CIRRELT

Centre interuniversitaire de recherche
sur les réseaux d'entreprise, la logistique et le transport

Interuniversity Research Centre
on Enterprise Networks, Logistics and Transportation

An Efficient Heuristic for Reliability Design Optimization Problems

Mohamed Ouzineb
Mustapha Nourelfath
Michel Gendreau

February 2009

CIRRELT-2009-05

Bureaux de Montréal :
Université de Montréal
C.P. 6128, succ. Centre-ville
Montréal (Québec)
Canada H3C 3J7
Téléphone : 514 343-7575
Télécopie : 514 343-7121

Bureaux de Québec :
Université Laval
Pavillon Palasis-Prince, local 2642
Québec (Québec)
Canada G1K 7P4
Téléphone : 418 656-2073
Télécopie : 418 656-2624

www.cirrelt.ca

An Efficient Heuristic for Reliability Design Optimization Problems

Mohamed Ouzineb[1,2], Mustapha Nourelfath[1,3,*], Michel Gendreau[1,2]

1. Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT)
2. Department of Computer Science and Operations Research, Université de Montréal, P.O. Box 6128, Station Centre-ville, Montréal, Canada H3C 3J7
3. Département de génie mécanique, Pavillon Adrien-Pouliot, Université Laval, Québec, Canada G1K 7P4

Abstract. This paper develops an efficient heuristic to solve two typical combinatorial optimization problems frequently met when designing highly reliable systems. The first one is the redundancy allocation problem (RAP) of series-parallel binary-state systems. The design goal of the RAP is to select the optimal combination of elements and redundancy levels to maximize system reliability subject to the system budget and to the system weight. The second problem is the expansion-scheduling problem (ESP) of multi-state series-parallel systems. In this problem, the study period is divided into several stages. At each stage, the demand is represented as a piecewise cumulative load curve. During the system lifetime, the demand can increase and the total productivity may become insufficient to assume the demand. To increase the total system productivity, elements are added to the existing system. The objective in the ESP is to minimize the sum of costs of the investments over the study period while satisfying availability constraints at each stage. The heuristic approach developed to solve the RAP and the ESP is based on a combination of space partitioning, genetic algorithms (GA) and tabu search (TS). After dividing the search space into a set of disjoint subsets, this approach uses GA to select the subspaces, and applies TS to each selected subspace. Numerical results for the test problems from previous research are reported and compared. The results show the advantages of the proposed approach for solving both problems.

Keywords. Tabu search, genetic algorithms, space partitioning, redundancy optimization, expansion planning.

Acknowledgements. The authors would like to thank anonymous reviewers for their comments and helpful questions. They would also like to thank the Natural Sciences and Engineering Research Council of Canada (NSERC) for financial support.

---

* Corresponding author: Mustapha.Nourelfath@cirrelt.ca

This document is also published as Publication #1331 by the Department of Computer Science and Operations Research of the Université de Montréal.

Dépôt légal – Bibliothèque et Archives nationales du Québec,
Bibliothèque et Archives Canada, 2009

1 Introduction

During the last decades, reliability optimization has attracted a great number of researchers, due to the critical importance of designing highly reliable systems in various industrial contexts. One option to improve system reliability or availability is to include redundant elements in parallel. Nevertheless, resources are usually required for any enhancement leading to constrained optimization problems. In general these problems are nonlinear integer programming problems, of combinatorial nature and NP-hard. For example, the redundancy allocation problem (RAP) was shown to be NP-hard in [3]. In recent works a major focus is on the development of heuristic methods that are based on meta-heuristics to solve reliability optimization problems [34].

The present contribution develops an efficient approach based on meta-heuristics to solve two problems of system reliability optimization, namely the redundancy allocation problem (RAP) of series-parallel binary-state systems (Problem 1), and the expansion-scheduling problem (ESP) of series-parallel multi-state systems (Problem 2). Problems 1 and 2 are chosen as typical representatives from binary-state reliability and multi-state reliability problems, respectively. The proposed heuristic approach combines genetic algorithms (GA), tabu search (TS), and the idea of space partitioning (SP). Because of such a combination, it is said to be hybrid. We call it space partitioning/tabu-genetic (SP/TG), SP and TG being acronyms of Space Partitioning and Tabu-Genetic, respectively.

The remainder of the paper is organized as follows. In Section 2, we present briefly a description, a literature review, and a mathematical formulation for problems 1 and 2. Section 3 develops our SP/TG approach. In Section 4, we apply this approach to solve problems 1 and 2. The test problems and the numerical results are presented in Section 5. Finally, some concluding remarks are given in Section 6.

2 Two typical reliability design optimization problems

2.1 The redundancy allocation in series-parallel binary-state systems

2.1.1 *Problem description and literature review*

Using binary-state reliability modeling, Problem 1 assumes that a system and its elements may experience only two possible states: good and failed. We also assume that failed elements are not repaired. Other classical assumptions include that all redundancy is active, and that the failures of individual elements are mutually s-independent. The system consists of components in series. Each component contains a number of elements connected in parallel. Different elements can be placed in parallel (*i.e.*, element mixing is allowed). A component of index i is functioning correctly if at least k_i of its elements are operational (k-out-of n:G component redundancy). This series-parallel system is a logic diagram representation. For each component, there are various element versions, which are proposed by the suppliers in the market. Each element is characterized by its cost, its weight and its reliability according to its version. Redundancy allows reliability improvement, but increases the total cost. The design goal is to select the optimal combination of elements and redundancy levels so that the total reliability is maximized, subject to budget and to weight constraints. This is a complex combinatorial optimization problem, which is very important in many industrial applications. It has been studied in many different forms, and by considering numerous approaches and techniques.

Assuming element mixing is not allowed, the RAP of binary-state series-parallel systems has been solved using exact methods (in [1, 6] using dynamic programming approach, in [2, 7, 10, 22] using integer programming and in [25] using mixed-integer and nonlinear programming). In [21], Nakagawa and Miyazaki have also proposed an exact algorithm named N&M to solve 33 variations of the Fyffe problem [6]; they have shown that their algorithm found optimal solutions for 30 of them. But, the algorithm did not converge to a feasible solution for the 3 other cases.

When element mixing is permitted, the problem becomes very complex due to the enormous size of the search space. To solve the problem, Coit and Smith [4] used a genetic algorithm while allowing to search in infeasible regions. A linear approximation for RAP with multiple element choices has been suggested by Hsieh [13]. An efficient tabu search has been developed in [14] to solve the RAP, while the ant colony optimization meta-heuristic was used in [17]. To the best of our knowledge, the best-published results have been provided by ant colony meta-heuristic with the degraded ceiling in [20] and by variable neighbourhood search in [18].

2.1.2 *Mathematical formulation*

Notation for Problem 1

n	number of components in the system
i	index for component, $i \in \{1, 2, ..., n\}$
m_i	number of available elements choices for component i
k_i	minimum number of elements in parallel required for component i
$\mathbf{k}$	$(k_1, k_2, ..., k_n)$
p_i	total number of elements used in component i
n_{max}	maximum number of elements in parallel
j	index (type) for element
x_{ij}	number of element of type j used in component i
$\mathbf{x}_i$	$(x_{i1}, x_{i2}, ..., x_{im_i})$
$\mathbf{X}$	a string of dimension $L = \sum_{i=1}^{n} m_i$ which defines the entire system structure, $\mathbf{X} = (x_{11}, x_{12}, ..., x_{1m_1}, x_{21}, x_{22}, ..., x_{2m_2}, ..., x_{n1}, x_{n2}, ..., x_{nm_n})$
M_{ij}	maximum number of elements of version j in parallel belonging to component i
r_{ij}	reliability of element of version j belonging to component i
c_{ij}	cost of each element of version j in component i
w_{ij}	weight of each element of version j in component i
$R(\mathbf{x}_i\|k_i)$	total reliability of system, given k_i
$R_i(\mathbf{x}_i\|k_i)$	reliability of component i, given k_i
$C_i(\mathbf{x}_i)$	total cost of component i
$W_i(\mathbf{x}_i)$	total weight of component i
C_0	cost limit
W_0	weight limit
$mnli$	maximum number of local iterations without improvement

The RAP formulation is given by:

$$\text{maximize} \quad R(\mathbf{x}_i|k_i) = \prod_{i=1}^{n} R_i(\mathbf{x}_i|k_i) \tag{1}$$

subject to

$$\sum_{i=1}^{n} C_i(\mathbf{x}_i) \leq C_0, \tag{2}$$

$$\sum_{i=1}^{n} W_i(\mathbf{x}_i) \leq W_0, \tag{3}$$

$$\sum_{j=1}^{m_i} x_{ij} \leq n_{max}, \quad i = 1, 2, ..., n, \tag{4}$$

$$\sum_{j=1}^{m_i} x_{ij} \geq k_i, \qquad i = 1, 2, ..., n, \tag{5}$$

$$x_{ij} \in \{0, 1, ..., M_{ij}\}, \quad i = 1, 2, ..., n; \quad j = 1, 2, ..., m_i. \tag{6}$$

The objective function (1) represents overall reliability of the series-parallel system. The system reliability is calculated by the product of n component reliabilities represented by $R_i(\mathbf{x}_i|k_i)$. The value $R_i(\mathbf{x}_i|k_i)$ is the reliability of component i given the minimum number of elements in parallel required for component i to function, *i.e.*, x_i.

Constraints (2) and (3) represent, respectively, the budget and the weight constraints. Constraint (4) specifies that, the number of elements to be included into component i cannot be higher than a pre-selected maximum number n_{max}. Constraint (5) represents the k-out-of n:G constraint. Constraint (6) specifies that, for each component i, number of elements of each type j is identified by integers from 0 to a maximum number of elements available in the market.

2.2 The expansion-scheduling of series-parallel multi-state systems

2.2.1 *Problem description and literature review*

The expansion-scheduling problem (ESP) of a series-parallel multi-state system was introduced by Levitin in [27]. The system is called a multi-state system (MSS) because it is considered to have a range of performance levels from perfect functioning to complete failure. In the formulation of the ESP of a series-parallel MSS, the system study horizon is divided into several periods. At each period the demand distribution is predicted in the form of a cumulative demand curve. As in many industrial cases the demand increases with time, the design problem concerns expansions (or reinforcements) to adjust the system productivity (or capacity) to the demand. This is ensured by adding elements which are chosen from the list of products available in the market. It is assumed that any failed element is repaired. These elements are binary-state and are characterized by their cost, productivity and own availability. The MSS availability is defined as the ability to satisfy consumer demand. The initial structure of the system may be given at the initial stage or it may be empty. The objective is to minimize the sum of investment-cost over the study period while satisfying availability constraints at each stage. The developed algorithm has to answer the following questions:

1. What elements must be added to the system?
2. Into which system-components the additional elements should be included?
3. At which period stage each element should be added?

The ESP is a difficult large-scale optimization problem due to the enormous size of the search space. It can be seen as a more complicated extension of the RAP. In fact, the single stage series-parallel MSS expansion-scheduling problem corresponds to the RAP of MSS. A good and extensive review of MSS literature can be found for example in [28] and [29]. The RAP for series-parallel MSS was first introduced in [30], where the universal moment generating function (UMGF) method was used for the reliability calculation [31]. Following these works, genetic algorithms were used in [15] and [16] for finding the minimal cost configuration of a series-parallel MSS under reliability or availability constraints. The others existing solution techniques of the RAP of MSS are ant colony optimization [35], a heuristic algorithm in [33] and tabu search [24]. The expansion-scheduling (or expansion planning) problem for multi-state series-parallel systems is more recent and has not been sufficiently studied. The only existing solution techniques of this problem are genetic algorithms [27] and an ant colony optimization algorithm in [35].

2.2.2 *Mathematical formulation*

Notation for Problem 2

Γ	number of expansion stages
γ	stage number, $1 \leq \gamma \leq \Gamma$
$J_{\gamma i}$	a number which identifies version (version number or index) of element to be included into component i at stage γ, $J_{\gamma i} \in \{1, 2, ..., Max(J_{\gamma i})\}$
$Max(J_{\gamma i})$	maximum number of versions available in the market for element to be included into component i at stage γ
$\mathbf{J}$	a matrix $(J_{\gamma i})$, $1 \leq \gamma \leq \Gamma$, $1 \leq i \leq n$, which specifies version numbers to be included to each component at each stage in the system
$H_{\gamma i}$	number of elements to be included into component i at stage γ
$Max(H_{\gamma i})$	maximum $H_{\gamma i}$ allowed (i.e., the upper bound of $H_{\gamma i}$)
$\mathbf{H}$	a matrix $(H_{\gamma i})$, $1 \leq \gamma \leq \Gamma$, $1 \leq i \leq n$, which specifies the numbers of elements to be included into each component in the system at each stage γ, $H_{\gamma i} \in \{1, 2, ..., Max(H_{\gamma i})\}$
$\mathbf{Y}$	a matrix $(Y_{\gamma l})$, $1 \leq \gamma \leq \Gamma$, $1 \leq l \leq 2n$ of dimension $2n\gamma$, which defines the entire expansion planning. For $1 \leq l \leq n$, the components of the matrix $\mathbf{Y}$ are those of $\mathbf{H}$. For $n+1 \leq l \leq 2n$, the components of the matrix $\mathbf{Y}$ are those of $\mathbf{J}$. This matrix is denoted by $\mathbf{Y} = (\mathbf{H}, \mathbf{J})$
$C_\gamma(\mathbf{Y})$	system expansion cost at stage γ, $C_\gamma(\mathbf{Y}) = C_\gamma(\mathbf{H}, \mathbf{J})$
$C(\mathbf{Y})$	total system expansion cost, $C(\mathbf{Y}) = \sum_{\gamma=1}^{\Gamma} C_\gamma(\mathbf{Y})$
$A_\gamma(\mathbf{Y})$	stationary availability index of the overall multi-state series-parallel system at stage γ
A_0	a specified minimum required level of system availability index
ρ	interest rate

$t(\gamma)$	time (in years) from initial stage to stage γ
j	index for element version ($j = J_{\gamma i}$)
A_{ij}	binary-state availability of element of version j belonging to component i
C_{ij}	cost of each element of version j in component i
W_{ij}	nominal performance level of element of version j in component i
A^0_{ij}	binary-state availability of element of version j belonging to i at stage 0
W^0_{ij}	nominal performance level of element of version j in component i at stage 0
K	number of partitioned intervals at each stage
k	index for partitioned intervals
$T(\gamma)$	MSS operation period at stage γ
$T_k(\gamma)$	a partitioned interval in $T(\gamma)$, $T(\gamma) = \sum_{k=1}^{K} T_k(\gamma)$
$W_k(\gamma)$	required MSS performance level for $T_k(\gamma)$
m	number of MSS state, $m \in \{1, 2, ..., M\}$, 1 is the worst state and M is the best state
W_m	MSS steady-state performance level associated with m
$W(t)$	output performance level of the MSS at time t, $W(t) \in \{W_1, ..., W_M\}$
p_m	$\lim_{t\to\infty}[Pr(W(t) = W_m)]$
Str_0	initial system structure
q	amplification parameter in the penalized objective function

The global period to be examined is divided into several stages. Each stage γ begins $t(\gamma)$ years after the stage 0. The total operation period at each stage is divided into K intervals of durations $(T_1(\gamma), T_2(\gamma), ..., T_K(\gamma))$, $(\gamma = 1, 2, ..., \Gamma)$ ($T_k(\gamma) = 0$ for redundant intervals), and each interval has a required demand level $(W_1(\gamma), W_2(\gamma), ..., W_K(\gamma))$, $(\gamma = 1, 2, ..., \Gamma)$. We consider an initial structure of the series-parallel system which consists of n components which are independent and connected in series. Each component i $(i = 1, 2, ..., n)$ is composed of actively redundant elements connected in parallel. Each element is characterized by its availability, its unit cost and its performance. Redundancy allows availability improvement, but increases the total cost. The objective is to minimize the sum of the costs of the investments over the study period, subject to availability constraints at each stage. At each stage, it is considered that once an element selection is made, only the same element type can be used to provide redundancy. That is, at each stage, for each component, one has to select one element type to be included and to determine the number of redundant elements. Figures 1–3 illustrate the expansion process in the ESP by considering an example. In reference [27], the ESP was studied for a power-station coal transportation system which supplies boilers, and which has 5 basic system-components.

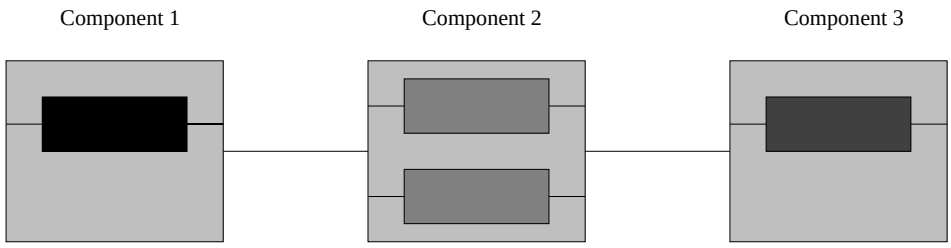

Figure 1: Initial structure of MSS

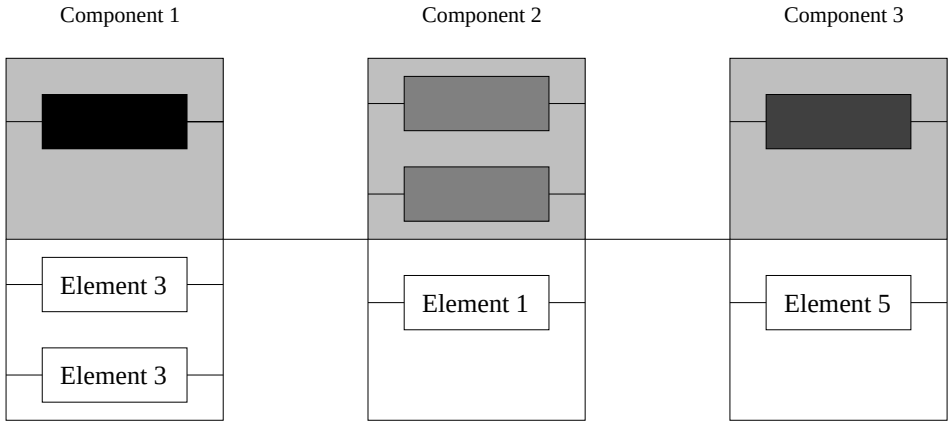

Figure 2: MSS structure for the first period

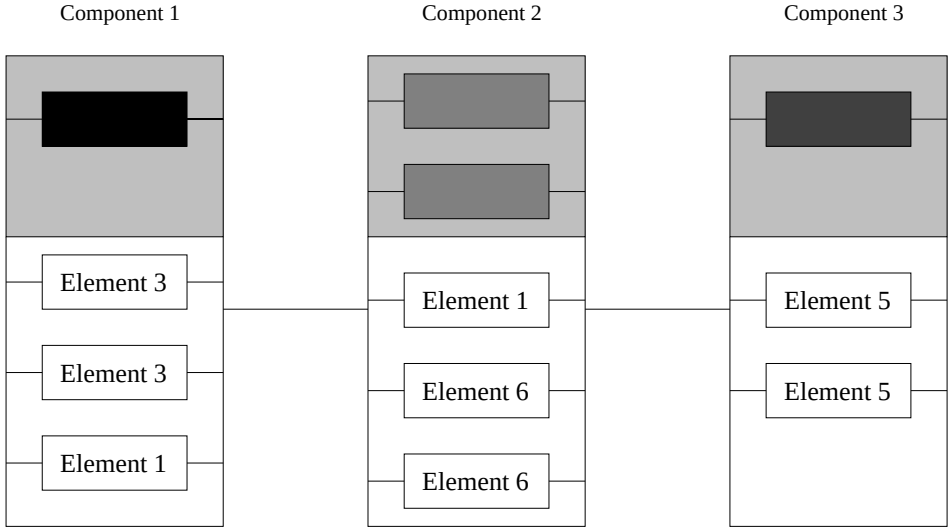

Figure 3: MSS structure for the second period

The total cost is calculated by adding the costs of the system expansion at each stage. For each system structure defined as a matrix $\mathbf{Y} = (Y_{\gamma l})$, $1 \leq \gamma \leq \Gamma$, $1 \leq l \leq 2n$, such as $\mathbf{Y}$=$(\mathbf{H}, \mathbf{J})$, the cost of the system expansion at stage γ is given by [27]:

$$C_\gamma(\mathbf{Y}) = \frac{1}{(1+\rho)^{t(\gamma)}} \sum_{i=1}^{n} H_{\gamma i} C_{iJ_{\gamma i}}. \tag{7}$$

Thus, the total cost is given by:

$$C(\mathbf{Y}) = \sum_{\gamma=1}^{\Gamma} \frac{1}{(1+\rho)^{t(\gamma)}} \sum_{i=1}^{n} H_{\gamma i} C_{iJ_{\gamma i}}. \tag{8}$$

The availability of total system at stage γ is given by [27]:

$$A_\gamma(\mathbf{Y}) = \frac{\sum_{k=1}^{K} P(W_S(\gamma) \geq W_k(\gamma)) T_k(\gamma)}{\sum_{k=1}^{K} T_k(\gamma)}, \tag{9}$$

where, $P(W_S(\gamma) \geq W_k(\gamma))$ is the probability that the total system capacity $W_S(\gamma)$ at stage γ is not less than the demand level $W_k(\gamma)$.

The problem formulation is given by:

$$\text{minimize} \quad C(\mathbf{Y}) = C(\mathbf{H}, \mathbf{J}) = \sum_{\gamma=1}^{\Gamma} \frac{1}{(1+\rho)^{t(\gamma)}} \sum_{i=1}^{n} H_{\gamma i} C_{iJ_{\gamma i}} \tag{10}$$

subject to

$$A_\gamma(\mathbf{H}, \mathbf{J}) \geq A_0, \quad \gamma = 1, 2, ..., \Gamma, \tag{11}$$

$$H_{\gamma i} \in \{0, 1, ..., Max(H_{\gamma i})\}, \quad i = 1, 2, ..., n; \quad \gamma = 1, 2, ..., \Gamma, \tag{12}$$

$$J_{\gamma i} \in \{1, ..., Max(J_{\gamma i})\}, \quad i = 1, 2, ..., n; \quad \gamma = 1, 2, ..., \Gamma. \tag{13}$$

The objective (10) is to minimize the total cost. Eq. (11) represents the availability constraint. Constraint (12) specifies that, at each stage γ, the number of elements to be included into component i is an integer which cannot be higher than a pre-selected maximum number $Max(H_{\gamma i})$. Constraint (13) specifies that, at each stage γ, for each component i, versions are identified by integers from 1 to a maximum number of versions available in the market. Given the solution structure ($\mathbf{Y} = (\mathbf{H}, \mathbf{J})$), the identical elements constraint is verified automatically. To estimate the availability function $A_\gamma(\mathbf{H}, \mathbf{J})$ we will use the U-function (universal z-transform) technique [31]. This mathematical technique is also called universal moment generating function (UMGF). It was proven to be very effective for high dimension combinatorial problems in [28] and [29]. For more details about MSS availability evaluation by using the UMGF method, the reader is referred to [31] or [29]. This method is used in [27] to calculate the availability function $A_\gamma(\mathbf{H}, \mathbf{J})$.

3 The space partitioning and tabu-genetic approach

The main idea of our approach consists in dividing the search space into a set of disjoint subsets, selecting subspaces by using genetic algorithms, and applying tabu search to each selected subspace. Each of these will now be described in more detail.

Dividing the search space into a set of disjoint subspaces

A given solution can be defined either as a vector or a matrix. It can be also defined as a string. We define an address which characterizes this solution. This address is usually an integer identifier which may depend on the system structure and parameters. It has to be chosen such that a set of solutions have the same address. In fact, we define each search subspace as the set of solutions which have the same address. The collection of non-empty search subspaces forms a *partition* of the search space.

Selecting subspaces by using genetic algorithms

When the number of subspaces is huge, the objective of this step is to locate promising search subspaces. The basic idea of the genetic algorithms is to evolve a population of individuals. Each individual in the population is represented as a string of finite integer numbers and represents only one region in the search space (*i.e.*, solutions which have the same address). We define the fitness of a given individual. At each iteration, two individuals are randomly selected and produces a new

solution (offspring) using a 2-points crossover procedure. The new solution is compared with the worst solution in the population. The better solution joints the population and the worse one is discarded. This version of GA is named GENITOR or "steady-state GA". It was developed in [26] and used in [16] and [27] for MSS reliability optimization. In general, GENITOR outperforms the basic "generational GA".

Applying tabu search to each selected subspace

The role of this step is to look (in each subset selected by GA) for solutions by using an efficient TS. We define the neighbourhood structure. At each iteration of TS, the local transformations (or moves), that can be applied to the current solution, define a set of neighbouring solutions in a selected subspace as: *Neighbourhood(current solution)* = {*new solution* obtained by applying a single *move* to the *current solution*}. The move is chosen in such a way that the solution address does not change after a local transformation of the solution and the search process remains in the same subspace.

At each iteration, the best solution in a subset of *Neighbourhood(current solution)* is selected and considered as a tabu solution for some next iterations. This subset (referred to as the effective neighbourhood) is generated by eliminating the tabu solutions from *Neighbourhood(current solution)*. Tabus are stored in a short-term memory of the search (tabu list). A previously visited solution is added to the tabu list in order to forbid the repetition of configurations. That is, tabus are used to prevent cycling when moving away from local optima through non-improving moves. The size of the tabu list (denoted by length) is an important parameter regarding the efficiency of the heuristic, but its best value is not easy to be determined. A dynamic length is used, as it is usually found that it is more efficient to use a variable size tabu list [8, 24]. The termination criterion used can be specified in terms of a maximum number of local iterations without finding an improvement in the best-known solution.

As an important additional feature of our proposed TS, we use a penalty function while allowing infeasible solutions. This penalty function discourages, but allows, the TS algorithm to search into the infeasible boundary region. The idea of exploring around boundaries is known to be efficient for past implementations to solve the RAP of series-parallel systems. For example, the authors of [5] in their use of GA observed that better final feasible solutions could be found by permitting the exploration of the infeasible region, but by penalizing those solutions on the basis of the infeasibility degree. This idea is also used in [14, 17, 20] for solving the RAP of series-parallel binary-state systems with other meta-heuristics, in [19] for reliability optimization of a series system with multiple-choice and budget constraints, and in [24] for solving the RAP of series-parallel multi-state systems with TS. In TS, allowing infeasible solutions is a well-known idea [9, 11, 12]. An interesting way to find correct weights for constraint violations is to use self-adjusting penalties, *i.e.*, weights are adjusted dynamically on the basis of the recent history of the search. Weights are increased if only infeasible solutions were encountered in the last few iterations, and decreased if all recent solutions were feasible; see [9] for further details. Penalty weights can also be modified systematically to drive the search to cross the feasibility boundary of the search space and thus induce diversification. This technique, known as strategic oscillation, was first introduced in [11] and used since in several successful TS procedures. In the sequel, the TS algorithm applied to subspaces will be called TS-Sub.

The general algorithm

The general algorithm can then be summarized as follows (see also Figure 4):

Step 1. Generate randomly an initial population of N_s solutions and consider a set E initially empty.
Step 2. Select randomly two solutions named parent 1 and parent 2.
Step 3. Use 2-points crossover procedure to produce a new solution named child or offspring.
Step 4. If address (offspring) is in E (*i.e.,* we have already applied TS-Sub to this subspace), pass to step 6.
Step 5. If address (offspring) is not in E, the best offspring is calculated by applying TS-Sub, and the address of the new solution is added to E (to avoid applying TS-Sub to the same subspace more than once).
Step 6. Decode the new offspring and compare its solution with the worst solution in the population. The better solution joins the population and the worse one is discarded. Equivalent solutions are eliminated. If the population contains only one solution, regenerate randomly new solutions in the population.
Step 7. Repeat steps (2 to 6) N_{rep} times.
Step 8. Terminate the GA after N_c genetic cycles.

The proposed SP/GA approach tends to provide a balance between diversification and intensification. First, the selection of subspaces by GA allows the identification of promising regions in the search space. One intended role of the used GA is to facilitate the exploration by guiding the search to unvisited subspaces. This leads to a diversification in subspaces. Second, the intended role of TS is to search carefully and intensively around good solutions found in the past search. This leads to intensification by exploiting each subspace selected by GA. Furthermore, as explained in Section 3, when using a penalty function while allowing infeasible solutions, we induce diversification by encouraging exploration of the boundary of the feasible region.

4 Application of the proposed SP/TG approach

4.1 Application to the RAP

A given solution is defined by a system structure which is defined as a string $\mathbf{X}$ of dimension $L = \sum_{i=1}^{n} m_i$. The address of $\mathbf{X}$ is defined by:

$$address(\mathbf{X}) \;=\; \sum_{l=1}^{L} \mathbf{X}_l. \tag{14}$$

As a simple illustrative example, let consider a system of 2 components in series. Component 1 contains 2 elements of version 5 and 1 element of version 3 in parallel, and component 2 contains

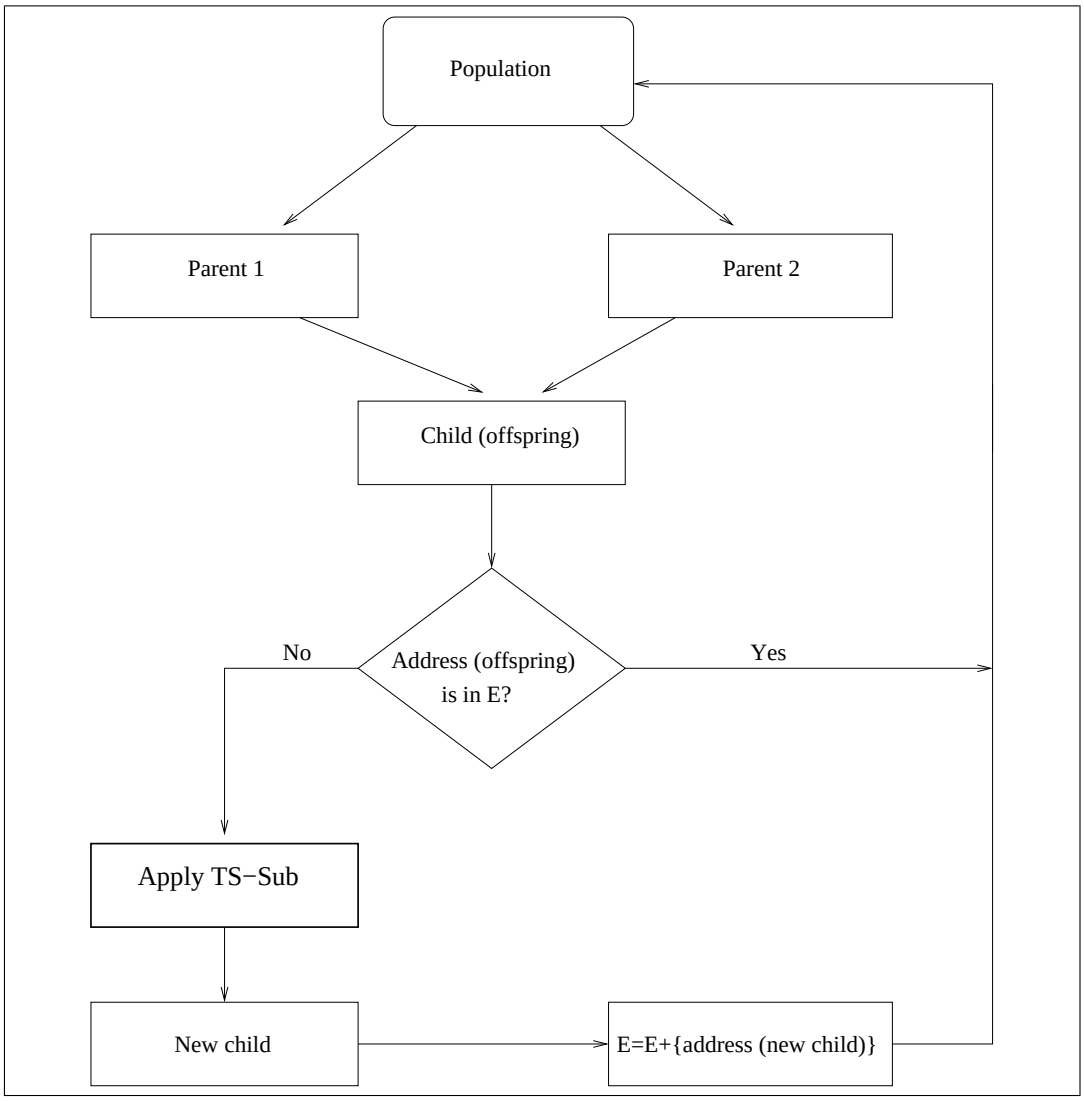

Figure 4: General algorithm

3 elements of version 9 in parallel. The address of $\mathbf{X}$ is simply obtained by summing the total number of elements (used in the system). That is, $address(\mathbf{X}) = 1 + 2 + 3 = 6$.

A search subspace of address r, denoted by S_r is defined as the set of solutions which have the same address, equal to r.

It follows from the above definitions that the lower bound of r is n and its upper bound is given by:

$$N = \sum_{i=1}^{n} \sum_{j=1}^{m_i} M_{ij}. \tag{15}$$

Note that $(S_r)_{n \leq r \leq N}$ is a partition of the search space S.

To let the GA look for the solution with maximum total reliability and with $W \geq W_0$ and

$C \geq C_0$, the fitness of a given individual is defined by:

$$F = \prod_{i=1}^{n} R_i(\mathbf{x}_i|k_i).\left(\frac{W}{W_0}\right)^{\alpha}.\left(\frac{C}{C_0}\right)^{\beta}. \quad (16)$$

By introducing a penalty function, we aim at encouraging the algorithm to explore the feasible region and infeasible region that is near the border of feasible area and discouraging, but allowing, search further into infeasible region.

In the previous subsection, the search space S has been partitioned into $(N-n+1)$ subspaces $(S_n, S_{n+1}, ..., S_N)$. To apply TS to a given subspace S_{r_i} $(n \leq r_i \leq N)$, let define the neighbourhood structure. At each iteration of TS, the local transformations (or moves), that can be applied to the current solution $\mathbf{X}$, define a set of neighboring solutions in S_{r_i} as:
neighbourhood $(\mathbf{X})$ = {series-parallel structures obtained by applying a single move to $\mathbf{X}$}.

The move applied to $\mathbf{X}$ consists in changing the number of elements in parallel by adding and subtracting one, if possible, for any component. In this way, $address(\mathbf{X})$ does not change after a local transformation of $\mathbf{X}$ and the search process remains in the same subspace. The penalized objective function to be minimized is given by F in Eq. (16).

4.2 Application to the ESP

Given an expansion plan for a system as a matrix $\mathbf{Y} = (\mathbf{H}, \mathbf{J})$, $(\mathbf{Y} = (Y_{\gamma l}),\ 1 \leq \gamma \leq \Gamma,\ 1 \leq l \leq 2n)$, the address of $\mathbf{Y}$ is defined by:

$$address(\mathbf{Y}) = \sum_{\gamma=1}^{\Gamma}\sum_{l=1}^{2n} Y_{\gamma l}. \quad (17)$$

For example, let consider the system such as the initial structure is empty, $n = 3,\ \Gamma = 3$,

$$\mathbf{H} = \begin{pmatrix} 2 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}, \mathbf{J} = \begin{pmatrix} 3 & 1 & 5 \\ 1 & 6 & 5 \\ 2 & 1 & 3 \end{pmatrix} \text{ and } \mathbf{Y} = \begin{pmatrix} 2 & 1 & 1 & 3 & 1 & 5 \\ 1 & 0 & 1 & 1 & 6 & 5 \\ 1 & 0 & 0 & 2 & 1 & 3 \end{pmatrix}.$$

The address of $\mathbf{Y}$ is simply obtained by summing the total number of elements (added in the total system) with the sum of version numbers, *i.e.*, $address(\mathbf{Y}) = 34$.

A search subspace of address r, denoted by S_r is defined as the set of solutions which have the same address, equal to r.

It follows from the above definitions that the lower bound of r is 0 and its upper bound is given by:

$$N = \sum_{\gamma=1}^{\Gamma}\sum_{i=1}^{n}[Max(H_{\gamma i}) + Max(J_{\gamma i})]. \quad (18)$$

$(S_r)_{0 \leq r \leq N}$ is a partition of the search space S.

The fitness of a given individual is defined as follows [27]:

$$F = \sum_{\gamma=1}^{\Gamma} \frac{1}{(1+\rho)^{t(\gamma)}} \sum_{i=1}^{n} H_{\gamma i} C_{iJ_{\gamma i}} + q \sum_{\gamma=1}^{\Gamma} \max\{0, A_0 - A_\gamma(\mathbf{H}, \mathbf{J})\}. \quad (19)$$

In the previous subsection, the search space S has been partitioned into $(N+1)$ subspaces $(S_0, S_1, ..., S_N)$. To apply TS to a given subspace S_{r_i} $(0 \leq r_i \leq N)$, let us define the neighbourhood structure. At each iteration of TS, the local transformations (or moves), that can be applied to the current solution $\mathbf{Y}$, define a set of neighboring solutions in S_{r_i} as: neighbourhood($\mathbf{Y}$) = {expansion plan for a system obtained by applying a single move to $\mathbf{Y}$}.

The move applied to $\mathbf{Y}$ consists in changing the number and the version of elements by adding and subtracting one, if possible, for any component of the solution matrix $\mathbf{Y} = (\mathbf{H}, \mathbf{J})$, ($\mathbf{Y} = (Y_{\gamma l})$, $1 \leq \gamma \leq \Gamma$, $1 \leq l \leq 2n$). In this way, $address(\mathbf{Y})$ does not change after a local transformation of $\mathbf{Y}$ and the search process remains in the same subspace. Each single move can be applied in three different ways:

1. Changing the number of elements in components ($H_{\gamma_1 l_1} \rightarrow H_{\gamma_1 l_1} + 1$ and $H_{\gamma_2 l_2} \rightarrow H_{\gamma_2 l_2} - 1$);
2. Changing version numbers in components ($J_{\gamma_1 l_1} \rightarrow J_{\gamma_1 l_1} + 1$ and $J_{\gamma_2 l_2} \rightarrow J_{\gamma_2 l_2} - 1$);
3. Changing both number of elements and version numbers in components ($H_{\gamma_1 l_1} \rightarrow H_{\gamma_1 l_1} + 1$ and $J_{\gamma_2 l_2} \rightarrow J_{\gamma_2 l_2} - 1$) or ($H_{\gamma_1 l_1} \rightarrow H_{\gamma_1 l_1} - 1$ and $J_{\gamma_2 l_2} \rightarrow J_{\gamma_2 l_2} + 1$).

The penalized objective function to be minimized is given by F in Eq. (19).

5 Computational results

For both problems, the algorithm was implemented in C^{++}. The numerical tests were completed on an Intel Pentium IV 3000 MHz DEC station 5000/240 with 1024 Mbytes of RAM running under Linux.

5.1 Redundancy allocation problem (RAP)

5.1.1 *Test problems for the RAP*

The test problems, used to evaluate the performance of the SP/TG algorithm when applied to the RAP, were originally proposed by Fyffe *et al.* in [6] and modified by Nakagawa and Miyazaki in [21]. Fyffe *et al.* [6] considered a system with 14 components and specified constraint limits of 130 units of system cost, 170 units of system weight and suppose 1-out-of n:G component redundancy. Nakagawa and Miyazaki [21] developed 33 variations of the Fyffe problem, where the cost constraint C is set to 130 and the weight constraint W is decreased incrementally from 191 units to 159 units. The element cost, weight and reliability values, as originally presented in [6], are reproduced in Table 1. For each component, there are three or four element choices.

Component i	Element choices											
	Choice 1			Choice 2			Choice 3			Choice 4		
	r_{i1}	c_{i1}	w_{i1}	r_{i2}	c_{i2}	w_{i2}	r_{i3}	c_{i3}	w_{i3}	r_{i4}	c_{i4}	w_{i4}
1	0.90	1	3	0.93	1	4	0.91	2	2	0.95	2	5
2	0.95	2	8	0.94	1	10	0.93	1	9	-	-	-
3	0.85	2	7	0.90	3	5	0.87	1	6	0.92	4	4
4	0.83	3	5	0.87	4	6	0.85	5	4	-	-	-

Component i	Element choices											
	Choice 1			Choice 2			Choice 3			Choice 4		
	r_{i1}	c_{i1}	w_{i1}	r_{i2}	c_{i2}	w_{i2}	r_{i3}	c_{i3}	w_{i3}	r_{i4}	c_{i4}	w_{i4}
5	0.94	2	4	0.93	2	3	0.95	3	5	-	-	-
6	0.99	3	5	0.98	3	4	0.97	2	5	0.96	2	4
7	0.91	4	7	0.92	4	8	0.94	5	9	-	-	-
8	0.81	3	4	0.90	5	7	0.91	6	6	-	-	-
9	0.97	2	8	0.99	3	9	0.96	4	7	0.91	3	8
10	0.83	4	6	0.85	4	5	0.90	5	6	-	-	-
11	0.94	3	5	0.95	4	6	0.96	5	6	-	-	-
12	0.79	2	4	0.82	3	5	0.85	4	6	0.90	5	7
13	0.98	2	5	0.99	3	5	0.97	2	6	-	-	-
14	0.90	4	6	0.92	4	7	0.95	5	6	0.99	6	9

Table 1: Data for RAP test problems [6]

5.1.2 *Size of the search space and number of subspaces for the RAP*

As in [18], the generation of initial solutions were controlled in a range between k_i and $M_{ij} - 4$ (inclusive). The total number of different solutions to be examined and the number of subspaces are simply given by the following equations:

$$\text{Size of the search space} = \prod_{i=1}^{n} \prod_{j=1}^{m_i} M_{ij}. \tag{20}$$

$$\text{Number of subspaces} = \sum_{i=1}^{n} \sum_{j=1}^{m_i} M_{ij} - n + 1. \tag{21}$$

Let us consider that different types of elements are allowed to reside in parallel, and assume that ($M_{ij} = 8\ \forall j,\ 1 \leq j \leq m_i$, and $\forall i,\ 1 \leq i \leq n$). This means simply that a maximum number of 8 s-identical elements are allowed for each component. In this case, the search space size is $8^{49} \approx 2.10^{44}$, while the number of subspaces is 371.

5.1.3 *Parameter settings for the RAP*

Preliminary numerical tests were used to set the values of the parameters. Different data are randomly generated and used to calibrate the parameters. Once the values of the parameters are set for these preliminary data, they are used for the variations of the problem instances to be solved in this paper. In this way, we avoid any parameter overfitting. The parameters' values are: $N_s = 50$, $N_{rep} = 50$, $N_c = 10$, $mnli = 200$, $\alpha = 1$ and $\beta = 0.3$.

5.1.4 *Comparing the best solutions of SP/TG and existing methods for the RAP*

The performance of the SP/TG heuristic is compared with the best-known heuristics for the RAP from previous literature, namely the genetic algorithm (GA) in [4], the linear approximation (LA) approach in [13], the tabu search (TS) algorithm in [14], the ant colony optimization (ACO)

in [17], the variable neighbourhood search (VNS) approach in [18], and the ant colony optimization coupled with the degraded ceiling algorithm (ACO/DC) in [20]. Ten runs of our algorithm were made using different random seeds for each problem instance. The best feasible solution over ten runs was first used for comparison. The results of the best solutions (*i.e.,* configuration, reliability, cost and weight) obtained by SP/TG for each of the 33 instances, are presented in Table 2. Table 3 gives a comparison between the best solutions of SP/TG and the best solutions obtained by the methods in references [4, 13, 14, 17, 18, 20]. Note that in these references, whenever the proposed algorithm is of a stochastic nature, 10 trials were performed and the best solution (among these 10 trials) was used as the final solution. The maximum reliability identified by these algorithms was then used to compare its performance to other algorithms. In Table 3, the best results are in boldface, indicating that generally SP/TG out-performed the existing approaches. More specifically, Table 3 shows that:

1. In 25 of the 33 test cases, the solutions found by our algorithm are better than those found by the genetic algorithm in [4], while the rest (*i.e.,* 8 cases) are as good as those they found.

2. Our algorithm out-performed the linear approximation approach in Hsieh [13] for all instances.

3. In 7 of the 33 test cases, the solutions found by our algorithm are better than those found by the tabu search algorithm in [14], while the rest (*i.e.,* 26 cases) are as good those they found.

4. In 9 of the 33 test cases, the solutions found by our algorithm are better than those found by the ant colony optimization in [17], while the rest (*i.e.,* 24 cases) are as good as those they found.

5. Our algorithm outperformed the variable neighbourhood search approach proposed in [18] for 6 instances, while the rest (*i.e.,* 27 instances) are as good as those they found.

6. In 2 of the 33 test cases, the solutions found by our algorithm are better than those found by the ant colony optimization coupled with the degraded ceiling algorithm in [20], while the rest (*i.e.,* 31 cases) are the same as those they found.

No	Solution encoding ($\mathbf{X}$)	$C(\mathbf{X})$	$W(\mathbf{X})$	$R(\mathbf{X})$
1	333,11,444,3333,222,22,111,1111,12,233,33,1111,11,34	130	191	0.986811
2	333,11,444,3333,222,22,111,1111,11,233,33,1111,12,34	130	190	0.986416
3	333,11,444,3333,222,22,111,1111,23,233,13,1111,11,34	130	189	0.985922
4	333,11,444,3333,222,22,111,1111,23,223,13,1111,12,34	130	188	0.985378
5	333,11,444,3333,222,22,111,1111,13,223,13,1111,22,34	130	187	0.984688
6	333,11,444,333,222,22,111,1111,23,233,33,1111,22,34	129	186	0.984176
7	333,11,444,3333,222,22,111,1111,23,223,13,1111,22,33	130	185	0.983505
8	333,11,444,333,222,22,111,1111,33,233,33,1111,22,34	130	184	0.982994
9	333,11,444,333,222,22,111,1111,33,223,33,1111,22,34	129	183	0.982256
10	333,11,444,333,222,22,111,1111,33,333,33,1111,22,33	130	182	0.981518
11	333,11,444,333,222,22,111,1111,33,233,33,1111,22,33	129	181	0.981027
12	333,11,444,333,222,22,111,1111,33,223,33,1111,22,33	128	180	0.980290

No	Solution encoding ($\mathbf{X}$)	$C(\mathbf{X})$	$W(\mathbf{X})$	$R(\mathbf{X})$
13	333,11,444,333,222,22,111,1111,33,223,13,1111,22,33	126	179	0.979505
14	333,11,444,333,222,22,111,1111,33,222,13,1111,22,33	125	178	0.978400
15	333,11,444,333,222,22,111,113,33,223,13,1111,22,33	126	177	0.977596
16	333,11,444,333,222,22,33,1111,33,223,13,1111,22,33	124	176	0.976690
17	333,11,444,333,222,22,13,1111,33,223,33,1111,22,33	125	175	0.975708
18	333,11,444,333,222,22,13,1111,33,223,13,1111,22,33	123	174	0.974926
19	333,11,444,333,222,22,13,1111,33,222,13,1111,22,33	122	173	0.973827
20	333,11,444,333,222,22,13,113,33,223,13,1111,22,33	123	172	0.973027
21	333,11,444,333,222,22,13,113,33,222,13,1111,22,33	122	171	0.971929
22	333,11,444,333,222,22,13,113,33,222,11,1111,22,33	120	170	0.970760
23	333,11,444,333,222,22,11,113,33,222,13,1111,22,33	121	169	0.969291
24	333,11,444,333,222,22,11,113,33,222,11,1111,22,33	119	168	0.968125
25	333,11,444,333,22,22,13,113,33,222,11,1111,22,33	118	167	0.966335
26	333,11,44,333,222,22,13,113,33,222,11,1111,22,33	116	166	0.965042
27	333,11,444,333,222,22,11,113,33,222,11,1111,22,33	117	165	0.963712
28	333,11,44,333,222,22,11,113,33,222,11,1111,22,33	115	164	0.962422
29	333,11,44,333,22,22,13,113,33,222,11,1111,22,33	114	163	0.960642
30	333,11,44,333,22,22,11,113,33,222,13,1111,22,33	115	162	0.959188
31	333,11,44,333,22,22,11,113,33,222,11,1111,22,33	113	161	0.958035
32	333,11,44,333,22,22,11,111,33,222,13,1111,22,33	112	160	0.955714
33	333,11,44,333,22,22,11,111,33,222,11,1111,22,33	110	159	0.954565

Table 2: Results of the best solutions obtained by SP/TG

No	W_0	GA[1]	LA[2]	TS[3]	ACO[4]	VNS[5]	ACO/DC[6]	SP/TG
1	191	0.98675	0.986711	**0.986811**	0.986745	**0.98681**	**0.986811**	**0.986811**
2	190	0.98603	0.986316	**0.986416**	0.985905	**0.98642**	0.986316	**0.986416**
3	189	0.98556	0.985724	**0.985922**	0.985773	**0.98592**	**0.985922**	**0.985922**
4	188	0.98503	0.985031	**0.985378**	0.985329	0.98487	**0.985378**	**0.985378**
5	187	0.98429	0.984153	**0.984688**	**0.984688**	0.98467	**0.984688**	**0.984688**
6	186	0.98362	0.983879	**0.984176**	0.983801	**0.98418**	**0.984176**	**0.984176**
7	185	0.98311	0.983387	**0.983505**	**0.983505**	**0.98351**	**0.983505**	**0.983505**
8	184	0.98239	0.982204	**0.982994**	**0.982994**	**0.98299**	**0.982994**	**0.982994**
9	183	0.98190	0.981466	**0.982256**	0.982206	**0.98226**	0.982225	**0.982256**
10	182	0.98102	0.979690	**0.981518**	0.981468	0.98147	**0.981518**	**0.981518**
11	181	0.98006	0.979280	**0.981027**	0.980681	**0.98103**	**0.981027**	**0.981027**
12	180	0.97942	0.978327	**0.980290**	**0.980290**	**0.98029**	**0.980290**	**0.980290**
13	179	0.97906	0.978055	**0.979505**	**0.979505**	**0.97951**	**0.979505**	**0.979505**
14	178	0.97810	0.976878	**0.978400**	**0.978400**	0.97838	**0.978400**	**0.978400**
15	177	0.97715	0.975400	0.977474	**0.977596**	**0.97760**	**0.977596**	**0.977596**
16	176	0.97642	0.974975	**0.976690**	0.976494	**0.97669**	**0.976690**	**0.976690**
17	175	0.97552	0.973500	**0.975708**	**0.975708**	**0.97571**	**0.975708**	**0.975708**
18	174	0.97435	0.972328	0.974788	**0.974926**	**0.97493**	**0.974926**	**0.974926**
19	173	0.97362	0.970531	**0.973827**	**0.973827**	0.97381	**0.973827**	**0.973827**

No	W_0	GA[1]	LA[2]	TS[3]	ACO[4]	VNS[5]	ACO/DC[6]	SP/TG
20	172	0.97266	0.969232	**0.973027**	**0.973027**	**0.97303**	**0.973027**	**0.973027**
21	171	0.97186	0.967896	**0.971929**	**0.971929**	**0.97193**	**0.971929**	**0.971929**
22	170	**0.97076**	0.966776	**0.970760**	**0.970760**	**0.97076**	**0.970760**	**0.970760**
23	169	0.96922	0.965612	**0.969291**	**0.969291**	**0.96929**	**0.969291**	**0.969291**
24	168	**0.96813**	0.964150	**0.968125**	**0.968125**	**0.96813**	**0.968125**	**0.968125**
25	167	**0.96634**	0.962990	**0.966335**	**0.966335**	**0.96634**	**0.966335**	**0.966335**
26	166	**0.96504**	0.961210	**0.965042**	**0.965042**	**0.96504**	**0.965042**	**0.965042**
27	165	**0.96371**	0.959923	**0.963712**	**0.963712**	**0.96371**	**0.963712**	**0.963712**
28	164	**0.96242**	0.958601	**0.962422**	**0.962422**	**0.96242**	**0.962422**	**0.962422**
29	163	**0.96064**	0.957317	0.959980	**0.960642**	**0.96064**	**0.960642**	**0.960642**
30	162	0.95912	0.955547	0.958205	**0.959188**	**0.95919**	**0.959188**	**0.959188**
31	161	**0.95803**	0.954101	0.956922	**0.958034**	**0.95804**	**0.958034**	**0.958035**
32	160	0.95567	0.952953	0.955604	**0.955714**	0.95567	**0.955714**	**0.955714**
33	159	0.95432	0.950800	0.954325	**0.954564**	**0.95457**	**0.954564**	**0.954565**

Table 3: Comparison of the best solutions among heuristics

[1] GA genetic algorithm [4]
[2] LA linear approximation [13]
[3] TS tabu search [14]
[4] ACO ant colony optimization [17]
[5] VNS variable neighborhood search [18]
[6] ACO/DC ant colony optimization and degraded ceiling [20]

5.1.5 *Robustness of the SP/TG algorithm for the RAP*

To measure the robustness of the SP/TG algorithm, the standard deviations and the average reliability, over ten runs in each instance, are given in Table 4. We remark from this table that for each instance, the standard deviation is very low. This implies that the proposed method is robust. The low standard deviation of SP/TG can be interpreted as a sign of insensitivity to the initial solution and the random number seed.

No	W_0	Max R	Mean R	Std dev	Average evaluations
1	191	0.986811	0.986811	0.000000	327,749
2	190	0.986416	0.986416	0.000000	386,116
3	189	0.985922	0.985922	0.000000	227,884
4	188	0.985378	0.985327	0.000091	480,042
5	187	0.984688	0.984593	0.000161	169,061
6	186	0.984176	0.983762	0.000764	213,086
7	185	0.983505	0.983465	0.000127	151,248
8	184	0.982994	0.982964	0.000009	133,120
9	183	0.982256	0.982081	0.000490	263,815
10	182	0.981518	0.981002	0.001124	285,481

No	W_0	Max R	Mean R	Std dev	Average evaluations
11	181	0.981027	0.980425	0.000390	231,039
12	180	0.980290	0.979630	0.000892	205,258
13	179	0.979505	0.978908	0.000449	195,794
14	178	0.978400	0.978151	0.000403	203,054
15	177	0.977596	0.977596	0.000000	184,477
16	176	0.976690	0.976651	0.000083	247,013
17	175	0.975708	0.975562	0.000308	282,445
18	174	0.974926	0.974675	0.000528	207,254
19	173	0.973827	0.973461	0.000743	183,166
20	172	0.973027	0.972591	0.000702	183,513
21	171	0.971929	0.971191	0.000952	228,454
22	170	0.970760	0.970090	0.000549	193,096
23	169	0.969291	0.968919	0.000743	170,768
24	168	0.968125	0.967766	0.001137	216,559
25	167	0.966335	0.965544	0.002500	286,547
26	166	0.965042	0.964826	0.000452	253,756
27	165	0.963712	0.963712	0.000000	216,990
28	164	0.962422	0.962422	0.000000	179,041
29	163	0.960642	0.960642	0.000000	303,201
30	162	0.959188	0.959138	0.000106	190,968
31	161	0.958035	0.958035	0.000000	242,367
32	160	0.955714	0.955714	0.000000	211,747
33	159	0.954565	0.954565	0.000000	190,198

Table 4: Maximum reliability, average reliability, standard deviation and average evaluation of the SP/TG algorithm

5.1.6 *Comparing the computational effort of SP/TG and existing methods for the RAP*

When considering the computational effort, the proposed algorithm requires less number of iterations (*i.e.,* generated solutions, during the whole search process) than TS in [14] and requires a larger number of iterations than GA in [4, 5] and VNS in [18]. From the previous literature, the numbers of iterations given for the following methods are:

- the number of solutions generated in GA [4, 5] is 48,000 (a population size of 40 chromosomes and 1200 generations),
- TS in [14] evaluated an average of 350,000 solutions,
- the ACO algorithm in [17] needs about 100,000 evaluations or more,
- the ACO/DC approach needs about 150,000 evaluations or more,
- the average number of evaluations used by VNS in [18] is around 120,000.

For each instance, the average numbers of solutions evaluated by SP/TG over ten runs are given in the last column of Table 4. The number of solutions evaluated by our SP/TG is approximately 231,645 on average (obtained by averaging the values in the last column of Table 4).

Furthermore, our approach is compared to the others with a similar amount of evaluations as follows: for each test case, we stop our algorithm after the number of evaluations needed for the other algorithm to reach its best solution, and we compare the solution of our algorithm over ten runs with this best solution. The results from this comparison are as follows:

- SP/TG solutions are better than GA solutions in 18 of the 33 test cases. GA solutions are better than SP/TG solutions in 9 of the 33 test cases.
- SP/TG solutions are better than ACO solutions in 7 of the 33 test cases. ACO solutions are better than SP/TG solutions in 6 of the 33 test cases.
- SP/TG solutions are better than VNS solutions in 3 of the 33 test cases. VNS solutions are better than SP/TG solutions in 5 of the 33 test cases.
- SP/TG solutions are better than ACO/DC solutions in 1 of the 33 test cases. ACO/DC solutions are better than SP/TG solutions in 3 of the 33 test cases.
- SP/TG solutions are better than TS solutions in 7 of the 33 test cases, while the rest are as good those they found.

5.2 Expansion-scheduling problem (ESP)

5.2.1 *Test problems for the ESP*

The test problems, used to evaluate the performance of the SP/TG algorithm when applied to the ESP, were proposed by Levitin in [27]. The system to be optimized is a power-station coal transportation system which supplies boilers, and which has 5 basic components connected in series. Each component may contain a set of parallel elements. For each component, different elements types are available. Each element is considered as a binary-state unit, while the system is a MSS with series-parallel structure. Two cases are considered:

- The case where the initial structure-set is empty ($Str_0 = \emptyset$). In this case, the problem is to determine the initial system structure as well as its expansion plan.
- The case where the initial system structure exists at stage 0 ($Str_0 = Str_0^*$). In this case, the problem is to determine the system expansion plan.

The reader is referred to reference [27] for all the input data including:

- The characteristics of available elements, *i.e.*, cost, availability and nominal capacity values.
- The data of the "piecewise cumulative boiler system demand curves" at 5 stages and times from the present to the beginning of these future stages.
- The initial system structure to be used in the second case above.

5.2.2 *Size of the search space and number of subspaces for the ESP*

As in [27], the generation of initial solutions were controlled in a range between 0 and 6. The total number of different solutions to be examined and the number of subspaces are simply given by the following equations:

$$\text{Size of the search space} = \prod_{\gamma=1}^{\Gamma} \prod_{i=1}^{n} Max(J_{\gamma i}) Max(H_{\gamma i}). \tag{22}$$

$$\text{Number of subspaces} = \sum_{\gamma=1}^{\Gamma} \sum_{i=1}^{n} Max(J_{\gamma i}) + Max(H_{\gamma i}) + 1. \tag{23}$$

Let us consider that different types of elements are allowed to reside in parallel, and assume that $((H_{\gamma i}) = 6 \ \forall \gamma, \ 1 \leq \gamma \leq \Gamma$, and $\forall i, \ 1 \leq i \leq n)$. This means simply that a maximum number of 8 elements are allowed for each component. In this case, the search space size is $3, 25.6^{25}.10^{18} \approx 10^{38}$, while the number of subspaces is 296.

5.2.3 *Parameter settings for the ESP*

To avoid parameter overfitting, a procedure similar to that used for the RAP was used again. That is, preliminary numerical tests were used to set the values of the ESP parameters. Different data are randomly generated and used to calibrate the parameters. Once the values of the parameters are set for these preliminary data, they are used for the problem instances to be solved. The parameters' values are: $N_s = 500$, $N_{rep} = 200$, $N_c = 100$, $mnli = 200$, $q = 500$.

5.2.4 *Comparing the best solutions of SP/TG and existing methods for the ESP*

While a large number of methods have been proposed for the RAP, the only existing methods to solve the ESP are a genetic algorithm (GA) in [27] and an ant colony optimization algorithm in [35]. However, the best-known results are those obtained by the genetic algorithm in [27]. Therefore, the performance of the SP/TG heuristic applied to the ESP is compared with this GA. The system availability of the final solution is rounded to three digits after the decimal point, in order to compare our results with those obtained in [27]. Ten runs of SP/TG algorithm were made using different random seeds for each problem instance. Tables **??** and **??** show the best expansion plans (among the ten runs) obtained for $Str_0 = Str_0^*$ and $Str_0 = \emptyset$, with three different desired values of A_0 ($A_0 = 0.950$, $A_0 = 0.970$ and $A_0 = 0.990$). Cost figures are in millions of dollars. The interest rate ρ is chosen to be 0.1. Expansion of each component i at stage γ is presented in the form $X_{i\gamma}(J_{i\gamma})$. Table 7 gives a comparison between the best solutions of SP/TG and the best solutions obtained with GA in [27], where the number of runs used is 100 and the best solution over these runs are given. By executing our heuristic over only 10 trials, the obtained results should be *a fortiori* conclusive. The percent that one solution improves upon another is defined in terms of objective function as:

$$\text{MPCI} = 100\% \times \frac{\text{(Minimal GA Cost - Minimal SP/TG Cost)}}{\text{Minimal GA Cost}}.$$

In Table 7, the best results are in boldface, indicating that SP/TG out-performed the existing approaches. More specifically, Table 7 shows that, in terms of the best objective function (minimum

over 10 runs), in 5 of 6 test problems, the solutions found by our algorithm SP/TG are better than those found by GA in [27], while in the remaining instance both algorithms returns the same solution.

A_0	i		$\gamma=1$	$\gamma=2$	$\gamma=3$	$\gamma=4$	$\gamma=5$
		$C(1)=0.645$					$C=4.127$
		A_γ	0.950	0.951	0.959	0.952	0.960
	1			1(4)			
	2		2(5)		1(5)	1(5)	
0.950	3				1(4)		1(2)
	4		1(7)	1(7)			
	5		1(3)			1(4)	
		$C(1)=3.323$					$C=6.484$
		A_γ	0.970	0.971	0.986	0.972	0.970
	1		1(6)	1(6)		1(6)	1(6)
	2		1(3)	1(5)		1(5)	2(5)
0.970	3		1(4)		1(2)		
	4		2(7)				
	5		1(3)				1(4)
		$C(1)=5.503$					$C=7.834$
		A_γ	0.991	0.991	0.990	0.996	0.995
	1		1(6)	1(6)	2(6)		
	2		1(5)	2(5)	1(5)		1(5)
0.990	3		1(2)			1(3)	
	4		1(7)	1(7)	1(7)		
	5		1(3)		1(3)		

Table 5: The best solution found by SP/TG algorithm among the 10 runs ($Str_0 = Str_0^*$)

A_0	i		$\gamma=1$	$\gamma=2$	$\gamma=3$	$\gamma=4$	$\gamma=5$
		$C(1)=10.160$					$C=14.578$
		A_γ	0.952	0.960	0.951	0.951	0.950
	1		3(4)				2(6)
	2		2(3)	1(5)	1(5)	1(3)	1(5)
0.950	3		1(1)	1(2)			
	4		3(7)	1(9)		1(9)	1(7)
	5		2(3)	1(3)		1(3)	
		$C(1)=12.727$					$C=16.909$
		A_γ	0.970	0.971	0.970	0.996	0.971
	1		3(4)				
	2		3(3)		2(5)		1(5)
0.970	3		2(2)	1(4)		1(2)	
	4		4(7)		1(9)		
	5		3(3)		1(3)		

A_0	i	$\gamma = 1$	$\gamma = 2$	$\gamma = 3$	$\gamma = 4$	$\gamma = 5$
		$C(1) = 17.285$				$C = 19.270$
		A_γ 0.994	0.993	0.990	0.996	0.991
	1	3(4)				
	2	2(3)	1(4)	3(5)		
0.990	3	3(2)			1(3)	
	4	3(7)	1(9)	1(7)		
	5	3(3)		1(3)		

Table 6: The best solution found by SP/TG algorithm among the 10 runs ($Str_0 = \emptyset$)

Str_0	A_0	Minimal SP/TG Cost	Minimal GA Cost	$\%MPCI$
$Str_0 = Str_0^*$	0.950	**4.127**	**4.127**	00.00
	0.970	**6.484**	6.519	00.54
	0.990	**7.834**	7.859	00.32
$Str_0 = \emptyset$	0.950	**14.578**	14.606	00.19
	0.970	**16.909**	17.206	01.72
	0.990	**19.270**	19.350	00.41

Table 7: Total costs for the proposed and existing algorithms for the ESP

5.2.5 *Robustness of the SP/TG algorithm for the ESP*

To measure the robustness of the SP/TG algorithm when applied to the ESP, the standard deviations and the average minimal cost, over ten runs in each instance, are given in Table 8. We remark from this table that for each instance, the standard deviation is low. This confirms that the proposed method is robust.

Str_0	A_0	Min cost	Mean cost	Std dev	Average evaluations
$Str_0 = Str_0^*$	0.950	4.127	4.135	0.008	1,267,647
	0.970	6.484	6.532	0.104	1,614,513
	0.990	7.834	7.862	0.030	1,827,918
$Str_0 = \emptyset$	0.950	14.578	14.587	0.006	3,140,892
	0.970	16.909	17.558	0.219	2,484,106
	0.990	19.270	19.474	0.246	2,937,715

Table 8: Minimum cost, average cost, standard deviation and average evaluation of the SP/TG algorithm

5.2.6 *Comparing the computational effort of SP/TG and existing methods for the ESP*

For each instance, the average numbers of solutions evaluated by SP/TG over ten runs are given in the last column of Table 8. The number of solutions evaluated by the proposed SP/TG is $2,2.10^6$ on average (obtained by averaging the values in the last column of Table 8). As it is reported in [27] that the number of solutions generated in GA is 2.10^6, the SP/TG algorithm requires only 10 % more computational effort to get the high quality solutions shown above.

5.3 Convergence

The convergence curves of SP/TG were drawn for all the test cases, showing similar behaviour in all these cases. Figures 5 and 6 show for example these curves for the first instance of each problem ($W_0 = 191$ for the RAP, and $A_0 = 0.950$; $Str_0 = Str_0^*$ for the ESP). We remark that even if the solutions of our algorithm are not good during the first iterations of the search process, they become quickly very good.

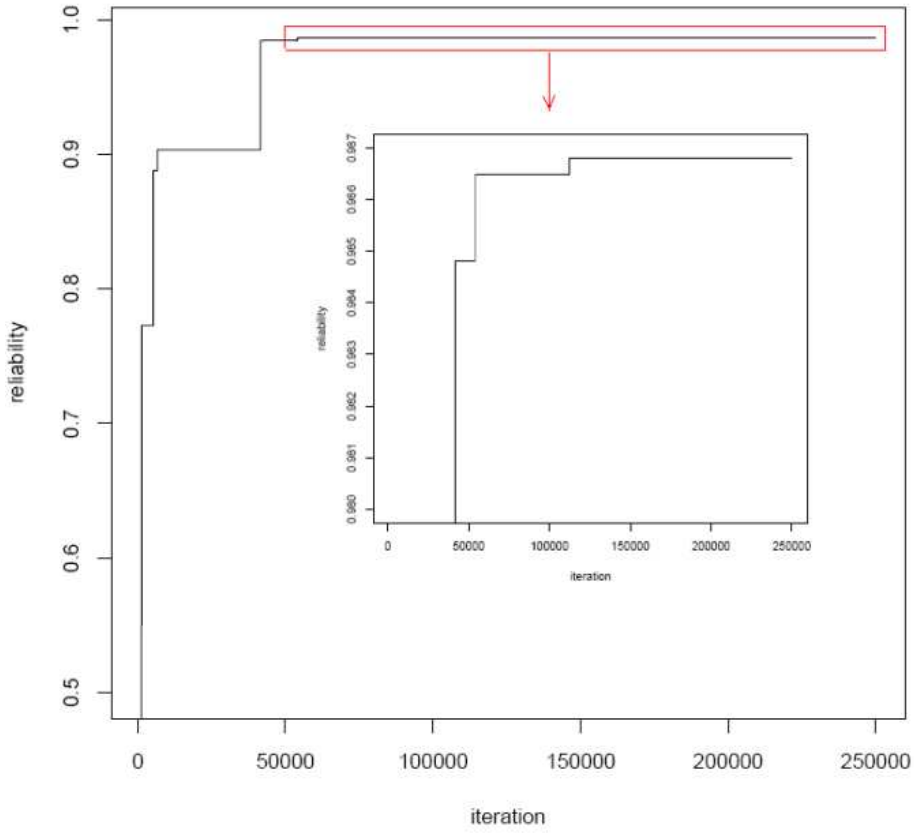

Figure 5: An example of convergence curve for the RAP

6 Conclusion

This paper developed an efficient approach, called SP/TG, to solve two reliability optimization problems for series-parallel systems. The first one is the redundancy allocation problem for

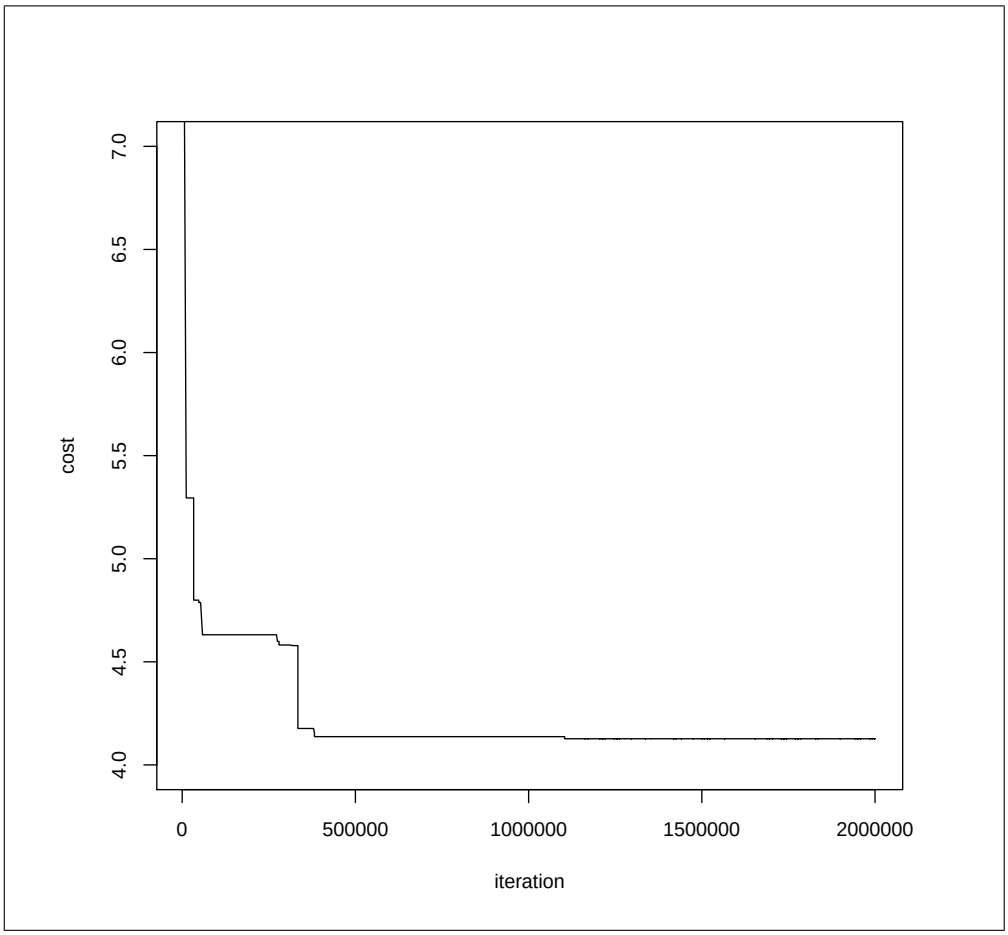

Figure 6: An example of convergence curve for the ESP

series-parallel binary-state systems, and the second one is the expansion-scheduling problem of series-parallel multi-state systems. The newly developed SP/TG approach combines GA, TS and the idea of partitioning the search space. After an appropriate division of the search space into a set of disjoint subspaces, the role of GA is to select the subspaces, while TS is applied to each selected subspace. By combining two meta-heuristics, the SP/TG provides a balance between diversification and intensification. For both problems, the experimental results showed that the solutions found by the SP/TG approach are better than or are comparable with the best-published results from the literature. As this hybrid approach has been successful for solving two typical design optimization problems from binary-state and multi-state reliability, it may represent a general approach to solve other reliability optimization problems where it is possible to properly partition the search space. Current works concern the application of the SP/TG approach developed in this paper to other reliability optimization problems.

Acknowledgements

The authors would like to thank the editor, and the anonymous reviewers for their comments and helpful questions. They would like to thank also the Natural Sciences and Engineering Research Council of Canada (NSERC) for financial support.

References

[1] Bellman R. E. and Dreyfus E., Dynamic programming and reliability of multicomponent devices, *Oper Res* 1958;6:200-206.

[2] Bulfin R. L. and Liu C. Y., Optimal allocation of redundant components for large systems, *IEEE Trans Reliab* 1985;34:241-247.

[3] Chern M. S. On the computational complexity of reliability redundancy allocation in a series system. *Operations Research Letters* 1992;11:309-315.

[4] Coit D.W. and Smith A.E., Reliability optimization of series-parallel systems using a genetic algorithm, *IEEE Trans Reliab* 1996;45(2):254-260.

[5] Coit, D.W. and Smith, A.E. Penalty guided genetic search for reliability design optimization. Computers & Industrial Engineering 1996;30(4):895-904.

[6] Fyffe D.E., Hines W.W. and Lee N.K. System reliability allocation and a computational algorithm. *IEEE Trans Reliab* 1968;R-17(2):64-69.

[7] Gen M., Ida K., Tsujimura Y. and Kim C. E., Large scale 01 fuzzy goal programming and its application to reliability optimization problem, *Comput Indust Eng* 1993;24:539-549.

[8] Gendreau M. Recent Advances in Tabu Search, in Essays and Surveys in Metaheuristics, C.C. Ribeiro and P.Hansen(eds.). Kluwer Academic Publishers 2002:369-377.

[9] Gendreau, M., Hertz, A. and Laporte, G. A tabu search heuristic for the vehicle routing problems. Management Science 1994;40:1276-1290.

[10] Ghare M. and Taylor R.E., Optimal redundancy for reliability in series system, *Oper Res* 1969;17:838-847.

[11] Glover F., Heuristics for Integer Programming Using Surrogate Constraints, *Decision Sciences* 1977;8:156-166.

[12] Glover, F. and Laguna M., Tabu Search. Kluwer Academic Publishers, Boston 1997.

[13] Hsieh Y. C., A linear approximation for redundant reliability problems with multiple component choices. *Computers & Industrial engineering* 2002;44:91-103.

[14] Kulturel-Konak S., Smith A.E. and Coit D.W., Efficiently solving the redundancy allocation problem using tabu search. *IIE transactions* 2003;35:515-526.

[15] Levitin G., Lisnianski A., Ben-Haim H. and Elmakis D., Structure optimization of power system with different redundant elements. *Electric Power Syst Res* 1997;43(1):19-27.

[16] Levitin G., Lisnianski A., Ben-Haim H. and Elmakis D., Redundancy optimization for series-parallel multi-state systems. IEEE Trans on Reliab 1998;7(2):165-172.

[17] Liang Y.C. and Smith A.E., An Ant Colony Optimization Algorithm for the Redundancy Allocation Problem (RAP). *IEEE transactions on Reliability* 2004;53(3):417-423.

[18] Liang Y.C. and Chen Y.C., Redundancy allocation of series-parallel systems using a variable neighborhood search algorithm. *Reliability Engineering and System Safety* 2007;92:323-331.

[19] Nahas N. and Nourelfath M., Ant system for reliability optimization of a series system with multiple-choice and budget constraints. *Reliab Eng Syst Saf* 2005;87(1):1-12.

[20] Nahas N., Nourelfath M. and Ait-Kadi D., Coupling ant colony and the degraded ceiling algorithm for the redundancy allocation problem of series-parallel systems. *Reliab Eng Syst Saf.* 2007;92(2):211-222.

[21] Nakagawa Y. and Miyazaki S., Surrogate constraints algorithm for reliability optimization problems with two constraints, *IEEE Trans Reliab* 1981;R-30(2):175-180.

[22] Misra K. B. and Sharma U., An efficient algorithm to solve integer-programming problems arising in system-reliability design, *IEEE Trans Reliab* 1991;40(1):81-91.

[23] Osman I. H. and Laporte G., Metaheuristics: a bibliography, *Annals of Operations Research* 1996;63:513-623.

[24] Ouzineb M., Nourelfath M. and Gendreau M., Tabu search for the redundancy allocation problem of homogenous series-parallel multi-state systems, *Reliab Eng Syst Safety* 2008;93(8):1257-1272.

[25] Tillman F.A., Hwang C-L., Kuo W., Determining component reliability and redundancy for optimum system reliability, *IEEE Trans on Reliab* 1997;R-26(3):162-165.

[26] Whitley D. and Kauth J., GENITOR: A different genetic algorithm, *Tech. ReportCS-88-101*, Colorado State University, 1988.

[27] Levitin G., Multistate Series-Parallel System Expansion-Scheduling Subject to Availability Constraints, *IEEE Transactions on reliability* 2000;49(1):71-79.

[28] Levitin G., Universal generating function in reliability analysis and optimization, Springer-Verlag, 2005.

[29] Lisnianski A. and Levitin G., Multi-state system reliability. Assesment, optimization and applications, World scientific 2003.

[30] Ushakov I., Optimal standby problems and a universal generating function, *Sov J Computing System Science* 1987;25(4);79-82.

[31] Ushakov I., Universal generating function, *Sov. J. Computing System Science* 1986;24(5);118-129.

[32] Nourelfath M., Nahas N. and Zeblah A., An ant colony approach to redundancy optimization for multi-state system, *Int Conf on Ind Engng and Prod Manage* (IEPM'2003) Porto-Portugal, 2003.

[33] Ramirez M. and Coit D. W., A Heuristic For Solving The Redundancy Allocation Problem For Multi-State Series-Parallel Systems, *Reliability Engineering and System Safety* 2004;83;341-349.

[34] Kuo W., Rajendra V.P., Tillman F.A., Hwang C.L., Optimal reliability design-fundamentals and applications. Cambridge University Press, Cambridge, 2001.

[35] Nourelfath M. and Zeblah A., Ant colony optimization for the expansion planning problem of multi-state systems, *5th International Industrial Engineering Conference*, Quebec, October 26-29, 2003.