Struct Multidisc Optim (2008) 37:131–147
DOI 10.1007/s00158-007-0222-3

RESEARCH PAPER

A ranking selection-based particle swarm optimizer for engineering design optimization problems

Jinhua Wang · Zeyong Yin

Received: 13 August 2007 / Revised: 30 October 2007 / Accepted: 2 December 2007 / Published online: 25 January 2008

Abstract Particle swarm optimization (PSO) algorithms have been proposed to solve optimization problems in engineering design, which are usually constrained (possibly highly constrained) and may require the use of mixed variables such as continuous, integer, and discrete variables. In this paper, a new algorithm called the ranking selection-based PSO (RSPSO) is developed. In RSPSO, the objective function and constraints are handled separately. For discrete variables, they are partitioned into ordinary discrete and categorical ones, and the latter is managed and searched directly without the concept of velocity in the standard PSO. In addition, a new ranking selection scheme is incorporated into PSO to elaborately control the search behavior of a swarm in different search phases and on categorical variables. RSPSO is relatively simple and easy to implement. Experiments on five engineering problems and a benchmark function with equality constraints were conducted. The results indicate that RSPSO is an effective and widely applicable optimizer for optimization problems in engineering design in comparison with the state-of-the-art algorithms in the area.

Keywords Particle swarm optimization · Mixed variables · Constrained optimization · Ranking selection · Equality constraints

J. Wang (✉)
School of Mechanical and Electrical Engineering,
Northwestern Polytechnical University,
Xi'an 710072, China
e-mail: wang.jinhua@yahoo.com.cn

Z. Yin · J. Wang
China Aviation Powerplant Research Institute,
Zhuzhou 412002, China

1 Introduction

Most engineering design optimization problems involve constraints, and in some occasions, these constraints cannot be expressed analytically in terms of design variables. In addition, they often contain integer or discrete variables. Without loss of generality, an optimization problem in engineering design can be formulated into a mixed variable-constrained optimization problem (MVCOP) as follows:

$$\text{Minimize } f(X) \tag{1}$$

Subject to:

$$\begin{aligned} &g_i(X) \le 0 \;\; i = 1, 2, ..., m \\ &h_i(X) = 0 \;\; i = 1, 2, ..., n \end{aligned} \tag{2}$$

where $f(X)$ is the objective function, $g_i(X)$, $h_i(X)$, m, and n are the inequality constraints, equality constraints, and their numbers, respectively, and X is the vector consisting of the design variables.

$$\begin{aligned} &X = \begin{pmatrix} X^{\mathrm{C}} \\ X^{\mathrm{I}} \\ X^{\mathrm{D}} \end{pmatrix} \\ &X^{\mathrm{C}} = [x_1, ..., x_{nC}]^{\mathrm{T}} \subseteq \mathrm{R}^{nC} \\ &X^{\mathrm{I}} = [x_{nC+1}, ..., x_{nC+nI}]^{\mathrm{T}} \subseteq \mathrm{Z}^{nI} \\ &X^{\mathrm{D}} = [x_{nC+nI+1}, ..., x_{nC+nI+nD}]^{\mathrm{T}} \\ &x_i^{\mathrm{Cl}} \le x_i \le x_i^{\mathrm{Cu}}, \; i = 1, 2, ..., nC \\ &x_i^{\mathrm{Il}} \le x_i \le x_i^{\mathrm{Iu}}, \; i = nC + 1, ..., nC + nI \\ &x_i \in D_i, \; i = nC + nI + 1, ..., nC + nI + nD \end{aligned} \tag{3}$$

where nC, nI, and nD are the numbers of the continuous, integer, and discrete variables, respectively. x_i^{Cl} and x_i^{Cu} are the lower and upper bounds of the continuous variable x_i.

x_i^{ll} and x_i^{lu} are the lower and upper bounds of the integer variable x_i. D_i is the sequence from which the discrete variable x_i takes on values.

To solve constrained optimization problems (COPs), a number of approaches have been proposed. Among them, evolutionary algorithms (EAs) combined with different constraint-handling techniques have attracted more attention in recent years because of its superior advantages. In EAs, objective or constraint functions are not required to be differentiable, continuous, or even explicit. Moreover, EAs have better global search ability over traditional mathematical programming. The most popular constraint-handling techniques utilized in EAs had been reviewed and classified into five categories by Coello (2002): (1) penalty functions, (2) special representations and operators, (3) repair algorithms, (4) separate objective and constraints, and (5) hybrid methods.

During the past decade, a novel algorithm called particle swarm optimization (PSO) was introduced by Kennedy and Eberhart (1995), which was inspired by the social behavior of animals such as birds flocking and fish schooling. Similar to EAs, PSO is a population-based optimization algorithm and can solve hard optimization problems, whereas it is simpler, easier to implement with only a few parameters to be tuned, and has a quicker convergence rate (Kennedy et al. 2001). Although the original PSO is usually applied to unconstrained continuous optimization problems, it has also been proposed to incorporate constraint-handling techniques or mixed variable-handling methods into PSO for solving constrained or mixed variable optimization problems. However, few investigations on this topic have been reported.

As to constraint handling in PSO, several techniques had been attempted. Parsopoulos and Vrahatis (2002a) used a nonstationary multistage assignment penalty function method to transform a constrained problem into an unconstrained one. In another undertaking by them (Parsopoulos and Vrahatis 2005), penalty factors were used to account for the number of constraints that are violated and the sum of violated constraints. Sedlaczek and Eberhard (2006) combined the standard PSO with an extended nonstationary penalty function approach, called augmented Lagrange multiplier method, for constraint handling. To avoid the difficulty of setting penalty factors, He and Wang (2007b) added a particle swarm to adapt factors automatically. Methods based on preserving feasibility of solutions were also employed by Hu and Eberhart (2002), He et al. (2004), in which each particle keeps tracking only feasible solutions. Pulido and Coello (2004) proposed a simple mechanism to handle constraints with PSO, which was based on the proximity of a particle to the feasible region. Similar to their work, He and Wang (2007a) applied a feasibility-based rule to deal with constraints. Lu and Chen (2006) also introduced a method, which is essentially based on the feasibility-based rule. Ray and Liew (2001) put forward a swarm algorithm with a multilevel information-sharing strategy to handle constraints.

To handle discrete variables in PSO, Venter and Sobieszczanski-Sobieski (2004) treated discrete design variables as continuous ones directly and applied round-off to the discrete components of the final result. Parsopoulos and Vrahatis (2002b) applied PSO to integer programming by simply truncating real values to integers and claimed that the method does not significantly affect the search performance. The same method was used to deal with integer variables by He et al. (2004), who directly optimized the index of the discrete variable instead of the discrete value itself. Kitayama et al. (2006) employed a penalty function approach to handle the discrete design variables, in which the discrete design variables were treated as continuous ones by penalizing at intervals. To solve discrete optimization problems, Kennedy and Eberhart (1997) introduced a discrete binary version of PSO and redefined the meanings of trajectory and velocity in terms of changes in probabilities that a bit will be in one state or the other. Other methods for solving discrete optimization problems with PSO have also been proposed by Clerc (2000), Liao et al. (2005), Jin et al. (2007), and so on.

Because of the parameters of PSO being problem dependent, there is a need to tune them to improve suboptimal solutions when a PSO algorithm is applied to a real-word problem. Tuning the parameters can control the behavior of a swarm, especially the convergence rate and diversity preservation. For a constrained problem, when a particle is located in the infeasible region, the primary objective of its search is to enter the feasible region. In contrast with that, its ultimate objective is to find the optimal solution after it has entered the feasible region. In this sense, it is reasonable to employ different parameter settings for controlling the search of particles in different search phases. However, none of the aforementioned PSO algorithms does so.

From the point of view of performance, the aforementioned PSO algorithms with different constraint or discrete variable-handling techniques have some limitations: (1) using the same parameter settings to guide the search of particles in different search phases; (2) although penalty function method is the most common constraint-handling technique, it has a major drawback that it is difficult for penalty factors to balance objective and penalty functions; (3) as constraint-handling techniques, the methods based on preserving feasibility of solutions require initial feasible solutions. However, it is nondeterministic polynomial-time hard to find feasible solutions of some problems; (4) the applicability of the methods using the feasibility-based rule is often seriously limited in highly constrained problems. The valuable information from the objective function values of infeasible solutions are usually neglected, which usually lead to inferior solutions; (5) treating integer and

discrete variables as continuous ones and then applying round-off to the final results is a popular method used in mixed variable problems (MVPs) but may result in a solution far from the optimum or even infeasible; (6) optimizing the index of a discrete variable instead of the discrete value of the variable directly is another simple method for handling discrete variables in MVPs. The method is based on the assumption that the neighboring values of a discrete variable tend to similarly contribute to the fitness. However, that is not always the case. For example, we may set steel=1, aluminum=2, etc., between which just qualitative distinctions exist; (7) handling integer and discrete variables through a penalty function is an approach through which the accurate values of integer and discrete variables cannot be reached; (8) methods redefining the meanings of velocity, etc. are more suitable for discrete optimization because solutions have to be manipulated (similar to coding/decoding operation in genetic algorithms) during optimizing in most cases, which make it inconvenient to be applied to MVPs.

The above review of the state-of-the-art PSO algorithms for MVPs or/and COPs shows a need to develop a new PSO algorithm. In this paper, we introduce a new PSO algorithm for MVCOPs called the ranking selection-based particle swarm optimizer (RSPSO). In RSPSO, the objective function and constraints are handled separately; discrete variables are partitioned into two types (ordinary discrete and categorical variables), which are dealt with using different methods, and a new ranking selection scheme is incorporated into PSO to guide the search of particles.

The remainder of this paper is organized as follows. Section 2 briefly introduces particle swarm optimizer. Section 3 presents the RSPSO for MVCOPs. In Section 4, five engineering design examples and a benchmark function with equality constraints were solved by RSPSO, and the results are compared with the ones from some relational literature. Section 5 is devoted to the discussion, and the paper is concluded in Section 6.

2 Particle swarm optimizer

PSO is a stochastic, population-based optimization algorithm, where the population is called a swarm and its individuals are called particles. In the standard PSO algorithm, each particle not only has its position and velocity but can also memorize the historical best position discovered so far by it. The position of each particle in the search space represents a candidate solution to an optimization problem.

In a standard PSO algorithm, the PSO is initialized with a group of randomly positioned particles and then tries to search for optima by updating the historical best position, current position, and velocity of each particle iteratively until a predefined stopping criterion is met. The updating process most commonly used at the kth ($k \geq 1$) iteration can be formalized as follows (Shi and Eberhart 1998).

$$Pbest_i^{(0)} = X_i^{(0)} \tag{4}$$

$$V_{i,d}^{(0)} = \text{rand}\left(0, V_{\max,d}\right) \tag{5}$$

$$Pbest_i^{(k)} = \begin{cases} Pbest_i^{(k-1)}, & \text{if } f\left(Pbest_i^{(k-1)}\right) \leq f\left(X_i^{(k-1)}\right) \\ X_i^{(k-1)}, & \text{if } f\left(Pbest_i^{(k-1)}\right) > f\left(X_i^{(k-1)}\right) \end{cases} \tag{6}$$

$$\begin{aligned} V_{i,d}^{(k)} = {} & w \times V_{i,d}^{(k-1)} + C_1 \times rand_1 \\ & \times \left(Pbest_{i,d}^{(k)} - X_{i,d}^{(k-1)}\right) + C_2 \times rand_2 \\ & \times \left(Gbest_d^{(k)} - X_{i,d}^{(k-1)}\right) \end{aligned} \tag{7}$$

$$X_{i,d}^{(k)} = X_{i,d}^{(k-1)} + V_{i,d}^{(k)} \tag{8}$$

where X_i, $Pbest_i$, V_i denote the position, historical best position, and velocity, respectively, of the ith particle. rand $(0,V_{\max,d})$ is a random number generated following a uniform distribution between 0 and $V_{\max,d}$. $V_{\max}$ is a vector containing the maximum velocity for all dimensions. $f()$ means the fitness value of a corresponding position. N is the swarm size, and w is the inertia weight. $V_{i,d}$ is the velocity on the dth dimension of the ith particle. C_1 and C_2 are the acceleration factors. $rand_1$ and $rand_2$ are two random numbers and generated for each dimension of each particle following a uniform distribution between 0 and 1. $Gbest$ is the best position discovered so far by the whole swarm (in global version of PSO) or the neighbors of the ith particle (in local version of PSO). If $\left|V_{i,d}^{(k)}\right| > \left|V_{\max,d}\right|$, then set $V_{i,d}^{(k)} = \text{sign}\left(V_{i,d}^{(k)}\right) V_{\max,d}$. $X_{i,d}$ is the position of the dth dimension of the ith particle.

3 Ranking selection-based particle swarm optimizer

3.1 Separation of objective function and constraints

In RSPSO, objective function and constraints are handled separately. The sum of violated constraints can be calculated as follows:

$$\Phi(X) = \sum_{i=1}^{m} \max\{0, g_i(X)\} + \sum_{i=1}^{n} |h_i(X)| \tag{9}$$

For infeasible solutions, their objective function values and sums of violated constraints are used to check their nondomination levels (the number 1 is assigned to the best level, the number 2 to the second best level, and so on), and then their nondomination levels and objective function values (or sums of violated constraints) will together determine their ranks. This strategy allows the valuable information from the objective function values of infeasible solutions to be utilized without requiring the use of penalty factors.

3.2 Partitioning of discrete variables

To apply the standard PSO to MVCOPs directly, He et al. (2004) relaxed discrete variables into continuous search space. This method is successful for handling ordinary discrete variables. However, it implies that the neighboring values of a discrete variable should tend to similarly contribute to the fitness, which is not always the case. For example, a few variables may be qualitative in a practical problem. To deal with them automatically, they are arbitrarily assigned discrete numerical values. For example, we may set steel=1, aluminum=2, copper=3 or copper=10, aluminum=20, steel=30, etc., and no inherent order exists between these values. Even for a discrete numerical variable, in some cases, a high fitness value may reveal little or even no information on whether the neighboring variable values have similar or improved fitness. This happens when the distances between the neighboring values of a discrete variable are far with respect to the landscape of the fitness function. To distinguish these from ordinary discrete variables, we call this type of variable categorical.

For a given MVCOP, the discrete variables should be identified and partitioned into two types: ordinary discrete and categorical. The ordinary discrete and categorical variables are denoted by X^{DO} and X^{DC}, respectively, i.e.

$$
\begin{aligned}
&X^{\mathrm{D}} = \begin{bmatrix} X^{\mathrm{DO}} \\ X^{\mathrm{DC}} \end{bmatrix} \\
&X^{\mathrm{DO}} = [x_{nC+nI+1}, \ldots, x_{nC+nI+nDO}]^{\mathrm{T}} \\
&X^{\mathrm{DC}} = [x_{nC+nI+nDO+1}, \ldots, x_{nC+nI+nDO+nDC}]^{\mathrm{T}} \\
&x_i \in DO_i, \quad i = nC + nI + 1, \ldots, nC + nI + nDO \\
&x_i \in DC_i, \quad i = nC + nI + nDO + 1, \ldots, nC + nI + nDO + nDC
\end{aligned} \tag{10}
$$

where nDO and nDC are the number of the ordinary discrete and categorical variables, respectively, and $nDO+nDC=nD$. DO_i and DC_i are the sequences from which the ordinary discrete and categorical variables take on values, respectively.

Based on the partition of discrete variables, a solution X can be partitioned into four segments corresponding to continuous, integer, ordinary discrete, and categorical variables, respectively, i.e.,

$$\text{Solution } X = \left[X^{\mathrm{C}}, X^{\mathrm{I}}, X^{\mathrm{DO}}, X^{\mathrm{DC}}\right]^{\mathrm{T}} \tag{11}$$

This strategy makes it possible for different types of discrete variables to be searched in different modes, and as a result, the performance of RSPSO on problems with categorical variables is enhanced.

3.3 Ranking selection

Ranking selection is a two-step process. First the list of individuals is ranked, and next, the individuals are selected with some form of probability distribution based on the ranks of the individuals. Blickle and Thiele (1996) performed a comparison of the selection schemes used in EA and claimed that the optimization task and the type of problem to be solved together dictate which scheme should be employed. The existing selection schemes used in genetic algorithms (GAs) are linear and exponential-ranking selection. In this paper, we propose a new selection scheme with a view to enhancing the performance of RSPSO on MVCOPs.

3.3.1 Ranking and memorizing of historical positions

In RSPSO, two kinds of ranking operations to determine the ranks of historical positions are introduced:

1. RankI: Rank feasible positions first according to the ascending order of their objective function values, followed by infeasible positions ranked according to the ascending order of their nondomination levels. The infeasible positions at the same nondomination level are further ranked according to the ascending order of the sums of violated constraints. Note that in RSPSO, the rank 1 is assigned to the best individual, the rank 2 to the second best individual, and so on.
2. RankF: same as the RankI operation except that the infeasible positions at the same nondomination level are further ranked according to the ascending order of their objective function values.

Generally, a large amount of historical positions will be produced during the run of a PSO algorithm. However, we have found that the computational effort required to preserve many positions with higher (inferior) ranks is much greater than the benefit obtained from them and that the top N historical positions can provide a good guide for the search of a swarm. A sequence *PbestSeq* is designed to memorize the top N historical positions with lower (better) ranks, and the elements in *PbestSeq* are denoted by *Pbest*, i.e., $PbestSeq=\{Pbest_i|i=1,2,\ldots,N\}$. Corresponding to two

ranking operations (RankI and RankF), there are two methods to update *PbestSeq* as follows.

1. Method I:

$$PbestSeq^{(k)} = \begin{cases} \text{RankI}\left(P^{(1)}\right), k = 1 \\ \text{RankI}\left(PbestSeq^{(k-1)} \text{U} P^{(k)}\right)[1:N], \quad k > 1 \end{cases} \tag{12a}$$

2. Method F:

$$PbestSeq^{(k)} = \begin{cases} \text{RankF}\left(P^{(1)}\right), k = 1 \\ \text{RankF}\left(PbestSeq^{(k-1)} \text{U} P^{(k)}\right)[1:N], \quad k > 1 \end{cases} \tag{12b}$$

where $P^{(k)}$ is the current positions of particles in a swarm at the kth iteration.

According to (12a, b), if only Q positions among the top N historical positions have entered the feasible region, then *PbestSeq*[1:Q] will track the descending of the objective function value and *PbestSeq*[Q+1:N] not only objective function value but also the constraint violation. Method I slightly biases *PbestSeq*[Q+1:N]'s track on the descending of the constraint violation and method F slightly on the descending of the objective function value. Figure 1 illustrates the evolving procedure of *PbestSeq* in highly constrained problems.

As described in Section 3.4.1, a swarm will learn only from its *PbestSeq*. The strategy of updating *PbestSeq* makes the search of a swarm through the infeasible region to aim at entering the feasible region while exploring the regions with smaller objective function values (where the sums of violated constraints may be larger or smaller), which can help some particles enter the feasible region from the infeasible regions with smaller objective function values. As a result, the obtained solutions (especially for a problem that has a very small feasible region or is highly constrained) may be improved.

In addition, the historical positions in *PbestSeq* are assigned different ranks. The rank of a feasible position is always lower (better) than any infeasible positions. Among two infeasible positions, the one with a lower nondomination level always has a lower (better) rank than another; among two infeasible positions at the same nondomination level, method I and method F assign a lower (better) rank to the position with smaller objective function value and a smaller sum of violated constraints, respectively. The difference between the ranks of the historical positions in *PbestSeq* makes it possible to further bias the search of a swarm in the most promising regions, which may enhance the efficiency of the search and the quality of the solutions obtained.

3.3.2 Ranking of permissible values

In RSPSO, every permissible value of each categorical variable memorizes its historical best value (HBV), which is the objective function value or the sum of violated constraints that had been experienced by it. Motivated by Deb (2000), the following rules (denoted by rule 1) are applied to update the HBV of any permissible value (assume that the value currently experienced by a permissible value, which is either the objective function value or the sum of violated constraints, is always denoted by VC):

1. If a permissible value has not a HBV, then its VC is memorized as its HBV.
2. If the existing HBV of a permissible value and its VC are all the objective function values, then the smaller is memorized as its HBV.

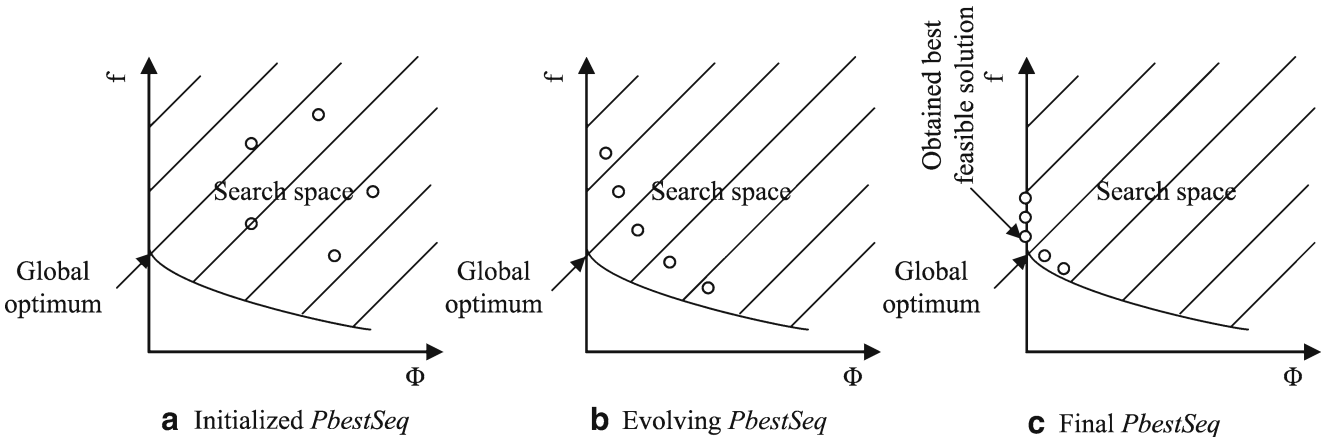


Fig. 1 Illustration of evolving procedure of PbestSeq in highly constrained problems (N=5)

3. If the existing HBV of a permissible value is the objective function value and its VC is the sum of violated constraints, then its HBV remains unchanged.
4. If the existing HBV of a permissible value is the sum of violated constraints and its VC is the objective function value, then the VC is memorized as its new HBV.
5. If the existing HBV of a permissible value and its VC are all the sums of violated constraints, then the smaller is memorized as its HBV.

When all permissible values of a categorical variable have their HBVs, they will be ranked as follows: rank permissible values with HBVs that are the objective function values first according to the ascending order of their HBVs, followed by the permissible values with HBVs that are the sums of violated constraints ranked according to the ascending order of their HBVs. This kind of ranking operation and the result of performing it on DC_i are denoted by RankV and DCV_i, respectively, i.e.,

$$DCV_i = \text{RankV}\ (DC_i) \tag{13}$$

RankV operation assigns rank to the permissible values of each categorical variable as follows: The rank of a permissible value that experienced a feasible solution is always lower (better) than any permissible value that has not experienced any feasible solution; among two permissible values that have (or not) experienced feasible solutions, the smaller HBV of which is, the lower (better) its rank. Note that rank 1 is assigned to the best one, rank 2 to the second best one, and so on.

As described in Section 3.4.2, when every permissible value of a categorical variable has been visited, the corresponding component of particles will take values from the ranked sequence of permissible values of that variable. The RankV operation assigns lower ranks to the more promising permissible values, which make it possible to further bias the search of a swarm on these more promising permissible values and consequently enhance the efficiency of the search and the quality of the solutions obtained.

3.3.3 Selection

In RSPSO, the selection scheme is used to control the probability of each element in a sequence being selected and if necessary, bias the search of a swarm in the more promising regions or on the more promising permissible values, thereby enhancing the efficiency of the search and the quality of the solutions obtained.

To do that, a parameter called the utilization-regulated parameter is introduced, which governs the selection pressure on the elements in a sequence. The selection scheme we propose can be expressed as follows.

$$g = \text{ceil}\left(\text{ceil}\left(M \times (rand_1)^{\frac{\log(ER)}{\log(0.5)}}\right) \times rand_2\right), 0 < ER \leq 1$$
$$\text{if} g = 0, \text{then set } g = 1 \tag{14}$$

where ER is the utilization-regulated parameter. M is the total number of elements in a sequence. ceil() is a ceiling operator, which returns the smallest integer that is larger than or equal to the specified value. g is the index of the element that will be selected from the sequence.

The value of ER represents the expected ratio of the number of the elements to be selected to M. When ER=1, the probability of each element in a sequence being selected is the same.

$$P\{g = i\} = \frac{1}{M},\ i = 1, 2, 3...M \tag{15}$$

When ER<1, the probability of the ith element in a sequence being selected is

$$P\{\text{g} = i\} = \sum_{k=i}^{N} \frac{k^{1/s} - (k-1)^{1/s}}{M^{1/s} \times k},\ \ i = 1, 2, 3...M \tag{16}$$

where s=log(ER)/log(0.5).

Figure 2 illustrates the probability of some elements being selected when M=40.

If we let ER=1, each element will have the same probability to be selected, which will help prevent the premature convergence of a swarm and obtain good solutions. However, it may lead to very slow convergence rates on some problems because of low selection pressure on the better elements; If we let ER≈0, then only the best element will almost exclusively be used, which may result in premature convergence. Clearly, it is preferable to set dissimilar ER values for different optimization problems to balance convergence rates and premature convergence. This is similar to the tuning of selection pressure in GAs and essentially a new selection scheme.

When (14) is applied to *PbestSeq*, M=N, but ER may take different values in different search phases. We simply divide the search of a swarm on a constrained problem into two phases: (1) all or some of top N historical positions are located in the infeasible region; (2) all of them are located in the feasible region. The ERs used in the former and latter cases are denoted by ERI and ERO, respectively. Adopting different ER values in different search phases

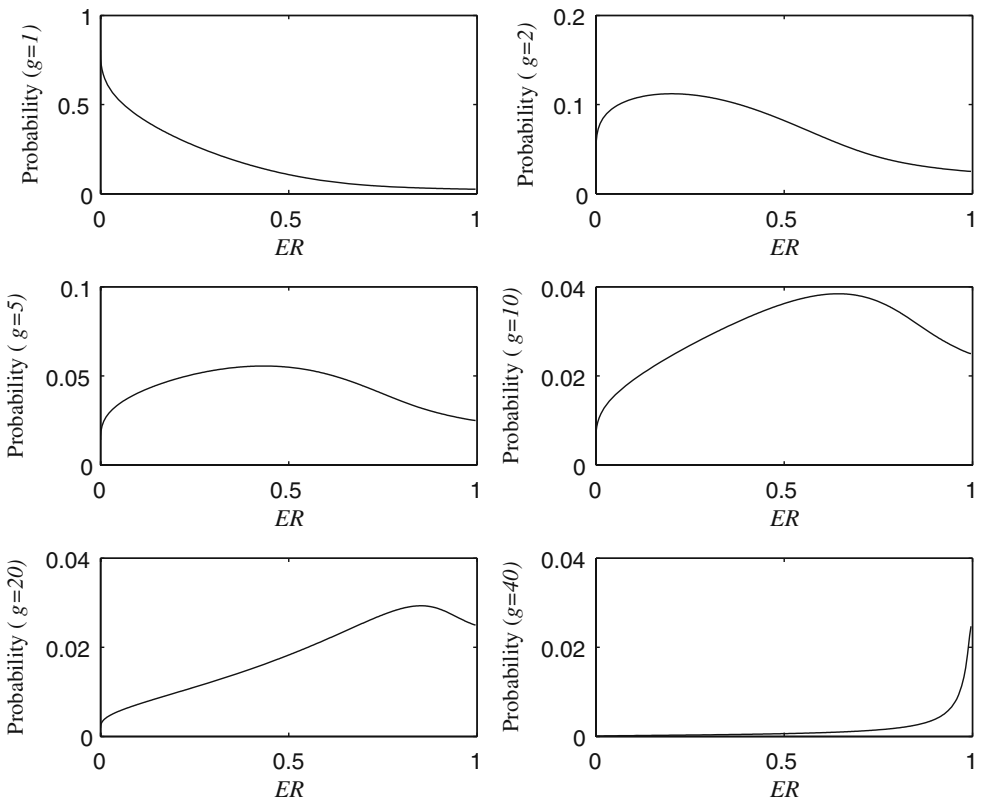


Fig. 2 Probability of some elements being selected (M=40)

help control the search behavior of a swarm, especially the convergence rate and the diversity preservation in each search phases. As a result, the obtained solutions may be improved.

When (14) is applied to the sequence of permissible values of a categorical variable, M is the total number of the permissible values. The *ER* used for categorical variables is denoted by *ERD*. As described in Section 3.4.2, categorical variables will be optimized in a mode different from continuous, integer, and ordinary discrete variables; adopting a special *ER* value for it is inevitable.

3.4 Updating of position and velocity of particle

As described in Section 3.2, a solution X can be partitioned into four segments X^C, X^{I}, X^{DO}, and X^{DC}. In RSPSO, X^{C}, X^{I}, and X^{DC} are searched directly. For X^{DO}, similar to He et al. (2004), the index of each ordinary discrete variable is searched as an integer variable instead of its discrete values directly. Therefore, the position of a particle will be

$$\text{Position } P=\left[X^{\mathrm{C}},X^{\mathrm{I}},P^{\mathrm{I}},X^{\mathrm{DC}}\right]^{\mathrm{T}} \tag{17}$$

where P^{I} is a vector comprising the integer variables, which represent the indices of the discrete values of the variables in X^{DO}. When evaluating the objective or constraints function, P^{I} should be replaced with the corresponding permissible values.

3.4.1 X^C, X^I, P^I segment

To effectively control the search behavior of a swarm, the X^{C}, X^{I}, P^{I} segment of all particles learn only from the *PbestSeq*, i.e, each particle will not know the historical best position found by itself so far. In RSPSO, two learning strategies different from the standard PSO are proposed:

1. Instead of learning from its own historical best position, each dimension (in X^{C}, X^{I}, P^{I} segment) of each particle learns from a position, which is nearest to it on the corresponding dimension among those belonging to the *PbestSeq* and being better than it in the objective function value (if it is feasible) or in either of the objective function value and the sum of violated constraints (if it is infeasible).
2. Instead of learning from the "best" position among the historical best positions of its neighbors, each dimension (in X^{C}, X^{I}, P^{I} segment) of each particle learns from a position randomly selected from the *PbestSeq*.

In addition, an operation INTR is introduced to deal with integer components of particles:

$$\mathrm{INTR}(r)=\begin{cases}\mathrm{floor}(r), & \text{if } rand > r-\mathrm{floor}(r)\\ \mathrm{ceil}(r), & \text{otherwise}\end{cases} \tag{18}$$

where floor() is a floor operator, which returns the largest integer that is less than or equal to the specified value *r*. *r* is the real number to be evaluated. *rand* is a number randomly generated following a uniform distribution between 0 and 1. The INTR operation can enhance the local search ability of individuals in comparison with a determinant method such as ceiling and floor operator.

Based on the introduced selection scheme, learning strategies, and INTR operation, the updating of the velocity and position of the X^{C}, X^{I}, P^{I} segment of the *i*th particle can be expressed as follows:

$$V_{i,d}^{(k+1)} = \begin{cases} w \times V_{i,d}^{(k)} + C_1 \times rand_1 \times \left(Pbest_{n,d}^{(k)} - P_{i,d}^{(k)}\right) + C_2 \times rand_2 \times \left(Pbest_{g,d}^{(k)} - P_{i,d}^{k}\right), \\ \qquad if\ 1 \le d \le nC \\ \mathrm{INTR}\left(w \times V_{i,d}^{(k)} + C_1 \times rand_1 \times \left(Pbest_{n,d}^{(k)} - P_{i,d}^{(k)}\right) + C_2 \times rand_2 \times \left(Pbest_{g,d}^{(k)} - P_{i,d}^{(k)}\right)\right), \\ \qquad if\ nC+1 \le d \le nC+nI+nDO \end{cases} \tag{19}$$

$$P_{i,d}^{(k+1)} = P_{i,d}^{(k)} + V_{i,d}^{(k+1)} \tag{20}$$

where, $Pbest_{n,d}$ is the *d*th dimension of a position $Pbest_n$, which is the nearest to the *i*th particle on the *d*th dimension among those belonging to the *PbestSeq* and being better than the *i*th particle in the objective fuction value (if the *i*th particle is feasible) or in either of the objective function value and the sum of violated constraints (if the *i*th particle is infeasible). If no position in the *PbestSeq* can be a $Pbest_n$ for the *i*th particle, the second item in the right hand side of (19) should be set to zero. $Pbest_g$ is randomly selected from the *PbestSeq* according to the following formula:

$$g = \mathrm{ceil}\left(\mathrm{ceil}\left(N \times (rand_1)^{\frac{\log(ER)}{\log(0.5)}}\right) \times rand_2\right), 0 < ER \le 1, \quad \text{if } g = 0, \text{ then set } g = 1$$
$$ER = \begin{cases} ERO, & \text{if all Pbests in the } PbestSeq \text{ are feasible} \\ ERI, & \text{otherwise} \end{cases} \tag{21}$$

3.4.2 X^{DC} segment

For a categorical variable, its value will be randomly selected from the sequence of its permissible values until every permissible value of it has been visited at least once, i.e.,

$$g = \mathrm{ceil}(N_d \times rand), \text{ if } g = 0 \text{ then set } g = 1 \tag{22}$$

$$P_{i,d}^{(k)} = DC_{d,g} \tag{23}$$

where $nC + nI + nDO + 1 \le d \le nC + nI + nDO + nDC$. *g* is the index of the selected value in DC_d. N_d is the total number of the values in DC_d.

When every permissible value of a categorical variable has been visited at least once, the corresponding sequence will be ranked according to (13), and then the value of this categorical variable will be randomly selected from the ranked sequence as follows:

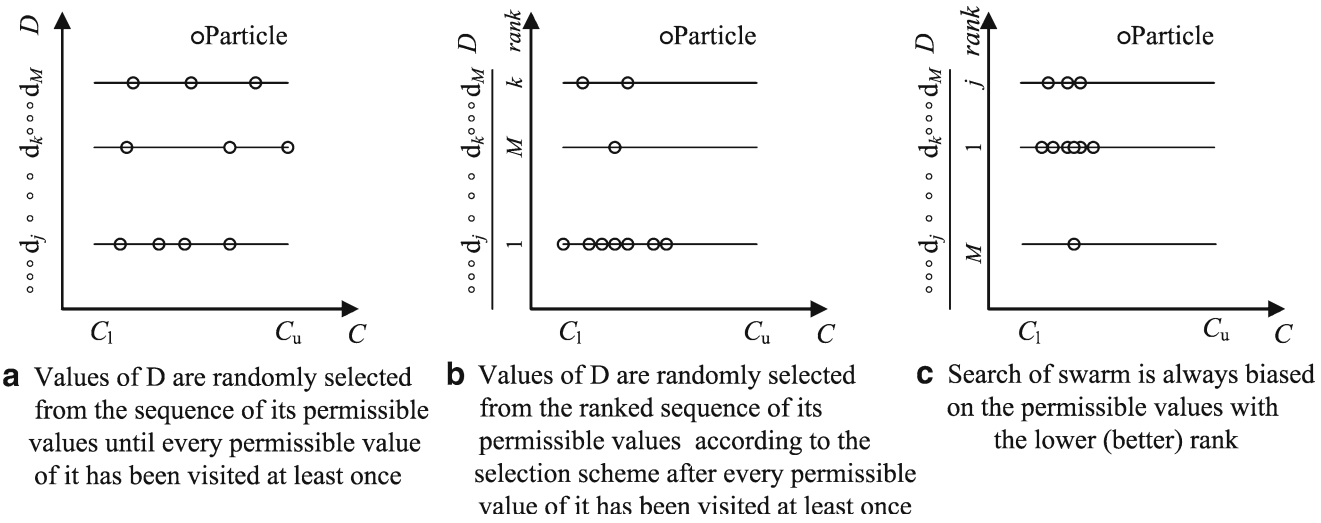
$$g = \mathrm{ceil}\left(\mathrm{ceil}\left(N_{\mathrm{d}} \times (rand_1)^{\frac{\log(ERD)}{\log(0.5)}}\right) \times rand_2\right), \quad 0 < ERD \le 1, \text{ if } g = 0, \text{ then set } g = 1 \tag{24}$$

$$P_{i,d}^{(k)} = DCV_{d,g} \tag{25}$$

where $DCV_{d,g}$ is the *g*th element in the DCV_d.

Expressions (22) to (25) is the selection mechanism employed in updating of the X^{DC} segment of a particle, and it can be seen that the concept of velocity in the standard PSO is not utilized anymore. The X^{DC} segment of a particle accurately reflects the permissible values. Figure 3 illustrates the search procedure of RSPSO on a problem with only two variables *C* and *D* ($C \in [C_{\mathrm{l}}, C_{\mathrm{u}}]$ is a continuous variable, *D* is a categorical variable with *M* permissible values).

From Fig. 3, it can be seen that the search space of a problem with categorical variables consists of disjoint regions. When RSPSO with $ERD<1$ was applied to these kinds of problems, the swarm always biases its search on the more promising disjoint regions corresponding to the permissible values with lower (better) ranks. As a result, the search efficiency is enhanced. At the same time, the particles randomly escape from these regions and explore other disjoint regions corresponding to the permissible values with higher ranks, which help prevent entrapment in one or more local optima.



a Values of D are randomly selected from the sequence of its permissible values until every permissible value of it has been visited at least once

b Values of D are randomly selected from the ranked sequence of its permissible values according to the selection scheme after every permissible value of it has been visited at least once

c Search of swarm is always biased on the permissible values with the lower (better) rank

Fig. 3 Illustration of search procedure of RSPSO on a problem with only two variables ($ERD<1$, $1<k<j<M$)

3.5 Algorithm framework

We incorporate the individual components described in detail above into a framework as follows.

```
Set N, maximum FFEs, Method to update PbestSeq, ER0, ERI, ERD, w, C1, C2
Randomly initialize P and set V=0   /*initialize the positions and velocity of particles*/
maximum iteration number= maximum FFEs/N
for k=1: maximum iteration number
    Evaluate f(Xi) and Φ(Xi), i=1,2,...,N   /* (1)(9) */
    Update the PbestSeq       /*(12a) or (12b)*/
    for d= nC+nI+nDO +1: nC+nI+nDO+nDC
        Update HBVs of all values in DCd with rule 1 .
        if each value in DCd has been visited at least once
            DCVd =RankV(DCd)   /*(13)*/
        end
    end
    for i=1:N
        for d=1: nC+nI+nDO
            Generate g     /*(21) */
            Use the gth element of the PbestSeq to calculate velocity Vi,d  /*(19)*/
            Update Pi,d   /*(20) */
        end
        for d= nC+nI+nDO +1: nC+nI+nDO+nDC
            if each value in DCd has not been visited at least once
                Update Pi,d with the value randomly selected from DCd  /*(22) (23)*/
            else
                Generate g /*(24) */
                Pi,d= DCVd,g   /*(25) */
            end
        end
    end
end
```

4 Numerical examples

In this section, five engineering design problems are used to investigate the performances of RSPSO. These problems are taken from the literature (Rao 1996) with exception of the welded beam design—case 2 (Deb and Goyal 1996)—and the four-stage gear train design (Pomrehn and Papalambros 1995a). To demonstrate the effectiveness and efficiency of RSPSO solving highly constrained optimization problems, two experiments on a well-studied benchmark function with equality constrains, which is one of the most difficult problems among the benchmark function suite (Runarsson and Yao 2000), were also conducted. The engineering problems and function are not described in this paper because of space limitation. If necessary, see the referenced literature.

Before optimizing a problem involving discrete variables, we identified and partitioned its discrete variables into ordinary discrete and categorical variables. To evaluate the performance of RSPSO on problems with categorical variables, the wire diameter $d(x_1)$ in the spring design (example 1) was treated as not only an ordinary discrete but also a categorical variable**,** although the difference between its neighboring values is small with respect to the landscape of the fitness function. The types of variables in the experiments are listed in Table 1.

As an intelligent optimization algorithm, the parameter settings of RSPSO are problem dependent and should be tuned for every given problem. Among the parameters of RSPSO, *ERO*, *ERI*, and *ERD* are the primary tunable parameters. To simplify the tuning, *ERO*, *ERI*, and *ERD* took on values only from the set {0.1, 0.25, 0.5, 0.75, 1.0}. Similar to the standard PSO, $C_{1,2}$ are also important parameters of RSPSO. In our experience, RSPSO with

Table 1 Types of variables and parameter settings for experiments

Experiment	Cont. var.	Int./ Ord. var.	Cat. var.	*N*	*Max. FFEs*	Meth.	*ERO*	*ERI*	*ERD*	*w*	C_1	C_2
Spring design (1)	1	2	0	20	15,000	I	0.5	1	N/A[c]	[d]	2.0	2.0
Spring design (2)[a]	1	1	1	20	15,000	I	0.25	1	0.25	[d]	2.0	2.0
Welded beam design—case 1	4	0	0	20	30,000	I	0.25	1	N/A	[d]	2.0	2.0
Welded beam design—case 2	1	3	2	20	15,000	I	0.25	1	0.25	[d]	2.0	2.0
Pressure vessel design	2	2	0	20	30,000	I	0.5	1	N/A	[d]	2.0	2.0
Four-stage gear train design	0	22	0	20	80,000	I	1	0.10	N/A	[d]	1.0	1.0
g13 (1)	5	0	0	20	350,000	I	1	1	N/A	[d]	1.0	1.0
g13 (2)[b]	5	0	0	20	80,000	I	0.5	1	N/A	[d]	1.0	1.0

[a] The wire diameter $d(x_1)$ was treated as a categorical variable
[b] The equality constraints were transformed into inequality ones with a tolerance value 1E-6
[c] *N/A* Not applicable
[d] The parameter *w* is set to be a random number and generated following the uniform distribution between 0.4 and 0.9

$C_{1,2}$=2.0 performs well on most problems and with $C_{1,2}$=1.0 on some hard problems. Therefore, when tuning them, first $C_{1,2}$=2.0 was tried, and if the results were not acceptable, then $C_{1,2}$=1.0 was tested. As to *maximum fitness function evaluations* (*FFEs*), it was set based on the relational literature and the complexity of the given problem. The rest (*N*, Method to update *PbestSeq*, and *w*) were kept unchanged in all the examples for their relatively less influence on the performance of RSPSO. The tuned parameter settings for the experiments are listed in Table 1.

All the experiments were performed in MATLAB. To demonstrate the robustness of RSPSO, 30 independent runs were performed for each experiment.

4.1 Example 1: spring design

This problem was first tackled by Sandgren (1990). Other approaches applied to this problem include a combined genetic adaptive search algorithms (Deb and Goyal 1997), a differential evolution algorithm (Lampinen and Zelinka 1999), and an improved PSO (He et al. 2004).

The best solutions found by the approaches mentioned above and RSPSO are showed in Table 2, and their statistical results are listed in Table 3. The results in the last row of Tables 2 and 3 are the ones when the wire diameter $d(x_1)$ was treated as a categorical variable**.**

From Tables 2 and 3, it can be seen that RSPSO obtained the same optimum as Lampinen and Zelinka (1999) and He et al. (2004). However, the mean result and standard deviation obtained by the former are smaller than those by He et al. (2004), although the FFEs consumed by them are the same (15,000). More FFEs (26,000) was adopted by Lampinen and Zelinka (1999).

When the wire diameter $d(x_1)$ was treated as a categorical variable, which takes value from 42 permissible values, the same optimum was still found by RSPSO. Furthermore, the mean result, standard deviation, and the worst solution are just slightly inferior to those when the wire diameter $d(x_1)$ was treated as an ordinary discrete variable.

4.2 Example 2: welded beam design—case 1

The approaches applied to this problem include a geometric programming (Ragsdell and Phillips 1976), a binary-coded GA combined with a traditional penalty function (Deb

Table 2 Best solution found by different algorithms for spring design

Algorithm	$d(x_1)$[a]	$D(x_2)$[a]	$N(x_3)$[a]	$f(X)$[a]
Sandgren (1990)	0.283	1.180701	10	2.7995
Deb and Goyal (1997)	0.283	1.226	9	2.665
Lampinen and Zelinka (1999)	0.283	1.223041010	9	2.65856
He et al. (2004)	0.283	1.223041010	9	2.65856
RSPSO (1)	0.283	1.223041010	9	2.65856
RSPSO (2)[b]	0.283	1.223041010	9	2.65856

[a] $d(x_1)$ The wire diameter, $D(x_2)$, the mean coil diameter, $N(x_3)$ the number of active coils, $f(X)$ the volume of the spring
[b] The wire diameter $d(x_1)$ was treated as a categorical variable

Table 3 Statistical results of different algorithms for spring design

Algorithm	*Max. FFEs*	Best	Mean	Std.	Worst
Sandgren (1990)	N/A	2.7995	NA[b]	NA	NA
Deb and Goyal (1997)	NA	2.665	NA	NA	NA
Lampinen and Zelinka (1999)	26,000	2.65856	NA	NA	NA
He et al. (2004)	15,000	2.65856	2.738024	0.107061	NA
RSPSO (1)	15,000	2.65856	2.728872	0.090461	2.95416
RSPSO (2)[a]	15,000	2.65856	2.799263	0.119812	3.06085

[a] The wire diameter $d(x_1)$ was treated as a categorical variable
[b] *NA* Not available

Table 4 Best solution found by different algorithms for welded beam design—case 1

Algorithm	$h(x_1)$[a]	$l(x_2)$[a]	$t(x_3)$[a]	$b(x_4)$[a]	$f(X)$[a]
Ragsdell and Phillips (1976)	0.2455	6.1960	8.2730	0.2455	2.385937
Deb (1991)	0.2489	6.1730	8.1789	0.2533	2.433116
Ray and Liew (2003)	0.244438276	6.237967234	8.288576143	0.244566182	2.3854347
He et al. (2004)	0.24436898	6.21751974	8.29147139	0.24436898	2.3809565827
RSPSO	0.24436898	6.21751971	8.29147140	0.24436898	2.3809565817

[a] $h(x_1)$ The thickness of the welded joint, $l(x_2)$ the length of the welded joint, $t(x_3)$ the width of the beam, $b(x_4)$ the thickness of the beam, $f(X)$ the cost of fabrication

1991), a society and civilization algorithm (Ray and Liew 2003), and a improved PSO (He et al. 2004).

The best solutions found by the approaches mentioned above and RSPSO are showed in Table 4, and their statistical results are listed in Table 5.

From Tables 4 and 5, we can see that the best solution, mean result, standard deviation, and worst solution obtained by RSPSO are better than those by Ray and Liew (2003) and He et al. (2004), although the FFEs used by the former is 30,000 and the latter 40,000 and 30,000, respectively.

4.3 Example 3: welded beam design—case 2

This problem is taken from Deb and Goyal (1996), where an adaptive genetic search algorithm was applied to solve it. The best solution found by Deb and Goyal (1996) and RSPSO are listed in Table 6, and their statistical results are shown in Table 7. From Tables 6 and 7, we can see that the best solution found by RSPSO is better than that by Deb and Goyal (1996). Moreover, the standard deviation obtained by RSPSO is small, and the mean result is very close to the optimum, although two categorical variables (material and configuration) exist.

4.4 Example 4: pressure vessel design

The approaches applied to this problem include a branch and bound technique (Sandgren 1990), an augmented Lagrange multiplier-based method (Kannan and Kramer 1994), a combined genetic search algorithm (Deb 1997), mixed variable evolutionary programming (Cao and Wu 1999), a GA-based technique employing coevolution to adapt penalty factors (Coello 2000), a GA using a dominance-based tournament selection scheme (Coello and Mezura-Montes 2001), an improved PSO (He et al. 2004), a hybrid PSO with a feasibility-based rule (He and Wang 2007a), and a coevolutionary PSO (He and Wang 2007b).

The best solutions found by the approaches mentioned above and RSPSO are showed in Table 8, and their statistical results are listed in Table 9.

From Tables 8 and 9, it can be seen that RSPSO found the same optimum as He et al. (2004) and He and Wang (2007a), but the mean result, standard deviation, and worst solution obtained by the former are much better than those by the latter. Moreover, the FFEs consumed by RSPSO are the same as that by He et al. (2004) and much less than 81,000 (adopted by He and Wang 2007a).

4.5 Example 5: four-stage gear train design

This problem has 22 discrete design variables (8 integer and 14 ordinary discrete variables) and 86 inequality constraints. Its solution space contains more than 4×10^{26} points, which is very large. However, its feasible region is very small. We did a random sampling test of 10^6evaluations, and no feasible solution was found. This is a hard nonlinear optimization problem.

The approaches applied to this problem include: a search space reduction method based on infeasibility and non-optimality tests (Pomrehn and Papalambros 1995b), a method based on constraint decomposition and designer interaction (Khorshid and Seireg 1999), a hybrid approach combining a heuristic search and a GA (Dolen et al. 2005). The method of Dolen et al. handled the constraints through

Table 5 Statistical results of different algorithms for welded beam design—case 1

Algorithm	*Max. FFEs*	Best	Mean	Std.	Worst
Ragsdell and Phillips (1976)	N/A	2.385937	NA	NA	NA
Deb (1991)	4,500	2.433116	NA	NA	NA
Ray and Liew (2003)	40,000	2.3854347	3.2551371	0.9590780	6.3996785
He et al. (2004)	30,000	2.3809565827	2.381932	0.005239371	NA
RSPSO	30,000	2.3809565817	2.380959	1.13689e−005	2.3810190

Table 6 Best solution found by different algorithms for welded beam design—case 2

Algorithm	$h(x_1)$[a]	$l(x_2)$[a]	$t(x_3)$[a]	$b(x_4)$[a]	Material	Configuration	$f(X)$[a]
Deb and Goyal (1996)	0.1875	1.6849	8.25	0.25	Steel	Four sided	1.9422
RSPSO	0.1875	1.6842	8.25	0.25	Steel	Four sided	1.9421

[a] $h(x_1)$ The thickness of the welded joint, $l(x_2)$ the length of the welded joint, $t(x_3)$ the width of the beam, $b(x_4)$ the thickness of the beam, $f(X)$ the cost of fabrication

a penalty function approach. Most importantly, in this method, they gradually rectified the most frequently violated constraints and eliminated some variables (i.e., modified the mathematical model) based on the times of each constraint being violated in every 100 runs and the analysis on the mathematical models. This method is essentially a search space reduction method.

The statistical results obtained by the approaches mentioned above and RSPSO are showed in Table 10.

For RSPSO, 29 out of the 30 runs got feasible solutions (3 out of the 29 runs reached feasible solutions before 10,000 FFEs). The average of the FFEs when the 29 runs first reached feasible solutions is 22,088, and the average of the volumes (the objective function values) when the 29 runs first reached feasible solutions is 56.72. The variable values of the best solution found by RSPSO are: 21, 45, 16, 35, 21, 42, 21, 44, 38.1, 76.2, 50.8, 38.1, 76.2, 76.2, 76.2, 38.1, 63.5, 76.2, 3.175, 3.175, 3.175, and 3.175. Note that the order of the variables is according to the literature (Dolen et al. 2005).

From Table 10 and the average of the FFEs when the 29 successful runs first reached feasible solutions, it can be seen that the result obtained by RSPSO is highly competitive with the method of Dolen et al. However, RSPSO is much easier to implement than the method of Dolen et al. RSPSO just regards the problem as a "black box" and does not require understanding it, although the understanding may be helpful. In contrast with this, to gradually rectify the most frequently violated constraints and eliminate some variables, the latter requires recording the times of each constraint violated in every 100 runs and thoroughly analyzing the mathematical models. In general, its efficiency is relatively lower and seriously depends on the designer. Moreover, it is relatively difficult for this method to be directly applied to other problems.

Table 7 Statistical results of different algorithms for welded beam design—case 2

Algorithm	*Max. FFEs*	Best	Mean	Std.	Worst
Deb and Goyal (1996)	NA	1.9422	NA	NA	NA
RSPSO	15,000	1.9421	1.9515	0.021412	1.9986

It should be pointed out that the *maximum FFEs* employed in this experiment is a tradeoff between the literature (Dolen et al. 2005) and the complexity of this problem, and a larger *maximum FFEs* is more suitable for the larger scale and difficulty of this problem. If a larger *maximum FFEs* and a larger *ER* value were together employed, RSPSO would obtain better results.

4.6 Example 6: g13 function

This problem had been solved by following approaches: a stochastic ranking-based evolution strategy (Runarsson and Yao 2000), a GA with a feasibility-based rule (Deb 2000), inverted-shrinkable Pareto archived evolution strategy method (Aguirre et al. 2004), a PSO incorporated with a turbulence operator and a decision-making scheme (Pulido and Coello 2004), a simple multimembered evolution strategy (Mezura-Montes and Coello 2005), a derivative-free filter-simulated annealing method (Hedar and Fukushima 2004), an improved stochastic ranking method (Runarsson and Yao 2005), and a dynamic-objective PSO (Lu and Chen 2006).

The results obtained by the approaches mentioned above and RSPSO are showed in Table 11. It is worth mentioning that the equality constraints were satisfied only approximately in all the approaches except RSPSO. For RSPSO, 25 out of the 30 runs got accurately feasible solutions (here, "accurately feasible" means that all the equality constraints are satisfied within MATLAB's precision bounds, not approximately like other approaches).The statistical results of these accurately feasible solutions are listed in the row labeled "RSPSO (1)" of Table 11. However, to compare RSPSO with other approaches, we also conducted an experiment where the equality constraints were converted into inequalities with a small tolerance value 1E−6, which is more stringent than others such as 1E−3 (Deb 2000, Lu and Chen 2006), 1E−4 (Runarsson and Yao 2000), and 3E−5 (Mezura-Montes and Coello 2005). In 29 out of the 30 runs, the sums of violated constraints are within the tolerance, and their statistical results are located in the row labeled "RSPSO (2)" of Table 11.

From Table 11, it can be seen that the mean result, standard deviation, and worst solution obtained by RSPSO are the best among the algorithms. Regardless of the original or transformed problem, RSPSO performed well,

Table 8 Best solution found by different algorithms for pressure vessel design

Algorithm	*Ts* (x_1)[a]	*Th* (x_2)[a]	$R(x_3)$[a]	$L(x_4)$[a]	$f(x)$[a]
Sandgren (1990)	1.1250	0.6250	48.9700	106.7200	7,982.5000
Kannan and Kramer (1994)	1.1250	0.6250	58.2910	43.6900	7,198.0428
Deb (1997)	0.9345	0.5000	48.3290	112.6790	6,410.3811
Cao and Wu (1999)	1.000	0.625	51.1958	90.7821	7,108.6160
Coello (2000)	0.8125	0.4375	40.3239	200.0000	6,288.7445
Coello and Mezura-Montes (2001)	0.8125	0.4375	40.097398	176.654047	6,059.94634
He et al. (2004)	0.81250000	0.43750000	42.09844560	176.63659584	6,059.7143
He and Wang (2007a)	0.8125	0.4375	42.0984	176.6366	6,059.7143
He and Wang (2007b)	0.8125	0.4375	42.0913	176.7465	6,061.0777
RSPSO	0.81250000	0.43750000	42.09844560	176.63659584	6,059.7143

[a] *Ts* (x_1) The thickness of the shell, *Th* (x_2) the thickness of the head, $R(x_3)$ the inner radius, $L(x_4)$ the length of the cylindrical section of the vessel, $f(X)$ the cost of fabrication

Table 9 Statistical results of different algorithms for pressure vessel design

Algorithm	*Max. FFEs*	Best	Mean	Std.	Worst
Sandgren (1990)	N/A	7,982.5000	NA	NA	NA
Kannan and Kramer (1994)	N/A	7,198.0428	NA	NA	NA
Deb (1997)	NA	6,410.3811	NA	NA	NA
Cao and Wu (1999)	NA	7,108.6160	NA	NA	NA
Coello (2000)	900,000	6,288.7445	6,293.8432	7.4133	6,308.1497
Coello and Mezura-Montes (2001)	80,000	6,059.94634	6,177.253268	130.929702	6,469.322010
He et al. (2004)	30,000	6,059.7143	6,289.92881	305.78	NA
He and Wang (2007a)	81,000	6,059.7143	6,099.9323	86.2022	6,288.6770
He and Wang (2007b)	200,000	6,061.0777	6,147.1332	86.4545	6,363.8041
RSPSO	30,000	6,059.7143	6,066.2032	13.3035	6,100.31956

Table 10 Statistical results of different algorithms for four-stage gear train design

Algorithm	Var. Num.	*Max. FFEs*	Runs	Succ. runs	Best	Mean	Std.	Worst
Pomrehn and Papalambros (1995b)	22	N/A	NA	NA	91.87	NA	NA	NA
Khorshid and Seireg (1999)	22	N/A	NA	NA	38.13	NA	NA	NA
Dolen et al. (2005)	22	10,000	100	0	–	–	–	–
	17	10,000	100	14	41.89	46.09	NA	NA
	17	10,000	100	21	41.82	46.19	NA	NA
	14	10,000	100	32	37.47	43.30	NA	NA
	14	10,000	100	60	35.40	39.78	NA	NA
RSPSO	22	10,000	30	3	44.58	55.37	12.13	68.50
		80,000		29	38.11	51.84	10.42	76.08

Table 11 Statistical results of different algorithms for g13 function

Algorithm	*Max. FFEs*	Tol.	Best	Mean	Std.	Worst
Runarsson and Yao (2000)	350,000	1E−4	0.053957	0.067543	3.1E−2	0.216915
Deb (2000)	350,050	1E−3	0.053950	0.241289[a]	NA	0.507761
Aguirre et al. (2004)	350,000	NA	0.05517	0.28184	1.8E−1	0.5471
Pulido and Coello (2004)	340,000	NA	0.068665	1.716426	NA	13.669500
Hedar and Fukushima (2004)	120,268[b]	1E−6	0.0539498	0.2977204	1.9E−1	0.4388511
Mezura-Montes and Coello (2005)	250,000	3E−5	0.053986	0.166385	1.8E−1	0.486294
Runarsson and Yao (2005)	350,000	NA	0.053942	0.096276	1.2E−1	0.438803
Lu and Chen (2006)	50,000	1E−3	0.0538666	0.6811235	4.0E−1	2.0428924
RSPSO (1)	350,000	MAT. pre.[d]	0.053987	0.056468	4.2E−3	0.071313
RSPSO (2)[c]	80,000	1E−6	0.053951	0.057657	4.1E−3	0.069198

[a] Median
[b] 120,268 is the average objective function evaluations; the average constraints function evaluations is 42,268
[c] The equality constraints were transformed into inequality ones with a tolerance value 1E−6
[d] MATLAB' default precision (double precision floating point according to IEEE Standard 754)

and the best solution found by it is very close to the global optimum, 0.0539498.

5 Discussions

Because of the diversity of real-world optimization problems and the fact that their optima are usually unknown, it is necessary to tune the parameters of PSO algorithms to find or approach the optima as close as possible. In RSPSO, there are nine tunable parameters (eight for a problem without categorical variables), among which, N, Method to update *PbestSeq*, *ERI*, and w were kept unchanged for all examples except the four-stage gear train design (ERI= 0.10); C_1 and C_2 also took the same values in all examples except g13 and the four-stage gear train design. The experiments indicate, for most engineering problems, that RSPSO may perform well when setting the Method to update *PbestSeq*, *ERI*, w, C_1, and C_2 to method I, 1.0, a random number between 0.4 and 0.9, 2.0, and 2.0, respectively. On the other hand, Method to update *PbestSeq* has only two choices (method I and method F); N, *maximum FFEs*, w, C_1, and C_2 differ little from the ones used in other PSO algorithms.

Among the parameters of RSPSO, *ERO*, *ERI*, and *ERD* are the primary tunable parameters to control the convergence rate/diversity preservation of a swarm. The lower the *ER* value, the faster the convergence and vice versa; for a smaller N, a relatively lower *ER* value will result in the rapid loss of diversity during optimizing and vice versa.

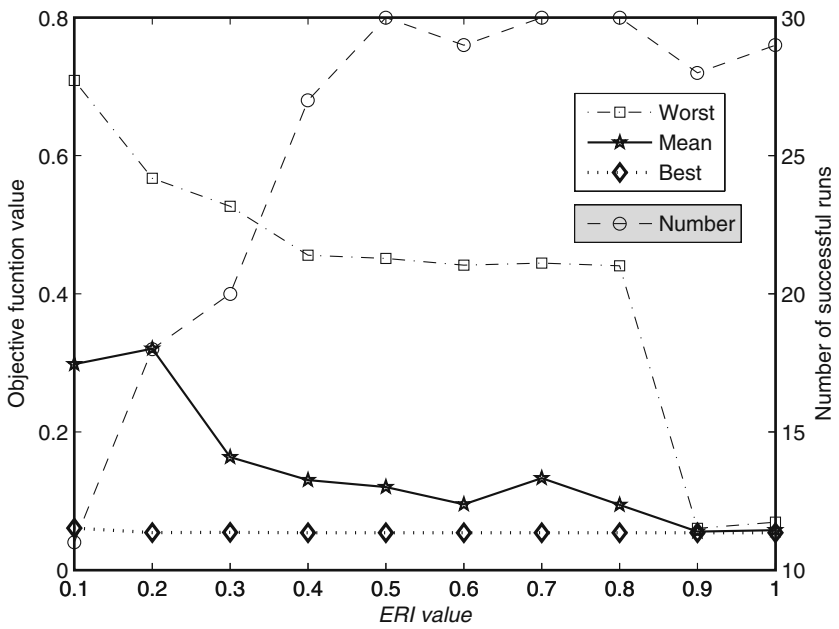


Fig. 4 Results under different *ERI* value for g13 (2)

To illustrate the relationship between the level of the ER value and the convergence rate/diversity preservation of a swarm, the experiment g13 (2) was performed with different *ERI* values (the settings of other parameters are the same as reported in the last column in Table 1). Thirty runs were performed under every *ERI* value. Figure 4 shows the number of runs that successfully reached the solution within the tolerance, the mean of objective function value of the feasible solutions obtained by RSPSO, as well as the best and worst ones. From Fig. 4, it can be seen that when $ERI \leq 0.3$, many runs could not reach the solution within the tolerance and the quality of feasible solutions is inferior because of premature convergence; when $ERI \geq 0.9$, almost all the runs successfully reached the solution within the required tolerance, and the quality of feasible solutions is greatly improved, which indicate that the a good balance between the convergence rate and diversity preservation has been achieved.

Theoretically, the quality of the solutions obtained by RSPSO for a given optimization problem vs *ER* value should be one of the following three kinds of functions:

- Monotonically decreasing function, if the *ER* value balancing convergence rate and diversity preservation is approximately zero
- Monotonically increasing function, if the *ER* value balancing convergence rate and diversity preservation is approximately or equal to 1
- Concave function, otherwise

The above functions feature very simple properties, which is the most crucial merit of the *ER* parameter. The simplicity makes it easy to seek an appropriate range of an ER value for a given problem. In our experience, for most engineering problems, tests on five *ER* values (0.10, 0.25, 0.50, 0.75, and 1.00) are sufficient for setting an *ER* value, as demonstrated by the examples. Although the tuning of *ER* (including *ERO*, *ERI*, and *ERD*) is not difficult, it may be interesting to conduct the studies of self-adaptive ER in the future because finding in these may further enhance the performance of RSPSO.

It should be pointed out that when RSPSO is applied to an optimization problem, the cost of tuning the parameters depends on the experience of the operator, the type and representation of the problem to be solved, the understanding of the results obtained by trial runs (to tune the parameters), the start point of the parameters, and the task restrictions (such as the schedule, hardware, and software). For example, example 5 (four-stage gear train design) consumed only 12 trial runs (six sets of parameter values in all) because of the experience on the other experiments, although it is the most difficult problem among the examples (it is a larger-scale and not well-studied problem with a very small feasible region). Note that in our experiments, a trial run may be incomplete; that is, once a set parameter values was judged to be helpless or not suitable based on the observation of the convergence over iteration, the trail run might be terminated immediately.

6 Conclusions

This paper presents a RSPSO to solve MVCOPs in engineering design. In this new approach, the objective function and constraints are handled separately, and feasible historical positions are ranked according to their objective function values. For infeasible historical positions, they are ranked according to their nondomination levels and objective function values (or sums of violated constraints) and are always assigned higher (inferior) ranks than feasible positions. The top N historical positions with lower (better) ranks are preserved in the *PbestSeq*. Discrete variables are partitioned into ordinary discrete and categorical variables. When all permissible values of a categorical variable have been visited, they are ranked according to their HBVs. The velocity and position of X^{C}, X^{I}, P^{I} segment of a particle learn only from the historical positions stored in the *PbestSeq* and its X^{DC} segment directly takes values from the sequences of permissible values of the corresponding categorical variables. To control the behavior of a swarm in different search phases and on categorical variables, a new selection scheme is proposed, in which the tunable parameter *ER* determines the probability of every element in the *PbestSeq* or the sequences of permissible values of a categorical variable being selected.

The separation of objectives function and constraints avoids the use of penalty factors, which are hard to be quantified. Compared with the algorithms based on preserving feasibility of solutions, RSPSO does not require initial feasible solutions. Making RSPSO even more attractive is that it can effectively solve highly constrained problems for which the algorithms using feasibility-based rule are seriously limited. As an extreme case, RSPSO can tackle problems with equality constraints without requiring the transformation of equality ones into inequalities that almost all other approaches require. Integer and discrete variables take on only permissible values without any approximation when evaluating fitness function. Categorical variables can be managed and searched directly. In contrast with this, traditional algorithms usually solve a problem with a fixed combination of all categorical variables at each run. From the point of view of technique, RSPSO is relatively simple and easy to implement.

The effectiveness, robustness, efficiency, and applicability to various problems of RSPSO have been demonstrated by five engineering design problems and a benchmark

function (highly constrained), three of which involve mixed variables. All the numerical examples indicate that RSPSO is an effective and widely applicable optimizer for optimization problems in engineering design when compared with the state-of-the-art algorithms in this area.

References

Aguirre AH et al (2004) Handling constraints using multiobjective optimization concepts. Int J Numer Meth Eng 59(15):1989–2017

Blickle T, Thiele L (1996) A comparison of selection schemes used in evolutionary algorithms. Available at: http://citeseer.ist.psu.edu/500714.html

Cao YJ, Wu QH (1999) A mixed variable evolutionary programming for optimisation of mechanical design. Eng Intell Syst Electr Eng Comm 7(2):77–82

Clerc M (2000) Discrete particle swarm optimization: a fuzzy combinatorial black box. Available at: http://clerc.maurice.free.fr/pso/Fuzzy_Discrete_PSO/Fuzzy_DPSO.htm

Coello CAC (2000) Use of a self-adaptive penalty approach for engineering optimization problems. Comput Ind 41:113–127

Coello CAC (2002) Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. Comput Meth Appl Mech Eng 191:1245–1287

Coello CAC, Mezura-Montes E (2001) Use of dominance-based tournament selection to handle constraints in genetic algorithms. In: Dagli C, Buczak AL, Ghosh J, Embrechts MJ, Ersoy O, Kercel S (eds) Proceedings of the Intelligent Engineering Systems through Artificial Neural Networks. vol. 11. ASME, St. Louis, MO, pp 177–182

Deb K (1991) Optimal design of a welded beam via genetic algorithms. AIAA J 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 497–451

Deb K (2000) An efficient constraint handling method for genetic algorithms. Comput Methods Appl Mech Eng 186:311–338

Deb K, Goyal M (1996) A combined genetic adaptive search (GeneAS) for engineering design. Comput Sci Inf 26(4):30–45

Deb K, Goyal M (1997) Optimizing engineering designs using a combined genetic search. In: Seventh International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, pp 521–528

Dolen M et al (2005) Discrete parameter-nonlinear constrained optimization of a gear train using genetic algorithms. Int J Comput Appl Tech 24(2):110–121

He Q, Wang L (2007a) A hybrid particle swarm optimization with a feasibility-based rule for constrained optimization. Appl Math Comput 86:1407–1422

He Q, Wang L (2007b) An effective co-evolutionary particle swarm optimization for constrained engineering design problems. Eng Appl Artif Intell 20:89–99

He S et al (2004) An improved particle swarm optimizer for mechanical design optimization problems. Eng Optim 36(5):585–605

Hedar A, Fukushima M (2004) Derivative-free filter simulated annealing method for constrained continuous global optimization. Dept Appl Math Phys, Kyoto Univ, Kyoto, Japan, Technical Report 2004-007

Hu XH, Eberhart RC (2002) Solving constrained nonlinear optimization problems with particle swarm optimization. In: Proceedings of the Sixth World Multiconference on Systematics, Cybernetics and Informatics, Orlando, FL, pp 203–206

Jin YX et al (2007) New discrete method for particle swarm optimization and its application in transmission network expansion planning. Electr Power Syst Res 77:227–233

Kannan BK, Kramer SN (1994) An augmented Lagrange multiplier based method for mixed integer discrete continuous optimization and its applications to mechanical design. ASME J Mech Des 116(2):405–411

Kennedy J et al (2001) Swarm intelligence. Morgan Kaufmann, San Mateo, CA

Kennedy J, Eberhart R (1995) Particle swarm optimization. In: IEEE International Conference on Neural Networks, IEEE, Piscataway, NJ, Perth, Australia 4:1942–1948

Kennedy J, Eberhart R (1997) A discrete binary version of the particle swarm algorithm. In: Proceedings of the world multiconference on systemics, cybernetics and informatics, Caracas, Venezuela, pp 4104–4109

Khorshid E, Seireg A (1999) Discrete nonlinear optimization by constraint decomposition and designer interaction. Int J Comput Appl Tech 12(2–5):233–244

Kitayama S et al (2006) Penalty function approach for the mixed discrete nonlinear problems by particle swarm optimization. Struct Multidisc Optim 32:191–202

Lampinen J, Zelinka I (1999) Mixed integer-discrete-continuous optimization by differential evolution. In: Proceedings of the 5th International Conference on Soft Computing, Iizuka, Fukuoka, Japan, pp 71–76

Liao CJ et al (2005) A discrete version of particle swarm optimization for flowshop scheduling problems. Comput Oper Res 34:3099–3111

Lu HY, Chen WQ (2006) Dynamic-objective particle swarm optimization for constrained optimization problems. J Comb Optim 12:409–419

Mezura-Montes E, Coello CAC (2005) A simple multimembered evolution strategy to solve constrained optimization problems. IEEE Trans Evol Comput 9(1):1–17

Parsopoulos KE, Vrahatis MN (2002a) Particle swarm optimization method for constrained optimization problems. In: Proceedings of the 2nd Euro-International Symposium on Computational Intelligence, IOS, Kosice, SK, pp 214–220

Parsopoulos KE, Vrahatis MN (2002b) Recent approaches to global optimization problems through Particle Swarm Optimization. Nat Comput 1:235–306

Parsopoulos KE, Vrahatis MN (2005) Unified particle swarm optimization for solving constrained engineering optimization problems. In: International Conference on Natural Computation, Springer, Berlin, pp 582–591

Pomrehn LP, Papalambros PY (1995a) Discrete optimal formulation with application to gear train design. ASME J Mech Des 117(3):419–424

Pomrehn LP, Papalambros PY (1995b) Infeasibility and non-optimality tests for solution space reduction in discrete optimal design. ASME J Mech Des 117(3):425–432

Pulido GT, Coello CAC (2004) A constraint-handling mechanism for particle swarm optimization. In: Proceedings of the 2004 Congress on Evolutionary Computation, IEEE, Piscataway, NJ, pp 1396–1403

Ragsdell KM, Phillips DT (1976) Optimal design of a class of welded structures using geometric programming. ASME J Eng Ind 98(3):1021–1025

Rao SS (1996) Engineering optimization: theory and practice, 3rd edn. Wiley, New York

Ray T, Liew KM (2001) A swarm with an effective information sharing mechanism for unconstrained and constrained single

objective optimization problems. In: Proceedings of the 2001 Congress on Evolutionary Computation, IEEE, Piscataway, NJ, pp 75–80

Ray T, Liew KM (2003) Society and civilization: an optimization algorithm based on the simulation of social behavior. IEEE Trans Evol Comput 7(4):386–396

Runarsson TP, Yao X (2000) Stochastic ranking for constrained evolutionary optimization. IEEE Trans Evol Comput 4(3):284–294

Runarsson TP, Yao X (2005) Search biases in constrained evolutionary optimization. IEEE Trans Syst Man Cybern C Appl Rev 35 (2):233–243

Sandgren E (1990) Nonlinear integer and discrete programming in mechanical design optimization. ASME J Mech Des 112(2):223–229

Sedlaczek K, Eberhard P (2006) Using augmented Lagrangian particle swarm optimization for constrained problems in engineering. Struct Multidisc Optim 32:277–286

Shi Y, Eberhart RC (1998) A modified particle swarm optimizer. In: Proceedings of the1998 IEEE Congress on Evolutionary Computation. IEEE, Piscataway, NJ, pp 69–73

Venter G, Sobieszczanski-Sobieski J (2004) Multidisciplinary optimization of a transport aircraft wing using particle swarm optimization. Struct Multidisc Optim 26:121–131