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

H. Karimi¹, A.A. Najafi²,*
¹Faculty of Industrial and Mechanical Engineering, Qazvin Branch, Islamic Azad University, Qazvin, Iran.
²Faculty of Industrial Engineering, K.N. Toosi University of Technology, Tehran, Iran.

Abstract

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.
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 $\rho_i$ [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 that in real-world problems, 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 \( \rho \). The components are assumed to be not repairable.

In each subsystem, there are \( m_i \) 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

\[
\begin{align*}
S & \quad \text{number of subsystems} \\
n_i & \quad \text{number of components used in the subsystem } i (i=1, 2, \ldots, S) \\
N & \quad \text{set of } n_i (n_1, n_2, \ldots, n_S) \\
n_{\text{max},i} & \quad \text{maximum number of components used in the subsystem } i \\
m_i & \quad \text{number of available components for the subsystem } i \\
z_i & \quad \text{index of components selected for the subsystem } i \\
Z & \quad \text{set of } z_i (z_1, z_2, \ldots, z_i) \\
t & \quad \text{mission time} \\
R(t; Z, N) & \quad \text{system reliability at time } t \text{ for solution vectors } Z \text{ and } N \\
r_i(t) & \quad \text{reliability at time } t \text{ for the } j\text{th components of the subsystem } i \\
C, W & \quad \text{cost and weight constraints} \\
C_{ij}, w_{ij} & \quad \text{cost and weight of the } j\text{th component of the subsystem } i \\
\rho & \quad \text{Probability of the switch working properly}
\end{align*}
\]

2.2. Formulation

\[
\begin{align}
\max MTTF & = \int_0^t R(t; Z, N) dt \\
R(t; Z, N) & = \prod_{i=1}^{S} \left( r_{z_i}(t) + \sum_{x=1}^{n_i-1} \rho^x \int_0^t r_{z_i}(t-u) f_{z_i}(u) du \right)
\end{align}
\]
Subject to:

\[
\sum_{i=1}^{S} c_{i, w} n_i \leq C \quad n_i \in \{1, 2, \ldots, n_{\text{max},i}\}
\]  

(3)

\[
\sum_{i=1}^{S} w_{i, z} n_i \leq W \quad z_i \in \{1, 2, \ldots, m_i\}
\]  

(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.

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,

- \( n\text{.sim} \) number of simulation
- \( S \) number of subsystem
- \( r_{ij} \) a random number assumed to be the time to failure of the \( j \)th component in subsystem \( i \)
- \( n_i \) redundancy level in subsystem \( i \)
- \( \rho \) probability of switch working properly
- \( TTF_i \) time to failure of subsystem \( i \)
- \( TTFS_k \) time to failure of the system for the \( k \)th simulation
- \( MTTF \) mean time to failure of the system

Table 1. Proposed simulation algorithm

```plaintext
For (k=1 to n.sim)
Begin
For (i=1 to S)
Begin
    Generate a random number \( r_{i1} \)
    \( TTF_i \leftarrow r_{i1} \)
For (j=2 to \( n_i \))
Begin
    Generate a random number \( R \in (0,1) \)
    If \( R < \rho \) then
        Generate a random number \( r_{ij} \)
        \( TTF_i \leftarrow TTF_i + r_{ij} \)
    Else
        Exit “For j” loop
End if
End for j
End for i
TTFS\( k \) = \( \min_i TTF_i \)
End for k
MTTF = \( \frac{\sum_{k=1}^{n\text{.sim}} TTFS_k}{n\text{.sim}} \)
```

According to the Table 1, the simulation is performed $n_{\text{sim}}$ 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 $\rho$, 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:

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

(5)

where, $f_{\text{penalized}}$ is the penalized objective function, $f_{\text{unpenalized}}$ is the objective function obtained by simulation, $f_{\text{best}}(\text{feas})$ is the value of the current best feasible solution, and $f_{\text{best}}(\text{all})$ is the value of the best solution without considering the penalty.

$\Delta w$ and $\Delta c$ 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:

$$NFT = \frac{NFT_0}{1 + \beta g^k}$$

(6)

where, $g$ is the generation number and $NFT_0$, $\beta$, 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:

\[
p_i = \frac{2r_i}{N(N+1)}
\]  

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.

![Figure 3. Crossover operator](image)

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 any one bit in the component selection vector where \( r_1 < P_m \), another component is selected, which replaces the previous one. For any one 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 \( n_{\text{max}} \), it is only permitted to decrease. An example of the mutation operator is depicted in Figure 4.

![Mutation operator](image)

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>
<thead>
<tr>
<th>( \lambda_{ij} )</th>
<th>( k_{ij} )</th>
<th>( c_{ij} )</th>
<th>( w_{ij} )</th>
<th>( \lambda_{ij} )</th>
<th>( k_{ij} )</th>
<th>( c_{ij} )</th>
<th>( w_{ij} )</th>
<th>( \lambda_{ij} )</th>
<th>( k_{ij} )</th>
<th>( c_{ij} )</th>
<th>( w_{ij} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.00532</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>0.000726</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>0.00499</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>0.00818</td>
<td>3</td>
<td>2</td>
<td>8</td>
<td>0.000619</td>
<td>1</td>
<td>1</td>
<td>10</td>
<td>0.00431</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0.0133</td>
<td>3</td>
<td>2</td>
<td>7</td>
<td>0.0110</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>0.0124</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0.00741</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>0.0124</td>
<td>3</td>
<td>3</td>
<td>6</td>
<td>0.00683</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>0.00619</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>0.00431</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>0.00818</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>0.00436</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>0.00567</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>0.00268</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>0.0105</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>0.00466</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>0.00394</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>0.0150</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>0.000105</td>
<td>1</td>
<td>5</td>
<td>7</td>
<td>0.0105</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>0.00268</td>
<td>2</td>
<td>2</td>
<td>8</td>
<td>0.000101</td>
<td>1</td>
<td>3</td>
<td>9</td>
<td>0.000408</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>0.0141</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>0.00683</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>0.00105</td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 2. The example data
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, $P_m=0.005$, $\alpha=2$, $NFT_{0\text{cost}}=100$, $NFT_{0\text{weight}}=W/1.3$, $\beta_{\text{cost}}=0.008$, $\beta_{\text{weight}}=0.08$, $k=1.6$.

Table 3 lists the experimental results for 33 problems, where,

<table>
<thead>
<tr>
<th>№</th>
<th>problem number</th>
<th>W</th>
<th>weight constraint</th>
<th>Best</th>
<th>objective function of the best solution in five executions</th>
<th>Mean</th>
<th>average of objective function in five executions</th>
<th>CV</th>
<th>coefficient of variance for values of objective function (standard deviation divided by average)</th>
<th>CT</th>
<th>average of CPU time in seconds for five executions</th>
<th>Best solution</th>
<th>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.</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>0.00394</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>0.00355</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0.00236</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>0.00769</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>0.0133</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>0.0110</td>
</tr>
<tr>
<td>13</td>
<td>0.00215</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>0.00436</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>0.00665</td>
<td>3</td>
<td>2</td>
<td>6</td>
<td>-</td>
</tr>
<tr>
<td>14</td>
<td>0.0110</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>0.00834</td>
<td>1</td>
<td>4</td>
<td>7</td>
<td>0.00355</td>
<td>2</td>
<td>5</td>
<td>6</td>
<td>0.00436</td>
</tr>
</tbody>
</table>

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.

```algorithms
counter ← 0
For (l=1 to n.sim)
  If TTFS_l equal or bigger than t then
    counter ← counter+1
  End for /
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>
<thead>
<tr>
<th>No</th>
<th>W</th>
<th>Best</th>
<th>Ave</th>
<th>CV</th>
<th>CT</th>
<th>BS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>159</td>
<td>382.461</td>
<td>380.781</td>
<td>0.003953</td>
<td>321.8</td>
<td>32432422231113-32332222123222</td>
</tr>
<tr>
<td>2</td>
<td>160</td>
<td>388.239</td>
<td>384.198</td>
<td>0.009568</td>
<td>379.0</td>
<td>32432422311113-32332222123222</td>
</tr>
<tr>
<td>3</td>
<td>161</td>
<td>392.464</td>
<td>392.070</td>
<td>0.000606</td>
<td>351.8</td>
<td>32432422311113-32332222123222</td>
</tr>
<tr>
<td>4</td>
<td>162</td>
<td>400.459</td>
<td>400.239</td>
<td>0.000331</td>
<td>309.2</td>
<td>32432422311113-32332222123222</td>
</tr>
<tr>
<td>5</td>
<td>163</td>
<td>400.513</td>
<td>399.186</td>
<td>0.007007</td>
<td>364.0</td>
<td>32432422311113-32332222123222</td>
</tr>
<tr>
<td>6</td>
<td>164</td>
<td>412.006</td>
<td>411.785</td>
<td>0.000549</td>
<td>281.6</td>
<td>32432432211113-32332222123222</td>
</tr>
<tr>
<td>7</td>
<td>165</td>
<td>412.109</td>
<td>407.680</td>
<td>0.013560</td>
<td>357.8</td>
<td>32432432211113-32332222123222</td>
</tr>
<tr>
<td>8</td>
<td>166</td>
<td>421.623</td>
<td>421.367</td>
<td>0.000506</td>
<td>302.2</td>
<td>32432432211113-32332222123222</td>
</tr>
<tr>
<td>9</td>
<td>167</td>
<td>421.68</td>
<td>421.451</td>
<td>0.000552</td>
<td>330.0</td>
<td>32432432211113-32332222123222</td>
</tr>
<tr>
<td>10</td>
<td>168</td>
<td>427.532</td>
<td>426.074</td>
<td>0.006152</td>
<td>311.8</td>
<td>32432432211113-32332222123222</td>
</tr>
<tr>
<td>11</td>
<td>169</td>
<td>430.129</td>
<td>426.222</td>
<td>0.009735</td>
<td>383.6</td>
<td>32432432211113-32332222123222</td>
</tr>
<tr>
<td>12</td>
<td>170</td>
<td>438.89</td>
<td>433.764</td>
<td>0.015780</td>
<td>353.6</td>
<td>32432432211113-32332222123222</td>
</tr>
<tr>
<td>13</td>
<td>171</td>
<td>443.657</td>
<td>442.150</td>
<td>0.009003</td>
<td>337.2</td>
<td>32432432211113-32332222123222</td>
</tr>
<tr>
<td>14</td>
<td>172</td>
<td>445.937</td>
<td>441.726</td>
<td>0.016936</td>
<td>315.0</td>
<td>32432432211113-32332222123222</td>
</tr>
<tr>
<td>15</td>
<td>173</td>
<td>448.726</td>
<td>443.447</td>
<td>0.016106</td>
<td>325.6</td>
<td>32432432211113-32332222123222</td>
</tr>
<tr>
<td>16</td>
<td>174</td>
<td>455.372</td>
<td>449.597</td>
<td>0.017074</td>
<td>333.4</td>
<td>32432432211113-32332222123222</td>
</tr>
<tr>
<td>17</td>
<td>175</td>
<td>456.782</td>
<td>452.295</td>
<td>0.010508</td>
<td>329.4</td>
<td>32432432211113-42343222123222</td>
</tr>
<tr>
<td>18</td>
<td>176</td>
<td>461.76</td>
<td>451.333</td>
<td>0.016292</td>
<td>298.0</td>
<td>32432432211113-42343222123222</td>
</tr>
<tr>
<td>19</td>
<td>177</td>
<td>466.271</td>
<td>462.322</td>
<td>0.011242</td>
<td>301.8</td>
<td>32432432211113-42343222123222</td>
</tr>
<tr>
<td>20</td>
<td>178</td>
<td>466.691</td>
<td>460.993</td>
<td>0.011978</td>
<td>366.0</td>
<td>32432432211113-42343222123222</td>
</tr>
<tr>
<td>21</td>
<td>179</td>
<td>467.928</td>
<td>465.394</td>
<td>0.004233</td>
<td>350.2</td>
<td>32432432211113-42344222123222</td>
</tr>
<tr>
<td>22</td>
<td>180</td>
<td>478.071</td>
<td>476.691</td>
<td>0.002865</td>
<td>306.6</td>
<td>32432432211113-32343232123222</td>
</tr>
<tr>
<td>23</td>
<td>181</td>
<td>477.55</td>
<td>474.069</td>
<td>0.007238</td>
<td>340.0</td>
<td>32432432211113-32343232123222</td>
</tr>
<tr>
<td>24</td>
<td>182</td>
<td>487.347</td>
<td>487.621</td>
<td>0.012955</td>
<td>337.6</td>
<td>32432432211113-42343222123222</td>
</tr>
<tr>
<td>25</td>
<td>183</td>
<td>490.551</td>
<td>487.212</td>
<td>0.008694</td>
<td>294.4</td>
<td>32432432211113-32343222123222</td>
</tr>
<tr>
<td>26</td>
<td>184</td>
<td>490.361</td>
<td>484.001</td>
<td>0.015448</td>
<td>372.0</td>
<td>32432432211113-32343222123222</td>
</tr>
<tr>
<td>27</td>
<td>185</td>
<td>501.498</td>
<td>494.646</td>
<td>0.018686</td>
<td>366.4</td>
<td>32432432211113-42343222123222</td>
</tr>
<tr>
<td>28</td>
<td>186</td>
<td>501.595</td>
<td>493.007</td>
<td>0.022069</td>
<td>351.8</td>
<td>32432432211113-42343222123222</td>
</tr>
<tr>
<td>29</td>
<td>187</td>
<td>502.081</td>
<td>499.850</td>
<td>0.005622</td>
<td>331.0</td>
<td>32432432211113-42343222123222</td>
</tr>
<tr>
<td>30</td>
<td>188</td>
<td>501.859</td>
<td>497.215</td>
<td>0.007501</td>
<td>293.2</td>
<td>32432432211113-42343222123222</td>
</tr>
<tr>
<td>31</td>
<td>189</td>
<td>517.192</td>
<td>506.885</td>
<td>0.023134</td>
<td>330.2</td>
<td>32432432211113-42343222123222</td>
</tr>
<tr>
<td>32</td>
<td>190</td>
<td>513.218</td>
<td>509.122</td>
<td>0.010504</td>
<td>344.2</td>
<td>32432432211113-42343222123222</td>
</tr>
<tr>
<td>33</td>
<td>191</td>
<td>530.712</td>
<td>527.421</td>
<td>0.008088</td>
<td>340.0</td>
<td>32432432211113-42343222123222</td>
</tr>
</tbody>
</table>
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


