. 1, pp. 662–671, 2017.

[5] M. Guendouz, A. Amine, and R. M. Hamou, "A discrete modified fireworks algorithm for community detection in complex networks," *Applied Intelligence*, vol. 46, no. 2, pp. 373–385, 2017.

[6] S. Ye, H. Ma, S. Xu, W. Yang, and M. Fei, "An effective fireworks algorithm for warehouse-scheduling problem," *Transactions of the Institute of Measurement and Control*, vol. 39, no. 1, pp. 75–85, 2017.

[7] E. Tuba, M. Tuba, and E. Dolicanin, "Adjusted fireworks algorithm applied to retinal image registration," *Studies in Informatics and Control*, vol. 26, no. 1, pp. 33–42, 2017.

[8] J. Liu, S. Zheng, and Y. Tan, "Analysis on global convergence and time complexity of fireworks algorithm," in *Evolutionary Computation (CEC), 2014 IEEE Congress on*. IEEE, 2014, pp. 3207–3213.

[9] J. Li and Y. Tan, "The bare bones fireworks algorithm: A minimalist global optimizer," *Applied Soft Computing*, vol. 62, pp. 454–462, 2018.

[10] K. Ding, S. Zheng, and Y. Tan, "A gpu-based parallel fireworks algorithm for optimization," in *Proceedings of the 15th annual conference on Genetic and evolutionary computation*. ACM, 2013, pp. 9–16.

[11] K. Ding and Y. Tan, "Attract-repulse fireworks algorithm and its cuda implementation using dynamic parallelism," *International Journal of Swarm Intelligence Research (IJSIR)*, vol. 6, no. 2, pp. 1–31, 2015.

[12] S. A. Ludwig and D. Dawar, "Parallelization of enhanced firework algorithm using mapreduce," *International Journal of Swarm Intelligence Research (IJSIR)*, vol. 6, no. 2, pp. 32–51, 2015.

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

[14] S. Zheng, A. Janecek, J. Li, and Y. Tan, "Dynamic search in fireworks algorithm," in *Evolutionary Computation (CEC), 2014 IEEE Congress on*. IEEE, 2014, pp. 3222–3229.

[15] J. Li, S. Zheng, and Y. Tan, "The effect of information utilization: Introducing a novel guiding spark in the fireworks algorithm," *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 1, pp. 153–166, 2017.

[16] Y. Pei, S. Zheng, Y. Tan, and H. Takagi, "An empirical study on influence of approximation approaches on enhancing fireworks algorithm," in *Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on*. IEEE, 2012, pp. 1322–1327.

[17] J. Li and Y. Tan, "Loser-out tournament based fireworks algorithm for multi-modal function optimization," *IEEE Transactions on Evolutionary Computation*, 2017.

[18] J. Li, S. Zheng, and Y. Tan, "Adaptive fireworks algorithm," in *Evolutionary Computation (CEC), 2014 IEEE Congress on*. IEEE, 2014, pp. 3214–3221.

[19] S. Cheng, Q. Qin, J. Chen, Y. Shi, and Q. Zhang, "Analytics on fireworks algorithm solving problems with shifts in the decision space and objective space," *International Journal of Swarm Intelligence Research (IJSIR)*, vol. 6, no. 2, pp. 52–86, 2015.