Finding Multiple Solutions in Nonlinear Integer Programming with Algebraic Test-Sets

M. I. Hartillo[1], J. M. Jiménez-Cobano[2], and J. M. Ucha[1,2]

[1] Departamento de Matemática Aplicada I., Universidad de Sevilla, Sevilla, Spain
{hartillo,ucha}@us.es
[2] Insituto de Matemáticas de la Universidad de Sevilla Antonio de Castro Brzezicki, Universidad de Sevilla, Sevilla, Spain
josjimcob@alum.us.es

Abstract. We explain how to compute all the solutions of a nonlinear integer problem using the algebraic test-sets associated to a suitable linear subproblem. These test-sets are obtained using Gröbner bases. The main advantage of this method, compared to other available alternatives, is its exactness within a quite good efficiency.

1 Introduction

In many real-life combinatorial optimization problems it is of great interest for the decision-maker to have not only one solution, but the set of *all* optimal solutions (see [15] or [11], for example). The information provided by this set can give some unexpected insights about the features of the solutions, and sometimes stands as a first step for multi-objective optimization as well.

On the other hand, sometimes these problems require nonlinear constraints to be modeled properly. In [14] a method for problems of the form

$$\begin{aligned} &\min\ cx^t \\ &\text{s.t. } Ax^t \leq b^t \\ &\quad x \in \Omega \\ &\quad x \in \mathbb{N}^n \end{aligned} \tag{1}$$

where $A \in \mathbb{Z}^{m\times n}, c \in \mathbb{Z}^n, b \in \mathbb{Z}^m$ (operator t stands for transposition) and the region Ω is finite and defined by linear and nonlinear constraints, was proposed. This method makes use of the so called *test-sets* associated to the linear subproblem

$$\begin{aligned} &\min\ cx^t \\ &\text{s.t. } Ax^t \leq b^t \end{aligned} \tag{2}$$

Definition 1. *A set $T \subset \mathbb{Z}^n$ is a test-set associated to problem 2 if $T \subset \ker(A)$, and for any non optimal x feasible for 2 there exists a $t \in T$ such that $x - t$ is feasible and $c(x - t) < c(x)$.*

As a consequence of this definition, starting from an optimal point $\hat{x}$ of problem 2 you can *recover* the set of all the feasible points, adding elements of the test-set until you eventually complete all the feasible region. In this way, you can obtain the optimal points of problem 1 *walking back* from the linear optimal point until you reach the region Ω. Technical details can be found in [14]. The feasible region will be supposed finite, although this is not strictly necessary.

There are several ways of computing test-sets (as a matter of fact, they can be computed depending only on the cost and the matrix of constraints, not taking into account the right hand side, for example). One of the most efficient and manageable ways is using Gröbner bases (see for example [7]) with the software `4ti2` (see [9]). This approach is based in the classical paper [4], that shows how to solve a Linear Integer problem obtaining Gröbner bases of a suitable *binomial* ideal with respect to an ordering compatible with the cost function.

In [3,10] the method of [14] is applied to real-life size problems with very competitive results. In this work, we (1) explain how to modify the walk-back method to obtain all the optimal points, and (2) compare its performance with the natural generalization of the algorithm presented in [15] and with the commercial software BARON (see [1]).

Remark 1. The ideals corresponding to the method that we present in this work are not zero-dimensional, so some efficient strategies as Triangular Decomposition can not be applied for the Gröbner bases computations. In principle, the method proposed in [2] would be an alternative to treat some Nonlinear Integer Optimization problems with zero-dimensional ideals, but as soon as some constraints can not be expressed in terms of polynomials or the rank of values of the variables is big the method is useless.

2 Finding All the Optimal Points with a General Integer Cut

In [15] a method is introduced to show how to compute all the optimal points in Integer Linear Problems. Once an optimal point is obtained, the idea is to add some conditions to make it unfeasible and solve again. More precisely, if you are solving the problem 1 and have obtained an optimal solution $(x_1^0, \ldots, x_n^0)$ you can add the constraint

$$\sum_{i=1}^{n} |x_i - x_i^0| \geq 1$$

to assure that $(x_1^0, \ldots, x_n^0)$ is unfeasible for this new problem. As you obtain new optimal solutions you have to add a similar constraint for each solution. This formulation can be linearized, as it is explained in [15, Prop. 1]. This method

can be used for Nonlinear Integer problems simply considering problems of the form

$$\begin{aligned} &\min cx^t \\ &\text{s.t. } Ax^t \leq b^t \\ &\quad \textstyle\sum_{i=1}^{n} |x_i - x_i^j| \geq 1, j = 1, \ldots, N \\ &\quad x \in \Omega \\ &\quad x \in \mathbb{N}^n \end{aligned}$$

with N constraints to try to obtain the $(N+1)$-th optimal point. One of the aims of this paper is to compare this natural approach to an alternative algebraic method.

3 Finding All the Optimal Points with Test-Sets

Our algorithm is based on the algebraic algorithm of [14] that provides one solution for a given nonlinear integer problem of the form

$$\begin{aligned} &\min cx^t \\ &\text{s.t. } Ax^t \leq b^t \\ &\quad x \in \Omega \\ &\quad x \in \mathbb{N}^n \end{aligned} \quad (P)$$

where $A \in \mathbb{Z}^{m \times n}$, $c \in \mathbb{Z}^n$, and $b \in \mathbb{Z}^m$. Let us describe the steps of our method:

- We start, as in the algorithm proposed in [14], in the optimal point for a suitable linear subproblem

$$\begin{aligned} &\min cx^t \\ &\text{s.t. } Ax^t \leq b^t \\ &\quad x \in \mathbb{N}^n \end{aligned} \quad (P_L)$$

Remark 2. The selection of the subproblem has to do first with the computability of the test-set, that can be a bottleneck as it is a computation of a Gröbner basis of a certain ideal (see [7]). Moreover, although the test-set of the whole linear part of problem 1 is available, sometimes it is better to compute the test-set associated to a submatrix of A that gives us a more manageable number of directions to be considered at any point during the walk-back. The constraints that are not included in the submatrix are simply added to the description of Ω.

- Then we systematically add the elements of the corresponding test-set, thus worsening the cost function trying to obtain in return feasible points for problem 1, until we get into Ω.
- The difference to the original method (that was designed to find *only one optimal point*) is that now we have to manage the searching of new possible optimal points inside Ω, once we have reached a candidate. While in the algorithm of [14] you discard new points with the same optimal value, we instead stock them in a provisional list. This list will be the set of optimal points as long as we find a new better value inside the region Ω.

– If we eventually find an improvement in the cost we delete the provisional list. Otherwise, we already have the set of optimal points.

The pseudocode of our algorithm is the following one:

INPUT: $c, A, b; \Omega$; optimal point β of 2; T associated test-set of problem 2.

$Opt := \emptyset$;
$Leaves := \{\beta + t | \forall t \in T\} \cap \mathbb{N}^n$
$costOpt = \infty$

IF $\beta \in \Omega$
THEN $Opt := \{\beta\}$;
$costOpt := c\beta^t$

WHILE ($Leaves \neq \emptyset$) DO
FOR $h \in Leaves$ DO
IF $c(h) < costOpt$
$Leaves = (Leaves \setminus \{h\}) \cup (\{h + t | \forall t \in T\} \cap \mathbb{N}^n)$
IF $h \in \Omega$
THEN $Opt = \{h\}$;
$costOpt = ch^t$;
$Leaves = (Leaves \setminus \{h\}) \cup (\{h + t | \forall t \in T\} \cap \mathbb{N}^n)$

♯ the list of old candidates is deleted
♯ and updated with a new candidate

ELSE IF $c(h) > costOpt$
THEN $Leaves = Leaves \setminus \{h\}$
♯ these branches are discarded

ELSE IF $c(h) = costOpt$
THEN $Leaves = (Leave \setminus \{h\}) \cup (\{h + t | \forall t \in T\} \cap \mathbb{N}^n)$
IF $h \in \Omega$
THEN $Opt = Opt \cup \{h\}$;
♯ a new candidate to be an optimal point has been obtained

END WHILE

OUTPUT: Opt the set of all optimal points of problem 1 with cost $costOpt$

Remark 3. It is straightforward to modify this algorithm to obtain the K best optimal points for a given K, as in [11].

4 Computational Experiments

We have run all the examples to test our algorithm coded in Python in a computer with an Intel Core i5, 3.5 Ghz, 8 Gb of RAM, under Ubuntu. The examples with BARON [1] and COUENNE [6] have been sent to `neos-server.org`.

We have studied two families of examples: the integer portfolio problem and the problem of reliability in series-parallel systems.

4.1 Integer Portfolio Problem

In the integer portfolio problem (see [3]) we have to solve problems of the form

$$\begin{array}{ll} \max & \sum_{i=1}^{n} c_i x_i \\ s.t. & xCx^t \leq R_0 \\ & \sum_{i=1}^{n} b_i x_i \leq B \\ & x \in \mathbb{N}^n \end{array} \tag{3}$$

where b_i are the prices today of n alternative investments or assets; c_i a forecast for their future prices; the matrix C has to do with the covariance matrix of the historical returns of the assets and it is a way of measuring the risk of a portfolio; R_0 stands for the maximum admissible risk; B is the available budget.

This so called *mean-variance portfolio model* was introduced by Markowitz with continuous variables (see [12] for the model and [5] or [16] for the mixed-integer case), but it is interesting to consider the case of integer variables: first to take into account the finite divisibility of the assets and second to consider some logical conditions that appear in these problems.

If you consider tailored examples for which two or more variables have the same price and risk you obtain many different optima and can compare our method to the generalization of [15]. As a general outcome we have obtained that as the number of optimal points increases our method overcomes by far the general cut method (coded in GAMS for COUENNE). Thereby, you can consider for example the simple case

$$\begin{array}{ll} \max & 2x_1 + x_2 + x_3 + \cdots + x_{n-1} + x_n \\ s.t. & x_1 + x_2 + x_3 + \cdots + x_{n-1} + x_n \leq B \\ & \left(x_1 \cdots x_n\right) C \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \leq R_0 \\ & x \in \mathbb{N}^n \end{array} \tag{4}$$

with C defined by $c_{ii} = 0.05, c_{ij} = -0.01$ if $i \neq j$ for $n = 10, 15, 20, 50, 100$, $B = 10$ and a not very tight R_0 to include many points. In this family of examples you can obtain thousands of optimal points. In average our method is more than a hundred times faster for big numbers of optimal points.

Remark 4. Comparing with the commercial software BARON, that has the option of computing the K best optimal solutions, we obtain better running times only in 15% of the cases. Nevertheless, our method obtains exactly the complete set of optimal points in all cases. BARON, in contrast to our method, fails in 11% of the cases: in 5% of the cases it does not obtain the optimal cost and in 6% does not find the complete set, due to rounding problems.

4.2 Reliability Problems

The *redundancy allocation problem* can be formulated as the minimization of the design cost of 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 (see [8,13] or [10]). It has the form

$$\begin{aligned} \min\ & \sum_{i=0}^{n} \sum_{j=1}^{k} c_{ij} x_{ij} \\ s.a.\ & R(x) \geq R_0 \\ & \sum_{j=1}^{k} x_{ij} \geq 1,\ \forall i = 1, \ldots, n \\ & 0 \leq x_{ij} \leq u_{ij},\ \forall i = 1, \ldots, n,\ j = 1, \ldots, k \\ & x_{ij} \in \mathbb{Z}^{+} \end{aligned} \tag{5}$$

with $R(x) = \prod_{i=1}^{n} \left(1 - \prod_{j=1}^{k} (1 - r_{ij})^{x_{ij}}\right)$. In this problem n is the number of subsystems (in series); k_i the number of different types of available components (in parallel) for the i- th subsystem, $1 \leq i \leq n$; r_{ij} the reliability of the j-th component for the i-th subsystem, $1 \leq i \leq n, 1 \leq j \leq k_i$; c_{ij}, the cost of the j-th component for the i-th subsystem, $1 \leq i \leq n, 1 \leq j \leq k_i$; l_{ij}, u_{ij}, lower/upper bounds of number of j components for the i-th subsystem, $1 \leq i \leq n, 1 \leq j \leq k_i$; R_0, an admissible level of reliability of the whole system; x_{ij}, number of j components used in the i-th subsystem, $1 \leq i \leq n, 1 \leq j \leq k_i$.

We have studied about 100 examples with 2, 3 and 4 subsystems and with 2 or 3 components in each subsystem (the costs and reliabilities generated randomly, $r_{ij} \in [0.90, 0.99]$ and $R_0 = 0.90$). The summary is in Tables 1, 2 and 3 and contains only the information about the examples with multiple number of solutions.

Table 1. Reliability examples $n = 3$, $k = 2, 3$.

	Average CPU Time	% Complete set of optimal solutions found
Test-set	0.2	100 %
BARON K-best	0.2	100 %
General cut	0.54	100 %

Table 2. Reliability examples $n = 4$, $k = 2$.

	Average CPU Time	% Complete set of optimal solution found
Test-set	0.46	100 %
BARON K-best	0.21	72.72 %
General cut	0.98	100 %

Table 3. Reliability examples $n = 4$, $k = 3$.

	Average CPU Time	% Complete set of optimal solution found
Test-set	1.1	100 %
BARON K-best	0.19	61.54 %
General cut	1.29	100 %

We can observe that:

- The test-set method and the general cut method are exact. BARON, on the contrary, does not give the complete set of optimal points or even one (in fact, usually provides only one although you ask for the best 3 or 4 best) in about 30 % of the cases.
- BARON is better in CPU time than the test-set method, and this is better than the general cut method, and much better as the number of optimal points increases substantially. If, for example, you treat an example with a hundred optimal points you have to solve 99 problems with 1, 2, ..., 99 new constraints, respectively.

5 Conclusions

We have presented an exact method to obtain the set of all optimal points for a given Nonlinear Integer problem as problem 1. A convenient linear subproblem is selected and then, walking back from an optimal point of this linear subproblem with the help of a test-set, the feasible region of the original problem is reached in all different ways, updating a list of optimal points. In this work we have studied two families of examples:

- Portfolio integer problems that can produce a huge number of optimal points and for which our algorithm overcomes a general cut approach as the one proposed in [15].
- Reliability problems, in which we point out problems of lack of exactness in the commercial software BARON compared with our approach.

References

1. Sahinidis, N. V.: BARON 14.4.0: Global Optimization of Mixed-Integer Nonlinear Programs. User's Manual (2014)
2. Bertsimas, D., Perakis, G., Tayur, S.: A new algebraic geometry algorithm for integer programming. Manag. Sci. **46**(7), 999–1008 (2000)
3. Castro, F.J., Gago, J., Hartillo, I., Puerto, J., Ucha, J.M.: An algebraic approach to integer portfolio problems. Eur. J. Oper. Res. **210**(3), 647–659 (2011)
4. Conti, P., Traverso, C.: Buchberger algorithm and integer programming. In: Mattson, H.F., Mora, T., Rao, T.R.N. (eds.) AAECC 1991. LNCS, vol. 539, pp. 130–139. Springer, Heidelberg (1991). https://doi.org/10.1007/3-540-54522-0_102

5. Corazza, M., Favaretto, D.: On the existence of solutions to the quadratic mixed-integer mean-variance portfolio selection problem. Eur. J. Oper. Res. **176**, 1947–1960 (2007)
6. Belotti, P.: COUENNE: a user's manual. Department of Mathematics Sciences, Clemson University. https://www.coin-or.org/Couenne/
7. Cox, D.A., Little, J., O'Shea, D.: Using Algebraic Geometry. Graduate Texts in Mathematics, vol. 185, 2nd edn. Springer, Hiedelberg (2005). https://doi.org/10.1007/978-1-4757-6911-1
8. Djerdjour, M., Rekab, K.: A branch and bound algorithm for designing reliable systems at a minimum cost. Appl. Math. Comput. **118**, 247–59 (2001)
9. 4ti2 team: 4ti2 - a software package for algebraic, geometric and combinatorial problems on linear spaces (2013). https://www.4ti2.de
10. Gago, J., Hartillo, I., Puerto, J., Ucha, J.M.: Exact cost minimization of a series-parallel reliable system with multiple component choices using an algebraic method. Comput. Oper. Res. **40**(11), 2752–2759 (2013)
11. Leão, A.A.S., Cherri, L.H., Arenales, M.N.: Determining the K-best solutions of knapsack problems. Comput. Oper. Res. **49**, 71–82 (2014)
12. Markowitz, H.: Portfolio selection. J. Finance **7**, 77–91 (1952)
13. Ruan, N., Sun, X.L.: An exact algorithm for cost minimization in series reliability systems with multiple component choices. Appl. Math. Comput. **181**, 732–41 (2006)
14. Tayur, S.R., Thomas, R.R., Natraj, N.R.: An algebraic geometry algorithm for scheduling in presence of setups and correlated demands. Math. Program. Ser. A **69**(3), 369–401 (1995). https://doi.org/10.1007/BF01585566
15. Tsai, J.-F., Lin, M.-H., Hu, Y.-C.: Finding multiple solutions to general integer linear programs. Eur. J. Oper. Res. **184**(2), 802–809 (2008)
16. Li, H.-L., Tsai, J.-F.: A distributed computation algorithm for solving portfolio problems with integer variables. Eur. J. Oper. Res. **186**(2), 882–891 (2008)