Available online at www.sciencedirect.com
ScienceDirect
Engineering Applications of Artificial Intelligence 20 (2007) 89–99
Engineering Applications of ARTIFICIAL INTELLIGENCE
www.elsevier.com/locate/engappai

An effective co-evolutionary particle swarm optimization for constrained engineering design problems

Qie He, Ling Wang*

Department of Automation, Tsinghua University, Beijing 100084, P.R. China

Received 13 June 2005; received in revised form 20 February 2006; accepted 29 March 2006
Available online 19 May 2006

Abstract

Many engineering design problems can be formulated as constrained optimization problems. So far, penalty function methods have been the most popular methods for constrained optimization due to their simplicity and easy implementation. However, it is often not easy to set suitable penalty factors or to design adaptive mechanism. By employing the notion of co-evolution to adapt penalty factors, this paper proposes a co-evolutionary particle swarm optimization approach (CPSO) for constrained optimization problems, where PSO is applied with two kinds of swarms for evolutionary exploration and exploitation in spaces of both solutions and penalty factors. The proposed CPSO is population based and easy to implement in parallel. Especially, penalty factors also evolve using PSO in a self-tuning way. Simulation results based on well-known constrained engineering design problems demonstrate the effectiveness, efficiency and robustness on initial populations of the proposed method. Moreover, the CPSO obtains some solutions better than those previously reported in the literature.

Keywords: Particle swarm optimization; Co-evolution; Penalty function

1. Introduction

Generally, a constrained optimization problem can be described as follows:

$$\text{find } \boldsymbol{x} \text{ to minimize } f(\boldsymbol{x}) \tag{1}$$

$$\text{Subject to}: g_i(\boldsymbol{x}) \leqslant 0, \quad i = 1, 2, \ldots, n, \tag{2}$$

$$h_j(\boldsymbol{x}) = 0, \quad j = 1, 2, \ldots, p, \tag{3}$$

where $\boldsymbol{x} = [x_1, x_2, \ldots, x_d]^{\mathrm{T}}$ denotes the decision solution vector, n is the number of inequality constraints and p is the number of equality constraints. In a common practice, equality constraint $h_j(\boldsymbol{x}) = 0$ can be replaced by a set of inequality constraints $h_j(\boldsymbol{x}) \geqslant \delta$ and $h_j(\boldsymbol{x}) \geqslant -\delta$ (δ is a small tolerant amount). Thus, all constraints can be transformed to $N = n + 2p$ inequality constraints.

Many engineering design problems can be formulated as constrained optimization problems. The presence of constraints may significantly affect the optimization performances of any optimization algorithms for unconstrained problems. With the increase of the research and applications based on evolutionary computation techniques (Wang, 2001), constraint handling used in evolutionary computation techniques has been a hot topic in both academic and engineering fields (Coello 2002; Michalewicz 1995). So far lots of constraint-handling techniques have been proposed for evolutionary algorithms (EAs).

For most constraint-handling techniques, both infeasible and feasible solutions could be generated at the search stage, and constraints are dealt with when evaluating solutions. The violation of constraints for each solution is considered separately, and the relationship between infeasible solutions and feasible regions is exploited to guide search. The difficulty lies in that there is no certain metric criterion to measure this relationship. Michalewicz (1995) proposed at least three choices: count the number of violations for a given solution; consider the amount of infeasibility in terms of constraints violation; and compute the effort of 'repairing' the individual. As we all know, the

*Corresponding author. Tel.: +86 10 62783125; fax: +86 10 62786911.
E-mail address: wangling@mail.tsinghua.edu.cn (L. Wang).

0952-1976/$ - see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.engappai.2006.03.003

penalty function method has been the most popular constraint-handling technique due to its simple principle and easy implementation. Constraints are incorporated into the objective function so as to transform constrained problems into unconstrained ones. The penalty factors are used to balance the search for the global optimum and satisfaction of the constraints. However, it is often not easy to set suitable penalty factors. Coello (2000, 2002) proposed a self-adaptive penalty approach based on a genetic algorithm (GA) by using a co-evolution model to adapt the penalty factors. Hamida and Schoenauer (2002) proposed an adaptive segregational constraint-handling technique with EA, which adapts penalty factors with the percentage of feasible solutions in the population at every generation and keeps infeasible solutions in the population to maintain diversity. Nevertheless, this approach introduces several extra parameters that need tuning empirically.

Another kind of constraint-handling techniques is inspired by multi-objective optimization techniques, in which objectives and constraints are considered separately. Runarsson and Yao (2000) proposed a constraint-handling technique (Stochastic Ranking, SR) from the viewpoint of balancing dominance between the objective and penalty functions, which does not need a penalty factor and focuses on the rank of the individuals directly using a bubble-sort like algorithm. Coello and Montes (2002) proposed a dominance-based selection scheme to handle constraints in a GA, motivated by the earlier constraint-handling technique known as NPGA (Horn et al., 1994). Recently, Montes and Coello (2005) applied a similar feasibility-based rule to propose a multimembered evolution strategy for constrained optimization problems.

Besides, some other constraint-handling methods are problem specific. In these methods, special operators are designed for generating solutions that do not violate constraints so as to keep the search always in feasible regions. For example, Koziel and Michalewicz (1999) proposed a homomorphous mapping (HM) between a high-dimensional cube and a feasible search space to transform the original problem to an unconstrained one. However, due to the complexity of constraints (e.g., linear, nonlinear, high dimensional and highly constrained), the design of special operators is problem-dependent and short of generality. Moreover, such a technique needs to find an initial set of feasible solutions, which in itself is also very hard for many constrained problems.

During the past decade, another novel evolutionary computation technique, particle swarm optimization (PSO), has been proposed and it has attracted much attention (Kennedy and Eberhart 1995; Kennedy et al., 2001). The development of PSO was based on observations of the social behavior of animals such as bird flocking, fish schooling and swarm theory. PSO is initialized with a population of random solutions. Each individual is assigned a random velocity according to both its own and its companions' flying experiences, and the individuals called particles are then flown through hyperspace. Compared with GAs, PSO has some attractive characteristics. It has memory, so knowledge of good solutions is retained by all particles; whereas in GAs, previous knowledge of the problem is destroyed once the population changes. In PSO, there is a mechanism of constructive cooperation and information sharing between particles. Due to the simple concept, easy implementation and quick convergence, PSO has gained much attention and been successfully applied in a variety of fields mainly for unconstrained continuous optimization problems (Kennedy et al., 2001). So far, as for the constrained optimization problems, relatively less work based on PSO can be found than those based on other EAs. Parsopoulos and Vrahatis (2002) proposed a non-stationary multi-stage assignment penalty function method to transform the constrained problem to the unconstrained problem. Simulation results showed that PSO outperformed other EAs, but the design of the multi-stage assignment penalty function is too complex. In the work of Hu and Eberhart (2002), the initial swarm contains only feasible solutions and a strategy to preserve feasibility is employed. Motivated by multi-objective optimization techniques, Ray and Liew (2001) proposed a swarm algorithm with a multilevel information sharing strategy to deal with constraints. In their work, a better performer list (BPL) is generated by a multilevel Pareto ranking scheme treating every constraint as an objective, while the particle which is not in the BPL gradually congregates round its closest neighbor in the BPL.

In this paper, a co-evolutionary particle swarm optimization approach (CPSO) is proposed for constrained optimization problems by applying PSO and employing the notion of co-evolution. In the CPSO, two kinds of swarms evolve interactively using PSO; one kind of multiple swarms is used for searching good solutions and another kind of a single swarm is used for evolving suitable penalty factors. In particular, the notion of co-evolution is employed to provide a framework to deal with both decision solutions and constraints, and PSO is applied for evolutionary exploration and exploitation in spaces of both solutions and penalty factors. The proposed CPSO is population based and easy to implement in parallel. Especially, penalty factors also evolve using PSO in a self-tuning way. Simulation results and comparisons based on three famous constrained engineering design problems demonstrate the effectiveness and efficiency of the proposed CPSO as well as its robustness on initial populations. Moreover, the CPSO obtains some solutions better than those previously reported in the literature.

The rest of this paper is organized as follows: Section 2 provides some basics for particle swarm optimization. In Section 3 the CPSO is proposed and explained in detail. Simulation results based on some engineering design problems and comparisons with previously reported resulted are presented in Section 4, and the discussion is provided in Section 5. Finally, we end the paper with some conclusions and future work in Section 6.

2. Basics of PSO

PSO is an evolutionary computation technique with the mechanism of individual improvement, population cooperation and competition, which is based on the simulation of simplified social models, such as bird flocking, fish schooling and the swarming theory (Kennedy and Eberhart, 1995). The theoretical framework of PSO is very simple, and PSO is easy to be coded and implemented, and it is computationally inexpensive in terms of memory requirements and CPU times (Kennedy et al., 2001). Nowadays PSO has gained much attention and been successfully applied in various fields, especially for unconstrained continuous optimization problems (Kennedy et al., 2001).

In PSO, it starts with the random initialization of a population (swarm) of individuals (particles) in the search space and works on the social behavior of the particles in the swarm. Therefore, it finds the global best solution by simply adjusting the trajectory of each individual towards its own best location and towards the best particle of the swarm at each time step (generation). However, the trajectory of each individual in the search space is adjusted by dynamically altering the velocity of each particle, according to its own flying experience and the flying experience of the other particles in the search space.

The position and the velocity of the ith particle in the d-dimensional search space can be represented as $X_i = [x_{i,1}, x_{i,2}, ..., x_{i,d}]^{\mathrm{T}}$ and $V_i = [v_{i,1}, v_{i,2}, ..., v_{i,d}]^{\mathrm{T}}$, respectively. Each particle has its own best position (pbest) $P_i = [p_{i,1}, p_{i,2}, ..., p_{i,d}]^{\mathrm{T}}$ corresponding to the personal best objective value obtained so far at time t. The global best particle (gbest) is denoted by P_{g}, which represents the best particle found so far at time t in the entire swarm. The new velocity of each particle is calculated as follows:

$$v_{i,j}(t+1) = wv_{i,j}(t) + c_1 r_1(p_{i,j} - x_{i,j}(t)) + c_2 r_2(p_{\mathrm{g},j} - x_{i,j}(t)), \quad j = 1, 2, ..., d \tag{4}$$

where c_1 and c_2 are constants called acceleration coefficients, w is called the inertia factor, r_1 and r_2 are two independent random numbers uniformly distributed in the range of [0, 1].

Thus, the position of each particle is updated in each generation according to the following equation:

$$x_{i,j}(t+1) = x_{i,j}(t) + v_{i,j}(t+1), \quad j = 1, 2, ..., d. \tag{5}$$

In the standard PSO, Eq. (4) is used to calculate the new velocity according to its previous velocity and to the distance of its current position from both its own best historical position and its neighbors' best position. Generally, the value of each component in V_i can be clamped to the range $[V_{i,\min}, V_{i,\max}]$ to control excessive roaming of particles outside the search space $[X_{i,\min}, X_{i,\max}]$. Then the particle flies toward a new position according to Eq. (5). The process is repeated until a user-defined stopping criterion is reached. We refer to Kennedy et al. (2001) for more detail.

The procedure of standard PSO is summarized as follows.

Step 1: Initialize a population of particles with random positions and velocities, where each particle contains d variables.

Step 2: Evaluate the objective values of all particles, set pbest of each particle and its objective value equal to its current position and objective value, and set gbest and its objective value equal to the position and objective value of the best initial particle.

Step 3: Update the velocity and position of every particle according to Eqs. (4) and (5).

Step 4: Evaluate the objective values of all particles.

Step 5: For each particle, compare its current objective value with the objective value of its pbest. If current value is better, then update pbest and its objective value with the current position and objective value.

Step 6: Determine the best particle of current whole population with the best objective value. If the objective value is better than the objective value of gbest, then update gbest and its objective value with the position and objective value of the current best particle.

Step 7: If a stopping criterion is met, then output gbest and its objective value; otherwise go back to Step (3).

3. Co-evolutionary PSO

3.1. Mechanism of co-evolution

Due to the simplicity of principle and easiness to implement, the penalty function method is the most popular technique to handle constraints. With respect to the main difficulty of setting appropriate penalty factors, Michalewicz and Attia (1994) indicated that a self-adaptive scheme is a promising direction. In the previous work by Coello (2000), a notion of co-evolution was proposed and incorporated into a GA to solve constrained optimization problems. In this paper, we will make some modifications on co-evolution and incorporate it into PSO for constrained optimization problems.

The principle of co-evolution model in CPSO is shown in Fig. 1. In our CPSO, two kinds of swarms are used. In particular, one kind of a single swarm (denoted by Swarm_2) with size M_2 is used adapt suitable penalty factors, another kind of multiple swarms (denoted by $\mathrm{Swarm}_{1,1}$, $\mathrm{Swarm}_{1,2}$, …, $\mathrm{Swarm}_{1,M2}$) each with size M_1 are used in parallel to search good decision solutions. Each particle B_j in Swarm_2 represents a set of penalty factors for particles in $\mathrm{Swarm}_{1,j}$, where each particle represents a decision solution.

In every generation of co-evolution process, every $\mathrm{Swarm}_{1,j}$ will evolve by using PSO for a certain number of generations (G_1) with particle B_j in Swarm_2 as penalty factors for solution evaluation to get a new $\mathrm{Swarm}_{1,j}$. Then the fitness of each particle B_j in Swarm_2 will be determined. After all particles in Swarm_2 are evaluated, Swarm_2 will

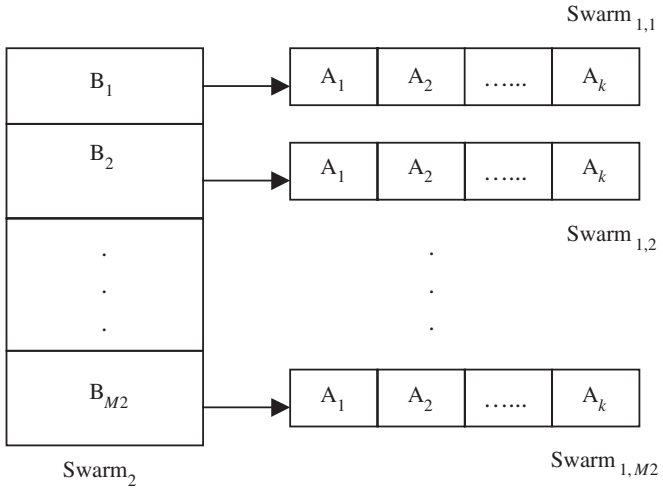

Fig. 1. Graphical illustration for the notion of co-evolution.

also evolve by using PSO with one generation to get a new Swarm_2 with adjusted penalty factors. The above co-evolution process will be repeated until a pre-defined stopping criterion is satisfied (e.g., a maximum number of co-evolution generations G_2 is reached).

In brief, two kinds of swarms evolve interactively, where $\text{Swarm}_{1,j}$ is used to evolve decision solutions while Swarm_2 is used to adapt penalty factors for solution evaluation. Due to the co-evolution, not only decision solutions are explored evolutionary, but also penalty factors are adjusted in a self-tuning way to avoid the difficulty of setting suitable factors by trial and error.

3.2. Evaluation function for $Swarm_{1,j}$

For constrained optimization problems, we design the penalty function following the guidance suggested by Richardson et al. (1989), i.e., not only how many constraints are violated but also the amounts in which such constraints are violated. In particular, the ith particle in $\text{Swarm}_{1,j}$ in CPSO is evaluated by using the following formula:

$$F_i(\boldsymbol{x}) = f_i(\boldsymbol{x}) + \text{sum\_viol} \times w_1 + \text{num\_viol} \times w_2, \quad (6)$$

where $f_i(\boldsymbol{x})$ is the objective value of the ith particle, sum_viol denotes the sum of all the amounts by which the constraints are violated, num_viol denotes the number of constraints violation, w_1 and w_2 are penalty factors corresponding to the particle B_j in Swarm_2.

The value of sum_viol is calculated as follows:

$$\text{sum\_viol} = \sum_{i=1}^{N} g_i(\boldsymbol{x}), \quad \forall g_i(\boldsymbol{x}) > 0 \quad (7)$$

where N is the number of inequality constraints (here it is assumed that all equality constraints have been transformed to inequality constraints).

3.3. Evaluation function for $Swarm_2$

Each particle in Swarm_2 represents a set of factors (w_1 and w_2). After $\text{Swarm}_{1,j}$ evolves for a certain number of generations (G_1), the jth particle B_j in Swarm_2 is evaluated as follows.

(1) If there is at least one feasible solution in $\text{Swarm}_{1,j}$, then particle B_j is evaluated using the following formula and is called a valid particle:

$$P(B_j) = \frac{\sum f_{\text{feasible}}}{\text{num\_feasible}} - \text{num\_feasible}, \quad (8)$$

where $\sum f_{\text{feasible}}$ denotes the sum of objective function values of feasible solutions in $\text{Swarm}_{1,j}$, and num_feasible is the number of feasible solutions in $\text{Swarm}_{1,j}$.

The reason for only considering feasible solutions is to bias the $\text{Swarm}_{1,j}$ towards feasible regions. Moreover, the subtraction of num_feasible in Eq. (8) is to avoid $\text{Swarm}_{1,j}$ stagnating at certain regions in which only very few particles will have good objective values or even be feasible. Consequently, $\text{Swarm}_{1,j}$ will be encouraged to move towards regions including a lot of feasible solutions with good objective values. In addition, num_feasible also acts as a scaling factor when used to divide $\sum f_{\text{feasible}}$.

(2) If there is no feasible solution in $\text{Swarm}_{1,j}$ (it can be considered that the penalty is too low), then particle B_j in Swarm_2 is evaluated as follows and is called an invalid particle.

$$P(B_j) = \max(P_{\text{valid}}) + \frac{\sum \text{sum\_viol}}{\sum \text{num\_viol}} - \sum \text{num\_viol}, \quad (9)$$

where $\max(P_{\text{valid}})$ denotes the maximum fitness value of all valid particles in Swarm_2, $\sum$sum_viol denotes the sum of constraints violation for all particles in $\text{Swarm}_{1,j}$, and $\sum$num_viol counts the total number of constraints violation for all particles in $\text{Swarm}_{1,j}$.

Obviously, by using Eq. (9), the particle in Swarm_2 that results in a smaller amount of constraints violation of $\text{Swarm}_{1,j}$ is considered better. Consequently, the search may bias $\text{Swarm}_{1,j}$ to the region where the sum of constraints violation is small (i.e. the boundary of the feasible region). Moreover, the addition of item $\max(P_{\text{valid}})$ is to assure that the valid particle is always better than the invalid one to guide the search to the feasible region. In addition, $\sum$num_viol acts as a scaling factor.

3.4. Evolution of $Swarm_{1,j}$ and $Swarm_2$

Particles in both the two kinds of swarms will evolve by using PSO procedure described in Section 2. In particular, the particle in Swarm_2 encodes a set of penalty factors (w_1 and w_2), while the particle in $\text{Swarm}_{1,j}$ encodes a set of decision variables. Both kinds of particles will apply Eqs. (4) and (5) to adjust their positions so as to obtain good decision solution and suitable penalty factors. Due to the merits of PSO, such a process can be implemented easily and is proved effective by later simulation results.

3.5. The framework of CPSO

After explaining the main elements of co-evolutionary PSO, the framework of CPSO is clearly illustrated in Fig. 2. The features of CPSO can be summarized as follows. (a) Two kinds of swarms evolve by using PSO interactively, where one is for decision solutions and the other is for penalty factors. (b) Penalty factors are adjusted by using a self-tuning approach. (c) The CPSO is population based and easy to implement in parallel.

4. Simulation results

In this section, we will carry out numerical simulation based on some well-known constrained engineering design problems to investigate the performances of the proposed CPSO. The selected problems have been well studied before as benchmarks by various approaches. We will also compare our results with some good results previously reported by EA-based methods and other traditional mathematical programming methods.

For each testing problem, the parameters of the CPSO are set as follows: $M_1 = 50$, $G_1 = 25$, $M_2 = 20$, $G_2 = 8$, $c_1 = c_2 = 2.0$, w in PSO linearly decreases from 0.9 to 0.4. The maximum and minimum positions for particles in $\text{Swarm}_{1,j}$ ($X_{1,\max}$ and $X_{1,\min}$) depend on the variable region given by the problems. The maximum and minimum positions of particles in Swarm_2 are set as $w_{1,\max} = w_{2,\max} = 1000$ and $w_{1,\min} = w_{2,\min} = 0$ for the first two problems and as $w_{1,\max} = w_{2,\max} = 10000$ and $w_{1,\min} = w_{2,\min} = 5000$ for the third problem. Moreover, the maximum and minimum velocities for particles in both the two kinds of swarms are set as $V_{i,\max} = 0.2 \times (X_{i,\max} - X_{i,\min})$ and $V_{i,\min} = -V_{i,\max}$ ($i = 1, 2$).

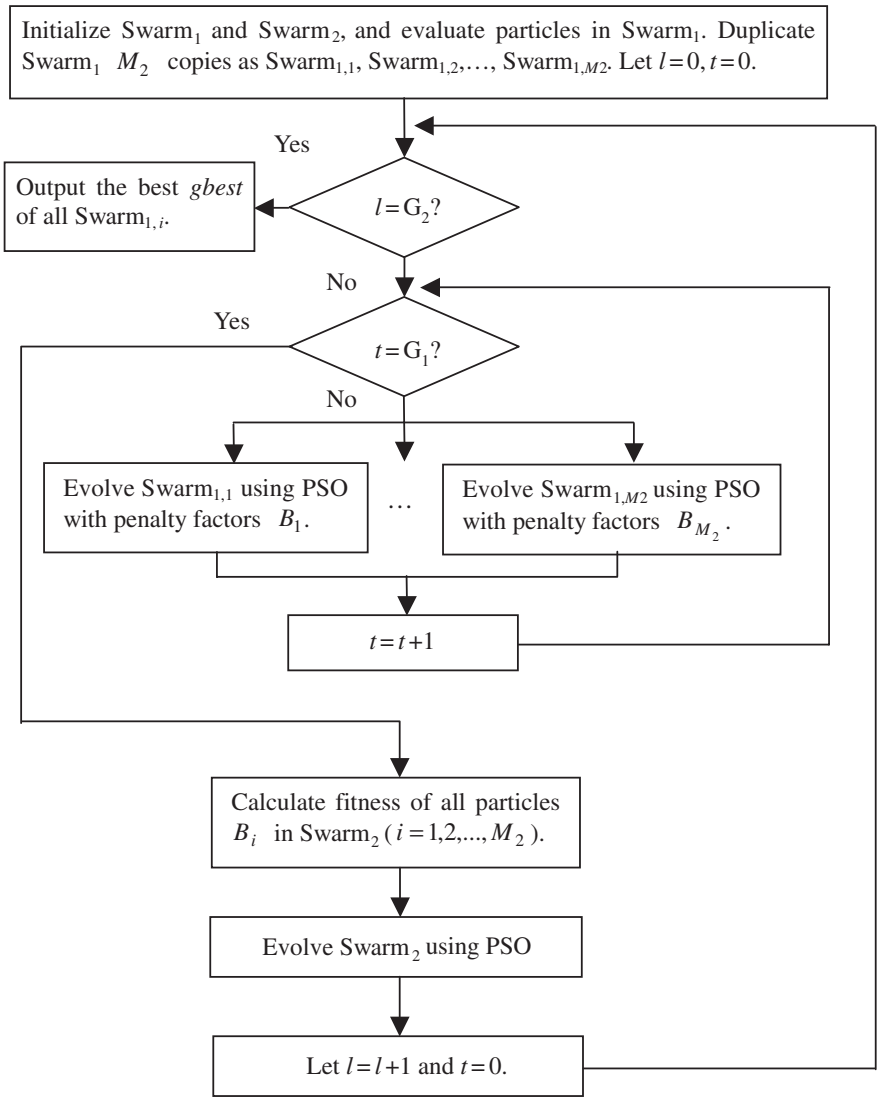

Fig. 2. Flow chart of co-evolutionary PSO.

4.1. Simulation results for welded beam design problem (Example 1)

The welded beam design problem is taken from Rao (1996), in which a welded beam is designed for minimum cost subject to constraints on shear stress (τ), bending stress in the beam (θ), buckling load on the bar (P_c), end deflection of the beam (δ), and side constraints. There are four design variables as shown in Fig. 3, i.e. h (x_1), l (x_2), t (x_3), and b (x_4).

The problem can be mathematically formulated as follows:

$$\text{Minimize} \quad f(\boldsymbol{x}) = 1.10471x_1^2x_2 + 0.04811x_3x_4(14.0 + x_2). \tag{10}$$

$$\text{subject to:} \quad g_1(\boldsymbol{x}) = \tau(\boldsymbol{x}) - 13000 \leqslant 0, \tag{11}$$

$$g_2(\boldsymbol{x}) = \sigma(\boldsymbol{x}) - 30000 \leqslant 0, \tag{12}$$

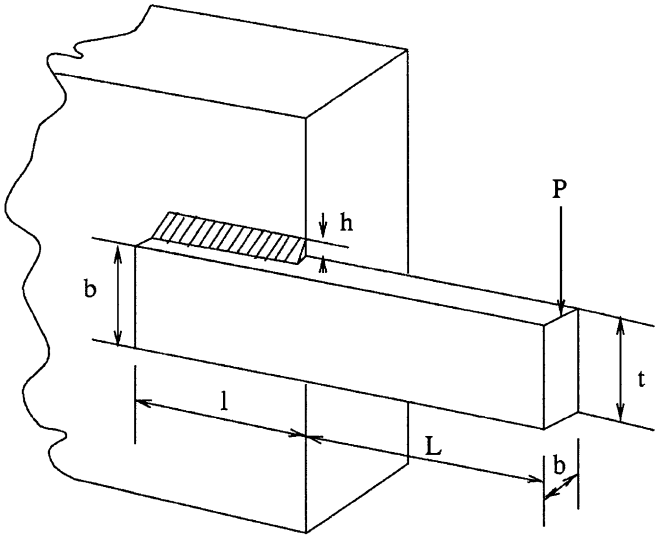

Fig. 3. Welded beam design problem (Example 1).

$$g_3(\boldsymbol{x}) = x_1 - x_4 \leqslant 0, \tag{13}$$

$$g_4(\boldsymbol{x}) = 0.10471x_1^2 + 0.04811x_3x_4(14.0 + x_2) - 5.0 \leqslant 0, \tag{14}$$

$$g_5(\boldsymbol{x}) = 0.125 - x_1 \leqslant 0, \tag{15}$$

$$g_6(\boldsymbol{x}) = \delta(x) - 0.25 \leqslant 0, \tag{16}$$

$$g_7(\boldsymbol{x}) = 6000 - P_c(x) \leqslant 0, \tag{17}$$

where

$$\tau(\boldsymbol{x}) = \sqrt{(\tau')^2 + 2\tau'\tau''\frac{x_2}{2R} + (\tau'')^2}, \tag{18}$$

$$\tau' = \frac{6000}{\sqrt{2}x_1x_2}, \tag{19}$$

$$\tau'' = \frac{MR}{J}, \tag{20}$$

$$M = 6000\left(14 + \frac{x_2}{2}\right), \tag{21}$$

$$R = \sqrt{\frac{x_2^2}{4} + \left(\frac{x_1 + x_3}{2}\right)^2}, \tag{22}$$

$$J = 2\left\{\sqrt{2}x_1x_2\left[\frac{x_2^2}{12} + \left(\frac{x_1 + x_3}{2}\right)^2\right]\right\}, \tag{23}$$

$$\sigma(\boldsymbol{x}) = \frac{504000}{x_4x_3^2}, \tag{24}$$

$$\delta(\boldsymbol{x}) = \frac{2.1952}{x_3^3x_4}, \tag{25}$$

$$P_c(\boldsymbol{x}) = 64746.022(1 - 0.0282346x_3)x_3x_4^3. \tag{26}$$

The approaches applied to this problem include geometric programming (Ragsdell and Phillips, 1976), genetic algorithm with binary representation and traditional penalty function (Deb, 1991), a GA-based co-evolution model (Coello, 2000) and a feasibility-based tournament selection

Table 1
Comparison of the best solution for Example 1 found by different methods

Design variables	CPSO	Ragsdell and Phillips (1976)	Deb (1991)	Coello (2000)	Coello and Montes (2002)
$x_1(h)$	0.202369	0.245500	0.248900	0.208800	0.205986
$x_2(l)$	3.544214	6.196000	6.173000	3.420500	3.471328
$x_3(t)$	9.048210	8.273000	8.178900	8.997500	9.020224
$x_4(b)$	0.205723	0.245500	0.253300	0.210000	0.206480
$g_1(\boldsymbol{x})$	−12.839796	−5743.826517	−5758.603777	−0.337812	−0.074092
$g_2(\boldsymbol{x})$	−1.247467	−4.715097	−255.576901	−353.902604	−0.266227
$g_3(\boldsymbol{x})$	−0.001498	0.000000	−0.004400	−0.001200	−0.000495
$g_4(\boldsymbol{x})$	−3.429347	−3.020289	−2.982866	−3.411865	−3.430043
$g_5(\boldsymbol{x})$	−0.079381	−0.120500	−0.123900	−0.083800	−0.080986
$g_6(\boldsymbol{x})$	−0.235536	−0.234208	−0.234160	−0.235649	−0.235514
$g_7(\boldsymbol{x})$	−11.681355	−3604.275002	−4465.270928	−363.232384	−58.666440
$f(\boldsymbol{x})$	1.728024	2.385937	2.433116	−1.748309	1.728226

Table 2
Statistical results of different methods for Example 1

Method	Best	Mean	Worst	Std Dev
CPSO	1.728024	1.748831	1.782143	0.012926
Ragsdell and Phillips (1976)	2.385937	N/A	N/A	N/A
Deb (1991)	2.433116	N/A	N/A	N/A
Coello (2000)	1.748309	1.771973	1.785835	0.011220
Coello and Montes (2002)	1.728226	1.792654	1.993408	0.074713

scheme inspired by the multi-objective optimization techniques (Coello and Montes, 2002). In this paper, the CPSO is run 30 times independently with the following variable regions: $0.1 \leqslant x_1 \leqslant 2$, $0.1 \leqslant x_2 \leqslant 10$, $0.1 \leqslant x_3 \leqslant 10$, $0.1 \leqslant x_4 \leqslant 2$. The best solutions obtained by the above-mentioned approaches are listed in Table 1, and their statistical simulation results are shown in Table 2.

From Table 1, it can be seen that the best feasible solution found by CPSO is better than the best solutions found by other techniques. From Table 2, it can be seen that the average searching quality of CPSO is also better than those of other methods, and even the worst solution found by CPSO is better than the best solution found by Ragsdell and Phillips (1976) and the best solution found by Deb (1991). In addition, the standard deviation of the results by CPSO in 30 independent runs is very small.

4.2. Simulation results for a tension/compression string design problem (Example 2)

This problem is from Arora (1989) and Belegundu (1982), which needs to minimize the weight (i.e. $f(\boldsymbol{x})$) of a tension/compression spring (as shown in Fig. 4) subject to constraints on minimum deflection, shear stress, surge frequency, limits on outside diameter and on design variables. 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:

$$\text{Minimize} \quad f(\boldsymbol{x}) = (x_3 + 2)x_2x_1^2. \tag{27}$$

$$\text{subject to:} \quad g_1(\boldsymbol{x}) = 1 - \frac{x_2^3x_3}{71785x_1^4} \leqslant 0, \tag{28}$$

$$g_2(\boldsymbol{x}) = \frac{4x_2^2 - x_1x_2}{12566(x_2x_1^3 - x_1^4)} + \frac{1}{5108x_1^2} - 1 \leqslant 0, \tag{29}$$

$$g_3(\boldsymbol{x}) = 1 - \frac{140.45x_1}{x_2^2x_3} \leqslant 0, \tag{30}$$

$$g_4(\boldsymbol{x}) = \frac{x_1 + x_2}{1.5} - 1 \leqslant 0. \tag{31}$$

The approaches applied to this problem include eight different numerical optimization techniques (Belegundu, 1982), a numerical optimization technique called constraint correction at constant cost (Arora, 1989), a GA-based co-evolution model (Coello, 2000) and a feasibility-based tournament selection scheme (Coello and Montes, 2002). In this paper, the CPSO is run 30 times independently with the following variable regions: $0.05 \leqslant x_1 \leqslant 2$, $0.25 \leqslant x_2 \leqslant 1.3$, $2 \leqslant x_3 \leqslant 15$. The best solutions obtained by the above-mentioned approaches are listed in Table 3, and their statistical simulation results are shown in Table 4.

From Table 3, it can be seen that the best feasible solution found by CPSO is better than the best solutions found by other techniques. From Table 4, it can be seen that the average searching quality of CPSO is also better than those of other methods, and even the worst solution found by CPSO is better than the best solutions found by Belegundu (1982) and the best solutions found by Arora (1989). Moreover, the standard deviation of the results by CPSO in 30 independent runs for this problem is also very small.

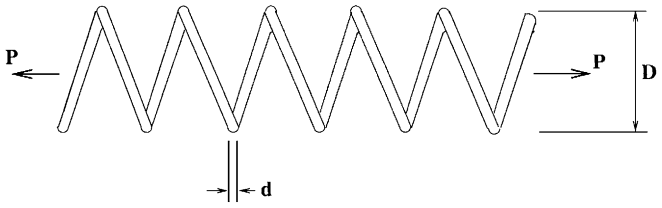

Fig. 4. Tension/compression string design problem (Example 2).

Table 3
Comparison of the best solution for Example 2 by different methods

Design variables	CPSO	Belegundu (1982)	Arora (1989)	Coello (2000)	Coello and Montes (2002)
$x_1(d)$	0.051728	0.050000	0.053396	0.051480	0.051989
$x_2(D)$	0.357644	0.315900	0.399180	0.351661	0.363965
$x_3(P)$	11.244543	14.250000	9.185400	11.632201	10.890522
$g_1(\boldsymbol{x})$	−0.000845	−0.000014	0.000019	−0.002080	−0.000013
$g_2(\boldsymbol{x})$	−1.2600e-05	−0.003782	−0.000018	−0.000110	−0.000021
$g_3(\boldsymbol{x})$	−4.051300	−3.938302	−4.123832	−4.026318	−4.061338
$g_4(\boldsymbol{x})$	−0.727090	−0.756067	−0.698283	−4.026318	−0.722698
$f(\boldsymbol{x})$	0.0126747	0.0128334	0.0127303	0.0127048	0.0126810

Table 4
Statistical results of different methods for Example 2

Method	Best	Mean	Worst	Std Dev
CPSO	0.0126747	0.012730	0.012924	5.198500e-005
Belegundu (1982)	0.0128334	N/A	N/A	N/A
Arora (1989)	0.0127303	N/A	N/A	N/A
Coello (2000)	0.0127048	0.012769	0.012822	3.939000e-005
Coello and Montes (2002)	0.0126810	0.0127420	0.012973	5.900000e-005

4.3. Simulation results for a pressure vessel design problem (Example 3)

In this problem, the objective is to minimize the total cost ($f(\boldsymbol{x})$), including the cost of the material, forming and welding. A cylindrical vessel is capped at both ends by hemispherical heads as shown in Fig. 5. There are four design variables: T_s (x_1, thickness of the shell), T_h (x_2, thickness of the head), R (x_3, inner radius) and L (x_4, length of the cylindrical section of the vessel, not including the head). Among the four variables, T_s and T_h are integer multiples of 0.0625in that are the available thicknesses of rolled steel plates, and R and L are continuous variables.

The problem can be formulated as follows (Kannan and Kramer, 1994):

$$\text{Minimize} \quad f(\boldsymbol{x}) = 0.6224x_1x_3x_4 + 1.7781x_2x_3^2 + 3.1661x_1^2x_4 + 19.84x_1^2x_3. \tag{32}$$

$$\text{subject to:} \quad g_1(\boldsymbol{x}) = -x_1 + 0.0193x_3 \leqslant 0, \tag{33}$$

$$g_2(\boldsymbol{x}) = -x_2 + 0.00954x_3 \leqslant 0, \tag{34}$$

$$g_3(\boldsymbol{x}) = -\pi x_3^2x_4 - \frac{4}{3}\pi x_3^3 + 1296000 \leqslant 0, \tag{35}$$

$$g_4(\boldsymbol{x}) = x_4 - 240 \leqslant 0. \tag{36}$$

The approaches applied to this problem include genetic adaptive search (Deb, 1997), an augmented Lagrangian multiplier approach (Kannan and Kramer, 1994), a branch and bound technique (Sandgren, 1988), a GA-based co-evolution model (Coello, 2000) and a feasibility-based tournament selection scheme (Coello and Montes, 2002). In this paper, the CPSO is run 30 times independently with the following variable regions: $1 \leqslant x_1 \leqslant 99$, $1 \leqslant x_2 \leqslant 99$, $10 \leqslant x_3 \leqslant 200$, $10 \leqslant x_4 \leqslant 200$. The best solutions obtained by the above mentioned approaches are listed in Table 5, and their statistical simulation results are shown in Table 6.

From Table 5, it can be seen that the best solution found by CPSO is better than the best solutions found by other techniques but slightly inferior to the result by Coello and Montes (2002). Comparing the results of CPSO with those by Coello and Montes (2002) statistically, it can be seen from Table 6 that the mean result, the worst result and the standard deviation by CPSO for this problem are all smaller than that by Coello and Montes (2002). In addition, from Table 6 it can be seen that the average searching quality of CPSO is also better than those of other methods, and even the worst solution found by CPSO is better than the best solutions found by Kannan and Kramer (1994) and the best solutions found by Sandgren (1988).

Based on the above simulation results and comparisons, it can be concluded that CPSO is of superior searching quality and robustness for constrained engineering design problems. Especially, our PSO-based co-evolution is more effective than GA-based co-evolution by Coello (2000). Moreover, the total number of fitness evaluations is

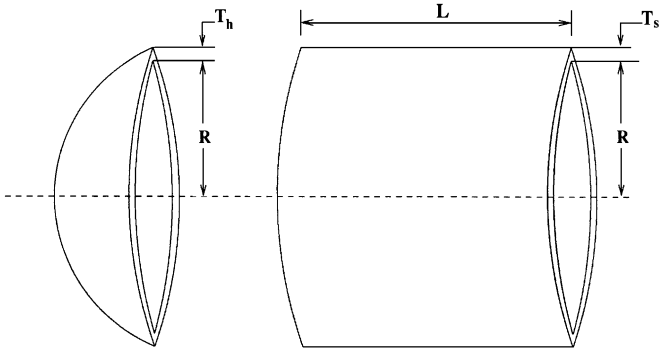

Fig. 5. Center and end section of pressure vessel design problem (Example 3).

Table 6
Statistical results of different methods for Example 3

Method	Best	Mean	Worst	Std Dev
CPSO	6061.0777	6147.1332	6363.8041	86.4545
Sandgren (1988)	8129.1036	N/A	N/A	N/A
Kannan and Kramer (1994)	7198.0428	N/A	N/A	N/A
Deb (1997)	6410.3811	N/A	N/A	N/A
Coello (2000)	6288.7445	6293.8432	6308.1497	7.4133
Coello and Montes (2002)	6059.9463	6177.2533	6469.3220	130.9297

Table 5
Comparison of the best solution for Example 3 found by different methods

Design variables	CPSO	Sandgren (1988)	Kannan and Kramer (1994)	Deb (1997)	Coello (2000)	Coello and Montes (2002)
$x_1(T_s)$	0.812500	1.125000	1.125000	0.937500	0.812500	0.812500
$x_2(T_s)$	0.437500	0.625000	0.625000	0.500000	0.437500	0.437500
$x_3(T_s)$	42.091266	47.700000	58.291000	48.329000	40.323900	42.097398
$x_4(T_s)$	176.746500	117.701000	43.690000	112.679000	200.000000	176.654050
$g_1(\boldsymbol{x})$	−0.000139	−0.204390	0.000016	−0.004750	−0.034324	−0.000020
$g_2(\boldsymbol{x})$	−0.035949	−0.169942	−0.068904	−0.038941	−0.052847	−0.035891
$g_3(\boldsymbol{x})$	−116.382700	54.226012	−21.220104	−3652.876838	−27.105845	−27.886075
$g_4(\boldsymbol{x})$	−63.253500	−122.299000	−196.310000	−127.321000	−40.000000	−63.345953
$f(\boldsymbol{x})$	6061.0777	8129.1036	7198.0428	6410.3811	6288.7445	6059.9463

900 000 in the GA-based co-evolution by Coello (2000), while in our CPSO the number of fitness evaluations is 200 000. So, it can be pointed out that PSO is a better alternative for constrained optimization. As for more discussion about the searching efficiency, it will be presented in Section 5.

5. Discussion

In this section, we will discuss the searching efficiency and the parameter setting of the CPSO.

Although the notion of the co-evolution model is introduced and two kinds of swarms evolve simultaneously in CPSO, it is not necessary to cost a large number of fitness evaluations to reach good solutions. Note that the total number of fitness evaluation is 900 000 in the GA-based co-evolution by Coello (2000), while in our CPSO it is 200 000. We investigate the smallest number of fitness evaluations for hitting the best known solution. The statistical information is listed in Table 7, where it can be seen that CPSO only requires small number of solution evaluation to reach the global optimum. In Fig. 6 a typical evolving process of fitness function for decision solutions when solving Example 1 is illustrated, from which it can be seen that CPSO can converge to the global optimum very quickly and much more solution evaluation is not necessary. Due to the easy implementation and parallel searching structure, the efficiency of CPSO can be improved further by using parallel processors or parallel computational techniques.

Table 7
Statistical smallest number of solution evaluation to hit the global optimum

Problem	Lowest number	Average number	Largest number
Example 1	30 000	34 500	39 000
Example 2	23 000	32 800	40 000
Example 3	24 000	32 500	36 000

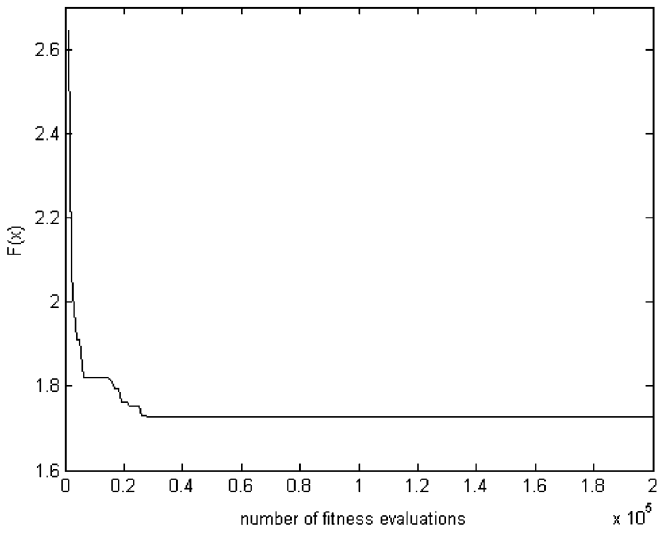

Fig. 6. A typical evolving process of fitness function for Example 1.

During the simulation experiments, it is found that the parameter M_1 (size of $\text{Swarm}_{1,j}$) plays a crucial role in the CPSO. Figs. 7 and 8 illustrate the average objective values that resulted from CPSO (30 independent runs are executed) and the total number of evaluation with different M_1 when solving Example 1. As shown in Fig. 7, if M_1 is too small, the solution region covered at each co-evolution is not enough so that the results are poor. As M_1 increases, the results become better at a cost of more fitness evaluations, but there is a threshold beyond which the results will not be affected in a significant manner. So,

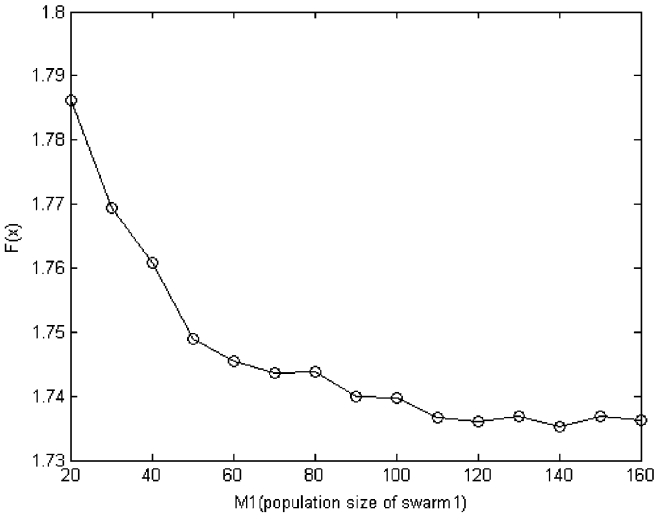

Fig. 7. Average objective values resulted by CPSO with different M_1 for Example 1.

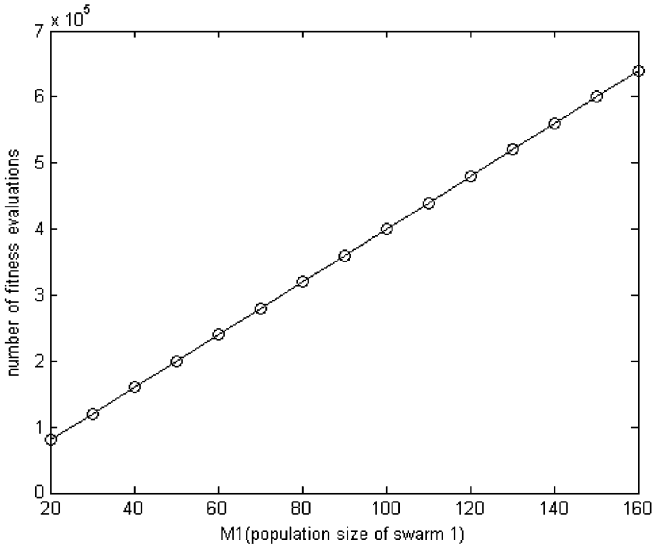

Fig. 8. Total number of fitness evaluations in CPSO with different M_1 for Example 1.

considering both the search quality and computational effort, it is recommended to choose M_1 between 50 and 70.

As for the two parameters used in $Swarm_2$, i.e., M_2 (the population size of $Swarm_2$) and G_2 (the max number of evolution generations for $Swarm_2$), it is found in simulation experiments that the increment of M_2 will not affect the search quality greatly. So, it is not necessary to set M_2 too large. Besides, it is found that the increment of G_2 is also not very beneficial to the search quality but will cause the increment of the number of evaluations, so it is preferred to set G_2 relatively smaller than G_1.

6. Conclusion

This paper has introduced a novel constraint-handling method—co-evolutionary particle swarm optimization. This is the first report to incorporate a co-evolution model into PSO to solve constrained optimization problems. In CPSO, two kinds of swarms evolve with PSO interactively, where one kind of multiple swarms are used for searching good solutions and another kind of a single swarm is used for evolving suitable penalty factors. Simulation results based on some well-known constrained engineering design problems and comparisons with previously reported results demonstrate the effectiveness, efficiency and robustness of the CPSO. Our future work is to incorporate suitable local search methods and diversity mechanisms into CPSO to further enhance and balance the exploration and exploitation abilities so as to achieve better performance. In addition, we will study the parallel implementation of CPSO and the application of CPSO for constrained combinatorial optimization problems.

Acknowledgments

The authors wish to thank the Editor-in-Chief Prof. R. Vingerhoeds and anonymous reviewers for their constructive and valuable comments. And this research is partially supported by National Science Foundation of China (60204008, 60374060 and 60574072) and 973 Program (2002CB312200).

References

Arora, J.S., 1989. Introduction to Optimum Design. McGraw-Hill, New York.

Belegundu, A.D., 1982. A study of mathematical programming methods for structural optimization. Department of Civil and Environmental Engineering, University of Iowa, Iowa City, Iowa.

Coello, C.A.C., 2000. Use of a self-adaptive penalty approach for engineering optimization problems. Computers in Industry 41, 113–127.

Coello, C.A.C., 2002. 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 191 (11/12), 1245–1287.

Coello, C.A.C., Montes, E.M., 2002. Constraint-handling in genetic algorithms through the use of dominance-based tournament selection. Advanced Engineering Informatics 16, 193–203.

Deb, K., 1991. Optimal design of a welded beam via genetic algorithms. AIAA Journal 29 (11), 2013–2015.

Deb, K., 1997. GeneAS: a robust optimal design technique for mechanical component design. In: Dasgupta, D., Michalewicz, Z. (Eds.), Evolutionary Algorithms in Engineering Applications. Springer, Berlin, pp. 497–514.

Hamida, S.B., Schoenauer, M., 2002. ASCHEA: new results using adaptive segregational constraint handling. In: Fogel, D.B. et al. (Eds.), Proceedings of the 2002 Congress on Evolutionary Computation IEEE Service Center, Piscataway, NJ, pp. 884–889.

Horn, J., Nafpliotis, N., Goldberg D.E., 1994. A niched pareto genetic algorithm for multiobjective optimization. In: Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence, IEEE Service Center, Piscataway, NJ, pp. 82-87.

Hu, X., Eberhart, R.C., 2002. Solving constrained nonlinear optimization problems with particle swarm optimization. In: Callaos, N. (Ed.), Proceedings of the Sixth World Multiconference on Systemics. Cybernetics and Informatics, Orlando, Fl, pp. 203–206.

Kannan, B.K., Kramer, S.N., 1994. An augmented Lagrange multiplier based method for mixed integer discrete continuous optimization and its applications to mechanical design. Transactions of the ASME, Journal of Mechanical Design 116, 318–320.

Kennedy, J., Eberhart, R.C., 1995. Particle swarm optimization. In: Proceedings of the 1995 IEEE International Conference on Neural Networks, IEEE Service Center, Piscataway, NJ, pp. 1942–1948.

Kennedy, J., Eberhart, R.C., Shi, Y. (Eds.), 2001. Swarm Intelligence. Morgan Kaufmann, San Francisco.

Koziel, S., Michalewicz, Z., 1999. Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization. Evolutionary Computation 7 (1), 19–44.

Michalewicz, Z., 1995. A survey of constraint handling techniques in evolutionary computation methods. In: McDonnell, J.R. et al. (Eds.), Proceedings of the fourth Annual Conference on Evolutionary Programming, MIT Press, Cambridge, MA, pp. 135–155.

Michalewicz, Z., Attia, N., 1994. Evolutionary optimization of constrained problems. In: Sebald, A.V., Fogel, L.J. (Eds.), Proceedings of the third Annual Conference on Evolutionary Programming. World Scientific, River Edge, NJ, pp. 98–108.

Montes, E.M., Coello, C.A.C., 2005. A simple multimembered evolution strategy to solve constrained optimization problems. IEEE Transactions on Evolutionary computation 9 (1), 1–17.

Parsopoulos, K.E., Vrahatis, M.N., 2002. Particle swarm optimization method for constrained optimization problems. In: Kvasnička, V. et al. (Eds), Proceedings of the second Euro-International Symposium on Computational Intelligence, Košice, Slovakia, pp. 214–220.

Ragsdell, K.M., Phillips, D.T., 1976. Optimal design of a class of welded structures using geometric programming. ASME Journal of Engineering for Industries 98 (3), 1021–1025.

Rao, S.S., 1996. Engineering Optimization. Wiley, New York.

Ray, T., Liew, K.M., 2001. A swarm with an effective information sharing mechanism for unconstrained and constrained single objective optimisation problems. In: Kim, J.H. et al. (Eds.), Proceedings of the 2001 Congress on Evolutionary Computation, IEEE Service Center, Piscataway, NJ, pp. 75–80.

Richardson, J.T., Palmer, M.R., Liepins, G., Hilliard, M., 1989. Some guidelines for genetic algorithms with penalty functions. In: Schaffer, J.D. (Ed.), Proceedings of the Third International Conference on Genetic Algorithms, George Mason University, Morgan Kaufmann, pp. 191–197.

Runarsson, T.P., Yao, X., 2000. Stochastic ranking for constrained evolutionary optimization. IEEE Transactions on Evolutionary Computation 4 (3), 284–294.

Sandgren, E., 1988. Nonlinear integer and discrete programming in mechanical design. In: Proceedings of the ASME Design Technology Conference, Kissimine, FL, pp. 95–105.

Wang, L., 2001. Intelligent optimization algorithms with application. Tsinghua University and Springer, Beijing.

Qie He is currently a MS student in Department of Automation at Tsinghua University. His current research interests include particle swarm optimization, constrained optimization.

Ling Wang completed Ph.D. in Department of Automation from Tsinghua University in 1999 and now is an Associate Professor. His current research interests include optimization theory and algorithms, production scheduling. Dr. Wang has published two books and over 100 refereed academic papers, and he gained Outstanding Paper Award in ICMLC'02 and National Natural Science Award (1st Place Prize) nominated by Ministry of Education of China in 2003.