... manuscript No.
(will be inserted by the editor)

On the exact separation of cover inequalities of maximum depth

Daniele Catanzaro · Stefano Coniglio · Fabio Furini

Received: date / Accepted: date

Abstract We investigate the problem of exactly separating cover inequalities of maximum depth and we develop a pseudo-polynomial-time algorithm for this purpose. Compared to the standard method based on the maximum violation, computational experiments carried out on knapsack and multi-dimensional knapsack instances show that, with a cutting-plane method based on the maximum-depth criterion, we can optimize over the closure of the cover inequalities in a smaller number of cuts. These results encourage the research of efficient exact and heuristic separation algorithms based on maximizing the cut depth, also for other classes of valid inequalities.

Keywords Knapsack Problem, Cover Inequalities · Dynamic Programming · Mixed Integer Nonlinear Programming · Cutting Plane Generation

1 Introduction

The generation of *maximum-violation* inequalities is the standard approach for separating valid inequalities in Mixed Integer Linear Programs (MILPs) [31]. In the literature, however, some authors have speculated that separating (fractional) solutions by using a criterion different from maximizing the *cut violation* could lead to a cutting-plane generation method capable of converging in fewer iterations [1,3,4,6,7,15]. The maximization of the *cut depth* [6] (also referred to as "geometric distance" [7] or "efficacy" [4,11]) is one such criterion. Previous works, including [26,32,34], tackled the problem of separating inequalities of maximum depth only heuristically, due to its intrinsic difficulty. In this work, we focus on the case of *cover inequalities* and we design an exact separation algorithm for generating cuts of *maximum depth* based on dynamic programming. Our goal is not to develop a method that improves on the computing time taken by state-of-the-art algorithms based on the exact maximization of the cut violation. Rather, we aim to assess whether the adoption of the cut depth within an exact separation algorithm can lead to the generation of a smaller number of cuts. This way, we experimentally answer the question raised in [1,3,4,6,7,15] for the family of cover inequalities.

Daniele Catanzaro
Center for Operations Research and Econometrics (CORE), Université Catholique de Louvain, Louvain-la-Neuve, Belgium.
Luxembourg Institute of Socio-Economic Research (LISER), Luxembourg.
E-mail: daniele.catanzaro@uclouvain.be

Stefano Coniglio
Department of Mathematical Sciences, University of Southampton, Southampton, UK.
E-mail: s.coniglio@soton.ac.uk

Fabio Furini
Istituto di Analisi dei Sistemi ed Informatica "Antonio Ruberti", Consiglio Nazionale delle Ricerche (IASI-CNR), Roma, Italy.
E-mail: f.furini@iasi.cnr.it

2 Maximum-violation cover inequalities

Given a set of items $N := \{1, 2, \dots, n\}$, a non-negative integer weight w_j for each item $j \in N$, a knapsack capacity c, and a vector of binary variables $y \in \{0,1\}^n$, consider the knapsack constraint $\sum_{j \in N} w_j\, y_j \leq c$, where each binary variable y_j models the decision of inserting item $j \in N$ into the knapsack. One (or several) such constraints are present in countless real-world applications involving the solution of an MILP [22,23,24,29].

Subsets of items $C \subseteq N$ with total weight exceeding the knapsack capacity are called *covers*. This notion plays a central role in determining strong valid inequalities for the convex hull of optimization problems featuring one or more knapsack constraints. Since no more than $|C| - 1$ items of a given cover $C \subseteq N$ can be simultaneously inserted into the knapsack, the following constraints, called *cover inequalities*, are valid:

$$\sum_{j \in C} y_j \leq |C| - 1 \qquad \forall\, C \subseteq N : \sum_{j \in C} w_j \geq c + 1. \tag{1}$$

In either its original form (1) or in its lifted form (see the survey [22]), this exponentially-large set of constraints constitutes one of the earliest families of combinatorial inequalities adopted in a general-purpose 0-1 Integer Programming (IP) solver [16], and it it still routinely generated (heuristically) in most state-of-the-art MILP solvers.

Given a (fractional) point $\bar{y} \in [0,1]^n$, the *Separation Problem* (SP) for the cover inequalities calls for an inequality of type (1) with strictly positive *cut violation*, i.e., with $\sum_{j \in C} \bar{y}_j - |C| + 1 > 0$, or for a proof that no such inequality exists. The standard solution approach for solving the SP consists in searching for a cover $C \subseteq N$ with a cut violation as large as possible. By introducing a binary variable z_j for each $j \in N$, equal to 1 if and only if item j belongs to the cover, the problem can be formulated as the following Integer Linear Program (ILP):

$$\max_{z \in \{0,1\}^n} \left\{ \sum_{j \in N} (\bar{y}_j - 1)\, z_j + 1 : \quad \sum_{j \in N} w_j\, z_j \geq c + 1 \right\}.$$

Complementing the z variables by introducing a new variable $x_j := 1 - z_j$ for each $j \in N$, the *Maximum-Violation SP* (MV-SP) can be cast as the following Knapsack Problem (KP):

$$v := \max_{x \in \{0,1\}^n} \left\{ \sum_{j \in N} q_j\; x_j + \Psi : \quad \sum_{j \in N} w_j\, x_j \leq \Phi \right\}, \tag{MV-SP}$$

where $q_j := 1 - \bar{y}_j$ $(j \in N)$, $\Psi := \sum_{j \in N} \bar{y}_j - n + 1$, $\Phi := \sum_{j \in N} w_j - (c+1)$, and v is the *maximum cut violation*. Let x^* denote an optimal solution to the (MV-SP) and let $C^* := \{j \in N : x^*_j = 0\}$. If $v > 0$, the cover inequality corresponding to C^* is maximally violated by $\bar{y}$, whereas, if $v \leq 0$, we have a proof that no violated cover inequality exists.

3 Maximum-depth cover inequalities

Let $H_C := \big\{ y \in \mathbb{R}^n : \sum_{j \in C} y_j = |C| - 1 \big\}$ denote the hyperplane corresponding to the points $y \in \mathbb{R}^n$ for which the inequality $\sum_{j \in C} y_j \leq |C| - 1$ is tight. Given a point $\bar{y} \in [0,1]^n$, the *depth* of the cover inequality corresponding to C is defined as the *Euclidean* (or 2-norm) *point-to-hyperplane distance* between $\bar{y}$ and the point belonging to H_C that is closest to $\bar{y}$ in Euclidean norm—such point is often called the *projection* of $\bar{y}$ onto H_C. We denote such distance as $d_2(\bar{y}, H_C)$. Letting $z \in \{0,1\}^n$ be the characteristic vector of C, we have:

$$d_2(\bar{y}, H_C) := \frac{\left| \sum_{j \in N} \bar{y}_j z_j - |C| + 1 \right|}{||z||_2}, \tag{2}$$

where $||z||_2 := \sqrt{\sum_{j \in N} z_j^2}$ is the Euclidean norm (or 2-norm) of z. A derivation of this classical result using the tools of nonlinear optimization can be found in [27]. Besides the aforementioned works belonging to the cutting-plane literature in which the cut depth has been adopted, articles in which such distance is either maximized or minimized include [2,10,13]. We report an illustration in the following example.

Example 1 Consider a point $\bar{y} = (0.75, 0.76, 0.5) \in \mathbb{R}^3$ and two covers $C_1 = \{3\}$ and $C_2 = \{1, 2\}$. The cover inequalities corresponding to C_1 and C_2 are $y_3 \leq 0$ and $y_1 + y_2 \leq 1$. The corresponding hyperplanes H_{C_1} and H_{C_2} have equations $y_3 = 0$ and $y_1 + y_2 = 1$. The Euclidean point-to-hyperplane distance between $\bar{y}$ and the two hyperplanes is $d_2(\bar{y}, H_{C_1}) = \frac{|0.5-1+1|}{\sqrt{1}} = 0.5$ and $d_2(\bar{y}, H_{C_2}) = \frac{|0.75+0.76-2+1|}{\sqrt{2}} = \frac{0.51}{\sqrt{2}} \simeq 0.36$. Hence, H_{C_1} is farther away from $\bar{y}$ than H_{C_2} in terms of the Euclidean 2-norm point-to-hyperplane distance, whereas, in terms of cut violation, C_1 has a smaller violation (of $0.5-1+1 = 0.5$) than C_2 (which has a larger violation of $0.75+0.76-2+1 = 0.51$). A graphical representation is reported in Figure 1. □

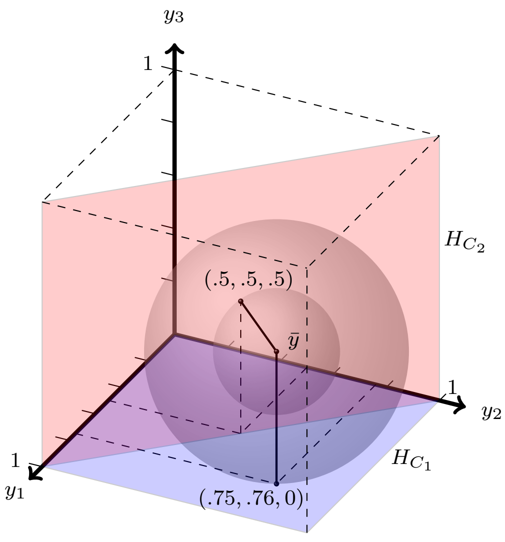

Fig. 1 An illustration of Example 1. The two solid segments connect $\bar{y}$ to the closest point in Euclidean point-to-hyperplane distance on either hyperplane H_{C_1} and H_{C_2}. According to such distance, H_{C_1} is farther away from $\bar{y}$ than H_{C_2}.

The separation problem calling for a cover inequality of maximum Euclidean-norm point-to-hyperplane distance, i.e., of *maximum depth*, can be cast as the following Mixed Integer Non-Linear Program (MINLP):

$$\max_{z \in \{0,1\}^n} \left\{ \frac{\sum_{j \in N} (\bar{y}_j - 1)\ z_j + 1}{||z||_2} : \quad \sum_{j \in N} w_j\, z_j \geq c + 1 \right\}.$$

The objective function is obtained from (2) by dropping the absolute value. It corresponds to the opposite of the *signed* 2-norm point-to-hyperplane distance between $\bar{y}$ and the set of all points $y \in \mathbb{R}^n$ which satisfy the cover inequality corresponding to $C := \{j \in N : z_j = 1\}$. It is therefore equal to $-d_2(\bar{y}, H_C)$ if $\bar{y}$ satisfies such inequality, and to $d_2(\bar{y}, H_C)$ otherwise.

Since the z variables are binary, $||z||_2 = \sqrt{\sum_{j \in N} |z_j|^2} = \sqrt{\sum_{j \in N} z_j}$. Complementing such variables as before, i.e., by introducing a binary variable $x_j := 1 - z_j$ for each $j \in N$, we obtain the following equivalent MINLP formulation:

$$\max_{x \in \{0,1\}^n} \left\{ \frac{\sum_{j \in N} q_j\ x_j + \Psi}{\sqrt{n - \sum_{j \in N} x_j}} : \quad \sum_{j \in N} w_j\, x_j \leq \Phi \right\},$$

The irrationality of the objective function prevents us from solving the problem exactly (i.e., up to an infinite precision) on a Turing machine. However, by discarding any solutions to the problem having a zero or negative violation, we can without loss of generality raise its objective function to the 2nd power. We obtain the following problem:

$$d := \max_{x\in\{0,1\}^n} \left\{ \frac{\left(\sum_{j\in N} q_j\ x_j + \Psi\right)^2}{n - \sum_{j\in N} x_j} : \quad \sum_{j\in N} w_j\, x_j \leq \Phi, \quad \sum_{j\in N} q_j\ x_j + \Psi > 0 \right\}, \qquad \text{(MD-SP)}$$

where d is the square of the maximum cut depth. Let x^* be an optimal solution to the (MD-SP) and let $C^* := \{j \in N : x^*_j = 0\}$. If the problem is feasible, which implies $d > 0$, C^* corresponds to a violated cover inequality of maximum depth. In contrast, if the problem is infeasible, we have a proof that no violated cover inequality exists. Note that the objective function is well-defined as its denominator is equal to 0 only for $x_j = 1$ for all $j \in N$, which is infeasible because $\sum_{j\in N} w_j \not\leq \Psi = \sum_{j\in N} w_j - (c+1)$.

4 Cutting-plane generation aspects

The (MD-SP) can be interpreted as a variant of the (MV-SP) in which, rather than a single objective function, the ratio of two different objective functions is maximized: the item profit incremented by the constant Ψ in the numerator and a measure of the sparsity of x, equal to the square root of the number of its zero entries, in the denominator. Due to the ratio, in the (MD-SP) we look for solutions with a large numerator and a small denominator, i.e., for solutions with a dense x, which, in turn, correspond to sparse covers. This results in a non-concave objective function, as shown in Figure 2.

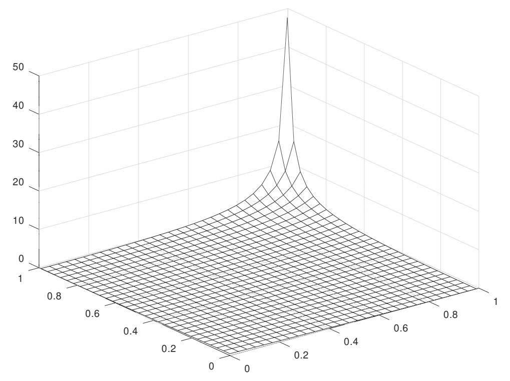

Fig. 2 An illustration of the objective function of the (MD-SP) for $n = 2$, $\bar{y} = (0, 0)$ and, accordingly, $q_1 = q_2 = 1$ and $\Psi = -1$, plotted for x_1 and $x_2 \in [0, 0.99]$. It corresponds to the non-concave function $\frac{(x_1+x_2-1)^2}{2-x_1-x_2}$.

It is well-known that a cover inequality is non-dominated whenever the corresponding cover C is *inclusion-wise minimal*, i.e., whenever, for any $j \in C$, $C \setminus \{j\}$ is not a cover. When separating maximum-depth cover inequalities, the nature of the denominator of the (MD-SP) implies the following:

Proposition 1 *Any cover corresponding to an optimal solution to the* (MD-SP) *is minimal.*

Proof Let x^* be an optimal solution to the (MD-SP) and let $C^* := \{j \in N : x^*_j = 0\}$ be the corresponding cover. If C^* is not minimal, there exists a minimal cover $\tilde{C}$ with $\tilde{C} \subset C^*$. Let $\tilde{x}_j \in \{0, 1\}^n$ such that $\tilde{C} = \{j \in N : \tilde{x}_j = 0\}$. As $\tilde{C} \subset C^*$, there is some $j \in N$ with $j \in C^*$ and

$j \notin \tilde{C}$. Thus, $x^*_j = 0$ and $\tilde{x}_j = 1$, implying that $\tilde{x}$ achieves a numerator at least as large as x^* and a strictly smaller denominator, which shows that x^* is not optimal. □

Besides this minimality property, the (MD-SP) establishes a trade-off between the cut violation (in the numerator) and the cut sparsity (in the denominator). The impact of sparse cuts in integer programming has been addressed in a number of papers, including [1,3,8,18]. We will further assess such impact in Section (6).

We conclude this section with a numerical example that shows how the sparsity of x may impact on the optimality of a solution to the (MV-SP) when compared to one which is optimal for the (MD-SP).

Example 2 Consider an instance of the (MV-SP) and (MD-SP) with $n = 4$ items, weights $w_1 = 3$, $w_2 = 1$, $w_3 = 2$, $w_4 = 5$, profits $q_1 = 0.25$, $q_2 = 0.23$, $q_3 = 0.24$, $q_4 = 0.5$, capacity $c = 5$. With these values, we have $\Phi = 5$ and $\Psi = -0.22$. Consider two solutions: $\{1, 3\}$ and $\{4\}$. From the perspective of the (MV-SP), $\{1, 3\}$ is suboptimal (it has value $0.25+0.24-0.22 = 0.27$), whereas $\{4\}$ is optimal (it has value $0.5 - 0.22 = 0.28$). From the perspective of the (MD-SP), $\{1, 3\}$ is optimal (it has value $\frac{(0.25+0.24-0.22)^2}{4-2} \approx 0.036$), whereas $\{4\}$ is suboptimal (it has value $\frac{(0.5-0.22)^2}{4-1} \approx 0.026$). This difference is due to the fact that, in the (MD-SP), the smaller profit of $\{1, 3\}$ is compensated by its smaller sparsity, outweighing the larger profit of $\{4\}$, which is sparser. □

5 A dynamic programming algorithm for (MD-SP)

The separation problem calling for a cover inequality of strictly positive violation (or for a proof that no such inequality exists) is known to be weakly $\mathcal{NP}$-hard [25]. As any algorithm for the solution of the (MD-SP) implicitly solves such separation problem, the (MD-SP) is *at least* weakly $\mathcal{NP}$-hard. In this section, we show that such a problem can be solved in pseudo-polynomial time, thus showing that it is indeed weakly $\mathcal{NP}$-hard.

The denominator of the objective function of the (MD-SP) takes discrete values in the set $\{1, \dots, n-1\}$ (recall that the (MD-SP) is infeasible for $k = n$) as a function of the number of items in the solution. We refer to this feature as the *cardinality* of a solution. It is not difficult to see that, by removing any item j from an optimal solution of cardinality $k \in \{1, \dots, n-1\}$, the remaining set of items in the solution must be optimal for the subproblem with capacity $\Phi - w_j$, item set $N \setminus \{j\}$, and cardinality $k-1$, as, if not, the solution of cardinality k is not optimal. Thus, the (MD-SP) exhibits the so-called *optimal-substructure property*, which can be exploited by an approach relying on Bellman's recursion [24,29].

Based on this observation, we now derive a Dynamic Programming (DP) algorithm for the (MD-SP). For each item $j \in N$, cardinality value $k \in \{1, \dots, n-1\}$, and capacity value $s \in \{0, \dots, \Phi\}$, we denote by $f^k_j(s)$ the maximum total profit in terms of the q values increased by the constant Ψ (this corresponds to the numerator of the objective function of (MD-SP)) that is achievable with k items belonging to $\{1, \dots, j\} \subseteq N$ with a total weight no larger than $s \le \Phi$. The following proposition holds:

Proposition 2 *The* (MD-SP) *can be solved in $O(n^2\Phi)$, and its optimal solution value is:*

$$d = \max_{k \in \{1,\dots,n-1\}} \left\{ \frac{f^k_n(\Phi)^2}{(n-k)} : f^k_n(\Phi) > 0 \right\},$$

whereas, if $f^k_n(\Phi) \le 0$ for all $k \in \{1, \dots, n-1\}$, the (MD-SP) *is infeasible.*

Proof For $j = 1$ and $s \in \{0, \dots, \Phi\}$, the following holds for all $k \in \{1, \dots, n-1\}$:

$$f^k_1(s) = \begin{cases} q_1 + \Psi & \text{if } k = 1 \text{ and } w_1 \le s \\ -\infty & \text{if } k = 1 \text{ and } w_1 > s \\ -\infty & \text{if } k \ge 2. \end{cases} \tag{3}$$

The last case is due to the fact that no more than a single item can be chosen for $j = 1$.

For $j \in \{2, \dots, n\}$, $s \in \{w_j, \dots, \Phi\}$, and $k = 1$, we have that either (i) item j is not chosen, leading to $f_j^1(s) = f_{j-1}^1(s)$, or (ii) item j is the only chosen item, leading to $f_j^1(s) = q_j + \Psi$. Therefore:

$$f_j^1(s) = \max\left\{f_{j-1}^1(s), q_j + \Psi\right\}. \tag{4}$$

Instead, for $j \in \{2, \dots, n\}$, $s \in \{0, \dots, w_j - 1\}$, and $k = 1$ we have $f_j^1(s) = f_{j-1}^1(s)$ as item j does not fit.

For $j \in \{2, \dots, n\}$, $s \in \{w_j, \dots, \Phi\}$, and $k \in \{2, \dots, n-1\}$, we have that either (i) item j is not chosen, leading to $f_j^k(s) = f_{j-1}^k(s)$, or (ii) item j is chosen, leading to $f_j^k(s) = f_{j-1}^{k-1}(s - w_j) + q_j$. Hence:

$$f_j^k(s) = \max\left\{f_{j-1}^{k-1}(s - w_j) + q_j, f_{j-1}^k(s)\right\}. \tag{5}$$

Instead, for $j \in \{2, \dots, n\}$, $s \in \{0, \dots, w_j - 1\}$, and $k \in \{2, \dots, n-1\}$ we have $f_j^k(s) = f_{j-1}^k(s)$ as item j does not fit.

The square of the optimal solution value of the (MD-SP) is obtained by evaluating the largest value taken by $\frac{(f_n^k(\Phi))^2}{(n-k)}$ over $k \in \{1, \dots, n-1\}$, and by ignoring any case with $f_n^k(\Phi) \leq 0$. This last step takes $O(n)$. If $f_n^k(\Phi) \leq 0$ for all k, the problem is infeasible and $\delta = -\infty$. As computing $f_j^k(s)$ for all $j \in N$, $k \in \{1, \dots, n-1\}$, and $s \in \{0, \dots, \Phi\}$ via (3), (4), and (5) takes $O(n^2 \Phi)$, the claim follows. □

We note that it is not necessary to compute $f_j^k(s)$ for all values of $k \in \{1, \dots, n-1\}$. In fact, it suffices to consider k up to $\bar{k} := \max_{x \in \{0,1\}^n} \left\{\sum_{j \in N} x_j : \sum_{j \in N} w_j \, x_j \leq \Phi\right\}$. This value can be computed in $O(n \log n)$ by sorting the items in N in non-increasing order of weights and adding them to the solution in that order until the capacity Φ is saturated. Thanks to this, the complexity of the algorithm reduces to $O(n \, \bar{k} \, \Phi)$. An additional speed-up can be obtained by considering, in the recursion, values of k up to the minimum between $\bar{k}$ and the index j of the item currently under examination.

We conclude this section illustrating the execution of the DP algorithm on an example.

Example 3 Consider again the instance of Example 2, with $n = 4$, weights $w_1 = 3$, $w_2 = 1$, $w_3 = 2$, $w_4 = 5$, profits $q_1 = 0.25$, $q_2 = 0.23$, $q_3 = 0.24$, $q_4 = 0.5$, $\Phi = 5$, and $\Psi = -0.22$. The maximum number of items that can be simultaneously chosen is $\bar{k} = 2$. The values of the recursive function $f_j^k(s)$ are shown in Table 1. For each item $j \in N$, the values corresponding to $k > j$ (which can be ignored) are denoted by "-". In order to retrieve the optimal solution value, it is sufficient to examine the values of $f_j(s)^k$ for $j = 4$ (the last item) and $s = \Phi = 5$ for all values of k. In particular, we have a solution of value $d = \frac{(0.5-0.22)^2}{4-1} \approx 0.026$ for $k = 1$ and one of value $d = \frac{(0.25+0.24-0.22)^2}{4-2} = 0.036$ for $k = 2$. The optimal solution value, which is obtained for $k = 2$, corresponds to the solution $x^* = (1, 0, 1, 0)$, which encodes the cover $C^* := \{j \in N : x_j^* = 0\} = \{2, 4\}$. □

Table 1 Execution of the DP algorithm on the instance of Example 3. For each item of index $j \in N$, the corresponding part of the table reports the value of $f_j^k(s)$ for all values of s and k.

	$j=1$							$j=2$					
	$s=0$	$s=1$	$s=2$	$s=3$	$s=4$	$s=5$		$s=0$	$s=1$	$s=2$	$s=3$	$s=4$	$s=5$
$k=1$	$-\infty$	$-\infty$	$-\infty$	0.03	0.03	0.03	$k=1$	$-\infty$	0.01	0.01	0.03	0.03	0.03
$k=2$	-	-	-	-	-	-	$k=2$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	0.26	0.26
	$j=3$							$j=4$					
	$s=0$	$s=1$	$s=2$	$s=3$	$s=4$	$s=5$		$s=0$	$s=1$	$s=2$	$s=3$	$s=4$	$s=5$
$k=1$	$-\infty$	0.01	0.02	0.03	0.03	0.03	$k=1$	$-\infty$	0.01	0.02	0.03	0.03	0.28
$k=2$	$-\infty$	$-\infty$	$-\infty$	0.25	0.26	0.27	$k=2$	$-\infty$	$-\infty$	$-\infty$	0.25	0.26	0.27

6 Computational experiments

In this section, we investigate the impact of generating maximum-depth cover inequalities within a cutting-plane method that relies on solving the (MD-SP). In order to evaluate the performance of this approach, we adopt as a baseline the *standard* cutting-plane method based on the generation of maximum-violation cover inequalities via the solution of the (MV-SP). Our primary goal is to assess whether the exact solution of a separation problem based on the cut depth may lead to the generation of a smaller number of cuts.

Specifically, we consider two cutting-plane generation methods: `MaxD` and `MaxV`. `MaxD` consists in the generation of cover inequalities of maximum-depth by solving the (MD-SP) via the DP algorithm we proposed in Section 5. We recall that, due to Proposition (1), every cover corresponding to an optimal solution to the (MD-SP) is minimal. `MaxV` is based on the generation of cover inequalities of maximum violation by solving the (MV-SP) via the standard $O(n\Phi)$ DP algorithm for the KP described in, e.g., [24,29]. Note that, if some of the components of the point $\bar{y}$ being separated are equal to 1 (which leads to a 0 coefficient in the objective function), the solution to the (MV-SP) may not be maximal, which implies that the corresponding cover may not be minimal. For this reason, we implemented the DP algorithm as suggested in [14] (see Proposition 3 at page 624), so that it generates by construction minimal cover of maximum violation with no impact on its computational complexity. As can be expected, the impact of generating minimal covers is substantial. Indeed, our experiments show that, with a standard DP algorithm such as the one in [24,29] which does not generate minimal covers by construction, the number of cover inequalities increases by up to 70%. It is worth noticing that, for a given violation, minimal covers are sparser, which is in line with the ratio in the objective function of the (MD-SP), where the cut sparsity corresponds to the denominator.

As a case study, we consider two combinatorial problems featuring one or more knapsack constraints: (*i*) the *Knapsack Problem* (KP) and (*ii*) the *Multi-dimensional Knapsack Problem* (MKP). This family of problems can be cast as the following MILP:

$$\max_{y\in\{0,1\}^n}\left\{\sum_{j\in N} p_j\, y_j : \sum_{j\in N} w_{ij}\, y_j \le c_i \quad i\in M\right\}, \tag{6}$$

where y_j is a binary variable equal to 1 if and only if item $j \in N$ is chosen, p_j is the item profit, and M is a set of m knapsack constraints, with capacities c_i $(i \in M)$ and item weights w_{ij} $(i \in M,\ j \in N)$. The KP, as well as its numerous variants, has been extensively studied in the literature for over a century [24,29], with works dating as early as 1897 [30]. We refer the reader to [17,19,20,21,24,29] for more details and the KP and its variants, and to [5,22,23] for works focused on its polyhedral aspects. The MKP is a multi-dimensional extension of the KP featuring $m > 1$ knapsack constraints. This problem has also been extensively studied in the literature, see e.g., [24,29,33].

Several classes of KP instances have been proposed. In this work, we consider six classes originally introduced in [28]: *Uncorrelated*, *Weakly Correlated*, *Strongly Correlated*, *Inversely Correlated*, *Almost Strongly Correlated*, and *Subset Sum*. As our experiments reveal that, on the *Strongly Correlated*, *Inversely Correlated*, and *Almost Strongly Correlated*, the number of cover inequalities that is generated is extremely small (2 or 3 cuts on average), in the following we only report the results for the remaining three classes. The instances depend on two user-supplied parameters: $\bar{c}$ and $\bar{R}$. The capacity c is defined as a percentage of the total item weight $W := \sum_{j=1}^{n} w_j$, and is set to $c := \lfloor \bar{c} \cdot W \rfloor + 1$. $\bar{R}$, which defines the order of magnitude of the item profits and weights, is set to 1,000. For each class, we generate 10 instances (with different seeds) per combination of n, $\overline{R}$, and $\bar{c}$, adopting $\overline{R} = 1000$, $n \in \{100, 120, 140, 160, 180, 200\}$, and $\bar{c} \in \{25\%, 50\%, 75\%\}$, for a total of 180 instances per class.

As for the MKP, in this work we consider the nine classes of instances belonging to the OR Lib [9] which are introduced in [12]. Experimentally, we observe that the number of inequalities generated on the instances of classes 4 to 9 is extremely small (no larger, on average, than 3 cuts for classes 4, 5, and 6, and equal to 0 for classes 7, 8, and 9). We also observe that, due to featuring $n = 500$ items, the instances of class 3 require a too large memory allocation for the DP algorithm of Section 5. For this reason, in the following we only report the results obtained for classes 1 and 2. Each class consists of 30 instances featuring $m = 5$ knapsack constraints and $n = 100$ items for class 1 and $n = 250$ items for class 2. We refer the reader to [12] for further details on these instances.

In order to better assess the performance of the different cutting plane methods, we adopt, as in [3,15,14,35], a clean setting in which, starting from the Linear Programming (LP) relaxation of (6), a single inequality is generated at each iteration per knapsack constraint. The DP algorithms are implemented in `C++` and compiled with `gcc` v9.2.1 with flag `-O3`. The LP relaxations are solved with `CPLEX` 12.9. The experiments are run on an 8-core Intel i7-3770 @ 3.4 GHz, equipped with 16 GB of RAM, and running Ubuntu 64-bit, release 19.10. The source code of the algorithms used in the experiments can be downloaded from `https://github.com/fabiofurini/COVER_MAX_DEPTH`.

As far as the KP instances are concerned, our results show that `MaxV` generates a total of 10,003 inequalities on the whole testbed, whereas `MaxD` only generates 8,051, for a reduction of 19.51%. When analyzing the three groups of KP instances individually, we observe that fewer inequalities are generated for the *Uncorrelated* and *Weakly Correlated* instances than for the *Subset Sum* instances, regardless of the cutting-plane method that is used. In particular, we observe that `MaxD` leads to a small reduction of 2.44% on the *Uncorrelated* instances, to a slightly larger one of 7.81% on the *Weakly Correlated* instances, and to a more substantial one of 24.99% on the *Subset Sum* instances. `MaxD` generates the smallest number of inequalities in ≈80% of the instances, whereas `MaxV` does so in only ≈ 60% of them.

Focusing on the *Subset Sum* instances, on which the number of generated inequalities by either algorithm is more substantial, we observe that `MaxV` generates an average number of inequalities ranging from 36.17 ($n = 100$) to 48.27 ($n = 120$). This is in contrast to `MaxD`, which generates an average number of inequalities ranging from 23.47 ($n = 100$) to 35.93 ($n = 180$). A similar behavior can be observed in terms of the maximum number of generated inequalities, which ranges from 101 ($n = 100$) to 151 ($n = 200$) for `MaxV` and from 82 ($n = 100$) to 131 ($n = 180$) for `MaxD`.

As far as the MKP instances are concerned, `MaxV` generates a total of 663 inequalities, while `MaxD` generates only 598, for a reduction of 11.68%. For the instances of class 1, the reduction is of 11.68% and, for those of class 2, it is of 7.69%. On the whole MKP testbed, `MaxD` generates the smallest number of inequalities in ≈85% of the instances, whereas `MaxV` does so in ≈65% of them, confirming that `MaxD` generates the smallest number of inequalities also for the MKP instances.

Table 2 summarizes the results obtained for `MaxV` and `MaxD`. Each line reports the total number of cuts generated by each method aggregated by the different classes of instances of the two problems.

Table 2 Total number of generated cover inequalities for the considered classes of instances.

Class	`MaxV`	`MaxD`
KP Uncorrelated	1,395	1,361
KP Weakly Correlated	1,358	1,252
KP Subset Sum	7,250	5,438
Total	10,003	8,051
MKP Class 1	351	310
MKP Class 2	312	288
Total	663	598

As the separation algorithm embedded in `MaxD` runs in $O(n\,\bar{k}\,\Phi)$, its computing time is larger than the one of the classical DP algorithm used in `MaxV`, which runs in $O(n\,\Phi)$. While this may be compensated by the reduction in the number of cutting-plane iterations (which also leads to solving a smaller number of LP relaxations), such reduction is, while substantial on the two testbeds we considered, not sufficient to obtain a cutting plane algorithm that is also faster in terms of computing time. Over the whole KP testbed, the total time taken by `MaxD` is of $\approx 65,000$ seconds, whereas the time taken by `MaxV` is of ≈ 650 seconds. For the MKP instances, `MaxD` requires a total time of $\approx 20,000$ seconds, while `MaxV` requires ≈ 150 seconds. This is in line with the fact that the (MD-SP) solved in `MaxD` is intrinsically harder than the (MV-SP) due to being a non-linear variant of the KP with a fractional objective function.

In order to also assess the performance of our DP algorithm for the exact solution of the (MD-SP) as a standalone problem, we compare it to `BARON` v19.12.7, which is one of the state-of-the-art solvers for global optimization. We adopt a time limit of 600 seconds. For these experiments, we set $n \in \{30, 40, 50, 75, 100, 150\}$ and $\bar{c} \in \{25\%, 50\%, 75\%\}$, $q_j := p_j$ for all $j \in N$, $\Phi := c$, and $\Psi := 1$, and consider all the KP instances of the six aforementioned classes, for a total of 144 instances of the (MD-SP). The experiments highlight the difficulties of `BARON` in solving such problem. Indeed, our DP algorithm is able to solve all of the 144 instances considered, while `BARON` only solves 132 (91.6%). In particular, `BARON` fails to solve at least an instance for every value of $n \geq 75$. Ignoring the instances on which the time limit is hit, the average computing time for `BARON` is of 3 seconds, whereas the average computing time for our DP algorithm is of 0.46 seconds.

We remark that it is rare for a state-of-the-art branch-and-cut algorithm (such as, e.g., `CPLEX`, `Gurobi`, and `Xpress`) to feature an exact cutting-plane separation algorithm, as, in general, the separation problem (for cuts that are generated to strengthen an LP relaxation) is typically solved heuristically. Our experiments suggest that taking the cut depth into consideration in such heuristic algorithms could lead to a reduction in the number of iterations, and it further supports the the adoption of depth-based measures in the cut-selection phase, see [1]. Furthermore, a reduced number of inequalities results in a more compact LP relaxation, which can be beneficial in branch-and-cut algorithms due to leading to smaller relaxations which are faster to reoptimize. A smaller LP relaxation is also likely to lead to smaller condition number of the inverse basic matrix, making the method less prone to numerical errors.

7 Conclusions

This work constitutes a first step in the generation of maximum-depth inequalities. In particular, it characterizes the separation problem of cover inequalities of maximum depth, providing a pseudo-polynomial-time separation algorithm based on dynamic programming. It also confirms, empirically, that a maximum-depth approach can lead to a substantial reduction in the number of cuts generated by a cutting-plane algorithm. Interesting lines of future research include measuring the impact of the generation of maximum-depth cover inequalities for general 0-1 integer programming problems involving knapsack constraints, as well as the study of the computational aspects of maximum-depth separation for other families of inequalities, also from a heuristic perspective.

Acknowledgments

D. Catanzaro acknowledges support from the Belgian National Fund for Scientific Research (FRS-FNRS) via the research grant "Crédit de Recherche" ref. S/25-MCF/OL J.0026.17, the Université Catholique de Louvain via the "Fonds Spéciaux de Recherche" (FSR) 2017-2021, and the Fondation Louvain via the research grant COALESCENS of the funding program "Le numérique au service de l'humain".

References

1. T. Achterberg. SCIP: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.
2. E. Amaldi and S. Coniglio. A distance-based point-reassignment heuristic for the k-hyperplane clustering problem. *European Journal of Operational Research*, 227(1):22–29, 2013.
3. E. Amaldi, S. Coniglio, and S. Gualandi. Coordinated cutting plane generation via multi-objective separation. *Mathematical Programming*, 143(1-2):87–110, 2014.
4. G. Andreello, A. Caprara, and M. Fischetti. Embedding {0, 1/2}-cuts in a branch-and-cut framework: A computational study. *INFORMS Journal on Computing*, 19(2):229–238, 2007.
5. P. Avella, M. Boccia, and I. Vasilyev. A computational study of exact knapsack separation for the generalized assignment problem. *Computational Optimization and Applications*, 45(3):543–555, 2010.
6. E. Balas, S. Ceria, and G. Cornuéjols. A lift-and-project cutting plane algorithm for mixed 0–1 programs. *Mathematical Programming*, 58(1-3):295–324, 1993.

7. E. Balas, S. Ceria, and G. Cornuéjols. Mixed 0-1 programming by lift-and-project in a branch-and-cut framework. *Management Science*, 42(9):1229–1246, 1996.
8. E. Balas and A. Saxena. Optimizing over the split closure. *Mathematical Programming*, 113(2):219–240, 2008.
9. J. E. Beasley. Or-library: distributing test problems by electronic mail. *Journal of the operational research society*, 41(11):1069–1072, 1990.
10. P. Bradely and O. Mangasarian. k-plane clustering. *Journal of Global Optimization*, 16:23–32, 2000.
11. S. Chopra and M. R. Rao. The Steiner tree problem II: Properties and classes of facets. *Mathematical Programming*, 64(1-3):231–246, 1994.
12. P. C. Chu and J. E. Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of heuristics*, 4(1):63–86, 1998.
13. S. Coniglio. The impact of the norm on the k-hyperplane clustering problem: Relaxations, restrictions, approximation factors, and exact formulations. In *Proceedings of the 10th Cologne-Twente Workshop on Graphs and Combinatorial Optimization (CTW), Frascati, Italy*, pages 118–121, 2011.
14. S. Coniglio, F. D'Andreagiovanni, and F. Furini. A lexicographic pricer for the fractional bin packing problem. *Operations Research Letters*, 47:622–628, 2019.
15. S. Coniglio and M. Tieves. On the generation of cutting planes which maximize the bound improvement. In *International Symposium on Experimental Algorithms*, pages 97–109. Springer, 2015.
16. H. Crowder, E. L. Johnson, and M. Padberg. Solving large-scale zero-one linear programming problems. *Operations Research*, 31(5):803–834, 1983.
17. C. D'Ambrosio, F. Furini, M. Monaci, and E. Traversi. On the product knapsack problem. *Optimization Letters*, 12(4):691–712, 2018.
18. S. S. Dey, M. Molinaro, and Q. Wang. Approximating polyhedra with sparse inequalities. *Mathematical Programming*, 154(1-2):329–352, 2015.
19. F. Furini, M. Iori, S. Martello, and M. Yagiura. Heuristic and exact algorithms for the interval min–max regret knapsack problem. *INFORMS Journal on Computing*, 27(2):392–405, 2015.
20. F. Furini, I. Ljubić, and M. Sinnl. An effective dynamic programming algorithm for the minimum-cost maximal knapsack packing problem. *European Journal of Operational Research*, 262(2):438–448, 2017.
21. F. Furini, M. Monaci, and E. Traversi. Exact approaches for the knapsack problem with setups. *Computers & Operations Research*, 90:208–220, 2018.
22. C. Hojny, T. Gally, O. Habeck, H. Lüthen, F. Matter, M. E. Pfetsch, and A. Schmitt. Knapsack polytopes: a survey. *Annals of Operations Research*, pages 1–49, 2019.
23. K. Kaparis and A. N. Letchford. Separation algorithms for 0-1 knapsack polytopes. *Mathematical programming*, 124(1-2):69–91, 2010.
24. H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, Berlin Heidelberg, 2004.
25. D. Klabjan, C. A. Tovey, and G. L. Nemhauser. The complexity of cover inequality separation. *Operations Research Letters*, 23(1-2):35–40, 1998.
26. M. Lübbecke. *Engine Scheduling by Column Generation*. PhD thesis, Technische Universität Braunschweig, 2001.
27. O. L. Mangasarian. Arbitrary-norm separating plane. *Operations Research Letters*, 24(1-2):15–23, 1999.
28. S. Martello, D. Pisinger, and P. Toth. Dynamic programming and strong bounds for the 0-1 knapsack problem. *Management Science*, 45(3):414–424, 1999.
29. S. Martello and P. Toth. *Knapsack problems: Algorithms and computer implementations*. John Wiley & Sons, New York, 1990.
30. G. B. Mathews. On the Partition of Numbers. *Proceedings of the London Mathematical Society*, s1-28(1):486–490, 11 1896.
31. G. L. Nemhauser and L. A. Wolsey. *Integer and combinatorial optimization*. Wiley-Interscience, New York, 1999.
32. N. Papadakos. Integrated airline scheduling. *Computers & Operations Research*, 36(1):176–195, 2009.
33. J. Puchinger, G. R. Raidl, and U. Pferschy. The multidimensional knapsack problem: Structure and algorithms. *INFORMS Journal on Computing*, 22(2):250–265, 2010.
34. F. Vanderbeck. *Decomposition and Column Generation for Integer Program*. PhD thesis, Faculty of Applied Sciences, Université Catholique de Louvain, 1994.
35. A. Zanette, M. Fischetti, and E. Balas. Lexicography and degeneracy: Can a pure cutting plane algorithm work? *Mathematical programming*, 130(1):153–176, 2011.