Explosion Gravitation Field Algorithm with Dust Sampling for Unconstrained Optimization

Xuemei Hu[1], Lan Huang[1,2],Yan Wang[1,*],Wei Pang[3,4*]

[1]College of Computer Science and Technology, Jilin University. Changchun, 130012, China.

Key Laboratory of Symbol Computation and Knowledge Engineering of Ministry of Education, Jilin University. Changchun, 130012, China.

[2]Zhuhai Laboratory of Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Department of Computer Science and Technology, Zhuhai College of Jilin University, Zhuhai, 519041, China.

[3]Department of Computing Science, University of Aberdeen, AB24 3UE, United Kingdom.

[4]Shaanxi Key Laboratory of Complex System Control and Intelligent Information Processing, Xi'an 710048, China

Corresponding authors: wy6868@jlu.edu.cn; pang.wei@abdn.ac.uk

Abstract

Gravitation Field Algorithm (GFA) is a novel optimization algorithm derived from the Solar Nebular Disk Model (SNDM) in astronomy and inspired by the formation process of planets. Although it has achieved good performance when solving many unconstrained optimization problems, which demonstrated its promising application potential in many real-world problems, GFA still has much room for improvement, especially when it comes to the accuracy and efficiency of the algorithm.

In this research, an improved GFA algorithm called Explosion Gravitation Field Algorithm (EGFA) is proposed for unconstrained optimization problems, with the introduction of two strategies: Dust Sampling (DS) and Explosion Operation. The task of DS is to locate the space that contains the optimal solution(s) by initializing the dust population randomly in the problem search space; while the Explosion Operator is to improve the accuracy of solutions and decrease the probability of the algorithm falling into local optima by generating the new population around the center dust to replace the original population.

A comparison of experimental results on six classical unconstrained benchmark problems with different dimensions demonstrates that the proposed EGFA outperforms the original GFA and several classical metaheuristic optimization algorithms, such as Genetic Algorithm (GA) and Particle Swarm

Optimization (PSO), in terms of accuracy and efficiency in lower dimensions. Additionally, the comparison of results on three real datasets indicate that EGFA performs better than the original GFA and k-means for solving clustering problems.

Key words：

Explosion gravitation field algorithm; Unconstrained optimization; Dust Sampling; Explosion operation

1. Introduction

Optimization problems exist in almost all fields of science and engineering, and they can be divided into two categories in terms of if there exist constrained conditions upon these problems: constrained optimization problems and unconstrained ones. Unconstrained optimization problems have been studied extensively for a long time, but we are still facing challenges in acquiring desirable solutions for many of these problems in terms of accuracy and efficiency. Traditional mathematical methods for solving these unconstrained optimization problems, such as the steepest descent method [1], the Newton method [2], the conjugate gradient method [3], and the quasi-Newton method [4], normally require or assume that the optimization problems are continuous and differentiable and often some prior knowledge is needed in advance. This makes it challenging for the above traditional algorithms to solve problems of higher dimensions.

To address the above challenges, many novel computational intelligence (CI) algorithms have been proposed over the past several decades, some of which have broken through the above rigorous limitations effectively. The significant advantages enable CI algorithms to be widely adopted to solve many unconstrained optimization problems. Over the past two decades, there have been many effective computational intelligence algorithms proposed, which are inspired by natural phenomena. For instance, Simulated Annealing (SA) algorithm proposed by Metropolis *et al*. in 1953 was inspired by annealing in metallurgy. Genetic algorithm (GA) [5][6], originally proposed by Holland in 1975, emulates the natural evolution selection process, and it is based on the Darwinian biological evolution theory. Ant colony optimization (ACO) [7][8] proposed by Dorigo in 1992 is based on the heuristic process of ant's food discovery and incorporated the communication mechanisms between the colony members. Particle swarm optimization (PSO) [9][10] proposed by Eberhart and Kennedy in 1995 simulates a simplified

social model [11].

Furthermore, there has been an innovative CI algorithm boom over the last twenty years. In 2013, Xing *et al*. [12] identified a vast amount of novel CI algorithms (more specifically, 134 in total) and grouping them into four large classes, i.e., biology- (99 in total) [13-16], physics- (28 in total) [17-19], chemistry- (5 in total) [20], and mathematics-based (2 in total) [21] CI algorithms. The core algorithm concerned in this research is the gravitation field algorithm (GFA), which belongs to the second class (physics-based CI). GFA [22-24] proposed by Zheng *et al.* in 2012 simulates the formation process of planets based on the Solar Nebular Disk Model (SNDM) [25] in astronomy. Based on the original GFA, we have developed an improved version of GFA called GFA-OD [26] (Optimal Detection). Almost all of these innovative CI algorithms have good performance and some of them have been successfully applied in many real-world optimization problems including unconstrained ones.

In this research, based on the original GFA and the earlier version of GFA-OD, we propose an improved version of GFA, which is called Explosion Gravitation Field Algorithm (EGFA), and we compare EGFA with popular approaches, including the original GFA, GA and PSO. More specifically, in EGFA, we develop an improved strategy called Dust Sampling (DS) to locate the space that contains the optimal solution(s) by initializing dust population randomly in a given problem search space, and this makes the algorithm more efficient. Additionally, to improve the accuracy of solutions and decrease the probability of the algorithm falling into local optima, an improved strategy named Explosion Operation is introduced, the aim of which is to improve the performance of EGFA by generating new dust population around center dust to replace the original dust. For implementing EGFA, we integrate two processes: movement and rotation, and modify the formula for updating dust population. From the experimental results on six complex unconstrained benchmark problems, we find that EGFA is superior to the original GFA and two other classical optimization algorithms (GA and PSO) in terms of accuracy and efficiency. In addition, further experiments show that EGFA outperforms the original GFA and k-means on three real datasets. All the experimental results demonstrate the application potential of EGFA in more complex problems.

2. Explosion Gravitation Field Algorithm

2.1 The Original Gravitation Field Algorithm

The original GFA was proposed in [23] as a novel nature-inspired heuristic search algorithm. The basic idea of GFA is to simulate the formation process of planets based on SNDM [25]. In GFA, all individuals can be mimicked as dust with mass, and each of them belongs to a certain group. In each group, the one with the biggest mass is regarded as the center dust and others are surrounding dust. Based on SNDM, each center dust attracts its surrounding dust by the gravitation field, and the field makes all surrounding dust move toward their center dust with heaviest mass. In addition, each dust has four characteristics: position, mass, group number, and a Boolean flag indicating whether it is a center. The position corresponds to a solution of the problem, the group number is initialized randomly, and the other two: mass and flag, are determined by the objective function. The flow chart of GFA is given in Fig.1. The details of GFA are summarized as follows, and all the parameters are set according to Zheng *et al.* [23].

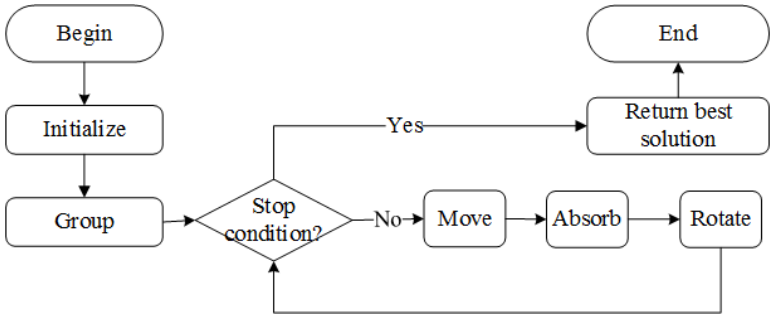

Fig. 1 The workflow of GFA

- Step 1: generate n dusts $D_i\left(i=1,2,\cdots n\right)$ randomly distributed in the mass function domain bound to establish the initial solution space when all parameters are set, where the position of the ith dust is defined by Eq. (1).

$$X_i=\left(x_{i,1},x_{i,2},\cdots,x_{i,k},\cdots,x_{i.d}\right),\ i=1,2,\cdots,N \tag{1}$$

In the above $x_{i,k}$ is the position of the ith dust in the kth dimension, and d is the number of dimensions of search space. In addition, every dust (solution) D_i is assigned a mass M_i, whose values are calculated based on the objective function or the benchmark function.

- Step 2: divide the search space into several subspaces randomly. Each dust D_i is allocated to a

subspace (called a group in GFA) randomly and the group number G_i is calculated by Eq. (2), where *ceil* is a function that rounds the elements to the nearest integers greater than or equal to it. In each subspace, the dust with the heaviest mass is called the center dust and we assign its flag $F_i = 1$, and others are the surrounding dusts and we assign their flags as $F_i = 0$.

$$G_i = ceil\left(rand[0,1] * group\_num\right) \tag{2}$$

- Step 3: move dust. Every surrounding dust move toward its center dust and the center dust remain stationary. The pace of movement is determined by Eq. (3).

$$Pace_i = w \times dis_i, \tag{3}$$

where dis_i is the Euclidean distance between the moving surrounding dust and its center dust in [22] and is also defined as the difference between two vector variables in [23]. w is a weight value for distance [22].

- Step 4: absorb dust. Some surrounding dusts which are close to their center dust enough are absorbed by their center dust and therefore eliminated from the initial search space for increasing the convergence speed of GFA.
- Step 5: perform rotation operation. The introduction of Rotation Operator in [23] is to push the surrounding dusts away from their center dust, which can decrease the probability of dusts falling into local optima to some extent. The rotation direction is not the original forward direction, but it could be any possible random direction. To prevent the surrounding dusts from being pushed too far, the pace is no greater than the max pace $withdraw_{\max}$, which is defined by Eq. (4).

$$withdraw_{\max} = 2 \times dis \times 0.0618 \tag{4}$$

The rotation operation is performed with the probability of Rotation Factor (RF) [23]. The value of RF is defined in Eq. (5).

$$factor^{t+1} = \begin{cases} factor_{\max}, & factor^t \geq factor_{\max} - 0.03 \\ factor^t + 0.03, & factor^t < factor_{\max} - 0.03 \end{cases}, \tag{5}$$

where $factor^{t+1}$ is the RF in the $(t+1)th$ iteration, $factor_{\max}$ is the max value of RF. This value cannot be too large, and it will gradually increase along with the running process of the algorithm. In [23], $factor^0 = 0$, and after several iterations, the RF will reach its max value and remain unchanged.

- Step 6: check the termination criterion. If the algorithm does not meet the stopping condition, GFA will go to Step 3, otherwise, the algorithm will terminate.

Algorithm 1: GFA

1. *initialdusts* ← ***Initialize****(bound, N, targetFun, M)* //initial
2. *[groupdusts, center]* ← ***Group****(initialdusts, G)* //divide search space into groups
3. *dusts* ← *groupdusts*
4. **while** the stop condition is not met **do**
5. *movedusts* ← ***Move****(dusts, center, targetFun)* //move
6. *absorbdusts* ← ***Absorb****(movedusts)* //absorb
7. *rotatedusts* ← ***Rotate****(absorbdusts, targetFun)* //rotate
8. *dusts* ← *rotatedusts*
9. **end**
10. **return** best solution(s)

According to the basic steps of GFA described above, every dust can be represented by a quadruple $\left(X_i, M_i, G_i, F_i\right)$, and the pseudo-code of the original GFA model is given in **Algorithm 1**, where N is the size of the population, M is the number of iterations; G is the number of groups; and *bound* is a $d*2$ matrix that records the boundary of search space. The first five steps of GFA previously mentioned correspond to five procedures: '***Initialize'***, '***Group'***, '***Move'***, '***Absorb'*** and '***Rotate'***. It is obvious that these five procedures have a computational complexity of $O\left(N\right)$, where N is the size of population, and the whole GFA model requires an $O\left(M \times N\right)$ of time complexity since the number of iteration M is one of the termination criteria.

The original GFA has been used to optimize several unconstrained optimization problems, and it has demonstrated excellent performance in these problems. It was reported that GFA was very effective to find approximate optimal solutions for several unconstrained optimization problems. However, there are still some limitations for the original GFA, as shown below:

1） The solutions that GFA obtains are particularly sensitive to the initial population. It is reported that the accuracy of solution is closely related to the initial population [26]. GFA tends to find a solution that usually could not meet the accuracy requirement of problems in given running time in many real-world problems if the initial dust population is generated randomly in the whole search space.

2） GFA may be in stagnation behavior easily during the iteration process because of falling into local optima, and in these cases the algorithm will return a local optimal solution or even a worse one if there is no appropriate strategy to jump out from the bad results.

2.2. Explosion Gravitation Field Algorithm

To address the limitations of GFA presented above, an improved Explosion Gravitation Field Algorithm (EGFA) was proposed with three aspects of improvement: two of them are two improved strategies called Dust Sampling (DS) and Explosion Operation in this research, and another one is the integration of the two processes: move and rotate. DS is performed when the parameter setting is completed and it is no longer performed once this process is over. It can efficiently locate a small enough search space which more likely contains the optimal solution(s). The second aspect of improvement, the Explosion Operation, is performed once one of the pre-specified conditions is met. The Explosion Operation is used to jump out from a local optimal solution with a certain probability and therefore improve the accuracy of solutions. Furthermore, to simplify the process of EGFA, there is another improvement: the processes of movement and rotation are integrated, and the formula for updating the dust population is modified. The whole core aspects of improvement in EGFA are described below and the workflow of EGFA is given in Fig. 2.

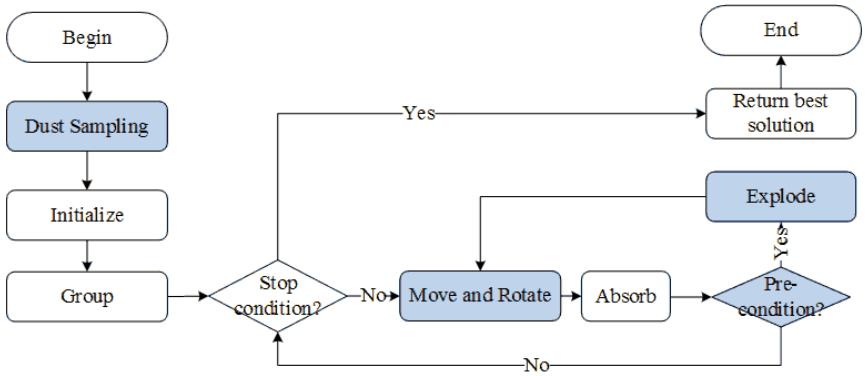

Fig. 2 The workflow of EGFA

2.2.1 Subspace location by dust sampling

As mentioned in Section 2.1, the original GFA starts with dust population being initialized randomly in a giving search space, this makes it a long iterative process to find a solution that meets the accuracy requirement for a given problem. The solution that the original GFA finds may be a suboptimal one whose accuracy could not be comparable to the solutions found by other classical metaheuristic search algorithms, such as GA and PSO. Therefore, to improve the accuracy and shorten the overall running time, we proposed a subspace location strategy named Dust Sampling (DS). The DS operator is not only

an important improvement in this research but also the core component of EGFA. The task of DS is to efficiently locate a small enough search space which more likely contains the optimal solution(s), and this shares some similarities with our original Optimal Detection (OD) operator [26]. The readers are referred to [26] for more details about the original OD operation.

The DS operation begins with all dusts being initialized randomly in the given search space of a problem, then as shown in Fig. 3.(a), the dusts in the top 20% of the population are selected to calculate the boundary of the '*Subspace'*, and the solution space containing the other 80% of dusts population is named *'Non-Subspace'*. The top 20% of dusts are selected based on their mass values calculated from the mass function, similar with the fitness values in GA and PSO. In the next DS, 80% of dusts are generated in the '*Subspace'* and just 20% of the dusts are generated in the '*Non-Subspace*' randomly, which is illustrated in Fig. 3.(b). This process will be executed several times until the '*Subspace'* is small enough. According to the above description, the pseudo-code of DS can be concluded in **Algorithm 2**.

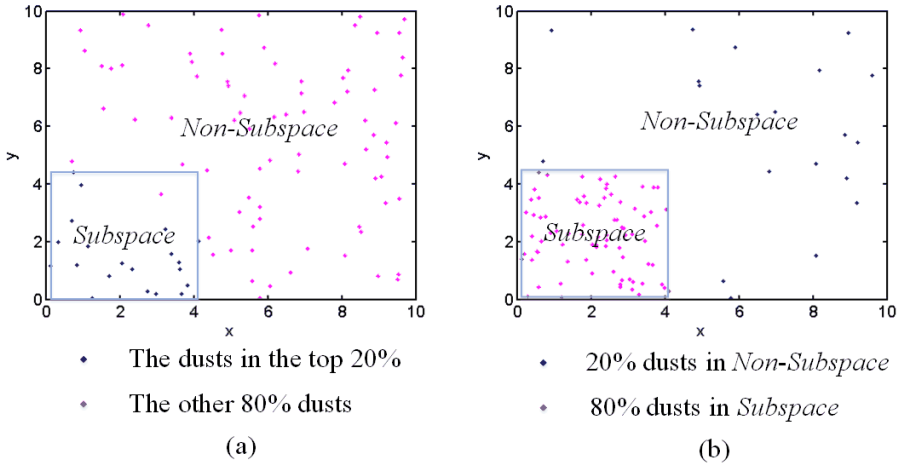

Fig. 3 The distribution of the dusts in *Subspace* and *Non-Subspace*

In [26], the algorithm can obtain a correct solution only when the hypothesis that the '*Subspace'* found by OD contains the optimal solution(s) is valid. If OD finds a wrong '*Subspace'*, the algorithm can only obtain a suboptimal solution. To overcome this limitation and decrease the probability of getting a suboptimal solution, every dust generated by DS is located in the '*Non-Subspace'* with the probability of 20%. Even if the '*Subspace'* does not contain the optimal solution(s), DS will still have a chance to locate the position of optimal solution(s) in the '*Non-Subspace'*. The choice of 20% and 80% when dividing the dusts for subspace and non-subspace is based on the Pareto principle, and in the future work we could further explore whether and how we can further improve this choice with better division.

The reason for the DS operator shortening the running time is because the main loop of the

algorithm just needs to find the optimal in a smaller search space and avoid a long iterative process to find a solution that meets the accuracy requirement, and this shares some similarities with our original OD [26]. However, a key to the original OD is that users have to specify the number of iterations and the size of the dusts population for the process to acquire a satisfactory '*Subspace'*, but it may be difficult for inexperienced users to provide such appropriate values. If the values are too large, the OD operation may take a long time, which lead to the decrease of the algorithm efficiency; if the value is too small, the algorithm may not be able to find a solution that meets the requirement of accuracy. The DS overcomes the above limitation and simplify the process of parameter setting for users. This operation provides default values for the number of iterations and the size of dusts population, according to the dimensions of the given problems. Specifically, the operation provides larger values for the complex problems with larger number of dimensions and provides smaller values for the uncomplex problems with smaller number of dimensions. The number of iterations and the size of population for the process increase with the increase of the complexity of a given problem. Let M_1 be the number of iterations for this process, and it is set as the stopping condition; if N is the size of population and d is the number of dimensions of the given problem, the process requires $O(M_1 \times N \times d)$ time complexity.

Algorithm 2: Subspace location by dust sampling
1. *Subspace* ← *Solution Space*;
2. *Non-Subspace* ← *Solution Space - Subspace*;
3. **While** the stopping conditions are not met **do**
4. **if** *Non-Subspace* is null **then**
5. generating all dust population in the *Subspace*;
6. **else**
7. generating 80% dust population in the *Subspace;*
8. generating 20% dust population in the *Non-Subspace*;
9. **end**
10. picking up the top 20% of the population and calculating the boundary of the subspace;
11. *Subspace* ← the boundary of the subspace;
12. *Non-Subspace* ← *Solution space – Subspace;*
13. **end**

Obviously, the fact that the number of the iterations and the size of dust population do not need to be assigned by the users in DS, and this makes the algorithm more robust to problems of different complexity compared with the original OD operation.

2.2.2 Explosion Operation

The algorithm may still fall into local optima or cannot find a solution that meets the accuracy requirement for the given problem if the '*Subspace'* identified by **Algorithm 2** does not contain the actual optimal solution. Regarding this, we introduce the Explosion operation to decrease the probability of dust falling into local optima compared with the original GFA, GA and PSO. The algorithm will perform the explosion operation when one of the following conditions is triggered.

1） *The algorithm is in stagnation behavior.* The explosion operation will be executed since the solutions that the algorithm finds will not be better in the future iteration process.

2） *The size of population is less than a pre-specified threshold*. The explosion operation is needed since the solution is less likely to get better in this case.

When one of above two conditions is triggered, all dusts will be removed except the center dust in each group. Then the algorithm will perform the explosion operation and generate a new population to replace the original one. The generation of the new population is not random, and it uses the knowledge from the last population. The process of explosion for one group is illustrated in Fig. 4.

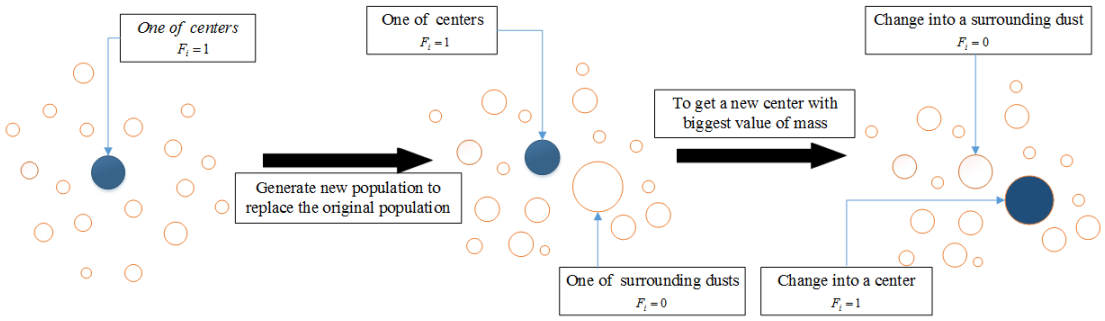

Fig. 4 One of the centers generate a new population to replace the original one

As shown in Fig. 4, the biggest blue dot with $F_i=1$ in a group is the center dust and other dots are the surrounding dust with $F_i=0$. Obviously, the task of Explosion Operation is to generate a new population around the center, and then attempt to find a better solution among the new population. The distribution of the new population is given by Eq. (5).

$$X_i^k = s\times(ub_k - lb_k)\times c\times normrnd[0,1]+X_{center_i}^k, k=1,2,\ldots,d, \quad (5)$$

where d is the number of dimensions, s is a positive or negative sign generated randomly in a mathematical operation, ub_k and lb_k are the upper and lower bounds of the kth dimension in search

space. The Gaussian function $normrnd[0,1]$ determines how new dust will be distributed in the *k*-dimensional search space, where $0 < normrnd[0,1] < 1$, and the mean and standard deviation for the Gaussian function are 0.025, 0.25, respectively. The coefficient c determines the degree of aggregation for dusts in each group, where $0 < c < 1$. For each group, the less the value of c is, the more concentrated the distribution of the dust is; the larger the value of c is, the more dispersed the distribution of dust is.

The Explosion is performed when one of the following two conditions is triggered: (1) the algorithm is in the stagnation behavior, and (2) the size of population is less than a pre-specific threshold. Obviously, the best solution the algorithm gets must be one of the centers. Provided that the algorithm has found an approximate optimal solution and the size of the population has been less than the pre-specified threshold, it is reasonable to believe that the better solution exists near one of the center dusts with high probability and the algorithm needs more dusts to explore the search space. Thus, the above approach can be used to find a better solution and improve the accuracy of historical best solution quickly. Additionally, if the algorithm has been in stagnation behavior because of falling into local optima, the algorithm can jump out from the local optimal solution with the help of surrounding dusts far away from the center dust with a certain probability. Therefore, the explosion operator introduced in EGFA can also be used to prevent the algorithm from being in stagnation behavior.

Algorithm 3: Explode

1. **for** *i = 1:C* //C is the size of each group
2. **for** *j = 1:G* //for each dust in a group
3. $index \leftarrow (i-1) * G + j$
4. **for** *k = 1:d* %for the *kth* demonstration
5. $X_i^k \leftarrow s \times (ub_k - lb_k) \times c \times normrnd[0,1] + X_{center_j}^k$ //the position of new dust
6. **end**
7. $M_{index} \leftarrow f(X_{index})$ //7-9 generate new dust
8. $G_{index} \leftarrow j$
9. $F_{index} \leftarrow 0$
10. **end**
11. **end**

Following the details about Explosion described above, the pseudo-code of one of the centers generating a new population to replace the original one is presented in **Algorithm 3**, which is the main part of Explosion operation. In **Algorithm 3**, G is the number of groups; N is the size of the population; C is the size of each group; $N = C \times G$; d is the number of dimensions of the search

space for a given problem. This process requires an overall of $O(C\times G\times d)=O(N\times d)$ computations. Note that only $O(N)$ storage is required for this process.

2.2.3 The Main loop

EGFA proposed in this research is an improved version of GFA, which introduces the DS and Explosion operations to improve the algorithm performance. In addition, to simplify the process of EGFA, we integrate the process of movement and rotation, and modify the formula for iteration. As a result, any dust D_i in a given search space is described by a quadruple $D_i=(X_i,M_i,G_i,F_i)$ and the new generation for the next iteration in the EGFA can be determined by Eq. (6).

$$X_i^{t+1}=X_i^t+Pace_i^t \quad , \tag{6}$$

where $Pace_i^t$ and X_i^t are the current pace and position of dust D_i at time t, respectively, and $Pace_i^t$ is determined by three components as follows:

$$Pace_i^t=\left(w_1*dis_i^t+w_2*Ibest^t\right)*\left(1-F_i\right)+w_3*rand\left(1,d\right), \tag{7}$$

where d is the dimension of the search space, w_1,w_2,w_3 stands for the weight values of the three parts, respectively, $w_i>w_2>0$, $w_3>0$, $F_i\in\{0,1\}$ (F_i is equal to 1 when D_i is a center, otherwise, F_i is equal to 0).

- The first component dis_i^t is the directional difference of the position between the moving surrounding dust D_i and its center at time t, which is calculated by Eq. (8) in EGFA.

$$dis_i^t=X_{center_{G_i}}^t-X_i^t \quad , \tag{8}$$

  Where X_i^t and $X(t)_{center_{G_i}}^t$ are the position of dust D_i and its center at time t, respectively.

- The second component $Ibest^t$ stands for the improvement vector of the global best's position at time t, and it is calculated by Eq. (9).

$$Ibest^t=X_{best}^t-X_{best}^{t-1} \quad , \tag{9}$$

  where X_{best}^t is the position of current global best dust and X_{best}^{t-1} is the stored position of the global best dust from the last generation. The concept of global best in this manuscript is similar to the

concept of global best Artificial Bee Colony (ABC), and reader can refer to [27] for more details about the concept. The main objective of this component is to modify the representative point by taking into account global directional moves so that to attract the path going to optima and reduce the time for reaching the global optima.

- The third component $rand(1,d)$ is a $1\times d$ matrix and produces random movements for all dusts, including the surrounding dusts and their centers, like all the planets in the universe are always in motion. As a result, the centers in EGFA are no longer still, and they move randomly toward any possible direction like surrounding dusts, which could help find a better center and improve the diversity of the population.

Furthermore, dust recovery is another interesting topic with regards to EGFA due to the fact that dust may move outside the search space and should be returned. There are many possible dust recovery methods. A useful one adopted in EGFA is the reposition factor (Frep for short) [19], which plays an important role in EGFA's convergence. They are described in Eq. (10) and Eq. (11) as below.

$$X_{i,k}^{t} = X_{k,\min} + \varepsilon\left(X_{i,k}^{t-1} - X_{k,\min}\right) \tag{10}$$

$$X_{i,k}^{t} = X_{k,\max} - \varepsilon\left(X_{k,\max} - X_{i,k}^{t-1}\right) \ , \tag{11}$$

where k is the current dimension; t is the current time step; i is the current dust index; and ε is a small positive number chosen by the user, typically 0.0005. EGFA uses Eq. (10) to reposition dimension of dust that have exceeded their minimum values, while Eq. (11) is used to reposition dimensions of dust that have exceeded their maximum values.

According to the above details of EGFA, the pseudo-code of the main steps of EGFA is given in **Algorithm 4**. In the above variable k is a monitor, which is employed to record the number of iterations of the algorithm being in stagnation behavior during the search process, variable '*bound*' gives the value range of X_i in all dimensions for each solution D_i, N is the size of population, G is the number of groups, d is the number of the dimensions of the search space.

The method '***DustSampling'*** corresponds to the process of Dust Sampling presented in Section 2.2.1 and stores the information of '*Subspace'* with the variable '*BestBound'*. The method '***MoveAndRotate***' integrates the process of movement and rotation and corresponds to Eq. (6) for iteration. The method '***Explode***' corresponds to the process of explosion described in Section 2.2.2 and there exists a very important threshold variable f_{th} in the method, which is usually proportional to the

number of the iterations. If the condition $k > f_{th}$ is met, the method '***Explode'*** will be performed. The method '***Initialize'***, '***Group'***, '***Absorb'*** still corresponds to the same processes as in the original GFA. The method '***GetCenters***' is used to get the center dust for each group and the method '***GetBest***' is devoted to update the historical best solution. Let M_2 be the number of iterations for the main loop and set as one of the stop conditions. Obviously, the methods '***Initial'***, '***Group'***, '***MoveAndRotate***', '***Absorb',*** '***GetCenters'*** and '***GetBest'*** just require $O(N)$ time complexity. Since **'*DustSampling'*** requires $O(M_1 \times N \times d)$ time complexity, and **'*Explode'*** requires $O(N \times d)$ time complexity, the main loop of EGFA require an overall of $O((M_1 + M_2) \times N \times d)$ time complexity. Note that only $O(N)$ space complexity is required for EGFA.

Algorithm 4: EGFA

1. *k=0*
2. *bestBound* ← ***DustSampling*** *(N, bound, targetFun);* //dust sampling
3. *initialdusts* ← ***Initialize*** *(bestBound, N, targeFun)* //initialisation
4. *[groupdusts,center]* ← ***Group*** *(initialdusts, G)* //group
5. *dust* ← *groupdusts*
6. **while** the stop conditions are not met **do**
7. *movedust* ← ***MoveAndRotate*** *(dust, center, targetFun);* //move and rotate
8. *absdust* ← ***Absorb*** *(center, movedust, bestBound);* //absorb
9. *dust* ← *absdust*
10. **if** pre-condition is met **then**
11. *[dust,k]* ← ***Explode*** *(k, dust, N, bound, targetFun, center);* //explode
12. **end if**
13. *[center,dust]* ← ***GetCenters*** *(G, dust);* //get the center dusts of each group
14. *[best,k]* ← ***GetBest*** *(best, center, k);* // the best dust
15. **end**
16. **return** optimal solution

The two strategies proposed in EGFA overcome the limitations of the original GFA. However, the introduction of these two strategies is at the cost of running time. It is noted that DS avoids the long iterative process and shortens the running time, although the time complexity has increased to $O((M_1 + M_2) \times N \times d)$. Last but not least, to prove the capacity in theory, the research discusses the convergence of EGFA in one-dimension simply in the supplementary material. As for the convergence in higher dimensional search space, it is a part of research in future work.

3. Experiments

3.1 Experiments on Benchmark problems

To assess the performance of EGFA proposed in this research, the following six benchmark unconstrained optimization problems in Table 1 are chosen, which are used to test the accuracy, running time of our algorithm with problems of different dimensions. At the same time, this series of classical test problems are also solved by the original GFA, GA and PSO to compare with EGFA. 100 trials of each algorithm are performed for solving these six benchmark problems, and we choose the mean value, the median value, the mean squared error and the Standard Deviation of solutions to evaluate the performance of the four algorithms. Finally, we present the experimental results and the discussions upon these results.

3.1.1 Benchmark problems and performance evaluation

The functions listed in Table 1 are some of the most commonly used functions used to assess the performance of unconstrained optimization algorithms. These benchmark problems are chosen from a number of significant past studies in unconstrained optimization. The functions [28] are known widely as the Sphere, Griewangk, Ackley, Zakharov, Rotated Hyper-Ellipsoid and Levy function, and they can be scaled to any number of variables (dimensions). Table 1 shows the domain, the objective function, and the global minimum for every benchmark problem. Additionally, the six benchmark problems have the same global minimum value $f\left(x^*\right)$, and the global minimum values of these six problems are equal to zero when the variables are equal to zero or one, regardless of the number of variables. The Zakharov, Sphere, and Rotated Hyper-Ellipsoid functions are continuous, convex, unimodal, and multidimensional. The first one is plate-shaped and the latter two are Bowl-Shaped in their two-dimensional forms. The others are multimodal, multidimensional, with a large number of local optima.

Table 1

The benchmark problems for testing EGFA, GFA, GA and PSO

Name	Variable ranges	Objective function	Optima
Sphere	$x_i \in [-50,50]$	$f(x)=\sum_{i=1}^{d} x_i^2$	$x^*=(0,\cdots,0)$ $f\left(x^*\right)=0$

Griewangk	$x_i \in [-5,5]$	$f(x)=\sum_{i=1}^{d}\frac{x_i^2}{4000}-\prod_{i=1}^{d}\cos\left(\frac{x_i}{\sqrt{i}}\right)+1$	$x^*=(0,\cdots,0)$ $f(x^*)=0$
Ackley	$x_i \in [-50,50]$	$f(x)=20+e-20e^{\left(-0.2\sqrt{\frac{1}{d}\sum_{i=1}^{d}x_i^2}\right)}-e^{\left(\frac{1}{d}\sum_{i=1}^{d}\cos(2\pi x_i)\right)}$	$x^*=(0,\cdots,0)$ $f(x^*)=0$
Zakharov	$x_i \in [-10,10]$	$f(x)=\sum_{i=1}^{d}x_i^2+\left(\sum_{i=1}^{d}0.5ix_i\right)^2+\left(\sum_{i=1}^{d}0.5ix_i\right)^4$	$x^*=(0,\cdots,0)$ $f(x^*)=0$
Rotated Hyper-Ellipsoid	$x_i \in [-50,50]$	$f(x)=\sum_{i=1}^{d}\sum_{j=1}^{i}x_j^2$	$x^*=(0,\cdots,0)$ $f(x^*)=0$
Levy	$x_i \in [-10,10]$	$f(x)=\sin^2(\pi w_1)+$ $\sum_{i=1}^{d-1}(w_i-1)^2\left[1+10\sin^2(\pi w_i+1)\right]+$ $(w_d-1)^2\left[1+\sin^2(2\pi w_d)\right]$ $where, w_i=1+\frac{x_i-1}{4}$	$x^*=(1,\cdots,1)$ $f(x^*)=0$

To investigate the accuracy and efficiency of EGFA, we choose four common performance metrics for evaluating the performance of EGFA in comparison with the original GFA and two other classical computational intelligence algorithms, i.e., GA and PSO. Four performance metrics are described as follows:

1) The mean value of solutions that are found by the four algorithms for 100 different trials.

2) The median value of solutions that are found by the four algorithms for 100 different trials.

3) The Mean squared error (MSE) [29]: it has the general definition as in Eq. (12), where *n* is the number of tests, $f(x_i)$ is the solution of *ith* run and $f_{opt}(x)$ is the true global minimum. Obviously, the lower the Mean squared error (MSE) is, the better performance the algorithm has.

$$MSE=\frac{1}{n}\sum_{i=1}^{n}\left(f(x_i)-f_{opt}(x)\right)^2, \tag{12}$$

4) The Standard Deviation (STD) [30]: it has the general definition as in Eq. (13), where n is the number of tests, $f(x_i)$ is the solution of *ith* run and $\overline{f(x)}$ is the mean of all $f(x_i)$. Same as the Mean squared error (MSE), the lower the Standard Deviation (STD) is, the better performance the algorithm has.

$$STD=\sqrt{\frac{1}{n-1}\sum_{i=1}^{n}\left(f(x_i)-\left(\overline{f(x)}\right)\right)^2}, \tag{13}$$

The mean value, the median value and the Mean squared error (MSE) are devoted to evaluate the

accuracy of solutions that are obtained by four different algorithms. The Standard Deviation (STD) is employed to measure the stability of the performance for the four test algorithms.

3.1.2 Parameter settings

The parameters and their values for the four algorithms are given in Table 2. The size of population is set as $100*d$, where d is the number of dimensions of a given search space. Moreover, $d = 2,3,5,10,20$ has been tested in this research. The maximum number of iterations is set $200*d$ for the four algorithms to control their running time in the same dimension. The number of groups is just set as 3 for EGFA and the original GFA to ensure multi-centers in this research. The parameter TolFun is the average change in value of the fitness function or mass function and it is changed from the default value of 1.0e-6 to 1.0e-30 to ensure that the optimal solution they acquire is accurate enough.

Table 2

The parameter settings for EGFA, GFA, GA and PSO

Parameters setting	**EGFA**	**GFA**	**GA**	**PSO**
Population size	100*d	100*d	100*d	100*d
The Max number iterations	200*d	200*d	200*d	200*d
The number of group	3	3	-	-
TolFun	1.0e-30	1.0e-30	1.0e-30	1.0e-30

In this research, every test problem shown in Table 1 is scaled to different number of variables, whose value range of each dimension is also set as in Table 1. The basic parameters of each algorithm are set as in Table 2. Four performance metrics, the mean value, the median value, MSE, and STD, are employed to measure the accuracy, efficiency of each algorithm in given running time. In this study, we do not make any systematic attempt to find the best parameters for EGFA and just focus on the accuracy of each algorithm in given running time. The higher accuracy of the optimal solution is, the better the performance the algorithm has.

3.1.3 Parametric statistical test

To compare the performance of EGFA, GFA, GA and PSO on unconstrained optimization problems with different numbers of dimensions in given running time, six test problems with $2,3,5,10,20$ dimensions

are used. The comparisons of four algorithms: EGFA, GFA, GA and PSO on the six test problems in mean value, median value, MSE and STD are shown in Figs. 5~8. More specifically, the results of Figs. 5~8 are the mean value, median value, MSE and the STD of 100 trials, which the size of the population is set as $100*d$, where d is the number of dimensions and $d = 2,3,5,10,20$, the iterations for EGFA, GFA, GA and PSO is set as $200*d$.

Fig. 5 shows that EGFA achieves better performance in terms of mean value than GFA, GA and PSO; Fig. 6 shows that EGFA achieves better performance in terms of median value than GFA, GA and PSO. Especially in search space of low dimensions, we can see that EGFA outperforms GFA, GA and PSO in terms of accuracy. Fig. 7 shows that the solution obtained by EGFA has less value of the MSE than GFA, GA and PSO, therefore we can see that EGFA has less error than the other three algorithms. Fig. 8 demonstrates that EGFA has more stable performance compared with GFA, GA and PSO since EGFA has the least value of STD among the four algorithms. In addition, the accuracy of solutions obtained by EGFA will decrease with the increase of the dimensions of search space like PSO, but EGFA has better accuracy than PSO as shown in Figs. 5-8.

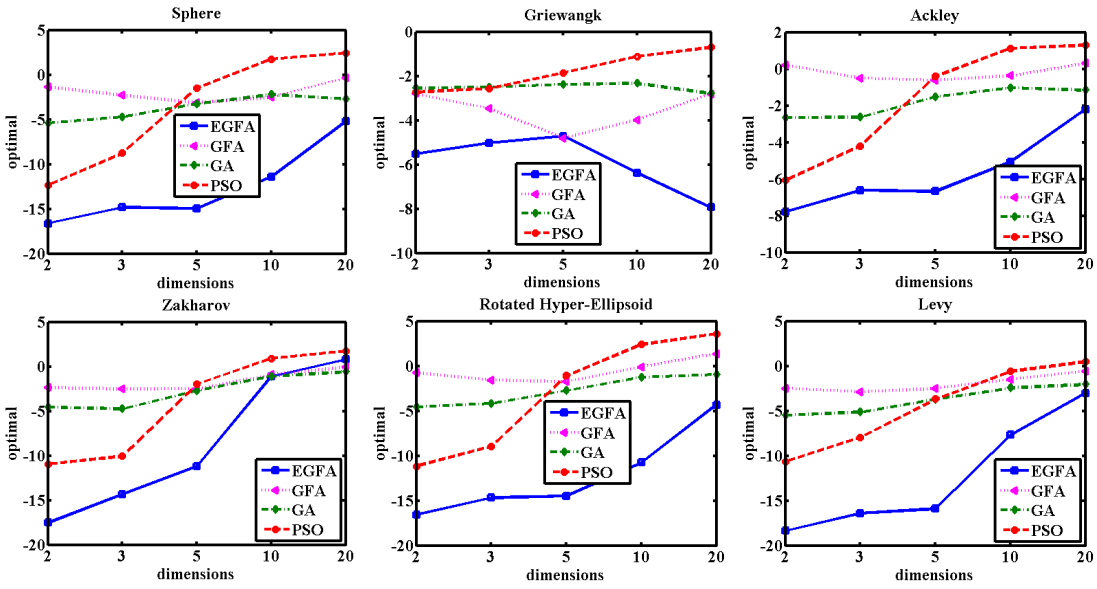

Fig. 5. A Comparison of EGFA, GFA, GA and PSO in $\log_{10}$(mean) with 2,3,5,10,20 dimensions

Besides the accuracy, the running time is another very important factor that we should consider in order to measure the efficiency of an algorithm. It is obvious that the introduction of DS and Explosion operations are at cost of time, but the optimal space the DS acquires can help to decrease the number of

iterations and shorten the overall running time. The average running time of the four algorithms for 100 trials on the six test problems with the number of dimensions $d = 2,3,5,10,20$ is demonstrated in Table 3 and Fig. 9. Table 3 shows that the performance of the four algorithms is controlled in similar running time, when they are executed on the same problem in same dimension. Fig. 9 shows that EGFA has the best efficiency for solving the six test problems in some content. And Fig. 9 also shows that the running time of the four algorithms on the test problems increases exponentially with the increase of the dimensions. All experiments were implemented on a PC (i5-4200M, 8GB, Windows 7, Matlab R2014a).

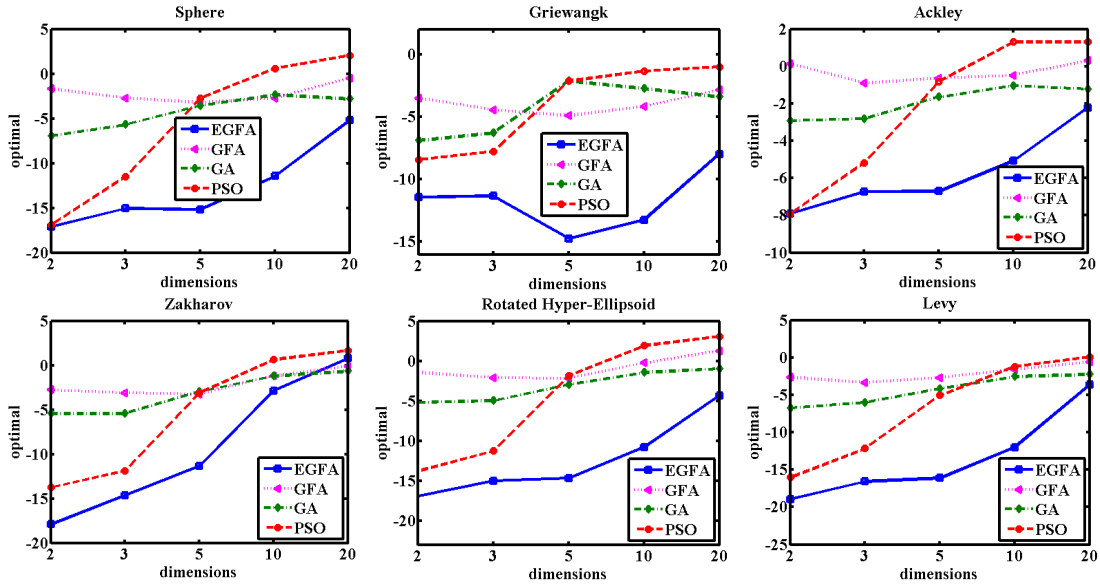

Fig. 6. A Comparison of EGFA, GFA, GA and PSO in $\log_{10}$(median) with 2,3,5,10,20 dimensions

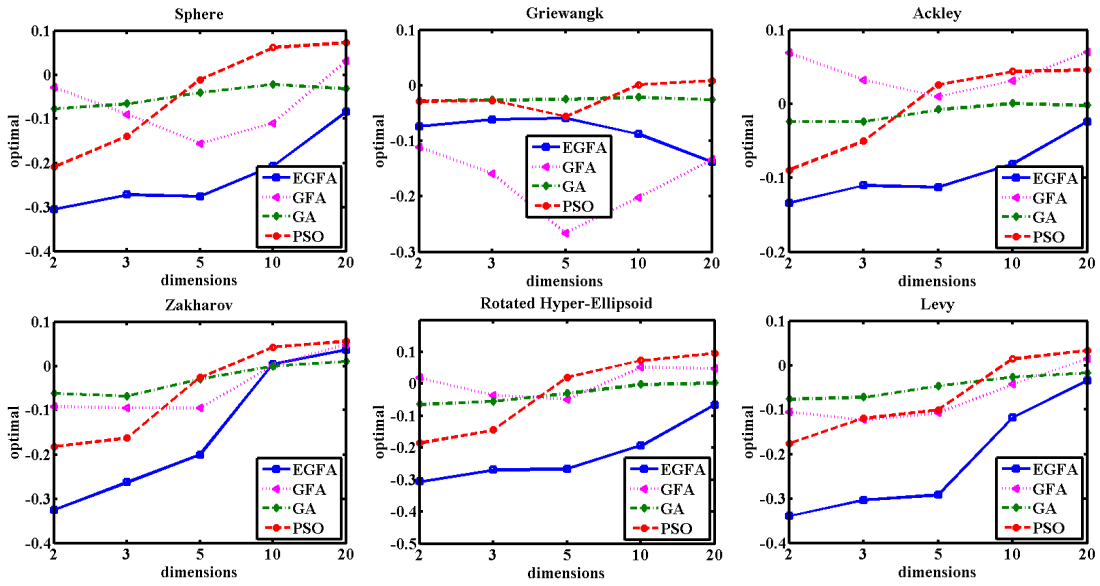

Fig. 7. A Comparison of EGFA, GFA, GA and PSO in log10(MSE) with 2,3,5,10,20 dimensions

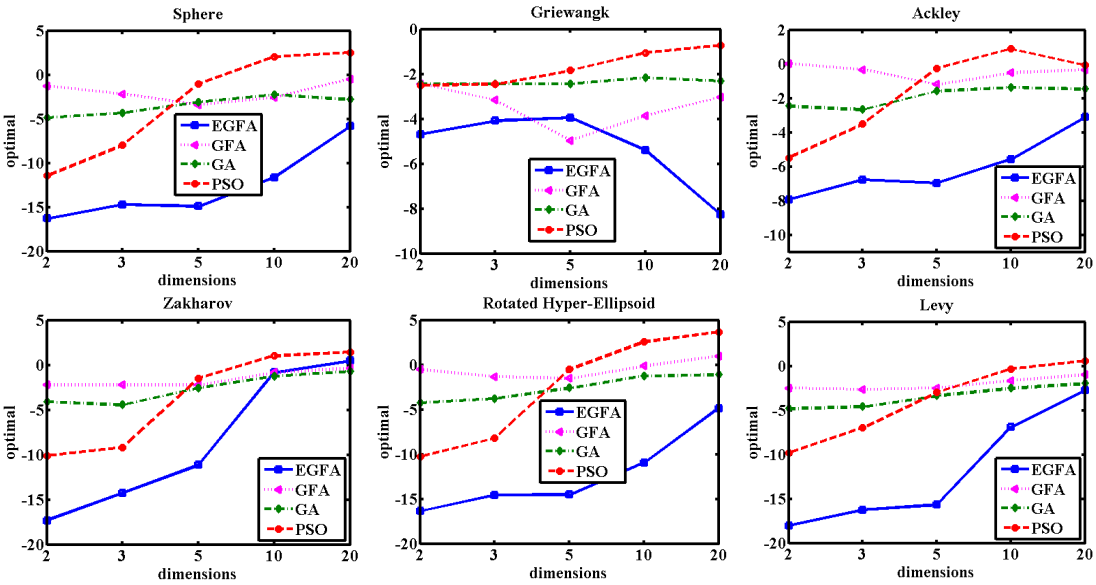

Fig. 8. A Comparison of EGFA, GFA, GA and PSO in $\log_{10}$(STD) value with 2,3,5,10,20 dimensions

Table 3

The average running time for 100 trials of six benchmark functions with dimension d=2,3,5,10,20

Benchmark Function		2	3	5	10	20
Sphere	EGFA	0.4381s	0.8898s	1.4093s	3.7177s	19.0255s
	GFA	0.5013s	0.9028s	1.4056s	3.7115s	19.0313s
	GA	0.5218s	0.9164s	1.4218s	3.7560s	19.0874s
	PSO	0.4315s	0.8820s	1.4350s	3.7997s	19.4872s
Griewangk	EGFA	0.4074s	0.9803s	1.4993s	3.8174s	17.0395s
	GFA	0.5018s	1.0027s	1.5081s	4.0279s	17.0497s
	GA	0.5225s	1.0276s	1.5361s	3.8566s	17.0977s
	PSO	0.4098s	0.9739s	1.5305s	3.9028s	17.4997s
Ackley	EGFA	0.4839s	0.9627s	1.5041s	4.0129s	16.0337s
	GFA	0.5015s	1.0029s	1.5077s	4.0257s	16.0513s
	GA	0.5186s	1.0221s	1.5372s	4.0426s	16.0824s
	PSO	0.4897s	0.9575s	1.5327s	4.1212s	16.5579s
Zakharov	EGFA	0.4158s	0.9771s	1.5019s	3.9152s	18.0285s
	GFA	0.5013s	1.0014s	1.5345s	4.0141s	19.0296s
	GA	0.4544s	1.0304s	1.5418s	3.9605s	18.1944s
	PSO	0.4081s	0.9750s	1.5384s	3.9908s	18.4605s
Rotated Hyper-Ellipsoid	EGFA	0.4074s	0.9125s	1.3598s	3.7141s	18.0331s
	GFA	0.4021s	0.9031s	1.4052s	3.8168s	18.0481s
	GA	0.4169s	0.9119s	1.4285s	3.7592s	18.1806s
	PSO	0.4080s	0.9125s	1.4314s	3.7941s	18.4442s
Levy	EGFA	0.4558s	0.9560s	1.8099s	4.2267s	19.0550s
	GFA	0.5028s	1.0036s	1.8073s	4.5340s	20.0777s

GA	0.5303s	1.0303s	1.8448s	4.2536s	19.2646s
PSO	0.5090s	1.0156s	1.8381s	4.4143s	19.5955s

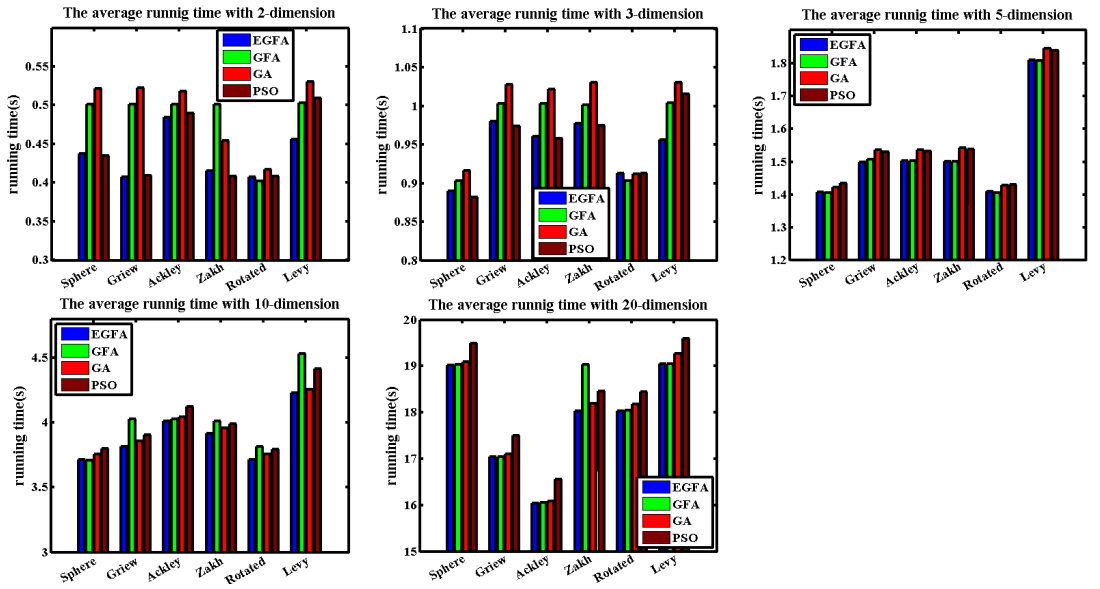

Fig. 9 The average running time for 100 trials of six benchmark functions with dimension d = 2, 3, 5, 10, 20

3.1.4 Non-parametric statistical test

To further verify the conclusion drawn from the part of parametric statistical test in Section 3.2.1, following [31] we use a non-parametric statistical method, called the Wilcoxon test [32] [33], to compare the performance of EGFA, GFA, GA and PSO. Table 4 shows the Wilcoxon test results of the 100 trials for the four algorithms on the six test problems, and readers can refer to [34] for more details about how to use the Wilcoxon test to compare different metaheuristic algorithms in detail.

In the Wilcoxon test presented in Table 4, we set the significance level p to be 0.05 ($p = 0.05$) and use the two-tailed hypothesis because the settings of the both are the most commonly. We have 100 trials (which means the sample size $N = 100$) for the four algorithms on the six test problems in $2, 3, 5, 10, 20$ dimensions, and calculate both p-value and h-value to compare the performance of each pair of the algorithms. If the p-value is less than the significance level ($p = 0.05$) and $h = 1$, the results indicate that there is a significant difference between the performance of the two algorithms. Otherwise, it indicates that there is not enough evidence to verify the significant difference between the performance of the two algorithms

Since we know that the performance of the two algorithms is significantly different, to further determine which algorithm performs better we will focus on the value of the rank sum for the former

algorithm $\sum R_1$ and the latter algorithm $\sum R_2$, that is, if $\sum R_1$ is less than $\sum R_2$, it indicates that the former algorithm outperforms the latter one, otherwise, the latter one outperforms the former one.

For each cell in Table 4, the first value is the *p*-value and is representative of the probability that the results for the two algorithms obey the same distribution. The second value is the *h*-value. If the *h*-value is equal to 1 ($h = 1$), it indicates that the performance of the two algorithms is significantly different. If there is no significant difference between the two algorithms, the *h*-value is equal to 0 ($h = 0$). A '-' sign means that the former one outperforms the latter one ($\sum R_1 < \sum R_2$). Similarly, a '+' sign means $\sum R_1 > \sum R_2$ and indicates that the latter algorithm outperforms the former one. For instance, from the three rows about the Rotated Hyper-Ellipsoid problem in Table 4, all the *p*-values are smaller than the significance level ($p = 0.05$), *h*-value is 1, and the '-' sign demonstrates that EGFA outperforms GFA, GA and PSO at $p = 0.05$ on the Rotated Hyper-Ellipsoid problem in $2, 3, 5, 10, 20$ dimensions because of $\sum R_1 < \sum R_2$. The results are consistent with the distribution of the results obtained by the four algorithms on 100 trials presented in Fig. 10. Fig. 11 presents the distribution of the results for the four algorithms on Rotated Hyper-Ellipsoid in 20-dimensional search space in detail, which corresponds to the cell '1.1720e-16/1/-', '2.5621e-34/1/-' and '2.1609e-23/1/-' in the Table 4.

From the results presented in Table 4, as well as in Fig. 10 and Fig. 11, we can see that results obtained from the Wilcoxon test confirm the conclusions drawn from Section 3.2.1, that is, EGFA outperforms GFA, GA and PSO on all six test problems in the overall level.

Table 4

Results of Wilcoxon rank sum test for statistically significance level at $p = 0.05$ for optimal solution over 100 runs on benchmark functions for 2,3,5,10,20 dimensions

Benchmark Function		2	3	5	10	20
		p/h/zval				
Sphere	EGFA vs GFA	3.0199e-11/1/-	3.0199e-11/1/-	3.0199e-11/1/-	3.0199e-11/1/-	3.0199e-11/1/-
	EGFA vs GA	2.5621e-34/1/-	2.5621e-34 /1/-	2.5621e-34 /1/-	2.5621e-34 /1/-	2.5621e-34/1/-
	EGFA vs PSO	0.2241 /0/-	4.9856e-29 /1/-	2.5621e-34 /1/-	2.5621e-34 /1/-	2.5621e-34 /1/-
Ackley	EGFA vs GFA	3.0199e-11/1/-	3.0199e-11/1/-	3.0199e-11/1/-	3.0199e-11/1/-	3.0199e-11/1/-
	EGFA vs GA	2.5621e-34/1/-	2.5621e-34/1/-	2.5621e-34/1/-	2.5621e-34/1/-	2.5621e-34/1/-
	EGFA vs PSO	0.7022/0/-	2.2224e-27/1/-	2.5621e-34/1/-	2.5621e-34/1/-	2.5621e-34 /1/-
Griewangk	EGFA vs GFA	3.0199e-11/1/-	3.1589e-10/1/-	1.0601e-07/1/-	3.1589e-10/1/-	3.0199e-11/1/-

	EGFA vs GA	3.7064e-23/1/-	3.2066e-29/1/-	1.3673e-31/1/-	9.6788e-31/1/-	2.5621e-34/1/-
	EGFA vs PSO	1.1068e-07/1/-	4.4804e-09/1/-	2.5074e-33/1/-	6.6802e-34/1/-	2.5621e-34/1/-
Levy	EGFA vs GFA	3.0199e-11/1/-	3.0199e-11/1/-	3.0199e-11/1/-	2.9543e-11/1/-	3.0199e-11/1/-
	EGFA vs GA	2.5621e-34/1/-	2.5621e-34/1/-	2.5621e-34/1/-	2.5306e-34/1/-	1.4447e-22/1/-
	EGFA vs PSO	8.2778e-22/1/-	3.6728e-34/1/-	2.5616e-34/1/-	2.5306e-34/1/-	2.5621e-34/1/-
Rotated Hyper-	EGFA vs GFA	3.0199e-11/1/-	3.0199e-11/1/-	3.0199e-11/1/-	3.0199e-11/1/-	1.1720e-16/1/-
Ellipsoid	EGFA vs GA	2.5621e-34/1/-	2.5621e-34/1/-	2.5621e-34/1/-	2.5621e-34/1/-	2.5621e-34/1/-
	EGFA vs PSO	2.1609e-23/1/-	5.7156e-30/1/-	2.5621e-34/1/-	2.5621e-34/1/-	2.5621e-34/1/-
Zakharov	EGFA vs GFA	3.0199e-11/1/-	3.0199e-11/1/-	8.1527e-11/1/-	9.2113e-05/1/-	6.1210e-10/1/+
	EGFA vs GA	2.5621e-34/1/-	2.5621e-34/1/-	3.6728e-34/1/-	5.3312e-12/1/-	3.0679e-34/1/+
	EGFA vs PSO	6.9521e-30/1/-	1.3281e-21/1/-	5.4201e-34/1/-	5.4201e-34/1/-	2.5621e-34/1/-

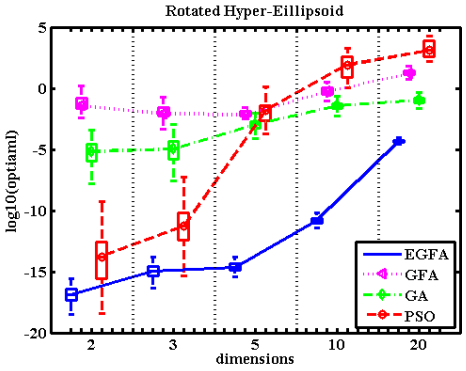

Fig. 10 The distribution of the results for EGFA, GFA, GA and PSO on Rotated Hyper-Ellipsoid in 2,3,5,10,20 dimensions

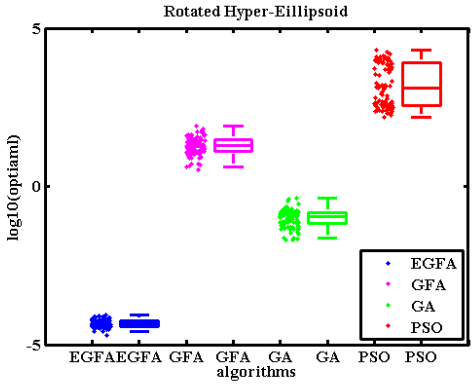

Fig. 11 The distribution of the results for EGFA, GFA, GA and PSO on Rotated Hyper-Ellipsoid in 20 dimensions

3.2 Experiments on real datasets

Clustering is an important data mining task and it has been explored extensively in different application areas. To assess the excellent performance of the model in real world applications, the GFA and EGFA

are applied to clustering problems in this research so that we can explore the application potential of GFA and EGFA on real world problems.

3.2.1 The Encoding of cluster centroid vector

In this research, we applied GFA and EGFA to clustering, based on the original idea of k-means. The process of clustering can be regarded as the unconstrained optimization problems since its objective function is constrained. In the context of clustering, a single dust represents the K cluster centroid vectors. That is, each dust $\vec{X}_i$ is constructed in the form of Eq. (14),

$$\vec{X}_i = \left(x_{i,1}, x_{i,2} \cdots x_{i,j} \cdots x_{i,K}\right) \ , \tag{14}$$

where $x_{i,j}$ refers to the j-th cluster centroid vector of the i-th dust. Therefore, the dust population represents candidate cluster centroid vectors.

The mass function of individual is measured as Eq. (15) [35], which is similar to the objective in k-means clustering algorithm.

$$\max massFunction = -\sum_{i=1}^{K} \sum_{s \in C_i} \left\| s - \vec{m}_i \right\|_2^2 \ , \tag{15}$$

where $\vec{m}_i$ is the i-th cluster centroid vector, s is the i-th sample that belongs to the cluster C_i, and K is the number of clusters.

The main loops of GFA and EGFA in data clustering are the same as the description in Section 2.1 and Section 2.2.3.

3.2.2 Datasets and performance evaluation

The datasets listed in Table 5 are the well-known real datasets from the UCI Machine Learning Repository [36], which are usually used to test the performance of clustering algorithms. These three real datasets are chosen from a series of past research in clustering. Table 5 shows the number of instances, the number of attributes, the number of clusters， and the distribution of the three datasets. Specifically, The Iris dataset consists of 150 instances with 4 attributes. The Seeds dataset consists of 210 instances with 7 attributes. The Wine dataset consists of 178 instances with13 attributes. There are 3 clusters of the three real datasets. This paper applies GFA and EGFA in clustering, and compared the results of them with k-means.

Table 5.

The real datasets for testing GFA, EGFA and k-means

Datasets	Number of instances	Number of attributes	Number of clusters	distribution
Iris	150	4	3	50/50/50
Seeds	210	7	3	70/70/70
Wine	178	13	3	59/71/48

In order to compare the performance of GFA, EGFA and k-means, The Adjusted Rand index (ARI) is adopted to assess clustering results of the three algorithms. The adjusted Rand Index is usually used to measures the agreement between two partitions. It is the corrected-for-chance version of Rand index (RI), which is simply defined as $RI = \frac{a+b}{a+b+c+d}$ (The range of the Rand index is between 0 and 1. When the two partition agree perfectly, the Rand index is 1. More details about Rand index are presented in [37] [38]). The original Adjusted Rand index is defined as Eq. (16).

$$ARI = \frac{Index - ExpectedIndex}{MaxIndex - ExpectedIndex} = \frac{\sum_i \binom{n_{ij}}{2} - \left[\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}\right] / \binom{n}{2}}{\frac{1}{2}\left[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}\right] - \left[\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}\right] / \binom{n}{2}}, \quad (16)$$

the bigger value the ARI is, the more agreement of the two partitions have. We adopt the Adjusted Rand index as the measure of the experimental results, and the Readers can refer [37] [38] for more details about how Adjusted Rand index assesses the agreement between two partitions.

3.2.3 Parameter settings

The settings of parameters for GFA, EGFA and k-means are showed in Table 6. The size of population is set as 20 for GFA and EGFA. The maximum number of iterations is set 100 for all three algorithms. There is a trick to setting the number of clusters in this research. Firstly, EGFA runs several times in different number of clusters. Then the value of ARI of those results is calculated. Lastly, the number of clusters is decided according the value of ARI, the one with largest value of ARI is desirable. Fig. 12 shows the average value of ARI for 30 trials on seeds dataset in different number of clusters. As Fig.12 shows the ARI is largest when the number of clusters is 3. In this way, the number of clusters is set 3 for

datasets: Iris, Seeds and Wine finally.

Table 6.

The parameters setting of GFA, EGFA and k-means

Parameters setting	Population Size	Max Number of iterations	Number of clusters
GFA	20	100	3
EGFA	20	100	3
k-means	-	100	3

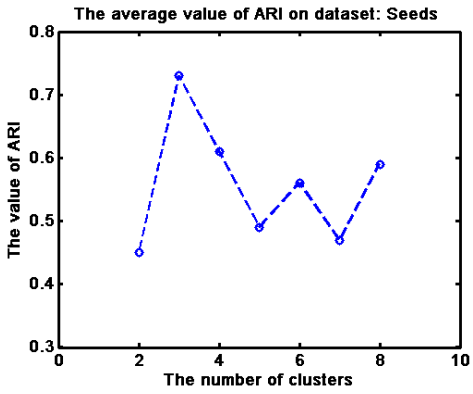

Fig. 12 The average value of ARI on Seeds dataset in different number of clusters

3.2.4 Experimental results and analysis

The comparisons of three algorithms: GFA, EGFA and k-means for 30 trials on three real datasets in average value of Adjusted Rand index (ARI) are shown in Table 7. Table 7 shows that the EGFA outperforms GFA and k-means on the three real datasets: Iris, Seeds, and Wine. Specifically, EGFA has the highest value of ARI among the three algorithms, which indicates that EGFA has the best performance compared with the original GFA and k-means in the three real datasets. In addition, GFA performs better than k-means on the datasets: Wine and Seeds, and k-means performs better than GFA on dataset Iris.

Table 7

The results of GFA, EGFA and k-means in ARI

ARI	GFA	EGFA	k-means
Iris	0.5791	**0.6846**	0.6836
Seeds	0.7027	**0.7302**	0.6998
Wine	0.3676	**0.3715**	0.3499

4. Conclusions and Future work

In this research, a novel EGFA is presented based on the original GFA. A novel accuracy improvement strategy called Dust Sampling (DS) is employed to quickly find the so-called optimal space that contains the optimal solution in search space. Another novel strategy named Explosion Operation is adopted to decrease the probability of dust falling into local optima, and the formulae for iteration are modified. Six benchmark problems and three real datasets previously used from literatures in unconstrained optimization are chosen to evaluate the performance of EGFA. The experimental results demonstrate that the proposed EGFA has achieved excellent performance in terms of efficiency, accuracy, and the capability of solving real world problems. All the results indicate that EGFA is of well convergence and higher search efficiency.

It is noted that the solutions EGFA finds are frequently closer to the actual optimal solutions than the other three algorithms in the lower dimensions on all the six benchmark problems and three real datasets, but at the same time, we also notice the fact that all the four optimization algorithms face challenges when dealing with problems in higher dimensions in terms of in accuracy and running time, especially when the dimension is larger than 20. This motivates us to make more efforts in our future research to investigate how to further improve EGFA along this line. The study on complex unconstrained optimization problems in higher dimensions by EGFA is in progress, and we will also further investigate how to find more effective methods to adjust the parameters according to the characteristics of specific problems.

Acknowledgement

This research was funded by the National Natural Science Foundation of China (Nos.61572227, 61772227, 61702214), the Development Project of Jilin Province of China (Nos 20170101006JC,

20180414012GH, 20170203002GX, 20190201293JC), Zhuhai Premier-Discipline Enhancement Scheme (Grant 2015YXXK02) and Guangdong Premier Key-Discipline Enhancement Scheme (Grant 2016GDYSZDXK036). This work was also supported by Jilin Provincial Key Laboratory of Big Date Intelligent Computing (No. 20180622002JC).

References

[1] B. Chatterjee, Steepest Descent Method: Springer US, 2013.

[2] F. Ahmad, E. Tohidi, and J. A. Carrasco, "A parameterized multi-step Newton method for solving systems of nonlinear equations," Numerical Algorithms, vol. 71, pp. 1-23, 2016.

[3] J. Zhao, E. A. H. Vollebregt, and C. W. Oosterlee, "A fast nonlinear conjugate gradient based method for 3D concentrated frictional contact problems," Journal of Computational Physics, vol. 288, pp. 86-100, 2015.

[4] R. H. Byrd, S. L. Hansen, J. Nocedal, and Y. Singer, "A Stochastic Quasi-Newton Method for Large-Scale Optimization," Siam Journal on Optimization, vol. 26, 2015.

[5] M. A. Salido, J. Escamilla, A. Giret, and F. Barber, "A genetic algorithm for energy-efficiency in job-shop scheduling," The International Journal of Advanced Manufacturing Technology, vol. 85, pp. 1303-1314, 2016.

[6] E. Elyan and M. M. Gaber, "A Genetic Algorithm Approach to Optimising Random Forests Applied to Class Engineered Data," Information Sciences, 2016.

[7] T. Liao, K. Socha, M. A. M. D. Oca, and T. Stützle, "Ant Colony Optimization for Mixed-Variable Optimization Problems," IEEE Transactions on Evolutionary Computation, vol. 18, pp. 503-518, 2014.

[8] Z. Wang, H. Xing, T. Li, and Y. Yang, "A Modified Ant Colony Optimization Algorithm for Network Coding Resource Minimization," IEEE Transactions on Evolutionary Computation, vol. 20, pp. 1-1, 2015.

[9] J. Kennedy and R. Eberhart, Particle swarm optimization: Springer US, 2011.

[10] X. L. Wen, J. C. Huang, D. H. Sheng, and F. L. Wang, "Conicity and cylindricity error evaluation using particle swarm optimization," Precision Engineering, vol. 34, pp. 338-344, 2010.

[11] J. Kennedy and R. Eberhart, "Particle swarm optimization," IEEE International Conference on

Neural Networks, vol. 4, pp. 1942-1948, 1995.

[12] B. Xing and W. J. Gao, Innovative Computational Intelligence: A Rough Guide to 134 Clever Algorithms: Springer Publishing Company, Incorporated, 2013.

[13] D. Karaboga and B. Basturk, "A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm," Journal of Global Optimization, vol. 39, pp. 459-471, 2007.

[14] C. J. A. B. Filho, F. B. D. L. Neto, A. J. C. C. Lins, A. I. S. Nascimento, and M. P. Lima, Fish School Search: Springer Berlin Heidelberg, 2009.

[15] K. M. Passino, Bacterial Foraging Optimization: IGI Global, 2010.

[16] S. C. Chu and P. W. Tsai, "Computational intelligence based on the behavior of cats," International Journal of Innovative Computing Information & Control Ijicic, vol. 3, pp. 163-173, 2006.

[17] O. K. Erol and I. Eksin, A new optimization method: Big Bang-Big Crunch: Elsevier Science Ltd., 2006.

[18] R. A. Formato, "Central Force Optimization: a New Metaheuristic with Applications in Applied Electromagnetics," vol. 77, pp. 425-491, 2007.

[19] E. Rashedi, H. Nezamabadi-Pour, and S. Saryazdi, "GSA: A Gravitational Search Algorithm," Intelligent Information Management, vol. 4, pp. 390-395, 2012.

[20] A. Y. S. Lam and V. O. K. Li, "Chemical-Reaction-Inspired Metaheuristic for Optimization," IEEE Transactions on Evolutionary Computation, vol. 14, pp. 381-399, 2010.

[21] S. A. Salem, "BOA: A novel optimization algorithm," in International Conference on Engineering and Technology, pp. 1-5, 2012.

[22] M. Zheng, G. Liu, C. Zhou, Y. Liang, and Y. Wang, "Gravitation field algorithm and its application in gene cluster," Algorithms for Molecular Biology, vol. 5, pp. 1-11, 2010.

[23] M. Zheng, Y. Sun, G. Liu, Y. Zhou, and C. Zhou, "Improved Gravitation Field Algorithm and Its Application in Hierarchical Clustering," PloS One, vol. 7, p. e49039, 2012.

[24] M. Zheng, G. X. Liu, Y. Zhou, and C. G. Zhou, "Reconstruction of gene regulatory network based on gravitation field algorithm," Journal of Jilin University, vol. 44, pp. 427-432, 2014.

[25] V. S. Safronov, "Evolution of the protoplanetary cloud and formation of the earthand planets," Trans. NASA TT F-677, 1972.

[26] L. Huang, X. Hu, Y. Wang, F. Zhang, Z. Liu, and W. Pang, "Gravitation field algorithm with optimal

626 detection for unconstrained optimization," in International Conference on Systems and Informatics,
627 2017, pp. 1411-1416.
628 [27] G. P. Zhu and S. Kwong, "Gbest-guided artificial bee colony algorithm for numerical function
629 optimization," Applied Mathematics and Computation, vol. 217, pp. 3166-3173, Dec 1 2010.
630 [28] D. E. Goldberg. "Genetic Algorithms in Search, Optimization and Machine Learning ", Addison-
631 Wesley Pub. Co. pp. 2104–2116, 1989.
632 [29] G. Steenackers and P. Guillaume, "Bias-specified robust design optimization: A generalized mean
633 squared error approach," Computers & Industrial Engineering, vol. 54, pp. 259-268, 2008.
634 [30] A. Majumder, "Application of Standard Deviation Method Integrated PSO Approach in
635 Optimization of Manufacturing Process Parameters," Handbook of Research on Artificial
636 Intelligence Techniques & Algorithms, 2015.
637 [31] W. Pang and G. M. Coghill, "QML-AiNet: An immune network approach to learning qualitative
638 differential equation models," Applied Soft Computing, vol. 27, pp. 148-157, 2015.
639 [32] S. Siegel and N. J. J. Castellan, "Non-Parametric Statistics for Behavioral Sciences," American
640 Catholic Sociological Review, vol. 18, 1957.
641 [33] G. W. Corder and D. I. Foreman, Nonparametric Statistics for Non-Statisticians: A Step-by-Step
642 Approach, 2009.
643 [34] S. García, D. Molina, M. Lozano, and F. Herrera, "A study on the use of non-parametric tests for
644 analyzing the evolutionary algorithms' behavior: a case study on the CEC'2005 Special Session on
645 Real Parameter Optimization," Journal of Heuristics, vol. 15, pp. 617-644, 2009.
646 [35] R. Duwairi and M. Abu-Rahmeh, "A novel approach for initializing the spherical K-means clustering
647 algorithm," Simulation Modelling Practice and Theory, vol. 54, pp. 49-63, May 2015.
648 [36] D. Dua and C. Graff, "UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. " Irvine,
649 CA: University of California, School of Information and Computer Science, 2019.
650 [37] D. Steinley, M. J. Brusco, and L. Hubert, "The Variance of the Adjusted Rand Index," Psychological
651 Methods, vol. 21, pp. 261-272, Jun 2016.
652 [38] R. Brouwer, "Extending the rand, adjusted rand and jaccard indices to fuzzy partitions," Journal of
653 Intelligent Information Systems, vol. 32, pp. 213-235, Jun 2009.