sensors

Article

An Adaptation Multi-Group Quasi-Affine Transformation Evolutionary Algorithm for Global Optimization and Its Application in Node Localization in Wireless Sensor Networks

Nengxian Liu [1], Jeng-Shyang Pan [1,2,3,*], Jin Wang [2,4] and Trong-The Nguyen [2,5]

1 College of Mathematics and Computer Science, Fuzhou University, Fuzhou 350116, China; lylnx@fzu.edu.cn
2 Fujian Provincial Key Lab of Big Data Mining and Applications, Fujian University of Technology, Fuzhou 350118, China; jinwang@csust.edu.cn (J.W.); vnthe@hpu.edu.vn (T.-T.N.)
3 College of Computer Science and Engineering, Shandong University of Science and Technology, Qingdao 266590, China
4 Hunan Provincial Key Laboratory of Intelligent Processing of Big Data on Transportation, School of Computer & Communication Engineering, Changsha University of Science & Technology, Changsha 410000, China
5 Department of Information Technology, University of Manage and Technology, Haiphong 180000, Vietnam
* Correspondence: jspan@cc.kuas.edu.tw

Received: 7 September 2019; Accepted: 21 September 2019; Published: 23 September 2019

Abstract: Developing metaheuristic algorithms has been paid more recent attention from researchers and scholars to address the optimization problems in many fields of studies. This paper proposes a novel adaptation of the multi-group quasi-affine transformation evolutionary algorithm for global optimization. Enhanced population diversity for adaptation multi-group quasi-affine transformation evolutionary algorithm is implemented by randomly dividing its population into three groups. Each group adopts a mutation strategy differently for improving the efficiency of the algorithm. The scale factor F of mutations is updated adaptively during the search process with the different policies along with proper parameter to make a better trade-off between exploration and exploitation capability. In the experimental section, the CEC2013 test suite and the node localization in wireless sensor networks were used to verify the performance of the proposed algorithm. The experimental results are compared results with three quasi-affine transformation evolutionary algorithm variants, two different evolution variants, and two particle swarm optimization variants show that the proposed adaptation multi-group quasi-affine transformation evolutionary algorithm outperforms the competition algorithms. Moreover, analyzed results of the applied adaptation multi-group quasi-affine transformation evolutionary for node localization in wireless sensor networks showed that the proposed method produces higher localization accuracy than the other competing algorithms.

Keywords: differential evolution; multi-group; quasi-affine transformation evolutionary algorithm; global optimization; distance vector-hop; node localization; wireless sensor networks

1. Introduction

Over the last few decades, global optimization problems have attracted a lot of research interest [1]. Many optimization algorithms have been developed based on inspiration from natural phenomenon, e.g., biological, swarm, physical aspects that are known as natural-inspired intelligent algorithms [2]. The natural-inspired smart algorithms have been widely applied to solve optimization problems successfully [3–5]. Genetic algorithm (GA) [6], particle swarm optimization (PSO) [7], differential evolution (DE) [8], ant colony optimization (ACO) [9], Hierarchical archive based mutation strategy with depth information of evolution (HARD-DE) [10] artificial bee colony optimization (ABC) [11], firefly algorithm (FA) [12], bat algorithm (BA) [13], fireworks algorithm (FWA) [14] and quasi-affine transformation evolution algorithm (QUATRE) [15], and cat swarm optimization (CSO) [16] are examples of such algorithms.

The natural-inspired algorithms have been proving robust in delivering optimal global solutions and assisting in resolving the limitations encountered in traditional methods [17]. The optimization process of the natural-inspired intelligent algorithms usually begins with generating a set of randomly initialized agents that combined, immigrated, or evolved over a predefined number of generations. Theses algorithms can be an efficient way to produce acceptable solutions by trial and error to a complex problem in a reasonably reasonable time [18]. There are two algorithms of the algorithms as mentioned above, PSO and DE have been paid much attention and widely used in diverse fields of science and engineering because of their simple and powerful.

The QUATRE [15] algorithm elaborated its relationship with PSO and DE algorithms. The QUATRE considered as the parameter-reduced or enhanced DE algorithm that conquers representational or positional bias of the DE algorithm. Several variants of the QUATRE algorithm were represented by the conventional notation "QUATRE/x/y" that is similar to the notation of DE "DE/x/y/z" [15,19].

The QUATRE algorithm is considered as a robust algorithm [5,20–22] with the advantages, e.g., simplicity, a few setting control parameters, easy implementation, and outputting excellent performance. However, similar to other stochastic optimization algorithms, QUATRE algorithm also exists some drawbacks when dealing with complicated problems such as the premature convergence and search stagnation. Some other QUATRE variants have been proposed to alleviate these weaknesses and to enhance the QUATRE's performance. C-QUATRE aimed to improve the QUATRE's performance by partitioning the entire population randomly into two groups that evolve the individual who loses in pairwise competition between two groups [20]. For S-QUATRE [21], it partitions the entire population into two groups, i.e., the better group and the worse group using sort strategy, and its individuals only changed in the worse group. Both of these variants used bi-group approaches and employed one mutation strategy to enhance the QUATRE's performance. Several algorithms introduced the enhanced population diversity based on dividing the entire population into some groups or subpopulations to improve their performance, such as MPADE [23], PPSO [24], and PCSO [25]. In QUATRE variants, the mutation schemes and control parameters (e.g., scale factor) play essential roles in the optimization performance of solving problems due to it would offer unusual exploration and exploitation abilities that caused to methods performance [15]. The searching ability and convergence speed are affected by a scaling factor. The experimental results [19,20] also recommended that setting the scaling factor to a specific constant number could produce functional optimization problems. However, the solution to different practical engineering problems, manual tuning of the control parameters with the appropriate value would be complicated and difficult implementing configuration. To avoid manual tuning of the control parameters, different categorizations of parameter adaptation techniques have been presented for other evolutionary algorithms in [26,27].

In this paper, motivated by multi-population approach and parameter adaptation technique, we propose a novel adaptation multi-group QUATRE algorithm (AMG-QUATRE) with an adaptation scale factor and adopting each group with different mutation scheme to overcome the mentioned weakness of the QUATRE. We evaluate the proposed algorithm by testing CEC2013 benchmark set, and a practical problem of node localization in wireless sensor networks (WSN). The simulation results show that the proposed AMG-QUATRE outperforms the competition algorithms, and improves localization accuracy of the distance vector-hop (DV-Hop) algorithm. Contributions behinds in this paper are as follows.

- A proposed novel AMG-QUATRE algorithm overcomes the deficiencies of the original QUATRE.
- Full use of different mutation strategies along with proper parameters makes a better trade-off between exploration and exploitation capability.

- Compared results with three QUATRE variants, two DE variants, and two PSO variants on testing CEC2013 benchmark is to evaluate confirming the performance of the proposed algorithm.
- An applied AMG-QUATRE to the node localization in WSN by modifying the average hop distance improving the DV-Hop algorithm and implementing the proposed algorithm to obtain the position of the nodes in WSN.

The remainder of the paper is organized as follows. Section 2 reviews the original QUATRE algorithm and the localization in WSN problem. Section 3 presents the proposed AMG-QUATRE algorithm and its application to node localization in WSN. Section 4 analyses the experimental results of the proposed algorithm for CEC2013 benchmark set with 28 benchmark functions and simulation results for node localization in WSN. Finally, the conclusion is given in Section 5.

2. Related Works

2.1. Original QUATRE Algorithm

QUATRE [15] is a swarm-based algorithm for solving single-objective optimization problems over continuous space. QUATRE is a combination of the acronym quasi-affine transformation evolution. Individuals in this algorithm evolve with a quasi-affine transformation form, which is extended from the affine transformation in geometry from one affine space to another. The exact evolution Equation for QUATRE is shown as follows.

$$\mathbf{X} \leftarrow \mathbf{M} \otimes \mathbf{X} + \overline{\mathbf{M}} \otimes \mathbf{B} \tag{1}$$

$\mathbf{X}$ represents the target population matrix consisting of ps target individuals, $\mathbf{X} = [\mathrm{X}_{1,G}, \mathrm{X}_{2,G}, \dots, \mathrm{X}_{\mathrm{ps},G}]^T$. $\mathrm{X}_{i,G} = [\mathrm{x}_{i1}, \mathrm{x}_{i2}, \dots, \mathrm{x}_{iD}]$, $i \in \{1, 2, \dots, \mathrm{ps}\}$, is the ith row vector of the $\mathbf{X}$, which represents the position of ith individual at the generation Gth, and it is a candidate solution to the optimization problem, and D represents the dimension of problem. $\mathbf{B}$ denotes the mutation/donor matrix, $\mathbf{B} = [\mathrm{B}_{1,G}, \mathrm{B}_{2,G}, \dots, \mathrm{B}_{\mathrm{ps},G}]^T$. $\otimes$ represents component-wise multiplication of the corresponding elements in the two matrices. $\mathbf{M}$ is an evolution matrix, and $\overline{\mathbf{M}}$ represents a binary operation of inverting all the elements in $\mathbf{M}$. All elements of these two matrices are binary values, either 1 or 0. The inverse values of one elements are zeros, and inverse values of zero elements are ones. Equation (1) implements an alternative method of crossover operation in the DE algorithm.

$\mathbf{M}$ is an automatically generated matrix which is transformed from an initial matrix $\mathbf{M}_{init}$. For simplicity, $\mathbf{M}_{init}$ is initialized according to a lower triangular matrix whose elements equaling to ones. Equation (2) shows an example of the initialization and transformation when the dimension number D equals to population size ps. There are two consecutive steps incorporated in transforming from $\mathbf{M}_{init}$ to $\mathbf{M}$, indicated by " $\sim$ " operator in Equation (2). First, the elements in each D-dimension row vector of $\mathbf{M}_{init}$ are randomly permuted. Second, the sequence of all the row vectors is randomly permuted with each row vector unchanged. When the population size ps is larger than the vector dimension, the initial matrix $\mathbf{M}_{init}$ requires to be extended according to ps. Equation (3) gives an example of extension with $ps = 2D + 2$. More generally, when $ps\%D = k$, the top k rows of the $D \times D$ lower triangular matrix are included in $\mathbf{M}_{init}$, and $\mathbf{M}$ adaptively changed in accordance with the change of $\mathbf{M}_{init}$ [15].

$$\mathbf{M}_{init} = \begin{bmatrix} 1 & & & \\ 1 & 1 & & \\ & & \dots & \\ 1 & 1 & \dots & 1 \end{bmatrix} \sim \begin{bmatrix} 1 & 1 & \dots & 1 \\ & 1 & 1 & \\ & & \dots & \\ & 1 & & \end{bmatrix} = \mathbf{M} \tag{2}$$

$$\mathbf{M}_{init} = \begin{bmatrix} 1 & & & \\ 1 & 1 & & \\ & & \dots & \\ 1 & 1 & \dots & 1 \\ 1 & & & \\ 1 & 1 & & \\ & & \dots & \\ 1 & 1 & \dots & 1 \\ & & \dots & \\ 1 & & & \\ 1 & 1 & & \end{bmatrix} \sim \begin{bmatrix} 1 & & \dots & 1 \\ & 1 & \dots & \\ & \dots & & \\ 1 & 1 & & \\ & 1 & & \\ 1 & \dots & 1 & 1 \\ & & 1 & \\ 1 & 1 & \dots & 1 \\ & & \dots & \\ 1 & \dots & 1 & 1 \\ & & 1 & \end{bmatrix} = \mathbf{M} \tag{3}$$

There are several different mutation schemes [19,28] introduced for calculation **B** in QUATRE algorithm. The seven mutation schemes are listed in Table 1. $\mathbf{X}_{gbest,G} = [X_{gbest,G}, X_{gbest,G}, \dots, X_{gbest,G}]^T$ in Table 1 is the global best matrix at generation Gth with each row vector equaling to the global best individual $X_{gbest,G}$. $\mathbf{X}_{ri,G}$, $i \in \{1, 2, \dots, 5\}$, in Table 1 is a random matrix which is generated by randomly permutating the sequence of row vectors in the target population matrix **X** at generation Gth. The control parameter F, so-called mutation scaling factor, is a positive number for scaling the difference matrix, whose recommended domain is $(0, 1]$ for most optimization problems.

Table 1. The seven schemes of donor matrix **B** calculation.

No.	QUATRE/x/y	Equation
1	QUATRE/rand/1	$\mathbf{B} = \mathbf{X}_{r1,G} + F \cdot (\mathbf{X}_{r2,G} - \mathbf{X}_{r3,G})$
2	QUATRE/best/1	$\mathbf{B} = \mathbf{X}_{gbest,G} + F \cdot (\mathbf{X}_{r1,G} - \mathbf{X}_{r2,G})$
3	QUATRE/target/1	$\mathbf{B} = \mathbf{X} + F \cdot (\mathbf{X}_{r1,G} - \mathbf{X}_{r2,G})$
4	QUATRE/target-to-best/1	$\mathbf{B} = \mathbf{X} + F \cdot (\mathbf{X}_{gbest,G} - \mathbf{X}) + F \cdot (\mathbf{X}_{r1,G} - \mathbf{X}_{r2,G})$
5	QUATRE/rand/2	$\mathbf{B} = \mathbf{X}_{r1,G} + F \cdot (\mathbf{X}_{r2,G} - \mathbf{X}_{r3,G}) + F \cdot (\mathbf{X}_{r4,G} - \mathbf{X}_{r5,G})$
6	QUATRE/best/2	$\mathbf{B} = \mathbf{X}_{gbest,G} + F \cdot (\mathbf{X}_{r1,G} - \mathbf{X}_{r2,G}) + F \cdot (\mathbf{X}_{r3,G} - \mathbf{X}_{r4,G})$
7	QUATRE/target/2	$\mathbf{B} = \mathbf{X} + F \cdot (\mathbf{X}_{r1,G} - \mathbf{X}_{r2,G}) + F \cdot (\mathbf{X}_{r3,G} - \mathbf{X}_{r4,G})$

2.2. Statement of the Location Problem

In some applications of WSN, node locations play essential roles in its successful implementation, e.g., fire monitoring system, battlefield monitoring, animal monitoring, web tracking [29–31]. Practical applications of WSN are challenging in design under constraints such as limited sensor power and low computational cost [32,33]. A suitable localization algorithm appreciates in developing application WSN. DV-Hop node localization has been widely used in many application because of its simplicity, easy implementing. However, the DV-Hop method has its accuracy still low. So, there are many improvements in the original DV-Hop algorithm have been proposed in the literature that attempts reducing its estimation error. The improved DV-Hop algorithms often have features of given anchor nodes, calculate hop distances, and estimate unknown nodes. Some approaches to improving localization algorithm included applying metaheuristics algorithms such as genetic algorithm (GA) for the DV-Hop and area coverage [34,35], PSO improved localization [36], differential evolution (DE) for localization [37], Improved DV-Hop algorithm based on teaching learning-based optimization (TLBO) [38], multi-objective firefly algorithm (MOFA) for node localization [39], shuffled frog leaping algorithm (SFLA) node localization, elephant herding optimization (EHO) for localization [40], elephant herding optimization (EHO) and tree growth algorithm adapted (TGA) for node localization [41]. There are three typical phases in the node localizations as follows.

In the first phase, every anchor node broadcasts a beacon packet $\{id, x_i, y_i, hop_i\}$, including its identifier id, location coordinate (x_i, y_i) and a zero-initialized hop-count value hop_i, to its neighbor nodes. Each node maintains its own hop-count table, which contains ids, location coordinates and minimum hop-count values of all anchor nodes. When any node receives a packet, it first checks if there is a record for the corresponding anchor node in the table. If the record does not exist, the anchor node is recorded in the table of this node. If the record exists and the hop-count value in the packet is less than the value recorded in the table, the value of the record will be updated. Then, the hop-count value is increased by one to form a new packet. This new packet will be sent to its neighbor nodes. Otherwise, if the hop-count value in the packet is greater than or equal to the value recorded in the table, the received packet is ignored. By this process, all nodes in the entire network get minimum hop-count value from every anchor node.

In the second phase, we estimate the distance between anchor node and the unknown node. Firstly, every anchor node estimates the average hop distances (hop size) using Equation (4):

$$Hopsize_i = \frac{\sum_{j=1,j\neq i}^{m} \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}}{\sum_{j=1,j\neq i}^{m} hop_{ij}} \tag{4}$$

where *i* is the id of anchor node *i*, *m* is the number of anchor nodes, hop_{ij} is the minimum hop-count value between anchor nodes *i* and *j*, (x_i, y_i), (x_j, y_j) are the position coordinates of anchor nodes *i* and *j*. Then, each anchor node broadcasts one more packet containing its $Hopsize_i$ in the entire network by using controlled flooding. The unknown node will store the first hop-size, which it receives and transmits the hop-size packet to neighbor nodes. Next, each unknown node calculates the distances to all anchor nodes using Equation (5) which is given as follow.

$$d_{uv} = hopsize_i \times hop_{uv} \tag{5}$$

where $hopsize_i$ is the average hop distances (hop size) obtained by unknown node *u* from nearest anchor node *i*, hop_{uv} is the minimum hop-count value between the unknown node *u* and the anchor node *v*.

In the last phase, we estimate the coordinate of each unknown node by using the multilateration method. Suppose the (x, y) denotes the position coordinate of unknown node U,(x_i, y_i) denotes position coordinate of *i*th anchor node, d_i, $i = 1 \ldots m$, represent the distance between the unknown node *U* and all anchor nodes. Then, we can obtain the following Equation (6).

$$\begin{cases} (x_1 - x)^2 + (y_1 - y)^2 = d_1^2 \\ (x_2 - x)^2 + (y_2 - y)^2 = d_2^2 \\ \quad \vdots \\ (x_m - x)^2 + (y_m - y)^2 = d_2^2 \end{cases} \tag{6}$$

Equation (7) can be obtained by expanding Equation (6) and subtract the first $(m - 1)$ as the equations follows.

$$\begin{cases} x_1^2 - x_m^2 + 2(x_1 - x_m)x + y_1^2 - y_m^2 - 2(y_1 - y_m)y = d_1^2 - d_n^2 \\ x_2^2 - x_m^2 + 2(x_2 - x_m)x + y_2^2 - y_m^2 - 2(y_2 - y_m)y = d_2^2 - d_n^2 \\ \vdots \\ x_{m-2}^2 - x_m^2 + 2(x_{m-2} - x_m)x + y_{m-2}^2 - y_m^2 - 2(y_{m-2} - y_m)y = d_{m-2}^2 - d_n^2 \\ x_{m-1}^2 - x_m^2 + 2(x_{m-1} - x_m)x + y_{m-1}^2 - y_m^2 - 2(y_{m-1} - y_m)y = d_{m-1}^2 - d_n^2 \end{cases} \tag{7}$$

Equation (7) can be written in matrix form of $\mathbf{AX} = \mathbf{B}$, where $\mathbf{A}$, $\mathbf{X}$, and $\mathbf{B}$ are given by the following Equations.

$$\mathbf{X} = \begin{bmatrix} x \\ y \end{bmatrix} \tag{8}$$

$$\mathbf{A} = \begin{bmatrix} 2(x_1 - x_m) & 2(y_1 - y_m) \\ 2(x_2 - x_m) & 2(y_2 - y_m) \\ \vdots & \\ 2(x_2 - x_m) & 2(y_{m-1} - y_m) \end{bmatrix} \tag{9}$$

$$B = \begin{bmatrix} x_1^2 + y_1^2 - x_m^2 - y_m^2 + d_m^2 - d_1^2 \\ x_2^2 + y_2^2 - x_m^2 - y_m^2 + d_m^2 - d_2^2 \\ \vdots \\ x_{m-1}^2 + y_{m-1}^2 - x_m^2 - y_m^2 + d_m^2 - d_{m-1}^2 \end{bmatrix} \tag{10}$$

Coordinates of the unknown nodes **U** are obtained by using least-squares method with Equation (11).

$$\mathbf{X}{=}(\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1}\mathbf{A}^{\mathrm{T}}\mathbf{B} \tag{11}$$

3. Adaptation Multi-Group QUATRE Algorithm and Its Application to Node Localization in WSN

3.1. Adaptation Multi-Group QUATRE Algorithm (AMG-QUATRE)

The main idea of the proposed AMG-QUATRE algorithm is described in this section. As mentioned above, the canonical QUATRE algorithm has the problem of easily trapping into local optima and premature convergence. In order to reduce above weaknesses, we proposed an improved QUATRE, called AMG-QUATRE, which is made up of population initialization, random population division, group evolution, and group recombining and adaption method for updating parameter scale factor F. Figure 1 shows an illustration of the main framework of proposed AMG-QUATRE algorithm.

3.1.1. Population Division and Mutation Schemes

QUATRE variants adopting different mutation schemes usually have different performance in solving different optimization problems, due to different mutation schemes having different exploration and exploitation capability [15]. The mutation scheme "QUATRE/best/1" employs the best individual found so far to guide the search direction, so it has very fast convergence speed and excellent local search capability around the best individual. This mutation scheme performs well when solving unimodal problems. But it is more likely to fall into local optimum and thereby lead to a premature convergence when solving multimodal problems. The mutation scheme "QUATRE/rand/1" is the most commonly used, powerful and robust scheme. Although it has slow convergence speed, it has stronger exploration capability to maintain population diversities. This mutation scheme is usually more suitable for solving multimodal problems. The mutation scheme "QUATRE/target-to-best/1" has good exploration and convergence capabilities, because the individuals using this scheme are guided by the best individual found so far and two randomly selected individuals. To make full use these different schemes, in our proposed algorithm, we utilize multi-group approach to incorporating their advantages. As can be seen from Figure 1, in our proposed algorithm, we first initialize the individual population and then randomly divide the population into three groups, say *group*1, *group*2 and *group*3, respectively. And each group use different mutation scheme, *group*1 uses the scheme "QUATRE/target-to-best/1", *group*2 employs the scheme "QUATRE/rand/1", and *group*3 utilizes the scheme "QUATRE/best/1", respectively. Therefore, this multi-group partition and different group adopting different mutation scheme approach can be traded off between the population diversity and convergence speed.

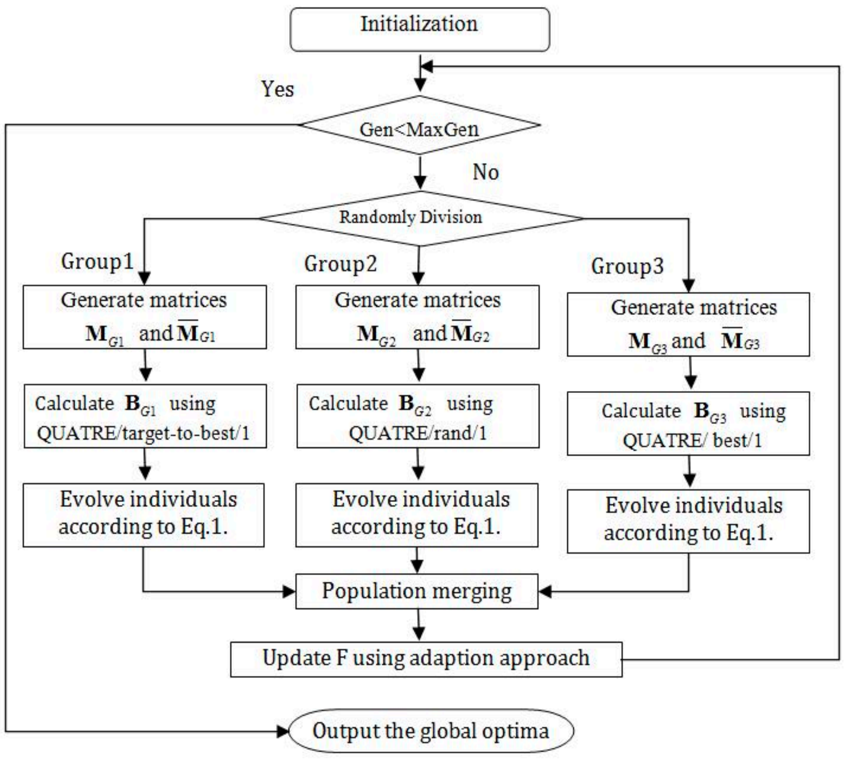
Initialization
Yes
Gen<MaxGen
No
Randomly Division
Group1
Group2
Group3
Generate matrices $\mathbf{M}_{G1}$ and $\overline{\mathbf{M}}_{G1}$
Generate matrices $\mathbf{M}_{G2}$ and $\overline{\mathbf{M}}_{G2}$
Generate matrices $\mathbf{M}_{G3}$ and $\overline{\mathbf{M}}_{G3}$
Calculate $\mathbf{B}_{G1}$ using QUATRE/target-to-best/1
Calculate $\mathbf{B}_{G2}$ using QUATRE/rand/1
Calculate $\mathbf{B}_{G3}$ using QUATRE/ best/1
Evolve individuals according to Eq.1.
Evolve individuals according to Eq.1.
Evolve individuals according to Eq.1.
Population merging
Update F using adaption approach
Output the global optima

Figure 1. The main framework of adaptation multi-group quasi-affine transformation evolutionary algorithm (AMG-QUATRE).

3.1.2. Adaptation Scale Factor

In the whole evolution phases, scale factor plays an important role in affecting the exploration and development capability of QUATRE algorithm. In [15], Meng et al. illustrate that QUATRE with different scaling factor values is suitable for different optimization problems. And there is no fixed parameter setting to obtain the optimal performance for all types of problems. Three categories parameter control methods have been proposed to adjust the evolutionary algorithms parameter values dynamically [27]. Nowadays, most of researchers focused on the adaptive control parameters. Hence, we use the adaptation scheme [22] to dynamically update the value of scale factor. In the proposed algorithm, scale factor F obeys Cauchy distribution, $F \sim C(\mu_F, \sigma_F)$, with location parameter μ_F and scale parameter σ_F. The initial value for μ_F is set to 0.5, and σ_F is set to a constant value 0.1 during the whole search process. Each individual in the population has its own F value which can be updated according to Equation (12).

$$F_i = C_i(\mu_F, \sigma_F) \tag{12}$$

In each generation μ_F is updated by weighted Lehmer mean according to Equation (13). S_F indicates the set of parameters F_i with which the associated individuals can obtain a better fitness value. $f(\mathrm{U}_{i,G})$ represents the fitness value for trial vector of the ith individual at generation G, and $f(\mathrm{U}_{i,G})$ represents the fitness value for target vector of the ith individual at generation G. Δf_i represents the corresponding fitness difference of the ith individual with the associated parameter F_i.

$$\begin{cases} w_{F_i} = \dfrac{\Delta f_i}{\sum_{F_i \in S_F} \Delta f_i} \\ \Delta f_i = f(\mathrm{U}_{i,G}) - f(\mathrm{X}_{i,G}) \\ \mu_F = \dfrac{\sum_{F_i \in S_F} w_{F_i}.F_i^2}{\sum_{F_i \in S_F} w_{F_i}.F_i} \end{cases} \tag{13}$$

Algorithm 1. Shows the Pseudo Code of AMG-QUATRE Algorithm.

```
// Initialization phase
Initialize the searching space V, dimension D, Set the generation counter Gen=1, randomly initialize the population $\mathbf{X}$ with $ps$ individuals, and evaluate fitness values of all individuals, set initial $\mu_F = 0.5$, $\sigma_F = 0.1$.
// Main loop
1: while stopping criterion is not satisfied do
2:     Randomly partition the population into three groups, $group1$, $group2$ and $group3$
3:     Generate matrices $\mathbf{M}_{group1}$ and $\overline{\mathbf{M}}_{group1}$, $\mathbf{M}_{group2}$ and $\overline{\mathbf{M}}_{group2}$, $\mathbf{M}_{group3}$ and $\overline{\mathbf{M}}_{group3}$, using Equation (3).
4:     Calculate mutation matrix $\mathbf{B}_{group1}$ using QUATRE/target-to-best/1, $\mathbf{B}_{group2}$ using QUATRE/rand/1, $\mathbf{B}_{group3}$ using QUATRE/best/1.
5:     Evolve individuals in each group using Equation (1).
6:     Evaluate fitness values of all individuals.
7:       for $i = 1; i \leq ps; i++$ do
8:         if $f(\mathrm{X}_i) \leq f(\mathrm{X}_{pbest,i})$ then
9:           $X_{pbest,i} = X_i$
10:        end if
11:      end for
12:    $\mathbf{X} = \mathbf{X}_{pbest}$, $\mathbf{X}_{gbest} = opt\{\mathbf{X}_{pbest}\}$.
13:    Update scale factor F according to Equation (12).
14:    $Gen = Gen + 1$
15: end while
Output: The global optimum $\mathbf{X}_{gbest}$, global best fitness value $f(\mathrm{X}_{pbset})$.
```

3.2. *Our Proposed Algorithm for DV-Hop Localization Algorithm*

In this subsection, we introduce the proposed AMG-QUATRE algorithm for the DV-Hop node localization in WSN. As we all know, the estimated distance between nodes is obtained multiplying the hop-count values by the hop size of the anchor node. However, when the number of hops between the anchor node and the unknown node is greater than two, this method suffers from a problem of poor distance estimation that leads decreasing accuracy of localization. The main aim of the localization problem in WSN is to minimize the estimation error and to improve the localization accuracy. In order to reduce the estimation error, we present an improved DV-Hop based on AMG-QUATRE for node localization in WSN. The proposed algorithm first refines the hop size of anchor nodes according to the least square error criterion, and then calculates the hop size of unknown nodes by weighting the received hop size from all anchor nodes. Finally, the proposed algorithm employs AMG-QUATRE algorithm to estimate the locations of unknown nodes.

3.2.1. Modification of Hop Size

The hop size of anchor nodes mainly determines the accuracy of the estimated distance. Localization accuracy could be improved by modifying the hop size of anchor nodes according to the references [42]. The least square error criterion is used to refine the average hop distance of each anchor node as follow.

$$Hopsize_i = \frac{\sum_{j=1,j\neq i}^{m} hop_{ij} d_{ij}}{\sum_{j=1,j\neq i}^{m} hop_{ij}^2} \tag{14}$$

where d_{ij} represents the distance from anchor node i to anchor node j. Then, the hop size of unknown nodes is calculated by weighting the m received hop size from all anchor nodes using Equations (15) and (16).

$$Hopsize_u = \sum_{i=1}^{m} w_i Hopsize_i \tag{15}$$

$$w_i = \frac{hop_i}{\sum_{j=1}^{m} hop_j} \tag{16}$$

where hop_i represents the hop-count value from anchor node i to unknown node u and m is the number of all anchor nodes.

3.2.2. Using AMG-QUATRE Algorithm to Locate the Unknown Nodes

In this subsection, the proposed AMG-QUATRE algorithm is applied to minimize squared error of the estimated distances that means to locate unknown nodes. The second phase of DV- Hop algorithm, the distance between the unknown node and anchor node d_{ui} is an estimated value, so it may be error prone. Let (x', y') is the actual position coordinate of unknown node, (x_i, y_i) denotes position coordinate of ith anchor node. The squared error of the estimated distances is defined as follows.

$$e = \sum_{i=1}^{m} \left(\sqrt{(x' - x_i)^2 - (y' - y_i)^2} - d_{ui} \right)^2 \tag{17}$$

The main aim of the localization problem in WSN is to minimize this error. Take into consideration that the distance estimation error also increases as the number of hops increases. So the Equation (17) can be deal in a weighted way by the reciprocal of hop-count. Hence, the fitness function of the AMG-QUATRE can be defined as follows.

$$f(x', y') = \min\left(\sum_{i=1}^{m} \left(\frac{1}{hop_{ui}}\right)^2 \left(\sqrt{(x' - x_i)^2 - (y' - y_i)^2} - d_{ui} \right)^2 \right) \tag{18}$$

For every unknown node, we use an independent AMG-QUATRE optimization process to reckon its location. The individual encoding of AMG-QUATRE for location estimation is two dimension variables (x_i, y_i) representing the coordinate of unknown node. The optimization process contains four steps as follows.

Step 1, initialize parameters of AMG-QUATRE and *ps* individuals.
Step 2, generate donor matrices.
Step 3, generate trial matrices.
Step 4, select a better vector between the trial vector and its corresponding target vector to enter the next generation. Repeat the above steps 2 to 4 until the stop condition is satisfied.

The whole flowchart of the proposed DV-Hop algorithm based on AMG-QUATRE is given in Figure 2.

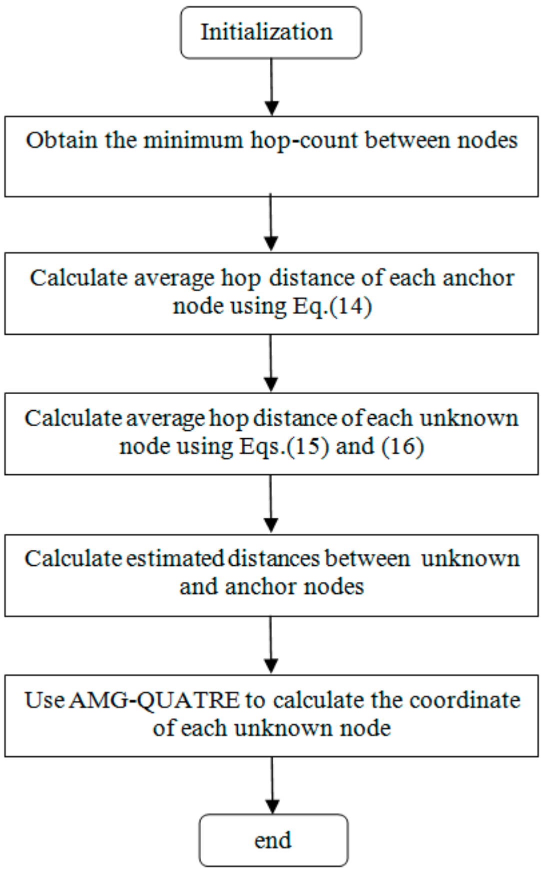
Initialization
Obtain the minimum hop-count between nodes
Calculate average hop distance of each anchor node using Eq.(14)
Calculate average hop distance of each unknown node using Eqs.(15) and (16)
Calculate estimated distances between unknown and anchor nodes
Use AMG-QUATRE to calculate the coordinate of each unknown node
end

Figure 2. The flowchart of the proposed distance vector-hop (DV-Hop) algorithm based on AMG-QUATRE.

4. Experimental Analysis

In this section, two groups of experiments are conducted to verify the performance of our proposed AMG-QUATRE algorithm and its application to node localization in WSN.

4.1. Simulation Results on Standard Bound-Constrained Benchmarks

A benchmark set of CEC2013 [43] is utilized to verify the performance of our proposed AMG-QUATRE algorithm in the following experiment.

CEC2013 has 28 benchmark functions, including five (f_1–f_5) unimodal functions, fifteen (f_6–f_{20}) multi-modal functions and eight (f_{21}–f_{28}) composition functions. These benchmark functions' definition can be found in [43]. All these benchmark functions are shifted to the same global minimum $O\{o_1, o_2, \ldots, o_d\}$.

The proposed AMG-QUATRE is compared with the three QUATRE variants [15,19] "QUATRE/target-to-best/1", "QUATRE/rand/1" and "QUATRE/best/1" because of AMG-QUATRE utilizing these three mutation schemes. We also compare it with DE [8], ODE [44], CLPSO [45], and SLPSO [46] due to the QUATRE algorithm has relationship with PSO and DE algorithms as described in paper [15]. The parameters of these algorithms are set in Table 2 according to the recommended values of the referenced papers. The dimension number D of all functions is set to 50. The maximal number of function evaluation (NFE) is set to $10000 \times D$. We run each of the contrasted algorithms on every benchmark function 20 times independently. Tables 3 and 4 collect the best and mean/standard deviation of the function error $\Delta f = f_i - f_i^*$ respectively. Symbols "+", "−" and "=" in parentheses after the values "Best" and "Mean/ standard deviation" means "better performance", "worse performance" and "similar performance" respectively. Wilcoxon's signed rank test with a level of significance $\alpha = 0.05$ is used for the evaluation of "Mean/Std". "Best" are measured by their associated arithmetic values, and we use the criterion "The smaller the better". The simulation results of all benchmark functions are plotted in Figures 3–6.

Table 2. Parameters settings.

Algorithm	Parameters Settings
DE	$F = 0.5, \mathrm{Cr} = 0.1, ps = 100$
ODE	$F = 0.5, \mathrm{Cr} = 0.1, Jr = 0.3, ps = 100$
CLPSO	$iw \in [0.9, 0.3], cc = 1.49455, Pc \in [0, 0.5], stay\_num = 7, v = \mathrm{rnd}, v\max = 0.2\mathrm{R}$
SLPSO	$M = 100, \mathrm{c}_3 = 0.005, PL \in [0, 1]$
QUATRE variants	$F = 0.7, ps = 100$
AMG-QUATRE	$\mu_F = 0.5, \sigma_F = 0.1, ps = 100$

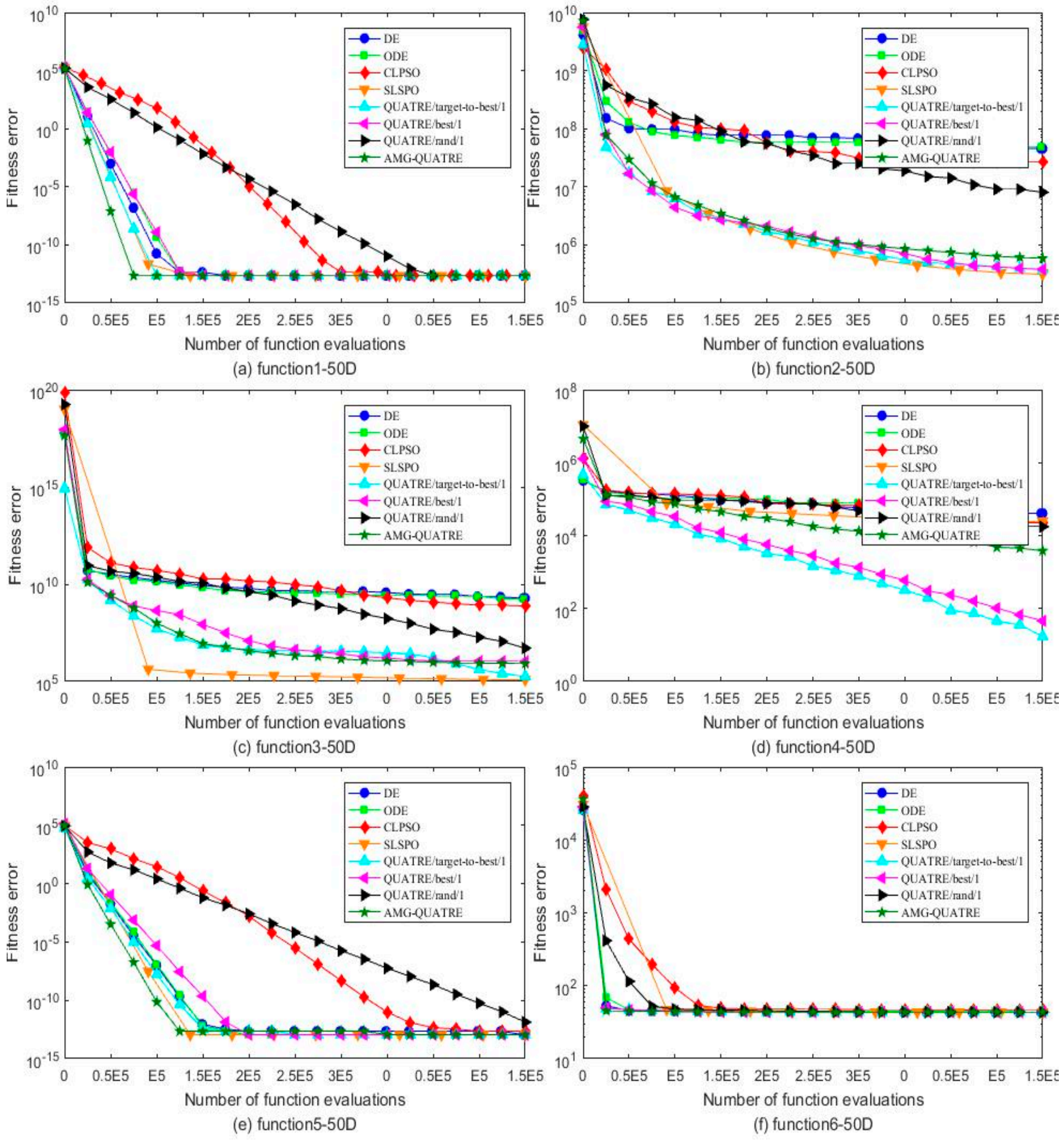

Figure 3. Comparison of the best of fitness errors for functions f_1–f_6 with 50D optimization. (**a**) f_1; (**b**) f_2; (**c**)f_3; (**d**)f_4; (**e**) f_5; (**f**) f_6.

Table 3. Comparison results of best value of 20-run fitness error among contrasted algorithms under CEC2013 test suite.

50D	DE/best/1/bin	ODE/best/1/bin	CLPSO	SLPSO	QUATRE/best	QUATRE/rand	QUATRE/target-to-best	AMP-QUATRE
1	2.273×10^{-13}(=)	2.273×10^{-13}(=)	2.2737×10^{-13}(=)	2.2737×10^{-13}(=)	$0.0000 \times 10^{+00}$(+)	$0.0000 \times 10^{+00}$(+)	2.2737×10^{-13}(=)	2.2737×10^{-13}
2	$4.5454 \times 10^{+07}$(−)	$4.871 \times 10^{+07}$(−)	$5.9535 \times 10^{+05}$(−)	$3.1277 \times 10^{+05}$(+)	$3.8146 \times 10^{+05}$(+)	$8.0401 \times 10^{+06}$(−)	$3.7393 \times 10^{+05}$(+)	$5.8732 \times 10^{+05}$
3	$1.9098 \times 10^{+09}$(−)	$1.775 \times 10^{+09}$(−)	$8.0541 \times 10^{+06}$(−)	$1.2111 \times 10^{+05}$(+)	$1.0726 \times 10^{+06}$(−)	$5.1302 \times 10^{+06}$(−)	$1.7985 \times 10^{+05}$(+)	$8.3722 \times 10^{+05}$
4	$4.0671 \times 10^{+04}$(−)	$4.629 \times 10^{+04}$(−)	$3.818 \times 10^{+03}$(−)	$2.3913 \times 10^{+04}$(−)	$4.5832 \times 10^{+01}$(+)	$1.7578 \times 10^{+04}$(−)	$1.7273 \times 10^{+01}$(+)	$3.8289 \times 10^{+03}$
5	$1.1369 \times 10^{+1}$ (=)	1.136×10^{-13}(=)	1.1369×10^{-13}(−)	1.1369×10^{-13}(=)	1.1369×10^{-13}(=)	1.3642×10^{-12}(−)	1.1369×10^{-13}(=)	1.1369×10^{-13}
6	$4.3447 \times 10^{+01}$(−)	$4.415 \times 10^{+01}$(−)	$4.3447 \times 10^{+01}$(−)	$4.3447 \times 10^{+01}$(=)	$4.3447 \times 10^{+01}$(=)	$4.3447 \times 10^{+01}$(−)	$4.3447 \times 10^{+01}$(=)	$4.3447 \times 10^{+01}$
7	$6.4767 \times 10^{+01}$(−)	$6.1526 \times 10^{+01}$(−)	$3.5117 \times 10^{+01}$(−)	7.1569×10^{-01}(+)	$2.9215 \times 10^{+01}$(+)	$3.1212 \times 10^{+01}$(+)	$9.8822 \times 10^{+00}$(+)	$3.3071 \times 10^{+01}$
8	$2.1041 \times 10^{+01}$(−)	$2.1044 \times 10^{+01}$(−)	$2.1060 \times 10^{+01}$(−)	$2.1044 \times 10^{+0}1$(−)	$2.1060 \times 10^{+01}$(−)	$2.1012 \times 10^{+01}$(−)	$2.1062 \times 10^{+01}$(−)	$2.1003 \times 10^{+01}$
9	$5.5049 \times 10^{+01}$(−)	$3.7639 \times 10^{+01}$(−)	$2.4972 \times 10^{+01}$(−)	$1.2712 \times 10^{+01}$(+)	$2.0720 \times 10^{+01}$(+)	$5.9709 \times 10^{+01}$(−)	$2.7689 \times 10^{+01}$(−)	$2.6244 \times 10^{+01}$
10	$1.1534 \times 10^{+00}$(−)	$1.7926 \times 10^{+00}$(−)	5.9149×10^{-02}(−)	1.0602×10^{-01}(−)	1.7241×10^{-02}(+)	9.4477×10^{-01}(−)	1.4780×10^{-02}(+)	6.6495×10^{-02}
11	5.6843×10^{-14}(+)	5.6843×10^{-14}(+)	$2.0090 \times 10^{+01}$(+)	$1.4924 \times 10^{+01}$(+)	$5.2875 \times 10^{+01}$(−)	$1.0379 \times 10^{+00}$(+)	$7.4948 \times 10^{+01}$(−)	$2.2921 \times 10^{+01}$
12	$2.5041 \times 10^{+02}$(−)	$1.5262 \times 10^{+02}$(−)	$6.4672 \times 10^{+01}$(−)	$3.0614 \times 10^{+02}$(−)	$7.1792 \times 10^{+01}$(−)	$2.5256 \times 10^{+02}$(−)	$2.2941 \times 10^{+02}$(−)	$6.6662 \times 10^{+01}$
13	$3.1407 \times 10^{+02}$(−)	$2.5190 \times 10^{+02}$(−)	$1.3242 \times 10^{+02}$(−)	$3.1176 \times 10^{+02}$(−)	$1.2214 \times 10^{+02}$(+)	$2.4339 \times 10^{+02}$(−)	$2.8762 \times 10^{+02}$(−)	$1.2265 \times 10^{+02}$
14	$6.0810 \times 10^{+00}$(+)	$6.3847 \times 10^{+00}$(+)	$3.7529 \times 10^{+02}$(+)	$6.9829 \times 10^{+02}$(−)	$1.2919 \times 10^{+03}$(−)	$7.9588 \times 10^{+01}$(+)	$3.6695 \times 10^{+03}$(−)	$6.0000 \times 10^{+02}$
15	$1.1449 \times 10^{+04}$(−)	$6.7771 \times 10^{+03}$(−)	$5.0860 \times 10^{+03}$(−)	$3.6314 \times 10^{+03}$(+)	$8.5293 \times 10^{+03}$(−)	$1.0372 \times 10^{+04}$(−)	$1.1384 \times 10^{+04}$(−)	$5.1193 \times 10^{+03}$
16	$2.9382 \times 10^{+00(-)}$	$2.5038 \times 10^{+00}$(−)	2.7308×10^{-01}(−)	$2.8265 \times 10^{+00}$(−)	$2.5942 \times 10^{+00}$(−)	$2.3831 \times 10^{+00}$(−)	$2.1997 \times 10^{+00}$(−)	7.0698×10^{-01}
17	$5.0786 \times 10^{+01}$(+)	$5.0800 \times 10^{+01}$(+)	$7.2026 \times 10^{+01}$(+)	$3.1820 \times 10^{+02}$(−)	$1.1320 \times 10^{+02}$(−)	$5.9894 \times 10^{+01}$(+)	$1.3404 \times 10^{+02}$(−)	$6.8152 \times 10^{+01}$
18	$4.0051 \times 10^{+02}$(−)	$3.6304 \times 10^{+02}$(−)	$9.8335 \times 10^{+01}$(−)	$3.7049 \times 10^{+02}$(−)	$2.6744 \times 10^{+02}$(−)	$3.5631 \times 10^{+02}$(−)	$3.5551 \times 10^{+02}$(−)	$1.0152 \times 10^{+02}$
19	$6.6847 \times 10^{+00}$(−)	$8.9514 \times 10^{+00}$(−)	$4.0347 \times 10^{+00}$(+)	$4.5031 \times 10^{+00}$(−)	$5.0660 \times 10^{+00}$(−)	$9.5681 \times 10^{+00}$(−)	$1.0231 \times 10^{+01}$(−)	$3.4843 \times 10^{+00}$
20	$2.1634 \times 10^{+01}$(−)	$2.1809 \times 10^{+01}$(−)	$1.8929 \times 10^{+01}$(−)	$2.1551 \times 10^{+01}$(−)	$2.0609 \times 10^{+01}$(−)	$2.1465 \times 10^{+01}$(−)	$2.0438 \times 10^{+01}$(−)	$1.7868 \times 10^{+01}$
21	$2.0000 \times 10^{+02}$(=)	$2.0000 \times 10^{+02}$(=)	$2.0000 \times 10^{+02}$(−)	$2.0000 \times 10^{+02}$(=)	$2.0000 \times 10^{+02}$(=)	$2.0000 \times 10^{+02}$(=)	$2.0000 \times 10^{+02}$(=)	$2.0000 \times 10^{+02}$
22	$2.6406 \times 10^{+01}$(+)	$3.0189 \times 10^{+01}$(+)	$6.2665 \times 10^{+02}$(+)	$7.3845 \times 10^{+02}$(−)	$1.7683 \times 10^{+03}$(−)	$1.0261 \times 10^{+02}$(+)	$3.3192 \times 10^{+03}$(−)	$4.8452 \times 10^{+02}$
23	$1.2346 \times 10^{+04}$(−)	$9.3380 \times 10^{+03}$(−)	$4.8609 \times 10^{+03}$(−)	$3.1419 \times 10^{+03}$(+)	$8.5494 \times 10^{+03}$(−)	$1.1668 \times 10^{+04}$(−)	$1.1481 \times 10^{+04}$(−)	$4.9707 \times 10^{+03}$
24	$3.1693 \times 10^{+02}$(−)	$3.2104 \times 10^{+02}$(−)	$2.4601 \times 10^{+02}$(−)	$2.3006 \times 10^{+02}$(+)	$2.5158 \times 10^{+02}$(−)	$2.2959 \times 10^{+02}$(+)	$2.3078 \times 10^{+02}$(+)	$2.4894 \times 10^{+02}$
25	$3.5881 \times 10^{+02}$(−)	$3.6268 \times 10^{+02}$(−)	$3.0172 \times 10^{+02}$(−)	$2.8333 \times 10^{+02}$(+)	$2.8807 \times 10^{+02}$(+)	$3.3466 \times 10^{+02}$(−)	$2.8322 \times 10^{+02}$(+)	$2.9743 \times 10^{+02}$
26	$2.0453 \times 10^{+02}$(−)	$2.0252 \times 10^{+02}$(−)	$2.0021 \times 10^{+02}$(−)	$2.0010 \times 10^{+02}$(+)	$2.0008 \times 10^{+02}$(+)	$2.0071 \times 10^{+02}$(−)	$2.0004 \times 10^{+02}$(+)	$2.0019 \times 10^{+02}$
27	$1.5258 \times 10^{+03}$(−)	$1.6157 \times 10^{+03}$(−)	$7.9749 \times 10^{+02}$(+)	$6.9280 \times 10^{+02}$(+)	$8.2461 \times 10^{+02}$(+)	$1.5090 \times 10^{+03}$(−)	$7.3735 \times 10^{+02}$(+)	$9.2220 \times 10^{+02}$
28	$4.0000 \times 10^{+02}$(=)	$4.0000 \times 10^{+02}$(=)	$4.0000 \times 10^{+02}$(=)	$4.0000 \times 10^{+02}$(=)	$4.0000 \times 10^{+02}$(=)	$4.0000 \times 10^{+02}$(=)	$4.0000 \times 10^{+02}$(=)	$4.0000 \times 10^{+02}$
−/=/+	20/4/4	20/4/4	20/2/6	12/5/11	14/4/10	19/2/7	14/5/9	−/−/−

Table 4. Comparison results of mean and standard deviation of 20-run fitness error among contrasted algorithms under CEC2013 test suite.

50D	**DE/best/1/bin**	**ODE/best/1/bin**	**CLPSO**	**SLPSO**
1	$2.2737 \times 10^{-13}/0.0000 \times 10^{+00}$(=)	$2.2737 \times 10^{-13}/0.0000 \times 10^{+00}$(=)	$2.2737 \times 10^{-13}/0.0000 \times 10^{+00}$(=)	$2.2737 \times 10^{-13}/0.0000 \times 10^{+00}$(=)
2	$6.7630 \times 10^{+07}/1.4092 \times 10^{+07}$(−)	$8.1854 \times 10^{+07}/1.7030 \times 10^{+07}$(−)	$3.9702 \times 10^{+07}/7.0886 \times 10^{+06}$(−)	$8.9731 \times 10^{+05}/3.1582 \times 10^{+05}$(=)
3	$3.2967 \times 10^{+09}/1.8024 \times 10^{+09}$(−)	$4.4957 \times 10^{+09}/1.4925 \times 10^{+09}$(−)	$1.8074 \times 10^{+09}/9.5165 \times 10^{+08}$(−)	$1.1610 \times 10^{+07}/1.4090 \times 10^{+07}$(+)
4	$4.9175 \times 10^{+04}/4.9989 \times 10^{+03}$(−)	$5.7660 \times 10^{+04}/8.7742 \times 10^{+03}$(−)	$3.3408 \times 10^{+04}/6.0160 \times 10^{+03}$(−)	$3.3850 \times 10^{+04}/1.0284 \times 10^{+04}$(−)
5	$2.2737 \times 10^{-13}/3.6885 \times 10^{-14}$(=)	$1.9895 \times 10^{-13}/5.0507 \times 10^{-14}$(=)	$2.8990 \times 10^{-13}/5.8028 \times 10^{-14}$(−)	$1.9895 \times 10^{-13}/5.0507 \times 10^{-14}$(=)
6	$4.4426 \times 10^{+01}/7.7214 \times 10^{-01}$(−)	$4.5560 \times 10^{+01}/1.4691 \times 10^{+00}$(−)	$4.6402 \times 10^{+01}/7.0628 \times 10^{-01}$(−)	$4.3447 \times 10^{+01}/1.2356 \times 10^{-11}$(+)
7	$8.3113 \times 10^{+01}/1.0175 \times 10^{+01}$(−)	$8.8732 \times 10^{+01}/1.2892 \times 10^{+01}$(−)	$1.0165 \times 10^{+02}/8.5250 \times 10^{+00}$(−)	$5.9876 \times 10^{+00}/4.8864 \times 10^{+00}$(+)
8	$2.1127 \times 10^{+01}/3.5876 \times 10^{-02}$(=)	$2.1143 \times 10^{+01}/3.7340 \times 10^{-02}$(=)	$2.1143 \times 10^{+01}/3.7719 \times 10^{-02}$(=)	$2.1119 \times 10^{+01}/3.3008 \times 10^{-02}$(=)
9	$5.8061 \times 10^{+01}/1.8481 \times 10^{+00}$(−)	$5.0049 \times 10^{+01}/5.9272 \times 10^{+00}$(−)	$5.3471 \times 10^{+01}/2.5860 \times 10^{+00}$(−)	$1.8053 \times 10^{+01}/3.5882 \times 10^{+00}$(+)
10	$3.9408 \times 10^{+00}/2.0661 \times 10^{+00}$(−)	$6.3503 \times 10^{+00}/4.4398 \times 10^{+00}$(−)	$6.0611 \times 10^{+00}/1.4295 \times 10^{+00}$(−)	$2.6597 \times 10^{-01}/1.1229 \times 10^{-01}$(−)
11	$1.9402 \times 10^{+00}/1.6920 \times 10^{+00}$(+)	$1.9402 \times 10^{+00}/1.5970 \times 10^{+00}$(+)	$8.8107 \times 10^{-14}/2.9014 \times 10^{-14}$(+)	$3.4565 \times 10^{+01}/1.1287 \times 10^{+01}$(=)
12	$3.1874 \times 10^{+02}/2.7910 \times 10^{+01}$(−)	$2.5954 \times 10^{+02}/3.3785 \times 10^{+01}$(−)	$2.7169 \times 10^{+02}/2.8911 \times 10^{+01}$(−)	$3.4056 \times 10^{+02}/1.4672 \times 10^{+01}$(−)
13	$3.4943 \times 10^{+02}/1.7881 \times 10^{+01}$(−)	$3.1646 \times 10^{+02}/3.3786 \times 10^{+01}$(−)	$3.5904 \times 10^{+02}/3.9979 \times 10^{+01}$(−)	$3.3874 \times 10^{+02}/1.0522 \times 10^{+01}$(−)
14	$9.2118 \times 10^{+01}/9.8833 \times 10^{+01}$(+)	$5.4027 \times 10^{+01}/6.5389 \times 10^{+01}$(+)	$4.3188 \times 10^{+01}/1.1086 \times 10^{+01}$(+)	$1.1953 \times 10^{+03}/3.3024 \times 10^{+02}$(−)
15	$1.2980 \times 10^{+04}/6.5858 \times 10^{+02}$(−)	$1.1202 \times 10^{+04}/1.7749 \times 10^{+03}$(−)	$9.2360 \times 10^{+03}/5.1031 \times 10^{+02}$(−)	$1.2144 \times 10^{+04}/2.9152 \times 10^{+03}$(−)
16	$3.3028 \times 10^{+00}/2.1856 \times 10^{-01}$(−)	$3.2672 \times 10^{+00}/3.4818 \times 10^{-01}$(−)	$2.6884 \times 10^{+00}/2.9832 \times 10^{-01}$(−)	$3.3398 \times 10^{+00}/2.5719 \times 10^{-01}$(−)
17	$5.0939 \times 10^{+01}/2.1260 \times 10^{-01}$(+)	$5.1808 \times 10^{+01}/1.0106 \times 10^{+00}$(+)	$5.3451 \times 10^{+01}/5.9901 \times 10^{-01}$(+)	$3.5993 \times 10^{+02}/2.4006 \times 10^{+01}$(−)
18	$4.2249 \times 10^{+02}/1.4402 \times 10^{+01}$(−)	$3.9685 \times 10^{+02}/1.5608 \times 10^{+01}$(−)	$4.0577 \times 10^{+02}/2.3800 \times 10^{+01}$(−)	$3.9239 \times 10^{+02}/1.2010 \times 10^{+01}$(−)
19	$8.7896 \times 10^{+00}/7.5759 \times 10^{-01}$(−)	$1.0217 \times 10^{+01}/4.9098 \times 10^{-01}$(−)	$3.0401 \times 10^{+00}/4.7453 \times 10^{-01}$(+)	$6.3564 \times 10^{+00}/9.8914 \times 10^{-01}$(−)
20	$2.2341 \times 10^{+01}/2.9931 \times 10^{-01}$(−)	$2.2343 \times 10^{+01}/2.7942 \times 10^{-01}$(−)	$2.3215 \times 10^{+01}/5.3751 \times 10^{-01}$(−)	$2.2119 \times 10^{+01}/3.1389 \times 10^{-01}$(−)
21	$6.3252 \times 10^{+02}/4.5144 \times 10^{+02}$(=)	$7.4552 \times 10^{+02}/3.8638 \times 10^{+02}$(=)	$3.5629 \times 10^{+02}/1.7059 \times 10^{+02}$(+)	$8.3775 \times 10^{+02}/3.5207 \times 10^{+02}$(=)
22	$2.1962 \times 10^{+02}/5.4330 \times 10^{+02}$(+)	$8.2888 \times 10^{+02}/8.9069 \times 10^{+02}$(=)	$1.1107 \times 10^{+02}/8.2297 \times 10^{+01}$(+)	$1.3757 \times 10^{+03}/3.8553 \times 10^{+02}$(−)
23	$1.3292 \times 10^{+04}/4.4167 \times 10^{+02}$(−)	$1.1793 \times 10^{+04}/1.2168 \times 10^{+03}$(−)	$1.0989 \times 10^{+04}/7.4371 \times 10^{+02}$(−)	$1.2284 \times 10^{+04}/2.2400 \times 10^{+03}$(−)
24	$3.2829 \times 10^{+02}/8.3062 \times 10^{+00}$(−)	$3.3833 \times 10^{+02}/1.0558 \times 10^{+01}$(−)	$3.4471 \times 10^{+02}/8.4855 \times 10^{+00}$(−)	$2.5367 \times 10^{+02}/1.0552 \times 10^{+01}$(+)
25	$3.7028 \times 10^{+02}/7.0668 \times 10^{+00}$(−)	$3.7432 \times 10^{+02}/4.9628 \times 10^{+00}$(−)	$3.8750 \times 10^{+02}/7.9298 \times 10^{+00}$(−)	$2.9793 \times 10^{+02}/7.6039 \times 10^{+00}$(+)
26	$2.0754 \times 10^{+02}/1.6936 \times 10^{+00}$(+)	$2.1829 \times 10^{+02}/5.4942 \times 10^{+01}$(+)	$2.0422 \times 10^{+02}/1.0483 \times 10^{+00}$(+)	$3.2412 \times 10^{+02}/4.4937 \times 10^{+01}$(+)
27	$1.7343 \times 10^{+03}/8.7052 \times 10^{+01}$(−)	$1.7591 \times 10^{+03}/7.7658 \times 10^{+01}$(−)	$1.5672 \times 10^{+03}/4.9764 \times 10^{+02}$(−)	$7.9272 \times 10^{+02}/6.7907 \times 10^{+01}$(+)
28	$8.7540 \times 10^{+02}/1.1611 \times 10^{+03}$(=)	$7.1055 \times 10^{+02}/9.5585 \times 10^{+02}$(=)	$4.0000 \times 10^{+02}/3.8809 \times 10^{-05}$(=)	$4.0000 \times 10^{+02}/1.8070 \times 10^{-13}$(+)
−/=/+	18/5/5	18/6/4	18/3/7	13/6/9

Table 4. *Cont.*

50D	QUATRE/best	QUATRE/rand	QUATRE/target-to-best	AMP-QUATRE
1	$2.1600 \times 10^{-13}/5.0842 \times 10^{-14}(=)$	$4.5475 \times 10^{-14}/9.3312 \times 10^{-14}(+)$	$2.2737 \times 10^{-13}/0.0000 \times 10^{+00}(=)$	$2.2737 \times 10^{-13}/0.0000 \times 10^{+00}$
2	$1.0164 \times 10^{+06}/3.7357 \times 10^{+05}(=)$	$1.5023 \times 10^{+07}/4.6299 \times 10^{+06}(-)$	$5.5836 \times 10^{+05}/1.8107 \times 10^{+05}(+)$	$1.0360 \times 10^{+06}/3.4365 \times 10^{+05}$
3	$2.3504 \times 10^{+07}/2.4206 \times 10^{+07}(+)$	$4.4566 \times 10^{+07}/3.8021 \times 10^{+07}(=)$	$3.6782 \times 10^{+06}/4.3941 \times 10^{+06}(+)$	$5.9671 \times 10^{+07}/7.3033 \times 10^{+07}$
4	$1.3953 \times 10^{+02}/1.1877 \times 10^{+02}(+)$	$2.7389 \times 10^{+04}/4.9350 \times 10^{+03}(-)$	$4.8391 \times 10^{+01}/3.2542 \times 10^{+01}(+)$	$6.8604 \times 10^{+03}/1.8305 \times 10^{+03}$
5	$1.5348 \times 10^{-13}/5.5634 \times 10^{-14}(+)$	$5.3547 \times 10^{-12}/2.1671 \times 10^{-12}(-)$	$1.9895 \times 10^{-13}/5.0507 \times 10^{-14}(=)$	$2.1600 \times 10^{-13}/5.0842 \times 10^{-14}$
6	$4.5714 \times 10^{+01}/1.0138 \times 10^{+01}(=)$	$4.3448 \times 10^{+01}/2.3788 \times 10^{-04}(-)$	$4.3447 \times 10^{+01}/1.5166 \times 10^{-13}(+)$	$4.3741 \times 10^{+01}/1.2778 \times 10^{+00}$
7	$6.7584 \times 10^{+01}/2.9108 \times 10^{+01}(=)$	$4.7842 \times 10^{+01}/8.1591 \times 10^{+00}(=)$	$3.2089 \times 10^{+01}/1.3516 \times 10^{+01}(+)$	$5.0112 \times 10^{+01}/1.1419 \times 10^{+01}$
8	$2.1186 \times 10^{+01}/4.1762 \times 10^{-02}(-)$	$2.1130 \times 10^{+01}/4.0969 \times 10^{-02}(=)$	$2.1132 \times 10^{+01}/3.3735 \times 10^{-02}(=)$	$2.1129 \times 10^{+01}/4.3053 \times 10^{-02}$
9	$3.7688 \times 10^{+01}/9.3369 \times 10^{+00}(=)$	$6.2639 \times 10^{+01}/1.3582 \times 10^{+00}(-)$	$5.1629 \times 10^{+01}/1.2135 \times 10^{+01}(-)$	$3.5635 \times 10^{+01}/5.3692 \times 10^{+00}$
10	$4.8407 \times 10^{-02}/2.7521 \times 10^{-02}(+)$	$1.0614 \times 10^{+00}/4.4795 \times 10^{-02}(-)$	$5.1479 \times 10^{-02}/2.2316 \times 10^{-02}(+)$	$1.6877 \times 10^{-01}/9.1756 \times 10^{-02}$
11	$8.2337 \times 10^{+01}/1.8602 \times 10^{+01}(-)$	$3.2253 \times 10^{+00}/1.5308 \times 10^{+00}(+)$	$8.7687 \times 10^{+01}/6.7867 \times 10^{+00}(-)$	$3.2670 \times 10^{+01}/7.8026 \times 10^{+00}$
12	$1.7239 \times 10^{+02}/5.2618 \times 10^{+01}(-)$	$2.8889 \times 10^{+02}/1.8586 \times 10^{+01}(-)$	$2.6828 \times 10^{+02}/2.2955 \times 10^{+01}(-)$	$9.6453 \times 10^{+01}/1.9272 \times 10^{+01}$
13	$2.4279 \times 10^{+02}/6.2197 \times 10^{+01}(-)$	$3.2962 \times 10^{+02}/2.8567 \times 10^{+01}(-)$	$3.2354 \times 10^{+02}/1.9088 \times 10^{+01}(-)$	$1.8994 \times 10^{+02}/3.9348 \times 10^{+01}$
14	$2.0942 \times 10^{+03}/4.5566 \times 10^{+02}(-)$	$1.0795 \times 10^{+02}/2.0880 \times 10^{+01}(+)$	$4.0460 \times 10^{+03}/2.6165 \times 10^{+02}(-)$	$9.4145 \times 10^{+02}/2.6942 \times 10^{+02}$
15	$1.0621 \times 10^{+04}/1.3099 \times 10^{+03}(-)$	$1.2616 \times 10^{+04}/7.7705 \times 10^{+02}(-)$	$1.2556 \times 10^{+04}/4.3565 \times 10^{+02}(-)$	$6.7499 \times 10^{+03}/9.2335 \times 10^{+02}$
16	$3.2825 \times 10^{+00}/3.6056 \times 10^{-01}(-)$	$3.2270 \times 10^{+00}/3.4962 \times 10^{-01}(-)$	$3.1910 \times 10^{+00}/3.5035 \times 10^{-01}(-)$	$2.1841 \times 10^{+00}/6.6439 \times 10^{-01}$
17	$1.4570 \times 10^{+02}/2.2191 \times 10^{+01}(-)$	$6.3096 \times 10^{+01}/2.0368 \times 10^{+00}(+)$	$1.4326 \times 10^{+02}/6.5122 \times 10^{+00}(-)$	$8.4913 \times 10^{+01}/1.1647 \times 10^{+01}$
18	$3.3174 \times 10^{+02}/4.0023 \times 10^{+01}(-)$	$3.9469 \times 10^{+02}/1.6971 \times 10^{+01}(-)$	$3.8234 \times 10^{+02}/1.6744 \times 10^{+01}(-)$	$1.3042 \times 10^{+02}/1.7501 \times 10^{+01}$
19	$8.9865 \times 10^{+00}/2.3915 \times 10^{+00}(-)$	$1.1791 \times 10^{+01}/1.0069 \times 10^{+00}(-)$	$1.1623 \times 10^{+01}/6.9306 \times 10^{-01}(-)$	$5.7073 \times 10^{+00}/1.2275 \times 10^{+00}$
20	$2.1561 \times 10^{+01}/6.1393 \times 10^{-01}(-)$	$2.2261 \times 10^{+01}/2.7190 \times 10^{-01}(-)$	$2.1631 \times 10^{+01}/4.1090 \times 10^{-01}(-)$	$1.9466 \times 10^{+01}/9.0364 \times 10^{-01}$
21	$7.5331 \times 10^{+02}/4.6351 \times 10^{+02}(=)$	$3.3833 \times 10^{+02}/3.3784 \times 10^{+02}(+)$	$6.9616 \times 10^{+02}/4.2920 \times 10^{+02}(=)$	$8.3451 \times 10^{+02}/3.9226 \times 10^{+02}$
22	$2.7567 \times 10^{+03}/5.0416 \times 10^{+02}(-)$	$1.5228 \times 10^{+02}/3.7961 \times 10^{+01}(+)$	$4.0302 \times 10^{+03}/3.6030 \times 10^{+02}(-)$	$1.0156 \times 10^{+03}/3.1553 \times 10^{+02}$
23	$1.0749 \times 10^{+04}/1.2205 \times 10^{+03}(-)$	$1.2862 \times 10^{+04}/5.9890 \times 10^{+02}(-)$	$1.2310 \times 10^{+04}/4.3309 \times 10^{+02}(-)$	$7.3067 \times 10^{+03}/1.1271 \times 10^{+03}$
24	$2.8054 \times 10^{+02}/1.6244 \times 10^{+01}(=)$	$2.5232 \times 10^{+02}/2.0385 \times 10^{+01}(+)$	$2.5944 \times 10^{+02}/1.5156 \times 10^{+01}(+)$	$2.7678 \times 10^{+02}/1.4085 \times 10^{+01}$
25	$3.1143 \times 10^{+02}/1.3222 \times 10^{+01}(=)$	$3.6970 \times 10^{+02}/1.5311 \times 10^{+01}(-)$	$3.0858 \times 10^{+02}/1.7011 \times 10^{+01}(=)$	$3.1547 \times 10^{+02}/1.2938 \times 10^{+01}$
26	$3.7335 \times 10^{+02}/4.4089 \times 10^{+01}(=)$	$2.7737 \times 10^{+02}/1.1776 \times 10^{+02}(=)$	$3.4616 \times 10^{+02}/6.6582 \times 10^{+01}(+)$	$3.7563 \times 10^{+02}/4.3086 \times 10^{+01}$
27	$1.1743 \times 10^{+03}/2.1424 \times 10^{+02}(=)$	$1.8024 \times 10^{+03}/1.0782 \times 10^{+02}(-)$	$9.5415 \times 10^{+02}/1.3437 \times 10^{+02}(+)$	$1.1754 \times 10^{+03}/1.3799 \times 10^{+02}$
28	$1.1486 \times 10^{+03}/1.3303 \times 10^{+03}(=)$	$4.0000 \times 10^{+02}/3.5987 \times 10^{-09}(+)$	$8.4294 \times 10^{+02}/1.0819 \times 10^{+03}(=)$	$1.1617 \times 10^{+03}/1.3536 \times 10^{+03}$
−/=/+	13/11/4	16/4/8	13/6/9	−/−/−

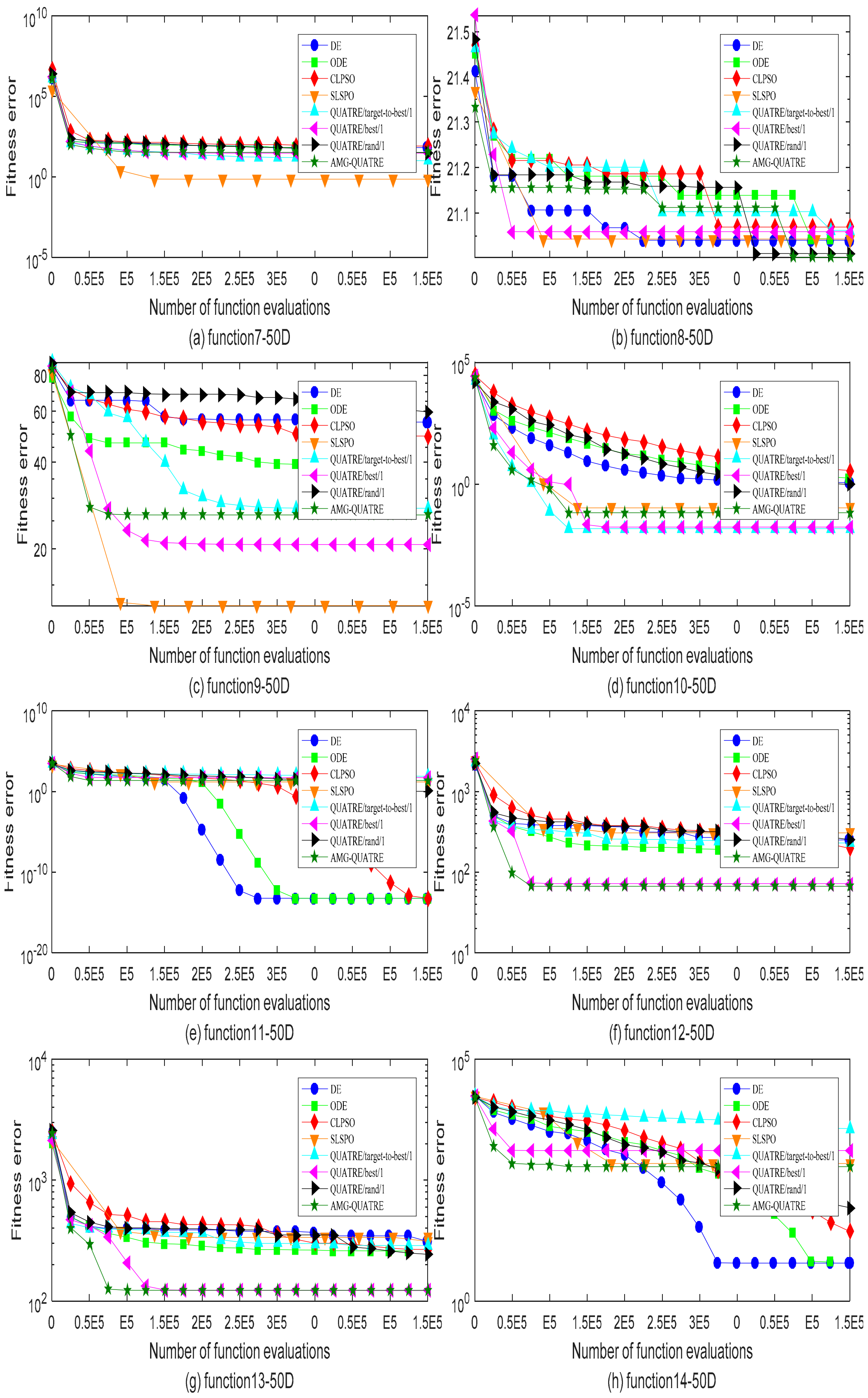

Figure 4. Comparison of the best of fitness errors for functions f_7–f_{14} with 50D optimization. (**a**) f_7; (**b**) f_8; (**c**)f_9; (**d**)f_{10}; (**e**) f_{11}; (**f**)f_{12}; (**g**) f_{13}; (**h**)f_{14}.

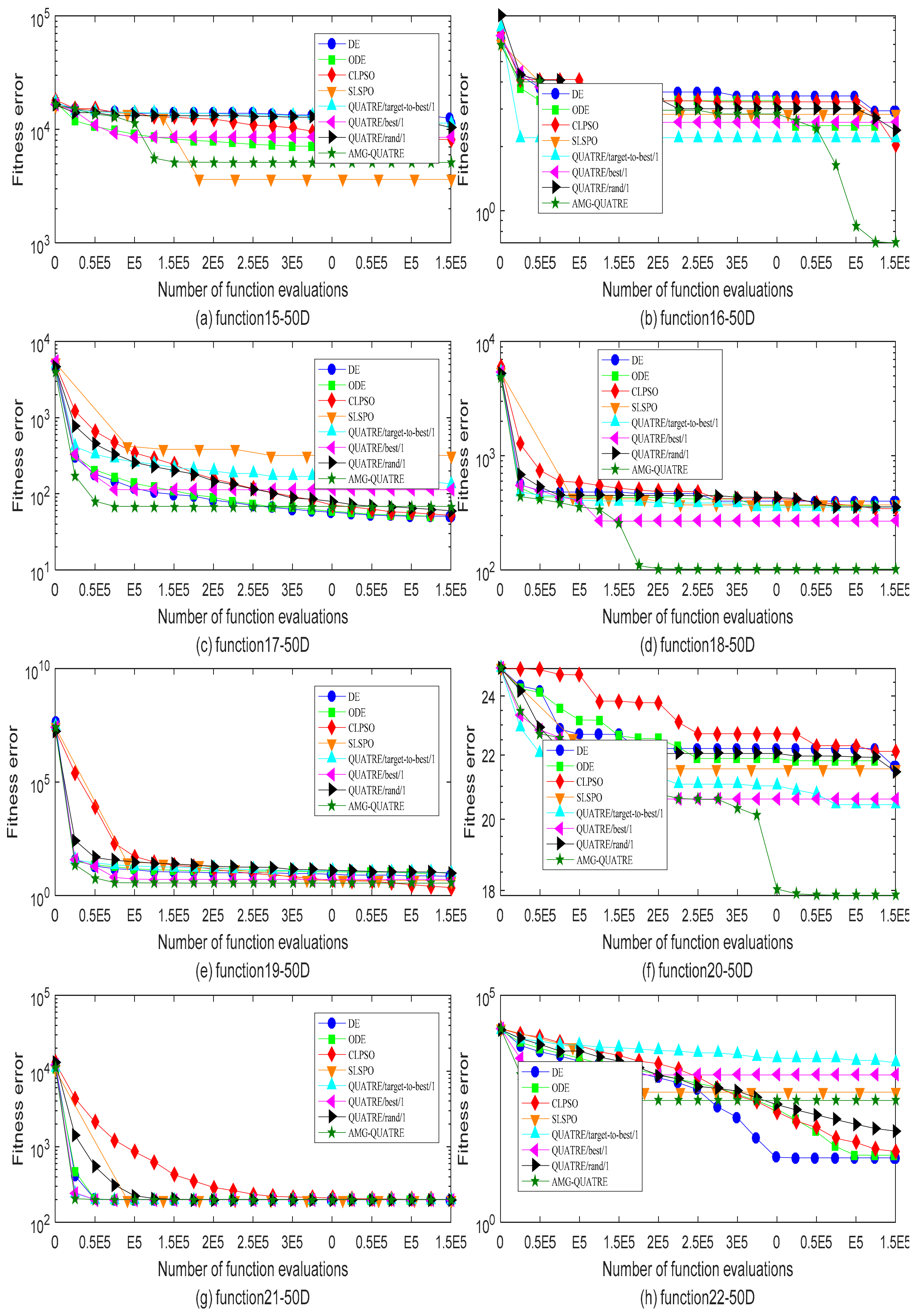

Figure 5. Comparison of the best of fitness errors for functions f_{15}–f_{22} with 50D optimization. (**a**) f_{15}; (**b**) f_{16}; (**c**) f_{17}; (**d**) f_{18}; (**e**) f_{19}; (**f**) f_{20}; (**g**) f_{21}; (**h**) f_{22}.

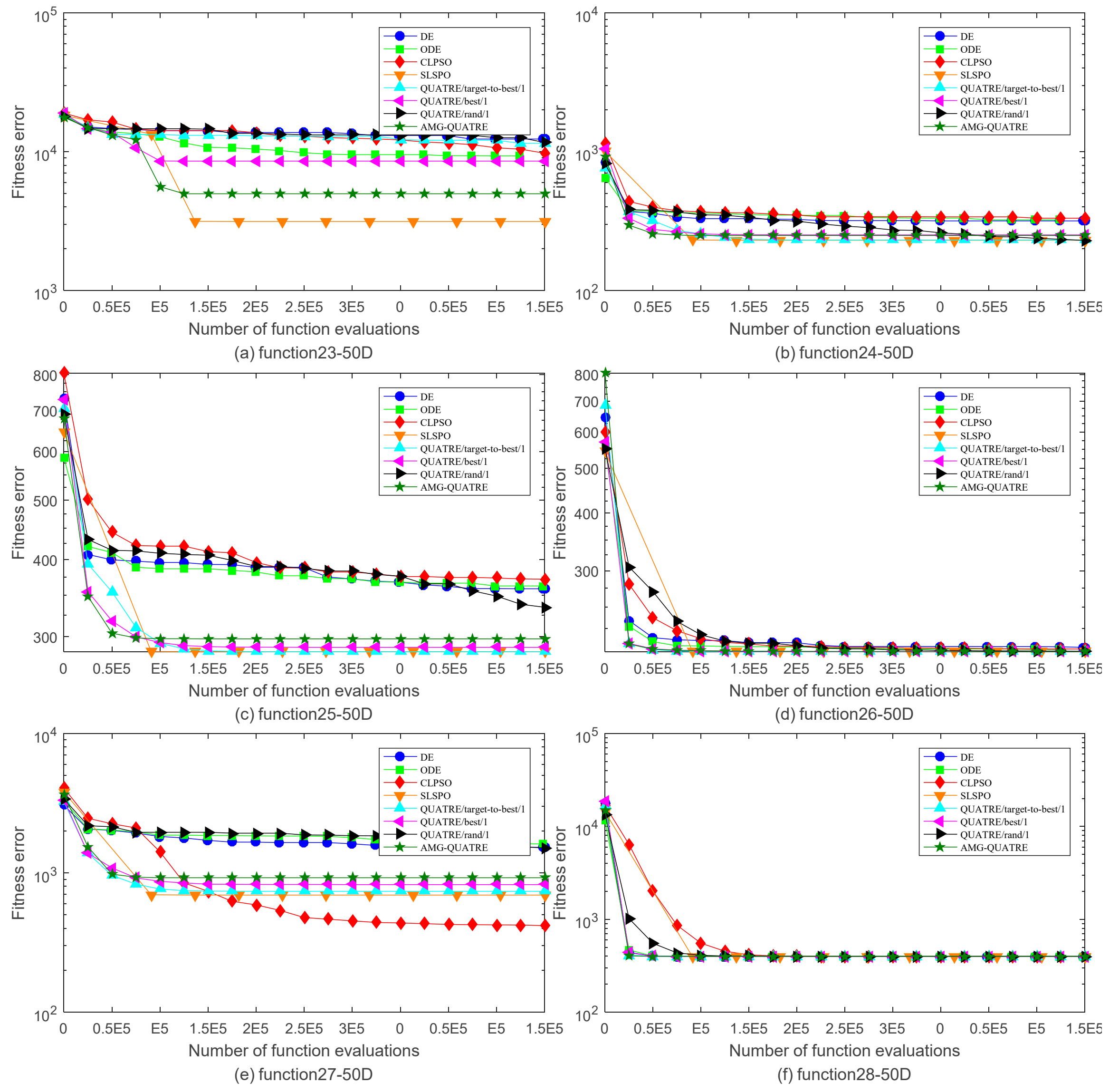

Figure 6. Comparison of the best of fitness errors for functions f_{23}–f_{28} with 50D optimization. (**a**) f_{23}; (**b**) f_{24}; (**c**)f_{25}; (**d**)f_{26}; (**e**) f_{27}; (**f**) f_{28}.

As can see from Table 3, the AMG-QUATRE outperform the other seven contrasted algorithms significantly over 28 benchmark functions from the optimization accuracy perspective. Comparing with DE algorithm, the proposed AMG-QUATRE algorithm obtained 20 better performances, 4 similar performances, and 4 worse performances out of 28 benchmarks. Comparing with ODE algorithm, it obtained 20 better performances, 4 similar performances, and 4 worse performances out of 28 benchmarks. Comparing with CLPSO algorithm, it obtained 20 better performances, 2 similar performances, and 6 worse performances out of 28 benchmarks. Comparing with SLPSO algorithm, it obtained 12 better performances, 5 similar performances, and 11 worse performances out of 28 benchmarks. Comparing with "QUATRE/best/1", it obtained 14 better performances, 4 similar performances, and 10 worse performances out of 28 benchmarks. Comparing with "QUATRE/rand/1", it obtained 19 better performances, 2 similar performances, and 7 worse performances out of 28 benchmarks. Comparing with "QUATRE/target-to-best/1", it obtained 14 better performances, 5 similar performances, and 9 worse performances out of 28 benchmarks. The convergence speeds of these algorithms are contrasted by plotting the convergence curves of best values in Figures 3–6.

As can see from the figures, the proposed AMG-QUATRE algorithm performs best on f_8, f_{12}, f_{13}, f_{16}, f_{18}, f_{20} than other competing algorithms.

From Table 4, we can see that, for the "Mean/standard deviation" value, the proposed AMG-QUATRE algorithm obtained 18 better performances, 5 similar performances, and 5 worse performances out of 28 benchmarks by comparing with DE algorithm. It obtained 18 better performances, 6 similar performances, and 4 worse performances by comparing with ODE algorithm. It obtained 18 better performances, 3 similar performances, and 7 worse performances by comparing with CLPSO algorithm. It obtained 13 better performances, 6 similar performances, and 9 worse performances by comparing with SLPSO algorithm. It obtained 13 better performances, 11 similar performances, and 4 worse performances by comparing with "QUATRE/best/1" algorithm. It obtained 16 better performances, 4 similar performances, and 8 worse performances by comparing with "QUATRE/rand/1" algorithm. It obtained 13 better performances, 6 similar performances, and 9 worse performances by comparing with "QUATRE/target-to-best/1" algorithm. Overall, our proposed AMG-QUATRE algorithm performed better than the other seven competing algorithms.

4.2. Simulation Results of Applied AMG-QUATRE to Node Localization in WSN

Simulations were conducted to verify the performance of our proposed DV-Hop based on AMG-QUATRE. We compared simulation results of our proposed algorithm with the standard DV-Hop, Hyperbolic-DV-hop, DV-Hop based on PSO, and DV-Hop based on DE. Hyperbolic-DV-hop is the algorithm that uses 2D Hyperbolic to replace the multilateration method in original DV-Hop. Using 2D Hyperbolic to estimate the location of unknown node can be found in references [42,47]. These five algorithms were implemented by MATLAB 2016a. The final performance of all the experimental results are the average of 20 runs of independent experiments. In simulations, sensor nodes were randomly scattered in 2-dimensional fixed square region of 100 m × 100 m. The parameter settings of simulation are shown in Table 5. The other parameters of AMG-QUATRE and DE are the same as Section 4.1. The parameters of PSO are $c_1 = c_2 = 2.05, w_{\max} = 0.9,$ and $w_{\min} = 0.4$.

Table 5. Parameter settings for simulation.

Simulation Parameters	Parameters Settings
Sensing region area	100 m × 100 m
Total number of sensor nodes	100–400
Communication range	15–40 m
Percentage of anchor nodes	5–40%
Initial population size	20
Maximum generations	100

The average localization error is considered as performance metrics, which is given as follows.

$$\text{average localization error} = \frac{\sum_{i=1}^{n} \sqrt{(x_i - x_i')^2 + (y_i - y_i')^2}}{n \times R} \tag{19}$$

where n is the total number of unknown nodes, (x_i, y_i) is the estimated coordinate of unknown node and $\left(x_i', y_i'\right)$ is the actual coordinates and R is the communication range of sensors.

4.2.1. Sensitivity of Varying Anchor Node Ratio

In this experiment, there are 200 sensor nodes, each with a communication range of 20 m, randomly scattered in a sensing region of 100 m × 100 m. The number of anchor nodes gradually varies from 5 to 40. The experimental results are shown in Figure 7 and Table 6. Convergence curve for metaheuristics of one single simulation run is shown in Figure 8.

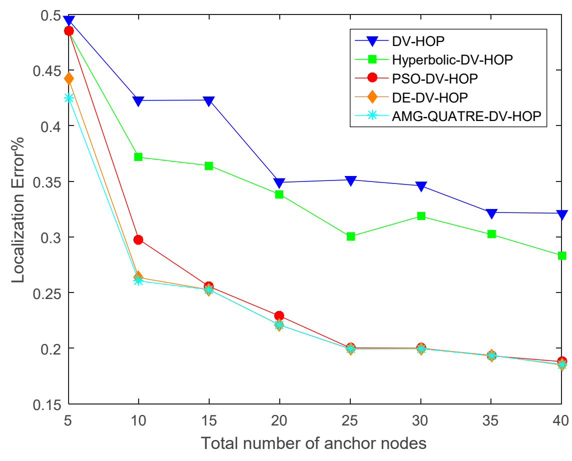

Figure 7. Comparison of location error of the applied AMG-QUATRE with the other methods for different anchor nodes.

Table 6. Comparison of location errors of the applied AMG-QUATRE with the other methods for different anchor nodes.

Anchor Nodes	5	10	15	20	25	30	35	40	Avg
DV-Hop	0.495	0.4227	0.423	0.349	0.3513	0.346	0.322	0.3213	0.378788
Hyperbolic-DV-Hop	0.4847	0.3716	0.3641	0.3382	0.3004	0.3185	0.3023	0.2834	0.3454
PSO-DV-Hop	0.4855	0.2979	0.2554	0.2289	0.2004	0.2001	0.1931	0.188	0.256163
DE-DV-Hop	0.4423	0.2634	0.2525	0.2207	0.1995	0.1997	0.1937	0.1849	0.244588
AMG-QUATRE-DV-Hop	0.4255	0.2605	0.2525	0.2209	0.1993	0.1995	0.1934	0.1855	0.242138

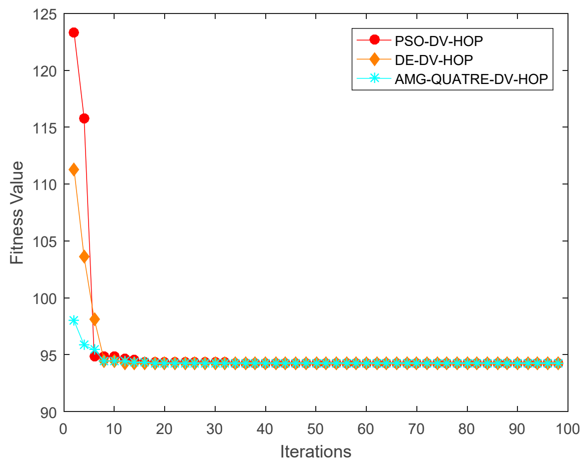

Figure 8. Comparison of convergence curve of the applied AMG-QUATRE with particle swarm optimization (PSO) and differential evolution (DE) methods for single simulation (Number of sensor node 200, Number of anchor node 40, Communication range 20).

We can see from Figure 7 and Table 6 that our proposed algorithm achieved better performance than the DV-Hop, Hyperbolic-DV-hop and PSO-DV-hop algorithms. Our proposed algorithm had similar a performance with the DE-DV-hop algorithm. The average localization error of the proposed algorithm is about 13.6%, 10.3% and 1.4% lower than the standard DV-Hop algorithm, Hyperbolic-DV-hop algorithm and PSO-DV-hop algorithm, respectively. It is also observed from Figure 7 that as the number of anchor node increased, the average localization error of all algorithms also decreased. This is because as the number of anchor nodes increases, the average hop distances of anchor nodes become more accurate and the errors of the estimated distances become smaller.

4.2.2. Sensitivity of Varying Communication Range

In this experiment, there were 200 sensor nodes with 10% anchor nodes randomly scattered in a sensing region of 100 m × 100 m. The communication range varies from 15 to 40 m. The experimental results are shown in Figure 9 and Table 7. Convergence curve for metaheuristics of one single simulation run is shown in Figure 10.

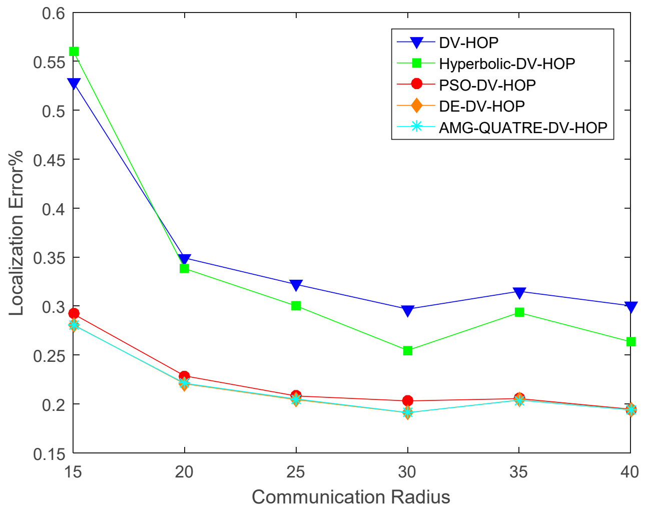

Figure 9. Comparison of location errors of the applied AMG-QUATRE with the other methods for a different communication range.

Table 7. Comparison of location errors of the applied AMG-QUATRE with the other methods for different communication ranges.

Communication Range	15	20	25	30	35	40	Avg.
DV-Hop	0.5286	0.349	0.3219	0.2968	0.3149	0.3002	0.3519
Hyperbolic-DV-Hop	0.5603	0.3382	0.3	0.2546	0.2931	0.2634	0.334933
PSO-DV-Hop	0.2919	0.2286	0.2081	0.203	0.2053	0.1945	0.2219
DE-DV-Hop	0.281	0.2204	0.2042	0.1909	0.2038	0.194	0.215717
AMG-QUATRE-DV-Hop	0.2806	0.2209	0.205	0.1911	0.2037	0.1939	0.215867

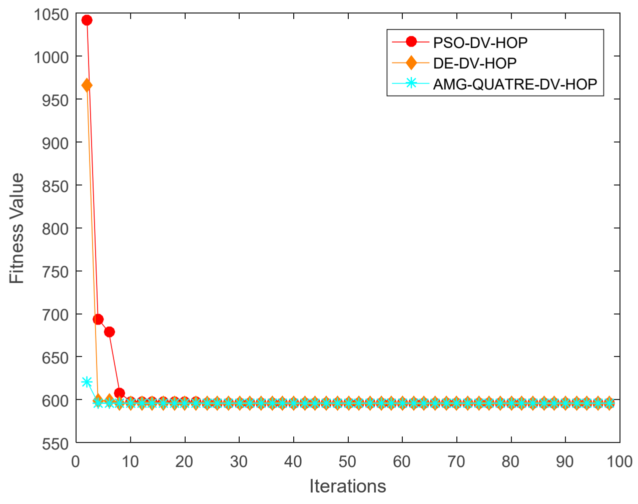

Figure 10. Comparison of convergence curve of the applied AMG-QUATRE with PSO and DE methods for single simulation (Number of sensor node 200, Number of anchor node 20, Communication range 40).

We can see from Figure 9 and Table 7 that our proposed algorithm achieved better performance than the DV-Hop, Hyperbolic-DV-hop and PSO-DV-hop algorithms. Our proposed algorithm had a similar performance with the DE-DV-hop algorithm. The average localization error of our proposed algorithm was about 13.6%, 11.9% and 0.6% lower than the standard DV-Hop algorithm, Hyperbolic-DV-hop algorithm and PSO-DV-hop algorithm, respectively. It was also observed that when the communication range exceeds 25m, the average localization error changes very little. This is because with the increase of communication range, the hop-count between sensor nodes decreases and hop size of anchor nodes increases.

4.2.3. Sensitivity of Node Density

In this simulation experiment, the number of sensor nodes varied from 100 to 400 with 10% anchor nodes randomly scattered in a 100 m × 100 m sensing region. The communication range of each sensor node is set to 20 m. The experimental results are shown in Figure 11 and Table 8. Convergence curve for metaheuristics of one single simulation run is shown in Figure 12.

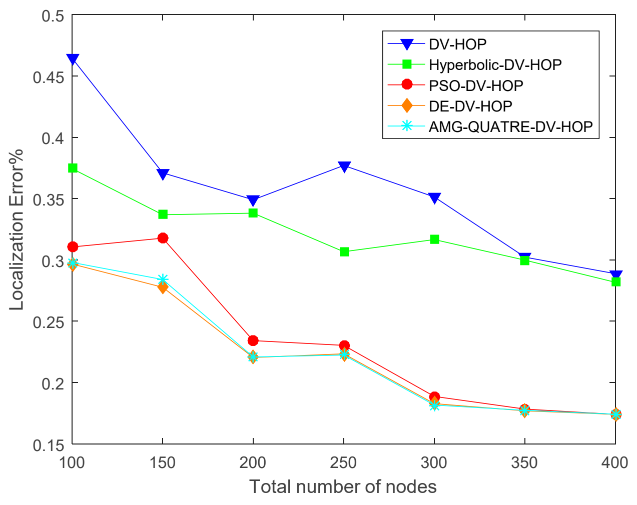

Figure 11. Comparison of average localization error of the applied AMG-QUATRE with the other methods for the different number of sensor nodes.

Table 8. Comparison of the applied AMG-QUATRE with the other methods by different sensor nodes.

Sensor Nodes	100	150	200	250	300	350	400	Avg.
DV-Hop	0.4645	0.3711	0.349	0.3771	0.3513	0.3022	0.2887	0.3577
Hyperbolic-DV-Hop	0.3745	0.3368	0.3382	0.3067	0.3166	0.2998	0.2818	0.322057
PSO-DV-Hop	0.3106	0.3178	0.2342	0.2303	0.1887	0.1784	0.1741	0.233443
DE-DV-Hop	0.2967	0.2777	0.2205	0.2234	0.1828	0.1769	0.1741	0.221729
AMG-QUATRE-DV-Hop	0.2979	0.284	0.2209	0.2225	0.1816	0.1773	0.1742	0.222629

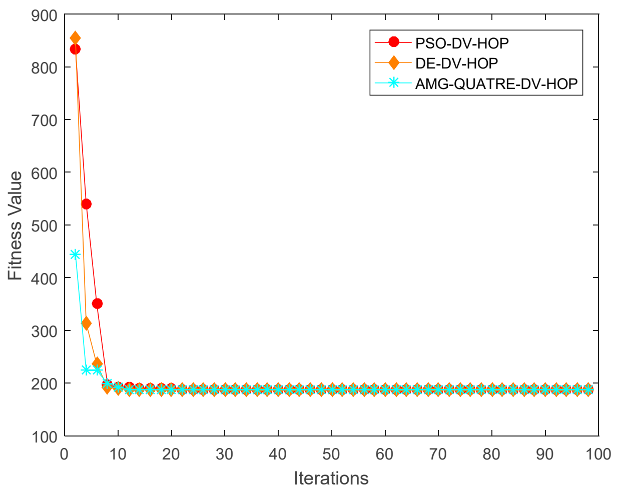

Figure 12. Comparison of convergence curve of the applied AMG-QUATRE with PSO and DE methods for single simulation (Number of the sensor node 400, Number of anchor node 40, Communication range 20).

We can see from Figure 11 and Table 8 that our proposed algorithm achieved better performance than the DV-Hop, Hyperbolic-DV-hop and PSO-DV-hop algorithms. Our proposed algorithm has similar performance with DE-DV-hop algorithm. The average localization error of our proposed algorithm was about 13.5%, 9.9% and 1.1% lower than the standard DV-Hop algorithm, Hyperbolic-DV-hop algorithm and PSO-DV-hop algorithm, respectively. It can be observed from Figure 11 that as the number of sensor nodes increased, the average localization error of all algorithms also decreased.

Figure 13 shows a summary of the average localization error for the three groups of experiments including the rate of anchor nodes, the communicating ranges, and the density of nodes. It can be seen from Figure 13 that the proposed algorithm achieved a better performance than the DV-Hop, Hyperbolic-DV-hop, PSO-DV-hop algorithms. However, it is similar to the DE-DV-hop algorithm.

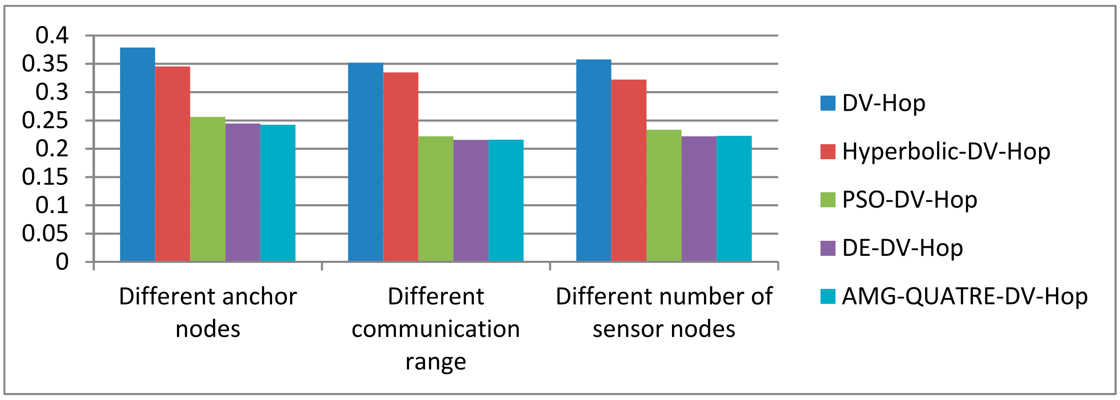

Figure 13. Comparison of average localization error of the applied AMG-QUATRE with the other methods.

5. Conclusions

In this paper, a novel adaptation multi-group QUATRE algorithm (AMG-QUATRE) was presented for the global optimization problems. In AMG-QUATRE, the population has randomly partitioned into several groups. Each group adopted a different mutation scheme to reserve population diversity and improve the efficiency of the algorithm. Also, the control parameter scale factor *F* of mutation was updated adaptively during the search process to make a trade-off between exploration and exploitation ability. The CEC2013 test suite was used to verify the performance of the proposed algorithm. The compared experimental results with the other algorithms in the literature demonstrate that the proposed AMG-QUATRE algorithm not only has better performance than three QUATRE variants but also has better performance than two DE variants and two PSO variants in terms of converging rates and quality performance.

The proposed AMG-QUATRE algorithm was also applied to enhance the node location accuracy of DV-Hop algorithm. In this application, we first refined the hop size of anchor nodes and then used AMG-QUATRE algorithm to estimate the locations of unknown nodes. We conducted several empirical scenarios of experiments such as on the different ratio of anchor nodes, diverse communication range, and different sizes of sensor networks. Simulated experimental results demonstrated that our proposed algorithm AMG-QUATRE-DV-hop achieves higher accuracy than the DV-Hop, Hyperbolic-DV-hop, and PSO-DV-hop algorithms.

In future work, we will further expand the adopted efficient schemes to improve the performance of the evolution and swarm algorithms [48,49]. We will also apply the improved algorithms to different kinds of WSN problems, e.g., clustering approaches [50–52], hierarchical routing [53], deployment, and coverage in WSN [54].

Author Contributions: Conceptualization, N.X.L. and J.-S.P.; methodology, N.X.L. and J.-S.P.; software, N.X.L.; validation, N.X.L. and J.-S.P.; formal analysis, J.-S.P., J.W.; investigation, N.X.L.; resources, J.-S.P., J.W.; data curation, N.X.L.; writing—original draft preparation, N.X.L., T.T.N.; writing—review and editing, N.X.L., J.W., T.T.N., J.W. and J.-S.P.; visualization, N.X.L.; supervision, J.-S.P.; project administration, J.-S.P.; funding acquisition, J.-S.P.

Funding: This research received no external funding.

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

References

1. Mahdavi, S.; Shiri, M.E.; Rahnamayan, S. Metaheuristics in large-scale global continues optimization: A survey. *Inf. Sci.* **2015**, *295*, 407–428. [CrossRef]
2. Beheshti, Z.; Shamsuddin, S.M.H. A review of population-based meta-heuristic algorithm. *Int. J. Adv. Soft Comput. Appl.* **2013**, *5*, 1–35.
3. Dao, T.-K.; Pan, T.-S.; Nguyen, T.-T.; Pan, J.-S. Parallel bat algorithm for optimizing makespan in job shop scheduling problems. *J. Intell. Manuf.* **2015**, *29*, 1–12. [CrossRef]
4. Nguyen, T.-T.; Pan, J.-S.; Dao, T.-K. A Compact Bat Algorithm for Unequal Clustering in Wireless Sensor Networks. *Appl. Sci.* **2019**, *9*, 1973. [CrossRef]
5. Liu, N.; Pan, J.-S.; Nguyen, T.-T. A bi-population QUasi-Affine TRansformation Evolution algorithm for global optimization and its application to dynamic deployment in wireless sensor networks. *EURASIP J. Wirel. Commun. Netw.* **2019**, *2019*, 175. [CrossRef]
6. Goldberg, D.E. Holland, J.H. Genetic Algorithms in Search, Optimization, and Machine Learning. *Mach. Learn.* **1988**, *3*, 95–99. [CrossRef]
7. Shi, Y.; Eberhart, R. A modified particle swarm optimizer. In Proceedings of the 1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360), Anchorage, AK, USA, 4–9 May 1998; pp. 69–73.
8. Storn, R.; Price, K. Differential Evolution—A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. *J. Glob. Optim.* **1997**, *11*, 341–359. [CrossRef]
9. Dorigo, M.; Di Caro, G. Ant colony optimization: A new meta-heuristic. In Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406), Washington, DC, USA, 6–9 July 1999.
10. Meng, Z.; Pan, J. HARD-DE: Hierarchical ARchive Based Mutation Strategy With Depth Information of Evolution for the Enhancement of Differential Evolution on Numerical Optimization. *IEEE Access* **2019**, *7*, 12832–12854. [CrossRef]
11. Baykasolu, A.; Oumlzbakr, L.; Tapk, P. Artificial Bee Colony Algorithm and Its Application to Generalized Assignment Problem. In *Swarm Intelligence, Focus on Ant and Particle Swarm Optimization*; IntechOpen: London, UK, 2007.
12. Yang, X.-S. Firefly Algorithm, Levy Flights and Global Optimization. In *Research and Development in Intelligent Systems XXVI*; Springer: London, UK, 2010; pp. 209–218.

13. Yang, X.-S.; Hossein Gandomi, A. Bat algorithm: a novel approach for global engineering optimization. *Eng. Comput. Int. J. Comput. Eng. Softw.* **2012**, *29*, 464–483. [CrossRef]
14. Tan, Y.; Tan, Y. Fireworks Algorithm (FWA). In *Fireworks Algorithm*; Springer: Berlin/Heidelberg, Germany, 2015; pp. 17–35.
15. Meng, Z.; Pan, J.-S.; Xu, H. QUasi-Affine TRansformation Evolutionary (QUATRE) algorithm: A cooperative swarm based algorithm for global optimization. *Knowledge-Based Syst.* **2016**, *109*, 104–121. [CrossRef]
16. Chu, S.-C.; Tsai, P.; Pan, J.-S. Cat Swarm Optimization. In *PRICAI 2006: Trends in Artificial Intelligence*; Yang, Q., Webb, G., Eds.; Springer: Berlin/Heidelberg, Germany, 2006; pp. 854–858.
17. Nguyen, T.; Pan, J.; Dao, T. An Improved Flower Pollination Algorithm for Optimizing Layouts of Nodes in Wireless Sensor Network. *IEEE Access* **2019**, *7*, 75985–75998. [CrossRef]
18. Nguyen, T.-T.; Pan, J.-S.; Wu, T.-Y.; Dao, T.-K.; Nguyen, T.-D. Node Coverage Optimization Strategy Based on Ions Motion Optimization. *J. Netw. Intell.* **2019**, *4*, 1–9.
19. Meng, Z.; Pan, J. QUasi-affine TRansformation Evolutionary (QUATRE) algorithm: The framework analysis for global optimization and application in hand gesture segmentation. In Proceedings of the 2016 IEEE 13th International Conference on Signal Processing (ICSP), Chengdu, China, 6–10 November 2016; pp. 1832–1837.
20. Meng, Z.; Pan, J. A Competitive QUasi-Affine TRansformation Evolutionary (C-QUATRE) Algorithm for global optimization. In Proceedings of the 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Budapest, Hungary, 9–12 October 2016; pp. 1644–1649.
21. Pan, J.-S.; Meng, Z.; Chu, S.-C.; Roddick, J.F. QUATRE Algorithm with Sort Strategy for Global Optimization in Comparison with DE and PSO Variants. In *Proceedings of the Fourth Euro-China Conference on Intelligent Data Analysis and Applications*; Krömer, P., Alba, E., Pan, J.-S., Snášel, V., Eds.; Springer: Cham, Switzerland, 2018; pp. 314–323.
22. Meng, Z.; Pan, J.-S. QUasi-Affine TRansformation Evolution with External ARchive (QUATRE-EAR): An enhanced structure for Differential Evolution. *Knowledge-Based Syst.* **2018**, *155*, 35–53. [CrossRef]
23. Cui, L.; Li, G.; Lin, Q.; Chen, J.; Lu, N. Adaptive Differential Evolution Algorithm with Novel Mutation Strategies in Multiple Sub-populations. *Comput. Oper. Res.* **2015**, *67*, 155–173. [CrossRef]
24. Chang, J.-F.; Chu, S.-C.; Roddick, J.F.; Pan, J.-S. A Parallel Particle Swarm Optimization Algorithm with Communication Strategies. *J. Inf. Sci. Eng.* **2005**, *21*, 809–818.
25. Tsai, P.-W.; Pan, J.-S.; Chen, S.-M.; Liao, B.-Y. Parallel Cat Swarm Optimization. In Proceedings of the Proceedings of the seventh international conference on machine learning and cybernetics, Kunming, China, 12–15 July 2008; pp. 3328–3333.
26. Angeline, P.J. Adaptive and Self-adaptive Evolutionary Computations. *Comput. Intell. A Dyn. Syst. Perspect.* **1995**, 152–163.
27. Eiben, A.E.; Hinterding, R.; Michalewicz, Z. Parameter control in evolutionary algorithms. *IEEE Trans. Evol. Comput.* **1999**, *3*, 124–141. [CrossRef]
28. Pan, J.-S.; Meng, Z.; Xu, H.; Li, X. QUasi-Affine TRansformation Evolution (QUATRE) algorithm: A new simple and accurate structure for global optimization. In Proceedings of the International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, Morioka, Japan, 2–6 August 2016; pp. 657–667.
29. Antoine-Santoni, T.; Santucci, J.-F.; De Gentili, E.; Silvani, X.; Morandini, F. Performance of a protected wireless sensor network in a fire. Analysis of fire spread and data transmission. *Sensors* **2009**, *9*, 5878–5893. [CrossRef]
30. Wang, J.; Ghosh, R.K.; Das, S.K. A survey on sensor localization. *J. Control Theory Appl.* **2010**, *8*, 2–11. [CrossRef]
31. Wang, J.; Gao, Y.; Liu, W.; Sangaiah, A.K.; Kim, H.-J. An intelligent data gathering schema with data fusion supported for mobile sink in wireless sensor networks. *Int. J. Distrib. Sens. Networks* **2019**, *15*, 1550147719839581. [CrossRef]
32. Wang, J.; Gao, Y.; Liu, W.; Sangaiah, K.A.; Kim, H.-J. An Improved Routing Schema with Special Clustering Using PSO Algorithm for Heterogeneous Wireless Sensor Network. *Sensors* **2019**, *19*, 671. [CrossRef] [PubMed]
33. Pan, J.S.; Kong, L.; Sung, T.W.; Tsai, P.W.; Snášel, V. A clustering scheme for wireless sensor networks based on genetic algorithm and dominating set. *J. Internet Technol.* **2018**, *19*, 1111–1118.
34. Peng, B.; Li, L. An improved localization algorithm based on genetic algorithm in wireless sensor networks. *Cogn. Neurodyn.* **2015**, *9*, 249–256. [CrossRef] [PubMed]

35. Hanh, N.T.; Binh, H.T.T.; Hoai, N.X.; Palaniswami, M.S. An efficient genetic algorithm for maximizing area coverage in wireless sensor networks. *Inf. Sci.* **2019**, *488*, 58–75. [CrossRef]
36. Singh, S.P.; Sharma, S.C. A PSO based improved localization algorithm for wireless sensor network. *Wirel. Pers. Commun.* **2018**, *98*, 487–503. [CrossRef]
37. Harikrishnan, R.; Kumar, V.J.S.; Ponmalar, P.S. Differential evolution approach for localization in wireless sensor networks. In Proceedings of the 2014 IEEE International Conference on Computational Intelligence and Computing Research, Coimbatore, India, 18–20 December 2014; pp. 1–4.
38. Sharma, G.; Kumar, A. Modified energy-efficient range-free localization using teaching–learning-based optimization for wireless sensor networks. *IETE J. Res.* **2018**, *64*, 124–138. [CrossRef]
39. Nguyen, T.-T.; Pan, J.-S.; Chu, S.-C.; Roddick, J.F.; Dao, T.-K. Optimization Localization in Wireless Sensor Network Based on Multi-Objective Firefly Algorithm. *J. Netw. Intell.* **2016**, *1*, 130–138.
40. Strumberger, I.; Beko, M.; Tuba, M.; Minovic, M.; Bacanin, N. Elephant Herding Optimization Algorithm for Wireless Sensor Network Localization Problem. In *Technological Innovation for Resilient Systems*; Camarinha-Matos, L.M., Adu-Kankam, K.O., Julashokri, M., Eds.; Springer: Cham, Switzerland, 2018; pp. 175–184.
41. Strumberger, I.; Minovic, M.; Tuba, M.; Bacanin, N. Performance of Elephant Herding Optimization and Tree Growth Algorithm Adapted for Node Localization in Wireless Sensor Networks. *Sensors* **2019**, *19*, 2515. [CrossRef] [PubMed]
42. Chen, X.; Zhang, B. Improved DV-Hop Node Localization Algorithm in Wireless Sensor Networks. *Int. J. Distrib. Sens. Networks* **2012**, *8*, 213980. [CrossRef]
43. Liang, J.J.; Qu, B.Y.; Suganthan, P.N.; Hernández-Díaz, A.G. Problem Definitions and Evaluation Criteria for the CEC 2013 Special Session on Real-Parameter Optimization. Technical Report 201212. Available online: https://www.al-roomi.org/multimedia/CEC_Database/CEC2013/RealParameterOptimization/CEC2013_RealParameterOptimization_TechnicalReport.pdf (accessed on 23 September 2019).
44. Rahnamayan, S.; Tizhoosh, H.R.; Salama, M.M.A. Opposition-Based Differential Evolution. *IEEE Trans. Evol. Comput.* **2008**, *12*, 64–79. [CrossRef]
45. Liang, J.J.; Qin, A.K.; Suganthan, P.N.; Baskar, S. Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. *IEEE Trans. Evol. Comput.* **2006**, *10*, 281–295. [CrossRef]
46. Cheng, R.; Jin, Y. A social learning particle swarm optimization algorithm for scalable optimization. *Inf. Sci.* **2015**, *291*, 43–60. [CrossRef]
47. Song, G.; Tam, D. Two novel DV-Hop localization algorithms for randomly deployed wireless sensor networks. *Int. J. Distrib. Sens. Networks* **2015**, *11*, 187670. [CrossRef]
48. Bui, D.T.; Moayedi, H.; Kalantar, B.; Osouli, A.; Pradhan, B.; Nguyen, H.; Rashid, A.S.A. A Novel Swarm Intelligence—Harris Hawks Optimization for Spatial Assessment of Landslide Susceptibility. *Sensors* **2019**, *19*, 3590. [CrossRef]
49. Pan, J.S.; Lee, C.Y.; Sghaier, A.; Zeghid, M.; Xie, J. Novel Systolization of Subquadratic Space Complexity Multipliers Based on Toeplitz Matrix-Vector Product Approach. *IEEE Trans. Very Large Scale Integr. Syst.* **2019**, *27*, 1614–1622. [CrossRef]
50. Wang, J.; Gao, Y.; Liu, W.; Wu, W.; Lim, S.-J. An Asynchronous Clustering and Mobile Data Gathering Schema based on Timer Mechanism in Wireless Sensor Networks. *Comput. Mater. Contin.* **2019**, *58*, 711–725. [CrossRef]
51. Wang, J.; Gao, Y.; Liu, W.; Sangaiah, A.K. Hye-Jin Kim Energy Efficient Routing Algorithm with Mobile Sink Support for Wireless Sensor Networks. *Sensors* **2019**, *19*, 1494. [CrossRef]
52. Wang, J.; Gao, Y.; Wang, K.; Sangaiah, A.K.; Lim, S.-J. An Affinity Propagation based Self-Adaptive Clustering Method for Wireless Sensor Networks. *Sensors* **2019**, *19*, 2579. [CrossRef]
53. Pan, J.-S.; Kong, L.; Sung, T.-W.; Tsai, P.-W.; Snasel, V. alpha-Fraction First Strategy for Hierarchical Wireless Sensor Networks. *J. Internet Technol.* **2018**, *19*, 1717–1726.
54. Nguyen, T.-T.; Dao, T.-K.; Kao, H.-Y.; Horng, M.-F.; Shieh, C.-S. Hybrid Particle Swarm Optimization with Artificial Bee Colony Optimization for Topology Control Scheme in Wireless Sensor Networks. *J. Internet Technol.* **2017**, *18*, 743–752.

© 2019 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).