Separation and extension of cover inequalities for second-order conic knapsack constraints with GUBs

Atamtürk, Alper; Muller, Laurent Flindt; Pisinger, David

Publication date:
2011

Document Version
Publisher's PDF, also known as Version of record

Separation and extension of cover inequalities for second-order conic knapsack constraints with GUBs

Report 6.2011

DTU Management Engineering

Laurent Flint Muller
Alper Atamtürk
David Pisinger
March 2011

DTU Management Engineering
Department of Management Engineering

Separation and extension of cover inequalities for second-order conic knapsack constraints with generalized upper bounds

Alper Atamtürk* Laurent Flindt Muller† David Pisinger†

*Department of Industrial Engineering and Operations Research,
University of California, Berkeley, CA 94720-1777, USA
atamturk@berkeley.edu

†Department of Management Engineering, Technical University of Denmark,
Produktionstorvet, Building 426, DK-2800 Kgs. Lyngby, Denmark
lafm@man.dtu.dk, pisinger@man.dtu.dk

Abstract

We consider the second-order conic equivalent of the classic knapsack polytope where the variables are subject to generalized upper bound constraints. We describe and compare a number of separation and extension algorithms which make use of the extra structure implied by the generalized upper bound constraints in order to strengthen the second-order conic equivalent of the classic cover cuts. We show that determining whether a cover can be extended with a variable is $\mathcal{NP}$-hard. Computational experiments are performed comparing the proposed separation and extension algorithms. These experiments show that applying these extended cover cuts can greatly improve solution time of second-order cone programs.

1 Introduction

We consider the second-order conic equivalent of the classic knapsack polytope, where the variables are subject to so-called *generalized upper bound* (GUB) constraints. In the following we first define what is understood by GUB constraints, and then define the second-order conic knapsack polytope.

Let N be a finite index set and let $Q_1, \ldots, Q_{|K|}$ be a division of N into $|K|$ independent sets. i.e., $\bigcup_{k\in K} Q_k = N$, and $Q_i \cap Q_j = \emptyset, \forall i, j \in K, i \neq j$. GUB constraints are a set of constraints of the form

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

where $x \in \{0,1\}^{|N|}$. In the following we will also refer to the sets $Q_1, \ldots, Q_{|K|}$ as *GUB-sets*.

For a subset $S \subseteq N$ and $k \in K$, define $S^{\cap k} := S \cap Q_k$ and $S^{\setminus k} := S \setminus Q_k$, and for some $v \in \mathrm{R}^{|N|}$ define $v(S) := \sum_{i\in S} v_i$. For a binary vector $x \in \{0,1\}^{|N|}$, define $S_x := \{i \in N : x_i = 1\}$. Let $f : 2^{|N|} \to \mathrm{R}$ be a set function defined as

$$f(S) := a(S) + \omega\sqrt{d(S)},$$

where $a \in \mathrm{R}_0^{+|N|}, d \in \mathrm{R}_0^{+|N|}$, and $\omega \geq 0$. The polytope considered here is

$$X := \left\{ x \in \{0,1\}^{|N|} : f(S_x) \leq b, \sum_{i\in Q_k} x_i \leq 1, \forall k \in K \right\},$$

where $b \geq 0$. We denote X the second-order conic knapsack polytope with GUB constraints. We use the term second-order conic because the constraint $f(S_x) \leq b$ is equivalent to the second-order cone constraint: $ax + \omega\|Dx\|_2 \leq b$, where $D_{ii} = \sqrt{d_i}$, and $D_{ij} = 0$ for $i \neq j$.

The motivation for considering the function f is that constraints of the form $f(S_x) \leq b$ arise when modelling certain types of chance-constraints of the form: $\mathbf{prob}\,(ax \leq b) \geq \epsilon$, where a is an n-vector of random variables, x is an n-vector of binary variables, $b \in \mathrm{R}$, and $\epsilon \in [0, 1]$. If each variable a_i is normally distributed with mean μ_i and variance σ_i^2, and $\epsilon \geq \frac{1}{2}$, the above chance-constraint can be formulated as the second-order cone constraint (see e.g. Boyd and Vandenberghe (2004)):

$$\sum_{i=0}^{n} \mu_i x_i + \Phi^{-1}(\epsilon)\sqrt{\sigma_i^2 x_i^2} \leq b,$$

where Φ is the cumulative distribution function. Since $x_i^2 = x_i$ for binary variables, the above is equivalent to $f(S_x) \leq b$ with $a = (\mu_1, \ldots, \mu_n)$, $d = (\sigma_1^2, \ldots, \sigma_n^2)$, and $\omega = \Phi^{-1}(\epsilon)$.

The motivation for considering GUB constraints is the same as for the linear case: GUB constraints may be used the strengthen cover inequalities. Note that if one changes the "$\leq$" to "$=$" in the GUB constraints the techniques described here are still applicable.

As mentioned earlier, we focus on cuts derived from X. The literature pertaining to these kind of cuts is quite sparse: Atamtürk and Narayanan (2010) and Cezik and Iyengar (2005) describe rounding cuts, Atamtürk and Narayanan (2009a) describe lifting procedures and Atamtürk and Narayanan (2009b) consider the sub-modular knapsack polytope, which is the same as the polytope considered here except that there are no GUB constraints, and f need only be sub-modular. For this polytope, the authors describe the conic equivalent of cover inequalities known from mixed integer linear programming, and present a heuristic for separating them based on a LP-relaxation of the separation problem. They additionally describe procedures for extending and lifting cover inequalities in order to strengthen them. As mentioned the polytope considered by the authors does not include GUB constraints, and this work can be seen as an extension to the case where GUB constraints are present.

Cover inequalities for linear knapsack constraints was introduced independently by Balas (1975), Hammer et al. (1975), and Wolsey (1975). Both Balas (1975) and Wolsey (1975) treat the lifting of such 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, these 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 examined by a number of people: 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 NP-hard, while Crowder et al. (1983) have shown that the problem is equivalent to solving a knapsack problem. A number of exact and heuristic methods exists for solving the separation problem, see for instance Gu et al. (1998) for an investigation of algorithmic and implementational issues with respect to branch-and-cut algorithms. For some recent surveys, see for instance Atamtürk (2005) or Kaparis and Letchford (2010).

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 knapsack constraints in the presence of GUB constraints. As a theoretical results we show that the problem of determining whether a cover may be extended with a variable is $\mathcal{NP}$-hard. Through computational experiments on a set of generated test instances the different proposed algorithms are compared with respect to time used and bounds produced. We show that the application of extended cover inequalities can greatly improve the solution time of second-order cone programs.

The outline of the remaining paper is as follows, in Section 2 covers, extended covers, and extended covers under the presence of GUB constraints are introduced, in Section 3 a number of algorithms for extending covers are proposed, in Section 4 separation of covers are introduced, and a number of separation algorithms are proposed, in Section 5 the efficiency of the proposed algorithms are evaluated on a set of generated test instances, we conclude in Section 6.

2 Cover inequalities

A subset $C \subseteq N$ is called a *cover* for X if $f(C) > b$. A cover C is called a *minimal cover* if the above property is not satisfied for any $C' \subset C$. If C is such that $|C^{\cap k}| \leq 1, \forall k \in K$, we call it a *base cover*. Given a cover C, the following inequality is valid for X (see Atamtürk and Narayanan (2009b))

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

Example. *Consider the polytope*

$$\left\{ \begin{array}{l} 3x_1 + 4x_2 + 2x_3 + 3x_4 + 1x_5 + \sqrt{2x_1 + 1x_2 + 2x_3 + 1x_4 + 10x_5} \leq 7 \\ x_1 + x_2 \leq 1, \quad x_3 + x_4 + x_5 \leq 1 \\ x_1, \ldots, x_5 \in \{0, 1\}. \end{array} \right\}$$

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

2.1 Extended cover inequalities

Given a cover C, the corresponding cover inequality may be strengthened by including additional variables. When chosen appropriately these variables will add to the left-hand side of (1) without raising the right-hand side. The process of adding variables to an existing cover is called *extending* the cover and can be seen as a lifting procedure where lifting coefficients may only take values 0 or 1. Atamtürk and Narayanan (2009b) describe a procedure for extending a minimal cover, when no GUB constraints are present. This procedure may still be used when GUB constraints are present, but the resulting cover inequalities may be weaker, than if GUB constraints are taken into account.

2.2 Extended cover inequalities with GUB constraints

We now describe how GUB constraints can be used to strengthen cover inequalities. For some $n \geq 0$, and subset $S \subseteq N$, define $\mathcal{W}(S, n) := \{T \subseteq S : |T| \geq n \wedge |T^{\cap k}| \leq 1, \forall k \in K\}$, 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 *extended cover* of size $n \geq 0$ if S is a cover $\forall S \in \mathcal{W}(C, n)$. An extended cover is called *minimal* if C is not an extended cover for any $n' < n$. Note that a base cover C is an extended cover of size $|C|$.

Proposition 1. *If C is an extended cover of size n, 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 $f(S_x) \leq f(T_x) \leq b$. □

Example (continued). *Assume the set C is extended with the element x_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 an extended cover of size $n = 2$ and the inequality $x_1 + x_2 + x_4 \leq 1$ is thus valid. C'' is also an extended cover of size $n = 3$, but C'' would then not be minimal resulting in the weaker inequality $x_1 + x_2 + x_4 \leq 2$.*

Proposition 2. *Let C be an extended cover and let $i^* \in Q_k \backslash C$ for some $k^* \in K$, be such that*

$$f(S \cup \{i^*\}) = a(S) + a_{i^*} + \omega \sqrt{d(S) + d_{i^*}} > b, \quad \forall S \in \mathcal{W}(C^{\backslash k^*}, n - 1), \tag{2}$$

then $C \cup \{i^*\}$ is an extended 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 $S \in \mathcal{W}(C^{\setminus k^*}, n-1)$, and thus $f(T) = f(S \cup \{i^*\}) > b$, and T is thus a cover. □

Proposition 2 suggest a method for extending a cover: Start with identifying a base cover, create some ordering of the variables currently not in the cover, then one at a time check if one of the variables not in the cover can be included by evaluating condition (2). This may be done by solving the following optimization problem:

$$OPT: \qquad \begin{array}{rl} \nu = \min & a(S) + a_{i^*} + \omega\sqrt{d(S) + d_{i^*}} \\ s.t. & S \in \mathcal{W}(C^{\setminus k^*}, n-1). \end{array}$$

If $\nu > b$, the variable can be added to the extended cover. OPT is the constrained minimization of a submodular function. For surveys of submodular function minimization we refer to Fujishige (2005), and Iwata (2008).

Example (continued). *Consider again the extended cover $C'' = \{1, 2, 4\}$. We have two GUB-sets: $Q_1 = \{1, 2\}$ and $Q_2 = \{3, 4, 5\}$. Assume we are considering extending C'' with the variable x_5. In this case $i^* = 5$ and $k^* = 2$. We have $\mathcal{W}(C''^{\setminus 2}, 1) = \{\{1\}, \{2\}\}$, and since $3 + 1 + \sqrt{2 + 10} > 7$, and $4 + 1 + \sqrt{1 + 10} > 7$ the cover may be extended with x_5.*

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

$$\min \quad \sum_{i \in C^{\setminus k^*}} a_i y_i + a_{i^*} + \omega \sqrt{\sum_{C^{\setminus k^*}} d_i y_i + d_{i^*}} \tag{3}$$

$$s.t. \quad \sum_{i \in C^{\cap k}} y_i \leq 1 \qquad \forall k \in K, k \neq k^* \tag{4}$$

$$\sum_{i \in C^{\setminus k^*}} y_i \geq n - 1 \tag{5}$$

$$y_i \in \{0, 1\} \qquad \forall i \in C^{\setminus k^*}, \tag{6}$$

where $y_i = 1$ if and only if the set S contains the ith element. Constraints (4) ensure that S contains at most one element from each GUB-set, and Constraints (5) ensure that S constains at least $n - 1$ elements.

Proposition 3. *Solving the optimization problem* (3)-(6) *is $\mathcal{NP}$-hard.*

Proof. For ease the exposition, let $\mathcal{I} = C^{\setminus k^*} = \{1, \ldots, p\}$, let $\mathcal{K} = K \setminus \{k^*\}$, 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

$$P: \qquad \begin{array}{lll} \min & \sum_{i=1}^p a_i y_i + \omega\sqrt{\sum_{i=1}^p d_i y_i} & \\ s.t. & \sum_{i \in \mathcal{Q}_k} y_i \leq 1 & k \in \mathcal{K} \\ & \sum_{i=1}^p y_i \geq m & \\ & y_i \in \{0, 1\} & \forall i = 1, \ldots p. \end{array}$$

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 $\sum_{i=1}^p a_i y'_i + \omega\sqrt{d_i y'_i} \leq l + D$.

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

$$P': \quad \begin{array}{ll} \sum_{i=1}^{p} a_i y_i + \omega\sqrt{\sum_{i=1}^{p} d_i y_i} + s = E & \\ \sum_{i\in\mathcal{Q}_k} y_i \leq 1 & k \in \mathcal{K} \\ \sum_{i=1}^{p} y_i \geq m & \\ y_i \in \{0,1\} & \forall i = 1,\ldots p \\ 0 \leq s \leq D. & \end{array}$$

The variable s is a slack variable, and since $u - l = D$ we can restrict s to be between 0 and D. Other cases of E are treated as follows: if $E < l$, we answer "no", while if $E > l + D$ we answer "yes" returning y' 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':

$$\begin{array}{ll} \sum_{i=1}^{q} 2Dw_i y_i + \omega\sqrt{\sum_{i=1}^{q} w_i y_{q+i}} + s = 2DC + \sqrt{C} & \\ y_i + y_{k+i} \leq 1 & k \in \mathcal{K} \\ \sum_{i=1}^{2q} y_i \geq q & \\ y_i \in \{0,1\} & \forall i = 1,\ldots p \\ 0 \leq s \leq D. & \end{array}$$

The constraints $y_i + y_{q+i} \leq 1$ and $\sum_{i=1}^{2q} y_i \geq q$ 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

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

This means that if

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

then both the following constraints are satisfied

$$\begin{array}{l} \sum_{i=1}^{q} w_i y_i = C \\ \sqrt{\sum_{i=1}^{q} w_i y_{q+i}} + s = \sqrt{C}. \end{array} \tag{7}$$

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

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

both of which are contradictions.

But the first equation of (7) 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 section 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 variable x_i can be added to the extended cover C. Given a lower bound LB on ν, the variable may be added 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, were not. There is thus a trade-off between the time spend extending the covers, and the strength of the resulting cover inequalities.

We now give a generic extension algorithm, which can be used with any procedure giving a lower bound on ν, starting with some initial base cover, C, of size $n = |C|$. In the following, unless otherwise stated, we will 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 $LB(C, i^*)$ gives a lower bound on OPT. Let $I = N \backslash C$, and assume that I is given some ordering. Different orderings will result in different extended covers. As the final aim is to find a violated cover inequality, and variables with a large value in the current solution is beneficial in this regard, the set I is sorted non-increasingly w.r.t. this value. The generic extension algorithm is shown in Algorithm 1.

Algorithm 1 Generic algorithm for extending a base cover C

Require: The initial base cover C to be extended.
  Let $I = N \backslash C$.
  Sort I non-increasingly w.r.t. the value of corresponding variables.
  for all $i^* \in I$ **do**
    $LB \leftarrow LB(C, i^*)$.
    if $LB > b$ **then**
      $C \leftarrow C \cup \{i^*\}$.
    end if
  end for
  return C

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 lower bounding approaches. Any of these approaches can be used for calculating $LB(C, i^*)$ in Algorithm 1.

3.1 Optimal

As we saw in the previous section OPT can be formulated as a CQIP. The resulting problem, is the minimization of a sub-modular function over a convex set. Atamtürk and Narayanan (2008) treat such a minimization problem using a cutting plane approach. For the computational experiments, we do however not employ this approach, but instead give the above model to a CQIP solver. 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 relaxing the CQIP (3) - (6) by allowing the y_i's to take fractional values. In the following we denote this bound by $LB1$.

3.3 Lower bound 2

Consider the minimization problem:

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

where

$$z_a = \min \sum_{i \in C^{\setminus k^*}} a_i y_i + a_{i^*} \tag{8}$$

$$s.t. \sum_{i \in C^{\cap k}} y_i \leq 1 \qquad \forall k \in K, k \neq k^* \tag{9}$$

$$\sum_{i \in C^{\setminus k^*}} y_i \geq n - 1 \tag{10}$$

$$y_i \in \{0, 1\} \qquad \forall i \in C^{\setminus k^*}, \tag{11}$$

and

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

$$s.t. \sum_{i \in C^{\cap k}} y_i \leq 1 \qquad \forall k \in K, k \neq k^* \tag{13}$$

$$\sum_{i \in C^{\setminus k^*}} y_i \geq n - 1 \tag{14}$$

$$y_i \in \{0, 1\} \qquad \forall i \in C^{\setminus k^*}. \tag{15}$$

ν' is a lower bound on ν, since the above optimization problem is a relaxation of OPT. The two optimization problems, (8) - (11), and (12) - (15), are solved independently of each other. A solution to the first problem can be found as follows in: Let $I^{min} = \{i_1^{min}, \ldots, i_{k^*-1}^{min}, i_{k^*+1}^{min}, \ldots, i_{|K|}^{min}\}$, where $i_k^{min} = \arg\min\{a_i : i \in C^{\cap k}\}$. If a $C^{\cap k} = \emptyset$, then no i_k^{min} is included. Order I^{min} non-decreasingly 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. The running time is $O(|I^{min}| \log |I^{min}|)$. In the following this bound is denoted $LB2$.

We now show that $LB1$ is stronger than $LB2$, i.e., $LB2 \leq LB1$.

Lemma 1. *Let μ^* be the optimal solution to the the minimization problem:*

$$\min \sum_{i \in \mathcal{I}} f_i \mu_i$$

$$s.t. \sum_{i \in \mathcal{Q}_k} \mu_i \leq 1 \qquad \forall k \in \mathcal{K}$$

$$\sum_{i \in \mathcal{I}} \mu_i \geq m$$

$$\mu_i \in \{0, 1\} \qquad \forall i \in \mathcal{I},$$

where $f_i \geq 0 \, \forall i \in \mathcal{I}$, $m \geq 0$, $\mathcal{Q}_k \cap \mathcal{Q}_{k'} = \emptyset \, \forall k, k' \in \mathcal{K} : k \neq k'$, and $\bigcup_{k \in \mathcal{K}} \mathcal{Q}_k = \mathcal{I}$. Then for any fractional solution $\tilde{\mu}$: $f\mu^* \leq f\tilde{\mu}$.

Proof. Let $\tilde{\mu}$ be the optimal solution, where the integer constraints have been relaxed. It is enough to show that $f\mu^* = f\tilde{\mu}$. We can assume there are at least least two fractional variables, since otherwise, the single fractional variable can be fixed to 0, producing an integer solution, μ^*, with $f\mu^* \leq f\tilde{\mu}$ because $f_i \geq 0 \, \forall i \in \mathcal{I}$, and because of the optimality of $\tilde{\mu}$ we have $f\mu^* = f\tilde{\mu}$.

Let $\tilde{\mu}_i, \tilde{\mu}_j$, be two fractional variables. Assume w.l.o.g. that $f_i \leq f_j$. Let $\epsilon = \min\{1 - \tilde{\mu}_i, \tilde{\mu}_j\}$. Then updating $\tilde{\mu}_i := \tilde{\mu}_i + \epsilon$, and $\tilde{\mu}_j = \tilde{\mu}_j - \epsilon$ produces a new feasible solution, $\tilde{\mu}'$, with $f\tilde{\mu}' \leq f\tilde{\mu}$ and at least one less fractional variable. Again because of the optimality of $\tilde{\mu}$ we have $f\tilde{\mu}' = f\tilde{\mu}$. Iterating this process produces an integer solution. □

Proposition 4. $LB2 \leq LB1$.

Proof. For ease of exposition assume $a_{i^*} = d_{i^*} = 0$. This assumption does not affect the correctness of the proof. Let $\mathcal{I} = C^{\setminus k^*}$, let $\tilde{y}$ be a (fractional) solution to the optimization problem corresponding to $LB1$, and let y^a, and y^d be (integer) solutions to the two optimization problems corresponding to $LB2$. Assume $LB1 < LB2$. We have

$$LB1 = \sum_{i\in\mathcal{I}} a_i\tilde{y}_i + \omega\sqrt{\sum_{\mathcal{I}} d_i\tilde{y}_i} < \sum_{i\in\mathcal{I}} a_i y_i^a + \omega\sqrt{\sum_{i\in\mathcal{I}} d_i y_i^d} = LB2 \iff$$

$$\sum_{i\in\mathcal{I}} a_i\tilde{y}_i - \sum_{i\in\mathcal{I}} a_i y_i^a < \omega\sqrt{\sum_{i\in\mathcal{I}} d_i y_i^d} - \omega\sqrt{\sum_{i\in\mathcal{I}} d_i\tilde{y}_i}.$$

Because of Lemma 1 we have $\sum_{i\in\mathcal{I}} a_i\tilde{y}_i - \sum_{i\in\mathcal{I}} a_i y_i^a \geq 0$, while again because of Lemma 1 and because the square root function is increasing, $\omega\sqrt{\sum_{i\in\mathcal{I}} d_i y_i^d} - \omega\sqrt{\sum_{i\in\mathcal{I}} d_i\tilde{y}_i} \leq 0$, which is a contradiction. □

4 Separation of cover inequalities

Disregarding GUB constraints, the separation of a cover inequality is the process, when given a fractional solution x^*, to find a cover C, such that $\sum_{i\in C} x_i^* > |C| - 1$, i.e., a violated cover inequality. As described by Atamtürk and Narayanan (2009b), a violated cover inequality can be separated (if one exists) by solving the minimization problem:

$$\eta = \min \sum_{i=1}^{n} (1 - x_i^*) y_i \tag{16}$$

$$s.t. \sum_{i=1}^{n} a_i y_i + \omega\sqrt{\sum_{i\in N} d_i y_i} \geq b + \epsilon \tag{17}$$

$$y \in \{0,1\}^{|N|}, \tag{18}$$

where ϵ is some small positive number. If $\eta < 1$, then a violated cover inequality has been found. Even though $\eta \geq 1$, a cover has been identified, and an attempt at extending the cover can be made. After extending the cover, the corresponding extended cover inequality may be violated, even though the original cover inequality was not.

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

$$\sum_{i\in Q_k} y_i \leq 1, \quad \forall k \in K. \tag{19}$$

This ensures that the resulting covers are base covers. Again if $\eta < 1$, a violated cover inequality has been found. In any case, a base cover has been found, and the earlier described generic extension algorithm coupled with a bound may by applied.

Atamtürk and Narayanan (2009b) solve the separation problem (16) - (18) heuristically based on the rounding of solutions to an LP-relaxation of an equivalent problem. Their approach does however not carry over well to the case with GUB constraints. In the following we describe a number of approaches for constructing good candidate base covers, which are to then to be extended using the genetic extension algorithm described earlier (see Algorithm 1) in conjunction with one of the lower bounds previously prestented.

4.1 Algorithms for separating base covers

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

4.1.1 Separation algorithm 1

For the first separation algorithm, we simply solve the separation problem (16) - (19), by giving it to a CQIP solver. The problem (16) - (19) is however not a CQIP in its present form and is thus reformulated as follows before being given to the solver:

$$\eta = \min \sum_{i=1}^{n} (1 - x_i^*) y_i \tag{20}$$

$$s.t. \sum_{i=1}^{n} a_i y_i + \omega z \geq b + \epsilon \tag{21}$$

$$z^2 \leq \sum_{i \in N} d_i y_i \tag{22}$$

$$\sum_{i \in Q_k} y_i \leq 1 \qquad \forall k \in K \tag{23}$$

$$y \in \{0,1\}^{|N|}, \quad z \geq 0, \tag{24}$$

where we have introduced the variable z. Because the separation problem for classic knapsack constraints is $\mathcal{NP}$-hard(see Ferreira et al. (1996), Klabjan et al. (1998), and Gu et al. (1999)), the above problem is likewise $\mathcal{NP}$-hard, and the problem can thus be computationally cumbersome, and is primarily included to evaluate the remaining separation algorithms. The separation algorithm is depicted in Algorithm 2.

Algorithm 2 Heuristic for finding violated cover inequalities

Require: Current solution x^*.
  Solve the problem (20) - (24) by the use of CQIP solver.
  if feasible solution found **then**
    Extend the found cover C using the generic extension algorithm and one of the bounds.
    if C constitutes a violated cover inequality **then**
      return C
    end if
  end if
  return no cover found.

4.1.2 Separation algorithm 2+3

This separation algorithm orders the variables, within each Q_k, by a weight calculated on the basis of the current fractional 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 some Q_k replaces the current variable from the same set and so on. The algorithm progresses until a cover is found or there are no more variables. Let x^* be the value of the current solution and let $w(x_i)$ be the weight associated with the variable x_i. We investigate two different weight functions: 1) $w(x_i) = x_i^*$, and 2) $w(x_i) = (x_i^* - 1)/(a_i + \omega\sqrt{d_i})$, giving rise to separation algorithm 2, and 3 respectively. Crowder et al. (1983) use a weight-function similar to 2) for the linear case. Algorithm 3 gives the details of these algorithms.

Algorithm 3 Heuristic for finding violated cover inequalities

Let C be the largest-weighted variable from each Q_j w.r.t. the current fractional solution x^* and weight function w.
Mark the variables in C.
while there are unmarked variables **do**
  if C is a cover **then**
    Extend C.
    if C constitutes a violated cover inequality **then**
      return C
    end if
  end if
  Add to C the largest-weighted unmarked variable and remove from C the variable from the same Q_j as the newly added variable.
  Mark the newly added variable.
end while
return no cover found.

5 Computational experiments

5.1 Test instances

In order to evaluate the different algorithms, a number of test instances are generated of the form:

$$\max \sum_{i \in N} c_i x_i \tag{25}$$

$$\sum_{i \in N} a_{im} x_i + \omega \sqrt{\sum_{i \in N} d_{im}^2 x_i} \leq b_m \qquad m = 1, \ldots, M \tag{26}$$

$$\sum_{i \in Q_k} x_i \leq 1 \qquad k \in K \tag{27}$$

$$x \in \{0,1\}^{|N|} \tag{28}$$

The size of N used is $\{50, 75, 100\}$. The size of M used is $\{10, 20\}$. The value for ω used is 3. For each instance the values of c_i is chosen at random in the integer interval $[1; 1000]$, the values of a_{im} is chosen at random in the integer interval $[0; 100]$, and the values of d_{im} is chosen at random in the integer interval $[0; a_{im}]$. The GUB-sets, Q_k, are created such that they are disjoint, each set contain a random number of variables in the interval $[0.1 \cdot N; 0.3 \cdot N]$, and such that $\bigcup_{k \in K} Q_k = N$. The value of b_m is set as

$$b_m = \beta \cdot \left(\sum_{i \in S} a_{im} + \omega \sqrt{\sum_{i \in T} d_{im}^2} \right),$$

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 d_{im} within each Q_k. The value of β used is $\{0.3, 0.5\}$. For each combination of N, M and β five instances are generated, giving a total of 60 test instances. These instances along with the source code is available for download at `http://diku.dk/~laurent`.

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 CPLEX, the MIP search strategy is set to traditional branch-and-bound, rather than the default dynamic search. The reason for this is that

we wish to investigate the effect of adding extended cover cuts using the proposed algorithms and bounds, and not to compare branch-and-bound with with extended cover cuts to dynamic search (it is not possible to add 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 2 Intel(R) Xeon(R) CPUs @ 2.67Ghz (16 logical cores), with 24 GB of RAM, and running Ubuntu 10.4.

5.3 Cuts

Depending on the combination of separation algorithm and bound used for the generic extension algorithm, cutting is either applied 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 bound arguments, but for the more computationally expensive the overhead of cutting in each node was to high. Table 1 lists how cutting is applied for the different combinations. In the following Sep1(conic), Sep2(x-sort), and Sep3(x/coef-sort) respectively refers to separation algorithm 1, 2, and 3, and Exact(conic), LB1(lprelax), and LB2(minsum) respectively refers to solving OPT, and the lower bounds $LB1$ and $LB2$.

	Sep1(conic)	**Sep2(x-sort)**	**Sep3(x/coef-sort)**
Exact(conic)	root	root	root
LB1(lprelax)	root	root	root
LB2(minsum)	all	all	all

Table 1: Table indicating whether cuts are applied only at the root (*root*) node, or throughout the branch-and-bound tree (*all*).

5.4 Results

We first compare the different combination of separation algorithms and bounds used for 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 the tables to come, the column **rgap** is the average optimality gap at the root node after addition of cuts. The **rgap** is calculated as $(UB - UB^*)/UB^*$, where UB is the value at the root node, and UB^* is the optimal solution. If no combination of algorithms could solve a given instance to optimality within the given time limit of 3600 secs, then UB^* is the best found solution across all the examined algorithms. In order to avoid cases with $UB^* = 0$, we add 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 cuts added at the root node, while 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. **nodes** is the average number of nodes of the branch-and-bound tree, **rt** is the time used in the root node in seconds, and **time** is the average total time used in seconds, where the number in parenthesis is how many of the 5 instances were solved to optimality within the time limit. Bold font indicates that all instances where solved to optimality.

Comparison of separation algorithms and bounds The branch-and-bound algorithm is run for each combination of separation algorithm and bound argument. Table 3, Table 4, and Table 5 contains the results for separation algorithm 1, 2, and 3 combined with the different bounds. Results from CPLEX can be seen in Table 2.

			CPLEX				
N	M	β	**rgap**	**cuts**	**nodes**	**rt**	**time**
50	10	0.3	77.8	0	865	0	**2(5)**
		0.5	22.87	0	1737	0	**9(5)**
	20	0.3	147.59	0	1335	1	**11(5)**
		0.5	38.83	0	2108	0	**83(5)**
75	10	0.3	80.24	0	2954	0	**32(5)**
		0.5	21.12	0	3594	0	**55(5)**
	20	0.3	183.4	0	2338	2	**29(5)**
		0.5	25.06	0	4724	2	760(4)
100	10	0.3	57.69	0	4664	0	**187(5)**
		0.5	7.78	0	1613	0	**13(5)**
	20	0.3	155.09	0	5175	3	1035(4)
		0.5	23.9	0	9279	3	2674(2)
Agg. time							4889(55)

Table 2: Results from CPLEX

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 can not be solved for $N = 75$ and $M = 20$, while for $N = 100$ all instances can be solved for $M = 10$, while 6 instances can be solved for $M = 20$. CPLEX solves a total of 55 instances using in total 4889 seconds.

We next compare the results of each combination of bound with separation algorithm 1 (Table 3), and compare these to CPLEX (Table 2). As can be seen, in general the impact of adding cuts using separation algorithm 1 has some effect on the computational time. For Exact(conic) and LB1(relax) the number of instances solved remains the same (55) but the computational time is reduced to 4131 and 4161 seconds respectively compared to the 4889 seconds of CPLEX. For LB2(minsum) the effect of cutting is quite noticable, the total number of solved instances increases to 59, and the total computational time is reduced to 1228 seconds. All combinations improves the root gaps compared to CPLEX. With respect to root gaps the best combination among the three is, as expected, Sep1+Exact(conic), but the time used 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(lprelax) using less time at the root node. Overall Sep1+LB1(lprelax) performs the best.

We next make a comparison, with separation algorithm 2 (Table 4). In general considerable more cuts are added at the root node, and as a consequence the root gap is lower than for separation algorithm 1. This may seem odd, as the separation problem is solved to optimality for separation algorithm 1. The reason is, separation algorithm 1 only attempts to extend the single cover corresponding to the solution of (20)–(24), while separation algorithm 2 will run through a number of covers, trying to extend each one. Extending the cover corresponding to the solution of (20)–(24) 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 explains why Sep2+Exact and Sep2+LB1 spends considerably more time spend in the root node, than their counterparts for separation algorithm 1. The additional cuts separated by the combinations of Exact(conic), and LB1(lprelax) with separation algorithm 2 does however not outweigh the additional time spend in the root node compared to separation algorithm 1, and the total computational time increases to respectively 13836 and 7077 seconds, while only a single extra instance is solved for LB1(lprelax). As the size of 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.

Finally considering separation algorithm 3 (Table 5), we see that the results are very similar to

			Sep1(conic)+Exact(conic)					Sep1(conic)+LB1(lprelax)				
N	M	β	**rgap**	**cuts**	**nodes**	**rt**	**time**	**rgap**	**cuts**	**nodes**	**rt**	**time**
50	10	0.3	0.4	35	1	8	**8(5)**	5.78	40	19	3	**3(5)**
		0.5	21.23	6	1597	4	**12(5)**	21.11	11	1663	2	**11(5)**
	20	0.3	28.38	55	70	14	**14(5)**	33.87	50	95	7	**7(5)**
		0.5	32.79	24	1526	11	**51(5)**	33.48	23	1970	3	**85(5)**
75	10	0.3	36.62	18	1365	5	**17(5)**	40.15	21	1346	3	**14(5)**
		0.5	16.68	18	3104	13	**68(5)**	17.2	22	3504	3	**113(5)**
	20	0.3	16.94	47	72	18	**19(5)**	34.7	51	136	10	**12(5)**
		0.5	20.7	20	5189	31	822(4)	22.56	16	4860	12	780(4)
100	10	0.3	51.44	14	2135	6	**16(5)**	53.08	12	2638	2	**16(5)**
		0.5	5.91	12	1365	31	**46(5)**	5.81	17	1623	5	**47(5)**
	20	0.3	87.48	32	816	22	**44(5)**	91.04	33	883	7	**26(5)**
		0.5	22.07	17	7906	55	3016(1)	22.46	14	10635	15	3048(1)
Agg. time							4131(55)					4161(55)

			Sep1(conic)+LB2(minsum)				
N	M	β	**rgap**	**cuts**	**nodes**	**rt**	**time**
50	10	0.3	2.22	45(42)	4	1	**1(5)**
		0.5	21.2	696(8)	191	1	**12(5)**
	20	0.3	25.41	91(51)	16	2	**4(5)**
		0.5	32.9	949(24)	228	1	**28(5)**
75	10	0.3	35.48	276(22)	59	1	**5(5)**
		0.5	17.3	1768(21)	467	1	**47(5)**
	20	0.3	20.73	84(50)	14	4	**5(5)**
		0.5	22.17	3296(14)	686	6	**123(5)**
100	10	0.3	52.29	414(13)	100	1	**9(5)**
		0.5	5.69	769(19)	223	2	**23(5)**
	20	0.3	92.59	256(32)	40	3	**9(5)**
		0.5	22.11	5450(20)	1087	12	962(4)
Agg. time							1228(59)

Table 3: Results from combinations of separation 1 and the different bounds.

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

If we compare the separation algorithms, separation algorithms 2, and 3 outperforms separation algorithm 1. The main reason is that for separation algorithm 1, a conic quadratic program needs to be solved, which is slow compared to the sorting done for separation algorithms 2, and 3. Also more cuts, can be separated per call, for the two latter, as more than one cover is attempted extended. 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 the different bounds LB2(minsum) has a clear advantage compared to the others. This is primarily because it is fast, and can thus be used to separate cuts throughout the branch-and-bound tree.

In order to better illustrate the advantage of cutting compared to CPLEX, Table 6 shows the results of CPLEX side-by-side with the best combination, i.e., separation algorithm 2, using

			Sep2(x-sort)+Exact(conic)					Sep2(x-sort)+LB1(lprelax)				
N	M	β	**rgap**	**cuts**	**nodes**	**rt**	**time**	**rgap**	**cuts**	**nodes**	**rt**	**time**
50	10	0.3	0.0	70	0	146	**146(5)**	0.67	120	2	57	**57(5)**
		0.5	17.12	52	1112	155	**161(5)**	17.54	58	1175	64	**68(5)**
	20	0.3	19.64	129	28	700	**700(5)**	26.33	192	27	186	**187(5)**
		0.5	26.51	92	1097	400	**496(5)**	28.43	102	1742	98	**232(5)**
75	10	0.3	20.57	113	103	699	**699(5)**	25.32	175	199	186	**187(5)**
		0.5	14.7	55	2635	271	**400(5)**	15.3	79	3191	77	**270(5)**
	20	0.3	13.52	161	12	998	**998(5)**	9.52	290	15	460	**460(5)**
		0.5	18.13	84	3881	644	1412(4)	19.57	99	4327	210	1158(4)
100	10	0.3	31.58	125	244	1710	**1711(5)**	36.73	231	477	428	**431(5)**
		0.5	4.79	35	785	339	**348(5)**	5.12	48	1518	63	**215(5)**
	20	0.3	42.1	277	58	2481	**2776(5)**	39.66	342	109	885	**887(5)**
		0.5	20.76	65	6961	1101	3989(1)	21.03	76	9223	228	2925(2)
Agg. time							13836(55)					7077(56)

			Sep2(x-sort)+LB2(minsum)				
N	M	β	**rgap**	**cuts**	**nodes**	**rt**	**time**
50	10	0.3	0.73	96(95)	1	0	**0(5)**
		0.5	17.29	931(56)	113	1	**1(5)**
	20	0.3	24.64	194(126)	17	1	**1(5)**
		0.5	27.56	1290(97)	140	1	**3(5)**
75	10	0.3	24.59	375(145)	33	1	**2(5)**
		0.5	14.57	2592(82)	238	0	**5(5)**
	20	0.3	12.43	238(206)	8	3	**3(5)**
		0.5	18.62	5013(111)	410	15	**30(5)**
100	10	0.3	37.19	652(160)	48	1	**2(5)**
		0.5	4.92	1702(47)	165	2	**5(5)**
	20	0.3	44.64	504(304)	32	6	**7(5)**
		0.5	20.73	13624(85)	860	21	**71(5)**
Agg. time							132(60)

Table 4: Results from combinations of separation 2 and the different bounds.

LB2(minsum).

Effect of extending covers In order to examine the effect of extending cover inequalities, we compare the results from running the best separation algorithm (separation algorithm 2), with and without the the bound resulting from the exact solution of OPT. The reason for using this bound, even though it is slow, is that it is optimal and should thus best 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 extension is used. The use of the slower exact extension however means that the time spend cutting at the root node, does not translate into a gain in total solution time.

Effect of using GUB information In order to examine the effect of using the GUB information when extending a cover, we perform two experiments. In the first experiment we compare the results of two runs, using separation algorithm 1 along with the optimal bound, i.e, solving OPT.

			Sep3(x/coef-sort)+Exact(conic)					Sep3(x/coef-sort)+LB1(lprelax)				
N	M	β	**rgap**	**cuts**	**nodes**	**rt**	**time**	**rgap**	**cuts**	**nodes**	**rt**	**time**
50	10	0.3	0.0	69	0	145	**145(5)**	0.47	108	4	57	**57(5)**
		0.5	18.54	32	1120	169	**175(5)**	18.97	31	1560	45	**55(5)**
	20	0.3	25.43	91	30	463	**463(5)**	27.79	157	40	152	**152(5)**
		0.5	28.38	64	1049	398	**420(5)**	30.81	63	1737	96	**176(5)**
75	10	0.3	22.74	102	137	685	**685(5)**	25.81	164	264	137	**138(5)**
		0.5	16.38	42	3432	289	**672(5)**	16.39	53	3517	64	**367(5)**
	20	0.3	11.59	200	20	1562	**1599(5)**	13.18	298	13	321	**321(5)**
		0.5	20.6	53	3943	710	1469(4)	21.35	59	4500	155	935(4)
100	10	0.3	35.36	134	253	1942	**1944(5)**	42.36	196	612	254	**258(5)**
		0.5	5.17	23	1163	411	**507(5)**	5.84	23	1729	56	**769(5)**
	20	0.3	44.88	309	123	3364	**3784(5)**	41.38	438	74	731	**732(5)**
		0.5	22.76	36	8416	906	3810(1)	22.03	30	8144	170	3355(1)
Agg. time							15673(55)					7316(55)

			Sep3(x/coef-sort)+LB2(minsum)				
N	M	β	**rgap**	**cuts**	**nodes**	**rt**	**time**
50	10	0.3	0.0	106(106)	0	0	**0(5)**
		0.5	20.42	868(24)	159	0	**1(5)**
	20	0.3	24.45	212(138)	17	1	**1(5)**
		0.5	29.78	1111(62)	166	1	**3(5)**
75	10	0.3	27.22	373(146)	38	1	**2(5)**
		0.5	16.28	2381(54)	432	0	**12(5)**
	20	0.3	11.96	295(246)	15	3	**4(5)**
		0.5	21.26	4155(52)	597	7	**40(5)**
100	10	0.3	39.33	625(172)	69	1	**2(5)**
		0.5	5.75	879(31)	184	1	**4(5)**
	20	0.3	46.13	539(393)	23	5	**7(5)**
		0.5	22.04	12336(34)	1674	13	**270(5)**
Agg. time							347(60)

Table 5: Results from combinations of separation 3 and the different bounds.

In the first run the GUB information is used, while in the second run it is not. Not using the GUB information means removing constraints (4) and constraints (23) from their respective mathematical programs. In both cases cutting is only applied at the root node. As can be seen from the results in Table 8 using GUB information results in an improvement of the root gap. This does however not translate into a better total solution time.

In order to get a better indication of the usefulness of using GUB information, we conduct a second experiment. In this experiment we compare the best separation and bound, i.e., Sep2+LB2(minsum), with an implementation of the separation and extension algorithm of Atamtürk and Narayanan (2008), which does not make use of GUB information. We do not include their advanced lifting procedure, but only their extension algorithm. The ordering of the variables used when extending is the same as for the other bounds examined. Cuts are applied throughout the branch-and-bound tree for both algorithms. As can be seen from the results in Table 9 there is a clear gain from employing GUB information when separating and extending cuts. It is surprising that for some of the instances so few cuts are separated at the root node by our implementation of the separation and extension described by Atamtürk and Narayanan

			CPLEX					Sep2(x-sort)+LB2(minsum)				
N	M	β	**rgap**	**cuts**	**nodes**	**rt**	**time**	**rgap**	**cuts**	**nodes**	**rt**	**time**
50	10	0.3	77.8	0	865	0	**2(5)**	0.73	96(95)	1	0	**0(5)**
		0.5	22.87	0	1737	0	**9(5)**	17.29	931(56)	113	1	**1(5)**
	20	0.3	147.59	0	1335	1	**11(5)**	24.64	194(126)	17	1	**1(5)**
		0.5	38.83	0	2108	0	**83(5)**	27.56	1290(97)	140	1	**3(5)**
75	10	0.3	80.24	0	2954	0	**32(5)**	24.59	375(145)	33	1	**2(5)**
		0.5	21.12	0	3594	0	**55(5)**	14.57	2592(82)	238	0	**5(5)**
	20	0.3	183.4	0	2338	2	**29(5)**	12.43	238(206)	8	3	**3(5)**
		0.5	25.06	0	4724	2	760(4)	18.62	5013(111)	410	15	**30(5)**
100	10	0.3	57.69	0	4664	0	**187(5)**	37.19	652(160)	48	1	**2(5)**
		0.5	7.78	0	1613	0	**13(5)**	4.92	1702(47)	165	2	**5(5)**
	20	0.3	155.09	0	5175	3	1035(4)	44.64	504(304)	32	6	**7(5)**
		0.5	23.9	0	9279	3	2674(2)	20.73	13624(85)	860	21	**71(5)**
Agg. time							4889(55)					132(60)

Table 6: Comparison of best combination to CPLEX.

(2008), but we believe that a reason could be that ILOG CPLEX 12.1 separates a number of unspecified conic cuts, which may be similar to cover cuts without GUB information (the version of CPLEX used in Atamtürk and Narayanan (2008) is 11.0 and does not separate any conic cuts). We note however that the implementation still beats CPLEX.

6 Conclusion

We have described how the second-order conic equivalent of cover cuts can be extended by using the structure imposed by GUB constraints. We have proposed a number of separation and extension algorithms, and compared these to one another and to CPLEX on a set of generated test instances. These experiments show that a relatively simple separation and extension algorithm, which employs the GUB constraints, can speed up the solution time considerably. Fast separation, and extension algorithms are an advantage, since this makes it possible to cut locally through-out the branch-and-bound tree as opposed to only in the root node.

As a theoretical contribution we have showed that the problem of deciding if a cover can be extended with a varible is $\mathcal{NP}$-hard, and have established the relation between two bounds: one based on an LP relaxation (LB1) and the other based on decomposing the problem (LB2).

			Sep2(x-sort)+Exact(conic)					Sep2(x-sort)				
N	M	β	**rgap**	**cuts**	**nodes**	**rt**	**time**	**rgap**	**cuts**	**nodes**	**rt**	**time**
50	10	0.3	0.0	70	0	146	**146(5)**	36.34	45	270	0	**1(5)**
		0.5	17.12	52	1112	155	**161(5)**	22.29	5	1758	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	1097	400	**496(5)**	37.86	4	1926	0	**85(5)**
75	10	0.3	20.57	113	103	699	**699(5)**	56.86	35	2173	1	**21(5)**
		0.5	14.7	55	2635	271	**400(5)**	19.18	10	3598	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	3881	644	1412(4)	24.95	1	5155	3	887(4)
100	10	0.3	31.58	125	244	1710	**1711(5)**	55.32	12	3916	0	**129(5)**
		0.5	4.79	35	785	339	**348(5)**	7.1	9	1512	1	**13(5)**
	20	0.3	42.1	277	58	2481	**2776(5)**	100.4	64	1856	4	**85(5)**
		0.5	20.76	65	6961	1101	3989(1)	23.6	5	9140	3	2836(2)
Agg. time							13836(55)					4145(56)

Table 7: Results from running separation algorithm 2 with and without the Exact(conic) bound.

			Sep1(conic)+Exact(conic)					Sep1(Conic)+Exact(conic)-GUB				
N	M	β	**rgap**	**cuts**	**nodes**	**rt**	**time**	**rgap**	**cuts**	**nodes**	**rt**	**time**
50	10	0.3	0.4	35	1	8	**8(5)**	0.83	43	1	10	**10(5)**
		0.5	21.23	6	1597	4	**12(5)**	22.13	4	1621	4	**12(5)**
	20	0.3	28.38	55	70	14	**14(5)**	33.53	43	90	15	**15(5)**
		0.5	32.79	24	1526	11	**51(5)**	36.32	12	2022	11	**120(5)**
75	10	0.3	36.62	18	1365	5	**17(5)**	40.53	22	1373	10	**21(5)**
		0.5	16.68	18	3104	13	**68(5)**	19.15	10	3390	11	**65(5)**
	20	0.3	16.94	47	72	18	**19(5)**	39.0	52	176	25	**27(5)**
		0.5	20.7	20	5189	31	822(4)	24.61	5	5025	14	**618(5)**
100	10	0.3	51.44	14	2135	6	**16(5)**	53.92	13	2603	4	**16(5)**
		0.5	5.91	12	1365	31	**46(5)**	6.53	12	1402	24	**33(5)**
	20	0.3	87.48	32	816	22	**44(5)**	95.34	28	953	20	**38(5)**
		0.5	22.07	17	7906	55	3016(1)	22.99	8	10812	36	2537(3)
Agg. time							4131(55)					3512(58)

Table 8: Results from running Sep1+Exact(conic) with and without use of GUB information.

			Sep2(x-sort)+LB2(minsum)					Atamtürk and Narayanan (2008)				
N	M	β	**rgap**	**cuts**	**nodes**	**rt**	**time**	**rgap**	**cuts**	**nodes**	**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	1193(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	1290(97)	140	1	**3(5)**	38.09	1861(1)	485	0	**17(5)**
75	10	0.3	24.59	375(145)	33	1	**2(5)**	73.67	1065(8)	163	1	**11(5)**
		0.5	14.57	2592(82)	238	0	**5(5)**	19.59	3313(9)	1161	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	5013(111)	410	15	**30(5)**	24.95	6021(1)	1433	3	**179(5)**
100	10	0.3	37.19	652(160)	48	1	**2(5)**	55.68	2011(8)	277	1	**37(5)**
		0.5	4.92	1702(47)	165	2	**5(5)**	7.46	1290(3)	489	1	**41(5)**
	20	0.3	44.64	504(304)	32	6	**7(5)**	123.24	1215(31)	69	6	**35(5)**
		0.5	20.73	13624(85)	860	21	**71(5)**	23.78	21045(2)	3405	3	**1411(5)**
Agg. time							132(60)					1811(60)

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

References

Atamtürk, A. Cover and pack inequalities for (mixed) integer programming. *Annals of Operations Research*, 139(1):21–38, 2005.

Atamtürk, A., Narayanan, V. Polymatroids and risk minimization in discrete optimization. *Operations Research Letters*, 36:618–622, 2008.

Atamtürk, A., Narayanan, V. Lifting for conic mixed-integer programming. *Mathematical Programming*, Onlne:1–13, 2009a.

Atamtürk, A., Narayanan, V. The submodular 0-1 knapsack polytope. *Discrete Optimization*, 6: 333–344, 2009b.

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

Balas, E. Facets of the knapsack polytope. *Mathematical Programming*, 8(1):146–164, 1975.

Boyd, S., Vandenberghe, L. *Convex Optimization*. Cambridge University Press, March 2004. ISBN 0521833787.

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

Crowder, H., Johnson, E., Padberg, M. Solving large-scale 0-1 linear programming problems. *Operations Research*, 31(5):803–834, 1983.

Ferreira, C., Martin, A., Weismantel, R. Solving multiple knapsack problems by cutting planes. *SIAM Journal on Optimization*, 6(3):858–877, 1996.

Fujishige, S. *Submodular functions and optimization*, volume 58. Elsevier Science Ltd, 2005.

Gu, Z., Nemhauser, G., Savelsbergh, M. Lifted cover inequalities for 0-1 integer programs: Computation. *INFORMS Journal on Computing*, 10:427–437, 1998.

Gu, Z., Nemhauser, G., Savelsbergh, M. Lifted cover inequalities for 0-1 integer programs: Complexity. *INFORMS Journal on Computing*, 11:117–123, 1999.

Hammer, P., Johnson, E., Peled, U. Facet of regular 0–1 polytopes. *Mathematical Programming*, 8(1):179–206, 1975.

Hartvigsen, D., Zemel, E. The complexity of lifted inequalities for the knapsack problem. *Discrete Applied Mathematics*, 39(2):113–123, 1992.

Iwata, S. Submodular function minimization. *Mathematical Programming*, 112(1):45–64, 2008.

Johnson, E. L., Padberg, M. W. A note of the knapsack problem with special ordered sets. *Operations Research Letters*, 1(1):18–22, 1981.

Kaparis, K., Letchford, A. Cover inequalities. Technical report, Lancaster University, 2010.

Karp, R. Reducibility among combinatorial problems. In Miller, R., Thatcher, J., editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

Klabjan, D., Nemhauser, G., Tovey, C. The complexity of cover inequality separation1. *Operations Research Letters*, 23(1-2):35–40, 1998.

Nemhauser, G., Vance, P. Lifted cover facets of the 0-1 knapsack polytope with gub constraints. *Operations Research Letters*, 16(5):255–263, 1994.

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

Wolsey, L. A. Valid inequalities for 0-1 knapsacks and mips with generalised upper bound constraints. *Discrete Applied Mathematics*, 29(2-3):251–261, 1990.

Zemel, E. Easily computable facets of the knapsack polytope. *Mathematics of Operations Research*, 14(4):760–764, 1989.

We consider the second-order conic equivalent of the classic knapsack polytope where the variables are subject to generalized upper bound constraints. We describe and compare a number of separation and extension algorithms which make use of the extra structure implied by the generalized upper bound constraints in order to strengthen the second-order conic equivalent of the classic cover cuts. We show that determining whether a cover can be extended with a variable is NP-hard. Computational experiments are performed comparing the proposed separation and extension algorithms. These experiments show that applying these extended cover cuts can greatly improve solution time of second-order cone programs.

DTU Management Engineering
Department of Management Engineering
Technical University of Denmark

Produktionstorvet
Building 424
DK-2800 Kongens Lyngby
Denmark
Tel. +45 45 25 48 00
Fax +45 45 93 34 35

www.man.dtu.dk