INFORMS Journal on Computing
Articles in Advance, pp. 1–12
ISSN 1091-9856 (print) | ISSN 1526-5528 (online)

http://dx.doi.org/10.1287/ijoc.1120.0511

Separation and Extension of Cover Inequalities for Conic Quadratic Knapsack Constraints with Generalized Upper Bounds

Alper Atamtürk
Department of Industrial Engineering and Operations Research, University of California, Berkeley, California 94720, atamturk@berkeley.edu

Laurent Flindt Muller, David Pisinger
Department of Management Engineering, Technical University of Denmark, Produktionstorvet, DK-2800 Kgs. Lyngby, Denmark
{lafm.man@gmail.com, pisinger@man.dtu.dk}

Motivated by addressing probabilistic 0–1 programs we study the conic quadratic knapsack polytope with generalized upper bound (GUB) constraints. In particular, we investigate separating and extending GUB cover inequalities. We show that, unlike in the linear case, determining whether a cover can be extended with a single variable is $\mathcal{NP}$-hard. We describe and compare a number of exact and heuristic separation and extension algorithms which make use of the structure of the constraints. Computational experiments are performed for comparing the proposed separation and extension algorithms. These experiments show that a judicious application of the extended GUB cover cuts can reduce the solution time of conic quadratic 0–1 programs with GUB constraints substantially.

Key words: programming: integer, nonlinear, convex, constraints; computational analysis
History: Accepted by Karen Aardal, Area Editor for Design and Analysis of Algorithms; received May 2011; revised November 2011, February 2012; accepted March 2012. Published online in *Articles in Advance*.

1. Introduction

We consider the conic quadratic knapsack polytope with *generalized upper bound* (GUB) constraints. The motivation for studying this polytope is to address 0-1 programming problems with probabilistic knapsack constraints. When the coefficients of a constraint are not deterministic, but are random variables, whether the constraint is satisfied or not is not only a function of the solution vector chosen, but also the realization of the random coefficients. In that case, one is interested in choosing a solution vector so that the constraint will be satisfied at least with a certain probability. Given a finite index set N, a probabilistic constraint over a binary vector $x \in \{0,1\}^N$ is stated as

$$\text{Prob}(\tilde{a}x \leq b) \geq \epsilon$$

for some $0 < \epsilon < 1$. If each coefficient $\tilde{a}_i$ is independent normally distributed with mean μ_i and variance σ_i^2, and $\epsilon \geq 0.5$, then the above probabilistic constraint can be formulated as a deterministic conic quadratic constraint (see, e.g., Boyd and Vandenberghe 2004):

$$\sum_{i \in N} \mu_i x_i + \Phi^{-1}(\epsilon) \sqrt{\sum_{i \in N} \sigma_i^2 x_i^2} \leq b,$$

where Φ is the standard normal cumulative distribution function.

GUB constraints frequently appear in practical 0–1 optimization problems and their utilization in polyhedral analysis and cut generation algorithms reduces the computational effort in solving mixed-integer programs significantly. Consider a nonempty partitioning of N indexed by K; that is, $\bigcup_{k \in K} Q_k = N$ and $Q_i \cap Q_j = \varnothing$ for all distinct $i, j \in K$. GUB constraints on the variables are upper bounding constraints of the form

$$\sum_{i \in Q_k} x_i \leq 1, \quad \forall k \in K.$$

In the following we will also refer to the sets $Q_1, \ldots, Q_{|K|}$ as *GUB-sets*.

In this paper we study the *conic quadratic knapsack set with GUB constraints*:

$$X := \left\{ x \in \{0,1\}^N \colon \sum_{i \in N} a_i x_i + \omega \sqrt{\sum_{i \in N} d_i x_i^2} \leq b, \right.$$
$$\left. \sum_{i \in Q_k} x_i \leq 1,\ \forall k \in K \right\},$$

where $a \in \mathbb{R}_+^N$, $d \in \mathbb{R}_+^N$, and $\omega > 0$. For $S \subseteq N$ and $k \in K$, define $S^{\cap k} := S \cap Q_k$ and $S^{\setminus k} := S \setminus Q_k$, and for some

$v \in \mathbb{R}^N$; define $v(S) := \sum_{i \in S} v_i$. For a binary vector $x \in \{0,1\}^N$, define $S_x := \{i \in N\colon x_i = 1\}$.

The literature on cuts for conic quadratic mixed integer programming (MIP) is quite sparse: Atamtürk and Narayanan (2010) and Cezik and Iyengar (2005) describe rounding cuts, Atamtürk and Narayanan (2011) describe lifting procedures and Atamtürk and Narayanan (2009) consider the sub-modular knapsack polytope, which is the same as the polytope considered here except that there are no GUB constraints. For this polytope, the authors describe cover inequalities and present a heuristic for separating them based on the convex continuous relaxation of the separation problem. They additionally describe procedures for extending and lifting cover inequalities in order to strengthen them. As the polytope considered by the authors does not include GUB constraints this work can be seen as an extension to the case where GUB constraints are present.

Cover inequalities for linear knapsack constraints were introduced independently by Balas (1975), Hammer et al. (1975), and Wolsey (1975). Both Balas (1975) and Wolsey (1975) treat the lifting of cover inequalities. Complexity results for obtaining lifted cover inequalities can be found in Zemel (1989) and Hartvigsen and Zemel (1992). If GUB constraints are present, they may be used during lifting to further strengthen the cover inequalities. Lifting has in this setting been treated by Johnson and Padberg (1981), Wolsey (1990), and Nemhauser and Vance (1994). The separation problem has been investigated in a number of studies: Crowder et al. (1983) have shown that the problem can be formulated as a knapsack problem, whereas Ferreira et al. (1996), Klabjan et al. (1998), and Gu et al. (1999) show that the separation problem for different classes of cover inequalities is $\mathcal{NP}$-hard. A number of exact and heuristic methods exist for solving the separation problem; see for instance Gu et al. (1998) for a detailed investigation of computational issues with respect to branch-and-cut algorithms. For recent surveys on cuts for linear knapsacks, the reader is referred to Atamtürk (2005) and Kaparis and Letchford (2010).

In the context of robust knapsack problems, cover and extended cover inequalities have been investigated by Klopfenstein and Nace (2009) and by Büsing et al. (2011).

The contribution of this work is the proposal and analysis of a number of separation and extension algorithms for cover inequalities for second-order conic knapsacks in the presence of GUB constraints. Unlike in the linear case, separation and extension of cover inequalities are themselves nonlinear 0–1 problems. We show that the problem of determining whether a cover may be extended with even a *single* variable is $\mathcal{NP}$-hard. Through computational experiments the proposed algorithms are mutually compared with respect to bound improvement and computation time. We show that a judicious application of extended cover inequalities can greatly improve the solution time of conic quadratic 0–1 programs with GUB constraints.

The outline of the paper is as follows. In §2 covers, extended covers, and extended covers under the presence of GUB constraints are introduced. In §3 a number of alternative algorithms for extending covers are proposed, whereas in §4 alternative separation algorithms are described. In §5 the efficiency of the proposed algorithms are evaluated computationally. We conclude in §6 with a few final remarks.

2. Cover Inequalities

A subset $C \subseteq N$ is called a *cover* for X if $a(C) + \omega\sqrt{d(C)} > b$. A cover C is called a *minimal cover* if no strict subset of C is a cover. If C satisfies $|C^{\cap k}| \le 1$, $\forall\, k \in K$, then it is called a *GUB cover*. Given a cover C, the *cover inequality*

$$\sum_{i \in C} x_i \le |C| - 1 \tag{1}$$

is valid for X (Atamtürk and Narayanan 2009).

Example. Consider the conic quadratic GUB knapsack given by the constraints

$$3x_1 + 4x_2 + 2x_3 + 3x_4 + 1x_5 + \sqrt{2x_1^2 + 1x_2^2 + 2x_3^2 + 1x_4^2 + 10x_5^2} \le 7, \tag{2}$$

$$x_1 + x_2 \le 1, \quad x_3 + x_4 + x_5 \le 1. \tag{3}$$

$C' = \{1, 2\}$ is a cover, but not a GUB cover as x_1 and x_2 belong to the same GUB-set. $C = \{1, 4\}$, on the other hand, is a GUB cover. Both C and C' are minimal.

Cover inequalities do not in general define facets of $\operatorname{conv}(X)$. However, a cover inequality may be strengthened by including variables not part of the cover. The process of adding variables to an existing cover is called *extending* the cover inequality and may be viewed as a special form of lifting procedure where lifting coefficients may take only values zero or one. Allowing lifting coefficients to be fractional may result in stronger inequalities; however, as shown by Atamtürk and Narayanan (2009), calculating the lifting coefficients of a variable requires solving an optimization problem over the conic quadratic 0–1 knapsack set, which is $\mathcal{NP}$-hard even when no GUB constraints are present. Considering only 0–1 lifting coefficients makes this problem simpler (although still $\mathcal{NP}$-hard), and in the present work we restrict our attention to this case. Atamtürk and Narayanan (2009) describe a procedure for extending a minimal cover for the conic quadratic knapsack set when no GUB constraints are present. While this procedure is valid, utilizing the GUB constraints in extending the cover inequalities can result in stronger inequalities for X.

2.1. Extending Cover Inequalities with GUB Constraints

We now describe how GUB constraints can be used to strengthen cover inequalities. For an integer $n \geq 0$, and subset $S \subseteq N$, define $\mathcal{W}(S, n) := \{T \subseteq S\colon |T| \geq n \wedge |T^{\cap k}| \leq 1, \forall k \in K\}$, i.e., the set of all subsets of S, of at least size n, which contain at most one element from each Q_k. We call a subset $C \subseteq N$ an *n-cover* if S is a cover $\forall S \in \mathcal{W}(C, n)$. An n-cover C is *minimal* if C is not an n'-cover for any $n' < n$. Note that a GUB cover C is a $|C|$-cover.

Proposition 1. *If C is an n-cover, then the following inequality is valid for X:*

$$\sum_{i \in C} x_i \leq n - 1.$$

Proof. Let $x \in X$. Assume for the sake of contradiction that $\sum_{i \in C} x_i \geq n$. Let $S = C \cap T_x$. We have $x \in X \Rightarrow T_x \cap Q_k \leq 1, \forall k \in K \Rightarrow S \cap Q_k \leq 1, \forall k \in K$, and $|S| = \sum_{i \in S} 1 = \sum_{i \in C \cap T_x} 1 = \sum_{i \in C} x_i \geq n$. Thus $S \in \mathcal{W}(C, n)$, which is a contradiction since

$$a(S_x) + \omega\sqrt{d(S_x)} \leq a(T_x) + \omega\sqrt{d(T_x)} \leq b,$$

as $x \in X$. □

Example (Continued). Let us extend the 2-cover $C = \{1, 4\}$ with element 2 resulting in the set $C'' = \{1, 2, 4\}$. $\mathcal{W}(C'', 2) = \{\{1, 4\}, \{2, 4\}\}$, and since $\{1, 4\}$, and $\{2, 4\}$ are both covers, the set C'' is a 2-cover as well and the inequality $x_1 + x_2 + x_4 \leq 1$ is thus valid.

Proposition 2. *Let C be an n-cover and $i^* \in Q_{k^*} \backslash C$ for some $k^* \in K$. If*

$$a(S) + a_{i^*} + \omega\sqrt{d(S) + d_{i^*}} > b, \quad \forall S \in \mathcal{W}(C^{\backslash k^*}, n-1) \tag{4}$$

holds, then $C \cup \{i^*\}$ is also an n-cover.

Proof. Let $T \in \mathcal{W}(C \cup \{i^*\}, n)$. If $i^* \notin T$, then T is a cover by assumption. Assume $i^* \in T$; then $T = S \cup \{i^*\}$ for some $S \in \mathcal{W}(C^{\backslash k^*}, n-1)$, and thus $a(T) + \omega\sqrt{d(T)} > b$, and T is hence a cover. Therefore $C \cup \{i^*\}$ is an n-cover. □

Proposition 2 suggests a method for extending a cover: Start with identifying a GUB cover and for some ordering of the variables currently not in the cover, check iteratively one at a time if the variable can be included by evaluating condition (4). This task can be accomplished by solving the following optimization problem:

$$\text{OPT:} \quad \nu = \min \ a(S) + a_{i^*} + \omega\sqrt{d(S) + d_{i^*}}$$
$$\text{s.t.} \ \ S \in \mathcal{W}(C^{\backslash k^*}, n-1).$$

If $\nu > b$, then n-cover C can be extended with i^*. OPT is a constrained minimization of a submodular function. For surveys of submodular function minimization we refer the reader to Fujishige (2005), and Iwata (2008).

Example (Continued). Consider now extending the 2-cover $C'' = \{1, 2, 4\}$ with element $i^* = 5$. Recall that the GUB-sets are: $Q_1 = \{1, 2\}$ and $Q_2 = \{3, 4, 5\}$. In this case $k^* = 2$. We have $\mathcal{W}(C''^{\backslash 2}, 1) = \{\{1\}, \{2\}\}$, and since $3 + 1 + \sqrt{2 + 10} > 7$, and $4 + 1 + \sqrt{1 + 10} > 7$, the cover may be extended with 5 and $\{1, 2, 4, 5\}$ is a 2-cover as well.

We now show that OPT is $\mathcal{NP}$-hard. First note that OPT is equivalent to the following conic quadratic integer program (CQIP):

$$\min \sum_{i \in C^{\backslash k^*}} a_i y_i + a_{i^*} + \omega\sqrt{\sum_{C^{\backslash k^*}} d_i y_i^2 + d_{i^*}} \tag{5}$$

$$\text{s.t.} \sum_{i \in C^{\cap k}} y_i \leq 1 \quad \forall k \in K,\ k \neq k^*, \tag{6}$$

$$\sum_{i \in C^{\backslash k^*}} y_i \geq n - 1, \tag{7}$$

$$y_i \in \{0, 1\} \quad \forall i \in C^{\backslash k^*}, \tag{8}$$

where $y_i = 1$ if and only if $i \in S$. Constraints (6) ensure that S contains at most one element from each GUB-set, and constraints (7) ensure that S contains at least $n - 1$ elements.

Proposition 3. *Optimization problem* (5)–(8) *is $\mathcal{NP}$-hard.*

Proof. For ease of exposition, let $\mathcal{I} = C^{\backslash k^*} = \{1, \ldots, p\}$, let $\mathcal{K} = K \backslash \{k^*\}$, let $\mathcal{Q}_k = C^{\cap k}$, $\forall k \in \mathcal{K}$, let $m = n - 1$, and let $a_{i^*} = d_{i^*} = 0$. The problem considered is

$$P: \quad \min \sum_{i=1}^{p} a_i y_i + \omega\sqrt{\sum_{i=1}^{p} d_i y_i^2},$$
$$\text{s.t.} \sum_{i \in \mathcal{Q}_k} y_i \leq 1 \quad k \in \mathcal{K},$$
$$\sum_{i=1}^{p} y_i \geq m,$$
$$y_i \in \{0, 1\} \quad \forall i = 1, \ldots, p.$$

If the second part of the objective is zero (e.g., $\omega = 0$), the problem may be solved in polynomial time using a simple greedy algorithm: Let $\underline{a}_k = \min\{a_i \in \mathcal{Q}_k\}$. Now choosing the m smallest values of

$\underline{a}_k$ gives an optimal solution y' with value l. The solution l is a lower bound for the general problem P.

We can also find an upper bound on an optimal solution value of P as follows: Let $D = \omega\sqrt{\sum_{i=1}^{p} d_i}$; then the optimal solution value is not bigger than $u = l + D$. To see this, assume an optimal solution has value larger than $l + D$. Now construct a new solution corresponding to y'; clearly $\sum_{i=1}^{p} a_i y'_i + \omega\sqrt{d_i y'_i} \le l + D$.

To prove that P is $\mathcal{NP}$-hard, we consider the decision problem:

$$
\begin{aligned}
P'\colon\quad & \sum_{i=1}^{p} a_i y_i + \omega\sqrt{\sum_{i=1}^{p} d_i y_i^2} + s = E\\
& \sum_{i\in\mathcal{Q}_k} y_i \le 1 \quad k\in\mathcal{K},\\
& \sum_{i=1}^{p} y_i \ge m,\\
& y_i \in \{0,1\} \quad \forall\, i = 1,\ldots,p,\\
& 0 \le s \le D.
\end{aligned}
$$

The variable s is a slack variable, and since $u - l = D$ we can restrict s to be between 0 and D. If P' is $\mathcal{NP}$-hard, then so is P, since instances of P' can be solved as follows: if $E < l$, we answer "no;" if $E > l + D$ we answer "yes" returning y' as a certificate; otherwise we solve P. If the objective value is above E, we answer "no;" otherwise we answer "yes" returning the solution to P as a certificate.

Consider the $\mathcal{NP}$-complete two-partition problem (see Karp 1972): Given a set of positive integers, $W = \{w_1, \ldots, w_q\}$. Is it possible to separate them into two sets, W_1 and W_2, such that $\sum_{i\in W_1} w_i = \sum_{i\in W_2} w_i = C = \frac{1}{2}\sum_{i=1}^{q} w_i$?

We reduce the two-partition problem to P' as follows. Let $p := 2\cdot q$, and for $i = 1, \ldots, q$ set $a_i := 2Dw_i$, $a_{q+i} := 0$, $d_i := 0$, $d_{q+i} := w_i$, set $\mathcal{K} := \{1, \ldots, q\}$, $\mathcal{Q}_k := \{i, k+i\}\forall k \in \mathcal{K}$, $m := q$, $E := 2DC + \sqrt{C}$, and $\omega := 1$. This leads to the following instance of P':

$$
\begin{gathered}
\sum_{i=1}^{q} 2Dw_i y_i + \omega\sqrt{\sum_{i=1}^{q} w_i y_{q+i}^2} + s = 2DC + \sqrt{C},\\
y_i + y_{k+i} \le 1 \quad k \in \mathcal{K},\\
\sum_{i=1}^{2q} y_i \ge q,\\
y_i \in \{0,1\} \quad \forall\, i = 1,\ldots,p,\\
0 \le s \le D.
\end{gathered}
$$

The constraints $y_i + y_{q+i} \le 1$ and $\sum_{i=1}^{2q} y_i \ge q$ together imply that $y_i + y_{q+i} = 1$.

Assume that two-partition has a feasible solution; i.e., there exists a binary vector y, such that $\sum_{i=1}^{q} w_i y_i = C$. Setting $y_{q+i} = 1 - y_i$, we find a solution to the above problem with $s = 0$.

Now assume the above problem has a feasible solution. The second part of the objective satisfies

$$0 \le \sqrt{\sum_{i=1}^{q} w_i y_{q+i}^2} + s \le 2D.$$

This means that if

$$\sum_{i=1}^{q} 2Dw_i y_i + \sqrt{\sum_{i=1}^{q} w_i y_{q+i}^2} + s = 2DC + \sqrt{C},$$

then both the following constraints are satisfied

$$
\begin{gathered}
\sum_{i=1}^{q} w_i y_i = C,\\
\sqrt{\sum_{i=1}^{q} w_i y_{q+i}^2} + s = \sqrt{C}.
\end{gathered}
\tag{9}
$$

To see this assume $\sum_{i=1}^{q} w_i y_i \ne C$. This means $\sum_{i=1}^{q} w_i y_i = C - k$ for some $k \in \mathcal{Z}$. We have

$$2D(C-k) + \sqrt{\sum_{i=1}^{q} w_i y_{q+i}^2} + s = 2DC + \sqrt{C},$$

implying that

$$\sqrt{\sum_{i=1}^{q} w_i y_{q+i}^2} + s = \sqrt{C} + k2D \begin{cases} > 2D, & \text{if } k \in \mathcal{Z}^+,\\ < 0, & \text{if } k \in \mathcal{Z}^-,\end{cases}$$

both of which are contradictions.

But the first equation of (9) above means we have found a solution to the two-partition problem. □

In the next section we give a number of algorithms, which can be used to check the condition of Proposition 2. The effectiveness of the proposed algorithms will be evaluated in §5.

3. Algorithms for Extending Cover Inequalities

First observe that it is not necessary to solve OPT to optimality in order to decide whether a cover C can be extended. Given a lower bound LB on ν, the cover can be extended if LB $> b$. Finding a lower bound may be computationally easier, but the resulting cover inequalities may be weaker, because certain variables, which could have been added to the cover, may be missed. Thus there is a trade-off between the time spent for extending the covers, and the strength of the resulting cover inequalities.

We now describe a generic extension algorithm, which can be used with any procedure giving a lower bound on ν, starting with some initial GUB cover C. In the following, unless otherwise stated, we assume that the variable considered for extension has index $i^* \in N$ and belongs to the GUB-set with index $k^* \in K$. Assume that given some extended cover C, the function $\mathrm{LB}(C, i^*)$ gives a lower bound on OPT. The generic extension algorithm is shown in Algorithm 1.

Algorithm 1 (Generic algorithm for extending a GUB cover C)

Require: The initial GUB cover C to be extended.
  Let $I = N \setminus C$ be an ordered set.
  for all $i^* \in I$ **do**
    $\mathrm{LB} \leftarrow \mathrm{LB}(C, i^*)$.
    if $\mathrm{LB} > b$ **then**
      $C \leftarrow C \cup \{i^*\}$.
    end if
  end for
  return C

Different orderings of I will result in different extended covers. As the final aim is to find a violated n-cover inequality, and variables with a large value in the relaxed solution could be more beneficial in this regard, the set I is sorted nonincreasingly w.r.t. the relaxed solution values.

In the following a number of lower bounding approaches along with an optimal solution approach is described. The latter is included in order to evaluate the quality of the lower bounding approaches. Any of these approaches can be used for computing $\mathrm{LB}(C, i^*)$ in Algorithm 1.

3.1. Optimal Extension

As we saw in the previous section OPT can be formulated as a CQIP. The resulting problem is a constrained submodular function minimization problem. Atamtürk and Narayanan (2008) treat such a minimization problem using a cutting plane approach. For the computational experiments, we do not, however, employ this approach, but instead solve the above model directly with a CQIP solver for the purpose of obtaining a reference to assess the quality of other lower bounding algorithms. Note that the optimization may be halted as soon as the current lower bound is above b.

3.2. Lower Bound 1

A simple lower bound is obtained by relaxing the CQIP (5)–(8) by allowing the y_i's to take fractional values. In the following we denote this convex relaxation bound as LB1.

3.3. Lower Bound 2

The second bound is obtained by decomposing the objective of OPT into linear and nonlinear parts. Namely, we consider the bound:

$$\nu' = z_a + z_d,$$

where

$$z_a = \min \sum_{i \in C \setminus k^*} a_i y_i + a_{i^*} \tag{10}$$

$$\text{s.t. } (6)\text{–}(8), \tag{11}$$

and

$$z_d = \min \ \omega \sqrt{\sum_{C \setminus k^*} d_i y_i^2 + d_{i^*}} \tag{12}$$

$$\text{s.t. } (6)\text{–}(8). \tag{13}$$

ν' is a lower bound on ν, since the above optimization problem is a relaxation of OPT as the two optimization problems, (10)–(11), and (12)–(13), are solved separately; i.e., z_a and z_d in general corresponds to different solution vectors. A solution to the first problem can be found as follows: Let

$$I^{\min} = \left\{ i_1^{\min}, \ldots, i_{k^*-1}^{\min}, i_{k^*+1}^{\min}, \ldots, i_{|K|}^{\min} \right\},$$

where $i_k^{\min} = \arg\min\{a_i \colon i \in C^{\cap k}\}$. If a $C^{\cap k} = \varnothing$, then no $i_k^{\min}$ is included. Order $I^{\min}$ nondecreasingly by the value of a_i. A solution is the first $n-1$ elements of $I^{\min}$. A solution to the second problem can be found similarly. Therefore, the running time is $O(|C| + |K| \log |K|)$. In the following this bound is denoted as LB2.

We now show that neither bound is dominated by the other. Consider the following instance of the optimization problem (5)–(8):

$$\begin{aligned} \min \ & 2y_1 + 1y_2 + 1 + \sqrt{1y_1^2 + 4y_2^2 + 1} \\ \text{s.t. } & (6)\text{–}(8). \end{aligned}$$

Here LB1 gives a lower bound of 4 (corresponding to $y_1 = y_2 = 0.5$), while LB2 gives a lower bound of $2 + \sqrt{2}$. For this instance LB1 thus dominates LB2. Now consider the instance:

$$\begin{aligned} \min \ & 1y_1 + 1y_2 + 1 + \sqrt{1y_1^2 + 1y_2^2 + 1} \\ \text{s.t. } & (6)\text{–}(8). \end{aligned}$$

Here LB1 gives the value $2 + \sqrt{1.5}$ (corresponding to $y_1 = y_2 = 0.5$), while LB2 gives a lower bound of $2 + \sqrt{2}$. For this instance LB2 thus dominates LB1.

4. Separation for Cover Inequalities

Given a fractional solution x^*, the separation for cover inequalities is to decide whether there exists a cover C, such that $\sum_{i \in C} x_i^* > |C| - 1$, i.e., a violated cover inequality, and if so, to construct it. Because the separation problem for cover inequalities is $\mathcal{NP}$-hard already for linear knapsack constraints (see Ferreira et al. 1996, Klabjan et al. 1998, Gu et al. 1999), it is also so for conic quadratic knapsack constraints.

In the absence of GUB constraints, as described by Atamtürk and Narayanan (2009), this problem can be answered by solving the minimization problem:

$$\eta = \min \sum_{i\in N}(1-x_i^*)y_i \tag{14}$$

$$\text{s.t.} \sum_{i\in N} a_i y_i + \omega\sqrt{\sum_{i\in N} d_i y_i^2} \geq b+\epsilon, \tag{15}$$

$$y \in \{0,1\}^N, \tag{16}$$

where ϵ is some small positive number. Here $y_i = 1$ if and only if $i \in C$. Note, that because of Constraint (15) the problem may appear to be a nonconvex 0–1 problem, but it can be reformulated as an equivalent quadratic mixed 0–1 problem (see §4.1.1). We have

$$\sum_{i\in C} x_i^* > |C|-1 \iff 1 > \sum_{i\in N}(1-x_i^*).$$

Therefore, if $\eta < 1$, then the optimal solution yields a violated cover inequality. Even if $\eta \geq 1$, a cover has been identified, and an attempt to extend it can be made. After extending the cover, the corresponding extended cover inequality may be violated, even though the original cover inequality is not.

When GUB constraints are present we instead wish to solve the above problem with the following set of constraints added:

$$\sum_{i\in Q_k} y_i \leq 1, \quad \forall k \in K. \tag{17}$$

This ensures that the resulting covers are GUB covers. Again, if $\eta < 1$, a violated cover inequality has been found. In any case, a GUB cover has been found, and Algorithm 1 may be applied.

Atamtürk and Narayanan (2009) solve the separation problem (14)–(16) heuristically based on the rounding continuous relaxation solutions that are derived from KKT conditions. Their approach does not, however, carry over well to the case with GUB constraints. In the following we describe a number of approaches for constructing good candidate GUB covers, which are then to be extended using Algorithm 1 in conjunction with one of the lower bounds previously presented.

4.1. Algorithms for Separating GUB Covers

The algorithms for separating GUB covers should identify a number of good candidate GUB covers for extension. A good candidate for a GUB cover may be one where the corresponding cover inequality is violated, or sufficiently close to being violated, but is also easy to extend.

4.1.1. Separation Algorithm 1. The first approach is to reformulate the separation problem (14)–(17) as an equivalent quadratic mixed 0–1 problem and solve it exactly using a CQIP solver. Observe that (14)–(17) can be restated as:

$$\eta = \min \sum_{i=1}^{n}(1-x_i^*)y_i \tag{18}$$

$$\text{s.t.} \sum_{i=1}^{n} a_i y_i + \omega z \geq b+\epsilon, \tag{19}$$

$$z^2 \leq \sum_{i\in N} d_i y_i, \tag{20}$$

$$\sum_{i\in Q_k} y_i \leq 1 \quad \forall k \in K, \tag{21}$$

$$y \in \{0,1\}^{|N|}, \quad z \geq 0, \tag{22}$$

where z is an auxiliary variable used to break the nonlinear constraint into two more convenient constraints. Because the y_i variables are binary we may perform the substitution $y_i = y_i^2$ for Constraint (20), which makes it convex quadratic. This approach is primarily included as a reference for evaluating the heuristic separation algorithms described next.

4.1.2. Separation Algorithms 2 and 3. Separation Algorithms 2 and 3 are greedy heuristics that attempt to find good solutions quickly to the separation problem. First, the variables within each GUB set are sorted according to a weight calculated on the basis of the current solution. Then, a set C is created containing the $|K|$ largest-weighted variables. If C is not a cover, a new set C is created where the second largest-weighted variable from the GUB sets replaces the current variable iteratively. The algorithm progresses until a cover is found or there are no more variables.

The variables are sorted using the following weights: (1) $w_i = x_i^*$, and (2) $w_i = (x_i^* - 1)/(a_i + \omega\sqrt{d_i})$, where x^* is the fractional solution to cut off. Separation Algorithm 2 uses the first weight function, whereas separation Algorithm 3 uses the second. Weight function 2 is a generalization of the weight function used by Crowder et al. (1983) for the linear case where $d_i = 0$ for all $i \in N$.

Once a cover inequality is found, it is extended using the extension algorithms described in the previous section.

5. Computational Experiments

In this section, we describe the computational experiments conducted to understand the value of using the additional structure imposed by the GUB constraints in conic quadratic MIP as well as to compare the proposed separation and extension algorithms.

5.1. Test Instances

As a test we constructed instances that have the general form:

$$\max \sum_{i\in N} c_i x_i \tag{23}$$

$$\sum_{i\in N} a_{im} x_i + \omega \sqrt{\sum_{i\in N} \sigma_{im}^2 x_i^2} \le b_m \quad m \in M, \tag{24}$$

$$\sum_{i\in Q_k} x_i \le 1 \quad k \in K, \tag{25}$$

$$x \in \{0,1\}^N. \tag{26}$$

In the following let $n = |N|$ and $m = |M|$. For each instance the values of c_i, a_{im}, and σ_{im} are respectively chosen at random based on the uniform distribution across the integer intervals [1; 1,000], [0; 100], and $[0; a_{im}]$. The value for ω is set to 3. The value of b_m is set as $b_m = \beta \cdot (\sum_{i\in S} a_{im} + \omega\sqrt{\sum_{i\in T} \sigma_{im}^2})$, where S is the index-set of variables with the maximal value of a_{im} within each Q_k, and T is likewise the index-set of variables with maximal value of σ_{im} within each Q_k. The GUB-sets, Q_k, are created such that they are disjoint, each set contains a random number of variables in the interval $[0.1 \cdot n; 0.3 \cdot n]$, and such that $\bigcup_{k\in K} Q_k = n$.

For each combination of n in $\{50, 75, 100\}$, m in $\{10, 20\}$ and β in $\{0.3, 0.5\}$, five random instances are generated, giving a total of 60 test instances. These instances along with the source code is available for download at http://or.man.dtu.dk/English/research/.

5.2. Test Setup

For the computational experiments, we use ILOG CPLEX 12.1 (CPLEX), which solves conic quadratic relaxations at the nodes of a branch-and-bound tree. CPLEX heuristics are turned off, and a single thread is used. When comparing to default CPLEX, the MIP search strategy is set to traditional branch-and-bound, rather than the default dynamic search as it is not possible to add user cuts in CPLEX while retaining the dynamic search strategy. When CPLEX is used in connection with a separation algorithm (separation Algorithm 1) or for calculating a bound (OPT and LB1) all settings are left at their default (except for the number of threads, which is set to one).

Experiments were performed on a machine with two Intel(R) Xeon(R) CPUs @ 2.67 Ghz (16 logical cores), with 24 GB of RAM, and running Ubuntu 10.4.

5.3. Cuts

In the following, Sep1(conic), Sep2(x-sort), and Sep3(x/coef-sort) refers to separation Algorithm 1, 2, and 3, respectively, and Exact(conic), LB1(convrelax), and LB2(minsum) refers to solving OPT, and the lower bounds LB1, and LB2, respectively.

Table 1 Table Indicating Whether Cuts Are Applied Only at the Root (*Root*) Node, or Throughout the Branch-and-Bound Tree (*All*)

	Sep1(conic)	Sep2(x-sort)	Sep3(x/coef-sort)
Exact(conic)	Root	Root	Root
LB1(convrelax)	Root	Root	Root
LB2(minsum)	All	All	All

Depending on which combination of separation algorithm (either Sep1(conic), Sep2(x-sort), or Sep3(x/coef-sort)) and lower bound used (either Exact(conic), LB1(convrelax), or LB2(minsum)), cutting is applied either only at the root node, or locally throughout the branch-and-bound tree. Cutting throughout the tree turned out to be effective for the "fast" separation algorithms and lower bound arguments, but for the more computationally expensive algorithms the overhead of cutting at each node was too high. Table 1 lists how cutting is applied for the different combinations.

5.4. Results

We first compare the different combinations of separation algorithms and bounds used in the generic extension algorithm. Next, we examine the effect of extending covers as compared to not extending them, and finally we examine the effect of using the GUB information to extend covers as compared to not using this information.

In Tables 2–7 below, the column **rgap** is the average optimality gap at the root node after addition of cuts. The **rgap** is calculated as $(\mathrm{UB} - \mathrm{LB}^*)/\mathrm{LB}^*$, where UB is the objective value at the root node and LB^* is the objective value of an optimal solution. If no algorithms solve a given instance to optimality within the given time limit of 3,600 secs, then LB^* is the objective value of the best-found solution across all algorithms. To avoid the case $\mathrm{LB}^* = 0$, we add the constant 1 to the objective function. For the combination of separation algorithms and bounds where cutting is only applied at the root node, the column **cuts** is the average number of cover cuts (user cuts) added at the root node, whereas for the combinations where cutting is applied throughout the branch-and-bound tree, the column is the average number of cuts added per node, and the number in parenthesis is the number of cuts added at the root node. The column **nc** is for the average number of nodes explored in the branch-and-bound tree, **rt** the time used in the root node in seconds, and finally, **time** is the average total time used in seconds, where the number in parenthesis shows how many of the five instances are solved to optimality within the time limit. Bold font indicates that all instances are solved to optimality. The Agg. time, Agg. node, and Solved rows, indicate the aggregate time used, the

Table 2 **Results from CPLEX**

n	m	β	CPLEX rgap	cuts	nc	rt	time
50	10	0.3	77.80	—	865	0	**2 (5)**
		0.5	22.87	—	1,737	0	**9 (5)**
	20	0.3	147.59	—	1,335	1	**11 (5)**
		0.5	38.83	—	2,108	0	**83 (5)**
75	10	0.3	80.24	—	2,954	0	**32 (5)**
		0.5	21.12	—	3,594	0	**55 (5)**
	20	0.3	183.40	—	2,338	2	**29 (5)**
		0.5	25.06	—	4,724	2	760 (4)
100	10	0.3	57.69	—	4,664	0	**187 (5)**
		0.5	7.78	—	1,613	0	**13 (5)**
	20	0.3	155.09	—	5,175	3	1,035 (4)
		0.5	23.90	—	9,279	3	2,674 (2)
Agg. time							4,889 (63)
Agg. node							40,386 (2,741)
Solved							55

aggregate number of branch-and-bound nodes visited, and the total number of instances solved, respectively. For Agg. time and Agg. node, the number in parenthesis is the geometric mean.

Comparison of Separation Algorithms and Bounds. The branch-and-bound algorithm is run for each combination of separation algorithm and bound argument. Tables 3–5 contain the results for separation Algorithms 1, 2, and 3, respectively, combined with the different bound choices. Results from CPLEX can be seen in Table 2.

We first consider the CPLEX results. As can be seen from Table 2 all instances could be solved up to $n = 75$, and $m = 10$. One instance cannot be solved for $n = 75$ and $m = 20$, while for $n = 100$ all instances can be solved for $m = 10$, and six instances can be solved for $m = 20$. CPLEX solves a total of 55 instances using, in total, 4,889 seconds and visiting 40,386 nodes.

We next compare the results of each combination of bound with separation Algorithm 1 (Table 3), and compare these to CPLEX (Table 2). Recall that separation Algorithm 1 solved the separation problem to optimality with CPLEX. As can be seen, in general adding cuts using separation Algorithm 1 has a positive effect on the computational time and the number of nodes visisted. For Exact(conic) and LB1(relax) the number of instances solved remains the same (55) but the computational time is reduced to 4,131 and 4,161 seconds, respectively, compared to the 4,889 seconds of CPLEX, and the number of nodes visited drops from 40,384 to 25,146 and 29,372, respectively. For LB2(minsum) the effect of cutting is quite noticable, the total number of solved instances increases to 59, the total computational time is reduced to 1,228 seconds, and the number of nodes visisted falls to 3,116. All combinations improve the root gaps compared to CPLEX. With respect to the root gaps the best combination among the three is, as expected, Sep1 + Exact(conic), but the time spent at the root node is also the largest, which is also as expected. The combination Sep1 + LB2(minsum) produces, in general, better root node gaps than Sep1 + LB1(convrelax) using less time at the root node. Overall, Sep1 + LB2(minsum) performs the best.

We next compare the use of extension lower bounds with separation Algorithm 2 (Table 4). In general considerably more cuts are added at the root node, and as a consequence the root gap is smaller compared to the case with separation Algorithm 1. While this may seem odd, as the separation problem is solved

Table 3 **Results from Combinations of Separation Algorithm 1 and the Different Bounds**

			Sep1(conic) + Exact(conic)					Sep1(conic) + LB1(convrelax)					Sep1(conic) + LB2(minsum)				
n	m	β	rgap	cuts	nc	rt	time	rgap	cuts	nc	rt	time	rgap	cuts	nc	rt	time
50	10	0.3	0.40	35	1	8	**8 (5)**	5.78	40	19	3	**3 (5)**	2.22	45 (42)	4	1	**1 (5)**
		0.5	21.23	6	1,597	4	**12 (5)**	21.11	11	1,663	2	**11 (5)**	21.20	696 (8)	191	1	**12 (5)**
	20	0.3	28.38	55	70	14	**14 (5)**	33.87	50	95	7	**7 (5)**	25.41	91 (51)	16	2	**4 (5)**
		0.5	32.79	24	1,526	11	**51 (5)**	33.48	23	1,970	3	**85 (5)**	32.90	949 (24)	228	1	**28 (5)**
75	10	0.3	36.62	18	1,365	5	**17 (5)**	40.15	21	1,346	3	**14 (5)**	35.48	276 (22)	59	1	**5 (5)**
		0.5	16.68	18	3,104	13	**68 (5)**	17.20	22	3,504	3	**113 (5)**	17.30	1,768 (21)	467	1	**47 (5)**
	20	0.3	16.94	47	72	18	**19 (5)**	34.70	51	136	10	**12 (5)**	20.73	84 (50)	14	4	**5 (5)**
		0.5	20.70	20	5,189	31	822 (4)	22.56	16	4,860	12	780 (4)	22.17	3,296 (14)	686	6	**123 (5)**
100	10	0.3	51.44	14	2,135	6	**16 (5)**	53.08	12	2,638	2	**16 (5)**	52.29	414 (13)	100	1	**9 (5)**
		0.5	5.91	12	1,365	31	**46 (5)**	5.81	17	1,623	5	**47 (5)**	5.69	769 (19)	223	2	**23 (5)**
	20	0.3	87.48	32	816	22	**44 (5)**	91.04	33	883	7	**26 (5)**	92.59	256 (32)	40	3	**9 (5)**
		0.5	22.07	17	7,906	55	3,016 (1)	22.46	14	10,635	15	3,048 (1)	22.11	5,450 (20)	1,087	12	962 (4)
Agg. time							4,131 (47)					4,161 (40)					1,228 (17)
Agg. node							25,146 (684)					29,372 (982)					3,116 (101)
Solved							55					55					59

Table 4 **Results from Combinations of Separation Algorithm 2 and the Different Bounds**

			Sep2(x-sort) + Exact(conic)					Sep2(x-sort) + LB1(convrelax)					Sep2(x-sort) + LB2(minsum)				
n	m	β	**rgap**	**cuts**	**nc**	**rt**	**time**	**rgap**	**cuts**	**nc**	**rt**	**time**	**rgap**	**cuts**	**nc**	**rt**	**time**
50	10	0.3	0.00	70	0	146	**146 (5)**	0.67	120	2	57	**57 (5)**	0.73	96 (95)	1	0	**0 (5)**
		0.5	17.12	52	1,112	155	**161 (5)**	17.54	58	1,175	64	**68 (5)**	17.29	931 (56)	113	1	**1 (5)**
	20	0.3	19.64	129	28	700	**700 (5)**	26.33	192	27	186	**187 (5)**	24.64	194 (126)	17	1	**1 (5)**
		0.5	26.51	92	1,097	400	**496 (5)**	28.43	102	1,742	98	**232 (5)**	27.56	1,290 (97)	140	1	**3 (5)**
75	10	0.3	20.57	113	103	699	**699 (5)**	25.32	175	199	186	**187 (5)**	24.59	375 (145)	33	1	**2 (5)**
		0.5	14.70	55	2,635	271	**400 (5)**	15.30	79	3,191	77	**270 (5)**	14.57	2,592 (82)	238	0	**5 (5)**
	20	0.3	13.52	161	12	998	**998 (5)**	9.52	290	15	460	**460 (5)**	12.43	238 (206)	8	3	**3 (5)**
		0.5	18.13	84	3,881	644	1,412 (4)	19.57	99	4,327	210	1,158 (4)	18.62	5,013 (111)	410	15	**30 (5)**
100	10	0.3	31.58	125	244	1,710	**1,711 (5)**	36.73	231	477	428	**431 (5)**	37.19	652 (160)	48	1	**2 (5)**
		0.5	4.79	35	785	339	**348 (5)**	5.12	48	1,518	63	**215 (5)**	4.92	1,702 (47)	165	2	**5 (5)**
	20	0.3	42.10	277	58	2,481	**2,776 (5)**	39.66	342	109	885	**887 (5)**	44.64	504 (304)	32	6	**7 (5)**
		0.5	20.76	65	6,961	1,101	3,989 (1)	21.03	76	9,223	228	2,925 (2)	20.73	13,624 (85)	860	21	**71 (5)**
Agg. time							13,836 (725)					7,077 (319)					132 (4)
Agg. node							16,915 (241)					22,006 (361)					2,064 (63)
Solved							55					56					60

to optimality for separation Algorithm 1, the reason is that separation Algorithm 1 only attempts to extend the single cover corresponding to the solution of (18)–(22), while separation Algorithm 2 will run through a number of covers, trying to extend each one. Extending the cover corresponding to the solution of (18)–(22) might not result in a violated inequality, while extending some of the covers examined by separation Algorithm 2 might. The numerous covers examined by separation Algorithm 2, also explain why Sep2 + Exact and Sep2 + LB1 spend considerably more time in the root node than their counterparts for separation Algorithm 1. The additional cuts separated by the combinations of Exact(conic), and LB1(convrelax) with separation Algorithm 2 does not, however, outweigh the additional time spent in the root node compared to separation Algorithm 1, and the total computational time increases to 13,836 and 7,077 seconds, respectively, while only a single extra instance is solved for LB1(convrelax). The number of nodes visited is reduced to 16,915 and 22,006, respectively, which is a consequence of the improvement in the root gap. As n grows, we see a clear advantage of using separation Algorithm 2 with LB2(minsum), both compared to separation Algorithm 1 and to CPLEX. This combination solves all 60 instances using only 132 seconds and visiting just 2064 nodes.

Finally considering separation Algorithm 3 (Table 5), we see that the results are very similar to separation Algorithm 2, but the performance is

Table 5 **Results from Combinations of Separation Algorithm 3 and the Different Bounds**

			Sep3(x/coef-sort) + Exact(conic)					Sep3(x/coef-sort) + LB1(convrelax)					Sep3(x/coef-sort) + LB2(minsum)				
n	m	β	**rgap**	**cuts**	**nc**	**rt**	**time**	**rgap**	**cuts**	**nc**	**rt**	**time**	**rgap**	**cuts**	**nc**	**rt**	**time**
50	10	0.3	0.00	69	0	145	**145 (5)**	0.47	108	4	57	**57 (5)**	0.00	106 (106)	0	0	**0 (5)**
		0.5	18.54	32	1,120	169	**175 (5)**	18.97	31	1,560	45	**55 (5)**	20.42	868 (24)	159	0	**1 (5)**
	20	0.3	25.43	91	30	463	**463 (5)**	27.79	157	40	152	**152 (5)**	24.45	212 (138)	17	1	**1 (5)**
		0.5	28.38	64	1,049	398	**420 (5)**	30.81	63	1,737	96	**176 (5)**	29.78	1,111 (62)	166	1	**3 (5)**
75	10	0.3	22.74	102	137	685	**685 (5)**	25.81	164	264	137	**138 (5)**	27.22	373 (146)	38	1	**2 (5)**
		0.5	16.38	42	3,432	289	**672 (5)**	16.39	53	3,517	64	**367 (5)**	16.28	2,381 (54)	432	0	**12 (5)**
	20	0.3	11.59	200	20	1,562	**1,599 (5)**	13.18	298	13	321	**321 (5)**	11.96	295 (246)	15	3	**4 (5)**
		0.5	20.60	53	3,943	710	1,469 (4)	21.35	59	4,500	155	935 (4)	21.26	4,155 (52)	597	7	**40 (5)**
100	10	0.3	35.36	134	253	1,942	**1,944 (5)**	42.36	196	612	254	**258 (5)**	39.33	625 (172)	69	1	**2 (5)**
		0.5	5.17	23	1163	411	**507 (5)**	5.84	23	1,729	56	**769 (5)**	5.75	879 (31)	184	1	**4 (5)**
	20	0.3	44.88	309	123	3,364	**3,784 (5)**	41.38	438	74	731	**732 (5)**	46.13	539 (393)	23	5	**7 (5)**
		0.5	22.76	36	8,416	906	3,810 (1)	22.03	30	8,144	170	3,355 (1)	22.04	12,336 (34)	1,674	13	**270 (5)**
Agg. time							15,673 (807)					7,316 (304)					347 (5)
Agg. node							19,688 (296)					22,193 (405)					3,372 (76)
Solved							55					55					60

Table 6 Comparison of the Best Separation Algorithm Combination with CPLEX

			CPLEX					Sep2(x-sort) + LB2(minsum)				
n	m	β	**rgap**	**cuts**	**nc**	**rt**	**time**	**rgap**	**cuts**	**nc**	**rt**	**time**
50	10	0.3	77.80	—	865	0	**2 (5)**	0.73	96 (95)	1	0	**0 (5)**
		0.5	22.87	—	1,737	0	**9 (5)**	17.29	931 (56)	113	1	**1 (5)**
	20	0.3	147.59	—	1,335	1	**11 (5)**	24.64	194 (126)	17	1	**1 (5)**
		0.5	38.83	—	2,108	0	**83 (5)**	27.56	1,290 (97)	140	1	**3 (5)**
75	10	0.3	80.24	—	2,954	0	**32 (5)**	24.59	375 (145)	33	1	**2 (5)**
		0.5	21.12	—	3,594	0	**55 (5)**	14.57	2,592 (82)	238	0	**5 (5)**
	20	0.3	183.40	—	2,338	2	**29 (5)**	12.43	238 (206)	8	3	**3 (5)**
		0.5	25.06	—	4,724	2	760 (4)	18.62	5,013 (111)	410	15	**30 (5)**
100	10	0.3	57.69	—	4,664	0	**187 (5)**	37.19	652 (160)	48	1	**2 (5)**
		0.5	7.78	—	1,613	0	**13 (5)**	4.92	1,702 (47)	165	2	**5 (5)**
	20	0.3	155.09	—	5,175	3	1,035 (4)	44.64	504 (304)	32	6	**7 (5)**
		0.5	23.90	—	9,279	3	2,674 **(2)**	20.73	13,624 (85)	860	21	**71 (5)**
Agg. time							4,889 (63)					132 (4)
Agg. node							40,386 (2,741)					2,064 (63)
Solved							55					60

slightly worse. This is not so surprising as the only difference between separation Algorithms 2 and 3 is the weight assigned to each variable when these are sorted.

Separation Algorithms 2 and 3 outperform separation Algorithm 1. The main reason is that for separation Algorithm 1, a conic quadratic integer program needs to be solved, which is slow compared to the sorting-based separation Algorithms 2 and 3. Also, many more cuts can be separated per call for Algorithms 2 and 3, as more than one extended cover is attempted. There seems to be a slight advantage to using separation Algorithm 2 over separation Algorithm 3, which seems to imply that the fractionality of a variable is more important than its weight, when attempting to find a violated inequality.

Comparing bounds used for extension algorithms, LB2(minsum) has a clear advantage compared to the others. This is primarily because it is very fast, and therefore, can be used to separate cuts throughout the branch-and-bound tree.

To better illustrate the advantage of utilizing cutting planes, we show in Table 6 the results from CPLEX side-by-side with the best combination, i.e., separation Algorithm 2 and using LB2(minsum) for extending covers.

Effect of Extending Covers. To examine the effect of extending cover inequalities, we compare the results from running the best separation algorithm (separation Algorithm 2), with and without applying the extension algorithm (using the OPT bound) to the covers. The reason for using the OPT bound, even though it is slow, is that it is optimal and should thus better illustrate the root bound quality gained from using extension. Cutting was in both cases only applied at the root node. As can be seen from the results in Table 7 there is a clear gain in quality of the root bound, in the number of cuts added, and in the number of branch-and-bound nodes when covers are extended. The use of the slower exact extension algorithm, however, means that the time spent for cutting at the root node does not translate into a gain in total solution time.

While Table 7 shows the best possible root gap improvement, comparing no extension (Sep2(x-sort) in Table 7) with the most effective way of extending covers using LB2 (Sep2(x-sort) + LB2(minsum) in Table 6), which takes only a total of 132 seconds and solves all sixty instances, it clearly shows the positive effect of extended covers in reducing the solution time dramatically.

Effect of Using GUB Information. To examine the effect of using the GUB information when separating and extending a cover, we perform two experiments: In the first experiment we compare the best separation algorithm and bound argument, i.e., Sep2+LB2(minsum), with an altered version that does not employ any GUB information. In the second experiment we compare Sep2+LB2(minsum), with an implementation of the separation and extension algorithm of Atamtürk and Narayanan (2008).

In relation to the first experiment, separation Algorithm 2 is altered as follows: The variables are still ordered w.r.t. their weight, but this is no longer done within each GUB-set; instead, all variables are ordered in a single list. Instead of creating a candidate cover by selecting the largest weighted variables from each GUB-set, a cover is created by selecting the first l variables from the ordered list, where l is chosen such

Table 7 Results from Running Separation Algorithm 2 With and Without the Extension Algorithm Based on the Exact(Conic) Bound

			Sep2(x-sort) + Exact(conic)					Sep2(x-sort)				
n	*m*	*β*	**rgap**	**cuts**	**nc**	**rt**	**time**	**rgap**	**cuts**	**nc**	**rt**	**time**
50	10	0.3	0.00	70	0	146	**146 (5)**	36.34	45	270	0	**1 (5)**
		0.5	17.12	52	1,112	155	**161 (5)**	22.29	5	1,758	0	**9 (5)**
	20	0.3	19.64	129	28	700	**700 (5)**	77.54	36	669	1	**6 (5)**
		0.5	26.51	92	1,097	400	**496 (5)**	37.86	4	1,926	0	**85 (5)**
75	10	0.3	20.57	113	103	699	**699 (5)**	56.86	35	2,173	1	**21 (5)**
		0.5	14.70	55	2,635	271	**400 (5)**	19.18	10	3,598	0	**58 (5)**
	20	0.3	13.52	161	12	998	**998 (5)**	83.23	80	681	2	**17 (5)**
		0.5	18.13	84	3,881	644	1,412 (4)	24.95	1	5,155	3	887 (4)
100	10	0.3	31.58	125	244	1,710	**1,711 (5)**	55.32	12	3,916	0	**129 (5)**
		0.5	4.79	35	785	339	**348 (5)**	7.10	9	1,512	1	**13 (5)**
	20	0.3	42.10	277	58	2,481	**2,776 (5)**	100.40	64	1,856	4	**85 (5)**
		0.5	20.76	65	6,961	1,101	3,989 (1)	23.60	5	9,140	3	2,836 (2)
Agg. time							13,836 (725)					4,145 (40)
Agg. node							16,915 (241)					32,654 (1,859)
Solved							55					56

Table 8 Results from Running Sep2 + LB2(minsum) With and Without Use of GUB Information

			Sep2(x-sort) + LB2(minsum)					Sep2(x-sort) + LB2(minsum)-GUB				
n	*m*	*β*	**rgap**	**cuts**	**nc**	**rt**	**time**	**rgap**	**cuts**	**nc**	**rt**	**time**
50	10	0.3	0.73	96 (95)	1	0	**0 (5)**	2.68	99 (93)	4	0	**1 (5)**
		0.5	17.29	931 (56)	113	1	**1 (5)**	21.87	1,701 (8)	306	0	**3 (5)**
	20	0.3	24.64	194 (126)	17	1	**1 (5)**	31.97	168 (119)	22	1	**2 (5)**
		0.5	27.56	1,290 (97)	140	1	**3 (5)**	35.65	2,312 (23)	344	1	**6 (5)**
75	10	0.3	24.59	375 (145)	33	1	**2 (5)**	34.67	441 (86)	55	1	**2 (5)**
		0.5	14.57	2,592 (82)	238	0	**5 (5)**	18.91	4,050 (20)	802	0	**13 (5)**
	20	0.3	12.43	238 (206)	8	3	**3 (5)**	24.35	212 (175)	11	4	**4 (5)**
		0.5	18.62	5,013 (111)	410	15	**30 (5)**	23.72	8,042 (16)	1,051	6	**65 (5)**
100	10	0.3	37.19	652 (160)	48	1	**2 (5)**	47.91	835 (77)	104	1	**3 (5)**
		0.5	4.92	1,702 (47)	165	2	**5 (5)**	6.15	1,922 (23)	471	1	**11 (5)**
	20	0.3	44.64	504 (304)	32	6	**7 (5)**	59.49	481 (236)	38	6	**7 (5)**
		0.5	20.73	13,624 (85)	860	21	**71 (5)**	23.19	28,468 (12)	3,279	6	**733 (5)**
Agg. time							132 (4)					849 (7)
Agg. node							2,064 (63)					6,486 (137)
Solved							60					60

that the selected variables are a cover. Iteratively, the largest weighted variable is removed from the cover, and in the order of the list, new variables are included until the selected variables are again a cover. This process continues until the end of the list is reached. Each cover generated in this way is extended using an altered LB2, where $I^{\min}$ contains all variables of the cover instead of the minimal element within each GUB-set of the cover.

As can be seen from the results in Table 8, using GUB information when separating and extending cuts gives a clear gain: the average root gaps are lower when using GUB information, and the total running time is improved from 849 seconds to 132 seconds.

To get further indication of the usefulness of exploiting GUB information, we conduct a second experiment, where we compare Sep2 + LB2(minsum), with an implementation of the separation and extension algorithm of Atamtürk and Narayanan (2008), that does not make use of GUB information. We do not include their advanced lifting procedure, but only their extension algorithm. Cuts are applied throughout the branch-and-bound tree for both algorithms. As it can be seen from the results in Table 9, there is again a clear gain due to employing GUB information when separating and extending cuts. These two comparisons clearly show the positive effect of using GUB information in extended cover cuts.

6. Conclusion

We have investigated using the special structure of GUB constraints in separating and extending cover

Table 9 Comparison of Sep2 + LB2(minsum) with Atamtürk and Narayanan (2008)

			Sep2(x-sort) + LB2(minsum)					Sep2(x-sort) + LB2(minsum)-GUB				
n	m	β	rgap	cuts	nc	rt	time	rgap	cuts	nc	rt	time
50	10	0.3	0.73	96 (95)	1	0	**0 (5)**	51.49	227 (24)	30	0	**1 (5)**
		0.5	17.29	931 (56)	113	1	**1 (5)**	22.85	1,193 (1)	441	0	**7 (5)**
	20	0.3	24.64	194 (126)	17	1	**1 (5)**	76.95	342 (30)	28	2	**4 (5)**
		0.5	27.56	1,290 (97)	140	1	**3 (5)**	38.09	1,861 (1)	485	0	**17 (5)**
75	10	0.3	24.59	375 (145)	33	1	**2 (5)**	73.67	1,065 (8)	163	1	**11 (5)**
		0.5	14.57	2,592 (82)	238	0	**5 (5)**	19.59	3,313 (9)	1,161	0	**60 (5)**
	20	0.3	12.43	238 (206)	8	3	**3 (5)**	114.11	412 (43)	26	3	**7 (5)**
		0.5	18.62	5,013 (111)	410	15	**30 (5)**	24.95	6,021 (1)	1,433	3	**179 (5)**
100	10	0.3	37.19	652 (160)	48	1	**2 (5)**	55.68	2,011 (8)	277	1	**37 (5)**
		0.5	4.92	1,702 (47)	165	2	**5 (5)**	7.46	1,290 (3)	489	1	**41 (5)**
	20	0.3	44.64	504 (304)	32	6	**7 (5)**	123.24	1,215 (31)	69	6	**35 (5)**
		0.5	20.73	13,624 (85)	860	21	**71 (5)**	23.78	21,045 (2)	3,405	3	**1,411 (5)**
Agg. time							132 (4)					1,811 (24)
Agg. node							2,064 (63)					8,007 (245)
Solved							60					60

inequalities for optimization problems with conic quadratic knapsack constraints. We have proposed a number of separation and extension algorithms, and compared them computationally. Our experiments show that relatively simple separation and extension algorithms, that employ the GUB constraints, can speed up the solution time of conic quadratic MIPs with GUB constraints substantially. Fast separation, and extension algorithms are an advantage as they make it possible to cut locally throughout the branch-and-bound tree as opposed to only in the root node.

As a theoretical contribution we have shown that the problem of deciding if a cover can be extended with a single variable is $\mathcal{NP}$-hard, and have established the nondominance between two bounds: one based on a convex relaxation (LB1) and the other based on a decomposition (LB2).

A direction for further research is to consider the more general lifting problem, and investigate how (approximate) lifting coefficients could be calculated for variables not in the current cover, taking into account the GUB constraints.

References

Atamtürk A (2005) Cover and pack inequalities for (mixed) integer programming. *Ann. Opre. Res.* 139:21–38.

Atamtürk A, Narayanan V (2008) Polymatroids and risk minimization in discrete optimization. *Oper. Res. Lett.* 36:618–622.

Atamtürk A, Narayanan V (2009) The submodular 0–1 knapsack polytope. *Discrete Optim.* 6:333–344.

Atamtürk A, Narayanan V (2010) Conic mixed-integer rounding cuts. *Math. Programming* 122:1–20.

Atamtürk A, Narayanan V (2011) Lifting for conic mixed-integer programming. *Math. Programming* 126:351–363.

Balas E (1975) Facets of the knapsack polytope. *Math. Programming* 8:146–164.

Boyd S, Vandenberghe L (2004) *Convex Optimization* (Cambridge University Press).

Büsing C, Koster AMCA, Kutschka M (2011) Recoverable robust knapsacks: The discrete scenario case. *Optim. Lett.* (To appear).

Cezik MT, Iyengar G (2005) Cuts for mixed 0–1 conic programming. *Math. Programming* 104:179–202.

Crowder H, Johnson EL, Padberg M (1983) Solving large-scale 0–1 linear programming problems. *Oper. Res.* 31:803–834.

Ferreira CE, Martin A, Weismantel R (1996) Solving multiple knapsack problems by cutting planes. *SIAM J. Optim.* 6:858–877.

Fujishige S (2005) *Submodular Functions and Optimization*, Vol. 58. (Elsevier Science Ltd.)

Gu Z, Nemhauser GL, Savelsbergh MWP (1998) Lifted cover inequalities for 0–1 integer programs: Computation. *INFORMS J. Comput.* 10:427–437.

Gu Z, Nemhauser GL, Savelsbergh MWP (1999) Lifted cover inequalities for 0–1 integer programs: Complexity. *INFORMS J. Comput.* 11:117–123.

Hammer PL, Johnson EL, Peled UN (1975) Facet of regular 0–1 polytopes. *Math. Programming* 8:179–206.

Hartvigsen D, Zemel E (1992) The complexity of lifted inequalities for the knapsack problem. *Discrete Appl. Math.* 39:113–123.

Iwata S (2008) Submodular function minimization. *Math. Programming* 112:45–64.

Johnson EL, Padberg MW (1981) A note of the knapsack problem with special ordered sets. *Oper. Res. Lett.* 1:18–22.

Kaparis K, Letchford AN (2010) Cover inequalities. Technical report, Lancaster University.

Karp RM (1972) Reducibility among combinatorial problems. Miller RE, Thatcher JW, eds. *Complexity of Computer Computations* (Plenum Press), 85–103.

Klabjan D, Nemhauser GL, Tovey C (1998) The complexity of cover inequality separation. *Oper. Res. Lett.* 23:35–40.

Klopfenstein O, Nace D (2009) Valid inequalities for a robust knapsack polyhedron–application to the robust bandwidth packing problem. *Proc. Internat. Network Optim. Conf. INOC*. (To appear in Networks).

Nemhauser GL, Vance PH (1994) Lifted cover facets of the 0–1 knapsack polytope with GUB constraints. *Opre. Res. Lett.* 16: 255–263.

Wolsey LA (1975) Faces for a linear inequality in 0–1 variables. *Math. Programming* 8:165–178.

Wolsey LA (1990) Valid inequalities for 0–1 knapsacks and mips with generalised upper bound constraints. *Discrete Appl. Math.* 29:251–261.

Zemel E (1989) Easily computable facets of the knapsack polytope. *Math. Oper. Res.* 14:760–764.

Author Queries

A1 Au: Please edit running head to shorter length.

A2 Au: Ok?

A3 Au: Ok?

A4 Au: Ok?

A5 Au: Ok?

A6 Au: Ok?

A7 Au: Ok?

A8 Au: City?

A9 Au: More info?

A10 Au: City?

A11 Au: City?

A12 Au: Update.