یازدهمین کنفرانس بین‌المللی مهندسی صنایع
۱۷ تا ۱۸ دیماه ۱۳۹۳
11th IIEC International Industrial Engineering Conference 7-8January 2015

Multi-Objective Reliability-Redundancy Allocation for Non-Exponential Multi-State Repairable Components

Ahmad Attar[a], Sadigh Raissi[b], Kaveh Khalili-Damghani[c]
Industrial Engineering Faculty, Islamic Azad University, South Tehran Branch
Tehran, Iran
[a] St_a_attar@azad.ac.ir, [b] Raissi@azad.ac.ir, [c] k_khalili@azad.ac.ir

***Abstract*— In recent years, little work has been done over the joint reliability-redundancy allocation and in all of that, exponential failure and repairs are assumed. In this paper we focus on investigating the reliability-redundancy allocation where the repair failure distributes are non-exponential. When failure/repair times follow a distribution other than exponential, analytical methods become too complicated or even useless. In these cases simulation methods like Mote Carlo and discrete event simulations are typically preferred, but they usually take relatively long time to estimate the availability of the system. This weakness gets more important when we want to use them in optimization algorithms. Here we offer a new discrete event computer simulation package which can estimate availability values in a reasonable time. Then we combine the mentioned simulation method with genetic algorithm to obtain optimal designs. These methods are illustrated with their application using some numerical examples.**

Keywords- Reliability Optimization; Non-Exponential; Multi-State; Reparability; Simulation; Multi-Objective.

I. Introduction and Background

Reliability Optimization and designing reliable systems with acceptable availability has always been a very important task for managers and engineers and that is because of the close relationship between reliability/availability and other concepts like income, quality and safety [1]. Many researchers have discussed the reliability optimization problem with different assumptions and each from certain point of view. However, in general we can distinguish three different approaches for this problem: (i) reliability allocation, (ii) redundancy allocation, and (iii) joint reliability-redundancy allocation.
The first type aims at determining the best reliability specification for all components in the system without making any changes in system structure [2-5]. While the second approach tries to determine the optimal number of redundant for each stage. This approach has been used by a number of papers such as [6-7].

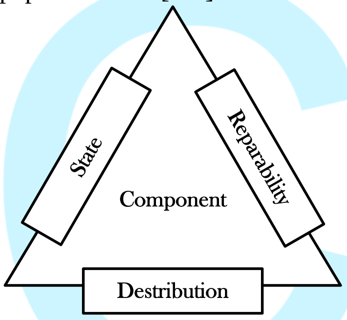


Fig. 1 Characterestics triangle

However, as explained in [8] , each of these approaches can only result in partial optimizations. Thus, Misra and Lobojevic first introduced the third approach by combining the previews allocation models. So this type of optimization utilizes the both options simultaneously, i.e. adjusting components' availability and determining the number of components in subsystems.
For each of these allocation methodologies, components have three major characteristics, namely component state, reparability, and Failure/Repair distribution. We call this the "characteristics triangle" (Fig 1). The number of states available for the component forms the first side of this triangle. Each component has at least two possible states: perfect functioning and complete failure. Engineering system can usually take some medial states in addition to these basic states. Components with only two possible states are commonly called binary state and those with more than two states are called multi-state components [1]. On the next side, we determine whether the component is repairable or not. Failure and repair

distributions are the other important attribute of a component that is indicated on the last side of this triangle. In reliability theory, Exponential, normal, lognormal and weibull are the commonly used life and repair distributions.

Among all the possible combinations of these characteristics, authors have always focused on some specific and simplified assumptions. For instance, in joint reliability-redundancy allocation, Misra and Lobojevic [8] discussed binary state and repairable components. Since then, many papers were limited to those characteristics and except for very few papers like [10], all the studies in binary state components neglected the repair option. Tian et.al. [11], for the first time, considered multi-state components with this approach and in [12] this problem was extended to include repairable components. Recently, Hamedani and Khorshidi [13] investigated the model formulated in [12]using time value of money. The last two references applied a practical approach for the joint reliability-redundancy optimization. In this new approach, instead of determining the failure and repair rates directly as design variables, they used technical and organization actions to promote the component failure and repairs distribution [13]. The key idea behind this is that actions like installing a condition monitoring system or applying certain maintenance program can affect the state transition rates (or distributions) of the multi-state component. Therefore this methodology involves three factors that can affect the system overall availability: (1) different component versions for each subsystem, (2) redundancy, (3) technical and organizational actions [12].

Back to the characteristics triangle, failure and repair distributions in reliability-redundancy optimization have always been considered to be exponential and other distributions have been neglected in this field. The main reason for this simplification is to use Marcov method to calculate components' state distribution and duo to this fact that common analytical methods can become too complicated or even inappropriate when dealing with more realistic assumptions. On the contrary, Bowles [14] pointed out the consequences of using such simplifications. He compared the reliability of an example system with weibull distribution in both original and simplified exponential conditions (with the same mean values) and concluded that such supposition can lead to either over estimating or underestimating the overall system reliability.

Monte Carlo (MC) and discrete event simulation (DES) are two possible alternatives to handle more realistic assumptions that support almost all distributions. Both of these methods have been used reliability and availability estimations on the presents of non-exponential distributions. For example, Marsequerra et.al. [9] proposed a MC simulation method for binary state non-exponential components. More recently, Lins and Droguett [5]offered a DES method for availability estimation with repairable binary state systems.

The main challenge for using simulation methods is that basically these are much more time consuming than the traditional analytical methods. This becomes more important when they are used in population based optimization algorithms like genetic algorithm (GA). Given that, in each iteration of these algorithms we have to evaluate the availability for all chromosomes which comes to several thousands of availability estimations.

In order to overcome this difficulty, both [9] and [5] have proposed methods to decrease the simulation time. The first reference proposed "drop-by-drop" simulation which is based on this fact that good chromosomes usually appear in successive generations (Elitism). They estimate the reliability of all chromosomes in every iteration with a relatively low accuracy and record them. These recorded values are updated in the next iteration if the same chromosome was selected for the new population. Consequently, the best chromosomes will gradually obtain reliability estimates with the desired accuracy. For this reason, they have analogized this process to filling a glass of water. Nevertheless, this method always suffers from unlikeness in the accuracy of the estimated reliability values among different chromosomes of a single population which causes imprecise comparisons and selections.

On the other hand, Lins and Droguett [5] divided the mission time into 30 steps and evaluated the system at the end of each step to avoid continuous calculations and reduce the simulation time. But limiting the evaluations to this relatively low number of steps will significantly affect the accuracy of the estimates. Moreover, regardless to the mentioned issues, these two methods, like other simulation methodologies offered for this, sill take at least few seconds to estimate the reliability/availability of an average system [15].

Here we focus on offering a multi-objective version the reliability-redundancy model proposed in [12] and extending it from the third side of the triangle to support non-exponential and more realistic distributions for multi-state repairable components.

We introduce a new DES for the series-parallel system in a relatively new simulation software which decreases the simulation time without suffering from the above mentioned issues, so it offers continuous availability check and homogeneous estimation accuracy at the same time. Furthermore, we combine the proposed DES with GA to get the optimum design.

The rest of the paper is organized as follows: in the next section we explain the designed availability estimation package and validate it using some analytical examples. Section 3 contains the mathematical model. The meta-heuristic algorithm is explained in section 4. An illustrative example is provided and discussed in section 5. And finally, some concluding remarks are given in section 6.

II. Discrete Event Computer Simulation

For the first time in reliability optimization, we take advantage of a relatively new simulation software named “Enterprise Dynamics (ED)”. This software is offers object oriented simulation modeling and excellent 2D and 3D environments that help modeling and understanding complex real systems. Here, we design new atoms (ED objects) for this software to add multi-state availability estimation capability to it. The designed package includes three atoms: *Multi-state Server*, *Subsystem atom*, and *Availability Monitor*.

A. Multi-state Server

This atom models the multi-state components. There are two major events that happen in multi-state components: Failure event, Repair event. The failure event is the event through which the component goes to a lower performance rate. Improving the component condition and restoring its performance rate is called repair event. These failure and repair events occur within random failure and repair times respectively and in each state, they follow certain distributions. For state j, we denote the failure and repair distributions by TTF_j and TTR_j respectively. **Fig. 2** illustrates the occurrence of these events for an example multi-state component.

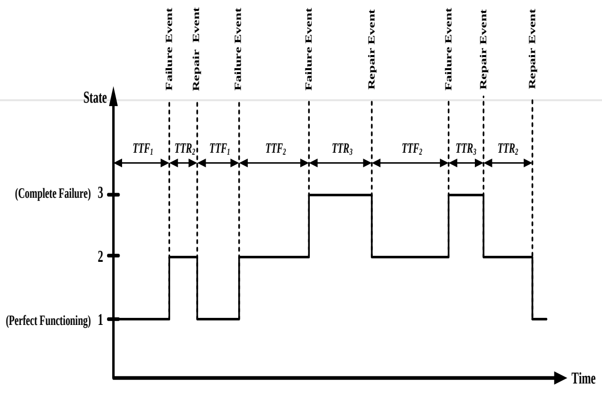


Fig. 2 Failure and repair events occurrence over time in a multi-state component

We imitate this process in the *Multi-state Server* atom by defining two events in the event handler of this object

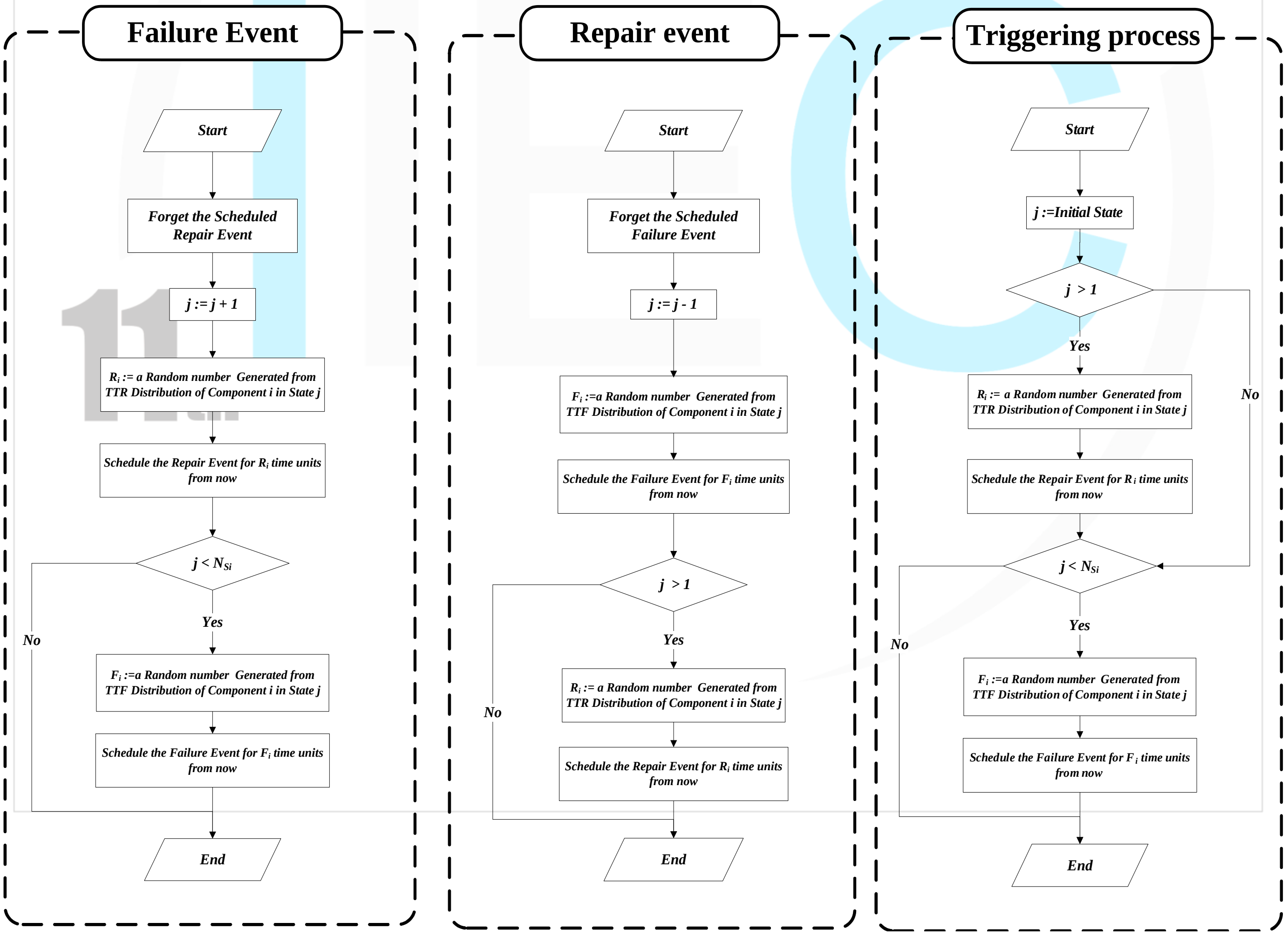

Failure Event
Start
Forget the Scheduled Repair Event
j := j + 1
Ri := a Random number Generated from TTR Distribution of Component i in State j
Schedule the Repair Event for Ri time units from now
j < NSi
Yes
No
Fi :=a Random number Generated from TTF Distribution of Component i in State j
Schedule the Failure Event for Fi time units from now
End
Repair event
Start
Forget the Scheduled Failure Event
j := j - 1
Fi :=a Random number Generated from TTF Distribution of Component i in State j
Schedule the Failure Event for Fi time units from now
j > 1
Yes
No
Ri := a Random number Generated from TTR Distribution of Component i in State j
Schedule the Repair Event for Ri time units from now
End
Triggering process
Start
j :=Initial State
j > 1
Yes
No
Ri := a Random number Generated from TTR Distribution of Component i in State j
Schedule the Repair Event for Ri time units from now
j < NSi
Yes
No
Fi :=a Random number Generated from TTF Distribution of Component i in State j
Schedule the Failure Event for Fi time units from now
End

Fig. 3 Failure and repair events and the triggering flowcharts in Multi-state Server

(Fig. 3). As seen in these flowcharts, both events reschedule the failure and repair events for the future and form an endless loop of failures and repairs, which is exactly what happens in real components. All we have to do is to trigger this loop. But this triggering process has to be done based on the initial states assumed for the components (Fig. 3). Note that we trigger the loop only once and at the very first moment of the simulation. In addition, we designed a 2D representation for this atom and included some indicators to visually monitor components' conditions. One of the advantages of this software and the designed atom is the resemblance of its 2D layout to usual reliability diagrams. For an example series-parallel system, Fig. 4 compares the ordinary diagram and the corresponding simulation model.

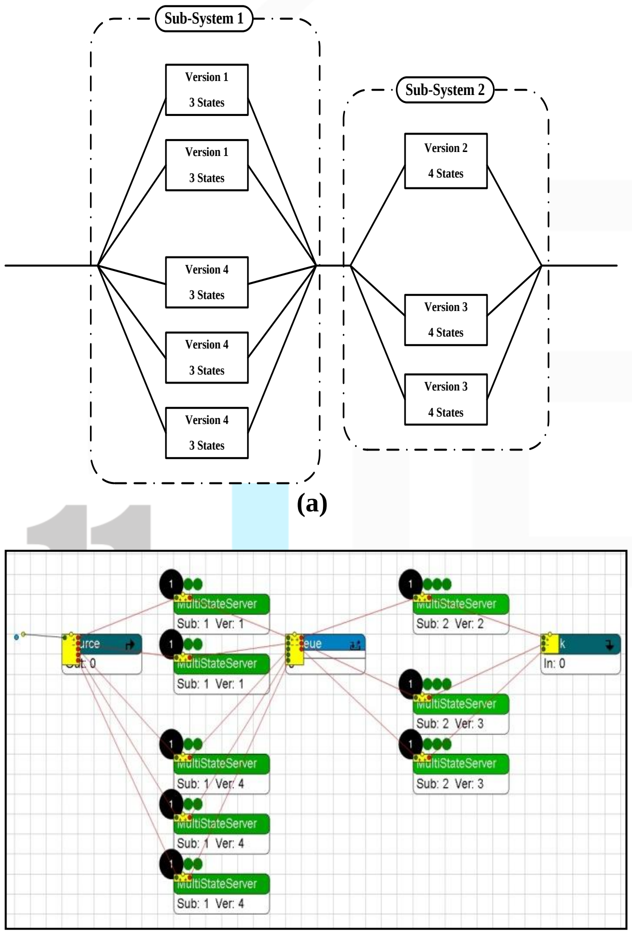


Fig. 4 an example series-parallel diagram and the related ED model

B. *Subsystem atom*

The main intentions of creating this atom is to link parallel components, aggregate the total performance rates/outputs of each subsystem, and determine its condition (up or down) regarding to the desired demand. In order to connect each *Multi-state Server* to the related subsystem, we just need to connect its *central* channel to an *input* channel on the subsystem.

C. *Availability Monitor*

This atom is where we calculate the availability of the system. In many cases we are mainly concerned with estimating the availability for systems that will run for a long time [1]. Hence, many papers with analytical approaches have considered this type of availability for their optimization model [12-13]. Eq. 1 is a common formula for the asymptotic or steady-state availability which uses the instantaneous availability concept to define the long run availability.

$$A_s = \lim_{t\to\infty} A(t) \tag{1}$$

However, steady-state availability has been neglected in most of the simulation based reliability studies. As seen in the formula, to achieve this type of availability using typical simulation models, we have to perform significantly long runs of simulation to get the instantaneous availability in the steady-state and this has always been impractical. For this reason, all simulation based studies have focused on small *t* values.

In this paper, we use another type of availability to overcome this limitation. There is an availability formula that is used for real running systems, called "Achieved Availability" [16]. In real systems there is no replication to get the instantaneous availability, but we have the down and up time that has actually happened. So the availability can be given by [16]:

$$A_A = \frac{Up\ time}{Total\ Time} \tag{2}$$

Since the simulation model imitates the real system behavior, we calculate and report A_A in this atom. Moreover, unlike the DES presented in [13], we take a continuous evaluation of the system availability and as seen Fig. 5 this atom shows the availability in real time.

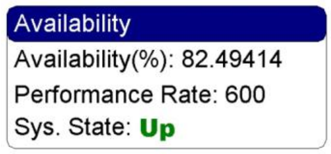

Availability
Availability(%): 82.49414
Performance Rate: 600
Sys. State: Up

Fig. 5 2D representation of the availability monitor atom in ED

For the real time evaluations, we use a signaling function in *Multi-state Servers*, so when it changes states, it triggers the evaluations and system state and total up and down times are updated. This way, we offer the real time availability with the maximum accuracy without the need to spending CPU-time over unnecessary evaluations.

Furthermore, this type of availability also converges to the steady-state availability value [16]:

$$A_S = \frac{Up\ Time}{Up\ Time + Down\ Time} \tag{3}$$

D. *Model validation*

Here we check the validity of the proposed simulation package using a numerical example from Ref. [12]. The configuration of this example system is given in Table 1. The desired demand is 1000 and the reported availability for this example is %95.39 which is derived by using Marcov and Universal generating function (UGF) methods in the reference paper.

Table 1 Validation example settings taken from [12]

# SubSystem	Version	Redundancy	Actions
1	1	5	{8}
	2	8	{8}
	3	1	{8}
2	1	4	{4}
	2	2	{4}
	3	2	{3,4}
	4	2	{4}

After setting the parameters in the related subsystem atoms, creating the simulation model, we perform 30 sample runs of this model and reported the data in Table 2. The duration of each run is considered to be 10,000 time units to make sure that all samples are taken at the steady-state conditions. Note that, each of these long simulation runs has just taken one second using this new simulation package that is much less than the time reported by other simulation methods in the literature [15].

Table 2 The estimated availabilities for the validation example

Estimate Availabilities					
0.8952	0.8946	0.8954	0.8944	0.8883	0.8929
0.8911	0.8940	0.8981	0.8927	0.8932	0.8912
0.8947	0.8910	0.9002	0.9003	0.8911	0.8934
0.8886	0.8947	0.8844	0.8892	0.8920	0.8998
0.8932	0.8953	0.8985	0.8912	0.8890	0.8851

In order to compare these results with the percentage reported by [12], we undertake a one sample t-test. This statistical test checks whether the mean value of the simulated availabilities is equal to the one reported by the reference. In other words, the following hypotheses are evaluated:

$$H_0\colon \mu = \%95.39 \text{ and } H_1\colon \mu \neq \%95.39$$

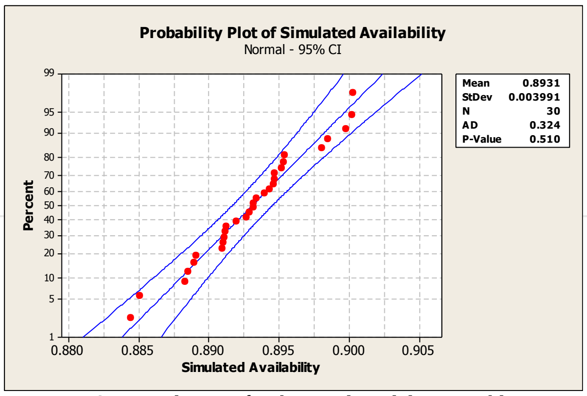
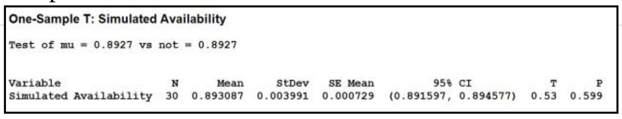


Fig. 6 Normality test for the simulateed data in Table 2

This test requires normal data, so we performed normality test for the simulated data (Fig. 6).The hypothesis test resulted in p-value of 0.599 (Fig. 7), which indicates that there is no sufficient evidence to reject H_0 and the simulation results are statistically acceptable.



```
One-Sample T: Simulated Availability

Test of mu = 0.8927 vs not = 0.8927


Variable                 N      Mean     StDev   SE Mean              95% CI          T      P
Simulated Availability  30  0.893087  0.003991  0.000729  (0.891597, 0.894577)  0.53  0.599
```

Fig. 7 t-test results for the validation example

III. Mathematical model

Here we propose a multi-objective version of the model presented in [12]:

$$Min\,\{1 - A(X)\,, C(X)\} \tag{4}$$

S.t.

$$A(X) \geq A_0 \tag{5}$$

$$C(X) \leq C_{Max} \tag{6}$$

$$n_{ih} \geq 0 \qquad ,(1 \leq i \leq N\,, 1 \leq h \leq H_i) \tag{7}$$

$$TK_{hk}^{i} = 0 \text{ } or \text{ } 1\,,(1 \leq i \leq N\,, 1 \leq h \leq H_i) \tag{8}$$

$$TK_{k}^{i} = 0 \text{ } or \text{ } 1 \quad ,(1 \leq i \leq N\,) \tag{9}$$

Where the design variable vector is denoted by X and is defined in Eq. (10). $A(X)$ and $C(X)$ are the availability and cost for the designed system. Cost and availability limits are symbolized by $C_{Max}$and A_0 respectively.

$$\begin{aligned} X = (&X_{11}, \dots, X_{1H_1}, TK_{K_1+1}^{1}, \dots\,, TK_{K_1+K_{S1}}^{1} \\ &, X_{21}, \dots, X_{2H_2}, TK_{K_2+1}^{2}, \dots\,, TK_{K_2+K_{S2}}^{2}, \\ &X_{N1}, \dots, X_{NH_N}, TK_{K_N+1}^{N}, \dots\,, TK_{K_N+K_{SN}}^{N}) \end{aligned} \tag{10}$$

$$X_{ih} = (n_{ih}, TK_{h1}^{i}, TK_{h2}^{i}, \dots, TK_{hk_i}^{i}) \tag{11}$$

Here TK_{hk}^{i} is a Boolean variable and represents performing action k on version h components of subsystem i. Similar to [12] and [13] here we take into account k_i component level actions for subsystem i , and k_{S_i} is the number of subsystem level actions.

Total cost of a parallel-series system is the summation of subsystems' costs.

$$C(X) = \sum_{i=1}^{N} C^i \tag{12}$$

$$C^i = C_{Com}^{i} + C_{act}^{i} \tag{13}$$

Subsystem cost includes the costs associated with components (C_{Com}^{i}) and costs of applying technical and organizational actions (C_{act}^{i}). Where the component cost consists of the related fixed cost of installing components in the subsystem (C_0^i) and the installation/purchase cost of each component. Action cost of subsystem i is given on Eq. (15). The first part of this equation represents the fixed (CT_{hk0}^{i}) and variable (CT_{hk}^{i}) costs of applying component level actions and the rest is related to the subsystem level actions.

$$C_{Com}^{i} = C_0^i + \sum_{h=1}^{H_i} n_{ih} C_h^i \tag{14}$$

$$\begin{aligned} C_{act}^{i} = &\sum_{k=1}^{K_i} TK_{hk}^{i} \left(CT_{hk0}^{i} + n_{ih} CT_{hk}^{i}\right) \\ &+ \sum_{k=K_i+1}^{K_i+K_{S_i}} TK_k^i CT_{k0}^{i} \end{aligned} \tag{15}$$

We utilize the weighting technique used by [5] to transform the above optimization model into a single objective model. In this case the objective function is presented by Eq. 16.

$$Min\ \ z = w_1\big(1 - A(X)\big) + w_2 C(X) \qquad (16)$$

w_1 and w_2 are arbitrary values that can be determined by the designer. Otherwise we can offer a set of Pareto optimal solutions by checking different weight values.

IV. Combining ED with Genetic Algorithm

In this paper we use genetic algorithm to solve the optimization model. This method has been used by many researchers and is the most applied meta-heuristic method in the field of reliability optimization [12-13]. One of the most important parts of utilizing this algorithm is to define the solution representation. We define an N parted chromosome for this problem, which contains all n_{ih} and TK_{hk}^{i} and TK_{k}^{i} variables. One of the challenges that have estranged researchers from using universal simulation programs (like ED) in reliability optimization is connecting the simulation software to the software in which the optimization algorithm is coded. We take advantage of the AxtiveX standard to transfer commands and data between ED and Matlab (where we coded GA). The following steps are taken for this connection:

(i) Establishing the ActiveX link and loading the necessary files in ED,
(ii) Converting the chromosome into the acceptable form for ED,
(iii) Setting the converted design in the related atoms,
(iv) Creating the simulation model and setting the speed and simulation time,
(v) Starting the simulation run and waiting for its completion,
(vi) Reading the availability from the simulation software.

Except for the first step which is only done once at the very initial iteration of the algorithm, Matlab is programmed to perform these steps every time that we have to estimate the availability of a new chromosome, i.e. hundreds of times in each iteration.

V. Illustrative Example

In this section, we investigate a non-exponential version of the numerical example discussed in [12] and [13]. Like the reference example, we assume a system with two subsystems, three component versions for the first subsystem and four versions for the components in the second subsystem. Number of actions is considered to be 8 and 4 for subsystem 1 and 2 respectively. C_0^i is assumed 50 and 60 for these subsystems.

In subsystem 1, actions 1-5 are component level actions and the rest is subsystem level action. Actions {1,2}, {3,4} and {6,7,8} in this subsystem are considered mutually exclusive. Table 3 contains the performance rates, failure and repair distributions in different states and the unit cost of the components in this subsystem. The effect of the actions on component versions 1-3 for subsystem 1 are given in Tables 4 to 6.

Table 4 Action effects over version 1 components of subsystem 1

Action type	l	FC_{jl}^{i}	VC_{jl}^{i}	AF_{l1}	AF_{l2}	AR_2	AR_3
Component level	1	0.5	5	1.1	1	1	1
	2	2	7.5	1.25	1	1	1
	3	4	15.5	1.25	1.1	1	1
	4	0	20	1.4	1.25	1	1
	5	10	2	1	1	1	0.7
Subsystem level	1	32	0	1	1	0.7	0.8
	2	40	0	1	1	0.7	0.7
	3	53	0	1	1	0.35	0.5

Table 5 Action effects over version 2 components of subsystem 1

Action type	l	FC_{jl}^{i}	VC_{jl}^{i}	AF_{l1}	AF_{l2}	AR_2	AR_3
Component level	1	0.5	5	1	1	1	1
	2	2.5	7.5	1	1	1	1
	3	4.5	15.5	1.25	1	1	1
	4	0	20	1.4	1.1	1	1
	5	10	2	1	1	1	0.6
Subsystem level	1	32	0	1	1	1	0.8
	2	40	0	1	1	0.8	0.6
	3	53	0	1	1	0.5	0.4

Table 3 parameters of components of subsystem 1

$Ver.$ j	PC_{j1}	R_{j1}^{1}	R_{j2}^{1}	TTF		TTR	
				State 1	State 2	State 2	State 3
1	20	600	300	$Weibull(19,1.5)$*	$Weibull(24,1.8)$	$Weibull(4,1.7)$	$Weibull(6,1.9)$
2	25	1000	500	$Weibull\ (10,1.9)$	$Weibull(11,1.7)$	$Weibull(5,1.4)$	$Weibull(7,2.1)$
3	40	1200	600	$Weibull(15,1.2)$	$Weibull(17,0.9)$	$Weibull(3,1.1)$	$Weibull(6,1.2)$

* Weibull (λ , k); λ : Scale parameter, k : Shape parameter

Table 6 Action effects over version 3 components of subsystem 1

Action type	l	FC_{jl}^i	VC_{jl}^i	AF_{l1}	AF_{l2}	AR_2	AR_3
Component level	1	0.5	5	1	1	1	1
	2	2	7.5	1	1	1	1
	3	5	15.5	1	1	1	1
	4	0	21	1	1.1	1	1
	5	10	2.5	1	1	1	0.7
Subsystem level	1	32	0	1	1	0.8	1
	2	40	0	1	1	0.7	0.7
	3	53	0	1	1	0.3	0.5

For subsystem 2, only one subsystem level action is assumed (action 4) and here all of the actions are independent. The number of states available for the components in subsystem 1 and 2 is assumed 3 and 2 respectively (exactly like [12]).

Table 7 Components specifications for subsystem 2

$Ver. j$	PC_{j2}	R_{j1}^2	TTF	TTR
1	30	800	$Weibull(20,2.2)$	$Weibull(7,3.1)$
2	35	1000	$Weibull(16,2.8)$	$Weibull(15,0.35)$
3	60	1500	$Weibull(30,2.5)$	$Weibull(40,0.40)$
4	80	1800	$Weibull(50,1.9)$	$Weibull(28,0.30)$

For each version of components in this subsystem, Table 7 presents the performance rates, failure and repair distributions and the unit cost and the effect of the actions is given in the following tables:

Table 8 Action effects for version 1 components in subsystem 2

Action type	l	FC_{jl}^i	VC_{jl}^i	AF_{l1}	AR_2
Component level	1	2	4	1.1	1
	2	0	16	1.7	1
	3	9	12	1	0.45
Subsystem level	1	1 0	0	1.1	0.4

Table 9 Action effects for version 2 components in subsystem 2

Action type	l	FC_{jl}^i	VC_{jl}^i	AF_{l1}	AR_2
Component level	1	2	5	1.1	1
	2	0	16	1.6	1
	3	9	14	1	0.5
Subsystem level	1	150	0	1	0.45

Table 10 Action effects for version 3 components in subsystem 2

Action type	l	FC_{jl}^i	VC_{jl}^i	AF_{l1}	AR_2
Component level	1	2	5	1	1
	2	0	16	1.3	1
	3	9	13	1	0.6
Subsystem level	1	150	0	1	0.55

Table 11 Action effects for version 4 components in subsystem 2

Action type	l	FC_{jl}^i	VC_{jl}^i	AF_{l1}	AR_2
Component level	1	2	5	1	1
	2	0	16	1.3	1
	3	9	13	1	0.45
Subsystem level	1	150	0	1	0.6

Like [12] and [13], for this example the population size is to 100 and we consider 200 iterations in the genetic algorithm. Besides, w_1 and w_2 values are supposed to be 100 and 0.25. and the demand is considered A_0 and C_{max} are considered to be 0.9 and 600 respectively. The following table shows the optimum design for this example.

Table 12 solution of the illustative example

# SubSystem	Version	Redundancy	Actions
1	1	1	{8}
	2	4	{8}
	3	1	{8}
2	1	0	
	2	1	{3}
	3	3	
	4	0	

VI. CONCLUSION

In this paper, we investigated the multi-objective joint reliability-redundancy allocation problem for multi-state series-parallel systems with repairable components and non-exponential distributions. We presented a new discrete event simulation package for Enterprise Dynamics simulation software that offered the availability estimations in a reasonable CPU-time. Then we combined this simulation method with genetic algorithm to get the best design for the problem. ActiveX programming standards were used for this purpose. We investigated the methods presented in this paper using a numerical example.

REFERENCES

[1] Xie, M., Dai, Y., & Poh, K. (2004). Computing System Reliability Models and Analysis, Springer

[2] Ramirez-Marquez, J. E., & Coit, D. W. (2004). A heuristic for solving the redundancy allocation problem for multi-state series-parallel systems. *Reliability Engineering & System Safety*, *83*(3), 341-349.

[3] Tian, Z., & Zuo, M. J. (2006). Redundancy allocation for multi-state systems using physical programming and genetic algorithms. *Reliability Engineering & System Safety*, *91*(9), 1049-1056

[4] Li, C. Y., Chen, X., Yi, X. S., & Tao, J. Y. (2010). Heterogeneous redundancy optimization for multi-state series–parallel systems subject to common cause failures. *Reliability Engineering & System Safety*, *95*(3), 202-207.

[5] Lins, I. D., & Droguett, E. L. (2011). Redundancy allocation problems considering systems with imperfect repairs using multi-objective genetic algorithms and discrete event simulation. *Simulation Modelling Practice and Theory*, *19*(1), 362-381.

[6] Yalaoui, A., Chu, C., & Châtelet, E. (2005). Reliability allocation problem in a series–parallel system. *Reliability engineering & system safety*, *90*(1), 55-61.

[7] Tian, P., Wang, J., Zhang, W., & Liu, J. (2009, May). A fault tree analysis based software system reliability allocation using genetic algorithm optimization. In *Software Engineering, 2009. WCSE'09. WRI World Congress on* (Vol. 2, pp. 194-198). IEEE.

[8] Misra, K. B., & Ljubojevic, M. D. (1973). Optimal reliability design of a system: a new look. *Reliability, IEEE Transactions on*, *22*(5), 255-258.

[9] Marseguerra, M., Zio, E., Podofillini, L., & Coit, D. W. (2005). Optimal design of reliable network systems in presence of uncertainty. *Reliability, IEEE Transactions on*, *54*(2), 243-253.

[10] Elegbede, C., & Adjallah, K. (2003). Availability allocation to repairable systems with genetic algorithms: a multi-objective formulation. *Reliability Engineering & System Safety*, *82*(3), 319-330.

[11] Tian, Z., Zuo, M. J., & Huang, H. (2008). Reliability-redundancy allocation for multi-state series-parallel systems. *Reliability, IEEE Transactions on*, *57*(2), 303-310.

[12] Tian, Z., Levitin, G., & Zuo, M. J. (2009). A joint reliability–redundancy optimization approach for multi-state series–parallel systems. *Reliability Engineering & System Safety*, *94*(10), 1568-1576.

[13] Hamadani, A. Z., & Khorshidi, H. A. (2013). System reliability optimization using time value of money. *The International Journal of Advanced Manufacturing Technology*, 1-10.

[14] Barlow, R. E., & Wu, A. S. (1978). Coherent systems with multi-state components. *Mathematics of Operations Research*, *3*(4), 275-281.

[15] Zio, E., Podofillini, L., & Levitin, G. (2004). Estimation of the importance measures of multi-state elements by Monte Carlo simulation. *Reliability Engineering & System Safety*, *86*(3), 191-204.

[16] Handbook, M. H. E. D. (1998). MIL-HDBK-338B. *US Department of Defense*, *1*.