

On the exact separation of mixed integer knapsack cuts

Ricardo Fukasawa Marcos Goycoolea

August 16, 2007

Abstract

During the last decades, much research has been conducted deriving classes of valid inequalities for single-row mixed integer programming polyhedrons. However, no such class has had as much practical success as the MIR inequality when used in cutting plane algorithms for general mixed integer programming problems. In this work we analyze this empirical observation by developing an algorithm which takes as input a point and a mixed integer knapsack polyhedron, and either proves the point is in the convex hull of said polyhedron, or finds a separating hyperplane, or knapsack cut. The main feature of this algorithm is a specialized subroutine for solving the Mixed Integer Knapsack Problem which exploits dominance relationships. To our knowledge, this is the first algorithm proposed for this problem. Exactly separating over the closure of mixed integer knapsack sets allows us to establish natural benchmarks by which to evaluate specific classes of knapsack cuts. Using these benchmarks on Miplib 3.0 and Miplib 2003 instances we analyze the performance of MIR inequalities. Our computations, which are performed in exact arithmetic, are surprising: Averaging over the 78 instances in which knapsack cuts afford bound improvements, MIR cuts alone achieve 95% of the observed gain.

1 Introduction

Consider positive integers n, m and let $d \in \mathbb{Q}^m$, $D \in \mathbb{Q}^{m \times n}$, $l \in \{\mathbb{Q} \cup \{-\infty\}\}^n$ and $u \in \{\mathbb{Q} \cup \{+\infty\}\}^n$. Let $I \subseteq N := \{1, \dots, n\}$ and consider the mixed integer set:

$$P = \{x \in \mathbb{R}^n : Dx \leq d, l \leq x \leq u, x_i \in \mathbb{Z}, \forall i \in I\}.$$

We say that a mixed integer knapsack set of the form,

$$K = \{x \in \mathbb{R}^n : ax \leq b, l \leq x \leq u, x_i \in \mathbb{Z}, \forall i \in I\}$$

with $b \in \mathbb{Q}$, $a \in \mathbb{Q}^n$ is *implied* by P if (a, b) is a non-negative linear combination of rows obtained from (D, d) . Observe that if K is implied by P , then $P \subseteq K$. Hence, any inequality which is valid for K is also valid for P . We henceforth call such inequalities *knapsack cuts* derived from K .

Deriving strong knapsack cuts is of great practical importance to Mixed Integer Programming (MIP). In fact, most cutting planes known for general mixed integer programming are knapsack cuts. For example, Gomory Mixed Integer (GMI) cuts [32, 45] are knapsack cuts derived from the tableaus of linear programming relaxations, and Lifted Cover Inequalities [19, 36] are knapsack

cuts derived from the original rows of P . Other classes of knapsack cuts include mixed-integer-rounding (MIR) cuts and their variations [17, 43, 46], split cuts [16], lift-and-project cuts [8], and group cuts [21, 33] – to name but a few.

In practice, the most successful class of knapsack cuts for MIP are the GMI/MIR cuts [13]. Dash and Günlük [22] have recently done an empirical study to better understand this practical success. They show that, in a significant number of instances, after adding GMI cuts for all the optimal tableau rows and reoptimizing the continuous relaxation, there are no more violated group cuts. Their results are quite surprising, and already suggest how strong are GMI cuts within the context of group cuts.

There are, however, natural questions left to answer: In the case where there are some violated group cuts, do these cuts give any significant improvement in the continuous relaxation bound? In the case where there are no violated group cuts, does there exist any other classes of knapsack cuts that are violated and can yield significantly better bounds? In fact, S. Dash (personal communication) proposed a closely related conjecture that for many problems in MIPLIB 3.0 the combined effect of GMI cuts from different tableau rows is to push the resulting solution x into the convex hull of the mixed-integer knapsack sets defined by the tableau rows.

To help resolve these questions, in this paper we discuss an empirical methodology for evaluating sub-classes of knapsack cuts using the objective function of the continuous relaxation as a measure of quality. Formally, consider P as defined above, $c \in \mathbb{Q}^n$, and \mathcal{C} a set of valid inequalities for P . Define,

$$z^*(\mathcal{C}) = \min\{cx : Dx \leq d, l \leq x \leq u, \pi x \leq \pi_o \ \forall (\pi, \pi_o) \in \mathcal{C}\}.$$

Observe that the value $z^*(\mathcal{C})$ defines a benchmark by which to evaluate classes of cuts that are subsets of \mathcal{C} . This idea will be applied in our context in the following way: Given a family of implied knapsack sets \mathcal{K} , let $\mathcal{C}^{\mathcal{K}}$ and $\mathcal{M}^{\mathcal{K}}$ represent, respectively, the set of all knapsack cuts and MIR cuts which can be derived from sets $K \in \mathcal{K}$. Since $\mathcal{M}^{\mathcal{K}} \subseteq \mathcal{C}^{\mathcal{K}}$ it is easy to see that $z^*(\mathcal{C}^{\mathcal{K}}) \geq z^*(\mathcal{M}^{\mathcal{K}})$ and that the proximity of these two values gives an indication of the strength of MIR inequalities derived from that particular family \mathcal{K} .

In this paper we compute the values $z^*(\mathcal{C}^{\mathcal{F}})$ and $z^*(\mathcal{C}^{\mathcal{T}})$, where \mathcal{F} is the set of original formulation rows and \mathcal{T} is the set of first tableau rows. Note that MIR cuts obtained from rows of a simplex tableau are simply GMI cuts. In this study we are able to obtain results for a large subset of MIPLIB 3.0 and MIPLIB 2003 instances, including general mixed integer problems.

There have been several related papers computing the value of the continuous relaxation using all cuts in a certain class of knapsack cuts. Boyd [15], Yan and Boyd [51] and Kaparis and Letchford [41] compute $z^*(\mathcal{C}^{\mathcal{F}})$, where \mathcal{F} is the set of original formulation rows. They perform these tests on a subset of pure and mixed 0-1 instances in MIPLIB 3.0 [12]. Balas and Saxena [9] and Dash et al. [22] compute $z^*(\mathcal{M}^{\mathcal{A}})$ for all MIPLIB 3.0 problems, generalizing the results of Fischetti and Lodi [28] and Bonami et al. [14] where they consider Chvátal-Gomory cuts and projected Chvátal-Gomory cuts. Fischetti and Lodi [27] compute $z^*(\mathcal{C}^{\mathcal{A}})$, where \mathcal{A} is the set of all implied knapsack polyhedra, for a very reduced test set of pure 0-1 problems. We summarize these results and compare them to ours in Table 1.

Finally, note that, since our method establishes benchmarks for knapsack cuts, we use exact arithmetic to ensure that the results obtained are accurate. Indeed, Boyd [15] already cites numerical difficulties in P2756, a pure 0-1 instance, and as a consequence, he gets different bounds than us and Kaparis and Letchford [41]. We expect that more numerical difficulties would arise in mixed integer instances with general bounds and considering other implied knapsack sets besides

Paper	Instance set	Implied Knapsack Set (\mathcal{K})	Class of Cuts
Boyd [15]	Pure 0-1	Formulation	Knapsack
Yan and Boyd [51]	Mixed 0-1	Formulation	Knapsack
Fischetti and Lodi [28]	Pure integer	All	CG
Bonami et al. [14]	General MIP	All	Pro-CG
Fischetti and Lodi [27]	Pure 0-1	All	Knapsack
Balas and Saxena [9]	General MIP	All	MIR
Dash et al. [22]	General MIP	All	MIR
Kaparis and Letchford [41]	Pure 0-1	Formulation	Knapsack
This paper	General MIP	Formulation and Tableaus	Knapsack

Table 1: Computational studies of classes of cuts

the formulation rows.

The organization of this paper is as follows. In the next section, we discuss how to solve the problem of separating over a single mixed integer knapsack set. In Section 3 we describe our methodology for a practical implementation of the separation algorithm. Computational results are presented in Section 4, while final remarks are given in Section 5.

2 Identifying a violated knapsack cut

Consider $x^* \in \mathbb{Q}^n$ and a feasible mixed integer knapsack polyhedron K . In this section we are concerned with developing an effective algorithm for the following questions: Is $x^* \in \text{conv}(K)$? If not, can we find an inequality $\pi x \leq \pi_o$ which is valid for K , and such that $\pi x^* > \pi_o$?

Throughout this section we assume that K has no free variables, since it is easy to substitute a free variable by two non-negative variables. Further, we assume that the bound constraints are tight. That is, for every finite bound there exists a feasible solution which meets that bound at equality. We may assume this condition because if the bounds are not tight it is trivial to strengthen and make them so. Finally, we assume that $l_i < u_i$ for all $i \in 1, \dots, n$, as otherwise the problem is either badly defined, or in case of equality, the variable can be ignored and treated as a constant.

Let $\{x^1, x^2, \dots, x^g\}$ and $\{r^1, r^2, \dots, r^t\}$ represent the extreme points and rays of $\text{conv}(K)$. The following proposition, which follows from the work of Applegate et. al [3] suggests a natural algorithm for addressing our concern.

Proposition 1 *Consider the following linear programming (LP) problem with variables $u, v, \pi \in$*

\mathbb{R}^n , and $\pi_o \in \mathbb{R}$:

$$\begin{aligned}
LP_1 : \quad & \min \quad \sum_{i=1}^n (u_i + v_i) \\
& \text{s.t.} \\
& \pi x^k - \pi_o \leq 0 \quad \forall k = 1 \dots q \quad (C1) \\
& \pi r^k \leq 0 \quad \forall k = 1 \dots t \quad (C2) \\
& \pi x^* - \pi_o = 1 \quad (C3) \\
& \pi + u - v = 0 \quad (C4) \\
& u \geq 0, v \geq 0.
\end{aligned}$$

If this problem is infeasible, then $x^* \in \text{conv}(K)$, and thus there exists no knapsack cut violated by x^* . Otherwise, this problem admits an optimal solution (u, v, π, π_o) such that inequality $\pi x \leq \pi_o$ is a valid knapsack cut maximizing:

$$\frac{\pi x^* - \pi_o}{\|\pi\|_1}$$

That is, the hyperplane defined by (π, π_o) maximizes the L_1 distance to x^* .

Because LP_1 has an exponential number of rows, we use a dynamic constraint generation algorithm to solve the problem. We begin with constraints (C3) – (C4) and a subset of constraints (C1) – (C2). The cut generation algorithm requires solving the problem $\max\{\pi x : x \in K\}$ at each iteration. If this problem is unbounded at any given iteration, then there exists an extreme ray r^j of $\text{conv}(K)$ such that $\pi r^j > 0$. That is, there exists a violated constraint in (C2) which can be added. If this problem is not unbounded, then there exists an optimal solution corresponding to an extreme point x^k of $\text{conv}(K)$. If $\pi x^k > \pi_o$ then this extreme point results in a violated constraint of (C1) which can be added. Otherwise, it means that all constraints of the problem are satisfied. Because there can only be a finite number of extreme points and rays for a set K , this dynamic cut generation algorithm is assured to converge in a finite number of iterations.

In order to solve the sub-problem $\max\{\pi x : x \in K\}$ we use a specialized mixed integer knapsack solver developed for this application. Details of this solver are described in Section 3. This solver is designed in such a way that if the sub-problem is unbounded, it will return an extreme ray of $\text{conv}(K)$ which proves so. However, when the sub-problem is finite, the solver may return an optimal solution which is not an extreme point of $\text{conv}(K)$. Despite this fact, we can still ensure that the dynamic constraint generation algorithm converges. In fact, it is easy to see that if we use any finite set of points in K containing the extreme points of $\text{conv}(K)$, Proposition 1 still remains valid and the dynamic cut generation converges finitely. We will show in Section 3 that our solver can be made to return optimal solutions within a finitely sized set, thus ensuring the convergence of this procedure.

We remark that in practice, it is faster to solve the dual of LP_1 , since it has a fixed number of rows. This requires implementing this procedure as a column generation algorithm rather than as a dynamic constraint generation algorithm. Also, observe that if this algorithm finds a violated knapsack cut, then this cut will correspond to a non-empty face of $\text{conv}(K)$. This follows from the fact that the optimal solution to LP_1 will be basic and that we are minimizing the norm. For more details, see Espinoza [26].

The dynamic cut generation algorithm, as presented, is sufficient to solve LP_1 and allows us to obtain separating knapsack cuts, or show that none exist. Unfortunately, the algorithm as

presented is prohibitively slow. In Sections 2.1 - 2.4 we present several steps which help speed up the procedure. We summarize the procedure in Algorithm 1.

2.1 Determine if the problem admits a trivial solution.

In order to achieve this we test trivial sufficient conditions for the following questions:

1. Is $x^* \in \text{conv}(K)$? For this, define $N_K = \{i \in 1, \dots, n : a_i \neq 0\}$. We know that $x^* \in \text{conv}(K)$ if any of the following conditions are met:
 - $a_i \in \{-1, 1\}$ for all $i \in N_K$ and $b \in \mathbb{Z}$,
 - $N_K \cap I = \emptyset$,
 - $x_i^* \in \mathbb{Z}$ for all $i \in I \cap N_K$.

In fact, the first two conditions imply that $\text{conv}(K)$ is defined by the knapsack and bound constraints, and the third condition implies that x^* is feasible for K .

2. Is $x^* \notin \text{conv}(K)$? For this, attempt to find a violated MIR inequality. In our implementation we use the MIR separation heuristic described in [34]. If this heuristic is successful we can terminate the algorithm early and utilize the violated MIR inequality as a separating knapsack cut. Observe that one could also attempt to find violated inequalities from among other classes of cuts which are easy to separate, such as lifted cover inequalities.

2.2 Eliminate variables from LP_1 .

Say that a knapsack cut for K is *trivial* if it is implied by the linear programming relaxation of K . A proof of the following result concerning non-trivial knapsack cuts can be found in Atamtürk [5].

Proposition 2 *Every non-trivial facet-defining knapsack cut $\pi x \leq \pi_o$ of $\text{conv}(K)$ satisfies the following properties:*

- (i) *If $a_i > 0$, then $\pi_i \geq 0$*
- (ii) *If $a_i < 0$, then $\pi_i \leq 0$*
- (iii) *$\pi_i = 0$ for all $i \notin I$ such that $a_i > 0$ and $u_i = +\infty$.*
- (iv) *$\pi_i = 0$ for all $i \notin I$ such that $a_i < 0$ and $l_i = -\infty$.*
- (v) *There exists a constant $\alpha > 0$ such that $\pi_i = \alpha a_i$ for all $i \notin I$ such that $a_i > 0$ and $l_i = -\infty$, and for all $i \notin I$ such that $a_i < 0$ and $u_i = +\infty$.*

The following result concerning violated and non-trivial knapsack cuts is a simple generalization of a remark made in Boyd [15].

Proposition 3 *Consider $x^* \notin \text{conv}(K)$. Let $H^+ = \{i \in 1, \dots, n : a_i > 0, x_i^* = l_i\}$ and $H^- = \{i \in 1, \dots, n : a_i < 0, x_i^* = u_i\}$. If there does not exist a trivial inequality separating x^* from $\text{conv}(K)$, then there exists a separating knapsack cut $\pi x \leq \pi_o$ such that $\pi_i = 0, \forall i \in H^+ \cup H^-$.*

Proof: Since $x^* \notin \text{conv}(K)$, then there exists a valid inequality $\hat{\pi}x \leq \hat{\pi}_o$ for $\text{conv}(K)$ violated by x^* . We may assume that $\hat{\pi}x \leq \hat{\pi}_o$ is nontrivial, since if there is no such cut, we conclude the proof. Define $\pi_o = \hat{\pi}_o - \sum_{i \in H^+} \hat{\pi}_i l_i - \sum_{i \in H^-} \hat{\pi}_i u_i$ and

$$\pi_i = \begin{cases} 0 & \text{if } i \in H^+ \cup H^- \\ \hat{\pi}_i & \text{otherwise.} \end{cases}$$

Consider $x \in \text{conv}(P_I)$. Consider $i \in H^+$. From Proposition 2 we know that $a_i > 0$ implies $\hat{\pi}_i \geq 0$, and thus, $\hat{\pi}_i x_i \geq \hat{\pi}_i l_i$. Likewise, consider $i \in H^-$. From Proposition 2 we know that $a_i < 0$ implies $\hat{\pi}_i \leq 0$, and thus, $\hat{\pi}_i x_i \geq \hat{\pi}_i u_i$. Hence, $\hat{\pi}_o \geq \hat{\pi}x \geq \pi x + \sum_{i \in H^+} \hat{\pi}_i l_i + \sum_{i \in H^-} \hat{\pi}_i u_i$, and so, $\pi x \leq \pi_o$. On the other hand, $\hat{\pi}_o < \hat{\pi}x^* = \pi x^* + \sum_{i \in H^+} \hat{\pi}_i l_i + \sum_{i \in H^-} \hat{\pi}_i u_i$. Thus, $\pi x^* > \pi_o$, and we conclude the result. ■

From Proposition 2 and Proposition 3 we can see that variables π_i with $i \in 1, \dots, n$ can be assumed to have value zero whenever any of the following conditions are met:

- $i \notin I$, $a_i > 0$ and $u_i = +\infty$,
- $i \notin I$, $a_i < 0$ and $l_i = -\infty$,
- $a_i > 0$ and $x_i^* = l_i$,
- $a_i < 0$ and $x_i^* = u_i$.

In these cases the corresponding variables can simply be eliminated from LP_1 . Further, from Proposition 2 it can be seen that if we add a non-negative continuous variable α to LP_1 , we can replace all variables π_i satisfying the conditions of (v) by the corresponding terms αa_i . This can effectively reduce the variable space by eliminating all remaining continuous unbounded variables. Finally, observe that Proposition 2 allows us to add non-negativity bounds on all variables π_i such that $a_i > 0$, and non-positivity bounds on all variables π_i such that $a_i < 0$.

2.3 Fix variables and lift back again

Consider a mixed integer polyhedron P , such as the one defined in Section 1:

$$P = \{x \in \mathbb{R}^n : Dx \leq d, l \leq x \leq u, x_i \in \mathbb{Z}, \forall i \in I\}.$$

Let U, L be disjoint subsets of $\{1, \dots, n\}$ such that $u_i \neq \infty$ for all $i \in U$ and such that $l_i \neq -\infty$ for all $i \in L$. Define,

$$P(L, U) = P \cap \{x \in \mathbb{R}^n : x_i = l_i \forall i \in L\} \cap \{x \in \mathbb{R}^n : x_i = u_i \forall i \in U\}$$

Given an inequality $\pi x \leq \pi_o$ which is valid for $P(L, U)$, *lifting* consists in obtaining from this an inequality $\hat{\pi}x \leq \hat{\pi}_o$ which is valid for P .

Given a mixed integer knapsack set K and a point x^* we are concerned in determining if $x^* \in \text{conv}(K)$, or if we can find a separating hyperplane. For this we use lifting in the following way. Define $U = \{i \in 1, \dots, n : x_i^* = u_i\}$ and $L = \{i \in 1, \dots, n : x_i^* = l_i\}$. It is easy to see that $x^* \in \text{conv}(K)$ if and only if $x^* \in \text{conv}(K(L, U))$. However, this latter problem is much easier to

solve since we can treat all variables x_i with $i \in L \cup U$ as constants, thus obtaining a problem of smaller dimension. In practice, this means we can apply the dynamic cut generation methodology on the smaller knapsack constraint:

$$\sum_{i=1, i \notin L \cup U}^n a_i x_i \leq b - \sum_{i \in L} a_i l_i - \sum_{i \in U} a_i u_i.$$

If solving this problem reveals that $x^* \in \text{conv}(K(L, U))$ then we conclude that $x^* \in \text{conv}(K)$. However, if it reveals that $x^* \notin \text{conv}(K(L, U))$ we will obtain an inequality $\pi x \leq \pi_o$ separating x^* from $\text{conv}(K(L, U))$ but which is not necessarily valid for K . In this case we simply apply lifting to obtain a new inequality which is valid for K .

Lifting methodologies typically come in two flavors: Sequential lifting [50, 52, 10, 24] and sequence-independent lifting [37, 6]. In our study we used the sequential lifting procedure described in Fukasawa et al. [31], which allowed us to lift both continuous and general integer variables. The sequence in which we lifted the variables was completely arbitrary (from lowest to highest index).

2.4 Early termination rules

Recall that the dynamic cut generation algorithm described for solving LP_1 iterated as follows: At the beginning of each iteration it started with a solution (π, π_o) satisfying $\pi x^* = \pi_o + 1$. Then, the algorithm solved the sub-problem $\max\{\pi x : x \in K\}$ to optimality. The stopping condition of the dynamic cut generation algorithm was that the optimal answer to this sub-problem should be finite, and of value less than or equal to π_o . In what follows we present two different ways in which this scheme can be modified in such a way as to still guarantee convergence.

- **Method 1:** If, while solving $\max\{\pi x : x \in K\}$, we encounter a solution \hat{x} that is feasible for K and such that $\pi \hat{x} > \pi_o$, we can stop the execution of the sub-problem and add the constraint $\pi \hat{x} \leq \pi_o$ to LP_1 rather than continuing to solve the problem to optimality.
- **Method 2:** Let v^* be the optimal value of $\max\{\pi x : x \in K\}$. If $v^* > \pi_o$, then $\pi x \leq \pi_o$ is clearly not valid for $\text{conv}(K)$. However, if $\pi_o < v^* < \pi_o + 1$, then inequality $\pi x \leq v^*$ is both valid for $\text{conv}(K)$ and violated by x^* . In this case we may terminate the dynamic cut generation algorithm and conclude that $x^* \notin \text{conv}(K)$, providing the inequality (π, π_o) as a separating cut.

Observe that both methods cannot be used simultaneously, given that the latter requires solving the sub-problem to optimality.

3 Solving the Mixed Integer Knapsack Problem

Consider K as defined in Section 1, let $c \in \mathbb{Q}^n$, and assume l_i is finite for each $i \in 1, \dots, n$. In this section we are concerned with solving the Mixed Integer Knapsack Problem (MIKP),

$$\max\{cx : x \in K\} \tag{1}$$

We will assume that the problem is feasible, and are interested in either (a) proving that the problem is unbounded by finding an extreme ray r^* of $\text{conv}(K)$, or (b) computing the optimal value of the problem by finding the optimal solution $x^* \in K$.

Algorithm 1: KNAPSACK_SEPARATOR(K, x^*)

Input: A mixed integer knapsack set K , and a vector x^* .

Output: An answer to the question “Is $x^* \in \text{conv}(K)$?”. In case that the answer is FALSE, the algorithm also returns a separating cut.

- 1 Test if x^* and K verify any of the sufficient conditions described in Section 2.1 for showing that $x^* \in \text{conv}(K)$. If the answer is affirmative then STOP and report that $x^* \in \text{conv}(K)$;
 - 2 Run the MIR separation heuristic using K and x^* as input. If this heuristic finds a violated MIR cut then STOP, report that $x^* \notin \text{conv}(K)$, and return the MIR inequality which was found;
 - 3 Simplify the problem by fixing variables which are at their bounds, as indicated in Section 2.3, and reduce to a smaller dimensional knapsack separation problem;
 - 4 Formulate problem LP_1 in the reduced variable space without adding any of the constraints in (C1) – (C2);
 - 5 Apply Propositions 2 and 3 to simplify LP_1 by eliminating variables from consideration and adding bounds;
 - 6 Solve LP_1 using the dynamic row generation algorithm. Be sure to apply one of the early termination rules described in Section 2.4;
 - 7 If solving LP_1 reveals that $x^* \in \text{conv}(K)$, then STOP and report that $x^* \in \text{conv}(K)$;
 - 8 Let $(\hat{\pi}, \hat{\pi}_o)$ represent the optimal solution to LP_1 ;
 - 9 Lift cut $\hat{\pi}x \leq \hat{\pi}_o$ to obtain a cut $\pi x \leq \pi_o$ which is valid for K ;
 - 10 STOP and return the cut (π, π_o) ;
-

Variants of MIKP have long been studied in the research literature. In these it is typically assumed that all coefficients defining the problem are integer, that all variables must take integer values (i.e. no continuous variables are allowed), and that $l_i = 0$ for all $i = 1, \dots, n$. In addition: In the Knapsack Problem (KP) $u_i = 1$ for all $i = 1, \dots, n$, in the Bounded Knapsack Problem (BKP) $u_i < \infty$ for all $i = 1, \dots, n$, and in the Unbounded Knapsack Problem (UKP) $u_i = \infty$ for all $i = 1, \dots, n$. Most modern algorithms for solving KP, BKP, and UKP are based either on branch and bound (following the work of Horowitz and Sahni [38]) and on dynamic programming (following the work of Bellman [11]). However, the most efficient codes seldom make explicit use of Linear Programming and in addition, they never consider the use of both integer and continuous variables. For excellent surveys describing the rich literature on this topic, the reader is advised to consult Kellerer et al [42] and Martello and Toth [44].

While it is reasonable to expect that many of these algorithms could be adapted for solving our general case with a mix of continuous, integer, bounded and unbounded variables, the fact that they are designed to work with integer coefficients raises certain concerns with regards to the application discussed in this paper. In fact, part of our motivation is to study the efficacy of cuts derived from tableau rows. However, not only are these rows rarely made up of integer coefficients, but they also are typically very ill conditioned. Thus, scaling them so as to obtain integer coefficients may result in extremely large numbers. Considering this important shortcoming, and the need to further study these algorithms in order to account for the mixed use of bounded, unbounded, continuous and integer variables, our approach has been to pursue an LP-based branch and bound approach, which seems naturally suited to mixed integer programming problems. This issue, however, is one which merits further research. In what follows we describe our algorithm for solving MIKP.

3.1 Detecting unbounded solutions

For each $i \in 1, \dots, n$ define the *efficiency* of variable x_i as $e_i = c_i/a_i$ if $a_i \neq 0$, as $e_i = +\infty$ if $a_i = 0$ and $c_i > 0$, and as $e_i = -\infty$ if $a_i = 0$ and $c_i < 0$.

Proposition 4 *If MIKP is feasible, then it is unbounded if and only if one of the following conditions hold,*

(a) *There exists $i \in 1, \dots, n$ such that*

$$(a_i \leq 0, c_i > 0, u_i = +\infty)$$

(b) *There exist $i, j \in 1, \dots, n$ such that $e_i > e_j$,*

$$(a_i > 0, c_i > 0, u_i = +\infty) \text{ and } (a_j < 0, c_j \leq 0, u_j = +\infty).$$

Proof: It is clear that if either condition holds then the problem must be unbounded. We prove the converse. Let $U = \{i \in 1, \dots, n : u_i = +\infty\}$, and consider the recession cone of K , which is given by:

$$C = \{x \in \mathbb{R}^n : ax \leq 0, x_i \geq 0 \forall i \in U, x_i = 0 \forall i \notin U\}$$

The extreme rays of K coincide with the extreme rays of C . These must satisfy $n - 1$ of the linear independent constraints defining C at equality. Thus, all extreme ray vectors have at most two-nonzero values. Now assume that (a) does not hold, and let x correspond to an extreme ray

satisfying $cx > 0$. If x had a single non-zero we would have that $ax \leq 0$. However, this would be contradictory with the fact that (a) does not hold. Thus, we know that x has exactly two non-zeroes. Assume that these non-zeroes are x_i and x_j . Given that $cx > 0$ we may assume that $c_i > 0$. Note that this implies $a_i > 0$, since otherwise x_i would satisfy condition (a). Since $ax = 0$, it follows that $a_j < 0$. However, this in turn implies that $c_j < 0$, or again, x_j would satisfy condition (a). Finally, $cx > 0$ implies that $c_i x_i > -c_j x_j$, and $ax = 0$ implies that $a_i x_i = -a_j x_j$. From this we have that $e_i > e_j$, and so we conclude. ■

Observe that Proposition 4 implies that it can be determined if MIKP is unbounded in linear time. In fact, this can be achieved by looping through the unbounded variables and keeping track of the most efficient one satisfying $(a_i > 0, c_i > 0)$, the least efficient one satisfying $(a_j < 0, c_j \leq 0)$, and checking if any satisfy $(a_i \leq 0, c_i > 0)$.

3.2 Preprocessing

We utilize a simple eight-step pre-processing procedure in order to speed our algorithm. Let $N^+ = \{i \in 1, \dots, n : a_i > 0\}$, and $N^- = \{i \in 1, \dots, n : a_i < 0\}$. We assume that for each $i \in I$ we have $l_i \in \mathbb{Z}$ and $u_i \in \mathbb{Z} \cup \{\infty\}$. If there exists $i \in N^-$ such that $u_i = +\infty$, define $L_{min} = -\infty$. Otherwise, define,

$$L_{min} = \sum_{i \in N^-} a_i u_i + \sum_{i \in N^+} a_i l_i$$

If there exists $i \in N^+$ such that $u_i = +\infty$, define $L_{max} = +\infty$. Otherwise,

$$L_{max} = \sum_{i \in N^-} a_i l_i + \sum_{i \in N^+} a_i u_i$$

Finally, let $sign(x) = 1$ if $x > 0$, $sign(x) = -1$ if $x < 0$ and $sign(x) = 0$ if $x = 0$.

1. **Detect infeasibility.** The problem is infeasible if and only if one of the following conditions hold: (a) $L_{min} \neq -\infty$ and $L_{min} > b$, or (b) there exists $i \in 1, \dots, n$ such that $u_i \neq \infty$ and $l_i > u_i$.
2. **Detect unboundedness.** Use the algorithm discussed in Section 3.1. Observe that after this procedure, all variables x_i satisfying $c_i \geq 0$ and $a_i \leq 0$ will be such that u_i is finite.
3. **Fix variables at their bounds.** Fix to u_i all variables x_i such that $c_i \geq 0$ and $a_i \leq 0$. Fix to l_i to all variables x_i such that $c_i \leq 0$ and $a_i \geq 0$. Observe that after fixing, all variables will be such that $(a_i > 0$ and $c_i > 0)$ or $(a_i < 0$ and $c_i < 0)$.
4. **Tighten upper bounds.** If $L_{min} \neq -\infty$, then the for each $i \in N^+$ we can re-define $u_i := \min\{u_i, (b - L_{min} + a_i l_i)/a_i\}$. If $i \in I$ we can further strengthen this by letting $u_i := \lfloor u_i \rfloor$.
5. **Tighten lower bounds.** If $L_{min} \neq \infty$, then the for each $i \in N^-$ we can re-define $l_i := \max\{l_i, (b - L_{min} + a_i u_i)/a_i\}$. If $i \in I$ we can further strengthen this by letting $l_i := \lceil l_i \rceil$.
6. **Sort variables** Sort variables in order of increasing efficiency. Break ties as follows: Integer variables come before continuous variables. Break second ties in order of increasing a_i .

7. **Aggregate integer variables.** If two integer variables x_i, x_j are such that $a_i = a_j$ and $c_i = c_j$ aggregate them into a new variable x_k of the same type such that $a_k = a_i = a_j$, $c_k = c_i = c_j$, $l_k = l_i + l_j$ and $u_k = u_i + u_j$.
8. **Aggregate continuous variables.** If two continuous variables x_i, x_j are such that $\frac{c_i}{a_i} = \frac{c_j}{a_j}$ and $\text{sign}(a_i) = \text{sign}(a_j)$ aggregate them into a new variable x_k of the same type such that $a_k = a_i$, $c_k = c_i$, $l_k = l_i + \frac{a_j}{a_i}l_j$ and $u_k = u_i + \frac{a_j}{a_i}u_j$.

Note that steps one through four can be performed together in a single loop, and that step seven can be performed in a single pass after sorting. For more information and for possible extensions see Savelsbergh [47]. From this point on, we will assume that we are always dealing with instances that have been preprocessed according to the steps above.

3.3 Branch and bound

We use a depth-first-search branch and bound algorithm which always branches on the unique fractional variable. We use a simple linear programming algorithm, a variation of Dantzig's algorithm [20], which runs in linear time by taking advantage of the fact that variables are sorted by decreasing efficiency. We do not use any cutting planes in the algorithm, nor any heuristics to generate feasible solutions. The algorithm uses variable reduced-cost information to improve variable bounds at each node of the tree.

3.4 Domination

Consider x^1 and x^2 , two feasible solutions of MIKP. We say that x^1 *cost-dominates* x^2 if $cx^1 > cx^2$ and $ax^1 \leq ax^2$. On the other hand, we say that x^1 *lexicographically-dominates* x^2 if $cx^1 = cx^2$, $ax^1 \leq ax^2$, and if in addition there exists $i \in \{1, \dots, n\}$ such that $x_i^1 < x_i^2$ and $x_k^1 = x_k^2$ for all $k < i$. We say that a solution is *dominated* if it is cost-dominated or lexicographically-dominated. Observe that there exists a unique non-dominated optimal solution (or none at all).

Traditional branch and bound algorithms work by pruning nodes when (a) they are proven infeasible, or (b) when it can be shown that the optimal solution in those nodes has value worse than a bound previously obtained. In our implementation, we additionally prune nodes when (c) it can be shown that every optimal solution in those nodes is dominated.

Using dominance to improve the branch and bound search can have an important impact on the effectiveness of the search [39]. In fact, lexicographic and cost dominance allow us to disregard feasible solutions that are not the unique lexicographically smallest optimum solution, hence significantly reducing the search space.

In general, the problem of detecting if a solution is dominated can be extremely difficult. For example, Fischetti et al. [30, 29] detect domination by means of solving an MIP. In what follows we describe a simple methodology for identifying specific cases of domination arising in instances of the mixed integer knapsack problem. Note however, that the notion of dominance as defined above is much more general, and could be used for general mixed integer programming problems as well [30, 29, 34]. As a final comment, note that Andonov et. al [2] have also used domination in the context of dynamic programming algorithms, having had great success in tackling the unbounded knapsack problem.

Throughout this section we will convene that $\infty + r = \infty, \forall r \in \mathbb{R}$.

3.4.1 Domination between pairs of integer variables

Consider indices $i, j \in I$, and non-zero integers k_i, k_j . If $a_i k_i + a_j k_j \geq 0$ and $c_i k_i + c_j k_j < 0$ we say that (i, j, k_i, k_j) defines an integer cost-domination tuple. If $i < j$, $k_i > 0$, $a_i k_i + a_j k_j \geq 0$ and $c_i k_i + c_j k_j = 0$ we say that (i, j, k_i, k_j) defines an integer lexicographic-domination tuple. The propositions below show how domination tuples allow for the easy identification of dominated solutions.

Proposition 5 *Consider an integer cost/lexicographic-domination tuple (i, j, k_i, k_j) and let x be a feasible MIKP solution. Define $\delta \in \mathbb{Z}^n$ such that $\delta_i = k_i$, $\delta_j = k_j$ and $\delta_q = 0$ for all $q \in \{1, \dots, n\} \setminus \{i, j\}$. If*

$$l_i + k_i \leq x_i \leq u_i + k_i \text{ and } l_j + k_j \leq x_j \leq u_j + k_j \quad (2)$$

Then x is cost/lexicographically-dominated by $x - \delta$. Moreover, we say that x violates the domination tuple (i, j, k_i, k_j) .

To see that this proposition is true, it is simply a matter of observing that condition (2) implies that $(x - \delta)$ is a feasible solution in terms of the bounds, and that $a_i k_i + a_j k_j \geq 0$ and $c_i k_i + c_j k_j \leq 0$ imply the domination.

The following Theorem follows directly from Proposition 5, and illustrates how domination tuples can be used to strengthen a branch and bound algorithm. This is achieved by preventing nodes with dominated solutions from being created through additional enforced bound changes.

Theorem 6 *Consider two integer type variables x_i and x_j and a domination tuple (i, j, k_i, k_j) . If in some node of the branch and bound tree we have that $l_i + k_i \leq x_i \leq u_i + k_i$ is satisfied by all solutions in that node, then:*

- *If $k_j > 0$ we can impose the constraint $x_j \leq l_j + k_j - 1$ in that node,*
- *If $k_j < 0$ we can impose the constraint $x_j \geq u_j + k_j + 1$ in that node,*

and in either case we will not cut off the unique non-dominated optimal solution to the original MIP.

Observe that whenever (c_i, c_j) and (a_i, a_j) are linearly independent there exist an infinite number of cost-domination pairs. Likewise, there exist an infinite number of lexicographic-domination tuples in the linear dependent case.

The two following propositions state that, in each case, there always exists a *minimal* domination tuple. That is, a domination tuple (i, j, k_i, k_j) such that all other domination tuples (i, j, k'_i, k'_j) defined for the same variables, satisfy $|k_i| \leq |k'_i|$ and $|k_j| \leq |k'_j|$.

Proposition 7 *If (a_i, a_j) and (c_i, c_j) are linearly independent, let,*

$$\mathcal{D} = \{(k_i, k_j) \in \mathbb{Z} \times \mathbb{Z} : k_i, k_j \neq 0, a_i k_i + a_j k_j \geq 0 \text{ and } c_i k_i + c_j k_j < 0\}$$

The following two properties hold:

(i) \mathcal{D} is contained in the interior of an orthant in $\mathbb{N} \times \mathbb{N}$. More precisely, for every $(k_i, k_j) \in \mathcal{D}$ we have that: $\left(c_i - \frac{c_j}{a_j} a_i\right) < 0$ if and only if $k_i > 0$ and $\left(c_j - \frac{c_i}{a_i} a_j\right) < 0$ if and only if $k_j > 0$.

(ii) There exists (k_i^o, k_j^o) in \mathcal{D} such that (k_i, k_j) in \mathcal{D} implies $|k_i^o| \leq |k_i|$ and $|k_j^o| \leq |k_j|$.

Proof: Consider $(k_i, k_j) \in \mathcal{D}$. Observe that since $\text{sign}(a_i) = \text{sign}(c_i)$ we can multiply constraint $a_i k_i + a_j k_j \geq 0$ by $-\frac{c_j}{a_j}$ and add it to $c_i k_i + c_j k_j < 0$ and we get that $\left(c_i - \frac{c_j}{a_j} a_i\right) k_i < 0$. Likewise $\left(c_j - \frac{c_i}{a_i} a_j\right) k_j < 0$. Thus, (i) follows.

Consider (x_i, x_j) and (y_i, y_j) in \mathcal{D} . Let $z_i = x_i$ if $|x_i| \leq |y_i|$. Otherwise, let $z_i = y_i$. Let $z_j = x_j$ if $|x_j| \leq |y_j|$. Otherwise, let $z_j = y_j$. Since \mathcal{D} is contained in a pointed cone in one of the four orthants, it is easy to see that $(z_i, z_j) \in \mathcal{D}$. Thus there must exist (k_i^o, k_j^o) in \mathcal{D} satisfying the required condition. ■

Proposition 8 If (a_i, a_j) and (c_i, c_j) are linearly dependent, let,

$$\mathcal{D} = \{(k_i, k_j) \in \mathbb{N} \times \mathbb{Z} : k_i, k_j \neq 0, a_i k_i + a_j k_j = 0 \text{ and } c_i k_i + c_j k_j = 0\}$$

There exists (k_i^o, k_j^o) in \mathcal{D} such that (k_i, k_j) in \mathcal{D} implies $|k_i^o| \leq |k_i|$ and $|k_j^o| \leq |k_j|$.

Proof: Given the linear dependence of (a_i, a_j) and (c_i, c_j) we have that,

$$\mathcal{D} = \left\{ (k_i, k_j) \in \mathbb{N} \times \mathbb{Z} : k_i, k_j \neq 0, \frac{k_i}{k_j} = -\frac{a_j}{a_i} \right\}$$

That this set is non-empty follows from the fact that a_i and a_j are rational numbers. Let (k_i^o, k_j^o) be the point in \mathcal{D} with the smallest k_i^o . Clearly, $|k_j^o| = \left| \frac{a_i}{a_j} \right| |k_i^o| \leq \left| \frac{a_i}{a_j} \right| |k_i| = |k_j|$ for all $(k_i, k_j) \in \mathcal{D}$. ■

From Proposition 8 we see that if (a_i, a_j) and (c_i, c_j) are linearly dependent, then it is trivial to compute a minimal domination pair. From Proposition 7 it follows that if (a_i, a_j) and (c_i, c_j) are linearly independent, a minimal domination tuple can be obtained by solving the following two-dimensional integer programming problem:

$$\begin{aligned} & \text{minimize} && |k_i| + |k_j| \\ & \text{subject to,} && \\ & && c_i k_i + c_j k_j \geq 0 \\ & && a_i k_i + a_j k_j \leq 0 \\ & && |k_i| \geq 1, |k_j| \geq 1, \\ & && k_i, k_j \in \mathbb{Z} \end{aligned} \tag{3}$$

Further, given that we know the feasible region of Problem (3) is contained in an orthant of $\mathbb{Z} \times \mathbb{Z}$, we can remove the absolute values and make the problem linear.

While in theory Problem (3) can be solved in polynomial time [25, 48, 40], in practice we found it easier to solve the problem by simple enumeration. In order to make the description of

our procedure more intuitive, we only consider the case in which $c_j > 0$, $(c_i - \frac{c_j}{a_j} a_i) < 0$ and $(c_j - \frac{c_i}{a_i} a_j) < 0$. However, all other cases are strictly analogous.

Let \mathcal{D} describe the feasible region of Problem (3). Observe that under our current assumptions we have that \mathcal{D} is contained in the positive orthant. That is, $(k_i, k_j) \in \mathcal{D}$ implies $k_i \geq 0$ and $k_j \geq 0$. Consider a fixed value $k_i^* \in \mathbb{N}$. Define k_j^a, k_j^c so that $a_i k_i^* + a_j k_j^a = 0$ and $c_i k_i^* + c_j k_j^c = 0$. Under our current assumptions, it is easy to verify that $k_j^c \leq k_j^a$. Let $k_j^* = \lceil k_j^c \rceil$. If $k_i^* \geq 1$ and $k_j^* \leq k_j^a$, then $(k_i^*, k_j^*) \in \mathcal{D}$. Further, k_j^* is the smallest integer k_j such that $(k_i^*, k_j) \in \mathcal{D}$. On the other hand, if $k_i^* \geq 1$ and $k_j^* > k_j^a$ then there exists no integer k_j such that $(k_i^*, k_j) \in \mathcal{D}$. Our enumeration algorithm works by enumerating values of k_i^* starting from one, and for each of these computing the value k_j^* . The algorithm stops whenever $(k_i^*, k_j^*) \in \mathcal{D}$. Given that \mathcal{D} cannot be empty, it is easy to see that the algorithm must eventually terminate. Given the existence of a minimal domination tuple, we know that when the algorithm terminates it will do so with the correct solution. Note that in practice we terminate the enumeration early if it goes too long without finding a domination pair.

In order to use the above propositions in the branch and bound algorithm we compute what we call a *domination table* before initiating the solve. This table is defined as a list of all possible (minimal) domination tuples. In order to obtain a domination table we must loop through all possible pairs of variables. Observe that because we will only use these tables to strengthen bounds after branching, as in Theorem 6, we only need to store domination tuples (i, j, k_i, k_j) such that $|k_i| \leq (u_i - l_i)$ and $|k_j| \leq (u_j - l_j)$.

3.4.2 Domination between integer and continuous variables

Let $C_{j,k} = \{i \in C : j \leq i \leq k\}$. In this section we study how branching on integer variable x_i can be used to strengthen the bounds on continuous variables x_j with $j \in C_{1,i}$ or those with $j \in C_{i+1,n}$. Our attention will focus on instances of MIKP satisfying $a_i > 0$ for all $i \in 1, \dots, n$. This assumption is made only to simplify the notation and analysis. All results can easily be extended to the more general case in which coefficients can be positive or negative.

Proposition 9 *Consider a pre-processed instance of MIKP, $\max\{cx : x \in K\}$. For any $x \in K$ and $i \in I$ we have that x is dominated if the following condition holds:*

$$x_i \geq l_i + 1 \text{ and } \sum_{j \in C_{1,i}} a_j (u_j - x_j) \geq a_i \quad (4)$$

We have that x is cost-dominated if there exists $k \in C_{i,n}$ such that $e_k < e_i$, $l_k < x_k$, and,

$$x_i \leq u_i - 1 \text{ and } \sum_{j \in C_{i,n}} a_j (x_j - l_j) \geq a_i \quad (5)$$

Proof: Assume that $x_i \leq u_i - 1$ and $\sum_{j \in C_{i,n}} a_j (x_j - l_j) \geq a_i$. Define $\delta \in \mathbb{R}^n$ as follows: let $\delta_i = -1$, $\delta_j = 0 \forall j \in I \setminus \{i\}$, and $\delta_j = 0 \forall j \in C \setminus C_{i,n}$. For $j \in C_{i,n}$ define δ_j so that (i) $0 \leq \delta_j \leq x_j - l_j$, (ii) $\sum_{j \in C_{i,n}} a_j \delta_j = a_i$, and (iii) $\delta_k > 0$. Define $x' = x - \delta$. It is easy to see that x' satisfies the integrality constraints. Further, (i) implies that x' satisfies all of the bound constraints, and (ii) implies that $ax' = ax$. Thus, $x' \in K$. Finally, since the variables are sorted

by decreasing efficiency, we have that $cx' \geq cx$, and from (iii) we obtain that this inequality is strict. Hence, x' cost-dominates x . To prove that (4) implies x is dominated is analogous. In fact, let $\delta \in \mathbb{R}^n$ such that $\delta_i = 1$, $\delta_j = 0 \forall j \in I \setminus \{i\}$, and $\delta_j = 0 \forall j \in C \setminus C_{1,i}$. For $j \in C_{1,i}$ define δ_j so that (i) $0 \leq \delta_j \leq u_j - x_j$ and (ii) $\sum_{j \in C_{1,i}} a_j \delta_j = a_i$. As before, it is easy to prove that $x' = x - \delta$

dominates x . In this case we don't have to worry about the strict efficiency condition because $x'_i < x_i$, and all other integer constrained variables remain the same. Thus, if $cx' = cx$, that is, if x' does not cost-dominate x , we will have that x' lexicographically-dominates x . ■

Consider $i \in I$. Define,

$$f(i) = \min\{k \in C_{i,n} : \sum_{j \in C_{i,k}} a_j(u_j - l_j) > a_i\}$$

Likewise, define:

$$g(i) = \max\{k \in C_{1,i} : \sum_{j \in C_{k,i}} a_j(u_j - l_j) > a_i\}$$

Proposition 10 Consider a pre-processed instance of MIKP, $\max\{cx : x \in K\}$, and $i \in I$. Let $f = f(i)$ and $g = g(i)$. If in some node of the branch and bound tree we have that $x_i \leq u_i - 1$ is satisfied by all solutions in that node, and in addition, $e_f < e_i$, then we may also impose that:

$$x_j = l_j \quad \forall j \in C(f+1, n) \tag{6}$$

and,

$$x_f \leq l_f + \frac{a_i - \sum_{j \in C_{i,f-1}} a_j(u_j - l_j)}{a_f} \tag{7}$$

Likewise, if in some node of the branch and bound tree we have that $x_i \geq l_i + 1$ is satisfied by all solutions in that node, then we may also impose that:

$$x_j = u_j \quad \forall j \in C(1, g-1), \tag{8}$$

and,

$$x_g \geq u_g - \frac{a_i - \sum_{j \in C_{g,i}} a_j(u_j - l_j)}{a_g}. \tag{9}$$

In either case we will only be cutting off dominated solutions to the original MIP.

Proof: Suppose that $e_f < e_i$. If there exists a solution x such that $\sum_{j \in C_{i,n}} a_j(x_j - l_j) > a_i$ then, necessarily, there must exist $j \in C_{i,n}$ such that $e_j < e_i$ and $l_j < x_j$. This follows from the fact that $\sum_{j \in C_{i,f-1}} a_j(u_j - l_j) \leq a_i$. Now assume that we have imposed $x_i \leq u_i - 1$. Given the previous observation, if $\sum_{j \in C_{i,n}} a_j(x_j - l_j) > a_i$, then by Proposition 9 we will have that x is cost-dominated. Thus, in this case we can impose the constraint $\sum_{j \in C_{i,n}} a_j(x_j - l_j) \leq a_i$. However, any optimal solution to the problem satisfying this condition must also satisfy conditions (6) and (7).

In fact, suppose first that (6) does not hold. Without loss of generality assume that $x_k > l_k$, where $k \in C_{f+1,n}$. By the definition of f we have that $\sum_{j \in C_{i,f}} a_j(u_j - l_j) > a_i$. Hence, there must exist $j \in C_{i,f}$ such that $x_j < u_j$. However, this in turn means that there exists some $\varepsilon > 0$ such that we can subtract ε from x_k and add ε to x_j obtaining a new feasible solution. Given that the variables are sorted by decreasing efficiency, and observing that no two continuous variables can have the same efficiency, we conclude that in this case x would be cost-dominated, thus contradicting the optimality assumption.

Now suppose that (6) holds, but that (7) does not. We have that,

$$\begin{aligned}
a_i &\geq \sum_{j \in C_{i,n}} a_j(x_j - l_j) \\
&\geq \left(\sum_{j \in C_{i,f-1}} a_j(x_j - l_j) \right) + a_f(x_f - l_f) \\
&> \left(\sum_{j \in C_{i,f-1}} a_j(x_j - l_j) \right) + \left(a_i - \sum_{j \in C_{i,f-1}} a_j(u_j - l_j) \right) \\
&> \left(\sum_{j \in C_{i,f-1}} a_j(x_j - u_j) \right) + a_i
\end{aligned}$$

However, this implies that $x_j < u_j$ for some $j \in C_{i,f-1}$, and we can conclude by the same cost-dominance argument used in the previous case. The second part of the Proposition is analogous. ■

3.5 Finiteness of Branch-and-Bound

In the following, we prove that the use of integer-domination branching, as described in Theorem 6, ensures that the branch-and-bound algorithm will always terminate in a finite number of iterations. Note that in principle, one can always make an unbounded problem bounded [46, 49] and achieve this same effect, but this requires imposing bounds which are exponentially large.

Observe that we can describe the nodes of a branch-and-bound tree in terms of the bounds which have been imposed in that node. Thus, if B is a node of a branch-and-bound tree, we will write $B = (l', u')$ and identify the corresponding LP relaxation region at that node as,

$$LP(B) := \{x \in \mathbb{R}^n : ax \leq b, l' \leq x \leq u'\}$$

If a branch-and-bound tree has an infinite number of nodes, then it must clearly contain a path $P = B_1, B_2, \dots$ of infinite length. The following Proposition establishes necessary conditions that such an infinite path must satisfy.

Proposition 11 *Consider a branch and bound tree, and a path $P = B_1, B_2, \dots$ of infinite length contained in this tree. There exists a pair of distinct indices $i, j \in 1, \dots, n$ such that for every integer t , there exists a node $B_k = (l^k, u^k)$ satisfying $l_i^k > t$ and $l_j^k > t$.*

Proof:

Because we are assuming all lower bounds are finite, it is easy to see that there exists at least one index $i \in I$ such that the property holds. That is, such that for every integer t there exists a positive integer k such that $l_i^k > t$. Now assume that for all $j \in I \setminus \{i\}$ the property does *not* hold. That is, there exists t such that for all $j \in I \setminus \{i\}$ and for all positive integers k , we have that $l_j^k \leq t$. This implies that there exists a non-negative integer p such that for all $k \geq p$, node B_k is obtained from node B_{k-1} by branching up on variable x_i . Now consider $k \geq p$ and the fractional solution x^k obtained by solving over $LP(B_k)$. Without loss of generality, we can assume that x^k is an extreme-point solution of $LP(B_k)$. Since we will be branching on variable x_i , it is clear that $l_i^k < x_i^k < u_i^k$. Since x^k is a basic solution there exist n linearly independent constraints which are tight for x^k in $LP(B_k)$. Since none of these constraints are the bound constraints of x_i , and further, since none of the other bound constraints have changed since node B_p , we conclude that x^k is also an extreme point solution of $LP(B_p)$. Since for every pair of distinct indices k_1, k_2 such that $p \leq k_1$ and $p \leq k_2$ we have that $x^{k_1} \neq x^{k_2}$ we observe a contradiction, since $LP(B_p)$ cannot have an infinite number of distinct extreme point solutions. Thus a second index satisfying the desired condition must exist. ■

Before presenting the main result of this section, we prove a small technical proposition.

Proposition 12 *Consider a domination tuple (i, j, k_i, k_j) corresponding to a bounded pre-processed instance of MIKP. If $u_i = +\infty$ and $u_j = +\infty$, then $k_i > 0$ or $k_j > 0$.*

Proof: This clearly holds for lexicographic-domination tuples, thus henceforth we assume that (a_i, a_j) and (c_i, c_j) are linearly-independent. Assume that $k_i < 0$ and $k_j < 0$. Since $c_i k_i + c_j k_j < 0$ it follows that $sign(c_i) \neq sign(c_j)$. Without loss of generality, assume $c_i > 0$, which in turn implies that $a_i > 0$. Further, from Proposition 7, part (i), it follows that $c_i - (c_j/a_j)a_i > 0$. Dividing by a_i we get that $(c_i/a_i) > (c_j/a_j)$. However, by Proposition 4 this would imply that the problem is unbounded. ■

Theorem 13 *Consider an instance of the Mixed Integer Knapsack Problem which has been pre-processed as described in Section 3.2. If at every node of the branch-and-bound tree we impose the constraints described in Theorem 6, then the branch-and-bound tree has a finite number of nodes.*

Proof: Suppose that the branch-and-bound tree has an infinite number of nodes, and let $P = B_1, B_2, \dots$ be a path of infinite length contained in this tree. From Proposition 11 we know that there exist distinct indices $i, j \in I$ such that for every integer t , there exists a node $B_k = (l^k, u^k)$ satisfying $l_i^k > t$ and $l_j^k > t$. Observe that this necessarily implies that $u_i^k = u_j^k = +\infty$ for each of these nodes. Let (i, j, k_i, k_j) be the minimal domination tuple associated to these variables. From Proposition 12 we know that $k_i > 0$ or $k_j > 0$, so we may assume, without loss of generality, that $k_i > 0$. Let $t = \max\{l_i + k_i, l_j + k_j\}$ and let k_o be such that $l_j^{k_o} \geq t$. Since $u_j = +\infty$, it is easy to see that for all $k \geq k_o$ every solution $x \in B_k$ will satisfy $l_j + k_j \leq x_j \leq u_j + k_j$. Thus, in every node B_k with $k \geq k_o$ we will have imposed that $x_i \leq l_i + k_i - 1$. However, this is contradictory with the fact that the $l_i^k > t$. From this contradiction we conclude that the branch-and-bound tree must be finite. ■

We remark that it seems likely that the reasoning used to show that the branch-and-bound algorithm is finite can be used to show that the algorithm of Fischetti and Toth [30] is finite as well.

3.6 Post-Processing

Given a mixed integer knapsack polyhedron K , it is well known that $\text{conv}(K)$ has a finite number of extreme points and rays. For the reasons discussed in Section 2, a desirable property of a mixed integer knapsack solver is that it should always provide as an answer either an extreme point or an extreme ray, or when not possible, only finitely many other possible solutions. We wish to make sure that our solver satisfies this property.

Proposition 4 shows that if the mixed integer knapsack problem is unbounded, our solver will return an extreme ray, so from now on we only focus on the case where the problem is not unbounded. Although it is hard to guarantee that our solver will only return extreme points of $\text{conv}(K)$ if the optimal solution is finite, it is easy to see that the extreme points of $\text{conv}(K)$ are within the set of possible solutions that our solver can return. Indeed, for every extreme point x^k of $\text{conv}(K)$, there exists an objective function for which x^k is the unique optimal solution and therefore, any mixed integer knapsack solver contains all extreme points of $\text{conv}(K)$ as possible answers.

The question that remains is if our solver has the property that only finitely many other possible solutions can be returned. Espinoza [26] proposes that this can be achieved if the solution x^* returned by it satisfies the following two conditions:

1. $|x_i^*| \leq L$ for some constant L , for all $i \in I$;
2. x^* is an extreme point of $K_C(x^*) = K \cap \{x \in \mathbb{R}^n : x_i = x_i^*, \forall i \in I\}$.

In the remainder of this section we present a simple post-processing algorithm which, given an optimal solution, returns another (or possibly the same) optimal solution satisfying the conditions proposed by Espinoza.

The post-processing procedure is described in Algorithm 2, and consists of two phases: the first phase (lines 1–6) ensures that the integer variables have bounded L_∞ norm by using domination, and the second phase (line 7) ensures that the solution satisfies condition 2.

For the first phase, note that we can use minimal integer-domination tuples for any pair of variables, regardless if they are integral or not. In particular, Proposition 5 is true even if i and/or j are not integer variables. So, with \mathcal{L} being the set of all minimal lexicographic integer-domination tuples obtained by considering all variables to be integer, phase 1 ensures that no tuple in \mathcal{L} is violated by the returned optimal solution after the first phase. In the following proposition, we show that this fact implies that the integer variables have bounded L_∞ norm.

Proposition 14 *Consider an instance of MIKP $\max\{cx : x \in K\}$ which has been pre-processed as indicated in steps 1–3 of Section 3.2. Let x^* correspond to an optimal solution of this instance which has been post-processed as indicated in lines 1–6 of Algorithm 2. There exists a constant L depending only on a, b, l and u , such that $|x_i^*| \leq L$ for all $i \in N$.*

Proof:

Consider an instance of MIKP and an optimal solution x^* obtained from the post-processing algorithm. Let $M_1 = \max\{(u_i - l_i) : u_i \neq \infty \text{ and } i \in I\}$. Let $M_2 = \max\{\max\{|k_i|, |k_j|\} : (i, j, k_i, k_j) \in \mathcal{L}\}$. Let $M = \max\{M_1, M_2\}$.

Observe, first, that at most one variable x_i can be such that $x_i > l_i + M$. In fact, suppose that there are two variables x_i, x_j such that $x_i > l_i + M$ and $x_j > l_j + M$. From Proposition 7 and Proposition 8 we know that there exists a minimal domination pair (i, j, k_i, k_j) . Define

$\delta = k_i e_i + k_j e_j \in \mathbb{Z}^n$. Since necessarily $u_i = u_j = \infty$ and $M \geq |k_i|$, $M \geq |k_j|$ it follows that x^* satisfies $l_i + k_i \leq x_i^* \leq u_i + k_i$ and $l_j + k_j \leq x_j^* \leq u_j + k_j$ and thus violates (i, j, k_i, k_j) . So, from Proposition 5 we know that x^* is dominated by $x^* - \delta$. Given that x^* is optimal, we know that x^* cannot be cost-dominated. Thus $(i, j, k_i, k_j) \in \mathcal{L}$, which contradicts the fact that after post-processing there is no violated tuple in \mathcal{L} .

Now let x_i be the single variable such that $x_i^* > l_i + M$. If $a_i > 0$, then $x_i^* \leq \frac{b - \sum_{j \neq i} a_j x_j^*}{a_i}$. Since for all $j \neq i$, we have that $x_j^* \leq l_j + M$, it is clear that x_i^* can be bounded by a constant α_i depending only on a, b, l, u and M . If $a_i < 0$, then let $\alpha_i = \max \left\{ 0, \frac{b - \sum_{j \neq i} a_j x_j^*}{a_i} \right\}$. If $x_i^* > \alpha_i$, then we can define a new solution x' by letting $x'_j = x_j^*, \forall j \neq i$ and $x'_i = \alpha_i$. Clearly x' is feasible, and since $c_i < 0$ we have that $cx' < cx^*$. Since x^* is optimal we obtain a contradiction. By letting $L = \max\{M, \alpha_i\}$ we obtain our result. ■

Algorithm 2: The Post-Processing Algorithm

Input: An instance of the mixed integer knapsack problem, an optimal solution x^* to this instance, and the set \mathcal{L} of all lexicographic-domination tuples associated to the instance obtained by considering all variables to be integer.

```

1 while  $\exists(i, j, k_i, k_j) \in \mathcal{L}$  that is violated by  $x^*$  do
2    $\delta_i \leftarrow k_i$ 
3    $\delta_j \leftarrow k_j$ 
4    $\delta_q \leftarrow 0, \forall q \neq i, j$ 
5   while  $x^* - \delta$  is feasible do
6      $x^* \leftarrow x^* - \delta$ 
7  $x^* \leftarrow$  optimal solution to  $\max\{cx : x \in K, x_i = x_i^*, \forall i \in I\}$ 

```

It is easy to see how step 7 ensures condition 2, since $K_C(x^*)$ is a polyhedron and the optimal solution to the LP in step 7 will be an extreme point of $K_C(x^*)$. Therefore, we have shown that, after applying Algorithm 2 all conditions proposed by Espinoza are satisfied.

Note that in the loop of lines 1–6, we are always getting a lexicographically smaller solution so it is easy to see that the loop terminates after a finite number of iterations.

As a final remark, we could use any other mixed-integer knapsack solver instead of ours, as long as we preprocess the instance and post-process the solution, and we would still satisfy the required conditions.

4 Computational experiments

In this section we describe the results of our computational experiments. All implementations were developed in the “C” and “C++” programming languages, using the Linux operating system (v2.4.27) on Intel Xeon dual-processor computers (2GB of RAM, at 2.66GHz). Since generating cuts which are invalid is a real point of concern, we use exact arithmetic rather than the floating

point arithmetic typically used in computer codes. Specifically, we used the exact LP solver of Applegate et al. [4] for solving LP_1 , and the GNU Multiple Precision (GMP) Arithmetic library [35] for implementing all other computations.

In order to test our results we use the MIPLIB 3.0 [12] and MIPLIB 2003 [1] data sets. Results are organized as follows. In Section 4.1 we utilize the methodology developed in Section 2 to benchmark the MIR inequalities. In Section 4.2 we analyze the performance of the mixed integer knapsack algorithm developed in Section 3.

4.1 Benchmarking the MIR inequalities

We now describe our experiments computing the values $z^*(\mathcal{C}^K)$ and $z^*(\mathcal{M}^K)$ and our use of these values to benchmark the performance of MIR inequalities, as detailed in Section 1. In these experiments we only consider the sets $\mathcal{K} = \mathcal{F}$, i.e., the family of knapsack sets induced by the original formulation rows, and $\mathcal{K} = \mathcal{T}$, i.e., the family of knapsack sets induced by the tableau rows of the optimal LP relaxation solution.

As a first step, we implemented the MIR separation heuristic described in Goycoolea [34]. This heuristic takes as input a mixed integer knapsack polyhedron K , a vector x^* , and attempts to find an MIR inequality which is valid for K and violated by x^* . In order to do this, the heuristic scales the knapsack set and complements variables in different ways attempting to obtain an MIR inequality each time by applying a rounding procedure. We henceforth refer to this heuristic as $\text{MIR_HEURISTIC}(K, x^*)$. Given a mixed integer programming problem and a set of implied mixed integer knapsack polyhedra we can use this heuristic to compute an approximation of $z^*(\mathcal{M}^K)$ as shown in Algorithm 3. This is only an approximation of the value $z^*(\mathcal{M}^K)$ because there could potentially be violated MIR inequalities for a polyhedron $K \in \mathcal{K}$ which MIR_HEURISTIC fails to find. A better approximation can probably be obtained by using the approach of Balas and Saxena [9] or the approach of Dash et al. [22] as a subroutine instead of using MIR_HEURISTIC .

Algorithm 3: Computing an approximation of $z^*(\mathcal{M}^K)$

Input: A vector $c \in \mathbb{R}^n$, a mixed integer polyhedron P , and a set of implied mixed integer knapsack polyhedra \mathcal{K} .

```

1 Let  $R$  be the linear programming relaxation of  $P$ ;
2 Let  $x^*$  be the optimal solution of  $\max\{cx : x \in R\}$ ;
3 Let  $\mathcal{C} = \emptyset$ ;
4 foreach  $K \in \mathcal{K}$  do
5   | if  $\text{MIR\_HEURISTIC}(K, x^*)$  finds a violated cut  $(\pi, \pi_0)$  then
6   |   | Add cut  $(\pi, \pi_0)$  to  $\mathcal{C}$ ;
7 if  $\mathcal{C} \neq \emptyset$  then
8   | Add cuts in  $\mathcal{C}$  to  $R$ ;
9   | Resolve  $\max\{cx : x \in R\}$  letting  $x^*$  correspond to the optimal solution;
10  | Go to line 3;
11 else
12  | Stop.
```

As a second step, we implemented the `KNAPSACK_SEPARATOR` algorithm described in Section 2 (for a summary see Algorithm 1). When running the algorithm we used the early termination rule “Method 2” as described in Section 2.4. Observe from Algorithm 1 that `KNAPSACK_SEPARATOR` needs an MIR generator as a subroutine. We use `MIR_HEURISTIC` for this purpose. The mixed integer knapsack solver which is also used as a subroutine was described in Section 3. Given a mixed integer programming problem and a set of implied mixed integer knapsack polyhedra we can use `KNAPSACK_SEPARATOR` to compute $z^*(\mathcal{C}^{\mathcal{K}})$ as shown in Algorithm 4.

Algorithm 4: Computing $z^*(\mathcal{C}^{\mathcal{K}})$

Input: A vector $c \in \mathbb{R}^n$, a mixed integer polyhedron P , and a set of implied mixed integer knapsack polyhedra \mathcal{K} .

```

1 Let  $R$  be the linear programming relaxation of  $P$ ;
2 Let  $x^*$  be the optimal solution of  $\max\{cx : x \in R\}$ ;
3 Let  $\mathcal{C} = \emptyset$ ;
4 foreach  $K \in \mathcal{K}$  do
5   | if KNAPSACK_SEPARATOR( $K, x^*$ ) finds a violated cut  $(\pi, \pi_o)$  then
6   |   | Add cut  $(\pi, \pi_o)$  to  $\mathcal{C}$ ;
7 if  $\mathcal{C} \neq \emptyset$  then
8   | Add cuts in  $\mathcal{C}$  to  $R$ ;
9   | Resolve  $\max\{cx : x \in R\}$  letting  $x^*$  correspond to the optimal solution;
10  | Go to line 3;
11 else
12  | Stop.
```

The following proposition and the fact that `KNAPSACK_SEPARATOR` returns a nonempty face of $\text{conv}(K)$ ensure the finiteness of Algorithm 4:

Proposition 15 *If each cut added in line 6 of Algorithm 4 defines a nonempty face of $\text{conv}(K)$, then algorithm 4 terminates in a finite number of iterations.*

Proof: Follows immediately since for every $K \in \mathcal{K}$, $\text{conv}(K)$ has only a finite number of nonempty faces. ■

Tables 4.1.1 and 4.1.2 present the results for $\mathcal{K} = \mathcal{F}$ and $\mathcal{K} = \mathcal{T}$ respectively. For each problem instance let z_{UB}^* represent the value of the optimal (or best known) solution and z_{LP}^* the LP relaxation value. For each set \mathcal{K} and each instance we compute the following performance measures:

ORIG-GAP: Performance of the original LP formulation. That is, the value of the LP relaxation gap:

$$\frac{z_{UB}^* - z_{LP}^*}{|z_{UB}^*|}.$$

MIR: Performance of MIR separation heuristic. That is, how much of the LP gap was closed by heuristic:

$$\frac{z_M^{\mathcal{K}} - z_{LP}^*}{z_{UB}^* - z_{LP}^*}.$$

KNAP: Performance of the knapsack cuts. That is, how much of the LP gap was closed by the knapsack cuts:

$$\frac{z^*(\mathcal{C}^{\mathcal{K}}) - z_{LP}^*}{z_{UB}^* - z_{LP}^*}.$$

MIR-REL: Relative performance of MIR separation heuristic. That is, how much of the LP gap closed by the knapsack cuts was closed by the MIR cuts:

$$\frac{z_M^{\mathcal{K}} - z_{LP}^*}{z^*(\mathcal{C}^{\mathcal{K}}) - z_{LP}^*}$$

4.1.1 Knapsack cuts derived from formulation rows

We now discuss the computational results obtained for $\mathcal{K} = \mathcal{F}$. Of the 92 instances in MIPLIB 3.0 and MIPLIB2003, we were able to compute the values $z_M^{\mathcal{K}}$ for 83 of them. The 9 problems which we were unable to solve were atlanta-ip, cap6000, dano3mip, harp2, momentum3, markshare1, markshare2, mkc and stp3d. Of the 83 problems solved, 4 of them were such that ORIG-GAP was equal to 0.0 (disctom, dsbmip, enigma, and noswot), and 53 of them were such that KNAP and MIR were both equal to 0.0. In Table 4.1.1 we present the results for the remaining 26 problems. That is, the only ones in which knapsack cuts improved the bound.

First, note that knapsack cuts alone can considerably close the remaining LP gap in some problems (column KNAP). In fact, in 11 problems out of the 26 problems in which knapsack cuts improved the gap, over 84% of the gap was closed, and in 15 out of 26 problems, over 50 % of the gap was closed. On average, the GAP closed by the knapsack cuts among these 26 instances is around 55%. Note, however, that in 53 instances knapsack cuts do nothing to improve the gap. If we consider the average GAP closed including these 53 instances, the average drops to 18.24%.

Second, consider the column MIR in which we can get an idea of how well the mixed integer rounding cut closure compares to the knapsack cut closure. Observe that of the 26 problems, in 21 of them, by using the MIR cuts alone, we close over 87% of the GAP closed by the knapsack cuts. This indicates that MIR inequalities are a very important subset of knapsack inequalities; at least for the instances considered. A natural question is the following: How much could we improve the value of MIR-PERF if we used an exact MIR separation algorithm as opposed to a heuristic? In an attempt to answer this question we fine-tuned the settings of the MIR heuristic for the problems p0033, p0548 and qnet1. In these, we managed to improve the value of MIR-PERF from 87.31% to 100%, from 62.76% to 63.66% and from 56.68% to 77.27% respectively. This indicates that the true value of MIR-REL might be much closer to 100% than suggested by the table.

4.1.2 Knapsack cuts derived from tableau rows

In order to obtain \mathcal{T} for a given problem instance, we would first solve the linear programming relaxation of the corresponding problem and let \mathcal{T} be the optimal basis tableau rows in the augmented variable space consisting of both structural and slack variables. When attempting to

Table 2: Benchmarks for Formulation Closure

Instance	ORIG-GAP	MIR	KNAP	MIR-REL
arki001	0.02%	12.99%	13.12%	99.01 %
fiber	61.55%	91.07%	93.82%	97.06 %
gen	0.16%	99.78%	99.78%	100 %
gesa2	1.18%	69.96%	71.03%	98.48 %
gesa3	0.56%	47.80%	49.33%	96.90 %
gt2	36.41%	92.56%	94.52%	97.93 %
l152lav	1.39%	0.01%	1.36%	0.41 %
lseu	25.48%	67.91%	76.09%	89.25 %
mitre	0.36%	89.14%	100.00%	89.14 %
mod008	5.23%	71.23%	89.21%	79.84 %
mod010	0.24%	18.34%	18.34%	100 %
msc98-ip	1.61%	0.16%	0.16%	100%
net12	91.94%	1.39%	1.39%	100%
nsrand-ipx	4.53%	9.93%	9.93%	100%
p0033	18.40%	76.33%	87.42%	87.31 %
p0201	9.72%	33.78%	33.78%	100 %
p0282	31.56%	94.08%	98.59%	95.42 %
p0548	96.37%	53.69%	84.34%	62.76 %
p2756	13.93%	44.46%	86.35%	51.49 %
qnet1	10.95%	50.48%	89.06%	56.68 %
qnet1_o	24.54%	84.32%	95.12%	88.65 %
rgn	40.63%	57.49	57.49%	100 %
roll3000	13.91%	55.28%	55.28%	100%
sp97ar	1.38%	1.70%	1.70%	100%
timtab1	96.25%	21.13%	21.13%	100%
timtab2	93.49%	12.79%	12.79%	100%

generate a cut from a tableau row our algorithms all worked in this augmented space. Whenever a cut was generated from a tableau row, we would first substitute out slack variables before adding it back to the LP. Note that slack variables were always assumed to be non-negative and continuous.

We now discuss the computational results obtained for $\mathcal{K} = \mathcal{T}$. Of the 92 instances in MIPLIB 3.0 and MIPLIB2003, we were able to compute the values z_M^T for 65 of them. Of these 65 problems, 4 of them were such that ORIG-GAP was equal to 0.0 (disctom, dsbmip, enigma, and noswot), and 9 of them were such that KNAP and MIR were both equal to 0.0 (glass, liu, msc98-ip, mzzv11, mzzv42z, net12, rd-rplusc-21, stein27 and stein45). In Table 4.1.1 we present the results for the remaining 52 problems. That is, the only ones in which knapsack cuts improved the bound.

First, it is important to remark that separating knapsack cuts from tableau rows is considerable more difficult than separating knapsack cuts from original formulation rows. This is due to several reasons: Tableau rows are typically much more dense, coefficients tend to be numerically very bad, and rows tend to have a lot of continuous variables. This added difficulty is reflected in the fact that out of 92 instances, after several days of runs we managed to solve 65 instances to completion,

Table 3: Benchmarks for Tableau Closure

Instance	ORIG-GAP	MIR	KNAP	MIR-REL
air03	0.38%	100.00%	100.00%	100%
alc1s1	91.33%	18.75%	18.75%	100%
aflow30a	15.10%	12.02%	12.02%	100%
aflow40b	13.90%	7.45%	7.45%	100%
bell3a	1.80%	60.15%	60.15%	100%
bell5	3.99%	14.53%	14.68%	98.94%
blend2	8.99%	20.63%	20.63%	100%
danoint	4.62%	1.74%	1.74%	100%
dcmulti	2.24%	50.46%	50.49%	99.94%
egout	73.67%	55.33%	55.33%	100%
fiber	61.55%	75.89%	77.27%	98.21%
fixnet6	69.85%	11.08%	11.08%	100%
flugpl	2.86%	11.74%	11.74%	100%
gen	0.16%	61.67%	61.97%	99.52%
gesa2	1.18%	28.13%	28.13%	99.98%
gesa2.o	1.18%	29.55%	29.65%	99.67%
gesa3	0.56%	45.76%	45.83%	99.85%
gesa3.o	0.56%	49.96%	49.99%	99.94%
gt2	36.41%	84.56%	90.02%	93.93%
khb05250	10.31%	75.14%	75.14%	100%
lseu	25.48%	61.21%	61.21%	100%
manna81	1.01%	30.08%	30.08%	100%
misc03	43.15%	7.24%	7.24%	100%
misc06	0.07%	26.98%	26.98%	100%
misc07	49.64%	0.72%	0.72%	100%
mod008	5.23%	22.57%	22.59%	99.92%
mod011	13.86%	22.17%	22.17%	100%
modglob	1.49%	18.05%	18.05%	100%
momentum2	41.32%	40.13%	40.13%	100%
nsrand-ipx	4.53%	23.34%	23.34%	100%
nw04	3.27%	22.37%	22.37%	100%
opt1217	25.13%	23.09%	23.09%	100%
p0033	18.40%	74.71%	74.71%	100%
p0201	9.72%	34.36%	34.36%	100%
p0282	31.56%	9.21%	12.48%	73.79%
p0548	96.37%	70.97%	92.74%	76.52%
p2756	13.93%	22.38%	45.42%	49.28%
pp08a	62.61%	50.97%	50.97%	100%
pp08aCUTS	25.43%	32.30%	32.54%	99.26%
protfold	35.35%	3.79%	3.79%	100%
qiu	601.15%	3.47%	3.47%	100%
rentacar	5.11%	27.42%	27.42%	100%
rgn	40.63%	9.78%	9.78%	100%
roll3000	13.91%	0.16%	0.16%	100%
set1ch	41.31%	39.18%	39.18%	100%
sp97ar	1.38%	4.23%	4.23%	100%
swath	28.44%	33.22%	33.58%	98.91%
timtab1	96.25%	23.59%	23.59%	100%
timtab2	93.49%	11.91%	11.91%	100%
tr12-30	89.12%	28.47%	28.47%	100%
vpm1	22.92%	47.27%	49.09%	96.30%
vpm2	28.08%	19.17%	19.39%	98.85%

as opposed to the 83 which we solved when considering formulation rows.

Second, it is interesting to note that the value KNAP is very erratic, uniformly ranging in values from 100% to 0.0%. In contrast to the case of formulation rows, only 9 instances are such that KNAP-PERF is 0.0%.

The last, and perhaps most startling observation, is that the MIR-REL is very often close to 100%. If this result were true in general, it would be very surprising. However, because there are

still 27 instances which have not been solved one must be very careful. Because of the way in which we computed these numbers, it could be the case that those instances with MIR-REL close to 100% are easier for our methodology to solve. It is very reasonable to expect that instances with MIR-REL well below 100% are more difficult to solve as they require more iterations of the knapsack separation algorithm as opposed to iterations of the MIR separation heuristic.

4.2 Performance of the mixed integer knapsack solver

We now compare the performance of our MIKP algorithm (*kbb*) with the performance of CPLEX 9.0 [18] (*cpx*), the only alternative for MIKP we know of to date. Note that CPLEX was used with all of its default settings, except for the tolerance, which was set to 10^{-6} . Note also that our MIKP algorithm was used with double floating arithmetic, with a tolerance of 10^{-6} . Finally, note that we also tested a hybrid algorithm (*cpp*) which combined the *kbb* pre-processor (as in Section 3.2) with the CPLEX solver.

In our first implementation of the separation algorithm we had incorporated a version of *kbb* which did not use domination branching nor reduced cost bound improvements. We quickly realized that this algorithm was not efficient enough to achieve our aim of separating knapsack cuts derived from MIPLIB instances. When running this version of the code, we saved all problems which took our algorithm more than 2.5 seconds to solve. This allowed us to save over 130,000 instances. Most of those instances were very similar to each other, since many would originate from separating over a same implied knapsack set. When a single implied knapsack set had more than a knapsack instance associate to it, we eliminated all of these but the one which was generated last. This choice was motivated by our observation that the last few oracle calls were usually the hardest ones to solve. This allowed us to reduce the number of instances to only 288. Most of these instances were poised in exact arithmetic. Thus, after converting them to double floating point, some of them became very easy to solve. Hence, from these 288, we removed all instances in which *kbb*, *cpx* and *cpp* all took less than 1 second to solve in double arithmetic. This left us with a total of 67 instances, which are the ones used in the coming experiments. Note that these instances can be downloaded (both in double and in exact arithmetic) from <http://mgoycool.uai.cl/>.

In Figure 4.2 we present the performance profiles of running times of *kbb* and *cpx*. Each point (x, y) of the curves tells us that in y percent of the instances, this particular solver was at most x times slower than the fastest solver for that instance (for a more detailed explanation of performance profiles, see [23]). For instance, by looking at the point where $x = 1$ we see that *kbb* is the fastest in about 77% of the instances and by looking at the point where $x = 10$, it is at most 10 times slower than either *cpx* or *cpp* in over 85% of the instances. Also, note that giving the preprocessed instance to CPLEX does not seem to make a big difference. This can be explained since CPLEX has its own preprocessing, which probably makes most of our preprocessing unnecessary.

It is clear from this Figure that the *kbb* algorithm outperforms *cpx* in this instance set. Note that this does not necessarily mean that *kbb* solves every instance faster than *cpx*, but rather, that cumulatively, *kbb* performs better. Moreover, *cpx/cpp* fails to find the optimum solution in 11/8 instances, since it runs out of memory after creating too large a branch and bound tree.

In Figure 4.2 we compare the different versions of our mixed integer knapsack solver:

- **kbo**: Without domination branching and without reduced-cost bound improvement.
- **kbr**: With reduced-cost bound improvement but without domination branching.

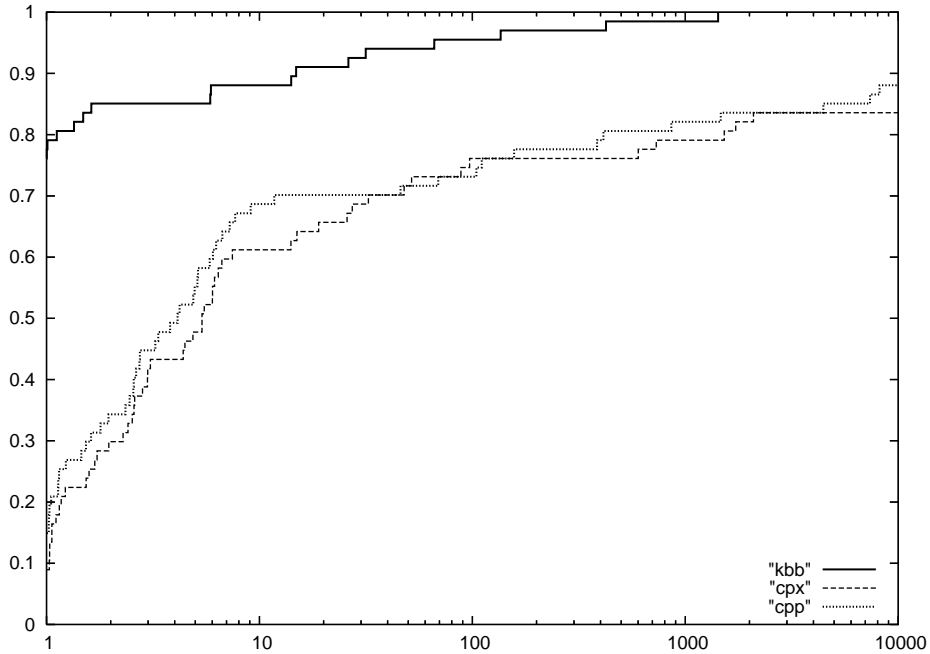


Figure 1: Performance profile comparing kbb, cpx and cpp.

- **kbd**: With domination branching but without reduced-cost bound improvement
- **kbb**: With both domination branching and reduced-cost bound improvement

It is interesting to see in Figure 4.2 how domination branching and reduced-cost fixing interact to improve the performance of the algorithm. Domination branching, when used as a single feature, clearly helps decrease solution times. This is not the case of reduced-cost fixing, however, which actually makes the algorithm perform worse. What is very surprising, however, is that when both features are used together, the performance is altogether markedly improved.

We believe this is due to the fact that we try to improve variable bounds by means of reduced cost in every node of the branch-and-bound tree, but these bound improvements do not reduce the size of the tree, which is the most important factor in determining the total running time of the algorithm. In fact, in all instances, the size of the branch-and-bound tree for *kbo* and *kbr* were exactly the same. Therefore, the additional time spent trying to improve bounds by reduced cost does not pay off. However, the use of domination reduces the size of the branch-and-bound tree and thus the total time. Now, since we use domination whenever certain bounds on variables are imposed, we believe the reduced cost bound improvements allow us to use domination more often, reducing even further the size of the branch-and-bound tree.

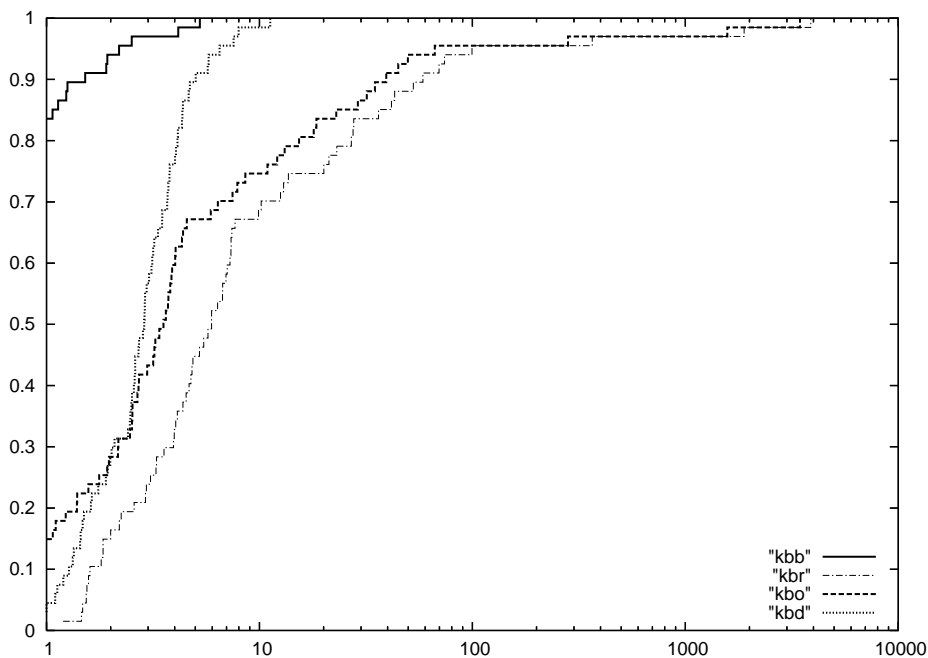


Figure 2: Performance profile comparing kbb, kbd, kbr, and kbo.

5 Final remarks

One of the goals of this study has been to assess the overall effectiveness of MIR inequalities relative to knapsack cuts. The motivation being the empirical observation that though much research has been conducted studying inequalities derived from single row systems, no such class of inequalities has been able to systematically improve upon the performance of MIRs. In this regard, the results we present are surprising.

We observe that in most test problems, the bound obtained by using just MIR inequalities is very similar in value (if not equal) to the bound obtained using all possible knapsack cuts. Though it is important to note that this observation is limited in the number of test problems considered, it does help explain the lack of success in generating other cuts from tableau and formulation rows, and, suggests that for further bound improvements we might have to consider new row aggregation schemes, or cuts derived from multiple row systems. At the very least, if there does exist some class of knapsack cuts which can significantly improve upon the performance of the MIR, our results show that to validate such a claim one would have to test using problems other than those in the MIPLIB data sets.

We put great care into ensuring that the generated cuts are valid and that the procedure runs correctly, but this makes the methodology very slow. For example, some of the computed KNAP values took as much as 5 days to obtain. Some of the unsolved instances have been ran for over a week without a final answer being reported. Part of the difficulty arises from the fact that exact

arithmetic is being employed. In average, we have observed that performing exact arithmetic computations takes several orders of magnitudes longer than floating-point computations.

As a final remark, we note that recently, knapsack cuts have been used in practice to help solve some combinatorial optimization problems [7]. Following up on this, it would be interesting to see if we can also use our knapsack cut generation procedure in practice to help solve other mixed-integer programming problems.

Acknowledgements The authors would like to thank William J. Cook for his support and encouragement in doing this work. They are also grateful to Sanjeeb Dash and Oktay Günlük for hosting them at the IBM T.J. Watson Research Center as student interns during the summers of 2004 and 2006. The discussions and questions raised during these internships were key factors in motivating this research.

References

- [1] T. Achterberg, T. Koch, and A. Martin. MIPLIB 2003. *Operations Research Letters*, 34(4):1–12, 2006. See <http://miplib.zib.de>.
- [2] R. Andonov, V. Poirriez, and S. Rajopadhye. Unbounded knapsack problem: dynamic programming revisited. *European Journal of Operational Research*, 123:394–407, 2000.
- [3] D. Applegate, R. E. Bixby, V. Chvátal, and W. Cook. TSP cuts which do not conform to the template paradigm. In *Computational Combinatorial Optimization, Optimal or Provably Near-Optimal Solutions [based on a Spring School]*, pages 261–304, London, UK, 2001. Springer-Verlag GmbH.
- [4] D. Applegate, W. Cook, S. Dash, and D. Espinoza. Exact solutions to linear programming problems. *Submitted to Operations Research Letters*, 2006.
- [5] A. Atamtürk. On the facets of the mixed-integer knapsack polyhedron. *Mathematical Programming*, 98:145–175, 2003.
- [6] A. Atamtürk. Sequence independent lifting for mixed-integer programming. *Operations Research*, 52:487–490, 2004.
- [7] P. Avella, M. Boccia, and I. Vasilyev. A computational study of exact knapsack separation for the generalized assignment problem. *Technical report available at Optimization Online*, July 2006.
- [8] E. Balas and M. Perregaard. A precise correspondence between lift-and-project cuts, simple disjunctive cuts, and mixed integer Gomory cuts for 0-1 programming. *Mathematical Programming*, 94:221–245, 2003.
- [9] E. Balas and A. Saxena. Optimizing over the split closure. *Mathematical Programming*, To appear.

- [10] E. Balas and E. Zemel. Facets of knapsack polytope from minimal covers. *SIAM Journal on Applied Mathematics*, 34(1):119–148, 1978.
- [11] R. E. Bellman. *Dynamic Programming*. Princeton University Press, 1957.
- [12] R. E. Bixby, S. Ceria, C. M. McZeal, and M. W. P. Savelsbergh. An updated mixed integer programming library: MIPLIB 3.0. *Optima*, (58):12–15, June 1998.
- [13] R.E. Bixby, Z. Gu, E. Rothberg, and R. Wunderling. Mixed integer programming: a progress report. In *The sharpest cut: the impact of Manfred Padberg and his work. MPS/SIAM Series on Optimization*, pages 309–326, 2004.
- [14] P. Bonami, G. Cornuéjols, S. Dash, M. Fischetti, and A. Lodi. Projected Chvátal Gomory cuts for mixed integer linear programs. *Mathematical Programming - to appear*, 2005.
- [15] A. E. Boyd. Fenchel cutting planes for integer programs. *Operations Research*, 42:53–64, 1992.
- [16] W. Cook, R. Kannan, and A. Schrijver. Chvátal closures for mixed integer programming problems. *Mathematical Programming*, 47:155–174, 1990.
- [17] G. Cornuéjols, Y. Li, and D. Vanderbussche. K-cuts: A variation of Gomory mixed integer cuts from the LP tableau. *INFORMS Journal on Computing*, 15(4):385–396, Fal 2003.
- [18] CPLEX. <http://www.ilog.com/products/cplex>.
- [19] H. Crowder, E.L. Johnson, and M. Padberg. Solving large-scale zero-one linear-programming problems. *Operations Research*, 31(5):803–834, 1983.
- [20] G. B. Dantzig. Discrete variable extremum problems. *Operations Research*, 5(2):266–277, 1957.
- [21] S. Dash and O. Günlük. On the strength of gomory mixed-integer cuts as group cuts. *IBM research report RC23967*, 2006.
- [22] S. Dash, O. Günlük, and A. Lodi. On the MIR closure of polyhedra. In *IPCO, Lecture notes in computer science, M. Fischetti and D. Williamson (Eds.)*, volume 4513, pages 337–351, 2007.
- [23] E. Dolan and J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2001.
- [24] T. Easton and T. Gutierrez. Sequential and simultaneous uplifting of general integer variables. *Submitted to Mathematical Programming*, 2007.
- [25] F. Eisenbrand and S. Laue. A linear algorithm for integer programming in the plane. *Mathematical Programming*, 102(2):249–259, 2005.
- [26] D. G. Espinoza. *On Linear Programming, Integer Programming and Cutting Planes*. PhD thesis, School of Industrial and Systems Engineering, Georgia Institute of Technology, March 2006.

- [27] M. Fischetti and A. Lodi. On the knapsack closure of 0-1 integer linear problems. Presentation at 10th International Workshop on Combinatorial Optimization, Aussois (2006). Available at <http://www-id.imag.fr/IWC02006/slides/Fischetti.pdf>.
- [28] M. Fischetti and A. Lodi. Optimizing over the first Chvátal closure. *Mathematical Programming B*, 110(1):3–20, 2007.
- [29] M. Fischetti and D. Salvagnin. A local dominance procedure for mixed-integer linear programming.
- [30] M. Fischetti and P. Toth. A new dominance procedure for combinatorial optimization problems. *Operations Research Letters*, 7:181–187, 1988.
- [31] R. Fukasawa, M. Goycoolea, and T. Easton, 2007. Working paper.
- [32] R. E. Gomory. Early integer programming (reprinted). *Operations Research*, 50(1):78–81, Jan 2002.
- [33] R. E. Gomory and E.L. Johnson. Some continuous functions related to corner polyhedra I. *Mathematical Programming*, 3:23–85, 1972.
- [34] M. Goycoolea. *Cutting Planes for Large Mixed Integer Programming Models*. PhD thesis, Georgia Institute of Technology, 2006.
- [35] T. Granlund. The GNU multiple precision arithmetic library. Available on-line at <http://www.swox.com/gmp/>.
- [36] Z. Gu, G. L. Nemhauser, and M. W. P. Savelsbergh. Lifted cover inequalities for 0-1 integer programs: Computation. *INFORMS Journal on Computing*, 10:427–437, 1998.
- [37] Z. Gu, G. L. Nemhauser, and M. W. P. Savelsbergh. Sequence independent lifting in mixed integer programming. *Journal Of Combinatorial Optimization*, 4(1):109–129, 2000.
- [38] E. Horowitz and S. Sahni. Computing partitions with applications to the knapsack problem. *Journal of the ACM*, 21:277–292, 1974.
- [39] T. Ibaraki. The power of dominance relations in branch-and-bound algorithms. *Journal of the ACM*, 24:264–279, 1977.
- [40] R. Kannan. A polynomial algorithm for the two-variable integer programming problem. *Journal of the ACM*, 27:118–122, 1980.
- [41] K. Kaparis and A. Letchford. Separation algorithms for 0-1 knapsack polytopes. *Technical report available at Optimization Online*, June 2007.
- [42] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, Berlin, Germany, 2004.
- [43] H. Marchand and L.A. Wolsey. Aggregation and mixed integer rounding to solve MIPs. *Operations Research*, 49:363–371, 2001.

- [44] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. J. Wiley, New York, 1990.
- [45] G. L. Nemhauser and L. A. Wolsey. A recursive procedure for generating all cuts for 0-1 mixed integer programs. *Mathematical Programming*, 46:379–390, 1990.
- [46] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Discrete Mathematics and Optimization. Wiley-Interscience, 1999.
- [47] M.W.P. Savelsbergh. Preprocessing and probing for mixed integer programming problems. *ORSA Journal on Computing*, 6:445–454, 1994.
- [48] H. E. Scarf. Production sets with indivisibilities – part II: the case of two activities. *Econometrica*, 49:395–423, 1981.
- [49] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley-Interscience, 1986.
- [50] L.A. Wolsey. Facets and strong valid inequalities for integer programs. *Operations Research*, 24(2):367–372, 1976.
- [51] X. Q. Yan and E. A. Boyd. Cutting planes for mixed-integer knapsack polyhedra. *Mathematical Programming*, 81(2):257–262, 1998.
- [52] E. Zemel. Lifting the facets of zero-one polytopes. *Mathematical Programming*, 15(1):268–277, 1978.