Comparing Parallel Simulated Annealing, Parallel Vibrating Damp Optimization and Genetic Algorithm for Joint Redundancy-Availability Problems in a Series-Parallel System with Multi-State Components

Mani Sharifi\(^a\)*, Morteza Mousa Khani\(^b\), Arash Zaretalab\(^c\)

\(^a\) Assistant Professor, Faculty of Industrial & Mechanical Engineering, Qazvin Branch, Islamic Azad University, Qazvin, Iran
\(^b\) Associate Professor, Faculty of Management & Accounting, Qazvin Branch, Islamic Azad University, Qazvin, Iran
\(^c\) MSc, Faculty of Industrial & Mechanical Engineering, Qazvin Branch, Islamic Azad University, Qazvin, Iran

Received 10 October, 2012; Revised 19 March, 2013; Accepted 20 May, 2013

Abstract

In this paper, we study different methods of solving joint redundancy-availability optimization for series-parallel systems with multi-state components. We analyzed various effective factors on system availability in order to determine the optimum number and version of components in each sub-system and consider the effects of improving failure rates of each component in each sub-system and improving reliability of each sub-system. The target is to determine optimum values of all variables for improving the availability level and decreasing the total cost of the system. At first, the exact values of variables are determined using a mathematical model; then, the results of SA-Parallel, VDO-Parallel and genetic algorithms are compared with the exact solution.

Keywords: SA; VDO; Parallel; Availability; Multi-state; Redundancy.

1. Introduction

The effective factors in system availability in general include component types in each sub-system, the number of components in each sub-system and the level of improving failure rates for each component in each sub-system, as well as improving reliability of each sub-system. The third factor concerns decreasing the failure rate of each component and increasing the reliability of subsystems by spending money. For example, if we consider more skilled workers in each subsystem, the reliability of the sub-system and in turn its availability will increase.

In traditional models, the components and subsystems have only two states: working, not working (failed). But it is obvious that they may be in the states between the two above states. These states are caused by slight failures and operational defects. These systems are called multi-state systems (MSS) (Barlow & Wu, 1987; Boedigheimer et al, 1994; Lisianaski et al, 2003; Zuo et al, 2006; Zuo et al, 2007).

In this regard, efforts have been made to optimize the level of redundancy in the various subsystems of a system. Levitin et al (1998) developed a model to determine the optimal version of subsystem components in a multi-state series-parallel system. Coit and Ramirez Marquez (2004) introduced a heuristic method for multi-state Redundancy Allocation Problem (RAP), and Tian and Zuo (2006) provided a method based on physical programming and genetic algorithm for multi-state RAP. In both models, RAP was the only factor in the improvement of system operation, but Tian et al (2009) introduced a modern method for optimizing series-parallel systems. In doing so, they provided two options for improving system reliability and availability: optimal redundancy allocation in various subsystems and improving the reliability of components through affecting functional rates. Furthermore, Tian et al (2005) presented a joint reliability-redundancy optimization method for multi-state series-parallel systems inspired by the method described above. In their method, transition rates affect component state distributions and redundancy is considered as a variable of the problem. After Chern (1972) proved that RAP belongs to NP hard problems, Mehta heuristic algorithms are used increasingly in the area. In this regard, Gen and Kim (1999) used a hybrid genetic algorithm to optimize reliability problems and Konak et al (2003) took advantage of using the tabu search for RAP. Liang et al (2004) solved RAP using the

* Corresponding author E-mail: mani.sharifi@yahoo.com
ant colony algorithm while Chen and Liang (2007) solved series-parallel RAP by VNS. Moreover, Lisnianski et al. (2008) described Universal Generating Function (UGF) for RAP in fuzzy and crisp situations.

In the same vein, in the present paper we use Simulated Annealing (SA) -parallel, Vibrating Damp Optimization (VDO) -parallel and genetic algorithms for solving series-parallel RAP as we calculate system availability by UGF. Comparing the results with other algorithms, it seems that the presented algorithms are effective.

Nomenclature

\( n \) : Number of sub-systems
\( n_i \) : Number of component(s) in sub-systems \( i \)
\( H_i \) : Number of versions in sub-systems \( i \)
\( H_{ih} \) : Number of components for version \( h \) in sub-system \( i \)
\( M_i \) : Maximum of components variety in sub-system \( i \)
\( M_{s,i} \) : Maximum states in subsystem \( i \)
\( M_{s,0} \) : Maximum states for series-parallel system
\( \lambda_{ji} \) : Failure rate from state \( j \) to state \( i \)
\( \mu_{ji} \) : Repair rate from state \( j \) to state \( i \)
\( p_i(t) \) : The probability that component is in \( i \) in time \( T \)
\( k_i \) : Number of technical and organizational activities in sub-system \( i \)
\( k_{si} \) : Number of available organizational activities in sub-system \( i \)
\( Tk_{ih} \) : A binary variable which equals 1 when technical activity \( k \) in sub-system \( i \) is done
\( Tk_{ik} \) : A binary variable which equals 1 when organizational activity \( k \) in sub-system \( i \) is done
\( X_{ih} \) : Vector for component version \( h \) in sub-system \( i \)
\( X \) : Variables vector
\( a_{k,j,i} \) : Represented factor for effects of activity \( k \) on transition rate from state \( j \) to state \( i \)
\( g_{ih} \) : Operation rate according to state \( i \) for component version \( h \) in sub-system \( i \)
\( U_i(Z) \) : U function for sub-system \( i \)
\( U(Z) \) : U function for series-parallel system
\( \otimes \) : UGF operator
\( A \) : System availability
\( A_i \) : System availability constraint
\( D \) : Demand level

The remainder of this paper is organized as follows. Section 2 describes the system definition under study along with the related mathematical model. Section 3 presents the mathematical model. Section 4 presents the proposed solution procedures. The experimental design and the numerical example appear in this Section. Finally, the paper is concluded in Section 5.

2. System Definition

2.1. Assumptions

The assumptions of this paper regarding multi-state systems are as follows:

- Components are independent,
- In different states, components have different operational rates,
- Operation rate of a parallel sub-system is equal to the sum of operation rates of all components,
- Operation rate of a series-parallel system is equal to minimum operation rates of sub-systems.

2.2. State diagram and Markovian model:

In multi-state systems, the first thing to calculate is the state probability distribution that depends on failure and repair rates of components. The transition rates can be estimated by the historical data obtained from past failure and repair times. In this paper, we concentrate on the differential equation obtained from Markovian model. Random process models such as Markov and semi-Markov models can be used to illustrate a multi-state component as shown in Figure 1.
Assume that each component has \((M+1)\) state(s) and transitions are possible between two adjacent states. So a system of differential equations must be solved to determine the state probability distributions at the time \(t\).

The systems of differential equations are as follows:

\[
\frac{dp_i(t)}{dt} = -\mu_{i,1} p_i(t) + \lambda_m p_i(t) + \lambda_{2,1} p_1(t) + \lambda_{2,2} p_2(t) - (\lambda_{1,0} + \mu_{1,2}) p_i(t) + \lambda_{M-1,0} p_{M-1}(t),
\]

And the initial conditions are:

\[
p_i(0) = 0; \quad i = 1, 2, \ldots, M-1
\]

Now it is necessary to discuss the effects of technical and organizational activities on the system reliability. When technical or organizational activities are done, the transition rate changes in order to improve the system availability. Generally, with these activities, the failure rate decreases and the repair rate increases. Assume that \(TK_k\) is a binary variable showing whether activity \(k\) is done or not. Now to determine the new values of transition rate, we have:

\[
\lambda_{ij} = \alpha_{k,i,j} \times \lambda_{ij} \quad \mu_{ij} = \alpha_{k,i,j} \times \mu_{ij}
\]

In this paper two different kinds of activities are considered:

- Activities at the component level called technical activities,
- Activities at the sub-system level called organizational activities.

\(TK_k\) is the technical activity that presents activity \(k\) on component version \(h\) on sub-system \(i\). The cost of this type of activities depends on the number of affected components. \(TK_k\) is the organizational activity that presents activity \(h\) on sub-system \(i\) and affects all components in sub-system \(i\). The cost of this kind of activity is independent of the number of sub-system components.

2.3. System variables

For each sub-system, different versions of components are available. We consider that for a specific sub-system, different versions of components have the same available states. Each version of components is defined and categorized by operation rates of different states, transition rates and unit costs.

Each sub-system can have different kinds of components. In addition to the number of components, the type of components in sub-systems must be determined. The maximum variety of component types and component versions in sub-system \(i\) is \(H_i\). The number of component version \(h\) on sub-system \(i\) is \(n_{ih}\). So for sub-system \(i\), \(1 \leq i \leq N\) the variables \(n_{i1}, n_{i2}, \ldots, n_{ih}\) must be determined.

In addition, vector \(TK_{1h}, TK_{2h}, \ldots, TK_{kh}\) is used to illustrate the set of technical activities done on components version \(h\). \(TK_{kh}\) is a binary variable. If activity \(k\) for sub-system \(i\) is performed for components version \(h\) then \(TK_{kh} = 1\), otherwise \(TK_{kh} = 0\).

\[
X_{ih} = (n_{ih}, TK_{1h}, TK_{2h}, \ldots, TK_{kh})
\]

The system design variables vector related to components version \(h\) in sub-system \(i\), \(1 \leq i \leq N\) for \(1 \leq h \leq H\) shown as \(X_{ih}\) is as follows:

\[
X_{ih} = (n_{ih}, TK_{1h}, TK_{2h}, \ldots, TK_{kh})
\]

To illustrate the set of technical activities on sub-system \(i\), we use \(TK_{1i}, TK_{2i}, \ldots, TK_{ki}\). In this vector,
In a vector of variables, we can identify the availability of a multi-state system as follows:

- For each version of components in a sub-system, identify the transition rate based on technical and organizational activities,
- For each version of components in a sub-system, identify state distributions based on transition rates using Markov model,
- Identify the system operation distribution based on the state distribution using UGF.

2.4. System cost:

\[
C^i = C^i_{com} + C^i_{act}
\]

\[
C^i_{com} = C^i_0 + \sum_{k=1}^{N} n_i \cdot c^i_h
\]

\[
C^i_{act} = \sum_{k=1}^{K} TK^i_{k,k} (c^i_{k,h} + n_i \cdot c^i_h) + \sum_{k=1}^{k_i} \sum_{i=1}^{K} TK^i_{k} c^i_{p}\]

The total cost of the system is equal to the sum of sub-system costs as follows:

\[
C(X) = \sum_{i=1}^{N} C^i
\]

2.5. UGF method


With regard to multi-state systems, it should be pointed out that when we study a multi-state component, for each performance mode of component a probability is considered. This increases the system performance state, but the classic method cannot be used to calculate the system reliability. Therefore, to calculate the system reliability and availability, there was a tendency for using mathematical methods. Using the UGF method for analyzing a multi-state system decreases the system state very much [17]. We use the UGF method to calculate the system availability as follows:

\[
U^i(z) = p^i_{h0}(t)z^h + p^i_{h1}(t)z^{h1} + \ldots + p^i_{h,M_i}(t)z^{hM_i}
\]

\[
U(z) = \bigcap_{i} U^i(z) = \bigcap_{i} U^i(z) = U^i(z) = U^i(z) = U^i(z) = U^i(z) = U^i(z)
\]

\[
A(w) = \sum_{j=1}^{M} p_i \cdot \mathbb{1}(g_j \geq w)
\]

3. Multi-State Series-Parallel System Optimization Model

In this section, the multi-state series-parallel system optimization model with \( n \) series subsystem with parallel components is presented. The objective function is to minimize the system cost in order to satisfy the system availability.

\[
\text{Min } \quad C(X)
\]

\[
s.t. \quad A \geq A_0
\]

\[
n_i \geq 0 \quad ;1 \leq i \leq N, 1 \leq h \leq H_i
\]

\[
TK^i_{h} = 0 \quad ;1 \leq i \leq N, 1 \leq h \leq H_i, 1 \leq k \leq k_i
\]

\[
TK^i_{1} = 0 \quad ;1 \leq i \leq N, K_i + 1 \leq k \leq k_i + K_i
\]

4. Solving Approach

4.1. SA-Parallel algorithm

In 1983, Kirkpatrick et al. (1983) developed simulated annealing (SA) algorithm which is a single point Meta-heuristic algorithm. In this algorithm, first, we obtain a random solution and introduce it to a predefined neighborhood and get another solution as the neighbor of the old solution. We can normalize the objective function than the old solution, it replaces the old solution. But if the new solution does not have a better objective function than the old solution, it still has the opportunity to replace the old solution under a probability. This probability defined under an acceptance function is as follows:

\[
\text{Uniform } (0,1) < e^{-\frac{\Delta}{\Delta(t)}}
\]

In equation number (10), the temperature in each repeat of algorithm is \( t(t) \), and \( \Delta \) is defined as follows:

\[
\Delta = f_2 - f_1
\]

In equation number (11), \( f_2 \) is the new solution and \( f_1 \) is the old solution. We can normalize the \( \Delta \) as follows:
\[ \Delta = \frac{f_2 - f_1}{f_1} \]  \hspace{1cm} (12) 

Clearly, this acceptance function which depends on the temperature is each repeat \( t(it) \) of the SA algorithm. After this stage, the old solution remains or is replaced by the new solution. In both situations, the result is again sent to the neighborhood function to get a new solution and this procedure repeats till the algorithm satisfies the stop conditions.

Now we present an improved version of the SA present that is called SA-Parallel. In the SA-Parallel the algorithm starts with generating more than one point (random solution) and these solutions are sent to the neighborhood function to generate new solutions. The number of these points is \( npop \). Also to produce more improvement, we can generate more than one new solution from each old solution. We show the number of new solutions obtained from an old solution as \( nmove \). So, in each iteration of the algorithm, the number of new solutions is \( npop \times nmove \). Then we rank all new solutions and select the first \( npop \) solutions. These new best solutions are again sent to the neighborhood function to generate \( nmove \) new solutions. This procedure repeats until the algorithm satisfies the stop conditions. To compare the new solutions with the old ones, each new solution is compared with the same rank old solution.

The structures of the solutions if the algorithm is Meta-heuristic are as follows:

- Structure related to redundancy: \( Position.\text{Redundancy}.n \),
- Structure related to technical activities: \( Position.tkh \),
- Structure related to organizational activities: \( Position.tk \).

4.1.1. Neighborhood function:

One of the most important bases in the SA algorithm is neighborhood function. Regarding this function, all solutions are categorized into two categories. The first category contains the solutions under SWAP or REVERSION. The second category contains the solutions under MUTATION.

4.1.1.1. First category

**SWAP**

In the SWAP condition, for each old solution in the neighborhood function, two elements of solution are selected randomly and then replaced by each other as follows:

![Image](image1.png)

**REVERSE**

In this condition in each old solution, two random elements are selected and then the places of the elements between these two points are reversed as follows:

![Image](image2.png)

Thus in the operators of the first category, entering new variables is impossible and only the places of the variables changed. But in the second category, it is possible to enter a new variable into a solution.

4.1.1.2. Second category **MUTATION**

In this category, a new matrix named \( M \) is generated with the same rank of the solution matrix. In the new matrix, each array is filled with a random variable between 0 and 1. Then in the solution matrix, the variables with a correspondence amount less than \( 0.2 \) in matrix \( M \) are replaced with a random new variable as follows:

![Image](image3.png)

\( M = \begin{pmatrix} 0.0054 & 0.7312 & 0.121 & 0.0005 & 0.0005 & 0.423 & 0.0829 \\ 
0.7312 & 0.0005 & 0.423 & 0.0829 & 0.0005 & 0.7312 & 0.121 \\ 
0.121 & 0.423 & 0.0005 & 0.0829 & 0.7312 & 0.0005 & 0.423 \\ 
0.0005 & 0.0829 & 0.7312 & 0.0005 & 0.121 & 0.423 & 0.0005 \\ 
0.423 & 0.0005 & 0.7312 & 0.0829 & 0.121 & 0.0005 & 0.423 \\ 
0.0829 & 0.7312 & 0.0005 & 0.423 & 0.121 & 0.0005 & 0.423 \\ 
0.121 & 0.423 & 0.0005 & 0.0829 & 0.423 & 0.0005 & 0.423 \\ 
0.0005 & 0.0829 & 0.7312 & 0.0005 & 0.121 & 0.423 & 0.0005 \\ 
\end{pmatrix} \)

In each new set of solutions in the SA algorithm, some of the results are obtained from the first category and the other results from the second category. So we have a wide range of results in each iteration. In Figure 2, the pseudo code of the neighborhood function is presented.

The flowchart of the SA-Parallel algorithm is as follows in Figure 3:

\[ e = \text{Uniform}(0, 1) \]

\begin{align*}
& if \quad e \leq 0.5 \\
& \quad \text{SWAP} \\
& \quad \text{REVERSE} \\
& \text{Else} \\
& \quad \text{MUTATION} \\
& \text{End}.
\end{align*}

Fig. 2. The pseudo code of neighborhood function
4.2. VDO-Parallel algorithm

The VDO algorithm developed by Mahdizade and Tavakoli-Moghadam (2009) is a single point Meta-heuristic algorithm which is very similar to SA. In this paper the performance of this algorithm is evaluated in reliability problems. The main difference between SA and VDO is their acceptance function. This function in the VDO algorithm is as follows:

\[
P = 1 - e^{-\frac{A^2}{2\sigma^2}}
\]  

The parameter \( A \) in each iteration of the algorithm is updated as:

\[
A = A_0 e^{-\frac{\gamma t}{2}}
\]

Where \( A_0, \gamma \) and \( \sigma \) must be predetermined. The steps of the VDO-Parallel algorithm are similar to the SA-Parallel.

4.3. Parameter tuning

We use the RSM methodology for tuning the SA and VDO parameters. It is similar to problem 1 used by Levitin et al. (2009) in 2009. The results are presented in Figures 4 and 5 along with the information in Tables 1 and 2. Also stop condition in both algorithms is 30 generation without improve.

Table 1
SA-Parallel parameters

<table>
<thead>
<tr>
<th></th>
<th>Lower value</th>
<th>Upper value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n_{pop} )</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>( n_{move} )</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>( t_0 )</td>
<td>1000</td>
<td>10000</td>
</tr>
</tbody>
</table>

\( t_0 \) is the starting temperature.
Analysis of Variance for Cost

Table 2
VDO-Parallel parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Optimal value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Npop</td>
<td>5</td>
</tr>
<tr>
<td>Nmove</td>
<td>20</td>
</tr>
<tr>
<td>t0</td>
<td>1000</td>
</tr>
</tbody>
</table>

Table 3
VDO-Parallel parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Lower value</th>
<th>Upper value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Npop</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Nmove</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>t0</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>A0</td>
<td>0.01</td>
<td>0.001</td>
</tr>
<tr>
<td>( \gamma )</td>
<td>4</td>
<td>10</td>
</tr>
</tbody>
</table>

Analysis of Variance for Cost

Table 4
VDO-Parallel parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Optimal value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Npop</td>
<td>5</td>
</tr>
<tr>
<td>Nmove</td>
<td>20</td>
</tr>
<tr>
<td>t0</td>
<td>1000</td>
</tr>
</tbody>
</table>

Fig. 4. ANOVA for SA

Fig. 5. ANOVA for VDO
4.4. Numerical examples

To illustrate the method introduced above, 6 numeric examples are provided in this section. Assume a system with two series sub-systems. In sub-system one, three different versions of components are available and in the second sub-system, four different components are available. The components in the first sub-system have three working states of 0, 1 and 2 while the components in the second sub-system have two working states of 0 and 1. Transitions are only available in the adjacent states.

The operation rates, transition rates, component costs in each version of subsystem 1 are presented in Table 5. Assume that in the first sub-system, 8 different activities are available numbered from 1 to 8. The actions numbered 1 to 5 are technical activities while the activities numbered 6, 7 and 8 are organizational activities. The effects of technical activities in the three versions of components on the first subsystem are presented in Tables 6, 7 and 8. The pre cost of components in the first sub-system is fixed on 50.

For subsystem 2, the operation rates, transition rates, and component costs of all versions are reported in Table 9. Assuming that in the second sub-system 4 different activities are available, the actions numbered 1 to 3 are technical activities and activity number 4 is an organizational activity. The effects of technical activities in the four versions of components on the second subsystem are presented in Tables 10, 11, 12 and 13. The pre cost of components in the second sub-system is fixed on 60.

In this example, we assume that the operation rate of each sub-system is equal to the sum of the components operation rates and the operation rate of the system is equal to the minimum operation rates of the two sub-systems. We calculate the availability and cost of the system and only a unique demand level $W$ is available (Tian & Zuo, 2009).
Table 8
Costs and effects of actions on version 3 components in sub-system 1

<table>
<thead>
<tr>
<th>Action k</th>
<th>$c_{\text{kh}}^{T_i}$</th>
<th>$c_{\text{ho}}^{T_i}$</th>
<th>$c_{\text{kh}}^{T_i}$</th>
<th>$a_{k,1,0}$</th>
<th>$a_{k,1,2}$</th>
<th>$a_{k,0,1}$</th>
<th>$a_{k,1,2}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.1</td>
<td>1.0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0.4</td>
<td>1.5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3.1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0.0</td>
<td>4.2</td>
<td>0.9</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>2.0</td>
<td>0.5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.4</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>6.4</td>
<td>0.0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.2</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>8.0</td>
<td>0.0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.4</td>
<td>1.4</td>
</tr>
<tr>
<td>8</td>
<td>10.6</td>
<td>0.0</td>
<td>1</td>
<td>1</td>
<td>2.0</td>
<td>3.2</td>
<td></td>
</tr>
</tbody>
</table>

Table 9
Components data for sub-system 2

<table>
<thead>
<tr>
<th>Version h</th>
<th>$c_{h}^{T_i}$</th>
<th>$c_{\text{h1}}^{T_i}$</th>
<th>$\lambda_{1,0}$</th>
<th>$\mu_{0,1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30</td>
<td>80</td>
<td>0.05</td>
<td>0.30</td>
</tr>
<tr>
<td>2</td>
<td>35</td>
<td>100</td>
<td>0.06</td>
<td>0.35</td>
</tr>
<tr>
<td>3</td>
<td>60</td>
<td>150</td>
<td>0.03</td>
<td>0.45</td>
</tr>
<tr>
<td>4</td>
<td>80</td>
<td>180</td>
<td>0.02</td>
<td>0.40</td>
</tr>
</tbody>
</table>

Table 10
Costs and effects of actions on version 1 components in sub-system 2

<table>
<thead>
<tr>
<th>Action k</th>
<th>$c_{\text{kh}}^{T_i}$</th>
<th>$c_{\text{ho}}^{T_i}$</th>
<th>$c_{\text{ho}}^{T_i}$</th>
<th>$a_{k,1,0}$</th>
<th>$a_{k,0,1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.4</td>
<td>0.8</td>
<td>0.9</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.0</td>
<td>3.2</td>
<td>0.6</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1.8</td>
<td>2.4</td>
<td>1</td>
<td>2.1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>30</td>
<td>0.0</td>
<td>1</td>
<td>2.2</td>
<td></td>
</tr>
</tbody>
</table>

Table 11
Costs and effects of actions on version 2 components in sub-system 2

<table>
<thead>
<tr>
<th>Action k</th>
<th>$c_{\text{kh}}^{T_i}$</th>
<th>$c_{\text{ho}}^{T_i}$</th>
<th>$c_{\text{ho}}^{T_i}$</th>
<th>$a_{k,1,0}$</th>
<th>$a_{k,0,1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.4</td>
<td>0.8</td>
<td>0.9</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.0</td>
<td>3.2</td>
<td>0.6</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1.8</td>
<td>2.6</td>
<td>1</td>
<td>1.6</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>30</td>
<td>0.0</td>
<td>1</td>
<td>1.8</td>
<td></td>
</tr>
</tbody>
</table>

Table 12
Costs and effects of actions on version 3 components in sub-system 2

<table>
<thead>
<tr>
<th>Action k</th>
<th>$c_{\text{kh}}^{T_i}$</th>
<th>$c_{\text{ho}}^{T_i}$</th>
<th>$c_{\text{ho}}^{T_i}$</th>
<th>$a_{k,1,0}$</th>
<th>$a_{k,0,1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.0</td>
<td>3.2</td>
<td>0.6</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1.8</td>
<td>2.6</td>
<td>1</td>
<td>1.6</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>30</td>
<td>0.0</td>
<td>1</td>
<td>1.8</td>
<td></td>
</tr>
</tbody>
</table>

Table 13
Costs and effects of actions on version 4 components in sub-system 2

<table>
<thead>
<tr>
<th>Action k</th>
<th>$c_{\text{kh}}^{T_i}$</th>
<th>$c_{\text{ho}}^{T_i}$</th>
<th>$c_{\text{ho}}^{T_i}$</th>
<th>$a_{k,1,0}$</th>
<th>$a_{k,0,1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.0</td>
<td>3.2</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1.8</td>
<td>2.6</td>
<td>1</td>
<td>2.2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>30</td>
<td>0.0</td>
<td>1</td>
<td>1.6</td>
<td></td>
</tr>
</tbody>
</table>

Example 1
In this example, $w = 500$ and $A_0 = 0.9$. Table 14 present the results of genetic, SA-Parallel, and VDO-Parallel algorithms and the fitness convergence graphs of the SA-Parallel and VDO-Parallel are depicted in Figures 6.

Example 2
In this example, $w = 500$ and $A_0 = 0.95$. The results of genetic, SA-Parallel, and VDO-Parallel algorithms are presented in Table 15. Figures 7 show the fitness convergence graphs of the SA-Parallel and VDO-Parallel.
Table 14
The results of the algorithms in example 1

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>Version</th>
<th>Redundancy</th>
<th>Actions</th>
<th>Availability</th>
<th>Cost</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td>GA</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>6</td>
<td></td>
<td></td>
<td>GA</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>3</td>
<td>0</td>
<td></td>
<td></td>
<td>GA</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td></td>
<td>0.9231</td>
<td>531</td>
<td>GA</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td></td>
<td>0.9134</td>
<td>486.7</td>
<td>SA-Parallel</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td></td>
<td>0.9134</td>
<td>486.7</td>
<td>SA-Parallel</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td>VDO-Parallel</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1.23</td>
<td></td>
<td>VDO-Parallel</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td>VDO-Parallel</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td>2</td>
<td></td>
<td></td>
<td>VDO-Parallel</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
<td>VDO-Parallel</td>
</tr>
</tbody>
</table>

Fig. 6. Comparing the fitness convergence graph for the VDO-Parallel and SA-Parallel in example 1

Table 15
The results of the algorithms in example 2

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>Version</th>
<th>Redundancy</th>
<th>Actions</th>
<th>Availability</th>
<th>Cost</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td>GA</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>6</td>
<td>0.9510</td>
<td>559.2</td>
<td>GA</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>3</td>
<td>0</td>
<td></td>
<td></td>
<td>GA</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td></td>
<td>GA</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td>GA</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td>GA</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
<td>GA</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td>SA-Parallel</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>6</td>
<td>0.9514</td>
<td>538.8</td>
<td>SA-Parallel</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>3</td>
<td>0</td>
<td></td>
<td></td>
<td>SA-Parallel</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td></td>
<td>SA-Parallel</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td></td>
<td>SA-Parallel</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td>0</td>
<td></td>
<td></td>
<td>SA-Parallel</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
<td>SA-Parallel</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>VDO-Parallel</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td>VDO-Parallel</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td>VDO-Parallel</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td>VDO-Parallel</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td>VDO-Parallel</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
<td>VDO-Parallel</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
<td>VDO-Parallel</td>
</tr>
</tbody>
</table>
Example 3
In this example, $w = 500$ and $A_0 = 0.99$. The results of genetic, SA-Parallel, and VDO-Parallel algorithms are reported in Table 16 and the fitness convergence graphs of the SA-Parallel and VDO-Parallel are shown in Figures 8.

Table 16
The results of the algorithms in example 3

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>version</th>
<th>Redundancy</th>
<th>Actions</th>
<th>availability</th>
<th>Cost</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>GA</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>8</td>
<td>2,3,4</td>
<td>0.9906</td>
<td>627.1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>SA-Parallel</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>VDO-Parallel</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example 4
In this example, $w = 1000$ and $A_0 = 0.9$. The results of genetic, SA-Parallel, and VDO-Parallel algorithms are shown in Table 17 and the fitness convergence graphs of the SA-Parallel and VDO-Parallel are shown in Figures 9.
Table 17
The results of the algorithms in example 4

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>version</th>
<th>Redundancy</th>
<th>Actions</th>
<th>availability</th>
<th>Cost</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td></td>
<td></td>
<td>GA</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>6</td>
<td>3</td>
<td></td>
<td></td>
<td>GA</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td>GA</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>9</td>
<td>4</td>
<td></td>
<td></td>
<td>GA</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>1,4</td>
<td></td>
<td></td>
<td>GA</td>
</tr>
</tbody>
</table>

0.9038   940   GA

SA-Parallel

| 1         | 1       | 7          | 8       |              |      | SA-Parallel |
| 1         | 2       | 7          | 8       |              |      | SA-Parallel |
| 1         | 3       | 0          |         |              |      | SA-Parallel |
| 2         | 1       | 0          |         |              |      | SA-Parallel |
| 2         | 2       | 8          | 3       |              |      | SA-Parallel |
| 2         | 3       | 3          |         |              |      | SA-Parallel |
| 2         | 4       | 0          |         |              |      | SA-Parallel |

0.9020   905.8   SA-Parallel

VDO-Parallel

| 1         | 1       | 7          | 1,6,8   |              |      | VDO-Parallel |
| 1         | 2       | 8          | 1,6,8   |              |      | VDO-Parallel |
| 1         | 3       | 1          | 6,8     |              |      | VDO-Parallel |
| 2         | 1       | 1          |         |              |      | VDO-Parallel |
| 2         | 2       | 8          |         |              |      | VDO-Parallel |
| 2         | 3       | 1          |         |              |      | VDO-Parallel |
| 2         | 4       | 2          |         |              |      | VDO-Parallel |

0.90   294.9379   VDO-Parallel

Example 5
In this example, \( w = 1000 \) and \( A_0 = 0.95 \). The results of genetic, SA-Parallel, and VDO-Parallel algorithms are presented in Table 18 and the fitness convergence graphs of the SA-Parallel and VDO-Parallel are shown in Figures 10.

Example 6
In this example, \( w = 1000 \) and \( A_0 = 0.99 \). Table 19 show the results of genetic, SA-Parallel, and VDO-Parallel algorithms, and the fitness convergence graphs of SA and VDO algorisms are illustrated in Figures 11.
Table 18
The results of the algorithms in example 5

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>version</th>
<th>Redundancy</th>
<th>Actions</th>
<th>availability</th>
<th>Cost</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td></td>
<td></td>
<td>GA</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>8</td>
<td>8</td>
<td></td>
<td></td>
<td>GA</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>8</td>
<td></td>
<td></td>
<td>GA</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
<td>GA</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td>GA</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3,4</td>
<td></td>
<td></td>
<td>GA</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td>GA</td>
</tr>
</tbody>
</table>

| 1         | 1       | 0          | 1,6     |              |      | SA-Parallel |
| 1         | 2       | 10         | 5,6     |              |      | SA-Parallel |
| 2         | 1       | 1          | 3       |              |      | SA-Parallel |
| 2         | 2       | 2          | 1       |              |      | SA-Parallel |
| 2         | 3       | 9          |         |              |      | SA-Parallel |
| 2         | 4       | 0          | 1       |              |      | SA-Parallel |

| 1         | 1       | 6          | 2       |              |      | VDO-Parallel |
| 1         | 2       | 9          |         |              |      | VDO-Parallel |
| 1         | 3       | 1          | 4       |              |      | VDO-Parallel |
| 2         | 1       | 6          | 1       |              |      | VDO-Parallel |
| 2         | 2       | 8          |         |              |      | VDO-Parallel |
| 2         | 3       | 1          |         |              |      | VDO-Parallel |
| 2         | 4       | 0          |         |              |      | VDO-Parallel |

| 1         | 1       | 2          | 1,2,3,6,8 |              |      | GA        |
| 1         | 2       | 7          | 1,6,8     |              |      | GA        |
| 1         | 3       | 3          | 6,8       |              |      | GA        |
| 2         | 2       | 8          | 4         |              |      | GA        |
| 2         | 3       | 3          | 4         |              |      | GA        |
| 2         | 4       | 1          | 2,3,4     |              |      | GA        |

| 1         | 1       | 0          | 2,5,6     |              |      | VDO-Parallel |
| 1         | 2       | 0          | 1,2,3,5,6 |              |      | VDO-Parallel |
| 1         | 3       | 10         | 1,3,4,5,6 |              |      | VDO-Parallel |
| 2         | 1       | 10         | 1,3,4     |              |      | VDO-Parallel |
| 2         | 2       | 10         | 2,4       |              |      | VDO-Parallel |
| 2         | 3       | 10         | 2,4       |              |      | VDO-Parallel |
| 2         | 4       | 9          | 2,3,4     |              |      | VDO-Parallel |

Table 19
The results of the algorithms in example 6

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>version</th>
<th>Redundancy</th>
<th>Actions</th>
<th>availability</th>
<th>Cost</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>6</td>
<td>1,8</td>
<td></td>
<td></td>
<td>GA</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>9</td>
<td>8</td>
<td></td>
<td></td>
<td>GA</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>GA</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>8</td>
<td>1,4</td>
<td></td>
<td></td>
<td>GA</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>5</td>
<td>3,4</td>
<td></td>
<td></td>
<td>GA</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
<td>GA</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>GA</td>
</tr>
</tbody>
</table>

| 1         | 1       | 2          | 1,2,3,6,8 |              |        | SA-Parallel |
| 1         | 2       | 7          | 1,6,8     |              |        | SA-Parallel |
| 1         | 3       | 3          | 6,8       |              |        | SA-Parallel |
| 2         | 2       | 8          | 4         |              |        | SA-Parallel |
| 2         | 3       | 3          | 4         |              |        | SA-Parallel |
| 2         | 4       | 1          | 2,3,4     |              |        | SA-Parallel |

| 1         | 1       | 0          | 2,5,6     |              |        | VDO-Parallel |
| 1         | 2       | 0          | 1,2,3,5,6 |              |        | VDO-Parallel |
| 1         | 3       | 10         | 1,3,4,5,6 |              |        | VDO-Parallel |
| 2         | 1       | 10         | 1,3,4     |              |        | VDO-Parallel |
| 2         | 2       | 10         | 2,4       |              |        | VDO-Parallel |
| 2         | 3       | 10         | 2,4       |              |        | VDO-Parallel |
| 2         | 4       | 9          | 2,3,4     |              |        | VDO-Parallel |

![Fig. 11. Comparing the fitness convergence graph for the VDO-Parallel and SA-Parallel in example 6](image-url)
As mentioned above, in example 1, the results of SA-Parallel and VDO-Parallel algorithms are the same and both of them are better than the genetic algorithm. The only difference between the two parallel algorithms is the number of iterations. In the SA-Parallel, after 45 iterations the algorithm reaches convergence but the number of iterations in the VDO-Parallel is 53. In the examples number 2, 4, 5 and 8, the VDO-Parallel yields better results than the SA-Parallel algorithm. Only in example number 3, the results of the SA-Parallel are better than those of the VDO-Parallel algorithm. Of course in all examples, both of the proposed parallel algorithms have better results than the genetic algorithm.

5. Conclusion and Further studies

All in all, in this paper we showed that parallel meta-heuristic algorithms like SA-parallel and VDO-Parallel have a better performance in the series-parallel multi-state systems than the ordinary ones such as genetic. Differences between the acceptance functions of SA and VDO algorithms enable them to avoid local solutions. Using this approach for the SA-parallel and VDO-Parallel improves the ability of these two algorithms. Of course, operation of these two algorithms depends on neighborhood function structures. Using the approach, one can improve the ability of other meta-heuristic algorithms.

In this paper we determined these decision variables: 1) Number of assigned components in each sub-system of each system, 2) The version of assigned components in each sub-system of each system, and 3) The type of technical and organizational activities.

References