Information Sciences 382–383 (2017) 216–233

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Sampling-based adaptive bounding evolutionary algorithm for continuous optimization problems

Linbo Luo[a,*], Xiangting Hou[b], Jinghui Zhong[c], Wentong Cai[b], Jianfeng Ma[a]

[a] School of Cyber Engineering, Xidian University, Xi'an 710071, China
[b] School of Computer Science and Engineering, Nanyang Technological University, Singapore
[c] School of Computer Science and Engineering, South China University of Technology, Guangzhou, China

ARTICLE INFO

Article history:
Received 1 June 2016
Revised 6 December 2016
Accepted 14 December 2016
Available online 14 December 2016

Keywords:
Search space narrowing
Adaptive bounding evolutionary algorithm
Continuous optimization problem
Simulation model calibration

ABSTRACT

This paper proposes a novel sampling-based adaptive bounding evolutionary algorithm termed SABEA that is capable of dynamically updating the search space during the evolution process for continuous optimization problems. The proposed SABEA adopts two bounding strategies, namely fitness-based bounding and probabilistic sampling-based bounding, to select a set of individuals over multiple generations and leverage the value information from these individuals to update the search space of a given problem for improving the solution accuracy and search efficiency. To evaluate the performance of this method, SABEA is applied on top of the classic differential evolution (DE) algorithm and a DE variant, and SABEA is compared to a state-of-the-art Distribution-based Adaptive Bounding Genetic Algorithm (DABGA) on a set of 27 selected benchmark functions. The results show that SABEA can be used as a complementary strategy for further enhancing the performance of existing evolutionary algorithms and it also outperforms DABGA. Finally, a practical problem, namely the model calibration for an agent-based simulation, is used to further evaluate SABEA. The results show SABEA's applicability to diverse problems and its advantages over the traditional genetic algorithm-based calibration method and DABGA.

1. Introduction

Continuous optimization problems are commonly observed in many real-world applications, such as scheduling, industrial process optimization, vehicle routing in large-scale networks and gene recognition in bioinformatics [4,7,8,23,24,26], just to name a few. To tackle these problems, evolutionary algorithms (EAs), including genetic algorithm (GA) [13], estimation of distribution algorithms (EDA) [19,39], differential evolution (DE) [14,28], particle swarm optimization (PSO) [34], Artificial Bee Colony (ABC) [1], etc., have been widely adopted. When solving continuous optimization problems using these methods, a population-based stochastic global search is performed in a search space constrained by the boundaries of the continuous variables of the problem. Candidate solutions are randomly generated within the search space and then iteratively evolved. While EA-based methods are recognized as effective approaches for continuous optimization problems, they may experience slow convergence when the search space of the problem becomes large. They may also suffer from premature convergence when solving complex multimodal problems.

* Corresponding author.
E-mail address: lbluo@xidian.edu.cn (L. Luo).

http://dx.doi.org/10.1016/j.ins.2016.12.023
0020-0255/

To enhance the convergence speed of EAs for solving continuous optimization problems, an effective method is to dynamically adapt the search space of problem while searching, i.e., instead of using a fixed search space throughout the search process, the boundaries of the variables are gradually updated such that new individuals can be generated within a smaller search space. As the search space is reduced during the search, this method can potentially speed up the convergence rate. Moreover, the search space updating method can be applied to many variants of EAs, including GA, DE and others. Thus, it can be used as a complementary strategy with other EAs. In this paper, we refer to an EA that uses the search space updating method as an adaptive bounding EA (ABEA).

For ABEA methods, the key question is how to effectively update the search space of a problem without missing global optimum. Several methods have been proposed in the literature. The major methods include GA with Parameter Space Size Adjustment (GAPSSA) [5], Successive Zooming GA (SZGA) [18] and Distribution-based Adaptive Bounding for GAs (DABGA) [27]. For GAPSSA, the search space is updated after each generation by narrowing the boundaries of each variable based on its average value over the whole population by a constant reduction rate. For SZGA, the search space is updated after every N_{sub} generations and the boundaries of variables are updated based on the current candidate optimal point with a fixed zooming factor. In contrast to GAPSSA and SZGA, where the reduction rate is set '*a priori*', DABGA utilizes the value distribution information of variables at the current generation to update the boundaries of each variable. DABGA has been shown to outperform GAPSSA and SZGA in terms of convergence speed and solution accuracy [27].

From the above ABEA methods, we observe some limitations. Firstly, the search space updating is based only on the information of individuals in a single generation (i.e., the generation before the search space updating). This may be problematic if the variables interact with each other. In such a case, the value of a variable in a single generation is easily affected by other interacting variables. The update of the boundaries of a variable can thus be affected by interacting variables, which can affect the search performance. Secondly, as the update of the search space relies on the current information of the variables, it is likely to be trapped into the local optima while reducing the search space. In the above methods, no explicit mechanism is provided to reduce the possibility of being trapped into the local optima. Lastly, the above methods have only been applied to the simplest form of EA, i.e., GA. The efficiency of these methods has not been tested over more advanced EAs, such as DE, and the evaluations have only conducted on relatively simple test functions.

To address the above-mentioned limitations, we propose a sampling-based adaptive bounding evolutionary algorithm (SABEA). In the proposed SABEA, the boundaries of the variables are updated based on information of individuals sampled over multiple generations. Without a fixed reduction rate, we can utilize the value information of individuals with better fitness over multiple generations to update the search space. To this end, a fitness-based bounding strategy is used to select a set of individuals with better fitness after every n_g generations. We assume that the search space constrained by these better individuals is more likely to contain the global optimum. Furthermore, we introduce an explicit mechanism to avoid the search being trapped into local optima. A probabilistic sampling-based bounding inspired by EDA [19] is adopted to randomly sample another set of individuals to improve the diversity of sampled individuals. The proposed method updates the search space based on information of individuals collected by both fitness-based bounding and probabilistic sampling-based bounding. In order to comprehensively evaluate the performance of SABEA, we apply this method on top of classic DE and two DE variants to solve a set of 27 benchmark problems taken from [37] and the CEC'05 [[31]. The experimental results clearly show the efficiency and effectiveness of SABEA for diverse problems. To demonstrate the robustness of SABEA for practical engineering problems, we further apply our method to solve the model calibration problem in an agent-based simulation. The experiments are conducted in two scenarios and the results show that the proposed method is effective at improving the convergence rate and solution accuracy.

The rest of this paper is organized as follows: Section 2 describes the related background method-DABGA. Section 3 presents the proposed SABEA. Section 4 describes the experimental studies. Section 5 concludes the paper.

2. Preliminaries

Our method is inspired and improved from the DABGA algorithm proposed by Peng et al. [27] for solving continuous optimization problems. In this section, we first introduce the continuous optimization problem and briefly describe the DABGA algorithm. Potential limitations of the DABGA algorithm are also discussed here.

The continuous optimization problem typically optimizes a real-value objective function $f(X)$. The input of $f(X)$ consists of a set of n variables $X = \{x_1, x_2, \ldots, x_n\}$, where $X \in \mathbb{R}^n$. Each variable x_i is constrained by a lower bound LB_i and an upper bound UB_i, i.e., $LB_i \le x_i \le UB_i$. The upper and lower bounds of all the input variables define the search space S of the given problem, where $S \subseteq \mathbb{R}^n$. The continuous optimization problem finds the optimum X_{opt} in the search space S such that $f(X_{opt})$ is the optimal value (e.g., the global minimum) for all $X \in S$.

When applying EAs for continuous optimization problems, the boundaries of the input variables of the objective function are usually fixed during the evolution process. However, if the search space can be dynamically reduced without missing X_{opt} during the evolution, this can potentially help to speed up the convergence and improve the precision of the results. To this end, the DABGA algorithm was proposed to dynamically update the search space of the input variables for continuous optimization problems. In DABGA, the upper bound and lower bound of the ith input variable x_i of an objective function are updated according to the following rules:

$$UB_i(k+1) = \tilde{x}_i(k) + b * [UB_i^{\{\beta\}}(k) - \tilde{x}_i(k)], \tag{1}$$

$$LB_i(k+1) = \bar{x}_i(k) - b * [\bar{x}_i(k) - LB_i^{\{\beta\}}(k)], \tag{2}$$

where k is the index of generation, $UB_i(k)$ and $LB_i(k)$ are the upper and lower bounds of variable x_i at the kth generation, and $\bar{x}_i(k)$ is the mean value of variable x_i over the population at the kth generation. b and β are referred to as the *bounding factor* and *bounding fraction* respectively, which take values in the ranges of (1, ∞) and (0, 1], respectively. b is usually set to be greater than 1, but close to 1 (e.g., $b = 1.25$).

The $UB_i^{\{\beta\}}(k)$ and $LB_i^{\{\beta\}}(k)$ in Eqs. (1) and (2) are determined using the following rule:

$$UB_i^{\{\beta\}}(k) - LB_i^{\{\beta\}}(k) = min_j\{x_{i,j+N_\beta}^k - x_{i,j+1}^k\}, 0 \le i \le N_{pop} - N_\beta, \tag{3}$$

where N_{pop} is the population size, $\{x_{i,j}^k$, j=1,... ,$N_{pop}\}$ is the set of values of variable x_i in increasing order, each $x_{i,j}^k$ in this set is the value of variable x_i of the jth individual at the kth generation, and N_β is the rounded integer of $\beta \times N_{pop}$. The interval $[LB_i^{\{\beta\}}(k), UB_i^{\{\beta\}}(k)]$ is thus the shortest interval that contains N_β values of the variable x_i. Here, β is set in the range of (0, 1] and is used as a bounding fraction. For example, if β is set to 0.9, this implies that 90% of the values of the variable x_i in the current population will be within the new boundaries.

In DABGA, the search space of the variables is not updated in every generation during the GA search. Instead, DABGA updates the variable boundaries only after a certain number of generations once GA starts, and then updates the boundaries for every n_g generations afterwards. The purpose of this is to prevent premature convergence of the GA search. The general idea of DABGA is that updating the boundaries for each variable depends on the value distribution of the variable over the whole population. The algorithm assumes that the range, where the values of a variable over the population are most densely distributed, has a high chance of containing the optimal value. Hence, the distribution statistics are utilized in the boundary updating process. While the DABGA algorithm has been shown to outperform several other search space updating algorithms, we observe two potential limitations of DABGA as follows:

Firstly, as shown in Eqs. (1) and (2), the new boundary of a variable is affected by $[LB^{\{\beta\}}(k), UB^{\{\beta\}}(k)]$, which is the most densely distributed range that contains N_β values of a variable. It is reasonable to assume that the global optimal value is most likely located within this range. However, within this range, the values of the variable may be evenly distributed or clustered within a smaller sub-range. If the values are clustered, it is desirable to set a smaller β to further narrow the range for faster convergence. The DABGA algorithm does not examine the pattern of the value distribution for a variable and only allows a fixed β to be set throughout the evolution process. This makes it difficult to account for changes in the distribution pattern of the variable.

Secondly, in DABGA, the boundaries of the variable are updated based on the distribution of the variable without considering the effect of interacting variables. In many continuous optimization problems, the variables do interact with each other. In such a case, the fitness of an objective function can be affected not only by the values of variables independently, but also via the interactions of these variables. For example, consider two simple functions $f_1(x_1, x_2) = x_1 + x_2$ and $f_2(x_1, x_2) = x_1{}^*x_2$. In f_1, the variables x_1 and x_2 are independent. The function value can only approach its optimum (i.e., minimum value) when both variables approach their optima (i.e., the minimum value within their value ranges). In f_2, as two variables are interacting, the function may reach a sub-optimal value even when the value of one variable is small and the value of the other variable is large (i.e., diverges from its optimal value). Thus, updating the boundaries based only on the value distribution of variables may not be effective when the objective function has complex dependencies among multiple input variables.

3. The proposed method

In this section, we propose a novel adaptive search space updating method for EAs. The proposed method is termed a sampling-based adaptive bounding evolutionary algorithm (SABEA). Similar to DABGA, our method starts to update the boundaries of the variables after n_o generations of the evolution process and then performs the boundary updating every n_g generations afterwards. However, when updating the boundaries of each variable, our method does not rely on the value distribution statistics of the variable over a previous generation. Instead, we adopt a *fitness-based bounding* strategy that selects the individuals with better fitness over multiple previous generations and updates the boundaries based on the search space confined by these best-so-far individuals. We assume that a search space confined by these individuals is most likely to contain the optimal values. To enhance this method's ability to jump out of local optima, we also introduce a *probabilistic sampling-based bounding* strategy to randomly sample additional individuals based on Gaussian distribution. The search space confined by the individuals collected by these two strategies is eventually used to update the boundaries of the variables.

The method starts with the initialization of N_{pop} individuals as the initial population. To avoid premature convergence of the EA search, the method first performs genetic operations (i.e., selection, crossover, mutation) over the population for n_o generations without updating the boundaries of the variables. After n_o generations, the method updates the boundaries of the variables every n_g generations (i.e., when $num = n_g$). If the generation number has not reached n_g, the method proceeds with normal evolution and records the individuals in each generation. Once the generation reaches n_g, the recorded individuals over the previous n_g generations are sorted in increasing order based on their fitness values (here, we assume

a smaller fitness value corresponds to better fitness). The fitness-based bounding is then performed on these individuals to select a set of individuals with better fitness values. The probabilistic sampling-based bounding adds random individuals to the set selected by the fitness-based bounding. The new boundaries of the variables are determined based on the search space confined by all the sampled individuals. Details of our search space updating technique are described below.

Step 1. Sorting the individuals based on fitness

The purpose of this step is to sort all the individuals recorded in n_g generations based on their fitness values. For a continuous optimization problem, each individual of a candidate solution can be expressed as an array X:

$$X = [x_1, x_2, \ldots, x_n], \tag{4}$$

where x_i is an input variable of the objective function for continuous optimization and n is the total number of variables. At the kth generation, the population of all the individuals A^k and the corresponding fitness values of individuals can then be expressed as:

$$A^k = [X_1^k, X_2^k, \ldots, X_j^k, \ldots, X_{N_{pop}}^k], FT^k = [\phi_1^k, \phi_2^k, \ldots, \phi_j^k, \ldots, \phi_{N_{pop}}^k], \tag{5}$$

where X_j^k is the jth individual at the kth generation, ϕ_j^k is the individual's corresponding fitness value and N_{pop} is the size of the population.

Step 1 of our method thus sorts $N_{pop}{}^*n_g$ individuals (i.e., the elements in A^k to A^{k+n_g-1}) based on their corresponding fitness values (i.e., FT^k to FT^{k+n_g-1}). To record all the individuals over n_g generations, we use an array Â with a size of $N_{pop}{}^*n_g$. The simplest way to perform sorting on Â is to sequentially add the elements in A^k to A^{k+n_g-1} into Â and then perform the sorting using an existing sorting algorithm. In this way, the best time complexity of sorting is $O((N_{pop}{}^*n_g)Log(N_{pop}{}^*n_g))$ using heap sort. However, for some optimization problems, the value of $N_{pop}{}^*n_g$ can be large. To reduce the time complexity of sorting, we adopt a two-step sorting strategy. That is, we first perform a heap sort on population A^k after every generation to obtain a sorted array $\widetilde{A}^k$. Once the generation number reaches n_g, we then sort n_g sorted arrays (i.e., $\widetilde{A}^k$ to $\widetilde{A}^{k+n_g-1}$) using a multiple merge sort with min-heap [17] and put the final sorted elements into Â. By using this two-step sorting strategy, the time complexity of sorting $N_{pop}{}^*n_g$ individuals based on their fitness values can be reduced to $O((N_{pop}{}^*n_g)Log(n_g))$. The space complexity for storing all individuals with the dimension of n is Â is $O(N_{pop}{}^*n_g{}^*n)$.

Step 2. Fitness-based bounding

In this step, we select the individuals with better fitness values in the n_g generations and use these individuals to update the boundaries (i.e., bounding) of the search space. In step 1, we obtained the sorted array Â containing $N_{pop}{}^*n_g$ individuals. Here, we simply select the top $\alpha{}^*n_g{}^*N_{pop}$ elements in the array Â as the candidate individuals, where α is a pre-determined constant referred to as the *selecting fraction*, and $0 < \alpha < 1$. The search space confined by the selected individuals can then be represented as $S^{fit} = \{[LB_1^{fit}, UB_1^{fit}], [LB_2^{fit}, UB_2^{fit}], \ldots, [LB_n^{fit}, UB_n^{fit}]\}$, where LB_i^{fit} and UB_i^{fit} are the minimum and maximum values respectively of the ith variable over the selected individuals and n is the dimension of the problem (i.e., the number of variables in each individual). As will be described later on, the new boundaries of each variable will be determined based on the search space S^{fit}. The main idea of the fitness-based bounding is thus to select the values of a variable that correspond to good fitness values.

The selecting fraction α is used to determine the number of individuals that should be selected based on their fitness values. For example, if $\alpha = 0.35$, it means that the top 35% of individuals with the highest fitness will be selected. The value that α is set at can affect the performance of the proposed method. If α is set close to 1, most individuals will be included in the sampled individuals. In this case, the updated boundaries will be close to the existing boundaries. This will affect the method's ability to speed up the convergence. If α is set as a small value (i.e., close to 0), it may select too few individuals. This may result in the range defined by the updated boundaries being too narrow, and may increase the possibility of missing the true optimal solution. Based on our experiments on multiple test problems, setting α at 0.2 is proper for most cases.

Step 3. Probabilistic sampling-based bounding

This step aims to refine the search space S^{fit} obtained in step 2 to avoid being trapped into local optima. To achieve this, we probabilistically sample individuals over the $N_{pop}{}^*n_g$ individuals and add them to the selected individuals from step 2. Our probabilistic sampling is inspired by Estimation of Distribution Algorithms (EDAs) [9], which use probabilistic models instead of crossover and mutation operators to generate individuals. We adopt a similar approach to generate individuals for probabilistic sampling, as EDAs have been proven to have better performance than genetic operator-based EAs for problems with complex interactions [25]. The basic procedure of the probabilistic sampling is outlined in Algorithm 1.

Specifically, a truncation selection operation is first performed to select $\delta{}^*N_{pop}{}^*n_g$ better individuals from the sorted array Â obtained in step 1. Here, the *truncation fraction* δ is usually set as 0.2, which selects 20% of the best individuals. Similar to EDAs, we use the selected individuals to construct a probabilistic model M for estimating the distribution of the selected individuals. In this paper, the model is constructed with Gaussian distribution, which is commonly used in EDAs for multivariate real-valued variables [6,19]. It should be noted that more advanced distribution models could also be used, depending on the test problems. The model M is represented as a vector of normal distributions, one for each variable. The

Algorithm 1 Probabilistic sampling pseudocode.

1: **Input**: The sorted array $\hat{A}$ containing $N_{pop} * n_g$ individuals, truncation fraction δ, sample size m.
2: **Output**: An array A^p containing m probabilistically sampled individuals.
3: select $\delta * N_{pop} * n_g$ better individuals S from $\hat{A}$
4: build the probabilistic model M from S
5: **for** $j = 0$; $j < m$; $j++$ **do**
6: $A^p[j] \leftarrow$ sample M to generate a new individual
7: **end for**

mean and standard deviation for each distribution is thus determined as follows:

$$\mu_i = \frac{\sum_{j=1}^{\delta * N_{pop} * n_g} x_{i,j}}{\delta * N_{pop} * n_g}, \sigma_i = \sqrt{\frac{\sum_{j=1}^{\delta * N_{pop} * n_g} (x_{i,j} - \mu)^2}{\delta * N_{pop} * n_g}}, \tag{6}$$

where μ_i is the mean for the ith variable and σ_i is the standard deviation for the ith variable. The probabilistic model M for the ith variable is thus constructed as $M(x_i) = \frac{1}{\sqrt{2\pi}} e^{\frac{(x_i-\mu_i)^2}{2\sigma_i^2}}$. Based on the constructed probabilistic model M, we then randomly sample m individuals and store them in the array A^p. The search space confined by the sampled individuals can then be represented as $S^{ps} = \{[LB_1^{ps}, UB_1^{ps}], [LB_2^{ps}, UB_2^{ps}], \ldots, [LB_n^{ps}, UB_n^{ps}]\}$, where LB_i^{ps} and UB_i^{ps} are the minimum and maximum values respectively of the ith variable over the individuals in A^p, and n is the dimension of the problem. It should be noted that, due to the nature of probabilistic sampling, it is possible to include some individuals with worse fitness. This can potentially improve the diversity of the population for updating the search space.

Step 4. Updating the boundaries

The last step is to update the boundaries of each variable according to the search space confined by the individuals selected in steps 2 and 3. The rules for updating the upper bound and lower bound of a variable x_i are as follows:

$$UB_i(k+1) = \bar{x}_i(k) + b * [max\{UB_i^{fit}, UB_i^{ps}\} - \bar{x}_i(k)], \tag{7}$$

$$LB_i(k+1) = \bar{x}_i(k) - b * [\bar{x}_i(k) - min\{LB_i^{fit}, LB_i^{ps}\}], \tag{8}$$

where k is the index of generation, $UB_i(k)$ and $LB_i(k)$ are the upper and lower bounds of variable x_i at the kth generation, $\bar{x}_i(k)$ is the mean value of variable x_i over the population at the kth generation, UB_i^{fit} and LB_i^{fit} are the boundaries of the variable x_i obtained from fitness-based bounding, UB_i^{ps} and LB_i^{ps} are the boundaries of the variable x_i obtained from probabilistic sampling-based bounding, and b is the *bounding factor* (the same as in DABGA), which is usually set to be greater than 1, but close to 1.

Eqs. (7) and (8) show that our method updates the boundaries of the variable by combining the search space confined by the individuals from fitness-based bounding and probabilistic sampling-based bounding (i.e., it takes the maximum for the upper bounds and the minimum for the lower bounds). After completing step 4, the proposed method repeats steps 1–4 after n_g generations and the entire evolution process ends until the termination condition is met.

The key features that distinguish our method from the DABGA algorithm are as follows. Firstly, we use a fitness-based strategy to determine the new boundaries of variables instead of relying on the value distribution information. This removes the need to specify the range where the values of the variable are most densely distributed (i.e., setting β value in DABGA). In practice, this range may change during the evolution process and may thus require dynamically adjusting β in DABGA. Secondly, while DABGA updates the boundaries based on information in the last generation, our method is based on the information of all individuals over n_g generations. This can help reduce the effect of interacting variables, as the values of the variable in one generation are more easily affected by other variables. Lastly, our method introduces a randomized method (i.e., probabilistic sampling), which can enhance the method's ability to avoid being trapped into local optima.

4. Experimental studies

This section investigates the performance of the proposed SABEA method on two classes of problems. The first class comprises 27 benchmark function optimization problems. The test function optimization problems contain a set of traditional scalable test functions [37] and a set of benchmark functions provided by the CEC 2005 special session [[31]. To demonstrate the efficiency and generality of the proposed method, SABEA is applied on top of the classic DE and a state-of-the-art DE named CoDE [32]. The second class of problems is the model calibration for an agent-based simulation. The model calibration problem can be considered to be a special type of continuous optimization problem where the input variables are the model parameters to be calibrated and the objective function is a distance function between the simulation outputs and the desired outcomes (e.g., real-world data). In both classes of problems, we compare our SABEA with DABGA operated on different EAs.

Table 1
The test functions.

	Function	Function name	Search space
Unimodal Functions	$f1$	Sphere problem	$[-100,100]^n$
	$f2$	Schwefels problem 2.22	$[-10,10]^n$
	$f3$	Schwefels problem 1.2	$[-100,100]^n$
	$f4$	Schwefels problem 2.23	$[-100,100]^n$
Multimodal Functions	$f5$	Rosenbrocks function	$[-30,30]^n$
	$f6$	Step function	$[-100,100]^n$
	$f7$	Quartic function with noise	$[-1.28,1.28]^n$
	$f8$	Schwefels problem 2.26	$[-500,500]^n$
	$f9$	Rastrigins function	$[-5.12,5.12]^n$
	$f10$	Ackleys function	$[-32,32]^n$
	$f11$	Griewank function	$[-600,600]^n$
	$f12$	Penalized function 1	$[-50,50]^n$
	$f13$	Penalized function 2	$[-50,50]^n$
Unimodal Functions	$f14$	Shifted sphere function	$[-100,100]^n$
	$f15$	Shifted Schwefels problem 1.2	$[-100,100]^n$
	$f16$	Shifted rotated high conditioned elliptic function	$[-100,100]^n$
	$f17$	Shifted Schwefels problem 1.2 withnoise inFitness	$[-100,100]^n$
	$f18$	Schwefels problem 2.6 with global optimum on bounds	$[-100,100]^n$
Basic Multimodal Functions	$f19$	Shifted Rosenbrocks function	$[-100,100]^n$
	$f20$	Shifted rotated Griewanks function without bounds	$[0,600]^n$
	$f21$	Shifted rotated Ackleys function with global optimum on bounds	$[-32,32]^n$
	$f22$	Shifted Rastrigins function	$[-5,5]^n$
	$f23$	Shifted rotated Rastrigins function	$[-5,5]^n$
	$f24$	Shifted rotated Weierstrass function	$[-0.5,0.5]^n$
	$f25$	Schwefels problem 2.13	$[-\pi, \pi]^n$
Expanded Multimodal Functions	$f26$	Expanded extended Griewanks plus Rosenbrocks function	$[-3,1]^n$
	$f27$	Shifted rotated expanded scaffers F6	$[-100,100]^n$

4.1. Experiments using traditional test function problems

4.1.1. Test functions

To comprehensively evaluate the performance of the proposed method, we adopt 27 well-known benchmark test functions as shown in Table 1. The details of these test functions are included in Appendix A. Among these test functions, the first 13 functions are taken from [37], $f_1 - f_4$ are unimodal functions, f_5 is the Rosenbrock function which is a unimodal function for $n = 2$ and a multimodal function for $n > 3$, and $f_8 - f_{13}$ are multimodal functions. The next 14 functions $f_{14} - f_{27}$ are taken from the CEC05 benchmark problems [31]. Among these 14 functions, $f_{14} - f_{18}$ are unimodal functions, $f_{19} - f_{25}$ are shifted rotated multimodal functions and $f_{26} - f_{27}$ are expanded multimodal functions.

4.1.2. Algorithms for evaluations and their settings

As mentioned in Section 1, the existing ABEA algorithms have only been applied to the simplest form of EA, that is, GA. In this work, in order to evaluate the ABEA algorithms' applicability to more advanced EA algorithms, we apply the DABGA [27] method and our SABEA method on top of DE, which is one of the most popular algorithms for continuous optimization problems [2]. The original DE was first proposed by Storn and Price [30] to solve the Chebyshev polynomial fitting problem. Since then, DE and its variants have emerged as one of the most effective and competitive EAs and are widely used to solve scientific and engineering problems [16,26,33].

In this work, we use the classic DE [30] and a state-of-the-art DE variant (CoDE [32]) as our base algorithms. The classic DE uses the "DE/rand/1" mutation scheme and the control parameters F and CR of DE are set to 0.5 and 0.9, respectively. The population size N_{pop} of the classic DE is set to 100. For CoDE, the individuals are sampled by several trial vector generation strategies [32]. Its control parameters, F and CR, are randomly selected from the parameter candidate pool. In our implementation, CoDE selects the combinations of F and CR from the parameter pool: $CR \in [0.1, 0.9, 0.2]$ and $F \in [1.0, 1.0, 0.8]$. The population size N_{pop} of CoDE is set to 30, according to [32].

To apply DABGA and our SABEA method on top of classic DE and CoDE, the search space is dynamically updated according to the strategies described in the respective methods. Once the search space is updated, we create a trial population within the updated search space and select a set of best individuals from the trial population according to their fitness values. These selected individuals are then used to replace the worst individuals in the existing population. Note that not all the individuals in the existing population are updated. In our current implementation, the best three individuals in the trial population are selected to replace the worst three individuals in the existing population. To create the trial population

Table 2
Results of DE, DE+DABGA, DE+SABEA on 27 test functions.

	Function	DE Mean error± std. dev.	DE+DABGA Mean error± std. dev.	DE+SABEA Mean error± std. dev.
Unimodal Functions	f1	2.17E−36 ± 3.06E−36$^{\downarrow}$	**1.82E−59 ± 4.14E−59**$^{\uparrow}$	1.25E−51± 1.35E−51
	f2	3.76E−18± 1.75E−18$^{\downarrow}$	**2.31E−32 ± 6.64E−32**$^{\uparrow}$	4.11E−25± 3.66E−25
	f3	3.72−6± 4.46E−06$^{\downarrow}$	2.23E+01± 1.51E+01$^{\downarrow}$	**1.09E−09 ± 2.00E−09**
	f4	**7.18E−03 ± 8.66E−03**$^{\updownarrow}$	5.77E+00 ± 1.67E+01$^{\downarrow}$	**8.33E−03 ± 1.14E−02**
Multimodal Functions	f5	**6.27E−01 ± 8.17E−01**$^{\updownarrow}$	1, 18E+01 ± 6.30E+00$^{\downarrow}$	**6.31**E−**01** ± **1.12**E+**00**
	f6	**0.00**E+**00** ± **0.00**E+**00**$^{\updownarrow}$	**0.00**E+**00** ± **0.00**E+**00**$^{\updownarrow}$	**0.00**E+**00** ± **0.00**E+**00**
	f7	5.19E−02± 3.17E−2$^{\downarrow}$	1.18E−01± 9.97E−02$^{\downarrow}$	**4.81E−03 ± 3.76E−03**
	f8	1.18E+01± 3.74E+01$^{\downarrow}$	5.01E+01± 9.52E+01$^{\downarrow}$	**3.24E−01 ± 1.12E−01**
	f9	7.87E+01± 2.78E+01$^{\downarrow}$	**1.59**E+**01** ± **4.39**E+**00**$^{\updownarrow}$	**2.04**E+**01** ± **6.28**E+**00**
	f10	**4.71E−15 ± 9.85E−16**$^{\updownarrow}$	8.81E−01± 6.79E−01$^{\downarrow}$	**4.44**E−**15** ± **0.00**E+**00**
	f11	7.58E−04± 2.73E−03$^{\downarrow}$	5.49−03± 7.46E−03$^{\downarrow}$	**1.76E−04 ± 4.19E−04**
	f12	**1.57E−32 ± 2.84E−48**$^{\updownarrow}$	**1.57E−32 ± 2.87E−32**$^{\updownarrow}$	**1.57E−32 ± 2.84E−48**
	f13	**1.34E−32 ± 2.24E−48**$^{\updownarrow}$	**1.76E−32 ± 3.15E−48**$^{\updownarrow}$	**1.31E−32 ± 2.17E−48**
Unimodal Functions	*f*14	**0.00**E+**00** ± **0.00**E+**00**$^{\updownarrow}$	3.24E−28± 2.24E−28$^{\downarrow}$	**0.00**E+**00** ± **0.00**E+**00**
	*f*15	3.82E−06± 5.05E−06$^{\downarrow}$	1.14E+02± 7.72E+01$^{\downarrow}$	**7.39E−10 ± 8.24E−10**
	*f*16	3.69E+05 ± 1.99E+05$^{\downarrow}$	5.93E+05 ± 1.46E+05$^{\downarrow}$	**1.09**E+**05** ± **9.82**E+**04**
	*f*17	3.76E−03 ± 3.85E−03$^{\downarrow}$	2.05E+04 ± 4.25E+03$^{\downarrow}$	**8.19E−04 ± 1.08E−04**
	*f*18	1.01E+01± 8.91E+00$^{\downarrow}$	3.78E+03± 5.12E+02$^{\downarrow}$	**4.65E−01 ± 3.92E−01**
Basic Multimodal Functions	*f*19	**8.65**E−**01** ± **1.07**E+**00**$^{\updownarrow}$	1.82E+01± 2.24E+01$^{\downarrow}$	**7.92**E−**01** ± **1.09**E+**00**
	*f*20	2.32E−02 ± 8.18E−03$^{\downarrow}$	3.12E+00 ± 1.27E+00$^{\downarrow}$	**7.21E−03 ± 1.36E−03**
	*f*21	**2.10**E+**01** ± **3.74**E−**01**$^{\updownarrow}$	**2.07**E+**01** ± **2.40**E−**01**$^{\updownarrow}$	**2.08**E+**01** ± **2.23**E−**01**
	*f*22	6.45E+01 ± 1.92E+01$^{\downarrow}$	**1.49**E+**01** ± **2.69**E+**00**$^{\updownarrow}$	**2.09**E+**01** ± **6.80**E+**00**
	*f*23	1.77E+02 ± 1.0E+01$^{\downarrow}$	**2.43**E+**01** ± **4.83**E+**00**$^{\uparrow}$	4.22E+01 ± 1.87E+01
	*f*24	3.99E+01 ± 6.08E+00$^{\downarrow}$	**2.67**E+**01** ± **4.61**E+**00**$^{\updownarrow}$	**1.23**E+**01** ± **4.28**E+**00**
	*f*25	7.22E+03 ± 2.56E+03$^{\downarrow}$	**1.70**E+**03** ± **1.66**E+**03**$^{\uparrow}$	4.18E+03 ± 1.06E+03
Expanded Multimodal Functions	*f*26	1.37E+01 ± 1.01E+00$^{\downarrow}$	**1.39**E+**00** ± **8.09**E−**01**$^{\updownarrow}$	**2.82**E+**00** ± **4.22**E−**01**
	*f*27	**1.33**E+**01** ± **2.01**E−**01** $^{\updownarrow}$	**1.31**E+**01** ± **3.38**E−**01**$^{\updownarrow}$	**1.26**E+**01** ± **4.91**E−**01**

Wilcoxons rank sum test at a 0.05 significance level is performed between DE+SABEA and the other two algorithms. $^{\downarrow}$ $^{\uparrow}$ $^{\updownarrow}$ represent that the performance of the corresponding algorithm is worse than, better than, and similar to that of the corresponding DE+SABEA, respectively.

within the updated search space, the following strategy is used:

$$x'_{ij}(k) = \begin{cases} LB_i(k) + (LB_i(k) - x_{ij}(k))\%(UB_i(k) - LB_i(k)) & x_{ij}(k) < LB_i(k) \\ UB_i(k) - (x_{ij}(k) - UB_i(k))\%(UB_i(k) - LB_i(k)) & x_{ij}(k) > UB_i(k), \end{cases} \tag{9}$$

where $x_{ij}(k)$ is the value of the ith variable of the jth individual at the kth generation in the existing population, $x'_{ij}(k)$ is the corresponding value in the new population, and $LB_i(k)$ and $UB_i(k)$ are the lower bound and upper bound, respectively, of the ith variable at the kth generation in the updated search space. Hence, if the values of the variable in the existing population are within the updated search space, these values are directly copied into the trial population. Otherwise, the values are modified according to the above equation.

When operating DABGA and SABEA on top of classic DE and CoDE, we use the same DE and CoDE settings as described above. For DABGA-specific settings, b and β are set to 1.25 and 0.97 respectively according to [27]. For our SABEA, the selecting fraction α is set to 0.20, the truncation fraction δ is 0.2, the sample size m is 6 and the bounding factor b is 1.25. When SABEA and DABGA are performed on DE, n_o (defined as n_{g0} in DABGA) is set to 50 and n_g is set to 5. When SABEA and DABGA are performed on CoDE, n_o (defined as n_{g0} in DABGA) is set to 80 and n_g is set to 8.

4.1.3. Experimental results

(a) Comparison of solution accuracy: In our experiment, the test functions shown in Table 1 are evaluated using DE, DE with DABGA (i.e., DE+DABGA), DE with SABEA (i.e., DE+SABEA), CoDE, CoDE with DABGA (i.e., CoDE+DABGA) and CoDE with SABEA (i.e., CoDE+SAEBA). For each test function, the dimension of the problem is set to 30. To ensure fair comparisons, all of the evaluations are terminated at 300,000 function evaluations. The performance of each algorithm is evaluated using the function error value. The function error value is defined as $|f(X) - f(X^*)|$, where X is the best solution found by the algorithm in a run and X^* is the global optimum of the test function. The averages and standard deviations of the function error values for the test functions obtained from 25 independent runs are summarized in Tables 2 and 3. In order to evaluate the statistical significance of the results, Wilcoxons rank sum test at a 0.05 significance level is conducted to perform pairwise comparisons between the algorithms. The test results are also indicated in Tables 2 and 3.

As shown in Table 2, DE+SABEA performs significantly better than, or at least comparable to, the classic DE in terms of solution accuracy. Comparing DE+SABEA with DE+DABGA for the traditional test functions $f_1 - f_{13}$, it is observed that

Table 3
Results of CoDE, CoDE+DABGA, CoDE+SABEA on 27 test functions.

	Function	CoDE Mean error± std. dev.	CoDE+DABGA Mean error ± std. dev.	CoDE+SABEA Mean error± std. dev.
	f1	5.01E−67 ± 5.95E−67$^{\downarrow}$	**3.24E−83 ± 2.98E−82**$^{\uparrow}$	2.71E−77 ± 1.98E−77
Unimodal	f2	3.28E−35 ± 2.76E−35$^{\downarrow}$	**4.28E−44 ± 1.21E−44**$^{\uparrow}$	7.21E−39 ± 5.78E−39
Functions	f3	2.68E−16 ± 3.69E−16$^{\downarrow}$	2.07E−01 ± 1.67E−01$^{\downarrow}$	**2.43E−17 ± 2.92E−17**
	f4	5.82E−16 ± 4.95E−16$^{\downarrow}$	1.83E−12 ± 4.85E−12$^{\downarrow}$	**6.60E−17 ± 6.53E−17**
	f5	2.52E−11 ± 4.71E−11$^{\downarrow}$	1.52E−09 ± 4.07E−09$^{\downarrow}$	**4.13E−12 ± 8.75E−12**
	f6	**0.00E+00 ± 0.00E+00**$^{\updownarrow}$	**0.00E+00 ± 0.00E+00**$^{\updownarrow}$	**0.00E+00 ± 0.00E+00**
	f7	2.82E−03 ± 1.29E−03$^{\downarrow}$	4.42E−02 ± 1.86E−02$^{\downarrow}$	**7.29E−04 ± 4.18E−04**
	f8	**3.63E−11 ± 0.00E+00**$^{\updownarrow}$	**3.63E−11 ± 0.00E+00**$^{\updownarrow}$	**3.63E−11 ± 0.00E+00**
	f9	**0.00E+00 ± 0.00E+00**$^{\updownarrow}$	**8.37E+00 ± 5.73E+00**$^{\updownarrow}$	**0.00E+00 ± 0.00E+00**
Multimodal	f10	**4.44E−15 ± 0.00E+00**$^{\updownarrow}$	**4.14E−15 ± 1.02E−15**$^{\updownarrow}$	**4.44E−15 ± 0.00E+00**
Functions	f11	**0.00E+00 ± 0.00E+00**$^{\updownarrow}$	1.84E−03 ± 3.34E−03$^{\downarrow}$	**0.00E+00 ± 0.00E+00**
	f12	**1.57E−32 ± 2.82E−48**$^{\updownarrow}$	**1.57E−32 ± 2.84E−48**$^{\updownarrow}$	**1.57E−32 ± 2.82E−48**
	f13	**1.34E−32 ± 2.24E−48**$^{\updownarrow}$	**1.64E−32 ± 2.96E−48**$^{\updownarrow}$	**1.34E−32 ± 2.24E−48**
	$f14$	**0.00E+00 ± 0.00E+00**$^{\updownarrow}$	**0.00E+00 ± 0.00E+00** $^{\updownarrow}$	**0.00E+00 ± 0.00E+00**
Unimodal	$f15$	1.69E−15 ± 3.95E−15$^{\downarrow}$	9.51E−01 ± 6.17E−01$^{\downarrow}$	**8.34E−17 ± 6.24E−17**
Functions	$f16$	1.05E+05 ± 6.25E+04$^{\downarrow}$	2.70E+05 ± 1.43E+05$^{\downarrow}$	**6.82E+04 ± 3.32E+04**
	$f17$	**5.81E−03 ± 1.38E−02**$^{\updownarrow}$	1.55E+04 ± 3.34E+03$^{\downarrow}$	**6.01E−03 ± 4.20E−03**
	$f18$	3.31E+02 ± 3.44E+02$^{\downarrow}$	2.60E+03 ± 3.73E+02$^{\downarrow}$	**1.56E+02 ± 6.38E+01**
	$f19$	1.60E−01 ± 7.85E−01$^{\downarrow}$	1.19E+01 ± 1.92E+01$^{\downarrow}$	**3.53E−02 ± 2.19E−02**
	$f20$	7.46E−03 ± 8.55E−03$^{\downarrow}$	2.78E−01 ± 1.69E−01$^{\downarrow}$	**2.12E−03 ± 1.34E−03**
Basic	$f21$	**2.01E+01 ± 1.41E−01**$^{\updownarrow}$	**2.06E+01 ± 3.40E−01**$^{\updownarrow}$	**2.01E+01 ± 1.07E−01**
Multimodal	$f22$	**0.00E+00 ± 0.00E+00**$^{\updownarrow}$	7.46E+00 ± 2.53E+00$^{\downarrow}$	**0.00E+00 ± 0.00E+00**
Functions	$f23$	4.15E+01 ± 1.16E+01$^{\updownarrow}$	**2.64E+01 ± 6.15E+00**$^{\uparrow}$	4.03E+01 ± 9.13E+00
	$f24$	**1.18E+01 ± 3.40E+00**$^{\updownarrow}$	3.03E+01 ± 7.96E+00$^{\downarrow}$	**1.25E+01 ± 3.64E+00**
	$f25$	3.05E+03 ± 3.80E+03$^{\downarrow}$	3.91E+03 ± 7.39E+03$^{\downarrow}$	**6.33E+02 ± 4.47E+02**
Expanded Multimodal	f26	**1.57E+00 ± 3.27E−01**$^{\updownarrow}$	**1.30E+00 ± 8.97E−01**$^{\updownarrow}$	**1.65E+00 ± 3.35E−01**
Functions	$f27$	**1.23E+01 ± 4.81E−01**$^{\updownarrow}$	**1.29E+01 ± 3.99E−01**$^{\updownarrow}$	**1.27E+01 ± 5.51E−01**

Wilcoxons rank sum test at a 0.05 significance level is performed between CoDE+SABEA and the other two algorithms. $^{\downarrow}$ $^{\uparrow}$ $^{\updownarrow}$ represent that the performance of the corresponding algorithm is worse than, better than, and similar to that of the corresponding CoDE+SABEA, respectively.

DE+SABEA is only worse than DE+DABGA for the functions f_1 and f_2. This is mainly due to the fact that f_1 and f_2 are relatively simple functions with no interacting variables. In such a case, DABGA's approach based on the value distribution of variables in a single generation can provide a good indication of the optimal solution. With the consideration of interacting variables, our SABEA does not directly utilize the variable's value distribution information in a single generation, which is easily distorted if there are interacting variables. Therefore, SABEA may perform slightly worse than DABGA for these simple functions. However, given the simplicity of these two test functions, both algorithms can achieve relatively small mean errors, as shown in Table 2.

For other traditional test functions, DE+SABEA outperforms DE+DABGA for the test functions f_3, f_4, f_5, f_7, f_8, f_{10}, and f_{11}. Among these functions, f_3 and f_5 have interacting variables. Thanks to SABEA's ability to deal with interacting variables, it can achieve much better results than DABGA for f_3 and f_5. The rest of the functions contain either a non-linear relationship or random numbers, which affect the value distribution of the variables. Thus, the distribution-based DABGA also performs worse than our SABEA. For f_6, f_9, f_{12} and f_{13}, classical DE can already achieve very small mean errors or global optimum. Thus, the results of DE+SABEA and DE+DABGA are comparable.

For the CEC 2005 benchmark functions $f_{14} - f_{27}$, DE+SABEA is significantly better than DE+DABGA for the unimodal functions $f_{14} - f_{18}$. These functions are shifted functions that introduce randomness of the value distribution of the variables. This may cause DABGA to be less effective as compared to our SABEA. For the basic multimodal functions f_{19}, f_{20}, DE+SABEA is also better than DE+DABGA. For the rest of the multimodal functions, the classical DE is not effective at dealing with the complexity of these functions. Thus, both DE+DABGA and DE+SABEA are limited by the performance of DE itself. Overall, 17 functions have been significantly improved by DE+SABEA over DE. In addition, 14 test functions show that DE+SABEA is better than DE+DABGA and 9 test functions, for which the two methods are comparable.

As shown in Table 3, the results achieved by CoDE are generally better than the results of the classic DE on these test functions. For instance, CoDE can achieve the global optimal solution for the test function f_{14}. When operating our SABEA method on top of CoDE, CoDE+SABEA is significantly better than, or at least comparable, to CoDE in terms of solution accuracy. It should be mentioned that for some test functions, such as $f_8 - f_{13}$, the effects of SABEA on CoDE are not as significant as those on the classic DE. This is mainly due to the fact that CoDE has a much better ability to search for the global minimum than the classic DE. When operating DABGA on top of CoDE, we observed that CoDE+DABGA is worse than CoDE for test functions such as $f_3 - f_5$, f_6, f_{11}, f_{13}, f_{15} and f_{20}. This is because CoDE can converge faster than the classic

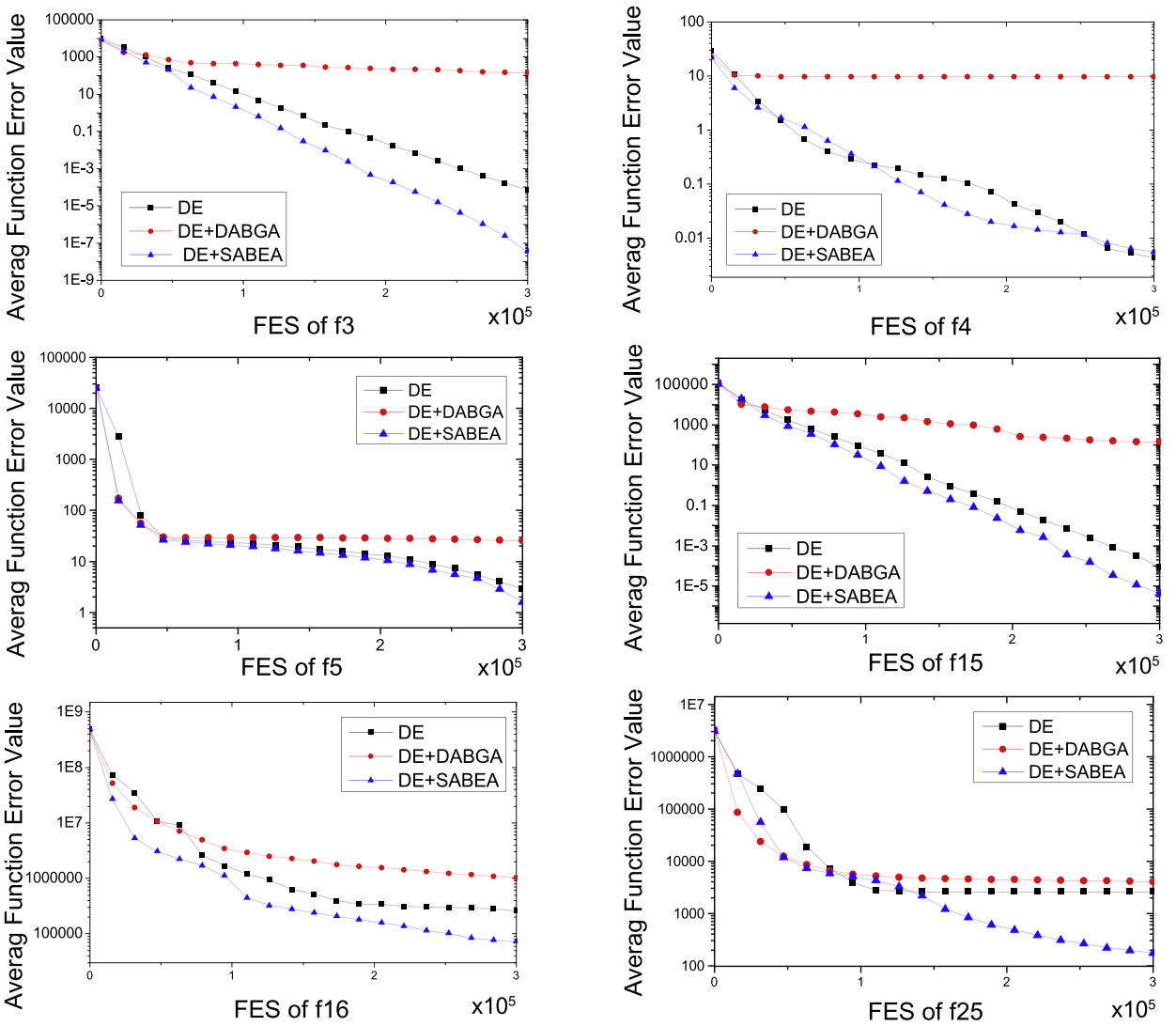

Fig. 1. Evolution of the mean function error values derived from DE, DE+DABGA and DE+SABEA.

DE and, if the updated boundaries by DABGA miss the global optimal, the effect will be more obvious. In contrast, our SABEA does not degrade the performance of CoDE for all the test functions. To compare CoDE+SABEA with CoDE+DABGA, CoDE+SABEA is significantly better than or at least comparable to, CoDE+DABGA for most of the test functions. Overall, there are 15 test functions where CoDE+SABEA is better than CoDE+DABGA and 9 test functions for which the two methods are comparable.

(b) Convergence performance and computational complexity: Apart from the solution accuracy, it is also important to evaluate the convergence speed of different algorithms. To show the convergence speed of the proposed SABEA method, we compared DE+SABEA with the classic DE and DE+DABGA and CoDE+SABEA with CoDE and CoDE+DABGA over six benchmark functions each with different characteristics. We chose two unimodal functions (f_3, f_4), one Rosenbrock function that is multimodal for $n > 3$ (f_5), one shifted function (f_{15}), one shifted rotated function (f_{16}) and one multimodal function (f_{25}). The convergence graphs of the algorithms over the test problems are shown in Figs. 1 and 2. The graphs are the mean best function error values of 25 independent runs. It can be seen that DE+SABEA and CoDE+SABEA have the fastest convergence speed in the two groups of comparisons. Both of these algorithms can also obtain the best solutions to the six test functions. It is also observed that DE+DABGA and CoDE+DABGA perform worse than DE and CoDE alone. This is mainly due to the complexity of test functions, such as variable linkage, rotation and shift. In these test functions, the updated boundaries of DABGA may miss the global optimal solution, which leads to the slow convergence.

To further investigate the computational complexity of the six tested a]lgorithms, the average search times of runs over all 27 test functions are measured. All algorithms are implemented in Matlab R2015b and the running times are measured

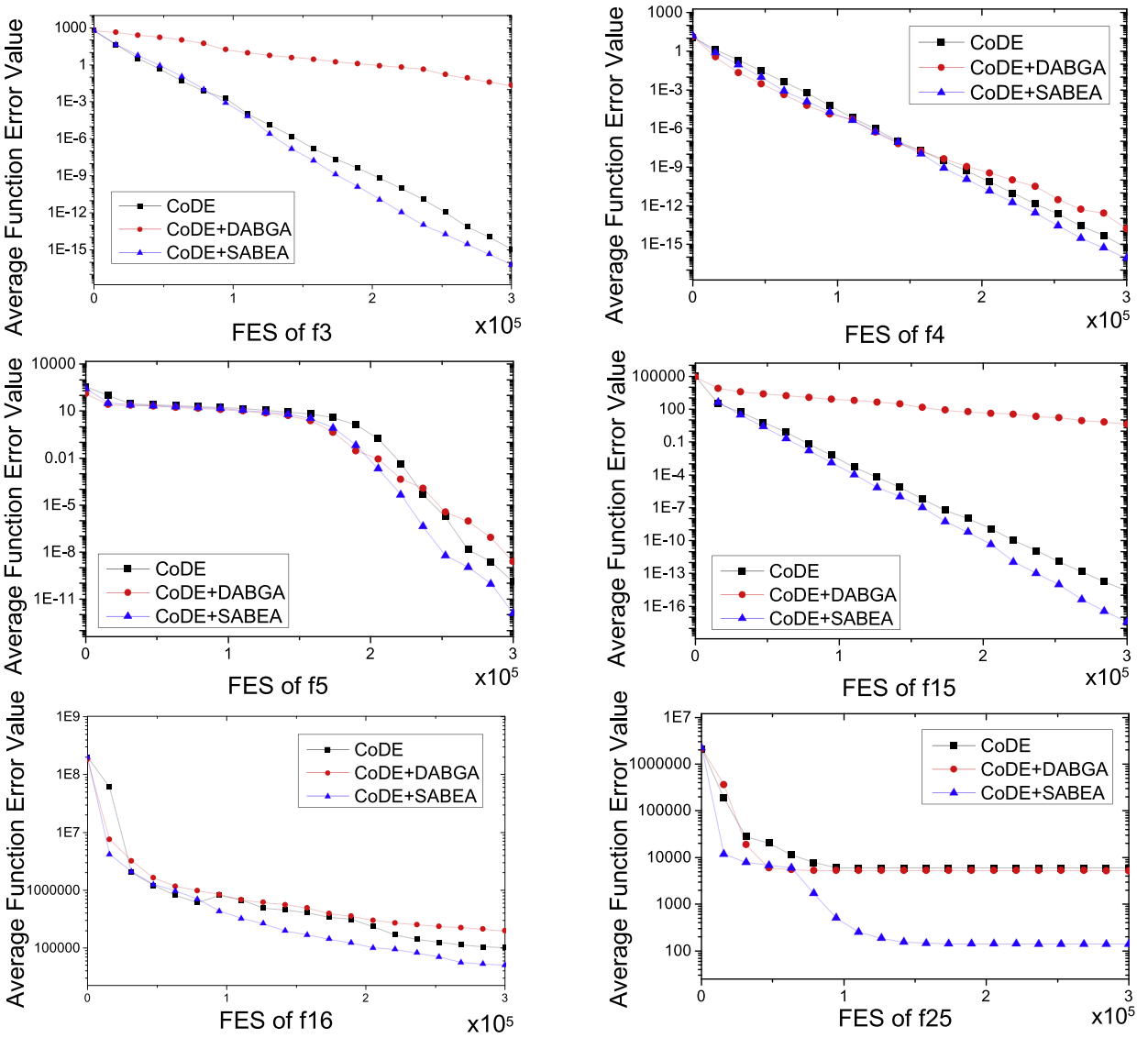

Fig. 2. Evolution of the mean function error values derived from CoDE, CoDE+DABGA and CoDE+SABEA.

Table 4
Average search time of the tested algorithms over 27 test functions.

Algorithm	Averaged search time (s)	Normalization
DE	11.19	1.00 (to DE)
DE + DABGA	12.39	1.11 (to DE)
DE + SABEA	12.07	1.09 (to DE)
CoDE	9.40	1.00 (to CoDE)
CoDE + DABGA	10.91	1.16 (to CoDE)
CoDE + SABEA	11.01	1.17 (to CoDE)

in a Dell Precision T3610 workstation with Intel(R) Xeon(R) CPU E5-1650v2@3.50 GHz and 16 GB RAM operated with a Microsoft Windows 10 system. To make a fair comparison, each run is performed with 300,000 function evaluations for all the algorithms and the corresponding results are shown in Table 4. It can be seen that both DABGA and our SABEA do not increase the search time significantly when operated on top of DE and CoDE. DABGA and SABEA are very close in terms of computational complexity, with less than 1 s difference. Memory usage is evaluated using the 'Profile' command in Matlab. For all the test functions, the maximum memory requirements for DE/CoDE+DABGA and DE/CoDE+SABEA are 40 MB and 55

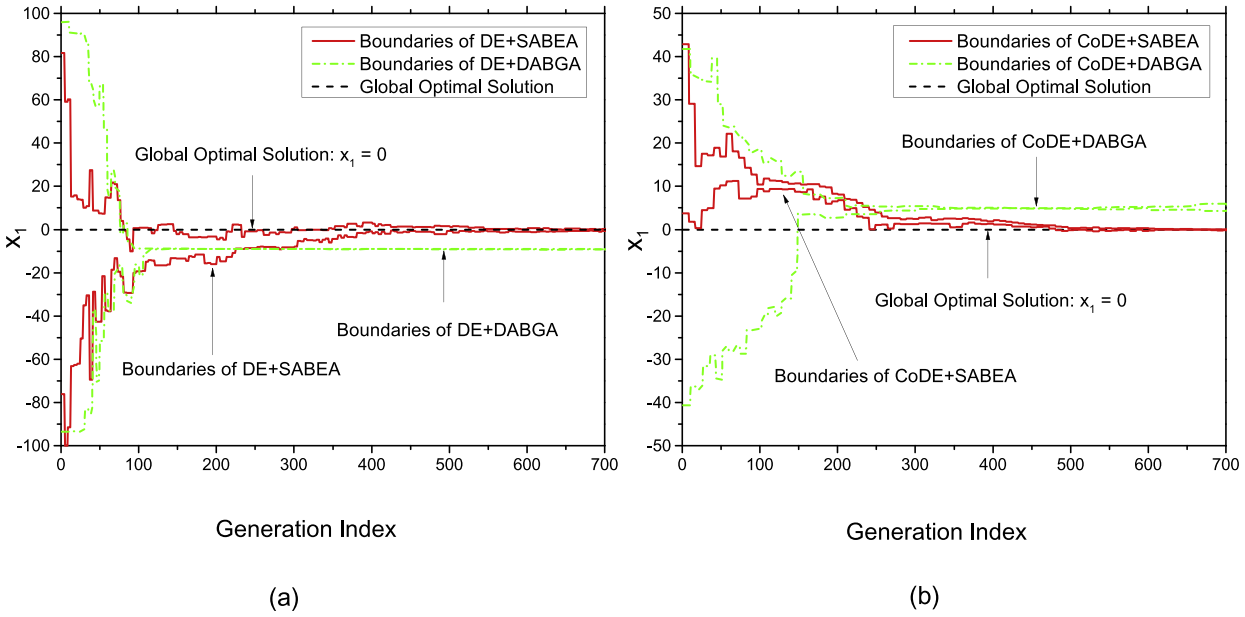

Fig. 3. Boundary curve on test function f_3.

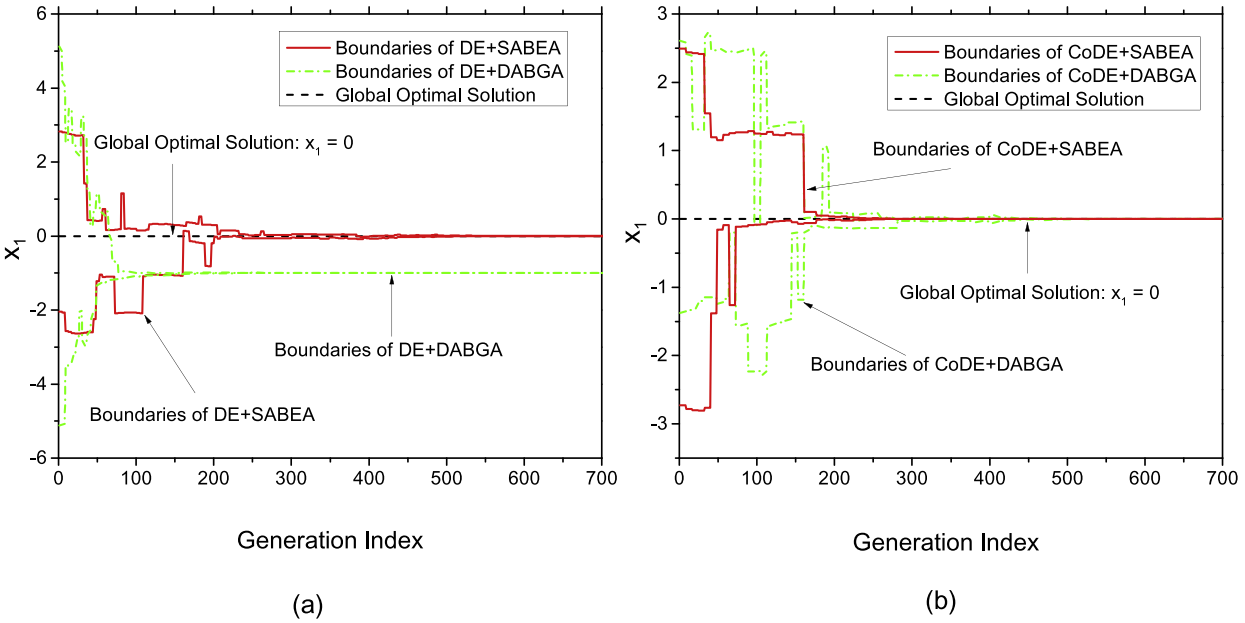

Fig. 4. Boundary curve on test function f_9.

MB, respectively. Our SABEA requires larger memory than DABGA. This is mainly because we need to store information of individuals over multiple generations rather than a single generation for search space updating. However, given a moderate memory configuration (usually $\geq$ 4GB) of a modern PC, the memory requirement of our SABEA algorithm is still affordable.

(c) Dynamics of boundary updating in the evolution process: For ABEA algorithms, it is of considerable interests to know how the boundary of a variable is dynamically updated during the evolution process. Figs. 3 and 4 are typical curves taken from one of the 25 tests of the test functions f_1 and f_9, when SABEA and DABGA are performed on DE and CoDE respectively. In these figures, the statistics of the variable x_1 over the whole population in each generation are shown. The boundaries in the figures are the upper and lower boundaries of variable x_1. The global optimal value of x_1 is also shown in these figures. It can be observed that for both DABGA and SABEA, the variable interval gradually shrinks, but does not decrease monotonically. This is mainly due to the introduction of the bounding factor b in both methods. In early generations, DABGA works well without missing the global optimal. However, in later generations, DABGA may miss the true optimum. While

Table 5
Impact of selecting fraction α.

α	f_3		f_5		f_{15}	
	Mean error	Std. dev.	Mean error	Std. dev.	Mean error	Std .dev.
0.05	2.80E−09	5.80E−09	1.16E+00	1.27E+00	2.33E−03	1.29E−04
0.1	**8.25E − 10**	2.93E−09	1.09E+00	1.20E+00	**7.14E − 10**	9.24E−10
0.2	1.09E−09	2.00E−09	**6.31E − 01**	1.12E+00	7.39E−10	8.24E−10
0.4	1.43E−08	2.77E−08	8.12E−01	1.23E+00	9.42E−09	4.22E−09
0.8	6.91E−07	9.28E−07	8.09E−01	1.33E+00	1.09E−08	2.81E−08

Table 6
Impact of truncation fraction δ.

δ	f_3		f_5		f_{15}	
	Mean error	Std. dev.	Mean error	Std. dev.	Mean error	Std. dev.
0.05	4.17E−09	5.54E−09	1.02E+00	1.92E+00	**5.89E − 10**	7.15E−10
0.1	**1.04E − 09**	5.21E−09	1.01E+00	1.98E+00	6.99E−10	8.81E−10
0.2	1.09E−09	2.00E−09	**6.31E − 01**	1.12E+00	7.39E−10	8.24E−10
0.4	4.28E−09	6.18E−9	**6.22E − 01**	1.32E+00	1.05E−09	2.14E−09
0.8	3.23E−08	6.07E−08	7.89E−01	2.01E+00	8.92E−09	1.07E−08

Table 7
Impact of sample size m.

m	f_3		f_5		f_{15}	
	Mean error	Std. dev.	Mean error	Std. dev.	Mean error	Std. dev.
3	4.78E−09	4.90E−09	1.66+01	1.06E+01	2.09E−09	4.71E−09
6	**1.09E − 09**	2.00E−09	**6.31E − 01**	1.12E+00	**7.39E − 10**	8.24E−10
12	8.31E−09	9.19E−09	9.24E−0	1.20E+00	1.01E−09	1.03E−09
24	3.34E−09	4.12E−08	1.54E+00	1.92E+00	8.25E−09	9.93E−09
48	1.11E−08	1.23E−08	1.72E+00	2.09E+00	9.94E−09	1.42E−09

SABEA may miss the true optimum in early generations for some cases (e.g., Fig. 3(b)), it shows a good ability to jump out of local optima and eventually approach the true optimum.

(d) Analysis of algorithm parameters: SABEA involves four key parameters: selecting fraction α, truncation fraction δ, sample size m and n_g. We study how different values of these parameters can affect the performance of our SABEA algorithm. For our purposes, we have selected three representative test functions f_3, f_5 and f_{15} and operate our SABEA on DE.

1) Impact of selecting fraction α:
The selecting fraction determines the number of individuals selected in the step 2 of SABEA. Given that $0 < \alpha < 1$, we varied the value of α to be 0.05, 0.1, 0.2, 0.4, and 0.8, while the other parameters remained the same as described in Section 4.1.2. For each parameter setting, 100 independent runs were performed. The results presented in Table 5 show that the α value should not be too small (i.e., less than 0.1) or too large (i.e., close to 1). If α is too small, the algorithm will be easily trapped into local optima due to the loss of population diversity. If α is too large, more individuals will be used to update the new boundaries, which makes the updated boundaries closer to the existing ones. Thus, a large α will slow down the search speed. It appears that the value of α should be set between 0.1 and 0.2.

2) Impact of truncation fraction δ:
The truncation fraction δ affects the number of individuals used for building the probabilistic model M in step 3 of SABEA. Given that $0 < \delta < 1$, we varied the value of δ to be 0.05, 0.1, 0.2, 0.4, and 0.8, while the other parameters were set the same as described in Section 4.1.2. The results presented in Table 6 show that the δ value should also not be too large or too small, similar to α. For instance, we observed that the algorithm is trapped into local optima of f_5, when δ is small (i.e., 0.05 or 0.1). In general, the results show that the value of δ has less effect on the algorithm performance than the value of α. This is mainly due to the probabilistic nature of the EDA-inspired sampling strategy adopted in step 3 of our algorithm. It appears that the $\delta = 0.2$ is a promising setting.

3) Impact of sample size m:
The sample size m determines the number of individuals sampled in step 3 of SABEA. Here we set the value of m as 3, 6, 12, 24 and 48 to investigate its impact. When m becomes large, more individuals are sampled for updating the search space of a given problem. The results in Table 7 show that the solution accuracy of the algorithm will decrease when m is large. As the probabilistic sampling in step 3 of SABEA is only used to prevent the early convergence of the algorithm,

Table 8
Impact of n_g.

n_g	f_3		f_5		f_{15}	
	Mean error	Std. dev.	Mean error	Std. dev.	Meanerror	Std. dev.
3	**4.15E − 10**	5.12E−10	1.43E+00	1.62E+00	**1.70E − 10**	3.78E−10
5	1.09E−09	2.00E−09	**6.31E − 01**	1.23E+00	7.39E−10	8.24E−10
7	7.15E−09	1.21E−08	6.81E−01	1.37E+00	7.33E−09	9.01E−09
9	2.18E−08	3.92E−08	7.13E−01	1.75E+00	1.21E−08	1.16E−08
11	7.03E−8	9.17E−08	7.19E−01	1.53E+00	4.26E−08	7.29E−08

the value of m should not be too large to include too many individuals in the sampling process. In general, $m = 6$ leads to better performance.

4) Impact of n_g:
The parameter n_g determines the total number of individuals utilized to update the search space of the variables. To assess the impact of n_g, we varied the value of n_g to 3, 5, 7, 9 and 11 while the other parameters remained the same as described in Section 4.1.2. The results presented in Table 8 show that the value of n_g should not be set to be too large. Even though a large n_g can increase the population diversity, it causes the search space updating to be performed less frequently. Furthermore, it increases the computational time and memory requirement for sorting and storing the individual information. Based on our observations, it appears that the value of the parameter n_g should be set between 5 and 8.

4.2. Experiments on the model calibration problems

In this section, we apply the SABEA method to a model calibration problem in an agent-based simulation. The objective of this class of problems is to calibrate the values of the model parameters in a simulation such that the simulation output matches some desired outcomes (e.g., real-world behavior of a crowd). Due to the dynamic nature of simulation, the model calibration problem has become a challenging issue to ensure the validity of simulation. Traditional deterministic optimization algorithms such as simplex algorithms [36] and regression methods [20] have been utilized to solve these problems, but they may become easily trapped into local optima. To avoid such problems, EA-based approaches have been adopted. Due to its simplicity and popularity, GA has been mostly used for model calibration in agent-based simulation [3,15,29,35]. Thus, for this class of problem, we use GA as our base algorithm. In the following sub-sections, we first give the definition of our model calibration problem. The simulation models, scenarios and experiment settings will then be described. Lastly, the experimental results using the GA, GA+DABGA and GA+SABEA methods will be presented and compared.

4.2.1. Definition of model calibration problem

The model calibration problem aims to tune the parameter settings of a simulation model so that the simulation output can match the desired outcomes. Such problems can be considered to be a special type of continuous optimization problems. The objective function in such a problem can be defined as $f(X) = |O_m^{\text{sim}} - O_m^{\text{desired}}|$, where O_m^{sim} is the simulation output using an evaluation metric m, and O_m^{desired} is the desired output. Here, the input variables X of the objective function are the model parameters. The settings of X will affect the simulation output O_m^{sim}. However, the relationship between the model parameters and simulation output is usually not described by a mathematical formula. In agent-based simulation, the relationship is governed by dynamic interactions of agents and the whole simulation, and thus can be considered to be a complex system. To avoid such problems, stochastic global search methods, such as genetic algorithm, have recently become a popular method.

In our work, we apply the SABEA method to solve the model calibration problem in an agent-based crowd simulation [22]. The crowd model calibration problem requires tuning parameters so that the simulation results match the desired objective crowd behaviors. To evaluate the quality of a simulated crowd behavior, we adopt the following density-based distance measure [38] as our objective function:

$$D(X) = |O_m^{\text{sim}} - O_m^{\text{desired}}| = \sum_{t=1}^{T} \left(\sum_{\text{q}=1}^{W \times L} (|\rho_{\text{q}}^t - \varrho_{\text{q}}^t|) \right), \tag{10}$$

where X is the model parameter setting, T is the total simulation time steps, $W \times L$ is the number of representative points in the simulation area of the crowd simulation and ρ_q^t and ϱ_q^t represent the crowd density value at the q_{th} representative point at time step t from the simulated result and real-world data respectively. Here, we evenly divide the virtual world in a crowd simulation into $W \times L$ discrete grids with the center of each cell as a representative point. To measure the local density at a representative point's location χ of time step t, the following formula is used:

$$\sum_{i=1}^{C} \frac{1}{\pi R^2} \exp\left[\frac{-||r_i(t) - \chi||^2}{R^2} \right], \tag{11}$$

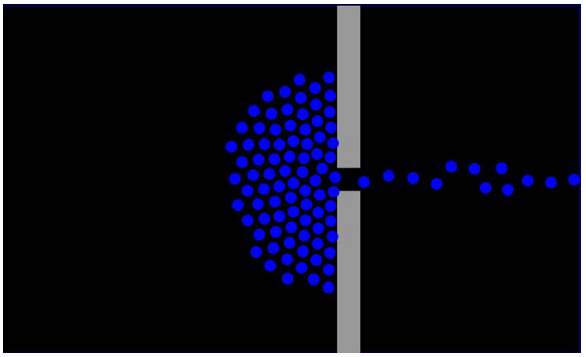

Fig. 5. Scenario 1.

Fig. 6. Scenario 2.

where C is the number of agents in the virtual world; $r_i(t)$ is the position of the ith agent at time step t and R is the kernel radius (e.g., R =2m). The greater the R, the smaller the variance of the local density values [11].

4.2.2. Simulation models and case studies

In this sub-section, we briefly introduce the crowd simulation model (i.e., social-force model) we used and then give descriptions of the simulation scenarios.

(a) Social-force model: The social-force model proposed by Helbing et al. [10] is one of the most popular crowd simulation models. In the social force model, the motion of agents is governed by virtual forces that can be expressed as:

$$f_i = m_i \frac{dv_i}{dt} = f_{i0} + \sum_{j(j\neq i)} f_{ij} + \sum_{w} f_{iw}, \tag{12}$$

where m_i is the mass of the subject agent, $\frac{dv_i}{dt}$ is the acceleration rate of the i_{th} agent, f_{i_0} is the attractive force from the goal, f_{ij} is the repulsive force from other agents and f_{iw} is the repulsive force from static obstacles such as walls. The detailed formulation of each force can be found in [10]. In the formulation of these forces, there are several key parameters to be tuned, including A, B, $k1$, $k2$, and τ where A reflects the strength of interaction, B determines the interaction range, $k1$ determines the obstructions of physical interactions, $k2$ determines the obstruction effects of physical interactions and τ stands for the time for an agent to adapt the actual velocity to the preference velocity.

(b) Case studies: The two scenarios we simulate with agent-based crowd simulation are as follows:

Case 1: This first case study is to calibrate a crowd evacuation scenario, where individuals are trying to evacuate from a room, as shown in Fig. 5. In this scenario, individuals are initially distributed randomly and then run towards the exit gate to evacuate the room. The observed area is evenly divided into 7 × 7 cells. 49 representative points are used to measure the local density as defined in Eq. (10). In this case study, the desired crowd behavior is set as the simulation output generated by the social-force model with some default values of its parameters. Next, the social-force model is initialized with another set of random parameter settings, and we use GA, GA+DABGA and GA+SABEA to tune the parameters so as to reproduce the desired behaviors.

Case 2: This scenario is to calibrate a crossing scenario where agents pass through a corridor in one direction as shown in Fig. 6, where the objective crowd behaviors are obtained from real-world data [12]. In this scenario, there are about 60 pedestrians crossing a 2 m × 14 m corridor area and the positions of the agents are recorded every 1/16 s. The observed area is evenly divided into 5 × 10 cells. 50 representative points are used to measure the local density as defined in Eq. (10).

Table 9
Parameters to be tuned for the social-force model.

Parameter	Lower bound	Upper bound	Default
A	1000	5000	1000
B	0.01	1	0.08
$K1$	10,000	300,000	120,000
$K2$	1000	300,000	240,000
τ	0.1	1	0.5

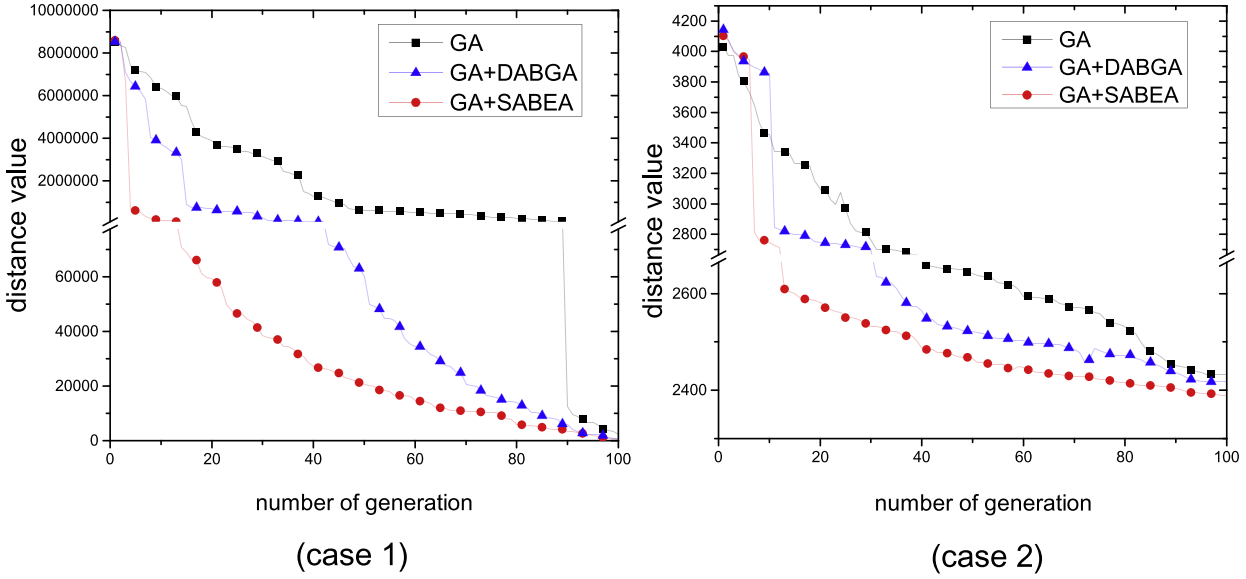

Fig. 7. Evolution curves of the best fitness values in case 1 and case 2, respectively.

In this case study, the desired crowd behavior is based on video data. The simulation is set based on the following rules: (a) the initial and destination positions of each agent are set to the first and the last position of the corresponding pedestrian in the video and (b) the desired speed of each agent is set as the average speed of its first three recorded steps. Next, GA, GA+DABGA and GA+SABEA are utilized to calibrate the social force model such that the simulation output matches the desired behaviors.

4.2.3. Experiment settings

We use the social-force model to simulate the two crowd scenarios described above. To ensure that the simulated results match the desired output, we tune the five key parameters A, B, $k1$, $k2$ and τ in the model. Table 9 shows the initial search space and default values for these parameters. For scenario case 1, we use the simulation results with the default parameter settings as the desired behaviors. In this case, the optimal values of the parameters are the default values set in Table 9. For scenario case 2, the desired behaviors correspond to the data directly extracted from the video. The optimal values of the parameters are obtained when the simulated results best match the real-world data.

The settings of GA, DABGA and SABEA are as follows. For all three algorithms, the population size is set as $N_{pop} = 50$ and the total number of function evaluations (i.e., $N_{pop} * N_{Gen}$) is set to 50,000. The crossover rate Pc and mutation rate Pm are set to 0.25 and 0.01, respectively. For DABGA, the bounding fraction $\beta = 0.97$, the bounding factor $b = 1.25$ and the parameter intervals are updated every three generations, i.e., $n_g = 3$. For SABEA, the selecting fraction $\alpha = 0.2$, the truncation fraction δ=0.2, the sample size m=5 and the bounding factor $b = 1.25$. The parameter intervals are also updated every three generations (i.e., n_g= 3).

4.2.4. Simulation results

Fig. 7 illustrates the average of the best distance values (i.e., $D(X)$ in Eq. (10)) found by the three algorithms versus the number of generations. Overall, it is observed that the simulated crowd behaviors become closer to the objective crowd behavior during the evolutionary process. Compared to GA and GA+DABGA, GA+SABEA has a higher convergence rate and reaches the smallest distance values with the given number of generations. It can also be observed that the distance values are smaller in case 1 than in case 2. This may be because the desired behaviors in case 1 are the sample simulation output, which can be exactly reproduced given the parameters' values are set as default values, while the desired behaviors (i.e., real-world data) in case 2 can only be approximated with the social-force model.

Table 10
Statistics of the three algorithms on two scenario cases.

Scenario	Algorithm	Minimum	Average	Maximum	Deviation
Case1	GA	1823.18	2061.68↓	2232.12	97.81
	GA+DABGA	710.76	888.57↓	942.16	59.26
	GA+SABEA	551.94	**628.14**	703.84	44.13
Case2	GA	2405.61	2421.15↓	2448.37	12.58
	GA+DABGA	2390.54	2410.28↓	2439.82	12.80
	GA+SABEA	2383.47	**2392.23**	2419.83	11.18

↓ Mean the results are significantly worse than those of the GA+SABEA by two-sample *t*-test with pooled or unpooled variance estimate at 0.05.

Table 10 lists the final best distance values by the three algorithms. It can be seen that the average of the best fitness values found by GA+SABEA is the best for both scenarios. In addition, a two-sample *t*-test with a pooled or unpooled variance estimate is performed to verify the statistical significance of the results. The *t*-test results illustrate that GA+SABEA is statistically better than GA and GA+DABGA at a 0.05 significance level.

5. Conclusions and future work

To improve the accuracy and efficiency of EAs for continuous optimization problems, this paper proposes a sampling-based adaptive bounding evolution algorithm (SABEA) that can dynamically adjust the search space of a problem during the evolution process. To update the variable boundaries of a given problem, the proposed method leverages the information of individuals collected via two sampling strategies, namely fitness-based bounding and probabilistic sampling-based bounding. The fitness-based bounding selects a set of individuals with better fitness over multiple generations (i.e., n_g generations) and we assume that the search space constrained by these individuals is more likely to contain the global optimum. To avoid being trapped into local optima, the probabilistic sampling-based bounding is used to probabilistically select another set of individuals based on Gaussian distribution. The two sets of individuals selected by the two sampling strategies are then combined and used to update the search space every n_g generations.

The proposed SABEA method is a general search space updating strategy that works on different EA methods. To test the effectiveness and generality of SABEA, we applied it on top of two EAs, (i.e., the classic DE and its variant CoDE) on 27 test functions with different characteristics, such as uni-modality, multi-modality, rotation, shift and ill-condition. The results show that SABEA is effective at improving the performance of these EAs in terms of solution accuracy and convergence speed. Our experimental study also compares the results of SABEA with results obtained by another adaptive bounding evolutionary algorithm called DABGA. The results show that SABEA is significantly better than, or at least comparable to, DABGA for most of the test functions. Moreover, as a practical engineering problem, the model calibration problem for agent-based simulation, is used to evaluate the performance of SABEA on real-world problems. In this model calibration experiment, SABEA achieves smaller calibration errors compared to the traditional GA-based calibration method and the DABGA method in two different cases.

This work is also expected to call more attention to the use of an adaptive search space updating strategy in EAs for continuous optimization problems. There are several future directions. One promising direction is to investigate other probabilistic models for probabilistic sampling to further enhance the method's ability to jump out of local optima. Another interesting direction is to extend the proposed method for large-scale optimization problems (e.g., problems with hundreds of variables). Lastly, applying the proposed method for more real-world applications [21] is also an interesting research topic.

Acknowledgments

This work is supported by National Natural Science Foundation of China (Grant no. 61502370), Natural Science Basic Research Plan in Shaanxi Province of China (Program no. 2016JQ6003), Fundamental Research Funds for the Central Universities (Grant no. JB150305) and China 111 Project (No. B16037) .

Appendix A. Benchmark functions

(1) $f_1(x) = \sum_{i=1}^{n} x_i^2;\ -100 \le x_i \le 100;\ f_1^*(X) = f_1^*(0,\ldots,0) = 0$
(2) $f_2(x) = \sum_{i=1}^{n} |x_i| + \prod_{i=1}^{n} |x_i|;\ -10 \le x_i \le 10;\ f_2^*(X) = f_2^*(0,\ldots,0) = 0$
(3) $f_3(x) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j\right)^2;\ -100 \le x_i \le 100;\ f_3^*(X) = f_3^*(0,\ldots,0) = 0$
(4) $f_4(x) = \max_i \{|x_i|, 1 \le i \le n\};\ -100 \le x_i \le 100;\ f_4^*(X) = f_4^*(0,\ldots,0) = 0$
(5) $f_5(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2];\ -30 \le x_i \le 30;\ f_5^*(X) = f_5^*(1,\ldots,1) = 0$
(6) $f_6(x) = \sum_{i=1}^{n} (\lfloor x_i + 0.5 \rfloor)^2;\ -100 \le x_i \le 100;\ f_6^*(X) = f_6^*(x_1,\ldots,x_n) = 0;\ -0.5 \le x_i \le 0.5$
(7) $f_7(x) = \sum_{i=1}^{n} i x_i^4 + random[0,1);\ -1.28 \le x_i \le 1.28;\ f_7^*(X) = f_7^*(0,\ldots,0) = 0$
(8) $f_8(x) = \sum_{i=1}^{n} (-x_i)\sin(\sqrt{|x_i|}) + n * 418.98288727243369;\ -500 \le x_i \le 500;\ f_8^*(X) = f_8^*(0,\ldots,0) = 0$

(9) $f_9(x) = \sum_{i=1}^{n}[x_i^2 - 10cos(2\pi x_i) + 10];\ -5.12 \le x_i \le 5.12;\ f_9^*(X) = f_9^*(0,\ldots,0) = 0$

(10) $f_{10}(x) = -20\exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2}) - \exp(\frac{1}{n}\sum_{i=1}^{n}cos2\pi x_i) + 20 + e;\ -32 \le x_i \le 32;\ f_{10}^*(X) = f_{10}^*(0,\ldots,0) = 0$

(11) $f_{11}(x) = \frac{1}{4000}\sum_{i=1}^{n}x_i^2 - \prod_{i=1}^{n}\cos(\frac{x_i}{\sqrt{i}} + 1);\ -600 \le x_i \le 600;\ f_{11}^*(X) = f_{11}^*(0,\ldots,0) = 0$

(12) $f_{12}(x) = \frac{\pi}{n}\{10sin^2(\pi y_i) + \sum_{i=1}^{n-1}(y_i - 1)^2[1 + sin^2(3\pi x_{i+1})] + (y_n - 1)^2\} + \sum_{i=1}^{n}u(x_i, 5, 100, 4);$

where, $y_i = 1 + \frac{1}{4}(x_i + 1)$ and $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, x_i > a \\ 0, -a < x_i < a \\ k(-x_i - a)^m, x_i < -a \end{cases}$ $-50 \le x_i \le 50;\ f_{12}^*(X) = f_{12}^*(-1,\ldots,-1) = 0$

(13) $f_{13}(x) = 0.1\{sin^2(3\pi x_1) + \sum_{i=1}^{n-1}(x_i - 1)^2[1 + sin^2(3\pi x_{i+1})] + (x_n - 1)[1 + sin^2(2\pi x_n)]\} + \sum_{i=1}^{n}u(x_i, 5, 100, 4)$

$-50 \le x_i \le 50;\ f_{13}^*(X) = f_{13}^*(1,\ldots,1) = 0$

(14) $f_{14}(x)$: Shifted Sphere Function; $-100 \le x_i \le 100;\ f_{14}^*(X) = -450$

(15) $f_{15}(x)$: Shifted Schwefels Problem 1.2; $-100 \le x_i \le 100;\ f_{15}^*(X) = -450$

(16) $f_{16}(x)$: Shifted Rotated High Conditioned Elliptic Function; $-100 \le x_i \le 100;\ f_{16}^*(X) = -450$

(17) $f_{17}(x)$: Shifted Schwefels Problem 1.2 with Noise in Fitness; $-100 \le x_i \le 100;\ f_{17}^*(X) = -450$

(18) $f_{18}(x)$: Schwefels Problem 2.6 with Global Optimum on Bounds; $-100 \le x_i \le 100;\ f_{18}^*(X) = -310$

(19) $f_{19}(x)$: Shifted Rosenbrocks Function; $-100 \le x_i \le 100;\ f_{14}^*(X) = 390$

(20) $f_{20}(x)$: Shifted Rotated Griewanks Function without Bounds; $0 \le x_i \le 600;\ f_{14}^*(X) = -180$

(21) $f_{21}(x)$: Shifted Rotated Ackleys Function with Global Optimum on Bounds; $-32 \le x_i \le 32;\ f_{14}^*(X) = -140$

(22) $f_{22}(x)$: Shifted Rastrigins Function; $-5 \le x_i \le 5;\ f_{22}^*(X) = -330$

(23) $f_{23}(x)$: Shifted Rotated Rastrigins Function; $-5 \le x_i \le 5;\ f_{23}^*(X) = -330$

(24) $f_{24}(x)$: Shifted Rotated Weierstrass Function; $-0.5 \le x_i \le 0.5;\ f_{24}^*(X) = 90$

(25) $f_{25}(x)$: Schwefels Problem 2.13; $-\pi \le x_i \le \pi;\ f_{25}^*(X) = -460$

(26) $f_{26}(x)$: Expanded Extended Griewanks plusRosenbrocks Function; $3 \le x_i \le 1;\ f_{14}^*(X) = -130$

(27) $f_{27}(x)$: Shifted Rotated Expanded Scaffers F6; $-100 \le x_i \le 100;\ f_{14}^*(X) = -300$

References

[1] B. Akay, D. Karaboga, A modified artificial bee colony algorithm for real-parameter optimization, Inf. Sci. 192 (2012) 120–142.
[2] I. Boussaïd, J. Lepagnot, P. Siarry, A survey on optimization metaheuristics, Inf. Sci. 237 (2013) 82–117.
[3] B. Calvez, G. Hutzler, Automatic tuning of agent-based models using genetic algorithms, in: Multi-Agent-Based Simulation VI, Springer, 2005, pp. 41–57.
[4] D. Dasgupta, Z. Michalewicz, Evolutionary algorithms in engineering applications, Springer Science & Business Media, 2013.
[5] A.B. Djurišić, Elite genetic algorithms with adaptive mutations for solving continuous optimization problems–application to modeling of the optical constants of solids, Opt. Commun. 151 (1) (1998) 147–159.
[6] W. Dong, T. Chen, P. Tino, X. Yao, Scaling up estimation of distribution algorithms for continuous optimization, IEEE Trans. Evol. Comput. 17 (6) (2013) 797–822.
[7] J.A. Egea, D. Henriques, T. Cokelaer, A.F. Villaverde, A. MacNamara, D.-P. Danciu, J.R. Banga, J. Saez-Rodriguez, Meigo: an open-source software suite based on metaheuristics for global optimization in systems biology and bioinformatics, BMC Bioinform. 15 (1) (2014) 136.
[8] L. Feng, Y.-S. Ong, A.-H. Tan, I.W. Tsang, Memes as building blocks: a case study on evolutionary optimization+ transfer learning for routing problems, Memetic Comput. 7 (3) (2015) 159–180.
[9] M. Hauschild, M. Pelikan, An introduction and survey of estimation of distribution algorithms, Swarm Evol. Comput. 1 (3) (2011) 111–128.
[10] D. Helbing, I. Farkas, T. Vicsek, Simulating dynamical features of escape panic, Nature 407 (6803) (2000) 487–490.
[11] D. Helbing, A. Johansson, H.Z. Al-Abideen, Dynamics of crowd disasters: an empirical study, Phys. Rev. E 75 (4) (2007) 046109.
[12] Hermes (2011), Hermes project: http://www.asim.uni-wuppertal.de/en/research/hermes.html.
[13] J.H. Holland, Adaptation in Natural and Artificial Systems. An Introductory Analysis with Application to Biology, Control, and Artificial Intelligence, University of Michigan Press, Ann Arbor, MI, 1975.
[14] D. Jia, G. Zheng, M.K. Khan, An effective memetic differential evolution algorithm based on chaotic local search, Inf. Sci. 181 (15) (2011) 3175–3187.
[15] A. Johansson, D. Helbing, P.K. Shukla, Specification of the social force pedestrian model by evolutionary adjustment to video tracking data, Adv. Complex Syst. 10 (supp02) (2007) 271–288.
[16] C.-F. Juang, Y.-H. Chen, Y.-H. Jhan, Wall-following control of a hexapod robot using a data-driven fuzzy controller learned through differential evolution, IEEE Trans. Ind. Electron. 62 (1) (2015) 611–619.
[17] D.E. Knuth, The Art of Computer Programming vol. 1, Fundamental Algorithms, Addison-Wesley, Reading, MA 9 (1968) 364–369.
[18] Y.-D. Kwon, S.-B. Kwon, S.-B. Jin, J.-Y. Kim, Convergence enhanced genetic algorithm with successive zooming method for solving continuous optimization problems, Comput. Struct. 81 (17) (2003) 1715–1725.
[19] P. Larranaga, J.A. Lozano, Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation, 2, Springer Science & Business Media, 2002.
[20] K.H. Lee, M.G. Choi, Q. Hong, J. Lee, Group behavior from video: a data-driven approach to crowd simulation, in: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation, Eurographics Association, 2007, pp. 109–118.
[21] L. Luo, H. Yin, W. Cai, J. Zhong, M. Lees, Design and evaluation of a data-driven scenario generation framework for game-based training, IEEE Trans. Comput. Intell. AI Games (2016), doi:10.1109/TCIAIG.2016.2541168.
[22] L. Luo, S. Zhou, W. Cai, M.Y.H. Low, F. Tian, Y. Wang, X. Xiao, D. Chen, Agent-based human behavior modeling for crowd simulation, Comput. Animat. Virtual Worlds 19 (3–4) (2008) 271–281.
[23] Y. Marinakis, G.-R. Iordanidou, M. Marinaki, Particle swarm optimization for the vehicle routing problem with stochastic demands, Appl. Soft Comput. 13 (4) (2013) 1693–1704.
[24] S. Mitra, I.E. Grossmann, J.M. Pinto, N. Arora, Optimal production planning under time-sensitive electricity prices for continuous power-intensive processes, Comput. Chem. Eng. 38 (2012) 171–184.

[25] H. Mühlenbein, T. Mahnig, A.O. Rodriguez, Schemata, distributions and graphical models in evolutionary optimization, J. Heuristics 5 (2) (1999) 215–247.
[26] Q.-K. Pan, L. Wang, L. Gao, W. Li, An effective hybrid discrete differential evolution algorithm for the flow shop scheduling with intermediate buffers, Inf. Sci. 181 (3) (2011) 668–685.
[27] J.-X. Peng, K. Li, S. Thompson, P.A. Wieringa, Distribution-based adaptive bounding genetic algorithm for continuous optimisation problems, Appl. Math. Comput. 185 (2) (2007) 1063–1077.
[28] A.K. Qin, V.L. Huang, P.N. Suganthan, Differential evolution algorithm with strategy adaptation for global numerical optimization, IEEE Trans. Evol. Comput. 13 (2) (2009) 398–417.
[29] F. Stonedahl, U. Wilensky, Finding forms of flocking: Evolutionary search in abm parameter-spaces, in: Multi-Agent-Based Simulation XI, Springer, 2010, pp. 61–75.
[30] R. Storn, K. Price, Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces, J. Global Optim. 11 (4) (1997) 341–359.
[31] P.N. Suganthan, N. Hansen, J.J. Liang, K. Deb, Y.-P. Chen, A. Auger, S. Tiwari, Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization, KanGAL report (2005).
[32] Y. Wang, Z. Cai, Q. Zhang, Differential evolution with composite trial vector generation strategies and control parameters, IEEE Trans. Evol. Comput. 15 (1) (2011) 55–66.
[33] Y. Wang, B. Li, T. Weise, Estimation of distribution and differential evolution cooperation for large scale economic load dispatch optimization of power systems, Inf. Sci. 180 (12) (2010) 2405–2420.
[34] Y. Wang, B. Li, T. Weise, J. Wang, B. Yuan, Q. Tian, Self-adaptive learning based particle swarm optimization, Inf. Sci. 181 (20) (2011) 4515–4538.
[35] D. Wolinski, S. J Guy, A.-H. Olivier, M. Lin, D. Manocha, J. Pettré, Parameter estimation and comparative evaluation of crowd simulations, Comput. Graph. Forum 33 (2) (2014) 303–312.
[36] K. Yamaguchi, A.C. Berg, L.E. Ortiz, T.L. Berg, Who are you with and where are you going? in: Proceedings of 2011 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2011, pp. 1345–1352.
[37] X. Yao, Y. Liu, G. Lin, Evolutionary programming made faster, IEEE Trans. Evol. Comput. 3 (2) (1999) 82–102.
[38] J. Zhong, N. Hu, W. Cai, M. Lees, L. Luo, Density-based evolutionary framework for crowd model calibration, J. Comput. Sci. 6 (2015) 11–22.
[39] J.-h. Zhong, J. Zhang, Z. Fan, Mp-eda: a robust estimation of distribution algorithm with multiple probabilistic models for global continuous optimization, in: Proceedings of the Asia-Pacific Conference on Simulated Evolution and Learning, Springer, 2010, pp. 85–94.