Volume 2, Number 2, 2013

International Journal of Research in Industrial Engineering

journal homepage: www.nvlscience.com/index.php/ijrie

A Hybrid Genetic Algorithm/Simulation Approach for Redundancy Optimization with Objective of Maximizing Mean Lifetime and Considering Component Selection

H. Karimi[1], A.A. Najafi[2,*]

[1]Faculty of Industrial and Mechanical Engineering, Qazvin Branch, Islamic Azad University, Qazvin, Iran.
[2]Faculty of Industrial Engineering, K.N. Toosi University of Technology, Tehran, Iran.

A R T I C L E I N F O

Article history :
Received:
February 5, 2013
Revised:
April 19, 2013
Accepted:
June 22, 2013

Keywords :
Reliability, redundancy allocation problem, Monte Carlo method, genetic algorithm, means time to failure.

A B S T R A C T

In this paper, we consider a reliability redundancy optimization problem in a series-parallel type system employing the redundancy strategy of cold-standby. The problem consists of two parts component selection and determination of redundancy level of each component—which need to be solved so that the mean lifetime of the system can be maximized. The redundancy allocation problem is non-deterministic polynomial-time hard and is solved by a combined genetic algorithm - simulation approach. Finally, this algorithm is tested on 33 benchmark problems.

1. Introduction

Designers are constantly attempting to improve the reliability of manufacturing systems to increase economic and production efficiencies. There are two ways to accomplish this. One is to improve the reliability of the system components, and the other is to add more components to the system. It has been shown that the first approach is not very effective. In the second approach, appropriately called the redundancy allocation problem (RAP), an optimal combination of the components and the redundancy level of each component must be determined. The only drawback of this approach is that it increases the weight, cost, and volume of the system.

There are four configurations for connecting the components in a system. They are series, parallel, series-parallel, and parallel-series. In this paper, we are considering the redundancy allocation problem for a series-parallel system. In this configuration (Figure 1), several subsystems are connected to each other in series, whereas the components in each subsystem are connected in parallel.

*Corresponding author
E-mail address: aanajafi@kntu.ac.ir

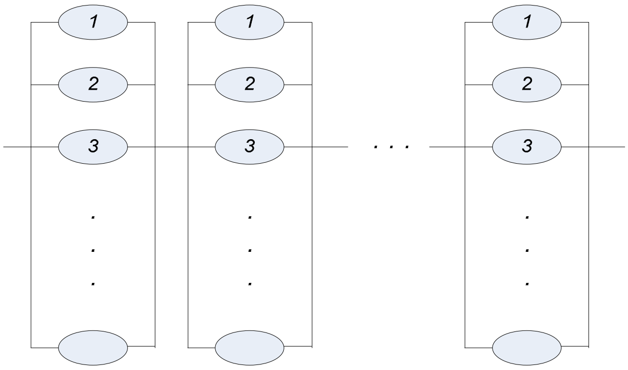

Figure 1. Series-parallel system

There are two types of redundancy strategies: active and standby. In the active strategy, all redundant components operate simultaneously from time zero, whereas in the standby strategy, only the needed components operate and the redundant components are idle until the active component fails.

There are three categories of standby redundancy: cold, warm, and hot. In cold-standby, components do not fail before operating. In warm standby, idle components can fail, but the probability of their failure is less than that of the failure of the operating component. In hot standby, the operating components and the idle components follow the same pattern of failure.

In a system with cold-standby redundancy configuration, a detection mechanism senses the failure of a component, and the system switches over to a safe component, if available. The detection and switching to the redundant component can be carried out in two ways. In the first scenario, the system is continually monitored for failure of detection. In this case, the failure of the detection/switching mechanism can occur at any time. In the second scenario, the failure of the detection/switching mechanism occurs only when a switching is required. In this scenario, the probability of the detection and the switching mechanism working properly and activating a redundant component in the subsystem i is denoted by ρi [1].

An RAP can be solved by an exact or a meta-heuristic method. The exact method includes dynamic programming [2, 3, 4, and 5], integer programming [6, 7, 8, and 9], and mixed-integer nonlinear programming [10]. Coit [11] proposed an integer programming solution for an RAP such that the reliability of a system employing the cold-standby redundancy strategy was maximized. Chern [12] demonstrated that even a simple RAP with linear constraints in a series system is non-deterministic polynomial-time hard. This persuaded researchers to develop heuristic and meta-heuristic approaches to solve RAP. These approaches can obtain near-optimal solutions within a reasonable computational time.

In previous studies, meta-heuristic methods, such as the genetic algorithm (GA) [13 and 14], variable neighborhood descent algorithm [15] and ant colony optimization [16 and 17] have been used to maximize system reliability for an RAP.

In all the above mentioned studies, the researchers intended to maximize the reliability of the system for a specific period of time. In contrast, this study attempts to maximize the mean lifetime of the system. The only work which has considered mean lifetime in the redundancy allocation problem was proposed by [18]. They considered mean time to failure (MTTF) of the system as mean lifetime index in the objective function. But, the component selection was not allowed in this model. Considering the fact the in real-world problem, the component selection is very important for a system designer. Hence, in this research, we consider a redundancy allocation problem in which the component selection is allowed. The Monte Carlo simulation is used to calculate the MTTF, and the GA is applied to solve the RAP in a system employing the cold-standby redundancy strategy; switch failure is only possible when switching is required. The probability that switching occurs properly in all subsystems is denoted by ρ . The components are assumed to be not repairable.

In each subsystem, there are mi options for selecting components, and the components within each subsystem must be of the same type. The time to failure of the components follows Erlang distribution function. Each available component has different levels of cost, weight, and reliability. There are redundancy level, cost, and weight constraints associated with components, and the problem is to select the component and identify the redundancy levels for each subsystem to maximize the MTTF of the system.

This paper is organized as follows. Section 2 contains the mathematical formulation for the problem. A genetic algorithm combined with the Monte Carlo method is presented in section 3. Section 4 presents the experimental results, and finally, section 5 states the conclusion and suggests possible future researches on this topic.

2. Problem formulation

2.1. Notations

S	number of subsystems
n_i	number of components used in the subsystem i (i=1, 2,…, S)
N	set of n_i (n_1, n_2, …, n_S)
$n_{\max,I}$	maximum number of components used in the subsystem i
m_i	number of available components for the subsystem i
z_i	index of components selected for the subsystem i
Z	set of z_i (z_1, z_2, …, z_s)
t	mission time
$R(t; Z, N)$	system reliability at time t for solution vectors Z and N
$r_i(t)$	reliability at time t for the jth components of the subsystem i
C, W	cost and weight constraints
$C_{i,j}$, $w_{i,j}$	cost and weight of the jth component of the subsystem i
ρ	Probability of the switch working properly

2.2. Formulation

$$\max MTTF = \int_0^t R(t;Z,N)dt \tag{1}$$

$$R(t;Z,N) = \prod_{i=1}^{S}\left(r_{iz_i}(t) + \sum_{x=1}^{n_i-1} \rho^x \int_0^t r_{iz_i}(t-u)\, f_{iz_i}^{(x)}(u)du \right) \tag{2}$$

Subject to:

$$\sum_{i=1}^{S} c_{iz_i} n_i \le C \quad n_i \in \{1, 2, \ldots, n_{\max,i}\} \tag{3}$$

$$\sum_{i=1}^{S} w_{iz_i} n_i \le W \quad z_i \in \{1, 2, \ldots, m_i\} \tag{4}$$

The R (t;Z,N) formulation is derived from [11]. Owing to the complexity of its calculation, Coit had to use an equivalent formulation. Exact calculation of the MTTF is more complex than that of R(t; Z,N). We use the Monte Carlo simulation to overcome the complexity. Constraints (3) and (4) represent the cost and weight constraints, respectively.

3. Proposed genetic algorithm

The GA is one of the most effective and applied meta-heuristic techniques for solving a variety of combinatorial problems. It has also been successfully employed for a variety of RAP [13 and 14]. The GA starts with a set of randomly generated initial solutions (chromosomes) called the initial population. A crossover operator is applied to improve the quality of solutions and a mutation operator is applied to increase the variety of solutions. Solutions for the crossover operator are selected by a mechanism called the selection strategy. The algorithm is repeated until the termination condition is met.

3.1.Solution representation

The solution is represented by two vectors, one representing the selected components, and the other, the level of redundancy. Figure 2 illustrates the solution representation for a system with seven subsystems. The *j*th bit in both vectors represents the selected component and the redundancy level for the *j*th subsystem.

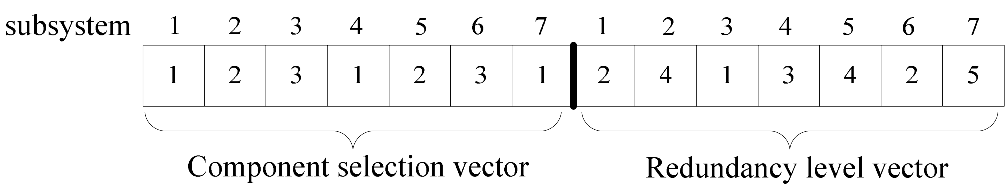

Figure 2. Solution representation

3.2.Fitness function

Each chromosome is evaluated by a fitness function. In this paper, the fitness function comprises the result of the Monte Carlo simulation and the penalty function.

3.2.1. Monte Carlo simulation

The Monte Carlo methods are a class of computational algorithms that rely on repeated random sampling to compute their results. These methods are often used for simulating physical and mathematical systems. They are used when it is infeasible to compute an exact result using a deterministic algorithm.

Calculating the reliability of a system with the cold-standby redundancy strategy at time t is possible only when the time to failure of the components follows a simple distribution function. Obtaining the MTTF of a system is very difficult, even when the components have a simple distribution function. The Monte Carlo simulation can be very useful in calculating both the reliability at time t and the MTTF objective functions when the distribution function of the time to failure of the component is complex.

The simulation algorithm for calculating the MTTF is presented in Table 1,

where,

$n.sim$	number of simulation
S	number of subsystem
r_{ij}	a random number assumed to be the time to failure of the jth component in subsystem i
n_i	redundancy level in subsystem i
ρ	probability of switch working properly
TTF_i	time to failure of subsystem i
$TTFS_k$	time to failure of the system for the kth simulation
MTTF	mean time to failure of the system

Table 1. Proposed simulation algorithm

```
For (k=1 to n.sim)
Begin
For (i=1 to S)
        Begin
                Generate a random number r_i1
        TTF_i ← r_i1
For (j=2 to n_i)
                Begin
                        Generate a random number R ∈ (0,1)
If R< ρ then
                        Begin
                                Generate a random number r_ij
                                TTF_i ← TTF_i + r_ij
End if
                        Else
                             Exit "For j" loop
                End for j
        End for i
        TTFS_k = min_i TTF_i
End for k
```

$$MTTF = \frac{\sum_{k=1}^{n.sim} TTFS_k}{n.sim}$$

According to the Table 1, the simulation is performed *n.sim* times, and the average of the simulation results gives the value of the MTTF. For each simulation, the time to failure values of the subsystems are calculated, the minimum value gives us the time to failure of the system.
For the simulation to estimate the time to failure of a subsystem, first the distribution function is used to generate a random number as the time to failure of the first component. This number is assumed to be the time to failure of the subsystem. If other components are available in that subsystem, a constant number between 0 and 1 that is generated randomly is used to determine if the switch has worked. If this number is less than ρ, it means that the switch has worked properly and another component has been activated. The time to failure of this component is added to the time to failure of the subsystem. The simulation terminates when this process is carried out for all components in the subsystem or when the switch does not work properly.

3.2.2. Penalty function

The crossover and mutation operators can produce infeasible solutions. The penalty method is the most effective approach to resolve this problem and provides an efficient search through the feasible and infeasible regions. Coit & Smith [13] used a penalty function in a GA to solve an RAP; this method significantly increased the quality of solutions. In this paper, a dynamic penalty function derived from [13] is used.
The fitness function is obtained by subtracting the penalty function from the result of the simulation:

$$f_{penalized} = f_{unpenalized} - \left(\left(\frac{\Delta w}{NFT_w} \right)^{\alpha} + \left(\frac{\Delta c}{NFT_c} \right)^{\alpha} \right) \left(f_{best}(all) - f_{best}(feas) \right) \qquad (5)$$

where, $f_{penalized}$ is the penalized objective function, $f_{unpenalized}$ is the objective function obtained by simulation, f_{best} (*feas*) is the value of the current best feasible solution, and f_{best} (*all*)is the value of the best solution without considering the penalty.
Δw and Δc denote the amount of violations of the weight and cost constraints for the components, respectively. *K* is a severity parameter. NFT_c and NFT_w are the "near-feasible thresholds (NFTs)" for the cost and weight constraints, respectively. The dynamic NFTs for the cost and weight constraints are calculated by the following formula:

$$NFT = \frac{NFT_0}{1 + \beta g^k} \qquad (6)$$

where, g is the generation number and NFT_0, β, and k are constant parameters.

3.3. Selection strategy

The chromosomes for crossing are selected by employing a linear ranking selection strategy. In this strategy, chromosomes are sorted according to the fitness function and all the chromosomes are assigned a rank $r_i \in \{1,2,...,N\}$, where N is the population size.

Rank N indicates the best solution, whereas rank 1 indicates the worst. The probability for selecting chromosome i (p_i) is obtained by the following formula:

$$p_i = \frac{2r_i}{N(N+1)} \tag{7}$$

3.4.Crossover operator

A uniform crossover operator is applied to both solution vectors. The operator first generates a random binary array called a mask. The size of the mask is equal to the size of the chromosomes. An offspring inherits genes from parents according to the mask. To produce an offspring, the mask is scanned bit by bit, and if the current bit is equal to 0, then the offspring inherits the corresponding gene from the first parent. Otherwise, the gene is inherited from the second parent. The second offspring is produced in a similar way, but with the difference that in the mask, 0 is changed to 1 and 1 to 0. Figure 3 depicts the uniform crossover with an example.

	1	2	3	4	5	6	7	1	2	3	4	5	6	7
Parent 1	2	2	1	3	1	3	3	2	1	3	2	4	3	3
Parent 2	1	3	3	1	2	2	1	4	4	2	2	1	5	3
Mask	0	1	1	0	1	0	0	0	1	0	1	1	0	1
Offspring 1	2	3	3	3	2	3	3	2	4	3	2	1	3	3
Offspring 2	1	2	1	1	1	2	1	4	1	2	2	4	5	3

Figure 3. Crossover operator

If P is the size of the population, then P offsprings are produced by crossover and are combined with the parents. Then P number of the best solutions among them are selected and they survive to the next generation. After deleting the inferior solutions from the population, mutation is performed on the remaining solutions. Mutation is not applied to the best solution and the best feasible solution.

3.5.Mutation operator

The probability of mutation of a gene is P_m. To mutate a chromosome, first a random number $r_1 \in (0,1)$ is generated for each gene. Then, for anyone bit in the component selection vector where $r_1 < P_m$, another component is selected, which replaces the previous one. For anyone bit in the redundancy level vector where $r_1 < P_m$, another random number $r_2 \in (0,1)$ is generated. If $r_2 < 0.5$, the redundancy level of the selected subsystem is decreased by one. Otherwise, it is increased by one. If the redundancy level of subsystem is 1, the subsystem is only permitted to increase and if the level is equal to n_{max}, it is only permitted to decrease. An example of the mutation operator is depicted in Figure 4.

	1	2	3	4	5	6	7	1	2	3	4	5	6	7
Befor mutation	1	2	3	1	3	1	2	2	4	1	3	6	2	5
Random numbers r_1	0.25	0.82	0.05	0.55	0.63	0.99	0.17	0.39	0.66	0.01	0.81	0.08	0.27	0.09
Random numbers r_2			0.65											0.34
After mutation	1	2	2	1	2	3	1	2	4	2	3	5	2	5

Figure 4. Mutation operator (P_m=0.1)

4. Experimental design

4.1.Test problems

Thirty-three problems provided by Coit [11] are used to test the algorithm. Only the weight constraint differs with each problem. These problems are applied to a system with 14 subsystems where each subsystem has three or four component types to choose from. Table 2 lists the shape and scale parameters for a Gamma distribution and the cost and weight of each component. The cost constraint is 130 for all problems, and the weight constraint ranges from 159 to 191. The maximum number of components in all subsystems is 6, and the reliability of the switch for all subsystems is 0.99.

Table 2. The example data

i	Choice 1				Choice 2				Choice 3				Choice 4			
	λ_{ij}	k_{ij}	c_{ij}	w_{ij}	λ_{ij}	k_{ij}	c_{ij}	w_{ij}	λ_{ij}	k_{ij}	c_{ij}	w_{ij}	λ_{ij}	k_{ij}	c_{ij}	w_{ij}
1	0.00532	2	1	3	0.000726	1	1	4	0.00499	2	2	2	0.00818	3	2	5
2	0.00818	3	2	8	0.000619	1	1	10	0.00431	2	1	9	-	-	-	-
3	0.0133	3	2	7	0.0110	3	3	5	0.0124	3	1	6	0.00466	2	4	4
4	0.00741	2	3	5	0.0124	3	4	6	0.00683	2	5	4	-	-	-	-
5	0.00619	1	2	4	0.00431	2	2	3	0.00818	3	3	5	-	-	-	-
6	0.00436	3	3	5	0.00567	3	3	4	0.00268	2	2	5	0.000408	1	2	4
7	0.0105	3	4	7	0.00466	2	4	8	0.00394	2	5	9	-	-	-	-
8	0.0150	3	3	4	0.00105	1	5	7	0.0105	3	6	6	-	-	-	-
9	0.00268	2	2	8	0.000101	1	3	9	0.000408	1	4	7	0.000943	1	3	8
10	0.0141	3	4	6	0.00683	2	4	5	0.00105	1	5	6	-	-	-	-

11	0.00394	2	3	5	0.00355	2	4	6	0.00314	2	5	6	-	-	-	-
12	0.00236	1	2	4	0.00769	2	3	5	0.0133	3	4	6	0.0110	3	5	7
13	0.00215	2	2	5	0.00436	3	3	5	0.00665	3	2	6	-	-	-	-
14	0.0110	3	4	6	0.00834	1	4	7	0.00355	2	5	6	0.00436	3	6	9

4.2.Experimental results

The algorithm was coded in Borland C++ and executed on a computer with 2.1 GHz Intel Core 2 Duo processor and 2GB of RAM. We executed the algorithms five times for each problem. The execution of the algorithm was terminated when the array of the best solution remained unchanged for 10 consecutive generations. The final solution obtained by the GA is simulated one million times to obtain the final objective function.

After extensive experiments with different penalty and genetic parameters, the following values were selected: population size=300, P_m=0.005, α=2, NFT_{0cost}=100, $NFT_{0weight}$=W/1.3,β_{cost} =0.008, β_{weight} =0.08, k=1.6.

Table 3 lists the experimental results for 33 problems,

where,

#	problem number
W	weight constraint
Best	objective function of the best solution in five executions
Mean	average of objective function in five executions
CV	coefficient of variance for values of objective function (standard deviation divided by average)
CT	average of CPU time in seconds for five executions
Best solution	best solution includes two vectors, A-B. A represents the selected components vector, and B represents the level of the redundant vector. For example, the solution shown in Figure 2 can be represented as 1231231-2413425.

The best and the average values of the objective functions for all problems are depicted in Figure 5. The values indicate that the reliability of the system improves with an increase in W.

Coit [11] used Hyper-LINDO software for solving problem 12 with the objective of maximizing the reliability of the system at time *t* (*t*= 100). He used the equivalent objective function instead of the original one and obtained a value of 0.9863.

We tested the algorithm on problem 12 as well, with the objective of maximizing the reliability of the system at time *t* (*t*= 100). To calculate this objective function, we replaced the last statement of the algorithm in Table 1 with the following statement.

counter ← 0
For (*l*=1 to n.sim)
 If $TTFS_l$ equal or bigger than *t* then
 counter←counter+1
End for *l*
Reliability ← counter / n.sim

The added statement compares the simulation result with *t*. If it is equal or bigger than *t*,it means the system has worked at least *t* times. The reliability of the system at time *t* is obtained by dividing the total number of simulations having a result equal or bigger than *t* by the number of simulation (n.sim).

Our algorithm is applied to problem 12 and the following results are obtained:

Best=0.9856, Average=0.9851, Dev/Ave=0.000396, CT=337.8,

BS: 31432213131223-32333222232322

Table 3. Performance of genetic algorithm for the problems provided by Coit (2001)

No	W	Best	Ave	CV	CT	BS
1	159	382.461	380.781	0.003953	321.8	32432422231113-32332222122322
2	160	388.239	384.198	0.009568	379.0	32432432231113-32233222122322
3	161	392.464	392.070	0.000606	351.8	32432432231113-32332222122322
4	162	400.459	400.239	0.000331	309.2	32432422231113-32333222122322
5	163	400.513	399.186	0.007007	364.0	32432422231113-32333222122322
6	164	412.006	411.785	0.000549	281.6	32432432231113-32333222122322
7	165	412.109	407.680	0.013560	357.8	32432432233113-32333222122322
8	166	421.623	421.367	0.000506	302.2	32432432233113-32333222122322
9	167	421.68	421.451	0.000552	330.0	32432432233113-32333222122322
10	168	427.532	426.074	0.006152	311.8	32432432231113-32343222122322
11	169	430.129	426.222	0.009735	383.6	32432432231113-32333222123322
12	170	438.89	433.764	0.015780	353.6	32432432233113-32343222122322
13	171	434.657	428.150	0.009003	337.2	32432432233113-32333222122422
14	172	445.937	441.726	0.016936	315.0	32432432233113-42343222122322
15	173	448.726	443.447	0.016106	325.6	32432432231113-32343222123322
16	174	455.372	449.597	0.017074	333.4	32432432233113-32343222122422
17	175	456.782	452.295	0.010508	329.4	32432432231113-42343222123322
18	176	461.76	451.333	0.016292	298.0	32432432233113-42343222122422
19	177	466.271	462.322	0.011242	301.8	324324 32231113-32343222123422
20	178	466.691	460.993	0.011978	366.0	32432432231113-42343222123422
21	179	467.928	465.394	0.004233	350.2	32432432233113-42344222122422
22	180	478.071	476.691	0.002865	306.6	32432422233113-32343232122422
23	181	477.55	474.069	0.007238	340.0	32432422233113-32343232122422
24	182	487.347	478.621	0.012955	337.6	32432422233113-42343232122422
25	183	490.551	487.212	0.008694	294.4	32432422231113-32343232123422
26	184	490.361	484.001	0.015448	372.0	32432422231113-32343232123422
27	185	501.498	494.646	0.018686	366.4	32432422231113-42343232123422
28	186	501.595	493.007	0.022069	351.8	32432422231113-42343232123422
29	187	502.081	499.850	0.005622	331.0	32432422233113-42343232122423
30	188	501.859	497.215	0.007501	293.2	32432422233113-42343233122422
31	189	517.192	506.885	0.023134	330.2	32432422231113-32343232123423
32	190	513.218	509.122	0.010504	344.2	32432422231113-42343232133422
33	191	530.712	527.421	0.008088	340.0	32432422231113-42343232123423

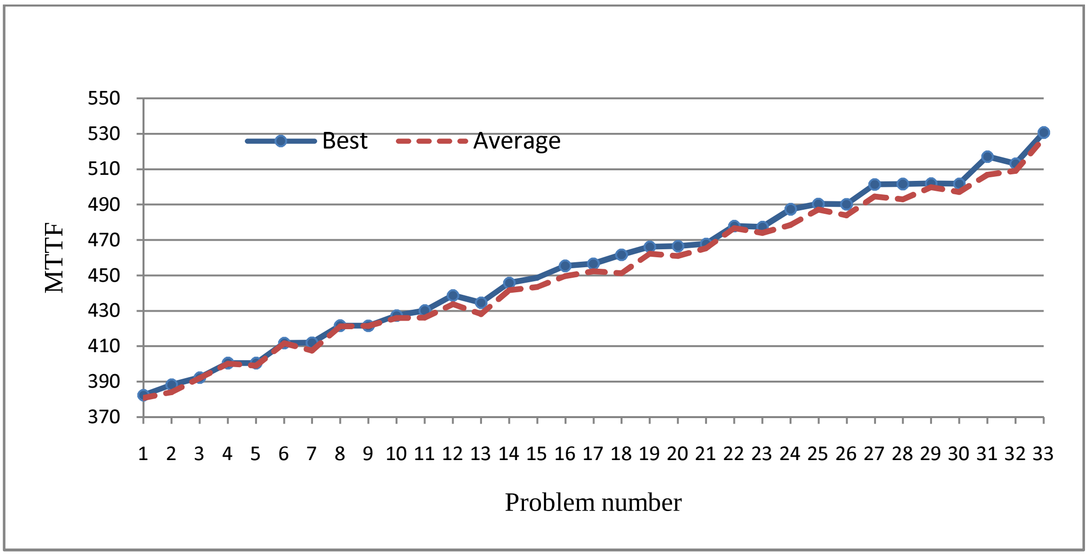

Figure 5. Best and average values of MTTF in 33 problems

5. Conclusions

In this study, we combined a genetic algorithm with a Monte Carlo simulation to solve the redundancy allocation problem (RAP). The objective was to maximize the mean time to failure (MTTF) of a series-parallel system employing the cold-standby redundancy strategy. We also expanded the algorithm to solve the RAP and hence to maximize the system reliability at time *t*. Figure 5 and the magnitude of the coefficient of variance in Table 3 imply that the algorithm is robust and very effective.

In future studies, this approach can be applied to solve RAPs where the components of the system are repairable, and the distribution function for time to failure of components is more complex. The simultaneous maximization of both the MTTF and system reliability at time *t* using multi-objective algorithms is an area that requires further study.

References

[1] Coit, D.W. (2003), *Maximization of system reliability with a choice of redundancy strategies*, IIE Trans., Vol. 35, pp. 535-544.

[2] Bellman, R. and Dreyfus, S. (1958), Dynamic programming and the reliability of multicomponent devices, *Oper. Res.*, Vol. 6, pp. 200-206.

[3] Fyffe, E.D., Hines, W.W. and Lee, N.K. (1968), *System reliability allocation and a computational algorithm*, IEEE Trans. Reliab., Vol. 17, pp. 64-69.

[4] Nakagawa, Y., and Miyazaki, S. (1981), *Surrogate constraints algorithm for reliability optimization problems with two constraints*, IEEE Trans. Reliab., Vol. 30, No. 2, pp. 175-80.

[5] Li, J. (1996), A bound dynamic programming for solving reliability optimization, *Microelectr. Reliab.*, Vol. 36, No. 10, pp. 1515-1520.

[6] Ghare, P.M. and Taylor, R.E. (1969), Optimal redundancy for reliability in series systems, *Oper. Res.*, Vol. 17, pp. 838-847.

[7] Bulfin, R.L. and Liu, C.Y. (1985), *Optimal allocation of redundant components for large systems*, IEEE Trans. Reliab., Vol. 34, No. 3, pp. 241-247.

[8] Misra, K.B. and Sharma, U. (1991), *An efficient algorithm to solve integer programming problems arising in system-reliability design*, IEEE Trans. Reliab., Vol. 40, No. 1, pp. 81-91.

[9] Coit, D.W. and Liu, J. (2000), System reliability optimization with k-out-of-n subsystems, *Int. J. Reliab. Qual. Safety Eng*., Vol. 7, No. 2, pp. 129-143.

[10] Tillman, F.A., Hwang, C.L. and Kuo, W. (1977), *Determining component reliability and redundancy for optimum system reliability*, IEEE Trans. Reliab., Vol. 26, No. 3, pp. 162-165.

[11] Coit, D.W. (2001), *Cold-standby redundancy optimization for non-repairable systems*, IIE Transactions, Vol. 33, pp. 471-478.

[12] Chern, M.S. (1992), On the computational complexity of reliability redundancy allocation in a series system, *Oper. Res. Lett*., Vol. 11, pp. 309-315.

[13] Coit, D.W. and Smith, A.E. (1996a), Penalty guided genetic search for reliability design optimization, *Comput. Ind. Eng*., Vol. 30, No. 4, pp. 895-904.

[14] Coit, D.W. and Smith, A.E. (1996b), *Reliability optimization of series-parallel systems using a genetic algorithm*, IEEE Trans. Reliab., Vol. 45, No. 2, pp. 254-260.

[15] Liang, Y.C. and Wu, C.C. (2005), A variable neighborhood descent algorithm for the redundancy allocation problem, *Ind. Eng. Manage. Syst*., Vol. 4, No. 1, pp. 109–116.

[16] Liang, Y.C. and Smith, A.E. (1999), *An ant system approach to redundancy allocation*, In Proceedings of the congress on evolutionary computation, Washington DC, USA, pp. 1478-1484.

[17] Liang, Y.C. and Smith, A.E. (2004), *An ant colony optimization algorithm for the redundancy allocation problem*, IEEE Trans. Reliab., Vol. 53, No. 3, pp. 417–423.

[18] Najafi, A.A., Karimi, H., Chambari, A. and Azimi, F. (2013), Two metaheuristics for solving the reliability redundancy allocation problem to maximize mean time to failure of a series–parallel system, *Scientia Iranica*, In Press.