Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2015, Article ID 276234, 6 pages
http://dx.doi.org/10.1155/2015/276234

Research Article

A Discrete-Binary Transformation of the Reliability Redundancy Allocation Problem

Marco Caserta[1] and Stefan Voß[2,3]

[1]*IE Business School, IE University, Maria de Molina 31B, 28006 Madrid, Spain*
[2]*Institute of Information Systems, University of Hamburg, Von-Melle-Park 5, 20146 Hamburg, Germany*
[3]*Escuela de Ingeniera Industrial, Pontificia Universidad Católica de Valparaíso, Avenida Brasil 2241, 2362807 Valparaíso, Chile*

Correspondence should be addressed to Stefan Voß; stefan.voss@uni-hamburg.de

Received 10 December 2014; Accepted 4 April 2015

Academic Editor: Wei-Chiang Hong

Given a reliability redundancy optimization problem in its *discrete* version, it is possible to transform such integer problem into a corresponding *binary* problem in log-time. A simple discrete-binary transformation is presented in this paper. The proposed transformation is illustrated using an example taken from the reliability literature. An immediate implication is that a standard exact dynamic programming approach may easily solve instances to optimality that were usually only solved heuristically.

1. Introduction

Dealing with system reliability is a central issue in a variety of fields. Hardware and software reliability are of paramount importance, since hardware and software components are pervasive in modern society. The current competitive business environment is placing further emphasis on effective product and system design. In particular, in engineering design with reliability in mind, it is most important to improve the competitive position and to save in engineering design and warranty costs. Academics as well as practitioners have devoted and still put special attention to the advancement of reliability design and analysis methods for complex systems, both hardware and software. Reliability is a strategic issue in a number of industries such as, for example, the aerospace, automotive, civil, defense, telecommunications, and power industries, where advanced systems like space shuttle, aerospace propulsion, nanocomposite structure, and bioengineering systems are designed and developed with reliability in mind. Electronics, mechanics, computer science, and industrial engineering are just other fields interested in current studies on reliability.

Design with reliability in mind provides a number of advantages, spanning from the ability to produce safer and, obviously, more reliable products to the improvement in the competitive position via significant reduction of costs.

One of the recent trends observed in the field of reliability is related to the growth in size and complexity of the systems studied. Due to the fact that hardware and software systems keep growing in size and complexity, there is a need to design and develop efficient methods, that is, algorithms, for reliability problems. More precisely, software solutions that are aimed at reducing the time required to design complex reliability systems, that are able to deal with large scale models, and that are robust with respect to the system configuration are deemed vital.

An immediate idea or technique commonly used to increase reliability of a complex system is via redundancy allocation. The underlying hypothesis is that the reliability value is directly correlated to the number of redundant components placed in each stage of the system. However, owing to a set of constraints describing limitations in terms of cost, physical space availability, and similar limitations, the optimal allocation of redundant components is a hard task.

Usually, algorithms developed for the reliability redundancy allocation problem (RAP) work on the discrete version of the problem (see, e.g., [1–7]). In this paper we present a simple transformation into the binary version of the problem.

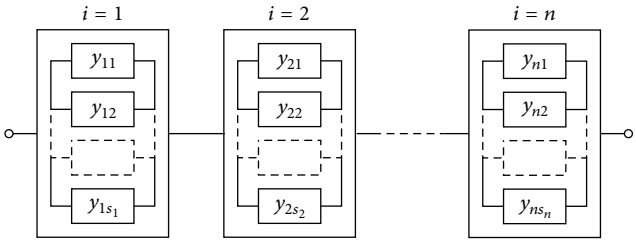

Figure 1: RAP series-parallel system.

Based on that, without loss of generality, one can work with the binary version of the RAP and use this for the future development of algorithms. As a result we can show that problem instances that are still solved heuristically in many recent papers (see, e.g., [8–10]) can be solved to optimality with a straightforward dynamic programming algorithm in reasonable computational time. The proposed discretization is supporting the dynamic programming problem as it allows treating incorporated problems of knapsack type directly as binary knapsack problems. In that sense, it allows binarizing a set of subproblems and then solving the general RAP via dynamic programming by linking or combining a set of binary knapsack problems.

In the next section we first provide a mathematical formulation for the RAP. In the sequel we transform the integer version of the problem into its analogous binary version (log-time transformation) in the spirit of, for example, the authors of [11] who developed similar ideas regarding the knapsack problem. After that, we show that a standard dynamic programming approach is able to quickly solve instances to optimality that were usually out of reach for exact approaches. We close with some conclusions.

2. A Nonlinear Formulation

Let us consider the RAP for the series-parallel configuration system, where n different subsystems are placed in series and, within each system, s_i parallel components are available, as illustrated in Figure 1. The objective of the problem is to determine which components and how many replications of each available component should be selected to maximize the overall system reliability. The problem is complicated by the existence of knapsack-type constraints, typically describing limitations in terms of volume, weight, and cost.

A nonlinear integer formulation for the RAP is

$$\begin{aligned} \text{RAP:} \quad & \\ \max \quad & R = \prod_{i=1}^{n} \left(1 - \prod_{k=1}^{s_i} \left(1 - R_{ik}\right)^{y_{ik}} \right) \\ \text{s.t.} \quad & \sum_{i=1}^{n} \sum_{k=1}^{s_i} g_{ik}^{q} y_{ik} \le b_q, \quad q = 1, \ldots, Q \\ & y_{ik} \in \mathbb{N}, \quad i = 1, \ldots, n, \; k = 1, \ldots, s_i, \end{aligned} \tag{1}$$

where s_i denotes the number of parallel components within subsystem i, $R_{ik} \in [0, 1]$ is the reliability of component k within subsystem i, with $i = 1, \ldots, n$ and $k = 1, \ldots, s_i$, $g_{ik}^{q} > 0$ accounts for the usage of resource q of component k within subsystem i (e.g., volume and cost), and $b_q \in \mathbb{R}_+$ provides the maximum availability of resource q, with $q = 1, \ldots, Q$. Within the set of decision variables, each y_{ik} indicates how many times component k of subsystem i is present in an optimal configuration. It is easy to see that, in order to achieve a nonnull system reliability, the constraint $\sum_k y_{ik} \ge 1$ must be implicitly satisfied for each subsystem i.

3. Transformation between the Discrete RAP and the Binary RAP

Let us indicate, for any $i = 1, \ldots, n$ and $q = 1, \ldots, Q$ with

$$\overline{g}_i^q = \min \left\{ g_{ik}^q : k = 1, \ldots, s_i \right\}, \tag{2}$$

the minimum amount of resource q required to select at least one component within each subsystem i. In addition, let b_q indicate the total amount of resource q available, and

$$\overline{b}_i^q = b_q - \sum_{w \ne i} \overline{g}_w^q, \tag{3}$$

the maximum amount of resource q that could be devoted to subsystem i, while ensuring that all the other subsystems will have enough spare resources to select at least one component. An upper bound on the number of replications of each component k within subsystem i is given by

$$u_{ik} = \left\lfloor \min_q \left\{ \frac{\overline{b}_i^q}{g_{ik}^q} \right\} \right\rfloor. \tag{4}$$

Consequently, possible encoding for the binary RAP is obtained by creating $\lceil \log_2(u_{ik} + 1) \rceil$ binary variables for each component within the system. For the sake of readability, let us focus on a single component k belonging to subsystem i and on a single resource q. Consequently, let us omit indexes i and q and let y_k indicate the kth component of subsystem i. Finally, let us indicate with $\hat{n}$ the total number of binary variables needed to encode the discrete variable y_k; that is,

$$\hat{n} = \left\lceil \log_2 \left(u_k + 1 \right) \right\rceil. \tag{5}$$

The discrete variable y_k, with $0 \le y_k \le u_k$, is substituted by a set of $\hat{n}$ binary variables; that is, $y_k = (x_{k_1}, \ldots, x_{k_h}, \ldots, x_{k_{\hat{n}}})$. Each of these $\hat{n}$ binary "components" corresponds to n_{k_h} replications of component k, where

$$n_{k_h} = \begin{cases} 2^{h-1}, & \text{if } h < \hat{n}; \\ u_{ik} - \sum_{h=1}^{\hat{n}-1} 2^{h-1}, & \text{if } h = \hat{n}, \end{cases} \tag{6}$$

with $h = 1, \ldots, \hat{n}$. Consequently, we have that

$$x_{k_h} = \begin{cases} 1, & \text{if } n_{k_h} \text{ replications of } k \text{ are taken;} \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

Therefore, coefficients n_{k_h} are defined such that they sum up to u_k, that is, the maximum number of replications of component k of subsystem i. Thus,

$$y_k = \sum_{h=1}^{\widehat{n}} n_{k_h} x_{k_h} \tag{8}$$

can take any integer value between 0 and u_k.

Finally, each binary variable x_{k_h} is characterized by the following coefficients:

$$\begin{aligned} a_{k_h} &= n_{k_h} g_k \\ r_{k_h} &= 1 - \left(1 - R_k\right)^{n_{k_h}}, \end{aligned} \tag{9}$$

where R_k and g_k indicate the reliability value and the amount of resource q used by component k within subsystem i, respectively.

Each subsystem i can thus be expressed through the use of a set of binary variables. Let us now reintroduce subindex i. Let us indicate with $\widehat{n}_{ik}$ the number of binary variables used to encode the discrete variable y_{ik}; that is,

$$\widehat{n}_{ik} = \left\lceil \log_2 \left(u_{ik} + 1\right) \right\rceil. \tag{10}$$

Each subsystem i is defined by $m_i = \sum_{k=1}^{s_i} \widehat{n}_{ik}$ binary variables. Let us call these variables, in a progressive fashion, $x_{i1}, \ldots, x_{ij}, \ldots, x_{im_i}$. In addition, let us indicate the progressive position of vector $\mathbf{x}$ where the block of binary variables corresponding to a discrete decision variable y_{ik} begins with

$$p_{ik} = \sum_{w=1}^{k} \widehat{n}_{iw} \tag{11}$$

with $k = 1, \ldots, s_i$. Therefore, the binary vector $\mathbf{x}_i$, that is, the set of binary variables used to describe subsystem i, is defined in the following way:

$$\mathbf{x}_i = \Bigg(\underbrace{x_{i1}, \ldots, x_{ip_{i1}}}_{y_{i1}} \mid \underbrace{x_{ip_{i1}+1}, \ldots, x_{ip_{i2}}}_{y_{i2}} \mid \underbrace{\ldots, \ldots}_{y_{ik}} \mid \underbrace{x_{ip_{is_i-1}+1}, \ldots, x_{ip_{is_i}}}_{y_{is_i}} \Bigg), \tag{12}$$

while vector $\mathbf{x}$, accounting for the whole system, is defined as $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$.

Finally, note that the RAP can be transformed into its corresponding binary version in $\mathcal{O}(m)$, where

$$m = \sum_{i=1}^{n} \sum_{k=1}^{s_i} \left\lceil \log_2 \left(u_{ik} + 1\right) \right\rceil = \sum_{i=1}^{n} m_i. \tag{13}$$

It is easy to see that since $\widehat{n}_{ik}$ is the minimum amount of binary variables required to encode the corresponding discrete variable, any other encoding will introduce the same amount of binary variables [11].

With this encoding, we finally define the RAP in its binary version:

RAP-B:

$$\begin{aligned} \max \quad & R = \prod_{i=1}^{n} \left(1 - \prod_{j=1}^{m_i} \left(1 - r_{ij}\right)^{x_{ij}} \right) \\ \text{s.t.} \quad & \sum_{i=1}^{n} \sum_{j=1}^{m_i} a_{ij}^{q} x_{ij} \le b_q, \quad q = 1, \ldots, Q \\ & x_{ij} \in \mathbb{B}, \quad i = 1, \ldots, n, \ j = 1, \ldots, m_i, \end{aligned} \tag{14}$$

where each x_{ij}, a_{ij}^{q}, and r_{ij} are defined as in (7) and (9).

In line with what is mentioned in [3] for the RAP, model RAP-B enables consideration of (i) multiple component choice for each subsystem—that is, each component can be selected more than once—and (ii) component mixing within a subsystem—that is, more than one component per subsystem can be selected.

An Example. Consider the first subsystem of the benchmark instances of [12]. We present the encoding of variables $y_{11}, \ldots, y_{14}$, accounting for the number of replications of components $1, \ldots, 4$ within subsystem 1. In Table 1 we summarize the relevant data, where R_{1k}, g_{1k}^{1}, and g_{ik}^{2} represent the reliability, cost, and weight of component k within subsystem 1, with $k = 1, \ldots, 4$.

Let us first consider component $k = 1$. From [12], the maximum number of replications for each component is equal to 4; that is, $u_{ik} = 4$. Therefore, we have $\widehat{n} = \lceil \log_2(4 + 1) \rceil = 3$. Consequently, we encode the *discrete* variable y_{11} through the use of three binary variables; that is, $y_{11} = (x_{11} \ x_{12} \ x_{13})$. Using (6), we compute the coefficients corresponding to the number of replications associated with each binary variable; that is, $n_{11} = 1$, $n_{12} = 2$, and $n_{13} = 1$. Therefore, as in (7), each binary variable has the following meaning:

$$\begin{aligned} x_{11} &= \begin{cases} 1, & \text{if 1 replication of component 1 is taken;} \\ 0, & \text{otherwise,} \end{cases} \\ x_{12} &= \begin{cases} 1, & \text{if 2 replications of component 1 are taken;} \\ 0, & \text{otherwise,} \end{cases} \\ x_{13} &= \begin{cases} 1, & \text{if 1 replication of component 1 is taken;} \\ 0, & \text{otherwise.} \end{cases} \end{aligned} \tag{15}$$

It is easy to see that every possible value of y_{11} can be expressed through an appropriate choice of values of x_{11}, x_{12},

Table 1: Input data of subsystem 1 of the instance from [12].

$i = 1$	$k = 1, \ldots, 4$			
	y_{11}	y_{12}	y_{13}	y_{14}
R_{1k}	0.90	0.93	0.91	0.95
g^1_{1k}	1	1	2	2
g^2_{1k}	3	4	2	5

and x_{13}. The coefficients of the binary variables are computed using (9):

$$\begin{aligned} r_{11} &= 1 - \left(1 - R_{11}\right)^{n_{11}} = 0.9 \\ r_{12} &= 1 - \left(1 - R_{11}\right)^{n_{12}} = 0.99 \\ r_{13} &= 1 - \left(1 - R_{11}\right)^{n_{13}} = 0.9 \\ a^1_{11} &= n_{11} g^1_{11} = 1 \times 1 = 1 \\ a^1_{12} &= n_{12} g^1_{11} = 2 \times 1 = 2 \\ a^1_{13} &= n_{13} g^1_{11} = 1 \times 1 = 1 \\ a^2_{11} &= n_{11} g^2_{11} = 1 \times 3 = 3 \\ a^2_{12} &= n_{12} g^2_{11} = 2 \times 3 = 6 \\ a^2_{13} &= n_{13} g^2_{11} = 1 \times 3 = 3. \end{aligned} \tag{16}$$

In a similar way, we encode variables y_{12}, y_{13}, and y_{14}. Consequently, the whole set of discrete variables referring to subsystem 1 is encoded using 12 binary variables as follows:

$$\mathbf{x}_1 = \Bigg(\underbrace{x_{11}, \ldots, x_{13}}_{y_{11}} \mid \underbrace{x_{14}, \ldots, x_{16}}_{y_{12}} \mid \underbrace{x_{17}, \ldots, x_{19}}_{y_{13}} \mid \underbrace{x_{1,10}, \ldots, x_{1,12}}_{y_{14}} \Bigg). \tag{17}$$

4. Utilizing the Transformation

Dynamic programming (DP) has been used for the RAP by, for example, the authors in [12–14] who present DP schemes for the single-constraint version of the RAP. The inherent difficulty of using DP in the multiconstraint case is related to the growth of the dimensions of the table for the DP return function. More precisely, both space complexity and computational complexity of the DP scheme grow with $\mathcal{O}(\prod_{q=1}^{Q} b_q)$, where Q indicates the number of knapsack-type constraints. Consequently, even with only a few constraints, the use of a DP approach can be cumbersome, though using the above transformation enables its use in settings that seemed out of reach without it.

In the next subsection we describe a DP algorithm for the RAP followed by a subsection presenting and discussing numerical results.

4.1. A Dynamic Programming Approach. We present a straightforward DP recursion for the RAP (note that this recursion can be used in various settings, e.g., as subroutine for metaheuristics; see [15]). We first separate each subsystem and transform the original problem into a series of "independent" knapsack problems (similar to [16]). The objective here is to maximize the reliability of subsystem i, indicated with R_i. The objective function of each subsystem can easily be linearized and the reliability of single subsystem i can be rewritten as

$$R'_i = \sum_{j=1}^{m_i} r'_{ij} x_{ij}, \tag{18}$$

where $R'_i = -\ln(1 - R_i)$ and $r'_{ij} = -\ln(1 - r_{ij})$. The problem of finding the optimal allocation of redundant components for subsystem i, with $i = 1, \ldots, n$, can be written as follows:

$$\begin{aligned} \text{RAP}\,(i): \quad & \\ \max \quad & \sum_{j=1}^{m_i} r'_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_{j=1}^{m_i} a^q_{ij} x_{ij} \le b_q \quad q = 1, \ldots, Q \\ & x_{ij} \in \{0, 1\} \quad j = 1, \ldots, m_i. \end{aligned} \tag{19}$$

Problem RAP(i) is equivalent to a standard multidimensional knapsack problem with Q constraints, which can be solved in pseudopolynomial time using DP ($\mathcal{O}(m_i \bar{b})$ where $\bar{b} = \prod_q b_q$). Let us indicate with $f_i(k, \mathbf{d})$ the optimal objective function value of the RAP(i) when only the first k components are considered, with $k = 1, \ldots, m_i$, and $\mathbf{d}^T = (d_1, \ldots, d_Q)$ such that $d_q = 0, \ldots, b_q$. A forward recursion formula for this problem is

$$f_i(k, \mathbf{d}) = \begin{cases} \begin{cases} 0 & \text{if } \mathbf{d} < \mathbf{a}_{ik} \\ r'_{ik} & \text{otherwise} \end{cases} & \text{for } k = 1, \\ \max\left\{ r'_{ik} + f_i\left(k - 1, \mathbf{d} - \mathbf{a}_{ik}\right), \right. & \\ \qquad \left. f_i\left(k - 1, \mathbf{d}\right) \right\} & \text{for } k = 2, \ldots, m_i, \end{cases} \tag{20}$$

where $\mathbf{a}^T_{ik} = (a^1_{ik}, \ldots, a^Q_{ik})$ indicates the resource consumption of component k, with $k = 1, \ldots, m_i$. Thus, $f_i(\mathbf{d}) = f_i(m_i, \mathbf{d})$ indicates the optimal solution with respect to subsystem i when $\mathbf{d}$ units of resources are used. In order to solve the RAP, let us now define $R(s, \mathbf{d})$ as the optimal objective function value of the RAP when the first s subsystems are considered, with $s = 1, \ldots, n$, and $\mathbf{d}^T = (d_1, \ldots, d_Q)$ such that $d_q = 0, \ldots, b_q$. A DP recursion for the "classical" series-parallel RAP (i.e., $k_i = 1$) is the following:

$$R(s, \mathbf{d}) = \begin{cases} f_1(\mathbf{d}) & \text{for } s = 1, \\ \max\limits_{\mathbf{w}=\mathbf{0}, \ldots, \mathbf{d}} \left\{ f_s(\mathbf{w}) \times R(s - 1, \mathbf{d} - \mathbf{w}) \right\} & \text{for } s = 2, \ldots, n. \end{cases} \tag{21}$$

Obviously, $R(n, \mathbf{b})$ is the optimal objective function value of the original RAP. The overall computational complexity of the recursive algorithm is $\mathcal{O}(n\bar{b}^2)$.

4.2. Numerical Results. The DP algorithm described in the previous subsection has been coded in C++ and compiled with the GNU C++ compiler using the -O option. We present results for a benchmark set made up of 33 instances originally proposed by [12] but also used by a number of researchers. Best known results have been reported by [7, 17] and, consequently, we will compare the results of the proposed algorithm with those reported by these authors. The 33 variations of the RAP are series-parallel systems with two knapsack-type constraints, representing weight and cost, and 14 different links. That is, the system characteristics are $n = 14$, $Q = 2$, and m_i either 3 or 4, depending on the value of i; see [7, 12] for details. Table 2 presents numerical results for these 33 benchmark instances. The wall-clock time is measured in seconds on a dual core Pentium 1.8 GHz Linux workstation with 4 Gb of RAM.

In Table 2, the first column provides the instances number and the second and third columns give the total weight W and cost C allowed. We solved all the instances to optimality using the DP scheme described above with the objective values and computational times as indicated in Table 2. Note that [7] also found these solutions, however, without being able to prove optimality. That is, the benefit of our transformation lies in applying a standard algorithm under this transformation to prove optimality of the instances within a short amount of time.

To make things more clear, we should emphasize that there are elaborate and versatile exact algorithms available that solve these instances to optimality as well as various heuristics and metaheuristics. Most interesting seem two avenues of research.

The first is the fact that we have successfully applied the transformation described in this paper in a subsequent paper [18], which not only is able to solve all benchmark instances from [12] to optimality, but for the first time also provides optimal solutions to the extended benchmark instances given in [3] that have not been solved to optimality before.

The second is the observation that the benchmark instances from [12] are still an object of intensive investigation for heuristics and metaheuristics. For instance, in a recent paper, [8], the authors provide a summary of various metaheuristics (including variable neighborhood search, tabu search, ant colony optimization, a genetic algorithm, and their own differential evolution algorithm; see Tables 2 and 3 of [8]) and none was able to solve all 33 benchmark instances to optimality (and they did not try those from [3]). While the computational times of our DP algorithm are higher than those of most of these metaheuristics, it is a standard DP approach which is easy to implement. That is, based on our results, there should be no good reason to study heuristics and metaheuristics for these instances just by themselves, unless the algorithms are applied to more general problems or at least more difficult instances than the ones from [12].

Table 2: Results on 33 benchmark instances.

Number	W	C	Objective	Time DP
1	191	130	0.986811	17.664
2	190	130	0.986416	17.556
3	189	130	0.985922	17.712
4	188	130	0.985378	17.22
5	187	130	0.984688	16.98
6	186	129	0.984176	16.728
7	185	130	0.983505	16.74
8	184	130	0.982994	16.692
9	183	129	0.982256	16.452
10	182	130	0.981518	16.008
11	181	129	0.981027	16.068
12	180	128	0.98029	16.14
13	179	126	0.979505	15.864
14	178	125	0.9784	15.336
15	177	126	0.977596	15.06
16	176	124	0.97669	14.916
17	175	125	0.975708	15.264
18	174	123	0.974926	14.628
19	173	122	0.973827	14.664
20	172	123	0.973027	14.184
21	171	122	0.971929	14.136
22	170	120	0.97076	13.944
23	169	121	0.969291	13.86
24	168	119	0.968125	13.56
25	167	118	0.966335	13.38
26	166	116	0.965042	13.284
27	165	117	0.963712	13.188
28	164	115	0.962422	13.056
29	163	114	0.960642	12.672
30	162	115	0.959188	12.732
31	161	113	0.958035	12.312
32	160	112	0.955714	12.372
33	159	110	0.954565	11.808

5. Conclusion

In this paper we have proposed a reformulation of the reliability redundancy allocation problem based on a binary discretization. That is, we proposed a simple discrete-binary transformation for reliability redundancy allocation. The discretization may be characterized as known in other fields, though it was not applied to this particular class of problems before. While our aim was not to investigate any new type of algorithm, we strongly believe that future research on algorithms for the redundancy allocation problem may favorably utilize this transformation as we have shown by means of a standard dynamic programming approach. Note in passing that we have applied this transformation in different context to obtain numerical results that were out of reach to the RAP community for quite some time. Moreover, for future research it should be of interest to apply our simple

transformation on a wider scale for other types of reliability problems wherever deemed practical.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] A. Azaron, H. Katagiri, K. Kato, and M. Sakawa, "A multi-objective discrete reliability optimization problem for dissimilar-unit standby systems," *OR Spectrum*, vol. 29, no. 2, pp. 235–257, 2007.

[2] A. Billionnet, "Redundancy allocation for series-parallel systems using integer linear programming," *IEEE Transactions on Reliability*, vol. 57, no. 3, pp. 507–516, 2008.

[3] D. W. Coit and A. Konak, "Multiple weighted objectives heuristic for the redundancy allocation problem," *IEEE Transactions on Reliability*, vol. 55, no. 3, pp. 551–558, 2006.

[4] S. Kulturel-Konak, A. E. Smith, and D. W. Coit, "Efficiently solving the redundancy allocation problem using tabu search," *IIE Transactions*, vol. 35, no. 6, pp. 515–526, 2003.

[5] J. E. Ramirez-Marquez, D. W. Coit, and A. Konak, "Redundancy allocation for series-parallel systems using a max-min approach," *IIE Transactions*, vol. 36, no. 9, pp. 891–898, 2004.

[6] B. Soylu and S. K. Ulusoy, "A preference ordered classification for a multi-objective maxmin redundancy allocation problem," *Computers & Operations Research*, vol. 38, no. 12, pp. 1855–1866, 2011.

[7] P.-S. You and T.-C. Chen, "An efficient heuristic for series-parallel redundant reliability problems," *Computers & Operations Research*, vol. 32, no. 8, pp. 2117–2127, 2005.

[8] N. Beji, B. Jarboui, P. Siarry, and H. Chabchoub, "A differential evolution algorithm to solve redundancy allocation problems," *International Transactions in Operational Research*, vol. 19, no. 6, pp. 809–824, 2012.

[9] G. Kanagaraj, S. G. Ponnambalam, and N. Jawahar, "A hybrid cuckoo search and genetic algorithm for reliability-redundancy allocation problems," *Computers & Industrial Engineering*, vol. 66, no. 4, pp. 1115–1124, 2013.

[10] E. Valian and E. Valian, "A cuckoo search algorithm by Lévy flights for solving reliability redundancy allocation problems," *Engineering Optimization*, vol. 45, no. 11, pp. 1273–1286, 2013.

[11] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*, John Wiley & Sons, 1990.

[12] Y. Nakagawa and S. Miyazaki, "Surrogate constraints algorithm for reliability optimization problems with two constraints," *IEEE Transactions on Reliability*, vol. 30, no. 2, pp. 175–180, 1981.

[13] K. Y. K. Ng and N. G. F. Sancho, "A hybrid 'dynamic programming/depth-first search' algorithm, with an application to redundancy allocation," *IIE Transactions*, vol. 33, no. 12, pp. 1047–1058, 2001.

[14] A. Yalaoui, E. Châtelet, and C. Chu, "A new dynamic programming method for reliability & redundancy allocation in a parallel-series system," *IEEE Transactions on Reliability*, vol. 54, no. 2, pp. 254–261, 2005.

[15] M. Caserta and S. Voß, "A corridor method based hybrid algorithm for redundancy allocation," *Journal of Heuristics*, 2014.

[16] R. L. Bulfin and C. Y. Liu, "Optimal allocation of redundant components for large systems," *IEEE Transactions on Reliability*, vol. 34, no. 3, pp. 241–247, 1985.

[17] J. Onishi, S. Kimura, R. J. W. James, and Y. Nakagawa, "Solving the redundancy allocation problem with a mix of components using the improved surrogate constraint method," *IEEE Transactions on Reliability*, vol. 56, no. 1, pp. 94–101, 2007.

[18] M. Caserta and S. Voß, "An exact algorithm for the reliability redundancy allocation problem," *European Journal of Operational Research*, vol. 244, no. 1, pp. 110–116, 2015.