Particle Swarm Optimization in Reliability Engineering

Gregory Levitin

The Israel Electric Corporation Ltd., Israel

Xiaohui Hu

Purdue School of Engineering and Technology, USA

Yuan-Shun Dai

Department of Computer & Information Science, Purdue University School of Science, USA

4.1 Introduction

Plenty of optimization meta-heuristics have been designed for various purposes in optimization. They have also been extensively implemented in reliability engineering. For example, Genetic Algorithm (Coit and Smith, 1996), Ant Colony Optimization (Liang and Smith, 2004), Tabu Search (Kulturel-Konak, et al., 2003), Variable Neighbourhood Descent (Liang and Wu, 2005), Great Deluge Algorithm (Ravi, 2004), Immune Algorithm (Chen and You, 2005) and their combinations (hybrid optimization techniques) exhibited effectiveness in solving various reliability optimization problems.

As proved by Wolpert and Macready (1997), no meta-heuristic is versatile, which could always outperform other meta-heuristics in solving all kinds of problems. Therefore, inventing or introducing new, good optimization approaches can be very helpful in some specific areas and benefit practitioners with more options.

Since the hybrid optimization technique becomes another promising direction, combining existing tools with new ones may produce robust and effective solvers. This consideration also encourages researchers to seek novel optimization meta-heuristics.

This chapter presents applications of a new Particle Swarm Optimization (PSO) meta-heuristic for single- and multi-objective reliability optimization problems.

Originally developed for the optimization of continuous unconstrained functions, PSO did not attract much attention from the reliability community because most reliability optimization problems are of discrete nature and have constraints. However, in this chapter we show that properly adapted PSO can be an effective tool for solving some discrete constrained reliability optimization problems.

4.2 Description of PSO and MO-PSO

PSO is a population-based stochastic optimization technique invented by Kennedy and Eberhart (Eberhart and Kennedy, 1995, Kennedy and Eberhart, 1995). PSO was originally developed to simulate the behavior of a group of birds searching for food in a cornfield. The early versions of the particle swarm model were developed for simulation purposes only. Later it was discovered that the algorithms were extremely efficient when optimizing continuous non-linear unconstrained functions. Due to its easy implementation and excellent performance, PSO has been gradually applied to many engineering fields in the last several years. Various improvements and modifications have been proposed and adapted to solve a wide range of optimization problems (Hu, et al., 2004).

4.2.1 Basic Algorithm

PSO is similar to Genetic Algorithm (GA) in that the system is initialized with a group of random particles (solutions) and each particle X_i ($1 \le i \le I$) is represented by a string (vector of coordinates in the space of solutions): $X_i = \{x_{id},\ 1 \le d \le D\}$. However, it is unlike GA in that a randomized velocity $V_i = \{v_{id},\ 1 \le d \le D\}$ is assigned to each particle i and new solutions in every PSO iteration are not generated by crossover or mutation operators but by the following formula:

$$v_{id} = w \times v_{id} + c_1 \times rand_1() \times (p_{id} - x_{id}) + c_2 \times rand_2() \times (p_{nd} - x_{id}) \quad (1)$$

$$x_{id} = x_{id} + v_{id} \quad (2)$$

Eq. (1) calculates a new velocity for each particle i based on its previous velocity v_{id}, the location at which it achieved the best fitness so far p_{id}, and the neighbor's location p_{nd} at which the best fitness in a neighborhood has been achieved so far. Eq. (2) updates the position of the particle in the problem space. In this equation, $rand_1()$ and $rand_2()$ are two random numbers independently generated, c_1 and c_2 are two learning factors that control the influence of p_{id} and p_{nd} on the search process. The weight w is the particle inertia that prevents it from making undesired jumps in the solution space.

It can be learned from Eq. (1) that each particle is updated by the following two "best" values. The first one is the best solution *pBest* a particle has achieved so far. The second one is the best solution *nBest* that any neighbor of a particle has achieved so far. The neighborhood of a particle is defined as a fixed subset of particles in the population. When a particle takes the entire population as its neighbors, the best neighborhood solution becomes the global best (*gBest*).

The process of implementing the PSO is as follows:

1. Initialize the particle population (position and velocity) randomly.
2. Calculate fitness values of each particle.

3. Update *pBest* for each particle: if the current fitness value is better than *pBest*, set *pBest* to current fitness value.
4. Update *nBest* for each particle: set *nBest* to the particle with the best fitness value of all neighbors.
5. Update particle velocity/position according to equation (1) and (2).
6. If stop criteria is not attained, go back to step 2.
7. Stop and return the best solution found.

4.2.2 Parameter Selection in PSO

It can be learned from the particle update formula that particles search for better solutions by learning from their own and their neighbors' experiences. The two equations, Eq. (1) and (2), are the core part of the PSO algorithm. The parameters used in the formula will determine the performance of the algorithm.

4.2.2.1 Learning Factors

The learning factors c_1 and c_2 in Eq. (1) represent the weights of the stochastic acceleration terms that pull each particle toward *pBest* and *nBest* positions. From a psychological standpoint, the second term in Eq. (1) represents cognition, or the private thinking of the particle (tendency of individuals to duplicate past behavior that have proven successful) whereas the third term in Eq. (1) represents the social collaboration among the particles (tendency to follow the successes of others).

Both c_1 and c_2 were set to 2.0 in initial PSO works (Eberhart and Kennedy, 1995, Eberhart and Shi, 1998). The obvious reason is it will make the search, which is centered at the *pBest* and *nBest*, cover all surrounding regions. Clerc (Clerc, 1999) introduced the constriction coefficient, which might be necessary to ensure convergence of PSO. $c_1 = c_2 = 1.49445$ is also used according to the work by Clerc (Eberhart and Shi, 2001b)

In most cases, the learning factors are identical, which puts the same weights on cognitive search and social search. Kennedy (Kennedy, 1997) investigated two extreme cases: a cognitive-only model and a social-only model, and found out that both parts are essential to the success of particle swarm search.

4.2.2.2 Inertia Weight

In the original version of particle swarm, there was no inertia weight. Inertia weight *w* was first introduced by Shi and Eberhart (Shi and Eberhart, 1998). The function of inertia weight is to balance global exploration and local exploitation. Linearly decreasing inertia weights were recommended. Clerc (Clerc, 1999) introduced the constriction coefficient and suggested it to ensure convergence of PSO. Randomized inertia weight is also used in several reports (Eberhart and Shi, 2001b, Hu and Eberhart, 2002c, Hu and Eberhart, 2002b, Hu and Eberhart, 2002a). The inertia weight can be set to $[0.5 + (rand_1/2.0)]$, which is selected in the spirit of Clerc's constriction factor (Eberhart and Shi, 2001a).

4.2.2.3 Maximum Velocity

Particles' velocities are clamped to a maximum velocity *Vmax*, which serves as a constraint to control the global explosion speed of particles. It limits the maximum step change of the particle, thus adjusting the moving speed of the whole population in the hyperspace. Generally, *Vmax* is set to the value of the dynamic range of each variable, which does not add any limit. If *Vmax* is set to a lower value, it might slow the convergence speed of the algorithm. However, it would help to prevent PSO from local convergence.

4.2.2.4 Neighborhood Size

As mentioned before, *nBest* is selected from a neighborhood. The neighborhood of a particle is usually pre-defined and does not change during iterations. The neighborhood size could vary from 1 to the maximum number of solutions in the population. This size affects the propagation of information about the best particle in the group. The bigger the neighborhood size, the faster the particles can learn from the global best solutions. In an extreme case, the global version of PSO, every particle knows every other particles' movements and can learn that within one step, making PSO converge very fast. However, it also causes premature convergence that can be avoided by the injection of new solutions. Small neighborhood size may prevent premature convergence at the price of slowing the convergence speed.

4.2.2.5 Termination Criteria

The PSO terminates when the pre-specified number of iterations has been performed or when no improvement of *gBest* has been achieved during a specified number of iterations.

4.2.3 Handling Constraints in PSO

Some studies that have reported the extension of PSO to constrained optimization problems (El-Gallad, et al., 2001, Hu, et al., 2003a, Hu and Eberhart, 2002a, Parsopoulos and Vrahatis, 2002, Ray and Liew, 2001). The goals of constrained optimization problems are to find the solution that optimizes the fitness function while satisfying a set of linear and non-linear constraints. The original PSO method needs to be modified in order to handle those constraints.

Hu and Eberhart (Hu and Eberhart, 2002a) introduced an effective method to deal with constraints based on a preserving feasibility strategy. Two modifications were made to the PSO algorithm: First, when updating the *pBest* values, all the particles consider only feasible solutions; Second, during the initialization process, only feasible solutions form the initial population. Various tests show that such modification of the PSO outperforms other evolutionary optimization techniques when dealing with optimization problems with linear or nonlinear inequity constraints (Hu, et al., 2003a, Hu and Eberhart, 2002a). The disadvantage of the method is that the initial feasible solution set is sometimes hard to find.

El-Gallad (El-Gallad, et al., 2001) introduced a similar method. The only difference is that when a particle gets outside of feasible region, it is reset to the last best feasible solution found for this particle. He, et al. (He, et al., 2004) reset the particle to a previous position instead of the last best feasible solution. However, if there are several isolated feasible regions, particles may be confined in their local regions with above approaches.

Parsopoulos, *et. al* (Parsopoulos and Vrahatis, 2002) converted the constrained optimization problem into a non-constrained problem by using a non-stationary multi-stage penalty function and then applied PSO to the converted problems. It was reported that the obtained PSO outperformed Evolution Strategy and GA on several benchmark problems (Parsopoulos and Vrahatis, 2002).

Ray, *et al.* (Ray and Liew, 2001) proposed a swarm metaphor with a multilevel information sharing strategy to deal with optimization problems. It is assumed that there are some better performers (leaders) in a swarm that set the direction of the search for the rest of the particles. A particle that does not belong to the better performer list (BPL) improves its performance by deriving information from its closest neighbor in BPL. The constraints are handled by a constraint matrix. A multilevel Pareto ranking scheme is implemented to generate the BPL based on the constraint matrix. In this case, the particle should be updated using a simple generational operator instead of the regular PSO formula. Tests of such PSO modifications have showed much faster convergence and much lower number of function evaluations compared to the GA approach (Ray and Liew, 2001)

The above mentioned works have showed that modified PSO can successfully handle linear or non-linear constraints.

4.2.4 Handling Multi-objective Problems with PSO

Multi-objective optimization addresses problems with several design objectives. In multi-objective optimization (MO) problems, objective functions may be optimized separately from one another and the best solution may be found for each objective. However, the objective functions are often in conflict among themselves and a Pareto front represents the set of optimal solutions. The family of solutions of a multi-objective optimization problem is composed of all those potential solutions such that the components of the corresponding objective vectors cannot be all simultaneously improved (concept of Pareto optimality). The Pareto optimum usually gives a group of solutions called non-inferior or non-dominated solutions instead of a single solution.

The traditional way of handling MO problems is to convert them to single objective problems by using weights. Multiple optimal solutions could be obtained through multiple runs with different weights. However, methods that find groups of Pareto optimal solutions simultaneously can save time and cost.

In PSO, a particle searches the problem space based on its own (*pBest*) and its peers' (*nBest*) experience. Both cognitive and social terms in Eq. (1) play crucial roles in guiding the search process. Thus, the selection of the cognitive and social leader (*pBest* and *nBest*) are key points of MO-PSO algorithms. The selections should satisfy two rules: first, it should provide effective guidance to the particle to reach the most promising Pareto front region; second, it should provide a balanced search along the Pareto front to maintain the population diversity.

The selection of cognitive leader (*pBest*) is almost the same as in the original PSO (Hu, et al., 2003b, Hu and Eberhart, 2002b). The only difference is that the comparison is based on Pareto optimality (*pBest* is updated only if the new solution dominates all solutions visited by the particle so far).

The selection of the social leader (*nBest*) consists of two steps. The first step is to define a candidate pool from which the leader is chosen, and the second step is to define the process of choosing the leader from the candidate pool. Usually the candidate pool is the collection of all particles' *pBest* positions or an external repository that includes all the Pareto optimal solutions found by the algorithm. For the selection procedure, two typical approaches have been suggested in the literature:

1. In the roulette wheel selection scheme approach (Coello Coello, et al., 2004, Coello Coello and Lechuga, 2002, Li, 2003, Ray and Liew, 2002), all candidates are assigned weights based on some criteria (such as crowding radius, crowding factor, niche count or other measures). The general rule is to distribute the particles evenly. If there are too many particles in a small region, the region is considered to be crowded, and the particles belonging to the crowd region have less chance to be selected.

Thus they do not attract particles to this region anymore. Then, random selection is used to choose the social leader. In this scheme, selection for a candidate is stochastic and proportional to the weights. This technique aims the process at maintaining the population diversity.

2. In the quantitative standard approach, the social leader is determined by some procedure without any random selection involved, such as dynamic neighborhood (Hu, et al., 2003b, Hu and Eberhart, 2002b), sigma distance (Mostaghim and Teich, 2003), dominated tree (Fieldsend and Singh, 2002), and *etc*.

4.3 Single-Objective Reliability Allocation

4.3.1 Background

Nowadays, considerable effort is concentrated on optimal system design that balances system reliability, cost and performance. Many systems perform their intended functions at multiple levels, ranging from perfectly working to completely failed. These kinds of systems are called multi-state systems.

In the case of a multi-state system, the concept corresponding to that of reliability in a binary system is state distribution. Having the system state distribution, one can determine its reliability as a probability of being in acceptable states and its expected performance named system utility (Aven, 1993).

There are two ways to improve the system reliability or utility: First, to provide redundancies of components at each subsystem; Second, to improve the component's performance/reliability, such as allowing a component to have more chances to stay at better states or allocating more test resources on the component for reliability growth (Dai, et al., 2003). Finding an optimal balance between these two factors is a classical reliability allocation problem that has been studied in many works (Hikita, et al., 1992, Prasad and Kuo, 2000, Tillman, et al., 1977) from different aspects and by various methods.

In this section, PSO has been tested on a single objective reliability allocation problem and then compared with GA that has been carefully tuned by Tian *et al.* (2005).

4.3.2 Problem Formulation

4.3.2.1 Assumptions

A multi-state series-parallel system consists of N subsystems connected in series. Each subsystem i has n_i identical components connected in parallel, as depicted in Fig. 1.

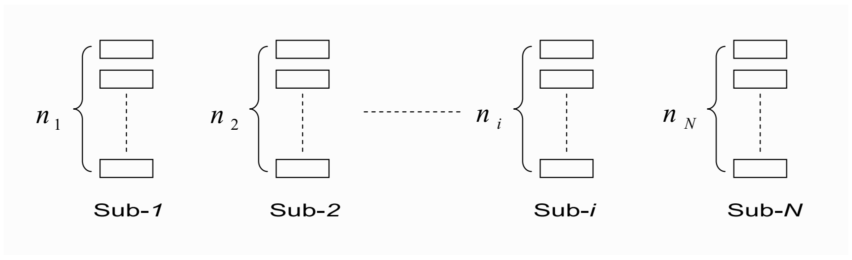

Fig. 1. Series-Parallel System

The components and the system have $M+1$ possible states: 0, 1, …, M. The states of the components in a subsystem are independent. The probability that the component belonging to subsystem i is in state j is p_{ij}. Since the component's states compose the complete group of mutually exclusive events $\sum_{j=0}^{M} p_{ij} = 1$. Therefore the state distribution of any element i is determined by M probabilities p_{ij} for $1\le j\le M$ and $p_{j0}=1-\sum_{j=1}^{M} p_{ij}$.

The system behavior can be improved by changing the number of parallel components in some subsystems or by changing the component state distribution.

4.3.2.2 Decision Variables

Two types of decision variables are of concern in the reliability allocation problem: real numbers p_{ij} ($1\le i\le N$, $1\le j\le M$) representing state distribution of the components in each subsystem *i*, and integer numbers n_i ($1 \le i \le N$) representing the number of components in each subsystem *i*. The total number of decision variables is *NM*+*N*.

4.3.2.3 Objective Function

In the optimization problem, the system utility should be maximized whereas its total cost should be limited within the given budget C_0. The system utility represents the expectperformance of multi-state systems. It is assumed that certain utility (performance) value u_s corresponds to any system state *s*. Having the probability that a multi-state series-parallel system is in state *s* or above in the form

$$\Pr[\phi(x) \ge s] = \prod_{i=1}^{N} \left[1 - \left(1 - \sum_{k=s}^{M} p_{ik} \right)^{n_i} \right], \tag{3}$$

one can obtain the multi-state system utility *U* as

$$U = \sum_{s=0}^{M} u_s \cdot \Pr[\phi(x) = s], \text{ and} \tag{4}$$

$\Pr[\phi(x) = s]$ can be derived from Eq. (3) by

$$\Pr[\phi(x) = s] = \begin{cases} \Pr[\phi(x) \ge s] - \Pr[\phi(x) \ge s+1] & (0 \le s < M) \\ \Pr[\phi(x) \ge s] & (s = M) \end{cases} \tag{5}$$

The cost model used in Tian, *et al.* (2005) adopts the cost-reliability relationship function suggested by Tillman, *et al.* (1977) which for subsystem *i* takes the form

$$C_i = c_i(r_i)[n_i + \exp(n_i / 4)], \tag{6}$$

$$c_i(r_i) = \alpha_i(-t / \ln r_i)^{\beta_i} \tag{7}$$

where r_i and $c_i(r_i)$ are reliability and cost of a single component, α_i and β_i are constants representing the inherent characteristics of components in subsystem *i,* and *t* is the operating time during which the component should not fail.

Eqs. (6) and (7) were adapted to fit the multi-state system model as follows: the cost of component *i* as a function of its state distribution is

$$c_i(p_{i1}, p_{i2},..., p_{iM}) = \sum_{j=1}^{M} (-t/\ln r_{ij})^{\beta_{ij}} \tag{8}$$

where

$$r_{ij} = p_{ij} / \sum_{k=0}^{M} p_{ik}\,, \qquad \text{for } 1 \le j \le M, \tag{9}$$

α_{ij} and β_{ij} are characteristic constants with respect to state *j*, and *t* is the operating time. The total system cost is

$$\text{C}= \sum_{i=1}^{N} c_i(p_{i1}, p_{i2},..., p_{iM})[n_i + \exp(n_i/4)] \tag{10}$$

4.3.2.4 The Problem

The single objective optimization problem is formulated as follows:

Maximize $U = \sum_{s=0}^{M} u_s \cdot \Pr[\phi(x) = s]$ (11)

Subject to: $C = \sum_{i=1}^{N} c_i(p_{i1}, p_{i2},..., p_{iM})\left[n_i + \exp\left(\frac{n_i}{4}\right)\right] \le C_0$

$p_{ij} \in [0,1]$, $1 \le i \le N$, $1 \le j \le M$,

$\sum_{j=1}^{M} p_{ij} \le 1$, $1 \le i \le N$,

$0 < n_i$ $1 \le i \le N$,

where C_0 is the maximum allowed system cost (budget).

4.3.3 Numerical Comparison

The multi-state series-parallel system considered by Tian *et al.* (2005) contains three subsystems connected in series. Any individual component and the entire system can have one of three states. The values of system utility u_s corresponding to its states are u_0=0, u_1=0.5, u_2=1.0. The cost function parameters are presented in Table 1. The system operating time is t = 1000.

Table 1. Cost function characteristic constants

Subsystem i	a_{i1}	a_{i2}	β_{i1}	β_{i2}
1	1.5E-5	4E-5	1.2	1.5
2	0.9E-5	3.2E-5	1.2	1.5
3	5.2E-5	9E-5	1.2	1.5

This problem was solved by GA in Tian, *et al.* (2005) using the physical programming framework. The optimal solution obtained by GA is shown in Table 2.

In order to compare the PSO results with results presented in Tian *et al.* (2005), C_0 was set to 89.476. The following PSO parameters were chosen: the population size of 40, the neighborhood size of 3. Maximum velocity was set to 20% of the dynamical range of the variables, the reason to choose a smaller maximum velocity is to control the convergence speed of the swarm. Learning factors c_1 and c_2 are set to 1.49445. The inertia weight w was set to [0.5 + ($rand_1$/2.0)] as mentioned in previous section. The number of iterations was 10000.

The best solution achieved by the PSO is shown in Table 2. This solution provides greater utility than one obtained by the GA with the same budget. The distribution of solutions obtained in 200 runs of the PSO with population size of 20 is shown in Table 2. The best result over 200 runs is U=0.9738, the worst one is U=0.9712, the average is U=0.9734 and the standard deviation is 0.000515. The mean value obtained by the PSO runs is the same as the best solution obtained by the GA.

Table 2. Comparison of the best solutions achieved by PSO and GA

	Genetic Algorithm			Particle Swarm Optimization		
Subsystem i	1	2	3	1	2	3
p_{i1}	0.2030	0.2109	0.2100	0.2124	0.2208	0.2042
p_{i2}	0.4200	0.4300	0.4000	0.4579	0.4712	0.4066
$\boldsymbol{n_i}$	8	8	7	7	7	7
System Utility		0.9734			0.9738	
System Cost		89.476			89.476	

4.4 Single-Objective Redundancy Allocation

The classical redundancy allocation problem belongs to the type of integer optimization problems. Many algorithms have been developed to solve the problem, including the GA (Coit and Smith, 1996), Ant Colony Optimization (Liang and Smith, 2004), Tabu Search (Kulturel-Konak, et al., 2003), Immune Algorithm (Chen and You, 2005), and Specialized Heuristic (You and Chen, 2005).

4.4.1 Problem Formulation

4.4.1.1 Assumptions

A system contains N subsystems connected in series. Each subsystem can contain multiple binary components connected in parallel (Fig. 1). Components composing each subsystem i can be different. They can be chosen from a list of M_i options. Different types of component are characterized by reliability, cost and weight. A subsystem fails if all its components fail. The entire system fails if any subsystem fails.

4.4.1.2 Decision Variable

The system structure in this problem is defined by integer numbers of components selected from the corresponding lists. The element σ_{ij} of the set of decision variables $\boldsymbol{\sigma} = \{\sigma_{ij},\ 1 \le i \le N,\ 1 \le j \le M_i\}$ determines the number of components of type j included in subsystem i. The total number of decision variables in the set $\boldsymbol{\sigma}$ is $\sum_{i=1}^{N} M_i$.

4.4.1.3 Objective Function

The general objective in this problem is to maximize the system reliability R subject to constraints on the total system cost and weight. Suppose the component of type j in subsystem i has reliability $r(i,j)$, cost $c(i,j)$, and weight $w(i,j)$. For the given set of chosen components $\boldsymbol{\sigma}$, the system reliability, cost and weight can be obtained by

$$R(\boldsymbol{\sigma}) = \prod_{i=1}^{N}\left(1 - \prod_{j=1}^{M_i}[1 - r(i,j)]^{\sigma_{ij}}\right) \tag{12}$$

$$C(\boldsymbol{\sigma}) = \sum_{i=1}^{N}\sum_{j=1}^{M_i} c(i,j)\sigma_{ij} \tag{13}$$

$$W(\boldsymbol{\sigma}) = \sum_{i=1}^{N}\sum_{j=1}^{M_i} w(i,j)\sigma_{ij} \tag{14}$$

The optimization problem can be formulated as follows:

Maximize $R(\boldsymbol{\sigma})$ (15)

Subject to: $C(\boldsymbol{\sigma}) \le C_0$, $W(\boldsymbol{\sigma}) \le W_0$,

$0 \le \sigma_{ij} \le M_i$, for $1 \le i \le N$,

$\sum_{j=1}^{M_i} \sigma_{ij} \le K_i$, $1 \le i \le N$,

where C_0 and W_0 are maximal allowed system cost and weight, and K_i is a maximal allowed number of components in subsystem i.

4.4.2 Numerical Comparison

The Fyffe, *et al.* problems as devised by Nakagawa & Miyazaki (1981) are used for comparison among different algorithms. The results of this comparison can be found in chapter 1 of this book. PSO has been tested on the first 12 problems (*W*=191 to 180). For each problem the results of 100 runs were obtained. The worst, best and average results over 100 runs are shown in Table 3. It can be seen that PSO performs very poor compared to the algorithm by You & Chen (2005). PSO just slightly outperforms the random search algorithm running for the same time (the best PSO results are slightly better than the results of random search whereas the worst PSO results are even worse than the worst random search results).

Table 3. Results from PSO and Random Search

W	PSO Worst	Rand Worst	PSO Mean	Rand Mean	PSO Best	Rand Best	Y&C-05
191	0.96918	0.97413	0.97792	0.97711	0.98209	0.97916	0.98681
190	0.96900	0.97342	0.97772	0.97605	0.98238	0.97859	0.98642
189	0.97017	0.97137	0.97673	0.97494	0.98214	0.97783	0.98592
188	0.96668	0.97153	0.97570	0.97467	0.98121	0.97773	0.98538
187	0.96812	0.96923	0.97480	0.97340	0.98047	0.97574	0.98469
186	0.96554	0.96963	0.97344	0.97356	0.97974	0.97654	0.98418
185	0.96594	0.96879	0.97201	0.97149	0.97984	0.97627	0.98350
184	0.96562	0.96803	0.97163	0.97168	0.97846	0.97554	0.98299
183	0.95826	0.96706	0.97032	0.96951	0.97802	0.97163	0.98226
182	0.95713	0.96556	0.96960	0.96872	0.97538	0.97072	0.98152
181	0.95800	0.96347	0.96793	0.96745	0.97416	0.97063	0.98103
180	0.96030	0.96334	0.96696	0.96684	0.97374	0.96854	0.98029

The major reason why PSO has such a poor performance is due to the regular coding scheme of particles. There are poor correlations among neighbors in the solutions space. The main assumption of PSO is that the neighbors of a good solution are also good. However, it is not true in the considered redundancy allocation problem.

4.5 Single Objective Weighted Voting System Optimization

4.5.1 Problem Formulation

Voting systems are widely used in human organization systems as well as in technical decision making systems. The weighted voting systems (WVS) are generalizations of the voting systems. The applications of WVS can be found in imprecise data handling, safety monitoring and self-testing, multi-channel signal processing, pattern recognition and target detection, etc. (Levitin, 2005a).

A WVS makes a decision about propositions based on the decisions of n statistically independent individual units of which it consists (for example, in target detecting system speed detectors and heat radiation detectors provide the system with their individual decisions without communicating among themselves). Each proposition is *a priori* right or wrong, but this information is available for the units in implicit form. Therefore the units are subject to the following three errors:

1. Acceptance of a proposition that should be rejected (fault of being too optimistic),
2. Rejection of a proposition that should be accepted (fault of being too pessimistic),
3. Abstaining from voting (fault of being unavailable or indecisive).

This can be modeled by considering system input I being either 1 (proposition to be accepted) or 0 (proposition to be rejected), which is supplied to each unit. Each unit j produces its decision (unit output) $d_j(I)$ which can be 1, 0 or x (in the case of abstention). Inequality $d_j(I)\neq I$ means that the decision made by the unit is wrong. The listed above errors can be expressed as

1. $d_j(0)=1$ (unit fails stuck-at-1),
2. $d_j(1)=0$ (unit fails stuck-at-0),
3. $d_j(I)=x$ (unit fails stuck-at-x).

Accordingly, reliability of each unit j can be characterized by probabilities of these errors: $q_{01}^{(j)}$ for the first one, $q_{10}^{(j)}$ for the second one, $q_{1x}^{(j)}$ and $q_{0x}^{(j)}$

for the third one (stuck-at-x probabilities can be different for inputs I=0 and I=1).

Each voting unit j has two weights that express its relative importance in the WVS: "negative" weight w^0_j, which is assigned to the unit when it votes for the proposition rejection, and "positive" weight w^1_j, which is assigned to the unit when it votes for the proposition acceptance. To make a decision about proposition acceptance, the system incorporates all the unit decisions into a unanimous system output D. The proposition is rejected by the WVS ($D(I)$=0) if the total weight of units voting for its acceptance is less than a pre-specified fraction τ of total weight of not abstaining units (τ is usually referred to as the threshold factor). The WVS abstains ($D(I)=x$) if all of its voting units abstain.

The system fails if $D(I)\neq I$. The entire WVS reliability can be defined as $R=\Pr\{D(I)=I\}$. One can see that the system reliability is a function of reliabilities of units it consists of. It also depends on the unit weights and the threshold. While the units' reliabilities usually can not be changed when the WVS is built, the weights and the threshold can be chosen in such a way that maximizes the entire WVS reliability $R(w^0{}_1,w^1{}_1,\ldots, w^0{}_n,w^1{}_n,\tau)$.

In many technical systems the time when the output (decision) of each voting unit is available is predetermined. For example, the decision time of a chemical analyzer is determined by the time of a chemical reaction. The decision time of a target detection radar system is determined by the time of the radio signal return and by the time of signal processing by the electronic subsystem. In both these cases, the variation of the decision times is usually negligible.

On the contrary, the decision time of the entire WVS composed from voting units with different constant decision times can vary. Indeed, the system does not need to wait for decisions of slow voting units, as long, as the system can make a correct decision with reliability higher than a pre-specified level. Moreover, in some cases the decisions of the slow voting units do not affect the decision of the entire system since this decision becomes evident after the fast units have voted. This happens when the total weight of units voting for the proposition acceptance or rejection is enough to guarantee the system decision independently of the decisions of the units that have not voted yet. In such situations, the voting process can be terminated without waiting for slow units' decisions, and the WVS decision can be made in a shorter time.

The number of combinations of unit decisions that allow the entire system decision to be obtained before the outputs of all of the units become available depends on the unit weight distribution and on the threshold value. By increasing the weights of the fastest units one makes the WVS more decisive in the initial stage of voting and therefore reduces the mean system decision time at the price of making it less reliable.

Since the units' weights and the threshold affect both the WVS's reliability and its expected decision time, the problem of the optimal system tuning can be formulated as follows: find the voting units' weights and the threshold that maximize the system reliability R while providing the expected decision time T not greater than a pre-specified value T^*:

Maximize $R(w^0{}_1, w^1{}_1, \ldots, w^0{}_n, w^1{}_n, \tau)$
Subject to: $T(w^0{}_1, w^1{}_1, \ldots, w^0{}_n, w^1{}_n, \tau) \leq T^*$

The method for calculating the WVS reliability and the expected decision time T, GA-based procedure for solving the optimization problem (16), was suggested in (Levitin 2005b). Here we compare PSO and GA optimization techniques on the numerical example presented in (Levitin 2005b).

4.5.2 Numerical Comparison

Experimentation was performed on a WVS consisting of six voting units with voting times and fault probabilities presented in Table 4.
Both GA and PSO require the solution to be coded as a finite length string. The natural representation of a WVS weight distribution is by an $2n+1$-length integer string $(s_1, \ldots, s_{2n+1})$ in which the values in $2j$-1 and $2j$ position corresponds to the weights $w^0{}_j$ and $w^1{}_j$ of j-th unit of the WVS and the value in position $2n+1$ corresponds to the threshold. The unit weights are further normalized in such a way that their total weight is always equal to a constant. As in (Levitin 2005b), the string elements take values in the range [0, 150], and the normalization takes the form:

$$w_j^0 = 10 s_{2j-1} / \sum_{j=1}^{n} s_{2j-1}, \quad w_j^1 = 10 s_{2j} / \sum_{j=1}^{n} s_{2j}, \quad \tau = s_{n+1} / 150. \quad (17)$$

Table 4. Parameters of voting units

No of unit j	t_j	$q_{01}(j)$	$q_{0x}(j)$	$q_{10}(j)$	$q_{1x}(j)$
1	10	0.22	0.31	0.29	0.12
2	12	0.35	0.07	0.103	0.30
3	38	0.24	0.08	0.22	0.15
4	48	0.10	0.05	0.2	0.01
5	55	0.08	0.10	0.15	0.07
6	70	0.08	0.01	0.10	0.05

Each new solution is decoded, and its objective function (fitness) value is estimated. In order to find the solution of Eq. (16), the fitness function is defined as:

$$F = R - \alpha \cdot \min(T - T^*, 0), \quad (18)$$

where α is a penalty coefficient. For solutions with $T<T^*$ the fitness of the solution depends only on WVS reliability.

The population size for both PSO and GA was chosen 50. Initial experimentation with the PSO showed that the best composition of parameters is: number of solution update cycles (PSO iterations) N=4500; $V_{\max}$=40; c_1=2; c_2=1.5, w linearly decreases as PSO proceeds: $w = 0.8+0.4(1-i/N)$.

In order to improve the PSO performance and avoid its convergence to local optima, in each M-th solution update cycle, the solutions (besides the best one in the population), instead of updating, were replaced by new randomly generated solutions with probability p. These new solutions had velocity 0. It is similar to the "mutation" operator in Genetic Algorithms, the experiments show that "injection of new solutions" improves the performance and the composition M=100, and p=1/3 gives the best improvement.

In order to compare PSO with and without injection of random solutions the optimal voting unit weights and thresholds were obtained for WVS with parameters given in Table 4 and with different values of T^*. For each T^*, 100 solutions were obtained (for the same problem) by both modifications of the PSO starting with different initial randomly generated sets of solutions. The average obtained reliability A, the coefficient of variation V of the obtained reliability over 100 solutions and the average running time Tr were calculated. These indices are presented in Table 5, as well as relative increase of average obtained reliability $\delta A=100\cdot(A_i-A)/A$ (%), increase in coefficient of variation $\Delta V= V_i-V$, and relative increase of average running time $\delta Tr=100\,(Tr_i-Tr)/Tr$ (%).

Table 5. Comparison of PSO with and without injection of random solutions

	No injection			Injection			Comparison		
T^*	A	V	Tr	A_i	V_i	Tr_i	δA	ΔV	δTr
50	0.9498	0.0999	18.20	0.9501	0.0906	18.49	0.023	-0.009	1.593
48	0.9447	0.1400	18.29	0.9450	0.1313	19.01	0.033	-0.009	3.937
46	0.9436	0.1761	18.31	0.9440	0.1680	19.05	0.045	-0.008	4.042
44	0.9377	0.2431	16.28	0.9388	0.0669	17.37	0.110	-0.176	6.695
42	0.9321	0.7384	14.98	0.9337	0.2244	14.78	0.172	-0.514	-1.335
40	0.9222	1.2532	10.93	0.9254	0.3576	11.10	0.344	-0.896	1.555
38	0.9123	1.4015	13.14	0.9155	0.2526	13.20	0.350	-1.149	0.457
36	0.9053	2.6553	12.96	0.9147	0.2954	14.53	1.033	-2.360	12.114
34	0.9019	2.7998	12.31	0.9128	0.2686	14.09	1.208	-2.531	14.460
32	0.8846	4.3692	9.82	0.9106	0.0604	12.02	2.934	-4.309	22.403
30	0.8734	4.6231	7.05	0.8954	2.8278	8.25	2.515	-1.795	17.021
28	0.8711	3.2375	5.69	0.8834	0.8038	7.04	1.410	-2.434	23.726
26	0.8612	3.1374	4.20	0.8740	1.6292	4.08	1.489	-1.508	-2.857

It can be seen that the PSO with injection of random solutions always outperforms the regular PSO. It produces better solutions with less variation at the price of 8% increase of running time (on average).

In order to compare PSO with injection of random solutions with GA, 100 solutions were obtained by GA starting with different initial populations (100 seeds) for each one of the optimization problems. The parameters of the GA were the same as in (Levitin, 2005b). The results of this comparison are presented in Table 6.

Table 6. Comparison of PSO and GA results

	GA					PSO					Comparison			
T^*	A_g	max_g	min_g	V_g	Tr_g	A_p	max_p	min_p	V_p	Tr_p	δA	δmax	ΔV	δTr
50	0.9502	0.9509	0.9478	0.0834	21.34	0.9501	0.9509	0.9466	0.0906	18.49	-0.018	0.0000	0.007	-13.355
48	0.9450	0.9466	0.9414	0.1661	21.96	0.9450	0.9466	0.9416	0.1313	19.01	0.003	0.0004	-0.035	-13.434
46	0.9420	0.9444	0.9265	0.2950	14.43	0.9440	0.9444	0.9324	0.1680	19.05	0.221	0.0000	-0.127	32.017
44	0.9379	0.9392	0.9336	0.1438	15.81	0.9388	0.9392	0.9358	0.0669	17.37	0.095	0.0004	-0.077	9.867
42	0.9317	0.9349	0.9225	0.3612	11.67	0.9337	0.9349	0.9232	0.2244	14.78	0.217	0.0005	-0.137	26.650
40	0.9252	0.9276	0.9192	0.3640	12.62	0.9254	0.9276	0.9181	0.3576	11.10	0.019	0.0000	-0.006	-12.044
38	0.9151	0.9180	0.9103	0.2154	16.15	0.9155	0.9180	0.9091	0.2526	13.20	0.045	0.0003	0.037	-18.266
36	0.9151	0.9159	0.9040	0.2867	16.74	0.9147	0.9160	0.9029	0.2954	14.53	-0.047	0.0008	0.009	-13.202
34	0.9130	0.9131	0.9123	0.0128	9.49	0.9128	0.9131	0.8887	0.2686	14.09	-0.020	0.0003	0.256	48.472
32	0.9107	0.9108	0.9086	0.0250	16.70	0.9106	0.9108	0.9074	0.0604	12.02	-0.013	0.0003	0.035	-28.024
30	0.9021	0.9036	0.8037	1.1327	12.13	0.8954	0.9037	0.8037	2.8278	8.25	-0.746	0.0049	1.695	-31.987
28	0.8736	0.8850	0.7950	3.0113	5.86	0.8834	0.8850	0.8214	0.8038	7.04	1.115	0.0000	-2.208	20.137
26	0.8695	0.8779	0.7460	2.8087	6.27	0.8740	0.8779	0.7893	1.6292	4.08	0.510	0.0010	-1.179	-34.928

For each problem, maximal, minimal and average reliability obtained over 100 seeds (*max* , *min* and *A* respectively) are presented in Table 6, as well as the coefficient of variation *V* and average running time *Tr* (seconds). Relative indices δA=100·(A_p-A_g)/A_g (%), δmax=100·(max$_p$-max$_g$)/max$_g$ (%), ΔV= V_p-V_g and δTr =100 (Tr_p-Tr_g)/Tr_g (%) are calculated. The comparison shows that GA and PSO produce very close results. However, PSO usually produces better solutions; the best PSO solutions over 100 seeds are always better or the same as the best GA solutions (δmax≥0). The average value of δA over all of the problems tested is 0.106%. PSO produces solutions with less variation (average value of ΔV over all of the problems tested is -0.133) in less running time (average value of δTr over all of the problems tested is -2.16%).

The difference in variability over all of the test problems between the PSO and the GA is shown in Fig. 2.

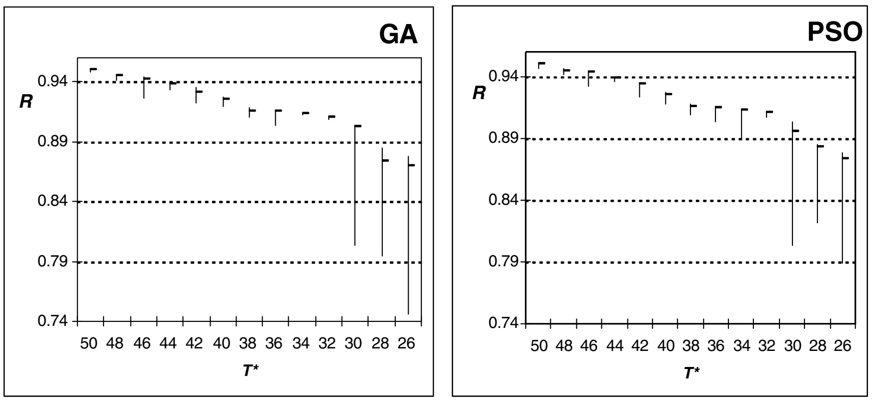

Fig. 2. Range of obtained WVS reliability over 100 seeds with mean shown as horizontal dash

4.6 Multi-Objective Reliability Allocation

4.6.1 Problem Formulation

In order to demonstrate PSO's ability to solve multi-objective optimization problems, it was applied to the multi-objective formulation of the reliability allocation problem presented in section 4.3. The problem formulation is as follows:

Maximize $$U = \sum_{s=0}^{M} u_s \cdot \Pr[\phi(x) = s] \tag{19}$$

and Minimize $$C = \sum_{i=1}^{N} c_i(p_{i1}, p_{i2}, \dots, p_{iM})[n_i + \exp(n_i/4)]$$

Subject to: $$U \ge U_0, \quad C \le C_0,$$

$$p_{ij} \in [0,1], \qquad 1 \le i \le N,\ 1 \le j \le M,$$

$$\sum_{j=1}^{M} p_{ij} \le 1, \qquad 1 \le i \le N,$$

$$0 < n_i \qquad 1 \le i \le N,$$

where U_0 is the lower bound of the system utility, and C_0 is the upper bound of the cost.

4.6.2 Numerical Comparison

The same example problem parameters as in section 4.3 were used while the budget and utility limits were C_0=200 and U_0=0.75. and.

A dynamic neighborhood PSO developed by Hu, *et al.* (Hu, et al., 2003b, Hu and Eberhart, 2002b) was employed to deal with this two-objective problem. The system utility was set to be the optimization objective and the cost was set to be the neighborhood objective. As mentioned before, the key point is to find the cognitive leader and the social leader. The cognitive leader (*pBest*) is updated when the particle is better than the old *pBest* in both system utility and cost values. All the Pareto optimal solutions found by PSO form the candidate pool for the selection of social leader *nBest*. First, the differences of cost between the particle and all candidates in the pool are calculated, then the particles with closest cost are chosen to be the neighbors of the particle, finally the candidate with the greatest system utility value became the social leader. Fig. 3 illustrates the selection process. The black curve is the final Pareto front. The dots are the current Pareto optimal solutions. The circle is a particle. The particle finds several nearest Pareto optimal solutions in term of cost as neighbors. Then the neighbor with highest utility fitness value is selected to be *nBest*.

The general procedure is as follows:

1. Initialize the population.
2. Calculate values of each objective function for the particles in the population.
3. For each solution (particle): if the current fitness values of the particle are better than any current solution in the Pareto optimal solution archive, then put the particle into the archive.
4. Find the *nBest* and *pBest* according to the dynamic neighborhood method.
5. Update particle velocity and position.

6. If maximum iteration number is not reached, go back to step 2.
7. Find all Pareto optimal solutions in the archive and generate solution set.

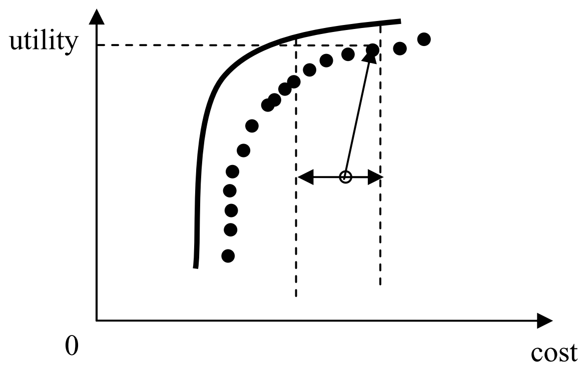

Fig. 3. Illustration of dynamic neighborhood strategy

The population size of 20 and neighborhood size of 3 were used in the optimization. Maximum velocity was set to 20% of the dynamical range of the variables. Learning factors c_1 and c_2 are set to 1.49445.

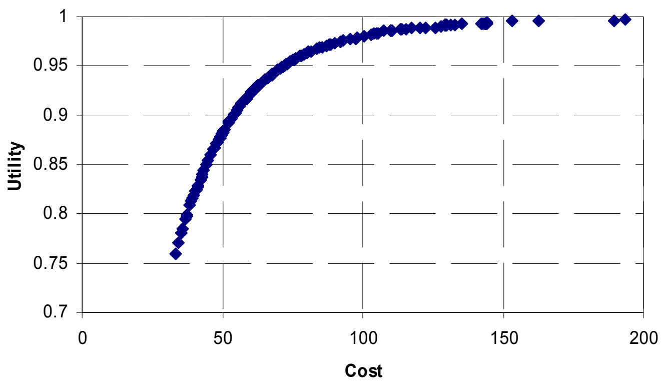

Fig. 4. Pareto optimal solutions obtained by MO-PSO

The inertia weight w was set to $[0.5 + (rand_1/2.0)]$ as mentioned in the previous section. Number of iterations was set to 10000. The average time for each run of the PSO on a HP Pentium IV 2.8GHz personal computer was less than 30 seconds.

The Pareto optimal solutions obtained from a single PSO run are presented in Fig. 4. A total of 200 Pareto optimal solutions have been found during the optimization. It can be seen in Fig. 4 that the solutions found by MO-PSO cover all the regions in the Pareto front and are almost evenly distributed along this front.

4.7 PSO Applicability and Efficiency

As it is reported in the PSO literature (Hu, et al., 2004), this algorithm has several advantages compared to other meta-heuristics:
-It can be easily implemented and adapted to complex problems (for the continuous optimization problems, the encoding is straightforward and does not need extra conversion);
- It allows simple constraint handling;
- Its convergence properties are almost insensitive to the design of fitness functions.

According to (Elbeltagi, et al., 2005), PSO often outperforms other meta-heuristics on the wide range of optimization problems. The tests performed in this study showed that PSO is able to at least get better results than those obtained by GA for several single- and multi-objective reliability optimization problems. However, PSO has showed poor performance when it was applied to the discrete redundancy allocation problem. This can be explained by the fact that the fundamental assumption in particle swarm is that the neighbor regions of a good solution are also good, i.e., small variations in vector representing the solution causes small variations in the fitness function (which lies on the base of the velocity update formula). This assumption usually holds in reliability allocation problems where fitness functions are relatively smooth. However, it is not always true in some discrete problems such as redundancy optimization.

It seems that the main cause preventing good performance of the PSO in solving discrete optimization problems with unsmooth fitness functions lies in its studying behavior based on gradual learning from the best solution. The basic PSO principle of emulating social behavior of learning from better examples becomes meaningless when approaching the best solution causes degradation.

The possible directions of PSO performance improvement are design of solution encoding schemes providing fitness function smoothness and combining the PSO with other heuristics and local search methods.

As an emerging stochastic optimization method, PSO exhibits great potential in solving reliability engineering problems. However, many issues remain unsolved, which requires further investigation. The adaptation of PSO for solving sequencing, partition and scheduling problems that arise in reliability engineering is the main challenge for further research.

References

Aven, T., "On performance-measures for multi-state monotone systems," *Reliability Engineering & System Safety*, vol. 41, no. 3, pp. 259-266, 1993.

Chen, T. C. and You, P. S., "Immune algorithms-based approach for redundant reliability problems with multiple component choices," *Computers In Industry*, vol. 56 pp. 195-205, 2005.

Clerc, M. The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. Proceedings of IEEE Congress on Evolutionary Computation, Vol.3pp. -1957, 1999

Coello Coello, C. A. and Lechuga, M. S. MOPSO: a proposal for multiple objective particle swarm optimization. Proceedings of IEEE Congress on Evolutionary Computation, Vol.2pp. 1051-1056, 2002

Coello Coello, C. A., Pulido, G. T., and Lechuga, M. S., "Handling multiple objectives with particle swarm optimization," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 3, pp. 256-279, 2004.

Coello Coello, C. A., "A comprehensive survey of evolutionary-based multiobjective optimization techniques," *Knowledge and Information Systems*, vol. 1, no. 3, pp. 269-308, Aug.1999.

Coit, D. W. and Smith, A. E., "Reliability optimization of series-parallel systems using a genetic algorithm," *IEEE Transactions on Reliability*, vol. 5, no. 2, pp. 254-260, 1996.

Dai, Y., Xie, M., Poh, K. L., and Yang, B., "Optimal testing-resource allocation with genetic algorithm for modular software systems," *Journal of Systems and Software*, vol. 66, no. 1, pp. 47-55, Jan.2003.

Eberhart, R. C. and Kennedy, J. A new optimizer using particle swarm theory. Proceedings of International Symposium on Micro Machine and Human Science, pp. 39-43, 1995

Eberhart, R. C. and Shi, Y. Tracking and optimizing dynamic systems with particle swarms. Proceedings of IEEE Congress on Evolutionary Computation, Vol.1pp. 94-100, 2001b

Eberhart, R. C. and Shi, Y. Particle swarm optimization: developments, applications and resources. Proceedings of IEEE Congress on Evolutionary Computation, Vol.1pp. 81-86, 2001a

Eberhart, R. C. and Shi, Y. Evolving artificial neural networks. Proceedings of International Conference on Neural Networks and Brain, Beijing, P. R. China. pp. PL5-PL13, 1998

El-Gallad, A. I., El-Hawary, M. E., and Sallam, A. A., "Swarming of intelligent particles for solving the nonlinear constrained optimization problem," *Engineering Intelligent Systems for Electrical Engineering and Communications*, vol. 9, no. 3, pp. 155-163, Sept.2001.

Elbeltagi, E., Hegazy, T., and Grierson, D., "Comparison among five evolutionary-based optimization algorithms," *Advanced Engineering Informatics*, vol. 19 pp. 43-53, 2005.

Fieldsend, J. E. and Singh, S. A multi-objective algorithm based upon particle swarm optimisation, an efficient data structure and turbulence. Proceedings of U.K.Workshop on Computational Intelligence, Birmingham, UK. pp. 37-44, 2002

He, S., Prempain, E., and Wu, Q. H., "An improved particle swarm optimizer for mechanical design optimization problems," *Engineering Optimization*, vol. 36, no. 5, pp. 585-605, Oct.2004.

Hikita, M., Nakagawa, Y., Nakashima, K., and Narihisa, H., "Reliability optimization of systems by a surrogatecon-straints algorithm," *IEEE Transactions on Reliability*, vol. 41, no. 3, pp. 473-480, 1992.

Hu, X. and Eberhart, R. C. Multiobjective optimization using dynamic neighborhood particle swarm optimization. Proceedings of IEEE Congress on Evolutionary Computation, Vol.2pp. 1677-1681, 2002b

Hu, X. and Eberhart, R. C. Solving constrained nonlinear optimization problems with particle swarm optimization. Proceedings of World Multiconference on Systemics, Cybernetics and Informatics, Orlando, USA. 2002a

Hu, X. and Eberhart, R. C. Adaptive particle swarm optimization: detection and response to dynamic systems. Proceedings of IEEE Congress on Evolutionary Computation, Vol.2pp. 1666-1670, 2002c

Hu, X., Eberhart, R. C., and Shi, Y. Engineering optimization with particle swarm. Proceedings of IEEE Swarm Intelligence Symposium, pp. 53-57, 2003a

Hu, X., Eberhart, R. C., and Shi, Y. Particle swarm with extended memory for multiobjective optimization. Proceedings of IEEE Swarm Intelligence Symposium, pp. 193-197, 2003b

Hu, X., Shi, Y., and Eberhart, R. C. Recent advances in particle swarm. Proceedings of IEEE Congress on Evolutionary Computation, Vol.1pp. 90-97, 2004

Kennedy, J. Minds and cultures: particle swarm implications. Proceedings of AAAI Fall Symposium: Socially Intelligent Agents, Menlo Park, CA. pp. 67-72, 1997

Kennedy, J. and Eberhart, R. C. Particle swarm optimization. Proceedings of IEEE International Conference on Neural Networks, Vol.4pp. 1942-1948, 1995

Kulturel-Konak, S., Coit, D. W., and Smith, A. E., "Efficiently solving the redundancy allocation problem using tabu search," *IIE Transactions*, vol. 35, no. 6, pp. 515-526, 2003.

Li, X. A non-dominated sorting particle swarm optimizer for multiobjective optimization. Lecture Notes in Computer Science, Chicago, IL, USA. Vol.2723pp. 37-48, 2003

Liang, Y.-C. and Smith, A. E., "An ant colony optimization algorithm for the redundancy allocation problem (RAP)," *IEEE Transactions on Reliability*, vol. 53, no. 3, pp. 527-423, 2004.

Liang, Y.-C. and Wu, C.-C., "A variable neighborhood descent algorithm for the redundancy allocation problem," *Industrial Engineering and Management Systems*, vol. 4, no. 1, pp. 109-116, 2005.

Mostaghim, S. and Teich, J. Strategies for finding good local guides in multi-objective particle swarm optimization (MOPSO). Proceedings of IEEE Swarm Intelligence Symposium, pp. 26-33, 2003

Nakagawa, Y. and Miyazaki, S., "Surrogate constraints algo-rithm for reliability optimization problems with two constraints," *IEEE Transactions on Reliability*, vol. R-30 pp. 175-180, 1981.

Parsopoulos, K. E. and Vrahatis, M. N. Particle swarm optimization method for constrained optimization problems. Proceedings of Euro-International Symposium on Computational Intelligence, 2002

Prasad, V. R. and Kuo, W., "Reliability optimization of coherent systems," *IEEE Transactions on Reliability*, vol. 49, no. 3, pp. 323-330, 2000.

Ravi, V., "Optimization of complex system reliability by a modified great deluge algorithm.," *Asia-Pacific Journal of Operational Research*, vol. 21, no. 4, pp. 487-497, 2004.

Ray, T. and Liew, K. M., "A swarm metaphor for multiobjective design optimization," *Engineering Optimization*, vol. 34, no. 2, pp. 141-153, 2002.

Ray, T. and Liew, K. M. A swarm with an effective information sharing mechanism for unconstrained and constrained single objective optimization problem. Proceedings of IEEE Congress on Evolutionary Computation, Seoul, Korea. pp. 75-80, 2001.

Shi, Y. and Eberhart, R. C. A modified particle swarm optimizer. Proceedings of IEEE Congress on Evolutionary Computation, pp. 69-73, 1998

Tian, Z., Zuo, M. J., and Huang, H. Reliability-redundancy allocation for multi-state series-parallel systems. Proceedings of European Safety and Reliability Conference, Tri City, Poland. pp. 1925-1930, 2005

Tillman, F. A., Hwang, C. L., and Kuo, W., "Determining component reliability and redundancy for optimum system reliability," *IEEE Transactions on Reliability*, vol. 26, no. 3, pp. 162-165, 1977.

Wolpert, D. H. and Macready, W. G., "No Free Lunch Theorems for Optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67-82, 1997.

You, P. S. and Chen, T. C., "An efficient heuristic for series-parallel redundant reliability problems," *Computers and Operations Research*, vol. 32 pp. 2117-2127, 2005.