International Journal of Innovative Computing, Information and Control
Volume **11**, Number **3**, June **2015**

 ISSN 1349-4198
pp. 1059–1073

HYBRID METHOD OF CHAOTIC GENETIC ALGORITHM AND BOUNDARY SIMULATION FOR CONSTRAINED OPTIMIZATION

Jie Yang[1], Hong Gao[2,*] and Wei Liu[2]

[1]Transportation Management College
[2]Department of Mathematics
Dalian Maritime University
No. 1, Linghai Road, Dalian 116026, P. R. China
*Corresponding author: gao_hong@qq.com; gaohong@dlmu.edu.cn

Received June 2014; revised November 2014

Abstract. *Many real-world problems are usually subject to some constraints and are posed as constrained optimization problems. In this paper, we present a hybrid method for constrained optimization problems, which combines the improved boundary simulation method, chaotic initialization method and genetic algorithm (GA). A novel boundary simulation method based on the backward binary search technique is proposed to get the boundary of the feasible region. In order to maintain diversity among initial solutions, a chaotic method is proposed to generate initial population from the boundary of the feasible region. Some self-adaptive parameters are designed in crossover and mutation to generate more valid feasible solutions, and a simple repair method is used to update infeasible solutions. The proposed hybrid method is tested on six benchmark functions and three engineering design problems. The results compared with those of some optimization algorithms show the competitive advantage of our algorithm.*
Keywords: Constrained optimization, Genetic algorithm, Boundary simulation, Backward binary search technique, Chaotic initialization

1. **Introduction.** The constrained optimization problem is a hot topic in the operations research. In order to handle constraints, many techniques which contain penalty methods, repair methods, multi-objective methods and hybrid methods have been proposed.

The penalty method is the most popular constraint-handling approach. Its main idea is to add a penalty term to the objective function. Then the original problem can be handled as an unconstrained problem. Joines and Houck [1] proposed a dynamic penalty scheme, in which some time-dependent parameters were introduced to adjust the constraint violation of the infeasible solution. Farmani et al. [2] assigned an infeasibility value to each solution and penalized infeasible solutions depending on their feasibility values. Huang et al. [3] designed a penalty function considering both the amount and the number of constraint violations and evolved penalty factors by differential evolution (DE) method.

The main idea behind repair method is to replace an infeasible solution with a feasible solution. Orvosh and Davis [4] concluded that the optimal result would be better when 5% original solutions were repaired. Chootinan and Chen [5] updated infeasible solutions using the gradient information of constraints. Takahama and Sakai [6] repaired infeasible solutions by gradient-based mutation with some probability and maintained infeasible solutions which have minimum constraint violation. More repair methods were introduced in [7].

The multi-objective method treats the constraints as objective functions and optimizes the constraints and objective simultaneously [8, 9]. Ray and Kang [10] ranked each solution and employed different mating strategies in terms of the constraint level to improve non-dominance solutions. Coello and Montes [11] considered each constraint violation as an objective. Venter and Haftka [12] constructed an additional objective function based on all constraints and solved the unconstrained bi-objective optimization problem.

Hybrid method is also used for constraint handling. Perzina and Ramik [13] incorporated a self-learning GA and heuristic local search operators. Ghodrati and Lotfi [14] applied a hybrid algorithm of cuckoo search and GA to the constrained optimization problem. Oh et al. [15] presented a genetic programming based approximation approach in combination with a multi-membered evolution strategy (GPA-ES).

In this paper, a hybrid GA is proposed to solve the constrained optimization problem. The rest of the paper is organized as follows. Section 2 presents a boundary simulation method based on our backward binary search technique. Using this method, the boundary of the feasible region can be simulated. Section 3 combines our boundary simulation method, chaos theory and GA to form a novel algorithm called boundary simulation chaotic genetic algorithm (BSCGA). In Section 4, we apply BSCGA to six benchmark functions. The results are compared with other methods. In Section 5, we apply BSCGA to three engineering design problems and compare the results with those of some state-of-the-art algorithms. Section 6 involves the conclusion.

2. A Novel Boundary Simulation Method.

The constrained optimization problem has four types of constraints, namely linear inequality, linear equality, nonlinear inequality and nonlinear equality constraints. Since equality constraints can be easily converted into inequality constraints, the constrained optimization problem can be described as follows.

$$\begin{aligned} &\min f(X) \\ &\text{s.t.} \begin{cases} g_j(X) \le 0, & j = 1, 2, \cdots, m, \\ x_i^l \le x_i \le x_i^u, & i = 1, 2, \cdots, n, \end{cases} \end{aligned} \tag{1}$$

where $f(X)$ is the objective function subject to m inequality constraints, $g_j(X)$ is the j-th inequality constraint function, $X = (x_1, x_2, \cdots, x_n)$ is n-dimensional decision vector, x_i^l and x_i^u are the lower and upper bounds of the i-th decision variable.

2.1. The outline of our boundary simulation method.

In order to describe our method, the following basic definitions are introduced.

Definition 2.1. *The feasible region* $\mathcal{F}$ *is*

$$\mathcal{F} = \left\{ X = (x_1, x_2, \cdots, x_n) \middle| \begin{array}{ll} g_j(X) \le 0, & j = 1, 2, \cdots, m, \\ x_i^l \le x_i \le x_i^u, & i = 1, 2, \cdots, n. \end{array} \right\}.$$

Definition 2.2. *The search space* $\mathcal{S}$ *is*

$$\mathcal{S} = \left\{ X = (x_1, x_2, \cdots, x_n) \middle| x_i^l \le x_i \le x_i^u, \quad i = 1, 2, \cdots, n \right\}.$$

Definition 2.3. *The outer space* $\mathcal{S}'$ *is*

$$\mathcal{S}' = \left\{ X = (x_1, x_2, \cdots, x_n) \middle| \tilde{x}_i^l \le x_i \le \tilde{x}_i^u, \quad i = 1, 2, \cdots, n \right\},$$

where $\tilde{x}_i^l < x_i^l$ *and* $\tilde{x}_i^u > x_i^u$. *The outer space* $\mathcal{S}'$ *envelops the search space* $\mathcal{S}$.

The outline of our boundary simulation method is as follows. Firstly, generate several feasible points from the search space (see Section 2.2). Secondly, construct an outer space which envelops the whole search space and generate some infeasible points from the boundary of the outer space (see Section 2.3). Finally, search for boundary points of the feasible region by the proposed backward binary search technique (see Section 2.4).

2.2. **Generate feasible points with GA.**

2.2.1. *Initialization and evaluation.* Randomly generate an initial population which contains *popsize* individuals and use the fitness function (2) to evaluate individuals.

$$f(X) = SP \times (c_{\max} - c_{\min}) + (c_{\max} - c(X)), \tag{2}$$

where SP is a positive parameter for selection, and it reflects the selection pressure. $c_{\max}$ and $c_{\min}$ are the maximum and minimum violation values among the current population. $c(X)$ is the violation value of individual X, and it is computed by Equations (3) to (5).

$$c(X) = \sum_{j=1}^{m} \tilde{q}_j(X), \tag{3}$$

$$\tilde{q}_j(X) = \frac{q_j(X) - q_j^{\min}}{q_j^{\max} - q_j^{\min}}, \tag{4}$$

$$q_j(X) = \begin{cases} g_j(X), & g_j(X) > 0, \\ 0, & g_j(X) \le 0, \end{cases} \tag{5}$$

where $q_j(X)$ is violation value of individual X for the j-th constraint; $q_j^{\max}$ and $q_j^{\min}$ are the maximum and minimum violation values for the j-th constraint among the current population.

2.2.2. *Selection.* In order to avoid the loss of the best individual, we select the individual using combined well-known proportional selection and elitist strategy. The method sorts all individuals in descending order according to their fitness values, and then selects the first EN (the elitist number) individuals as parents. Other parents are selected by the proportional selection operator. All the parents are stored in a mating pool in disorder.

2.2.3. *Crossover.* In the crossover, two self-adaptive parameters are used to make sure the offspring are in the search space. The process can be described by three steps.
Step 1. Select two parents from the mating pool.
Step 2. Generate a random number η from the interval $[0, 1]$. For a given crossover probability p_c, if $\eta > p_c$, randomly duplicate a parent twice as new offspring; if $\eta \le p_c$, generate the new offspring according to Equation (6).

$$x_i^1 = x_i^{mi} + r_i^1(x_i^{ma} - x_i^{mi});\ x_i^2 = x_i^{ma} + r_i^2(x_i^{mi} - x_i^{ma}), \tag{6}$$

where x_i^1 and x_i^2 are the i-th decision variables of the two offspring; x_i^{mi} is the smaller one of the i-th decision variable of the two parents, and x_i^{ma} is the bigger one. r_i^1 and r_i^2 are crossover parameters, and their values are randomly generated from the following intervals.

$$r_i^1 \in \left[\frac{x_i^l - x_i^{mi}}{x_i^{ma} - x_i^{mi}}, \frac{x_i^u - x_i^{mi}}{x_i^{ma} - x_i^{mi}}\right];\ r_i^2 \in \left[\frac{x_i^u - x_i^{ma}}{x_i^{mi} - x_i^{ma}}, \frac{x_i^l - x_i^{ma}}{x_i^{mi} - x_i^{ma}}\right].$$

Step 3. Replace the parents with their offspring and put them into the mating pool.

2.2.4. *Mutation.* In the mutation, we use a self-adaptive parameter to make sure the mutated offspring are in the search space.

The process is to generate a random number η from the interval $[0, 1]$. For a given mutation probability p_m, if $\eta > p_m$, duplicate the original individual as new offspring; if $\eta \leq p_m$, randomly select a decision variable x_i and add a variable bias δ_i to x_i. The value of δ_i is calculated according to Equation (7).

$$\delta_i = r(x_i^u - x_i^l), \tag{7}$$

where r is mutation parameter, and its value is randomly generated from the following interval.

$$\left[\frac{x_i^l - x_i}{x_i^u - x_i^l}, \frac{x_i^u - x_i}{x_i^u - x_i^l}\right].$$

2.2.5. *End condition.* We check the individuals of each generation. If there exists a feasible point, pause the program and output all the feasible points. If the number of feasible points is less than the pre-established value, continue the program; otherwise terminate.

2.3. **Generate infeasible points from the boundary of the outer space.** To generate infeasible points from the boundary of the outer space as defined by Definition 2.3, the two steps are as follows.

Step 1. Find the boundary of the outer space. The bounds on the i-th dimension of the outer space, $\tilde{x}_i^l$ and $\tilde{x}_i^u$, are calculated according to Equation (8), which can make sure the feasible region is in the center of the outer space as much as possible.

$$\tilde{x}_i^l = x_i^{\min} - (x_i^u - x_i^l);\ \tilde{x}_i^u = x_i^{\max} + (x_i^u - x_i^l), \tag{8}$$

where $x_i^{\min}$ and $x_i^{\max}$ are the minimum and maximum values of decision variable x_i among feasible points (generated in Section 2.2).

Step 2. Generate infeasible points. Firstly, select some points in the outer space randomly. Secondly, project these points to the boundary. Fix one component and convert others to corresponding upper or lower bounds.

2.4. **Calculate boundary points of the feasible region with the backward binary search technique.** We present a backward binary search technique to find the boundary point of the feasible region. The detailed steps are as follows.

Step 1. Choose a feasible point a and an infeasible point b.

Step 2. Calculate the midpoint c of interval $[a, b]$.

- **Step 2.1.** If c is feasible, let $c = a$ and go to Step 2.4; otherwise record the interval $[a, c]$ and calculate the midpoint d of interval $[c, b]$.
- **Step 2.2.** If d is feasible, clear the record $[a, c]$, let $d = a$ and go to Step 2.4; otherwise go to Step 2.3.
- **Step 2.3.** If the distance between d and b is less than ε, let $d = b$ and return to Step 2; otherwise let $d = c$ and return to Step 2.1.
- **Step 2.4.** If the distance between a and b is less than ε, output point a; otherwise return to Step 2.

Perform these steps until the program terminates. Then we will get the feasible point a on the boundary of the feasible region.

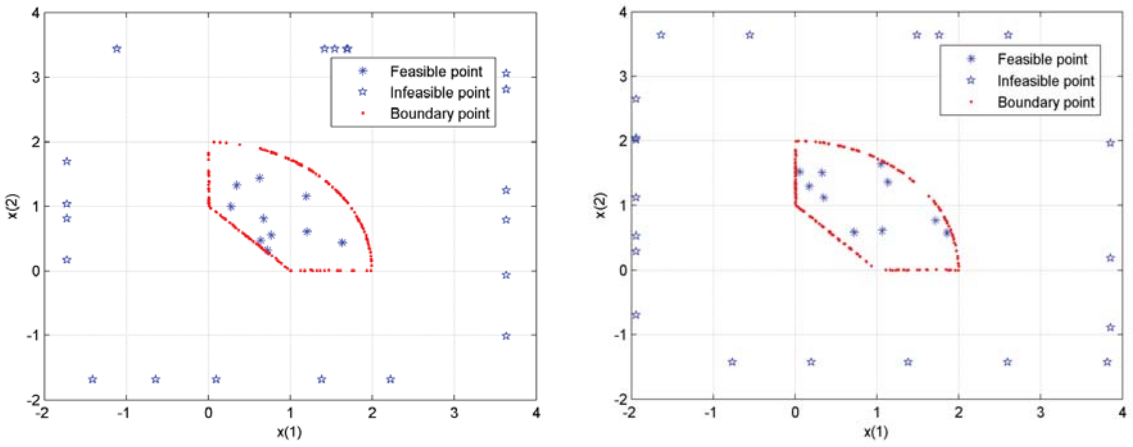
2.5. **Examples.** In order to test the effectiveness of our boundary simulation method, we simulate the boundary of the feasible region for two optimization problems and compare the results with those obtained by the binary search method presented in [16].

The parameters of the boundary simulation method are set as follows: $FP = 10$, $IFP = 20$, $BP = FP \times IFP = 200$, $\varepsilon = 0.001$, where FP, IFP and BP are the numbers of feasible, infeasible and boundary points respectively, ε represents the accuracy.

Example 2.1. *The feasible region is simply connected. Let* $X = (x_1, x_2)$, *and the constrained optimization problem is*

$$\min f(X)$$
$$\text{s.t.} \begin{cases} x_1^2 + x_2^2 \le 4, \\ x_1 + x_2 \ge 1, \\ 0 \le x_1, x_2 \le 2. \end{cases} \tag{9}$$

The results of boundary simulation for this example are shown in Figure 1. The left (a) is by our backward binary search method, and the right (b) is by the binary search method. We can see that, the boundaries simulated are similar. For the constrained optimization problem with connected feasible region, the two search methods are comparative.



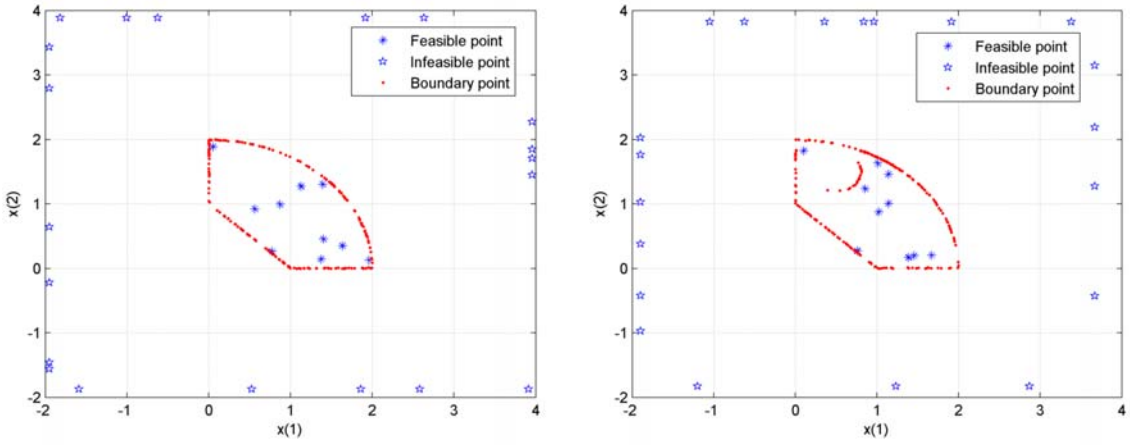
(a) (b)

FIGURE 1. Boundary simulation results for Example 2.1: (a) by the backward binary search, (b) by the binary search method

Example 2.2. *The feasible region is multiply connected. Let* $X = (x_1, x_2)$, *and the constrained optimization problem is*

$$\min f(X)$$
$$\text{s.t.} \begin{cases} x_1^2 + x_2^2 \le 4, \\ x_1 + x_2 \ge 1, \\ (x_1 - 0.5)^2 + (x_2 - 1.5)^2 \ge 0.09, \\ 0 \le x_1, x_2 \le 2. \end{cases} \tag{10}$$

The results of boundary simulation for this example are shown in Figure 2. The left (a) is by our backward binary search method, and the right (b) is by the binary search method.



(a) (b)

FIGURE 2. Boundary simulation results for Example 2.2: (a) by the backward binary search method, (b) by the binary search method

In this example, the feasible region is constructed by digging a round hole whose center is at the point $(0.5, 1.5)$ and radius is 0.3. Figure 2(b) shows not only the external boundary but also the internal boundary of the feasible region. Figure 2(a) only shows the largest boundary which is the boundary we need for initialing population (see Section 3.2).

The binary search method is based on a sorted array. So it is only suitable for the connected feasible region. Our backward binary search method can simulate boundary well not only for connected feasible region but also for multiply connected feasible region.

3. BSCGA.

In this section, we present the boundary simulation chaotic genetic algorithm (BSCGA) for constrained optimization problems. The new hybrid algorithm combines the improved boundary simulation method, chaotic initialization and genetic algorithm.

3.1. Simulation of the boundary of the feasible region.

Using the novel boundary simulation method (see Section 2), we can get BP boundary points of the feasible region. BP is greater than *popsize* which is the population size.

3.2. Chaotic initialization.

Chaotic search is characterized by randomness, ergodicity and sensitive to initial conditions. Based on these characters, we propose a chaotic initialization method to generate the initial population from the boundary of the feasible region.

The chaotic initialization method contains the following three steps.

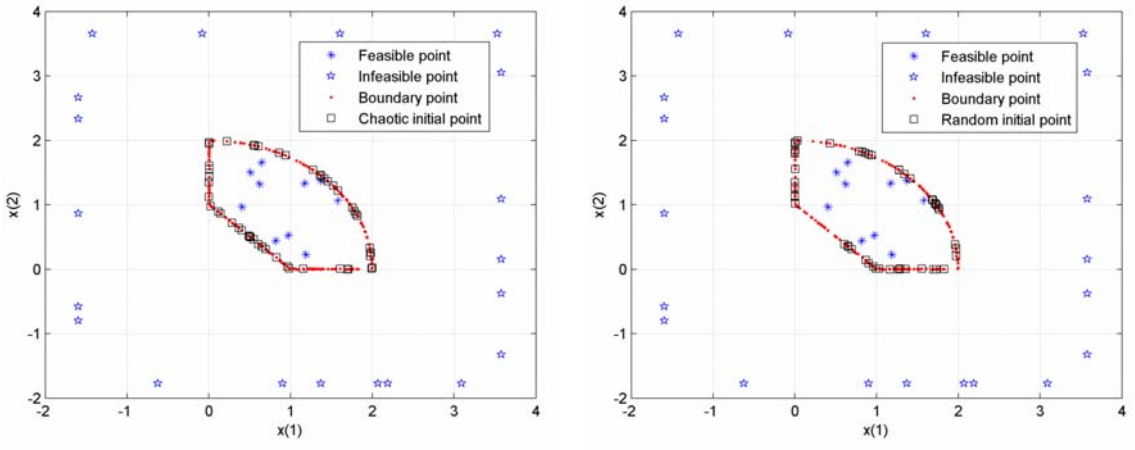
Step 1. Generate n chaotic sequences $\{t_k^1\}, \{t_k^2\}, \cdots, \{t_k^n\}$ with different initial values using the logistic mapping (11), where $k = 1, 2, \cdots, l + popsize$, l is a large positive integer, t_k^i is the k-th iteration value of the i-th chaotic sequence.

$$t_{k+1}^i = 4t_k^i(1 - t_k^i). \tag{11}$$

Step 2. Generate *popsize* chaotic mapping points with the mapping (12).

$$x_k^i = Min_i + t_k^i(Max_i - Min_i), \tag{12}$$

where Min_i and Max_i are minimal and maximal values of the i-th decision variable among the simulated boundary points of the feasible region. Thus, the *popsize* chaotic mapping points $X_k = (x_k^1, x_k^2, \cdots, x_k^n)$ $(k = l + 1, l + 2, \cdots, l + popsize)$ can be generated.



(a) (b)

FIGURE 3. The initial populations: (a) by chaotic initialization method, (b) by random initialization method

Step 3. Select the nearest point with the chaos mapping point from the boundary point set. If Y_k is the nearest point with X_k, then we can get the initial population $\{Y_{l+1}, Y_{l+2}, \cdots, Y_{l+popsize}\}$.

To test the validity of the method, we use our chaotic initialization method and random generation method to initialize the population for Example 2.1. The results are shown in Figure 3. The left (a) is by our chaotic initialization method and the right (b) is by the random generation method.

From Figure 3, we can be see that the initial population generated by chaotic initialization method is uniformly distributed on the boundary of the feasible region. We also compute the radius of the standard deviational circle using Equation (13) to measure the discreteness of the initial population.

$$r = \sqrt{\sum_{i=1}^{popsize} \frac{(x_1 - \bar{x_1})^2 + (x_2 - \bar{x_2})^2}{popsize - 2}}, \tag{13}$$

where $\bar{x_1}$ and $\bar{x_2}$ are the mean values of x_1 and x_2 among the initial points $X = (x_1, x_2)$.

We run each method 20 times. For initial populations generated by chaotic initialization method, we get larger mean value and smaller standard deviation (st.dev.) of the radii of standard deviational circles. It indicates that our chaotic initialization method can better maintain the diversity of the initial population.

3.3. **Genetic operators.** In the genetic process, we use steady-state evolutionary model. The best individual in the current generation is preserved for the next generation, and the worst individual in the current generation is replaced by the best individual so far.

In the evaluation, the fitness value of individual X is computed as Equation (14).

$$f(X) = SP \times (y_{\max} - y_{\min}) + (y_{\max} - y(X)). \tag{14}$$

where $y(X)$ is the objective value of individual X, $y_{\min}$ and $y_{\max}$ are the minimal and maximal objective values among the current population.

In the selection, we use the selection operator (see Section 2.2.2).

In the crossover, we use the crossover operator (see Section 2.2.3). And the values of crossover parameters (in Equation (6)) are generated from the following intervals.

$$r_i^1 \in \left[\frac{Min_i - x_i^{mi}}{x_i^{ma} - x_i^{mi}}, \frac{Max_i - x_i^{mi}}{x_i^{ma} - x_i^{mi}}\right];\ r_i^2 \in \left[\frac{Max_i - x_i^{ma}}{x_i^{mi} - x_i^{ma}}, \frac{Min_i - x_i^{ma}}{x_i^{mi} - x_i^{ma}}\right],$$

where Min_i and Max_i are the minimal and maximal values of the i-th decision variable among simulated boundary points of the feasible region.

In the mutation, we use the mutation operator (see Section 2.2.4). To narrow the search area, we use Equation (15) to compute the variable bias δ_i instead of Equation (7).

$$\delta_i = r(Max_i - Min_i), \tag{15}$$

where r is mutation parameter and its value ranges in the following interval.

$$\left[\frac{Min_i - x_i}{Max_i - Min_i}, \frac{Max_i - x_i}{Max_i - Min_i}\right].$$

To enhance the performance of genetic search and the quality of solutions, we propose a simple binary repair method to repair infeasible solutions. The steps are as follows.

Step 1. Choose a feasible point a and an infeasible point b that needs to be repaired.
Step 2. Calculate the midpoint c of interval $[a, b]$.
Step 3. If c is feasible, terminate; otherwise let $b = c$ and return to Step 2.
Step 4. Replace the infeasible point b with c.

4. **Experiments for Benchmark Functions.** The BSCGA is coded by Matlab M script programming language and compiled with Matlab 7.8. In this section, we apply it to six benchmark functions (see Section 4.1) which have been widely used in optimization researches. All results are compared with some state-of-the-art algorithms.

4.1. **Benchmark functions.** The six benchmark functions are as follows.

g01.

$$\min f(X) = \frac{\sin^3(2\pi x_1)\sin(2\pi x_2)}{x_1^3(x_1 + x_2)}$$
$$\text{s.t.} \begin{cases} x_1^2 - x_2 + 1 \le 0, \\ 1 - x_1 + (x_2 - 4)^2 \le 0, \\ 0 \le x_i \le 10,\ i = 1, 2. \end{cases}$$

g02.

$$\min f(X) = 5.357847x_3^2 + 0.8356891x_1x_5 + 37.293239x_1 - 40792.141$$
$$\text{s.t.} \begin{cases} 0 \le 85.334407 + 0.0056858x_2x_5 + 0.0006262x_1x_4 - 0.0022053x_3x_5 \le 92, \\ 90 \le 80.51249 + 0.0071317x_3x_5 + 0.0029955x_1x_2 + 0.0021813x_3^2 \le 110, \\ 20 \le 9.300961 + 0.0047026x_3x_5 + 0.0012547x_1x_3 + 0.0019085x_3x_4 \le 25, \\ 78 \le x_1 \le 102,\ 33 \le x_2 \le 45,\ 27 \le x_i \le 45,\ i = 3, 4, 5. \end{cases}$$

g03.

$$\min f(X) = (x_1 - 10)^3 + (x_2 - 20)^3$$
$$\text{s.t.} \begin{cases} 100 - (x_1 - 5)^2 - (x_2 - 5)^2 \le 0, \\ (x_1 - 6)^2 + (x_2 - 5)^2 - 82.81 \le 0, \\ 13 \le x_1 \le 100,\ 0 \le x_2 \le 100. \end{cases}$$

g04.

$$\min f(X) = 5\sum_{i=1}^{4} x_i - 5\sum_{i=1}^{4} x_i^2 - \sum_{i=5}^{13} x_i$$

$$\text{s.t.} \begin{cases} 2x_1 + 2x_2 + x_{10} + x_{11} - 10 \le 0, \\ 2x_1 + 2x_3 + x_{10} + x_{12} - 10 \le 0, \\ 2x_2 + 2x_3 + x_{11} + x_{12} - 10 \le 0, \\ -8x_1 + x_{10} \le 0, \\ -8x_2 + x_{11} \le 0, \\ -8x_3 + x_{12} \le 0, \\ -2x_4 - x_5 + x_{10} \le 0, \\ -2x_6 - x_7 + x_{11} \le 0, \\ -2x_8 - x_9 + x_{12} \le 0, \\ 0 \le x_i \le 1,\ i = 1, 2, \cdots, 9, \\ 0 \le x_i \le 100,\ i = 10, 11, 12, \\ 0 \le x_{13} \le 1. \end{cases}$$

g05.

$$\min f(X) = (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)^2 + 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7$$

$$\text{s.t.} \begin{cases} 127 - 2x_1^2 - 3x_2^4 - x_3 - 4x_4^2 - 5x_5 \ge 0, \\ 282 - 7x_1 - 3x_2 - 10x_3^2 - x_4 + x_5 \ge 0, \\ 196 - 23x_1 - x_2^2 - 6x_6^2 + 8x_7 \ge 0, \\ -4x_1^2 - x_2^2 + 3x_1x_2 - 2x_3^2 - 5x_6 + 11x_7 \ge 0, \\ -10 \le x_i \le 10,\ i = 1, 2, \cdots, 7. \end{cases}$$

g06.

$$\min f(X) = x_1 + x_2 + x_3$$

$$\text{s.t.} \begin{cases} 1 - 0.0025(x_4 + x_6) \ge 0, \\ 1 - 0.0025(x_5 + x_7 - x_4) \ge 0, \\ 1 - 0.01(x_8 - x_5) \ge 0, \\ x_1x_6 - 833.33252x_4 - 100x_1 + 83333.333 \ge 0, \\ x_2x_7 - 1250x_5 - x_2x_4 + 1250x_4 \ge 0, \\ x_3x_8 - x_3x_5 + 2500x_5 - 1250000 \ge 0, \\ 100 \le x_1 \le 10000, \\ 1000 \le x_i \le 10000,\ i = 2, 3, \\ 10 \le x_i \le 1000,\ i = 4, 5, \cdots, 8. \end{cases}$$

These problems have various objective functions (i.e., quadratic, cubic, linear, nonlinear and polynomial functions) and a wide range of feasible ratio from 0.001% to 52.123%.

4.2. **Parameters selection.** In the boundary simulation method of BSCGA, there are three parameters involved, namely FP, IFP and ε (see Section 2.5). The three parameters are experimentally set as follows: $FP = 10$, $IFP = 100$, $\varepsilon = 0.001$.

In the chaotic initialization method of BSCGA, in order to guarantee the diversity of the initial population, the parameter l should be a large number. We set $l = 1000$.

Other parameters are conventional parameters of GA. They are set as follows: population size $popsize = 200$, selection pressure $SP = 1$, elitist number $EN = 40$, crossover probability $p_c = 0.8$, mutation probability $p_m = 0.2$, and the maximum number of function evaluation $MFE = 350,000$ which is the terminal condition.

4.3. **Results.** We run BSCGA 25 times for each benchmark function. The best, mean, worst results and the standard deviation of the optimum results are shown in Table 1.

TABLE 1. The results of BSCGA for six benchmark problems

	g01	g02	g03	g04	g05	g06
best	–0.105279	–30681.670	–6961.891	–15.000	680.631	7049.253
mean	–0.096940	–30679.392	–6961.882	–15.000	680.631	7051.546
worst	–0.094283	–30673.589	–6961.880	–15.000	680.636	7054.655
st.dev.	1.01E–03	2.45E+00	3.42E–03	0.00E+00	2.61E–03	3.29E+00

We also compare our results with some other algorithms, such as stochastic ranking strategy (SR) [17], self-adaptive fitness representation (SFR) [2], gradient-based repair method (GR) [5] and GPA-ES [15]. The comparisons of the "best" results and "mean" results are shown in Table 2 and Table 3.

TABLE 2. Comparison of the best results for benchmark functions

	g01	g02	g03	g04	g05	g06
BSCGA	**–0.105279**	**–30681.670**	**–6961.891**	–15.000	680.631	**7049.253**
SR	–0.095825	–30665.539	–6961.814	–15.000	680.630	7054.316
SFR	–0.095825	–30665.500	–6961.800	–15.000	680.640	7049.340
GR	–0.095825	–30665.539	–6961.814	–15.000	680.630	7049.261
GPA-ES	–0.095825	–30665.539	–6961.814	–15.000	680.630	7081.948

TABLE 3. Comparison of the mean results for benchmark functions

	g01	g02	g03	g04	g05	g06
BSCGA	**–0.096940**	**–30679.392**	**–6961.882**	–15.000	**680.631**	7051.546
SR	–0.095825	–30665.539	–6961.814	–15.000	680.640	7372.613
SFR	–0.095825	–30665.200	–6961.800	–15.000	680.720	7627.890
GR	–0.095825	–30665.534	–6961.814	–15.000	680.638	7049.566
GPA-ES	–0.095825	–30648.853	–6961.814	–15.000	680.648	7342.196

From Table 2, we can see that for problem g01, g02, g03, g06, our "best" results are better than those of other algorithms. For problem g04, the "best" result is the same as others. For problem g05, our "best" result is very close to the best solution.

From Table 3, we can see that for problem g01, g02, g03, g05, our "mean" results are better than those of other algorithms. For problem g04, the "mean" result is the same as others. For problem g06, our "mean" result is very close to that of GR which is the best "mean" result.

5. **Experiments for Engineering Design Problems.** In this section, we apply BSCGA to three engineering design problems, and each problem has been solved 25 times. The parameter settings are the same as those in Section 4.2 except that MFE is set to 80,000. The results are compared with those of other optimization methods including GA with a dominance-based selection scheme (DS-GA) [11], hybrid particle swarm optimization (HPSO) [18], DE based on a co-evolution mechanism (CDE) [3], hybrid Nelder-Mead simplex search method and PSO (NM-PSO) [19], hybrid PSO and a spreadsheet "Solver" (PSOLOVER) [20] and improved constrained electromagnetism-like mechanism (ICEM) [21].

5.1. **The pressure vessel design problem.** This problem aims to minimize the total cost of material, forming and welding subject to four inequality constraints. It has four variables: thickness of the shell (x_1), thickness of the head (x_2), inner radius (x_3) and length of the cylindrical section of the vessel (x_4). The formulated problem can be found in [12].

In Table 4, the best, mean, worst results, the standard deviation of the optimum results and NFE which is the number of function evaluation are compared with those of other methods. The values of decision variables and constrained functions corresponding to the "best" result are called the best solutions. The comparison of the best solutions with those of other methods is shown in Table 5.

TABLE 4. Comparison of the results for pressure vessel design problem

	best	mean	worst	st.dev.	NFE
BSCGA	5918.4422	5948.4601	6025.8620	3.44E+01	80,000
DS-GA	6059.9463	6177.2533	6469.3220	1.31E+02	80,000
HPSO	6059.7143	6099.9323	6288.6770	8.62E+01	81,000
CDE	6059.7340	6085.2303	6371.0455	4.30E+01	204,800
NM-PSO	5930.3137	5946.7901	5960.0557	9.16E+00	80,000
PSOLOVER	6059.7143	6059.7143	6059.7143	4.63E–12	310
ICEM	6059.7143	6059.7143	6059.7143	9.10E–13	80,000

TABLE 5. Comparison of the best solutions for pressure vessel design problem

	$f(X)$(best)	x_1	x_2	x_3	x_4
BSCGA	5918.4422	0.7970	0.3940	41.2953	186.8497
DS-GA	6059.9463	0.8125	0.4375	42.0974	176.6540
HPSO	6059.7143	0.8125	0.4375	42.0984	176.6366
CDE	6059.7340	0.8125	0.4375	42.0984	176.6377
NM-PSO	5930.3137	0.8036	0.3972	41.6392	182.4120
PSOLOVER	6059.7143	0.8125	0.4375	42.0984	176.6366
ICEM	6059.7143	0.8125	0.4375	42.0984	176.6366
	$g_1(X)$	$g_2(X)$	$g_3(X)$	$g_4(X)$	
BSCGA	–7.10E–07	–4.28E–05	–3.68E–01	–53.15	
DS-GA	–2.00E–05	–3.59E–02	–2.79E+01	–63.35	
HPSO	–8.00E–11	–3.59E–02	–2.72E–04	–23.36	
CDE	–6.68E–07	–3.59E–02	–3.68E+00	–63.36	
NM-PSO	3.66E–05	3.80E–05	–1.59E+00	–17.59	
PSOLOVER	–8.00E–11	–3.59E–02	–2.72E–04	–23.36	
ICEM	–8.00E–11	–3.59E–02	–2.72E–04	–23.36	

From Table 4, we can see that the "best" result obtained by BSCGA is clearly better than those of other algorithms. And the best solution of BSCGA satisfies all constraints, while the best solution of NM-PSO violates two constraints (see Table 5). BSCGA gets a better "mean" result which is very close to the best "mean" result (obtained by NM-PSO), and the standard deviation obtained by BSCGA is acceptable.

5.2. **The tension/compression spring design problem.** This problem aims to minimize the weight of a tension/compression spring subject to four inequality constraints with three variables such as the wire diameter (x_1), the mean coil diameter (x_2) and the number of active coils (x_3). The details of this engineering design problem can be found in [12].

The comparison of the results is shown in Table 6. The comparison of the best solutions is shown in Table 7. From Table 6, we can see that the "best" result obtained by BSCGA is the second smallest. The "best" result obtained by NM-PSO is the smallest, but its decision vector corresponding to the "best" result is not feasible. The constraints $g_1(X)$ and $g_2(X)$ are violated, and the violation value 0.001 is the biggest in Table 7. Although the best solutions obtained by DS-GA and CDE satisfy all the constraints, their objective values are not very small. HPSO, PSOLOVER and ICEM get the same

TABLE 6. Comparison of the results for tension/compression spring design problem

	best	mean	worst	st.dev.	NFE
BSCGA	0.0126640	0.0126819	0.0127213	2.43E–05	80,000
DS-GA	0.0126810	0.0127420	0.0129730	5.90E–05	80,000
HPSO	0.0126652	0.0127072	0.0127190	1.58E–05	81,000
CDE	0.0126702	0.0127030	0.0127900	2.70E–05	204,800
NM-PSO	0.0126302	0.0126314	0.0126330	8.74E–07	80,000
PSOLOVER	0.0126652	0.0126652	0.0126652	2.46E–09	253
ICEM	0.0126652	0.0126653	0.0126653	3.67E–08	80,000

TABLE 7. Comparison of the best solutions for tension/compression spring design problem

	$f(X)$(best)	x_1	x_2	x_3
BSCGA	0.0126640	0.051704	0.357070	11.267666
DS-GA	0.0126810	0.051989	0.363965	10.890522
HPSO	0.0126652	0.051706	0.357126	11.265083
CDE	0.0126702	0.051609	0.354714	11.410831
NM-PSO	0.0126302	0.051620	0.355498	11.333272
PSOLOVER	0.0126652	0.051686	0.356650	11.292950
ICEM	0.0126652	0.051689	0.356718	11.288960
	$g_1(X)$	$g_2(X)$	$g_3(X)$	$g_4(X)$
BSCGA	9.63E–05	6.23E–06	–4.055	–0.727
DS-GA	–1.30E–05	–2.10E–02	–4.061	–0.723
HPSO	–3.07E–06	1.39E–06	–4.055	–0.727
CDE	–3.86E–05	–1.83E–04	–4.048	–0.729
NM-PSO	1.00E–03	1.00E–03	–4.061	–0.729
PSOLOVER	–2.00E–05	1.33E–05	–4.054	–0.728
ICEM	–6.41E–06	3.90E–06	–4.054	–0.728

minimum objective value 0.0126652 at the cost of a minor violation of constraint $g_2(X)$. Our algorithm BSCGA achieves a balance between objective value and constraint violation and gets a better "best" result than ICEM with acceptable violations of constraints $g_1(X)$ and $g_2(X)$.

5.3. **The welded beam design problem.** This problem aims to minimize the cost subject to seven inequality constraints with four variables. The details of this engineering design problem can be found in [12].

The comparison of the results is shown in Table 8. BSCGA gets the smallest "best" result which is also obtained by NM-PSO and PSOLOVER. Compared with NM-PSO, BSCGA obtains better "mean", "worst" results and a smaller standard deviation.

TABLE 8. Comparison of the results for welded beam design problem

	best	mean	worst	st.dev.	NFE
BSCGA	1.724717	1.724991	1.731462	1.07E–03	80,000
DS-GA	1.728226	1.792654	1.993408	7.47E–02	80,000
HPSO	1.724852	1.749040	1.814295	4.01E–02	81,000
CDE	1.733461	1.768158	1.824105	2.22E–02	204,800
NM-PSO	1.724717	1.726373	1.733393	3.50E–03	80,000
PSOLOVER	1.724717	1.724717	1.724717	1.62E–11	297
ICEM	1.724852	1.724852	1.724852	8.88E–12	80,000

TABLE 9. Comparison of the best solutions for welded beam design problem

	$f(X)$(best)	$g_1(X)$	$g_2(X)$	$g_3(X)$
BSCGA	1.724717	–8.62E–04	–1.04E+00	1.12E–04
DS-GA	1.728226	–7.41E–02	–2.66E–01	–4.95E–04
HPSO	1.724852	–2.54E–02	–5.31E–02	0.00E+00
CDE	1.733461	–4.46E+01	–4.47E+01	–3.04E–03
NM-PSO	1.724717	–2.53E–02	–5.31E–02	1.00E–04
PSOLOVER	1.724717	–2.53E–02	–5.31E–02	1.00E–04
ICEM	1.724852	–2.54E–02	–5.31E–02	0.00E+00
	$g_4(X)$	$g_5(X)$	$g_6(X)$	$g_7(X)$
BSCGA	–3.4332	–8.08E–02	–0.2355	–1.87E–02
DS-GA	–3.4300	–8.10E–02	–0.2355	–5.87E+01
HPSO	–3.4330	–8.07E–02	–0.2355	–3.16E–02
CDE	–3.4237	–7.81E–02	–0.2356	–3.80E+01
NM-PSO	–3.4332	–8.08E–02	–0.2355	–3.16E–02
PSOLOVER	–3.4332	–8.08E–02	–0.2355	–3.16E–02
ICEM	–3.4330	–8.07E–02	–0.2355	–3.16E–02

The comparison of the best solutions is shown in Table 9. The optimum decision vector obtained by BSCGA is $X = (0.205841, 3.468028, 9.036795, 0.205729)$. Its objective function value is 1.724717 and the violation value of constraint $g_3(X)$, 1.12E–04, is almost the same as those of NM-PSO and PSOLOVER.

6. **Conclusion.** In this paper, a new hybrid method (BSCGA) is proposed to solve the constrained optimization problem. The algorithm mainly includes three processes: boundary simulation, population initialization and genetic operation.

Firstly, the boundary simulation method based on the backward binary search technique is presented to simulate the boundary of the feasible region. This method can perform well not only for simply connected feasible region but also for multiple connected and disconnected feasible region.

Secondly, chaotic initialization method is proposed to generate the initial population from the simulated boundary of the feasible region. This method has better performance on maintaining the diversity of the initial population.

Finally, BSCGA uses three genetic operators to search the optimal solution. And a simple binary repair method is presented to repair infeasible individuals.

In this paper, six benchmark functions and three engineering design problems are solved by BSCGA. The results are compared with those of other algorithms. And the comparisons show the competitive advantage of our algorithm.

Further work should be aimed at extending the BSCGA to solve the constrained multiobjective optimization problem.

Acknowledgment. This work is partially supported by the Natural Science Foundation of China (No. 61203283, No. 61203082) and Fundamental Research Funds for the Central Universities (No. 3132014036, No. 3132014324). The authors also gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the presentation.

REFERENCES

[1] J. A. Joines and C. R. Houck, On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with gas, *Proc. of the 1st IEEE Conf. on Evolutionary Computation*, Orlando, FL, pp.579-584, 1994.

[2] R. Farmani, J. A. Wright, D. A. Savic and G. A. Walters, Self-adaptive fitness formulation for evolutionary constrained optimization of water systems, *Journal of Computing in Civil Engineering*, vol.19, no.2, pp.212-216, 2005.

[3] F. Z. 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.

[4] D. Orvosh and L. Davis, Using a genetic algorithm to optimize problems with feasibility constraints, *Proc. of the 1st IEEE Conf. on Evolutionary Computation*, Orlando, FL, pp.548-553, 1994.

[5] P. Chootinan and A. Chen, Constraint handling in genetic algorithms using a gradient-based repair method, *Computers & Operations Research*, vol.33, no.8, pp.2263-2281, 2006.

[6] T. Takahama and S. Sakai, Constrained optimization by the ε constrained differential evolution with gradient-based mutation and feasible elites, *IEEE Congress on Evolutionary Computation*, Vancouver, British Columbia, pp.1-8, 2006.

[7] S. Salcedo-Sanz, A survey of repair methods used as constraint handling techniques in evolutionary algorithms, *Computer Science Review*, vol.3, no.3, pp.175-192, 2009.

[8] Y. Guo, X, Cao and J. Zhang, Constraint handling based multiobjective evolutionary algorithm for aircraft landing scheduling, *International Journal of Innovative Computing, Information and Control*, vol.5, no.8, pp.2229-2238, 2009.

[9] Y. G. Woldesenbet, G. G. Yen and B. G. Tessema, Constraint handling in multiobjective evolutionary optimization, *IEEE Trans. Evolutionary Computation*, vol.13, no.3, pp.514-525, 2009.

[10] T. Ray and T. Kang, An evolutionary algorithm with a multilevel pairing strategy for single and multiobjective optimization, *Foundations of Computing and Decision Sciences*, vol.26, no.1, pp.75-98, 2001.

[11] C. A. C. 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] G. Venter and R. T. Haftka, Constrained particle swarm optimization using a bi-objective formulation, *Structural and Multidisciplinary Optimization*, vol.40, no.1-6, pp.65-76, 2010.

[13] R. Perzina and J. Ramik, Self-learning genetic algorithm for a timetabling problem with fuzzy constraints, *International Journal of Innovative Computing, Information and Control*, vol.9, no.11, pp.4565-4582, 2013.

[14] A. Ghodrati and S. Lotfi, A hybrid CS/GA algorithm for global optimization, *Proc. of the International Conf. on Soft Computing for Problem Solving*, Roorkee, India, pp.397-404, 2011.

[15] S. Oh, Y. Jin and M. Jeon, Approximate models for constraint functions in evolutionary constrained optimization, *International Journal of Innovative Computing, Information and Control*, vol.7, no.11, pp.6585-6603, 2011.

[16] X. Li and G. Du, Inequality constraint handling in genetic algorithms using a boundary simulation method, *Computers & Operations Research*, vol.39, no.3, pp.521-540, 2012.

[17] T. P. Runarsson and X. Yao, Stochastic ranking for constrained evolutionary optimization, *IEEE Trans. Evolutionary Computation*, vol.4, no.3, pp.284-294, 2000.

[18] Q. He and L. Wang, A hybrid particle swarm optimization with a feasibility-based rule for constrained optimization, *Applied Mathematics and Computation*, vol.186, no.2, pp.1407-1422, 2007.

[19] E. Zahara and Y. T. Kao, Hybrid Nelder-Mead simplex search and particle swarm optimization for constrained engineering design problems, *Expert Systems with Applications*, vol.36, no.2, pp.3880-3886, 2009.

[20] A. H. Kayhan, H. Ceylan, M. T. Ayvaz and G. Gurarslan, PSOLVER: A new hybrid particle swarm optimization algorithm for solving continuous optimization problems, *Expert Systems with Applications*, vol.37, no.10, pp.6798-6808, 2010.

[21] C. Zhang, X. Li, L. Gao and Q. Wu, An improved electromagnetism-like mechanism algorithm for constrained optimization, *Expert Systems with Applications*, vol.40, no.14, pp.5621-5634, 2013.