Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2014, Article ID 975916, 14 pages
http://dx.doi.org/10.1155/2014/975916

Research Article

Adaptive Randomness: A New Population Initialization Method

Weifeng Pan,[1] Kangshun Li,[2] Muchou Wang,[3] Jing Wang,[4] and Bo Jiang[1]

[1] *School of Computer Science and Information Engineering, Zhejiang Gongshang University, Hangzhou, Zhejiang 310018, China*
[2] *College of Information, South China Agricultural University, Guangzhou, Guangdong 510642, China*
[3] *Wenzhou University Library, Wenzhou University, Wenzhou, Zhejiang 325035, China*
[4] *School of Software and Communication Engineering, Jiangxi University of Finance and Economics, Nanchang, Jiangxi 330013, China*

Correspondence should be addressed to Weifeng Pan; panweifeng1982@gmail.com

Received 2 August 2014; Accepted 15 October 2014; Published 13 November 2014

Academic Editor: Erik Cuevas

Population initialization is a crucial task in population-based optimization methods, which can affect the convergence speed and also the quality of the final solutions. Generally, if no a priori information about the solutions is available, the initial population is often selected randomly using random numbers. This paper presents a new initialization method by applying the concept of adaptive randomness (AR) to distribute the individuals as spaced out as possible over the search space. To verify the performance of AR, a comprehensive set of 34 benchmark functions with a wide range of dimensions is utilized. Conducted experiments demonstrate that AR-based population initialization performs better than other population initialization methods such as random population initialization, opposition-based population initialization, and generalized opposition-based population initialization in the convergence speed and the quality of the final solutions. Further, the influences of the problem dimensionality, the new control parameter, and the number of trial individuals are also investigated.

1. Introduction

Evolutionary algorithms (EAs) are population-based stochastic optimization algorithms. For each optimization problem, they maintain a set of candidate solutions to play the role of individuals in a population, perform crossover and mutation operations on this set to generate different solutions, and use a fitness function to determine the environment within which the solutions live. In the last decade, EAs have been applied successfully to solve many real-world and benchmark optimization problems. However, as population-based algorithms, EAs such as the genetic algorithm (GA) [1] and differential evolution (DE) [2, 3] all have common drawbacks—long computational time and premature convergence, especially when the solution space is hard to explore.

Since reducing computation time needed to reach optimal solutions and improve the quality of the final solutions would be beneficial, many efforts have been already done. However, most of the work mainly focused on the introduction and improvement of selection mechanisms, crossover and mutation operators, parameter adjustments, and some other hybrid strategies. If no information about the solution is available, the most commonly used method to generate the initial population is random initialization. Little work has been done on the population initialization, even though it is a crucial task in EAs and can affect the convergence speed and also the quality of the final solution. Maaranen et al. [4] used quasirandom sequences to generate the initial population for GAs. The experimental results showed that their approach could improve the quality of the final solutions but with no noteworthy improvement for convergence speed. Rahnamayan et al. [5] proposed an opposition-based population initialization method, which achieved a fast convergence speed. Wang et al. [6] presented a population initialization method based on space transformation search (in their following work, such a method is renamed as generalized opposition-based initialization [7]). Experimental results showed that their approach when combined with other strategies outperformed the traditional random initialization and opposition-based initialization.

This paper proposes a new approach for population initialization by employing the adaptive randomness (AR) to

improve the quality of the final solutions and also accelerate the convergence speed. AR initialization is an enhanced version of random initialization. It is simple and easy to be implemented. The main idea of AR is to make use of the difference between individuals to make them more evenly spread over the entire search space and then a better approximation for the current candidate solution is obtained. Although this paper only embeds the AR for population initialization of classical DE, the idea is general enough to be applied to all other EAs. Experimental results on 34 well-known benchmark problems show that the proposed approach performs better than the random initialization, opposition-based initialization, and generalized opposition-based initialization both in the quality of the final solutions and the convergence speed.

The remainder of this paper is organized as follows: in Section 2, the concept of AR is briefly explained. In Section 3, the classical DE is briefly reviewed. In Section 4, the proposed AR-based population initialization algorithm is presented. Experimental results are given in Section 5 with focus on the test functions used, parameter setting, results, and results' analysis. In Section 6, we conclude the work and all benchmark functions are listed in the appendix.

2. Adaptive Randomness

Traditionally, EAs imitate natural evolution in a population. The population is a set of candidate solutions to an optimization problem, making us consider several solutions at the same time. The population evolves from one generation to another as the individuals are crossbred and mutated until the predefined criteria are satisfied. If no a priori information about the solution is available, the initial population is often selected randomly using random numbers [4]. Obviously, the computation time is directly related to the distance of the random numbers from optimal solutions [5].

In practice, random numbers cannot be generated algorithmically. The algorithmically generated numbers (usually called pseudorandom numbers) only try to imitate random numbers. However, it is usually more important that the numbers are as evenly distributed as possible than that they imitate random numbers [4], for they provide much more information about the fitness function. This forms the basis of our approach for population initialization, namely, adaptive randomness (AR).

AR slightly modifies the random initialization by controlling the individuals that can come into the initial population. When adding a new individual to the initial population, AR needs to make sure that the individual should not be too close to any of the previous individuals already in the initial population.

To achieve this, AR should maintain two sets of individuals, that is, partial initial population (PP) and set of trial individuals (ST). Before concentrating on AR-based population initialization, we define the two sets first.

Definition 1. Let $P(\mathrm{ps}) = \{X_1, X_2, \ldots, X_{\mathrm{ps}}\}$ be the initial population of a specific optimization method, where ps is the population size and X_i $(i = 1, 2, \ldots, \mathrm{ps})$ is the candidate solution in a D-dimensional space. Then PP is defined by $\mathrm{PP} \subseteq P$.

Definition 2. $\mathrm{ST}(k) = \{Y_1, Y_2, \ldots, Y_k\}$ is the set of k trial individuals such that $\mathrm{ST} \cap \mathrm{PP} = \emptyset$. Each trial individual Y_i $(i = 1, 2, \ldots, k)$ is randomly chosen from the D-dimensional search space and k is the predefined number of trial individuals.

Obviously trial individuals are those individuals that are randomly generated from the search space but have not been added into PP.

To distribute the individuals in PP as spaced out as possible, when adding a new individual into PP, AR first generates k trial individuals to form ST and then the trial individual that is farthest away from all individuals in PP is selected to be added into PP. Such a process is repeated until the number of individuals in PP reaches ps.

In AR the distance between every pair of individuals $X_i(x_{i1}, x_{i2}, \ldots, x_{iD}) \in \mathrm{PP}$ and $Y_j(y_{j1}, y_{j2}, \ldots, y_{jD}) \in \mathrm{ST}$ is calculated by Euclidean distance as

$$d\left(X_i, Y_j\right) = \left\|X_i, Y_j\right\| = \sqrt{\sum_{t=1}^{D} \left(x_{it} - y_{jt}\right)^2}. \tag{1}$$

So when adding a new individual into PP the trial individual $Y_n \in \mathrm{ST}$ will be chosen such that for all $j \in \{1, 2, \ldots, k\}$,

$$\min_{1 \le i \le |\mathrm{PP}|} d\left(X_i, Y_n\right) \ge \min_{1 \le i \le |\mathrm{PP}|} d\left(X_i, Y_j\right), \tag{2}$$

where |PP| counts the number of individuals in PP and min returns the minimum value in a set of values. The rationale for using (2) is to evenly distribute the individuals in initial population by maximizing the minimum distance between the trial individual selected and the individuals already in PP.

Figure 1 gives a simple example to show you how to select the trial individual to update PP. Assume that PP already has two individuals (i.e., X_1 and X_2) and we generate ST with two (suppose that $k = 2$) trial individuals (i.e., Y_1 and Y_2). According to (1), we can obtain $d(X_1, Y_1) = 7.076722$, $d(X_2, Y_1) = 4.060788$, $d(X_1, Y_2) = 5.507268$, and $d(X_2, Y_2) = 3.2$. So $\min(\{d(X_1, Y_1), d(X_2, Y_1)\}) = d(X_2, Y_1) = 4.060788$ and $\min(\{d(X_1, Y_2), d(X_2, Y_2)\}) = d(x_2, Y_2) = 3.2$. Since $d(X_2, Y_1) > d(X_2, Y_2)$, the trial individual Y_1 will be added into PP.

Before introducing the AR-based population initialization algorithm, the classical DE is briefly reviewed in the following section.

3. Brief Review of the Classical DE

DE, firstly proposed by Storn and Price in [2, 3], is a population-based stochastic optimization algorithm and has been successfully used in both benchmark test functions and real-world applications. It is simple yet effective and robust. A plethora of experimental studies show its better performance than other EAs.

The proposed algorithm is also based on this DE scheme. Let us assume that $X_i(t)$ $(i = 1, 2, \ldots, \mathrm{ps})$ is the ith individual

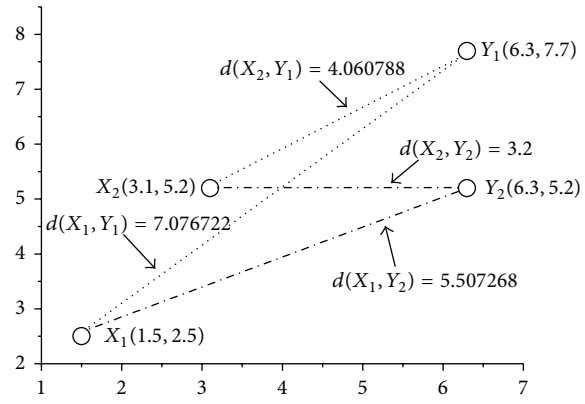

Figure 1: Illustration of the selection of a trial individual into PP in a two-dimensional space.

in population $P(t)$, where ps is the population size, t is the generation index, and $P(t)$ is the population in the ith generation. The main idea of DE is to generate trial vectors. Mutation and crossover are used to produce new trial vectors, and selection determines which of the vectors will be successfully selected into the next generation.

For classical DE (DE/rand/1/bin), the mutation, crossover, and selection operators can be defined as follows.

Mutation. For each vector $X_i(t)$ in generation t, a mutant vector $V_i(t+1)$ is defined by

$$V_i(t+1) = X_{i1}(t) + F\left(X_{i2}(t) - X_{i3}(t)\right) \quad i \neq i_1 \neq i_2 \neq i_3, \tag{3}$$

where $i = 1, 2, \ldots, \text{ps}$ and i_1, i_2, and i_3 are randomly selected integer indices from $[1, \text{ps}]$. Further, $i \neq i_1 \neq i_2 \neq i_3$, so the population size ps satisfies $\text{ps} \geq 4$. $F \in [0, 2]$ is a real number which determines the amplification of the differential variation $(X_{i2}(t) - X_{i3}(t))$. Larger values of F result in higher diversity in the generated population and lower values in faster convergence.

Crossover. As many other EAs do, DE also takes a crossover operation to increase the diversity of population and generate the trial vectors. The trial vector can be defined as

$$U_i(t+1) = \left(U_{i1}(t+1), U_{i2}(t+1), \ldots, U_{iD}(t+1)\right), \tag{4}$$

where D is the problem dimension. The classical DE uses the DE/rand/1/bin scheme to generate the trial vector

$$U_{ij}(t+1) = \begin{cases} V_{ij}(t+1), & \text{if } \text{rand}_j(0,1) \leq \text{CR} \\ X_{ij}(t), & \text{otherwise,} \end{cases} \tag{5}$$

where $\text{CR} \in [0, 1]$ is the predefined crossover probability, $\text{rand}_j(0, 1)$ is a random number within $[0, 1]$ for the jth dimension, and $j \in \{1, 2, \ldots, D\}$ is a random parameter index.

Selection. A greedy selection mechanism is used as

$$X_i(t+1) = \begin{cases} U_i(t+1), & \text{if } f\left(U_i(t+1)\right) \leq f\left(X_i(t)\right) \\ X_i(t), & \text{otherwise,} \end{cases} \tag{6}$$

where $f(\cdot)$ is the fitness function. Without loss of generality, this paper only considers minimization problems. If, and only if, the trial vector $U_i(t+1)$ is better than $X_i(t)$ (i.e., $f(U_i(t+1)) \leq f(X_i(t))$), $X_i(t+1)$ is set to $U_i(t+1)$; otherwise, the $X_i(t+1)$ remains unchanged; that is, $X_i(t+1) = X_i(t)$. Hence the population either gets better or remains the same with respect to the fitness function but never deteriorates.

Though there are many variants of DE [2, 3], to maintain a general comparison, this paper only uses the classical DE in the conducted experiments to demonstrate the improvement of the convergence speed and the quality of the final solutions by using AR-based population initialization.

4. The Proposed AR-Based Population Initialization Algorithm

For a specific optimization problem, when lacking a priori information about the solutions, the initial population is usually created using random numbers. AR makes full use of the distance information during the process of population initialization. By applying the AR strategy, we can distribute the individuals as spaced out as possible and obtain a better approximation for the current candidate solutions. So instead of using a pure random initialization, we propose the following AR-based population initialization algorithm (see Algorithm 1).

As is shown in Algorithm 1, PP is initially empty and the first individual of PP is randomly chosen from the search space. During population initialization, PP will be incrementally updated with the individuals selected from ST until the number of individuals in PP reaches the population size ps.

The flowcharts of DE with random population initialization, opposition-based population initialization, generalized opposition-based population initialization, and AR-based population initialization are shown in Figure 2.

AR-based population initialization will be embedded in the classical DE in Section 5 to show its effectiveness in the improvement of the convergence speed and the quality of the final solutions.

5. Empirical Study

To investigate the effectiveness of the proposed AR-based population initialization algorithm in the improvement of convergence speed and the final solution, we embedded it in the classical DE and conducted controlled experiments. Our experiments were carried out on a PC at 2.3 GHz with 2 GB of RAM.

In the following subsections, we provide details on the test functions of our study (Section 5.1), parameter settings

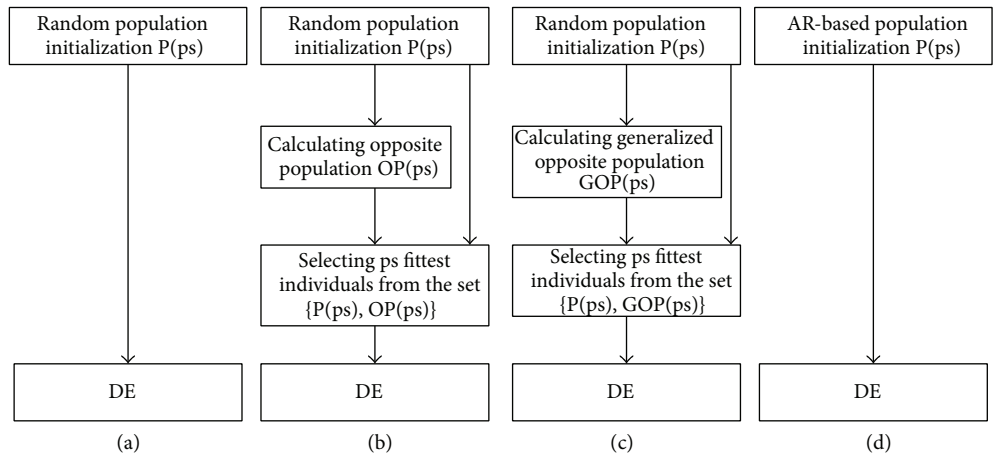

Figure 2: DE with (a) random population initialization, (b) opposition-based population initialization, (c) generalized opposition-based population initialization, and (d) AR-based population initialization.

(1) Set PP = ∅, ST = ∅, pps = 0
(2) Randomly generate an individual X from a D-dimensional search space
(3) PP = PP ∪ $\{X\}$
(4) pps++
(5) **while** pps ≤ ps **do**
(6) Randomly initialize ST = $\{Y_1, Y_2, \ldots, Y_k\}$ with k trial individuals
(7) Select the trial individual Y_n such that for all $j \in \{1, 2, \ldots, k\}$: $\min_{1 \le i \le |PP|} d(X_i, Y_n) \ge \min_{1 \le i \le |PP|} d(X_i, Y_j)$
(8) PP = PP ∪ $\{Y_n\}$
(9) pps++
(10) Set ST = ∅
(11) **end while**
(12) **return** Set the final PP as the initial population P

Algorithm 1: AR-based population initialization algorithm.

(Section 5.2), and our experiment results and analysis (Section 5.3).

5.1. Test Functions. In order to compare the convergence speed and the quality of the final solutions of DE with random population initialization (DE_r), DE with opposition-based population initialization (DE_o), DE with generalized opposition-based population initialization (DE_{go}), and DE with AR-based population initialization (DE_{ar}), a comprehensive test set with 34 numerical benchmark functions is employed. The test set includes well-known unimodal as well as highly multimodal minimization problems [8, 9]. The definition, the range of search space, and also the global optimum(s) for each function are given in the appendix. The dimensionality of these problems varies from 2 to 100, covering a wide range of problem complexity.

5.2. Parameter Settings. For all the conducted experiments, the parameters of the classical DE, namely, population size (ps), differential amplification factor (F), crossover probability constant (CR), and maximum number of function calls (MAX_{NFC}), if no a change is mentioned, are fixed to 100, 0.5, 0.9, and 10^6, respectively. Such a setting follows the suggestions given in the literature (e.g., [10–12]). And the parameter k of ST in AR-based population initialization is set to 3 unless a change is mentioned.

5.3. Results and Analysis. The experiments are categorized as follows. In Section 5.3.1, DE_r, DE_o, DE_{go}, and DE_{ar} are compared in terms of convergence speed and robustness. In Section 5.3.2, DE_r, DE_o, DE_{go}, and DE_{ar} are compared in terms of the quality of the final solution. In Section 5.3.3, the effect of problem dimensionality is investigated. In Section 5.3.4, the effect of parameter k is studied. All the experiments are conducted 1,000 times with different random seeds, and the average results throughout the optimization runs are recorded. It should be noted that, in the experiments, we find that for a small value of conducted times, the values of evaluation times and the final solutions are not stable.

5.3.1. Comparison of DE_r, DE_o, DE_{go}, and DE_{ar} in Terms of Convergence Speed and Robustness. By the suggestions in the literature (e.g., [5, 6, 9]), we compare the convergence speed of DE_r, DE_o, DE_{go}, and DE_{ar} by measuring the number of function calls (NFC). For each optimization problem, NFC is recorded when a specific algorithm reduces the best value to a value smaller than the value-to-reach (VTR) before meeting MAX_{NFC}. In order to minimize the effect of the stochastic nature of the algorithms, all the reported NFCs are averaged over 1,000 independent trials. Obviously, a smaller NFC means a higher convergence speed. In order to compare the convergence speed between two specific algorithms, we introduce another metric, acceleration rate (ARE), which is defined as

$$\text{ARE} = \frac{\text{NFC}_{\text{algA}}}{\text{NFC}_{\text{algB}}}, \tag{7}$$

where NFC_{algA} and NFC_{algB} are the NFCs for the two algorithms algA and algB (algA and algB are all chosen from $\{\text{DE}_r, \text{DE}_o, \text{DE}_{go}, \text{DE}_{ar}\}$). So ARE $>$ 1 means that algB is faster. The VTR is set to 10^{-6} for all benchmark functions. The same setting has been used in the literature (e.g., [13, 14]).

We also compare the robustness of DE_r, DE_o, DE_{go}, and DE_{ar} by measuring the success rate (SR) [13]. In the current work, a successful running means that a specific algorithm successfully reaches the VTR for each test function in the allowed MAX_{NFC}. So SR can be calculated as

$$\text{SR} = \frac{\text{number of times reached VTR}}{\text{total number of trials}}. \tag{8}$$

SR is a commonly used metric to characterize the robustness of a specific algorithm; that is, a larger SR means that the algorithm is more robust.

Further the average NFC (NFC_{avg}), the average SR (SR_{avg}), and the average ARE (ARE_{avg}) over the n test functions are calculated as

$$\begin{aligned} \text{NFC}_{\text{avg}} &= \frac{1}{n}\sum_{i=1}^{n}\text{NFC}_i, \\ \text{ARE}_{\text{avg}} &= \frac{1}{n}\sum_{i=1}^{n}\text{ARE}_i, \\ \text{SR}_{\text{avg}} &= \frac{1}{n}\sum_{i=1}^{n}\text{SR}_i. \end{aligned} \tag{9}$$

Table 1 summarizes the numerical results when solving the 34 benchmark functions shown in the appendix. The best result of NFC for each function is highlighted in boldface and NFC_{avg}, SR_{avg}, and ARE_{avg} are shown in the last row. Since comparing the algorithms with different SR values seems meaningless, the reported average values are calculated only on the functions where all the algorithms have the same success rate.

As seen, DE_{ar} outperforms DE_r on 22 test functions (about 64.7% of the problems). Though DE_r is faster than DE_{ar} on 4 functions (i.e., f_6, f_{16}, f_{26}, and f_{31}), its SR is worse. Except for f_6, the rest 3 functions are functions with a low dimensionality ($D \leq 10$). DE_{ar} outperforms DE_o on 22 test functions (about 64.7% of the problems), while DE_o surpasses DE_{ar} only on 4 functions (i.e., f_6, f_{18}, f_{26}, and f_{29}). Though DE_o is faster than DE_{ar} on f_6 and f_{26}, its SR is worse. DE_{ar} outperforms DE_{go} on 24 test functions (about 70.6% of the problems), while DE_{go} surpasses DE_{ar} only on f_4 and f_6. Though DE_{go} is faster than DE_{ar} on f_6, its SR is worse. All the algorithms fail to solve 6 functions (i.e., f_{19}, f_{21}, f_{22}, f_{23}, f_{24}, and f_{25}). On f_{15} and f_{30}, only DE_{ar} can solve them with a very small ARE while the other three algorithms all fail. The ARE_{avg} between DE_r and DE_{ar} is 1.0116, which means that DE_{ar} is on average 1.16% faster than DE_r. Similarly, DE_{ar} is on average 1.52% faster than DE_r and DE_{ar} is on average 1.38% faster than DE_{go}.

So we can conclude that DE_{ar} shows better convergence speed than the other 3 algorithms with the same parameter settings and fixing the k for the DE_{ar} ($k = 3$). Some sample bar charts for the performance comparison of the 4 algorithms are given in Figure 3.

5.3.2. Comparison of DE_r, DE_o, DE_{go}, and DE_{ar} in Terms of the Quality of Final Solutions. In this section, DE_{ar} is compared with DE_r, DE_o, and DE_{go} with respect to the quality of the final solutions. All the experiments were conducted 1,000 times, and the mean function error value and standard deviation of the results are recorded. The results of the 4 DE algorithms on the 34 test problems are presented in Tables 2 and 3, where "Mean" indicates the mean function error value and "Std. Dev." stands for the standard deviation. The best results among the 4 DE algorithms are shown in boldface.

From the results, it can be seen that DE_{ar} achieves better results than DE_r, DE_o, and DE_{go} on 24 test functions (about 70.5% of the test functions). DE_{ar} achieves the same performance as the DE_{go} on function f_{26}, and they are both much better than the other algorithms on this problem. For the rest of the 9 functions, all the algorithms achieve the same results.

To compare the performance of multiple algorithms on the test suite, the average ranking of the Friedman test is conducted by the suggestions considered in [7, 15]. Table 4 shows the average ranking of the 4 DE algorithms on functions f_1–f_{34}. These algorithms can be sorted by average ranking into the following order: DE_{ar}, DE_r, DE_{go}, and DE_o. It means that DE_{ar} and DE_o are the best and worst ones among the four algorithms, respectively. So as seen, although opposition-based population initialization can accelerate the convergence speed on some test problems, when compared with DE_r, it cannot improve the quality of the final solutions. So if we want to obtain a high quality solution, opposition-based population initialization cannot be used alone.

To investigate the significant differences between the behavior of two algorithms, we conduct four tests, that is, Nemenyi's, Holm's, Shaffer's, and Bergmann-Hommel's [7, 15]. For each test, we calculate the adjusted P values on pairwise comparisons of all algorithms. Table 5 shows the results of adjusted P values. Under the null hypothesis, the two algorithms are equivalent. If the null hypothesis is rejected, then the performances of these two algorithms are

Table 1: Comparison of convergence speed (NFC) and success rate (SR) for DE with random population initialization (DE_r), with opposition-based population initialization (DE_o), with generalized opposition-based population initialization (DE_{go}), and with AR-based population initialization (DE_{ar}).

F	D	DE_r		DE_o		DE_{go}		DE_{ar}		ARE		
		NFC	SR	NFC	SR	NFC	SR	NFC	SR	$NFC_{DE_r}/NFC_{DE_{ar}}$	$NFC_{DE_o}/NFC_{DE_{ar}}$	$NFC_{DE_{go}}/NFC_{DE_{ar}}$
f_1	30	49725	1	49670	1	49510	1	**49505**	1	1.0044	1.0033	1.0001
f_2	30	55650	1	55735	1	55620	1	**55530**	1	1.0022	1.0037	1.0016
f_3	20	121666	1	120482	1	121136	1	**120314**	1	1.0112	1.0014	1.0068
f_4	30	327670	1	327185	1	**324500**	1	326530	1	1.0035	1.0020	0.9938
f_5	10	230525	0.95	254545	0.95	237175	0.95	**227935**	0.95	1.0114	1.1167	1.0405
f_6	30	63895	0.98	62490	0.96	**62148**	0.96	64725	**1**	0.9872	0.9655	0.9602
f_7	30	11751	1	11645	1	11698	1	**11621**	1	1.0112	1.0021	1.0066
f_8	30	97624	1	97816	1	97437	1	**97411**	1	1.0022	1.0042	1.0003
f_9	2	3180	1	3150	1	3193	1	**3137**	1	1.0137	1.0041	1.0179
f_{10}	4	13272	1	13470	1	13544	1	**13255**	1	1.0013	1.0162	1.0218
f_{11}	2	15250	1	15488	1	15380	1	**15220**	1	1.0020	1.0176	1.0105
f_{12}	2	4325	1	4475	1	4410	1	**4215**	1	1.0261	1.0574	1.0463
f_{13}	30	8164	1	8157	1	8153	1	**8129**	1	1.0043	1.0034	1.0030
f_{14}	2	2681	1	2705	1	2716	1	**2678**	1	1.0011	1.0101	1.0142
f_{15}	4	—	0	—	0	—	0	**1541**	**0.01**	—	—	—
f_{16}	10	**33032**	0.24	53652	0.36	50060	0.32	49760	**0.36**	0.6638	1.0782	1.0060
f_{17}	30	250184	1	250848	1	251408	1	**249712**	1	1.0019	1.0045	1.0068
f_{18}	30	102236	1	**101268**	1	102068	1	101344	1	1.0088	0.9993	1.0071
f_{19}	30	—	0	—	0	—	0	—	0	—	—	—
f_{20}	30	22485	1	22370	1	22402	1	**22331**	1	1.0069	1.0017	1.0032
f_{21}	30	—	0	—	0	—	0	—	0	—	—	—
f_{22}	4	—	0	—	0	—	0	—	0	—	—	—
f_{23}	4	—	0	—	0	—	0	—	0	—	—	—
f_{24}	4	—	0	—	0	—	0	—	0	—	—	—
f_{25}	4	—	0	—	0	—	0	—	0	—	—	—
f_{26}	2	7980	0.97	**7923**	0.96	8136	0.98	8061	**0.98**	0.9900	0.9829	1.0093
f_{27}	2	856	1	859	1	839	1	**801**	1	1.0687	1.0724	1.0474
f_{28}	30	172487	1	172489	1	177257	1	**171543**	1	1.0055	1.0055	1.0333
f_{29}	2	4952	1	**4781**	1	4924	1	4845	1	1.0221	0.9868	1.0163
f_{30}	5	—	0	**21**	**0.02**	—	0	—	0	—	—	—
f_{31}	5	**28396**	0.99	29204	1	28974	1	28903	1	0.9825	1.0104	1.0025
f_{32}	2	1567	1	1558	1	1564	1	**1549**	1	1.0116	1.0058	1.0097
f_{33}	2	6386	1	6305	1	6309	1	**6230**	1	1.0250	1.0120	1.0127
f_{34}	2	4110	1	4083	1	4086	1	**4069**	1	1.0101	1.0034	1.0042
Average values	—	68488	0.997727	69504	0.997727	68879	0.997727	**68087**	0.997727	1.0116	1.0152	1.0138

Note. The reported average values (the last row) are calculated only on the functions where all the algorithms have the same success rate (i.e., all SRs = 1 and one SR = 0.95).

Table 2: Comparison among the 4 DE algorithms on test problems f_1–f_{22}, where "Mean" indicates the mean function error value and "Std. Dev." stands for the standard deviation. The best results among the four algorithms are shown in boldface.

	D		DE_r	DE_o	DE_{go}	DE_{ar}
f_1	30	Mean	8.81228e − 009	8.81301e − 009	8.79674e − 009	**8.77417e − 009**
		Std. Dev.	1.10323e − 009	8.84418e − 010	1.08043e − 009	8.7124e − 010
f_2	30	Mean	9.04527e − 009	9.10281e − 009	9.06893e − 009	**8.92437e − 009**
		Std. Dev.	7.96893e − 010	6.70026e − 010	7.70684e − 010	7.79631e − 010
f_3	20	Mean	9.34818e − 009	9.39706e − 009	9.38949e − 009	**9.33293e − 009**
		Std. Dev.	5.33216e − 010	4.83679e − 010	5.58047e − 010	5.48986e − 010
f_4	30	Mean	9.26684e − 009	9.32035e − 009	9.3298e − 009	**9.11962e − 009**
		Std. Dev.	7.17271e − 010	6.50894e − 010	5.82968e − 010	7.65305e − 010
f_5	10	Mean	0.049748	0.049748	0.049748	0.049748
		Std. Dev.	0.22248	0.22248	0.22248	0.22248
f_6	30	Mean	0.000172542	0.00029585	0.000320462	**8.80458e − 009**
		Std. Dev.	0.00122635	0.00146403	0.00159228	1.20596e − 009
f_7	30	Mean	6.24585e − 009	6.62518e − 009	6.86516e − 009	**6.20522e − 009**
		Std. Dev.	2.32947e − 009	2.28414e − 009	2.42006e − 009	2.35472e − 009
f_8	30	Mean	9.3102e − 009	9.33032e − 009	9.2814e − 009	**9.25572e − 009**
		Std. Dev.	4.90595e − 010	5.23764e − 010	6.79597e − 010	5.84208e − 010
f_9	2	Mean	4.46327e − 009	4.39462e − 009	4.67599e − 009	**4.31941e − 009**
		Std. Dev.	2.95843e − 009	3.02071e − 009	2.72012e − 009	2.98226e − 009
f_{10}	4	Mean	6.27386e − 009	6.25378e − 009	6.40042e − 009	**5.92566e − 009**
		Std. Dev.	2.57066e − 009	2.69058e − 009	2.27469e − 009	2.58577e − 009
f_{11}	2	Mean	4.72646e − 009	4.63412e − 009	4.47891e − 009	**3.90594e − 009**
		Std. Dev.	3.03952e − 009	2.74639e − 009	3.01052e − 009	2.94237e − 009
f_{12}	2	Mean	4.65101e − 008	4.65101e − 008	4.65101e − 008	4.65101e − 008
		Std. Dev.	0	0	0	0
f_{13}	30	Mean	0	0	0	0
		Std. Dev.	0	0	0	0
f_{14}	2	Mean	5.04676e − 009	4.58853e − 009	4.67473e − 009	**4.51194e − 009**
		Std. Dev.	2.87757e − 009	3.1021e − 009	2.83989e − 009	2.96938e − 009
f_{15}	4	Mean	0.00967981	0.0121192	0.0154611	**0.0047311**
		Std. Dev.	0.0461986	0.048523	0.0506461	0.0142838
f_{16}	10	Mean	0.0320855	0.0287587	0.0247149	**0.0239915**
		Std. Dev.	0.023079	0.0368454	0.0268588	0.0306438
f_{17}	30	Mean	9.16118e − 009	9.11199e − 009	9.30319e − 009	**9.10653e − 009**
		Std. Dev.	8.92376e − 010	8.15186e − 010	7.58129e − 010	8.8449e − 010
f_{18}	30	Mean	9.34244e − 009	9.38564e − 009	9.42362e − 009	**9.22727e − 009**
		Std. Dev.	5.34739e − 010	4.09769e − 010	4.70872e − 010	7.64185e − 010
f_{19}	30	Mean	1.73273	2.07072	1.40319	**1.38389**
		Std. Dev.	1.11967	2.26245	1.01423	1.17433
f_{20}	30	Mean	0	0	0	0
		Std. Dev.	0	0	0	0
f_{21}	30	Mean	0.000697643	0.000727313	0.000740714	**0.000656965**
		Std. Dev.	0.00020979	0.000291491	0.000261093	0.0001786
f_{22}	4	Mean	0.00405371	0.00405371	0.00405371	0.00405371
		Std. Dev.	8.89894e − 019	8.44228e − 019	8.89894e − 019	8.44228e − 019

Table 3: Comparison among the 4 DE algorithms on test problems f_{23}–f_{34}, where "Mean" indicates the mean function error value and "Std. Dev." stands for the standard deviation. The best results among the four algorithms are shown in boldface.

	D		DE_r	DE_o	DE_{go}	DE_{ar}
f_{23}	4	Mean	10.1876	10.1876	10.1876	10.1876
		Std. Dev.	5.46751e − 015	5.46751e − 015	5.31347e − 015	5.31347e − 015
f_{24}	4	Mean	10.4503	10.4503	10.4503	10.4503
		Std. Dev.	1.8225e − 015	1.8225e − 015	1.8225e − 015	1.8225e − 015
f_{25}	4	Mean	10.6103	10.6103	10.6103	10.6103
		Std. Dev.	1.8225e − 015	1.8225e − 015	1.8225e − 015	1.8225e − 015
f_{26}	2	Mean	0.03	0.04	0.02	0.02
		Std. Dev	0.171447	0.196946	0.140705	0.140705
f_{27}	2	Mean	2.72741e − 009	2.94925e − 009	2.78605e − 009	**2.63247e − 009**
		Std. Dev.	2.76876e − 009	2.8458e − 009	2.79248e − 009	2.76724e − 009
f_{28}	30	Mean	9.28689e − 009	9.28566e − 009	9.28304e − 009	**9.27877e − 009**
		Std. Dev.	6.31616e − 010	5.9591e − 010	6.40915e − 010	6.07695e − 010
f_{29}	2	Mean	4.63641e − 009	4.15234e − 009	4.89748e − 009	**3.94368e − 009**
		Std. Dev.	3.24716e − 009	3.01035e − 009	2.81979e − 009	2.7603e − 009
f_{30}	5	Mean	0.0452575	0.0437805	0.043716	**0.0430663**
		Std. Dev.	0.0220521	0.0262655	0.0233605	0.0220867
f_{31}	5	Mean	0.00523817	6.93483e − 009	7.14794e − 009	**6.31104e − 009**
		Std. Dev.	0.0523817	2.25948e − 009	2.13661e − 009	2.19293e − 009
f_{32}	2	Mean	0	0	0	0
		Std. Dev.	0	0	0	0
f_{33}	2	Mean	4.70104e − 009	4.89609e − 009	4.7038e − 009	**4.44655e − 009**
		Std. Dev.	2.89836e − 009	2.82738e − 009	3.08112e − 009	2.68808e − 009
f_{34}	2	Mean	4.75055e − 009	4.73653e − 009	4.52236e − 009	**4.23427e − 009**
		Std. Dev.	2.96249e − 009	3.10385e − 009	3.18058e − 009	2.89249e − 009

Table 4: Average ranking of the 4 DE algorithms.

Algorithm	Ranking
DE_{ar}	3.5588235294117663
DE_r	2.205882352941176
DE_{go}	2.1470588235294117
DE_o	2.0882352941176463

significantly different. In this paper, we only discuss whether the hypotheses is rejected at the 0.05 level of significance. As we can see, all the four tests reject hypotheses 1–3.

Besides the above four tests, we also conduct Wilcoxon's test to recognize significant differences between the behavior of two algorithms [7, 15]. Table 6 shows the P values of applying Wilcoxon's test among DE_{ar} and the other three DE algorithms. The P values below 0.05 (the significant level) are shown in boldface. From the results, it can be seen that DE_{ar} is significantly better than DE_r, DE_o, and DE_{go}.

5.3.3. Scalability Test: Effect of Problem Dimensionality. The performance of most EAs (including DE) deteriorates quickly with the growth of the dimensionality of the problem. The main reason is that, in general, the complexity of the problem (search space) increases exponentially with its dimension. Here, we show a scalable test of DE_{ar} for $D/2$, D, and $2D$ for each scalable function in our test set. Table 7 summarizes the comparison results of the four DE algorithms for $D/2$, D, and $2D$. From the results, it can be seen that DE_{ar} is not always affected by the growth of dimensionality. For f_1, f_2, f_3, f_7, f_8, f_{17}, f_{18}, f_{21}, and f_{28}, DE_{ar} achieves similar performance when the dimension increases from $D/2$ to $2D$. The performance of DE_{ar} deteriorates quickly with the growth of dimension for five functions (f_4, f_5, f_6, f_{19}, and f_{20}). For the remaining one function (f_{13}), the growth of dimension does not affect the performance of DE_{ar}.

5.3.4. Effect of Different k Settings. In DE_{ar}, a new control parameter k (the number of trial individuals) is added to DE's parameters (ps, F, and CR). Since k denotes the number of trial individuals, k should be a positive integer. And when $k = 1$, DE_{ar} is equal to DE_r. So in our study we restrict k to the positive integers within the range of [2, ps]. As mentioned above, k was fixed to 3 in all experiments. Such a value was set without any effort to find an optimal value. However the performance of DE_{ar} may be influenced by different settings of k values.

We investigate the correlation between k and the quality of the final solutions using the Spearman correlation [16]. We repeat the conducted experiments in Section 5.3.3 for $k \in [10, 100]$ (since ps = 100) with step size of 10 (i.e., 1,000 trials per function per $k \in \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$).

Table 5: Adjusted P values.

i	Hypothesis	Unadjusted P	P_{Neme}	P_{Holm}
1	DE_o versus DE_{ar}	$2.6442139326214958e-6$	$1.5865283595728974e-5$	$1.5865283595728974e-5$
2	DE_{go} versus DE_{ar}	$6.519075482494892e-6$	$3.9114452894969356e-5$	$3.2595377412474464e-5$
3	DE_r versus DE_{ar}	$1.5536056185614817e-5$	$9.32163371136889e-5$	$6.214422474245927e-5$
4	DE_r versus DE_o	0.7071142312899601	4.24268538773976	2.12134269386988
5	DE_o versus DE_{go}	0.8509806870320539	5.1058841221923235	2.12134269386988
6	DE_r versus DE_{go}	0.8509806870320561	5.105884122192337	2.12134269386988
i	Hypothesis	P_{Shaf}	P_{Berg}	
1	DE_o versus DE_{ar}	$1.5865283595728974e-5$	$1.5865283595728974e-5$	
2	DE_{go} versus DE_{ar}	$1.9557226447484678e-5$	$1.9557226447484678e-5$	
3	DE_r versus DE_{ar}	$4.660816855684445e-5$	$3.1072112371229634e-5$	
4	DE_r versus DE_o	2.12134269386988	2.12134269386988	
5	DE_o versus DE_{go}	2.12134269386988	2.12134269386988	
6	DE_r versus DE_{go}	2.12134269386988	2.12134269386988	

Table 6: Wilcoxon test between DE_{ar} and the other three DE algorithms on functions f_1–f_{34}. The P values below 0.05 are shown in boldface.

DE_{ar} versus	P values
DE_r	**0.000012**
DE_{go}	**0.000018**
DE_o	**0.000012**

For the limitation of space, we do not show all the results of the final solutions; only the final solutions obtained on f_2 and f_3 are shown in Table 8 for illustration. But almost a similar behavior has been observed for all functions that the quality of the final solution is better than that of DE_r, DE_o, and DE_{go} when $k > 1$.

Table 9 shows the Spearman correlation test results between k and the final solutions obtained on each test function. As seen, there is not a significant correlation between k and the quality of the final solutions. It means that k, like other control parameters of DE, has a problem-oriented value. Since the larger the k value is, the more time to initialize the population is required, especially when the dimensionality of the problem is also large. Our limited experiments suggest to use a small value of k.

6. Conclusions

This paper employs the concept of adaptive randomness (AR) for population initialization. The main idea of AR is to make use of the difference between individuals to make them more evenly spread over the search space and then a better approximation for the current candidate solution is obtained. In order to investigate the performance of the AR-based population initialization, the classical DE has been utilized. By embedding AR within DE, DE_{ar} was proposed. Experiments are conducted on 34 benchmark functions. The experimental results can be summarized as follows.

(i) DE_{ar} is compared with other DE algorithms such as DE with random initialization (DE_r), DE with opposition-based learning (DE_o), and DE with generalized opposition-based learning (DE_{go}) with respect to the convergence speed and robustness. The results demonstrate that DE_{ar} performs better than the other three DE algorithms at least on 64.7% of the test functions. Although the other three DE algorithms outperform DE_{ar} on some functions, their success rates are always worse.

(ii) DE_{ar} is further compared with DE_r, DE_o, and DE_{go} with respect to the quality of the final solutions. The results show that DE_{ar} performs better than the other three DE algorithms on the majority (about 70.5%) of test functions. And on the rest of functions, DE_{ar} obtains no worse results than the results of the other three DE algorithms. Statistical comparisons also show that DE_{ar} is the best of the four DE algorithms.

(iii) A scalability test of DE_{ar} over 15 test functions with different problem dimensions ($D/2$, D, and $2D$) are conducted. The 15 functions are scalable and chosen from the 34 test functions. The results show that DE_{ar} is not always affected by the growth of dimensionality. For 9 functions, DE_{ar} achieves similar performance when the dimension increases from $D/2$ to $2D$. For 5 functions, the performance of DE_{ar} deteriorates quickly with the growth of dimension while for the remaining 1 function, the growth of dimension does not affect the performance of DE_{ar}.

(iv) The influence of k (number of trial individuals) was studied by investigating the Spearman correlation between k and the quality of the final solutions. The results obtained on the 34 test functions show that there is not a significant correlation between k and the quality of the final solutions. But the quality of the final solution is better than that of the other three DE algorithms when $k > 1$.

The main motivation of the current work was the introduction of the concept of adaptive randomness for population initialization. Although this paper only embeds the AR within

Table 7: Mean function error values of DE_{ar} for $D/2$, D, and $2D$ for each scalable function in our test set.

F	DE_{ar}			F	DE_{ar}		
	$D/2$	D	$2D$		$D/2$	D	$2D$
f_1	8.22347e − 009	8.77417e − 009	9.09346e − 009	f_{13}	0	0	0
f_2	8.27444e − 009	8.92437e − 009	9.25353e − 009	f_{17}	8.20248e − 009	9.10653e − 009	7.33223e − 006
f_3	8.33435e − 009	9.33293e − 009	9.3625e − 009	f_{18}	9.14822e − 009	9.22727e − 009	9.65751e − 009
f_4	8.30962e − 009	9.11962e − 009	1.50496	f_{19}	9.05241e − 009	1.38389	11.5557
f_5	7.0724e − 009	0.049748	2.53715	f_{20}	0	0	0.35
f_6	8.04869e − 009	8.80458e − 009	0.000369811	f_{21}	0.000222243	0.000656965	0.00434487
f_7	5.81354e − 009	6.20522e − 009	6.63678e − 009	f_{28}	8.92662e − 009	9.27877e − 009	9.54968e − 009
f_8	9.00816e − 009	9.25572e − 009	9.64823e − 009				

Note. The values of D for each test function are shown as those in Table 1.

Table 8: Final solutions obtained on f_2 and f_3 on interval [10, 100] of k with step size of 10.

	$k = 10$	$k = 20$	$k = 30$	$k = 40$	$k = 50$
f_2	8.78572e − 009	8.85414e − 009	8.77919e − 009	8.7689e − 009	8.79035e − 009
f_3	8.67476e − 009	8.8475e − 009	8.83617e − 009	8.85011e − 009	8.79553e − 009
	$k = 60$	$k = 70$	$k = 80$	$k = 90$	$k = 100$
f_2	8.83882e − 009	8.64561e − 009	8.75898e − 009	8.73418e − 009	8.8088e − 009
f_3	8.74038e − 009	8.8981e − 009	8.81404e − 009	8.76988e − 009	8.80807e − 009

classical DE, the idea is general enough to be applied to all other population-based methods (e.g., GA and PSO). Further since AR is new, studies are still required to investigate its benefits, weaknesses, and limitations. The current work can be considered as a first step in applying AR.

Appendix

List of Benchmark Functions

The 34 test functions we employed are given below. All the functions used in this paper are to be minimized.

(1) Sphere Model. Consider

$$f_1(X) = \sum_{i=1}^{n} x_i^2, \tag{A.1}$$

where $x_i \in [-5.12, 5.12]$ and the global optimum is 0 at $(0, 0, \ldots, 0)$. f_1 is a unimodal, scalable, convex, and easy function.

(2) Axis Parallel Hyperellipsoid. Consider

$$f_2(X) = \sum_{i=1}^{n} i x_i^2, \tag{A.2}$$

where $x_i \in [-5.12, 5.12]$ and the global optimum is 0 at $(0, 0, \ldots, 0)$. f_2 is a unimodal, scalable, convex, and easy function.

(3) Schwefel's Problem 1.2. Consider

$$f_3(X) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2, \tag{A.3}$$

where $x_i \in [-65, 65]$ and the global optimum is 0 at $(0, 0, \ldots, 0)$. f_3 is a unimodal and scalable function.

(4) Rosenbrock's Valley. Consider

$$f_4(X) = \sum_{i=1}^{n-1} \left[100 \left(x_{i+1} - x_i^2 \right)^2 + \left(1 - x_i \right)^2 \right], \tag{A.4}$$

where $x_i \in [-2, 2]$ and the global optimum is 0 at $(1, 1, \ldots, 1)$. f_4 is a nonconvex unimodal function. Its optimum is inside a long, narrow, parabolic shaped flat valley.

(5) Rastrigin's Function. Consider

$$f_5(X) = 10n + \sum_{i=1}^{n} \left(x_i^2 - 10 \cos\left(2\pi x_i \right) \right), \tag{A.5}$$

where $x_i \in [-5.12, 5.12]$ and the global optimum is 0 at $(0, 0, \ldots, 0)$. f_5 is highly mutimodel.

(6) Griewangk's Function. Consider

$$f_6(X) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}} \right) + 1, \tag{A.6}$$

where $x_i \in [-600, 600]$ and the global optimum is 0 at $(0, 0, \ldots, 0)$.

(7) Sum of Different Powers. Consider

$$f_7(X) = \sum_{i=1}^{n} \left| x_i \right|^{(i+1)}, \tag{A.7}$$

Table 9: Spearman's correlation test results between k and the final solutions on all the test functions.

	SC		SC		SC		SC		SC		SC
f_1	−0.399	f_7	0.394	f_{13}	0.068	f_{19}	−0.261	f_{25}	−0.327	f_{31}	−0.304
f_2	−0.333	f_8	0.576	f_{14}	−0.124	f_{20}	0.287	f_{26}	−0.108	f_{32}	0.196
f_3	−0.018	f_9	−0.321	f_{15}	−0.120	f_{21}	0.168	f_{27}	−0.282	f_{33}	0.292
f_4	0.624	f_{10}	−0.394	f_{16}	0.317	f_{22}	0.127	f_{28}	−0.267	f_{34}	−0.174
f_5	−0.065	f_{11}	−0.333	f_{17}	0.258	f_{23}	−0.157	f_{29}	0.260		
f_6	0.060	f_{14}	−0.245	f_{18}	0.197	f_{24}	−0.388	f_{30}	−0.142		

Note. SC denotes the correlation coefficient of the Spearman correlation test.

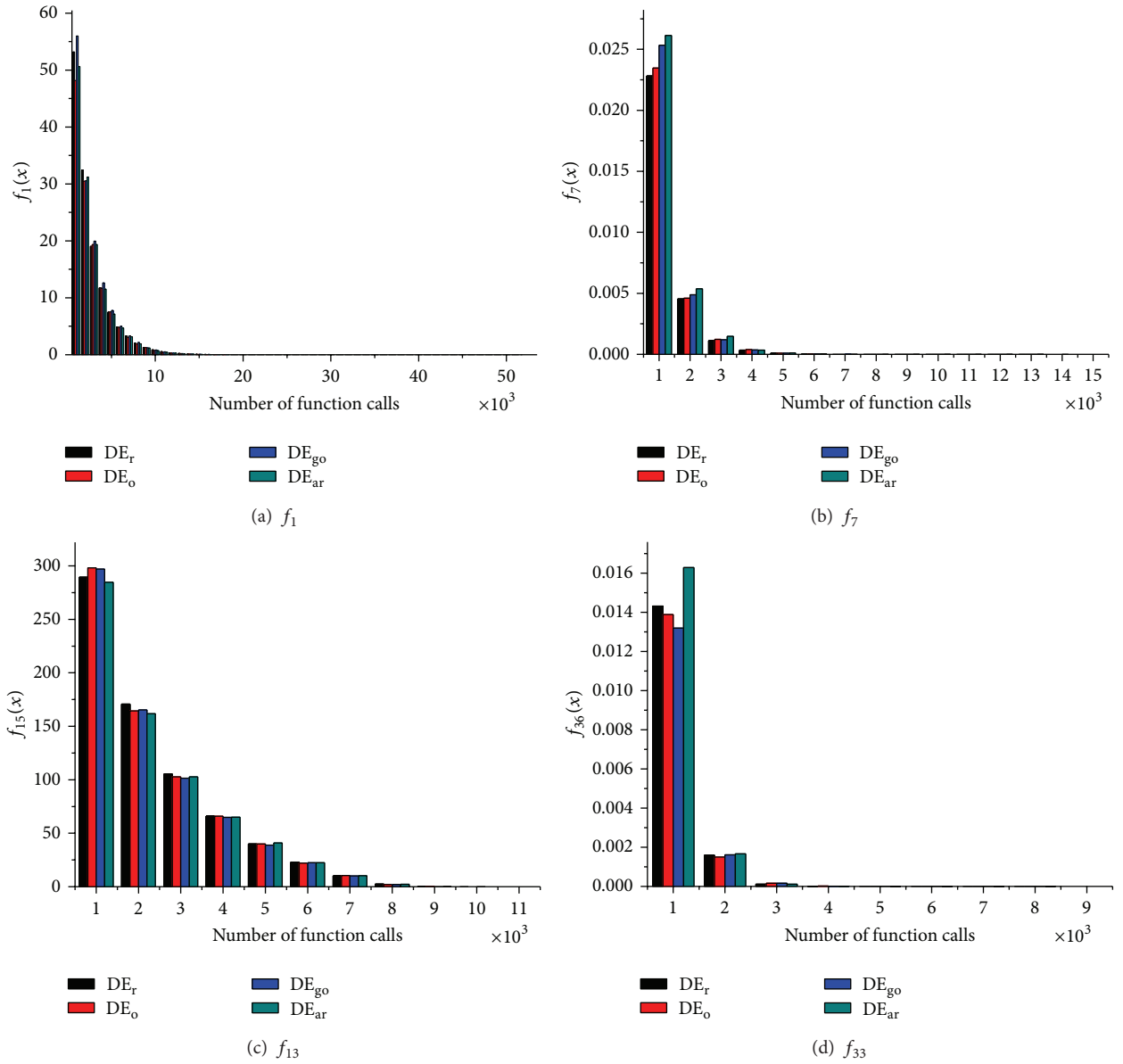

Figure 3: Some sample bar charts for the performance comparison of the 4 DE algorithms. The values are calculated at every 1,000 function calls.

where $x_i \in [-1, 1]$ and the global optimum is 0 at $(0, 0, \ldots, 0)$. f_7 is a unimodal and scalable function.

(8) Ackley's Problem. Consider

$$f_8(X) = -20 \exp\left(-0.2\sqrt{\frac{\sum_{i=1}^{n} x_i^2}{n}}\right) - \exp\left(\frac{\sum_{i=1}^{n} \cos(2\pi x_i)}{n}\right) + 20 + e, \tag{A.8}$$

where $x_i \in [-32, 32]$ and the global optimum is 0 at $(0, 0, \ldots, 0)$.

(9) Beale's Function. Consider

$$f_9(X) = \left[1.5 - x_1(1 - x_2)\right]^2 + \left[2.25 - x_1\left(1 - x_2^2\right)\right]^2 + \left[2.625 - x_1\left(1 - x_2^3\right)\right]^2, \tag{A.9}$$

where $x_i \in [-4.5, 4.5]$ and the global optimum is 0 at $(3, 0.5)$.

(10) Colville's Function. Consider

$$f_{10}(X) = 100\left(x_2 - x_1^2\right)^2 + (1 - x_1)^2 + 90\left(x_4 - x_3^2\right)^2 + (1 - x_3)^2 + 10.1\left((x_2 - 1)^2 + (x_4 - 1)^2\right) + 19.8(x_2 - 1)(x_4 - 1), \tag{A.10}$$

where $x_i \in [-10, 10]$ and the global optimum is 0 at $(1, 1, 1, 1)$.

(11) Easom's Function. Consider

$$f_{11}(X) = -\cos(x_1)\cos(x_2)\exp\left(-(x_1 - \pi)^2 - (x_2 - \pi)^2\right), \tag{A.11}$$

where $x_i \in [-100, 100]$ and the global optimum is 0 at (π, π). f_{11} is unimodal and its global minimum lays in a narrow area relative to the search space.

(12) Six-Hump Camel Back Function. Consider

$$f_{12}(X) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4 + 1.0316285, \tag{A.12}$$

where $x_i \in [-5, 5]$ and the global optimum is 0 at $(0.0898, -0.7126)$ and $(-0.0898, 0.7126)$. f_{12} has six local minima, two of them are global.

(13) Levy's Function. Consider

$$f_{13}(X) = \sin^2(3\pi x_1) + \sum_{i=1}^{n-1}(x_i - 1)^2\left(1 + \sin^2(3\pi x_{i+1})\right) + (x_n - 1)\left(1 + \sin^2(2\pi x_n)\right), \tag{A.13}$$

where $x_i \in [-10, 10]$ and the global optimum is 0 at $(1, 1, \ldots, 1)$. f_{13} has about 15^n local minima.

(14) Matyas Function. Consider

$$f_{14}(X) = 0.26\left(x_1^2 + x_2^2\right) - 0.48x_1x_2, \tag{A.14}$$

where $x_i \in [-10, 10]$ and the global optimum is 0 at $(0, 0)$. f_{14} is unimodal.

(15) Perm Function. Consider

$$f_{15}(X) = \sum_{k=1}^{n}\left[\sum_{i=1}^{n}\left(i^k + 0.5\right)\left(\left(\frac{1}{i}x_i\right)^k - 1\right)\right]^2, \tag{A.15}$$

where $x_i \in [-n, n]$ and the global optimum is 0 at $(1, 2, 3, \ldots, n)$. f_{15} is unimodal.

(16) Michalewicz's Function. Consider

$$f_{16}(X) = -\sum_{i=1}^{n}\sin(x_i)\left(\sin\left(\frac{ix_i^2}{\pi}\right)\right)^{2m} + 9.66015, \tag{A.16}$$

where $x_i \in [0, \pi]$, $m = 10$, $n = 10$, and the global optimum is 0.

(17) Zakharov's Function. Consider

$$f_{17}(X) = \sum_{i=1}^{n}x_i^2 + \left(\sum_{i=1}^{n}0.5ix_i\right)^2 + \left(\sum_{i=1}^{n}0.5ix_i\right)^4, \tag{A.17}$$

where $x_i \in [-5, 10]$ and the global optimum is 0 at $(0, 0, \ldots, 0)$. f_{17} is unimodal.

(18) Schwefel's Problem 2.22. Consider

$$f_{18}(X) = \sum_{i=1}^{n}|x_i| + \prod_{i=1}^{n}|x_i|, \tag{A.18}$$

where $x_i \in [-10, 10]$ and the global optimum is 0 at $(0, 0, \ldots, 0)$. f_{18} is unimodal.

(19) Schwefel's Problem 2.21. Consider

$$f_{19}(X) = \max\left\{|x_i|,\ 1 \le i \le n\right\}, \tag{A.19}$$

where $x_i \in [-100, 100]$ and the global optimum is 0 at $(0, 0, \ldots, 0)$. f_{19} is unimodal.

(20) Step Function. Consider

$$f_{20}(X) = \sum_{i=1}^{n}\left(\lfloor x_i + 0.5\rfloor\right)^2, \tag{A.20}$$

where $x_i \in [-100, 100]$ and the global optimum is 0 when $x_i \in [-0.5, 0.5]$.

(21) Noisy Quartic Function. Consider

$$f_{21}(X) = \sum_{i=1}^{n}ix_i^4 + \text{random}[0, 1), \tag{A.21}$$

where $x_i \in [-1.28, 1.28]$ and the global optimum is 0 at (0, 0, ..., 0).

(22) Kowalik's Function. Consider

$$f_{22}(X) = \sum_{i=1}^{11} \left[a_i - \frac{x_1 \left(b_i^2 + b_i x_2\right)}{b_i^2 + b_i x_3 + x_4} \right]^2, \tag{A.22}$$

where $x_i \in [-5, 5]$, $\alpha = [0.1957, 0.1947, 0.1735, 0.1600, 0.0844, 0.0627, 0.0456, 0.0342, 0.0323, 0.0235, 0.0246]$, $b^{-1} = [0.25, 0.5, 1, 2, 4, 6, 8, 10, 12, 14, 16]$, and the global optimum is 0.0003075 at (0.19, 0.19, 0.12, 0.14). f_{22} is multimodal.

(23) Shekel 5 Problem. Consider

$$f_{23}(X) = -\sum_{i=1}^{5} \frac{1}{\sum_{j=1}^{4} \left(x_j - a_{ij}\right)^2 + c_i} + 10.1499, \tag{A.23}$$

where

$$x_j \in [0, 10],$$
$$c_i = [0.1, 0.2, 0.2, 0.4, 0.4],$$
$$a_{ij} = \begin{bmatrix} 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 3 & 7 & 3 & 7 \end{bmatrix}, \tag{A.24}$$

and the global optimum is 0 at (4, 4, 4, 4).

(24) Shekel 7 Problem. Consider

$$f_{24}(X) = -\sum_{i=1}^{7} \frac{1}{\sum_{j=1}^{4} \left(x_j - a_{ij}\right)^2 + c_i} + 10.3999, \tag{A.25}$$

where

$$x_i \in [0, 10],$$
$$c_i = [0.1, 0.2, 0.2, 0.4, 0.4, 0.6, 0.3],$$
$$a_{ij} = \begin{bmatrix} 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 3 & 7 & 3 & 7 \\ 2 & 9 & 2 & 9 \\ 5 & 5 & 3 & 3 \end{bmatrix}, \tag{A.26}$$

and the global optimum is 0 at (4, 4, 4, 4).

(25) Shekel 10 Problem. Consider

$$f_{25}(X) = -\sum_{i=1}^{10} \frac{1}{\sum_{j=1}^{4} \left(x_j - a_{ij}\right)^2 + c_i} + 10.5319, \tag{A.27}$$

where

$$x_i \in [0, 10],$$
$$c_i = [0.1, 0.2, 0.2, 0.4, 0.4, 0.6, 0.3, 0.7, 0.5, 0.5],$$
$$a_{ij} = \begin{bmatrix} 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 3 & 7 & 3 & 7 \\ 2 & 9 & 2 & 9 \\ 5 & 5 & 3 & 3 \\ 8 & 1 & 8 & 1 \\ 6 & 2 & 6 & 2 \\ 7 & 3.6 & 7 & 3.6 \end{bmatrix}, \tag{A.28}$$

and the global optimum is 0 at (4, 4, 4, 4).

(26) Tripod Function. Consider

$$\begin{aligned} f_{26}(X) = {} & p(x_2)\left(1 + p(x_1)\right) \\ & + \left|\left(x_1 + 50p(x_2)\left(1 - 2p(x_1)\right)\right)\right| \\ & + \left|\left(x_2 + 50\left(1 - 2p(x_2)\right)\right)\right|, \end{aligned} \tag{A.29}$$

where $x_i \in [-100, 100]$, $p(x) = 1$ for $x \leq 0$; otherwise $p(x) = 0$, and the global optimum is 0 at (0, −50).

(27) 4th De Jong. Consider

$$f_{27}(X) = \sum_{i=1}^{n} i x_i^4, \tag{A.30}$$

where $x_i \in [-1.28, 1.28]$ and the global optimum is 0 at (0, 0, ..., 0).

(28) Alpine Function. Consider

$$f_{28}(X) = \sum_{i=1}^{n} \left| x_i \sin(x_i) + 0.1 x_i \right|, \tag{A.31}$$

where $x_i \in [-10, 10]$ and the global optimum is 0 at (0, 0, ..., 0). f_{28} is multimodal and not symmetrical.

(29) Schaffer's Function 6. Consider

$$f_{29}(X) = 0.5 + \frac{\sin^2 \sqrt{\left(x_1^2 + x_2^2\right)} - 0.5}{1 + 0.01\left(x_1^2 + x_2^2\right)^2}, \tag{A.32}$$

where $x_i \in [-10, 10]$ and the global optimum is 0 at (0, 0). f_{29} is multimodal.

(30) Pathological Function. Consider

$$f_{30}(X) = \sum_{i=1}^{n-1} \left(0.5 + \frac{\sin^2 \sqrt{\left(100 x_i^2 + x_{i+1}^2\right)} - 0.5}{1 + 0.001\left(x_i^2 - 2x_i x_{i+1} + x_{i+1}^2\right)^2} \right), \tag{A.33}$$

where $x_i \in [-100, 100]$ and the global optimum is 0 at (0, 0, ..., 0). f_{30} is multimodal and extremely complex.

(31) Inverted Cosine Wave Function. Consider

$$f_{31}(X) = -\sum_{i=1}^{n-1}\left(\exp\left(\frac{-\left(x_i^2 + x_{i+1}^2 + 0.5x_i x_{i+1}\right)}{8}\right) \times \cos\left(4\sqrt{x_i^2 + x_{i+1}^2 + 0.5x_i x_{i+1}}\right)\right) + n - 1, \tag{A.34}$$

where $x_i \in [-5, 5]$ and the global optimum is $1 - n$ at (0, 0, ..., 0). f_{31} is multimodal.

(32) Aluffi-Pentini's Problem. Consider

$$f_{32}(X) = 0.25x_1^4 - 0.5x_1^2 + 0.1x_1 + 0.5x_2^2 + 0.3523, \tag{A.35}$$

where $x_i \in [-10, 10]$ and the global optimum is 0 at (−1.0465, 0).

(33) Becker and Lago Problem. Consider

$$f_{33}(X) = \left(|x_1| - 5\right)^2 + \left(|x_2| - 5\right)^2, \tag{A.36}$$

where $x_i \in [-10, 10]$ and the global optimum is 0 at (±5, ±5).

(34) Bohachevsky 1 Problem. Consider

$$f_{34}(X) = x_1^2 + 2x_2^2 - 0.3\cos\left(3\pi x_1\right)0.4\cos\left(4\pi x_2\right) + 0.3, \tag{A.37}$$

where $x_i \in [-50, 50]$ and the global optimum is 0 at (0, 0).

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (nos. 61202048 and 61202200), the Commonwealth Project of Science and Technology Department of Zhejiang Province (no. 2014C23008), the Zhejiang Provincial Nature Science Foundation of China (no. LY13F020010), and the Open Foundation of State Key Laboratory of Software Engineering of Wuhan University of China (no. SKLSE-2012-09-21).

References

[1] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*, MIT Press, Cambridge, UK, 1992.

[2] R. Storn and K. Price, "Differential evolution: a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[3] K. Price, R. Storn, and J. Lampinen, *Differential Evolution—A Practical Approach to Global Optimization*, Springer, Berlin, Germany, 2005.

[4] H. Maaranen, K. Miettinen, and M. M. Mäkelä, "Quasi-random initial population for genetic algorithms," *Computers & Mathematics with Applications*, vol. 47, no. 12, pp. 1885–1895, 2004.

[5] S. Rahnamayan, H. R. Tizhoosh, and M. M. A. Salama, "A novel population initialization method for accelerating evolutionary algorithms," *Computers & Mathematics with Applications*, vol. 53, no. 10, pp. 1605–1614, 2007.

[6] H. Wang, Z. Wu, Y. Liu, J. Wang, D. Jiang, and L. Chen, "Space transformation search: a new evolutionary technique," in *Proceedings of the 1st ACM/SIGEVO Summit on Genetic and Evolutionary Computation (GEC '09)*, pp. 537–544, Shanghai, China, June 2009.

[7] H. Wang, Z. Wu, and S. Rahnamayan, "Enhanced opposition-based differential evolution for solving high-dimensional continuous optimization problems," *Soft Computing*, vol. 15, no. 11, pp. 2127–2140, 2011.

[8] G. C. Onwubolu and B. V. Babu, *New Optimiza tion Techniques in Engineering*, Springer, Berlin, Germany, 2004.

[9] J. Andre, P. Siarry, and T. Dognon, "An improvement of the standard genetic algorithm fighting premature convergence in continuous optimization," *Advances in Engineering Software*, vol. 32, no. 1, pp. 49–60, 2001.

[10] J. Brest, S. Greiner, B. Bošković, M. Mernik, and V. Zumer, "Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 6, pp. 646–657, 2006.

[11] X. Yao, Y. Liu, and G. Lin, "Evolutionary programming made faster," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 2, pp. 82–102, 1999.

[12] C.-Y. Lee and X. Yao, "Evolutionary programming using mutations based on the Levy probability distribution," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 1, pp. 1–13, 2004.

[13] R. S. Rahnamayan, H. R. Tizhoosh, and M. M. A. Salama, "Opposition-based differential evolution," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 1, pp. 64–79, 2008.

[14] P. N. Suganthan, N. Hansen, J. J. Liang et al., "Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization," Tech. Rep. 2005005, Nanyang Tech. Univ., Singapore and KanGAL, Kanpur Genetic Algorithms Lab., IIT, Kanpur, India, May 2005.

[15] S. Garcia, A. Fernandez, J. Luengo, and F. Herrera, "Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: experimental analysis of power," *Information Sciences*, vol. 180, no. 10, pp. 2044–2064, 2010.

[16] C. R. Kothari, *Research Methodology: Methods and Techniques*, New Age International, New Delhi, India, 2007.