EXACT COST MINIMIZATION OF A SERIES-PARALLEL RELIABLE SYSTEM WITH MULTIPLE COMPONENT CHOICES USING AN ALGEBRAIC METHOD

J. GAGO-VARGAS, M.I. HARTILLO-HERMOSO, J. PUERTO, AND J.M. UCHA-ENRÍQUEZ

Abstract. The redundancy allocation problem is formulated minimizing the design cost for a series-parallel system with multiple component choices while ensuring a given system reliability level. The obtained model is a nonlinear integer programming problem with a nonlinear, nonseparable constraint. We propose a method based on the construction of a test set of an integer linear problem, which allows us to obtain an exact solution of the problem. It is compared to other approaches in the literature and standard nonlinear solvers.

1. Introduction

System reliability is considered an important measure in the engineering design process. A series system is similar to a chain composed of links, each one representing a subsystem. The failure of one of these components means the failure of the whole system. In order to avoid this, it is usual to use redundant components in parallel to guarantee a certain level of reliability. These systems are called series-parallel systems.

Determining the optimal number of components in each subsystem is the so called reliability optimization problem. Two different approaches are usual:

- maximize system reliability subject to system budget constraint, or
- minimize system cost subject to a required level of reliability.

Both problems are nonlinear integer programming problems, and they are NP-hard [6]. There are very few papers looking for their exact solutions, due to the difficulty of the problems. Those works use essentially dynamic programming [21], branch and bound methods [12], or Lagrangian relaxation [16], among other techniques. On the contrary, in the literature there are many heuristics and metaheuristic algorithms, such as those based on Genetic Algorithms [7], Tabu Search [15] or Ant Colony Optimization [2], among others.

In this paper we study the exact solution of one of the versions of the problem that minimizes the cost function of the chosen design, subject to a nonlinear constraint which describes the reliability of the considered system. For a fixed subsystem, its inner components can be considered equal, as in [12], or different, as in [16] and [21]. If the components are equal, the reliability function is separable and convex, and the problem can be reduced to a linear knapsack problem [12].

In the case of multiple component choices, the reliability function is no longer separable. In [21], the solution is found using dynamic programming methods. That approach presents two stages. In the first one the problem is restricted to each

1991 *Mathematics Subject Classification.* Primary: 90C10, 13P10. Secondary: 90B25.
Key words and phrases. Reliability, Integer programming, Test set.

subsystem, with a level of reliability. Under this assumption the reliability function is separable, and the optimization problem reduces to a knapsack problem. Those knapsack problems need discrete values to be solved, and the level of accuracy is determined by a constant L. So the exactness of solution is not guaranteed, a priori. The reliability levels of the subsystems are determined using some lower and upper bounds that also depend on the level of accuracy L, and a new dynamic programming process is needed.

The solution method shown in [16] uses an algorithm based on Lagrangian relaxations over two linear relaxations of the original problem. The first relaxation consists of deleting the nonlinear reliability constraint, and adding certain linear constraints, one for each subsystem. The second relaxation assumes that the same type of component is going to be used in every subsystem, so that the problem reduces to one similar as in [12].

The algorithm of [16] is what their authors call a *cut and partition scheme* (a geometric branch and bound). The solution space is partitioned in boxes, which are divided and discarded under certain conditions. The cuts are built from the best bound feasible solution of the above mentioned Lagrangian relaxations. Such bounds allow to remove certain boxes depending on the improvement with respect to the current best point.

We address the problem via a different approach based on Gröbner bases, which in this framework gives better computational results than other methods in the literature. As an introduction on this subject, we recommend the text books [1], [4] and [10].

Gröbner bases were applied to integer linear programming, for the first time, in [8]. Later, Tayur et al. [20] introduced a new application framework, which solves nonlinear integer programming problems, with a linear objective function. This is exactly our framework, as in [5].

First, we consider a relaxed integer programming problem where all the restrictions are linear. Then we find the solution of the relaxed integer problem by computing a test set. Next, using the so called reverse test set, we can solve the complete problem, generating paths from the solution of the relaxed problem to a solution of the complete one. These paths increase the cost function at each step. We also obtain a feasible solution with a greedy algorithm to begin with a good upper bound with respect to the cost function.

A test set for a linear integer programming problem is a set of directions that can be used to design descending algorithms with respect to a linear cost function. A test set can be computed from a Gröbner basis of the toric ideal associated with the linear restrictions, with respect to an order given by the cost function.

One of the most time consuming tasks in the process described above is usually the calculation of the Gröbner basis. In this paper we consider the relaxed linear integer programming problem obtained by removing the nonlinear reliability function. For this problem we explicitly give the associated Gröbner basis, and so the test set to solve the main problem. We point out that for this problem the Gröbner basis can be computed in polynomial time with respect to the number of systems n and the number of different types of available components k_i for the i-th subsystem, $i = 1, \ldots, n$.

The organization of the paper is as follows. In Section 2, we introduce the notation and describe the model of a parallel-series system with multiple component

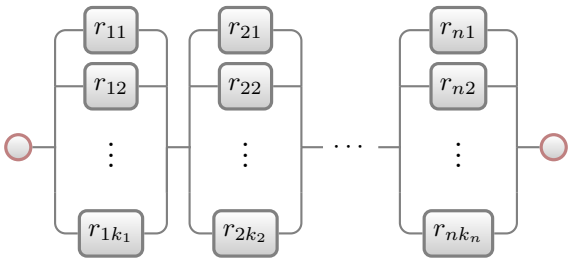

FIGURE 1. A series-parallel system with multiple choice components

choices. In Section 3, a greedy algorithm is described in order to compute a feasible point. Section 4 is devoted to a brief introduction to the essential facts about Gröbner bases and how they can be applied to our problem. Section 5 contains our main result: a closed formula for the test set of the underlying integer linear problem. Our computational experiments are reported in Section 6, with comparisons to other exact approaches. Finally we draw some concluding remarks in Section 7.

2. General model

In order to formulate the problem, some notation used throughout the paper is first introduced.

- n number of subsystems.
- k_i number of different types of available components for the i-th subsystem, $i = 1, \ldots, n$.
- r_{ij} reliability of the j-th component for the i-th subsystem, $i = 1, \ldots, n$, $j = 1, \ldots, k_i$.
- c_{ij} cost of the j-th component for the i-th subsystem, $i = 1, \ldots, n$, $j = 1, \ldots, k_i$.
- l_{ij}, u_{ij} lower/upper bounds of number of j components for the i-th subsystem, $i = 1, \ldots, n$, $j = 1, \ldots, k_i$.
- R_0 admissible level of reliability of the whole system.
- x_{ij} number of j components used in the i-th subsystem, $i = 1, \ldots, n$, $j = 1, \ldots, k_i$.

In our model, some assumptions are considered:

- Components have two states: working or failed.
- The reliability of each component is known and is deterministic.
- Failure of individual components are independent.
- Failed components do not damage other components or the system, and they are not repaired.

This model is illustrated in Figure 1. It is a system with n subsystems with the notation introduced before. The optimization problem can be formulated as:

$$
\begin{array}{lll}
(RP) \quad \min & \sum_{i=1}^{n}\sum_{j=1}^{k_i} c_{ij}x_{ij} & \\
\text{s. t.} & & \\
& R(x) \geq R_0, & \\
& \sum_{j=1}^{k_i} x_{ij} \geq 1, & i = 1,\ldots,n, \\
& 0 \leq l_{ij} \leq x_{ij} \leq u_{ij}, & i = 1,\ldots,n, \\
& & j = 1,\ldots,k_i, \\
& x_{ij} \in \mathbb{N} & \text{for all } i,j,
\end{array}
\tag{1}
$$

where $R(x) = \prod_{i=1}^{n}(1-\prod_{j=1}^{k_i}(1-r_{ij})^{x_{ij}})$. The first n linear inequalities in Problem (RP) assert that each subsystem must have, at least, one component.

As usual, we can make a change of variables $y_{ij} = x_{ij} - l_{ij}$, so that we can assume $l_{ij} = 0$. This does not alter the equations of Problem (RP), and some of the last equations can be redundant. Hence it can be assumed $l_{ij} = 0$ without loss of generality, and:

$$
\begin{array}{lll}
(RP) \quad \min & \sum_{i=1}^{n}\sum_{j=1}^{k_i} c_{ij}x_{ij} & \\
\text{s. t.} & & \\
& R(x) \geq R_0, & \\
& \sum_{j=1}^{k_i} x_{ij} \geq 1, & i = 1,\ldots,n. \\
& 0 \leq x_{ij} \leq u_{ij}, & i = 1,\ldots,n, \\
& & j = 1,\ldots,k_i, \\
& x_{ij} \in \mathbb{N} & \text{for all } i,j.
\end{array}
$$

A feasible solution is sometimes called a reliable solution because it ensures a reliability greater than or equal to R_0.

3. Computing a reliable system with a greedy procedure

The main step used in the algebraic algorithm described in this article is to consider an integer linear problem (LRP), obtained by relaxing the nonlinear constraint of the original problem (RP). There is only one nonlinear constraint in (RP), the equation which ensures the reliability of the whole system. Removing the nonlinear constraint, we get an integer linear programming problem:

$$
\begin{array}{lll}
(LRP) \quad \min & \sum_{i=1}^{n}\sum_{j=1}^{k_i} c_{ij}x_{ij} & \\
\text{s. t.} & & \\
& \sum_{j=1}^{k_i} x_{ij} \geq 1, & i = 1,\ldots,n. \\
& 0 \leq x_{ij} \leq u_{ij}, & i = 1,\ldots,n, \\
& & j = 1,\ldots,k_i, \\
& x_{ij} \in \mathbb{N} & \text{for all } i,j.
\end{array}
$$

In our approach, we start from the solution of the linear programming problem (LRP), and following the directions given by the test set of (LRP) we follow a descent path to the solution of the complete Problem (RP).

The process is greatly improved if we have a feasible point y^0 of (RP). There are lots of heuristic methods to obtain such a point. In our case, we use a greedy algorithm similar to [12] or [16].

At the beginning of the greedy algorithm, y^0 describes the system with the maximum number of components of every type. If the reliability of that system is less than R_0, then the problem has an empty feasible region. I will be the set of all pairs (i,j), which describes the j-th component for the i-th subsystem. For each (i,j), we calculate the rate $t_{ij} = \frac{c_{ij}}{-\log(1-r_{ij})}$ between cost and reliability, and sort I by decreasing rates (ties are solved by lex order, for example). For the first index (i_0, j_0) in I, we subtract components of type (i_0, j_0) from y^0 until it is not reliable, or there is no such component or the i_0-th subsystem is empty. If the solution obtained by this process is not reliable, or the i_0-th subsystem is empty, one (i_0, j_0) component is added. Then we take the next index in the set I and repeat the procedure, until the index set I has been completely processed.

Data: r_{ij}, vector $\boldsymbol{c}$
Result: $\boldsymbol{y}^0$ feasible point
$\boldsymbol{y}^0 = (u_{11}, \ldots, u_{nk_n})$;
$t = \left(\frac{c_{11}}{-\log(1-r_{11})}, \ldots, \frac{c_{nk_n}}{-\log(1-r_{nk_n})}\right)$;
$I = \{(1,1), \ldots, (1,k_1) \ldots, (n,k_n)\}$;
Sort I decreasingly by $t_{i,j}$;
forall the $(i,j) \in I$ **do**
  Reliable=TRUE ;
  SubsystemNonEmpty=TRUE ;
  while *Reliable* **and** *SubsystemNonEmpty* **and** $y^0_{i,j} > 0$ **do**
    $y^0_{i,j} = y^0_{i,j} - 1$;
    if $\sum_k y^0_{ik} < 1$ **then**
      SubsystemNonEmpty=FALSE ;
    end
    if $R(y^0) < R_0$ **then**
      Reliable=FALSE ;
    end
    if *Reliable=FALSE* **or** *SubsystemNonEmpty=FALSE* **then**
      $y^0_{ij} = y^0_{ij} + 1$;
    end
  end
end

Algorithm 1: Greedy algorithm

Using the above greedy algorithm, we obtain a feasible point y^0, with a cost $\sum_{ij} c_{ij} y^0_{ij} = c^0$. Obviously, the optimal solution of (RP) has a cost less than or equal to c^0.

4. A review on integer programming and Gröbner bases

In this section, we recall the concepts and algorithms used to solve integer linear programming problems from an algebraic point of view, and the walk back procedure for nonlinear integer programming problems based on test sets. To this end, we have followed [19] and [20].

4.1. **Gröbner bases.** Denote by $\mathbb{Q}[\boldsymbol{x}] = \mathbb{Q}[x_1, \ldots, x_N]$ the ring of polynomials with coefficients in $\mathbb{Q}$. The ideal generated by a subset $\mathcal{F} \subset \mathbb{Q}[\boldsymbol{x}]$ is the set $\langle \mathcal{F} \rangle$

consisting of all linear combinations:

$$\langle \mathcal{F} \rangle = \{h_1 f_1 + \cdots + h_r f_r \ : \ f_1, \ldots, f_r \in \mathcal{F}, h_1, \ldots, h_r \in \mathbb{Q}[\boldsymbol{x}]\}.$$

A term order on $\mathbb{N}^N$ is a total order $\prec$ satisfying the following properties:

- $\prec$ is compatible with sums, i.e., $\alpha \prec \beta \Rightarrow \alpha+\gamma \prec \beta+\gamma$, for all $\alpha, \beta, \gamma \in \mathbb{N}^N$.
- $\prec$ is a well-ordering, i.e., $0 \prec \alpha$ for all $\alpha \in \mathbb{N}^N$, $\alpha \neq 0$.

If we fix a term order $\prec$, then every nonzero polynomial f has a unique initial term $\text{in}_\prec(f) = a\boldsymbol{x}^\alpha$. It is the monomial $a\boldsymbol{x}^\alpha$ where α is the maximizing term appearing in f for the term order $\prec$. We are particularly interested in two term orders:

(1) The lexicographic order $<_{\text{lex}}$. For every $\alpha, \beta \in \mathbb{N}^N$, we say $\alpha >_{\text{lex}} \beta$ if, in the vector difference $\alpha - \beta \in \mathbb{Z}^N$, the first nonzero entry is positive.
(2) The vector induced order $<_{\boldsymbol{c}}$. Consider a vector $\boldsymbol{c} \in \mathbb{N}^N$. Given $\alpha, \beta \in \mathbb{N}^N$, we say $\alpha >_{\boldsymbol{c}} \beta$ if

$$\begin{cases} \boldsymbol{c}^t\alpha > \boldsymbol{c}^t\beta \\ \quad \text{or} \\ \boldsymbol{c}^t\alpha = \boldsymbol{c}^t\beta \quad \text{and } \alpha >_{\text{lex}} \beta. \end{cases}$$

For example, consider the polynomial $f = 6x_1x_2^2x_3 + 7x_3^2 - 5x_1^3 + 4x_1^2x_3^2$ and the vector $\boldsymbol{c} = (3,2,2)^t$. The monomials in f are represented by the vectors

$$\alpha_1 = (1,2,1)^t, \alpha_2 = (0,0,2)^t, \alpha_3 = (3,0,0)^t, \alpha_4 = (2,0,2)^t.$$

With respect to the lexicographic order $<_{\text{lex}}$ we have

$$\alpha_3 >_{\text{lex}} \alpha_4 >_{\text{lex}} \alpha_1 >_{\text{lex}} \alpha_2, \text{ so } \text{in}_{<_{\text{lex}}}(f) = -5x_1^3.$$

With respect to the induced order $<_{\boldsymbol{c}}$, we compute

$$\boldsymbol{c}^t\alpha_1 = 9, \boldsymbol{c}^t\alpha_2 = 4, \boldsymbol{c}^t\alpha_3 = 9, \boldsymbol{c}^t\alpha_4 = 10,$$

so

$$\alpha_4 >_{\boldsymbol{c}} \alpha_3 >_{\boldsymbol{c}} \alpha_1 >_{\boldsymbol{c}} \alpha_2, \text{ so } \text{in}_{<_{\boldsymbol{c}}}(f) = 4x_1^2x_3^2.$$

Of course, we can reorder the variables x_i, and get a new term order. In general, the notation $<_{\boldsymbol{c}}$ means a term order which is compatible with the partial order defined by the vector $\boldsymbol{c}$ and then a tie-break term order. If we change the lexicographic order in the definition on $<_{\boldsymbol{c}}$, we get another vector induced order.

Suppose that J is an ideal in $\mathbb{Q}[\boldsymbol{x}]$, and $\prec$ is a given term order. Then its initial ideal $\text{in}_\prec(J)$ is the ideal generated by the initial terms of the polynomials in J, that is, $\text{in}_\prec(J) = \langle \text{in}_\prec(f) \ : \ f \in J \rangle$. A finite subset $\mathcal{G}$ of J is a Gröbner basis with respect to the term order $\prec$ if the initial terms of the elements in $\mathcal{G}$ suffice to generate the initial ideal. In other words, $\text{in}_\prec(J) = \langle \text{in}_\prec(g) \ : \ g \in \mathcal{G} \rangle$.

Given an ideal and a term order, a Gröbner basis is not unique. The uniqueness is guaranteed by adding two more conditions. The reduced Gröbner basis of J with respect to $\prec$ is a Gröbner basis $\mathcal{G}_\prec$ of J such that:

- $\text{in}_\prec(g_i)$ has unit coefficient for each $g_i \in \mathcal{G}_\prec$.
- For each $g_i \in \mathcal{G}_\prec$, no monomial in g_i lies in $\langle \text{in}_\prec(\mathcal{G}_\prec \backslash \{g_i\}) \rangle$.

Every ideal J has a unique reduced Gröbner basis for each term order.

4.2. **Test set.** Consider an integer linear programming problem:

$$\begin{array}{lll} ILP(\boldsymbol{b}) & \min & \boldsymbol{c}^t \cdot \boldsymbol{x} \\ & \text{s. t.} & \\ & & A \cdot \boldsymbol{x} = \boldsymbol{b}, \\ & & \boldsymbol{x} \in \mathbb{N}^N, \end{array}$$

where $A \in \mathbb{Z}^{d\times N}, \boldsymbol{b} \in \mathbb{Z}^d, \boldsymbol{c} \in \mathbb{R}^N$. The notation $ILP(\boldsymbol{b})$ denotes the integer linear programming problem with right-hand-side fixed to $\boldsymbol{b}$. When we write (ILP), we note the set of all the integer linear programming problems obtained by varying the right-hand-side vector $\boldsymbol{b}$, fixing A and the cost function $\boldsymbol{c}$. Consider the map $\pi : \mathbb{N}^N \to \mathbb{Z}^d$ defined by $\pi(\boldsymbol{x}) = A\boldsymbol{x}$. Given a vector $\boldsymbol{b} \in \mathbb{Z}^d$, the set $\pi^{-1}(\boldsymbol{b}) = \{\boldsymbol{u} \in \mathbb{N}^N \ : \ \pi(\boldsymbol{u}) = \boldsymbol{b}\}$ is the fiber of (ILP) over $\boldsymbol{b}$.

We group points in $\mathbb{N}^N$ according to increasing cost value $\boldsymbol{c}^t\boldsymbol{x}$, and refine this order to a total order $<_{\boldsymbol{c}}$ breaking ties among points with the same cost value by adopting some term order (lexicographic, for example, as defined in the previous section). It is the vector induced order.

A set $G_{<_{\boldsymbol{c}}} \subset \mathbb{Z}^N$ is a test set for the family of integer linear problems (ILP) with respect to the matrix A and the order $<_{\boldsymbol{c}}$ if

- for each nonoptimal point α in each fiber of (ILP), there exists $g \in G_{<_{\boldsymbol{c}}}$ such that $\alpha - g$ is a feasible solution in the same fiber and $\alpha - g <_{\boldsymbol{c}} \alpha$,
- for the optimal point β in a fiber of (ILP), $\beta - g$ is not a feasible point for every $g \in G_{<_{\boldsymbol{c}}}$

A test set for (ILP) gives an obvious algorithm to solve an integer program, provided we know a feasible solution to this problem. At every step of this algorithm, we have two different cases:

- There exists an element in the test set which, when subtracted from the current point, yields an improved point. We are then in a nonoptimal point, but we get a better one.
- There will not exist such an element in the set, so we are in the optimum of the fiber.

4.3. **Toric ideal.** We define I_A the toric ideal associated with A as

$$I_A = \langle \boldsymbol{x}^\alpha - \boldsymbol{x}^\beta \ : \ A\alpha = A\beta, \alpha, \beta \in \mathbb{N}^N \rangle.$$

Given an integral vector $\gamma \in \mathbb{Z}^N$, we can write it uniquely as $\gamma = \gamma^+ - \gamma^-$, where $\gamma^+, \gamma^- \in \mathbb{N}^N$ and have disjoint supports. It is well known [19] that

$$I_A = \langle \boldsymbol{x}^{\alpha^+} - \boldsymbol{x}^{\alpha^-} \ : \ A\alpha = \boldsymbol{0}, \alpha \in \mathbb{Z}^N \rangle.$$

The relationship between the previous concepts is that the reduced Gröbner basis $\mathcal{G}_{<_{\boldsymbol{c}}}$ of I_A with respect to the order $<_{\boldsymbol{c}}$ allows us to compute a uniquely defined minimal test set $G_{<_{\boldsymbol{c}}}$ for (ILP). The reduced Gröbner basis is formed by binomials

$$\mathcal{G}_{<_{\boldsymbol{c}}} = \{\boldsymbol{x}^{\alpha_i} - \boldsymbol{x}^{\beta_i}, i = 1, 2, \ldots, r\}, \text{ with } \mathrm{in}_{<_{\boldsymbol{c}}}(\boldsymbol{x}^{\alpha_i} - \boldsymbol{x}^{\beta_i}) = \boldsymbol{x}^{\alpha_i},$$

and then the test set is expressed as

$$G_{<_{\boldsymbol{c}}} = \{\alpha_i - \beta_i, i = 1, 2, \ldots, r\}.$$

Data: Matrix A, vectors $\boldsymbol{b}, \boldsymbol{c}$, nonlinear restrictions
Result: Optimum
β = optimum for relaxed LRP ;
$\mathcal{P} = \{P(\beta)\}$;
$y^0 = \texttt{greedy}(RP)$;
$Y = \{y^0\}$;
$\mathcal{G}_{<_{\boldsymbol{c}}} = \texttt{groebner}(I_A)$ with respect to $<_{\boldsymbol{c}}$;
repeat
  forall the $P(\alpha) \in \mathcal{P}$ **do**
    forall the $g \in \mathcal{G}_{<_{\boldsymbol{c}}}$ **do**
      $w = \alpha + g$;
      if *w is a feasible point of (LRP)* **then**
        if *w is feasible for (RP)* **then**
          $Y = Y \cup \{w\}$;
          Prune $P(w)$;
        else
          if *$y <_{\boldsymbol{c}} w$ for some $y \in Y$* **then**
            Prune $P(w)$;
          end
          $\mathcal{P} = \mathcal{P} \cup \{P(w)\}$;
        end
      end
    end
  end
  Delete $P(\alpha)$ from $\mathcal{P}$;
until *all paths in $\mathcal{P}$ are pruned*;
Optimum = Select minimum $<_{\boldsymbol{c}}$ element from Y;

Algorithm 2: Walk back procedure

4.4. **Walk back procedure.** Basically the walk back procedure induces an algorithm which computes the optimum for a nonlinear integer programming problem under some conditions. The integer programming problem (RP) introduced in Section 2 is not linear. It has a nonlinear constraint (the reliability condition), while the rest of the restrictions are linear and the cost function is also linear. These are the conditions required to use the walk back procedure, introduced in [20].

In Algorithm 2, it is used the directed graph defined by the Gröbner basis over the feasible points, but directions are reversed in the skeleton. In each step, elements $w = \alpha + g$ in the reverse skeleton are computed, where $A\alpha = \boldsymbol{b}$ and g is an element in the Gröbner basis.

In general, Algorithm 2 uses the following notation. We denote by (RP) the entire nonlinear integer programming problem, (LRP) the relaxed linear integer programming problem which arises from (RP). Let β be the optimum of (LRP). If β is a feasible point for (RP), then it is the solution to (RP). If it is not a feasible point, then the reverse skeleton is needed.

Let $P(\alpha)$ denote the path, in the directed graph (reversed) of the linear integer programming problem (LRP), from the optimum β for (LRP) to a feasible point

α for (LRP). There is always one. Any solution of (RP) is feasible for (LRP), so the objective is to find such a path, in an ordered way. In each reversed step the cost function increases, so the minimum cost feasible points for (RP) are found first. Observe that the feasible point y^0 obtained in Section 3 is the first value for variable Y, that contains feasible points of the Problem (RP).

The cost of the point y^0 is an upper bound for the cost of any feasible point we will find in the procedure. If a feasible w point for Problem (RP) is found and its cost is greater than the cost for some $y \in Y$, then the branch of the point w is pruned.

5. The test set of the relaxed integer linear problem

In the relaxed integer linear problem (LRP), each inequality must be converted to an equality, so we must introduce a new slack variable for each inequality:

$$\begin{array}{lll} (LRP) \quad \min & \sum_{i=1}^{n}\sum_{j=1}^{k_i} c_{ij}x_{ij} & \\ \text{s.t.} & & \\ & \sum_{j=1}^{k_i} x_{ij} - d_i = 1, & i = 1,\ldots,n, \\ & x_{ij} + t_{ij} = u_{ij}, & i = 1,\ldots,n, \\ & & j = 1,\ldots,k_i, \\ & x_{ij}, d_i, t_{ij} \in \mathbb{N} & \text{for all } i,j. \end{array}$$

If we denote $N = k_1 + \ldots + k_n$ and

$$D_{n\times N} = \begin{pmatrix} \overbrace{1\ldots1}^{k_1} & \overbrace{0\ldots0}^{k_2} & \ldots & \overbrace{0\ldots0}^{k_n} \\ 0\ldots0 & 1\ldots1 & \ldots & 0\ldots0 \\ & & \ddots & \\ 0\ldots0 & 0\ldots0 & \ldots & 1\ldots1 \end{pmatrix},$$

the constraints in matrix form can be written as $A\boldsymbol{z} = \boldsymbol{b}$, where

$$A = \begin{pmatrix} D & -I_n & \mathbf{0}_{n\times N} \\ I_N & \mathbf{0}_{N\times n} & I_N \end{pmatrix}, \boldsymbol{b} = \begin{pmatrix} \mathbf{1}_n \\ \boldsymbol{u}_1 \\ \vdots \\ \boldsymbol{u}_n \end{pmatrix}, \boldsymbol{z} = \begin{pmatrix} \boldsymbol{x}_{N\times 1} \\ \boldsymbol{d}_{n\times 1} \\ \boldsymbol{t}_{N\times 1} \end{pmatrix}, \tag{2}$$

and

$$\boldsymbol{x} = \begin{pmatrix} x_{11} \\ \vdots \\ x_{1k_1} \\ \vdots \\ x_{n1} \\ \vdots \\ x_{nk_n} \end{pmatrix}, \boldsymbol{t} = \begin{pmatrix} t_{11} \\ \vdots \\ t_{1k_1} \\ \vdots \\ t_{n1} \\ \vdots \\ t_{nk_n} \end{pmatrix}, \boldsymbol{d} = \begin{pmatrix} d_1 \\ \vdots \\ d_n \end{pmatrix}, \boldsymbol{u}_i = \begin{pmatrix} u_{i1} \\ \vdots \\ u_{ik_i} \end{pmatrix}, i = 1,\ldots,n.$$

The vector $\mathbf{1}_n$ denotes the vector with all the n components equal to 1. We can assume that, for each $i = 1,\ldots,n$, the costs c_{ij} are sorted in descending order:

$c_{iq} \geq c_{ip}$ if $q < p$. We identify $\boldsymbol{z}^{\boldsymbol{y}}$ with

$$\boldsymbol{z}^{\boldsymbol{y}} = x_{11}^{X_{11}} \cdots x_{nk_n}^{X_{nk_n}} d_1^{D_1} \cdots d_n^{D_n} t_{11}^{T_{11}} \cdots t_{nk_n}^{T_{nk_n}}.$$

Consider the following set of binomials in $\mathbb{Q}[\boldsymbol{z}]$:

$$\mathcal{G} = \{\underline{x_{ik} d_i} - t_{ik}, \underline{x_{iq} t_{ip}} - x_{ip} t_{iq}\}, \tag{3}$$

for $i = 1, \ldots, n, k = 1, \ldots, k_i, 1 \leq q < p \leq k_i$. Let $>$ be a term order in $\mathbb{Q}[\boldsymbol{z}]$ such that $\boldsymbol{x} > \boldsymbol{d} > \boldsymbol{t}$. Within each block, the variables are lexicographically ordered as follows:

$$\begin{aligned} x_{11} > \cdots > x_{1k_1} > x_{21} > \cdots > x_{nk_n}, \\ d_1 > \cdots > d_n, \\ t_{11} > \cdots > t_{1k_1} > t_{21} > \cdots > t_{nk_n}. \end{aligned}$$

Theorem 1. *The set $\mathcal{G}$ is the reduced Gröbner basis of the toric ideal I_A with respect to the term order $>$. Moreover, $\mathcal{G}$ is the reduced Gröbner basis with respect to the order $<_{\boldsymbol{c}}$ induced by the cost vector* $\mathbf{c}$.

Proof. The proof follows similar steps and notation as that in [20, Thm. 4]. First of all, the set $\mathcal{G}$ is a subset of I_A, because all the binomials $\boldsymbol{z}^\alpha - \boldsymbol{z}^\beta$ in $\mathcal{G}$ verify $A\alpha = A\beta$.

The initial term of every binomial in $\mathcal{G}$ with respect to $>$ is the underlined term. It is enough to show that for every binomial $\boldsymbol{z}^\alpha - \boldsymbol{z}^\beta \in I_A$, with initial term $\boldsymbol{z}^\alpha$, there is some $g \in \mathcal{G}$ whose initial term divides $\boldsymbol{z}^\alpha$. By definition of toric ideal, $\boldsymbol{z}^\alpha - \boldsymbol{z}^\beta \in I_A$ if and only if $\alpha - \beta \in K = \{\boldsymbol{y} \in \mathbb{Z}^s \ : \ A\boldsymbol{y} = \boldsymbol{0}\}, s = n + 2N$. We denote an element $\boldsymbol{y}$ in K by $\boldsymbol{y} = (y_x, y_d, y_t)$ to indicate the correspondence between components of $\boldsymbol{y}$ and the columns of A. In addition, we denote the components of y_x by $(X_{11}, \ldots, X_{nk_n})$, and similarly for the others. We classify the elements in K in the following manner:

(1) Let $K_1 = \{\boldsymbol{y} \in K \ : \ y_x = 0\}$. Now $\boldsymbol{y} \in K_1$ if and only if $(y_d, y_t) \in \mathbb{Z}^{s_1}, s_1 = n + N$ belongs to the lattice $S' = \{\boldsymbol{w} \in \mathbb{Z}^{s_1} \ : \ A'\boldsymbol{w} = 0\}$ where

$$A' = \begin{pmatrix} -I_n & \\ & I_N \end{pmatrix}.$$

But $S' = 0$ since A' is a nonsingular matrix. Therefore $K_1 = 0$. This implies that there are no binomials of the form $\boldsymbol{z}^\alpha - \boldsymbol{z}^\beta$ that do not contain the variables x_{ij}.

(2) Let $K_2 = \{\boldsymbol{y} \in K \ : \ y_d = 0\}$. Again $\boldsymbol{y} \in K_2$ if and only if $(y_x, y_t) \in \mathbb{Z}^{s_2}, s_2 = 2N$ belongs to the lattice $S'' = \{\boldsymbol{w} \in \mathbb{Z}^{s_2} \ : \ A''\boldsymbol{w} = 0\}$, where

$$A'' = \begin{pmatrix} D & 0 & 0 \\ I_N & I_N & 0 \end{pmatrix}.$$

Let X_{iq} be the left most nonzero component of y_x. We may assume that $X_{iq} > 0$ since S is the set of integer points in a vector space which implies that it contains the negative of every element in it. The i-th row of matrix D in A'' implies that there exists some $p > q$ such that $X_{ip} < 0$. Therefore,

$$x_{iq} \text{ divides } \boldsymbol{z}^{\boldsymbol{y}^+} \text{ and } x_{ip} \text{ divides } \boldsymbol{z}^{\boldsymbol{y}^-}.$$

Consider now the rows given by the block $\begin{pmatrix} I_N & I_N & 0 \end{pmatrix}$ in A''. These rows imply that $T_{iq} = -X_{iq} < 0$ and $T_{ip} = -X_{ip} > 0$. Therefore,

$$x_{iq}t_{ip} \text{ divides } \boldsymbol{z}^{\boldsymbol{y}^+} \text{ and } x_{ip}t_{iq} \text{ divides } \boldsymbol{z}^{\boldsymbol{y}^-}.$$

The initial term of $\boldsymbol{z}^{\boldsymbol{y}^+} - \boldsymbol{z}^{\boldsymbol{y}^-}$ with respect to $>$ is $\boldsymbol{z}^{\boldsymbol{y}^+}$ since x_{iq} divides $\boldsymbol{z}^{\boldsymbol{y}^+}$ and x_{iq} is the greatest variable that appears in this binomial. But this implies that the initial term of $x_{iq}t_{ip} - x_{ip}t_{iq} \in \mathcal{G}$ divides the initial term of $\boldsymbol{z}^{\boldsymbol{y}^+} - \boldsymbol{z}^{\boldsymbol{y}^-}$. Therefore, the initial term of all binomials associated with K_2 is divisible by the initial term of an element in $\mathcal{G}$.

(3) Consider now a general element in $S = \{\boldsymbol{y} \in \mathbb{Z}^s \ : \ A\boldsymbol{y} = 0\}, s = n + 2N$, with no variables restricted to be zero. By the previous cases we may assume $y_x \neq 0, y_d \neq 0$. Let D_i be the first nonzero component of y_d. As before, we may assume that $D_i > 0$. Therefore,

$$d_i \text{ divides } \boldsymbol{z}^{\boldsymbol{y}^+}.$$

Then there exists $X_{ik} > 0$, so

$$x_{ik}d_i \text{ divides } \boldsymbol{z}^{\boldsymbol{y}^+}.$$

Similarly, there exists $T_{ik} < 0$ and

$$t_{ik} \text{ divides } \boldsymbol{z}^{\boldsymbol{y}^-},$$

so the initial term of $\boldsymbol{z}^{\boldsymbol{y}^+} - \boldsymbol{z}^{\boldsymbol{y}^-}$ is $\boldsymbol{z}^{\boldsymbol{y}^+}$ because $d_i > t_{ik}$. This initial term is divisible by $x_{ik}d_i$, which is the initial term of $x_{ik}d_i - t_{ik}$. Therefore

$$\text{in}_<(I_A) = \langle \text{in}_<(\mathcal{G}) \rangle,$$

which proves that $\mathcal{G}$ is a Gröbner basis of I_A with respect to the term order $<$. Clearly, it is a reduced basis.

Moreover, with respect to the term order $<_{\boldsymbol{c}}$,

$$\text{in}_{<_{\boldsymbol{c}}}(x_{ik}d_i - t_{ik}) = x_{ik}d_i,$$

because the weight of the first monomial is equal to $c_{ik} > 0$, and the weight of the second monomial is equal to zero. Similarly,

$$\text{in}_{<_{\boldsymbol{c}}}(x_{iq}t_{ip} - x_{ip}t_{iq}) = x_{iq}t_{ip}, 1 \leq q < p < k_i,$$

because the weight of the first monomial is $c_{iq} \geq c_{ip}$, which is the weight of the second monomial. If $c_{iq} = c_{ip}$, the tie is broken with the lexicographical order $x_{iq} > x_{ip}$. □

The above theorem gives a *reduced* Gröbner basis with respect to the term order induced by the objective function of (LRP). For comparison purposes, we mention that [20, Thm. 4] only provides a Gröbner basis with respect to a lexicographical order, and not with respect to the term order needed for the computation of the test set. Therefore, one more computational step is required to get it. Our construction gives directly the answer with its consequent saving.

Remark 1. *It is worth observing that the constraint matrix A in Equation (2) is totally unimodular. Therefore, the set of circuits $\mathcal{C}(A)$ is the universal Gröbner basis $\mathcal{U}(A)$ associated with the toric ideal I_A* [19, Prop. 4.11, Prop. 8.11]*. The set of circuits can be computed, as proposed in* [19, p. 35]*. Nevertheless, the elements that form our set $\mathcal{G}$ described in (3) must be extracted to construct the vectors that*

define the test set. Theorem 1 ensures that these two operations can be done directly in a single step, providing the test set in a straightforward way. Moreover, from our result we can state that the computational effort to get $\mathcal{G}$ has worst case complexity $O(nk^2)$, where $k = \max\{k_i \ : \ i = 1, \ldots, n\}$.

6. Computational results

The closed form construction of the test set for this particular problem is used in the computational experiments of our algorithm. In all the examples, the test set is computed in less than 0.1 seconds. A standard approach to compute the Gröbner basis using a program like 4ti2 ([4ti2 team(2008)]) is fast too, although Theorem 1 ensures that in our problem the computation does not depend on the implementation of Buchberger's algorithm.

The Algorithm 2 has been coded in Matlab and run on an Intel Xeon X5660 (2.8 GHz) with 32 GB RAM. For Table 1, all data in the test problems have been randomly generated from uniform distributions, with $l_{ij} = 0, u_{ij} = 4$ and $r_{ij} \in [0.99, 0.998]$, as in [16]. The linear cost function $\sum_{i=1}^{n}\sum_{j=1}^{k_i} c_{ij}x_{ij}$ has coefficients $c_{ij} \in [10, 20]$.

First of all, we compare in Table 1 our results (column Walk-Back) against the data from [16] (column Ruan-Sun). In the exact method described in [12], there appear only illustrative examples, and they do not consider different types of components within a subsystem.

We report the results in the following format. Columns n and k refer to the number of subsystems, and the number of different components in each subsystem, respectively.

Table 1. $R_0 = 0.90, r_{ij} \in [0.99, 0.998]$, 30 instances

		Walk-Back 2.8 GHz, 32 GB			Ruan-Sun 2 GHz, 256 MB	
n	k	Iter.	CPU	> 80000	Iter.	CPU
10	2	0	0.0	0	696	0.9
10	3	0	0.0	0	5797	13.5
10	5	0	0.0	0	26427	109.1
15	2	0	0.0	0	15184	34.5
15	3	0	0.1	0	85103	316.6
15	4	0	0.0	0	New	
17	3	84	0.1	0	New	
17	4	58	0.1	1	New	
19	2	3389	40.1	1	New	
20	2	7254	173.7	6	294747	1031.9

The remaining data are grouped in two blocks. The first one reports the results of our Walk-Back procedure, and the second one the data corresponding to the Ruan-Sun algorithm (extracted from [16, Table 1]). We include number of iterations and CPU time.

The average CPU time, and the average number of iterations (generated nodes) of the Walk-Back procedure has been obtained by running the program for 30 instances. The column "> 80000" contains the number of instances with more

than 80000 processed nodes. In that case, the process was terminated with a good (probably optimum) point, although no certificate of optimality is proven. The average CPU time is computed over the finished processes.

Since we have solved larger instances than those treated in [16], we label as "New" those cases not contained there. Comparing our results with [16], not only the CPU time is improved, but also the effort measured by the number of iterations by the Walk-Back procedure is less than the number of iterations in [16]. We also point out that the iterations in [16] compute two Lagrangian discrete relaxations and their corresponding solutions for the best value, each time. After that, the reliability of that solution is needed to discard remaining boxes in their branch and bound tree. In each iteration of the Walk-Back procedure, a node is easily computed by adding a vector, and its reliability is compared to R_0. Clearly, these iterations are less costly than theirs.

In [21] complete data for reliability and cost do not appear for all the examples, but only for the illustrative example of [21, Sect. 3].

In order to better illustrate the results, new tests have been done with data sets imposing the additional hypothesis that reliability and cost are correlated: the components with a greater reliability are the most expensive. Note that if there is no correlation between cost and reliability of a component (as in [16]), then it is likely that certain components are not going to be used, because only more reliable components would be chosen regardless their cost. Hence the dimensionality of the problem is artificially reduced. In order to consider these more realistic design problems we have also reported data generated with this correlation pattern, that appears in Table 2. This correlation is also considered in [21].

TABLE 2. $R_0 = 0.90, r_{ij} \in [0.99, 0.998]$ (**ordered**), 30 instances

n	k	Iter.	CPU (s)	> 80000
10	2	0	0.0	0
10	3	0	0.0	0
10	5	0	0.0	0
15	2	155.8	0.6	0
15	3	6965	322.8	1
15	4	17629	571.1	13
17	2	6465	92.7	1

It is clear the increasing computational effort shown by rows $n = 15, k = 3$ and $n = 15, k = 4$. Even if we increase the limit of iterations to 120000, there remain 9 instances in $n = 15, k = 4$ where we cannot ensure that we have reached the optimum. The average CPU time for the instances that were solved to optimality increases to 3442.1 seconds, and the average number of iterations to 34461. From the above, we conclude that the computational experiments for this model should be done with this additional hypothesis of correlation between cost and reliability for each component.

We also note that the algorithm is very sensitive to changes in the value of the reliability parameters r_{ij}. For example, for less reliable components, $r_{ij} \in [0.90, 0.99]$ and the same value, $R_0 = 0.90$, for the overall reliability, we have obtained the results in Table 3. The reader may observe that the values for n and k that can

TABLE 3. $R_0 = 0.90, \mathbf{r}_{ij} \in [\mathbf{0.90}, \mathbf{0.99}]$ (**ordered**), 30 instances

n	k	Iter.	CPU (s)	> 80000
6	4	11	0	0
6	5	49	0.1	0
7	4	1775	3.1	0
7	5	6157	23.6	0
8	4	32728	593.5	7

be solved are smaller. However, we have to point out that an exact solution has been found in almost all the instances. Although [21] does not give enough data to do comparisons, we observe that the best time for $n = 7, k = 5$ in [21, Table X] corresponding to a constant $L = 10^4$ is 105.29 CPU seconds, over 8 running tests. In Table 3 we report 23.6 CPU seconds, with a sample of 30 instances.

In order to perform further comparisons with standard integer nonlinear solvers as BARON ([18]) and COUENNE ([3]), the problem (RP) has been modeled in GAMS format. This encoding has allowed us to launch our data sets in the Neos Server ([11, 14, 13]) under these solvers. We have chosen the 30 instances for systems with $n = 8, k = 4$ from Table 3, because it contains a good assortment of cases. The results are reported in Table 4. Under column COUENNE, cells with $xxx(yyy)$ denote an error in the output of the program. In those data sets, COUENNE gets good bounds, namely yyy, but the returned value xxx is not the optimal value. In BARON, cells in the form $xxx(*)$ denote that the program has not reached the optimal value. The column "Minimum" in WB contains the certified minimum, obtained with the Walk-Back procedure by extending the search. Entries marked with "N/F" in the table indicate that the corresponding process has been stopped because a limit of time or nodes has been reached. We can make the following remarks:

- Comparison between Walk-Back and COUENNE.
  - WB is, in general, faster than COUENNE. In average, WB is 1.7 times faster than COUENNE.
  - WB does not certify optimality in 7 instances, and COUENNE does not do it in 9.
- Comparison between Walk-Back and BARON.
  - BARON achieves better CPU times than WB. In average, BARON is 9.1 times faster than WB.
  - BARON ends in all the cases. However there are 7 instances where BARON returns a nonoptimal point, although it is very close to the minimum cost (under 1% of optimality gap). Therefore, we conclude that termination, in general, does not mean certification for optimality.
  - One of the advantages of using Walk-Back is that it is an exact method, and operates with integer arithmetic. As pointed in [17, 9], available software packages to solve mixed integer programming problems use floating-point arithmetic that can lead to calculation errors in the solution of mathematical-programming relaxations and in the methods used for creating cutting planes to improve these relaxations. These issues result in a lack of certification of optimality for these software

TABLE 4. $R_0 = 0.90, r_{ij} \in [0.90, 0.99]$, (ordered), 30 instances

	Couenne		Baron		WB	
	CPU (s)	Minimum	CPU (s)	Minimum	CPU (s)	Minimum
1	81.7	113 (134)	13.7	113	23.7	113
2	1082	120	40.6	120	627.8	120
3	366.5	116	37.6	116	542.2	116
4	N/F		34.1	120(*)	522.4	119
5	133	113	18.1	113	34.9	113
6	1982.9	118	27.5	118	541.1	118
7	2927	121	72.4	122(*)	1915	121
8	N/F		143.3	133(*)	2274.7	132
9	N/F		101.7	125	N/F	125
10	313.7	125	25.9	125	197.9	125
11	3596.8	123	172.0	124(*)	N/F	123
12	N/F		155.4	113	N/F	113
13	N/F		105.3	123	1944.6	123
14	N/F		151.7	113	N/F	113
15	N/F		85.6	123	1108.7	123
16	1017.5	121	60.9	122(*)	920.4	121
17	649.6	122	20.9	122	216.9	122
18	247.8	113	29.7	113	88.9	113
19	68.6	114	9.7	114	20.6	114
20	N/F		144.2	131	N/F	131
21	139.7	123	15.6	123	38.2	123
22	N/F		126.5	134(*)	N/F	133
23	63.1	109	12.1	110(*)	14.8	109
24	4340.3	111	109.7	111	N/F	111
25	499.8	114 (754)	23.5	114	212.6	114
26	1419.2	117	67.2	117	1359.1	117
27	159.8	115	23.2	115	150.3	115
28	255.5	123	32.1	123	127.3	123
29	1289.6	111	51.3	111	605.3	111
30	805.5	124	38.8	124	162.2	124

packages. As opposite to that, Walk-Back certifies optimality upon termination because it is an exact method that operates in integer arithmetic.

An interesting observation is that the elapsed time in the Walk-Back procedure is significantly reduced if the algorithm is stopped with the first feasible point found in the process. Obviously, this approach does not guarantee optimality but it gives very accurate approximations. From this observation, we think that a promising open field is the combination of this technique with heuristic methods to get good approximated solutions.

7. Conclusion

We have presented in this paper an exact method for solving a nonlinear integer programming problem arising from the design of series-parallel reliability systems.

The method is based on the construction of a test set of an integer linear problem through the theory of Gröbner bases. We provide an explicit formula of the test set. Computational experiments show that this approach improves some existing methods in the literature already applied for this problem.

This paper deepens the challenge given in [20] to yield efficient algorithms for integer problems based on attractive bases.

Acknowledgments

This paper has been partially supported by Junta de Andalucía under grant FQM-5849, and Ministerio de Ciencia e Innovación MTM2010-19336, MTM2010-19576 and FEDER.

References

4ti2 team(2008). 4ti2 team, 2008. 4ti2—a software package for algebraic, geometric and combinatorial problems on linear spaces. Available at www.4ti2.de.

1. W.W. Adams, P. Loustaunau, *An introduction to Gröbner bases*, Graduate Studies in Mathematics, vol. 3, American Mathematical Society, Providence, RI, 1994.
2. F. Ahmadizar, H. Soltanpanah, Reliability optimization of a series system whith multiple-choice and budget constraints using an efficient ant colony approach, *Expert systems with Applications*, 38, (2011), 3640–3646.
3. P. Belloti, in `www.coin-or.org/Couenne/`.
4. D. Bertsimas, R. Weismantel, *Optimization over integers*, Dynamic ideas, 2005.
5. F. Castro, J. Gago, I. Hartillo, J. Puerto, J.M. Ucha, An algebraic approach to integer portfolio problems. *European J. Oper. Res.*, 210, (2011), 647–659.
6. M.S. Chern, On the computational-complexity of reliability redundancy allocation in a series system, *Oper. Res. Lett.*, 11, (1992), 309–315.
7. D.W. Coit, A.E. Smith, Reliability optimization of series-parallel systems using a genetic algorithm, *IEEE Trans. Reliab.*, 45, (1996) 254–260.
8. P. Conti, C. Traverso, Buchberger algorithm and integer programming, in Applied Algebra, Algebraic Algorithms and Error-Correcting Codes. *Lecture Notes in Comput. Sci.*, 539, (1991), 130–139., second ed.
9. W. Cook, T. Koch, D.E. Steffy, D.E. and K. Wolter, An exact rational mixed-integer programming solver, in IPCO2011: Integer Programming and Combinatorial Optimization, *Lecture Notes in Comput. Sci.* 6655, (2011), 104–116.
10. D.A. Cox, J. Little, D. O'Shea, *Using Algebraic Geometry*, Graduate Texts in Mathematics, vol. 185, Springer, New York, 2005.
11. J. Czyzyk, M.P. Mesnier, and J.J. Moré. The NEOS server. *IEEE Computational Science and Engineering*, 5(3):68–75, 1998.
12. M. Djerdjour, K. Rekab, A branch and bound algorithm for designing reliable systems at a minimum cost, *Appl. Math. Comput*, 118,(2001) 247–259.
13. E. Dolan. The NEOS server 4.0 administrative guide. Technical Report Technical Memorandum ANL/MCS-TM-250, Mathematics and Computer Science Division, Argonne National Laboratory, 2001.
14. W. Gropp and J.J. Moré. *Optimization environments and the NEOS server*, pages 167–182. Approximation theory and optimization (Cambridge, 1996). Cambridge Univ. Press, Cambridge, 1997.
15. M. Ouzineb, M. Nourelfath, M. Gendreau, Tabu search for the redundancy allocation problem of homogenous series-parallel multi-state systems, *Reliab. Eng. Syst. Saf.*, 93, (2008), 1257–1272.
16. N. Ruan, XL. Sun, An exact algorithm for cost minimization in series reliability systems with multiple component choices, *Appl. Math. Comput.*, 181, (2006) 732–741.
17. D.E. Steffy, K. Wolter, Valid linear programming bounds for exact mixed-integer programming, *INFORMS J. Comput.*, 25(2), (2013), 271–284.
18. N.V. Sahinidis, BARON: A general purpose global optimization software package, *Journal of Global Optimization*, 8(2), (1996), 201–205.

19. B. Sturmfels, *Gröbner Bases and Convex Polytopes*, University Lecture Series, vol. 8, American Mathematical Society, Providence, Rhode Island, 1996.
20. S.R. Tayur, R.R. Thomas, N.R. Natraj, An algebraic geometry algorithm for scheduling in presence of setups and correlated demands, *Math. Program.*, 69, (1995) 369–401.
21. A. Yalaoui, E. Chatelet, C.B. Chu, A new dynamic programming method for reliability & redundancy allocation in a parallel-series system, *IEEE Trans. Reliab.*, 54, (2005), 254–261.

Depto. de Álgebra, Universidad de Sevilla. Apdo. 1160, E-41080 Sevilla (Spain)
E-mail address: gago@us.es

Dpto. de Matemática Aplicada I, E.T.S. de Ingeniería Informática, Av. Reina Mercedes, s/n, 41012 Sevilla, Spain
E-mail address: hartillo@us.es

Dpto. de Estadística e I.O., Facultad de Matemáticas, apdo. 1160, 41080 Sevilla, Spain
E-mail address: puerto@us.es

Depto. de Álgebra, Universidad de Sevilla. Apdo. 1160, E-41080 Sevilla (Spain)
E-mail address: ucha@us.es