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

Extended Formulations in Mixed Integer Conic Quadratic Programming

Juan Pablo Vielma, Iain Dunning, Joey Huchette and Miles Lubin

Jan 6th, 2015. Revised Mar 4th, 2015. Revised Feb 15th, 2016. Revised Jul 10th, 2016.

Abstract In this paper we consider the use of extended formulations in LP-based algorithms for mixed integer conic quadratic programming (MICQP). Extended formulations have been used by Vielma, Ahmed and Nemhauser (2008) and Hijazi, Bonami and Ouorou (2013) to construct algorithms for MICQP that can provide a significant computational advantage. The first approach is based on an extended or lifted polyhedral relaxation of the Lorentz cone by Ben-Tal and Nemirovski (2001) that is extremely economical, but whose approximation quality cannot be iteratively improved. The second is based on a lifted polyhedral relaxation of the euclidean ball that can be constructed using techniques introduced by Tawarmalani and Sahinidis (2005). This relaxation is less economical, but its approximation quality can be iteratively improved. Unfortunately, while the approach of Vielma, Ahmed and Nemhauser is applicable for general MICQP problems, the approach of Hijazi, Bonami and Ouorou can only be used for MICQP problems with convex quadratic constraints. In this paper we show how a homogenization procedure can be combined with the technique by Tawarmalani and Sahinidis to adapt the extended formulation used by Hijazi, Bonami and Ouorou to a class of conic mixed integer programming problems that include general MICQP problems. We then compare the effectiveness of this new extended formulation against traditional and extended formulation-based algorithms for MICQP. We find that this new formulation can be used to improve various LP-based algorithms. In particular, the formulation provides an easy-to-implement procedure that, in our benchmarks, significantly improved the performance of commercial MICQP solvers.

1 Introduction

Most state of the art solvers for (convex) mixed integer nonlinear programming (MINLP) can be classified as nonlinear programming (NLP) based branch-and-bound algorithms or linear programming (LP) based branch-and-bound algorithms. NLP-based algorithms (e.g. [11, 29, 36, 43]) are the natural extension of

J. P. Vielma
Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA 02139
E-mail: jvielma@mit.edu
I. Dunning
Operations Research Center, Massachusetts Institute of Technology, Cambridge, MA 02139
E-mail: idunning@mit.edu
J. Huchette
Operations Research Center, Massachusetts Institute of Technology, Cambridge, MA 02139
E-mail: huchette@mit.edu
M. Lubin
Operations Research Center, Massachusetts Institute of Technology, Cambridge, MA 02139
E-mail: mlubin@mit.edu

LP-based branch-and-bound algorithms for mixed integer linear programming (MILP) and solve the NLP relaxation of the MINLP in each node of the branch-and-bound tree. In contrast, LP-based algorithms (e.g. [3, 9, 17, 18, 23, 41, 46, 47]) try to avoid solving the NLP relaxation as much as possible. To achieve this they use a polyhedral relaxation of the nonlinear constraints to construct an LP that is solved in the nodes of the branch-and-bound tree. The relaxation is iteratively refined through the branch-and-bound procedure, which is combined with sporadic solutions to the NLP relaxation.

Traditional LP-based algorithms construct the polyhedral relaxations in the original variable space used to describe the nonlinear constraints. However, an emerging trend is to use auxiliary variables to construct extended or *lifted* polyhedral relaxations of the nonlinear constraints. This approach exploits the fact that the projection of a polyhedron can have significantly more inequalities than the original polyhedron. Hence, it is sometimes possible to construct small lifted polyhedral relaxations with the same approximation quality as significantly larger relaxations in the original space. Two algorithms that use lifted polyhedral relaxations to solve quadratic MINLP problems are the *lifted LP branch-and-bound* (LiftedLP) algorithm developed in [45] and the variant of BONMIN introduced in [32]. The two main differences between these approaches are the class of lifted polyhedral approximation used and the range of problems to which they are applicable. The LiftedLP algorithm from [45] uses a lifted polyhedral relaxation of the Lorentz (or second order) cone introduced by Ben-Tal and Nemirovski [6] and hence is applicable to any mixed integer conic quadratic programming (MICQP) problem. This relaxation can be tailored to any approximation quality, but once this approximation is fixed the relaxation cannot be easily refined. More specifically, the only known way to improve the approximation quality is to reconstruct the relaxation from scratch. In this paper we refer to approximations with this property as *static lifted polyhedral relaxations*. The inability to easily refine a static relaxation is clearly undesirable. However, the number of constraints and additional variables in the approximation by Ben-Tal and Nemirovski grows so slowly that the LiftedLP algorithm is able to use a fixed relaxation throughout its execution. While the approximation quality has to be selected a priori, this choice is often a minor calibration exercise and the resulting algorithm can significantly outperform traditional LP and NLP-based algorithms. Still, if the approximation quality is chosen poorly, the only way to adjust it is to reconstruct the relaxation and restart the algorithm. One potential solution to this issue is the variant of BONMIN from [32]. This algorithm uses a lifted polyhedral relaxation of the euclidean ball that can be constructed using techniques introduced by Tawarmalani and Sahinidis [44]. This approximation is not as efficient as the one by Ben-Tal and Nemirovski, but it is straightforward to iteratively refine it to improve the approximation accuracy. In this paper we refer to approximations with this property as *dynamic lifted polyhedral relaxations*. Using this approximation can also significantly improve the performance of traditional LP-based algorithms. However, the approach is only applicable for MINLP problems with convex quadratic constraints and hence cannot be used for all problems for which the LiftedLP algorithm is applicable.

In this paper we study the effectiveness of dynamic lifted polyhedral relaxations for the solution of general MICQP problems. In particular, we introduce three dynamic lifted polyhedral relaxations of the Lorentz cone. The first relaxation is a straightforward adaptation of a portion of the original static relaxation introduced by Ben-Tal and Nemirovski [6]. The second relaxation we construct by combining a homogenization procedure with the relaxation technique by Tawarmalani and Sahinidis [44]. This yields a relaxation for various conic sets including the Lorentz cone. The final relaxation is a simple combination of the previous two. All three approximations can be used to improve the LiftedLP algorithm from [45]. However, we show that they can also be used directly by interpreting them as lifted conic quadratic formulations of the Lorentz cone, which lead to extended MICQP re-formulations of any MICQP problem. We computationally evaluate the effectiveness of all three formulations when used by themselves and as an improvement of the LiftedLP algorithm. Our computational results show that the extended MICQP re-formulation version of the homogenized variant of the Tawarmalani and Sahinidis relaxation can provide a significant computational advantage over traditional LP and NLP algorithms and other lifted polyhedral relaxation approaches.

The remainder of this paper is organized as follows. Section 2 provides a brief introduction to LP-based algorithms for MINLP, a detailed description of existing lifted polyhedral relaxations and a brief description of their use in LP-based algorithms. This section also compares and contrasts the advantages of extremely

economical static lifted polyhedral relaxations that cannot be iteratively refined, to less economical dynamic relaxations that can be refined. In particular, the section introduces the relaxation from [32, 44] as an extremely effective compromise between these objectives, that nonetheless has some modeling limitations. Then, Section 3 shows how the modeling limitations of the relaxation from [32, 44] can be eliminated through a homogenization procedure that adapts it to all MICQP problems. In this section we also discuss the relation of the homogenization procedure with existing perspective reformulation techniques. The new lifted polyhedral relaxation leads to some variants of the LiftedLP algorithm from [45], which are described in detail in Section 4. This section also shows how the relaxations can be used directly by interpreting them as extended MICQP reformulations. Section 5 then provides a computational comparison of these algorithms and traditional LP-based and NLP-based algorithms. Finally, Section 6 discusses some possible improvements for specific classes of problems and some open questions concerning the best possible approximation quality for dynamic lifted polyhedral approximations. Omitted proofs and additional figures and tables are provided in Appendices A and B.

To improve readability we use the convention of naming polyhedral sets in a natural original variable space with upper case letters (e.g. P) and non-polyhedral sets in the same space with bold upper case letters (e.g. $\mathbf{C}$). Lifted sets defined in the space of original and auxiliary variables additionally appear with a "hat" (e.g. $\widehat{P}$ for lifted polyhedral sets and $\widehat{\mathbf{C}}$ for lifted non-polyhedral sets). Additional definitions and notation will be introduced as they are needed through the paper.

2 LP Based Algorithms and Lifted Polyhedral Relaxations

We consider a MICQP of the form

$$\begin{aligned}
\text{obj}_{\mathsf{MICQP}} := \max \quad & cx && \text{(1a)}\\
s.t. \quad & \\
& Ex \leq h, && \text{(1b)}\\
& \left\| A^l x + b^l \right\|_2 \leq a^l \cdot x + b_0^l, \quad \forall l \in \{1, \ldots, q\}, && \text{(1c)}\\
& x \in \mathbb{R}^n, && \text{(1d)}\\
& x_j \in \mathbb{Z}^n, \qquad \forall j \in I, && \text{(1e)}
\end{aligned}$$

where $c \in \mathbb{R}^n$, $E \in \mathbb{R}^{m \times n}$, $h \in \mathbb{R}^m$, $\|\cdot\|_2$ is the euclidean norm, $a^l \cdot x$ is the inner product between a^l and b, and where for each $l \in \{1, \ldots, q\}$ there exist d_l such that $A^l \in \mathbb{R}^{d_l \times n}$, $b^l \in \mathbb{R}^{d_l}$, $a^l \in \mathbb{R}^n$ and $b_0^l \in \mathbb{R}$. We denote problem (1) as MICQP.

We start by noting that constraint (1c) can be written as

$$A^l x + b^l = y, \quad a^l \cdot x + b_0^l = y_0, \quad (y_0, y) \in \mathbf{L}^{d_l},$$

where $\mathbf{L}^d$ is the $(d+1)$-dimensional Lorentz cone given by $\mathbf{L}^d := \left\{(y_0, y) \in \mathbb{R}^{d+1} : \|y\|_2 \leq y_0\right\}$. Hence, to get an LP-based algorithm for (1) it suffices to construct a polyhedral relaxation of $\mathbf{L}^d$.

We can construct a polyhedral relaxation of $\mathbf{L}^d$ through its semi-infinite representation given by

$$\mathbf{L}^d = \left\{ (y_0, y) \in \mathbb{R}^{d+1} : \sum_{j=1}^{d} \omega_j y_j \leq y_0, \forall \omega \in \mathbf{S}^{d-1} \right\}.$$

where $\mathbf{S}^{d-1} := \left\{\omega \in \mathbb{R}^d : \|\omega\|_2 = 1\right\}$ is the unit sphere. Then for any $\Omega \subseteq \mathbf{S}^{d-1}$ with $|\Omega| < \infty$, the set

$$O^d(\Omega) := \left\{ (y_0, y) \in \mathbb{R}^{d+1} : \sum_{j=1}^{d} \omega_j y_j \leq y_0, \forall \omega \in \Omega \right\} \tag{2}$$

is a polyhedron such that $\mathbf{L}^d \subseteq O^d(\Omega)$. This polyhedral relaxation can easily be refined by noting that if $(y_0, y) \in O^d(\Omega) \backslash \mathbf{L}^d$ then $\sum_{j=1}^d \omega_j(y) y_j > y_0$ where $\omega(y) = \frac{y}{\|y\|_2} \in \mathbf{S}^{d-1}$ and hence $(y_0, y) \notin O^d(\Omega \cup \{\omega(y)\})$. Polyhedral relaxations such as O^d, which do not use auxiliary variables and can be iteratively refined, lead to traditional LP-based algorithms usually known as outer approximation algorithms. The simplest version of such algorithms is the MILP-based algorithm by Duran and Grossmann [17]. In the context of MICQP, this algorithm is based on the MILP relaxation of MICQP given by

$$\begin{aligned}
\text{obj}_{\mathsf{MILP}\left(\{\Omega^l\}_{l=1}^k\right)} := \max \quad & cx && && \text{(3a)}\\
s.t. \quad & \\
& Ex \leq h, && && \text{(3b)}\\
& A^l x + b^l = y^l, && \forall l \in \{1,\dots,q\}, && \text{(3c)}\\
& a^l \cdot x + b_0^l = y_0^l, && \forall l \in \{1,\dots,q\}, && \text{(3d)}\\
& \left(y_0^l, y^l\right) \in O^{d_l}\left(\Omega^l\right), && \forall l \in \{1,\dots,q\}, && \text{(3e)}\\
& x \in \mathbb{R}^n, && && \text{(3f)}\\
& x_j \in \mathbb{Z}^n, && \forall j \in I. && \text{(3g)}
\end{aligned}$$

We denote this relaxation by $\mathsf{MILP}\left(\left\{\Omega^l\right\}_{l=1}^q\right)$.

A basic version of the MILP-based outer approximation algorithm for solving MICQP is given in Algorithm 1, where for simplicity we have assumed the existence of an initial approximating set $\left\{\tilde{\Omega}^l\right\}_{l=1}^q$ for which $\mathsf{MILP}\left(\left\{\tilde{\Omega}^l\right\}_{l=1}^q\right)$ is bounded.

Algorithm 1: A Basic LP Based Algorithm.

```
Set Ω^l = Ω̃^l for all l ∈ {1,...,q}.
Solve MILP({Ω^l}_{l=1}^q).
if MILP({Ω^l}_{l=1}^q) is infeasible then
    Declare MICQP infeasible.
else
    Let (x̄, ȳ, {(ȳ_0^l, ȳ^l)}_{l=1}^q) be an optimal solution to MILP({Ω^l}_{l=1}^q).
end
if (ȳ_0^l, ȳ^l) ∈ L^{d_l} for all l ∈ {1,...,q} then
    Return (x̄, ȳ) as the optimal solution to MICQP.
else
    for l = 1 to q do
        if (ȳ_0^l, ȳ^l) ∉ L^{d_l} then
            Set Ω^l = Ω^l ∪ {ω(ȳ^l)}.
        end
    end
end
Go to 2
```

Algorithm 1 converges in finite time for pure integer problems ($p = 0$) with bounded feasible regions and simple modifications can ensure convergence for the mixed integer case too (e.g. see [9, 17, 18]). However, the following example from [32] shows that an exponential number of iterations may be needed even for very simple pure integer problems.

Example 1 Let $\mathbf{F}^n := \left\{x \in \mathbb{R}^n : \sum_{j=1}^n \left(x_j - \frac{1}{2}\right)^2 \leq \frac{n-1}{4}\right\}$ and consider the description of $\mathbf{F}^n$ as the feasible region of MICQP given by

$$x_j - \frac{1}{2} = y_j, \quad \forall j \in \{1, \ldots, n\}, \tag{4a}$$

$$\sqrt{\frac{n-1}{4}} = y_0, \tag{4b}$$

$$(y_0, y) \in \mathbf{L}^n. \tag{4c}$$

It is shown in [32] that if $P \subseteq \mathbb{R}^n$ is a polyhedron such that $\mathbf{F}^n \subseteq P$ and $P \cap \mathbb{Z}^n = \emptyset$, then P must have at least 2^n facets. In particular, $\left\{(x, y_0, y) \in \mathbb{R}^{2n+1} : (4a)\text{–}(4b), \quad (y_0, y) \in O^n(\Omega)\right\} \neq \emptyset$ for any $\Omega \subseteq \mathbf{S}^{n-1}$ such that $|\Omega| < 2^n$ and hence Algorithm 1 will require an exponential number of iterations to prove that $\mathbf{F}^n \cap \mathbb{Z}^n = \emptyset$.

We can check that $\tilde{F}^n := \left\{x \in \mathbb{R}^n : \sum_{j=1}^n \left|x_j - \frac{1}{2}\right| \leq \frac{\sqrt{n}\sqrt{n-1}}{2}\right\}$ is such that $\mathbf{F}^n \subseteq \tilde{F}^n$ and $\tilde{F}^n \cap \mathbb{Z}^n = \emptyset$. As expected $\tilde{F}^n$ has an exponential number of facets, but using standard LP techniques we can construct the lifted LP representation of $\tilde{F}^n$ given by

$$\begin{aligned} x_j - \frac{1}{2} &\leq z_j, && \forall j \in \{1, \ldots, n\}, \\ \frac{1}{2} - x_j &\leq z_j, && \forall j \in \{1, \ldots, n\}, \\ \sum_{j=1}^n z_j &\leq \frac{\sqrt{n}\sqrt{n-1}}{2}. \end{aligned}$$

Then, this formulation is a lifted polyhedral relaxation of $\tilde{F}^n$ with a linear number of constraints and additional variables that can be used to prove $\mathbf{F}^n \cap \mathbb{Z}^n = \emptyset$. Hence, the exponential behavior from Example 1 can be avoided by using auxiliary variables. In the following subsection we describe two lifted polyhedral relaxation approaches that can be used to prove $\mathbf{F}^n \cap \mathbb{Z}^n = \emptyset$ and are also applicable to more general MICQPs.

2.1 Lifted Polyhedral Relaxations

We begin with a formal definition of a lifted polyhedral relaxation of $\mathbf{L}^d$ (the definition naturally generalizes to arbitrary convex sets).

Definition 1 A polyhedron $P := \left\{(y_0, y, z) \in \mathbb{R}^{d+1+m_2} : A\begin{pmatrix} y_0 \\ y \end{pmatrix} + Dz \leq b\right\}$ for $A \in \mathbb{R}^{m_1 \times (d+1)}$, $D \in \mathbb{R}^{m_1 \times m_2}$ and $b \in \mathbb{R}^{m_1}$ is a *lifted polyhedral relaxation* of $\mathbf{L}^d$ if and only if

$$\mathbf{L}^d \subseteq \mathrm{Proj}_{(y_0, y)}(P),$$

where $\mathrm{Proj}_{(y_0,y)}$ is the orthogonal projection onto the (y_0, y) variables.

Two desirable properties of a lifted polyhedral relaxation are small number of inequalities (m_1) and number of auxiliary variables (m_2), and high approximation quality (quantified through various measures). For instance, the first approximation we consider additionally satisfies

$$\mathbf{L}^d \subseteq \mathrm{Proj}_{(y_0,y)}(P) \subseteq \left\{(y_0, y) \in \mathbb{R}^{d+1} : ||y||_2 \leq (1+\varepsilon)y_0\right\}, \tag{5}$$

which we refer to as having approximation quality ε, and $m_1, m_2 \leq \psi(d, \ln(1/\varepsilon))$ for a low degree polynomial ψ. This approximation was introduced by Ben-Tal and Nemirovski [6] (see also [24]) and its construction begins by building the lifted polyhedral relaxation of $\mathbf{L}^2$ given in the following proposition.

Proposition 1 *Let*

$$\widehat{N}_s^2 := \left\{ (y_0, y_1, y_2, v) \in \mathbb{R}_+ \times \mathbb{R}^2 \times \mathbb{R}^{2s} \; : \; \begin{array}{rl} y_0 = v_{2s-1}\cos\left(\frac{\pi}{2^s}\right) + v_{2s}\sin\left(\frac{\pi}{2^s}\right), & \\ v_1 = y_1\cos(\pi) + y_2\sin(\pi), & \\ v_2 \geq |y_2\cos(\pi) - y_1\sin(\pi)|, & \\ v_{2(i+1)-1} = v_{2i-1}\cos\left(\frac{\pi}{2^i}\right) + v_{2i}\sin\left(\frac{\pi}{2^i}\right), & \forall i \in \{1,\dots,s-1\}, \\ v_{2(i+1)} \geq \left| v_{2i}\cos\left(\frac{\pi}{2^i}\right) - v_{2i-1}\sin\left(\frac{\pi}{2^i}\right)\right|, & \forall i \in \{1,\dots,s-1\} \end{array} \right\}.$$

Then, $\widehat{N}_s^2$ is a lifted polyhedral relaxation of $\mathbf{L}^2$ with approximation quality $\varepsilon = \cos\left(\frac{\pi}{2^s}\right)^{-1} - 1$.

To construct a lifted polyhedral relaxation of $\mathbf{L}^d$ for $d > 2$, Ben-Tal and Nemirovski observe that

$$\mathbf{L}^d = \left\{ (y_0, y) \in \mathbb{R}^{d+1} \; : \; \exists t \in \mathbb{R}^{\lfloor d/2 \rfloor} \text{ s.t. } \begin{array}{rl} \sum_{k=1}^{\lceil d/2 \rceil} t_k^2 \leq y_0^2, & \\ y_{2k-1}^2 + y_{2k}^2 \leq t_k^2, & \forall k \in \{1,\dots,\lfloor d/2 \rfloor\}, \\ t_{\lceil d/2 \rceil} = y_d, & \text{if } d \text{ is odd} \end{array} \right\}.$$

Recursively repeating this construction for the $\lceil d/2 \rceil + 1$ dimensional Lorentz cone inside this representation we can construct a version with additional variables that only uses $d-1$ three dimensional Lorentz cones. This construction was denoted the *tower of variables* by Ben-Tal and Nemirovski and yields the lifted representation $\mathbf{L}^d = \text{Proj}_{(y_0,y)}\left(\widehat{\mathbf{T}}^d\right)$ for

$$\widehat{\mathbf{T}}^d = \left\{ (y_0, y, t) \in \mathbb{R}^{d+1+R(d)} \; : \; \begin{array}{rl} y_0 = t_1^K, & \\ t_i^0 = y_i, & \forall i \in \{1,\dots,d\}, \\ (t_i^{k+1}, t_{2i-1}^k, t_{2i}^k) \in \mathbf{L}^2, & \forall i \in \{1,\dots,\lfloor r_k/2 \rfloor\}, \\ & k \in \{0,\dots,K-1\}, \\ t_{r_k}^k = t_{\lceil r_k/2 \rceil}^{k+1}, & \forall k \in \{0,\dots,K-1\} \\ & \text{s.t. } r_k \text{ is odd} \end{array} \right\} \tag{6}$$

where $K = \lceil \log_2(d) \rceil$, $\{r_k\}_{k=0}^K$ is defined by the recursion $r_0 = d$, $r_{k+1} = \lceil r_k/2 \rceil$ for $k \in \{0,\dots,K-1\}$ and $R(d) = \sum_{k=0}^K r_k$. A lifted polyhedral relaxation of $\mathbf{L}^d$ can then be constructed by replacing every occurrence of $\mathbf{L}^2$ in $\widehat{\mathbf{T}}^d$ with an appropriate polyhedral relaxation. The original approximation of Ben-Tal and Nemirovski uses $\widehat{N}_s^2$ as a lifted polyhedral relaxation of $\mathbf{L}^2$ and leads to essentially the smallest possible approximation of $\mathbf{L}^d$ (see Section 3 of [6]). However, we will see that using alternative approximations could lead to a computational advantage in our context, so the following proposition first presents a generic version of the approximation and then specializes it to the original Ben-Tal and Nemirovski approximation.

Proposition 2 *Let $K = \lceil \log_2(d) \rceil$, $\{r_k\}_{k=0}^K$ be defined by the recursion $r_0 = d$, $r_{k+1} = \lceil r_k/2 \rceil$ for $k \in \{0,\dots,K-1\}$ and $R(d) = \sum_{k=0}^K r_k$. Let $D := \left\{d_{i,k} \; : \; i \in \{1,\dots,\lfloor r_k/2 \rfloor\}, k \in \{0,\dots,K-1\}\right\} \subseteq \mathbb{Z}_+$, $G(D) := \sum_{k=0}^{K-1}\sum_{i=1}^{\lfloor r_k/2 \rfloor} d_{i,k}$ and $\mathcal{P} := \left\{P_{i,k} \; : \; i \in \{1,\dots,\lfloor r_k/2 \rfloor\}, k \in \{0,\dots,K-1\}\right\}$ be a family of polyhedra such that, for each i and k, $P_{i,k} \subseteq \mathbb{R}^{3+d_{i,k}}$ is a (lifted) polyhedral relaxation of $\mathbf{L}^2$ with approximation quality $\varepsilon_k \in (0,1]$.*

Then

$$\widehat{T}^d\left(\mathcal{P}, D\right) := \left\{ (y_0, y, t, v) \in \times \mathbb{R}^{d+1+R(d)+G(D)} \; : \; \begin{array}{rl} y_0 = t_1^K, & \\ t_i^0 = y_i, & \forall i \in \{1,\ldots,d\}, \\ (t_i^{k+1}, t_{2i-1}^k, t_{2i}^k, v^{i,k}) \in P_{i,k}, & \forall i \in \{1,\ldots,\lfloor r_k/2 \rfloor\}, \\ & k \in \{0,\ldots,K-1\}, \\ t_{t_k}^k = t_{\lceil r_k/2 \rceil}^{k+1}, & \forall k \in \{0,\ldots,K-1\} \\ & \text{s.t. } r_k \text{ is odd} \end{array} \right\}.$$

is a lifted polyhedral relaxation of $\mathbf{L}^d$ *with approximation quality* $\varepsilon = \left(\prod_{k=0}^{K-1} 1+\varepsilon_k\right) - 1$. *In particular, let* $\widehat{L}_s^d := \widehat{T}^d\left(\mathcal{N}_s, D_{2s}\right)$ *where* $\mathcal{N}_s := \left\{\widehat{N}_s^2 \; : \; i \in \{1,\ldots,\lfloor r_k/2 \rfloor\},\, k \in \{0,\ldots,K-1\}\right\}$[1] *and*

$$D_{2s} := \left\{2s \; : \; i \in \{1,\ldots,\lfloor r_k/2 \rfloor\},\, k \in \{0,\ldots,K-1\}\right\}.$$

Then, for any $\varepsilon \in (0,1)$ *there exists an* $s\left(\varepsilon\right) \in \mathbb{Z}_+$ *such that* $\widehat{L}_{s(\varepsilon)}^d$ *has* $O\left(d \ln\left(K/\varepsilon\right)\right)$ *variables and constraints, and is a lifted polyhedral relaxation of* $\mathbf{L}^d$ *with approximation quality* ε.

It is not hard to see that for an appropriately chosen ε, $\widehat{L}_{s(\varepsilon)}^d$ is a lifted polyhedral relaxation with a polynomial number of variables and constraints that can prove $\mathbf{F}^n \cap \mathbb{Z}^n = \emptyset$. However, this can also be achieved with the tower of variables construction without the use of $\widehat{N}_s^2$. To show that we will use the following straightforward corollary of Proposition 2, which approximates each $\mathbf{L}^2$ in $\widehat{\mathbf{T}}^d$ with non-lifted approximation $O^d\left(\Omega\right)$ introduced in (2).

Corollary 1 *Let* $\Omega := \left\{\Omega_{i,k} \; : \; i \in \{1,\ldots,\lfloor t_k/2 \rfloor\}, \quad k \in \{0,\ldots,K-1\}\right\}$ *be such that* $\Omega_{i,k} \subseteq \mathbf{S}^1$ *and* $\left|\Omega_{i,k}\right| < \infty$ *for all* i,k. *For any such,* Ω *define* $\widehat{T}^d\left(\Omega\right) := \widehat{T}^d\left(\mathcal{O}, D_0\right)$ *for*

$$\mathcal{O} := \left\{O^2\left(\Omega_{i,k}\right) \; : \; i \in \{1,\ldots,\lfloor r_k/2 \rfloor\},\; k \in \{0,\ldots,K-1\}\right\}$$

and $D_0 := \{0 \; : \; i \in \{1,\ldots,\lfloor r_k/2 \rfloor\},\; k \in \{0,\ldots,K-1\}\}$. *Then* $\widehat{T}^d\left(\Omega\right)$, *is a lifted polyhedral relaxation of* $\mathbf{L}^d$ *with*

$$\sum_{k=0}^{K-1} \sum_{i=1}^{\lfloor t_k/2 \rfloor} \left|\Omega_{i,k}\right| \leq (d-1) \max_{\Omega_{i,k} \in \Omega} \left|\Omega_{i,k}\right|$$

constraints and $d-2$ *auxiliary variables*[2].

The following example shows how the tower of variables approximation based on both $\widehat{N}_s^2$ and $O^2\left(\Omega\right)$ result in small lifted polyhedral approximations that prove $\mathbf{F}^n \cap \mathbb{Z}^n = \emptyset$.

Example 2 Let $\Omega_{0,0} = \left\{\left(s_1/\sqrt{2}, s_2/\sqrt{2}\right)\right\}_{s \in \{-1,1\}^2}$, then we can check that

$$\mathrm{Proj}_{(y_0,y)}\left(\widehat{N}_s^2\right) = O^2\left(\Omega_{0,0}\right) = \left\{(y_0, y) \in \mathbb{R}_+ \times \mathbb{R}^2 \; : \; |y_1| + |y_2| \leq \sqrt{2} y_0\right\}.$$

Then, if we let $\Omega := \{\Omega_{0,0} \; : \; i \in \{1,\ldots,\lfloor t_k/2 \rfloor\}, \quad k \in \{0,\ldots,K-1\}\}$ then

$$\mathrm{Proj}_{(y_0,y)}\left(\widehat{L}_2^d\right) = \mathrm{Proj}_{(y_0,y)}\left(\widehat{T}^d\left(\Omega\right)\right).$$

[1] Or more precisely, $\mathcal{N} := \left\{N_{i,k} \; : \; i \in \{1,\ldots,\lfloor r_k/2 \rfloor\},\, k \in \{0,\ldots,K-1\}\right\}$ where $N_{i,k} = \widehat{N}_s^2$ for each i,k

[2] After removing redundant variables through the equalities of the formulation.

We can also check that

$$\min\left\{y_0 \,:\, (y_0,\bar{y}) \in \text{Proj}_{(y_0,y)}\left(\widehat{L}_2^d\right)\right\} = \frac{1}{2}\left(\frac{2}{\sqrt{2}}\right)^{\log_2(d)} = \sqrt{\frac{d}{4}}$$

for all $\bar{y} \in \{0,1\}^d$. Hence

$$\left\{(x,y,y_0) \in \mathbb{R}^{2n+1} \,:\, x_j - \frac{1}{2} = y_j \;\forall j \in \{1,\dots,n\},\; \sqrt{\frac{n-1}{4}} = y_0,\; (y_0,y) \in \text{Proj}_{(y_0,y)}(P)\right\} \cap \mathbb{Z}^n = \emptyset \quad (7)$$

for $P = \widehat{L}_2^d$ or $P = \widehat{T}^d(\Omega)$ and both of these formulations can be used to prove $\mathbf{F}^n \cap \mathbb{Z}^n = \emptyset$. Formulation $\widehat{T}^d(\Omega)$ achieves this with $4n-4$ constraints and $n-2$ auxiliary variables and $\widehat{L}_2^d$ achieves it with $4n-4$ constraints and $2n-1$ auxiliary variables.

From Examples 1 and 2 we have that the number of constraints in the projection onto the (y_0,y) variables of both $\widehat{T}^d(\Omega)$ and $\widehat{L}_s^d$ can have a number of facets that is exponential on their original number of variables and constraints. Hence, both formulations can provide a significant advantage over the traditional outer approximation formulation $O^d(\Omega)$. However, the full potential of the approximation by Ben-Tal and Nemirovski approximation is only achieved when $\widehat{N}_s^2$ is used (e.g. as in $\widehat{L}_s^d$). For instance, if $d=2$, $\widehat{L}_s^d$ reduces to $\widehat{N}_s^2$, which has a linear (in s) number of variables and constraints and which has a projection onto the (y_0,y) variables with an exponential number of constraints[3]. In contrast, for $d=2$, $\widehat{T}^d(\Omega)$ reduces to $O^d(\Omega)$ and hence we do not get any constraint multiplying effect through projections. Nonetheless, $\widehat{T}^d(\Omega)$ does have one advantage over $\widehat{L}_s^d$. While $\widehat{T}^d(\Omega)$ can be iteratively refined by simply augmenting Ω, the only way to refine $\widehat{L}_s^d$ is to refine each (or some) of the approximations $\widehat{N}_s^2$ used in its construction. Regrettably, the complexity of $\widehat{N}_s^2$ does not easily lend itself to an iterative refinement and the only known way to refine $\widehat{N}_s^2$ is to replace it with $N_{s'}^2$ for $s' > s$ constructed from scratch. Fortunately, there is an alternative lifted approximation that provides a middle ground between $\widehat{T}^d(\Omega)$ and $\widehat{L}_s^d$. This approximation can be constructed using the following proposition from [44].

Proposition 3 *Let $f_j : \mathbb{R} \to \mathbb{R}$ be strictly convex differentiable functions for each $j \in \{1,\dots,n\}$ and let $\Gamma := \left\{\Gamma_j \,:\, j \in \{1,\dots,n\}\right\}$ be such that $\Gamma_j \subseteq \mathbb{R}$ and $\left|\Gamma_j\right| < \infty$. If $\mathbf{G} := \left\{(y_0,y) \in \mathbb{R}^{n+1} \,:\, \sum_{j=1}^n f_j(y_j) \le y_0\right\}$, then*

$$f_j(\gamma) + f_j'(\gamma)(y_j - \gamma) \le w_j, \quad \forall \gamma \in \Gamma_j, \quad j \in \{1,\dots,n\}, \quad (8a)$$

$$\sum_{j=1}^n w_j \le y_0 \quad (8b)$$

is a lifted polyhedral relaxation of $\mathbf{G}$ with $1+\sum_{j=1}^n \left|\Gamma_j\right|$ constraints and whose projection onto the (y_0,y) variables can have up to $\prod_{j=1}^n \left|\Gamma_j\right|$ constraints.

While set $\mathbf{G}$ in Proposition 8 has a somewhat restrictive structure, [32] uses it to constructs the following linear sized polyhedral approximation that is able to prove $\mathbf{F}^n \cap \mathbb{Z}^n = \emptyset$.

Example 3 An alternative extended formulation of set $\mathbf{F}^n$ from Example 1 is given by

$$x_j - \frac{1}{2} = y_j, \qquad \forall j \in \{1,\dots,n\}, \quad (9a)$$

$$y_0 = \sqrt{\frac{n-1}{4}} \quad (9b)$$

$$\sum_{j=1}^n y_j^2 \le y_0^2. \quad (9c)$$

[3] For instance, one can check that $\left\{y \in \mathbb{R}^2 \,:\, \text{Proj}_{(1,y)}\left(\widehat{N}_s^2\right)\right\}$ is a regular 2^s polygon.

Constraint (9c) is not convex, but because y_0 is fixed to a constant, without loss of generality, we can replace (9b) by $y_0 = \frac{n-1}{4}$ and (9c) by $\sum_{j=1}^{n} y_j^2 \leq y_0$. Applying Proposition 3 for $f_j(y_j) = y_j^2$ and $\Gamma_j = \{-1/2, 1/2\}$ for all $j \in \{1, \ldots, n\}$ yields the polyhedral relaxation of (9), given by

$$x_j - \frac{1}{2} = y_j, \qquad \forall j \in \{1, \ldots, n\}, \tag{10a}$$

$$y_j - \frac{1}{4} \leq w_j, \qquad \forall j \in \{1, \ldots, n\}, \tag{10b}$$

$$-y_j - \frac{1}{4} \leq w_j, \qquad \forall j \in \{1, \ldots, n\}, \tag{10c}$$

$$\sum_{j=1}^{n} w_j \leq \frac{n-1}{4}. \tag{10d}$$

We can then check that $\left\{(x, y, w) \in \mathbb{R}^{3n} : (10), \quad x \in \mathbb{Z}^n\right\} = \emptyset$.

Formulation (8) shares with $\widehat{T}^d(\Omega)$ the possibility of being iteratively refined. Furthermore, while it does not reach the efficiency of $\widehat{L}_s^d$, it does provide a tangible improvement over $\widehat{T}^d(\Omega)$. For instance for $d = 2$ (8) with $r_1 + r_2 + 1$ constraints projects to a polyhedron with at most $r_1 \times r_2$ constraints. Hence (8) can have an up to quadratic constraint multiplying effect (cf. the exponential constraint multiplying effect of L_s^2 and the lack of constraint multiplying effect of $T^2(\Omega)$). However, while formulation (8) yields a polyhedral relaxation of $\mathbf{F}^n$, $\|y\|_2$ cannot be written as a sum of univariate functions and hence formulation (8) cannot be used directly to construct a polyhedral relaxation of $\mathbf{L}^d$. Fortunately, as we will show in the following section, a simple homogenization technique can be used to adapt (8) to yield lifted polyhedral relaxations of $\mathbf{L}^d$ and other *conic* sets. These lifted polyhedral relaxations share with (8) the possibility of being iteratively refined. To simplify exposition, from now on we refer to relaxations that can be iteratively refined as *dynamic* relaxations and to those that cannot be easily refined as *static* relaxations.

3 Dynamic Lifted Polyhedral Relaxations of Conic Sets

Proposition 3 cannot be directly used to construct a polyhedral relaxation of $\mathbf{L}^d$, but it can be used to construct relaxations of the euclidean ball $\mathbf{B}^d := \left\{y \in \mathbb{R}^d : \sum_{i=1}^{d} y_i^2 \leq 1\right\} = \left\{y \in \mathbb{R}^d : (1, y) \in \mathbf{L}^d\right\}$. Furthermore, $\mathbf{L}^d$ can be constructed from $\mathbf{B}^d$ through the homogenization $\mathbf{L}^d = \text{cone}\left(\{1\} \times \mathbf{B}^d\right)$. Hence, any relaxation of $\mathbf{B}^d$ can be transformed to a relaxation of $\mathbf{L}^d$ through a similar homogenization. This approach can be formally stated through the following proposition, which we prove in Appendix A (see Section 6 for an alternative derivation).

Proposition 4 *Let $f_j : \mathbb{R} \to \mathbb{R}$ be closed convex functions such that $\lim_{x\to\infty} \frac{f_j(x)}{|x|} = \infty$ for all $j \in \{1, \ldots, d\}$ so that the closure of the perspective function of f is given by*

$$\left(cl\tilde{f}\right)(t, x) = \begin{cases} tf(x/t) & t > 0 \\ 0 & x = 0 \text{ and } t = 0 \\ \infty & o.w. \end{cases}.$$

If $\mathbf{C} := \left\{y \in \mathbb{R}^d : \sum_{j=1}^{d} f_j(y_j) \leq 1\right\}$ and

$$\widehat{\mathbf{C}} := \left\{(y_0, y, w) \in \mathbb{R}^{2d+1} : \left(cl\tilde{f}_j\right)(y_0, y_j) \leq w_j, \quad \forall j \in \{1, \ldots, d\}, \quad \sum_{j=1}^{d} w_j \leq y_0\right\}, \tag{11}$$

then $\text{cone}(\{1\} \times \mathbf{C}) = \text{Proj}_{(y_0,y)}\left(\widehat{\mathbf{C}}\right)$.

Furthermore, if functions f_j are differentiable, then for any $\Gamma := \left\{\Gamma_j : j \in \{1,\dots,n\}\right\}$ such that $\Gamma_j \subseteq \mathbb{R}$ and $\left|\Gamma_j\right| < \infty$, the set

$$\widehat{C}(\Gamma) := \left\{ (y_0,y,w) \in \mathbb{R}^{2d+1} : \begin{array}{c} \left(f(\gamma) - \gamma f'(\gamma)\right) y_0 + f'(\gamma)\, y_j \leq w_j, \quad \forall \gamma \in \Gamma_j,\ j \in \{1,\dots,n\} \\ \sum_{j=1}^{n} w_j \leq y_0 \end{array} \right\}$$

is a lifted polyhedral relaxation of $\operatorname{cone}(\{1\} \times \mathbf{C})$ *with* $1 + \sum_{j=1}^{n} \left|\Gamma_j\right|$ *constraints. If the functions are additionally strictly convex the projection onto the variables (y_0, y) of this relaxation can have up to $\prod_{j=1}^{n} \left|\Gamma_j\right|$ constraints.*

Finally, if $(y_0,y,w) \in \widehat{C}(\Gamma) \setminus \widehat{\mathbf{C}}$, then there exists $j \in \{1,\dots,n\}$ and $\gamma \in \mathbb{R}$ such that $(y_0,y,w) \notin \widehat{C}(\Gamma)$ if we augment Γ_j to $\Gamma_j \cup \{\gamma\}$.

Obtaining a dynamic lifted polyhedral relaxation for $\mathbf{L}^d$ is a direct corollary of Proposition 4 by letting $f_j(x_j) = x_j^2$ and using $\mathbf{L}^d = \operatorname{cone}\left(\{1\} \times \mathbf{B}^d\right)$ (See [5, p. 109] for an alternative derivation). One notable difference between the general approximation from Proposition 4 and the approximation of $\mathbf{L}^d$ from Corollary 2 below is the inclusion of constraint $y_0 \geq 0$ in the later. This constraint is excluded in Proposition 4, but is implied by the nonlinear constraints defining $\widehat{\mathbf{C}}$ and can be arbitrarily approximated by the linear constraints of $\widehat{C}(\Gamma)$ by appropriately selecting Γ (e.g. see the proof of Lemma 2 in Appendix A). The constraint is explicitly included in Corollary 2 to ensure conic quadratic representability of $\widehat{\mathbf{H}}^d$ and for computational convenience and numerical stability in $\widehat{H}^d(\Gamma)$.

Corollary 2 *Let*

$$\widehat{\mathbf{H}}^d := \left\{ (y_0,y,w) \in \mathbb{R}^{2d+1} : y_j^2 \leq w_j y_0, \quad \forall j \in \{1,\dots,d\}, \quad \sum_{j=1}^{d} w_j \leq y_0, \quad 0 \leq y_0 \right\}, \tag{12}$$

then $\mathbf{L}^d = \operatorname{Proj}_{(y_0,y)}\left(\widehat{\mathbf{H}}^d\right)$ *and hence $\widehat{\mathbf{H}}^d$ is a lifted reformulation of $\mathbf{L}^d$ with d rotated two-dimensional conic quadratic constraints, one linear constraint*[4] *and d auxiliary variables.*

Furthermore, for any $\Gamma := \left\{\Gamma_j : j \in \{1,\dots,n\}\right\}$ for $\Gamma_j \subseteq \mathbb{R}$ with $\left|\Gamma_j\right| < \infty$, the set

$$\widehat{H}^d(\Gamma) := \left\{ (y_0,y,w) \in \mathbb{R}^{2d+1} : \begin{array}{c} 2\gamma y_j - \gamma^2 y_0 \leq w_j, \quad \forall j \in \{1,\dots,d\},\ \gamma \in \Gamma_j, \\ \sum_{j=1}^{d} w_j \leq y_0, \\ 0 \leq y_0. \end{array} \right\}$$

is a lifted polyhedral relaxation of $\mathbf{L}^d$.

Finally, if $(y_0,y,w) \in \widehat{H}^d(\Gamma) \setminus \widehat{\mathbf{H}}^d$ and $y_0 \neq 0$, then there exists $j \in \{1,\dots,d\}$ such that $(y_0,y,w) \notin \widehat{H}^d(\Gamma)$ if we augment Γ_j to $\Gamma_j \cup \left\{\gamma(y_0,y_j)\right\}$ where $\gamma(y_0,y_j) := y_j/y_0$. Similarly if $(y_0,y,w) \in \widehat{H}^d(\Gamma) \setminus \widehat{\mathbf{H}}^d$ and $y_0 = 0$, then $(y_0,y,w) \notin \widehat{H}^d(\Gamma)$ if we augment Γ_j for all $j \in \{1,\dots,d\}$ to $\Gamma_j \cup \{-\gamma,\gamma\}$ for any $\gamma > 0$.

With a slight abuse of terminology, we refer to $\widehat{H}^d(\Gamma)$ as the separable relaxation to emphasize the fact that it considers nonlinearities one variable at a time in a similar way to Proposition 3.

[4] Note that the complete description of each rotated two-dimensional conic quadratic constraints is $y_j^2 \leq w_j y_0$ and $0 \leq y_0$.

3.1 Relation to Perspective Reformulations

Perspective functions have been used to model unions of convex sets [12, 25, 43] for many years, with a recent emphasis on modeling of the union of a point and a single convex set [19, 21, 22, 26, 27, 28, 31]. We now show how these techniques can be adapted to give an alternative construction of the lifted separable reformulation $\widehat{\mathbf{C}}$.

The alternate construction of $\widehat{\mathbf{C}}$ is based on the lifted reformulation of $\mathbf{C} := \left\{ y \in \mathbb{R}^d \, : \, \sum_{j=1}^d f_j(y_j) \leq 1 \right\}$ given by

$$\mathbf{C}_{+,1} := \left\{ (y, y_0, w) \in \mathbb{R}^{2d+1} \, : \, f_j(y_j) \leq w_j, \quad \forall j \in \{1, \ldots, d\}, \quad \sum_{j=1}^d w_j \leq y_0, \quad y_0 = 1 \right\}$$

and the set $\mathbf{C}_{+,0} := \left\{ (y, y_0, w) \in \mathbb{R}^{2d+1} \, : \, y_0 = y_i = w_i = 0, \quad \forall i \in \{1, \ldots, d\} \right\}$. Using known results (e.g. Section 3 of [27]) we have that, under the assumptions of Proposition 4,

$$\operatorname{conv}\left(\mathbf{C}_{+,1} \cup \mathbf{C}_{+,0}\right) = \left\{ (y, y_0, w) \in \mathbb{R}^{2d+1} \, : \, \begin{array}{c} \left(cl\tilde{f}_j\right)(y_0, y_j) \leq w_j, \; \forall j \in \{1, \ldots, d\}, \\ \sum_{j=1}^d w_j \leq y_0, \\ 0 \leq y_0 \leq 1 \end{array} \right\}. \tag{13}$$

We can also check that $\operatorname{cone}\left(\{1\} \times \mathbf{C}\right) = \operatorname{Proj}_{(y_0,y)}\left(\operatorname{cone}\left(\operatorname{conv}\left(\mathbf{C}_{+,1} \cup \mathbf{C}_{+,0}\right)\right)\right)$ and that $\operatorname{cone}\left(\operatorname{conv}\left(\mathbf{C}_{+,1} \cup \mathbf{C}_{+,0}\right)\right)$ is obtained by removing $y_0 \leq 1$ from the right hand side of (13), from which we precisely obtain $\widehat{\mathbf{C}}$[5]. Finally, inequalities $\left(f(\gamma) - \gamma f'(\gamma)\right) y_0 + f'(\gamma)\, y \leq w_j$ from the polyhedral approximation of $\mathcal{C}$ can be obtained from the *perspective cuts* from [19].

4 Lifted LP-Based Algorithms

We can construct lifted LP-based algorithms for MICQP by combining lifted polyhedral relaxations of $\mathbf{L}^d$ with the general algorithmic framework of traditional LP-based branch-and-bound algorithms [3, 9, 10, 18, 41]. For static relaxations such as $\widehat{L}^d_{s(\varepsilon)}$, [45] introduces the LiftedLP algorithm, which uses branching and the sporadic solutions of nonlinear relaxations to avoid the need to refine the polyhedral relaxation. Similarly, for dynamic polyhedral relaxations such as $\widehat{H}^d(\Gamma)$ we can follow the approach of [32] and adapt a traditional LP-based branch-and-bound algorithm to construct and refine the polyhedral relaxation on the space of original and auxiliary variables. However, for the polyhedral relaxations that are based on a nonlinear reformulation of $\mathbf{L}^d$, such as $\widehat{\mathbf{T}}^d$ defined in (6), we can more easily adapt a traditional LP-based algorithm by simply giving it this reformulation. We now describe how to do this for three relaxations. We then describe two versions of the LiftedLP algorithm [45] that we test in Section 5.

4.1 Algorithms from Nonlinear Reformulations

The first two relaxations we consider correspond to the tower of variables reformulation $\widehat{\mathbf{T}}^d$ and the separable reformulation $\widehat{\mathbf{H}}^d$ from Corollary 2. The third one is a combination of these two relaxations. For all three

[5] Remember that, while $\widehat{\mathbf{C}}$ does not explicitly include $y_0 \geq 0$, any point in $\widehat{\mathbf{C}}$ satisfies this constraint under the assumptions of Proposition 4 (e.g. see proof of Lemma 2 in Appendix A).

relaxations we give to the LP-based branch-and-bound algorithm a reformulation of MICQP of the form

$$\begin{aligned}
\text{obj}_{\mathsf{MICQP}\left(\{\widehat{\mathbf{Q}}^{d_l}\}_{l=1}^{k}\right)} := \max \quad & cx && && (14a)\\
s.t. \quad & \\
& Ex \leq h, && && (14b)\\
& A^l x + b^l = y^l, && \forall l \in \{1,\dots,q\}, && (14c)\\
& a^l \cdot x + b_0^l = y_0^l, && \forall l \in \{1,\dots,q\}, && (14d)\\
& \left(y_0^l, y^l, z^l\right) \in \widehat{\mathbf{Q}}^{d_l}, && \forall l \in \{1,\dots,q\}, && (14e)\\
& x \in \mathbb{R}^n, && && (14f)\\
& x_j \in \mathbb{Z}^n, && \forall j \in I, && (14g)
\end{aligned}$$

where $\widehat{\mathbf{Q}}^{d_l}$ is a lifted reformulation of $\mathbf{L}^{d_l}$ such that $\text{Proj}_{\left(y_0^l, y^l\right)}\left(\widehat{\mathbf{Q}}^{d_l}\right) = \mathbf{L}^{d_l}$. Because the LP-based algorithm will consider auxiliary variables z^l as formulation variables it will construct and refine a polyhedral relaxation of nonlinear constraints (14e) in the $\left(y_0^l, y^l, z^l\right)$ variable space. This will effectively construct a lifted polyhedral relaxation of $\mathbf{L}^{d_l}$ using auxiliary variables z^l.

The nonlinear reformulation associated to the tower of variables relaxation from Corollary 1 is equal $\widehat{\mathbf{T}}^d$ defined in (6). This reformulation uses auxiliary variables $z^l = t^l \in \mathbb{R}^{R(d_l)}$ for $R(d)$ defined in Proposition 2 and corresponds to replacing (14e) with

$$\begin{aligned}
y_0^l = t_1^{l,K_l}, \quad & \forall l \in \{1,\dots,q\}, && (15a)\\
t_i^{l,0} = y_i^l, \quad & \forall i \in \{1,\dots,d_l\}, l \in \{1,\dots,q\}, && (15b)\\
(t_i^{l,k+1}, t_{2i-1}^{l,k}, t_{2i}^{l,k}) \in \mathbf{L}^2, \quad & \forall i \in \{1,\dots,\lfloor r_{l,k/2} \rfloor\}, k \in \{0,\dots,K_l-1\}, l \in \{1,\dots,q\}, && (15c)\\
t_{r_{l,k}}^{l,k} = t_{\lceil r_{l,k}/2 \rceil}^{l,k+1}, \quad & \forall k \in \{0,\dots,K_l-1\} \text{ s.t. } r_{l,k} \text{ is odd}, l \in \{1,\dots,q\}, && (15d)
\end{aligned}$$

where K_l and $\{r_{l,k}\}_{k=0}^{K_l}$ correspond to K and $\{r_k\}_{k=0}^{K}$ defined in Proposition 2 for $d = d_l$.

The nonlinear reformulation associated to the separable relaxation described in Corollary 2 is equal to $\widehat{\mathbf{H}}^d$ defined in (12). This reformulation uses auxiliary variables $z = w \in \mathbb{R}^d$ and corresponds to replacing (14e) with

$$\begin{aligned}
\left(y_j^l\right)^2 \leq w_j^l y_0^l, \quad & \forall j \in \{1,\dots,d_l\}, l \in \{1,\dots,q\}, && (16a)\\
\sum_{j=1}^{d} w_j^l \leq y_0^l, \quad & \forall l \in \{1,\dots,q\}, && (16b)\\
0 \leq y_0^l, \quad & \forall l \in \{1,\dots,q\}. && (16c)
\end{aligned}$$

The final reformulation is a combination of the previous two that replaces $\mathbf{L}^2$ in (15c) with $\widehat{\mathbf{H}}^2$. This reformulation uses auxiliary variables $z^l = \left(t^l, v^l\right) \in \mathbb{R}^{R(d_l)+G\left(D_2^l\right)}$ for $R(d)$ and $G(D)$ defined in Proposition 2 and $D_2^l := \left\{2 : i \in \{1,\dots,\lfloor r_{l,k}/2 \rfloor\},\ k \in \{0,\dots,K_l-1\}\right\}$ where K_l and $\{r_{l,k}\}_{k=0}^{K_l}$ correspond to K

and $\{r_k\}_{k=0}^K$ defined in Proposition 2 for $d = d_l$. The version of constraint (14e) for this case is

$$y_0^l = t_1^{l,K}, \qquad \forall l \in \{1,\dots,q\}, \tag{17a}$$
$$t_i^{l,0} = y_i^l, \qquad \forall i \in \{1,\dots,d\}, l \in \{1,\dots,q\}, \tag{17b}$$
$$\left(t_{2i-1}^{l,k}\right)^2 \le v_{i,1}^{l,k} t_i^{l,k+1}, \quad \forall i \in \{1,\dots,\lfloor r_{l,k}/2 \rfloor\}, k \in \{0,\dots,K_l-1\}, l \in \{1,\dots,q\}, \tag{17c}$$
$$\left(t_{2i}^{l,k}\right)^2 \le v_{i,2}^{l,k} t_i^{l,k+1}, \quad \forall i \in \{1,\dots,\lfloor r_{l,k}/2 \rfloor\}, k \in \{0,\dots,K_l-1\}, l \in \{1,\dots,q\}, \tag{17d}$$
$$v_{i,1}^{l,k} + v_{i,1}^{l,k} \le t_i^{l,k+1}, \quad \forall i \in \{1,\dots,\lfloor r_{l,k}/2 \rfloor\}, k \in \{0,\dots,K_l-1\}, l \in \{1,\dots,q\}, \tag{17e}$$
$$t_{r_{l,k}}^{l,k} = t_{\lceil r_{l,k}/2 \rceil}^{l,k+1}, \quad \forall k \in \{0,\dots,K_l-1\} \text{ s.t. } r_{l,k} \text{ is odd}, l \in \{1,\dots,q\}. \tag{17f}$$

In Section 5 we compare the effectiveness of these nonlinear reformulations with two versions of the LiftedLP algorithm of [45], which we now describe in detail.

4.2 Branch-based LiftedLP Algorithm and Cut-based Adaptation

The original LiftedLP algorithm of [45] uses a version of $\widehat{L}^d_{s(\varepsilon)}$ introduced in [24]. This version corresponds to the static lifted polyhedral relaxation described by the following corollary.

Corollary 3 *Let $K = \lceil \log_2(d) \rceil$ and $\{r_k\}_{k=0}^K$ be defined by the recursion $r_0 = d$, $r_{k+1} = \lceil r_k/2 \rceil$ for $k \in \{0,\dots,K-1\}$. Furthermore, for any $\varepsilon \in (0, 1/2)$ let*

$$s_k(\varepsilon) = \left\lceil \frac{k+1}{2} \right\rceil - \left\lceil \log_4 \left(\frac{16}{9} \pi^{-2} \log(1+\varepsilon) \right) \right\rceil$$

for each $k \in \{0,\dots,K-1\}$. Finally, let $L^d_\varepsilon := \widehat{T}^d(\mathcal{N}_\varepsilon, D_\varepsilon)$ where

$$\mathcal{N}_\varepsilon := \left\{ N^2_{s_k(\varepsilon)} : i \in \{1,\dots,\lfloor r_k/2 \rfloor\}, k \in \{0,\dots,K-1\} \right\}$$

and $D_\varepsilon := \{2s_k(\varepsilon) : i \in \{1,\dots,\lfloor r_k/2 \rfloor\}, k \in \{0,\dots,K-1\}\}$. Then, L^d_ε is a lifted polyhedral relaxation of $\mathbf{L}^d$ with approximation quality ε and $O(d \log(1/\varepsilon))$ variables and constraints.

The LiftedLP algorithm replaces every occurrence $\mathbf{L}^d$ with L^d_ε for an appropriately chosen ε. To avoid the need to refine these approximations the algorithm sporadically solves some nonlinear relaxations and follows a specialized branching convention. This branch-based approach to the LiftedLP algorithm was motivated by the ineffectiveness of traditional polyhedral relaxations of $\mathbf{L}^d$ in certain classes of problems. However, the advent of dynamic lifted polyhedral relaxations suggests the possible effectiveness of a cut-based version of the LiftedLP algorithm. We now concurrently describe the original branch-based and the cut-based variant of the LiftedLP algorithm. The cut-based variant combines static relaxation L^d_ε with a dynamic relaxation of $\mathbf{L}^d$. While any of the relaxations described in the previous section can be used, preliminary computational test showed that $\widehat{H}^d(\Gamma)$ provides comparable or better performance than the other relaxations. For this reason, and because the extension to other dynamic relaxations is straightforward, we here only describe the variant for $\widehat{H}^d(\Gamma)$.

Both versions of the LiftedLP algorithm are based on the LP relaxation of MICQP defined in (3) given by

$$\begin{aligned}
\text{obj}_{\mathsf{LP}\left(\mathbf{l},\mathbf{u},\left\{\Gamma^l\right\}_{l=1}^k\right)} := \max \quad & cx && && \text{(18a)}\\
s.t. \quad & \\
& Ey \leq h, && && \text{(18b)}\\
& A^l x + b^l = y^l, && l \in \{1,\dots,q\}, && \text{(18c)}\\
& a^l \cdot x + b_0^l = y_0^l, && l \in \{1,\dots,q\}, && \text{(18d)}\\
& \left(y^l, y_0^l, t^l, v^l\right) \in L_{\varepsilon}^{d_l}, && l \in \{1,\dots,q\}, && \text{(18e)}\\
& \left(y^l, y_0^l, w^l\right) \in \widehat{H}^{d_l}\left(\Gamma^l\right), && l \in \{1,\dots,q\}, && \text{(18f)}\\
& \mathbf{l}_j \leq x_j \leq \mathbf{u}_j, && j \in \{1,\dots,n\}. && \text{(18g)}
\end{aligned}$$

where Γ^l corresponds to the set of inequalities currently used by the algorithms. If $\Gamma^l = \emptyset$ for all $l \in \{1,\dots,q\}$ constraints (18f) do not restrict $\left(y^l, y_0^l\right)$ in any way, so we assume that in such case (18f) is simply omitted. Vectors $\mathbf{l}, \mathbf{u} \in (\mathbb{R} \cup \{-\infty, \infty\})^n$ correspond to variable lower and upper bounds that are used to define a node in the branch-and-bound tree. These bounds are initially infinite ($\mathbf{l}_j = -\infty$ an $\mathbf{u}_j = \infty$ for all $j \in \{1,\dots,n\}$) and are only modified for integer constrained variables x_j for $j \in I$. The infinite bounds for the continuous variables are included to simplify notation. We denote the relaxation defined in (18) $\mathsf{LP}\left(\mathbf{l}, \mathbf{u}, \left\{\Gamma^l\right\}_{l=1}^k\right)$. The algorithms also use the following nonlinear relaxation to sporadically compute node-bounds or as a heuristic to find feasible solutions.

$$\begin{aligned}
\text{obj}_{\mathsf{CP}(\mathbf{l},\mathbf{u})} := \max \quad & cx && && \text{(19a)}\\
s.t. \quad & \\
& Ey \leq h, && && \text{(19b)}\\
& A^l x + b^l = y^l, && l \in \{1,\dots,q\}, && \text{(19c)}\\
& a^l \cdot x + b_0^l = y_0^l, && l \in \{1,\dots,q\}, && \text{(19d)}\\
& \left(y^l, y_0^l\right) \in \mathbf{L}^{d_l}, && l \in \{1,\dots,q\}, && \text{(19e)}\\
& \mathbf{l}_j \leq x_j \leq \mathbf{u}_j, && j \in \{1,\dots,n\}. && \text{(19f)}
\end{aligned}$$

We denote this relaxation $\mathsf{CP}(\mathbf{l}, \mathbf{u})$.

The final ingredient in the algorithms is a list of branch-and-bound nodes to be processed, which we denote $\mathcal{M}$. The nodes in this list are characterized by variable upper and lower bounds $(\mathbf{l}, \mathbf{u})$ and an estimated upper bound of the nonlinear relaxation $\mathsf{CP}(\mathbf{l}, \mathbf{u})$. The objective of the algorithm is to simulate a nonlinear branch-and-bound based on $\mathsf{CP}(\mathbf{l}, \mathbf{u})$, but by solving $\mathsf{LP}\left(\mathbf{l}, \mathbf{u}, \left\{\Gamma^l\right\}_{l=1}^k\right)$ in each node and only sporadically solving $\mathsf{CP}(\mathbf{l}, \mathbf{u})$. A generic version of this procedure is described in Algorithm 2. The basis of Algorithm 2 is a branch-and-bound procedure that solves $\mathsf{LP}\left(\mathbf{l}, \mathbf{u}, \left\{\Gamma^l\right\}_{l=1}^k\right)$ in each branch-and-bound node. If the optimal value of this relaxation is worse than that of the incumbent feasible solution the node is fathomed by bound in line 9. If the optimal value is better than the incumbent and the solution to the LP relaxation does not satisfy the integrality constraints of MICQP, the algorithm branches on an integer constrained variable with a fractional value in the traditional way. This is done in lines 23–28 of the algorithm. If the optimal value is better than the incumbent and the solution of the LP relaxation satisfies the integrality constraints of MICQP, the algorithm first checks if the nonlinear constraints of MICQP happen

Algorithm 2: A Generic lifted LP-based Branch-and-Bound Algorithm.

1 Set global lower bound $\text{LB} := -\infty$.
2 Set $\mathbf{l}_j^0 := -\infty$, $\mathbf{u}_j^0 := +\infty$ for all $j \in \{1, \ldots, n\}$.
3 Set $\text{UB}^0 = +\infty$.
4 Set node list $\mathcal{M} := \{(\mathbf{l}^0, \mathbf{u}^0, \text{UB}^0)\}$.
5 Set initial approximation $\Gamma^l := \Gamma_0^l$ for all $l \in \{1, \ldots, q\}$.
6 **while** $\mathcal{M} \neq \emptyset$ **do**
7 Select and remove a node $(\mathbf{l}^k, \mathbf{u}^k, \text{UB}^k) \in \mathcal{M}$.
8 Solve $\mathsf{LP}\left(\mathbf{l}^k, \mathbf{u}^k, \left\{\Gamma^l\right\}_{l=1}^q\right)$.
9 **if** $\mathsf{LP}\left(\mathbf{l}^k, \mathbf{u}^k, \left\{\Gamma^l\right\}_{l=1}^q\right)$ *is feasible* **and** $\text{obj}_{\mathsf{LP}\left(\mathbf{l}^k, \mathbf{u}^k, \left\{\Gamma^l\right\}_{l=1}^q\right)} > \text{LB}$ **then**
10 Let $\left(\bar{x}, \left\{\left(\bar{y}^l, \bar{y}_0^l, \bar{z}^l\right)\right\}_{l=1}^q\right)$ be the optimal solution to $\mathsf{LP}\left(\mathbf{l}^k, \mathbf{u}^k, \left\{\Gamma^l\right\}_{l=1}^q\right)$.
11 **if** $\bar{x}_j \in \mathbb{Z}$ *for all* $j \in I$ **then**
12 **if** $\left(\bar{y}^l, \bar{y}_0^l\right) \in \mathbf{L}^{d_l}$ *for all* $l \in \{1, \ldots, q\}$ **then**
13 Set $\text{LB} := \text{obj}_{\mathsf{LP}\left(\mathbf{l}^k, \mathbf{u}^k, \left\{\Gamma^l\right\}_{l=1}^q\right)}$.
14 **else** `/* Run Heuristic and Refine */`
15 Let $\mathbf{l}_j = \mathbf{l}_j^k$, $\mathbf{u}_j = \mathbf{u}_j^k$ for all $j \in \{1, \ldots, n\} \setminus I$.
16 Let $\mathbf{l}_j = \mathbf{u}_j = \bar{x}_j$ for all $j \in I$.
17 Solve $\mathsf{CP}(\mathbf{l}, \mathbf{u})$.
18 **if** $\mathsf{CP}(\mathbf{l}, \mathbf{u})$ *is feasible* **and** $\text{obj}_{\mathsf{CP}(\mathbf{l}, \mathbf{u})} > \text{LB}$ **then**
19 $\text{LB} := \text{obj}_{\mathsf{CP}(\mathbf{l}, \mathbf{u})}$.
20 **end**
21 Call $\mathsf{REFINE}\left(\mathbf{l}^k, \mathbf{u}^k, \left\{\left(\bar{y}^l, \bar{y}_0^l, \bar{z}^l\right)\right\}_{l=1}^q, \text{LB}, \text{obj}_{\mathsf{LP}(\mathbf{l}^k, \mathbf{u}^k)}, \mathcal{M}, \left\{\Gamma^l\right\}_{l=1}^q\right)$.
22 **end**
23 **else** `/* Branch on` $\bar{x}$ `*/`
24 Pick j_0 in $\{j \in I : \bar{x}_j \notin \mathbb{Z}\}$.
25 Let $\mathbf{l}_j = \mathbf{l}_j^k$, $\mathbf{u}_j = \mathbf{u}_j^k$ for all $j \in \{1, \ldots, n\} \setminus \{j_0\}$.
26 Let $\mathbf{u}_{j_0} = \lfloor \bar{x}_{j_0} \rfloor$, $\mathbf{l}_{j_0} = \lfloor \bar{x}_{j_0} \rfloor + 1$.
27 $\mathcal{M} := \mathcal{M} \cup \left\{\left(\mathbf{l}^k, \mathbf{u}, \text{obj}_{\mathsf{LP}(\mathbf{l}^k, \mathbf{u}^k)}\right), \left(\mathbf{l}, \mathbf{u}^k, \text{obj}_{\mathsf{LP}(\mathbf{l}^k, \mathbf{u}^k)}\right)\right\}$
28 **end**
29 **end**
30 Remove every node $(\mathbf{l}^k, \mathbf{u}^k, \text{UB}^k) \in \mathcal{M}$ such that $\text{UB}^k \leq \text{LB}$.
31 **end**

to be satisfied in line 12. If the nonlinear constraints are satisfied the incumbent solution is updated in line 13 (as stated the algorithm only updates the value of the solution; the modification to update the solution itself is straightforward). If the nonlinear constraints are not satisfied, the algorithm first attempts to find a feasible solution to MICQP that has the same values in the integer variables as the solution to $\mathsf{LP}\left(\mathbf{l}, \mathbf{u}, \left\{I^l\right\}_{l=1}^k\right)$ for the current node. This is done by solving $\mathsf{CP}\left(\mathbf{l}, \mathbf{u}\right)$ for appropriately chosen bounds in lines 15–17. If this heuristic is successful and yields a better solution the incumbent is updated in lines 18–20. Finally, a generic refinement procedure for the current branch-and-bound node is called in line 21.

The original LiftedLP algorithm from [45] is obtained when we let $\Gamma_0^l = \emptyset$ for all $l \in \{1, \ldots, q\}$ and we use the branch-based refinement procedure described in Algorithm 3. The procedure begins by solving the nonlinear relaxation that would have been solved by an NLP-based branch-and-bound algorithm in the current node. The procedure then processes this node exactly as an NLP-based branch-and-bound algorithm would. In line 2 it first checks if the node can be fathomed by bound or infeasibility. If this fails, the procedure attempts to fathom by integrality in lines 4–6. Finally, if all the previous steps fail, the procedure branches on an integer constrained variable with a fractional value in lines 6–11.

Algorithm 3: Branch-based refinement, Branch. REFINE $\left(\mathbf{l}^k, \mathbf{u}^k, \left\{\left(\bar{y}^l, \bar{y}_0^l, \bar{w}^l\right)\right\}_{l=1}^q, \text{LB}, \text{obj}_{\mathsf{LP}(\mathbf{l}^k,\mathbf{u}^k)}, \mathcal{M}, \left\{\Gamma^l\right\}_{l=1}^q\right)$

Input: Lower and upper variable bounds $\left(\mathbf{l}^k, \mathbf{u}^k\right)$, solution $\left\{\left(\bar{y}^l, \bar{y}_0^l, \bar{w}^l\right)\right\}_{l=1}^q$, lower bound LB, node bound $y_{\mathsf{LP}(\mathbf{l}^k,\mathbf{u}^k)}$, node list $\mathcal{M}$ and cut list $\left\{\Gamma^l\right\}_{l=1}^q$.

Solve $\mathsf{CP}(\mathbf{l}^k, \mathbf{u}^k)$.
if $\mathsf{CP}(\mathbf{l}^k, \mathbf{u}^k)$ *is feasible* **and** $\text{obj}_{\mathsf{CP}(\mathbf{l}^k,\mathbf{u}^k)} > \text{LB}$ **then**
  Let $(\tilde{x}^k, \tilde{y}^k)$ be the optimal solution to $\mathsf{CP}(\mathbf{l}^k, \mathbf{u}^k)$.
  if $\tilde{x}^k \in \mathbb{Z}^n$ **then** `/* Fathom by Integrality */`
    $\text{LB} := \text{obj}_{\mathsf{CP}(\mathbf{l}^k,\mathbf{u}^k)}$.
  else `/* Branch on` $\tilde{x}^k$ `*/`
    Pick j_0 in $\{j \in I : \tilde{x}_j^k \notin \mathbb{Z}\}$.
    Let $\mathbf{l}_j = \mathbf{l}_j^k$, $\mathbf{u}_j = \mathbf{u}_j^k$ for all $j \in \{1, \ldots, n\} \setminus \{j_0\}$.
    Let $\mathbf{u}_{j_0} = \lfloor \tilde{x}_{j_0}^k \rfloor$, $\mathbf{l}_{j_0} = \lfloor \tilde{x}_{j_0}^k \rfloor + 1$.
    $\mathcal{M} := \mathcal{M} \cup \left\{\left(\mathbf{l}^k, \mathbf{u}, \text{obj}_{\mathsf{CP}(\mathbf{l}^k,\mathbf{u}^k)}\right), \left(\mathbf{l}, \mathbf{u}^k, \text{obj}_{\mathsf{CP}(\mathbf{l}^k,\mathbf{u}^k)}\right)\right\}$
  end
end

Finally, a cut-based version of the algorithm that truly uses $H^{d_l}\left(\Gamma^l\right)$ is obtained when we use the cut-based refinement procedure described in Algorithm 4. This procedure first checks for a violation of the

Algorithm 4: Cut-based refinement, Cut. REFINE $\left(\mathbf{l}^k, \mathbf{u}^k, \left\{\left(\bar{y}^l, \bar{y}_0^l, \bar{w}^l\right)\right\}_{l=1}^q, \text{LB}, \text{obj}_{\mathsf{LP}(\mathbf{l}^k,\mathbf{u}^k)}, \mathcal{M}, \left\{\Gamma^l\right\}_{l=1}^q\right)$

Input: Lower and upper variable bounds $\left(\mathbf{l}^k, \mathbf{u}^k\right)$, solution $\left\{\left(\bar{y}^l, \bar{y}_0^l, \bar{w}^l\right)\right\}_{l=1}^q$, lower bound LB, node bound $y_{\mathsf{LP}(\mathbf{l}^k,\mathbf{u}^k)}$, node list $\mathcal{M}$ and cut list $\left\{\Gamma^l\right\}_{l=1}^q$.

1 **for** $l = 1$ **to** q **do**
2 **if** $\left(\bar{y}^l, \bar{y}_0^l\right) \notin \mathbf{L}^{d_l}$ **then**
3 **for** $j = 1$ **to** d_l **do**
4 **if** $\left(\bar{y}_j^l\right)^2 > \bar{w}_j^l \bar{y}_0^l$ **then**
5 Set $\Gamma_j^l := \Gamma_j^l \cup \left\{\bar{y}_j^l / \bar{y}_0^l\right\}$
6 **end**
7 **end**
8 **end**
9 **end**
10 $\mathcal{M} := \mathcal{M} \cup \left\{\left(\mathbf{l}^k, \mathbf{u}^k, \text{obj}_{\mathsf{LP}(\mathbf{l}^k,\mathbf{u}^k)}\right)\right\}$

nonlinear constraints by the current node's solution in lines 1–2. If a violation is found, then the separation procedure for $H^{d_l}\left(\Gamma^l\right)$ is called in line 3. This separation procedure updates Γ^l with one or more inequality violated by the node solution.

5 Computational Experiments

In this section we compare the performance of the different algorithms on the portfolio optimization instances considered in [45]. We begin by describing the portfolio optimization instances and how the different algorithms are implemented. We then present the results of the computational experiments.

5.1 Instances

The instances from [45] correspond to three classes of portfolio optimization instances with limited diversification or cardinality constraints [7, 8, 14, 40] which are formulated as MICQPs. All three problems construct a portfolio out of n assets with an expected return $\bar{a} \in \mathbb{R}^n$. The objective is to maximize the return of the portfolio subject to various risk constraints and limitation on the number of assets considered in the portfolio. To simplify the description of the three versions we deviate from the conventions of (1) and use different names for continuous and integer constrained variables. With this the first class of problems we consider are of the form

$$\begin{aligned}
\max \quad & \bar{a}x && \text{(20a)}\\
s.t. \quad & \\
& \left\|Q^{1/2}x\right\|_2 \leq \sigma, && \text{(20b)}\\
& \sum_{j=1}^{n} x_j = 1, && \text{(20c)}\\
& x_j \leq z_j, \qquad \forall j \in \{1,\ldots,n\}, && \text{(20d)}\\
& \sum_{j=1}^{n} z_j \leq K, && \text{(20e)}\\
& z \in \{0,1\}^n, && \text{(20f)}\\
& x \in \mathbb{R}^n_+, && \text{(20g)}
\end{aligned}$$

where x indicates the fraction of the portfolio invested in each asset, $Q^{1/2}$ is the positive semidefinite square root of the covariance matrix of the returns of the stocks, σ is the maximum allowed risk and $K < n$ is the maximum number of assets that can be included in the portfolio. We refer to this class of instances as the *classical* instances.

The second class of problems is obtained by replacing constraint (20b) with a shortfall constraint considered in [37, 38]. This constraint can be formulated as

$$\Phi^{-1}(\eta_i)\left\|Q^{1/2}y\right\|_2 \leq \bar{a}y - W_i^{low}, \qquad \forall i \in \{1,2\}$$

where W_i^{low} and η_i are given parameters and $\Phi(\cdot)$ is the cumulative distribution function of the Normal distribution with zero mean and unit standard deviation. We refer to this class of instances as the *shortfall* instances.

The third and final class of problems correspond to a robust version of (20) introduced in [13]. This model introduces an additional continuous variable t, replaces the objective function (20a) with $\max t$ and adds the constraint $\bar{a}x - \alpha\left\|R^{1/2}y\right\|_2 \geq t$ where $R^{1/2}$ is the positive semidefinite square root of a given matrix R and α is a given scalar parameter. We refer to this class of instances as the *robust* instances.

We note that only the classical instances can be handled by the lifted polyhedral relaxation considered in [32].

5.2 Implementation and Computational Settings

All algorithms and models were implemented using the JuMP modeling language [2, 39] and solved with CPLEX v12.6 [34] and Gurobi v5.6.3 [30]. The complete code for this implementation is available at `https://github.com/juan-pablo-vielma/extended-MIQCP`.

Our base benchmark algorithms are CPLEX and Gurobi's standard algorithms for solving MICQP. Both solvers implement an NLP-based branch-and-bound algorithm and a standard LP-based branch-and-bound algorithm. Each of these implementations include advanced features such as cutting planes, heuristics, preprocessing and elaborate branching and node selection strategies. We refer to the NLP-based algorithms as CPLEXCP and GurobiCP, and to the LP-based algorithms as CPLEXLP and GurobiLP. All four algorithms can be used by simply giving the model to the appropriate solver and setting a specialized parameter value.

We also implemented a branch-based and a cut-based version of Algorithm 2. The branch-based version corresponds to the LiftedLP algorithm from [45] and its implementation requires access to a branch callback, which is not provided by Gurobi. For this reason we only implemented a CPLEX version similar to the original implementation from [45]. This version was developed using the branch, heuristic and incumbent callbacks for CPLEX provided by the CPLEX.jl library [1]. We refer to this algorithm as LiftedLP. Implementing the cut-based versions of Algorithm 2 only requires access to a *lazy constraint* callback and to a heuristic callback. These callbacks can be accessed for CPLEX and Gurobi through the solver independent callback interface provided by JuMP, which allowed us to implement a version of this algorithm that is not tied to either solver. We implemented the cut-based LiftedLP algorithm for all three lifted polyhedral relaxations considered in Section 4.1. However, as noted in Section 4.2, preliminary experiments showed that the version based on $\widehat{H}^d(\Gamma)$ provides comparable or better performance than the other two versions. For this reason we here only present results for that version, which corresponds to Algorithm 2 using the cut-based refinement described in Algorithm 4. We refer to the implementation of this algorithm using CPLEX and Gurobi as base solvers as CPLEXSepLazy and GurobiSepLazy respectively.

Finally, instead of implementing LP-based algorithms that only use a dynamic lifted polyhedral relaxation (i.e. that does not use L^d_{ε} or $L^d_{s(\varepsilon)}$), we simply solve the three lifted reformulations described in Section 4.1 with CPLEX and Gurobi's LP-based algorithms. We refer to the implementations based on reformulation (15) as CPLEXTowerLP and GurobiTowerLP, to the implementations based on reformulation (16) as CPLEXSepLP and GurobiSepLP and to the implementations based on reformulation (17) as CPLEXTowerSepLP and GurobiTowerSepLP.

5.3 Results

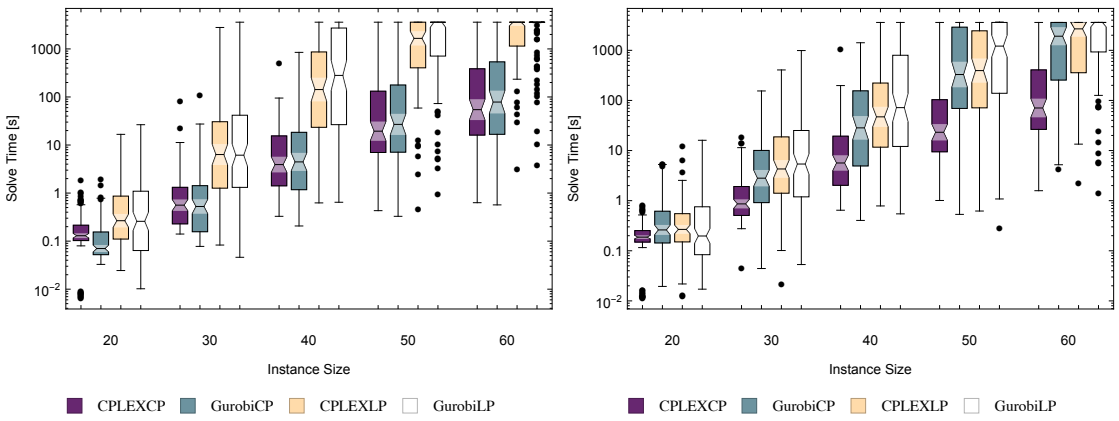
All computational results in this section are from tests on a Intel i7-3770 3.40GHz Linux workstation with 32GB of RAM. All algorithms were limited to a single thread by appropriately setting CPLEX and Gurobi parameters and to a total run time of 3600 seconds. We consider the same portfolio optimization instances from [45], which correspond to the three classes of problems described in Section 5.1 for $K = 10$ and $n \in \{10, 20, 30, 40, 50, 60, 100, 200, 300\}$. For each problem class and choice of n we test 100 randomly generated instances. We refer the reader to [45] for more details on how the instances were generated. All instances and results are available at `https://github.com/juan-pablo-vielma/extended-MIQCP`. Results are presented in two types of charts. The first type are box-and-whisker charts generated by the BoxWhiskerChart function in Mathematica v10 [48]. These charts consider solve times in seconds in a logarithmic scale and show the median solve times, 25% and 75% quantiles of the solve times and minimum and maximum solve times excluding outliers, which are shown as dots. We note that to ensure the graphs are easily legible in black and white print we use the same colors for each chart. Hence, a given algorithm may be assigned different colors in different charts. The second type are performance profiles introduced by Dolan and Moré [15] with solve time as a performance metric. Finally, tables with summary statistics for all methods and instances are included in the Appendix B.1.

5.4 Comparison with Traditional Algorithms and Initial Calibration

In this section we present some initial results that evaluate the difficulty of the considered instances, compare the lifted algorithms to standard algorithms and compare the dynamic lifted relaxations. Because

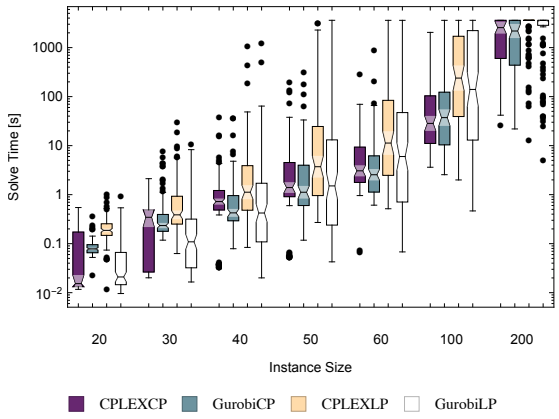
the Classical and Shortfall instances are extremely difficult for many algorithms tested in this section we only consider $n \in \{10, 20, 30, 40, 50, 60\}$ for these instances. Similarly we exclude $n = 300$ for the Robust instances.

The first set of results are presented through box-and-whisker charts in Figure 1. These charts compare the performance of CPLEX and Gurobi's NLP and LP-based algorithms (CPLEXCP, GurobiCP, CPLEXLP and GurobiLP). The results in [45] showed that NLP-based algorithms had a significant advantage over



(a) Classical.

(b) Shortfall.

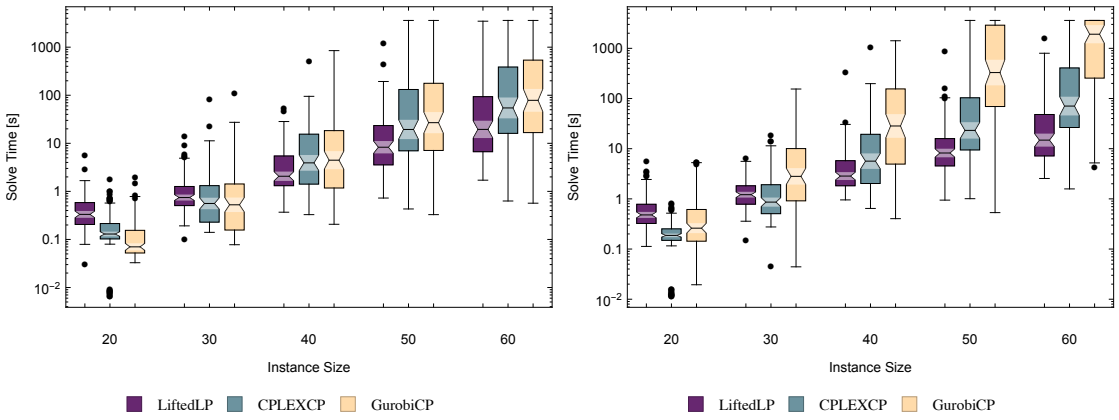


(c) Robust.

Fig. 1 Solution times for standard LP-based and NLP-based algorithms [s].

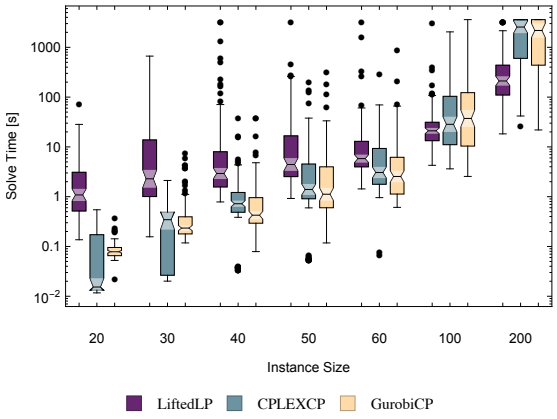
traditional LP-based algorithms for the portfolio optimization instances. Figure 1 shows that, for sufficiently large values of n, this advantage still holds for current versions of CPLEX and Gurobi.

The second set of results are also presented through box-and-whisker charts in Figure 2. These charts compare the performance of the branch-based LiftedLP algorithm (LiftedLP) and the two NLP-based algorithms (CPLEXCP and GurobiCP). Figure 2 confirms that, for sufficiently large values of n, LiftedLP



(a) Classical.

(b) Shortfall.



(c) Robust.

Fig. 2 Solution times for branch-based LiftedLP and NLP-based algorithms [s].

still provides an advantage over CPLEXCP and GurobiCP for the considered instances.

The final set of results in this section compares the performance of the three dynamic lifted polyhedral relaxations in their nonlinear reformulation versions described in Section 4.1 (CPLEXSepLP, GurobiSepLP, CPLEXTowerLP, GurobiTowerLP, CPLEXTowerSepLP, GurobiTowerSepLP). This time the results are presented through a performance profile in Figure 3. Figure 3 shows that for a fixed solver (CPLEX or Gurobi), the separable relaxation outperforms the other relaxation (i.e. CPLEXSepLP outperforms CPLEXTowerLP and CPLEXTowerSepLP, and GurobiSepLP outperforms GurobiTowerLP and GurobiTowerSepLP). Furthermore, CPLEXSepLP and GurobiSepLP have comparable or better performance than all other combinations of solver and relaxation. These results align with our preliminary experiments which showed that the separable relaxation performed best among the cut-based LiftedLP algorithms considered in Section 4.2. Because the separable relaxation is additionally the simplest, from now on we concentrate on this relaxation among the three dynamic ones. Box-and-whisker charts with more details concerning this experiment are presented in Figure 6 in Appendix B.3. For instance, Figure 6 shows that the pure tower

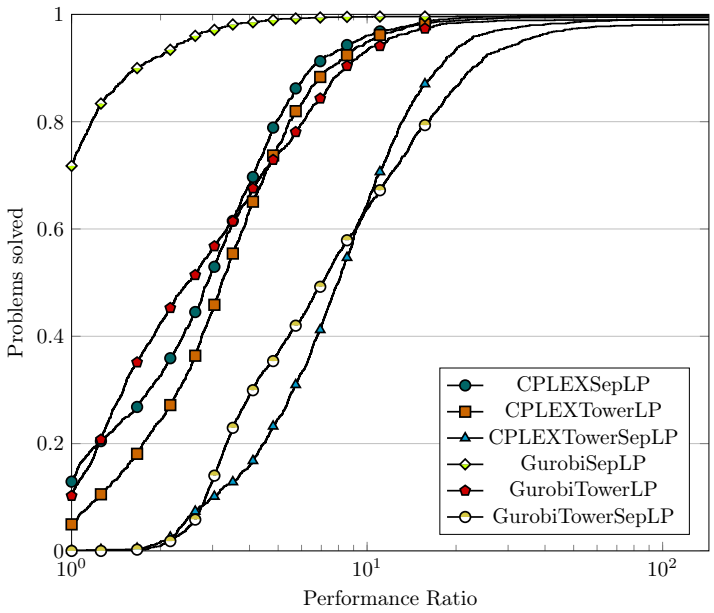
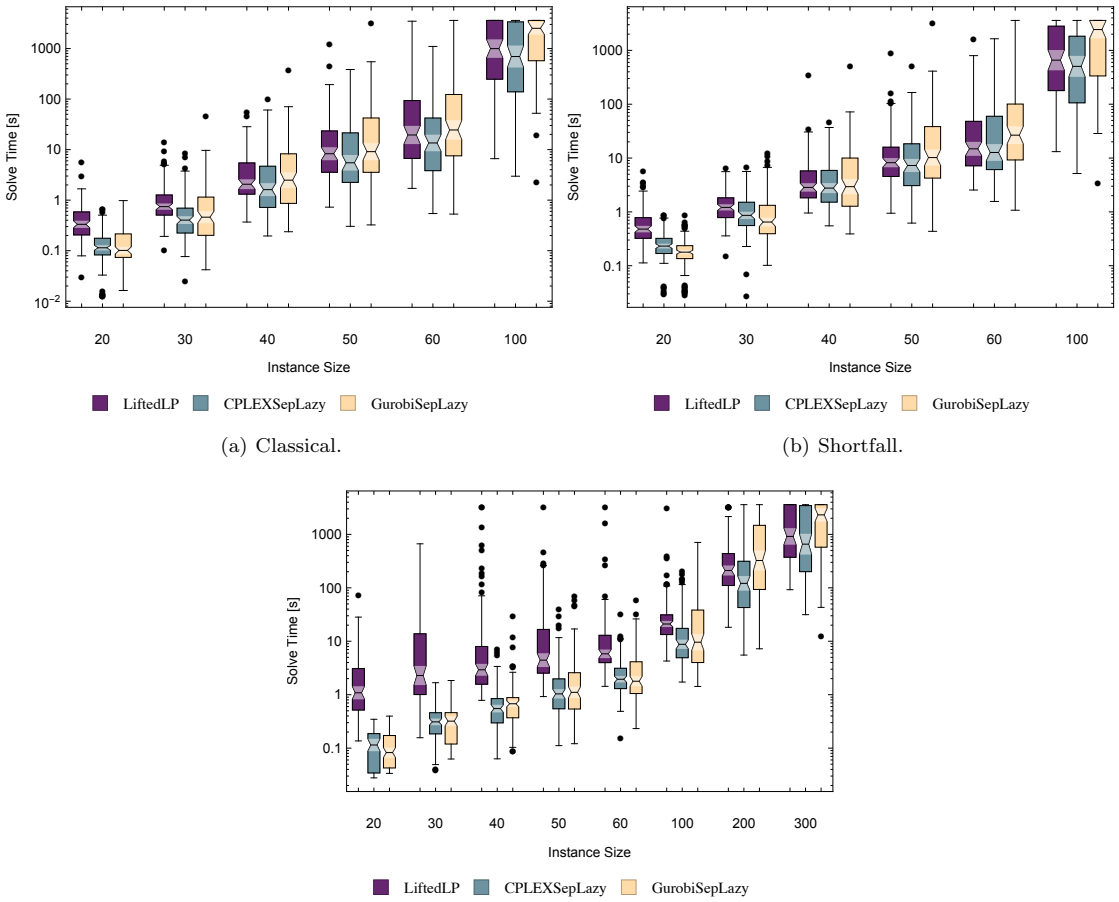


Fig. 3 Performance profiles of solution times for dynamic lifted polyhedral relaxations solved by standard LP-based algorithms.

relaxation (CPLEXTowerLP and GurobiTowerLP) can provide an advantage over the separable relaxations (CPLEXSepLP and GurobiSepLP) for the smallest instances, but the combined tower-separable relaxation (CPLEXTowerSepLP and GurobiTowerSepLP) is consistently outperformed by the other two. Additional details are also included in summary tables in Appendices B.1 and B.2. The tables in Appendix B.1 present summary statistics for solve times by class of instance and size, which can be used for even more detailed comparisons. For example, the tables confirm the potential advantage of the tower relaxation for the smallest instances by showing that GurobiTowerLP is the fastest option in 54 out of the 100 shortfall instances for $n = 20$. The tables in Appendix B.2 present summary statistics for the feasibility quality of the solutions obtained. These statistics show that using the separable reformulation resulted in a significant reduction on the feasibility quality of the solutions obtained, particularly when using Gurobi. However, this reduction in quality is not surprising as errors in the 3-dimensional rotated conic constraints (16a) can easily add up to a larger error in the original conic constraint. Fortunately, this can be easily resolved by increasing the precision for constraints (16a) or by simply correcting the final or intermediate incumbent solutions using the original conic constraint (i.e. by solving the original conic quadratic relaxation with the integer variables fixed appropriately).

5.5 Comparison between Lifted LP-based Algorithms

In this section we compare the performance of the lifted algorithms. Classical and Shortfall instances are still extremely difficult for many of these algorithms so we exclude results for $n \in \{200, 300\}$ for these instances. We begin by comparing the branch-based (LiftedLP) and cut-based LiftedLP algorithms (CPLEXSepLazy and GurobiSepLazy) in Figure 4. The results show that all three methods have comparable overall performances. The cut-based algorithms do sometimes provide a computational advantage, particularly for



(a) Classical.

(b) Shortfall.

(c) Robust.

Fig. 4 Solution times branch-based and cut-based LiftedLP algorithms [s].

the smaller instances. Furthermore, cut-based algorithms also have the practical advantage of being easily implemented for both CPLEX and Gurobi through JuMP.

Our final set of experiments compare the branch-based and cut-based LiftedLP algorithms (LiftedLP, CPLEXSepLazy and GurobiSepLazy) the separable dynamic lifted polyhedral relaxation in its nonlinear reformulation version (CPLEXSepLP and GurobiSepLP). The results are presented through a performance profile in Figure 5, which also includes the four traditional algorithms (CPLEXCP, GurobiCP, CPLEXLP and GurobiLP) as a reference. Figure 5 confirms that the traditional LP-based algorithms have the worst performance and are rather consistently outperformed by the NLP-based algorithms. In addition, the branch-based LiftedLP algorithm outperforms the traditional algorithms with the exception of CPLEX's non-linear based algorithm (CPLEXCP). Furthermore, the additional advantage provided by the cut-based LiftedLP algorithms allows them to consistently outperform all traditional algorithms. However, the best performance is achieved by the separable dynamic lifted polyhedral relaxations in their nonlinear reformulation versions (CPLEXSepLP and GurobiSepLP). Again, box-and-whisker charts and summary statistics

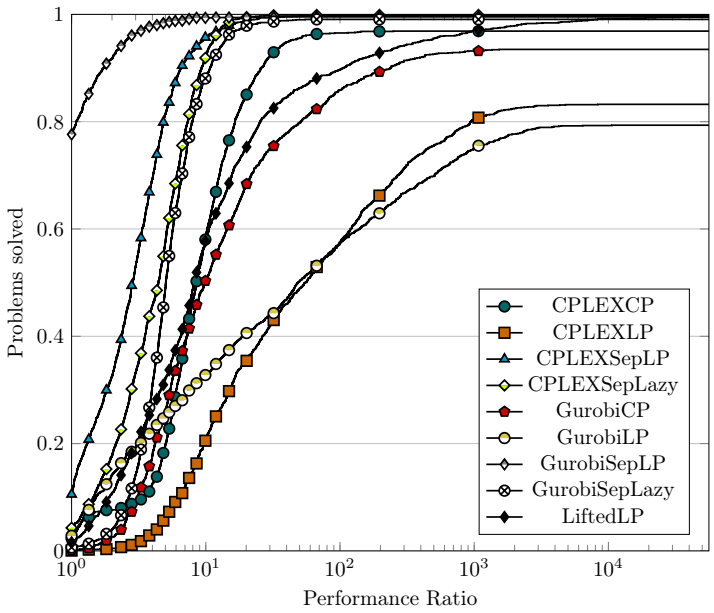


Fig. 5 Solution times for static and dynamic lifted LP-based, standard LP-based and standard NLP-based algorithms [s].

tables with more details concerning solve times and solution quality are included in Appendix B. One notable insight from these tables is that the cut-based LiftedLP algorithms can be competitive for the hardest instances. For instance, CPLEXSepLazy was the fastest option in 37 out of the 100 shortfall instances for $n = 100$.

The JuMP implementation of the cut-based LiftedLP algorithms results on an extremely flexible framework that can significantly outperform traditional algorithms. On the other hand, the separable reformulation provides a consistently better performance without the need for callbacks or any additional programming beyond a simple transformation of the conic constraints. A possible explanation of this performance may be that callbacks interfere with (or even result in the deactivation of) advanced features of the MICQP solvers (e.g. CPLEX turns off the dynamic search feature when control callbacks such as heuristic, lazy constraint and branch callbacks are used [33]). Hence, it is possible that an internal implementation of the LiftedLP algorithms may provide an advantage in some classes of problems. Furthermore, from a purely theoretical standpoint (i.e. disregarding the mentioned algorithmic details and implementation issues), the only difference between the cut-based LiftedLP algorithms and the separable reformulation algorithms is the inclusion of the Ben-Tal and Nemirovski lifted polyhedral relaxation L_ε^d by the first class of algorithms. This suggests that including L_ε^d as an initial polyhedral relaxation can sometimes be useful for large instances.

6 Possible Extensions and Open Questions

The computational results show that the separable reformulations provide a clear advantage over the original LiftedLP algorithm and standard LP-based and NLP-based algorithms for all three variants of the portfolio optimization problem. However, a further strengthening of the reformulation may be possible for the classical instances. In addition, some theoretical questions remain about the approximation quality provided by the reformulation.

6.1 Formulation Strengthening for Portfolio Optimization Through Perspective Reformulations

The relation between the separable reformulation and perspective formulations of unions of convex sets described in Section 3.1 can also be used to strengthen the separable reformulation for the classical portfolio optimization instances (20) for $Q^{1/2} = I$. If we rewrite (20b) using separable reformulation $\widehat{\mathbf{H}}^d$ defined in (12) we obtain

$$x_0 \leq \sigma, \tag{21a}$$
$$x_j^2 \leq w_j x_0, \qquad \forall j \in \{1,\dots,n\}, \tag{21b}$$
$$\sum_{j=1}^{d} w_j \leq x_0, \tag{21c}$$
$$\sum_{j=1}^{n} x_j = 1, \tag{21d}$$
$$x_j \leq z_j, \qquad \forall j \in \{1,\dots,n\}, \tag{21e}$$
$$\sum_{j=1}^{n} z_j \leq K, \tag{21f}$$
$$z \in \{0,1\}^n, \tag{21g}$$
$$x \in \mathbb{R}^n_+. \tag{21h}$$

However, using known results (e.g. [42] and Section 3.4 of [28]), we may replace (21a)–(21b) with

$$x_0 \leq \sigma^2, \tag{22a}$$
$$x_j^2 \leq w_j z_j, \quad \forall j \in \{1,\dots,n\} \tag{22b}$$

to obtain a stronger formulation. Having $Q^{1/2} = I$ is essential for this improvement, but extending it to general $Q^{1/2}$ may be possible by using other known techniques [16, 20, 35].

6.2 Approximation Quality of Dynamic Lifted Polyhedral Relaxations

It is well known that constructing a non-lifted (i.e. $m_2 = 0$ in Definition 1) polyhedral relaxation of $\mathbf{L}^d$ with approximation quality ε in (5), requires at least $\exp\left(\frac{d}{2(1+\varepsilon)^2}\right)$ linear inequalities [4]. Hence, as discussed just before Proposition 3, the approximation quality of $\widehat{L}^d_{s(\varepsilon)}$ and L^d_ε given by Proposition 2 and Corollary 3 seems strongly dependent on the fact that the projection of $\widehat{N}^2_s$ onto the variables y has an exponential in s number of inequalities. This suggests that neither the tower of variables nor the separable polyhedral relaxations can achieve the approximation efficiency of $\widehat{L}^d_{s(\varepsilon)}/L^d_\varepsilon$ with regard to number of linear inequalities. However it would still be interesting to understand what level of efficiency these approximations can achieve. We formalize this in the following open questions.

Question 1 (Smallest tower of variables polyhedral relaxation) Let

$$\Omega := \left\{\Omega_{i,k} : i \in \{1,\dots,\lfloor t_k/2 \rfloor\}, \quad k \in \{0,\dots,K-1\}\right\}$$

be such that $\Omega_{i,k} \subseteq \mathbf{S}^1$ and $m(\Omega) := \sum_{k=0}^{K-1} \sum_{i=1}^{\lfloor t_k/2 \rfloor} |\Omega_{i,k}|$.

For a given $\varepsilon > 0$, what is the smallest $m(\Omega)$ such that $\widehat{T}^d(\Omega)$ is a polyhedral relaxation of $\mathbf{L}^d$ with approximation quality ε?

Question 2 (Smallest separable polyhedral relaxation) Let $\Gamma := \{\Gamma_j : j \in \{1,\ldots,n\}\}$ and $m(\Gamma) := \sum_{j=1}^{n} |\Gamma_j|$.

For a given $\varepsilon > 0$, what is the smallest $m(\Gamma)$ such that $\widehat{H}^d(\Gamma)$ is a polyhedral relaxation of $\mathbf{L}^d$ with approximation quality ε?

Finally, [5] shows that $\widehat{L}^d_{s(\varepsilon)}/L^d_\varepsilon$ are essentially the smallest possible (static or dynamic) polyhedral relaxations of $\mathbf{L}^d$ with an approximation quality ε. Then some natural follow-up questions are: what is the smallest possible size of a dynamic polyhedral relaxation of $\mathbf{L}^d$, and how close are the tower of variables and separable approximations to this lower bound.

Acknowledgements We thank the review team for their thoughtful and constructive comments, which improved the exposition of the paper. Support for Juan Pablo Vielma was provided by the National Science Foundation under grant CMMI-1351619, support for Miles Lubin was provided by the DOE Computational Science Graduate Fellowship under grant DE-FG02-97ER25308 and support for Joey Huchette was provided by the National Science Foundation Graduate Research Fellowship under grant 1122374.

References

1. Julia interface for the CPLEX optimization software. https://github.com/JuliaOpt/CPLEX.jl
2. JuMP Modeling language for Mathematical Programming. https://github.com/JuliaOpt/JuMP.jl
3. Abhishek, K., Leyffer, S., Linderoth, J.: FilMINT: An outer approximation-based solver for convex mixed-integer nonlinear programs. INFORMS Journal on computing **22**(4), 555–567 (2010)
4. Ball, K.M.: An elementary introduction to modern convex geometry. In: S. Levy (ed.) Flavors of Geometry, *Mathematical Sciences Research Institute Publications*, vol. 31, pp. 1–58. Cambridge University Press, Cambridge (1997)
5. Ben-Tal, A., Nemirovski, A.: Lectures on modern convex optimization: analysis, algorithms, and engineering applications, vol. 2. Siam (2001)
6. Ben-Tal, A., Nemirovski, A.: On polyhedral approximations of the second-order cone. Math. Oper. Res. **26**(2), 193–205 (2001)
7. Bertsimas, D., Shioda, R.: Algorithm for cardinality-constrained quadratic optimization. Computational Optimization and Applications **43**(1), 1–22 (2009)
8. Bienstock, D.: Computational study of a family of mixed-integer quadratic programming problems. Math. Program. **74**(2), 121–140 (1996)
9. Bonami, P., Biegler, L.T., Conn, A.R., Cornuéjols, G., Grossmann, I.E., Laird, C.D., Lee, J., Lodi, A., Margot, F., Sawaya, N., et al.: An algorithmic framework for convex mixed integer nonlinear programs. Discrete Optimization **5**(2), 186–204 (2008)
10. Bonami, P., Kilinç, M., Linderoth, J.: Algorithms and software for convex mixed integer nonlinear programs. In: Mixed Integer Nonlinear Programming, pp. 1–39. Springer (2012)
11. Borchers, B., Mitchell, J.E.: An improved branch and bound algorithm for mixed integer nonlinear programs. Comput. Oper. Res. **21**(4), 359–367 (1994)
12. Ceria, S., Soares, J.: Convex programming for disjunctive convex optimization. Mathematical Programming **86**, 595–614 (1999)
13. Ceria, S., Stubbs, R.A.: Incorporating estimation errors into portfolio selection: Robust portfolio construction. J. Asset Manage. **7**(2), 109–127 (2006)
14. Chang, T.J., Meade, N., Beasley, J.E., Sharaiha, Y.M.: Heuristics for cardinality constrained portfolio optimisation. Comput. and Oper. Res. **27**, 1271–1302 (2000)
15. Dolan, E.D., Moré, J.J.: Benchmarking optimization software with performance profiles. Math. Program. **91**(2), 201–213 (2002)
16. Dong, H., Linderoth, J.: On valid inequalities for quadratic programming with continuous variables and binary indicators. In: M.X. Goemans, J.R. Correa (eds.) Integer Programming and Combinatorial Optimization - 16th International Conference, IPCO 2013, Valparaíso, Chile, March 18-20, 2013. Proceedings, *Lecture Notes in Computer Science*, vol. 7801, pp. 169–180. Springer (2013)
17. Duran, M.A., Grossmann, I.E.: An outer-approximation algorithm for a class of mixed-integer nonlinear programs. Math. Program. **36**(3), 307–339 (1986)
18. Fletcher, R., Leyffer, S.: Solving mixed integer nonlinear programs by outer approximation. Math. Program. **66**(3), 327–349 (1994)

19. Frangioni, A., Gentile, C.: Perspective cuts for a class of convex 0–1 mixed integer programs. Mathematical Programming **106**, 225–236 (2006)
20. Frangioni, A., Gentile, C.: SDP diagonalizations and perspective cuts for a class of nonseparable MIQP. Operations Research Letters **35**, 181–185 (2007)
21. Frangioni, A., Gentile, C.: A computational comparison of reformulations of the perspective relaxation: Socp vs. cutting planes. Operations Research Letters **37**, 206–210 (2009)
22. Frangioni, A., Gentile, C., Grande, E., Pacifici, A.: Projected perspective reformulations with applications in design problems. Operations research **59**, 1225–1232 (2011)
23. Geoffrion, A.: Generalized benders decomposition. J. of Optim. Theory and Appl. **10**(4), 237–260 (1972)
24. Glineur, F.: Computational experiments with a linear approximation of second order cone optimization. Image Technical Report 0001, Service de Mathématique et de Recherche Opérationnelle, Faculté Polytechnique de Mons, Mons, Belgium (2000)
25. Grossmann, I.E., Lee, S.: Generalized convex disjunctive programming: Nonlinear convex hull relaxation. Computational Optimization and Applications **26**, 83–100 (2003)
26. Günlük, O., Linderoth, J.: Perspective relaxation of mixed integer nonlinear programs with indicator variables. In: A. Lodi, A. Panconesi, G. Rinaldi (eds.) Integer Programming and Combinatorial Optimization, 13th International Conference, IPCO 2008, Bertinoro, Italy, May 26-28, 2008, Proceedings, *Lecture Notes in Computer Science*, vol. 5035, pp. 1–16. Springer (2008)
27. Günlük, O., Linderoth, J.: Perspective reformulations of mixed integer nonlinear programs with indicator variables. Math. Program. **124**, 183–205 (2010)
28. Günlük, O., Linderoth, J.: Perspective reformulation and applications. In: J. Lee, S. Leyffer (eds.) Mixed Integer Nonlinear Programming, *The IMA Volumes in Mathematics and its Applications*, vol. 154, pp. 61–89. Springer New York (2012)
29. Gupta, O.K., Ravindran, A.: Branch and bound experiments in convex nonlinear integer programming. Manage. Sci. **31**(12), 1533–1546 (1985)
30. Gurobi Optimization: The Gurobi Optimizer. http://www.gurobi.com
31. Hijazi, H., Bonami, P., Cornuéjols, G., Ouorou, A.: Mixed-integer nonlinear programs featuring on/off constraints. Computational Optimization and Applications **52**, 537–558 (2012)
32. Hijazi, H., Bonami, P., Ouorou, A.: An outer-inner approximation for separable mixed-integer nonlinear programs. INFORMS Journal on Computing **26**(1), 31–44 (2013)
33. IBM Corp.: User's Manual for CPLEX. IBM Corp (2014)
34. IBM ILOG: CPLEX High-performance mathematical programming engine. http://www.ibm.com/software/integration/optimization/cplex/
35. Jeon, H., Linderoth, J., Miller, A.: Quadratic cone cutting surfaces for quadratic programs with on-off constraints. Optimization Online (2015). URL http://www.optimization-online.org/DB_HTML/2015/01/4746.html
36. Leyffer, S.: Integrating SQP and branch-and-bound for mixed integer nonlinear programming. Comput. Optim. and Appl. **18**, 295–309 (2001)
37. Lobo, M.S., Fazel, M., Boyd, S.: Portfolio optimization with linear and fixed transaction costs. Annals of Operations Research **152**(1), 341–365 (2007)
38. Lobo, M.S., Vandenberghe, L., Boyd, S.: Application of second-order cone programming. Linear Algebra Appl. **284**, 193–228 (1998)
39. Lubin, M., Dunning, I.: Computing in operations research using julia. arXiv preprint arXiv:1312.1431 (2013)
40. Maringer, D., Kellerer, H.: Optimization of cardinality constrained portfolios with a hybrid local search algorithm. OR Spectrum **25**(4), 481–495 (2003)
41. Quesada, I., Grossmann, I.: An LP/NLP based branch and bound algorithm for convex MINLP optimization problems. Comput. and Chem. Engrg. **16**, 937–947 (1992)
42. Stubbs, R.A.: Branch-and-cut methods for mixed 0-1 convex programming. Ph.D. thesis (1996)
43. Stubbs, R.A., Mehrotra, S.: A branch-and-cut method for 0-1 mixed convex programming. Math. Program. **86**(3), 515–532 (1999)
44. Tawarmalani, M., Sahinidis, N.V.: A polyhedral branch-and-cut approach to global optimization. Mathematical Programming **103**(2), 225–249 (2005)
45. Vielma, J.P., Ahmed, S., Nemhauser, G.: A lifted linear programming branch-and-bound algorithm for mixed integer conic quadratic programs. INFORMS Journal on Computing **20**, 438–450 (2008)
46. Westerlund, T., Pettersson, F.: An extended cutting plane method for solving convex minlp problems. Comput. and Chem. Engrg. **19**, S131–S136 (1995)

47. Westerlund, T., Pettersson, F., Grossmann, I.: Optimization of pump configurations as a minlp problem. Comput. and Chem. Engrg. **18**(9), 845–858 (1994)
48. Wolfram Research Inc.: Mathematica, Version 10.0. Wolfram Research, Inc., Champaign, Illinois (2014)

A Proof of Proposition 4

To prove Proposition 4 we begin with the following lemma, which gives a characterization of the homogenization of a convex set described by nonlinear constraints.

Lemma 1 *Let $f : \mathbb{R}^d \to \mathbb{R}$ be a closed convex function such that $\lim_{\|x\|_2 \to \infty} \frac{f(x)}{\|x\|_2} = \infty$ so that the closure of the perspective function of f is given by*

$$\left(cl\tilde{f}\right)(t,x) = \begin{cases} tf(x/t) & t > 0 \\ 0 & x = 0 \text{ and } t = 0 \\ \infty & o.w. \end{cases}.$$

If $\mathbf{C} := \left\{y \in \mathbb{R}^d \,:\, f(y) \leq 1\right\}$, then

$$\operatorname{cone}(\{1\} \times \mathbf{C}) = \left\{(y_0, y) \in \mathbb{R}^{d+1} \,:\, \left(cl\tilde{f}\right)(y_0, y) \leq y_0\right\}. \tag{23}$$

Proof Let $\mathbf{D} := \left\{(y_0, y) \in \mathbb{R}^{d+1} \,:\, \left(cl\tilde{f}\right)(y_0, y) \leq y_0\right\}$. We have $\{1\} \times \mathbf{C} \subseteq \mathbf{D}$ and $\mathbf{D}$ is a convex cone because $\left(cl\tilde{f}\right)(y_0, y)$ is a homogeneous function, so $\operatorname{cone}(\{1\} \times \mathbf{C}) \subseteq \mathbf{D}$.

For the reverse inclusion, let $(y_0, y) \in \mathbf{D}$. If $y_0 = 0$ then $y = 0$ and hence $(y_0, y) \in \operatorname{cone}(\{1\} \times \mathbf{C})$. If $y_0 > 0$ then $(y_0, y)/y_0 \in \{1\} \times \mathbf{C}$ and hence $(y_0, y) \in \operatorname{cone}(\{1\} \times \mathbf{C})$.

The final ingredient for the proof of Proposition 4 is the following lemma that shows how to translate the polyhedral approximation for univariate functions to their homogenization.

Lemma 2 *Let $f : \mathbb{R} \to \mathbb{R}$ be a closed convex differentiable function such that $\lim_{x \to \infty} \frac{f(x)}{|x|} = \infty$. Then*

$$\begin{aligned} \operatorname{epi}\left(cl\tilde{f}\right) :&= \left\{(y_0, y, w) \in \mathbb{R}^3 \,:\, \left(cl\tilde{f}\right)(y_0, y) \leq w\right\} \\ &= \left\{(y_0, y, w) \in \mathbb{R}^3 \,:\, \left(f(\gamma) - \gamma f'(\gamma)\right) y_0 + f'(\gamma)\, y \leq w \quad \forall \gamma \in \mathbb{R}\right\}. \end{aligned}$$

Furthermore, $(y_0, y, w) \in \operatorname{epi}\left(cl\tilde{f}\right)$ if and only if either $y_0 = y = 0 \leq w$ or if $y_0 > 0$ and

$$\left(f(\gamma(y_0, y)) - \gamma(y_0, y) f'(\gamma(y_0, y))\right) y_0 + f'(\gamma(y_0, y))\, y \leq w \tag{24}$$

for $\gamma(y_0, y)$ defined in Corollary 2.

Proof First note that $\mathbf{D} := \left\{(y_0, y, w) \in \mathbb{R}^3 \,:\, \left(f(\gamma) - \gamma f'(\gamma)\right) y_0 + f'(\gamma)\, y \leq w \quad \forall \gamma \in \mathbb{R}\right\}$ is a closed convex cone, $\operatorname{epi}\left(cl\tilde{f}\right) = \overline{\operatorname{cone}}(\{1\} \times \operatorname{epi}(f))$ and

$$\operatorname{epi}(f) = \left\{(y, w) \in \mathbb{R}^2 \,:\, \left(f(\gamma) - \gamma f'(\gamma)\right) + f'(\gamma)\, y \leq w \quad \forall \gamma \in \mathbb{R}\right\}.$$

Then $\{1\} \times \operatorname{epi}(f) \subseteq \mathbf{D}$ implies $\operatorname{epi}\left(cl\tilde{f}\right) \subseteq \mathbf{D}$.

For the reverse inclusion, let $(y_0, y, w) \in \mathbf{D}$. We first show that $y_0 \geq 0$, by assuming $y_0 < 0$ and reaching a contradiction. For this we consider cases $y \neq 0$ and $y = 0$ separately. For both cases note that $\lim_{x \to \infty} \frac{f(x)}{|x|} = \infty$ implies that $\lim_{x \to +\infty} f'(x) = +\infty$ and $\lim_{x \to -\infty} f'(x) = -\infty$.

For case $y \neq 0$, note that if $y_0 < 0$, then $y_0 f(0) \leq y_0 \left(f(\gamma) - \gamma f'(\gamma)\right)$ for all $\gamma \in \mathbb{R}$ by convexity of f. Hence, if $(y_0, y, w) \in \mathbf{D}$ and $y_0 < 0$ then

$$y_0 f(0) + f'(\gamma)\, y \leq w \tag{25}$$

for all $\gamma \in \mathbb{R}$. Taking limit for $\gamma \to \infty$ when $y > 0$ and for $\gamma \to -\infty$ when $y < 0$ in (25) we arrive at a contradiction with $w < \infty$.

For case $y = 0$, note that by convexity of f and the mean value theorem we have that $f(\gamma) - \gamma f'(\gamma) \leq f(\gamma_0) - f'(\gamma)\gamma_0$ for any $0 < \gamma_0 < \gamma$. Then, $\lim_{\gamma\to\infty} f(\gamma) - \gamma f'(\gamma) = -\infty$. Now, if $(y_0, y, w) \in \mathbf{D}$ and $y = 0$ then

$$\left(f(\gamma) - \gamma f'(\gamma)\right) y_0 \leq w \tag{26}$$

for all $\gamma \in \mathbb{R}$. Then, if $y_0 < 0$, by taking limit for $\gamma \to \infty$ in (26) we again arrive at a contradiction with $w < \infty$.

We now show that any $(y_0, y, w) \in \mathbf{D}$ with $y_0 \geq 0$ also belongs to $\text{epi}\left(cl\tilde{f}\right)$. We divide the proof into cases $y_0 = 0$ and $y_0 > 0$.

For case, $y_0 = 0$, note that $(y_0, y, w) \in \mathbf{D}$ implies $f'(\gamma)\, y \leq w$ for all $\gamma \in \mathbb{R}$. Taking limit for $\gamma \to \infty$ when $y > 0$ and for $\gamma \to -\infty$ when $y < 0$, we conclude that $y = 0$ and $w \geq 0$. Hence, $(y_0, y, w) \in \text{epi}\left(cl\tilde{f}\right)$.

For case, $y_0 > 0$ we can check that $(y, w)/y_0 \in \text{epi}(f)$ and then $(y_0, y, w)/y_0 \in \{1\} \times \text{epi}(f)$. Hence, $(y_0, y, w) \in \text{cone}(\{1\} \times \text{epi}(f)) \subseteq \text{epi}\left(cl\tilde{f}\right)$.

For the final statement, it suffices to prove that $(y_0, y, w) \in \text{epi}\left(cl\tilde{f}\right)$ if $y_0 > 0$ and (y_0, y, w) satisfies (24). For this note that, if the last two conditions hold, then $(y, w)/y_0 \in \text{epi}(f)$, which we have already shown implies $(y_0, y, w) \in \text{epi}\left(cl\tilde{f}\right)$.

Combining these lemmas we obtain the following straightforward proof of Proposition 4.

Proof (of Proposition 4) Let $f(y) = \sum_{j=1}^{d} f_j(y_j)$. Then (11) follows by noting that by Lemma 1 we have $\text{cone}(\{1\} \times \mathbf{C}) = \left\{(y_0, y) \in \mathbb{R}^{d+1} : \left(cl\tilde{f}\right)(y_0, y) \leq y_0\right\}$ and by the definition of perspective function we have $\left(cl\tilde{f}\right)(y_0, y) = \sum_{j=1}^{d} \left(cl\tilde{f}_j\right)(y_0, y_j)$. All other statements are directly from Lemma 2 and from Proposition 3 by fixing $y_0 = 1$.

B Additional Graphs and Tables

B.1 Summary Statistics for Solve Times

Tables 1–8 show some summary statistics for the solve times for the different algorithms and instances. These statistics include the minimum, average and maximum solve times, together with their standard deviation. The tables also include the number of each solver was the fastest (wins) and the number of times a solver has a solution time that was within 1% and 10% of the fastest solver (1% and 10% win).

Algorithm	min	avg	max	std	wins	1% win	10% win
CPLEXCP	0.01	**0.23**	2.06	0.30	0	0	2
GurobiCP	0.03	**0.17**	2.22	0.31	0	0	0
CPLEXLP	0.02	**1.23**	16.67	2.69	0	0	0
GurobiLP	0.01	**1.63**	26.48	4.04	0	0	0
LiftedLP	0.03	**0.53**	6.44	0.74	0	0	0
CPLEXSepLp	0.01	**0.09**	0.53	0.09	4	4	6
CPLEXTowerLp	0.01	**0.11**	0.71	0.10	0	0	3
CPLEXTowerSepLp	0.01	**0.23**	1.49	0.23	0	0	0
GurobiSepLp	0.01	**0.03**	0.18	0.02	58	58	76
GurobiTowerLp	0.01	**0.03**	0.21	0.03	38	38	58
GurobiTowerSepLp	0.01	**0.15**	1.06	0.22	0	0	0
CPLEXSepLazy	0.01	**0.16**	0.74	0.14	0	0	0
GurobiSepLazy	0.02	**0.17**	0.98	0.18	0	0	0

(a) Classical.

Algorithm	min	avg	max	std	wins	1% win	10% win
CPLEXCP	0.01	**0.23**	0.91	0.18	1	1	1
GurobiCP	0.02	**0.76**	6.12	1.32	2	2	3
CPLEXLP	0.01	**0.69**	13.57	1.59	0	0	1
GurobiLP	0.02	**0.97**	16.03	2.18	0	0	0
LiftedLP	0.11	**0.82**	6.27	0.95	0	0	0
CPLEXSepLp	0.01	**0.15**	0.81	0.13	2	2	3
CPLEXTowerLp	0.01	**0.12**	0.42	0.09	5	5	6
CPLEXTowerSepLp	0.03	**0.29**	0.80	0.19	1	1	1
GurobiSepLp	0.01	**0.04**	0.12	0.02	35	37	53
GurobiTowerLp	0.01	**0.04**	0.14	0.02	54	55	67
GurobiTowerSepLp	0.03	**0.17**	1.15	0.19	0	0	0
CPLEXSepLazy	0.03	**0.28**	0.98	0.19	0	0	0
GurobiSepLazy	0.03	**0.21**	0.97	0.15	0	0	0

(b) Shortfall.

Algorithm	min	avg	max	std	wins	1% win	10% win
CPLEXCP	0.01	**0.10**	0.55	0.10	19	20	33
GurobiCP	0.03	**0.09**	0.41	0.05	0	0	0
CPLEXLP	0.01	**0.23**	1.14	0.17	0	0	0
GurobiLP	0.01	**0.07**	1.05	0.14	17	17	30
LiftedLP	0.14	**4.03**	81.41	9.25	0	0	0
CPLEXSepLp	0.01	**0.08**	0.34	0.08	7	9	30
CPLEXTowerLp	0.01	**0.07**	0.32	0.07	16	18	33
CPLEXTowerSepLp	0.03	**0.15**	0.61	0.14	0	0	0
GurobiSepLp	0.01	**0.03**	0.07	0.01	39	40	48
GurobiTowerLp	0.01	**0.04**	0.20	0.03	2	2	5
GurobiTowerSepLp	0.03	**0.08**	0.36	0.06	0	0	0
CPLEXSepLazy	0.03	**0.12**	0.35	0.09	0	0	0
GurobiSepLazy	0.03	**0.12**	0.40	0.09	0	0	0

(c) Robust.

Table 1 Summary statistics of Solve Times for $n = 20$ [s].

Algorithm	min	avg	max	std	wins	1% win	10% win
CPLEXCP	0.14	**2.44**	91.52	9.54	0	0	0
GurobiCP	0.08	**2.95**	124.47	12.81	0	0	0
CPLEXLP	0.08	**101.30**	2795.96	391.38	0	0	0
GurobiLP	0.05	**174.52**	3600.01	605.37	0	0	0
LiftedLP	0.11	**1.35**	15.74	2.10	0	0	0
CPLEXSepLp	0.01	**0.49**	4.45	0.72	1	2	2
CPLEXTowerLp	0.01	**0.68**	8.36	1.15	0	0	1
CPLEXTowerSepLp	0.02	**1.49**	21.18	2.65	0	0	0
GurobiSepLp	0.01	**0.21**	3.94	0.46	65	66	72
GurobiTowerLp	0.02	**0.27**	5.83	0.69	34	37	46
GurobiTowerSepLp	0.04	**2.65**	96.69	10.10	0	0	0
CPLEXSepLazy	0.03	**0.80**	9.53	1.38	0	0	0
GurobiSepLazy	0.04	**1.59**	51.04	5.30	0	0	0

(a) Classical.

Algorithm	min	avg	max	std	wins	1% win	10% win
CPLEXCP	0.05	**2.04**	20.97	3.31	0	0	0
GurobiCP	0.04	**11.88**	154.60	22.32	0	0	0
CPLEXLP	0.02	**22.79**	406.01	62.02	0	0	0
GurobiLP	0.05	**42.13**	989.04	145.49	0	0	0
LiftedLP	0.17	**1.55**	7.20	1.23	0	0	0
CPLEXSepLp	0.03	**0.72**	8.32	0.90	4	4	4
CPLEXTowerLp	0.02	**0.75**	7.74	0.97	2	2	2
CPLEXTowerSepLp	0.05	**1.90**	21.70	2.69	0	0	0
GurobiSepLp	0.02	**0.23**	2.87	0.43	94	94	95
GurobiTowerLp	0.03	**0.83**	8.87	1.53	0	0	1
GurobiTowerSepLp	0.05	**2.09**	25.97	4.25	0	0	0
CPLEXSepLazy	0.03	**1.21**	7.64	1.11	0	0	0
GurobiSepLazy	0.10	**1.39**	13.96	2.24	0	0	0

(b) Shortfall.

Algorithm	min	avg	max	std	wins	1% win	10% win
CPLEXCP	0.02	**0.41**	2.12	0.39	4	4	8
GurobiCP	0.12	**0.69**	8.47	1.35	0	0	0
CPLEXLP	0.06	**1.56**	33.38	4.20	0	0	0
GurobiLP	0.02	**0.68**	11.97	1.69	6	6	11
LiftedLP	0.16	**23.53**	666.01	77.12	0	0	0
CPLEXSepLp	0.02	**0.26**	0.84	0.23	15	16	23
CPLEXTowerLp	0.02	**0.25**	1.14	0.25	11	11	18
CPLEXTowerSepLp	0.05	**0.55**	2.67	0.54	0	0	0
GurobiSepLp	0.02	**0.10**	1.33	0.14	62	62	64
GurobiTowerLp	0.02	**0.16**	0.96	0.19	2	2	3
GurobiTowerSepLp	0.05	**0.38**	2.67	0.48	0	0	0
CPLEXSepLazy	0.04	**0.35**	1.68	0.26	0	0	0
GurobiSepLazy	0.06	**0.39**	1.84	0.34	0	0	0

(c) Robust.

Table 2 Summary statistics of Solve Times for $n = 30$ [s].

Algorithm	min	avg	max	std	wins	1% win	10% win
CPLEXCP	0.33	**17.43**	568.02	58.58	0	0	0
GurobiCP	0.21	**24.56**	847.80	88.11	0	0	0
CPLEXLP	0.62	**731.57**	3600.42	1132.98	0	0	0
GurobiLP	0.64	**1157.74**	3600.30	1462.59	0	0	0
LiftedLP	0.37	**5.36**	60.57	9.19	0	0	0
CPLEXSepLp	0.13	**3.42**	70.71	7.78	3	4	5
CPLEXTowerLp	0.20	**4.89**	102.78	12.01	1	1	1
CPLEXTowerSepLp	0.28	**10.43**	207.66	24.01	0	0	0
GurobiSepLp	0.05	**1.28**	28.18	3.24	94	95	96
GurobiTowerLp	0.11	**8.85**	249.85	26.53	1	1	1
GurobiTowerSepLp	0.17	**21.59**	646.23	67.72	0	0	0
CPLEXSepLazy	0.20	**5.78**	111.98	14.02	0	0	0
GurobiSepLazy	0.24	**10.71**	415.53	42.17	1	1	1

(a) Classical.

Algorithm	min	avg	max	std	wins	1% win	10% win
CPLEXCP	0.64	**29.08**	1168.34	118.88	0	0	0
GurobiCP	0.40	**152.28**	1416.22	293.82	0	0	0
CPLEXLP	0.78	**325.52**	3600.01	738.07	0	0	0
GurobiLP	0.55	**685.09**	3600.86	1160.38	0	0	0
LiftedLP	0.95	**9.23**	379.92	38.05	1	1	2
CPLEXSepLp	0.33	**4.34**	64.07	8.61	3	4	6
CPLEXTowerLp	0.22	**4.43**	84.74	9.68	1	1	1
CPLEXTowerSepLp	0.64	**11.51**	168.16	21.32	0	0	0
GurobiSepLp	0.08	**2.45**	74.77	7.90	93	93	96
GurobiTowerLp	0.12	**8.80**	218.17	23.77	0	0	0
GurobiTowerSepLp	0.31	**31.74**	1393.77	141.23	0	0	0
CPLEXSepLazy	0.55	**5.45**	52.10	7.71	2	2	3
GurobiSepLazy	0.39	**13.79**	578.08	58.35	0	0	0

(b) Shortfall.

Algorithm	min	avg	max	std	wins	1% win	10% win
CPLEXCP	0.04	**1.86**	43.15	4.95	2	2	2
GurobiCP	0.08	**1.92**	41.76	6.11	0	0	0
CPLEXLP	0.08	**22.79**	1197.49	130.23	0	0	0
GurobiLP	0.02	**21.77**	1392.90	149.35	4	4	10
LiftedLP	0.78	**114.09**	3600.00	530.85	0	0	1
CPLEXSepLp	0.03	**0.71**	8.93	1.10	9	9	11
CPLEXTowerLp	0.03	**0.84**	13.61	1.79	8	8	13
CPLEXTowerSepLp	0.07	**1.87**	34.23	3.97	0	0	0
GurobiSepLp	0.03	**0.26**	4.54	0.51	69	69	74
GurobiTowerLp	0.03	**0.83**	19.90	2.66	7	7	13
GurobiTowerSepLp	0.07	**2.07**	52.12	6.82	0	0	0
CPLEXSepLazy	0.06	**0.85**	7.78	1.24	1	1	1
GurobiSepLazy	0.10	**1.30**	32.36	3.54	0	0	0

(c) Robust.

Table 3 Summary statistics of Solve Times for $n = 40$ [s].

Algorithm	min	avg	max	std	wins	1% win	10% win
CPLEXCP	0.43	**188.32**	3600.25	511.63	0	0	0
GurobiCP	0.33	**254.29**	3600.01	619.08	0	0	0
CPLEXLP	0.52	**1941.58**	3600.07	1515.59	0	0	0
GurobiLP	1.06	**2375.47**	3608.86	1488.32	0	0	0
LiftedLP	0.73	**42.46**	1368.69	147.18	1	1	2
CPLEXSepLp	0.14	**21.88**	471.47	57.54	7	8	9
CPLEXTowerLp	0.24	**33.96**	858.26	104.09	1	1	2
CPLEXTowerSepLp	0.57	**67.57**	1957.18	210.80	0	0	0
GurobiSepLp	0.07	**21.63**	1255.85	126.13	87	88	91
GurobiTowerLp	0.08	**102.69**	3600.00	382.69	0	0	1
GurobiTowerSepLp	0.31	**197.25**	3600.00	501.15	0	0	0
CPLEXSepLazy	0.30	**28.20**	384.63	65.40	4	4	4
GurobiSepLazy	0.32	**88.79**	3600.00	369.09	0	0	0

(a) Classical.

Algorithm	min	avg	max	std	wins	1% win	10% win
CPLEXCP	1.01	**160.11**	3600.22	427.35	0	0	0
GurobiCP	0.53	**1148.66**	3600.02	1460.67	0	0	0
CPLEXLP	0.62	**1231.56**	3600.44	1446.50	0	0	0
GurobiLP	0.32	**1773.43**	3658.96	1614.40	0	0	0
LiftedLP	0.94	**28.42**	1000.65	102.41	2	3	3
CPLEXSepLp	0.35	**16.64**	458.53	48.68	12	12	15
CPLEXTowerLp	0.35	**24.49**	889.35	90.93	4	4	5
CPLEXTowerSepLp	0.54	**77.90**	2216.07	261.45	0	0	0
GurobiSepLp	0.12	**20.27**	876.51	91.10	78	79	79
GurobiTowerLp	0.19	**74.46**	3600.00	363.09	1	1	1
GurobiTowerSepLp	0.58	**147.20**	3600.00	405.19	0	0	0
CPLEXSepLazy	0.62	**21.93**	572.22	62.57	3	3	6
GurobiSepLazy	0.44	**74.11**	3562.78	359.48	0	0	0

(b) Shortfall.

Algorithm	min	avg	max	std	wins	1% win	10% win
CPLEXCP	0.06	**9.35**	226.54	31.22	3	3	3
GurobiCP	0.12	**10.76**	355.45	42.73	0	0	0
CPLEXLP	0.27	**133.90**	3600.02	563.01	0	0	0
GurobiLP	0.04	**162.41**	3604.89	709.57	3	4	4
LiftedLP	0.92	**62.95**	3600.01	365.07	1	1	1
CPLEXSepLp	0.04	**1.95**	20.83	3.70	14	14	19
CPLEXTowerLp	0.04	**2.92**	72.92	8.85	1	1	3
CPLEXTowerSepLp	0.10	**7.49**	190.99	23.24	0	0	0
GurobiSepLp	0.04	**1.64**	57.29	6.40	71	71	77
GurobiTowerLp	0.05	**4.75**	103.85	16.26	6	6	11
GurobiTowerSepLp	0.14	**11.63**	253.89	39.10	0	0	0
CPLEXSepLazy	0.11	**2.65**	45.20	6.24	1	1	1
GurobiSepLazy	0.12	**4.40**	76.78	12.17	0	0	0

(c) Robust.

Table 4 Summary statistics of Solve Times for $n = 50$ [s].

Algorithm	min	avg	max	std	wins	1% win	10% win
CPLEXCP	0.63	**440.28**	3602.68	851.14	0	0	0
GurobiCP	0.57	**574.65**	3600.00	1016.49	0	0	0
CPLEXLP	3.50	**2664.89**	3601.10	1354.02	0	0	0
GurobiLP	4.35	**2969.53**	3617.48	1237.05	0	0	0
LiftedLP	1.71	**141.92**	3471.98	452.83	1	1	2
CPLEXSepLp	0.31	**68.85**	1293.39	174.87	4	4	8
CPLEXTowerLp	0.38	**110.91**	2728.20	318.85	0	0	1
CPLEXTowerSepLp	0.67	**212.91**	3600.00	526.40	0	0	0
GurobiSepLp	0.08	**80.51**	3600.00	374.64	74	75	77
GurobiTowerLp	0.15	**254.80**	3600.00	649.20	0	0	0
GurobiTowerSepLp	0.31	**477.73**	3600.00	886.20	0	0	0
CPLEXSepLazy	0.54	**56.38**	1100.44	149.15	20	20	20
GurobiSepLazy	0.53	**200.00**	3600.00	541.28	1	1	1

(a) Classical.

Algorithm	min	avg	max	std	wins	1% win	10% win
CPLEXCP	1.58	**447.80**	3600.00	849.68	0	0	0
GurobiCP	4.71	**1947.87**	3600.03	1594.42	0	0	0
CPLEXLP	2.51	**2065.17**	3601.08	1569.51	0	0	0
GurobiLP	1.61	**2440.88**	3609.79	1453.93	0	0	0
LiftedLP	2.55	**75.88**	1793.78	211.91	9	9	10
CPLEXSepLp	0.82	**61.51**	2284.68	234.88	9	9	13
CPLEXTowerLp	0.51	**97.22**	3600.95	406.79	2	3	3
CPLEXTowerSepLp	1.93	**238.37**	3600.00	558.70	0	0	0
GurobiSepLp	0.16	**84.67**	3600.00	383.37	65	65	68
GurobiTowerLp	0.32	**181.66**	3600.00	493.36	0	0	1
GurobiTowerSepLp	0.75	**386.34**	3600.01	772.94	0	0	0
CPLEXSepLazy	1.56	**59.45**	1643.17	175.33	15	15	18
GurobiSepLazy	1.07	**141.33**	3600.00	416.08	0	0	0

(b) Shortfall.

Algorithm	min	avg	max	std	wins	1% win	10% win
CPLEXCP	0.08	**12.35**	325.85	34.59	1	1	1
GurobiCP	0.61	**21.46**	1006.61	103.22	0	0	0
CPLEXLP	0.52	**197.03**	3600.02	608.43	0	0	0
GurobiLP	0.07	**211.17**	3600.01	681.26	1	1	2
LiftedLP	1.43	**71.35**	3600.01	399.94	3	3	4
CPLEXSepLp	0.41	**2.77**	26.13	3.25	10	10	16
CPLEXTowerLp	0.06	**4.12**	62.27	7.32	2	2	3
CPLEXTowerSepLp	0.15	**9.01**	124.45	14.65	0	0	0
GurobiSepLp	0.13	**2.28**	29.70	4.60	71	72	73
GurobiTowerLp	0.09	**5.29**	113.54	13.76	11	11	14
GurobiTowerSepLp	0.21	**14.59**	300.93	37.99	0	0	0
CPLEXSepLazy	0.17	**3.31**	36.10	4.39	0	0	2
GurobiSepLazy	0.23	**5.10**	64.45	9.08	1	1	2

(c) Robust.

Table 5 Summary statistics of Solve Times for $n = 60$ [s].

Algorithm	min	avg	max	std	wins	1% win	10% win
LiftedLP	6.62	**1624.83**	3600.02	1471.37	3	17	22
CPLEXSepLp	1.66	**1484.69**	3600.21	1403.10	14	28	33
GurobiSepLp	0.34	**1439.88**	3600.31	1437.49	42	56	57
CPLEXSepLazy	2.98	**1452.46**	3600.02	1463.32	40	40	45
GurobiSepLazy	2.55	**2155.33**	3600.02	1477.84	1	15	16

(a) Classical.

Algorithm	min	avg	max	std	wins	1% win	10% win
LiftedLP	13.16	**1345.62**	3600.12	1391.11	19	24	30
CPLEXSepLp	2.39	**1232.79**	3600.11	1302.56	14	22	27
GurobiSepLp	0.58	**1493.44**	3600.19	1493.08	29	37	40
CPLEXSepLazy	5.16	**1129.58**	3600.12	1305.08	37	42	46
GurobiSepLazy	3.73	**2081.67**	3600.08	1506.22	1	8	8

(b) Shortfall.

Algorithm	min	avg	max	std	wins	1% win	10% win
CPLEXCP	3.62	**162.04**	2043.65	375.08	0	0	0
GurobiCP	2.55	**196.83**	3600.03	509.70	0	0	0
CPLEXLP	2.00	**1056.43**	3600.10	1362.59	0	0	0
GurobiLP	0.47	**1079.13**	3603.90	1472.19	1	1	1
LiftedLP	4.27	**71.45**	3430.89	345.06	2	2	6
CPLEXSepLp	1.38	**17.48**	144.82	26.37	25	25	30
CPLEXTowerLp	0.99	**29.77**	535.74	62.07	0	0	0
CPLEXTowerSepLp	2.42	**104.37**	2712.22	324.25	0	0	0
GurobiSepLp	0.39	**32.04**	595.06	86.96	57	57	59
GurobiTowerLp	0.62	**58.55**	658.53	124.77	5	6	7
GurobiTowerSepLp	1.01	**167.12**	1750.92	343.94	0	0	0
CPLEXSepLazy	1.73	**23.22**	229.88	41.66	7	7	10
GurobiSepLazy	1.43	**54.06**	706.76	126.63	3	3	5

(c) Robust.

Table 6 Summary statistics of Solve Times for $n = 100$ [s].

Algorithm	min	avg	max	std	wins	1% win	10% win
CPLEXCP	28.85	**2172.25**	3600.07	1441.31	0	1	1
GurobiCP	21.90	**2053.05**	3600.09	1435.75	0	1	1
CPLEXLP	14.74	**3036.45**	3600.11	1143.01	0	1	1
GurobiLP	5.63	**2841.03**	3604.06	1342.22	0	1	1
LiftedLP	18.26	**529.57**	3600.04	832.69	6	8	11
CPLEXSepLp	5.00	**426.90**	3600.04	854.99	23	25	33
CPLEXTowerLp	6.77	**740.08**	3600.06	1163.97	0	1	2
CPLEXTowerSepLp	12.89	**1151.67**	3600.40	1271.52	0	1	1
GurobiSepLp	1.03	**593.35**	3600.05	1064.61	54	55	56
GurobiTowerLp	3.02	**1033.21**	3600.05	1294.31	1	2	2
GurobiTowerSepLp	6.31	**1685.98**	3600.07	1411.79	0	1	1
CPLEXSepLazy	5.51	**406.56**	3600.01	752.93	16	17	24
GurobiSepLazy	7.23	**954.15**	3600.09	1245.66	0	1	2

(a) Robust.

Table 7 Summary statistics of Solve Times for $n = 200$ [s].

Algorithm	min	avg	max	std	wins	1% win	10% win
LiftedLP	92.22	**1660.09**	3602.30	1426.85	1	20	20
CPLEXSepLp	27.42	**1381.03**	3600.08	1417.33	23	42	46
GurobiSepLp	4.11	**1429.40**	3600.12	1432.61	45	65	66
CPLEXSepLazy	31.46	**1390.95**	3600.05	1422.92	22	31	34
GurobiSepLazy	13.96	**2114.17**	3600.05	1470.40	9	19	19

(a) Robust.

Table 8 Summary statistics of Solve Times for $n = 300$ [s].

B.2 Summary Statistics for Solution Quality

Tables 9–13 present summary statistics for the maximum violation of constraints (1c) by the optimal solution returned by the solver. This value is calculated as

$$\max_{l=1}^{q} \left\| A^l \bar{x} + b^l \right\|_2^2 - \left(a^l \bar{x} + b_0^l \right)^2$$

where $\bar{x}$ is the optimal solution.

Algorithm	min	avg	max	std
LiftedLP	0.00e+00	**5.87e-09**	8.31e-08	1.20e-08
CPLEXSepLp	9.36e-11	**5.66e-09**	5.50e-08	8.56e-09
CPLEXTowerLp	0.00e+00	**6.24e-09**	6.36e-08	9.71e-09
CPLEXTowerSepLp	0.00e+00	**4.16e-09**	3.59e-08	7.12e-09
GurobiSepLp	3.84e-04	**5.98e-04**	8.28e-04	8.31e-05
GurobiTowerLp	3.88e-04	**5.68e-04**	7.76e-04	8.14e-05
GurobiTowerSepLp	4.02e-06	**3.92e-05**	1.54e-04	3.13e-05
CPLEXSepLazy	4.68e-06	**6.40e-06**	8.82e-06	8.99e-07
GurobiSepLazy	4.11e-06	**6.31e-06**	8.72e-06	9.17e-07

(a) Classical.

Algorithm	min	avg	max	std
LiftedLP	0.00e+00	**0.00e+00**	0.00e+00	0.00e+00
CPLEXSepLp	0.00e+00	**5.19e-10**	1.30e-08	2.03e-09
CPLEXTowerLp	0.00e+00	**1.67e-11**	1.09e-09	1.23e-10
CPLEXTowerSepLp	0.00e+00	**6.25e-10**	5.18e-08	5.21e-09
GurobiSepLp	4.19e-04	**6.47e-04**	8.46e-04	8.87e-05
GurobiTowerLp	3.81e-04	**5.65e-04**	8.39e-04	9.37e-05
GurobiTowerSepLp	4.82e-06	**5.36e-05**	2.50e-04	4.29e-05
CPLEXSepLazy	0.00e+00	**6.37e-06**	9.33e-06	1.35e-06
GurobiSepLazy	0.00e+00	**6.17e-06**	8.90e-06	1.47e-06

(b) Shortfall.

Algorithm	min	avg	max	std
LiftedLP	0.00e+00	**6.59e-09**	8.05e-08	1.35e-08
CPLEXSepLp	1.26e-11	**4.74e-09**	3.57e-08	6.12e-09
CPLEXTowerLp	7.06e-11	**5.09e-09**	8.23e-08	1.01e-08
CPLEXTowerSepLp	1.02e-11	**2.40e-09**	2.69e-08	4.59e-09
GurobiSepLp	4.51e-04	**6.27e-04**	1.02e-03	9.93e-05
GurobiTowerLp	1.92e-06	**3.07e-05**	1.19e-04	2.11e-05
GurobiTowerSepLp	2.27e-06	**4.90e-05**	1.44e-04	3.46e-05
CPLEXSepLazy	2.29e-06	**4.83e-06**	8.02e-06	1.16e-06
GurobiSepLazy	2.29e-06	**4.71e-06**	7.50e-06	1.05e-06

(c) Robust.

Table 9 Summary statistics of conic constraint violation for $n = 20$.

Algorithm	min	avg	max	std
LiftedLP	0.00e+00	**7.78e-09**	9.01e-08	1.57e-08
CPLEXSepLp	0.00e+00	**9.56e-09**	7.86e-08	1.41e-08
CPLEXTowerLp	0.00e+00	**5.75e-09**	5.76e-08	1.02e-08
CPLEXTowerSepLp	0.00e+00	**1.67e-09**	1.95e-08	3.52e-09
GurobiSepLp	4.91e-04	**7.52e-04**	1.03e-03	1.10e-04
GurobiTowerLp	4.77e-04	**7.10e-04**	1.04e-03	1.18e-04
GurobiTowerSepLp	2.71e-06	**4.19e-05**	1.69e-04	3.20e-05
CPLEXSepLazy	4.73e-06	**8.32e-06**	1.17e-05	1.25e-06
GurobiSepLazy	4.93e-06	**8.23e-06**	1.19e-05	1.34e-06

(a) Classical.

Algorithm	min	avg	max	std
LiftedLP	0.00e+00	**0.00e+00**	0.00e+00	0.00e+00
CPLEXSepLp	0.00e+00	**2.32e-09**	2.40e-08	4.58e-09
CPLEXTowerLp	0.00e+00	**0.00e+00**	0.00e+00	0.00e+00
CPLEXTowerSepLp	0.00e+00	**0.00e+00**	0.00e+00	0.00e+00
GurobiSepLp	5.46e-04	**7.91e-04**	1.15e-03	1.12e-04
GurobiTowerLp	0.00e+00	**3.93e-05**	2.42e-04	3.31e-05
GurobiTowerSepLp	0.00e+00	**5.30e-05**	3.60e-04	5.19e-05
CPLEXSepLazy	0.00e+00	**8.38e-06**	1.11e-05	1.38e-06
GurobiSepLazy	0.00e+00	**8.20e-06**	1.20e-05	1.95e-06

(b) Shortfall.

Algorithm	min	avg	max	std
LiftedLP	0.00e+00	**6.47e-09**	1.12e-07	1.50e-08
CPLEXSepLp	2.17e-11	**4.49e-09**	5.83e-08	7.64e-09
CPLEXTowerLp	1.13e-11	**3.68e-09**	4.03e-08	6.82e-09
CPLEXTowerSepLp	0.00e+00	**9.52e-10**	1.42e-08	1.74e-09
GurobiSepLp	5.57e-04	**8.22e-04**	1.23e-03	1.30e-04
GurobiTowerLp	1.04e-06	**2.95e-05**	1.42e-04	2.65e-05
GurobiTowerSepLp	8.51e-07	**4.09e-05**	1.71e-04	3.84e-05
CPLEXSepLazy	2.70e-06	**6.75e-06**	1.00e-05	1.39e-06
GurobiSepLazy	2.21e-06	**6.31e-06**	9.71e-06	1.46e-06

(c) Robust.

Table 10 Summary statistics of conic constraint violation for $n = 30$.

Algorithm	min	avg	max	std
LiftedLP	0.00e+00	**4.47e-09**	1.23e-07	1.68e-08
CPLEXSepLp	3.96e-11	**9.40e-09**	8.67e-08	1.56e-08
CPLEXTowerLp	0.00e+00	**3.38e-09**	3.01e-08	6.07e-09
CPLEXTowerSepLp	0.00e+00	**1.02e-09**	4.12e-08	4.36e-09
GurobiSepLp	6.61e-04	**9.23e-04**	1.31e-03	1.33e-04
GurobiTowerLp	1.60e-06	**3.91e-05**	2.31e-04	3.83e-05
GurobiTowerSepLp	3.59e-06	**6.26e-05**	2.20e-04	5.33e-05
CPLEXSepLazy	6.73e-06	**9.94e-06**	1.32e-05	1.32e-06
GurobiSepLazy	7.19e-06	**1.03e-05**	1.41e-05	1.48e-06

(a) Classical.

Algorithm	min	avg	max	std
LiftedLP	0.00e+00	**0.00e+00**	0.00e+00	0.00e+00
CPLEXSepLp	0.00e+00	**4.15e-09**	4.05e-08	6.54e-09
CPLEXTowerLp	0.00e+00	**0.00e+00**	0.00e+00	0.00e+00
CPLEXTowerSepLp	0.00e+00	**1.75e-11**	1.55e-09	1.56e-10
GurobiSepLp	7.17e-04	**9.36e-04**	1.26e-03	1.16e-04
GurobiTowerLp	2.82e-07	**5.86e-05**	3.51e-04	6.24e-05
GurobiTowerSepLp	5.73e-06	**6.74e-05**	6.29e-04	8.16e-05
CPLEXSepLazy	7.61e-06	**1.05e-05**	1.41e-05	1.32e-06
GurobiSepLazy	0.00e+00	**1.00e-05**	1.30e-05	1.65e-06

(b) Shortfall.

Algorithm	min	avg	max	std
LiftedLP	0.00e+00	**5.48e-09**	8.07e-08	1.35e-08
CPLEXSepLp	6.25e-12	**3.72e-09**	3.74e-08	6.68e-09
CPLEXTowerLp	0.00e+00	**2.25e-09**	3.65e-08	4.38e-09
CPLEXTowerSepLp	0.00e+00	**1.54e-09**	2.68e-08	3.31e-09
GurobiSepLp	7.19e-04	**9.86e-04**	1.35e-03	1.57e-04
GurobiTowerLp	3.04e-06	**2.97e-05**	1.38e-04	2.42e-05
GurobiTowerSepLp	2.77e-06	**4.00e-05**	2.34e-04	3.66e-05
CPLEXSepLazy	3.38e-06	**8.47e-06**	1.25e-05	1.90e-06
GurobiSepLazy	3.58e-06	**8.51e-06**	1.32e-05	1.94e-06

(c) Robust.

Table 11 Summary statistics of conic constraint violation for $n = 40$.

Algorithm	min	avg	max	std
LiftedLP	0.00e+00	**5.00e-09**	1.46e-07	2.11e-08
CPLEXSepLp	1.27e-11	**9.51e-09**	5.63e-08	1.20e-08
CPLEXTowerLp	0.00e+00	**3.34e-09**	5.17e-08	6.74e-09
CPLEXTowerSepLp	0.00e+00	**1.14e-09**	4.82e-08	5.31e-09
GurobiSepLp	7.63e-04	**1.13e-03**	1.52e-03	1.56e-04
GurobiTowerLp	0.00e+00	**6.45e-05**	3.19e-04	5.47e-05
GurobiTowerSepLp	0.00e+00	**6.97e-05**	3.03e-04	6.10e-05
CPLEXSepLazy	8.87e-06	**1.23e-05**	1.64e-05	1.51e-06
GurobiSepLazy	7.96e-06	**1.21e-05**	1.62e-05	1.43e-06

(a) Classical.

Algorithm	min	avg	max	std
LiftedLP	0.00e+00	**0.00e+00**	0.00e+00	0.00e+00
CPLEXSepLp	0.00e+00	**4.62e-09**	3.75e-08	6.74e-09
CPLEXTowerLp	0.00e+00	**0.00e+00**	0.00e+00	0.00e+00
CPLEXTowerSepLp	0.00e+00	**0.00e+00**	0.00e+00	0.00e+00
GurobiSepLp	7.21e-04	**1.10e-03**	1.54e-03	1.49e-04
GurobiTowerLp	0.00e+00	**8.81e-05**	3.75e-04	9.06e-05
GurobiTowerSepLp	0.00e+00	**9.94e-05**	4.00e-04	8.90e-05
CPLEXSepLazy	6.48e-06	**1.22e-05**	1.70e-05	1.91e-06
GurobiSepLazy	0.00e+00	**1.22e-05**	1.66e-05	2.05e-06

(b) Shortfall.

Algorithm	min	avg	max	std
LiftedLP	0.00e+00	**6.56e-09**	2.55e-07	2.76e-08
CPLEXSepLp	2.47e-11	**5.58e-09**	6.47e-08	1.26e-08
CPLEXTowerLp	0.00e+00	**2.99e-09**	3.95e-08	5.81e-09
CPLEXTowerSepLp	0.00e+00	**5.75e-10**	1.49e-08	1.87e-09
GurobiSepLp	8.24e-04	**1.15e-03**	1.54e-03	1.66e-04
GurobiTowerLp	2.00e-06	**3.71e-05**	2.38e-04	4.06e-05
GurobiTowerSepLp	2.56e-06	**4.73e-05**	1.94e-04	4.30e-05
CPLEXSepLazy	4.09e-06	**1.01e-05**	1.48e-05	2.07e-06
GurobiSepLazy	3.59e-06	**1.01e-05**	1.65e-05	2.17e-06

(c) Robust.

Table 12 Summary statistics of conic constraint violation for $n = 50$.

Algorithm	min	avg	max	std
LiftedLP	0.00e+00	**7.16e-09**	1.57e-07	2.34e-08
CPLEXSepLp	0.00e+00	**1.14e-08**	1.32e-07	2.02e-08
CPLEXTowerLp	0.00e+00	**2.23e-09**	2.41e-08	4.45e-09
CPLEXTowerSepLp	0.00e+00	**3.57e-10**	2.38e-08	2.41e-09
GurobiSepLp	0.00e+00	**1.31e-03**	1.69e-03	2.19e-04
GurobiTowerLp	0.00e+00	**6.18e-05**	2.65e-04	5.12e-05
GurobiTowerSepLp	0.00e+00	**8.76e-05**	4.45e-04	8.17e-05
CPLEXSepLazy	1.11e-05	**1.39e-05**	1.83e-05	1.56e-06
GurobiSepLazy	8.65e-06	**1.41e-05**	1.90e-05	1.89e-06

(a) Classical.

Algorithm	min	avg	max	std
LiftedLP	0.00e+00	**0.00e+00**	0.00e+00	0.00e+00
CPLEXSepLp	0.00e+00	**6.95e-09**	1.61e-07	1.79e-08
CPLEXTowerLp	0.00e+00	**1.06e-11**	1.06e-09	1.06e-10
CPLEXTowerSepLp	0.00e+00	**0.00e+00**	0.00e+00	0.00e+00
GurobiSepLp	0.00e+00	**1.24e-03**	1.63e-03	1.85e-04
GurobiTowerLp	0.00e+00	**1.14e-04**	5.25e-04	9.93e-05
GurobiTowerSepLp	0.00e+00	**1.16e-04**	5.57e-04	1.09e-04
CPLEXSepLazy	9.24e-06	**1.43e-05**	2.01e-05	1.97e-06
GurobiSepLazy	1.06e-05	**1.43e-05**	2.19e-05	2.03e-06

(b) Shortfall.

Algorithm	min	avg	max	std
LiftedLP	0.00e+00	**6.00e-09**	8.58e-08	1.42e-08
CPLEXSepLp	2.80e-11	**7.93e-09**	1.14e-07	1.46e-08
CPLEXTowerLp	0.00e+00	**2.10e-09**	4.10e-08	4.76e-09
CPLEXTowerSepLp	0.00e+00	**5.02e-10**	1.48e-08	1.83e-09
GurobiSepLp	9.63e-04	**1.31e-03**	1.56e-03	1.27e-04
GurobiTowerLp	2.06e-06	**3.58e-05**	1.27e-04	2.96e-05
GurobiTowerSepLp	2.53e-06	**4.82e-05**	2.06e-04	4.36e-05
CPLEXSepLazy	8.41e-06	**1.27e-05**	1.71e-05	1.45e-06
GurobiSepLazy	5.04e-06	**1.21e-05**	1.77e-05	1.83e-06

(c) Robust.

Table 13 Summary statistics of conic constraint violation for $n = 60$.

Algorithm	min	avg	max	std
LiftedLP	0.00e+00	**4.18e-09**	1.13e-07	1.40e-08
CPLEXSepLp	0.00e+00	**9.51e-09**	1.13e-07	1.97e-08
GurobiSepLp	0.00e+00	**1.68e-03**	3.04e-03	9.73e-04
CPLEXSepLazy	1.58e-05	**2.12e-05**	2.59e-05	1.97e-06
GurobiSepLazy	1.39e-05	**2.12e-05**	2.69e-05	2.31e-06

(a) Classical.

Algorithm	min	avg	max	std
LiftedLP	0.00e+00	**0.00e+00**	0.00e+00	0.00e+00
CPLEXSepLp	0.00e+00	**9.92e-09**	1.21e-07	2.03e-08
GurobiSepLp	0.00e+00	**1.39e-03**	2.29e-03	8.42e-04
CPLEXSepLazy	1.41e-05	**2.14e-05**	2.98e-05	2.61e-06
GurobiSepLazy	0.00e+00	**2.19e-05**	2.93e-05	3.30e-06

(b) Shortfall.

Algorithm	min	avg	max	std
CPLEXCP	0.00e+00	**6.93e-09**	1.78e-07	2.69e-08
GurobiCP	0.00e+00	**2.55e-08**	1.18e-06	1.30e-07
CPLEXLP	0.00e+00	**4.17e-09**	3.42e-08	6.27e-09
GurobiLP	0.00e+00	**6.43e-05**	9.99e-05	3.59e-05
LiftedLP	0.00e+00	**8.18e-09**	1.01e-07	1.88e-08
CPLEXSepLp	0.00e+00	**1.57e-08**	1.61e-07	3.26e-08
CPLEXTowerLp	0.00e+00	**1.20e-09**	3.95e-08	4.32e-09
CPLEXTowerSepLp	0.00e+00	**1.15e-10**	7.55e-09	7.60e-10
GurobiSepLp	1.51e-03	**1.86e-03**	2.35e-03	1.94e-04
GurobiTowerLp	2.34e-06	**5.25e-05**	2.63e-04	4.32e-05
GurobiTowerSepLp	2.60e-06	**6.64e-05**	4.05e-04	6.14e-05
CPLEXSepLazy	1.43e-05	**2.04e-05**	2.77e-05	2.52e-06
GurobiSepLazy	8.77e-06	**1.93e-05**	2.44e-05	2.37e-06

(c) Robust.

Table 14 Summary statistics of conic constraint violation for $n = 100$ [s].

Algorithm	min	avg	max	std
CPLEXCP	0.00e+00	**1.73e-09**	8.96e-08	9.49e-09
GurobiCP	0.00e+00	**2.69e-08**	7.83e-07	1.07e-07
CPLEXLP	0.00e+00	**2.04e-09**	5.48e-08	7.18e-09
GurobiLP	0.00e+00	**2.61e-05**	9.86e-05	3.97e-05
LiftedLP	0.00e+00	**6.70e-09**	1.66e-07	2.14e-08
CPLEXSepLp	0.00e+00	**2.19e-07**	7.79e-07	1.74e-07
CPLEXTowerLp	0.00e+00	**2.92e-11**	1.14e-09	1.61e-10
CPLEXTowerSepLp	0.00e+00	**4.47e-10**	1.28e-08	2.09e-09
GurobiSepLp	0.00e+00	**3.40e-03**	4.28e-03	9.77e-04
GurobiTowerLp	0.00e+00	**6.96e-05**	2.94e-04	6.28e-05
GurobiTowerSepLp	0.00e+00	**7.14e-05**	2.58e-04	6.26e-05
CPLEXSepLazy	2.34e-05	**3.84e-05**	4.78e-05	3.89e-06
GurobiSepLazy	3.41e-08	**3.80e-05**	4.50e-05	5.21e-06

(a) Robust.

Table 15 Summary statistics of conic constraint violation for $n = 200$ [s].

Algorithm	min	avg	max	std
LiftedLP	0.00e+00	**4.61e-09**	9.94e-08	1.35e-08
CPLEXSepLp	0.00e+00	**2.97e-07**	9.07e-07	2.48e-07
GurobiSepLp	0.00e+00	**4.09e-03**	6.35e-03	2.39e-03
CPLEXSepLazy	4.47e-05	**5.79e-05**	6.91e-05	4.48e-06
GurobiSepLazy	3.90e-05	**5.60e-05**	6.81e-05	5.63e-06

(a) Robust.

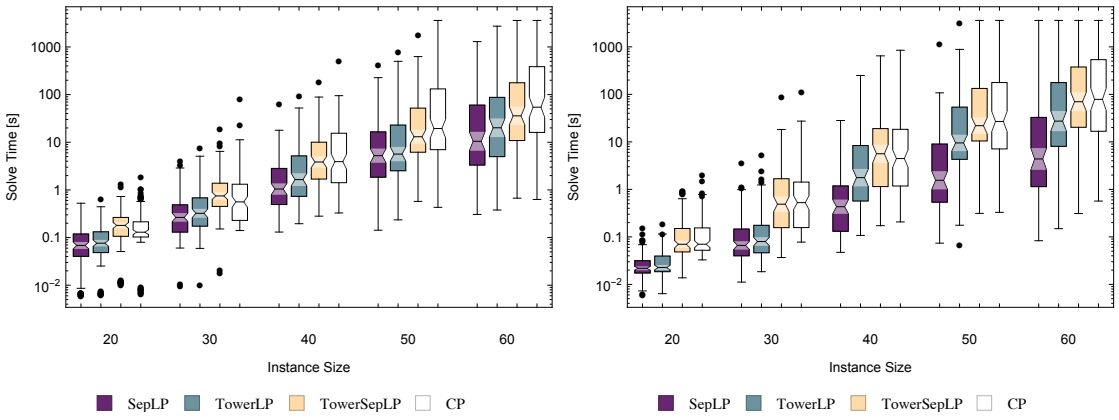
Table 16 Summary statistics of conic constraint violation for $n = 300$ [s].

B.3 Additional Graphs

Figure 6 compares the performance of the three dynamic lifted polyhedral relaxations in their nonlinear reformulation version (CPLEXSepLP, GurobiSepLP, CPLEXTowerLP, GurobiTowerLP, CPLEXTowerSepLP, GurobiTowerSepLP). We also include as a reference the NLP-based algorithms CPLEXCP and GurobiCP.

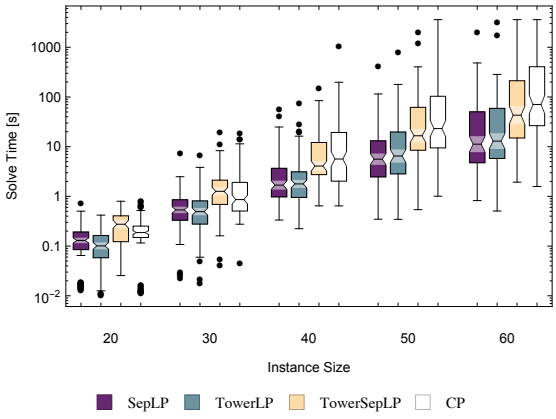
LiftedLP Our final set of charts compare the branch- and cut-based LiftedLP algorithms (LiftedLP, CPLEXSepLazy and GurobiSepLazy) with the separable dynamic lifted polyhedral relaxation in its nonlinear reformulation version (CPLEXSepLP and GurobiSepLP). To minimize the complexity of the graphics we divide the comparison into two parts. In Figure 7 we compare the branch-based algorithm with the separable reformulations and in Figure 8 we compare the cut-based algorithms with the separable reformulations.

From Figure 7 we see that the separable reformulations provide an advantage over the branch-based LiftedLP algorithm for all instances. However, this advantage becomes smaller as the problems sizes increase and the LiftedLP algorithm can provide an advantage in some instances (e.g Table 7 shows the algorithms is the fastest in 10% of the robust instances for $n = 200$). Figure 8 shows a similar behavior for the cut-based algorithms.

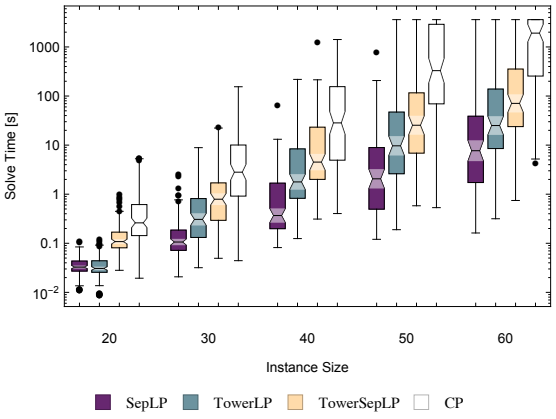


(a) Classical-CPLEX.

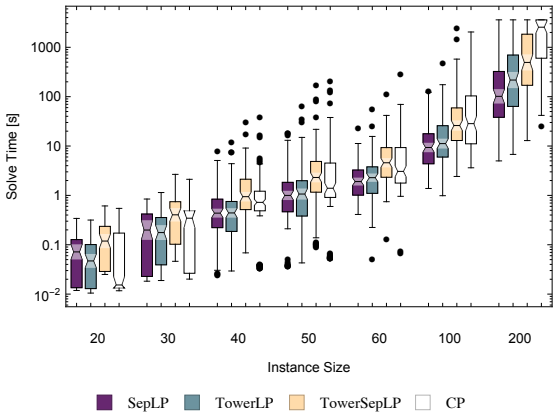
(b) Classical-Gurobi.



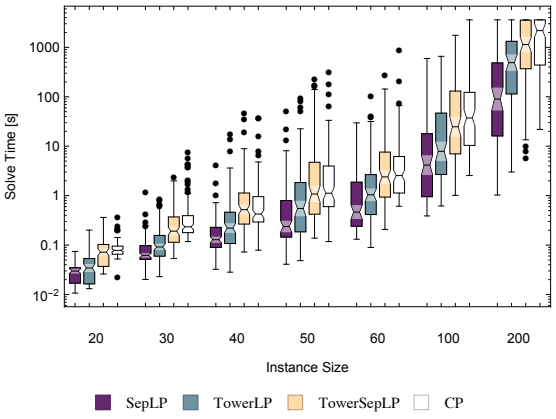
(c) Shortfall-CPLEX.



(d) Shortfall-Gurobi.

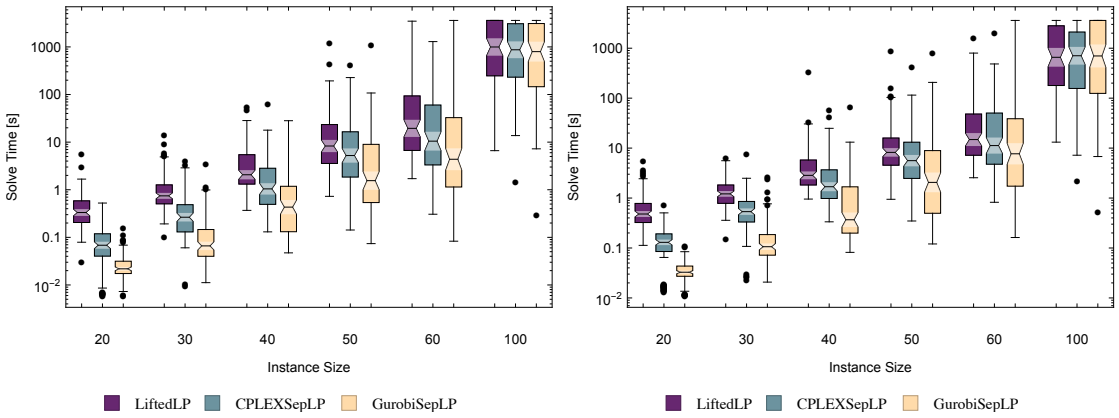


(e) Robust-CPLEX.



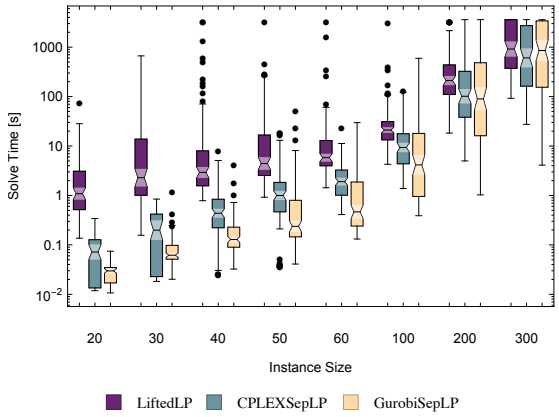
(f) Robust-Gurobi.

Fig. 6 Solution times for dynamic lifted polyhedral relaxations solved by standard LP-based algorithms [s].



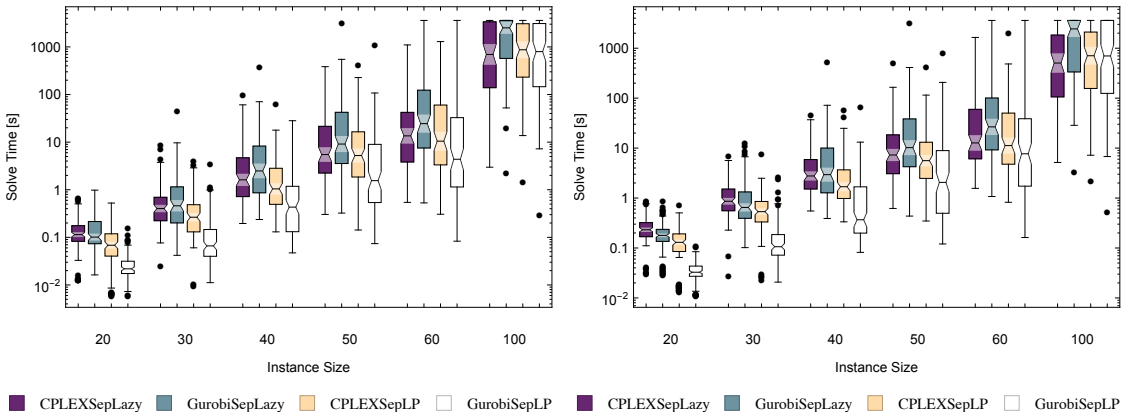
(a) Classical.

(b) Shortfall.



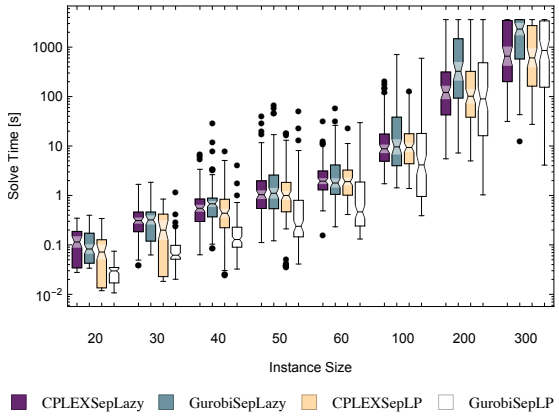
(c) Robust.

Fig. 7 Solution times for the branch-based LiftedLP algorithm and best LP-based (separable re-formulation) algorithms [s].



(a) Classical.

(b) Shortfall.



(c) Robust.

Fig. 8 Solution times for the cut-based LiftedLP algorithms and best LP-based (separable re-formulation) algorithms [s].