Int J Adv Manuf Technol (2013) 68:317–338
DOI 10.1007/s00170-013-4730-6

ORIGINAL ARTICLE

# A hybrid GA–PSO approach for reliability optimization in redundancy allocation problem

**M. Sheikhalishahi · V. Ebrahimipour · H. Shiri · H. Zaman · M. Jeihoonian**

Received: 5 June 2010 / Accepted: 2 January 2013 / Published online: 26 January 2013


**Abstract** This paper presents a novel hybrid genetic algorithm (GA)-particle swarm optimization (PSO) approach for reliability redundancy allocation problem (RRAP) in series, series–parallel, and complex (bridge) systems. The proposed approach maximizes overall system reliability while minimizing system cost, system weight and volume, simultaneously, under nonlinear constraints. To meet these objectives, an adaptive hybrid GA–PSO approach is developed to identify the optimal solutions and improve computation efficiency for these NP-hard problems. An illustrative example is applied to show the capability and effectiveness of the proposed approach. According to the results, in all three cases, reliability values are improved. Moreover, computational time and variance are decreased compared to the similar studies. The proposed approach could be helpful for engineers and managers to better understand their system reliability and performance, and also to reach a better configuration.

**Keywords** Redundancy allocation problem · Genetic algorithm · Particle swarm optimization · Reliability optimization · Mixed-integer problems

## 1 Introduction

Due to the importance of reliability applications in industries, especially in complex systems including nuclear powers, powerhouses, etc., it is studied from different points of view in the literature. The primary goal of reliability engineering is to improve the system reliability. In the initial design activity, the redundancy allocation is a direct way of enhancing system reliability. Both reliability and redundancy optimization (reliability–redundancy optimization) should be considered as a gainful way for designing systems that are largely assembled and manufactured [1]. The integer values of redundancy levels lead to continuous zero–one interval for component's reliability. Following this point, one can notice that the reliability redundancy-allocation problem (RRAP) falls into category of mixed-integer programming. The RRAP can be formulated in two forms: (1) minimizing the total cost under constraints as volume, weight, etc. and (2) maximizing system reliability subject to such constraints [2].

In reality, systems usually include four types of series, parallel, series–parallel, and complex (bridge) configurations. Chern [3] proved that even in an ordinary structure such as series system, RRAP is organized as a NP-hard class. Hikita et al. [4] developed an algorithm to solve the RRAP in cases of series, series–parallel, and complex system based on surrogate-constraint method. The method made the solutions of the surrogate dual problem and omitted the surrogate gap by using dynamic programming in a series–parallel case under one constraint and two special conditions of monotonicity and separability.

It is ordinary that in optimization problems, some decision variables have only discrete values. Finding optimal solutions to such problems is defined as combinatorial optimization. Classical approaches such as enumeration, Lagrangian relaxation, decomposition, and cutting plane techniques or their combinations may not be computationally feasible or efficient to solve a combinatorial optimization problem of a practical size. Therefore, researchers focus on heuristic techniques that seek a good solution to a complex combinatorial problem within a reasonable time [5]. To improve computation efficiency, hybrid optimization algorithms have been increasingly used to achieve their

V. Ebrahimipour
Mathematics and Industrial Engineering Department, Ecole Polytechnique de Montreal, Québec, Canada

M. Sheikhalishahi · V. Ebrahimipour (✉) · H. Shiri · H. Zaman · M. Jeihoonian
Department of Industrial Engineering, College of Engineering, University of Tehran, Tehran, Iran
e-mail: vahid.ebrahimipour@polymtl.ca

V. Ebrahimipour
e-mail: vebrahimi@ut.ac.ir

individual advantages, simultaneously. Ant colony optimization, genetic algorithm (GA), and simulated annealing (SA) have been widely and successfully employed to deal with reliability optimization problems [6–12]. GA has been combined with other meta-heuristics and heuristics to attain more efficiency from computational point of view [2].

Altumi et al. [13] developed a mathematical model to achieve the overall system reliability and cost optimization through allocation of required spares tool in a flexible manufacturing system. Lee et al. [14] applied max–min approach and the Nakagawa and Nakashima method for the series–parallel redundancy allocation problem and compared them with each other based on quality and computational complexity point of view. They proved that Min–Max approach is better in terms of quality but inferior in computational complexity in comparison with Nakagawa and Nakashima method. Lillo and Neuts [15] proposed a heuristic search method to find the optimal spacing of inspections of a parallel system in which the given database contains general life time distribution. Zhongliang et al. [16] addressed a reliability optimization approach based on neural networks to specify the choice of components in series–parallel systems. Their neural network is based on McCulloch–Pittes neuron network model which minimizes the energy function of the neural networks to obtain optimum solution. Li et al. [17] proposed a new exact method which is the combination of the convexification transformation and branch-and-bound method to find an optimal solution of reliability optimization in complex systems. They applied a new exact approach for four typical cases to show the efficiency of the method. Norkin and Onishchenko [18] maximized the mean life of a network in an optimal redundancy problem. They applied branch-and-bound method to solve this stochastic optimization problem. Valdebenito et al. [19] provided a survey on various methods existing in the field of reliability optimization and proposed a qualitative comparison between different approaches to clarify the application of methods.

Genetic algorithms have been successfully applied for RRAP in the literature [20–22]. Hsieh et al. [23] presented the mixed-integer nonlinear reliability problems in types of series, parallel, series–parallel, and complex systems and proposed an efficient genetic algorithm as a solution technique. The application of simulated annealing in optimal reliability design can be found in the work of Kuo et al. [24]. Moreover, Kim and Bae [25] addressed SA algorithm to solve RRAP and showed that it provides better solutions in comparison to the previous works [4, 23, 26]. In another similar work, Coelho [27] considered series and complex (bridge) systems and proposed particle swarm optimization (PSO) algorithm to solve them. The presented method calling PSO-GC is based on chaotic sequence and Gaussian distribution. The solution quality improvement over the hybrid methods, PSO–CA and PSO–CO, has been shown through two examples.

According to the literature GA and PSO are widely used for solving redundancy allocation problem. In this paper a hybrid GA–PSO approach is applied to benefit from advantages of both genetic algorithm and particle swarm optimization. However, both models have strengths and weaknesses. Comparisons between GAs and PSOs have been performed by Eberhart and Angeline and both conclude that a hybrid of the standard GA and PSO models could lead to further advances [28–30]. In a GA, if an individual is not selected the information contained by that individual is lost, but PSOs have memory. However, without a selection operator, PSOs may waste resources on poor individuals. While GAs have trouble finding an exact solution and are best at reaching a global region, a PSO's group interactions boosts the search for an optimal solution, [29].

This paper considers mixed-integer nonlinear programming models for series, series–parallel, and complex (bridge) configurations. To improve the computation efficiency, a GA–PSO approach is proposed and developed to obtain better solutions along with less variation and processing time in comparison to the similar studies in this area. The remainder of this paper is organized as follows. Section 2 introduces three typical cases of the reliability-redundancy optimization problems. Section 3 elaborates the proposed hybrid GA–PSO approach as well as a brief overview of GA and PSO methods. In Section 4, the performance of the proposed approach is illustrated via numerical experiments, and concluding remarks are presented in Section 5.

## 2 Problem definition

The following formulation is considered as the general reliability–redundancy allocation problem, in which reliability is considered as the objective function. The problem is to determine the optimal combination of components and redundancy levels in order to maximize the overall system reliability under volume, cost, and weight constraints. The mixed-integer nonlinear programming model with reliability maximization objective function is shown as follows.

### Nomenclature

| | |
|---|---|
| $W$ | The upper bound on the weight of the system |
| $V$ | The upper bound on the volume of the system |

| n | | | | | r | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $n_1$ | $n_2$ | $n_3$ | $n_4$ | $n_5$ | $r_1$ | $r_2$ | $r_3$ | $r_4$ | $r_5$ |

**Fig. 1** A chromosome in GA population

| | |
|---|---|
| $C$ | The upper bound on the cost of the system |
| $W_i$ | The component weight of subsystem |
| $v_i$ | The component volume of subsystem $i$ |
| $c_i$ | The component cost of subsystem $\boldsymbol{i}$ |
| $\boldsymbol{n}$, $\boldsymbol{r}$ | The vector of the redundancy allocation and component reliabilities of the system |
| $\boldsymbol{r}$ | $(r_1,\ldots,r_m)$ |
| $b$ | The upper limit on the resource |
| $N$ | Population of $n$ |
| $GA_{PopSize}$ | Population size of GA |
| $PSO_{MaxPopSize}$ | Final population size in PSO |
| $PSO_{PopIncRate}$ | Increasing rate of PSO population size |
| $PSO_{MinIteration}$ | First maximum iteration number of PSO |
| $PSO_{NumMin}$ | Minimum number of individuals that effected by PSO |
| $PSO_{NumDecRate}$ | Decreasing rate of number of individuals that effected by PSO |
| $PSO_{MaxIter}$ | Current max iteration number in PSO |
| $keep_{Per}$ | Elite percentage |
| $c_1$ | Individual intelligence coefficient |
| $w$ and $w_{final}$ | Initial and final inertia |
| $m$ | The number of subsystems |
| $u_i$ | The upper limit of redundancy level of subsystem $i$ |
| $r_i$ | The component reliability of subsystem $i$ |

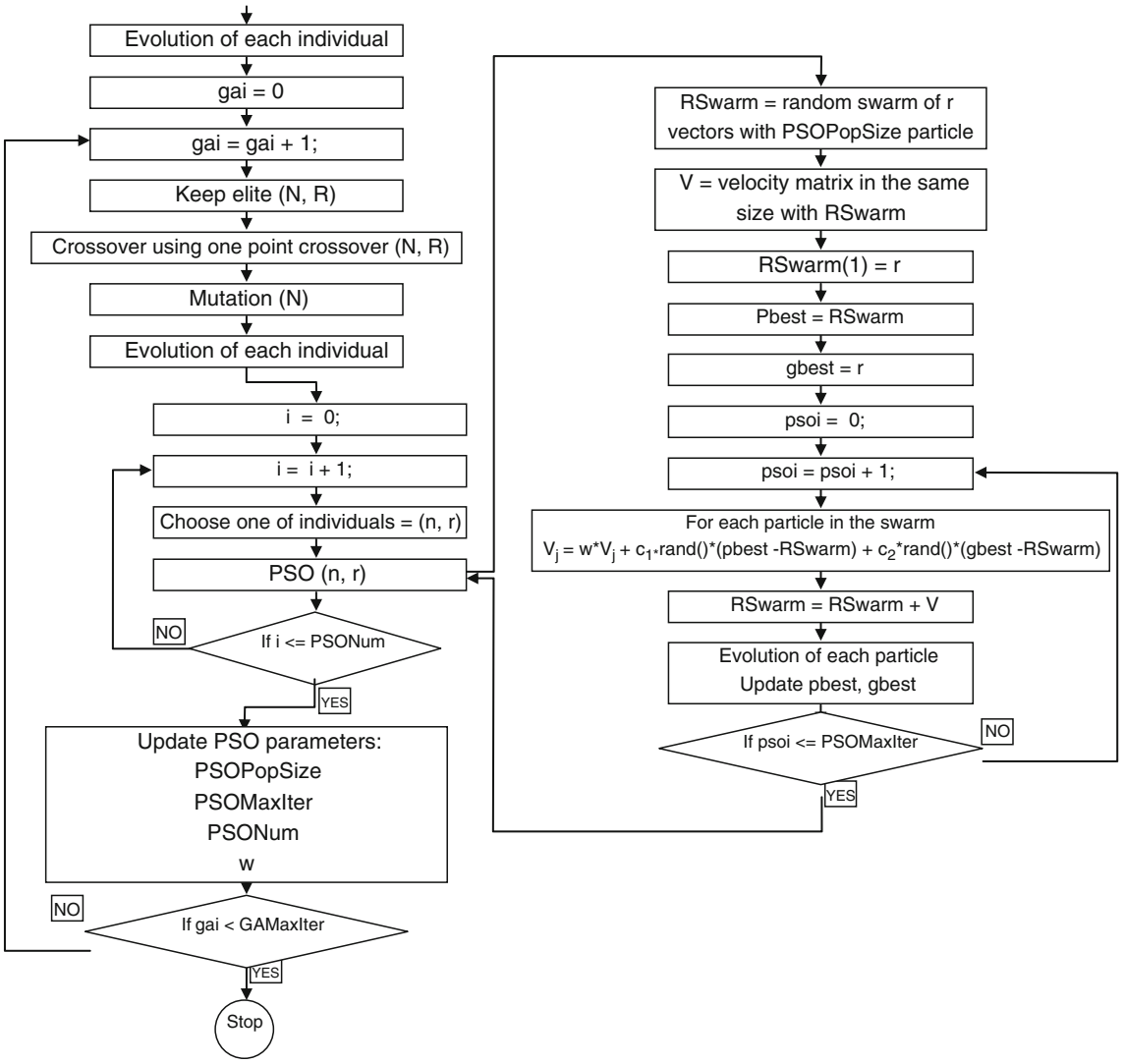


**Fig. 2** Procedure of hybrid approach

| | |
|---|---|
| $g_j$ | The $j$th constraint function |
| $n_i$ | The number of redundancy of subsystem $\boldsymbol{i}$ ($i=1,2,\ldots,m$) |
| $\boldsymbol{N}$ | $(n_1,\ldots,n_m)$ |
| $R_i$ | The reliability of subsystem $i$ |
| $R_s$ | The system reliability |
| $\alpha_i$ ,$\beta_i$ | Physical feature of each component |
| $R$ | Population of r |
| $\mathrm{GA}_{\mathrm{MaxIteration}}$ | Maximum iteration of GA |
| $\mathrm{PSO}_{\mathrm{MinPopSize}}$ | First population size in PSO |
| $\mathrm{PSO}_{\mathrm{MaxIteration}}$ | Last maximum iteration number of PSO |
| $\mathrm{PSO}_{\mathrm{IterationIncRate}}$ | Increasing rate of PSO maximum iteration |
| $\mathrm{PSO}_{\mathrm{NumMax}}$ | Maximum number of individuals that are affected by PSO |
| $\mathrm{PSO}_{\mathrm{PopSize}}$ | Current population size in PSO |
| $\mathrm{PSO}_{\mathrm{Num}}$ | Current number of individuals that effected by PSO |
| $\mathrm{crossover}_{\mathrm{Per}}$ | Crossover percentage |
| $c_2$ | Social intelligence coefficient |
| $w_{\mathrm{alpha}}$ | Decreasing rate of inertia |

$$\begin{aligned} &Max\, R_s = f(r,n) \\ &\quad s.t \\ &\quad g(r,n) \le b \\ &1 \le i \le m; 0 \le r_i \le 1; n_i \in Z^+ \end{aligned} \tag{1}$$

The inequality $g(r,n) \le b$ has been described in the literature [23, 25, 27], and it is formulated as follows:

$$g_1(n,r) = \sum_{i=1}^{m} w_i v_i^2 n_i^2 \le V \tag{2}$$

$$g_2(n,r) = \sum_{i=1}^{m} \alpha_i \left(\frac{-1000}{\ln r_i}\right)^{\beta_i} \left(n_i + \exp\left(\frac{n_i}{4}\right)\right) \le C \tag{3}$$

$$g_3(n,r) = \sum_{i=1}^{m} w_i n_i \exp\left(\frac{n_i}{4}\right) \le w \tag{4}$$

**Table 1** Input data for series system

| Stage | $10^5\alpha_i$ | $\beta_i$ | $v_i$ | $w_i$ | $V$ | $C$ | $W$ |
|---|---|---|---|---|---|---|---|
| 1 | 1.0 | 1.5 | 1 | 6 | 250 | 400 | 500 |
| 2 | 2.3 | 1.5 | 2 | 6 | | | |
| 3 | 0.3 | 1.5 | 3 | 8 | | | |
| 4 | 2.3 | 1.5 | 2 | 7 | | | |

**Table 2** Input data for series–parallel system

| Stage | $10^5\alpha_i$ | $\beta_i$ | $v_i$ | $w_i$ | $V$ | $C$ | $W$ |
|---|---|---|---|---|---|---|---|
| 1 | 2.500 | 1.5 | 2 | 3.5 | 180 | 175 | 100 |
| 2 | 1.450 | 1.5 | 4 | 4 | | | |
| 3 | 0.541 | 1.5 | 5 | 4 | | | |
| 4 | 0.541 | 1.5 | 8 | 3.5 | | | |
| 5 | 2.100 | 1.5 | 4 | 4.5 | | | |

$$1 \le i \le m; 0 \le r_i \le 1 \tag{5}$$

Constraint (2) indicates a combination of weight ($w$), redundancy allocation ($n$), and volume ($v$) in which $V$ represents the upper limit on the sum of the subsystems' products of volume and weight. According to constraint (3), the system overall cost is restricted to an upper limit denoted by $C$. The parameters $\alpha_i$ and $\beta_i$ are physical features of system components. In this constraint, 1,000 is the operating time during which the component must not fail. Finally, Eq. (4) constrains the system total weight. The term $exp(n_i/4)$ accounts for the interconnecting hardware. In this paper series, series–parallel, and complex (bridge) systems are used as three typical configurations of RRAP problems [4, 23–27, 31]. Noteworthy, in each case, the constraints are the same whereas the objective function is different, and it is defined due to system structure. Thus, Eqs. (6)–(8) state the overall reliability function for series, series–parallel, and complex (bridge) systems, respectively. In order to compare the results of the proposed GA–PSO approach with previous works, four subsystems are considered for series system, and five subsystems are considered for series–parallel and complex (bridge) systems.

$$f(n,r) = R_1R_2R_3R_4 \tag{6}$$

$$\begin{aligned} f(n,r) = {} & 1 - (1 - R_1R_2) \\ & \times (1 - (1 - (1 - R_3)(1 - R_4))R_5) \end{aligned} \tag{7}$$

**Table 3** Input data for complex system

| Stage | $10^5\alpha_i$ | $\beta_i$ | $v_i$ | $w_i$ | $V$ | $C$ | $W$ |
|---|---|---|---|---|---|---|---|
| 1 | 2.330 | 1.5 | 1 | 7 | 110 | 175 | 200 |
| 2 | 1.450 | 1.5 | 2 | 8 | | | |
| 3 | 0.541 | 1.5 | 3 | 8 | | | |
| 4 | 8.050 | 1.5 | 4 | 6 | | | |
| 5 | 1.950 | 1.5 | 2 | 9 | | | |

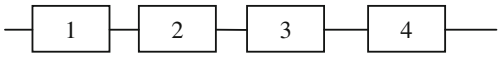


**Fig. 3** The structure of series system

$$f(n,r) = R_1R_2 + R_3R_4 + R_1R_4R_5 + R_2R_3R_5 - R_1R_2R_3R_4 - R_1R_2R_3R_5 - R_1R_3R_4R_5 - R_2R_3R_4R_5 - R_1R_2R_4R_5 + 2R_1R_2R_3R_4R_5 \quad (8)$$

## 3 Methodology: hybrid GA–PSO approach

In this paper a hybrid GA–PSO approach is applied to benefit from advantages of both genetic algorithm and particle swarm optimization. However, both models have strengths and weaknesses. In a GA if an individual is not selected the information contained by that individual is lost, but PSOs have memory. However, without a selection operator PSOs may waste resources on poor individuals. While GAs have trouble finding an exact solution and are best at reaching a global region, a PSO's group interactions boosts the search for an optimal solution, [29]. These key points and RRAP structure lead us to take the advantages of both algorithms into account, simultaneously. It means GA is applied for searching space of vector $n$, and PSO is used for searching space of vector $r$. GA makes a population of vectors ($\boldsymbol{n}$, $\boldsymbol{r}$), in which every pair constitutes a chromosome (Fig. 1). In this hybrid approach, vectors $\boldsymbol{n}$ and $\boldsymbol{r}$ are modified by GA; however, PSO only operates in direction of improving vector $\boldsymbol{r}$. In other words, PSO searches a better solution of $\boldsymbol{r}$ for an associated vector $\boldsymbol{n}$.

After forming a new generation in GA iterations, some chromosomes of new population are selected and then PSO is applied for each of them separately. The selected chromosomes of vectors $r$ will be altered in direction of fitness function incensement. PSO is not applied on the whole population, because it is very time consuming. Two critical questions, therefore, would necessarily be answered: How many and which chromosomes should be evolved in each GA renewal generation? We consider Eqs. (9) and (10) as the most comfortable and flexible answers to these questions ($GA_i$ denotes current GA iteration):

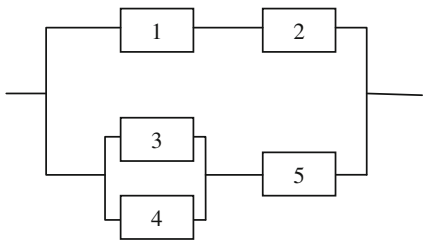


**Fig. 4** The structure of series–parallel system

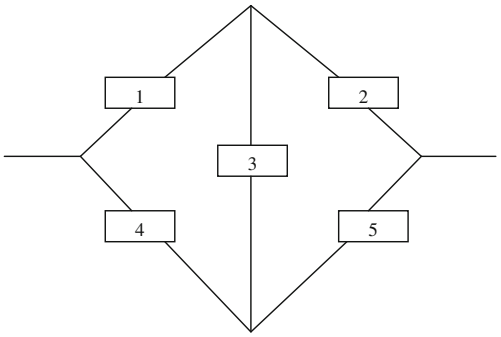


**Fig. 5** The structure of complex (bridge) system

$$PSO_{Num} = PSO_{NumMax} - \left(\frac{GA_i}{GA_{MaxIteration}}\right)^{PSO_{NumDecRate}} (PSO_{NumMax} - PSO_{NumMin}) \quad (9)$$

$$Max - Index = \left(1 - \frac{GA_i}{GA_{MaxIteration}}\right).\left(GA_{PopSize} - PSO_{Num}\right) + PSO_{Num} \quad (10)$$

The candidate chromosomes should be selected between the first and the Max-Index chromosome of the current

**Table 4** Control parameter for the proposed hybrid GA–PSO

| | |
|---|---|
| $GA_{PopSize}$ | 200 |
| $GA_{MaxIteration}$ | 100 |
| $PSO_{MaxPopSize}$ | 200 |
| $PSO_{MinPopSize}$ | 5 |
| $PSO_{PopIncRate}$ | 15 |
| $PSO_{MaxIteration}$ | 400 |
| $PSO_{MinIteration}$ | 5 |
| $PSO_{IterationIncRate}$ | 15 |
| $PSO_{NumMax}$ | 3 |
| $PSO_{NumMin}$ | 1 |
| $PSO_{NumDecRate}$ | 2 |
| $keep_{Per}$ | 10 % |
| $crossover_{Per}$ | 70 % |
| $c_1$ | 1 |
| $c_2$ | 1 |
| $w$ | 0.99 |
| $w_{final}$ | 0.8 |
| $w_{alpha}$ | 1.0021 |

**Table 5** Comparison of the best GA–PSO results with other algorithms—series system

| Parameter | Simulated annealing algorithms [25] | Genetic algorithm [26] | Particle swarm approach [27] | Proposed GA–PSO algorithm |
|---|---|---|---|---|
| $f(\boldsymbol{r}, \boldsymbol{n})$ | 0.99946800 | 0.99994500 | 0.99995300 | 0.99995467 |
| $n_1$ | 3 | 5 | 5 | 5 |
| $n_2$ | 6 | 5 | 6 | 5 |
| $n_3$ | 3 | 5 | 4 | 4 |
| $n_4$ | 5 | 5 | 5 | 6 |
| $r_1$ | 0.965593 | 0.895644 | 0.902231 | 0.901628 |
| $r_2$ | 0.760592 | 0.885878 | 0.856325 | 0.888230 |
| $r_3$ | 0.972646 | 0.912184 | 0.948145 | 0.948121 |
| $r_4$ | 0.804660 | 0.887785 | 0.883156 | 0.849921 |
| Slack[a] of the first constraint | 92 | 50 | 55 | 55 |
| Slack[a] of the second constraint | 70.733576− | 0.938000 | 0.975465 | 0.000006 |
| Slack[a] of the third constraint | 127.583189 | 28.803700 | 24.801882 | 15.363463 |

[a] Slack: the unused resource

population. In order to decrease time and memory usage, PSO iterations number and initial population size are not fixed, and have a direct relationship with current GA iteration ($GA_i$). The following equations satisfy this purpose:

$$PSO_{PopSize} = \left(\frac{GA_i}{GA_{MaxIteration}}\right)^{PSO_{PopIncRate}} \left(PSO_{MaxPopSize} - PSO_{MinPopSize}\right) + PSO_{MinPopSize} \tag{11}$$

$$PSO_{MaxIter} = \left(\frac{GA_i}{GA_{MaxIteration}}\right)^{PSO_{IterationIncRate}} \left(PSO_{MaxIteration} - PSO_{MinIteration}\right) + PSO_{MinIteration} \tag{12}$$

Number of the selected chromosomes will decrease when the algorithm iterations increase; however, PSO iterations number and population size will gradually increase to their final values. Lastly, inertia coefficient has been used to control particles velocity. It begins from an initial value

**Table 6** Comparison of the best GA–PSO results with other algorithms—series–parallel system

| Parameter | Surrogate-constraint algorithm [4] | Genetic algorithm [23] | Simulated annealing algorithms [25] | Proposed GA–PSO algorithm |
|---|---|---|---|---|
| $f(\boldsymbol{r}, \boldsymbol{n})$ | 0.99996875 | 0.99997418 | 0.99997631 | 0.99997665 |
| $n_1$ | 3 | 2 | 2 | 2 |
| $n_2$ | 3 | 2 | 2 | 2 |
| $n_3$ | 1 | 2 | 2 | 2 |
| $n_4$ | 2 | 2 | 2 | 2 |
| $n_5$ | 3 | 4 | 4 | 4 |
| $r_1$ | 0.838193 | 0.785452 | 0.812161 | 0.819640 |
| $r_2$ | 0.855065 | 0.842998 | 0.853346 | 0.845091 |
| $r_3$ | 0.878859 | 0.885333 | 0.897597 | 0.895482 |
| $r_4$ | 0.911402 | 0.917958 | 0.900710 | 0.895517 |
| $r_5$ | 0.850355 | 0.870318 | 0.866316 | 0.868430 |
| Slack[a] of the first constraint | 27 | 40 | 40 | 40 |
| Slack[a] of the second constraint | 0 | 1.194440 | 0.007300 | 0.000001 |
| Slack[a] of the third constraint | 7.518918 | 1.609289 | 1.609289 | 1.609289 |

[a] Slack: the unused resource

**Table 7** Comparison of the best GA–PSO results with other algorithms—complex system

| Parameter | Surrogate-constraint algorithm [4] | Genetic algorithm [23] | Simulated annealing algorithms [25] | Particle swarm approach [27] | Proposed GA–PSO algorithm |
|---|---|---|---|---|---|
| $f(\boldsymbol{r}, \boldsymbol{n})$ | 0.9997894 | 0.99987916 | 0.99988764 | 0.99988957 | 0.99988964 |
| $n_1$ | 3 | 3 | 3 | 3 | 3 |
| $n_2$ | 3 | 3 | 3 | 3 | 3 |
| $n_3$ | 2 | 3 | 3 | 2 | 2 |
| $n_4$ | 3 | 3 | 3 | 4 | 4 |
| $n_5$ | 2 | 1 | 1 | 1 | 1 |
| $r_1$ | 0.814483 | 0.814090 | 0.807263 | 0.826678 | 0.828134 |
| $r_2$ | 0.821383 | 0.864614 | 0.868116 | 0.857172 | 0.857831 |
| $r_3$ | 0.896151 | 0.890291 | 0.872862 | 0.914629 | 0.914192 |
| $r_4$ | 0.713091 | 0.701190 | 0.712673 | 0.648918 | 0.648069 |
| $r_5$ | 0.814091 | 0.734731 | 0.751034 | 0.715290 | 0.704476 |
| Slack[a] of the first constraint | 18 | 18 | 40 | 5 | 5 |
| Slack[a] of the second constraint | 1.854075 | 0.376347 | 0.007300 | 0.000339 | 0.000000 |
| Slack[a] of the third constraint | 4.264770 | 4.264770 | 1.609289 | 1.560466 | 1.560466 |

[a] Slack: the unused resource

and gradually decreases with $w_{\text{alpha}}$ factor; hence, the velocity of particles will decrease and PSO algorithm focuses on the solutions precision according to the following equations:

$$w = w_{alpha}.w \tag{13}$$

$$w_{alpha} = \left(\frac{w_{final}}{w}\right)^{\left(\frac{1}{GA_{MaxIteration}}\right)} \tag{14}$$

To sum up, a representation of the described procedure is presented in Fig. 2.

## 3.1 Genetic algorithm

GA is a powerful optimization method based on natural selection introduced by Holland [32]. The basic elements of a GA that must be specified for any given implementation are representation, population, evaluation, selection, operators and parameters. The procedure for implementing this algorithm is given briefly by the following steps:

Step 1: A random initial population is generated;
Step 2: The fitness function for each individual in the current population is evaluated;
Step 3: Predefined stopping criteria is checked;
Step 4: Reproduction, crossover, and mutation are performed on the current population;
Step 5: The new generation is formed from Step4' individuals. Steps 2 to 5 will be repeated. The problem is categorized as partition and allocation problems according to the Levitin category classification [33].

## 3.2 Particle swarm optimization

PSO algorithm first introduced by Kennedy and Eberhart [34, 35], and it is a population-based method modeled after

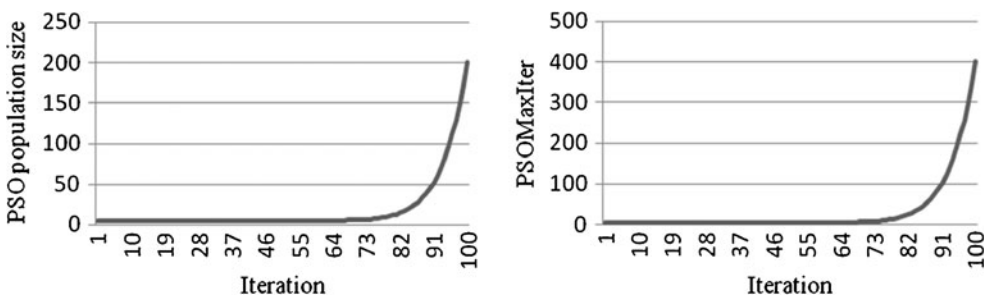


**Fig. 6** Population size and maximum number of iterations of PSO

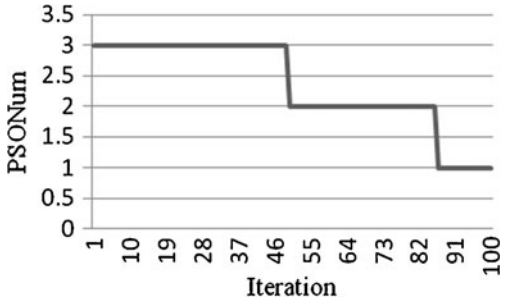


**Fig. 7** Number of individuals that affected by PSO ($PSO_{num}$) during Iterations

the simulation of the social behavior of bird flocks. The main purpose is to reach the best position of each particle. Each particle moves in direction of continuously updating velocity vector. All particles memorized two parameters: the local best (pbest), and the global best position (gbest). Each particle position is determined by considering these parameters. First, the velocity vector ($v_i$) is calculated through the previous velocity, the local best, and the global best position according to Eq. (15), and then the next particle position ($x_i$) is derived due to Eq. (16).

$$v_i(t) = w.v_i(t-1) + c_1.r_1(pbest - x_i(t-1)) + c_2.r_2(gbest - x_i(t-1)) \quad (15)$$

$$x_i(t) = x_i(t-1) + v_i(t-1) \quad (16)$$

Where $w$ is decreasing factor; $r_1$ and $r_2 \sim U(0, 1)$; $c_1$ and $c_2$ are individual and social coefficients, respectively.

## 4 Numerical results

In this section, the performance of the proposed hybrid GA–PSO approach in terms of the solution quality and computation time is evaluated through three well-known test problems. The parameters defining the characteristics of these series, series- parallel, and complex (bridge) systems are presented in Tables 1, 2, and 3 [4, 23, 25–27, 31]. The proposed GA–PSO approach is coded in MATLAB and performed on a core 2 Duo CPU processor with a 2.4 GHz and 4 GB of RAM (MATLAB codes are provided in appendices I, II, III). In each case, 50 independent runs were performed for the proposed optimization approach, and then the results compared with previous works in the literature. The first reliability–redundancy problem belongs to the category of series system, and the mathematical formulation is defined in Eqs. (2)–(5) and (6). The structure of the series system is illustrated in Fig. 3.

Tables 1, 2, and 3 show input data which are used for series, series–parallel, and complex systems, respectively.

The second and third problems are known as series–parallel and complex system. As stated before, the difference between these three systems is their objective functions definition. Figures 4 and 5 show a representation of both series–parallel and complex system.

Table 4 shows a set of control parameters which are adjusted in the proposed GA–PSO approach.

Tables 5, 6, and 7 compared the obtained results in this paper with those of previous works in the literature, and by individual GA and PSO based results. In these tables, the second row indicates the reliability objective function value. Rows containing $n_i$ and $r_i$ are associated with number and reliability of components in each subsystem, respectively. Finally, the slack rows imply the remainders of resources vector.

Table 5 reveals that the solutions of the series problem ($R_s$=0.99995467) found by GA–PSO are better than those reported by previous works [25–27].

Table 6 reports that the GA–PSO solutions for the series–parallel system ($R_s$=0.99997665) is better than previous works [4, 23, 25].

Finally, better performance of the proposed approach in comparison with previous works (4, 23, 25, and 27) for the represented complex (bridge) system is shown in Table 7 ($R_s$=0.99988964). According to the results, although in some cases reliability improvements are marginal, it is noteworthy that the proposed GA–PSO algorithm found

**Table 8** Overall reliability results of series, series–parallel, and complex systems

| | Problems | Maximum (best) | Minimum (worst) | Mean | Standard deviation |
|---|---|---|---|---|---|
| This paper | Series | 0.99995467 | 0.99995467 | 0.99995467 | 1.00E−16 |
| | Series–parallel | 0.99997665 | 0.99997015 | 0.99997613 | 4.533E−12 |
| | Complex | 0.99988964 | 0.99988935 | 0.999889623 | 2.2826E−11 |
| Coelho [27] | Series | 0.99995300 | 0.99963800 | 0.99990700 | 0.000011 |
| | Complex | 0.99988957 | 0.99987750 | 0.99988594 | 6.90E−07 |

**Table 9** Best results for series, series–parallel and complex system (50 runs)

| Parameter | Series | Series–parallel | Complex |
|---|---|---|---|
| $f(\boldsymbol{r},\boldsymbol{n})$ | 0.99995467 | 0.99997665 | 0.99988964 |
| $n_1$ | 5 | 2 | 3 |
| $n_2$ | 5 | 2 | 3 |
| $n_3$ | 4 | 2 | 2 |
| $n_4$ | 6 | 2 | 4 |
| $n_5$ | – | 4 | 1 |
| $r_1$ | 0.901628 | 0.819640 | 0.828134 |
| $r_2$ | 0.888230 | 0.845091 | 0.857831 |
| $r_3$ | 0.948121 | 0.895482 | 0.914192 |
| $r_4$ | 0.849921 | 0.895517 | 0.648069 |
| $r_5$ | – | 0.868430 | 0.704476 |
| Slack[a] of the first constraint | 55 | 40 | 5 |
| Slack[a] of the second constraint | 0.000006 | 0.000001 | 0.000000 |
| Slack[a] of the third constraint | 15.363463 | 1.609289 | 1.560466 |

[a]Slack: the unused resource

better results in all three series, series–parallel and complex (bridge) systems in comparison to the previously best-known solutions. The $PSO_{PopSize}$, $PSO_{MaxIter}$, and $PSO_{num}$ curves behavior during the running iterations of the proposed approach for all three series, series–parallel, and complex systems are displayed in Figs. 6 and 7. As it was expected, PSO current population size ($PSO_{PopSize}$) is increased gradually.

Table 8 presents statistical analysis of the GA–PSO approach including the best and the worst solutions, as well as the standard deviation for all three problems in comparison with those of Coelho [27]. According to the results, the standard deviation of the proposed approach is very small in 50 independent runs. Notably, it reveals that the worst solution obtained by the proposed GA–PSO approach is better than the best of Kim and Bae [25] and Yokota et al. [26]. Furthermore, the worst solution of the proposed GA–PSO approach in case of complex system is better than the best of similar previous works, except in that of Coelho [27]. Table 8 compares the results of the proposed GA–PSO approach with the best of the literature [27].

The best results of vectors $\boldsymbol{n}$, $\boldsymbol{r}$, and unused resources values as well as the overall reliability of the proposed approach are shown in Table 9.

**Table 10** Average and standard deviation time (50 runs)

| Problems | Average time (s) | Standard deviation (s) |
|---|---|---|
| Series | 3.14 | 0.06 |
| Series–parallel | 3.36 | 0.14 |
| Complex | 3.32 | 0.09 |

Table 10 shows both average and standard deviation of CPU time for implementing the hybrid GA–PSO approach in 50 independent runs for all three cases.

## 5 Conclusion

This paper addresses a hybrid approach based on GA and PSO to solve reliability redundancy allocation problem (RRAP). Three different cases including series, series–parallel, and complex (bridge) systems are considered. The objective of the proposed approach is to maximize overall system reliability subject to volume, cost, and weight constraints. To improve computation efficiency, a hybrid GA–PSO approach has been applied for searching solution space more efficiently. Moreover, the proposed approach benefits from advantages of both genetic algorithm and particle swarm optimization. As illustrated, the solutions found by the proposed hybrid approach are better than the results obtained by the other heuristic and meta-heuristic algorithms that are reported in the literature, for all series, series–parallel and complex (bridge) systems. Furthermore, the proposed approach shows lower variance (at most $e^{-10}$), as well as very low CPU time to derive the near global optimal solutions. The proposed GA–PSO could be recommended as a profitable solution method for MINLP problems.

**Acknowledgments** The authors are grateful for the valuable comments and suggestion from the respected reviewers. Their valuable comments and suggestions have enhanced the strength and significance of our paper. The authors would like to acknowledge the financial support of University of Tehran for this research under grant number 27775/01/06.

## Appendix I

```
function result = EvalRelOfSys
tic
%% initialization
% problem parameters
% N = matrix of n, (population of n)
% n = vector of ni
% ni = the number of redundancy of ith subsystem
% R = matrix of r, (population of r)
% r = vector of ri
% ri = reliability of component of ith subsystem
s = 4; %number of sub systems
rimin = 0.5;
rimax = 1-10^-6;
nimin = 1;
nimax = 10;
% algorithm parameters
GAPopSize = 200;
GAMaxIteration = 100;
PSOMaxPopSize = 200;
PSOMinPopSize = 5;
PSOPopIncRate = 15;
PSOMaxIteration = 400;
PSOMinIteration = 5;
PSOIterationIncRate = 15;
PSONumMax = 3;
PSONumMin = 2;
PSONumDecRate = 2;

PSOPopSize = PSOMaxPopSize;
PSOMaxIter = PSOMinIteration;
PSONum = PSONumMax;
keepPer = 0.1;
crossoverPer = 0.7;
keepNum = round(GAPopSize * keepPer);
crossoverNum = round(GAPopSize * crossoverPer);
c1 = 1;
c2 = 1;
w = 0.99;
wfinal = 0.8;
wAlpha = (wfinal/w)^(1/GAMaxIteration);
% Genetic initialization_____________________________________________________
N = round(rand(GAPopSize, s) * (nimax - nimin)) + nimin;
R = rand(GAPopSize, s).*(rimax-rimin) + rimin;
newN = zeros(GAPopSize, s);
newR = zeros(GAPopSize, s);
RelOfPop = EvalRelOfPop(N, R, GAPopSize);
gai = 0;
%% main cycle
while (gai < GAMaxIteration)
    gai = gai + 1;
    [RelOfPop, idx] = sort(RelOfPop, 'descend');
    %keep elite
    newN(1, :) = N(idx(1), :);
    newR(1, :) = R(idx(1), :);
    j = 1;
    i = 1;
```

```
while i < keepNum
   j = j + 1;
   use = true;
   for k= 1:i
      if ~any(N(idx(j), :) ~= newN(k, :))
         use = false;
         break;
      end
   end
   if use == true
      i = i + 1;
      newN(i, :) = N(idx(j), :);
      newR(i, :) = R(idx(j), :);
   end
end
%selection
slctd = randperm(GAPopSize);
slctd = slctd(1: crossoverNum);
%one point crossover
for i= 1:2:crossoverNum
   k = ceil(rand() * s);
   newN(i + keepNum, :) = [N(slctd(i), 1:k), N(slctd(i + 1), k+1:s)];
   newN(i + keepNum + 1, :) = [N(slctd(i + 1), 1:k), N(slctd(i), k+1:s)];
   newR(i + keepNum, :) = [R(slctd(i), 1:k), R(slctd(i + 1), k+1:s)];
   newR(i + keepNum + 1, :) = [R(slctd(i + 1), 1:k), R(slctd(i), k+1:s)];
end
%mutation
for i= 1 + keepNum + crossoverNum: GAPopSize
   p = ceil(rand() * GAPopSize);
   newN(i, :) = N(p, :);
   newR(i, :) = R(p, :);
   choice = round(rand() * 3);
   switch choice
      case 0
         mask = round(rand(1, s) * 2 - 1);
         newN(i, :) = newN(i, :) + mask(1, :);
      case 1
         mask = round(rand(1, s));
         mask(mask == 0) = -1;
         newN(i, :) = newN(i, :) + mask(1, :);
      case 2
         k = ceil(rand() * s);
         newN(i, k) = newN(i, k) + 1;
      case 3
         k = ceil(rand() * s);
         newN(i, k) = newN(i, k) - 1;
      otherwise
   end
end
newN(newN < nimin) = nimax;
```

```
    newN(newN > nimax) = nimin;
    N = newN;
    R = newR;
    RelOfPop = EvalRelOfPop(N, R, GAPopSize);
    domin = (1 - gai/GAMaxIteration) * (GAPopSize - PSONum) + PSONum;
    selectedn = randperm(round(domin));
    sn = selectedn(1:PSONum);
    %PSO_____________________________________________________________________
    for i= 1:PSONum
      RSwarmVel = rand(PSOPopSize, s)./50;
      mask = randint(PSOPopSize, s);
      RSwarmVel(mask == 0) = -RSwarmVel(mask == 0);
      RSwarm = rand(PSOPopSize, s).*(rimax-rimin) + rimin;
      RSwarm(1, :) = R(sn(i), :);
      ROfPSOPop = EvalRelOfSwarm(N(sn(i), :), RSwarm, PSOPopSize);
      gbestCost = RelOfPop(sn(i));
      gbest = RSwarm(1, :);
      pbestCost = ROfPSOPop;
      pbest = RSwarm;
      psoi = 0;
      while (psoi < PSOMaxIter)
        psoi = psoi + 1;
        r1 = rand(PSOPopSize, s);
        r2 = rand(PSOPopSize, s);
        RSwarmVel = (w*RSwarmVel + ...
          c1*r1.*(pbest - RSwarm) + ...
          c2*r2.*(ones(PSOPopSize, 1)*gbest - RSwarm));
        RSwarm = RSwarm + RSwarmVel;
        RSwarm(RSwarm < rimin) = rimin;
        RSwarm(RSwarm > rimax) = rimax;
        ROfPSOPop = EvalRelOfSwarm(N(sn(i), :), RSwarm, PSOPopSize);
        for k = 1:PSOPopSize
          if ROfPSOPop(k) > pbestCost(k)
            pbest(k, :) = RSwarm(k, :);
            pbestCost(k) = ROfPSOPop(k);
          end
        end
        [bestOfIter, gbestIdx]  = max(ROfPSOPop);
        if bestOfIter > gbestCost
          gbestCost = bestOfIter;
          gbest = RSwarm(gbestIdx, :);
        end
      end
      R(sn(i), :) = gbest;
      RelOfPop(sn(i)) = gbestCost;
    end
    %update PSO parameters
    PSOPopSize = round(((gai/GAMaxIteration)^PSOPopIncRate)*(PSOMaxPopSize-
  PSOMinPopSize)+PSOMinPopSize);
```

```
    PSOMaxIter = round(((gai/GAMaxIteration)^PSOIterationIncRate)*(PSOMaxIteration-
PSOMinIteration)+ PSOMinIteration);
    PSONum = PSONumMax - round(((gai/GAMaxIteration)^PSONumDecRate) *
(PSONumMax - PSONumMin));
    w = wAlpha * w;
end
%% print results
totalTime = toc;
[RelOfPop, idx] = sort(RelOfPop, 'descend');
N = N(idx, :);
R = R(idx, :);
result = RelOfPop(i);
bestnf = N(1, :);
bestrf = R(1, :);
[s1, s2, s3] = EvalSlacks(bestnf, bestrf);
%{
fprintf('\n population data\n [i] [N] [R] [system reliability] \n');
for i= 1:GAPopSize
    fprintf('%5d', i);
    fprintf('%5d', N(i,:));
    fprintf('%10.6f', R(i,:));
    fprintf('%20.8f', RelOfPop(i));
    fprintf('\n');
end
%}
fprintf('\nfinal solution = \n');
fprintf('%5d', bestnf);
fprintf('%10.6f', bestrf);
fprintf('%20.8f', RelOfPop(1));
fprintf('\n');
fprintf('\nslack 1 = %10.6f', s1);
fprintf('\nslack 2 = %10.6f', s2);
fprintf('\nslack 3 = %10.6f', s3);
fprintf('\nTotal time = %5.8f second\n', totalTime);
end
```

## Appendix II

```
function result = EvalRelOfSys
tic
%% initialization
% problem parameters
% N = matrix of n, (population of n)
% n = vector of ni
% ni = the number of redundancy of ith subsystem
% R = matrix of r, (population of r)
% r = vector of ri
% ri = reliability of component of ith subsystem
s = 5; %number of sub systems
rimin = 0.5;
rimax = 1-10^-6;
nimin = 1;
nimax = 5;
% algorithm parameters
GAPopSize = 200;
GAMaxIteration = 100;
PSOMaxPopSize = 200;
PSOMinPopSize = 5;
PSOPopIncRate = 15;
PSOMaxIteration = 400;
PSOMinIteration = 5;
PSOIterationIncRate = 15;
PSONumMax = 3;
PSONumMin = 2;
PSONumDecRate = 2;
PSOPopSize = PSOMaxPopSize;
PSOMaxIter = PSOMinIteration;
PSONum = PSONumMax;
keepPer = 0.1;
crossoverPer = 0.7;
keepNum = round(GAPopSize * keepPer);
crossoverNum = round(GAPopSize * crossoverPer);
c1 = 1;
c2 = 1;
w = 0.99;
wfinal = 0.8;
wAlpha = (wfinal/w)^(1/GAMaxIteration);
% Genetic initialization_____________________________________________________
N = round(rand(GAPopSize, s) * (nimax-nimin)) + nimin;
R = rand(GAPopSize, s).*(rimax-rimin) + rimin;
newN = zeros(GAPopSize, s);
newR = zeros(GAPopSize, s);
RelOfPop = EvalRelOfPop(N, R, GAPopSize);
gai = 0;
%% main cycle
while (gai < GAMaxIteration)
   gai = gai + 1;
   [RelOfPop, idx] = sort(RelOfPop, 'descend');
   %keep elite
   newN(1, :) = N(idx(1), :);
   newR(1, :) = R(idx(1), :);
   j = 1;
   i = 1;
   while i < keepNum
      j = j + 1;
      use = true;
      for k= 1:i
         if ~any(N(idx(j), :) ~= newN(k, :))
            use = false;
            break;
         end
```

```
    end
    if use == true
        i = i + 1;
        newN(i, :) = N(idx(j), :);
        newR(i, :) = R(idx(j), :);
    end
end
%selection
slctd = randperm(GAPopSize);
slctd = slctd(1: crossoverNum);
%one point crossover
for i= 1:2:crossoverNum
    k = ceil(rand() * s);
    newN(i + keepNum, :) = [N(slctd(i), 1:k), N(slctd(i + 1), k+1:s)];
    newN(i + keepNum + 1, :) = [N(slctd(i + 1), 1:k), N(slctd(i), k+1:s)];
    newR(i + keepNum, :) = [R(slctd(i), 1:k), R(slctd(i + 1), k+1:s)];
    newR(i + keepNum + 1, :) = [R(slctd(i + 1), 1:k), R(slctd(i), k+1:s)];
end
%mutation
for i= 1 + keepNum + crossoverNum: GAPopSize
    p = ceil(rand() * GAPopSize);
    newN(i, :) = N(p, :);
    newR(i, :) = R(p, :);

    choice = round(rand() * 3);
    switch choice
        case 0
            mask = round(rand(1, s) * 2-1);
            newN(i, :) = newN(i, :) + mask(1, :);
        case 1
            mask = round(rand(1, s));
            mask(mask == 0) =-1;
            newN(i, :) = newN(i, :) + mask(1, :);
        case 2
            k = ceil(rand() * s);
            newN(i, k) = newN(i, k) + 1;
        case 3
            k = ceil(rand() * s);
            newN(i, k) = newN(i, k) -1;
        otherwise
    end
end
newN(newN < nimin) = nimax;
newN(newN > nimax) = nimin;
N = newN;
R = newR;
RelOfPop = EvalRelOfPop(N, R, GAPopSize);
domin = (1-gai/GAMaxIteration) * (GAPopSize -PSONum) + PSONum;
selectedn = randperm(round(domin));
sn = selectedn(1:PSONum);
```

```
%PSO_____________________________________________________________________
for i= 1:PSONum
   RSwarmVel = rand(PSOPopSize, s)./50;
   mask = randint(PSOPopSize, s);
   RSwarmVel(mask == 0) = -RSwarmVel(mask == 0);
   RSwarm = rand(PSOPopSize, s).*(rimax-rimin) + rimin;
   RSwarm(1, :) = R(sn(i), :);
   ROfPSOPop = EvalRelOfSwarm(N(sn(i), :), RSwarm, PSOPopSize);
   gbestCost = RelOfPop(sn(i));
   gbest = RSwarm(1, :);
   pbestCost = ROfPSOPop;
   pbest = RSwarm;
   psoi = 0;
   while (psoi < PSOMaxIter)
      psoi = psoi + 1;
      r1 = rand(PSOPopSize, s);
      r2 = rand(PSOPopSize, s);
      RSwarmVel = (w*RSwarmVel + ...
         c1*r1.*(pbest - RSwarm) + ...
         c2*r2.*(ones(PSOPopSize, 1)*gbest - RSwarm));
      RSwarm = RSwarm + RSwarmVel;
      RSwarm(RSwarm < rimin) = rimin;
      RSwarm(RSwarm > rimax) = rimax;

      ROfPSOPop = EvalRelOfSwarm(N(sn(i), :), RSwarm, PSOPopSize);
      for k = 1:PSOPopSize
         if ROfPSOPop(k) > pbestCost(k)
            pbest(k, :) = RSwarm(k, :);
            pbestCost(k) = ROfPSOPop(k);
         end
      end
      [bestOfIter, gbestIdx]  = max(ROfPSOPop);
      if bestOfIter > gbestCost
         gbestCost = bestOfIter;
         gbest = RSwarm(gbestIdx, :);
      end
   end
   R(sn(i), :) = gbest;
   RelOfPop(sn(i)) = gbestCost;
end
%update PSO parameters
PSOPopSize = round(((gai/GAMaxIteration)^PSOPopIncRate)*(PSOMaxPopSize-
PSOMinPopSize)+PSOMinPopSize);
PSOMaxIter = round(((gai/GAMaxIteration)^PSOIterationIncRate)*(PSOMaxIteration-
PSOMinIteration)+ PSOMinIteration);
PSONum = PSONumMax - round(((gai/GAMaxIteration)^PSONumDecRate) *
(PSONumMax - PSONumMin));
w = wAlpha * w;
end
%% print results
```

```
totalTime = toc;
[RelOfPop, idx] = sort(RelOfPop, 'descend');
N = N(idx, :);
R = R(idx, :);
result = RelOfPop(i);
bestnf = N(1, :);
bestrf = R(1, :);
[s1, s2, s3] = EvalSlacks(bestnf, bestrf);
%{
fprintf('\n population data\n [i] [N] [R] [system reliability] \n');
for i= 1:GAPopSize
   fprintf('%5d', i);
   fprintf('%5d', N(i,:));
   fprintf('%10.6f', R(i,:));
   fprintf('%20.8f', RelOfPop(i));
   fprintf('\n');
end
%}
fprintf('\nfinal solution = \n');
fprintf('%5d', bestnf);
fprintf('%10.6f', bestrf);
fprintf('%20.8f', RelOfPop(1));
fprintf('\n');

fprintf('\nslack 1 = %10.6f', s1);
fprintf('\nslack 2 = %10.6f', s2);
fprintf('\nslack 3 = %10.6f', s3);
fprintf('\nTotal time = %5.8f second\n', totalTime);
end
```

## Appendix III

```
function result = EvalRelOfSys
tic
%% initialization
% problem parameters
% N = matrix of n, (population of n)
% n = vector of ni
% ni = the number of redundancy of ith subsystem
% R = matrix of r, (population of r)
% r = vector of ri
% ri = reliability of component of ith subsystem
s = 5; %number of sub systems
rimin = 10^-6;
rimax = 1 - 10^-6;
nimin = 1;
nimax = 10;
% algorithm parameters
GAPopSize = 200;
GAMaxIteration = 100;
PSOMaxPopSize = 200;
PSOMinPopSize = 5;
PSOPopIncRate = 15;
PSOMaxIteration = 400;
PSOMinIteration = 5;
PSOIterationIncRate = 15;
PSONumMax = 3;
PSONumMin = 2;
PSONumDecRate = 2;
PSOPopSize = PSOMaxPopSize;
PSOMaxIter = PSOMinIteration;
PSONum = PSONumMax;
keepPer = 0.1;
crossoverPer = 0.7;
keepNum = round(GAPopSize * keepPer);
crossoverNum = round(GAPopSize * crossoverPer);
c1 = 1;
c2 = 1;
w = 0.99;
wfinal = 0.8;
wAlpha = (wfinal/w)^(1/GAMaxIteration);
% Genetic initialization_____________________________________________________
N = round(rand(GAPopSize, s) * (nimax - nimin)) + nimin;
R = rand(GAPopSize, s).*(rimax-rimin) + rimin;

newN = zeros(GAPopSize, s);
newR = zeros(GAPopSize, s);
RelOfPop = EvalRelOfPop(N, R, GAPopSize);
gai = 0;
%% main cycle
while (gai < GAMaxIteration)
   gai = gai + 1;
   [RelOfPop, idx] = sort(RelOfPop, 'descend');
   %keep elite
   newN(1, :) = N(idx(1), :);
   newR(1, :) = R(idx(1), :);
   j = 1;
   i = 1;
   while i < keepNum
      j = j + 1;
      use = true;
      for k= 1:i
         if ~any(N(idx(j), :) ~= newN(k, :))
            use = false;
            break;
         end
      end
      if use == true
         i = i + 1;
         newN(i, :) = N(idx(j), :);
         newR(i, :) = R(idx(j), :);
```

```
    end
end
%selection
slctd = randperm(GAPopSize);
slctd = slctd(1: crossoverNum);
%one point crossover
for i= 1:2:crossoverNum
    k = ceil(rand() * s);
    newN(i + keepNum, :) = [N(slctd(i), 1:k), N(slctd(i + 1), k+1:s)];
    newN(i + keepNum + 1, :) = [N(slctd(i + 1), 1:k), N(slctd(i), k+1:s)];
    newR(i + keepNum, :) = [R(slctd(i), 1:k), R(slctd(i + 1), k+1:s)];
    newR(i + keepNum + 1, :) = [R(slctd(i + 1), 1:k), R(slctd(i), k+1:s)];
end
%mutation
for i= 1 + keepNum + crossoverNum: GAPopSize
    p = ceil(rand() * GAPopSize);
    newN(i, :) = N(p, :);
    newR(i, :) = R(p, :);
    choice = round(rand() * 3);
    switch choice
        case 0
            mask = round(rand(1, s) * 2 - 1);
            newN(i, :) = newN(i, :) + mask(1, :);
        case 1
            mask = round(rand(1, s));
            mask(mask == 0) = -1;
            newN(i, :) = newN(i, :) + mask(1, :);
        case 2
            k = ceil(rand() * s);
            newN(i, k) = newN(i, k) + 1;
        case 3
            k = ceil(rand() * s);
            newN(i, k) = newN(i, k) - 1;
        otherwise
    end
end
newN(newN < nimin) = nimax;
newN(newN > nimax) = nimin;
N = newN;
R = newR;
RelOfPop = EvalRelOfPop(N, R, GAPopSize);
domin = (1 - gai/GAMaxIteration) * (GAPopSize - PSONum) + PSONum;
selectedn = randperm(round(domin));
sn = selectedn(1:PSONum);
%PSO_____________________________________________________________________________
for i= 1:PSONum
    RSwarmVel = rand(PSOPopSize, s)./50;
    mask = randint(PSOPopSize, s);
    RSwarmVel(mask == 0) = -RSwarmVel(mask == 0);
    RSwarm = rand(PSOPopSize, s).*(rimax-rimin) + rimin;
```

```
        RSwarm(1, :) = R(sn(i), :);
        ROfPSOPop = EvalRelOfSwarm(N(sn(i), :), RSwarm, PSOPopSize);
        gbestCost = RelOfPop(sn(i));
        gbest = RSwarm(1, :);
        pbestCost = ROfPSOPop;
        pbest = RSwarm;
        psoi = 0;
        while (psoi < PSOMaxIter)
          psoi = psoi + 1;
          r1 = rand(PSOPopSize, s);
          r2 = rand(PSOPopSize, s);
          RSwarmVel = (w*RSwarmVel + ...
            c1*r1.*(pbest - RSwarm) + ...
            c2*r2.*(ones(PSOPopSize, 1)*gbest - RSwarm));
          RSwarm = RSwarm + RSwarmVel;
          RSwarm(RSwarm < rimin) = rimin;
          RSwarm(RSwarm > rimax) = rimax;
          ROfPSOPop = EvalRelOfSwarm(N(sn(i), :), RSwarm, PSOPopSize);
          for k = 1:PSOPopSize
            if ROfPSOPop(k) > pbestCost(k)
              pbest(k, :) = RSwarm(k, :);
              pbestCost(k) = ROfPSOPop(k);
            end
          end
          [bestOfIter, gbestIdx]  = max(ROfPSOPop);
          if bestOfIter > gbestCost
            gbestCost = bestOfIter;
            gbest = RSwarm(gbestIdx, :);
          end
        end
        R(sn(i), :) = gbest;
        RelOfPop(sn(i)) = gbestCost;
      end
      %update PSO parameters
      PSOPopSize = round(((gai/GAMaxIteration)^PSOPopIncRate)*(PSOMaxPopSize-
PSOMinPopSize)+PSOMinPopSize);
      PSOMaxIter = round(((gai/GAMaxIteration)^PSOIterationIncRate)*(PSOMaxIteration-
PSOMinIteration)+ PSOMinIteration);
      PSONum = PSONumMax - round(((gai/GAMaxIteration)^PSONumDecRate) *
(PSONumMax - PSONumMin));
      w = wAlpha * w;
    end
    %% print results
    totalTime = toc;
    [RelOfPop, idx] = sort(RelOfPop, 'descend');
    N = N(idx, :);
    R = R(idx, :);
    result = RelOfPop(i);
    bestnf = N(1, :);
    bestrf = R(1, :);
```

```
[s1, s2, s3] = EvalSlacks(bestnf, bestrf);
%{
fprintf('\n population data\n [i] [N] [R] [system reliability] \n');
for i= 1:GAPopSize
   fprintf('%5d', i);
   fprintf('%5d', N(i,:));
   fprintf('%10.6f', R(i,:));
   fprintf('%20.8f', RelOfPop(i));
   fprintf('\n');
end
%}
fprintf('\nfinal solution = \n');
fprintf('%5d', bestnf);
fprintf('%10.6f', bestrf);
fprintf('%16.8f', RelOfPop(1));
fprintf('\n');
fprintf('\nslack 1 = %10.6f', s1);
fprintf('\nslack 2 = %10.6f', s2);
fprintf('\nslack 3 = %10.6f', s3);
fprintf('\nTotal time = %5.8f second\n', totalTime);
end
```

## References


1. Kulturel-Konak S, Smith AE, Coit DW (2003) Efficiently solving the redundancy allocation problem using tabu search. IIE Trans 35 (6):515–526
2. Kuo W, Wan R (2007) Recent advances in optimal reliability allocation. Comput Intell Reliab Eng Stud Comput Intell 39:1–36
3. Chern MS (1992) On the computational complexity of reliability redundancy allocation in a series system. Oper Res Lett 11(5):309–315
4. Hikita M, Nakagawa Y, Narihisa H (1992) Reliability optimization of systems by a surrogate constraints algorithm. IEEE Trans Reliab 41(3):473–480
5. Levitin G (2007) Computational intelligence in reliability engineering: evolutionary techniques in reliability analysis and optimization. Springer, New York
6. Nahas N, Nourelfath M (2005) Ant system for reliability optimization of a series system with multiple-choice and budget constraints. Reliab Eng Syst Saf 87:1–12
7. Shelokar PS, Jayaraman VK, Kulkarni BD (2002) Ant algorithm for single and multi objective reliability optimization problems. Qual Reliab Eng Int 18(6):497–514
8. Suman B (2003) Simulated annealing-based multi-objective algorithm and their application for system reliability. Eng Optim 35(4):391–416
9. Sung CS, Cho YK (2000) Reliability optimization of a series system with multiple-choice and budget constraints. Eur J Oper Res 48:158–171
10. Yun WY, Kim JW (2004) Multi-level redundancy optimization in series systems. Comput Ind Eng 46(2):337–346
11. Zafiropoulos EP, Dialynas EN (2004) Reliability and cost optimization of electronic devices considering the component failure rate uncertainty. Reliab Eng Syst Saf 84:271–284
12. Zhao R, Liu B (2004) Redundancy optimization problems with uncertainty of combining randomness and fuzziness. Eur J Oper Res 157:716–735
13. Altumi AA, Philipose AM, Taboun SM (2000) Reliability optimisation of FMS with spare tooling. Int J Adv Manuf Technol 16:551–558
14. Lee H, Kuo W, Ha C (2003) Comparison of max–min approach and NN method for reliability optimization of series–parallel system. J Syst Sci Syst Eng J Sci Syst Eng 12(1):39–48
15. Lillo Rosa E, Neuts Marcel F (2000) Empirical optimization in a reliability problem. Methodol Comput Appl Probab 2(4):413–424
16. Pan Z, Chen L, Zhang G (2007) A neural network method for reliability optimizations of complex systems. Wuhan Univ J Natl Sci 12(1):139–142
17. Li D, Sun X, Mckinnon K (2005) An exact solution method for reliability optimization in complex systems. Ann Oper Res 133:129–148
18. Norkina VI, Onishchenko BO (2008) Reliability optimization of a complex system by the stochastic branch and bound method. Cybern Syst Anal 44(3):418–428
19. Valdebenito MA, Schuëller GI (2010) A survey on approaches for reliability-based optimization. Struct Multidiscip Optim 42:645–663
20. Dengiz B, Altiparmak F, Smith AE (1997) Local search genetic algorithm for optimal design of reliable networks. IEEE Trans Evol Comput 1(3):179–188
21. Tavakkoli-Moghaddama R, Safari J, Sassani F (2008) Reliability optimization of series–parallel systems with a choice of redundancy strategies using a genetic algorithm. Reliab Eng Syst Saf 93:550–556
22. Ye Z, Li Z, Min X (2010) Some improvements on adaptive genetic algorithms for reliability-related applications. Reliab Eng Syst Saf 95:120–126

23. Hsieh YC, Chen TC, Bricker DL (1998) Genetic algorithms for reliability design problems. Microelectron Reliab 38:1599–1605
24. Kuo W, Prasad VR, Tillman FA, Hwang CL (2001) Optimal reliability design: fundamentals and applications. Cambridge University Press, Cambridge
25. Kim HG, Bae C (2006) Reliability-redundancy optimization using simulated annealing algorithms. J Qual Maint Eng 12(4):354–363
26. Yokota T, Gen M, Li YX (1996) Genetic algorithm for non-linear mixed-integer programming and its applications. Comput Ind Eng 30(4):905–917
27. Coelho LDS (2009) An efficient particle swarm approach for mixed-integer programming in reliability–redundancy optimization applications. Reliab Eng Syst Saf 94:830–837
28. Angeline PJ (1998) Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences. In V. William Porto and et al., editors, Evolutionary Programming, vol. 1447 of Lecture Notes in Computer Science, pp. 601–610. Springer, 1998
29. Eberhart RC, Shi Y (1998) Comparison between genetic algorithms and particle swarm optimization. In et. al. V. William Porto, editor, Evolutionary Programming, vol. 1447 of Lecture Notes in Computer Science, pp. 611–616. Springer, 1998
30. Settles M, Soule T (2005) Breeding swarms: a GA/PSO hybrid. Proceedings of the 2005 conference on Genetic and evolutionary computation, pp. 161–168
31. Xu Z, Kuo W, Lin HH (1990) Optimization limits in improving system reliability. IEEE Trans Reliab 39(1):51–60
32. Holland J (1975) Adaptation in natural and artificial systems. University of Michigan Press
33. Levitin G (2005) The universal generating function in reliability analysis and optimization. Springer, London
34. Eberhart RC, Kennedy JF (1995) A new optimizer using particle swarm theory. Proceeding of the 6th international symposium on micro machine and human science, pp. 39–43
35. Kennedy JF, Eberhart RC (1995) Particle swarm optimization. Proc IEEE Int Conf Neural Netw 4:1942–1948