Combining Multiobjective Optimization with Differential Evolution to Solve Constrained Optimization Problems

Yong Wang, *Member, IEEE,* and Zixing Cai, *Senior Member, IEEE*

***Abstract*—During the past decade, solving constrained optimization problems with evolutionary algorithms has received considerable attention among researchers and practitioners. Cai and Wang's method (abbreviated as CW method) is a recent constrained optimization evolutionary algorithm proposed by the authors. However, its main shortcoming is that a trial-and-error process has to be used to choose suitable parameters. To overcome the above shortcoming, this paper proposes an improved version of the CW method, called CMODE, which combines multiobjective optimization with differential evolution to deal with constrained optimization problems. Like its predecessor CW, the comparison of individuals in CMODE is also based on multiobjective optimization. In CMODE, however, differential evolution serves as the search engine. In addition, a novel infeasible solution replacement mechanism based on multiobjective optimization is proposed, with the purpose of guiding the population toward promising solutions and the feasible region simultaneously. The performance of CMODE is evaluated on 24 benchmark test functions. It is shown empirically that CMODE is capable of producing highly competitive results compared with some other state-of-the-art approaches in the community of constrained evolutionary optimization.**

***Index Terms*—Constrained optimization problems, constraint-handling technique, differential evolution, multiobjective optimization.**

I. Introduction

In REAL-WORLD applications, most optimization problems are subject to different types of constraints. These problems are known as constrained optimization problems (COPs). In the minimization sense, general COPs can be formulated as follows:

$$\begin{aligned} &\text{minimize} \quad f(\vec{x}) \\ &\text{subject to} \quad g_j(\vec{x}) \le 0, \quad j = 1, \ldots, q \\ &\qquad\qquad h_j(\vec{x}) = 0, \quad j = q+1, \ldots, m \end{aligned}$$

Manuscript received December 15, 2008; revised July 2, 2009, April 18, 2010, July 12, 2010, September 3, 2010, and November 1, 2010; accepted November 4, 2010. Date of publication January 11, 2012; date of current version January 31, 2012. This work was supported in part by the National Natural Science Foundation of China, under Grants 60805027, 90820302, and 61175064, in part by the Research Fund for the Doctoral Program of Higher Education of China, under Grant 200805330005, and in part by the Graduate Innovation Fund of Hunan Province of China, under Grant CX2009B039. This paper was recommended by Associate Editor C. A. Coello Coello.

The authors are with the School of Information Science and Engineering, Central South University, Changsha 410083, China (e-mail: ywang@csu.edu.cn; zxcai@csu.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TEVC.2010.2093582

where $\vec{x} = (x_1, \ldots, x_n) \in S, S$ is the decision space defined by the parametric constraints:

$$L_i \le x_i \le U_i, \quad 1 \le i \le n \tag{1}$$

$g_j(\vec{x})$ is the jth inequality constraint, and $h_j(\vec{x})$ is the $(j-q)$th equality constraint.

The feasible region $\Omega \subseteq S$ is defined as follows:

$$\begin{aligned} \Omega = \{\vec{x} | g_j(\vec{x}) \le 0,\ j = 1, \ldots, q; \\ h_j(\vec{x}) = 0,\ j = q+1, \ldots, m; \vec{x} \in S\}. \end{aligned} \tag{2}$$

If an inequality constraint satisfies $g_j(\vec{x}) = 0$ ($j \in \{1, \ldots, q\}$) at any point $\vec{x} \in \Omega$, we say it is *active* at $\vec{x}$. All equality constraints $h_j(\vec{x})$ ($j = q+1, \ldots, m$) are considered *active* at all points of Ω.

The use of evolutionary algorithms (EAs) for COPs has significantly grown in the past decade, giving rise to a large number of constrained optimization evolutionary algorithms (COEAs) [2], [3]. It is necessary to note that EAs are unconstrained optimization methods that need additional mechanisms to deal with constraints when solving COPs [4]. As a result, a variety of constraint-handling techniques targeted at EAs have been developed [5].

Penalty function methods are the most common constraint-handling technique. They use the amount of constraint violation to punish an infeasible solution so that it is less likely to survive into the next generation than a feasible solution. The main limitation of penalty function methods is that they require fine tuning of the penalty factors. In order to address this limitation, methods based on the preference of feasible solutions over infeasible solutions have been proposed. For example, Deb [6] proposed a feasibility-based rule to pairwise compare individuals:

1) any feasible solution is preferred to any infeasible solution;
2) among two feasible solutions, the one that has a better objective function value is preferred;
3) among two infeasible solutions, the one that has a smaller degree of constraint violation is preferred.

In addition, some researchers have employed multiobjective optimization techniques to handle constraints. The main idea of this kind of method is that after converting COPs into unconstrained multiobjective optimization problems,

multiobjective optimization techniques are exploited to tackle the converted problems [7].

Although some researchers have suggested that multiobjective optimization techniques are not suitable for solving COPs [8], [9], this kind of technique has still attracted considerable interest in the community of constrained evolutionary optimization in recent years and many approaches have been proposed [1], [5], [7], [10]–[16]. Among these approaches, CW [1] is a very recent one. It consists of two main components: the first is the population evolution model, and the second is the infeasible solution archiving and replacement mechanism. However, as pointed out in [1] and [15], the main drawback of this approach is that values must be determined for some problem-dependent parameters, such as the expanding factor in simplex crossover [17], which limits its real-world applications.

The main motivation of this paper is to overcome the above drawback of CW and, as a result, a new method, called CMODE, is proposed. Apart from using differential evolution (DE) as the search engine, CMODE also proposes a novel infeasible solution replacement mechanism based on multiobjective optimization. Twenty-four benchmark test functions collected for the special session on constrained real-parameter optimization of the 2006 IEEE congress on evolutionary computation (IEEE CEC2006) [18] are used to demonstrate the effectiveness of CMODE. Experimental results suggest that the performance of CMODE is very competitive with that of several state-of-the-art methods in the community of constrained evolutionary optimization.

The remainder of this paper is organized as follows. Section II introduces DE and briefly reviews the previous work on constrained optimization using DE. Section III describes our proposal in detail. The experimental results and the performance comparisons are given in Section IV. Section V discusses the effectiveness of some mechanisms proposed in this paper and the effect of parameter settings on the performance of CMODE. Finally, Section VI concludes this paper and provides some possible paths for future research.

II. DE and Its Application in Constrained Evolutionary Optimization

DE, proposed by Storn and Price [19] in 1995, is an efficient and simple EA. The initial population of DE is randomly generated within the decision space. The population of DE consists of N_p n-dimensional vectors:

$$\vec{x}_{i,t} = (x_{i,1,t}, x_{i,2,t}, \ldots, x_{i,n,t}), \quad i = 1, 2, \ldots, N_p \tag{3}$$

where t denotes the generation number. The idea behind DE is to take advantage of mutation and crossover operations to yield a trial vector $\vec{u}_{i,t}$ for each target vector $\vec{x}_{i,t}$. Thereafter, a selection operation is executed between the trial vector $\vec{u}_{i,t}$ and the target vector $\vec{x}_{i,t}$.

Several variants of DE have been proposed. In this paper, the most often used DE algorithm (i.e., *DE/rand/1/bin*) is utilized. The mutation, crossover, and selection operations of this DE algorithm are explained as follows.

A. Mutation Operation

Taking into account each target vector $\vec{x}_{i,t}$ at generation t, a mutant vector $\vec{v}_{i,t} = (v_{i,1,t}, v_{i,2,t}, \ldots, v_{i,n,t})$ is defined by

$$\vec{v}_{i,t} = \vec{x}_{r_1,t} + F \cdot (\vec{x}_{r_2,t} - \vec{x}_{r_3,t}) \tag{4}$$

where indexes r_1, r_2, and r_3 represent mutually different integers that are different from i and that are randomly generated over $[1, N_p]$, and F is the scaling factor.

In this paper, if a component $v_{i,j,t}$ of a mutant vector $\vec{v}_{i,t}$ violates the boundary constraint, this component is reset as follows:

$$v_{i,j,t} = \begin{cases} \min\{U_j, 2L_j - v_{i,j,t}\}, & \text{if} \quad v_{i,j,t} < L_j \\ \max\{L_j, 2U_j - v_{i,j,t}\}, & \text{if} \quad v_{i,j,t} > U_j. \end{cases} \tag{5}$$

B. Crossover Operation

The target vector $\vec{x}_{i,t}$ is mixed with the mutant vector $\vec{v}_{i,t}$, using a binomial crossover operation (also known as uniform discrete crossover), to form the trial vector:

$$u_{i,j,t} = \begin{cases} v_{i,j,t}, & \text{if } rand_j(0,1) \le C_r \quad \text{or} \quad j = j_{rand} \\ x_{i,j,t}, & \text{otherwise} \end{cases} \tag{6}$$

where $i = 1, 2, \ldots, N_p, j = 1, 2, \ldots, n$, index j_{rand} is a randomly chosen integer within the range $[1, n]$, $rand_j(0, 1)$ is the jth evaluation of a uniform random number generator, and $C_r \in [0, 1]$ is the crossover control parameter. The condition "$j = j_{rand}$" is introduced to ensure that the trial vector $\vec{u}_{i,t}$ differs from its target vector $\vec{x}_{i,t}$ by at least one element.

C. Selection Operation

After evaluating the target vector $\vec{x}_{i,t}$ and the trial vector $\vec{u}_{i,t}$, the trial vector $\vec{u}_{i,t}$ is compared against the target vector $\vec{x}_{i,t}$ and the better one is preserved for the next generation:

$$\vec{x}_{i,t+1} = \begin{cases} \vec{u}_{i,t}, & \text{if } f(\vec{u}_{i,t}) \le f(\vec{x}_{i,t}) \\ \vec{x}_{i,t}, & \text{otherwise.} \end{cases} \tag{7}$$

Many studies have been conducted to solve COPs by DE. Below, we briefly review some of them.

Storn [20] proposed a method called CADE, which combines the idea of constraint adaptation with DE. CADE first relaxes all constraints so that all individuals in the population are feasible and then gradually tightens the constraints. In addition, CADE employs the concept of aging to prevent a vector from surviving for excessive generations. Lin *et al.* [21] introduced a hybrid DE with a multiplier updating method to solve COPs. Lampinen [22] extended DE to handle nonlinear constraint functions. In this method, when the trial vector and the target vector are infeasible, the one which Pareto dominates the other in the constraint space will be selected. In addition, if these two vectors are incomparable with each other, the trial vector is allowed to enter the population to avoid stagnation. Runarsson and Yao [8] proposed an improved version of stochastic ranking [23]. This method contains a differential variation operator, which resembles the mutation operator of DE. Mezura-Montes *et al.* [24] presented an alternative method which can be considered as the first proposal to incorporate a diversity mechanism

into DE. In this method, the diversity mechanism allows infeasible solutions with a good value of the objective function, regardless of the degree of constraint violation, to remain in the population. Furthermore, each parent is able to generate more than one offspring in this method. DE, coupled with a cultural algorithm is proposed by Becerra and Coello Coello [25].

Besides the above methods, several related approaches were proposed at the IEEE CEC2006 special session on constrained real-parameter optimization. Takahama and Sakai [26] proposed εDE which applies an ε-constrained method to DE. In this method, a gradient-based mutation is introduced, which uses the gradient of constraints at an infeasible point to find a feasible point. Huang *et al.* [27] introduced a self-adaptive DE for COPs, in which the choice of the trial vector generation strategies and the two control parameters (F and C_r) are not required to be predefined. During evolution, the suitable strategies and parameter settings are gradually self-adapted according to the learning experience. Tasgetiren and Suganthan [28] presented a multi-populated DE. This method regroups the individuals in certain periods of a run. Kukkonen and Lampinen [29] proposed a generalized DE to solve COPs. In this approach, the trial vector is selected to replace the target vector if it weakly dominates the target vector in the space of constraint violations or objective function. Brest *et al.* [30] also proposed a self-adaptive DE, in which three DE strategies are applied and the control parameters F and C_r of DE are self-adapted. Mezura-Montes *et al.* [31] presented a modified DE for COPs, in which a new mutation operator is designed. The new mutation operator combines information of both the best solution in the current population and the current parent to find new search directions. Zielinski and Laur [32] integrated DE with Deb's feasibility-based rule [6] for constrained optimization.

More recently, Mezura-Montes and Cecilia-López-Ramírez [33] established a performance comparison of four bio-inspired algorithms with the same constraint-handling technique (i.e., Deb's feasibility-based rule) to solve 24 benchmark test functions. These four bio-inspired algorithms are DE, genetic algorithm, evolution strategy, and particle swarm optimization. The overall results indicate that DE is the most competitive among all of the compared algorithms for this set of test functions. In addition, Gong and Cai [34] proposed a multiobjective DE algorithm for constrained optimization. This method uses orthogonal design to generate the initial population, and the comparison of the individuals is based on Pareto dominance. Takahama and Sakai [35] proposed an improved ε-constrained DE to solve COPs with equality constraints. In this approach, dynamic control of allowable constraint violation for equality constraints is introduced, and the amount of allowable violation is specified by the ε-level. Zhang *et al.* [36] proposed a dynamic stochastic selection scheme based on stochastic ranking [23] and combined it with the multimember DE [24]. Zielinski and Laur [37] investigated several stopping criteria for DE in constrained optimization, which consider the improvement, movement or distribution of population members to determine when DE should be terminated.

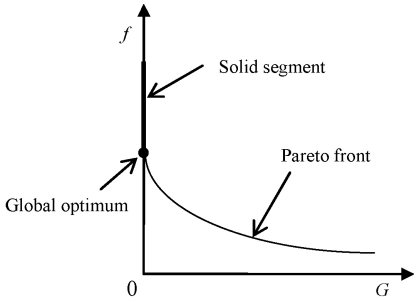

Fig. 1. Graph representation for $\mathbf{f}(\vec{x})$. The Pareto optimal set is mapped to the Pareto front. The feasible region Ω is mapped to the solid segment. The global optimum $\vec{x}^*$ is mapped to the intersection of the Pareto front and the solid segment. The search space S is mapped to points on and above the Pareto front.

III. Proposed Approach

A. CW Approach

In the CW approach [1], the degree of constraint violation of an individual $\vec{x}$ on the jth constraint is defined as

$$G_j(\vec{x}) = \begin{cases} \max\{0, g_j(\vec{x})\}, & 1 \le j \le q \\ \max\{0, |h_j(\vec{x})| - \delta\}, & q+1 \le j \le m \end{cases} \tag{8}$$

where δ is a positive tolerance value for equality constraints. Then $G(\vec{x}) = \sum_{j=1}^{m} G_j(\vec{x})$ reflects the degree of constraint violation of the individual $\vec{x}$.

In principle, the CW approach converts COPs into multiobjective optimization problems (MOPs) in which two objectives are considered: the first is to optimize the original objective function $f(\vec{x})$, and the second is to minimize the degree of constraint violation $G(\vec{x})$. For the sake of clarity, let $\mathbf{f}(\vec{x}) = (\mathbf{f}_1(\vec{x}), \mathbf{f}_2(\vec{x})) = (f(\vec{x}), G(\vec{x}))$.

An important concept of multiobjective optimization is that of Pareto dominance. In the context of this paper, an individual $\vec{x}_i$ is said to Pareto dominate another individual $\vec{x}_j$ (denoted as $\vec{x}_i \prec \vec{x}_j$) if $\forall k \in \{1, 2\}$, $\mathbf{f}_k(\vec{x}_i) \le \mathbf{f}_k(\vec{x}_j)$, and $\exists k \in \{1, 2\}$, such that $\mathbf{f}_k(\vec{x}_i) < \mathbf{f}_k(\vec{x}_j)$. The nondominated individuals of the population refer to those that are not Pareto dominated by any member of the population. The set of nondominated individuals is called the Pareto optimal set. The Pareto front is the image of the Pareto optimal set in the objective space. According to the above definitions, the detailed illustration of $\mathbf{f}(\vec{x})$ is shown in Fig. 1 [1].

The CW approach attempts to optimize $\mathbf{f}(\vec{x})$ through multiobjective optimization techniques. As analyzed in [1], the essential difference between the solution of $\mathbf{f}(\vec{x})$ and the solution of the general MOPs is that the aim of the former is to find the global optimal solution in the feasible region (i.e., the global optimum in Fig. 1); however, the goal of the latter is to obtain a final population with a diversity of nondominated individuals, i.e., the image of the population in the objective space should be distributed as uniformly as possible in the Pareto front (i.e., the Pareto front in Fig. 1). Therefore, the solution of $\mathbf{f}(\vec{x})$ is not equivalent to that of the general MOPs. Moreover, it is unnecessary to uniformly distribute the nondominated individuals found during evolution when solving $\mathbf{f}(\vec{x})$.

Realizing the above difference, the CW approach only exploits Pareto dominance that is often used in multiobjective optimization to compare the individuals in the population. The CW approach includes two main components: the first is the population evolution model inspired by the proposal in [38], and the second is the infeasible solution archiving and replacement mechanism which is proposed in the hope of steering the population toward the feasible region. A detailed description of the CW approach has been presented in [1]. The CW approach has been tested on 13 benchmark test functions collected by Runarsson and Yao [23]. It has been shown to be successful for solving COPs with different constraint types, especially COPs with equality constraints. However, the major limitation of this method is that for a specific problem, one often needs to carry out extensive tuning of the parameters, such as the expanding factor in the simplex crossover [17].

B. CMODE Approach

To overcome the shortcomings of the CW approach, this paper proposes a new implementation of the CW approach, which we call CMODE.

At each generation, CMODE maintains:

1) a population of N_p individuals, i.e., $P_{(t)} = \{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_{N_p}\}$ where t denotes the generation number;
2) their objective function values $f(\vec{x}_1), f(\vec{x}_2), \ldots, f(\vec{x}_{N_p})$ and their degree of constraint violations $G(\vec{x}_1), G(\vec{x}_2), \ldots, G(\vec{x}_{N_p})$.

The framework of CMODE is shown in Fig. 2. Initially, population $P_{(t)}$ is randomly produced in the decision space defined by $[L_i, U_i], 1 \leq i \leq n$. Subsequently, λ individuals (set Q) are randomly chosen from $P_{(t)}$ to yield λ offspring (set C) by DE operations and are deleted from $P_{(t)}$. Thereafter, the nondominated individuals (set R) are identified from C and replace the dominated individuals (if they exist) in Q. As a result, Q is updated. After combining the updated Q with $P_{(t)}$, the update of $P_{(t)}$ is also achieved. Additionally, if R contains only infeasible solutions, meaning C is also entirely composed of infeasible solutions, then the infeasible solution with the lowest degree of constraint violation in R is stored into archive A. Every k generations, all of the infeasible individuals in A are used to replace the same number of individuals in $P_{(t)}$. It is noteworthy that the above replacement is executed based on an infeasible solution replacement mechanism inspired by multiobjective optimization. The above procedure is repeated until the maximum number of function evaluations (FES) is reached.

The detailed features of our algorithm are the following.

1) *DE operations*: In CMODE, DE is considered to be the search engine. However, it is important to note that we only use DE's crossover and mutation operations to create the offspring population. Moreover, the selection operation of DE is not applied. With respect to the classical DE, the trial vector created by crossover and mutation operations is directly compared with its target vector and the better one will survive into the next population. Nevertheless, in this paper only the nondominated individuals of the offspring population are identified and exploited to replace the dominated individuals (if they exist) in the parent population, since the nondominated individuals represent the most important feature of the population to which they belong [1].

2) *Infeasible solution replacement mechanism*: This mechanism consists of two main parts: the deterministic replacement and the random replacement.

The primary motivation of the deterministic replacement is to enhance the quality and feasibility of the individuals in population $P_{(t)}$ simultaneously, by replacing the worst individuals in $P_{(t)}$ with the infeasible solutions in archive A. The worst individuals in $P_{(t)}$ are measured by two performance indicators.

The first performance indicator measures the quality of the individuals. We consider that the greater the number of individuals Pareto dominating a given individual, the worse the quality of this individual. According to this viewpoint, the fitness assignment scheme proposed by Zitzler *et al.* [39] for MOPs is chosen as the first performance indicator since in this paper, COPs are treated as MOPs. As in [39], each individual $\vec{x}_i$ in population $P_{(t)}$ is assigned a strength value $s(\vec{x}_i)$. $s(\vec{x}_i)$ represents the number of individuals in $P_{(t)}$ Pareto dominated by $\vec{x}_i$, that is

$$s(\vec{x}_i) = \#\{\vec{x}_j | \vec{x}_j \in P_{(t)} \wedge \vec{x}_i \prec \vec{x}_j\}, \quad i = 1, \ldots, N_p \tag{9}$$

where # is the cardinality of the set. On the basis of the strength value, the rank value $R_1(\vec{x}_i)$ of the individual $\vec{x}_i$ is calculated using the following equation:

$$R_1(\vec{x}_i) = \sum_{\vec{x}_j \in P_{(t)} \bigcap \vec{x}_j \prec \vec{x}_i} s(\vec{x}_j), \quad i = 1, \ldots, N_p. \tag{10}$$

The second performance indicator measures the feasibility of the individuals. In this performance indicator, the individuals in population $P_{(t)}$ are sorted according to the following criteria:

a) the feasible solutions are listed in front of the infeasible solutions;
b) the feasible solutions are sorted in ascending order by their objective function values;
c) the infeasible solutions are sorted in ascending order by their degree of constraint violations.

After the above process, each individual is assigned another rank value $R_2(\vec{x}_i)$ by subtracting 1 from its sequence number.

A final objective function is established by adding the normalized rank values of $R_1(\vec{x}_i)$ and $R_2(\vec{x}_i)$ together

$$F(\vec{x}_i) = \frac{R_1(\vec{x}_i)}{\max\limits_{j=1,\ldots,N_p} R_1(\vec{x}_j)} + \frac{R_2(\vec{x}_i)}{\max\limits_{j=1,\ldots,N_p} R_2(\vec{x}_j)}, \quad i = 1, \ldots, N_p. \tag{11}$$

In (11), $R_1(\vec{x}_i)$ and $R_2(\vec{x}_i)$ are normalized so that their values are of an order of magnitude of 1.

In the deterministic replacement, all of the individuals in archive A are selected to replace the same number of individuals with the largest $F(\vec{x}_i)$ in population $P_{(t)}$. The first term on the right-hand side of (11) aims to eliminate the worst individuals in the population on the basis of Pareto dominance, and as a result, to motivate the population toward the solutions with higher quality. However, it is necessary to emphasize that this term does not consider the feasibility of the individuals explicitly and cannot effectively guide the

Input: N_p: the size of the population
λ: the number of individuals chosen to take part in DE operations
k: the generation interval for implementing the infeasible solution replacement mechanism
Max_FES: maximum number of function evaluations

Output: $\vec{x}^*$: the best solution in the final population

Step 1 Initialization

Step 1.1 Randomly generate an initial population of size N_p from the decision space S.

Step 1.2 Evaluate the f value and the G value for each individual in the initial population.

Step 1.3 Set $FES = N_p$. /* FES denotes the number of function evaluations */

Step 1.4 Set $A = \phi$. /* A denotes the archive which is used to store the infeasible individual with the lowest degree of constraint violation */

Step 1.5 $t = 1$. /* t denotes the generation number */

Step 2 Population evolution model

Step 2.1 Choose λ individuals (set Q) from population $P_{(t)}$ randomly.

Step 2.2 $P_{(t)} = P_{(t)} - Q$.

Step 2.3 For each individual in Q, an offspring is created by using the mutation and crossover operations of DE as explained in Section II. Thus, λ offspring (set C) are generated from Q.

Step 2.4 Evaluate the f value and the G value for each individual in C.

Step 2.5 $FES = FES + \lambda$.

Step 2.6 Identify the nondominated individuals (set R) from C. Assuming there exist m' nondominated individuals, denoted as $\vec{x}_1, \ldots, \vec{x}_{m'}$.

Step 2.7 Suppose that there are n' individuals in Q dominated by $\vec{x}_1$. If $n' \geq 1$, then one of the dominated individuals chosen at random is replaced by $\vec{x}_1$. Thereafter, the remaining nondominated individuals of R will continue the same process for the updated Q in turn.

Step 2.8 $P_{(t)} = P_{(t)} \cup Q$.

Step 3 Infeasible solution archiving and replacement

Step 3.1 If there do not exist any feasible solutions in R, then $A = A \cup \vec{x}'$, where $\vec{x}'$ is the infeasible individual with the lowest degree of constraint violation in R.

Step 3.2 If $\mathrm{mod}(t, k) = 0$, then execute the infeasible solution replacement mechanism and set $A = \phi$. /* the infeasible solution replacement mechanism will be explained later */

Step 4 Set $t = t + 1$.

Step 5 Stopping Criterion: If $FES \geq Max\_FES$, then stop and output the best solution $\vec{x}^*$ in $P_{(t)}$, otherwise go to step 2.

Fig. 2. Framework of CMODE.

population toward the feasible region since the individuals with a large degree of constraint violations might have small values with respect to this term. Note that the survival of such solutions will influence the speed at which the population enters the feasible region. Hence, we introduce the second term on the right-hand side of (11) to motivate the population approaching or entering the feasible region quickly. By combining the first and second terms, the purpose of the deterministic replacement can be achieved.

With respect to the random replacement, all of the individuals in archive A are used to eliminate the same number of individuals selected from population $P_{(t)}$ at random. Note, however, that the replacement is not applied to the best individual in $P_{(t)}$. It is necessary to emphasize that if the population contains only infeasible solutions, the best individual denotes the infeasible solution with the lowest degree of constraint violation, or else the best individual denotes the feasible solution with the smallest objective function value in the population. The main reason why the best individual is not replaced is explained as follows.

For some particular COPs in which the optimal solutions are located on the boundaries of the feasible region, the following scenario might arise:

a) the region surrounding the optimal solution consists of a very large infeasible region yet a very small feasible region;
b) the population either converges or is very close to the global optimum in the later evolutionary stage.

Under the above conditions, if the evolution of the population does not finish, the offspring population created by the DE operations might always only involve infeasible solutions. Note that if the current offspring population is entirely composed of infeasible solutions, the infeasible solution with the lowest degree of constraint violation will be archived. Subsequently, some individuals of population $P_{(t)}$ will be randomly replaced by the infeasible individuals in archive A. Thus, the worst case is that the feasibility proportion of $P_{(t)}$ might gradually decrease to zero due to the random replacement. The above issue can be addressed by preventing the best individual from being replaced since once the population contains one or more feasible solutions, the best feasible solution will not be replaced by the infeasible solutions in archive A.

Based on our experiments which are explained in Section V-A, the performance of our algorithm will be more competitive if the deterministic replacement has a slightly higher chance of being implemented than the random replacement. The infeasible solution replacement mechanism is executed as follows:

```
if rand < 0.75
    execute the deterministic replacement;
else
    execute the random replacement;
end
```

where *rand* is a uniformly generated random number between 0 and 1.

TABLE I
DETAILS OF 24 BENCHMARK TEST FUNCTIONS

Prob.	n	Type of objective function	ρ	LI	NI	LE	NE	a	$f(\vec{x}^*)$
g01	13	Quadratic	0.0111%	9	0	0	0	6	−15.0000000000
g02	20	Nonlinear	99.9971%	0	2	0	0	1	−0.8036191041
g03	10	Polynomial	0.0000%	0	0	0	1	1	−1.0005001000
g04	5	Quadratic	51.1230%	0	6	0	0	2	−30665.5386717833
g05	4	Cubic	0.0000%	2	0	0	3	3	5126.4967140071
g06	2	Cubic	0.0066%	0	2	0	0	2	−6961.8138755802
g07	10	Quadratic	0.0003%	3	5	0	0	6	24.3062090682
g08	2	Nonlinear	0.8560%	0	2	0	0	0	−0.0958250414
g09	7	Polynomial	0.5121%	0	4	0	0	2	680.6300573744
g10	8	Linear	0.0010%	3	3	0	0	0	7049.2480205287
g11	2	Quadratic	0.0000%	0	0	0	1	1	0.7499000000
g12	3	Quadratic	4.7713%	0	1	0	0	0	−1.0000000000
g13	5	Nonlinear	0.0000%	0	0	0	3	3	0.0539415140
g14	10	Nonlinear	0.0000%	0	0	3	0	3	−47.7648884595
g15	3	Quadratic	0.0000%	0	0	1	1	2	961.7150222900
g16	5	Nonlinear	0.0204%	4	34	0	0	4	−1.9051552585
g17	6	Nonlinear	0.0000%	0	0	0	4	4	8853.5338748065
g18	9	Quadratic	0.0000%	0	13	0	0	0	−0.8660254038
g19	15	Nonlinear	33.4761%	0	5	0	0	0	32.6555929502
g20	24	Linear	0.0000%	0	6	2	12	16	0.2049794002
g21	7	Linear	0.0000%	0	1	0	5	6	193.7245100697
g22	22	Linear	0.0000%	0	1	8	11	19	236.4309755040
g23	9	Linear	0.0000%	0	2	3	1	6	−400.0551000000
g24	2	Linear	79.6556%	0	2	0	0	2	−5.5080132716

C. Computational Time Complexity

The basic operations of CMODE and their worst-case complexities at one generation are as follows.

1) The identification of the nondominated individuals in set C requires $2\lambda\bar{\lambda}$ comparisons of individuals [40], where $\bar{\lambda}$ is the number of the nondominated individuals in set C.
2) Using the nondominated individuals in set C to replace the dominated individuals in set Q needs $2\lambda\bar{\lambda}$ comparisons.
3) The infeasible solution replacement mechanism includes two parts. In the deterministic replacement, computing the rank value $R_1(\vec{x}_i)$ and the rank value $R_2(\vec{x}_i)$ requires $2N_p(N_p - 1)$ comparisons and $N_p\log(N_p)$ comparisons in population $P_{(t)}$, respectively. In addition, sorting the individuals in population $P_{(t)}$ according to (11) requires $N_p\log(N_p)$ comparisons. Since the replacement is implemented every k iterations, the worst complexity of computing the rank value $R_1(\vec{x}_i)$ and the rank value $R_2(\vec{x}_i)$ and sorting the individuals according to (11) at one generation is $2N_p(N_p-1)/k$, $N_p\log(N_p)/k$, and $N_p\log(N_p)/k$, respectively. The computational time complexity can be neglected in the random replacement.

So, the overall computational time complexity of CMODE is $4\lambda\bar{\lambda} + 2N_p(N_p - 1 + \log(N_p))/k$.

D. Similarities and Differences Between CW and CMODE

We would like to make the following remarks on the similarities and differences between CMODE and CW.

1) Both CW and CMODE involve two main components: the population evolution model and the infeasible solution archiving and replacement mechanism.
2) CW includes a problem-dependent parameter, i.e., the expanding factor in the simplex crossover, which should be tuned according to the number of the decision variables of the problems at hand, in order to produce robust performance. However, in CMODE the parameter settings are kept the same for different problems by taking advantage of DE as the search engine.
3) In CW, only one nondominated individual in set C is used to replace one dominated individual (if it exists) in set Q. However, CMODE allows all of the nondominated individuals in set C to replace the corresponding dominated individuals (if they exist) in set Q. Moreover, the replacement procedure of CMODE is simpler than that of CW.
4) The infeasible solution replacement mechanism in CW is similar to the random replacement in CMODE. The only difference is that the best individual in the population is never replaced in the random replacement of CMODE. In CMODE, the deterministic replacement is introduced and combined with the random replacement, which makes the algorithm more effective when solving complex COPs.
5) Instead of only randomly choosing several infeasible individuals from archive A to replace the same number of individuals in population $P_{(t)}$ as in CW, all of the infeasible individuals in archive A replace the same number of individuals in population $P_{(t)}$ in

CMODE, which eliminates an additional parameter for CMODE.

IV. Experimental Study

A. Experimental Settings

The 24 benchmark test functions collected in [18] were employed to demonstrate the capability of CMODE. The details of these test cases are reported in Table I, where n is the number of decision variables, $\rho = |\Omega|/|S|$ is the estimated ratio between the feasible region and the search space, LI is the number of linear inequality constraints, NI is the number of nonlinear inequality constraints, LE is the number of linear equality constraints, NE is the number of nonlinear equality constraints, a is the number of constraints active at the optimal solution, and $f(\vec{x}^*)$ is the objective function value of the best known solution. It is necessary to emphasize that we only show 4 digits after the decimal point in the fourth column of Table I, so ρ is equal to 0.0000% for some test functions. However, it does not mean that there are no feasible solutions in the search space.

Since an improved best known solution has been found in this paper for test function g17, the $f(\vec{x}^*)$ in Table I is different from that in [18] for this test function. The best known solution reported in [18] for test function g17 is $\vec{x}^* = (201.784467214524, 100, 383.071034852773, 420, -10.907658451429, 0.073148231208)$ with $f(\vec{x}^*) = 8853.53967480648$. The improved solution found in this paper for test function g17 is $\vec{x}^* = (201.784462493550, 100, 383.071034852773, 420, -10.907665625756, 0.073148231208)$ with $f(\vec{x}^*) = 8853.533874806484$. The above result for test function g17 has also been reported in [29].

CMODE includes the following five parameters: the population size (N_p), the scaling factor (F) and the crossover control parameter (C_r) of DE, the size of set Q (λ), and the interval of generations for infeasible solution archiving and replacement (k). Usually, F is chosen from the interval [0, 1], and the best reported values are typically between 0.5 and 0.9 [41], [42]. C_r is also chosen between 0 and 1; however, high values, such as 0.9 and 1.0, lead to good results for most applications. It is necessary to note that this paper adopts a steady-state EA,[1] thus the size of $P_{(t)}$ (i.e., N_p) is recommended to be much larger than that of set Q (i.e., λ). In addition, we intend to let each individual in population $P_{(t)}$ carry out the DE operations about once before being replaced. So, N_p should be approximately equal to λk. In this paper, the actual parameter values are set as follows: $N_p = 180$, F is randomly chosen between 0.5 and 0.6, C_r is randomly chosen between 0.9 and 0.95, $\lambda = 8$, and $k = 22$.

B. General Performance of the Proposed Algorithm

As suggested by Liang *et al.* [18], 25 independent runs were performed for each test case using 5×10^5 FES at maximum, and the tolerance value δ for the equality constraints was set to 0.0001. Note that we present our experimental results in the way also suggested by Liang *et al.* [18]. The best, median, worst, mean, and standard deviation of the error value ($f(\vec{x}) - f(\vec{x}^*)$) for the best-so-far solution $\vec{x}$ after 5×10^3, 5×10^4, and 5×10^5 FES in each run are recorded in Tables II–V. In these tables, c is the number of violated constraints at the median solution: the sequence of three numbers indicates the number of violations (including inequality and equality constraints) by more than 1.0, between 0.01 and 1.0, and between 0.0001 and 0.01, respectively. $\bar{v}$ is the mean value of the violations of all the constraints at the median solution. The numbers in the parentheses after the error values of the best, median, and worst solutions are the number of unsatisfied constraints at the best, median, and worst solutions, respectively.

As shown in Tables II–V, for 12 out of 24 test functions (i.e., g01, g02, g04, g06, g07, g08, g09, g11, g12, g16, g19, and g24) feasible solutions can be found in every run by using 5×10^3 FES. In 5×10^4 FES, feasible solutions can be consistently found for all the test functions with the exception of test functions g20, g22, and g23. For test function g23, CMODE enters the feasible region within 5×10^5 FES. It is worth noting that with regard to test function g20, the best known solution is a little infeasible, and no feasible solution has been found so far. Despite the fact that several constraints are violated in the best, median, and worst solutions for this test function by using 5×10^5 FES, only the first constraint is violated for slightly more than 0.01 based on our observation of CMODE. In terms of test function g22, the results derived from CMODE are still far away from the feasible region, which means this test function is very difficult for CMODE to solve. It can also be seen from Tables II–V that CMODE is able to find a good feasible approximation of the "known" optimal solutions for 9 test functions (i.e., test functions g05, g06, g08, g11, g12, g13, g15, g16, and g24) in 5×10^4 FES. The results achieved by CMODE are very close to or even equal to the "known" optimal solutions for 22 test functions in all runs by using 5×10^5 FES, except for test functions g21 and g22. The results for test function g21 can reach the "known" optimal solution for a majority of runs.

Table VI records the number of FES needed in each run for satisfying the success condition as suggested by Liang *et al.* [18]: $f(\vec{x}) - f(\vec{x}^*) \leq 0.0001$ and $\vec{x}$ is feasible. For test function g20, when the population of CMODE is very close to the feasible region in the later stage, the objective function values of the individuals in the population are smaller than that of the best known solution, thus the success condition is changed to $|f(\vec{x}) - f(\vec{x}^*)| \leq 0.0001$. Table VI also records the feasible rate, the success rate, and the success performance for 24 test functions. The feasible rate denotes the percentage of runs where at least one feasible solution is found in 5×10^5 FES. The success rate denotes the percentage of runs where the algorithm finds a solution that satisfies the success condition. The success performance denotes the mean number of FES for successful runs multiplied by the number of total runs and divided by the number of successful runs.

As shown in Table VI, the feasible rate of 100% has been accomplished for all of the test cases except for test functions

[1] Unlike the generational EA (i.e., standard EA), in which all the individuals in the population are used to generate the offspring population, in the steady-state EA only several individuals in the population are chosen to produce the offspring population, in order to make the evolution of the population more steady.

TABLE II
FUNCTION ERROR VALUES ACHIEVED WHEN FES = 5×10^3, FES = 5×10^4, AND FES = 5×10^5 FOR TEST FUNCTIONS G01-G06

FES	Prob.	g01	g02	g03	g04	g05	g06
5×10^3	Best	6.2987E+00 (0)	4.1673E-01 (0)	8.6595E-01 (0)	8.3040E+01 (0)	3.5276E+02 (3)	9.3309E+00 (0)
	Median	8.4124E+00 (0)	4.9163E-01 (0)	8.5741E-01 (1)	1.4440E+02 (0)	2.1365E+01 (3)	4.2506E+01 (0)
	Worst	1.0242E+01 (0)	5.2903E-01 (0)	7.6268E-01 (1)	2.3471E+02 (0)	1.3974E+02 (4)	1.4798E+02 (0)
	c	0, 0, 0	0, 0, 0	0, 0, 1	0, 0, 0	2, 1, 0	0, 0, 0
	$\bar{v}$	0	0	1.2800E-04	0	8.6147E-01	0
	Mean	8.5197E+00	4.8832E-01	9.1635E-01	1.4686E+02	9.4818E+01	4.3961E+01
	Std	9.3137E-01	3.1699E-02	1.4038E-01	3.8310E+01	1.4295E+02	2.8949E+01
5×10^4	Best	3.3935E-02 (0)	1.6609E-01 (0)	1.4327E-03 (0)	1.4489E-03 (0)	1.7750E-08 (0)	2.1464E-08 (0)
	Median	7.0387E-02 (0)	2.0310E-01 (0)	7.1021E-03 (0)	1.3733E-02 (0)	1.4309E-07 (0)	2.4520E-07 (0)
	Worst	1.6472E-01 (0)	2.6952E-01 (0)	1.9600E-02 (0)	4.4097E-02 (0)	5.2276E-07 (0)	1.4298E-06 (0)
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0
	$\bar{v}$	0	0	0	0	0	0
	Mean	7.8005E-02	2.0225E-01	7.2294E-03	1.7592E-02	1.6482E-07	4.4921E-07
	Std	3.2951E-02	2.7097E-02	4.2851E-03	1.1460E-02	1.1939E-07	4.2333E-07
5×10^5	Best	0.0000E+00 (0)	4.1726E-09 (0)	2.3964E-10 (0)	7.6398E-11 (0)	−1.8190E-12 (0)	3.3651E-11 (0)
	Median	0.0000E+00 (0)	1.1372E-08 (0)	1.1073E-09 (0)	7.6398E-11 (0)	−1.8190E-12 (0)	3.3651E-11 (0)
	Worst	0.0000E+00 (0)	1.1836E-07 (0)	2.5794E-09 (0)	7.6398E-11 (0)	−1.8190E-12 (0)	3.3651E-11 (0)
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0
	$\bar{v}$	0	0	0	0	0	0
	Mean	0.0000E+00	2.0387E-08	1.1665E-09	7.6398E-11	−1.8190E-12	3.3651E-11
	Std	0.0000E+00	2.4195E-08	5.2903E-10	2.6382E-26	1.2366E-27	1.3191E-26

TABLE III
FUNCTION ERROR VALUES ACHIEVED WHEN FES = 5×10^3, FES = 5×10^4, AND FES = 5×10^5 FOR TEST FUNCTIONS G07-G12

FES	Prob.	g07	g08	g09	g10	g11	g12
5×10^3	Best	2.4705E+01 (0)	7.3110E-07 (0)	1.7204E+01 (0)	3.6769E+03 (0)	4.2572E-05 (0)	3.4327E-05 (0)
	Median	5.3070E+01 (0)	2.3934E-04 (0)	4.9003E+01 (0)	7.5273E+03 (0)	5.7988E-04 (0)	1.3876E-04 (0)
	Worst	9.3853E+01 (0)	1.1690E-03 (0)	7.5658E+01 (0)	7.1095E+03 (1)	1.6314E-02 (0)	4.1130E-04 (0)
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0
	$\bar{v}$	0	0	0	0	0	0
	Mean	5.8455E+01	3.1711E-04	4.8373E+01	7.2982E+03	2.3246E-03	1.4508E-04
	Std	1.8767E+01	3.2230E-04	1.5605E+01	2.1056E+03	4.3639E-03	9.0524E-05
5×10^4	Best	1.3917E-01 (0)	8.1968E-11 (0)	7.7915E-04 (0)	1.7843E+01 (0)	1.1792E-10 (0)	0.0000E+00 (0)
	Median	2.4984E-01 (0)	1.1650E-08 (0)	1.7339E-03 (0)	2.9539E+01 (0)	1.6769E-09 (0)	0.0000E+00 (0)
	Worst	3.4206E-01 (0)	2.8863E-07 (0)	7.3459E-03 (0)	5.1087E+01 (0)	9.0309E-09 (0)	0.0000E+00 (0)
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0
	$\bar{v}$	0	0	0	0	0	0
	Mean	2.3893E-01	3.4410E-08	2.4239E-03	3.0954E+01	1.7298E-09	0.0000E+00
	Std	5.3937E-02	6.3509E-08	1.5793E-03	7.1813E+00	1.7427E-09	0.0000E+00
5×10^5	Best	7.9783E-11 (0)	8.1964E-11 (0)	−9.8225E-11 (0)	6.2755E-11 (0)	0.0000E+00 (0)	0.0000E+00 (0)
	Median	7.9793E-11 (0)	8.1964E-11 (0)	−9.8225E-11 (0)	6.2755E-11 (0)	0.0000E+00 (0)	0.0000E+00 (0)
	Worst	7.9811E-11 (0)	8.1964E-11 (0)	−9.8111E-11 (0)	6.3664E-11 (0)	0.0000E+00 (0)	0.0000E+00 (0)
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0
	$\bar{v}$	0	0	0	0	0	0
	Mean	7.9793E-11	8.1964E-11	−9.8198E-11	6.2827E-11	0.0000E+00	0.0000E+00
	Std	7.6527E-15	6.3596E-18	4.9554E-14	2.5182E-13	0.0000E+00	0.0000E+00

TABLE IV
FUNCTION ERROR VALUES ACHIEVED WHEN FES = 5×10^3, FES = 5×10^4, AND FES = 5×10^5 FOR TEST FUNCTIONS G13-G18

FES	Prob.	g13	g14	g15	g16	g17	g18
5×10^3	Best	9.4524E-01 (3)	−3.4672E+01 (3)	1.6702E-01 (2)	3.3010E-02 (0)	1.1711E+01 (4)	9.4076E-01 (2)
	Median	2.8416E-01 (3)	−7.1386E+01 (3)	1.8488E-01 (2)	8.9566E-02 (0)	1.9234E+02 (4)	1.0787E+00 (5)
	Worst	9.6130E-02 (3)	−1.0251E+02 (3)	−9.2489E-01 (2)	1.6484E-01 (0)	−1.1476E+02 (4)	7.1992E-02 (10)
	c	0, 3, 0	3, 0, 0	0, 1, 1	0, 0, 0	2, 2, 0	1, 4, 0
	$\bar{v}$	1.9430E-01	2.2977E+00	2.0680E-02	0	1.8689E+00	2.5338E-01
	Mean	1.2345E+00	−7.2797E+01	4.6228E-01	9.8091E-02	7.6342E+01	7.0585E-01
	Std	2.6306E+00	2.3303E+01	1.0900E+00	3.4533E-02	1.4688E+02	3.2805E-01
5×10^4	Best	8.3118E-09 (0)	4.6637E-02 (0)	1.6336E-10 (0)	2.9208E-07 (0)	4.7053E-03 (0)	3.9508E-03 (0)
	Median	3.8234E-08 (0)	1.6697E-01 (0)	3.4606E-10 (0)	8.9034E-07 (0)	3.8689E-01 (0)	8.3102E-03 (0)
	Worst	1.8106E-07 (0)	3.1410E-01 (0)	1.2237E-09 (0)	2.4040E-06 (0)	3.6328E+00 (0)	1.3964E-02 (0)
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0
	$\bar{v}$	0	0	0	0	0	0
	Mean	6.0691E-08	1.5457E-01	4.5651E-10	9.2580E-07	9.5947E-01	7.9920E-03
	Std	5.5890E-08	6.9440E-02	3.0279E-10	4.4351E-07	1.2452E+00	2.5298E-03
5×10^5	Best	4.1897E-11 (0)	8.5123E-12 (0)	6.0822E-11 (0)	6.5213E-11 (0)	1.8189E-12 (0)	1.5561E-11 (0)
	Median	4.1897E-11 (0)	8.5194E-12 (0)	6.0822E-11 (0)	6.5213E-11 (0)	1.8189E-12 (0)	1.5561E-11 (0)
	Worst	4.1897E-11 (0)	8.5194E-12 (0)	6.0822E-11 (0)	6.5213E-11 (0)	1.8189E-12 (0)	1.5561E-11 (0)
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0
	$\bar{v}$	0	0	0	0	0	0
	Mean	4.1897E-11	8.5159E-12	6.0822E-11	6.5213E-11	1.8189E-12	1.5561E-11
	Std	1.0385E-17	3.6230E-15	0.0000E+00	2.6382E-26	1.2366E-27	6.5053E-17

TABLE V
FUNCTION ERROR VALUES ACHIEVED WHEN FES = 5×10^3, FES = 5×10^4, AND FES = 5×10^5 FOR TEST FUNCTIONS G19-G24

FES	Prob.	g19	g20	g21	g22	g23	g24
5×10^3	Best	1.1203E+02 (0)	3.1573E+00 (20)	1.2672E+02 (5)	2.0789E+03 (19)	1.9244E+02 (5)	6.8253E-04 (0)
	Median	2.6721E+02 (0)	4.4912E+00 (20)	3.2566E+02 (5)	1.4253E+03 (19)	−9.7940E+02 (5)	9.1187E-03 (0)
	Worst	4.5526E+02 (0)	4.7287E+00 (20)	1.3427E+02 (5)	7.2212E+03 (19)	−6.6565E+02 (5)	2.0908E-02 (0)
	c	0, 0, 0	2, 17, 1	1, 4, 0	20, 0, 0	2, 3, 0	0, 0, 0
	$\bar{v}$	0	2.7709E+00	4.7008E-01	3.4231E+06	5.8659E-01	0
	Mean	2.7639E+02	5.1024E+00	1.7669E+02	2.1077E+03	−4.0060E+02	9.4422E-03
	Std	7.4543E+01	9.4581E-01	1.8408E+02	2.2120E+03	4.9291E+02	5.9393E-03
5×10^4	Best	3.4097E+00 (0)	−5.1880E-02 (20)	2.5883E-02 (0)	−2.3148E+02 (20)	3.7442E+01 (0)	1.4111E-09 (0)
	Median	4.5266E+00 (0)	−4.4471E-02 (20)	1.0766E-01 (0)	−2.3581E+02 (20)	1.1020E+02 (0)	2.0215E-08 (0)
	Worst	5.9388E+00 (0)	−8.6126E-02 (20)	1.3195E+02 (0)	−2.3516E+02 (20)	1.5480E+02 (3)	5.0833E-07 (0)
	c	0, 0, 0	0, 8, 12	0, 0, 0	20, 0, 0	0, 0, 0	0, 0, 0
	$\bar{v}$	0	2.6343E-02	0	2.9974E+04	0	0
	Mean	4.5782E+00	−6.5402E-02	3.5214E+01	−2.3462E+02	1.1587E+02	6.9295E-08
	Std	6.8379E-01	1.0365E-02	5.7623E+01	1.9303E+00	6.8442E+01	1.1091E-07
5×10^5	Best	1.1027E-10 (0)	−1.1525E-05 (6)	−3.1237E-10 (0)	−2.3643E+02 (20)	1.8758E-12 (0)	4.6735E-12 (0)
	Median	2.1582E-10 (0)	−1.6567E-05 (8)	−2.9436E-10 (0)	−2.3643E+02 (20)	1.5859E-11 (0)	4.6735E-12 (0)
	Worst	5.4446E-10 (0)	−1.9427E-05 (9)	1.3097E+02 (0)	−2.3643E+02 (20)	2.8063E-10 (0)	4.6735E-12 (0)
	c	0, 0, 0	0, 1, 7	0, 0, 0	8, 10, 0	0, 0, 0	0, 0, 0
	$\bar{v}$	0	7.1975E-03	0	5.6257E+01	0	0
	Mean	2.4644E-10	−2.1729E-05	2.6195E+01	−2.3643E+02	4.4772E-11	4.6735E-12
	Std	1.0723E-10	9.2521E-06	5.3471E+01	8.7023E-14	7.3264E-11	8.2445E-28

TABLE VI
NUMBER OF FES TO ACHIEVE THE SUCCESS CONDITION, SUCCESS RATE, FEASIBLE RATE, AND SUCCESS PERFORMANCE

Prob.	Best	Median	Worst	Mean	Std	Feasible Rate	Success Rate	Success Performance
g01	101 908	122 324	136 228	121 077	8355.8	100%	100%	121 077
g02	170 372	189 204	222 468	189 820	1269.2	100%	100%	189 820
g03	63 364	75 860	86 772	75 085	6271.1	100%	100%	75 085
g04	63 540	73 572	79 556	72 748	3869.7	100%	100%	72 748
g05	26 580	28 692	31 508	28 873	1256.7	100%	100%	28 873
g06	26 932	35 908	41 716	35 464	3200.5	100%	100%	35 464
g07	142 388	156 644	166 148	155 968	4865.1	100%	100%	155 968
g08	2820	5988	8276	5885	1383.7	100%	100%	5885
g09	63 540	70 404	83 780	71 122	6044.7	100%	100%	71 122
g10	171 252	183 924	192 900	183 255	5757.7	100%	100%	183 255
g11	3532	6164	8100	6023	1061.3	100%	100%	6023
g12	1764	5460	8100	5009	1735.1	100%	100%	5009
g13	19 484	30 980	42 316	30 689	4247.1	100%	100%	30 689
g14	97 684	106 660	118 452	107 976	5515.3	100%	100%	107 976
g15	10732	12 868	14 788	12 855	851.2	100%	100%	12 855
g16	27 460	29 396	32 388	29 332	1114.1	100%	100%	29 332
g17	75 460	134 644	294 452	139 746	5949.8	100%	100%	139 746
g18	93 812	104 196	116 340	105 020	7236.0	100%	100%	105 020
g19	241 476	251 684	269 284	251 676	7047.5	100%	100%	251 676
g20	20 076	457 692	487 524	440 121	9064.1	0%	100%	440 121
g21	85 012	95 332	224 756	103 006	27844.2	100%	80%	128 758
g22	–	–	–	–	–	0%	0%	–
g23	208 036	240 772	326 484	244 612	26323.2	100%	100%	244 612
g24	13 908	23 060	31 684	21 820	5171.2	100%	100%	21 820

g20 and g22. For these two test functions, no feasible solutions have been found. Regarding the success rate, CMODE is capable of achieving a value of 100% for all of the test functions with the exception of test functions g21 and g22. There is no successful run for test function g22; however, the successful runs arise in a majority of trials for test function g21. By making use of the indicator "success performance," one can conclude that CMODE requires less than 5×10^4 FES for 9 test functions, less than 2.6×10^5 FES for 22 test functions, and less than 5×10^5 FES for 23 test functions, to achieve the target error accuracy level.

The convergence graphs of $\log(f(\vec{x}) - f(\vec{x}^*))$ over FES at the median run are plotted in Figs. 3–8. Since test function g22 cannot be solved, its convergence graph is not included in Figs. 3–8. The value of $(f(\vec{x}) - f(\vec{x}^*))$ in the later stage is negative for test function g20; consequently, there is no convergence graph of this test case in Figs. 3–8. It is important to note that in Figs. 3–8, the points which satisfy $f(\vec{x}) - f(\vec{x}^*) \leq 0$ are not plotted. Figs. 3–8 clearly indicate that CMODE requires less than 2×10^5 FES for most of test functions to accomplish the target error accuracy level, which further verifies the results shown in Table VI.

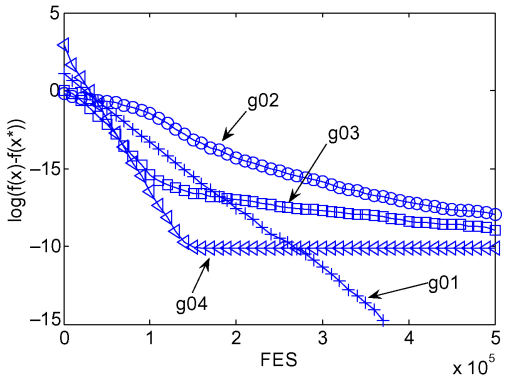

Fig. 3. Convergence graph for g01-g04.

C. Comparison with CW

The main motivation of this paper is to overcome the shortcomings of CW, that is, the problem-dependent parameters, such as the expanding factor in the simplex crossover. In order to validate the motivation for developing CMODE, we compared the performance of CMODE with that of CW on 24 benchmark test functions.

In [1], the authors have recommended the ranges for the population size and the expanding factor in the simplex crossover for different test functions based on the number of decision variables. According to the recommendation in [1], we used the following population size for CW when solving the 24 benchmark test functions:

$$N_p = \begin{cases} 50, & 0 < n < 5 \\ 100, & 5 \leq n < 15 \\ 150, & 15 \leq n < 20 \\ 200, & 20 \leq n < 25. \end{cases} \tag{12}$$

In addition, according to the suggestion in [1], the expanding factor of the simplex crossover in this paper ranged from 3 to 6 if $2 \leq n < 10$, ranged from 8 to 11 if $10 \leq n < 20$, and ranged from 13 to 16 if $20 \leq n < 25$. Moreover, we randomly chose an integer from the range specified for the expanding factor at each generation in order to ease the setting of this parameter and make the comparison fair. The remaining parameters of CW were kept the same as in [1].

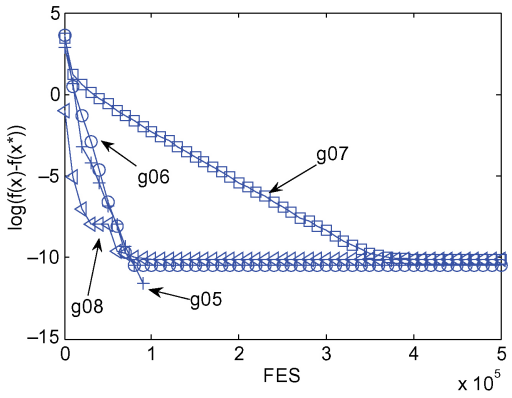

Fig. 4. Convergence graph for g05-g08.

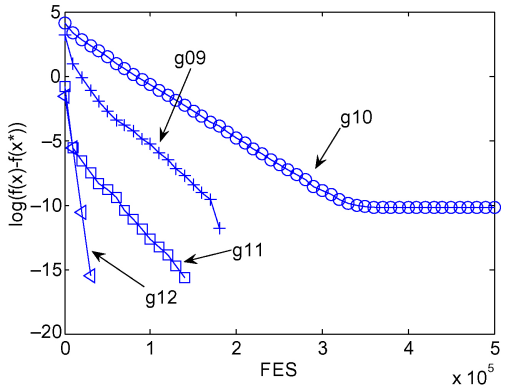

Fig. 5. Convergence graph for g09-g12.

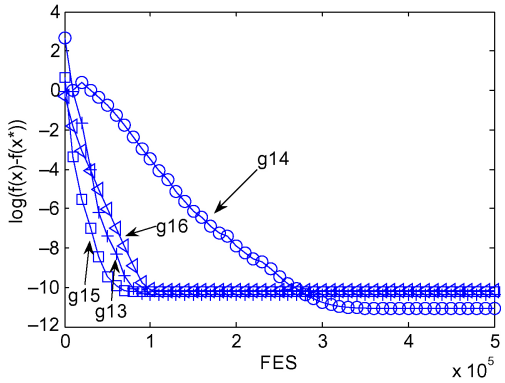

Fig. 6. Convergence graph for g13-g16.

For CMODE and CW, 25 dependent runs were executed on each test function and the maximum number of FES was 5×10^5. In this subsection, only the results of those test functions for which the performance difference between CW and CMODE is obvious are reported due to space limitations. Table VII summarizes the experimental results for test functions g02, g09, g10, g13, g17, g18, g20, g21, and g23, in terms of three indicators: mean objective function value, success rate, and feasible rate.

As shown in Table VII, CW suffers from frequent premature convergence for test functions g02, g09, g10, g13, and g17, although CW can consistently enter the feasible region for them. For these five test functions, the success rates provided by CW are clearly less than those of CMODE. Moreover, CW

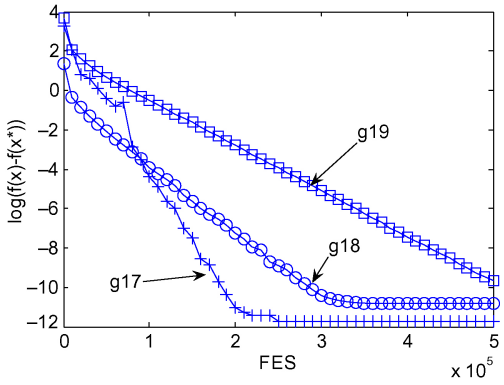

Fig. 7. Convergence graph for g17-g19.

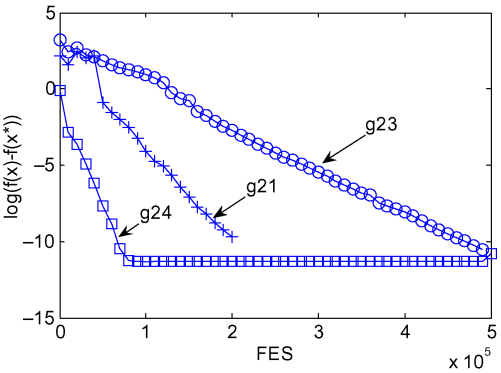

Fig. 8. Convergence graph for g21, g23, and g24.

fails to solve test functions g02, g10, and g17 in any run. In addition, CW cannot achieve the target error accuracy level for test function g20. For test functions g18, g21, and g23, CW is incapable of consistently finding the feasible solutions. More importantly, CW cannot find the feasible solutions even in one run for test functions g21 and g23. Compared to CW, CMODE achieves remarkably better performance in terms of the three indicators.

The above comparison verifies the superiority of CMODE and indicates that the performance of CW is sensitive to the expanding factor in the simplex crossover.

D. Comparison with Some Other DE-Based Approaches in Constrained Evolutionary Optimization

We compared CMODE against six DE-based approaches: εDE [26], a variant of SaDE [27], MPDE [28], GDE [29], jDE-2 [30], and MDE [31], using three performance metrics: feasible rate, success rate, and success performance. It is necessary to note that DE also serves as the search engine in these six approaches. The experimental results of these six approaches are directly taken from the references and are compared with those of CMODE in Tables VIII and IX.

In terms of the mean feasible rate, the performance of CMODE is worse than three methods, i.e., εDE, SaDE, and jDE-2, similar to one method, i.e., MDE, and superior to two methods, i.e., MPDE and GDE. εDE and SaDE achieve the best mean feasible rate among the seven methods. However, both εDE and SaDE require gradient information.

With respect to the mean success rate, CMODE performs significantly better than the other six methods. It is noteworthy

TABLE VII
COMPARISON OF CMODE WITH RESPECT TO CW ON TEST FUNCTIONS G02, G09, G10, G13, G17, G18, G20, G21, AND G23

Prob.	Mean Objective Function Value (Success Rate) [Feasible Rate]		Prob.	Mean Objective Function Value (Success Rate) [Feasible Rate]	
	CMODE	CW		CMODE	CW
g02	**−0.8036 (100%) [100%]**	−0.7655 (0%) [100%]	g18	**−0.8660 (100%) [100%]**	−0.5557 (0%) [68%]
g09	**680.6301 (100%) [100%]**	680.6374 (24%) [100%]	g20	**0.2050 (100%) [0%]**	0.1848 (0%) [0%]
g10	**7049.2480 (100%) [100%]**	9532.9256 (0%) [100%]	g21	**219.9201 (80%) [100%]**	419.0050 (0%) [0%]
g13	**0.0539 (100%) [100%]**	0.0709 (76%) [100%]	g23	**−400.0551 (100%) [100%]**	−342.3291 (0%) [0%]
g17	**8853.5339 (100%) [100%]**	8934.4539 (0%) [100%]			

The better result for each test function between the two compared algorithms is highlighted in boldface.

TABLE VIII
COMPARISON OF CMODE WITH RESPECT TO εDE [26], SADE [27], MPDE [28], GDE [29], JDE-2 [30], AND MDE [31] IN TERMS OF FEASIBLE RATE AND SUCCESS RATE

Prob.	Feasible Rate							Success Rate						
	εDE	SaDE	MPDE	GDE	jDE-2	MDE	CMODE	εDE	SaDE	MPDE	GDE	jDE-2	MDE	CMODE
g02	100%	100%	100%	100%	100%	100%	100%	**100%**	84%	92%	72%	92%	16%	**100%**
g03	**100%**	**100%**	**100%**	96%	**100%**	**100%**	**100%**	**100%**	96%	84%	4%	0%	**100%**	**100%**
g05	**100%**	**100%**	**100%**	96%	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	92%	68%	**100%**	**100%**
g11	100%	100%	100%	100%	100%	100%	100%	**100%**	**100%**	96%	**100%**	96%	**100%**	**100%**
g13	**100%**	**100%**	88%	88%	**100%**	**100%**	**100%**	**100%**	**100%**	48%	40%	0%	**100%**	**100%**
g14	100%	100%	100%	100%	100%	100%	100%	**100%**	80%	**100%**	96%	**100%**	**100%**	**100%**
g15	100%	100%	100%	100%	100%	100%	100%	**100%**	**100%**	**100%**	96%	96%	**100%**	**100%**
g17	**100%**	**100%**	96%	76%	**100%**	**100%**	**100%**	**100%**	4%	28%	16%	4%	**100%**	**100%**
g18	**100%**	**100%**	**100%**	84%	**100%**	**100%**	**100%**	**100%**	92%	**100%**	76%	**100%**	**100%**	**100%**
g19	100%	100%	100%	100%	100%	100%	100%	**100%**	**100%**	**100%**	88%	**100%**	0%	**100%**
g20	0%	0%	0%	0%	**4%**	0%	0%	0%	0%	0%	0%	0%	0%	**100%**
g21	**100%**	**100%**	**100%**	88%	**100%**	**100%**	**100%**	**100%**	60%	68%	60%	92%	**100%**	80%
g22	**100%**	**100%**	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
g23	**100%**	**100%**	**100%**	88%	**100%**	**100%**	**100%**	**100%**	88%	**100%**	40%	92%	**100%**	**100%**
Mean	**95.83%**	**95.83%**	91%	88.17%	91.83%	91.67%	91.67%	91.67%	83.5%	84%	74.17%	76.67%	84%	**95%**

The best result for each test function among the compared algorithms is highlighted in boldface.

that for test function g20, the other six methods fail in reaching the target error accuracy level. However, CMODE can achieve a 100% success rate for this test function. Furthermore, we have provided an improved solution for test function g17. Based on this improved solution, the other six methods, except for GDE, cannot solve this test function even in one run according to the target error accuracy level.[2] Moreover, GDE reaches this improved solution only a few times out of 25 trials.

When performance is reliable, then success performance reflects the convergence speed of an algorithm. Since SaDE employs the sequential quadratic programming method as the local search operator, we cannot exactly calculate how many number of FES is spent by the local search. As a result, the results of SaDE are not included in Table IX. In Table IX, "*NA*" denotes that the success performance is not available since the corresponding success rate provided is 0%. In constrained optimization, the computation time of an algorithm mainly depends on the evaluation of the objective function and the degree of constraint violation, hence, the sum of success performance reflects the overall convergence speed of the algorithm to a certain degree. It can be observed from Table IX that CMODE certainly converges faster than MPDE, GDE, and jDE-2 in terms of the sum of success performance. In addition, the overall convergence performance among εDE, MDE, and CMODE are incomparable since we cannot exactly measure the number of FES for "*NA*." On the other hand, we rank the performance of each algorithm [1 (best) to 6 (worst)] on each test function, and sum the rank numbers for each algorithm over all the test functions, to assess the overall convergence speed of each algorithm. From Table IX, we can see that MDE exhibits the fastest overall convergence speed and the overall convergence performance of CMODE is not very good compared with the other competitors, in terms of the sum of rank.

In summary, the overall performance of CMODE is highly competitive with the six methods compared. Although CMODE does not perform very well in terms of the feasible rate, it is significantly superior to the other six approaches with respect to the success rate.

V. DISCUSSION

This section aims at discussing the effectiveness of some mechanisms proposed in this paper and the effect of parameter settings on the performance of CMODE. It is important to

[2]Recall that we directly take the results of εDE, SaDE, MPDE, GDE, jDE-2, and MDE from the original papers to make the comparison fair, therefore, the success rates of εDE, SaDE, MPDE, jDE-2, and MDE in Table VIII are greater than 0% for test function g17. However, based on the improved solution provided in this paper, the success rates of εDE, SaDE, MPDE, jDE-2, and MDE should be 0%.

TABLE IX
COMPARISON OF CMODE WITH RESPECT TO εDE [26], SADE [27], MPDE [28], GDE [29], JDE-2 [30], AND MDE [31] IN TERMS OF SUCCESS PERFORMANCE

Prob.	Success Performance						Prob.	Success Performance					
	εDE	MPDE	GDE	jDE-2	MDE	CMODE		εDE	MPDE	GDE	jDE-2	MDE	CMODE
g01	5.9E+04	4.3E+04	**4.1E+04**	5.0E+04	7.5E+04	1.2E+05	g13	3.5E+04	7.4E+05	8.7E+05	*NA*	**2.2E+04**	3.1E+04
g02	1.5E+05	3.0E+05	1.5E+05	1.5E+05	**6.0E+04**	1.9E+05	g14	1.1E+05	**4.3E+04**	2.3E+05	9.8E+04	2.9E+05	1.1E+05
g03	8.9E+04	**2.5E+04**	3.5E+06	*NA*	4.5E+04	7.5E+04	g15	8.4E+04	2.0E+05	7.5E+04	2.4E+05	**1.0E+04**	1.3E+04
g04	2.6E+04	2.1E+04	**1.5E+04**	4.1E+04	4.2E+04	7.3E+04	g16	1.3E+04	1.3E+04	1.3E+04	3.2E+04	**8.7E+03**	2.9E+04
g05	9.7E+04	2.2E+05	1.9E+05	4.5E+05	**2.1E+04**	2.9E+04	g17	9.9E+04	7.3E+05	2.1E+06	1.1E+07	**2.6E+04**	1.4E+05
g06	7.4E+03	1.1E+04	6.5E+03	2.9E+04	**5.2E+03**	3.5E+04	g18	5.9E+04	**4.4E+04**	4.8E+05	1.0E+05	1.0E+05	1.1E+05
g07	7.4E+04	**5.7E+04**	1.2E+05	1.3E+05	1.9E+05	1.6E+05	g19	3.5E+05	**1.2E+05**	2.3E+05	2.0E+05	*NA*	2.5E+05
g08	1.1E+03	1.5E+03	1.5E+03	3.2E+03	**9.2E+02**	5.9E+03	g20	*NA*	*NA*	*NA*	*NA*	*NA*	**4.4E+05**
g09	2.3E+04	2.1E+04	3.0E+04	5.5E+04	**1.6E+04**	7.1E+04	g21	1.4E+05	2.1E+05	5.8E+05	1.3E+05	**1.1E+05**	1.3E+05
g10	1.1E+05	**4.8E+04**	8.3E+04	1.5E+05	1.6E+05	1.8E+05	g22	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*
g11	1.6E+04	2.3E+04	8.5E+03	5.4E+04	**3.0E+03**	6.0E+03	g23	**2.0E+05**	2.1E+05	1.1E+06	3.6E+05	3.6E+05	2.4E+05
g12	4.1E+03	4.2E+03	3.1E+03	6.4E+03	**1.3E+03**	5.0E+03	g24	3.0E+03	4.3E+03	3.1E+03	1.0E+04	**1.8E+03**	2.2E+04

Sum of success performance	εDE	1.7E+06+2**NA*
	MPDE	3.1E+06+2**NA*
	GDE	9.8E+06+2**NA*
	jDE-2	1.3E+07+4**NA*
	MDE	1.5E+06+3**NA*
	CMODE	2.5E+06+1**NA*
Sum of rank	εDE	67
	MPDE	68
	GDE	79
	jDE-2	101
	MDE	**58**
	CMODE	95

The best result for each test function among the compared algorithms is highlighted in boldface.

note that only the results of those test functions which cause remarkable difference on the performance of the compared methods are summarized in this section and that other results with negligible difference are omitted for clarity. In addition, this section only reports on mean objective function value, success rate, and feasible rate.

A. Effectiveness of Some Mechanisms Proposed

1) *Effectiveness of the Infeasible Solution Replacement Mechanism:* In order to ascertain whether combining the deterministic replacement with the random replacement exhibits better search performance than the individual replacement mechanisms, we performed computational trials using different replacement schemes. The algorithm with only the deterministic replacement is denoted as CMODE-1, and the algorithm with only the random replacement is denoted as CMODE-2. Table X provides the experimental results of test functions g02, g21, and g23 for CMODE-1 and CMODE, and Table XI provides the experimental results of test functions g10, g13, g14, g20, and g21 for CMODE-2 and CMODE.

As shown in Table X, a negative impact on the performance occurs for test functions g02, g21, and g23 when applying CMODE-1. This may be because the diversity of the population is not good enough for some test cases due to the deterministic replacement.

Additionally, the success rates drastically decrease for test functions g10, g13, g14, g20, and g21 when implementing CMODE-2. Moreover, with respect to test functions g10 and g20, CMODE-2 cannot find the optimal solutions even in one run. Based on our observation, although both CMODE and CMODE-2 can succeed in solving test function g23 in all runs, the results derived from CMODE are of a higher quality than those of CMODE-2. We attribute the above behavior to the fact that the random replacement gives rise to some important individuals being replaced unreasonably.

It is worth remembering that the deterministic replacement is implemented with a higher frequency than the random replacement in CMODE. One may be interested in the performance of the algorithm when applying the deterministic replacement and the random replacement with the same probability (i.e., 0.5). Thus, another algorithm named CMODE-3 is executed for the above purpose. Table XII summarizes the experimental results of test functions g13 and g20 for CMODE and CMODE-3.

As shown in Table XII, CMODE-3 cannot converge to the global optima of test functions g13 and g20 consistently, which suggests a slight drop in the algorithm performance when using CMODE-3.

The above discussions verify that combining these two kinds of replacements is appropriate and that applying the deterministic replacement with a higher frequency than the random replacement is also effective. Moreover, the deterministic replacement seems to be a necessary ingredient in our infeasible solution replacement mechanism.

2) *Effectiveness of the Two Kinds of Rank Value in the Deterministic Replacement:* The deterministic replacement proposed in Section III-B involves two kinds of rank value, which are shown in (11). In order to verify that these two

TABLE X
COMPARISON OF CMODE WITH CMODE-1 ON TEST FUNCTIONS G02, G21, AND G23 OVER 25 RUNS

Prob.	Mean Objective Function Value (Success Rate) [Feasible Rate]		Prob.	Mean Objective Function Value (Success Rate) [Feasible Rate]	
	CMODE	CMODE-1		CMODE	CMODE-1
g02	**−0.8036 (100%) [100%]**	−0.8031 (96%) [100%]	g23	**−400.0551 (100%) [100%]**	−388.0548 (96%) [100%]
g21	**219.9201 (80%) [100%]**	225.1593 (76%) [100%]			

The better result for each test function between the two compared algorithms is highlighted in boldface.

TABLE XI
COMPARISON OF CMODE WITH CMODE-2 ON TEST FUNCTIONS G10, G13, G14, G20, AND G21 OVER 25 RUNS

Prob.	Mean Objective Function Value (Success Rate) [Feasible Rate]		Prob.	Mean Objective Function Value (Success Rate) [Feasible Rate]	
	CMODE	CMODE-2		CMODE	CMODE-2
g10	**7049.2480 (100%) [100%]**	7049.3051 (0%) [100%]	g20	**0.2050 (100%) [0%]**	0.2010 (0%) [0%]
g13	**0.0539 (100%) [100%]**	0.2541 (48%) [100%]	g21	**219.9201 (80%) [100%]**	240.8767 (64%) [100%]
g14	**−47.7649 (100%) [100%]**	−47.7111 (40%) [100%]			

The better result for each test function between the two compared algorithms is highlighted in boldface.

TABLE XII
COMPARISON OF CMODE WITH CMODE-3 ON TEST FUNCTIONS G13 AND G20 OVER 25 RUNS

Prob.	Mean Objective Function Value (Success Rate) [Feasible Rate]		Prob.	Mean Objective Function Value (Success Rate) [Feasible Rate]	
	CMODE	CMODE-3		CMODE	CMODE-3
g13	**0.0539 (100%) [100%]**	0.0693 (96%) [100%]	g20	**0.2050 (100%) [0%]**	0.2049 (88%) [0%]

The better result for each test function between the two compared algorithms is highlighted in boldface.

kinds of rank value are necessary for CMODE, two additional experiments were executed. The first experiment only contains the first rank value in CMODE and is denoted as CMODE-4, and the second experiment only incorporates the second rank value into CMODE and is denoted as CMODE-5. Tables XIII and XIV summarize the results of test functions g10, g20, g21, and g23 for CMODE and CMODE-4, and the results of test functions g03, g13, g17, g20, and g21 for CMODE and CMODE-5, respectively.

As shown in Table XIII, CMODE-4 has difficulty in solving test functions g10 and g20. Besides, CMODE-4 cannot consistently enter the feasible region for test functions g21 and g23.

It is evident from Table XIV that the performance of CMODE-5 is much worse than that of CMODE. We also observe that although both CMODE and CMODE-5 can reach the optimal solution in all runs for test function g23, the results provided by CMODE are of a higher quality.

Based on the results provided in the above experiments, we can conclude that both kinds of ranking are important for the performance of CMODE.

3) *Effectiveness of DE:* One may be interested in how much of the improved performance of CMODE is due to the use of DE. In order to answer this question, an additional experiment named CMODE-6 was implemented in which DE was replaced by the simplex crossover for CMODE. In our experiment, the setting of the expanding factor in the simplex crossover is the same as in Section IV-C. Table XV summarizes the experimental results of test functions g02, g09, g10, g13, g17, g18, g21, and g23 for CMODE and CMODE-6.

As shown in Table XV, CMODE provides higher success rates than CMODE-6 for test functions g02, g09, g10, g13, and g17. For the rest of the three test functions (i.e., g18, g21, and g23), CMODE-6 is unable to consistently find the feasible solutions in all runs.

The above comparison demonstrates that under our framework, DE is more suitable than the simplex crossover as the search engine.

B. Effect of the Parameter Settings

1) *Effect of the Population Size N_p:* Table XVI summarizes the experimental results of test functions g02, g20, and g21 in the case of the population size N_p alone being changed to 120, 150, 180, 200, and 240.

When N_p = 120, the mean objective function value and the success rate of test function g02 are much worse than those when N_p is larger than 120. Also, in the case of N_p = 240, the algorithm provides the worst mean objective function values and success rates for test functions g20 and g21. Moreover, under this condition, the algorithm cannot converge to the optimal solution for test function g20 even in one run.

These results encourage the use of a value between 150 and 200 for the population size.

2) *Effect of the Parameter λ:* As pointed out previously, the parameter λ should be much smaller than the population size N_p since we use a steady-state EA in this paper. In order to analyze the impact on the performance, we varied this parameter and performed runs using: 4, 6, 8, 10, and 12. It is necessary to note that a bigger value of λ signifies a lower number of iterations. Table XVII summarizes the experimental results of test functions g02, g20, g21, and g23.

We can see that the performance degradation tends to occur for test functions g02, g20, and g23 when using a relatively

TABLE XIII
COMPARISON OF CMODE WITH CMODE-4 ON TEST FUNCTIONS G10, G20, G21, AND G23 OVER 25 RUNS

Prob.	Mean Objective Function Value (Success Rate) [Feasible Rate]		Prob.	Mean Objective Function Value (Success Rate) [Feasible Rate]	
	CMODE	CMODE-4		CMODE	CMODE-4
g10	**7049.2480 (100%) [100%]**	7093.4229 (0%) [100%]	g21	**219.9201 (80%) [100%]**	224.0578 (76%) [76%]
g20	**0.2050 (100%) [0%]**	0.1972 (0%) [0%]	g23	**−400.0551 (100%) [100%]**	−388.0288 (0%) [96%]

The better result for each test function between the two compared algorithms is highlighted in boldface.

TABLE XIV
COMPARISON OF CMODE WITH CMODE-5 ON TEST FUNCTIONS G03, G13, G17, G20, AND G21 OVER 25 RUNS

Prob.	Mean Objective Function Value (Success Rate) [Feasible Rate]		Prob.	Mean Objective Function Value (Success Rate) [Feasible Rate]	
	CMODE	CMODE-5		CMODE	CMODE-5
g03	**−1.0005 (100%) [100%]**	−0.7789 (0%) [100%]	g20	**0.2050 (100%) [0%]**	0.2032 (0%) [0%]
g13	**0.0539 (100%) [100%]**	0.3031 (16%) [100%]	g21	**219.9201 (80%) [100%]**	246.1158 (60%) [100%]
g17	**8853.5339 (100%) [100%]**	8856.5086 (88%) [100%]			

The better result for each test function between the two compared algorithms is highlighted in boldface.

TABLE XV
COMPARISON OF CMODE WITH RESPECT TO CMODE-6 ON TEST FUNCTIONS G02, G09, G10, G13, G17, G18, G21, AND G23

Prob.	Mean Objective Function Value (Success Rate) [Feasible Rate]		Prob.	Mean Objective Function Value (Success Rate) [Feasible Rate]	
	CMODE	CMODE-6		CMODE	CMODE-6
g02	**−0.8036 (100%) [100%]**	−0.7964 (0%) [100%]	g17	**8853.5339 (100%) [100%]**	8926.2709 (0%) [100%]
g09	**680.6301 (100%) [100%]**	680.6309 (48%) [100%]	g18	**−0.8660 (100%) [100%]**	−0.6703 (0%) [92%]
g10	**7049.2480 (100%) [100%]**	9674.3599 (0%) [100%]	g21	**219.9201 (80%) [100%]**	442.9414 (0%) [28%]
g13	**0.0539 (100%) [100%]**	0.0847 (84%) [100%]	g23	**-400.0551 (100%) [100%]**	−289.7387 (0%) [0%]

The better result for each test function between the two compared algorithms is highlighted in boldface.

TABLE XVI
MEAN OBJECTIVE FUNCTION VALUE, SUCCESS RATE, AND FEASIBLE RATE ON TEST FUNCTIONS G02, G20, AND G21 WITH VARYING N_p OVER 25 RUNS

Prob.	Mean Objective Function Value (Success Rate) [Feasible Rate]				
	120	150	180	200	240
g02	−0.8003 (80%) [100%]	−0.8032 (96%) [100%]	−0.8036 (100%) [100%]	−0.8036 (100%) [100%]	−0.8036 (100%) [100%]
g20	0.2050 (100%) [0%]	0.2050 (100%) [0%]	0.2050 (100%) [0%]	0.2049 (88%) [0%]	0.2041 (0%) [0%]
g21	240.8767 (64%) [100%]	230.3984 (72%) [100%]	219.9201 (80%) [100%]	219.9201 (80%) [100%]	247.2258 (42%) [100%]

TABLE XVII
MEAN OBJECTIVE FUNCTION VALUE, SUCCESS RATE, AND FEASIBLE RATE ON TEST FUNCTIONS G02, G20, G21, AND G23 WITH VARYING λ OVER 25 RUNS

Prob.	Mean Objective Function Value (Success Rate) [Feasible Rate]				
	4	6	8	12	15
g02	−0.8029 (92%) [100%]	−0.8033 (96%) [100%]	−0.8036 (100%) [100%]	−0.8036 (100%) [100%]	−0.8032 (96%) [100%]
g20	0.2048 (52%) [0%]	0.2050 (100%) [0%]	0.2050 (100%) [0%]	0.2050 (100%) [0%]	0.2049 (88%) [0%]
g21	219.9201 (80%) [100%]	219.9201 (80%) [100%]	219.9201 (80%) [100%]	219.9201 (80%) [100%]	237.6486 (64%) [100%]
g23	−400.0550 (80%) [100%]	−400.0551 (100%) [100%]	−400.0551 (100%) [100%]	−400.0551 (100%) [100%]	−388.0547 (96%) [100%]

TABLE XVIII
MEAN OBJECTIVE FUNCTION VALUE, SUCCESS RATE, AND FEASIBLE RATE ON TEST FUNCTIONS G13, G20, AND G21 WITH VARYING k OVER 25 RUNS

Prob.	Mean Objective Function Value (Success Rate) [Feasible Rate]				
	8	16	22	28	34
g13	0.0539 (100%) [100%]	0.0539 (100%) [100%]	0.0539 (100%) [100%]	0.0539 (100%) [100%]	0.0693 (96%) [100%]
g20	0.2050 (100%) [0%]	0.2050 (100%) [0%]	0.2050 (100%) [0%]	0.2049 (96%) [0%]	0.2049 (88%) [0%]
g21	240.8767 (64%) [100%]	235.6375 (68%) [100%]	219.9201 (80%) [100%]	230.3984 (72%) [100%]	261.2825 (48%) [100%]

TABLE XIX
MEAN OBJECTIVE FUNCTION VALUE, SUCCESS RATE, AND FEASIBLE RATE ON TEST FUNCTIONS G02, G17, G20, G21, AND G23 WITH VARYING F OVER 25 RUNS

Prob.	Mean Objective Function Value (Success Rate) [Feasible Rate]				
	0.4	0.5	0.5∼0.6	0.6	0.7
g02	−0.8001 (76%) [100%]	−0.8023 (88%) [100%]	−0.8036 (100%) [100%]	−0.8036 (100%) [100%]	−0.8020 (84%) [100%]
g17	8866.0388 (48%) [100%]	8856.4961 (96%) [100%]	8853.5339 (100%) [100%]	8853.5339 (100%) [100%]	8853.5339 (100%) [100%]
g20	0.2120 (0%) [0%]	0.2050 (100%) [0%]	0.2050 (100%) [0%]	0.2048 (56%) [0%]	0.2014 (0%) [0%]
g21	219.9201 (80%) [100%]	219.9201 (80%) [100%]	219.9201 (80%) [100%]	219.9201 (80%) [100%]	230.3984 (72%) [100%]
g23	−388.0547 (96%) [100%]	−400.0551 (100%) [100%]	−400.0551 (100%) [100%]	−400.0551 (100%) [100%]	−400.0549 (68%) [100%]

TABLE XX
MEAN OBJECTIVE FUNCTION VALUE, SUCCESS RATE, AND FEASIBLE RATE ON TEST FUNCTIONS G02, G20, G21, AND G23 WITH VARYING C_r OVER 25 RUNS

Prob.	Mean Objective Function Value (Success Rate) [Feasible Rate]				
	0.85	0.9	0.9∼0.95	0.95	1.0
g02	−0.8036 (100%) [100%]	−0.8036 (100%) [100%]	−0.8036 (100%) [100%]	−0.8029 (92%) [100%]	−0.7792 (32%) [100%]
g20	0.2045 (0%) [0%]	0.2049 (92%) [0%]	0.2050 (100%) [0%]	0.2050 (100%) [0%]	0.1956 (0%) [0%]
g21	267.0723 (44%) [100%]	219.9201 (80%) [100%]	219.9201 (80%) [100%]	219.9201 (80%) [100%]	198.9636 (96%) [100%]
g23	−400.0551 (88%) [100%]	−400.0551 (100%) (100%)	−400.0551 (100%) (100%)	−400.0551 (100%) (100%)	−400.0551 (100%) [100%]

small value for this parameter (i.e., 4). In addition, a relatively big value for this parameter also has a negative effect on the performance since the algorithm cannot find the optimal solutions consistently for test functions g02, g20, and g23 and exhibits the worst performance on test function g21 when $\lambda = 15$.

The results in Table XVII reveal that a value between 6 and 12 is a suitable choice for this parameter.

3) *Effect of the Parameter k in the Infeasible Solution Replacement Mechanism:* It is worth noting that k represents the frequency of the infeasible solution replacement mechanism being applied. For investigating the effect of the parameter k on the search ability of CMODE, we tested five different k: 10, 16, 22, 28, and 34. The experimental results of test functions g13, g20, and g21 are provided in Table XVIII.

As shown in Table XVIII, the mean objective function value and the success rate of test function g21 are not good in the case of $k = 8$. In the case of $k = 34$, the mean objective function values and the success rates of test functions g13, g20, and g21 are worse than the other corresponding results.

Based on the above discussion, a value between 16 and 28 is recommended for this parameter.

4) *Effect of the Scaling Factor F in DE:* Five experiments have been performed to study the effect of the scaling factor F in DE by only changing this parameter to 0.4, 0.5, 0.5∼0.6, 0.6, and 0.7, where 0.5∼0.6 denotes this parameter is randomly chosen between 0.5 and 0.6. Table XIX summarizes the experimental results of test functions g02, g17, g20, g21, and g23.

As shown in Table XIX, in the case of $F = 0.4$, the algorithm exhibits the worst performance for test functions g02 and g17. Meanwhile, in the case of $F = 0.7$, the algorithm provides the worst success rates for test functions g21 and g23. Moreover, the algorithms with $F = 0.4$ and 0.7 fail to find the optimal solution for test function g20 in any run. It is interesting to note that the algorithm with $F = 0.5$ and the algorithm with $F = 0.6$ are overall complementary to each other since the former can solve test function g20 and suffers from premature convergence for test functions g02 and g17; however, the latter can solve test functions g02 and g17 and is unable to consistently reach the optimal solution for test function g20.

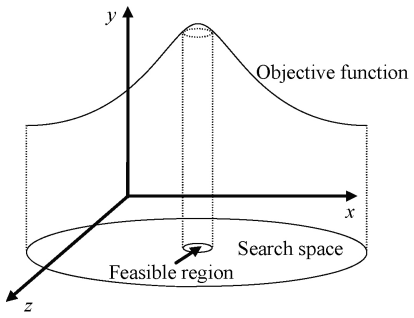

Fig. 9. Schematic diagram of the search space, the feasible region, and the objective function.

The above discussion reveals that randomly choosing a value between 0.5 and 0.6 is suitable for the parameter F.

5) *Effect of the Crossover Control Parameter C_r in DE:* Table XX summarizes the experimental results of test functions g02, g20, g21, and g23 in the case of the crossover control parameter C_r alone being changed to 0.85, 0.9, 0.9∼0.95, 0.95, and 1.0. Note that 0.9∼0.95 means the value of this parameter is randomly chosen from the interval [0.9, 0.95].

Table XX indicates that decreasing C_r causes convergence instability. For instance, the success rates are only 0%, 44%, and 88% for test functions g20, g21, and g23, respectively, in the case of $C_r = 0.85$. In addition, in the case of $C_r = 1.0$ the mean objective function values and the success rates of test functions g02 and g20 are of a much worse quality compared with the other corresponding results, while the algorithm provides the best mean objective function value and success rate for test function g21. Again, the algorithm with $C_r = 0.9$ and the algorithm with $C_r = 0.95$ exhibit the overall

complementary performance, especially for test functions g02 and g20.

From the above discussion, a value between 0.9 and 0.95 is recommended for the crossover control parameter C_r in DE.

VI. Conclusion and Future Work

This paper proposed an improved version of the CW method, called CMODE, which combines multiobjective optimization with differential evolution to deal with COPs. Compared with its previous version, CMODE has two main differences: 1) instead of using the simplex crossover, CMODE exploits DE as the search engine, and 2) a novel infeasible solution replacement mechanism is presented based on multiobjective optimization. The effectiveness of CMODE is demonstrated by 24 benchmark test functions collected in the IEEE CEC2006 special session on constrained real-parameter optimization. The experimental results suggest that CMODE is very competitive with six state-of-the-art methods in the community of constrained evolutionary optimization. CMODE can provide the optimal solutions for 23 test functions and successfully solve 22 test functions consistently. The effectiveness of some mechanisms proposed in this paper and the effect of parameter settings are also demonstrated by various experiments. Due to its effectiveness and robustness, CMODE should gain increasing attention from both researchers and practitioners in the near future.

As shown in (11), we apply equal emphasis to combine the two kinds of rank value for the deterministic replacement in this paper. As a part of our future work, we would like to study whether the performance of CMODE can be further improved by altering the emphasis of these two kinds of rank value when creating the combined rank value.

Despite multiobjective optimization being an effective technique to solve COPs, if we only use Pareto dominance as the way to accept new individuals into the population, the algorithm might not converge to the global optimum, which primarily relies on the characteristics of the problem to be solved. For instance, the features of the search space, the feasible region, and the objective function of the problem to be solved are shown in Fig. 9. From Fig. 9, it is clear that the ratio between the feasible region and the entire search space is very small. Under this condition, suppose that the initial population does not contain any feasible solutions. After some generations, the offspring population might include some feasible solutions. However, such feasible solutions fail to Pareto dominate the infeasible solutions in the parent population based on the characteristics of the problem shown in Fig. 9; thus, they cannot be accepted for the next population. Therefore, in this case if the algorithm requires a new solution to Pareto dominate an old one in order to be accepted, the population cannot contain any feasible solutions in the end. As a result, the algorithm does not have the ability to achieve the global optimum in the feasible region.

Indeed, relatively little effort has been devoted to studying the global convergence property of COEAs so far in the community of constrained evolutionary optimization. In the future, we intend to research the global convergence property of COEAs intensively and to study whether there exists a unified framework under which the algorithm holds the global convergence property.

The source code of CMODE is written in MATLAB and can be obtained from the first author upon request.

References

[1] Z. Cai and Y. Wang, "A multiobjective optimization-based evolutionary algorithm for constrained optimization," *IEEE Trans. Evol. Comput.*, vol. 10, no. 6, pp. 658–675, Dec. 2006.

[2] Z. Michalewicz and M. Schoenauer, "Evolutionary algorithm for constrained parameter optimization problems," *Evol. Comput.*, vol. 4, no. 1, pp. 1–32, Feb. 1996.

[3] C. A. Coello Coello, "Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art," *Comput. Meth. Appl. Mech. Eng.*, vol. 191, nos. 11–12, pp. 1245–1287, Jan. 2002.

[4] E. Mezura-Montes and C. A. Coello Coello, "A simple multimembered evolution strategy to solve constrained optimization problems," *IEEE Trans. Evol. Comput.*, vol. 9, no. 1, pp. 1–17, Feb. 2005.

[5] S. Venkatraman and G. G. Yen, "A generic framework for constrained optimization using genetic algorithms," *IEEE Trans. Evol. Comput.*, vol. 9, no. 4, pp. 424–435, Aug. 2005.

[6] K. Deb, "An efficient constraint handling method for genetic algorithms," *Comput. Methods Appl. Mech. Eng.*, vol. 186, nos. 2–4, pp. 311–338, 2000.

[7] C. A. Coello Coello, "Treating constraints as objectives for single-objective evolutionary optimization," *Eng. Opt.*, vol. 32, no. 3, pp. 275–308, 2000.

[8] T. P. Runarsson and X. Yao, "Search biases in constrained evolutionary optimization," *IEEE Trans. Syst. Man Cybern. C.*, vol. 35, no. 2, pp. 233–243, May 2005.

[9] E. Mezura-Montes and C. A. Coello Coello, "Multiobjective-based concepts to handle constraints in evolutionary algorithms," in *Proc. 4th Mexican Int. Conf. Comput. Sci. (ENC)*, 2003, pp. 192–199.

[10] Y. Zhou, Y. Li, J. He, and L. Kang, "Multiobjective and MGG evolutionary algorithm for constrained optimization," in *Proc. CEC*, 2003, pp. 1–5.

[11] C. A. Coello Coello, "Constraint handling using an evolutionary multiobjective optimization technique," *Civil Eng. Environ. Syst.*, vol. 17, no. 4, pp. 319–346, 2000.

[12] C. A. Coello Coello and E. Mezura-Montes, "Constraint-handling in genetic algorithms through the use of dominance-based tournament selection," *Adv. Eng. Informatics*, vol, 16, no. 3, pp. 193–203, 2002.

[13] A. H. Aguirre, S. B. Rionda, C. A. Coello Coello, G. L. Lizáraga, and E. M. Montes, "Handling constraints using multiobjective optimization concepts," *Int. J. Numer. Methods Eng.*, vol. 59, no. 15, pp. 1989–2017, Apr. 2004.

[14] T. Ray and K. M. Liew, "Society and civilization: An optimization algorithm based on the simulation of social behavior," *IEEE Trans. Evol. Comput.*, vol. 7, no. 4, pp. 386–396, Aug. 2003.

[15] Y. Wang, Z. Cai, Y. Zhou, and W. Zeng, "An adaptive tradeoff model for constrained evolutionary optimization," *IEEE Trans. Evol. Comput.*, vol. 12, no. 1, pp. 80–92, Feb. 2008.

[16] Y. Wang, Z. Cai, G. Guo, and Y. Zhou, "Multiobjective optimization and hybrid evolutionary algorithm to solve constrained optimization problems," *IEEE Trans. Syst. Man Cybern. B Cybern.*, vol. 37, no. 3, pp. 560–575, Jun. 2007.

[17] S. Tsutsui, M. Yamamure, and T. Higuchi, "Multiparent recombination with simplex crossover in real coded genetic algorithms," in *Proc. GECCO*, 1999, pp. 657–664.

[18] J. J. Liang, T. P. Runarsson, E. Mezura-Montes, M. Clerc, P. N. Suganthan, C. A. Coello Coello, and K. Deb, "Problem definitions and evaluation criteria for the CEC 2006," *Special Session on Constrained Real-Parameter Optimization*, Nanyang Technol. Univ., Singapore, Tech. Rep., 2006.

[19] R. Storn and K. Price, "Differential evolution: A simple and efficient adaptive scheme for global optimization over continuous spaces," Int. Comput. Sci. Instit., Berkeley, CA, Tech. Rep. TR-95-012, 1995.

[20] R. Storn, "System design by constraint adaptation and differential evolution," *IEEE Trans. Evol. Comput.*, vol. 3, no. 1, pp. 22–34, Apr. 1999.

[21] Y. C. Lin, K. S. Hwang, and F. S. Wang, "Hybrid differential evolution with multiplier updating method for nonlinear constrained optimization problems," in *Proc. CEC*, 2002, pp. 872–877.
[22] J. Lampinen, "A constraint handling approach for the differential evolution algorithm," in *Proc. CEC*, 2002, pp. 1468–1473.
[23] T. P. Runarsson and X. Yao, "Stochastic ranking for constrained evolutionary optimization," *IEEE Trans. Evol. Comput.*, vol. 4, no. 3, pp. 284–294, Sep. 2000.
[24] E. Mezura-Montes, J. Velázquez-Reyes, and C. A. Coello Coello, "Promising infeasibility and multiple offspring incorporated to differential evolution for constrained optimization," in *Proc. GECCO*, vol. 1. 2005, pp. 225–232.
[25] R. L. Becerra and C. A. Coello Coello, "Cultured differential evolution for constrained optimization," *Comput. Methods Appl. Mech. Eng.*, vol. 195, nos. 33–36, pp. 4303–4322, 2006.
[26] T. Takahama and S. Sakai, "Constrained optimization by the ε constrained differential evolution with gradient-based mutation and feasible elites," in *Proc. CEC*, 2006, pp. 1–8.
[27] V. L. Huang, A. K. Qin, and P. N. Suganthan, "Self-adaptive differential evolution algorithm for constrained real-parameter optimization," in *Proc. CEC*, 2006, pp. 17–24.
[28] M. F. Tasgetiren and P. N. Suganthan, "A multi-populated differential evolution algorithm for solving constrained optimization problem," in *Proc. CEC*, 2006, pp. 33–40.
[29] S. Kukkonen and J. Lampinen, "Constrained real-parameter optimization with generalized differential evolution," in *Proc. CEC*, 2006, pp. 207–214.
[30] J. Brest, V. Zumer, and M. S. Maucec, "Self-adaptive differential evolution algorithm in constrained real-parameter optimization," in *Proc. CEC*, 2006, pp. 215–222.
[31] E. Mezura-Montes, J. Velázquez-Reyes, and C. A. Coello Coello, "Modified differential evolution for constrained optimization," in *Proc. CEC*, 2006, pp. 332–339.
[32] K. Zielinski and R. Laur, "Constrained single-objective optimization using differential evolution," in *Proc. CEC*, 2006, pp. 927–934.
[33] E. Mezura-Montes and B. Cecilia-López-Ramírez, "Comparing bio-inspired algorithms in constrained optimization problems," in *Proc. IEEE CEC*, 2007, pp. 662–669.
[34] W. Gong and Z. Cai, "A multiobjective differential evolution algorithm for constrained optimization," in *Proc. CEC*, Jun. 2008, pp. 181–188.
[35] T. Takahama and S. Sakai, "Constrained optimization by ε constrained differential evolution with dynamic ε-level control," in *Advances in Differential Evolution*, U. K. Chakraborty, Ed. Berlin, Germany: Springer, 2008, pp. 139–154.
[36] M. Zhang, W. Luo, and X. Wang, "Differential evolution with dynamic stochastic selection for constrained optimization," *Information Sci.*, vol. 178, no. 15, pp. 3043–3074, 2008.
[37] K. Zielinski and R. Laur, "Stopping criteria for differential evolution in constrained single-objective optimization," in *Advances in Differential Evolution*, U. K. Chakraborty, Ed. Berlin, Germany: Springer, 2008, pp. 111–138.
[38] K. Deb, "A population-based algorithm-generator for real-parameter optimization," *Soft Comput.*, vol. 9, no. 4, pp. 236–253, 2005.
[39] E. Zitzler, M. Laumannns, and L. Thiele, "SPEA2: Improving the strength Pareto evolutionary algorithm for multiobjective optimization," in *Proc. Evol. Methods Des. Optimization Control Applicat. Ind. Prob. (EUROGEN)*, 2001, pp. 95–100.
[40] S. Tang, Z. Cai, and J. Zheng, "A fast method of constructing the non-dominated set: Arena's principle," in *Proc. 4th ICNC*, 2008, pp. 391–395.
[41] J. Lampinen and R. Storn, "Differential evolution," in *New Optimization Techniques in Engineering*, G. C. Onwubolu and B. Babu, Eds. Berlin/Heidelberg, Germany: Springer-Verlag, 2004, pp. 123–166.
[42] R. Gamperle, S. D. Muller, and P. Koumoutsakos, "A parameter study for differential evolution," in *Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation*, A. Grmela and N. Mastorakis, Eds. WSEAS Press, 2002, pp. 293–298.

Yong Wang (M'08) was born in Hubei, China, in 1980. He received the B.S. degree in automation from the Wuhan Institute of Technology, Wuhan, China, in 2003, and the M.S. degree in pattern recognition and intelligent systems and the Ph.D. degree in control science and engineering both from the Central South University, Changsha, China, in 2006 and 2011, respectively.

Currently, he is a Lecturer with the School of Information Science and Engineering, CSU. His current research interests include evolutional computation, differential evolution, global optimization, constrained optimization, multiobjective optimization, and their real-world applications.

Dr. Wang is a member of the IEEE Task Force on Nature-Inspired Constrained Optimization. He was a reviewer and also has published several papers in the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, the *Evolutionary Computation* (MIT Press), and the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS, PART B: CYBERNETICS.

Zixing Cai (SM'98) received the Diploma degree from the Department of Electrical Engineering, Jiao Tong University, Xi'an, China, in 1962.

He has been teaching and doing research with the School of Information Science and Engineering, Central South University (CSU), Changsha, China, since 1962. From May 1983 to December 1983, he visited the Center of Robotics, Department of Electrical Engineering and Computer Science, University of Nevada, Reno. Then, he visited the Advanced Automation Research Laboratory, School of Electrical Engineering, Purdue University, West Lafayette, IN, from December 1983 to June 1985. From October 1988 to September 1990, he was a Senior Research Scientist with the National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences, Beijing, China, and the National Laboratory of Machine Perception, Center of Information, Beijing University, Beijing. Since February 1989, he has become an Expert of United Nations granted by UNIDO. From September 1992 to March 1993, he visited the NASA Center for Intelligent Robotic Systems for Space Exploration, and the Department of Electrical, Computer and System Engineering, Rensselaer Polytechnic Institute, Troy, NY, as a Visiting Professor. From April 2004 to July 2004, he visited the Institute of Informatics and Automation, Russia Academy of Sciences, Moscow, Russia. From April 2007 to August 2007, he visited Denmark Technical University, Lyngby, Copenhagen, Denmark, as a Visiting Professor. He has authored or co-authored over 600 papers and 30 books/textbooks. His current research interests include intelligent systems, artificial intelligence, intelligent computation, and robotics.

Prof. Cai has received over 30 state, province, university awards in science, technology, and teaching. One of the most recent prizes is the State Eminent Professor Prize of China.