Annals of Operations Research 117, 71–93, 2002

Cutting and Surrogate Constraint Analysis for Improved Multidimensional Knapsack Solutions

MARÍA A. OSORIO aosorio@cs.buap.mx
School of Computer Sciences, Autonomous University of Puebla, Puebla 72560, México

FRED GLOVER
Hearin Center for Enterprise Science, School of Business, University of Mississippi, MS 38677, USA

PETER HAMMER
Rutgers Center for Operations Research, Rutgers University, Piscataway, NJ 08854-8003, USA

Abstract. We use surrogate analysis and constraint pairing in multidimensional knapsack problems to fix some variables to zero and to separate the rest into two groups – those that tend to be zero and those that tend to be one, in an optimal integer solution. Using an initial feasible integer solution, we generate logic cuts based on our analysis before solving the problem with branch and bound. Computational testing, including the set of problems in the OR-library and our own set of difficult problems, shows our approach helps to solve difficult problems in a reasonable amount of time and, in most cases, with a fewer number of nodes in the search tree than leading commercial software.

Keywords: multidimensional knapsack problem, surrogate constraints, duality, constraint pairing, logic cuts

1. Introduction

In this paper, we present an integrated cutting and surrogate constraint analysis strategy for the NP-hard, multidimensional knapsack problem (MKP), which can be formulated as:

$$\text{Maximize } z = \sum_{j \in N} c_j x_j \tag{1}$$

$$\text{subject to} \quad \sum_{j \in N} a_{ij} x_i \leqslant b_i, \ i \in M, \tag{2}$$

$$x_j \in \{0, 1\}, \ j \in N, \tag{3}$$

where $N = \{1, 2, \ldots, n)$, $M = \{1, 2, \ldots, m\}$, $c_j \geqslant 0$, for all $j \in N$, $a_{ij} \geqslant 0$, for all $i \in M$, $j \in N$. Each of the m constraints of (2) is called a knapsack constraint.

The multidimensional knapsack problem has received wide attention from the operations research community, because it embraces many practical problems. Applications include resource allocation in distributed systems, capital budgeting and cutting

stock problems [16,18,52]. In addition, the MKP can be seen as a general model for any kind of binary problems with positive coefficients (see [35,38]).

Most of the research on knapsack problems deals with the much simpler single constraint version ($m = 1$). For the single constraint case the problem is not strongly NP-hard and effective approximation algorithms have been developed for obtaining near-optimal solutions. A good review of the single knapsack problem and its associated exact and heuristic algorithms is given by [44]. Important recent advances are found in [43,48].

The development of exact algorithms for the MKP began several decades ago [2,9, 19]. There exists a main stream of algorithms that try to find upper or lower bounds for the objective value, to reduce the problem size and use information from its relaxations, and to employ search trees in branch and bound schemes. The algorithm presented in this paper belongs in this mainstream.

The Constraint Pairing ideas used here were developed by Hammer, Padberg and Peled [27] and used later by Dembo and Hammer [10], as a support of a reduction algorithm for knapsack problems that uses constraint pairing in a Lagrangian relaxation framework. Glover established the main principles of his surrogate constraint duality theory in the same year [22], stimulating a series of algorithmic developments that used surrogate constraint analysis as an alternative to relying on weaker relaxations provided by Lagrangian relaxation theory. Recently, Glover, Sherali and Lee [24] generated cuts from surrogate constraint analysis for zero–one and multiple choice programming. Those cuts proved to be effective and stronger than a variety of those previously introduced in the literature.

Other kinds of exact algorithms include a dynamic programming based method [18], an enumerative algorithm based on the Fourier–Motzkin elimination [5], and an iterative scheme using linear programs to generate subproblems solved with implicit enumeration [53].

One of the first algorithms that used branch and bound to solve the MKP was published by Shih [52]. He obtained an upper bound for the problem by computing the objective function value associated with an optimal fractional solution obtained related to every single constraint knapsack problem separately. He found an upper bound for the MKP using the minimum of the objective values for these separate problems and tested his algorithm for random instances with sizes up to five knapsacks and ninety variables. He obtained better results than the zero–one additive algorithm developed by Balas [2].

Gavish and Pirkul [16] developed another branch and bound algorithm using the Lagrangian, the surrogate and the composite relaxation of the problem. They evaluated the quality of the bounds generated by these different relaxations. They also developed new algorithms for obtaining surrogate bounds and for reducing the size of the problem. For randomly generated instances containing up to 5 constraints and 200 variables, their algorithm was proved to be faster than the Shih algorithm by at least one order of magnitude. Their algorithm was also found to be faster than the Sciconic commercial software and was used as a heuristic, terminating before the tree search was finished. In this way, it achieved better results than the heuristic developed by Loulou and Michaelides [42].

Crama and Mazzola [7] showed that although the bounds derived from the well-known relaxations, such as Lagrangian, surrogate, or composite, are stronger than the bounds obtained from the linear programming (LP) relaxation, the improvement obtained with these relaxations is limited. In particular, they showed that the improvement in the quality of the bounds using any of these relaxations cannot exceed the magnitude of the largest coefficient in the objective function.

The first statistical analysis for this problem was conducted by Fontanari [13], who investigated the dependence of the objective function on the knapsack capacities and on the number of capacity constraints, in the case when all objects were assigned the same profit value and the knapsack coefficients were uniformly distributed over the unit interval. He obtained a rigorous upper bound on the optimal profit by employing annealed approximation and compared the results with the exact value obtained through a Lagrangian relaxation method.

An asymptotic analysis was presented by Schilling [50] who computed the asymptotic (n – with m fixed) objective function value for problems where the objective and knapsack coefficients were uniformly (and independently) distributed over the unit interval and where the RHS in the knapsack was set to the value 1. Szkatula [54] generalized that analysis to the case where the RHS in the constraints were not restricted to be one (see also Szkatula [55]).

Finally, heuristic algorithms developed for the MKP can be roughly grouped into the early heuristic approaches, bound based heuristics (which make use of an upper bound on the optimal solution to the MKP), tabu search heuristics, genetic algorithm heuristics, and analyzed heuristics (with some theory relating to worst-case or probabilistic performance). Hanafi and Fréville [28] developed an efficient tabu search approach for the 0–1 multidimensional knapsack problem. Chu and Beasley [6] presented an extensive analysis of the problem, using a genetic algorithm and testing it in a data set with 270 problems, now in the OR-library, that included instances with 5, 10 and 30 knapsacks, 100, 250 and 500 variables and an RHS generated as 0.25, 0.5 and 0.75 of the coefficient sums in every knapsack. We used this data set in one of our computational experiments.

2. Some results on surrogate constraint duality

Duality theory in mathematical programming is not new. For years, it has customarily been based upon the use of a generalized Lagrangian function to define the dual. Elegant results have emerged linking optimality conditions for the dual to those for the primal. Out of these results have arisen solution strategies for the primal that exploit the properties of the primal dual interrelations. Some of these strategies have been remarkably successful, particularly for problems in which the duality gap – the amount by which optimal objective function values for the two problems differ – is nonexistent or small. A different type of solution strategy has been proposed for solving mathematical programs in which duality gaps are likely to be large.

In contrast to the Lagrangian strategy, which absorbs a set of constraints into the objective function, a different strategy proposed by Glover [22] replaces the original constraints by a new one called a surrogate constraint. Since their introduction by Glover [21], surrogate constraints have been proposed by a variety of authors for use in solving nonconvex problems, especially those of integer programming. Surrogate constraints that were 'strongest' for 0–1 integer programming under certain relaxed assumptions were suggested by Balas [3] and Geoffrion [17]. The paper by Geoffrion also contained a computational study that demonstrated the practical usefulness of such proposals. Later, Dyer [12] provided a major treatment of surrogate duality. Methods for generating strongest surrogate constraints according to other definitions, in particular, segregating side conditions and introducing normalizations, were subsequently proposed by Glover [20]. However, a significant price was paid by the relaxations used in some of these early developments, whose effect was to replace the original nonconvex surrogate IP problem by a linear programming problem. The structure of this LP problem is sufficiently simple that the distinction between the surrogate constraint approach and the Lagrangian approach vanished.

The first proposal for surrogate constraints [21] used notions that were later used in surrogate duality theory. It defined a strongest surrogate the same way as in this theory and presented a theorem that led to a procedure for searching optimal surrogate multipliers that could obtain stronger surrogate constraints for a variety of problems. Later, Greenberg and Pierskalla [26] provided the first major treatment of surrogate constraints in the context of general mathematical programming. These authors showed that the dual surrogate is quasiconcave, thus assuring that any local maximum for it is a global maximum, and noted that the surrogate approach has a smaller duality gap than the Lagrangian approach. They provided sufficient conditions for the nonoccurrence of surrogate duality gaps.

The work of Glover [22] developed a surrogate duality theory that provides exact conditions under which surrogate duality gaps cannot occur. These conditions (both necessary and sufficient) are less confining that those governing the absence of Lagrangian duality gaps. Furthermore, they give a precise characterization of the difference between surrogate and Lagrangian relaxation, and give a framework for combining these relaxations. Useful relationships for combining these relaxations are also developed in [36].

2.1. Definitions

The primal problem of mathematical programming can be written:

$$\text{P:} \quad \min_{x \in X} f(x) \quad \text{subject to } g(x) \leqslant 0,$$

where f and each component $g_i(x)$ of the vector $g(x)$ are real-valued functions defined on X. No special characteristics of these functions or of X will be assumed unless otherwise specified.

A surrogate constraint for P is a linear combination of the component constraints of $g(x) \leqslant 0$ that associates a multiplier u_i with each $g_i(x)$ to produce the inequality $ug(x) \leqslant 0$, where $u = (u_1, \ldots, u_m)$. Clearly, this inequality is implied by $g(x) \leqslant 0$ whenever $u \geqslant 0$. Correspondingly, we define the surrogate problem as:

$$\begin{aligned} \text{SP}(u): \quad & \min_{x \in X} f(x) \\ & \text{subject to } ug(x) \leqslant 0. \end{aligned}$$

The optimal objective function value for SP(u) will be denoted by $s(u)$, or more precisely, as:

$$s(u) = \inf_{x \in X(u)} f(x), \quad \text{where } X(u) = \{x \in X\colon ug(x) \leqslant 0\}.$$

Since SP(u) is a relaxation of P (for u nonnegative), $s(u)$ cannot exceed the optimal objective function value for P and approaches this value more closely as $ug(x) \leqslant 0$. Choices for the vector u that improve the proximity of SP(u) to P – i.e., that provide the greatest values of $s(u)$ – yield strongest surrogate constraints in a natural sense, and motivate the definition of the surrogate dual:

$$\text{SD:} \quad \max_{u \geqslant 0} s(u).$$

The surrogate dual may be compared with the Lagrangian dual LD: $\max_{u \geqslant 0} L(u)$, where $L(u)$ is the function given by $L(u) = \inf_{x \in X}\{f(x) + ug(x)\}$. It should be noted that $s(u)$ is defined relative to the set $X(u)$, which is more restrictive than the set X relative to which the Lagrangian $L(u)$ is defined. Also, modifying the definition of $L(u)$ by replacing X with $X(u)$, while possibly increasing $L(u)$, will nevertheless result in $L(u) \leqslant s(u)$ because of the restriction $ug(x) \leqslant 0$; that is, $L(u)$ may be regarded as an 'underestimating' function for both the surrogate and primal problems.

Another immediate observation is that any optimal solution to the surrogate problem that is feasible for the primal is automatically optimal for the primal and no complementary slackness conditions are required, in contrast to the case for the Lagrangian. These notions have been embodied in the applications of surrogate constraints since they were first proposed. Taken together, they provide what may be called a 'first duality theorem' for surrogate mathematical programming.

2.2. First duality theorem for surrogate mathematical programming

Optimality conditions for surrogate duality are the requirements that the surrogate multiplier vector u is non-negative, x is optimal for the surrogate problem, and x is feasible for the primal problem. 'Strong' optimality conditions add the requirement of complementary slackness $ug(x) = 0$. The surrogate optimality conditions are:

(i) $u \geqslant 0$,

(ii) x is optimal for SP(u),

(iii) $g(x) \leqslant 0$.

Theorem [22]. The surrogate optimality conditions imply x is optimal for the primal, u is optimal for the surrogate dual, and their optimal objective function values are equal.

Using the basic equalities, we can also go a step farther and define the strong optimality conditions of complementary slackness:

(iv) $ug(x) = 0$.

The strong surrogate optimality conditions are necessary and sufficient for optimality, and they can be used in a variety of new inferences for surrogate constraints in mathematical programming. The methodology proposed here draws on these fundamental results.

3. Constraint pairing

The main ideas about constraint pairing in integer programming were exposed by Hammer, Padberg and Peled [27]. Based on the objective of getting bounds for most variables, the strategy is to pair constraints in the original problem to produce bounds for some variables.

It is well known that a major parameter in characterizing the computational complexity of an integer programming problem is the number n of integer valued variables. This is the reason why most algorithms involve as a first step an attempt to fix values or at least find sharp bounds on the integer variables of the problem. However, in most cases this attempt is performed on individual constraints examined one at time. It is apparent on the other hand that the examination of the full constraint system will usually lead to much better conclusions. The computational effort required by such an approach, however, can be excessive. To strike a compromise between the two extremes – examining the full constraint system on one hand, and examining individual constraints on the other – one is naturally led to the idea of examining pairs of constraints.

Obviously, the selection of the 'pair' to be examined will play an important role in this procedure. Based on the results exposed in the last section, the dual surrogate constraint provides a useful relaxation of the constraint set, and can be paired with the objective function. In order to get the dual surrogate values, we will use the bounds $x_j \leqslant 1$ as part of the LP relaxation of the problem. The resulting surrogate will have the following properties:

$$\sum_{j\in N}\left(v_j + \sum_{i\in M} u_i a_{ij}\right)x_j \leqslant \sum_{i\in M} u_i b_i + \sum_{j\in N} v_j. \tag{4}$$

For the resulting surrogate, $\sum_{i\in M} u_i b_i + \sum_{j\in N} v_j =$ continuous LP relaxed solution, our current upper bound (UB).

Another interesting property of (4) is that the coefficient values for the x's whose relaxed values in the LP problem are strictly greater than zero, will be the same as in the

objective function, i.e.:

$$\sum_{j\in N}\left(v_j + \sum_{i\in M} u_i a_{ij}\right) = c_j. \tag{5}$$

Now, we pair the system,

$$\sum_{j\in N} c_j x_j \geqslant LB, \tag{6}$$

$$\sum_{j\in N}\left(v_j + \sum_{i\in M} u_i a_{ij}\right) x_j \leqslant UB. \tag{7}$$

Using the property in (5) for eliminating all terms whose x values in the relaxed LP solution are strictly greater than zero, we get a new constraint with fewer variables in the left side and with a right-hand side equal to the gap, $UB - LB$.

Defining S as a set that contains the indexes of x variables whose values in the relaxed LP solution are equal to zero, and making $\{v_j + \sum_{i\in M} u_i a_{ij}\} = s_j$, the resulting combined constraint can be expressed as

$$\sum_{j\in N}(s_j - c_j)x_j \leqslant UB - LB. \tag{8}$$

A straightforward way to use this constraint is to strengthen the bounds on the components of x. Obviously, if a $s_j - c_j$ is greater than the value $UB - LB$, the corresponding x_j must be zero in the integer solution. Because we depend on the gap $UB - LB$ and UB cannot be changed because it is the LP-continuous relaxed solution of the problem, a better LB given by the best integer solution known, can increase the number of integer variables fixed to zero. The inequality (8) will not only be used to fix x variables to zero, but also will be used as a knapsack with positive coefficients to get effective logic cuts, as we describe in the following section.

4. Logic and nested cuts

A logic cut for an MIP model has been characterized as an implication of the constraint set. Actually any logical formula implied by the constraint set as a whole is a logic cut, and a logic cut is true if it satisfies the constraints even if it is not implied by the constraints. Logic cuts can be defined in an even more general sense that allows them to be nonvalid. A cut may be added to the problem without changing the optimal solution, but it may exclude feasible solutions (see [31]). An intuitive understanding of a problem can suggest logic cuts, both valid and nonvalid, even when no further polyhedral cuts are easily identified. The idea of a (*possibly nonvalid*) logic cut was defined by Hooker et al. [34], who used process synthesis as an example. Other examples include structural design problems in [4], and a series of standard 0–1 problems discussed by Wilson [58].

Whereas a cut in the traditional sense is an inequality, a logic cut can take the form of any restriction on the possible values of the integer variables, whether or not expressed as an inequality. Logic cuts can therefore be used to prune a search tree even when they are not expressed as inequality constraints in an MIP model. But they can also be imposed as inequalities within an MIP model, in which case they can tighten the linear relaxation and cut off fractional solutions as traditional cuts do.

Knapsack constraints are often used to generate logic cuts, in the form of extended inequalities, which can be easily manipulated. The logical clauses implied by a knapsack constraint are identical to the well-known "covering inequalities" for the constraint, and their derivation is straightforward (see [25]). Examples with logic cuts generated from knapsack constraints include location problems in [46], and multilevel generalized assignment problems in [45].

The MIP solver in CPLEX provides effective and efficient uses of cover inequalities to facilitate the solution of MIP problems. The cover inequalities are derived by looking at each all-binary inequality with non-unit coefficients. For a constraint with all non-negative coefficients, a minimal cover is a subset of the variables in the inequality such that if all variables were set to 1, the inequality would be violated, but if any one variable were excluded, the inequality would be satisfied. Cover inequalities take the form where the sum of the variables in the cover must not exceed the size of the cover less one. If a cover inequality is violated by the solution to the current subproblem, it is added to the problem.

While it is hard to derive all the extended inequalities implied by a knapsack constraint, it is easy to derive all *contiguous cuts*. Consider a 0–1 inequality $dy \geqslant \delta$ for which it is assumed, without loss of generality, that $d_1 \geqslant d_2 \geqslant \cdots \geqslant d_n > 0$. Note that if $d_j < 0$, its sign is reversed and d_j is added to δ. A contiguous cut for $dy \geqslant \delta$ has the form,

$$\sum_{j=t}^{t+w+k-1} y_j \geqslant k, \tag{9}$$

where k is the degree of the cut and $w < n$ is the "weakness" (with $w = 0$ indicating a cut that fixes all of its variables). In particular (9) is a t-cut because the first term is y_t and it is valid if and only if

$$\sum_{j=1}^{t+k-1} d_j + \sum_{t+w+k}^{n} d_j < \delta. \tag{10}$$

Furthermore, Hooker and Osorio [33] showed that every t-cut of weakness w for $dy \geqslant \delta$ is implied by a 1-cut of weakness w. Therefore, generating 1-cuts can be equivalent to generate all t-cuts in terms of infering values for the binary variables. Figure l shows the algorithm that generates the 1-cuts we used for this problem. These cuts are generated in linear time. To illustrate, consider the knapsack constraint

$$13y_1 + 9y_2 + 8y_3 + 6y_4 + 5y_5 + 3y_6 \geqslant 30.$$

```
Let k = 1, s = Σ_{j=1}^{n} d_j, k_last = 0.
For j = 1, ..., n:
  {Let s = s − d_j.
   If (s < δ) then
     {While (s + d_k < δ)
        {Let s = s + d_k
         Let k = k + 1
        }
      If (k > k_last) then
        {Generate the cuty y_1 + ··· + y_j ⩾ k.
         Let k_last = k
        }
     }
  }
```

Figure 1. An algorithm for generating all 1-cuts for a knapsack constraint $dy \geqslant \delta$ in which $d_1 \geqslant d_2 \geqslant \cdots \geqslant d_n > 0$.

This knapsack constraint gives rise to the following cuts,

$$y_1 + y_2 \geqslant 1,$$
$$y_1 + y_2 + y_3 \geqslant 2,$$
$$y_1 + y_2 + y_3 + y_4 + y_5 \geqslant 3.$$

Note that the first cut becomes redundant, once the second one is formulated.

The preceding cut inequalities are a special case of nested inequalities, where two inequalities overlap in their unit coefficients only if the nonzero coefficients of one are contained in the other. Algorithms for efficiently generating strongest nested inequalities in general, simultaneously bounded from above as well as below, are presented for surrogate constraints having both positive and negative coefficients in Glover [21]. We restrict attention here to the simpler types of nested inequalities where each is strictly "contained in" the next member of the progression.

4.1. Nested logic cuts from pairing

For our procedure, we do not generate the nested cuts directly from the knapsack constraint system in (2), because these cuts are too numerous and the number of nonzero coefficients in every cut is so large that the cut is not very useful in the optimization procedure. Instead, we generate the cuts using the modified paired constraint defined in (8) and the objective function used as a constraint in (6).

The inequality (8) has information from the dual surrogate constraint and from the objective equation. It also has fewer variables with nonzero coefficients than a customary logic cut, because we set $x_j = 0$ for all s_j values greater than the value of $UB - LB$. In addition, $s_j = 0$ for all variables x_j whose values in the LP relaxed problem are strictly

greater than zero. Now, we have only one knapsack, with a reduced number of variables, to generate cuts.

We will split the variables in (6), sending to the right-hand side the terms whose x's are presented in the last cut generated from (8). Now, using the worst case, we get the following inequality,

$$\sum_{j \in N'} c_j x_j \geqslant LB - \max\left(\sum_{j \in N''} c_j x_j \right),$$

where N'' is a set that contains the indexes of the variables present in the last cut generated from (6) and N' is a set with the indexes not present in that cut. The cuts generated from (8) impose a limit on the value of $\max(\sum_{j \in N''} c_j x_j)$. We take that limit and make $z' = LB - \max(\sum_{j \in N''} c_j x_j)$, to yield

$$\sum_{j \in N'} c_j x_j \geqslant z'. \tag{11}$$

Now, we use (11) for generating nested logic cuts of type "greater equal". Again, the inequality in (11) has fewer variables than the original in (6). The cuts generated from this knapsack will likewise have fewer variables and will be more effective.

To test the effectiveness of these cuts in our experiments we use the number of nodes in the search tree because there is no reasonable theory of "tightness" for cutting planes (other than to check whether a cut is supporting or facet defining). Usually, the measure of the effectiveness of covering cuts and logic cuts can be obtained by comparing the size of the search tree (number of nodes) with and without the cuts. The search tree size is relatively an intrinsic measure [33].

5. Example

We illustrate the procedure in the following example. Table 1 shows the coefficients for a multidimensional knapsack problem with 15 variables and 4 knapsack constraints. Referring to the formulation presented in (1)–(3), the c_j are the objective function coefficients and the a_{ij} are the knapsack coefficients.

After solving the relaxed LP problem, the corresponding dual variable values are:

$$u_i = \{0.66,\ 0.52,\ 0.62,\ 2.79,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 23.13,\ 22.44,\ 0,\ 0\}.$$

This problem has a continuous LP solution value of 335.6 (UB), and we already know an integer feasible solution with a value of 301 (LB). In table 2, we show the coefficient values for the surrogate constraint s_j, defined in (7), the objective function c_j, defined in (6), and the resulting paired constraint $s_j - c_j$, defined in (8). In order to explain the properties found in the paired constraint, we also show the x_j values for the continuous LP relaxation and for the integer problems.

A useful property of the paired constraint, is the fact that a zero coefficient results for any x_j whose value in the relaxed LP problem is strictly greater than zero. In con-

Table 1
Data for the multidimensional knapsack example.

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
c_j	36	83	59	71	43	67	23	52	93	25	67	89	60	47	64	RHS
a_{1j}	7	19	30	22	30	44	11	21	35	14	29	18	3	36	42	87
a_{2j}	3	5	7	35	24	31	25	37	35	25	40	21	7	17	22	75
a_{3j}	20	33	17	45	12	21	20	2	7	17	21	11	11	9	21	65
a_{4j}	15	17	9	11	5	5	12	21	17	10	5	13	9	7	13	55

Table 2
Surrogate and paired constraints.

j	1	2	3	4	5	6	7	8	9
s_j	60.4	83	59	91.5	53.7	72.1	66.2	92.9	93
c_j	36	83	59	71	43	67	23	52	93
$s_j - c_j$	24.4	0	0	20.5	10.7	5.1	43.2	40.9	0
x_j	0	0.72	0.49	0	0	0	0	0	0.9
x_j	0	0	1	0	0	0	0	0	1

j	10	11	12	13	14	15	RHS	
s_j	60.7	67	89	60	57.6	88.4	335.6	*UB*
c_j	25	67	89	60	47	64	301	*LB*
$s_j - c_j$	35.7	0	0	0	10.6	24.4	34.6	*UB* − *LB*
x_j	0	0.22	1	1	0	0	Cont. LP solution	
x_j	0	0	1	1	0	0	Integer solution	

sequence, the paired constraint will contain nonzero coefficients only for variables that are zero in the continuous LP relaxed problem and most of these variables will tend to be zero in the integer solution.

The paired constraint, as defined in (8), is

$$24.4x_1+20.5x_4+10.7x_5+5.1x_6+43.2x_7+40.9x_8+35.7x_{10}+10.6x_{14}+24.4x_{15} \leqslant 34.6.$$

In order to be able to satisfy this paired constraint, variables x_7, x_8 and x_{10} must be zero because their coefficients are greater than the right-hand side value of 34.6 in this inequality. Fixing those x variables to zero and sorting the constraint coefficients in descending order, we have

$$x_7 = 0, \quad x_8 = 0, \quad x_{10} = 0, \quad \text{and}$$
$$24.4x_1 + 24.4x_{15} + 20.5x_4 + 10.7x_5 + 10.6x_{14} + 5.1x_6 \leqslant 34.6.$$

Applying the procedure described in figure 1, we get the following nested not redundant logic cuts:

$$x_1 + x_{15} + x_4 \leqslant 1,$$
$$x_1 + x_{15} + x_4 + x_5 + x_{14} \leqslant 2.$$

Now, from the objective function constraint (6), we have

$$36x_1 + 83x_2 + 59x_3 + 71x_4 + 43x_5 + 67x_6 + 23x_7 + 52x_8 + 93x_9 + 25x_{10} + 67x_{11} + 89x_{12} + 60x_{13} + 47x_{14} + 64x_{15} \geqslant 301.$$

Clearing terms and sending to the right-hand side the variables included in the preceding logic cut, we obtain

$$83x_2 + 59x_3 + 67x_6 + 23x_7 + 52x_8 + 93x_9 + 25x_{10} + 67x_{11} + 89x_{12} + 60x_{13} \geqslant 301 - \{36x_1 + 71x_4 + 43x_5 + 47x_{14} + 64x_{15}\}.$$

In the worst case, with the coefficients sorted in descending order and eliminating the terms with a x value of zero, we have

$$93x_9 + 89x_{12} + 83x_2 + 67x_{11} + 67x_6 + 60x_{13} + 59x_3 \geqslant 301 - \text{Max}\{71x_4 + 64x_{15} + 47x_{14} + 43x_5 + 36x_1\}.$$

Using $x_1 + x_{15} + x_4 + x_5 + x_{14} \leqslant 2$, this implies $93x_9 + 89x_{12} + 83x_2 + 67x_{11} + 67x_6 + 60x_{13} + 59x_3 \geqslant 166$.

This last inequality, mostly contains variables not included in the paired constraint, i.e., variables that will tend to equal 1 in the integer solution. Applying the procedure shown in figure 1, we get the following no redundant logic cut,

$$x_9 + x_{12} + x_2 + x_{11} + x_6 + x_{13} \geqslant 2.$$

This last cut is of the "greater equal" type in contrast to the cuts obtained from the paired constraint, and contains variables not included in the other logic cuts. The addition of these two types of cuts to the original problem proves to be a powerful tool to improve the CPU time required to solve MKPs to optimality using a classical branch and bound procedure.

We do not have any way to know the quality of an integer feasible solution in advance. Yet the cuts obtained from the paired constraint and from the objective function strongly depend in the integer feasible solution used to generate them. Such a solution may be obtained from a data set or from another paper in the topic. Another source is to apply a heuristic procedure or to run a commercial code for a fixed number of nodes and to take the best solution generated.

6. Experimental results

We tested our approach in three experiments. To test the number of variables fixed by pairing the dual surrogate with the objective function, we used the set of 270 large problems developed by Chu and Beasley [6] and available in the OR-library. In order to test the efficiency of the logic cuts generated, we present two experiments. For the first one, we used real-world problems already published in the literature and available in the OR-library, and for the second one, we combined different ideas published in the literature to build our own generator of hard MKPs.

6.1. *Large problems in OR-library and number of variables fixed*

For this experiment, we used the set with 270 large MKP instances used by Chu and Beasley [6] and made publicly available in the OR-library, with WWW access at http://mscmga.ms ic.ac.uk/.

This data set was generated using the procedure suggested by Fréville and Plateau [15]. The number of constraints m was set to 5, 10 and 30, and the number of variables n was set to 100, 250 and 500. Thirty problems were generated for each $m - n$ combination, giving a total of 270 problems.

The a_{ij} are integer numbers drawn from the discrete uniform generator $U(0, 1000)$. For each $m - n$ combination, the right-hand side coefficients (b_i's) are set using the relation, $b_i = \alpha \sum_{j \in N} a_{ij}$, where α is a tightness ratio and $\alpha = 0.25$ for the first ten problems, $\alpha = 0.50$ for the next ten problems and $\alpha = 0.75$ for the remaining ten problems. The objective function coefficients (c_j's) were correlated to a_{ij} and generated as follows: $c_j = \sum_{i \in M} a_{ij}/m + 500 d_j \; j \in N$, here d_j is a real number drawn form the continuous uniform generator $U(0, 1)$. In general, correlated problems are more difficult to solve than uncorrelated problems [16,47].

We used these instances to test the number of variables fixed with our procedure, pairing the surrogate obtained with the dual values and the objective function. Table 3 shows the value for the ratio m/n, the average number of variables fixed and cuts added for every combination of m, n and α. It also shows the correlation between the coefficients in the objective function and the coefficients in the knapsacks for every subset. There is a very high correlation coefficient between the ratio m/n and the number of variables fixed for every α in this data set. We also found high correlation coefficients between the number of variables (m) and the average number of cuts generated for every m and n combination. There is a high negative correlation coefficient between the number of knapsacks in the problems (n) and the average correlation coefficient in the data for each m and n, and between the number of cuts generated and the tightness value in the knapsacks (α).

We solved these problems using CPLEX V6.5.2 installed on Windows NT in a Pentium III with 450 MHz and 64 MB RAM, without modifying its default parameters, both with and without the addition of our procedure. We allowed a maximum solution time of 3 hours (10,800 secs) and a memory tree size of 250 MB. We compared the objective value obtained, the number of problems solved to optimality with a gap of 0.0001 and the number of problems finished when the time or the tree size memory were exceeded in both methodologies. We also compared the gap value between the best solution value found and the continuous LP relaxation value for the best node remaining in the tree, i.e., (best LP value of remaining nodes − best integer value)/(1.0 + best LP value of remaining nodes), getting averages values every 10 instances with the same m, n and α.

The first three columns in table 4 indicate the sizes (n and m) and the tightness ratio (α) of a particular problem structure, with each problem structure containing 10 problem instances. The next three columns report the average objective value obtained with

Table 3
Variables fixed, cuts and correlation factors for the OR-library instances.

m	n	Relation	Variables fixed			Correl.	Cuts added			Average	Correlation in data			Average
		m/n	$\alpha = 0.25$	$\alpha = 0.5$	$\alpha = 0.75$	α& vars	$\alpha = 0.25$	$\alpha = 0.5$	$\alpha = 0.75$	cuts	$\alpha = 0.25$	$\alpha = 0.5$	$\alpha = 0.75$	correl.
100	5	20	22.6	11	0.6	−1	11.6	12.8	12.9	12.433	0.28	0.3	0.3	0.2922
100	10	10	6.3	0.4	0	−0.893	12.6	14.4	13.8	13.6	0.18	0.17	0.17	0.1719
100	30	3.3333	0	0	0	N/A	13.6	15.7	13.6	14.3	0.06	0.06	0.07	0.0651
250	5	50	107	51.1	6.7	−0.998	16.4	19.1	18.3	17.933	0.31	0.3	0.3	0.3037
250	10	25	57.3	20.4	0	−1	18.9	21.3	21.9	20.7	0.16	0.15	0.16	0.1557
250	30	8.3333	0.1	0	0	N/A	22.8	27.3	25.7	25.267	0.06	0.07	0.06	0.0644
500	5	100	246	140	35.6	−1	23.4	24	24.5	23.967	0.29	0.29	0.31	0.2974
500	10	50	99	24.9	0	−0.961	26.8	31.3	30.1	29.4	0.17	0.18	0.17	0.1698
500	30	16.667	0	0	0	N/A	34.7	37.4	40.3	37.467	0.06	0.06	0.06	0.0629
Correlation (m/n & vars fixed)			0.99	0.96	0.89	(m & cuts)	0.92	0.9	0.89	0.9044	Corr (n & CorrFactor)			−0.9166

Table 4
Computational results for OR-library problems.

Vars	Constraints		Objective value				GAP		Proved optimal[c]		Time finished[b]		Memory finished[a]	
n	m	α	OR-library	Fix + Cuts	CPLEX	Imp	Fix + Cuts	CPLEX	Fix + Cut	CPLEX	Fix + Cut	CPLEX	Fix + Cut	CPLEX
100	5	0.25	24197	24197	24197.2	−0	0.0001	0.0001	10	10	0	0	0	0
100	5	0.5	43253	43253	43252.9	0.1	0.0001	0.0001	10	10	0	0	0	0
100	5	0.75	60471	60471	60471	0	0.0001	0.0001	10	10	0	0	0	0
100	10	0.25	22602	22602	22601.9	0.1	0.0001	0.0001	10	10	0	0	0	0
100	10	0.5	42659	42661	42660.6	0.4	0.0001	0.0001	10	10	0	0	0	0
100	10	0.75	59556	59556	59555.6	0.4	0.0001	0.0001	10	10	0	0	0	0
100	30	0.25	21654	21656	21660.2	−4	0.0042	0.0039	2	2	8	8	0	0
100	30	0.5	41431	41437	41440.4	−3	0.0019	0.0018	1	1	9	9	0	0
100	30	0.75	59199	59202	59201.8	0.2	0.0002	0.0002	8	8	2	2	0	0
250	5	0.25	60411	60413	60413.5	−1	0.0002	0.0003	6	4	4	6	0	0
250	5	0.5	109285	109293	109293	0	0.0001	0.0001	7	5	3	5	0	0
250	5	0.75	151556	151560	151560	0	0.0001	0.0001	10	10	0	0	0	0
250	10	0.25	58994	59019	59018	1	0.0021	0.0024	0	0	10	0	0	10
250	10	0.5	108582	108607	108592	15	0.0011	0.0012	0	0	9	0	1	10
250	10	0.75	151330	151363	151344	19	0.0005	0.0006	0	0	9	5	1	5
250	30	0.25	56876	56959	56887.7	71	0.0066	0.0074	0	0	3	0	7	10
250	30	0.5	106629	106686	106679	7	0.00317	0.00325	0	0	3	0	7	10
250	30	0.75	150444	150467	150479	−12	0.0015	0.0014	0	0	10	10	0	0
500	5	0.25	120616	120610	120619	−9	0.0005	0.0004	0	0	9	3	1	7
500	5	0.5	219503	219504	219506	−2	0.0002	0.0002	0	0	10	2	0	8
500	5	0.75	302355	302361	302358	3	0.0001	0.0001	6	5	4	0	0	5
500	10	0.25	118566	118584	118597	−13	0.0002	0.0002	0	0	0	0	10	10
500	10	0.5	217275	217297	217290	7	0.0007	0.0008	0	0	2	0	8	10
500	10	0.75	302556	302562	302573	−11	0.0005	0.0005	0	0	2	0	8	10
500	30	0.25	115470	115520	115497	23	0.0048	0.0051	0	0	10	0	0	10
500	30	0.5	216187	216180	216151	29	0.0021	0.0023	0	0	10	0	0	10
500	30	0.75	302353	302373	302366	7	0.0013	0.0013	0	0	10	0	0	10
	Total		3244134	3244514	3244389	125			100	95	127	50	43	125
	Average		120153	120167	120163	4.6	0.0012	0.00126						

[a] Memory finished = 250 MB treesize memory. [b] Time finished = 10800 s of solution time. [c] Proved optimal, gap < 0.0001.

each methodology, the genetic algorithm procedure proposed by Chu and Beasley [6], CPLEX with our fixed variables and cuts and CPLEX by itself. In column 7 we present the difference in the average objective value between CPLEX with our cuts and CPLEX alone. Columns 8 and 9 present the average gap value at the moment CPLEX stopped the execution for both methodologies. The next 6 columns show the number of instances finished in each possible way: proved optimality with a gap less or equal to 0.0001, time finished or tree size memory finished.

The average objective value for the genetic algorithm is 120,153.1, for CPLEX, 120,162.5 and for our procedure, 120,167.2. We have a total improvement of 125.2 in the sum of objective values and an average improvement of 4.637 over the average CPLEX values. The improvement in the gap values is also significant, with 0.0012 for our methodology and 0.00126 for CPLEX. The gap value is the most accurate way to know how close the solution found will be to the real optimal value of the problem. In addition, our procedure was able to solve 100 problems to optimality, while CPLEX alone could solve only 95. It is interesting to notice that in the rest of the problems, CPLEX usually terminated because the tree size memory was exceeded. Our procedure kept the tree size memory within the limits for a larger number of instances and finished because the time limit was reached.

6.2. Real problems in OR-library

The second experiment for testing the influence of these cuts was made in real-world problems consisting of $m = 2$ to 30 and $n = 6$ to 105 included in the OR-library and whose optimal solutions are known. Many of these problems have been used by other authors [1,6,8,11,23,30,37,40,41,56].

We solved these problems on a SUN Ultrix 10, Solaris 7 and 250 MB RAM, using the general-purpose CPLEX 7.1 mixed integer programming (MIP) solver, including the options for adding cuts and fixing variables (Cuts + Fix). The first five columns indicate the problem set name, the authors who first published them, the number of problems in the set, the average number of variables and the average number of constraints in each set. Columns 6 and 7 show the average number of cuts generated and variables fixed to zero, using our procedure, for each set.

All problems from the test set were solved to optimality in less than 3 seconds of CPU time and the difference by CPLEX in order to solve these simple problems with and without our procedure was practically nil. Instead, the effect of the cuts for these problem instances can be better appreciated by considering the average number of nodes needed to solve each set. The relative difference in the number of nodes needed by CPLEX was on average 41.31%, favoring the version that was augmented by our method.

6.3. Problem generation and results for difficult problems

Problem generation methodologies are currently a very important topic of research. A main part of applying new techniques consists of empirically testing algorithm per-

Table 5
Computational results for small problems.

File	First published in	No. problems	Average variables	Average constraints	Av. cuts	Vars. fixed	Average nodes	
							Cuts + Fix	Original
Petersen	Petersen (Balas) [3]	7	24	8.57	5.7	3	7.29	15
SENTO	Senju and Toyoda [51]	2	60	30	8.5	7	273.5	308.5
WEING	Weingartner and Ness [57]	7	47.25	2	6.3	15	3.5	6.375
WEISH	Shi [52]	30	60	5	6.1	28	10	12.47
PB	Fréville and Plateau [14]	6	31.17	13.33	6.2	1	153.67	185.17
HP	Fréville and Plateau [14]	2	31.5	4	6.5	0	27	27.5

formance across a representative range of problems. An inadequate set of test problems can provide misleading information about algorithm performance.

MKP problem generation has been intensively studied in the last decade. Using the gap as a measure of hardness, Pirkul [47] concludes that for a given number of constraints and a given degree of difficulty, the gap reduces as the number of variables increases. For a given number of variables and a given degree of difficulty, the gap increases as the number of constraints increases. Finally, holding the number of variables and constraints constant, the gap increases as the degree of difficulty increases (in particular, as the constraints become tighter).

Fréville and Plateau [15] suggested a procedure for generating hard MKP problems. This procedure was adopted by Chu and Beasley [6] in the generation of the MKP dataset in the OR-library used for the second experiment in this paper. The approach algorithm uses independently uniform random generation with an interval of (0,1000) for the coefficients in the knapsacks. It also averages the constraint coefficients of each variable to generate the coefficient in the objective function, getting, in this way, a positive correlation between the constraint coefficients and the objective. This way of generating the knapsacks provides independence and a wide range in the coefficients, characteristics that contribute to the hardness of these problems (see [49]). The positive correlation between the objective and the knapsack coefficients, even if highly influenced by the number of knapsacks in the problem, contributes to the problems hardness (see table 5 and [29]).

Hill and Reilly [29] presented an empirical study of the effects of coefficient correlation structure and constraint slackness settings on the performance of solution procedures. This work examined synthetic two-dimensional knapsack problems (2KP), and conducted tests on representative branch-and-bound and heuristic procedures. Population correlation structure, and in particular the interconstraint component of the correlation structure, was found to be a significant factor influencing the performance of algorithms. In addition, the interaction between constraint slackness and population correlation structure was found to influence solution procedure performance and independent sampling seems to provide information closer to median performance. These results agree with prior studies. Tighter constraints and constraint coefficients with a

wider range of values yield harder problems, especially when both of these elements are combined in a problem.

In a different setting, Laguna et al. [39] developed a procedure to create hard problem instances for MGAP (Multilevel Generalized Assignment Problems). This approach embodies a random problem generator, labeled E, that draws constraint coefficients from an exponential distribution and generates the coefficients in the objective function to be inversely correlated. The effectiveness of this approach for generating difficult problems led us to consider adapting some of its features to the MKP setting.

Combining all the ideas described above for generating hard problems, we developed a generator that produces challenging MKP problems. Our approach uses independently exponential distributions over a wide range to generate the constraint coefficients, and the corresponding average for each variable is used to calculate directly correlated coefficients in the objective function. The RHS values are set to 0.25 of the sum of the corresponding constraint coefficients for the first part of the experiment, using the concept that tighter constraints yield difficult problems [29,47]. In the second part of the experiment, this multiple was set at 0.25, 0.5 and 0.75.

The problem generator we used to create the random instances of MKPs is designed as follows. The a_{ij} are integer numbers drawn from the exponential distribution $a_{ij} = 1.0 - 1000\ln(U(0,1))$, $i \in M$, $j \in N$. Again, for each $m - n$ combination, the right-hand side coefficients are set using the relation, $b_i = \alpha \sum_{j \in N} a_{ij}$, $i \in M$, where α is a tightness ratio and $\alpha = 0.5$ for the first part of the experiment. In the second part, $\alpha = 0.25$ for the first ten problems, $\alpha = 0.5$ for the next ten problems and $\alpha = 0.75$ for the remaining ten problems. The objective function coefficients (c_j's) were correlated to a_{ij} and generated as: $c_j = 10(\sum_{i \in M} a_{ij}/m) + 10U(0,1)$, $j \in N$. In all cases, $U(0,1)$ is a real number drawn form the continuous uniform generator.

The first part of the last experiment was performed on a SUN Ultrix 10, Solaris 7 and 250 MB RAM, using CPLEX V7.1. We use the number of nodes used by CPLEX as a measure of hardness for the instances generated in this way. The average number of nodes, in 10 instances, needed to solve a problem with 100 variables, 5 constraints and a tightness of 0.25 for the OR-library problems is 36,434.2. For a problem with the same characteristics using our generator, the average number of nodes is 1,366,447.

For this experiment we chose problems consisting of 100 variables, 5 constraints and $\alpha = 0.25$. These settings generate problems that can usually be solved to optimality

Table 6
Results for problems with 100 variables, 5 constraints and a tightness of 0.25.

	Corr. coeff.	Best	Objective	No. cuts	Var. fix	Cuts + fixed		CPLEX	
						Nodes	Sol'n time	Nodes	Sol'n time
Average	0.448163	2432	2514.2	21	0.3	1211665	875.547	1366447	767.82
Std.desv.	0.032846	109.5	107.02	4	1.6	559327.8	399.46	612223.9	345.41
Maximum	0.498274	2634	2728	28	9	2992562	2027.98	2685657	1483.83
Minimum	0.376742	2243	2323	10	0	363727	262.56	247035	139.54
Sum		72973	75425	64	9	36349947	26266.41	40993405	23034.5

Table 7
Computational results for our generator.

Vars.		Vars.	No.	Correlation	Objective value			GAP		Proved optimal[c]		Time finished[b]		Memory finished[a]	
n	α	fixed	cuts	coefficient	Fix + Cut	CPLEX	Imp.	Fix + Cuts	CPLEX	Fix + Cut	CPLEX	Fix + Cut	CPLEX	Fix + Cut	CPLEX
100	0.25	0	22	0.394	2504	2504	0	0.0001	0.0024	10	5	0	5	0	0
100	0.5	0	21	0.4484	5084	5081	3	0.005	0.0059	0	0	0	0	10	10
100	0.75	0	22	0.45077	7764	7761	3	0.00425	0.0048	0	0	0	0	10	10
250	0.25	0.1	23	0.44276	2667	2665	2	0.00368	0.0044	0	0	0	0	10	10
250	0.5	0.1	27	0.45968	5312	5312	0	0.00217	0.0024	1	0	0	1	9	9
250	0.75	0.2	27	0.44744	7932	7928	4	0.00156	0.0022	0	0	0	0	10	10
500	0.25	13	21	0.45055	2875	2875	0	0.00258	0.00261	0	0	0	0	10	10
500	0.5	6.5	30	0.44633	5674	5674	0	0.00142	0.00141	0	0	0	0	10	10
500	0.75	2.5	43	0.44723	8380	8380	0	0.00097	0.00098	0	0	0	0	10	10
Total					48192	48180	12			11	5	0	6	79	79
Average					5355	5353	2	0.00241	0.00301						

[a] Memory finished = 250 MB treesize memory.
[b] Time finished = 10800 s of solution time.
[c] Proved optimal, gap $\leqslant$ 0.0001.

in less than three hours. We generated 30 problems and allowed each one to run without a limit on solution time. In table 6, we show the average number of nodes and the solution time needed by CPLEX with and without our procedure.

The average number of nodes generated by our procedure reduces the average required by CPLEX by 154,782 (from 1,366,447 to 1,211,665) even though our approach was only implemented at a single node of the tree (corresponding to the first LP solution). More significantly, the reduction is not only in the average number of nodes, but in fact occurred for every setting tested. These instances were solved in CPLEX 6.5.2 and CPLEX 7.1. The CPU time changed considerably, but the difference in the number of nodes remained stable in favor of our procedure.

For the second part of the experiment, we used a Pentium III with 500 MHz and 128 MB in RAM. We concentrated in problems with 5 constraints and 100, 250 and 500 variables, and examined tightness values of 0.25, 0.5 and 0.75. These problems were solved using CPLEX V6.5.2 for Windows 98, without modifying its default parameters, both with and without the addition of our procedure. We allowed a maximum solution time of 3 hours (10,800 secs) and a memory tree size of 250 MB. We compared the objective value obtained, the number of problems solved to optimality with a gap of 0.0001 and the number of problems finished when the time or the tree size memory were exceeded in both methodologies. We also compared the gap values in both options.

The results are shown in table 7. The first two columns indicate the number of variables (n) and the tightness ratio (α) of a particular problem structure, with each problem structure containing 10 problem instances. Columns 3 and 4 present the number of variables fixed and the number of cuts generated with our procedure. Column 5 shows the average correlation coefficient between the objective function and the constraint coefficients for each setting. In columns 6 and 7 we present the average objective value for CPLEX with our cuts and for CPLEX alone. Column 8 presents the absolute difference between columns 6 and 7. Columns 9 and 10 show the average gap value at the moment CPLEX stopped the execution for both methodologies. The next 6 columns show the number of instances finished in each possible way: proved optimality with a gap less or equal to 0.0001, the time elapsed or the tree size memory used upon termination.

In all the settings tested, CPLEX performed much better on average when augmented with our procedure. As the table discloses, the use of the nested cuts allowed CPLEX to solve problems to optimality that could not otherwise be solved by using search trees of reasonable sizes.

7. Conclusions

Our procedure augments a branch and cut framework, by a process of fixing variables and adding global cuts. The approach can be applied every time the branch and cut method gets a better integer solution or it can be used as a preprocessing algorithm (as we have done), based on assuming a bound on an optimal objective value. Our computational experiments show that the preprocessing approach creates an enhanced method that allows problems to be solved by constructing smaller search trees.

References

[1] R. Aboudi and K. Jörnsten, Tabu search for general zero–one integer programs using the pivot and complement heuristics, ORSA Journal on Computing 6 (1994) 82–93.

[2] E. Balas, An additive algorithm for solving linear programs with zero–one variables, Operations Research 13 (1965) 517–546.

[3] E. Balas, Discrete programming by the filter method, Operations Research 19 (1967) 915–957.

[4] S. Bollapragada, O. Ghattas and J.N. Hooker, Optimal design of truss structures by mixed logical and linear programming, Manuscript, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh (1995).

[5] A.V. Cabot, An enumeration algorithm for knapsack problems, Operations Research 18 (1970) 306–311.

[6] P. Chu and J. Beasley, A genetic algorithm for the multidimensional knapsack problem, Journal of Heuristics 4 (1998) 63–86.

[7] Y. Crama and J. Mazzola, On the strength of relaxations of multidimensional knapsack problems, INFOR 32(4) (1994) 219–225.

[8] S. Dammeyer and S. Voss, Dynamic tabu list of management using reverse elimnation method, Annals of Operations Research 41 (1993) 31–46.

[9] G.B. Dantzig, Discrete variables problems, Operations Research 5 (1957) 266–277.

[10] R. Dembo and P. Hammer, A reduction algorithm for knapsack problems, Methods of Operations Research 36 (1980) 49–60.

[11] A. Drexel, A simulated annealing approach to the multiconstraint zero–one knapsack problem, Computing 40 (1988) 1–8.

[12] H.E. Dyer, Calculating surrogate constraints, Mathematical Programming 12 (1980) 225–278.

[13] J.F. Fontanari, A statistical analysis of the knapsack problem, Journal of Physics A: Mathematical and General 28 (1995) 4751–4759.

[14] A. Fréville and G. Plateau, Hard 0–1 multiknapsack test problems for size reduction methods, Investigación Operativa 1 (1990) 251–270.

[15] A. Fréville and G. Plateau, An efficient preprocessing procedure for the multidimensional 0–1 knapsack problem, Discrete Applied Mathematics 49 (1994) 189–212.

[16] B. Gavish and H. Pirkul, Efficient algorithms for solving multiconstraint zero–one knapsack problems to optimality, Mathematical Programming 31 (1985) 78–105.

[17] A. Geoffrion, An improved implicit enumeration approach for integer programming, Operations Research 17 (1969) 437–454.

[18] P.C. Gilmore and R.E. Gomory, The theory and computation of knapsack functions, Operations Research 14 (1966) 1045–1075.

[19] F. Glover, A multiphase-dual algorithm for the zero–one integer programming problem, Operations Research 13 (1965) 879–919.

[20] F. Glover, Surrogate constraints, Operations Research 16 (1968) 741–749.

[21] F. Glover, Flows in arborescences, Management Science 17(9) (1971) 568–586.

[22] F. Glover, Surrogate constraint duality in mathematical programming, Operations Research 23(3) (1975) 434–451.

[23] F. Glover and G.A. Kochenberger, Critical event tabu search for multidimensional knapsack problems, in: *Meta-Heuristics: Theory and Applications*, eds. I.H. Osman and J.P. Kelly (Kluwer Academic, 1996) pp. 407–427.

[24] F. Glover, H. Sherali and Y. Lee, Generating cuts from surrogate constraint analysis for zero–one and multiple choice programming, Computational Optimization and Applications 8 (1997) 151–172.

[25] F. Granot and P.L. Hammer, On the use of Boolean functions in 0–1 linear programming, Methods of Operations Research (1971) 154–184.

[26] H. Greenberg and W. Pierskalla, Surrogate mathematical programs, Operations Research 18 (1970) 924–939.

[27] P. Hammer, M. Padberg and U. Peled, Constraint pairing in integer programming, INFOR 13(1) (1975) 68–81.
[28] S. Hanafi and A. Fréville, An efficient tabu search approach for the 0–1 multidimensional knapsack problem, EJOR 106 (1998) 659–675.
[29] R. Hill and Ch. Reilly, The effects of coefficient correlation structure in two-dimensional knapsack problems on solution procedure performance, Management Science 46(2) (2000) 302–317.
[30] A. Hoff, A. Løkketangen and I. Mittet, Genetic algorithms for 0/1 multidimensional knapsack problems, Working Paper, Molde College, Molde, Norway (1996).
[31] J.N. Hooker, Logic-based methods for optimization, in: *Principles and Practice of Constraint Programming*, ed. A. Borning, Lecture Notes in Computer Science, Vol. 874 (1994) pp. 336–349.
[32] J.N. Hooker and N.R. Natraj, Solving 0–1 optimization problems with k-tree relaxation (1999), in preparation.
[33] J.N. Hooker and M.A. Osorio, Mixed logical/linear programming, Discrete Applied Mathematics 96–97 (1999) 395–442.
[34] J.N. Hooker, H. Yan, I. Grossmann and R. Raman, Logic cuts for processing networks with fixed charges, Computers and Operations Research 21 (1994) 265–279.
[35] R.E. Jeroslow and J.K. Lowe, Modeling with integer variables, Mathematical Programming Studies 22 (1984) 167–184.
[36] M.H. Karwan and R.L. Rardin, Some relationships between Lagrangian and surrogate duality in integer programming, Mathematical Programming 17 (1979) 230–334.
[37] S. Khuri, T. Back and J. Heitkotter, The zero/one multiple knapsack problem and genetic algorithms, in: *Proceedings of the 1994 ACM Symposium on Applied Computing (SAC'94)* (ACM Press, 1994) pp. 188–193.
[38] G. Kochenberger, G. McCarl and F. Wymann, A heuristic for general integer programming, Decision Sciences 5 (1974) 36–44.
[39] M. Laguna, J.P. Kelly, J.L. Gonzalez Velarde and F. Glover, Tabu search for the multilevel generalized assignment problem, European Journal of Operational Research 82 (1995) 176–189.
[40] A. Løkketangen and F. Glover, Probabilistic move selection in tabu search for zero–one mixed integer programming problems, in: *Meta-Heuristics: Theory and Applications*, eds. I.H. Osman and J.P. Kelly (1996) pp. 467–487.
[41] A. Løkketangen, K. Jörnsten and S. Storøy, Tabu search within a pivot and complement framework, International Transactions of Operations Research 1 (1994) 305–316.
[42] R. Loulou and E. Michaelides, New greedy-like heuristics for the multidimensional 0–1 knapsack problem, Operations Research 27 (1979) 1101–1114.
[43] S. Martello, D. Pisinger and P. Toth, New trends in exact algorithms for the 0–1 knapsack problem, European Journal of Operational Research 123(2) (1999) 325–336.
[44] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations* (Wiley, New York, 1990).
[45] M.A. Osorio and M. Laguna, Logic cuts for the multilevel generalized assignment problem (2002), to appear in European Journal of Operational Research.
[46] M.A. Osorio and R. Mújica, Logic cuts generation in a branch arid cut framework for location problems, Investigación Operativa 8(1–3) (1999) 155–166.
[47] H. Pirkul, A heuristic solution procedure for the multiconstraint zero–one knapsack problem, Naval Research Logistics 34 (1987) 161–172.
[48] D. Pisinger, Contributed research articles: a minimal algorithm for the bounded knapsack problem, ORSA Journal on Computing 12(1) (2000) 75–84.
[49] Ch. Reilly, Input models for synthetic optimization problems, in: *Proceedings of the 1999 Winter Simulation Conference* (1999) pp. 116–121.
[50] K.E. Schilling, The growth of m-constraint random knapsacks, European Journal of Operations Research 46 (1990) 109–112.

[51] S. Senju and Y. Toyoda, An approach to linear programming with 0–1 variables, Management Science 15 (1968) 196–207.
[52] W. Shih, A branch and bound method for the multiconstraint zero–one knapsack problem, Journal of Operation Research Society of Japan 30 (1979) 369–378.
[53] A.L. Soyster, B. Lev and W. Slivka, Zero–one programming with many variables and few constraints, European Journal of Operational Research 2 (1978) 195–201.
[54] K. Szkatula, The growth of multi-constraint random knapsacks with various right-hand sides of the constraints, European Journal of Operational Research 73 (1994) 199–204.
[55] K. Szkatula, The growth of multi-constraint random knapsacks with large right-hand sides of the constraints, Operations Research Letters 21 (1997) 25–30.
[56] J. Thiel and S. Voss, Some experiences on solving multiconstraint zero–one knapsack problems with genetic algorithms, INFOR 32 (1994) 226–242.
[57] H.M. Weingartner and D.N. Ness, Methods for the solution of the multidimensional 0/1 knapsack problem, Operations Research 15 (1967) 83–103.
[58] J.M. Wilson, Generating cuts in integer programming with families of specially ordered sets, European Journal of Operational Research 46 (1990) 101–108.