Math. Program., Ser. B (2010) 124:69–91
DOI 10.1007/s10107-010-0359-5

FULL LENGTH PAPER

Separation algorithms for 0-1 knapsack polytopes

Konstantinos Kaparis · Adam N. Letchford

Received: 1 June 2008 / Accepted: 10 August 2009 / Published online: 8 May 2010
© Springer and Mathematical Programming Society 2010

Abstract Valid inequalities for 0-1 knapsack polytopes often prove useful when tackling hard 0-1 Linear Programming problems. To generate such inequalities, one needs *separation algorithms* for them, i.e., routines for detecting when they are violated. We present new exact and heuristic separation algorithms for several classes of inequalities, namely *lifted cover*, *extended cover*, *weight* and *lifted pack* inequalities. Moreover, we show how to improve a recent separation algorithm for the 0-1 knapsack polytope itself. Extensive computational results, on MIPLIB and OR Library instances, show the strengths and limitations of the inequalities and algorithms considered.

Keywords Integer programming · Knapsack problems · Cutting planes

Mathematics Subject Classification (2000) 90C10 · 90C27 · 90C57

1 Introduction

A *0-1 knapsack problem* (0-1 KP) is a problem of the form:

$$\max \{c^T x : a^T x \leq b, \ x \in \{0,1\}^n\},$$

where $c \in \mathbb{Z}^n_+$ is the vector of *profits*, $a \in \mathbb{Z}^n_+$ is the vector of *weights* and $b \in \mathbb{Z}_+$ is the *knapsack capacity*. It is well-known that the 0-1 KP is NP-hard, but can be solved in $O(nb)$ time by dynamic programming.

K. Kaparis
School of Mathematics, University of Southampton, Southampton SO17 1BJ, UK
e-mail: K.Kaparis@soton.ac.uk

A. N. Letchford (✉)
Department of Management Science, Lancaster University, Lancaster LA1 4YX, UK
e-mail: A.N.Letchford@lancaster.ac.uk

A *0-1 knapsack polytope* is the convex hull of feasible solutions to a 0-1 KP, i.e., a polytope of the form:

$$KP(a, b) = \text{conv}\{x \in \{0, 1\}^n : a^T x \leq b\}.$$

Such polytopes have been widely studied and many classes of valid inequalities are known, such as the *lifted cover inequalities* (LCIs) of Balas [3] and Wolsey [31], the *extended cover inequalities* (ECIs) of Balas [3], the *weight inequalities* (WIs) of Weismantel [29] and the *lifted pack inequalities* (LPIs) of Atamtürk [2].

One motive for studying these polytopes is that valid inequalities for them can be used as cutting planes for *general* 0-1 linear programs (0-1 LPs). The idea is to consider each individual constraint of a 0-1 LP as a 0-1 knapsack constraint (complementing variables if necessary to obtain non-negative coefficients) and derive cutting planes from the associated polytope [13,16,24].

The *separation problem* for 0-1 knapsack polytopes is this: given n, a, b and a vector $x^* \in [0, 1]^n$, either find an inequality that is valid for $KP(a, b)$ and violated by x^*, or prove that none exists. One can define the separation problem for a given class of inequalities (such as LCIs, ECIs, WIs or LPIs) analogously. Separation algorithms are needed if one wishes to use valid inequalities as cutting planes (see, e.g., Nemhauser and Wolsey [24]).

Several exact and heuristic separation algorithms for 0-1 knapsack polytopes already exist in the literature [7–10,13,16,19,24,29]. The purpose of the present paper is to review these algorithms and present several new ones. The structure of the paper is as follows:

- In Sect. 2, we present a brief literature review.
- In Sect. 3, we present two exact algorithms and a heuristic for ECIs, and then show how to convert them into heuristics for LCIs, which are more general.
- In Sect. 4, we present two exact algorithms and a heuristic for WIs, and then show how to convert them into heuristics for LPIs, which are more general.
- In Sect. 5, we examine an exact algorithm for the 0-1 knapsack polytope itself, due to Boccia [7], and show how to improve its performance.
- In Sect. 6, we give extensive computational results, conducted on MIPLIB instances [1] and OR Lib instances [5].
- Finally, concluding remarks are made in Sect. 7.

Throughout the paper, we use the notation $N = \{1, \ldots, n\}$ and $a_{\max} = \max_{j \in N} a_j$.

2 Literature review

2.1 Lifted cover inequalities

A set $C \subseteq N$ is called a *cover* if it satisfies $\sum_{j \in C} a_j > b$. Given any cover C, the *cover inequality* (CI) $\sum_{j \in C} x_j \leq |C| - 1$ is valid for $KP(a, b)$. The strongest CIs are obtained when the cover C is *minimal*, in the sense that no proper subset of C is also a cover.

Given a minimal cover C, there exists at least one facet-defining *lifted cover inequality* (LCI) of the following form [3,31]:

$$\sum_{j\in C} x_j + \sum_{j\in N\setminus C} \alpha_j x_j \le |C| - 1. \tag{1}$$

The *lifting coefficients* α_j are usually computed *sequentially*, i.e., one at a time. Zemel [32] showed that, given a fixed cover C and a fixed lifting sequence, one can compute all lifting coefficients in $O(n|C|)$ time. His algorithm can be made even faster using results in [4,18], which enable one to determine most of the lifting coefficients in $O(n + |C|\log|C|)$ time.

As pointed out by Balas [3], a fast but naive way of lifting a CI is as follows. We compute $a^* := \max_{j\in C} a_j$ and define the *extension* of C as $E(C) := C \cup \{j \in N\setminus C : a_j \ge a^*\}$. Then the *extended* cover inequality (ECI) $\sum_{j\in E(C)} x_j \le |C| - 1$ is valid, but not guaranteed to be facet-defining.

Van Roy and Wolsey [27] noted that one can derive more general LCIs of the form:

$$\sum_{j\in C\setminus D} x_j + \sum_{j\in N\setminus C} \alpha_j x_j + \sum_{j\in D} \beta_j x_j \le |C\setminus D| + \sum_{j\in D} \beta_j - 1,$$

where C is a cover and $D \subset C$. We will follow Gu et al. [16] in referring to the computation of the α and β coefficients as *up-lifting* and *down-lifting*, respectively, and in referring to the inequalities (1) as 'simple' LCIs. The general LCIs include, not only the simple LCIs, but also the so-called '$(1, k)$-configuration' inequalities of Padberg [25] as special cases. Gu et al. [16] remark that Zemel's algorithm can be adapted to perform down-lifting in $O(|C|n^3)$ time. In practice, lifting can usually be performed very quickly.

The separation problems for CIs, LCIs and simple LCIs are all *NP*-hard [14,17,23], and the same seems likely for ECIs. Crowder et al. [13] noted that the separation problem for CIs can itself be written as a 0-1 KP, of the following form:

$$\min \sum_{j\in N} (1 - x_j^*) y_j \tag{2}$$

$$\text{s.t.} \sum_{j\in N} a_j y_j > b \tag{3}$$

$$y_j \in \{0, 1\} \quad (j \in N). \tag{4}$$

A CI is violated if and only if the optimal solution y^* has a cost less than one, in which case setting $C = \{j \in N : y_j^* = 1\}$ yields the violated CI. To save time, Crowder et al. solved this 0-1 KP heuristically, simply inserting items into C in non-decreasing order of $(1 - x_j^*)/a_j$. They then converted any CI found into a simple LCI, to increase the violation.

Hoffman and Padberg [20] recommended using general LCIs rather than simple LCIs. They obtained the best results by setting $D = \{j \in C : x_j^* = 1\}$, and using the following lifting sequence: first up-lift the variables with $x_j^* > 0$, then down-lift

the variables in D, then up-lift the variables with $x_j^* = 0$. Gu et al. [16] used the same lifting sequence, but recommended an alternative greedy heuristic for building the cover: simply insert items into C in non-increasing order of x_j^*.

In Kaparis and Letchford [21], we pointed out a small drawback of the Hoffman-Padberg lifting scheme: if there exists a $k \in N \backslash C$ such that $x_k^* > 0$ and $a_k > b - \sum_{j \in D} a_j$, then the up-lifting coefficient of x_k will be undefined. To resolve this problem, one can iteratively delete items from D, in arbitrary order, until no such item k exists.

Finally, we mention the paper by Gabrel and Minoux [15], which gives an exact separation algorithm for ECIs, based on the solution of at least n 0-1 KPs. Since our own algorithms (Subsect. 3.1 and 3.2) are both simpler and faster, we do not go into details.

2.2 Lifted pack inequalities

A set $P \subset N$ is called a *pack* if $\sum_{j \in P} a_j \leq b$. Given any pack P, the *pack inequality* (PI) $\sum_{j \in P} a_j x_j \leq \sum_{j \in P} a_j$ is trivially valid for $KP(a, b)$. PIs are very weak, being dominated by the upper bounds $x_j \leq 1$ for all $j \in P$. Nevertheless, there exist non-dominated *lifted* pack inequalities (LPIs).

A simple form of LPIs, involving up-lifting only, was presented by Weismantel [29]. Given a pack P, let us define the *residual capacity* $r = b - \sum_{j \in P} a_j$. Note that, if there exists a $j \in N \backslash P$ such that $a_j > r$, then x_j can be up-lifted. Based on this observation, Weismantel proposed the following *weight inequalities* (WIs):

$$\sum_{j \in P} a_j x_j + \sum_{j \in N \backslash P} \max\{0, a_j - r\} x_j \leq \sum_{j \in P} a_j. \tag{5}$$

The WIs can be thought of as being obtained from PIs by a *simultaneous* up-lifting procedure. The up-lifting coefficients are, however, not best possible. Atamtürk [2] showed how to derive slightly stronger LPIs using superadditive functions. Some other related inequalities can be found in [29], but we do not go into details, for the sake of brevity.

Weismantel [29] also derived a pseudo-polynomial-time exact separation algorithm for WIs. The idea is to write the WI (5) in the following form:

$$\sum_{j \in P: a_j \leq r} a_j x_j + \sum_{j \in P: a_j > r} r x_j \leq b - r - \sum_{j \in N: a_j > r} (a_j - r) x_j. \tag{6}$$

For *fixed* r, the separation problem amounts to finding the set P that maximises the left-hand side of (6). This can be formulated as an equality-constrained 0-1 KP, which can be solved in pseudo-polynomial time. Doing this for each possible value of r yields the desired separation algorithm.

Helmberg and Weismantel [19] presented a fast separation heuristic for WIs, that simply inserts items into P in non-increasing order of x^* value. To our knowledge,

there are no other papers on separation for WIs, and none at all on separation for other kinds of LPIs.

2.3 Separation from the polytope itself

Finally, we turn our attention to separation algorithms for $KP(a, b)$ itself. To our knowledge, four such algorithms have been proposed: three due to Boyd [8–10] and one due to Boccia [7]. For the sake of brevity, we concentrate on the method of Boccia, which in our experience gives rather good results.

Boccia begins by using the idea of polarity (see, e.g., Nemhauser and Wolsey [24]), to formulate the separation problem itself as the following LP:

$$\max \sum_{j \in N} x_j^* \alpha_j \tag{7}$$

$$\text{s.t.} \sum_{j \in P} \alpha_j \leq 1 \ (\forall P \subset N : \sum_{j \in P} a_j \leq b) \tag{8}$$

$$\alpha_j \in [0, 1] \ (j \in N). \tag{9}$$

The constraints (8) ensure that the inequality $\alpha^T x \leq 1$ is valid for $KP(a, b)$. If the optimal solution α^* has a profit larger than 1, then the inequality $(\alpha^*)^T x \leq 1$ is violated by x^*.

Since the separation LP has an exponential number of constraints, Boccia solves it with a cutting plane method. To check if a constraint (8) is violated by a given $\alpha^* \in [0, 1]^n$, it suffices to solve the following 0-1 KP:

$$\max \sum_{j \in N} \alpha_j^* y_j \tag{10}$$

$$\text{s.t.} \sum_{j \in N} a_j y_j \leq b \tag{11}$$

$$y_j \in \{0, 1\} \quad (j \in N). \tag{12}$$

If the solution y^* to this 0-1 KP has a profit larger than 1, and we set $P = \{j \in N : y_j^* = 1\}$, then the inequality $\sum_{j \in P} \alpha_j \leq 1$ is violated by α^*.

As Boccia points out, this naive approach suffers from very long computing times, and significant problems with rounding errors. To address these problems, he proposes a three-stage procedure. In the first stage, variables that satisfy $x_j^* \in \{0, 1\}$ are temporarily eliminated from the problem, so that the separation LP can be solved in the subspace of the fractional variables. If a violated inequality is found, one proceeds to the second stage, in which the inequality is scaled and rounded to integers. Boccia does this by solving a small integer linear program. In the third and final stage, the violated inequality is lifted to make it valid and facet-inducing for $KP(a, b)$. To do this, Boccia solves a sequence of 0-1 KPs. As in Gu et al. [16], Boccia recommends down-lifting the variables with $x_j^* = 1$ before up-lifting the variables with $x_j^* = 0$.

3 On ECI and LCI separation

In this section, we present two exact algorithms and a heuristic for ECI separation, and then show how all three algorithms can be modified to yield heuristics for the separation of the (stronger and more general) LCIs.

3.1 An $O(n^2 b)$ exact algorithm for ECIs

We begin by presenting a fairly simple exact algorithm for ECI separation, that involves the solution of a sequence of 0-1 KPs.

Recall from Subsect. 2.1 that Crowder et al. [13] formulated the separation problem for CIs as the 0-1 KP (2)–(4). We will need the following lemma:

Lemma 1 *Let $C^* \subset N$ be the cover obtained by solving the 0-1 KP (2)–(4), and let $a^* := \max_{j \in C^*} a_j$. If the ECI with $C = C^*$ is not violated, then a violated ECI must satisfy $C \subset \{j \in N : a_j \leq a^*\}$.*

Proof The violation of an ECI with cover C is equal to the sum of two terms:

1. $\sum_{j \in C} x_j^* - |C| + 1$ (the violation of the CI with cover C)
2. $\sum_{j \in N \setminus C : a_j \geq a^*} x_j^*$.

By definition, C^* is the cover that maximises the first term. So, if we replace C^* with a different cover C, the first term cannot increase. Moreover, if $\max_{j \in C} a_j > a^*$, then the second term cannot increase either. □

If one could make the inequality in Lemma 1 strict, an iterative algorithm for ECI separation would immediately follow: solve the 0-1 KP (2)–(4), eliminate the items with $a_j \geq a^*$, and repeat until the total weight of the remaining items no longer exceeds the knapsack capacity. However, we were unable to determine whether the lemma is valid with strict inequality. Fortunately, we have the following refinement of Lemma 1:

Proposition 1 *Let C^* and a^* be defined as in Lemma 1, let $S^* = \{j \in C^* : a_j = a^*\}$ and let k^* be an item in S^* of minimum x^* value. If the ECI corresponding to C^* is not violated, yet there exists a violated ECI, then there exists a violated ECI such that $C \subset \{j \in N \setminus \{k^*\} : a_j \leq a^*\}$.*

Proof Suppose that the ECI corresponding to C^* is not violated, but the cover C yields a violated ECI. By Lemma 1, we can assume that $C \subset \{j \in N : a_j \leq a^*\}$. Now recall again the two components of the violation mentioned in the proof of Lemma 1. If S^* were a subset of C, then the second term would be no larger for C than it was for C^*, and the ECI would not be violated. Thus, S^* cannot be a subset of C. Since k^* is the item with smallest x^* value in S^*, we can assume that k^* does not belong to C. □

The following separation algorithm follows immediately from Proposition 1:

1. Set $\tilde{N} := \{j \in N : x_j^* > 0\}$.

2. Solve the 0-1 KP:

$$\min\left\{\sum_{j\in\tilde{N}}(1-x_j^*)y_j : \sum_{j\in\tilde{N}}a_j y_j > b,\ y\in\{0,1\}^{\tilde{N}}\right\}.$$

3. Let C^* be the resulting cover. If the corresponding ECI is violated, output it.

4. Compute a^* and k^* as above. Delete from $\tilde{N}$ the item k^* and all items in $\tilde{N}$ with weight larger than a^*.

5. If $\sum_{j\in\tilde{N}} a_j \le b$, stop. Otherwise return to 2.

This algorithm involves the solution of at most n 0-1 KPs of 'greater-than' type. We show in the next subsection that each such 0-1 KP can be solved in $O(nb)$ time. So, the algorithm runs in $O(n^2b)$ time. In practice, however, only a small number of 0-1 KPs, of rapidly decreasing size, typically need to be solved. They can often be solved quickly by branch-and-bound. Moreover, the algorithm can return more than one violated inequality.

3.2 An $O(nb)$ exact algorithm for ECIs

In this subsection, we present a faster exact algorithm for ECI separation. First, we show explicitly how to solve the separation problem for the weaker CIs [i.e., the 0-1 KP (2)–(4)] in $O(nb)$ time. Then, we will show how to modify the algorithm in order to solve the separation problem for the ECIs themselves in $O(nb)$ time.

For $k = 1,\ldots,n$ and $r = 0,\ldots,b$, define:

$$f(k,r) := \min\left\{\sum_{j=1}^{k}(1-x_j^*)y_j : \sum_{j=1}^{k}a_j y_j = r,\ y\in\{0,1\}^k\right\}.$$

Also, for $k = 1,\ldots,n$, define:

$$g(k) := \min\left\{\sum_{j=1}^{k}(1-x_j^*)y_j : \sum_{j=1}^{k}a_j y_j \ge b+1,\ y\in\{0,1\}^k\right\}.$$

The following dynamic programming algorithm computes all of the $f(k,r)$ and $g(k)$ values, and thereby solves the CI separation problem:

Set $f(k,r) := \infty$ for $k = 1,\ldots,n$ and $r = 0,\ldots,b$. Set $f(0,0) := 0$.
Set $g(k) := \infty$ for $k = 1,\ldots,n$.
For $k = 1,\ldots,n$
  For $r = 0,\ldots,b$
    If $f(k-1,r) < f(k,r)$
      Set $f(k,r) := f(k-1,r)$.

For $r = 0, \ldots, b - a_k$
  If $f(k-1, r) + (1 - x_k^*) < f(k, r + a_k)$
    Set $f(k, r + a_k) := f(k-1, r) + (1 - x_k^*)$.
For $r = b - a_k + 1, \ldots, b$
  If $f(k-1, r) + (1 - x_k^*) < g(k)$
    Set $g(k) := f(k-1, r) + (1 - x_k^*)$.
If $g(k) < 1$, output the violated CI.

It is easy to see that this algorithm runs in $O(nb)$ time.

Suppose now that the items in N have been sorted in non-decreasing order of a_j. Under this assumption, we have a violated ECI if and only if $g(k) - \sum_{j=k+1}^{n} x_j^* < 1$ for some $k \in N$. Thus, to convert the exact CI separation algorithm into an exact ECI separation algorithm, it suffices to make the following two minor changes:

(i) Sort the items in non-decreasing order of a_j before running the algorithm.

(ii) Change the last 'if' statement to:

If $g(k) - \sum_{j=k+1}^{n} x_j^* < 1$, output the violated ECI.

This leads to the following result:

Theorem 1 *The separation problem for ECIs can be solved exactly in $O(nb)$ time.*

Proof The initial sorting can be performed in $O(n + a_{\max})$ time, using bucket sort (see [12]). This time is dominated by the time taken to solve the dynamic program, $O(nb)$. □

We remark that this algorithm can return more than one violated inequality. Moreover, it can be made faster in practice by excluding variables with $x_j^* = 0$ from the DP, although, each time a violated ECI is found, one should check whether any such variables can be inserted into the extension $E(C)$.

3.3 A fast heuristic for ECIs

The exact algorithm for ECIs presented in Subsect. 3.1 can be easily converted into a fast heuristic for ECIs: one simply solves the subproblems in step 2 heuristically instead of exactly. Following Crowder et al. [13], we simply insert items into C in non-decreasing order of $(1 - x_j^*)/a_j$. This leads to the following algorithm:

1. Sort the items in N in non-decreasing order of $(1 - x_j^*)/a_j$, and store them in a list L. Initialise the cover C as the empty set and initialise $a^* = b$.
2. Remove an item from the head of the sorted list L. If its weight is larger than a^*, ignore it, otherwise insert it into C. If C is now a cover, go to step 4.
3. If L is empty, stop. Otherwise, return to step 2.
4. If the ECI corresponding to C is violated by x^*, output it.

5. Find k^*, the heaviest item in C, decrease a^* accordingly, and delete k^* from C. Return to step 2.

This heuristic can easily be implemented so that it runs in $O(n^2)$ time. Like the exact algorithms, it can return more than one violated inequality.

3.4 Heuristics for LCIs

Recall that ECIs are a special case of, and dominated by, the LCIs. Any of the algorithms for ECI separation presented in the previous three subsections can be easily converted into a heuristic for LCI separation. Namely, each time we generate an ECI, we convert it into an LCI that is violated by at least as much. This can be done by taking the cover C, setting $D = \{j \in C : x_j^* = 1\}$, and using the Hoffman-Padberg lifting order mentioned in Subsect. 2.1.

4 On WI and LPI separation

In this section, we present three algorithms for WI separation: an enhanced version of Weismantel's exact algorithm, an alternative exact algorithm with a reduced running time, and a new heuristic algorithm. We then show how all three algorithms can be modified to yield heuristics for the separation of other LPIs.

4.1 Enhancements to Weismantel's algorithm

The worst-case running time of Weismantel's exact algorithm for WI separation is rather high, as shown in the following proposition:

Proposition 2 *Weismantel's algorithm runs in $O(nba_{\max})$ time.*

Proof An equality-constrained 0-1 KP has to be solved for each possible value of the residual capacity $r = b - \sum_{j \in P} a_j$. We can assume that $0 < r < a_{\max}$, since the WI is redundant when either $r = 0$ or $r \geq a_{\max}$. Thus, only $a_{\max} - 1$ equality-constrained 0-1 KPs need to be solved. It takes $O(nb)$ time to solve one such 0-1 KP by dynamic programming. □

We have found three simple ways to speed up the algorithm, which make a dramatic difference in practice:

1. It can be shown that, if $x_k^* = 0$, then one can assume that $k \notin P$ without losing any violated WIs. Similarly, if $x_k^* = 1$, then one can assume that $k \in P$. This reduces the number of variables in the equality-constrained 0-1 KPs.
2. If, for a given value of r, no $P \subset N$ exists satisfying $\sum_{j \in P} a_j = b - r$, the equality-constrained 0-1 KP will be infeasible. To avoid wasting time solving infeasible problems, one can compute the possible values that $\sum_{j \in P} a_j$ can take. This can easily be done in $O(nb)$ time and $O(b)$ space, by dynamic programming.

3. If one solves the remaining equality-constrained 0-1 KPs via branch-and-bound, one can abort the branch-and-bound run as soon as the upper bound is less than or equal to the right-hand side of (6).

Another way to reduce running times is, of course, to run a separation heuristic first, and only call the exact algorithm when the heuristic fails.

4.2 An $O((n + a_{\max})b)$ exact algorithm for WIs

Improving on the running time bound of $O(nba_{\max})$ turns out to be a non-trivial exercise. We found it helpful to treat the two terms on the left-hand side of (6), and the right-hand side of (6), separately. Thus, we make the following three definitions.

Definition 1 For given integers r, s, with $0 < r < a_{\max}$ and $0 \leq s \leq b-r$, we define:

$$f(r,s) := \max \left\{ \sum_{j \in N: a_j \leq r} a_j x_j^* y_j : \sum_{j \in N: a_j \leq r} a_j y_j = s,\ y_j \in \{0,1\}\ (j \in N : a_j \leq r) \right\}.$$

Definition 2 For given integers r, s, with $0 < r < a_{\max}$ and $0 \leq s \leq b-r$, we define:

$$g(r,s) := \max \left\{ \sum_{j \in N: a_j > r} x_j^* y_j : \sum_{j \in N: a_j > r} a_j y_j = s,\ y_j \in \{0,1\}\ (j \in N : a_j > r) \right\}.$$

Definition 3 For a given integer r, with $0 < r < a_{\max}$, we define:

$$h(r) := b - r - \sum_{j \in N: a_j > r} (a_j - r) x_j^*.$$

Armed with these definitions, we propose the following exact separation algorithm for WIs:

1. Sort the items in N in non-decreasing order of weight.
2. Compute $f(r,s)$ for $r = 1, \ldots, a_{\max} - 1$ and for $s = 0, \ldots, b - r$.
3. Compute $g(r,s)$ for $r = 1, \ldots, a_{\max} - 1$ and for $s = 0, \ldots, b - r$.
4. Compute $h(r)$ for $r = 1, \ldots, a_{\max} - 1$.
5. For $r = 1, \ldots, a_{\max} - 1$, compute the maximum possible violation of a WI with $\sum_{j \in P} a_j = b - r$. That is, compute:

$$\max_{0 \leq s \leq b-r} f(r,s) + r g(r, b - r - s) - h(r).$$

If this quantity is positive, output the violated WI.

We have the following theorem.

Theorem 2 *The separation problem for WIs can be solved exactly in* $O((n + a_{\max})b)$ *time.*

Proof As noted in Subsect. 3.2, Step 1 can be performed in $O(n + a_{\max})$ time. Step 2 can be performed in $O(nb)$ time: just use the standard dynamic programming algorithm for the 0-1 KP, but introduce new items in non-decreasing order of weight. Similarly, Step 3 can be performed in $O(nb)$ time: use the standard dynamic programming algorithm for the 0-1 KP, but introduce new items in non-increasing order of weight. Step 4 can be performed in $O(na_{\max})$ time. Finally, step 5 takes $O(ba_{\max})$ time. The terms $O(n + a_{\max})$ and $O(na_{\max})$ are dominated by $O(nb)$. □

4.3 A simple new heuristic for WIs

In our implementation of the heuristic of Helmberg and Weismantel [19], the items are sorted in non-increasing order of x^* value, and then placed in a list. We then proceed through the list, iteratively inserting items into the pack P, until we encounter an item whose weight is greater than or equal to the residual capacity. We then stop and output the pack P obtained.

A small modification of this procedure enables one to construct additional packs: simply continue through the sorted list and, each time an item is encountered whose weight is less than the residual capacity, insert it into the pack and output the pack thus formed. In our experiments, this modified procedure typically produced two or three packs, rather than only one as in the Helmberg–Weismantel heuristic.

4.4 Heuristics for LPIs

Recall that WIs can be viewed as a special kind of LPI, derived by up-lifting simultaneously (Subsect. 2.2). We have found that, in practice, *sequential* lifting usually yields stronger inequalities. To derive sequentially-lifted LPIs, we simply take each pack P generated by one of the above-mentioned heuristics (the Helmberg–Weismantel heuristic or our modified version of it), and apply sequential lifting. Specifically, we set the down-lifting set D to $\{j \in P : x^*_j = 1\}$, and then follow the Hoffman-Padberg lifting order (see Subsect. 2.1).

LPIs differ from LCIs, however, in two respects. First, LPIs are not guaranteed to induce facets of $KP(a, b)$, even if the lifting coefficients are computed exactly. All we can say is that they induce faces of dimension at least $n - |P \backslash D|$. Second, computing lifting coefficients exactly is non-trivial. In fact, it can be easily shown (by reduction from the *subset-sum* problem [22]) that computing even a single up-lifting coefficient of an LPI is NP-hard.

On the positive side, we have discovered a dynamic programming algorithm that computes all lifting coefficients exactly in $O(nb)$ time. For the sake of brevity, we describe the algorithm for the case of up-lifting only.

Assume, without loss of generality, that $P = \{1, \ldots, p\}$, and we wish to compute up-lifting coefficients for $x_{p+1}, \ldots, x_n$, in that order. Let $\alpha_{p+1}, \ldots, \alpha_n$ denote these

up-lifting coefficients. For $k = 1, \ldots, p$ and $r = 0, \ldots, b$, let $f(k, r)$ denote the solution to the following 0-1 KP:

$$\max \left\{ \sum_{j=1}^{k} a_j y_j : \sum_{j=1}^{k} a_j y_j \leq r, \ y \in \{0,1\}^k \right\}.$$

Note that the first up-lifting coefficient, α_{p+1}, is equal to $f(p, b) - f(p, b - a_k)$. Now, for $k = p + 1, \ldots, n$ and $r = 0, \ldots, b$, let $f(k, r)$ denote instead the solution to the following 0-1 KP:

$$\max \left\{ \sum_{j=1}^{p} a_j y_j + \sum_{j=p+1}^{k} \alpha_j y_j : \sum_{j=1}^{k} a_j y_j \leq r, \ y \in \{0,1\}^k \right\}.$$

The kth up-lifting coefficient, α_k, is then equal to $f(k-1, b) - f(k-1, b-a_k)$.

The following dynamic programming algorithm computes the desired $f(k, r)$ and α_k values:

Initialise $f(0, r) := 0$ for $r = 0, \ldots, b$.
For $k = 1, \ldots, p$ do:
  For $r = 0, \ldots, b$ do:
    Set $f(k, r) := f(k-1, r)$.
  For $r = a_k, \ldots, b$ do:
    If $f(k-1, r-a_k) + a_k > f(k, r)$
      Set $f(k, r) := f(k-1, r-a_k) + a_k$.
For $k = p+1, \ldots, n$ do:
  Set $\alpha_k := f(k-1, b) - f(k-1, b-a_k)$.
  For $r = 0, \ldots, b$ do:
    Set $f(k, r) := f(k-1, r)$.
  For $r = a_k, \ldots, b$ do:
    If $f(k-1, r-a_k) + \alpha_k > f(k, r)$
      Set $f(k, r) := f(k-1, r-a_k) + \alpha_k$.

This routine runs in $\mathcal{O}(nb)$ time, and can be implemented to take only $\mathcal{O}(b)$ space. Moreover, it can be made faster in practice using the following observation: for any $k \in \{p+1, \ldots, n\}$, the quantity $f(p, b) - f(p, b - a_k)$ is an upper bound on α_k. Thus, once all of the $f(p, r)$ have been computed, one can compute an upper bound on the violation of the LPI in linear time. If this upper bound is non-positive, one need not proceed any further.

We remark that our lifting algorithm can be easily adapted to compute exact lifting coefficients for *any* valid inequality in $O(nb)$ time.

5 Enhancing Boccia's algorithm

In this section, we present some simple and effective ways to enhance Boccia's separation algorithm for $KP(a, b)$ itself. Throughout this section, N^0, N^1 and F denote $\{j \in N : x_j^* = 0\}$, $\{j \in N : x_j^* = 1\}$ and $\{j \in N : 0 < x_j^* < 1\}$, respectively.

5.1 Eliminating rows from consideration

Suppose that $x^* \in [0, 1]^n$ and $a^T x^* \leq b$. It can be shown that, if $\sum_{j \in N \setminus N^0} a_j \leq b$, then $x^* \in KP(a, b)$. There is therefore no point in searching for violated inequalities in that case. The same applies if $a \in \{0, 1\}^n$, since in that case $KP(a, b)$ is an integral polytope.

5.2 Looking for violated LCIs first

Our best algorithm for LCI separation (see Subsect. 6.5) is very fast and effective. Moreover, the LCIs are typically less dense and better behaved numerically than general facet-inducing inequalities. Thus, we recommend calling the LCI separation algorithm first, and resorting to exact knapsack separation only when one does not find a violated LCI.

5.3 Solving knapsack subproblems heuristically

To find violated constraints of the form (8), one has to repeatedly solve 0-1 KPs of the form (10)–(12). We solve these 0-1 KPs heuristically, and only call an exact routine when the heuristic fails. We run a simple greedy heuristic, based on profit-to-weight ratio, and then try to improve the solution obtained via local search. A 'move' in our local search algorithm consists in deleting one item from the knapsack and inserting one or more other items.

5.4 Warm-starting the LP

In each call of the separation algorithm, apart from the first, it helps a lot to 'warm-start' the LP (7)–(9) by including, not only the trivial bounds (9), but also the constraints (8) that were binding in the previous call. Although this is a simple idea, there is a complication: the separation LP is solved in the space of the fractional variables F, but the set F can change from one call to the next. To deal with this, we iteratively delete items from the corresponding packs P, until feasible packs are obtained. We then enlarge the packs, if possible, by iteratively inserting items that have moved from $N^0 \cup N^1$ to F since the last call.

5.5 A fast scaling heuristic

Recall that, once a violated inequality is found, it has to be scaled to integers. Boccia formulates this scaling problem itself as an integer linear program. We have found that the following heuristic works in over 90% of cases: let $\alpha_{\min}$ be the smallest *fractional* α^* value. Divide the entire inequality by $\alpha_{\min}$, and check if the resulting inequality is integral (to within a tolerance of 10^{-5}). To prevent rounding errors, we check that the resulting inequality is valid before proceeding to the lifting step.

5.6 One other minor consideration

We mentioned in Subsect. 2.1 that, when deriving LCIs, one or more items may have to be removed from the down-lifting set D if there exists an item $k \in N \backslash (C \cup N^0)$ such that $a_k > b - \sum_{j \in D} a_j$. An analogous issue arises in Boccia's method, if there exists an item $k \in F$ such that $a_k > b - \sum_{j \in N^1} a_j$. When this problem arises, we iteratively move items from N^1 to F until no such item exists.

6 Computational experiments

In this section, we present the results of some computational experiments, to assess the usefulness of our separation algorithms when applied to 0-1 LPs. We used the following very simple cutting plane algorithm:

1. Solve the initial LP relaxation of the problem.
2. If the solution is integer, stop.
3. For each row of the problem, call one or more separation algorithms. If any violated inequalities are found, add all of them to the LP, re-optimise and go to step 2.
4. Output the final upper bound and stop.

We used the primal simplex algorithm to solve the initial LP in step 1 and dual simplex to re-optimise after cutting planes have been added.

The algorithm was implemented in Microsoft Visual Studio.Net 2003 and called on functions from version 10.0 of the ILOG CPLEX Callable Library. All experiments were performed using a PC with an Intel Pentium IV, 2.8 GHz processor and 512 MB of RAM.

6.1 The MIPLIB instances

Following Gu et al. [16], we conducted experiments on 15 instances taken from MIPLIB [1]. Table 1 shows the following for each instance: instance name, number of variables n, number of constraints m, density of the constraint matrix (as a percentage), number of 'genuine' knapsack constraints (i.e., constraints whose left-hand-side coefficients are not all 0, 1 or -1), cost of the integer optimum, lower bound obtained by solving the initial LP relaxation (before cuts are added), and the corresponding integrality gap, expressed as a percentage of the optimum. We remark that the seven 'p' instances were used by both Crowder et al. [13] and Boyd [9,10] as benchmarks.

The instances `l152lav`, `lp41` and `mod010` have only one knapsack constraint each (an equation, treated as two inequalities in the table). We found that no family of knapsack-based cutting planes, not even the general knapsack facets, closed a significant proportion of the integrality gap for these instances. Thus, we omit them in the following tables.

In a few cases, we observed knapsack constraints for which one or more left-hand-side coefficients exceeded the right-hand side. This phenomenon indicates the presence of one or more variables that can be fixed permanently at either zero or

Table 1 Description of the MIPLIB instances

Name	n	m	%Dens.	KPs	Opt.	Init. LB	IG (%)
bm23	27	20	88.5	20	34.0	20.6	39.50
l152lav	1,989	97	50.8	2	4,722	4,656.4	1.39
lp41	1,086	85	12.0	2	2,967	2,942.5	0.83
lseu	89	28	12.4	11	1,120	834.7	25.47
mod008	319	6	64.9	6	307	290.9	5.24
mod010	2,655	146	5.8	2	6,548	6,532.1	0.24
p0033	33	16	17.4	11	3,089	2,520.6	18.40
p0040	40	24	11.3	13	62,027	61,796.5	0.37
p0201	201	134	5.0	107	7,615	6,875.0	9.72
p0282	282	242	2.0	44	258,411	1,76867.5	31.56
p0291	291	253	1.9	14	5,223.75	1,705.1	67.36
p0548	548	177	1.8	92	8,691	315.3	96.37
p2756	2,756	756	0.4	382	3,124	2,688.7	13.93
pipex	48	25	16.0	9	788.263	773.8	1.88
sentoy	60	30	100.0	30	−7772	−7839.3	0.87

one. We performed this fixing right at the start, to avoid problems with our separation routines.

6.2 Cover inequalities

Table 2 reports the results obtained when applying CIs to the MIPLIB instances. The columns headed '% gap closed' report the percentage of the integrality gap closed by the inequalities, and the columns headed 'time (s)' report the total running time of the cutting plane algorithm, in seconds. The columns headed 'CJP' were obtained using the greedy heuristic of Crowder et al. [13]. The column headed 'Exact' was obtained by solving the separation problem exactly. The column headed 'Ex1' reports the time taken when using the exact separation algorithm in its original form. The column headed 'Ex2' reports the time taken by a hybrid approach, in which the exact algorithm is called only when the Crowder et al. heuristic fails.

We see that the CIs, though theoretically weak, close half of the gap on average. Moreover, the Crowder et al. heuristic is very effective. All running times are negligible.

6.3 Extended cover inequalities

Table 3 reports the results obtained when applying ECIs to the MIPLIB instances. The columns headed 'CJP' were again obtained using the Crowder et al. heuristic. The columns headed 'GNS' were obtained using the alternative greedy heuristic of Gu et al. [16]. The columns headed 'N.H.' were obtained using the new heuristic presented

Table 2 Results obtained with cover inequalities

Name	%Gap closed		Time (s)		
	CJP	Exact	CJP	Ex1	Ex2
bm23	5.57	5.57	0.002	0.105	0.080
lseu	39.87	39.87	0.001	0.024	0.013
mod008	3.34	3.75	0.015	0.074	0.043
p0033	63.55	63.55	0.002	0.020	0.010
p0040	76.82	76.82	0.001	0.002	0.001
p0201	33.78	33.78	0.005	0.018	0.010
p0282	94.25	94.27	0.020	0.325	0.174
p0291	95.19	95.23	0.011	0.099	0.033
p0548	66.89	67.68	0.042	0.732	0.292
p2756	86.26	86.26	0.154	0.379	0.169
pipex	16.69	16.69	0.001	0.014	0.009
sentoy	16.33	16.58	0.004	0.351	0.208
Aver.	49.88	50.00	0.022	0.179	0.087

Table 3 Results obtained with extended cover inequalities

Name	%Gap closed				Time (s)					
	CJP	GNS	N.H.	Exact	CJP	GNS	N.H.	Ex1	Ex2	Ex3
bm23	15.36	15.36	15.82	15.82	0.00	0.00	0.00	0.08	0.02	0.05
lseu	39.87	39.85	61.01	61.36	0.00	0.00	0.02	0.13	3.16	0.06
mod008	4.58	4.42	6.31	17.59	0.00	0.02	0.20	0.44	236.6	0.34
p0033	66.50	66.50	71.51	71.93	0.00	0.02	0.02	0.06	0.09	0.03
p0040	76.82	76.82	100.00	100.00	0.00	0.02	0.00	0.00	0.19	0.00
p0201	33.78	33.78	33.78	33.78	0.02	0.00	0.02	0.02	0.19	0.00
p0282	94.19	93.48	92.72	94.35	0.03	0.03	0.02	0.44	0.39	0.22
p0291	95.19	95.15	94.96	95.23	0.03	0.02	0.02	0.09	5.71	0.08
p0548	67.55	66.48	67.37	67.68	0.06	0.08	0.06	1.05	16.9	0.30
p2756	86.26	78.16	86.26	86.26	0.17	0.22	0.16	0.52	70.6	0.20
pipex	16.68	16.68	23.82	27.96	0.00	0.00	0.00	0.06	0.03	0.09
sentoy	16.53	15.99	17.27	21.57	0.02	0.00	0.02	0.94	8.06	0.70
Aver.	51.11	50.22	55.90	57.79	0.03	0.03	0.05	0.32	28.5	0.17

in Subsect. 3.3. The column headed 'Exact' was obtained by solving the separation problem exactly. The column headed 'Ex1' corresponds to the $O(n^2b)$ exact algorithm described in Subsect. 3.1. The column headed 'Ex2' corresponds to the $O(nb)$ exact algorithm described in Subsect. 3.2. Finally, the column headed 'Ex3' corresponds to a hybrid approach, in which the $O(n^2b)$ exact algorithm is called only when the new heuristic fails.

We found these results rather surprising, for several reasons. First, the ECIs do not close a substantially larger proportion of the integrality gap than the CIs. Second, the Gu et al. heuristic performs no better than the Crowder et al. heuristic. Third, although the $O(nb)$ exact algorithm is theoretically faster than the $O(n^2b)$ exact algorithm, it is much slower in practice. Indeed, the running times are negligible for all schemes apart from the $O(nb)$ exact algorithm.

All things considered, the new ECI heuristic and the new hybrid exact ECI algorithm look promising.

6.4 Simple lifted cover inequalities

Next, we converted the ECI algorithms into simple LCI algorithms, as explained in Subsect. 3.4. We were disappointed to find that the simple LCIs closed no more gap than the ECIs, for any instance. Further investigation revealed that up-lifting coefficients larger than 1 occurred very rarely. We do not report the running times, for brevity, but they were all slightly larger than the running times for ECIs.

6.5 General lifted cover inequalities

Table 4 reports the results obtained when applying general LCIs to the MIPLIB instances. Four different separation heuristics are compared. The columns headed 'CJP' and 'GNS' were again obtained using the Crowder et al. and Gu et al. heuristics to generate the covers. The columns headed 'N.H.1' were obtained using our best exact ECI separation algorithm (labelled 'Ex3' in Subsect. 6.3) to generate the covers. The columns headed 'N.H.2' were obtained using a hybrid method, in which our best ECI algorithm is called only when the Gu et al. heuristic fails to yield a violated LCI. In all four cases, the Hoffman-Padberg lifting sequence was used to derive the general LCIs.

The general LCIs close significantly more gap than the ECIs on several instances. This confirms the value of down-lifting. Moreover, in our view, the running times are promising. All things considered, we recommend using the strategy represented by 'N.H.2'.

6.6 Weight inequalities and other lifted pack inequalities

Table 5 reports results obtained with weight inequalities and sequentially-lifted pack inequalities. The columns headed 'WI1', 'WI2' and 'WI3' were obtained with the Helmberg–Weismantel WI heuristic, the new WI heuristic described in Subsect. 4.3, and the enhanced version of Weismantel's exact WI algorithm presented in Subsect. 4.1. (We do not report results for the exact WI algorithm presented in Subsect. 4.2, because it ran into time and/or memory problems for several instances.) The columns headed 'sLPI' and 'LPI' were obtained with our heuristics for simple LPIs and general LPIs, respectively.

Table 4 Results obtained with lifted cover inequalities

Name	%Gap closed				Time (s)			
	CJP	GNS	N.H.1	N.H.2	CJP	GNS	N.H.1	N.H.2
bm23	16.11	16.00	16.11	16.11	0.000	0.000	0.047	0.047
lseu	56.99	59.28	61.56	66.20	0.000	0.016	0.062	0.062
mod008	17.81	17.55	18.24	18.24	0.047	0.031	0.359	0.094
p0033	70.36	80.62	80.62	80.62	0.000	0.015	0.031	0.031
p0040	100.00	100.00	100.00	100.00	0.000	0.000	0.000	0.000
p0201	33.78	33.78	33.78	33.78	0.015	0.016	0.016	0.016
p0282	95.97	94.84	96.15	96.21	0.046	0.031	0.625	0.437
p0291	95.19	96.39	95.41	96.40	0.015	0.031	0.110	0.047
p0548	67.56	66.81	67.71	67.71	0.062	0.078	0.437	0.297
p2756	86.26	80.52	86.26	86.26	0.187	0.219	0.265	0.250
pipex	72.78	72.62	73.34	73.34	0.000	0.000	0.062	0.032
sentoy	16.53	16.50	22.69	22.72	0.000	0.016	0.641	0.813
Aver.	60.78	61.24	62.66	63.13	0.031	0.038	0.221	0.178

Table 5 Results obtained with weight inequalities and sequentially-lifted pack inequalities

Name	%Gap closed					Time (s)				
	WI1	WI2	WI3	sLPI	LPI	WI1	WI2	WI3	sLPI	LPI
bm23	1.81	1.81	1.81	6.21	16.90	0.004	0.005	0.117	0.007	0.013
lseu	15.23	15.24	15.25	21.54	73.60	0.007	0.008	1.037	0.593	0.641
mod008	2.62	2.62	2.62	3.73	66.95	0.504	0.579	3.534	130.6	351.9
p0033	6.44	6.44	6.44	83.80	85.77	0.001	0.002	0.028	0.066	0.052
p0040	29.24	29.24	29.24	84.29	100.0	0.003	0.003	0.008	0.072	0.029
p0201	12.50	12.50	12.50	12.50	33.78	0.010	0.011	0.653	0.152	0.059
p0282	76.96	90.48	93.99	92.57	94.97	0.022	0.044	18.28	0.258	0.482
p0291	88.72	88.72	95.23	89.18	90.57	0.008	0.008	2.612	0.458	0.849
p0548	43.27	45.58	70.12	46.47	53.39	0.038	0.045	47.87	3.158	3.568
p2756	21.07	25.15	63.63	26.90	44.85	0.159	0.210	4.650	10.80	20.41
pipex	4.56	4.56	4.56	55.97	64.06	0.001	0.001	0.037	0.024	0.036
sentoy	5.85	5.91	7.34	6.10	19.39	0.005	0.006	62.47	0.840	2.385
Aver.	25.69	27.35	33.56	44.11	61.10	0.064	0.077	11.77	12.25	31.70

Our new heuristic for WIs gives slightly better bounds than the Helmberg-Weismantel heuristic, but the results obtained with WIs are not impressive. Simple LPIs perform a little better, but it is only the general LPIs that give bounds comparable to those obtained using LCIs. On the other hand, the running time is excessively high for some instances.

We also tried using both LCIs and LPIs in combination. The average amount of gap closed was 67.74%, which shows that LCIs and LPIs are to some extent complementary.

Table 6 Results obtained with general knapsack facets

Name	%Gap	T1	T2	T3
bm23	20.08	2.734	0.578	0.703
lseu	76.09	3.297	0.687	1.625
mod008	89.23	68.906	41.438	57.687
p0033	87.42	0.594	0.109	0.265
p0040	100.0	0.015	0.000	0.016
p0201	33.78	1.031	0.031	0.718
p0282	98.59	29.188	5.172	7.734
p0291	99.43	9.454	1.875	3.579
p0548	84.34	37.250	2.250	16.391
p2756	86.36	38.469	1.000	24.547
pipex	86.55	4.640	0.797	1.188
sentoy	30.97	38.844	3.829	11.141
Aver.	74.49	19.533	4.814	10.466

6.7 General knapsack facets

Table 6 reports results obtained using general facets of the knapsack polytope. The percentage gap closed is shown, along with the running times for three variants of Boccia's separation scheme. The columns headed 'T1' and 'T2' correspond to Boccia's original scheme and our enhanced scheme, respectively. The column headed 'T3' was obtained with a scheme that incorporates all of our enhancements apart from the one mentioned in Subsect. 5.2, i.e., a scheme that does not use LCIs.

Several things are apparent from the table. First, we see that the general knapsack facets close significantly more of the integrality gap than the LCIs and LPIs. Second, our enhanced scheme is around four times faster, on average, than the original one. Third, it definitely pays off to call LCI separation first, provided of course that one has a fast and effective separation heuristic for LCIs. Fourth, our best exact separation algorithm for general knapsack facets is (bizarrely) less time-consuming than our best exact separation algorithms for WIs and LPIs.

A couple of further comments are in order. First, we were using a general-purpose branch-and-bound solver to solve the knapsack subproblems. One could probably obtain a substantial speed-up using a specialised algorithm for the 0-1 KP (Boccia himself uses a modified version of the `MINKNAP` algorithm of Pisinger [26]). Second, we wish to point out that, for the '`p`' instances, the percentage gaps closed by the general knapsack facets are exactly the same as those reported by Boyd [9,10], with one exception: for `p2756`, Boyd obtained a slightly smaller value of 86.16%. Indeed, this instance contains a few very dense constraints that, according to Boyd, caused his algorithm to run into difficulties.

6.8 Experiments with multi-dimensional knapsack instances

To gain further insight into the relative performance of the different inequalities, and also to explore the limits of our approaches, we also performed experiments on the

Table 7 Percentage gap closed by inequalities on 0-1 MKP instances

n	m	$\alpha(\%)$	IG (%)	CIs	ECIs	LCIs	WIs	sLPIs	LPIs	All
100	5	25	0.99	2.39	3.66	4.20	1.96	2.19	9.53	17.96
100	5	50	0.45	2.88	5.13	5.81	2.82	2.88	10.52	21.65
100	5	75	0.32	3.48	5.41	6.77	3.15	3.18	12.14	22.88
100	10	25	1.59	0.34	0.62	0.79	0.13	0.14	1.55	5.55
100	10	50	0.80	0.51	1.59	1.73	0.60	0.62	2.75	7.33
100	10	75	0.48	0.17	0.72	1.01	0.21	0.21	2.47	7.23
100	30	25	2.97	0.00	0.00	0.00	0.00	0.00	0.00	0.16
100	30	50	1.34	0.00	0.01	0.01	0.00	0.00	0.00	0.49
100	30	75	0.83	0.00	0.01	0.01	0.00	0.00	0.00	0.52
250	5	25	0.25	0.93	1.68	2.02	1.02	1.03	5.28	14.56
250	5	50	0.11	1.44	2.39	3.04	1.19	1.20	5.97	15.68
250	5	75	0.08	1.21	2.82	3.94	0.96	0.96	7.77	17.48
250	10	25	0.46	0.10	0.37	0.58	0.12	0.12	1.35	4.53
250	10	50	0.23	0.21	0.40	0.47	0.14	0.14	1.28	4.48
250	10	75	0.14	0.14	0.39	0.58	0.09	0.10	1.65	5.03
500	5	25	0.07	1.28	1.63	2.14	1.30	1.31	5.17	13.80
500	5	50	0.04	0.88	1.61	1.90	0.82	0.83	4.39	11.91
500	5	75	0.02	0.87	1.64	1.98	0.88	0.88	74.41	13.70
			Aver.	0.94	1.67	2.06	0.86	0.88	4.24	10.28

0-1 *multi-dimensional knapsack problem* (0-1 MKP). We used the library of 0-1 MKP instances created by Chu and Beasley [11], which are available in the OR Lib [5]. In these instances, the constraint matrix A consists entirely of positive integers—and is therefore 100% dense. Although the library contains 270 instances, the largest instances have not yet been solved to proven optimality (see, e.g., Vimont et al. [28]). Thus, we restrict our attention to 180 of the instances, for which the optima are known.

The 180 instances are arranged in blocks of ten. Table 7 reports the following information for each block. The first three columns show the number of variables n, the number of constraints m and the so-called *tightness ratio* α. The next column shows the percentage integrality gap of the initial LP relaxation (before the addition of cutting planes). The remaining columns show the percentage of the integrality gap closed by each of seven classes of inequalities. Each figure in the last seven columns is averaged over the given ten instances.

Before interpreting the results, there are three points that we need to make. First, just as with the MIPLIB instances, the simple LCIs gave exactly the same results as the ECIs, and therefore there is no separate column for them. Second, the column headed 'WIs' was obtained using the heuristic mentioned in Subsect. 4.3, since our exact algorithm ran into time and memory problems for most of these instances. Third, some of the results in the table have been reported elsewhere in the literature

(Bektas and Oguz [6], Gabrel and Minoux [15], Kaparis and Letchford [21]), but only for $n = 100$ and only for CIs, ECIs and LCIs.

Compared to the results obtained with the MIPLIB instances, there are some striking differences. The first is that simple LPIs perform no better than WIs. The second is that LPIs perform significantly better than both LCIs and simple LPIs. The third is that general knapsack facets close a much larger proportion of the gap than either LCIs or LPIs. This suggests that, for constraints involving many variables, LCIs and LPIs give a rather poor partial description of the knapsack polytope. Finally, even the general knapsack facets close only a small proportion of the gap when $m > 5$. This demonstrates that the intersection of the individual 0-1 knapsack polytopes give a poor approximation to the true polytope for problems with more than a few dense constraints.

For the sake of brevity, we do not report the running times in detail. We just want to say that the average running time was a fraction of a second for the CIs, ECIs, LCIs and WIs, several seconds for the simple and general LPIs, but around 5 min for the general knapsack facets. This latter figure may seem excessive, but it should be borne in mind that the separation algorithms of Boyd [8–10] run into serious difficulties as soon as a constraint has more than about 30 non-zeroes on the left-hand side, rather than hundreds of non-zeroes as we have in this case.

7 Concluding remarks

In this paper, we have presented new or improved separation algorithms for several classes of valid inequalities for the 0-1 knapsack polytope—namely, the CIs, ECIs, LCIs, WIs and LPIs—and also for the 0-1 knapsack polytope itself. We have also presented extensive computational results.

Our computational results with LCIs confirm the claim in [13] that the LCIs make very useful cutting planes for sparse unstructured 0-1 LPs. They also confirm the claim in [16,20] that general LCIs are more useful than simple LCIs. One surprising result, however, is that the simple LCIs generated in practice tend to be 'mere' ECIs. Thus, if one does not wish to bother implementing down-lifting, then one might as well not bother implementing up-lifting either, and instead work directly with ECIs.

To our knowledge, this paper is the first to report detailed computational results for WIs and sequentially-lifted LPIs. The WIs and simple LPIs exhibited rather disappointing performance, but the general LPIs look more promising. In any case, we believe that further research on WIs and LPIs is warranted. In particular, it would be nice to determine whether or not WI separation is NP-hard. It would also be good to further examine alternative exact and approximate lifting techniques for LPIs, such as the simultaneous-lifting procedure of Atamtürk [2].

Finally, we recall that, on some instances, the general knapsack facets close considerably more of the integrality gap than the LCIs and LPIs. This indicates that the LCIs and WIs give a poor approximation of the 0-1 knapsack polytope, especially when the number of variables in the knapsack constraint is large. Since separation of general knapsack facets is still rather time-consuming, it might be worthwhile devising and

testing separation algorithms for other known classes of valid inequalities (such as the *extended* weight inequalities [29]). In fact, we believe that the 0-1 knapsack polytope itself deserves further study.

References

1. Achterberg, T., Koch, T., Martin, A.: MIPLIB 2003. Oper. Res. Lett. **34**, 361–372
2. Atamtürk, A.: Cover and pack inequalities for (mixed) integer programming. Ann. of Oper. Res. **139**, 21–38 (2005)
3. Balas, E.: Facets of the knapsack polytope. Math. Program. **8**, 146–164 (1975)
4. Balas, E., Zemel, E.: Facets of the knapsack polytope from minimal covers. SIAM J. Appl. Math. **34**, 119–148 (1978)
5. Beasley, J.E.: OR-Library: distributing test problems by electronic mail. J. Oper. Res. Soc. **41**, 1069–1072 (1990)
6. Bektas, T., Oguz, O.: On separating cover inequalities for the multidimensional knapsack problem. Comp. Oper. Res. **34**, 1771–1776 (2007)
7. Boccia M.: Using exact knapsack separation for the single-source capacitated facility location problem. Working paper, Department of Engineering, University of Sannio (2006)
8. Boyd, E.A.: A pseudo-polynomial network flow formulation for exact knapsack separation. Networks **22**, 503–514 (1992)
9. Boyd, E.A.: Generating Fenchel cutting planes for knapsack polyhedra. SIAM J. Optimization **3**, 734–750 (1993)
10. Boyd, E.A.: Fenchel cutting planes for integer programs. Oper. Res. **42**, 53–64 (1994)
11. Chu, P.C., Beasley, J.E.: A genetic algorithm for the multidimensional knapsack problem. J. Heur. **4**, 63–86 (1998)
12. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. MIT Press, Cambridge, MA (2001)
13. Crowder, H., Johnson, E., Padberg, M.: Solving large-scale 0-1 linear programming programs. Oper. Res. **31**, 803–834 (1983)
14. Ferreira, C.E., Martin, A., Weismantel, R.: Solving multiple knapsack problems by cutting planes. SIAM J. Opt. **6**, 858–877 (1996)
15. Gabrel, V., Minoux, M.: A scheme for exact separation of extended cover inequalities and application to multidimensional knapsack problems. Oper. Res. Lett. **30**, 252–264 (2002)
16. Gu, Z., Nemhauser, G.L., Savelsbergh, M.W.P.: Lifted cover inequalities for 0-1 integer programs: computation. INFORMS J. Computing **10**, 427–437 (1998)
17. Gu, Z., Nemhauser, G.L., Savelsbergh, M.W.P.: Lifted cover inequalities for 0-1 integer programs: complexity. INFORMS J. Computing **11**, 117–123 (1999)
18. Gu, Z., Nemhauser, G.L., Savelsbergh, M.W.P.: Sequence independent lifting in mixed integer programming. J. Comb. Opt. **4**, 109–129 (2000)
19. Helmberg, C., Weismantel, R.: Cutting plane algorithms for semidefinite relaxations. In: Pardalos, P.M., Wolkowicz, H. (eds.) Fields Institute Communications. vol. **18**, pp. 197–213 (1998)
20. Hoffman, K.L., Padberg, M.W.: Improving LP-representations of zero-one linear programs for branch-and-cut. ORSA J. Comput. **3**, 121–134 (1991)
21. Kaparis, K., Letchford, A.N.: Local and global lifted cover inequalities for the multidimensional knapsack problem. Eur. J. Opl Res. **186**, 91–103 (2008)
22. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (eds.) Complexity of Computer Computations., Plenum, New York (1972)
23. Klabjan, D., Nemhauser, G.L., Tovey, C.: The complexity of cover inequality separation. Oper. Res. Lett. **23**, 35–40 (1998)
24. Nemhauser, G.L., Wolsey, L.A.: Integer and Combinatorial Optimization. Wiley, New York (1988)
25. Padberg, M.W.: $(1, k)$-configurations and facets for packing problems. Math. Program. **18**, 94–99 (1980)
26. Pisinger, D.: A minimal algorithm for the 0-1 knapsack problem. Oper. Res. **46**, 758–767 (1995)
27. Van Roy, T.J., Wolsey, L.A.: Solving mixed integer programming problems using automatic reformulation. Oper. Res. **35**, 45–57 (1987)

28. Vimont, Y., Boussier, S., Vasquez, M.: Reduced costs propagation in an efficient implicit enumeration for the 0-1 multidimensional knapsack problem. J. Comb. Opt. **15**, 165–178 (2008)
29. Weismantel, R.: On the 0-1 knapsack polytope. Math. Program. **77**, 49–68 (1997)
30. Williams, J.W.J.: Algorithm 232—Heapsort. Commun. ACM **7**, 347–348 (1964)
31. Wolsey, L.A.: Faces for a linear inequality in 0-1 variables. Math. Program. **8**, 165–178 (1975)
32. Zemel, E.: Easily computable facets of the knapsack polytope. Math. Oper. Res. **14**, 760–765 (1989)