Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2014, Article ID 617905, 15 pages
http://dx.doi.org/10.1155/2014/617905

Research Article

Composite Differential Evolution with Modified Oracle Penalty Method for Constrained Optimization Problems

Minggang Dong,[1,2] Ning Wang,[2] Xiaohui Cheng,[1] and Chuanxian Jiang[1]

[1] *College of Information Science and Engineering, Guilin University of Technology, Guilin 541004, China*
[2] *National Laboratory of Industrial Control Technology, Institute of Cyber-Systems and Control, Zhejiang University, Hangzhou 310027, China*

Correspondence should be addressed to Ning Wang; nwang@iipc.zju.edu.cn

Received 19 April 2014; Revised 1 August 2014; Accepted 13 August 2014; Published 12 October 2014

Academic Editor: Swagatam Das

Motivated by recent advancements in differential evolution and constraints handling methods, this paper presents a novel modified oracle penalty function-based composite differential evolution (MOCoDE) for constrained optimization problems (COPs). More specifically, the original oracle penalty function approach is modified so as to satisfy the optimization criterion of COPs; then the modified oracle penalty function is incorporated in composite DE. Furthermore, in order to solve more complex COPs with discrete, integer, or binary variables, a discrete variable handling technique is introduced into MOCoDE to solve complex COPs with mix variables. This method is assessed on eleven constrained optimization benchmark functions and seven well-studied engineering problems in real life. Experimental results demonstrate that MOCoDE achieves competitive performance with respect to some other state-of-the-art approaches in constrained optimization evolutionary algorithms. Moreover, the strengths of the proposed method include few parameters and its ease of implementation, rendering it applicable to real life. Therefore, MOCoDE can be an efficient alternative to solving constrained optimization problems.

1. Introduction

Most of optimization problems in the real life involve finding a solution that not only is optimum, but also satisfies one or more constraints; these problems are known as constrained optimization problems (COPs). In general, constrained optimization problems are intractable; solving COPs has been known as a challenging research area in computer science, operations research, and optimization domains [1]. In the minimization sense, the general constrained optimization problem can be formulated as follows:

$$\begin{aligned} &\min && f(\vec{x}) \\ &\text{subject to:} && \begin{cases} h_j(\vec{x}) = 0, & j = 1, 2, \ldots, me \\ g_j(\vec{x}) \le 0, & j = me+1, \ldots, m \\ \vec{x} = (x_1, x_2, \ldots, x_n), & x_i \in [l_i, u_i], \\ & i = 1, 2, \ldots, n, \end{cases} \end{aligned} \tag{1}$$

where $f(\vec{x})$, $h(\vec{x})$, and $g(\vec{x})$ are objective function, equality constraint, and inequality constraint, respectively. me is the number of equality constraints, m is the number of all constraints, and n denotes the number of variables. l_i, u_i represent the low bounder and up bounder of x_i, respectively.

Evolutionary algorithms (EAs) have a long history of successfully solving COPs. Compared with traditional gradient based optimization methods, EAs have many merits, including being population-based, derivative-free, and easy to implement. Therefore, research of constrained optimization evolutionary algorithms (COEAs) has become a hot area and attracted a lot of research interests. Extensive surveys of COEAs can be found in recent literature [2–4].

It should be noted that EAs are unconstrained optimization methods that need additional mechanisms to deal with constraints while solving COPs. Therefore, constraint-handling techniques for EAs attract many researchers, and different constraint-handling techniques that are proposed to cope with constraints can be found in the recent survey paper

[4]. According to recent advancement on constraint-handling techniques for EAs, like the work of Wang et al. [5], we divide them into three major categories: penalty function methods, feasibility rules methods, and multiobjective optimization methods.

Penalty function is the most widely used technique to handle constraints due to its simple principle and easy implementation, especially for continuous constrained problems. By adding penalty functions into the objective function, a constrained problem is transformed into an unconstrained one. In general, there are three main penalty methods: death penalty, static penalty, and adaptive penalty [6].

Feasibility rules methods are based on the rules of biasing feasible over infeasible solutions [7]. There are three rules when comparing pairwise individuals: (1) feasible solution is preferred to infeasible solution, (2) the one with a better objective function value is chosen between two feasible solutions, and (3) the one with smaller constraint violation is chosen between two infeasible solutions.

In recent years, multiobjective concepts have been increasingly used in evolutionary algorithm to handle constraints. The common idea of these algorithms is to convert constraints into one or more objectives. According to the different principles of handling constraints, there are two multiobjective methods; one has two objectives: the original objective function and degree of all constraints violation. And the other treats each constraint as an objective; therefore, it has $m + 1$ objectives, where m is the number of constraints [5].

DE, which is a new efficient stochastic population-based heuristic for global optimization in continuous spaces, is presented by Storn and Price [18]. Due to its simple concept, easy implementation, and quick convergence, DE has been successfully applied to a variety of unconstrained continuous optimization problems [19–23]. However, like other nature inspired techniques, there is no provision for constraint handling in its original formulation: relatively fewer works based on DE can be found than those based on other EAs for constrained optimization. A brief review of DE approaches for COPs is provided below.

The first known extension of DE, a multimember DE, for COPs, was proposed by Storn [24]. Kukkonen and Lampinen [25] presented a generalised DE-based approach to solve constrained multiobjective optimization problems. Zielinski and Laur [26] employed feasibility rules with DE to solve COPs. A DE algorithm based on dynamic control of the allowable constraint violation specified by the ε-level was developed by Takahama and Sakai [27]. Multipopulated DE was also proposed by Tasgetiren and Suganthan [28]. And a cooperative-coevolutionary-based DE algorithm was presented by Huang et al. [13] and Yang et al. [29]. Recently, DE was demonstrated as the most competitive algorithm among genetic algorithms (GA), evolution strategies (ES), and particle swarm optimizations (PSO) with the feasibility rules [30]. Combining dynamic stochastic ranking with multimember DE, Zhang et al. [31] presented a hybrid DE for constrained optimization. And DE with a pattern search-based local exploration method is proposed for constrained global optimization [32]. Santana-Quintero et al. [33] proposed a DE with rough set approach to handle constrained multiobjective problems. Mallipeddi and Suganthan [34] proposed an ensemble of four constraint-handling techniques with DE to solve COPs. Mezura-Montes and Miranda-Varela [1] evaluated the performance of DE in constrained numerical optimization. Zou et al. [14] proposed an adaptive DE with a common penalty function method for COPs. da Silva et al. [9] presented an adaptive penalty technique within DE for constrained engineering optimization. Elsayed et al. [10] implement a self-adaptive multioperator differential evolution (SAMO-DE) for solving COPs. Sardar et al. [35] proposed a novel constrained optimizer based on DE by incorporating the idea of gradient-based repair with a DE/rand/1/bin scheme. Compared with two best-known constrained optimizers published, it is very competitive. Combining two-objective method with differential evolution is proposed by Wang and Cai and shows good performance for COPs [36].

In essence, COEAs can be considered as constraint-handling techniques plus EAs [5]. Therefore, to design a new approach to solve COPs, an effective constraint-handling technique needs to be in conjunction with an efficient EA to obtain competitive performance. Recently, a new effective adaptive constraint-handling technique named oracle penalty function was proposed by Schlüter and Gerdts [6], and it was employed in ant colony optimization for solving complex COPs [37, 38]; the results illustrated that oracle penalty function was a new way to handle constrained optimization problems within stochastic metaheuristics. It is robust and easy to implement and handle, and it keeps a high potential in finding global optimum solutions where other methods fail [6, 37, 38]. In addition, more recently, CoDE, which was a DE with composite trial vector generation strategies and control parameters [22], showed better performance in unconstrained continuous optimization problems.

Although the performance of ACO with oracle penalty function has been studied for MINLPs [37, 38], to our best knowledge, there are no reports about DE in conjunction with oracle penalty function. Considering the excellent performances of CoDE in continuous optimization, we try to extend its application domain to practical COPs in engineering. As a result, a new method, called modified oracle penalty function-based CoDE (MOCoDE), is proposed for numerical and engineering constrained optimization problems. To achieve this purpose, the original oracle penalty function approach is modified so as to satisfy the optimization criterion of COPs, and then the modified oracle penalty function is combined with CoDE. Furthermore, in order to solve more complex COPs with discrete, integer, or binary variables, discrete variables handling approach is also introduced into MOCoDE.

To verify the performances of MOCoDE, eleven benchmark functions and seven well-known engineering constrained optimization problems are chosen. The experimental results show that CoDE with the modified oracle penalty function is beneficial, and the proposed method achieves

competitive performances with respect to some other state-of-the-art approaches in the community of constrained optimization evolutionary algorithms.

This paper is organized as follows. After the introduction, in Section 2, CoDE is briefly summarized. Oracle penalty function approach is described in Section 3. In Section 4, the MOCoDE is proposed and explained in detail. Extensive simulation results and comparisons are presented in Section 5. Finally, we end this paper with some conclusions and comments for further research in Section 6.

2. Composite Differential Evolution Algorithm (CoDE)

2.1. Brief Introduction of DE. DE is a new alternative metaheuristic algorithm, and the main idea behind DE is searching for the global optimum solutions with the differences between contemporary individuals. This algorithm has three main stages: mutation, crossover, and selection. In the mutation stage, by adding a weighted difference of two random members to a third member, a trial individual is generated. In the crossover stage, an operator applies between generated trial individual and a predetermined one to generate an offspring. After the generation of a new member, in the selection stage based on a greedy criterion, the offspring will be evaluated using cost function and if the fitness of a new member was better than the fitness of the current member, this candidate replaces the current member.

There are several versions of DE that were reported in the literature [39]. The main difference of these versions is their applied schemes for mutation stage. DEs' merits include simple operator to generate new member, easy implementation, and fast convergence; therefore, DE has been utilized in solving a variety of benchmark problems as well as many real world applications [23, 40]. More information can be found in the recent surveys of DE [41, 42].

2.2. CoDE. Considering that trial vector generation strategies and control parameters have a significant influence on the performance of DE, Wang et al. [22] proposed a novel DE method, CoDE, which combines several trial vector generation strategies with a number of control parameter settings to generate new trial vectors. There are three trial vector generation strategies and three control parameters in CoDE. The selected trial vector generation strategies are

"rand/1/bin",

$$u_{i,j}^{G} = \begin{cases} x_{r1,j}^{G} + F \cdot \left(x_{r2,j}^{G} - x_{r3,j}^{G}\right), & \text{if rand} < C_r \text{ or } j = j_{\text{rand}} \\ x_{i,j}^{G} & \text{otherwise,} \end{cases} \tag{2}$$

"rand/2/bin",

$$u_{i,j}^{G} = \begin{cases} x_{r1,j}^{G} + F \cdot \left(x_{r2,j}^{G} - x_{r3,j}^{G}\right) \\ \quad + F \cdot \left(x_{r4,j}^{G} - x_{r5,j}^{G}\right), & \text{if rand} < C_r \text{ or } j = j_{\text{rand}} \\ x_{i,j}^{G} & \text{otherwise,} \end{cases} \tag{3}$$

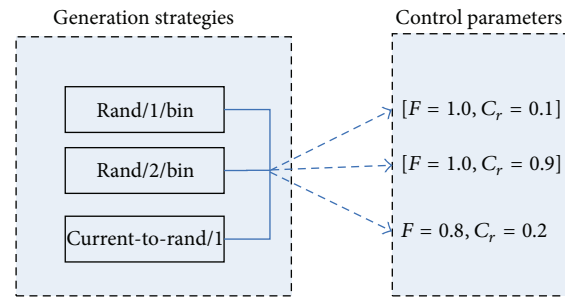

Figure 1: The structure of CoDE.

"current-to-rand/1"

$$\vec{u}_i^{G} = \vec{x}_i^{G} + \text{rand} \cdot \left(\vec{x}_{r1}^{G} - \vec{x}_i^{G}\right) + F \cdot \left(\vec{x}_{r2}^{G} - \vec{x}_{r3}^{G}\right). \tag{4}$$

And the three control parameter settings are

$$\begin{gathered} \left[F = 1.0,\ C_r = 0.1\right], \qquad \left[F = 1.0,\ C_r = 0.9\right], \\ \left[F = 0.8,\ C_r = 0.2\right], \end{gathered} \tag{5}$$

where F is the scaling factor and C_r is the crossover control parameter.

The above strategies and parameter settings are usually used in many DE variants and their properties have been well studied. This method can be illustrated in Figure 1. In CoDE, its strategy candidate pool includes three generation strategies: "rand/1/bin," "rand/2/bin," and "current-to-rand/1," and its parameter candidate pool includes three control parameter settings: $[F = 1.0, C_r = 0.1]$, $[F = 1.0, C_r = 0.9]$, and $[F = 0.8, C_r = 0.2]$. At each generation, three generation strategies with randomly chosen control parameter settings are used to create three new trial vectors. For each target vector, three trial vectors are generated. Then, the best one enters the next generation if it is better than its target vector. More information can be found in [22].

3. Oracle Penalty Function Method

Oracle penalty function approach is a new constraint-handling technique proposed by Schlüter and Gerdts [6]; it belongs to an adaptive penalty method. The key idea of the oracle penalty method is a transformation of the objective function $f(\vec{x})$ of COPs into an additional equality constraint where $g_0(\vec{x}) = f(\vec{x}) - \Omega = 0$. The parameter Ω is named oracle. An objective function is redundant in the transformed problem definition and can be declared as a constant zero function $\tilde{f}(\vec{x}) \equiv 0$. Therefore, the new formulation about COP can be described as follows:

$$\begin{aligned} \min \quad & \tilde{f}(\vec{x}) \equiv 0 \\ \text{Subject to:} \quad & g_0(\vec{x}) = f(\vec{x}) - \Omega = 0 \\ & g_j(\vec{x}) = 0, \quad j = 1, 2, \ldots, me \\ & g_j(\vec{x}) \geq 0, \quad j = me + 1, \ldots, m. \end{aligned} \tag{6}$$

Based on the new formulation form, the rate of objective function and residual function in penalty function can be adaptively adjusted. The description about penalty function $p(\vec{x})$ is given in the following:

$$p(\vec{x}) = \begin{cases} \alpha \cdot \left| f(\vec{x}) - \Omega \right| \\ \quad + (1-\alpha) \cdot \operatorname{res}(\vec{x}), & \text{if } f(\vec{x}) > \Omega \text{ or } \operatorname{res}(\vec{x}) > 0 \\ -\left| f(\vec{x}) - \Omega \right|, & \text{if } f(\vec{x}) \le \Omega,\ \operatorname{res}(\vec{x}) = 0, \end{cases} \tag{7}$$

where $f(\vec{x})$ is objective function and Ω is oracle parameter. As for (7), it can be seen that the first case affects any iterate with an objective function value greater than Ω or any infeasible iterate, while the second case concerns only feasible iterates with an objective function value lower than Ω. And any feasible iterate $\vec{x}$ with $f(\vec{x}) < \Omega$ will be penalized with a negative value. Considering infeasible iterates corresponding to objective function value slower than the oracle parameter Ω and a highly undesired effect when $f(\vec{x}) > \Omega$ and $\operatorname{res}(\vec{x}) < |f(\vec{x}) - \Omega|/3$, α is split into four cases and the value of α is given in (8), and detailed derivation of (8) is given in [6]:

$$\alpha = \begin{cases} \dfrac{\left| f(x) - \Omega \right| \cdot \left(6\sqrt{3} - 2\right)/6\sqrt{3} - \operatorname{res}(\vec{x})}{\left| f(x) - \Omega \right| - \operatorname{res}(\vec{x})}, \\ \qquad \text{if } f(\vec{x}) > \Omega,\ \operatorname{res}(\vec{x}) < \dfrac{\left| f(\vec{x}) - \Omega \right|}{3} \\ 1 - \dfrac{1}{2\sqrt{\left| f(\vec{x}) - \Omega \right| / \operatorname{res}(\vec{x})}}, \\ \qquad \text{if } f(\vec{x}) > \Omega, \\ \qquad \dfrac{\left| f(\vec{x}) - \Omega \right|}{3} \le \operatorname{res}(\vec{x}) \le \left| f(\vec{x}) - \Omega \right| \\ \dfrac{1}{2}\sqrt{\dfrac{\left| f(\vec{x}) - \Omega \right|}{\operatorname{res}(\vec{x})}}, \\ \qquad \text{if } f(\vec{x}) > \Omega,\ \operatorname{res}(\vec{x}) > \left| f(\vec{x}) - \Omega \right| \\ 0 \qquad \text{if } f(\vec{x}) \le \Omega. \end{cases} \tag{8}$$

$\operatorname{res}(\vec{x})$ is the residual function, which measures the constraint violations by applying a norm over all m constraint violations of COPs. Following residual functions based on the l^1, l^2, and l^∞ norm functions are commonly used:

$$\begin{aligned} l^1\text{: } \operatorname{res}(\vec{x}) &= \sum_{i=1}^{me} \left| g_i(\vec{x}) \right| - \sum_{i=me+1}^{m} \min\left\{0, g_i(\vec{x})\right\}, \\ l^2\text{: } \operatorname{res}(\vec{x}) &= \sqrt{\sum_{i=1}^{me} \left| g_i(\vec{x}) \right|^2 + \sum_{i=me+1}^{m} \min\left\{0, g_i(\vec{x})\right\}^2}, \\ l^\infty\text{: } \operatorname{res}(\vec{x}) &= \max\left\{ \left| g_i(\vec{x}) \right|_{i=1,\ldots,me}, \right. \\ &\qquad \left. \left| \min\left\{0, g_i(\vec{x})\right\}_{i=me+1,\ldots,m} \right| \right\}. \end{aligned} \tag{9}$$

Equations (9) are 1-norm, 2-norm, and infinite-norm for residual functions of COPs, respectively. Ω is the only parameter, and the recommend initial values are 10^9 or 10^6, and thanks to subtle design, this approach is robust to Ω. Furthermore, a simple but effective update rule for the oracle parameter is presented in [6]:

$$\Omega^i = \begin{cases} f^{i-1}, & \text{if } f^{i-1} < \Omega^{i-1},\ \operatorname{res}^{i-1} = 0 \\ \Omega^{i-1}, & \text{else.} \end{cases} \tag{10}$$

More information about oracle penalty method can be found in [6]. From the work of Schlüter and Gerdts, oracle penalty method has the following merits. Firstly, it has only one parameter Ω, which needs to be tuned. Secondly, this approach is robust regarding oracle parameter selection. Thirdly, it has shown better performance compared to traditional methods. Since it is universal, it can be easily applied in stochastic metaheuristic approaches.

4. The Proposed MOCoDE Algorithm

4.1. Modified Oracle Penalty Function Method. According to criteria of conference evolutionary computing (CEC) reported about COPs, a solution $\vec{x}$ is regarded as feasible if $|h_j(\vec{x})| - \varepsilon \le 0$, $j = 1, 2, \ldots, me$ and $g_j(\vec{x}) \le 0$, $j = me+1, \ldots, m$, where ε is violation tolerance and 0.0001 is recommended [43]. Although the oracle penalty method shows better performance than other penalty methods, it cannot be employed for COPs directly due to the different constraints violation tolerance criterion and model formulations. More specifically, in oracle penalty method, constraint violation tolerance is applied to all constraints including equality and inequality constraints; however, violation tolerance is only for equality constraint according to the CEC standard [43]. Furthermore, the inequality constraints require $g_j(\vec{x}) \ge 0$ in oracle penalty method while $g_j(\vec{x}) \le 0$ is used in usual COPs formulation. Therefore, oracle penalty method must be modified if it is employed for constraint handling of COPs.

Here, we define new constraint function $g'(\vec{x})$ as follows:

$$g'_j(\vec{x}) = \begin{cases} \varepsilon - \left| h_j(\vec{x}) \right| & j = 1, \ldots, me \\ -g_j(\vec{x}) & j = me+1, \ldots, m. \end{cases} \tag{11}$$

Based on the definition of $g'(\vec{x})$, we can get the following theorem.

Theorem 1. *If $\vec{x}$ is a feasible solution of COPs, then $g'_j(\vec{x}) \ge 0$, $j = 1, 2, \ldots, m$.*

Proof. If $\vec{x}$ is feasible, it means that $|h_j(\vec{x})| - \varepsilon \le 0$ for $j = 1, 2, \ldots, me$ and $g_j(\vec{x}) \le 0$ for $j = me+1, \ldots, m$. Therefore, $\varepsilon - |h_j(\vec{x})|$ is no less than 0 for all equality constraints and $-g_j(\vec{x})$ is also no less than 0 for all inequality constraints. Then, according to the definition of $g'_j(\vec{x})$, we can obtain that $g'_j(\vec{x}) \ge 0$ for $j = 1, 2, \ldots, m$.

Since $g'(\vec{x})$ is introduced, all equality constraints are transformed into inequality. Therefore, all constraints are

inequality, and the residual function with l^1, l^2, and l^∞ norm can be reduced as follows:

$$l^1\colon \operatorname{res}(\vec{x}) = \sum_{i=1}^{m} \min\left\{0, g_i'(\vec{x})\right\}, \tag{12}$$

$$l^2\colon \operatorname{res}(\vec{x}) = \sqrt{\sum_{i=1}^{m} \min\left\{0, g_i'(\vec{x})\right\}^2}, \tag{13}$$

$$l^\infty\colon \operatorname{res}(\vec{x}) = \max\left\{\left|\min\left\{0, g_i'(\vec{x})\right\}_{i=1,\ldots,m}\right|\right\}. \tag{14}$$

According to the standard of feasible solution, the violation tolerance is 0.

In brief, the procedure of modified oracle penalty can be described as follows:

(1) using (11) to transform all original constraints (including equality and inequality constraints) into inequality constraint, which is great than 0;

(2) using (14) to calculate the residual function value;

(3) evaluation of the candidate solution with (7) and (8).

It can be seen that, like oracle penalty function method, modified oracle penalty function is universal as it is applicable to any kind of optimization algorithm, and it is easy to be employed in stochastic metaheuristics, for instance, CoDE. □

4.2. Generalized Discrete Variables Handling Method. According to the work of Liao [17], discrete variables can be divided into four categories: (i) binary variables, (ii) integer variables with uniformly spaced values, (iii) real variables with uniformly spaced values, and (iv) discrete integer or real variables bounded within a range.

To handle different types of discrete variables, here, we adopted the generalized discrete variables handling method proposed in [17]; all discrete variables are also represented as continuous variables. However, for evaluating the fitness function and constraints, they will be converted into corresponding discrete values. In summary, the generalized method for handling discrete variables can be reformulated as follows.

Suppose the possible values of a discrete variable belong to a discrete set $S = \{s_1, s_2, \ldots, s_n\}$; then we can define a position variable p with the following equation (15):

$$p = \operatorname{round}(x), \quad x \in [1, n], \tag{15}$$

where n is the size of S, x is a continuous variable between 1 and n, and round operation is employed to transform x into an integer value. Then, we can get the value d of the discrete variable with the following equation (16):

$$d = S(p) = s_p, \tag{16}$$

where p is the position variable described above.

4.3. MOCoDE Algorithm. Since CoDE is a very competitive metaheuristic algorithm for continuous unconstrained problems, modified oracle penalty function is an effective method for constraint handling, and generalized discrete variables handling method can transform discrete variables into continuous ones easily; they are general and complementary; therefore, it is natural to incorporate them to solve more complex COPs. Therefore, a hybrid algorithm named MOCoDE is presented. The main steps of MOCoDE can be described as follows: (1) set the input parameters; (2) if there are discrete variables, then the generalized discrete variables handling method is used to transform discrete variables into continuous variables; (3) generate the initial population and evaluate the initial individuals with modified oracle penalty function; (4) use CoDE to generate three new trial vectors; (5) according to the value of modified oracle penalty function, select the best one from the three trial vectors; (6) continue steps (4) and (5) until the termination criterion is met. Algorithm 1 gives a detailed description of MOCoDE algorithm.

It should be noted that most parameters of CoDE and modified oracle penalty function approach are adaptive, and MOCoDE has only three parameters: population size *NP*, the maximum number of function evaluation *Max_FES*, and oracle parameters Ω. Furthermore, they are very easy to handle.

5. Experimental Results and Discussions

In this section, numerical experiments are carried out with 11 well-known benchmark functions and 7 classical constrained engineering problems in real life. MOCoDE is coded in Matlab 7.5 and all experiments are executed on a Pentium Dual-Core 2.7 GHz CPU with 2 GB memory PC. According to the relative literature, for benchmark functions suite, three parameters, *NP*, *MAX_FES*, and Ω are set to 30 [22], 240000 [13], and 10^9 [6], respectively. And 50 independent runs were made. For constrained engineering problems, to fairly compare, *NP*, *MAX_FES*, and Ω are set to 30, 90000, and 10^9, respectively. And 100 independent runs were made for 7 constrained engineering problems. The best, the mean, and the worst and standard deviation of objective values are recorded and selected for comparisons. In all cases, the results compared refer to the feasible final solutions found, and a "—" indicates that the value is not available.

5.1. Benchmark Functions Suite. The first experiment concerns a popular suite of problems presented in [43]. These problems have been well studied before as benchmarks by various approaches. All the testing problems can be found in [43] in detail. The summary of the 11 problems of the suite is given in Table 1, where "n", "LI", "NI", "LE", and "NE" denote the number of decide variables, line inequality, no-line inequality, line equality, and no-line inequality, respectively. And "f^*" is the known optimum value. The results of recent different approaches for the G-Suite are summarized in Table 2, which presents statistical results "best," "worst," "mean," and "std" (standard deviation) for each problem.

Input:
NP: the population size.
Max_FES: maximum number of function evaluation.
Ω: the oracle pamameter.
the generation strategies pool: "rand/1/bin", "rand/2/bin", and "current-to-rand/1".
the control parameters pool: $[F = 1.0, C_r = 0.1]$, $[F = 1.0, C_r = 0.9]$, and $[F = 0.8, C_r = 0.2]$.
(1) $G = 0$;
(2) if there are discrete variables, then the generalized discrete variables handling method is used,
(3) generate an initial population $P^0 \{X^0{}_1, \ldots, X^0{}_{\mathrm{NP}}\}$ by uniformly and randomly sampling from the search landspace,
(4) measure the constraint violations with the residual function res(x) for each individual in current population,
(5) evaluate the modified oracle penatly function values $p(x)$ for each individual in current population,
(6) FES = NP;
(7) **while** FES < Max_FES **do**
(8) $P^{G+1} = \emptyset$;
(9) **for** $i = 1 : \mathrm{NP}$ **do**
(10) use the three strategies, with a control parameter setting randomly selected from the control parameter pool, to generate three trial vectors $u^G{}_{i\_1}, u^G{}_{i\_2}$ and $u^G{}_{i\_3}$ for the target vector $X^G{}_i$;
(11) measure the constraint violations with the residual function values res(x) of the three trial vectors $u^G{}_{i\_1}, u^G{}_{i\_2}$ and $u^G{}_{i\_3}$;
(12) evaluate the modified oracle penatly function values $p(x)$ of the three trial vectors $u^G{}_{i\_1}, u^G{}_{i\_2}$ and $u^G{}_{i\_3}$;
(13) choose the best trial vector and add it into P^{G+1};
(14) FES = FES + 3;
(15) **end for**
(16) $G = G + 1$;
(17) **end while**
(18) select the individual X' with the smallest modified oracle penatly function value in the population
Output: the objective function value $f(X')$

Algorithm 1: The framework of MOCoDE algorithm.

Table 1: The information of 11 benchmark functions.

Function	n	Type of $f(x)$	LI	NI	LE	NE	$f^*(x)$
g_1	13	Quadratic	9	0	0	0	−15.0000
g_2	20	Nonlinear	0	2	0	0	−0.8036
g_3	10	Polynomial	0	0	0	1	−1.0005
g_4	5	Quadratic	0	6	0	0	−30665.5387
g_5	4	Cubic	2	0	0	3	5126.4967
g_6	2	Cubic	0	2	0	0	−6961.8139
g_7	10	Quadratic	3	5	0	0	24.3062
g_8	2	Nonlinear	0	2	0	0	−0.0958
g_9	7	Polynomial	0	4	0	0	680.6301
g_{10}	8	Linear	3	3	0	0	7049.2480
g_{11}	2	Quadratic	0	0	0	1	0.7499

Implementation of each algorithm is indicated by its reference.

From Table 2, it can be seen that MOCoDE can obtain the optimum solutions for all benchmark functions and shows good statistical performances for all test functions except for g_3 and g_5. Compared with DUVDE + APM, though DUVDE+APM show better performance for g_5, MOCoDE outperforms DUVDE+APM in terms of both worst and mean indexes for g_1, g_2, g_3, g_7, g_{10}, and g_{11}. Moreover, it needs to be mentioned that the number of function evaluation for DUVDE+APM is 350000 while it is 240000 for MOCoDE. Compared with the state-of-the-art approach SAMO-GA, MOCoDE obtains better mean values for g_2, g_7, g_9, and g_{10}. Furthermore, MOCoDE can find the optimum solution of g_2 while SAMO-GA fails. Compared with SAMO-DE, both of them can find the optimum solutions for all testing functions. Although MOCoDE gets worse mean values for g_3 and g_5, it produces better mean values for g_2, g_7, and g_{10} than SAMO-DE. It is evident that the standard deviation of MOCoDE is also very small; this may mean that the proposed method is robust for these problems.

The comparisons illustrate that the performance of MOCoDE is very competitive with that of several state-of-the-art approaches in constrained evolutionary optimization.

Table 2: The comparison results of different methods.

(a)

Function	HEA-ACT [5]				ES-DE [8]			
	Worst	Best	Mean	Std.	Worst	Best	Mean	Std.
g_1	−8.8	**−15**	−11.9	2.57320	−13.0000	**−15.0000**	−14.8511	$5.02E-01$
g_2	−0.6066	−0.8036	−0.7405	0.04906	−0.530496	−0.803311	−0.738181	$6.67E-02$
g_3	0	−0.2864	−0.0169	0.06382	−1.0000	−1.0000	−1.0000	$0.00E+00$
g_4	−30665.5	−30665.5	−30665.5	0	−30665.54	−30665.54	−30665.54	$2.20E-11$
g_5	5126.4965	5126.4965	5126.4965	0	5129.420	5126.500	5127.290	$1.17E+00$
g_6	−6961.8	−6961.8	−6961.8	0	−6961.814	−6961.814	−6961.814	$2.18E-12$
g_7	29.0591	24.306	25.048	1.37164	24.3077	24.3062	24.3065	$3.97E-04$
g_8	−0.09582	−0.09582	−0.09582	0	−0.095825	−0.095825	−0.095825	$7.80E-17$
g_9	680.63	680.63	680.63	0	680.6301	680.6301	680.6301	$4.46E-13$
g_{10}	8832.12	7049.25	7290.75	379.12207	7050.22	7049.253	7049.418	$1.88E-01$
g_{11}	1	1	1	0	0.7500	0.7500	0.7500	$0.00E+00$

(b)

Function	DUVDE + APM [9]				MOCoDE			
	Worst	Best	Mean	Std.	Worst	Best	Mean	Std.
g_1	−6	−15	−12.5	2.37254	**−15**	**−15**	**−15**	$4.1689e-008$
g_2	−0.6709	−0.8036	−0.7688	0.03568	**−0.79818**	**−0.80362**	**−0.80351**	0.00076891
g_3	−0	−1.0	−0.2015	0.34508	−0.066992	**−1.0005**	−0.77806	0.23028
g_4	−30665.5	−30665.5	−30665.5	0	**−30665.5387**	**−30665.5387**	**−30665.5387**	$1.6909e-008$
g_5	5126.4965	5126.4965	5126.4965	0	5191.2685	**5126.4967**	5131.5314	14.322
g_6	−6961.8	−6961.8	−6961.8	0	**−6961.8139**	**−6961.8139**	**−6961.8139**	$1.7299e-008$
g_7	121.747	24.306	30.404	21.56839	**24.3062**	**24.3062**	**24.3062**	$4.5915e-008$
g_8	−0.09582	−0.09582	−0.09582	0	**−0.095825**	**−0.095825**	**−0.095825**	$7.1126e-009$
g_9	680.63	680.63	680.63	0.00003	**680.6301**	**680.6301**	**680.6301**	$3.7532e-008$
g_{10}	8332.12	7049.25	7351.17	525.62430	**7049.248**	**7049.248**	**7049.248**	$3.6634e-007$
g_{11}	1	0.75	0.98749	0.05590	**0.7499**	**0.7499**	**0.7499**	$1.2157e-008$

Result in boldface indicates that a better result is reached.

(c)

Function	SAMO-GA [10]				SAMO-DE [10]			
	Worst	Best	Mean	Std.	Worst	Best	Mean	Std.
g_1	—	−15.0000	−15.0000	$0.00E+00$	—	−15.0000	−15.0000	$0.00E+00$
g_2	—	−0.80359052	−0.79604769	$5.8025E-03$	—	−0.8036191	−0.79873521	$8.8005E-03$
g_3	—	−1.0005	−1.0005	$0.00E+00$	—	−1.0005	−1.0005	$0.00E+00$
g_4	—	−30665.5386	−30665.5386	$0.00E+00$	—	−30665.5386	−30665.5386	$0.00E+00$
g_5	—	5126.497	5127.976432	$1.1166E+00$	—	5126.497	5126.497	$0.00E+00$
g_6	—	−6961.813875	−6961.813875	$0.00E+00$	—	−6961.813875	−6961.813875	$0.00E+00$
g_7	—	24.3062	24.4113	$4.5905E-02$	—	24.3062	24.3096	$1.5888E-03$
g_8	—	−0.09582504	−0.09582504	$0.00E-00$	—	−0.09582504	−0.09582504	$0.00E+00$
g_9	—	680.630	680.634	$1.4573E-03$	—	680.630	680.630	$1.1567E-05$
g_{10}	—	7049.248	7144.40311	$6.7860E+01$	—	7049.2481	7059.81345	$7.856E+00$
g_{11}	—	0.7499	0.7499	$0.00E+00$	—	0.7499	0.7499	$0.00E+00$

This motivates us to study the performance of MOCoDE for more complex COPs in real life and it will be discussed in the following sections.

5.2. Constrained Engineering Problems. To investigate the performances of MOCoDE on engineering constrained optimization problems in real life, 7 well-studied engineering examples are selected. These problems include three difficult nonconvex optimization problems and four constrained optimization problems with continuous and discrete variables.

5.2.1. Welded Beam Design Problem. The welded beam design problem is a practical design problem that is often employed as a benchmark for testing different optimization methods

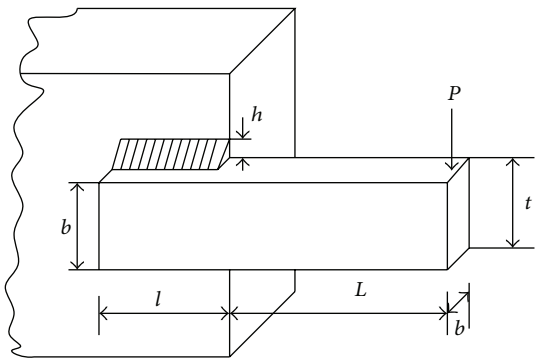

Figure 2: Welded beam design problem.

[11–13]. The objective is to minimize fabricating cost of the welded beam subject to constraints on shear stress (τ), bending stress in the beam (σ), buckling load on the bar (P_c), end deflection of the beam (δ), and side constraints. There are four design variables as shown in Figure 2, $h(x_1), l(x_2), t(x_3)$, and $b(x_4)$.

The problem can be mathematically formulated as follows:

$$\min \quad f(\vec{x}) = 1.10471x_1^2x_2 + 0.04811x_3x_4(14.0 + x_2)$$

$$\text{s.t.} \quad \begin{cases} g_1(\vec{x}) = \tau(\vec{x}) - \tau_{\max} \le 0 \\ g_2(\vec{x}) = \sigma(\vec{x}) - \sigma_{\max} \le 0 \\ g_3(\vec{x}) = x_1 - x_4 \le 0 \\ g_4(\vec{x}) = 0.10471x_1^2 \\ \qquad +0.04811x_3x_4\left(14.0 + x^2\right) - 5.0 \le 0 \\ g_5(\vec{x}) = 0.125 - x_1 \le 0 \\ g_6(\vec{x}) = \delta(\vec{x}) - \delta_{\max} \le 0 \\ g_7(\vec{x}) = P - P_c(\vec{x}) \le 0, \end{cases}$$

$$\tau(\vec{x}) = \sqrt{(\tau')^2 + 2\tau'\tau''\frac{x_2}{2R} + (\tau'')^2},$$

$$\tau' = \frac{P}{\sqrt{2}\,x_1x_2},$$

$$\tau'' = \frac{MR}{J},$$

$$M = P\left(L + \frac{x_2}{2}\right),$$

$$R = \sqrt{\frac{x_2^2}{4} + \frac{(x_1 + x_3)^2}{4}},$$

$$J = 2\left\{\sqrt{2}x_1x_2\left[\frac{x_2^2}{12} + \frac{(x_1 + x_3)^2}{4}\right]\right\},$$

$$\sigma(\vec{x}) = \frac{6PL}{x_4x_3^2},$$

$$\delta(\vec{x}) = \frac{4PL^3}{Ex_3^3x_4},$$

$$P_c(\vec{x}) = \frac{4.013E\sqrt{x_3^2x_4^6/36}}{L^2}\left(1 - \frac{x_3}{2L}\sqrt{\frac{E}{4G}}\right),$$

$$P = 6000lp,$$

$$L = 14in,$$

$$E = 30 \times 10^6 psi,$$

$$G = 12 \times 10^6 psi,$$

$$\tau_{\max} = 13600psi,$$

$$\sigma_{\max} = 30000psi,$$

$$\delta_{\max} = 0.25in$$

$$x_1, x_4 \in [0.1, 2], \quad x_2, x_3 \in [0.1, 10]. \tag{17}$$

The best solutions obtained by different approaches are listed in Table 3, and their statistical results are shown in Table 4.

From the results described in Table 3, it can be seen that the best feasible solution found by MOCoDE is better than those found by other approaches. From Table 4, it can be found that the average searching quality of MOCoDE is also better than those of other methods in terms of best, mean, worst, and standard deviation. It should be noted that even the worst solution found by MOCoDE is better than all the best solutions found by the compared methods. Furthermore, the standard deviation of the results by MOCoDE is the smallest among all considered approaches.

5.2.2. Experimental Results for a Tension/Compression Spring Design Problem. This problem consists of minimizing the weight of a tension/compression spring subject to constraints on shear stress, surge frequency, and minimum deflection as shown in Figure 3. The design variables are the mean coil diameter $D(x_2)$, the wire diameter $d(x_1)$, and the number of active coils $P(x_3)$. The mathematical formulation of this problem can be described as follows:

$$\min \quad f(\vec{x}) = (x_3 + 2)x_2x_1^2$$

$$\text{s.t.} \quad \begin{cases} g_1(\vec{x}) = 1 - \dfrac{x_2^3x_3}{71785x_1^4} \le 0 \\ g_2(\vec{x}) = \dfrac{4x_2^2 - x_1x_2}{12566\left(x_2x_1^3 - x_1^4\right)} + \dfrac{1}{5108x_1^2} - 1 \le 0 \\ g_3(\vec{x}) = 1 - \dfrac{140.45x_1}{x_2^2x_3} \le 0 \\ g_4(\vec{x}) = \dfrac{x_2 + x_1}{1.5} - 1 \le 0 \end{cases}$$

$$x_1 \in [0.05, 2], \quad x_2 \in [0.25, 1.3], \quad x_3 \in [2, 15]. \tag{18}$$

This problem is also recently studied by Huang et al. [13], Zou et al. [14], and Gandomi et al. [8]. The best solutions obtained by different approaches are listed in Table 5, and their statistical results are shown in Table 6.

Table 3: Comparison of the best solutions for welded beam design problem.

Design variables	Coello Coello and Montes [11]	CPSO [12]	CDE [13]	MoCoDE
$x_1(h)$	0.205986	0.202369	0.203137	0.205729639198702
$x_2(l)$	3.471328	3.544214	3.542998	3.470488748041664
$x_3(t)$	9.020224	9.048210	9.033498	9.036623927483428
$x_4(b)$	0.206480	0.205723	0.206179	0.205729641930671
$g_1(x)$	−0.074092	−12.839796	−44.578568	−0.000238821934545
$g_2(x)$	−0.266227	−1.247467	−44.663534	−0.000426438829891
$g_3(x)$	−0.000495	−0.001498	−0.003042	−0.000000002731969
$g_4(x)$	−3.430043	−3.429347	−3.423726	−3.432983758766147
$g_5(x)$	−0.080986	−0.079381	−0.078137	−0.080729639198702
$g_6(x)$	−0.235514	−0.235536	−0.235557	−0.235540322817697
$g_7(x)$	−58.666440	−11.681355	−38.028268	−0.000195113635527
$f(x)$	1.728226	1.728024	1.733462	1.7248523

Table 4: Statistical results of different methods for welded beam design problem.

Method	Best	Mean	Worst	Std.	t
Coello Coello and Montes [11]	1.728226	1.792654	1.993408	—	—
CPSO [12]	1.728024	1.748831	1.782143	0.012926	−18.5138
CDE [13]	1.733461	1.768158	1.824105	0.022194	−19.4909
MOCoDE	1.7249	1.7249	1.7249	1.1106e − 008	—

Table 5: Comparison of the best solutions for spring design problem.

Design variables	Coello Coello and Montes [11]	CPSO [12]	CDE [13]	NMDE [14]	ES-DE [8]	MoCoDE
$x_1(d)$	0.051989	0.051728	0.051609	0.051689	—	0.051718175810237
$x_2(D)$	0.363965	0.357644	0.354714	0.356723	—	0.357418368943598
$x_3(P)$	10.890522	11.244543	11.410831	11.288649	—	11.248015806467105
$g_1(x)$	−0.000013	−0.000845	−0.000039	−1.775e − 11	—	−0.000000060916373
$g_2(x)$	−0.000021	−1.2600e − 05	−0.000183	−7.6e − 13	—	−0.000000447510206
$g_3(x)$	−4.061338	−4.051300	−4.048627	−4.053796	—	−4.055164429852527
$g_4(x)$	−0.722698	−0.727090	−0.729118	−0.727725	—	−0.727242303497443
$f(x)$	0.0126810	0.0126747	0.0126702	0.012665	0.012665	0.012665259791822

Table 6: Statistical results of different methods for spring design problem.

Method	Best	Mean	Worst	Std.	t
Coello Coello and Montes [11]	0.0126810	0.0127420	0.012973	5.900000e − 005	−13.0508
CPSO [12]	0.0126747	0.012730	0.012924	5.198500e − 005	−12.5036
CDE [13]	0.0126702	0.012703	0.012790	2.7000e − 005	−14.0741
NMDE [14]	0.012665	0.012665	0.012665	—	—
ES-DE [8]	0.012665	0.012665	0.012665	3.58E − 009	0
MOCoDE	0.012665	0.012665	0.012665	1.8721e − 008	—

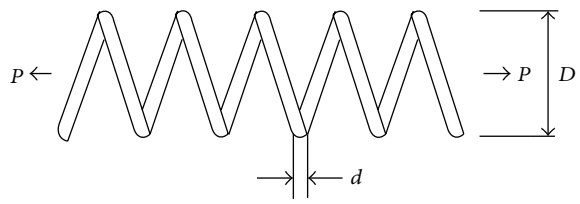

Figure 3: Tension/compression spring design problem.

From Tables 5 and 6, it can be seen that MOCoDE, NMDE, and ES-DE perform better than others and they can find the optimum solution for each time. However, for CDE and CPSO, they only find the approximate optimum solution. MOCoDE, NMDE, and ES-DE obtain the very same statistic performances for this case. However, it should be noted that numbers of parameters of NMDE and ES-DE are 6 and 4, respectively, while MOCoDE has only 3 parameters.

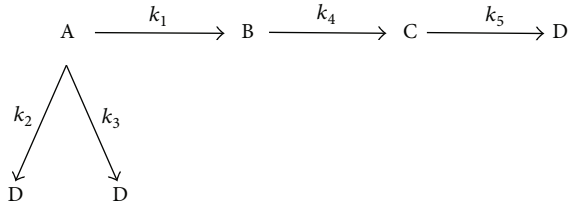

Figure 4: Isothermal continuous stirred tank reactor problem.

5.2.3. Optimization of an Isothermal Continuous Stirred Tank Reactor (CSTR). For the reaction scheme shown below, a problem of finding the best isothermal yield of a product C in a CSTR was chosen. In this reaction scheme, A is the feed material; B is a transient intermediate; C is the required product; D is an unwanted side product. The reaction scheme is depicted in Figure 4.

The rate constants are given by the equation

$$k_i = C_i \exp\left\{\frac{-E_i}{R}\left(\frac{1}{T} - \frac{1}{658}\right)\right\}, \tag{19}$$

where C_i is the frequency factor, E_i is activation energy (cal/mol), R is the gas constant (1.9872 cal/mol/K), and T is absolute temperature in K. The values of C_i and E_i for five rate constants are $C = [1.02, 0.93, 0.386, 3.28, 0.084]$ and $E = [16000, 14000, 15000, 10000, 15000]$. The kinetic equations for the reaction scheme are given as

$$\frac{dx_1}{dt} = -\left(k_1 + k_2 + k_3\right)x_1, \qquad \frac{dx_2}{dt} = k_1 x_1 - k_4 x_2, \qquad \frac{dx_3}{dt} = k_4 x_2 - k_5 x_3, \tag{20}$$

where x_1, x_2, and x_3 are the concentrations of materials A, B, and C, respectively. In this problem, the best isothermal yield of C (f) is found by maximizing this expression with respect to the contact time (t) and the temperature (T) and $f = x_3 = 0.423084$. More information can be found in [15]. A run is declared as successful if the global optimum solution was found within 10^{-5} error. The percentage of successful rates (SR) and average numbers of function evaluation (NFE) are listed in Table 7.

As for CSTR problem, DE, MDE, and MOCoDE can find the optimum solution for each time, while MOCoDE can reduce NFE drastically. It is interesting that though DETL has larger NFE, it obtains lower successful rate than the proposed method.

5.2.4. Reactor Network Design (RND). This example is a reactor network design problem with four discrete variables, describing the system shown in Figure 5. It involves the design of a sequence of two CSTR reactors where the consecutive reaction A → B → C takes place. The objective is to maximize the concentration of product B ($x_4 = C_{B2}$) in the exit stream. This example constitutes a very difficult test problem, as it possesses a local minimum with an objective function value that is very close to that of the global solution

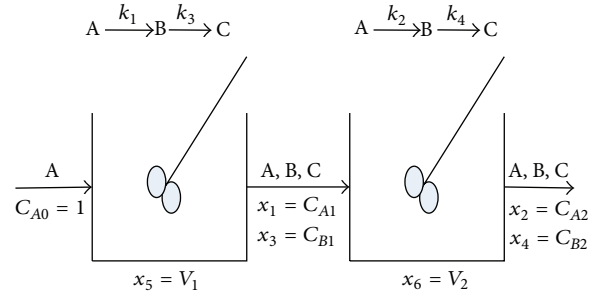

Figure 5: Reactor network design problem.

$f = -0.388812$. The local solutions are with $f = -0.37461$ and $f = -0.38808$:

$$\begin{aligned} &\min \quad f(\vec{x}) = -x_4 \\ &\text{s.t.} \quad \begin{cases} g_1(\vec{x}) = x_1 + k_1 x_1 x_5 - 1 = 0 \\ g_2(\vec{x}) = x_2 - x_1 + k_2 x_2 x_6 = 0 \\ g_3(\vec{x}) = x_3 + x_1 + k_3 x_3 x_5 - 1 = 0 \\ g_4(\vec{x}) = x_4 - x_3 + x_2 - x_1 + k_4 x_4 x_6 = 0 \\ g_5(\vec{x}) = x_5^{0.5} + x_6^{0.5} - 4 \le 0 \end{cases} \\ &x_i \in \{0, 1\}, \quad i = 1, 2, 3, 4, \\ &x_i \in \left[10^{-5}, 16\right], \quad i = 5, 6, \end{aligned} \tag{21}$$

where $k_1 = 0.09755988$, $k_2 = 0.99k_1$, $k_3 = 0.0391908$, and $k_4 = 0.9k_3$. The computational results of recent DE variants are listed in Table 7. From the results of Table 7, as for RND problem, though the NFE for MOCoDE is larger than those provided by DE [15], MDE, [15] and DETL [16], it is totally acceptable and the successful rate of MOCoDE approach is 100% for 100 times run. Although SR of DETL is also high, it should be noted that the high successful rate of DETL depends on a local SQP optimization technique [16].

5.2.5. Process Flowsheeting Problem. This case is a process flowsheeting problem, and it is first studied by Floudas [44]. It is nonconvex because of the first constraint and has two continuous variables and one binary variable. More recently, two hybrid differential evolution algorithms were used to solve this problem [17]. This problem is formulated as follows:

$$\begin{aligned} &\min \quad f(\vec{x}) = -0.7x_3 + 5\left(x_1 - 0.5\right)^2 + 0.8 \\ &\text{s.t.} \quad \begin{cases} g_1(\vec{x}) = -\exp\left(x_1 - 0.2\right) - x_2 \le 0 \\ g_2(\vec{x}) = x_1 - 1.2x_3 - 0.2 \le 0 \\ g_3(\vec{x}) = x_2 + 1.1x_3 + 1.0 \le 0 \end{cases} \\ &x_1 \in [0.2, 1], \quad x_2 \in [-2.22554, -1], \quad x_3 \in \{0, 1\}. \end{aligned} \tag{22}$$

The computational results are listed in Tables 8 and 9.

The comparison of solutions turned very clearly to the advantage of MOCoDE. It can be seen from Table 8 that

Table 7: Computational results of four DE variants for RND and CSTR.

Problem	DE [15]		MDE [15]		DETL [16]		MOCoDE	
	SR[a] (%)	NFE[b]	SR[a] (%)	NFE[b]	SR (%)	NFE[c]	SR (%)	NFE
CSTR	100 (100)	8600 (7996)	100 (100)	7351 (7685)	77	1983 + 15	100	1245
RND	10 (57)	1605 (1468)	8 (44)	1289 (1253)	98	1468 + 25	100	7780

[a]SR with relocating to boundary method (SR with relocating to interior method).
[b]NFE with relocating to boundary method (NFE with relocating to interior method).
[c]The two numbers are NFE required by the method DETL + NFE required for the local optimization.

Table 8: Statistical results of DE approaches for process flowsheeting problem.

Global optimum	MDE′ [17]		MA-MDE′ [17]		MDE′-HIS [17]		MOCoDE	
	SR	NFE	SR	NFE	SR	NFE	SR	NFE
1.076543	40	30986	53.3	25766	83.3	22146	100	11134

MOCoDE can find the optimum solution for each time, and it is superior to MDE′, MA-MDE′, and MDE′-HIS in terms of SR and NFE. From the statistical results of Table 9, MOCoDE get consistent results with very small standard deviation; moreover, even the worst result is better than the mean of the compared approaches.

5.2.6. Process Synthesis Problem. To further demonstrate the performance of MOCoDE, we select a classical process synthesis problem which features nonlinearities in both continuous and binary variables. This problem has seven variables with nine constraints. Recently, this problem was studied by Angira and Babu [15], Liao [17], and Zou et al. [14]. This problem is formulated as follows:

$$\begin{aligned}
\min \quad & f(\vec{x}) = (x_4 - 1)^2 + (x_5 - 1)^2 \\
& + (x_6 - 1)^2 - \ln(x_7 + 1) + (x_1 - 1)^2 \\
& + (x_2 - 2)^2 + (x_3 - 3)^2 \\
\text{s.t.} \quad & \begin{cases} g_1(\vec{x}) = x_4 + x_5 + x_6 + x_1 + x_2 + x_3 - 5 \le 0 \\ g_2(\vec{x}) = x_6^2 + x_1^2 + x_2^2 + x_3^2 - 5.5 \le 0 \\ g_3(\vec{x}) = x_4 + x_1 - 1.2 \le 0 \\ g_4(\vec{x}) = x_5 + x_2 - 1.8 \le 0 \\ g_5(\vec{x}) = x_6 + x_3 - 2.5 \le 0 \\ g_6(\vec{x}) = x_7 + x_1 - 1.2 \le 0 \\ g_8(\vec{x}) = x_6^2 + x_3^2 - 4.25 \le 0 \\ g_9(\vec{x}) = x_5^2 + x_3^2 - 4.64 \le 0 \end{cases} \\
& x_1 \in [0, 1.2], \quad x_2 \in [0, 1.8], \quad x_3 \in [0, 2.5], \\
& x_4, x_5, x_6, x_7 \in \{0, 1\}.
\end{aligned} \tag{23}$$

The global optimum solution is $x = (0.2, 1.28062, 1.954482)$ and $y = (1, 0, 0, 1)$ with $f = 3.557461$. It has several local minima.

To demonstrate the performance of MOCoDE, the results of different methods are given in Tables 10 and 11. From the reported results in Table 10, it can be seen that MOCoDE obtain the best SR while the NFE is not increasing greatly. Furthermore, from the computational results given in Table 11, MOCoDE provide the best results in terms of mean, worst, and standard deviation indicators. In particular, MOCoDE can almost reach the same result and the standard deviation is extremely small.

Results given in Table 10 indicate that MOCoDE algorithm can locate the optimum solution for each time without increasing NFE a lot. As described in Table 11, in terms of the selected performance measures, it can be seen that MOCoDE show the best performance among these methods.

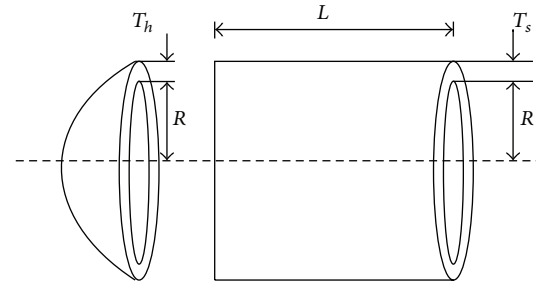

Figure 6: Cylindrical pressure vessel design problem.

5.2.7. Pressure Vessel Design Problem. A cylindrical vessel is capped at both ends by hemispherical heads as shown in Figure 6. The objective is to minimize the total cost, including the cost of material, forming, and welding. It is a mixed discrete-continuous constrained optimization problem. There are four design variables: the thickness of the pressure vessel T_s (x_1), the thickness of the head T_h (x_2), the inner radius of the vessel R (x_3), and the length of the cylindrical component L (x_4). T_s and T_h are the available thicknesses of rolled steel plates, which are integer multiples of 0.0625 inch, and R and L are continuous variables. The problem can be mathematically formulated as follows:

$$\begin{aligned}
\min \quad & f(\vec{x}) = 0.6224 x_1 x_3 x_4 + 1.7781 x_2 x_3^2 \\
& + 3.1661 x_1^2 x_4 + 19.84 x_1^2 x_3
\end{aligned}$$

Table 9: Computational results of DE approaches for process flowsheeting problem.

Method	Best	Mean	Worst	Std.	t
MDE′ [17]	—	1.124453	—	0.075163	−6.37413
MA-MDE′ [17]	—	1.099805	—	0.055618	−4.18244
MDE′-HIS [17]	—	1.094994	—	0.052898	−3.48801
MOCoDE	1.07654309	1.076543128	1.076543152	$1.47e-008$	—

Table 10: Statistical results of DE approaches for process synthesis problem.

Global optimum	MDE′ [17]		MA-MDE′ [17]		MDE′-HIS [17]		MOCoDE	
	SR	NFE	SR	NFE	SR	NFE	SR	NFE
3.557461	30	37739	86.7	20116	83.3	27116	100	27494

Table 11: Computational results of DE approaches for process synthesis problem.

Method	Best	Median	Mean	Worst	Std.	t
SADE [14]	3.557461	3.557461	3.594041	4.63273	—	—
ODE [14]	3.557461	3.557461	3.585754	3.714353	—	—
NMDE [14]	3.557461	3.557461	3.580045	3.896224	—	—
MDE′ [17]	—	—	3.599903	—	0.059012	−7.1921
MA-MDE′ [17]	—	—	3.564912	—	0.029017	−2.56781
MDE′-HIS [17]	—	—	3.561157	—	0.008381	−4.40997
MOCODE	3.557461	3.557461	3.557461	3.557461	$1.1848e-008$	—

Table 12: Comparison of the best solutions for pressure vessel design problem.

Design variables	CPSO [12]	CDE [13]	DUVDE + APM [9]	ES-DE [8]	MOCoDE
$x_1(T_s)$	0.812500	0.812500	0.81250	—	0.812500
$x_2(T_h)$	0.437500	0.437500	0.43750	—	0.437500
$x_3(R)$	42.091266	42.098411	42.09844	—	42.09844559585
$x_4(L)$	176.746500	176.637690	176.63678	—	176.63659584337
$g_1(x)$	−0.000139	$-6.677e-07$	—	—	$-5.4e-14$
$g_2(x)$	−0.035949	−0.035881	—	—	−0.035880829
$g_3(x)$	−116.382700	−3.683016	—	—	−0.000004978
$g_4(x)$	−63.253500	−63.36231	—	—	−63.363404156
$f(x)$	6061.0777	6059.7340	6059.71826	6059.71	6059.7143

Table 13: Statistical results of different methods for pressure vessel design problem.

Method	Best	Mean	Worst	Std	t
CPSO [12]	6061.0777	6147.1332	6363.8041	86.4545	−10.1116
CDE [13]	6059.7340	6085.2303	6371.0455	43.0130	−5.93216
DUVDE + APM [9]	6059.71826	6059.71826	6059.71826	0	−2036618
ES-DE [8]	6059.71	6059.71	6059.71	$9.77E-012$	0
MOCoDE	6059.7143	6059.7143	6059.7143	$1.9444e-008$	—

$$\text{s.t.} \quad \begin{cases} g_1(\vec{x}) = -x_1 + 0.0193x_3 \le 0 \\ g_2(\vec{x}) = -x_2 + 0.00954x_3 \le 0 \\ g_3(\vec{x}) = -\pi x_3^2 x_4 - \dfrac{4}{3}\pi x_3^3 + 1296000 \le 0 \\ g_4(\vec{x}) = x_4 - 240 \le 0 \end{cases}$$

$$x_1, x_2 \in \{0.0625, 0.125, \ldots, 5\}, \quad x_3, x_4 \in [10, 200]. \tag{24}$$

The variables are also limited by $0.0625 \le x_1, x_2 \le 5$, in constant steps of 0.0625 and $10 \le x_3, x_4 \le 200$. The reported optimum solution is 6059.7143. This is a constrained optimization problem with discrete-continuous variables. The results are shown in Tables 12 and 13. Table 12 presents the values of the design variables corresponding to the best solution found by each implementation.

From the results listed in Table 12, it is evident that MOCoDE obtain the best solution among these compared

Table 14: t-test of the significance of difference of MoCoDE with other algorithms.

	HEA-ACT	ES-DE	DUVDE + APM	SAMO-GA	SAMO-DE
g_1	−8.518696649	−2.0973745	−7.45094689	0	0
g_2	−9.080580332	−6.92526383	−6.87723632	−9.01494661	−3.82191113
g_3	−22.5234955	6.814976508	−9.82715065	6.830329703	6.830329703
g_4	0	0	0	0	0
g_5	2.485834334	2.087114192	2.485834334	1.749851202	2.485587473
g_6	0	0	0	0	0
g_7	−3.824121565	−5.34337614	−1.9991273	−16.1892872	−15.1319427
g_8	0	0	0	0	0
g_9	0	0	0	−18.9234643	0
g_{10}	−4.50429335	−6.39405068	−4.06166712	−9.91524072	−9.50980313
g_{11}	−145469610.9	−58164.5785	−30.0539356	0	0
k	7	7	7	5	5

approaches. From Table 13, the statistical results of different methods also demonstrate that the proposed method can solve this COP with discrete-continuous variables effectively and provide competitive statistical results with state-of-the-art algorithms. It should be noted the results of ES-DE do not denote that it can find better solutions due to the accuracy.

5.3. Significance Tests and Discussions. To investigate the statistically significant MoCoDE, independent-samples t-tests have been conducted for benchmark functions and engineering problems based on the above computational results.

5.3.1. Benchmark Functions. t-tests were conducted based on the statistical results of algorithms from Table 2. Significance level is 0.05, sample size is 50, and degree of freedom is 98, $t_{0.05}(98) = 1.984$. t values are listed in Table 14, and the number of significance of difference k is given in the last row of Table 14. It can be seen that MoCoDE has more significance of difference than HEA-ACT, ES-DE, and DUVDE + APM. As for SAMO-GA and SAMO-DE, MoCoDE can obtain competitive results and better performances for g_2, g_7, and g_{10}.

5.3.2. Engineering Problems. Exception problem 3 and problem 4, t-tests were conducted based on the statistical results for the rest of engineering problems. Significance level is 0.05, sample size is 100, and degree of freedom is 198, $t_{0.05}(198) = 1.972$. The t values are given in the last column of Tables 4, 6, 9, 11, and 13.

It can be seen that MoCoDE has more significance of difference than CPSO and CDE for Problems 1, 2, and 7. As for Problem 2, MoCoDE shows more significance of difference than Coello and Montes approach, while it gets competitive result with ES-DE. For Problems 5 and 6, MoCoDE also show better statistical results than MDE′, MA-MDE′, and MDE′-HIS proposed in [43]. As for Problem 7, MoCoDE obtains better performance than DUVDE+APM and gets the same statistical performance with ES-DE.

Compared with recently constrained optimization evolutionary algorithms, these t-test results of significance of difference demonstrate that the proposed algorithm is very competitive. Based on the above simulation results and comparisons, it can be concluded that MOCoDE is of superior searching quality and robustness for constrained problems. Particularly, MOCoDE can almost reach the same results for each run, and the standard deviations are extremely small. In particular, for the constrained engineering optimization problems, including more complex problems with mix discrete-continuous variables, the proposed algorithm has found the global optimum solutions in all runs, which is really interesting.

The effectiveness of MOCoDE lies in that it combines the merits of CoDE and modified penalty method. More specifically, on one hand, with the help of modified oracle penalty to handle constraints, MOCoDE can approach or land in the feasible region of the search space rapidly. On the other hand, thanks to the effective search ability of CoDE, MOCoDE can explore the feasible space sufficiently. Furthermore, since most of the parameters are adaptive, MOCoDE require very few user-defined parameters, rendering it more suitable to real life. Therefore, it can be pointed out that MOCoDE is an alternative approach for constrained optimization problems.

6. Conclusions

A novel constrained differential evolution algorithm is proposed to solve constrained optimization problems in this paper. In order to handle constraints effectively, a modified oracle penalty function method is presented. In addition, a modified oracle method is embedded into CoDE. Furthermore, to solve COPs with mix variables, a generalized discrete variables handling method is introduced to MOCoDE to achieve this purpose.

The numerical results and real world constraint engineering applications reveal the strength of MOCoDE in terms of robustness and global optimality. From the experimental results, it is evident that the proposed approach has substantial potential in handling various COPs, and its performance remarkably outperforms other algorithms presented in recent

literature. Meanwhile, it should be noted that there are fewer parameters to be tuned by user. These characters are extremely important merits for potential implementation and application in real life. Therefore, MOCoDE is a new alternative effective algorithm for COPs. In the future, how to extend MOCoDE to more complex mix integer COPs in real life should be investigated.

Conflict of Interests

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

Acknowledgments

The authors would like to thank the editor and three anonymous reviewers for their constructive and valuable suggestions and comments. This paper has been supported by the National Natural Science Foundations of China under Grants no. 61203109, 61262075, 61262076, and 60874072, Guangxi Natural Science Foundation under Grant no. 2014GXNSFAA118371, and foundations of Guilin University of Technology.

References

[1] E. Mezura-Montes and M. E. a. Miranda-Varela, "Differential evolution in constrained numerical optimization: an empirical study," *Information Sciences*, vol. 180, no. 22, pp. 4223–4262, 2010.

[2] C. A. Coello Coello, "Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art," *Computer Methods in Applied Mechanics and Engineering*, vol. 191, no. 11-12, pp. 1245–1287, 2002.

[3] Y. Wang, Z. X. Cai, Y. R. Zhou, and C. X. Xiao, "Constrained optimization evolutionary algorithms," *Journal of Software*, vol. 20, no. 1, pp. 11–29, 2009 (Chinese).

[4] E. Mezura-Montes and C. A. Coello Coello, "Constraint-handling in nature-inspired numerical optimization: past, present and future," *Swarm and Evolutionary Computation*, vol. 1, no. 4, pp. 173–194, 2011.

[5] Y. Wang, Z. X. Cai, Y. R. Zhou, and Z. Fan, "Constrained optimization based on hybrid evolutionary algorithm and adaptive constraint-handling technique," *Structural and Multidisciplinary Optimization*, vol. 37, no. 4, pp. 395–413, 2009.

[6] M. Schlüter and M. Gerdts, "The oracle penalty method," *Journal of Global Optimization*, vol. 47, no. 2, pp. 293–325, 2010.

[7] K. Deb, "An efficient constraint handling method for genetic algorithms," *Computer Methods in Applied Mechanics and Engineering*, vol. 186, no. 2–4, pp. 311–338, 2000.

[8] A. H. Gandomi, X. Yang, S. Talatahari, and S. Deb, "Coupled eagle strategy and differential evolution for unconstrained and constrained global optimization," *Computers & Mathematics with Applications*, vol. 63, no. 1, pp. 191–200, 2012.

[9] E. K. da Silva, H. J. C. Barbosa, and A. C. C. Lemonge, "An adaptive constraint handling technique for differential evolution with dynamic use of variants in engineering optimization," *Optimization and Engineering*, vol. 12, no. 1-2, pp. 31–54, 2011.

[10] S. M. Elsayed, R. A. Sarker, and D. L. Essam, "Multi-operator based evolutionary algorithms for solving constrained optimization problems," *Computers & Operations Research*, vol. 38, no. 12, pp. 1877–1896, 2011.

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

[12] Q. He and L. Wang, "An effective co-evolutionary particle swarm optimization for constrained engineering design problems," *Engineering Applications of Artificial Intelligence*, vol. 20, no. 1, pp. 89–99, 2007.

[13] F. Huang, L. Wang, and Q. He, "An effective co-evolutionary differential evolution for constrained optimization," *Applied Mathematics and Computation*, vol. 186, no. 1, pp. 340–356, 2007.

[14] D. Zou, H. Liu, L. Gao, and S. Li, "A novel modified differential evolution algorithm for constrained optimization problems," *Computers & Mathematics with Applications*, vol. 61, no. 6, pp. 1608–1623, 2011.

[15] R. Angira and B. V. Babu, "Optimization of process synthesis and design problems: a modified differential evolution approach," *Chemical Engineering Science*, vol. 61, no. 14, pp. 4707–4721, 2006.

[16] M. Srinivas and G. P. Rangaiah, "Differential evolution with tabu list for solving nonlinear and mixed-integer nonlinear programming problems," *Industrial and Engineering Chemistry Research*, vol. 46, no. 22, pp. 7126–7135, 2007.

[17] T. W. Liao, "Two hybrid differential evolution algorithms for engineering design optimization," *Applied Soft Computing Journal*, vol. 10, no. 4, pp. 1188–1199, 2010.

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

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

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

[21] A. K. Qin, V. L. Huang, and P. N. Suganthan, "Differential evolution algorithm with strategy adaptation for global numerical optimization," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 2, pp. 398–417, 2009.

[22] Y. Wang, Z. X. Cai, and Q. F. Zhang, "Differential evolution with composite trial vector generation strategies and control parameters," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 1, pp. 55–66, 2011.

[23] Y. Wang, Z. X. Cai, and Q. F. Zhang, "Enhancing the search ability of differential evolution through orthogonal crossover," *Information Sciences*, vol. 185, pp. 153–177, 2012.

[24] R. Storn, "System design by constraint adaptation and differential evolution," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 1, pp. 22–34, 1999.

[25] S. Kukkonen and J. Lampinen, "Constrained real-parameter optimization with generalized differential evolution," in *Proceedings of the Congress on Evolutionary Computation*, pp. 911–918, IEEE Service Center, Piscataway, NJ, USA, 2006.

[26] K. Zielinski and R. Laur, "Constrained single-objective optimization using differential evolution," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '06)*, pp. 927–934, Piscataway, NJ, USA, July 2006.

[27] T. Takahama and S. Sakai, "Constrained optimization by the ε constrained differential evolution with gradient-based mutation and feasible elites," in *Proceeding of the IEEE Congress on Evolutionary Computation (CEC '06)*, pp. 1–8, Vancouver, Canada, July 2006.

[28] M. F. Tasgetiren and P. N. Suganthan, "A multi-populated differential evolution algorithm for solving constrained optimization problem," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '06)*, pp. 33–40, Piscataway, NJ, USA, July 2006.

[29] Z. Yang, K. Tang, and X. Yao, "Large scale evolutionary optimization using cooperative coevolution," *Information Sciences*, vol. 178, no. 15, pp. 2985–2999, 2008.

[30] E. Mezura-Montes and B. Cecilia-López-Ramírez, "Comparing bio-inspired algorithms in constrained optimization problems," in *Proceedings of the IEEE Congress on Evolutionary Computation*, W. Brauer, K. Berns, and T . Luksch, Eds., pp. 662–669, IEEE Press, Piscataway, NJ, USA, 2007.

[31] M. Zhang, W. Luo, and X. Wang, "Differential evolution with dynamic stochastic selection for constrained optimization," *Information Sciences*, vol. 178, no. 15, pp. 3043–3074, 2008.

[32] M. M. Ali and Z. Kajee-Bagdadi, "A local exploration-based differential evolution algorithm for constrained global optimization," *Applied Mathematics and Computation*, vol. 208, no. 1, pp. 31–48, 2009.

[33] L. V. Santana-Quintero, A. G. Hernández-Díaz, J. Molina, C. A. Coello Coello, and R. Caballero, "DEMORS: a hybrid multiobjective optimization algorithm using differential evolution and rough set theory for constrained problems," *Computers & Operations Research*, vol. 37, no. 3, pp. 470–480, 2010.

[34] R. Mallipeddi and P. N. Suganthan, "Ensemble of constraint handling techniques," *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 4, pp. 561–579, 2010.

[35] S. Sardar, S. Maity, S. Das, and P. N. Suganthan, "Constrained real parameter optimization with a gradient repair based differential evolution algorithm," in *Proceedings of the IEEE Symposium on Differential Evolution (SDE '11)*, pp. 1–8, Paris, France, April 2011.

[36] Y. Wang and Z. Cai, "Combining multiobjective optimization with differential evolution to solve constrained optimization problems," *IEEE Transactions on Evolutionary Computation*, vol. 16, no. 1, pp. 117–134, 2012.

[37] M. Schlüter, J. A. Egea, and J. R. Banga, "Extended ant colony optimization for non-convex mixed integer nonlinear programming," *Computers & Operations Research*, vol. 36, no. 7, pp. 2217–2229, 2009.

[38] M. Schlüter, J. A. Egea, L. T. Antelo, A. A. Alonso, and J. R. Banga, "An extended ant colony optimization algorithm for integrated process and control system design," *Industrial and Engineering Chemistry Research*, vol. 48, no. 14, pp. 6723–6738, 2009.

[39] K. Price, R. Storn, and J. Lampinen, *Differential Evolution: A Practical Approach to Global Optimization*, Natural Computing Series, Springer, Berlin, Germany, 2005.

[40] M. A. Ahandani, N. P. Shirjoposh, and R. Banimahd, "Three modified versions of differential evolution algorithm for continuous optimization," *Soft Computing*, vol. 15, no. 4, pp. 803–830, 2011.

[41] F. Neri and V. Tirronen, "Recent advances in differential evolution: a review and experimental analysis," *Artificial Intelligence Review*, vol. 33, pp. 61–106, 2010.

[42] S. Das and P. N. Suganthan, "Differential evolution: a survey of the state-of-the-art," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 1, pp. 4–31, 2011.

[43] J. J. Liang, T. P. Runarsson, E. Mezura-Montes et al., "Problem definitions and evaluation criteria for the cec 2006 special session on constrained real-parameter optimization," Tech. Rep., Nanyang Technological University, Singapore, 2006.

[44] C. A. Floudas, *Nonlinear and Mixed-Integer Optimization*, Oxford University Press, New York, NY, USA, 1995.