Outer Approximation With Conic Certificates For Mixed-Integer Convex Problems[*]

Chris Coey[§], Miles Lubin[§], and Juan Pablo Vielma[‡]

[§]MIT Operations Research Center
[‡]MIT Sloan School of Management
coey@mit.edu, miles.lubin@gmail.com, jvielma@mit.edu

December 14, 2019

Abstract

A mixed-integer convex (MI-convex) optimization problem is one that becomes convex when all integrality constraints are relaxed. We present a branch-and-bound LP outer approximation algorithm for an MI-convex problem transformed to *MI-conic* form. The polyhedral relaxations are refined with $\mathcal{K}^*$ *cuts* derived from *conic certificates* for continuous primal-dual conic subproblems. Under the assumption that all subproblems are *well-posed*, the algorithm detects infeasibility or unboundedness or returns an optimal solution in finite time. Using properties of the conic certificates, we show that the $\mathcal{K}^*$ cuts imply certain practically-relevant guarantees about the quality of the polyhedral relaxations, and demonstrate how to maintain helpful guarantees when the LP solver uses a positive feasibility tolerance. We discuss how to *disaggregate* $\mathcal{K}^*$ cuts in order to tighten the polyhedral relaxations and thereby improve the speed of convergence, and propose fast heuristic methods of obtaining useful $\mathcal{K}^*$ cuts. Our new open source MI-conic solver *Pajarito* (github.com/JuliaOpt/Pajarito.jl) uses an external mixed-integer linear (MILP) solver to manage the search tree and an external continuous conic solver for subproblems. Benchmarking on a library of mixed-integer second-order cone (MISOCP) problems, we find that Pajarito greatly outperforms Bonmin (the leading open source alternative) and is competitive with CPLEX's specialized MISOCP algorithm. We demonstrate the robustness of Pajarito by solving diverse MI-conic problems involving mixtures of positive semidefinite, second-order, and exponential cones, and provide evidence for the practical value of our analyses and enhancements of $\mathcal{K}^*$ cuts.

[*]M. Lubin is now at Google. The authors thank MOSEK for access to alpha releases of MOSEK 9 and Russell Bent and Emre Yamangil for support for and early contributions to Pajarito. This work has been partially funded by the National Science Foundation under grant CMMI-1351619, the Department of Energy Computational Science Graduate Fellowship under grant DE-FG02-97ER25308, and the Office of Naval Research under grant N00014-18-1-2079.

Contents

1 Mixed-Integer Convex Optimization

A *mixed-integer convex* (MI-convex) problem is a finite-dimensional optimization problem that minimizes a convex objective function over convex constraints and integrality restrictions on a subset of the variables. Belotti et al. [2013] and Bonami et al. [2012] review MI-convex applications and Lubin et al. [2017a] characterize which nonconvex feasible regions are MI-convex-representable. Since an MI-convex problem without integrality restrictions is just a convex problem, MI-convex optimization generalizes both mixed-integer linear optimization (MILP) and convex optimization. This structure also leads to effective branch-and-bound (B&B) algorithms, which recursively partition the possible values of the integer variables in a search tree and obtain objective bounds and feasible solutions from efficiently-solvable subproblems.

1.1 Branch-And-Bound Algorithms

A *nonlinear B&B* (B&B-NL) algorithm for a MI-convex problem solves a nonlinear subproblem that includes all of the convex constraints at every node of the search tree. The Bonmin solver package [Bonami et al., 2008] implements a B&B-NL variant by calling the derivative-oracle-based nonlinear programming (NLP) solver Ipopt to solve the subproblems. The relatively new SCIP-SDP [Gally et al., 2018] B&B-NL implementation for mixed-integer semidefinite (MISDP) problems uses a primal-dual conic interior-point solver for the SDP subproblems.

Typically, B&B-NL methods need to solve a large number of very similar nonlinear subproblems to near-global optimality and feasibility in order to obtain accurate objective bounds. Linear optimization (LP) solvers based on the Simplex algorithm are able to rapidly reoptimize after variable bounds are changed or linear cuts are added, thus typically benefiting from warm-starting much more so than state-of-the-art NLP or conic solvers. *B&B LP outer approximation* (B&B-OA) algorithms take advantage of LP warm-starting by solving a polyhedral relaxation, or LP outer approximation (LP OA), of the nonlinear subproblem at every node. Implementations often take advantage of the speed and stability of advanced MILP branch-and-cut solvers.

B&B-OA algorithms differ in how they refine the polyhedral relaxations of the nonlinear constraints and how they obtain feasible solutions. In a *separation-based* algorithm, no nonlinear solver is used. At each node the optimal point of the LP OA is first checked for feasibility for the convex constraints; if the violation exceeds a positive tolerance, valid cuts separating the point are added to the LP, otherwise the point may be accepted as a new incumbent if it is integral.[1] Quesada and Grossmann [1992] and Leyffer [1993] describe *subproblem-based* B&B-OA algorithms that solve smooth subproblems at a subset of the nodes.

[1]Commercial mixed-integer second-order cone optimization (MISOCP) solvers use separation-based algorithms, but also occasionally solve SOCP subproblems to obtain feasible solutions and fathom nodes. SCIP-SDP offers both B&B-NL and separation-based B&B-OA methods for mixed-integer semidefinite problems.

The subproblems provide points at which cuts based on gradient inequalities can be derived.[2]

Bonami et al. [2008] found that Bonmin's B&B-OA method generally outperforms its B&B-NL method. Since both of these methods rely on NLP subproblems, they frequently fail in the presence of nonsmoothness. Continuous conic solvers are more numerically robust than derivative-oracle-based NLP solvers on nonsmooth problems (such as SOCPs and SDPs). For the special case of MISOCP, Drewes and Ulbrich [2012] propose a conic subproblem-based B&B-OA algorithm that derives cuts from subgradients satisfying subproblem KKT optimality conditions, and hence does not require smoothness assumptions.

Another advantage of conic solvers is that they return simple *certificates* proving primal or dual infeasibility or optimality of a primal-dual solution pair [Permenter et al., 2015]. Using the theory of conic duality, it is possible to describe an elegant OA algorithm for generic MI-conic problems that uses conic certificates returned by primal-dual conic solvers, with no need to examine KKT conditions or solve a second modified subproblem in the case of infeasibility (as in the algorithm by Drewes and Ulbrich [2012]). Lubin et al. [2016, 2018] propose this idea in an iterative OA algorithm. However a B&B algorithm using a single search tree, instead of solving a sequence of MILP instances each with their own search tree, is more flexible and likely to be significantly faster in practice. We fill this gap with the first conic-certificate-based B&B-OA algorithm.

1.2 Mixed-Integer Conic Form

We use the following general form for a mixed-integer conic (MI-conic) problem:

$$\mathfrak{M} \begin{cases} \inf_{\boldsymbol{x} \in \mathbb{R}^N} \ \boldsymbol{c}^T \boldsymbol{x} : & \quad (2a) \\ \quad \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x} \in \mathcal{K} \subset \mathbb{R}^M & \quad (2b) \\ \quad\quad x_i \in \mathbb{Z} \qquad \forall i \in [\![I]\!], & \quad (2c) \end{cases}$$

where $\mathcal{K}$ is a *closed convex cone*, i.e. a closed subset of $\mathbb{R}^M$ that contains all conic (nonnegative) combinations of its points [Ben-Tal and Nemirovski, 2001a]:

$$\alpha_1 \boldsymbol{y}_1 + \alpha_2 \boldsymbol{y}_2 \in \mathcal{K} \qquad \forall \alpha_1, \alpha_2 \geq 0 \qquad \forall \boldsymbol{y}_1, \boldsymbol{y}_2 \in \mathcal{K}. \tag{3}$$

The decision variables in $\mathfrak{M}$ are represented by the column vector $\boldsymbol{x} \in \mathbb{R}^N$, so the objective (2a) minimizes a linear function of $\boldsymbol{x}$ subject to (denoted by ':') the constraints (2b) and (2c). The index set of integer decision variables is $[\![I]\!] = \{1, \ldots, I\}$, so the integrality constraints (2c) restrict only the first I variables $x_1, \ldots, x_I$ to the set of integers $\mathbb{Z}$. The conic constraint (2b), which restricts the affine transformation $\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}$ of $\boldsymbol{x}$ to $\mathcal{K}$, is a convex constraint, so

[2]For a convex function $f : \mathbb{R}^n \to \mathbb{R}$, the set $\mathcal{X} = \{\boldsymbol{x} \in \mathbb{R}^n : f(\boldsymbol{x}) \leq 0\}$ is convex. If f is smooth, then given a point $\bar{\boldsymbol{x}} \in \mathbb{R}^n$, the following *gradient cut* yields a polyhedral relaxation of $\mathcal{X}$:

$$f(\bar{\boldsymbol{x}}) + (\nabla f(\bar{\boldsymbol{x}}))^T (\boldsymbol{x} - \bar{\boldsymbol{x}}) \leq 0. \tag{1}$$

relaxing the integrality constraints (2c) results in a convex conic optimization problem.

Any MI-convex problem can be expressed in MI-conic form, by homogenizing the convex constraints, for example through perspective transformations [Boyd and Vandenberghe, 2004, Lubin et al., 2016]. *Disciplined Convex Programming* (DCP) modeling packages such as CVX [Grant and Boyd, 2014], CVXPy [Diamond and Boyd, 2016], and Convex.jl [Udell et al., 2014] perform conic transformations automatically, conveniently enabling modelers to access powerful conic solvers such as ECOS [Domahidi et al., 2013], SCS [O'Donoghue et al., 2016], MOSEK [Mosek ApS, 2016], and CSDP [Borchers, 1999].

Conic solvers recognize the cone $\mathcal{K}$ as a Cartesian product of standard *primitive* cones [Friberg, 2016]. A primitive closed convex cone cannot be written as a Cartesian product of two or more lower-dimensional closed convex cones. Lubin et al. [2018] claim that the following classes of standard primitive cones are extremely versatile, encoding all of the problems in the Conic Benchmark Library (CBLIB) compiled by Friberg [2016], and all 333 MI-convex problems in MINLPLIB2 [Vigerske, 2018].

Linear cones naturally express affine constraints; any mixed-integer linear optimization (MILP) problem can be written in $\mathfrak{M}$ form using nonnegative, nonpositive, and zero cones.

Second-order cones (and rotated-second-order cones) are widely used to model rational powers, norms, and geometric means, as well as convex quadratic objectives and constraints [Ben-Tal and Nemirovski, 2001a].

Positive semidefinite cones can model robust norms and functions of eigenvalues [Ben-Tal and Nemirovski, 2001a], and sum-of-squares constraints for polynomial optimization problems [Parrilo, 2003].

Exponential cones can model exponentials, logarithms, entropy, and powers, as well as log-sum-exp functions that arise from convex transformations of geometric programs [Serrano, 2015].

We note that conic representations are useful for constructing tight formulations for disjunctions or unions of convex sets [Lubin et al., 2017a,b, Vielma, 2018].

1.3 Overview And Contributions

In Section 2, we start by reviewing the relevant foundations of conic duality and certificates. We then introduce the notion of $\mathcal{K}^*$ *cuts*, and describe how to refine LP OAs of conic constraints using *certificate* $\mathcal{K}^*$ *cuts* obtained from certificates returned by continuous primal-dual conic solvers. For a MI-conic problem $\mathfrak{M}$, we propose the first B&B-OA algorithm based on conic certificates. We show that our algorithm detects infeasibility or unboundedness or terminates with an optimal solution in finite time under minimal assumptions.

In Section 3, we demonstrate that a $\mathcal{K}^*$ cut from a conic certificate implies useful guarantees about the infeasibility or optimal objective of an LP OA, suggesting that our algorithm can often fathom a node immediately after solving the LP rather than proceeding to the expensive conic subproblem solve. We consider how these guarantees may be lost in the more realistic setting of an LP solver with a positive feasibility tolerance, and propose a practical methodology for scaling a certificate $\mathcal{K}^*$ cut to recover similar guarantees.

In Section 4, we describe how to strengthen the LP OAs by *disaggregating* $\mathcal{K}^*$ cuts, and show that this methodology maintains the guarantees from Section 3. We argue for initializing the LP OAs using *initial fixed* $\mathcal{K}^*$ *cuts*, and offer a procedure for cheaply obtaining *separation* $\mathcal{K}^*$ *cuts* to cut off an infeasible LP OA solution. These proposed techniques require minimal modifications to our algorithm and are practical to implement. In Appendix A, we specialize these techniques for the second-order, positive semidefinite, and exponential cones. We note that the ideas from Sections 3 to 4 can be applied (with minor adjustments) to the iterative conic OA method described by Lubin et al. [2016].

In Section 5, we describe the software architecture and algorithmic implementation of Pajarito, our open source MI-convex solver.[3] This section may be of particular interest to advanced users and developers of mathematical optimization software. We emphasize that our implementations diverge from the idealized algorithmic description in Section 2, because of our decision to leverage powerful external mixed-integer linear (MILP) solvers through limited, solver-independent interfaces. In Appendix B, we describe how Pajarito lifts $\mathcal{K}^*$ cuts for the second-order cone using an *extended formulation*, resulting in tighter LP OAs. In Appendix C, we show how Pajarito can optionally tighten OAs for PSD cone constraints by strengthening $\mathcal{K}^*$ cuts to rotated second-order cone constraints, which can be added to an MISOCP OA model.

In Section 6, we summarize computational experiments demonstrating the speed and robustness of Pajarito. We benchmark Pajarito and several MISOCP solver packages, and conclude that Pajarito is the fastest and most-reliable open source solver for MISOCP. Finally, we compare the performance of several of Pajarito's algorithmic variants on MI-conic instances involving mixtures of positive semidefinite, second-order, and exponential cones, demonstrating practical advantages of several methodological contributions from Sections 3 and 4 and Appendix A.

2 A Branch-And-Bound LP Outer Approximation Algorithm

For a MI-conic problem $\mathfrak{M}$, we propose a branch-and-bound LP outer approximation (B&B-OA) algorithm, the first such method based on conic certificates.

[3]The new version of Pajarito that we implemented for this paper is the first conic-certificate-based OA solver. Although Pajarito solver was introduced in Lubin et al. [2016], this early implementation used NLP solvers instead of primal-dual conic solvers for continuous subproblems, and was built to assess the value of *extended formulations* by counting iterations before convergence. To avoid confusing users, we recently moved this old NLP-based functionality out of Pajarito and into *Pavito* solver at github.com/JuliaOpt/Pavito.jl.

In Section 2.1, we describe the continuous conic subproblems that a nonlinear branch-and-bound (B&B-NL) algorithm would solve at each node, and review the relevant foundations of conic duality. In Section 2.2, we introduce the notion of $\mathcal{K}^*$ cuts and describe how to refine polyhedral relaxations of the conic subproblems using information from conic certificates. Finally, we outline our B&B-OA algorithm in Section 2.3, and discuss finiteness of convergence. For the sake of exposition and analysis, this algorithm is quite minimal, and we postpone a description of optional enhancements to Sections 3 to 4. As we discuss in Section 5, the methods in Pajarito solver deviate from this minimal algorithm due to limitations of the external mixed-integer linear (MILP) solvers and interfaces we use in our implementations.

2.1 Continuous Subproblems And Conic Duality

Recall from $\mathfrak{M}$ that the first I variables in $\boldsymbol{x}$ are constrained to be integer. Branch-and-bound algorithms recursively partition the valid integer assignments, so for convenience we assume known finite lower bounds $\boldsymbol{l}^0 \in \mathbb{Z}^I$ and upper bounds $\boldsymbol{u}^0 \in \mathbb{Z}^I$ on the integer variables $x_1, \dots x_I$. At a node of the branch-and-bound search tree with lower bounds $\boldsymbol{l} \in \mathbb{Z}^I$ and upper bounds $\boldsymbol{u} \in \mathbb{Z}^I$ on $x_1, \dots, x_I$, the natural continuous conic subproblem is $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$:

$$\mathfrak{C}(\boldsymbol{l},\boldsymbol{u}) \left\{ \begin{aligned} \inf_{\boldsymbol{x}} \quad & \boldsymbol{c}^T \boldsymbol{x} : && && \text{(4a)} \\ & \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x} \in \mathcal{K} && && \text{(4b)} \\ & l_i - x_i \in \mathbb{R}_- && \forall i \in [\![I]\!] && \text{(4c)} \\ & u_i - x_i \in \mathbb{R}_+ && \forall i \in [\![I]\!] && \text{(4d)} \\ & \boldsymbol{x} \in \mathbb{R}^N, && && \text{(4e)} \end{aligned} \right.$$

where the bound constraints (4c) and (4d) are expressed in conic form using the nonpositive cone $\mathbb{R}_-$ (the nonpositive reals) and the nonnegative cone $\mathbb{R}_+$ (the nonnegative reals).

There exist primal-dual conic algorithms for $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$ that are powerful in both theory and practice. The foundation for these methods and for much of this paper is the elegant theory of conic duality, described by Ben-Tal and Nemirovski [2001a], Boyd and Vandenberghe [2004]. Recall that the cone $\mathcal{K}$ in $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$ is a closed convex cone; we let $\mathcal{K}^*$ denote the *dual cone* of $\mathcal{K}$, i.e. the set of points that have nonnegative inner product with all points in $\mathcal{K}$:

$$\mathcal{K}^* = \{\boldsymbol{z} \in \mathbb{R}^M : \boldsymbol{y}^T \boldsymbol{z} \geq 0, \forall \boldsymbol{y} \in \mathcal{K}\}. \tag{5}$$

$\mathcal{K}^*$ is also a closed convex cone [Boyd and Vandenberghe, 2004]. The standard conic dual of $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$ can be written as $\mathfrak{C}^*(\boldsymbol{l}, \boldsymbol{u})$:

$$\mathfrak{C}^*(\boldsymbol{l},\boldsymbol{u}) \begin{cases} \sup_{\boldsymbol{z},\boldsymbol{\mu},\boldsymbol{\nu}} \quad -\boldsymbol{b}^T\boldsymbol{z} - \boldsymbol{l}^T\boldsymbol{\mu} - \boldsymbol{u}^T\boldsymbol{\nu} : & \text{(6a)} \\ \boldsymbol{z} \in \mathcal{K}^* & \text{(6b)} \\ \boldsymbol{\mu} \in \mathbb{R}_-^I & \text{(6c)} \\ \boldsymbol{\nu} \in \mathbb{R}_+^I & \text{(6d)} \\ \boldsymbol{c} + \boldsymbol{A}^T\boldsymbol{z} + \boldsymbol{\mu}' + \boldsymbol{\nu}' \in \{0\}^N, & \text{(6e)} \end{cases}$$

where for ease of exposition we let $\boldsymbol{\mu}' = (\mu_1, \ldots, \mu_I, 0, \ldots, 0) \in \mathbb{R}^N$ and similarly for $\boldsymbol{\nu}'$. Note that the nonnegative and nonpositive cones are both *self-dual*, i.e. $\mathbb{R}_-^* = \mathbb{R}_-$ and $\mathbb{R}_+^* = \mathbb{R}_+$. The zero cone $\{0\}$ (containing only the origin) is dual to the free cone $\mathbb{R}$. The variables $\boldsymbol{z}$ in the dual constraint (6b) are associated with the primal constraint (4b), and similarly for (6c) and (4c), (6d) and (4d), and (4e) and (6e).

If the conic primal-dual pair $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$–$\mathfrak{C}^*(\boldsymbol{l}, \boldsymbol{u})$ is *well-posed*, then conic duality can be thought of as a simple generalization of LP duality.[4] In particular, the inf and sup can be replaced with min and max, and the possible status combinations for $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$ and $\mathfrak{C}^*(\boldsymbol{l}, \boldsymbol{u})$ are: both infeasible, one unbounded and the other infeasible, or both feasible and bounded with equal objective values attained by optimal solutions. The conditions for well-posedness in conic duality are described by Friberg [2016], and are outside the scope of this paper, so we assume that any primal-dual subproblem we encounter has the well-posed property.

Friberg [2016] discusses certificates that provide easily-verifiable proofs of unboundedness or infeasibility of the primal or dual problems or of optimality of a given pair of primal and dual points. In terms of the primal subproblem $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$, the three possible mutually-exclusive cases and their interpretations are as follows.

A dual improving ray certifies that $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$ is infeasible, via the conic generalization of Farkas' lemma. The improving ray $(\bar{\boldsymbol{z}}, \bar{\boldsymbol{\mu}}, \bar{\boldsymbol{\nu}}) \in \mathbb{R}^{M+2I}$ of $\mathfrak{C}^*(\boldsymbol{l}, \boldsymbol{u})$ is a feasible direction for $\mathfrak{C}^*(\boldsymbol{l}, \boldsymbol{u})$ along which the objective value of any feasible point of $\mathfrak{C}^*(\boldsymbol{l}, \boldsymbol{u})$ can be improved indefinitely. It satisfies the following conditions:

$$-\boldsymbol{b}^T\bar{\boldsymbol{z}} - \boldsymbol{l}^T\bar{\boldsymbol{\mu}} - \boldsymbol{u}^T\bar{\boldsymbol{\nu}} > 0 \tag{7a}$$
$$\bar{\boldsymbol{z}} \in \mathcal{K}^* \tag{7b}$$
$$\bar{\boldsymbol{\mu}} \leq \boldsymbol{0} \tag{7c}$$
$$\bar{\boldsymbol{\nu}} \geq \boldsymbol{0} \tag{7d}$$
$$\boldsymbol{A}^T\bar{\boldsymbol{z}} + \bar{\boldsymbol{\mu}}' + \bar{\boldsymbol{\nu}}' = \boldsymbol{0}. \tag{7e}$$

Clearly, if $\mathfrak{C}^*(\boldsymbol{l}, \boldsymbol{u})$ itself has a feasible point, then it is unbounded, otherwise it is infeasible. We note that conditions (7b) to (7e) imply that $(\bar{\boldsymbol{z}}, \bar{\boldsymbol{\mu}}, \bar{\boldsymbol{\nu}})$ is feasible for a modified $\mathfrak{C}^*(\boldsymbol{l}, \boldsymbol{u})$ problem in which $\boldsymbol{c} = \boldsymbol{0}$.

[4]If $\mathcal{K}$ is polyhedral, then $\mathcal{K}^*$ is polyhedral, and hence $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$ and $\mathfrak{C}^*(\boldsymbol{l}, \boldsymbol{u})$ are both LPs. All LPs are well-posed.

A primal improving ray and a primal feasible point certifies that $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$ is unbounded, because the improving ray is a feasible direction along which the objective value of the feasible point can be improved indefinitely. The improving ray $\bar{\boldsymbol{x}} \in \mathbb{R}^N$ of $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$ also implies infeasibility of $\mathfrak{C}^*(\boldsymbol{l}, \boldsymbol{u})$ and satisfies the following conditions:

$$\boldsymbol{c}^T \bar{\boldsymbol{x}} < 0 \tag{8a}$$

$$-\boldsymbol{A}\bar{\boldsymbol{x}} \in \mathcal{K} \tag{8b}$$

$$\bar{x}_i = 0 \qquad \forall i \in [\![I]\!], \tag{8c}$$

and the feasible point $\hat{\boldsymbol{x}} \in \mathbb{R}^N$ of $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$ simply satisfies the primal feasibility conditions (4b) to (4e). Note that if the objective coefficients of the continuous variables are all zero ($c_{I+1} = \ldots = c_N = 0$), then conditions (8a) and (8c) can never be satisfied, so there cannot be a primal improving ray. This matches intuition because if the continuous variables have zero objective coefficients and the integer variables are bounded, $\mathfrak{M}$ cannot be unbounded.

A complementary solution pair implies conic strong duality holds and certifies optimality for $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$ of the primal feasible point $\hat{\boldsymbol{x}} \in \mathbb{R}^N$ in the pair $(\hat{\boldsymbol{x}}, (\hat{\boldsymbol{z}}, \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\nu}}))$. The dual feasible point $(\hat{\boldsymbol{z}}, \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\nu}}) \in \mathbb{R}^{M+2I}$ is also optimal for $\mathfrak{C}^*(\boldsymbol{l}, \boldsymbol{u})$, and the pair have equal primal and dual objective values:

$$\boldsymbol{c}^T \hat{\boldsymbol{x}} = -\boldsymbol{b}^T \hat{\boldsymbol{z}} - \boldsymbol{l}^T \hat{\boldsymbol{\mu}} - \boldsymbol{u}^T \hat{\boldsymbol{\nu}}. \tag{9}$$

2.2 Dynamic Polyhedral Relaxations

Recall from equation (5) that $\boldsymbol{y} \in \mathcal{K}$ if and only if $\boldsymbol{z}^T \boldsymbol{y} \geq 0, \forall \boldsymbol{z} \in \mathcal{K}^*$. This implies that a nonpolyhedral conic constraint $\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x} \in \mathcal{K}$ has the following equivalent semi-infinite linear representation:

$$\boldsymbol{z}^T(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}) \geq 0 \qquad \forall \boldsymbol{z} \in \mathcal{K}^*. \tag{10}$$

We refer to a point $\boldsymbol{z} \in \mathcal{K}^*$ as a $\mathcal{K}^*$ *point*, and call the corresponding linear constraint $\boldsymbol{z}^T(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}) \geq 0$ a $\mathcal{K}^*$ *cut*. A $\mathcal{K}^*$ cut cannot exclude any point $\boldsymbol{x}$ that satisfies $\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x} \in \mathcal{K}$, so any finite set of $\mathcal{K}^*$ cuts defines a valid polyhedral relaxation of the conic constraint (4b).

Given a finite set $\mathcal{Z} \subset \mathcal{K}^*$ of $\mathcal{K}^*$ points, consider modifying the subproblem $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$ by relaxing the conic constraint and instead imposing the finite number of $\mathcal{K}^*$ cuts implied by $\mathcal{Z}$. We refer to the resulting LP as $\mathfrak{P}(\mathcal{Z}, \boldsymbol{l}, \boldsymbol{u})$, which we choose to write in inequality form rather than conic form:

$$\mathfrak{P}(\mathcal{Z}, \boldsymbol{l}, \boldsymbol{u}) \begin{cases} \min\limits_{\boldsymbol{x}} \quad \boldsymbol{c}^T \boldsymbol{x} : & \text{(11a)} \\ \qquad x_i \geq l_i \qquad \forall i \in [\![I]\!] & \text{(11b)} \\ \qquad x_i \leq u_i \qquad \forall i \in [\![I]\!] & \text{(11c)} \\ \boldsymbol{z}^T(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}) \geq 0 \qquad \forall \boldsymbol{z} \in \mathcal{Z}. & \text{(11d)} \end{cases}$$

Since $\mathfrak{P}(\mathcal{Z}, \boldsymbol{l}, \boldsymbol{u})$ and $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$ have the same objective function, and the feasible set of $\mathfrak{P}(\mathcal{Z}, \boldsymbol{l}, \boldsymbol{u})$ is a polyhedral relaxation of the feasible set of $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$, solving $\mathfrak{P}(\mathcal{Z}, \boldsymbol{l}, \boldsymbol{u})$ with an LP solver may give us useful information about $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$. If $\mathfrak{P}(\mathcal{Z}, \boldsymbol{l}, \boldsymbol{u})$ is infeasible, then $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$ must be infeasible. If $\mathfrak{P}(\mathcal{Z}, \boldsymbol{l}, \boldsymbol{u})$ has an optimal objective value of L, then $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$ is either infeasible or has an optimal objective no smaller than L. In these cases, we may be able to immediately fathom the node by infeasibility or by bound, or even use a fractional optimal solution for $\mathfrak{P}(\mathcal{Z}, \boldsymbol{l}, \boldsymbol{u})$ to make a branching decision, without needing to solve $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$. However, if $\mathfrak{P}(\mathcal{Z}, \boldsymbol{l}, \boldsymbol{u})$ is unbounded, it does not provide useful information about the status or optimal value of $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$.

After every infeasible or bounded conic subproblem solve, we add a new $\mathcal{K}^*$ cut obtained from the conic certificate found by the conic subproblem solver. Suppose that at some node, a primal-dual conic subproblem solver yields a dual improving ray $(\bar{\boldsymbol{z}}, \bar{\boldsymbol{\mu}}, \bar{\boldsymbol{\nu}})$: from condition (7b), $\bar{\boldsymbol{z}} \in \mathcal{K}^*$, so $\bar{\boldsymbol{z}}$ is a $\mathcal{K}^*$ point. Now suppose that the subproblem solver yields a complementary solution $(\hat{\boldsymbol{x}}, (\hat{\boldsymbol{z}}, \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\nu}}))$: by the dual feasibility condition (6b), $\hat{\boldsymbol{z}} \in \mathcal{K}^*$, so $\hat{\boldsymbol{z}}$ is a $\mathcal{K}^*$ point. In both cases, a subvector of the ray or solution for the dual subproblem $\mathfrak{C}^*(\boldsymbol{l}, \boldsymbol{u})$ allows us to augment $\mathcal{Z} \subset \mathcal{K}^*$, refining our LP outer approximation (LP OA) model $\mathfrak{P}(\mathcal{Z}, \boldsymbol{l}, \boldsymbol{u})$. In Section 3, we use conic duality theory to show that these certificate $\mathcal{K}^*$ cuts derived from conic subproblems encode important information about the subproblems into the subsequent polyhedral relaxations.

2.3 The Conic-Certificate-Based Algorithm

Our conic-certificate-based B&B-OA algorithm for the MI-conic problem $\mathfrak{M}$ is outlined in Algorithm 1. Recall that $\mathfrak{M}$ is in minimization form. Algorithm 1 maintains an upper bound U (initially ∞), a corresponding best feasible solution set $\mathcal{X}$ (initially empty), and a set of active nodes $\mathcal{N}$ of the search tree. A node $(\boldsymbol{l}, \boldsymbol{u}, L)$ is characterized by the finite variable bound vectors $\boldsymbol{l}$ and $\boldsymbol{u}$ and a lower bound value L. The node's lower bound L signifies that all feasible solutions for $\mathfrak{M}$ that satisfy the node's bounds on integer variables have an objective value of at least L. The node set $\mathcal{N}$ is initialized to contain only the root node $(\boldsymbol{l}^0, \boldsymbol{u}^0, -\infty)$, where $\boldsymbol{l}^0, \boldsymbol{u}^0 \in \mathbb{R}^I$ are the finite initial global bounds on the integer variables.

On Line 5, the main loop removes a node $(\boldsymbol{l}, \boldsymbol{u}, L)$ from $\mathcal{N}$. If the node's lower bound L is no smaller than the current global best upper bound U, Line 7 fathoms the node by bound as it cannot yield a better incumbent. Otherwise, Line 8 solves the node's LP OA model $\mathfrak{P}(\mathcal{Z}, \boldsymbol{l}, \boldsymbol{u})$, taking advantage of an LP warm-start from a previous node.

If $\mathfrak{P}(\mathcal{Z}, \boldsymbol{l}, \boldsymbol{u})$ is infeasible, Line 10 immediately fathoms the node by infeasibility. If $\mathfrak{P}(\mathcal{Z}, \boldsymbol{l}, \boldsymbol{u})$ has an optimal solution $\hat{\boldsymbol{x}}$, then its optimal objective value is the tightest lower bound known for $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$ (in Section 3.1.2, we prove $\boldsymbol{c}^T\hat{\boldsymbol{x}} \geq L$ is a consequence of the $\mathcal{K}^*$ cuts), so Line 12 updates L to $\boldsymbol{c}^T\hat{\boldsymbol{x}}$. Line 14 fathoms the node by bound if L is no better than the incumbent value U, otherwise if $\hat{\boldsymbol{x}}$ is fractional (i.e. it violates an integrality constraint (2c)), Line 17

Algorithm 1: Conic-certificate-based branch-and-bound LP outer approximation for $\mathfrak{M}$.

```
initialize incumbent solution set $\mathcal{X}$ to $\varnothing$, upper bound $U$ to $\infty$
initialize $\mathcal{K}^*$ point set $\mathcal{Z}$ to $\varnothing$
initialize node list $\mathcal{N}$ with root node $(\boldsymbol{l}^0, \boldsymbol{u}^0, -\infty)$
while $\mathcal{N}$ contains nodes do
    remove a node $(\boldsymbol{l}, \boldsymbol{u}, L)$ from $\mathcal{N}$
    if lower bound $L \geq U$ then
        continue                                          ▷ fathomed by bound
    call LP solver on $\mathfrak{P}(\mathcal{Z}, \boldsymbol{l}, \boldsymbol{u})$
    if get an infeasibility proof then
        continue                                          ▷ fathomed by infeasibility
    else if get an optimal solution $\hat{\boldsymbol{x}}$ then
        update $L$ to $\boldsymbol{c}^T \hat{\boldsymbol{x}}$
        if $L \geq U$ then
            continue                                      ▷ fathomed by bound
        else if $\hat{\boldsymbol{x}}$ is fractional then
            add branch nodes to $\mathcal{N}$ using $\hat{\boldsymbol{x}}$ and $L$
            continue                                      ▷ branched
    call primal-dual continuous conic solver on $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$
    if get a dual improving ray $(\bar{\boldsymbol{z}}, \bar{\boldsymbol{\mu}}, \bar{\boldsymbol{\nu}})$ then
        add $\mathcal{K}^*$ point $\bar{\boldsymbol{z}}$ to $\mathcal{Z}$
        continue                                          ▷ fathomed by infeasibility
    else if get a primal improving ray $\bar{\boldsymbol{x}}$ and feasible point $\hat{\boldsymbol{x}}$ then
        if $\hat{\boldsymbol{x}}$ is integral then
            update $U$ to $-\infty$
            break                                         ▷ proven unbounded
    else if get a complementary solution $(\hat{\boldsymbol{x}}, (\hat{\boldsymbol{z}}, \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\nu}}))$ then
        add $\mathcal{K}^*$ point $\hat{\boldsymbol{z}}$ to $\mathcal{Z}$
        update $L$ to $\boldsymbol{c}^T \hat{\boldsymbol{x}}$
        if $L \geq U$ then
            continue                                      ▷ fathomed by bound
        else if $\hat{\boldsymbol{x}}$ is integral then
            update $\mathcal{X}$ to $\{\hat{\boldsymbol{x}}\}$ and $U$ to $\boldsymbol{c}^T \hat{\boldsymbol{x}}$
            continue                                      ▷ fathomed by integrality
    add branch nodes to $\mathcal{N}$ using $\hat{\boldsymbol{x}}$ (fractional) and $L$
return $\mathcal{X}$, $U$
```

branches on it.[5] The branch procedure strictly partitions the node's integer bounds $\boldsymbol{l}$ and $\boldsymbol{u}$ by picking an $i \in [\![I]\!] : \hat{x}_i \notin \mathbb{Z}$ and adding two child nodes to $\mathcal{N}$: $(\boldsymbol{l}, (u_1, \dots, \lfloor \hat{x}_i \rfloor, \dots, u_N), L)$ and $((l_1, \dots, \lceil \hat{x}_i \rceil, \dots, l_N), \boldsymbol{u}, L)$.

If the node is not fathomed or branched on immediately after the LP solve (before Line 18), then $\mathfrak{P}(\mathcal{Z}, \boldsymbol{l}, \boldsymbol{u})$ is either unbounded or has an optimal solution $\hat{\boldsymbol{x}}$ that is integral (i.e. $\hat{x}_i \in \mathbb{Z}, \forall i \in [\![I]\!]$) with optimal value $\boldsymbol{c}^T \hat{\boldsymbol{x}} < U$. Then Line 18 solves the conic subproblem $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$ with the primal-dual continuous conic solver. Recall from Section 2.1 our assumption that the primal-dual subproblem pair $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$–$\mathfrak{C}^*(\boldsymbol{l}, \boldsymbol{u})$ is well-posed, so the conic solver returns one of the three possible certificates, which we handle as follows.

A dual improving ray on Line 19 provides a $\mathcal{K}^*$ point, which Line 20 adds to $\mathcal{Z}$ (as described in Section 2.2). This certificate proves that $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$ is infeasible, so Line 21 fathoms the node by infeasibility.

A primal improving ray and feasible point on Line 22 certifies that $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$ is unbounded. Since the primal improving ray conditions (8a) to (8c) are the same for any conic subproblem, every subproblem is infeasible or unbounded, so $\mathfrak{M}$ is either infeasible or unbounded. The incumbent solution set must be empty and $U = \infty$. Line 23 checks whether the feasible point $\hat{\boldsymbol{x}}$ is integral. If so, it is a feasible solution for $\mathfrak{M}$, so $\mathfrak{M}$ is unbounded and Line 25 terminates the main loop.

A complementary solution on Line 26 provides a $\mathcal{K}^*$ point that Line 27 adds to $\mathcal{Z}$ (as described in Section 2.2) and an optimal solution $\hat{\boldsymbol{x}}$ for $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$. The optimal objective value gives the tightest lower bound known for the node, so Line 28 updates L to $\boldsymbol{c}^T \hat{\boldsymbol{x}}$, and Line 30 fathoms by bound if this value is no better than U. Line 31 checks if $\hat{\boldsymbol{x}}$ is integral, in which case it becomes the new incumbent solution for $\mathfrak{M}$ on Line 32, and the node is fathomed by integrality on Line 33.

If the node is not fathomed immediately after the conic solve (before Line 34), then $\hat{\boldsymbol{x}}$ is a feasible solution for $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$ that is fractional. L is either ∞ (in the primal improving ray case) or finite (in the complementary solution case), and is the best known lower bound for the node. Line 34 branches on $\hat{\boldsymbol{x}}$ using the same branch procedure we describe above for Line 17.

Since the initial bounds on the integer variables are finite, and the main loop of Algorithm 1 either fathoms each node or strictly partitions its integer bounds or terminates the algorithm, it follows that Algorithm 1 terminates finitely. From the fact that $\mathfrak{P}(\mathcal{Z}, \boldsymbol{l}, \boldsymbol{u})$ is a valid polyhedral relaxation of $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$, and from the correctness of our inferences from the subproblem certificates, it is clear Algorithm 1 terminates correctly, under the assumption of well-posed conic subproblems. On Line 35, if $U = \infty$, then $\mathfrak{M}$ is proven infeasible, otherwise if U is finite, then $\mathcal{X}$ contains an optimal solution for $\mathfrak{M}$, otherwise $U = -\infty$ and $\mathfrak{M}$ is proven unbounded.

[5]We could instead remove lines 15-17 and solve the conic subproblem even if the LP solution is fractional, rather than branching. This variation may perform better if the conic subproblem solves are quite fast in practice.

For finite convergence, it is not necessary to solve conic subproblems for which $\boldsymbol{l} \neq \boldsymbol{u}$. However, Algorithm 1 solves a conic subproblem whenever the LP solution is integral (which can happen when $\boldsymbol{l} \neq \boldsymbol{u}$), because this may reduce the size of the branch and bound tree (an idea that has been exploited in the MINLP setting in Bonmin solver's *hybrid* algorithm [Bonami et al., 2008]).[6] We note that without using the LP $\mathfrak{P}(\mathcal{Z}, \boldsymbol{l}, \boldsymbol{u})$ (i.e. removing Lines 8 to 17 and not creating and augmenting $\mathcal{Z}$ on Lines 2, 20 and 27), we get a simple conic-certificate-based B&B-NL algorithm for $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$, for which finite termination guarantees and correctness follow from the same assumptions and arguments. We have omitted any discussion of node selection or fractional variable selection for branching. MILP solvers can use LP certificates to make intelligent selections, and we expect that some of these LP-based criteria are generalizeable to the conic case, as conic duality theory is a simple extension of LP duality under the well-posed assumption.

3 Polyhedral Relaxation Guarantees From Conic Certificates

Recall from Section 2.2 that $\mathcal{K}^*$ cuts yield valid polyhedral relaxations of the conic constraint $\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x} \in \mathcal{K}$, and a certificate $\mathcal{K}^*$ cut can be obtained directly from the conic certificate for an infeasible or bounded and feasible subproblem $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$. We demonstrate in Section 3.1 that a certificate $\mathcal{K}^*$ cut implies useful guarantees about the infeasibility or optimal objective of the LP OAs, suggesting that Algorithm 1 can often fathom a node immediately after solving the LP OA rather than proceeding to the expensive conic subproblem solve.[7] In Section 3.2, we consider how these guarantees may be lost in the more realistic setting of an LP solver with a positive feasibility tolerance, and propose a practical methodology for *scaling* a certificate $\mathcal{K}^*$ cut to recover similar guarantees.

3.1 Under An Exact LP Solver

We continue to assume well-posedness of every conic subproblem at every node. We consider what a certificate $\mathcal{K}^*$ cut from the conic subproblem $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$ at a node with bounds $\boldsymbol{l}, \boldsymbol{u}$ on the integer variables implies about the LP OA $\mathfrak{P}(\mathcal{Z}, \underset{\sim}{\boldsymbol{l}}, \underset{\sim}{\boldsymbol{u}})$ at a different node with bounds $\underset{\sim}{\boldsymbol{l}}, \underset{\sim}{\boldsymbol{u}}$.

[6]From personal communications with Zonghao Gu of Gurobi Optimization and Felipe Serrano of ZIB in 2017, we learned that both the Gurobi and SCIP MISOCP implementations solve SOCP restrictions where $\boldsymbol{l} \neq \boldsymbol{u}$ as heuristics to find feasible solutions. However, the dual information from these conic subproblem solves is discarded.

[7]By similar arguments, we expect that the certificate $\mathcal{K}^*$ cut may be useful at nearby nodes for duality-based preprocessing such as reduced cost fixing [Gally et al., 2018, sec. 7] or conflict analysis [Witzig et al., 2017].

3.1.1 Certificate Cuts From Dual Improving Rays

Suppose $(\bar{\boldsymbol{z}}, \bar{\boldsymbol{\mu}}, \bar{\boldsymbol{\nu}})$ is an improving ray of the dual subproblem $\mathfrak{C}^*(\boldsymbol{l}, \boldsymbol{u})$, certifying infeasibility of $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$. Using properties (7a) to (7e) of this certificate, any point $\boldsymbol{x} \in \mathbb{R}^N$ satisfying the bounds $\utilde{l}_i \leq x_i \leq \utilde{u}_i, \forall i \in \llbracket I \rrbracket$ at the new node also satisfies:

$$\begin{aligned}
\bar{\boldsymbol{z}}^T(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}) &= \boldsymbol{b}^T \bar{\boldsymbol{z}} - \boldsymbol{x}^T \boldsymbol{A}^T \bar{\boldsymbol{z}} && \text{(12a)} \\
&= \boldsymbol{b}^T \bar{\boldsymbol{z}} + \boldsymbol{x}^T \bar{\boldsymbol{\mu}}' + \boldsymbol{x}^T \bar{\boldsymbol{\nu}}' && \text{(12b)} \\
&\leq \boldsymbol{b}^T \bar{\boldsymbol{z}} + \boldsymbol{x}^T \bar{\boldsymbol{\mu}}' + \boldsymbol{x}^T \bar{\boldsymbol{\nu}}' + \sum_{i \in \llbracket I \rrbracket} ((\utilde{l}_i - x_i)\bar{\mu}_i + (\utilde{u}_i - x_i)\bar{\nu}_i) && \text{(12c)} \\
&= \boldsymbol{b}^T \bar{\boldsymbol{z}} + \utilde{\boldsymbol{l}}^T \bar{\boldsymbol{\mu}} + \utilde{\boldsymbol{u}}^T \bar{\boldsymbol{\nu}} && \text{(12d)} \\
&= (\boldsymbol{b}^T \bar{\boldsymbol{z}} + \boldsymbol{l}^T \bar{\boldsymbol{\mu}} + \boldsymbol{u}^T \bar{\boldsymbol{\nu}}) - (\boldsymbol{l} - \utilde{\boldsymbol{l}})^T \bar{\boldsymbol{\mu}} - (\boldsymbol{u} - \utilde{\boldsymbol{u}})^T \bar{\boldsymbol{\nu}}. && \text{(12e)}
\end{aligned}$$

From property (7a) of the certificate, $\boldsymbol{b}^T \bar{\boldsymbol{z}} + \boldsymbol{l}^T \bar{\boldsymbol{\mu}} + \boldsymbol{u}^T \bar{\boldsymbol{\nu}} < 0$. If $l_i \leq \utilde{l}_i \leq \utilde{u}_i \leq u_i, \forall i \in \llbracket I \rrbracket$, then $(\boldsymbol{l} - \utilde{\boldsymbol{l}})^T \bar{\boldsymbol{\mu}} \geq 0$ and $(\boldsymbol{u} - \utilde{\boldsymbol{u}})^T \bar{\boldsymbol{\nu}} \geq 0$. In this case, the value (12e) is negative, so from (12a) to (12e), the certificate $\mathcal{K}^*$ cut $\bar{\boldsymbol{z}}^T(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}) \geq 0$ is violated. Therefore, the certificate $\mathcal{K}^*$ cut from the infeasible subproblem $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$ guarantees infeasibility of any LP OA $\mathfrak{P}(\mathcal{Z}, \utilde{\boldsymbol{l}}, \utilde{\boldsymbol{u}})$ in the subtree of the node with bounds $\boldsymbol{l}, \boldsymbol{u}$.

More importantly for Algorithm 1, the certificate $\mathcal{K}^*$ cut is likely to remain violated at 'nearby' nodes outside of this subtree, as the conditions (12a) to (12e) have a natural interpretation from global sensitivity analysis. Perturbing the bounds on the integer variables from $\boldsymbol{l}, \boldsymbol{u}$ to $\utilde{\boldsymbol{l}}, \utilde{\boldsymbol{u}}$ changes the upper bound on $\bar{\boldsymbol{z}}^T(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x})$ through a linear dependence on the values $\boldsymbol{\mu} \leq 0$ and $\boldsymbol{\nu} \geq 0$ of the dual variables in the improving ray of $\mathfrak{C}^*(\boldsymbol{l}, \boldsymbol{u})$.

3.1.2 Certificate Cuts From Dual Optimal Solutions

Suppose $(\hat{\boldsymbol{x}}, (\hat{\boldsymbol{z}}, \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\nu}}))$ is a complementary solution pair for the subproblem $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$, certifying optimality of the solution pair. Using the strong duality conditions (property (9) and feasibility for $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$ and $\mathfrak{C}^*(\boldsymbol{l}, \boldsymbol{u})$), any point $\boldsymbol{x} \in \mathbb{R}^N$ satisfying the bounds $\utilde{l}_i \leq x_i \leq \utilde{u}_i, \forall i \in \llbracket I \rrbracket$ at the new node and the certificate

$\mathcal{K}^*$ cut $\hat{\boldsymbol{z}}^T(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}) \geq 0$ has objective value:

$$
\begin{align}
\boldsymbol{c}^T\boldsymbol{x} &= -(\boldsymbol{A}^T\hat{\boldsymbol{z}} + \hat{\boldsymbol{\mu}}' + \hat{\boldsymbol{\nu}}')^T\boldsymbol{x} \tag{13a}\\
&= -\hat{\boldsymbol{z}}^T\boldsymbol{A}\boldsymbol{x} - \boldsymbol{x}^T(\hat{\boldsymbol{\mu}}' + \hat{\boldsymbol{\nu}}') \tag{13b}\\
&= -\boldsymbol{b}^T\hat{\boldsymbol{z}} + \hat{\boldsymbol{z}}^T(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x})^T - \boldsymbol{x}^T(\hat{\boldsymbol{\mu}}' + \hat{\boldsymbol{\nu}}') \tag{13c}\\
&\geq -\boldsymbol{b}^T\hat{\boldsymbol{z}} - \boldsymbol{x}^T(\hat{\boldsymbol{\mu}}' + \hat{\boldsymbol{\nu}}') \tag{13d}\\
&\geq -\boldsymbol{b}^T\hat{\boldsymbol{z}} - \boldsymbol{x}^T(\hat{\boldsymbol{\mu}}' + \hat{\boldsymbol{\nu}}') - \sum_{i \in [\![I]\!]} ((\utilde{l}_i - x_i)\hat{\mu}_i + (\utilde{u}_i - x_i)\hat{\nu}_i) \tag{13e}\\
&= -\boldsymbol{b}^T\hat{\boldsymbol{z}} - \utilde{\boldsymbol{l}}^T\hat{\boldsymbol{\mu}} - \utilde{\boldsymbol{u}}^T\hat{\boldsymbol{\nu}} \tag{13f}\\
&= (-\boldsymbol{b}^T\hat{\boldsymbol{z}} - \boldsymbol{l}^T\hat{\boldsymbol{\mu}} - \boldsymbol{u}^T\hat{\boldsymbol{\nu}}) + (\boldsymbol{l} - \utilde{\boldsymbol{l}})^T\hat{\boldsymbol{\mu}} + (\boldsymbol{u} - \utilde{\boldsymbol{u}})^T\hat{\boldsymbol{\nu}} \tag{13g}\\
&= \boldsymbol{c}^T\hat{\boldsymbol{x}} + (\boldsymbol{l} - \utilde{\boldsymbol{l}})^T\hat{\boldsymbol{\mu}} + (\boldsymbol{u} - \utilde{\boldsymbol{u}})^T\hat{\boldsymbol{\nu}}. \tag{13h}
\end{align}
$$

If $l_i \leq \utilde{l}_i \leq \utilde{u}_i \leq u_i, \forall i \in [\![I]\!]$, then $(\boldsymbol{l} - \utilde{\boldsymbol{l}})^T\hat{\boldsymbol{\mu}} \geq 0$ and $(\boldsymbol{u} - \utilde{\boldsymbol{u}})^T\hat{\boldsymbol{\nu}} \geq 0$. In this case, the value (13h) is no smaller than $\boldsymbol{c}^T\hat{\boldsymbol{x}}$, the lower bound from the subproblem $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$. Therefore, the certificate $\mathcal{K}^*$ cut from the feasible subproblem $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$ guarantees that the optimal value of any LP OA $\mathfrak{P}(\mathcal{Z}, \utilde{\boldsymbol{l}}, \utilde{\boldsymbol{u}})$ in the subtree of the node with bounds $\boldsymbol{l}, \boldsymbol{u}$ on the integer variables does not decrease, but may actually improve.[8]

More importantly for Algorithm 1, at 'nearby' nodes outside of this subtree, the objective bounds implied by the certificate $\mathcal{K}^*$ cut in the LP OA model are likely to remain fairly tight. Perturbing the bounds on the integer variables from $\boldsymbol{l}, \boldsymbol{u}$ to $\utilde{\boldsymbol{l}}, \utilde{\boldsymbol{u}}$ changes the lower bound on $\boldsymbol{c}^T\boldsymbol{x}$ through a linear dependence on the values $\boldsymbol{\mu} \leq 0$ and $\boldsymbol{\nu} \geq 0$ of the dual variables in the complementary solution pair for $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$.

3.2 Under An LP Solver With A Feasibility Tolerance

So far, we have been assuming that the LP solver computes a solution that satisfies all the $\mathcal{K}^*$ cuts in the LP OAs exactly. In practice, LP solvers based on the Simplex method (except those that use rational arithmetic) enforce constraints up to an absolute constraint-wise violation tolerance $\delta > 0$ (typically set by the user). Therefore, a more realistic assumption is that any solution returned by the LP solver does not violate any $\mathcal{K}^*$ cut by more than δ, i.e. a $\mathcal{K}^*$ point $\boldsymbol{z}$ effectively yields a 'relaxed $\mathcal{K}^*$ cut' $\boldsymbol{z}^T(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}) \geq -\delta$. Under this relaxed condition, we may lose the 'within-subtree' guarantees described in Section 3.1. However, noting that any positive scaling of a $\mathcal{K}^*$ point is still a $\mathcal{K}^*$ point, we demonstrate how to recover the infeasibility guarantee from Section 3.1.1 exactly, and the objective bound guarantee from Section 3.1.2 to within a given relative objective gap tolerance. Such an analysis appears to be novel in the MI-convex literature.

[8]If Algorithm 1 branches on Line 34 after solving a bounded and feasible conic subproblem to get the tightest lower bound, then when examining a child node, this objective guarantee ensures the node's lower bound L does not decrease when we update it to the optimal value of the LP OA on Line 12.

3.2.1 Certificate Cuts From Dual Improving Rays

Suppose $(\bar{\boldsymbol{z}}, \bar{\boldsymbol{\mu}}, \bar{\boldsymbol{\nu}})$ is an improving ray of the dual subproblem $\mathfrak{C}^*(\boldsymbol{l}, \boldsymbol{u})$. From the property (7a) of the certificate and the conditions (12a) to (12e), any point $\boldsymbol{x} \in \mathbb{R}^N$ satisfying the bounds $l_i \leq x_i \leq u_i, \forall i \in [\![I]\!]$ and the relaxed certificate $\mathcal{K}^*$ cut condition $\bar{\boldsymbol{z}}^T(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}) \geq -\delta$ must satisfy:

$$0 > \boldsymbol{b}^T\bar{\boldsymbol{z}} + \boldsymbol{l}^T\bar{\boldsymbol{\mu}} + \boldsymbol{u}^T\bar{\boldsymbol{\nu}} \geq \bar{\boldsymbol{z}}^T(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}) \geq -\delta. \tag{14}$$

Therefore, if $\delta > 0$ is sufficiently large, the relaxed certificate $\mathcal{K}^*$ cut condition fails to enforce the infeasibility guarantee from Section 3.1.1.

However, for a positive multiplier $\bar{\gamma} > 0$ satisfying:

$$\bar{\gamma} > \frac{\delta}{-\boldsymbol{b}^T\bar{\boldsymbol{z}} - \boldsymbol{l}^T\bar{\boldsymbol{\mu}} - \boldsymbol{u}^T\bar{\boldsymbol{\nu}}} > 0, \tag{15}$$

we have $\bar{\gamma}(\boldsymbol{b}^T\bar{\boldsymbol{z}} + \boldsymbol{l}^T\bar{\boldsymbol{\mu}} + \boldsymbol{u}^T\bar{\boldsymbol{\nu}}) < -\delta$. Therefore, the relaxed *scaled* certificate $\mathcal{K}^*$ cut condition $\bar{\gamma}\bar{\boldsymbol{z}}^T(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}) \geq -\delta$ recovers the infeasibility guarantee within the subtree of the node from which the certificate is obtained. Note that the scaling factor (15) depends only on δ, problem data, and the certificate for the infeasible subproblem $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$. We can modify Algorithm 1 on Line 20 to add the scaled $\mathcal{K}^*$ point $\bar{\gamma}\bar{\boldsymbol{z}}$ to $\mathcal{Z}$.

3.2.2 Certificate Cuts From Dual Optimal Solutions

Suppose $(\hat{\boldsymbol{x}}, (\hat{\boldsymbol{z}}, \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\nu}}))$ is a complementary solution pair for the subproblem $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$. From the conditions (13a) to (13h), any point $\boldsymbol{x} \in \mathbb{R}^N$ satisfying the bounds $l_i \leq x_i \leq u_i, \forall i \in [\![I]\!]$ and the relaxed certificate $\mathcal{K}^*$ cut condition $\hat{\boldsymbol{z}}^T(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}) \geq -\delta$ has objective value:

$$\boldsymbol{c}^T\boldsymbol{x} \geq -\boldsymbol{b}^T\hat{\boldsymbol{z}} + \hat{\boldsymbol{z}}^T(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}) - \boldsymbol{l}^T\hat{\boldsymbol{\mu}} - \boldsymbol{u}^T\hat{\boldsymbol{\nu}} \geq L - \delta. \tag{16}$$

Recall $L = \boldsymbol{c}^T\hat{\boldsymbol{x}} = -\boldsymbol{b}^T\hat{\boldsymbol{z}} - \boldsymbol{l}^T\hat{\boldsymbol{\mu}} - \boldsymbol{u}^T\hat{\boldsymbol{\nu}}$ is the optimal objective value of $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$ and $\mathfrak{C}^*(\boldsymbol{l}, \boldsymbol{u})$. Therefore, the relaxed certificate $\mathcal{K}^*$ cut condition only enforces the objective guarantee from Section 3.1.2 to an absolute tolerance of δ. In general, it makes little sense for an objective guarantee to depend on the the LP solver's feasibility tolerance.

Instead, for a relative optimality gap tolerance $\epsilon > 0$, we can easily motivate a relative objective gap condition such as:

$$\frac{L - \boldsymbol{c}^T\boldsymbol{x}}{|L| + \theta} \leq \epsilon, \tag{17}$$

where θ is a small positive value (e.g. 10^{-5}) that is used to avoid division by zero. Consider a positive multiplier $\hat{\gamma} > 0$ satisfying:

$$\hat{\gamma} \geq \frac{\delta}{\epsilon(|L| + \theta)} > 0. \tag{18}$$

Modifying the conditions (16) for the relaxed *scaled* certificate $\mathcal{K}^*$ cut condition $\hat{\gamma}\hat{\boldsymbol{z}}^T(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}) \geq -\delta$, we get $\boldsymbol{c}^T\boldsymbol{x} \geq L - \delta/\hat{\gamma}$. Rearranging, this implies:

$$\frac{L - \boldsymbol{c}^T\boldsymbol{x}}{|L| + \theta} \leq \frac{\delta}{\hat{\gamma}(|L| + \theta)} \leq \epsilon, \tag{19}$$

so by scaling the certificate $\mathcal{K}^*$ cut by $\hat{\gamma}$, we achieve the relative objective gap guarantee (17) within the subtree of the node from which the certificate is obtained. Note that the scaling factor (18) depends only on ϵ, δ, problem data, and the certificate for the bounded and feasible subproblem $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$. We can modify Algorithm 1 on Line 27 to add the scaled $\mathcal{K}^*$ point $\hat{\gamma}\hat{\boldsymbol{z}}$ to $\mathcal{Z}$.

4 Tightening Polyhedral Relaxations

In Section 4.1, we outline a two-stage procedure for *disaggregating* $\mathcal{K}^*$ cuts to get stronger polyhedral relaxations, and show how to maintain the certificate $\mathcal{K}^*$ cut guarantees from Section 3. In Section 4.2, we argue for initializing the polyhedral relaxations using *initial fixed* $\mathcal{K}^*$ *cuts*, and in Section 4.3, we describe a procedure for cheaply obtaining *separation* $\mathcal{K}^*$ *cuts* to cut off an infeasible LP OA solution. All of our proposed techniques for tightening the LP OAs require minimal modifications to Algorithm 1 and are practical to implement.

4.1 Extreme Ray Disaggregation

Consider a set of $\mathcal{K}^*$ points $\mathcal{Z} = \{\boldsymbol{z}^1, \ldots, \boldsymbol{z}^J\} \subset \mathcal{K}^*$. By *aggregating* the corresponding $\mathcal{K}^*$ cuts, we see they imply infinitely many $\mathcal{K}^*$ cuts:

$$\boldsymbol{z}^T(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}) \geq 0 \qquad \forall \boldsymbol{z} \in \operatorname{cone}(\mathcal{Z}), \tag{20}$$

where $\operatorname{cone}(\mathcal{Z})$ is the *conic hull* of $\mathcal{Z}$, i.e. the set of conic (nonnegative) combinations of $\boldsymbol{z}^1, \ldots, \boldsymbol{z}^J$:

$$\operatorname{cone}(\mathcal{Z}) = \{\alpha^1\boldsymbol{z}^1 + \cdots \alpha^J\boldsymbol{z}^J : \alpha^1, \ldots, \alpha^j \geq 0\} \subset \mathcal{K}^*. \tag{21}$$

Thus for a redundant $\mathcal{K}^*$ point $\boldsymbol{z}^{J+1} \in \operatorname{cone}(\mathcal{Z})$, the polyhedral relaxation of the conic constraint $\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x} \in \mathcal{K}$ implied by $\mathcal{Z} \cup \{\boldsymbol{z}^{J+1}\}$ is no stronger than that implied by $\mathcal{Z}$ alone. An *extreme ray* of $\mathcal{K}^*$ is a point $\boldsymbol{z} \in \mathcal{K}^*$ that cannot be written as a nontrivial conic combination of other points in $\mathcal{K}^*$ that are not positive rescalings of $\boldsymbol{z}$. To maximize the efficiency of our polyhedral relaxations, we propose adding only extreme rays of $\mathcal{K}^*$ to the $\mathcal{K}^*$ point set $\mathcal{Z}$ maintained by Algorithm 1.

Recall from Section 1.2 that our closed convex cone $\mathcal{K}$ is encoded as a Cartesian product $\mathcal{K} = \mathcal{K}_1 \times \cdots \times \mathcal{K}_K$ of standard primitive cones $\mathcal{K}_1, \ldots, \mathcal{K}_K$ (e.g. nonnegative, second-order, exponential, and positive semidefinite cones). A primitive closed convex cone cannot be written as a Cartesian product of two or more lower-dimensional closed convex cones [Friberg, 2016]. If $\mathcal{K}$ is separable, then its dual cone $\mathcal{K}^*$ is also separable:

$$\mathcal{K}^* = (\mathcal{K}_1 \times \cdots \times \mathcal{K}_K)^* = \mathcal{K}_1^* \times \cdots \times \mathcal{K}_K^*. \tag{22}$$

We exploit this separability and our understanding of the structure of the standard primitive cones to disaggregate a $\mathcal{K}^*$ point $\boldsymbol{z}$ into extreme rays of $\mathcal{K}^*$.

First, we note that $\boldsymbol{z} = (\tilde{\boldsymbol{z}}^1, \dots, \tilde{\boldsymbol{z}}^K) \in \mathcal{K}^*$, where $\tilde{\boldsymbol{z}}^k \in \mathcal{K}_k^*, \forall k \in [\![K]\!]$. Second, for each $k \in [\![K]\!]$, we disaggregate $\tilde{\boldsymbol{z}}^k$ into extreme rays of the primitive standard dual cone $\mathcal{K}_k^*$. This step is trivial for linear cones. For second-order, positive semidefinite, and exponential cones, we describe practical computational procedures for dual disaggregation in Appendix A.[9] We have $\tilde{\boldsymbol{z}}^k = \sum_{j \in [\![J_k]\!]} \tilde{\boldsymbol{z}}^{k,j}$, where $\tilde{\boldsymbol{z}}^{k,j} \neq 0$ is an extreme ray of $\mathcal{K}_k^*$, for all $j \in [\![J_k]\!]$. We choose these extreme rays so that none is a positive scaling of another, and J_k does not exceed $\dim(\mathcal{K}_k^*)$. Note that $J_k = 0$ if $\tilde{\boldsymbol{z}}^k = \boldsymbol{0}$.

For some $k \in [\![K]\!]$ and $j \in [\![J_k]\!]$, consider the point $\boldsymbol{z}^{k,j} = (0, \dots, 0, \tilde{\boldsymbol{z}}^{k,j}, 0, \dots, 0)$, which is nonzero only on the elements corresponding to the kth primitive dual cone. Since any cone contains the origin $\boldsymbol{0}$, and $\tilde{\boldsymbol{z}}^{k,j} \in \mathcal{K}_k^*$, $\boldsymbol{z}^{k,j} \in \mathcal{K}^*$ by equation (22). Furthermore, since $\tilde{\boldsymbol{z}}^{k,j}$ is an extreme ray of $\mathcal{K}_k^*$, it cannot be written as a nontrivial sum of extreme rays of $\mathcal{K}_k^*$, and so $\boldsymbol{z}^{k,j}$ cannot be written as a nontrivial sum of extreme rays of $\mathcal{K}^*$. Thus $\boldsymbol{z}^{k,j}$ is an extreme ray of $\mathcal{K}^*$.

Our two-stage disaggregation procedure for $\boldsymbol{z} \in \mathcal{K}^*$ yields $\sum_{k \in [\![K]\!]} J_k \leq \dim(\mathcal{K}) = M$ extreme rays of $\mathcal{K}^*$:

$$\boldsymbol{z} = \sum_{k \in [\![K]\!]} \sum_{j \in [\![J_k]\!]} \boldsymbol{z}^{k,j}. \tag{23}$$

Besides adding potentially multiple $\mathcal{K}^*$ points to $\mathcal{Z}$, no modifications are needed to the description of Algorithm 1. Since $\boldsymbol{z}$ is clearly contained in the conic hull of these $\mathcal{K}^*$ points, there is no loss of strength in the polyhedral relaxations, so the certificate $\mathcal{K}^*$ guarantees from Section 3.1 are maintained. The polyhedral relaxations are potentially much tighter, improving the power of the LP OA for fathoming a node by infeasibility or objective bound without proceeding to an expensive conic subproblem solve.[10]

We can also recover the guarantees from Section 3.2 for an LP solver with a feasibility tolerance $\delta > 0$. We assume $\boldsymbol{z}$ is a certificate $\mathcal{K}^*$ point that has already been scaled according to Section 3.2. After disaggregating $\boldsymbol{z}$, we scale each extreme ray up by $J = \sum_{k \in [\![K]\!]} J_k$ before adding it to $\mathcal{Z}$. The J relaxed scaled disaggregated $\mathcal{K}^*$ cut conditions are:

$$(J\boldsymbol{z}^{k,j})^T(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}) \geq -\delta \qquad \forall k \in [\![K]\!], j \in [\![J_k]\!]. \tag{24}$$

Summing and using equation (23), and dividing by J, we see that these conditions imply the relaxed scaled original $\mathcal{K}^*$ cut condition $\boldsymbol{z}^T(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}) \geq -\delta$.

[9]For example, if $\mathcal{K}_k$ is a positive semidefinite cone, we disaggregate $\tilde{\boldsymbol{z}}^k \in \mathcal{K}_k^*$ by performing an eigendecomposition on it; see Appendix A.3.2.

[10]The LP solver may need to deal with more cuts at nodes visited early in the search tree, but is ultimately likely to need to examine fewer nodes overall and solve fewer expensive conic subproblems, so the tradeoff can be worthwhile.

4.2 Initial Fixed Polyhedral Relaxations

We can modify Algorithm 1 on Line 2 to initialize a nonempty set $\mathcal{Z}$ of *initial fixed* $\mathcal{K}^*$ extreme rays that are not derived from subproblem certificates, but depend only on the geometry of $\mathcal{K}^*$. If $\mathcal{K}$ is a separable product of standard primitive cones, we can obtain initial fixed $\mathcal{K}^*$ extreme rays by treating each primitive cone constraint separately. In particular, a linear cone constraint need not be relaxed at all, since it is equivalent to one $\mathcal{K}^*$ cut (for a nonnegative or nonpositive cone) or two $\mathcal{K}^*$ cuts (for the zero cone). In Appendix A, we describe simple sets of initial fixed $\mathcal{K}^*$ extreme rays for second-order, positive semidefinite, or exponential primitive cones.[11] We show in Appendix A how knowledge of the initial fixed $\mathcal{K}^*$ extreme rays allows us to tailor our extreme ray disaggregation procedures from Section 4.1 for certificate $\mathcal{K}^*$ points to further increase the strength of the polyhedral relaxations and reduce redundancy in $\mathcal{Z}$.[12]

4.3 Separation Of Infeasible Points

Inspired by separation-based OA algorithms, we can modify Algorithm 1 on Line 16 to add *separation* $\mathcal{K}^*$ *points* to $\mathcal{Z}$ that cut off a fractional optimal LP solution $\hat{\boldsymbol{x}}$ that violates the conic constraint, right before branching on $\hat{\boldsymbol{x}}$. We show that a separation $\mathcal{K}^*$ point exists when $\boldsymbol{b} - \boldsymbol{A}\hat{\boldsymbol{x}} \notin \mathcal{K}$. Since $\mathcal{K}$ is closed and convex, there exists a hyperplane $(\boldsymbol{z}, \theta)$ that separates $\hat{\boldsymbol{y}} = \boldsymbol{b} - \boldsymbol{A}\hat{\boldsymbol{x}}$ from $\mathcal{K}$, i.e. $\boldsymbol{z}^T\hat{\boldsymbol{y}} < \theta$ and $\boldsymbol{z}^T\boldsymbol{y} \geq \theta, \forall \boldsymbol{y} \in \mathcal{K}$. Since the problem $\inf_{\boldsymbol{y}\in\mathcal{K}} \boldsymbol{z}^T\boldsymbol{y}$ is homogeneous (as $\mathcal{K}$ is a cone) and the optimal value is bounded below by finite θ, the optimal value must equal zero. So $\theta \leq 0$, implying $\boldsymbol{z}^T\hat{\boldsymbol{y}} < 0$ and $\boldsymbol{z}^T\boldsymbol{y} \geq 0, \forall \boldsymbol{y} \in \mathcal{K}$. Thus $\boldsymbol{z} \in \mathcal{K}^*$ (by definition (5) of $\mathcal{K}^*$), and it implies a $\mathcal{K}^*$ cut that separates $\hat{\boldsymbol{x}}$ from the feasible set of the conic constraint.

A separation $\mathcal{K}^*$ point may fail to improve the objective lower bound from the LP OA, and does not in general possess the sort of guarantees from Section 3 that a certificate $\mathcal{K}^*$ point implies. However, deriving a separation $\mathcal{K}^*$ point can be much cheaper than solving a continuous conic subproblem. If $\mathcal{K}$ is a separable product of standard primitive cones, we can obtain separation $\mathcal{K}^*$ extreme rays easily by treating each primitive cone constraint separately. In Appendix A, we describe practical computational methods for obtaining separation $\mathcal{K}^*$ extreme rays for primitive conic constraints involving second-order, positive semidefinite, or exponential cones.[13]

[11] For example, for a positive semidefinite cone, we use the extreme rays of the polyhedral cone of diagonally dominant symmetric matrices as initial fixed $\mathcal{K}^*$ extreme rays; see Appendix A.3.1.

[12] However, to be able to recover the guarantees from Section 3.2 under an LP solver with a feasibility tolerance, we would need the ability to dynamically scale up the initial fixed $\mathcal{K}^*$ points.

[13] For example, we obtain separation $\mathcal{K}^*$ extreme rays for a point that violates a positive semidefinite cone constraint by performing an eigendecomposition on it; see Appendix A.3.3.

5 Pajarito Solver And Related Software

We describe the software architecture and algorithmic implementation of Pajarito, our open source MI-convex solver. This section may be of particular interest to advanced users and developers of mathematical optimization software. In Pajarito's readme file (github.com/JuliaOpt/Pajarito.jl) we provide more guidance on the recommended ways of using the solver, as well as default options and tolerances. We emphasize that our implementations diverge from the idealized description of Algorithm 1 in Section 2.3, because of our decision to leverage powerful external mixed-integer linear (MILP/MIP) solvers through limited, solver-independent interfaces. Developers of MI-conic software with low-level control of the MIP search tree are able to implement features of Algorithm 1 that we are not capable of in Pajarito.

5.1 Julia And MathProgBase

Pajarito is the first MI-convex solver written in the relatively young Julia language [Bezanson et al., 2017]. MI-convex solvers such as α-ECP, Artelys Knitro, Bonmin, DICOPT, FilMINT, MINLP_BB, and SBB, which are reviewed by Bonami et al. [2012], are to our knowledge written in C, C++, or Fortran. Julia is a high-level programming language that can match the performance of these lower-level languages for writing solvers with much less boilerplate code [Lubin and Dunning, 2015]. Pajarito's compact codebase is thoroughly commented, and conveniently reusable and extensible by other researchers. We implement an extensive testing infrastructure with hundreds of unit tests. Since Pajarito's first release, several other MINLP solvers have been written in Julia and are available through MathProgBase, such as POD [Nagarajan et al., 2017], Juniper [Kröger et al., 2018], and Katana.[14]

Pajarito is integrated with the powerful MathProgBase abstraction layer. MathProgBase is a standardized API in Julia for interacting with optimization solvers, designed in part to allow the user to write solver-independent code.[15] The breadth of problem classes covered by MathProgBase is described at juliaopt.org and distinguishes it from similar abstraction layers such as OSI [Saltzman et al., 2004], a COIN-OR library in C++. It includes specifications for continuous and mixed-integer solvers that use linear/quadratic, conic, or oracle-based NLP (nonlinear programming) forms.

In Section 5.2, we describe accessing Pajarito through MathProgBase's conic interface (see top of Figure 1). The user specifies external MIP and continuous primal-dual conic solvers (including solver options) from the available solvers (i.e., those accessible though MathProgBase) and passes each solver object as an option into a function that creates a Pajarito solver object. In Section 5.3, we summarize Pajarito's main algorithmic implementations. Pajarito uses the mod-

[14]See github.com/lanl-ansi/POD.jl, github.com/lanl-ansi/Juniper.jl, and github.com/lanl-ansi/Katana.jl.

[15]MathProgBase is being replaced by a redesigned API, MathOptInterface. The process of building Pajarito has motivated many of the planned improvements in MathOptInterface.

eling package JuMP to conveniently build and manage the external MIP solver's OA model. JuMP itself interacts with the MIP solver via MathProgBase's linear/quadratic interface (see bottom right of Figure 1). To solve a continuous conic subproblem for a conic certificate, Pajarito calls the external primal-dual conic solver through the conic interface (see bottom left of Figure 1).[16]

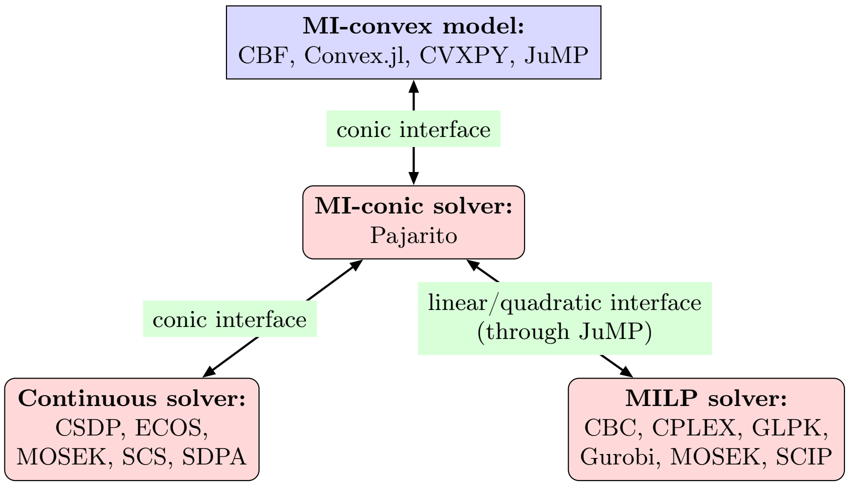


Figure 1: Pajarito's integration with MathProgBase.

5.2 Accessing Pajarito

Pajarito's use of conic form is a significant architectural difference from most existing MI-convex solvers, which interact with a MI-convex instance almost exclusively through oracles to query values and derivatives of the constraint and objective functions. MathProgBase conic form can be described compactly from a constraint matrix in sparse or dense format, right-hand side and objective coefficient vectors, variable and constraint cones expressed as lists of standard primitive cones (1-dimensional vector sets) with corresponding ordered row indices, and a vector of variable types (each continuous, binary, or general integer). In addition to the basic linear cones (nonnegative, nonpositive, zero, and free cones), Pajarito recognizes three standard primitive nonpolyhedral cones introduced in Section 1.2: exponential cones (see Appendix A.1), second-order cones (see Appendix A.2), and positive semidefinite cones (see Appendix A.3).[17]

Friberg [2016] designed the Conic Benchmark Format (CBF) as a file format originally to support mixed-integer second-order cone (SOCP) and positive semidefinite cone (SDP) instances. In collaboration with Henrik Friberg, we extended the format to support exponential cones in Version 2, and developed a

[16]MathProgBase documents the conic and linear/quadratic interfaces at mathprogbasejl.readthedocs.io/en/latest. JuMP is documented at juliaopt.org/JuMP.jl/0.18/.

[17]As we note in Appendix A.2, Pajarito also recognizes rotated second-order cones, but for simplicity converts them to second-order cones during preprocessing.

Julia interface ConicBenchmarkUtilities.jl to provide utilities for translating between CBF and MathProgBase conic format. One may use Pajarito to solve any instance in the Conic Benchmark Library (CBLIB), which contains thousands of benchmark problems from a wide variety of sources. Pajarito's extensive unit tests rely on small example instances loaded from CBF files.

Lubin et al. [2016] demonstrate that all 333 known MI-convex instances in MINLPLib2 [Vigerske, 2018] are representable with linear, second-order, exponential, and *power* cones. Since a power cone constraint is representable with linear and exponential cone constraints, Pajarito can be used to solve any of the MI-convex instances in MINLPLib2. We translated 115 instances from the MINLPLIB2 library to CBF and contributed them to CBLIB.[18] Many of the MINLPLIB2 instances have tiny values artificially-introduced in order to work around potential numerical issues with smooth derivative-based NLP solvers [Günlük and Linderoth, 2012], which we manually removed before converting to conic form. For example, the instance 'rsyn0805h' has a constraint:

$$\left(\frac{x_{289}}{10^{-6} + b_{306}} - \frac{6}{5} \log \left(1 + \frac{x_{285}}{10^{-6} + b_{306}} \right) \right) (10^{-6} + b_{306}) \leq 0, \tag{25}$$

where $b_{306}, x_{285}, x_{289}$ are scalar variables. Without the artificial 10^{-6} values, a conic encoding of the NLP constraint (25) in terms of the exponential cone $\mathcal{E}$ is:

$$(b_{306} + x_{285}, b_{306}, {}^{5}\!/\!{}_{6}\, x_{289}) \in \mathcal{E}. \tag{26}$$

Within Julia, the modeling packages JuMP [Dunning et al., 2017] and Convex.jl [Udell et al., 2014] each provide a convenient way for users to specify MI-convex problems, call Pajarito solver, and interpret solutions. JuMP is particularly useful for a large, sparse problem involving complex indexing schemes for variables, expressions, or constraints. It efficiently builds a MathProgBase conic form representation of a problem involving second-order or positive semidefinite cones, but currently does not recognize exponential cones.

Convex.jl, unlike JuMP, is a Disciplined Convex Programming (DCP) modeling package. It defines a list of *atoms* for the user to model a MI-convex problem with and performs automatic verification of convexity of the continuous relaxation by applying simple composition rules described by Grant et al. [2006]. It converts the problem into a MI-conic instance in MathProgBase conic form through epigraph and perspective transformations that introduce additional variables and constraints in conic form using only the standard primitive cones recognized by Pajarito. CVXPY [Diamond and Boyd, 2016] is a Python-based DCP modeling package analogous to Convex.jl. In collaboration with Steven Diamond and Baris Ungun, we developed cmpb.jl (github.com/mlubin/cmpb), a prototype C API to MathProgBase that enables Pajarito to be called on a problem modeled with CVXPY.

[18]Lubin et al. [2016] first translated these instances from the MINLPLIB2 library into Convex.jl models. We used ConicBenchmarkUtilities.jl to translate these to CBF. The instances, available at github.com/mlubin/MICPExperiments, are 48 'rsyn' instances, 48 'syn' instances, 6 'tls' instances, 12 'clay' instances, and the challenging 'gams01' instance.

We illustrate Convex.jl and JuMP modeling using a simple MI-convex example described by Boyd and Vandenberghe [2004, ch. 7.5]: 'E-optimal experiment design'.[19] While we can solve E-optimal experiment design exactly using Pajarito, Boyd and Vandenberghe [2004, ch. 7.5] choose to relax the integrality constraints in order to use a continuous convex solver before rounding the fractional solution heuristically. To begin, we set up the Pajarito solver object `mysolver` using a GLPK MILP solver object and a SCS conic solver object, each with internal options set.

```
using Pajarito, GLPKMathProgInterface, SCS #load packages
mysolver = PajaritoSolver(log_level = 3, #use verbose output
    mip_solver = GLPKSolverMIP(msg_lev = GLPK.MSG_OFF), #set MIP solver
    cont_solver = SCSSolver(eps = 1e-6, verbose = 0)) #set conic solver
```

Pajarito performs a sanity check on the combination of options and solvers specified.[20] Next, we model and solve the problem using Convex.jl as follows, where $p, m, n \in \mathbb{R}$ and $V \in \mathbb{R}^{n \times p}$ are problem data.

```
using Convex
mp = Variable(p, Positive(), :Int) #create p nonneg. integer variables
eOpt = maximize(lambdamin(V * diagm(mp./m) * V'), #max. min. eigenvalue
    sum(mp) <= m) #add linear constraint
solve!(eOpt, mysolver) #solve model using Pajarito solver
@show eOpt.status, eOpt.optval, mp.value #show solve status and results
```

Alternatively, we model and solve the problem using JuMP as follows.

```
using JuMP
eOpt = Model(solver = mysolver) #initialize model using Pajarito solver
@variable(eOpt, mp[1:p] >= 0, Int) #create p nonneg. integer variables
@constraint(eOpt, sum(mp) <= m) #add linear constraint
@variable(eOpt, t) #create auxiliary variable
FI = V * diagm(mp./m) * V' #create linear expression matrix
@SDconstraint(eOpt, FI - t * eye(n) >= 0) #add PSD constraint on matrix
@objective(eOpt, Max, t) #maximize linear objective
@show solve(eOpt) #solve model and show status
@show getobjectivevalue(eOpt), getvalue(mp) #show solve results
```

Pajarito manipulates the conic data and performs sanity checks. We refer to the resulting preprocessed representation of the instance as $\mathfrak{M}$. After Pajarito executes one of the OA algorithms described in Section 5.3 on $\mathfrak{M}$, the user can use Convex.jl or JuMP to conveniently query information such as Pajarito's solve status, objective bound, objective value, and solution.

[19]More Pajarito examples can be found at github.com/JuliaOpt/Pajarito.jl/blob/master/examples.

[20]MathProgBase does not attempt to provide an abstraction for solver parameters like convergence tolerances. In cases where we need certain tolerances on the continuous conic and MIP solvers in order for Pajarito to converge to a requested tolerance, it is the user's responsibility to set these tolerances. For example, we ask users to manually adjust the MIP solver's linear feasibility tolerance and integer feasibility tolerance for improved convergence behavior. These cases are documented in Pajarito's readme file.

5.3 Basic Algorithmic Implementations

We discuss the main conic-certificate-based methods Pajarito uses to solve the preprocessed MI-conic model $\mathfrak{M}$. We omit many options, enhancements, and numerical details that can be understood from the Pajarito readme file and from browsing the high-level Julia code and comments. Note that for explaining our computational experiments, Section 6.3 briefly introduces several other algorithmic variants that we do not discuss here, such as separation-based methods that do not utilize conic certificates. In Section 5.3.1, we summarize the initialization procedure for the OA model, an MILP relaxation of $\mathfrak{M}$ that Pajarito constructs and later refines (with extreme ray $\mathcal{K}^*$ cuts) using JuMP. In Section 5.3.2, we describe the 'iterative' method, an extension of the simple sequential OA algorithm by Lubin et al. [2018]. In Section 5.3.3, we describe the 'MIP-solver-driven' (MSD) method, so-called because it relies on the power of the branch-and-cut MIP solver to manage convergence in a single tree. Since MathProgBase's solver-independent abstraction for MIP solver callbacks is designed primarily around shared behavior between CPLEX and Gurobi, Pajarito is limited to interacting with the MIP solver through a *lazy cut callback* function and a *heuristic callback* function. Although the MSD method is generally much faster than the iterative method, the latter may be used with MILP solvers for which callback functionality is unavailable or unreliable.

5.3.1 Initializing The MIP OA Model

We first solve the continuous relaxation of $\mathfrak{M}$ (in which only the integrality constraints are relaxed), using the primal-dual conic solver (see top of Figure 2). This conic problem is analogous to the first node subproblem in Algorithm 1, but without finite bounds on the integer variables. Note that we preprocess this conic model slightly to tighten any non-integral bounds on the integer variables. If the conic solver indicates this relaxation is infeasible, then $\mathfrak{M}$ must be infeasible, so we terminate with an 'infeasible' status. If the conic solver returns a complementary solution pair, the optimal value gives an objective lower bound $L > -\infty$ for $\mathfrak{M}$. Otherwise, we set $L = -\infty$. We initialize the objective upper bound U for $\mathfrak{M}$ to ∞.

Using JuMP, we build the initial OA model, adding the variables and integrality constraints and setting the objective (see bottom of Figure 2). We then add initial fixed cuts for each primitive cone, as we describe in Section 4.2. Primitive linear cone constraints are imposed entirely (as equivalent LP equality or inequality constraints), and for each primitive nonpolyhedral cone, we add a small number of initial fixed cuts (defined in Appendix A).

A complementary solution pair from the conic relaxation solve yields a $\mathcal{K}^*$ point, so we perform an extreme ray disaggregation from Section 4.1 and add certificate cuts for each primitive nonpolyhedral cone (using the procedures in Appendix A). These continuous relaxation certificate cuts technically guarantee that the root node of the OA model has an optimal value no smaller than L.[21]

[21]This can be seen from a simple modification of the complementary solution case polyhedral

This is important because we cannot handle unboundedness of the OA model.

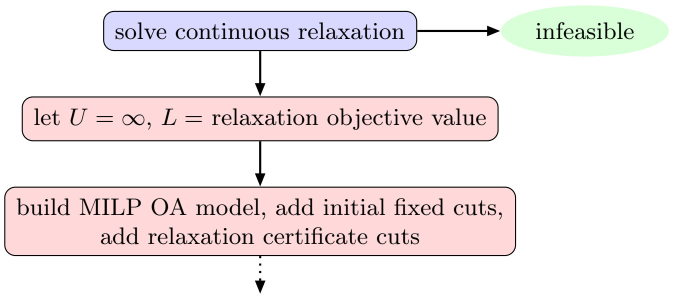

solve continuous relaxation
infeasible
let $U = \infty$, L = relaxation objective value
build MILP OA model, add initial fixed cuts, add relaxation certificate cuts

Figure 2: Pajarito's OA model initialization.

5.3.2 Iterative Method

The iterative method, following initialization in Figure 2, is outlined in Figure 3. At each iteration of the main loop, Pajarito solves the current OA model using the MIP solver.[22] If the OA model is infeasible, $\mathfrak{M}$ must be infeasible, so we terminate with an 'infeasible' status. If it is unbounded, Pajarito fails with an 'OA fail' error status, as we are unable to handle unbounded rays. If the MIP solver returns an optimal solution to the OA model, this OA solution satisfies the integrality constraints and initial fixed cuts, but in general not all of nonpolyhedral primitive cone constraints. The MIP solver's objective bound provides a lower bound for $\mathfrak{M}$, so we update L. Pajarito terminates with an 'optimal' status if the relative optimality gap condition (17) on L, U is satisfied.[23]

If after solving the OA model we have an optimal OA solution and the objective bounds haven't converged, we check whether the OA sub-solution on the integer variables has been encountered before. If not, then we solve a continuous conic subproblem in which the integer variables are fixed to their values in the new integer sub-solution. This subproblem is analogous to $\mathfrak{C}(\boldsymbol{l}, \boldsymbol{u})$ from Section 2.1, with $\boldsymbol{l} = \boldsymbol{u}$, and it has not been solved during any previous iteration. Note that in preprocessing, we remove any subproblem equality constraints that effectively have no variables when an integer sub-solution is fixed. For efficient loading of the subproblem data at each iteration, we only change the constant vector $\boldsymbol{b}$ of the preprocessed conic subproblem, as this is the only data that changes. Since it is more constrained than the OA model, the conic subproblem is bounded or infeasible. If the conic subproblem solver fails to return a certificate, we backtrack and perform the separation procedure (as if the integer sub-solution repeated). Otherwise, we scale the certificate's dual solution or dual ray according to Section 3.2 (using the tolerance values set as

relaxation guarantee we prove in Section 3.1.2, with trivial bounds on the integer variables.

[22]We suggest the user set the MIP solver's relative optimality gap tolerance to its smallest possible value.

[23]Pajarito uses $\theta = 10^{-5}$, to avoid division by zero. The gap tolerance $\epsilon > 0$ is specified by the user, but defaults to $\epsilon = 10^{-5}$.

Pajarito options), then disaggregate the scaled $\mathcal{K}^*$ point and add extreme ray certificate cuts to the OA model (as we described for the continuous relaxation certificate in Section 5.3.1). In the case of a complementary solution pair, the primal solution yields a feasible point for $\mathfrak{M}$, since it satisfies both the integrality and conic constraints. If it has an objective value better than U, we update U and the incumbent solution and check the relative optimality gap condition again.[24]

If instead the integer sub-solution has been encountered at a previous iteration, then we have already solved the corresponding conic subproblem, and doing so again would only yield redundant information and lead to cycling. In this case, we check the conic feasibility of the OA solution by calculating the absolute violation on each primitive nonpolyhedral cone constraint as the violation of the appropriate separation cut (defined in Appendix A). If the worst absolute violation does not exceed Pajarito's feasibility tolerance (set by the user), then the OA solution is considered feasible. In this case, since the solution is optimal for the OA model, we can consider it optimal for $\mathfrak{M}$, so we update the incumbent and upper bound and terminate the solve immediately. If the OA solution is not considered feasible, we add all of the separation cuts that are (significantly) violated to the OA model.

After adding separation or certificate cuts, we warm-start the MIP solver with our incumbent and re-execute the main loop. The procedure in Figure 3 is iterated until L and U converge or the MIP solver detects infeasibility.[25] Note that since we only add cuts to the OA model on every loop, if the first OA model is bounded, then all subsequent (refined) OA models are bounded or infeasible, and the sequence of lower bounds L is nondecreasing. If we assume that all subproblems are solved exactly, our iterative method converges correctly and finitely, provided that there are no ill-posed conic subproblems. If ill-posedness occurs, Pajarito may fail to converge, as there exists no finite set of cuts that can tighten the lower bound sufficiently to meet the upper bound (see Lubin et al. [2016] for a discussion of strong duality in OA).

5.3.3 MIP-Solver-Driven Method

The MSD method, following initialization in Figure 2, is outlined in Figure 4. As in the iterative method, Pajarito returns an 'OA fail' status if the MIP solver detects unboundedness (as we are unable to handle unbounded rays), or an 'infeasible' status in the case of infeasibility.[26] The MIP-solver-independent callback interface allows us to pass in lazy cuts during a lazy callback and feasible

[24]Conic solvers typically do not use an absolute primitive constraint-wise feasibility tolerance, as Pajarito does for checking feasibility of OA solutions for the conic constraint. Our incumbent may not satisfy this notion of feasibility, since we do not perform a feasibility check on the conic solver's primal subproblem solutions.

[25]If the user sets a time limit, Pajarito may terminate with the status 'user limit'. Pajarito sets the time limit on each MIP or conic solve to the remaining time. Note that the vast majority of Pajarito execution time is spent in MIP or conic solves.

[26]If the user sets a time limit, Pajarito sets a time limit on the MIP solver, and terminates with a 'user limit' status if this limit is reached.

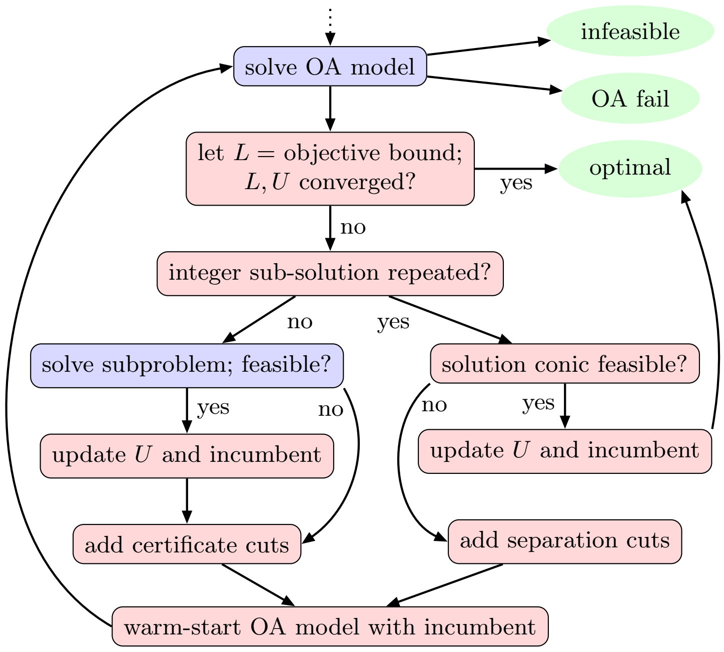


Figure 3: Pajarito's iterative method, following initialization.

solutions during a heuristic callback, however we cannot exert any control over branching decisions, node selection, fathoming, or node lower bound updating.

The MIP solver calls the lazy callback function whenever it finds an integer-feasible OA solution at a node. During a lazy callback, we first check whether the integral OA solution from the MIP solver is repeated. If so, we derive separation cuts to add as lazy constraints; if none can be added, the MIP solver considers the solution feasible and may update its incumbent. If the integer sub-solution has not been encountered before, we solve a new (bounded or infeasible) conic subproblem. Since we lack the ability to query the node's bounds on the integer variables, we only solve subproblems with fixed integer sub-solutions, as in the iterative method. If the conic solver returns a certificate, we scale and disaggregate the $\mathcal{K}^*$ point (as we described for the iterative method in Section 5.3.2), and add extreme ray cuts as lazy constraints.[27] In the case of a complementary solution pair, the primal solution yields a feasible point for $\mathfrak{M}$, which we store. During a heuristic callback, if there is a stored feasible solution to $\mathfrak{M}$ that has never been added as a heuristic solution, we add it.

Since there are no guarantees on when or how frequently the MIP solver calls the heuristic callback function, we may not be able to indirectly update

[27]The MIP solver is not guaranteed to respect the cuts that we add, and we may need to re-add the same cuts during multiple lazy callbacks (unlike in the iterative method, where cuts previously added are respected). We actually store a dictionary from the integer sub-solution to the cuts. For each repeated integer sub-solution, we re-add these saved certificate cuts, in addition to the new separation cuts.

the MIP solver's incumbent and upper bound when we are able to. Partly for this reason, Pajarito maintains its own upper bound and incumbent, not illustrated in Figure 4, which we update during lazy callbacks. During each lazy callback, we ask the MIP solver for its lower bound and check our relative optimality gap condition (as we described for the iterative method). If the condition is met, we force the MIP solver to terminate early. In this case, or if the MIP solver terminates with an optimal solution and we verify that the relative optimality gap condition is met, we return our incumbent solution with an 'optimal' status.[28] Even if we assume that all subproblems are solved exactly and ill-posed subproblems are not encountered, our MSD method may not behave as expected. Across different MIP solvers we encountered variability on whether all lazy cuts we add are respected. Also, the potential inconsistency between Pajarito's incumbent and the MIP solver's incumbent creates opportunities for undefined behavior, e.g., the MIP solver may have no branching candidates and be unable to fathom a node by bound if it does not have the latest incumbent, although we did not observe this occurring in practice.

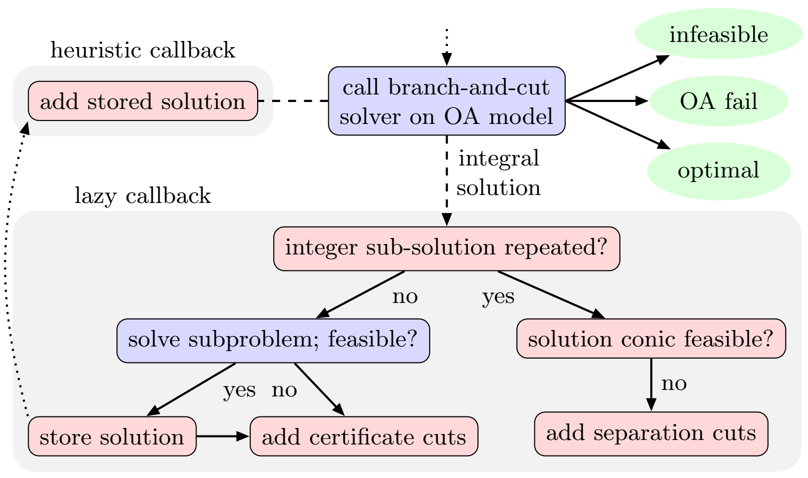


Figure 4: Pajarito's MIP-solver-driven (MSD) method, following initialization.

5.4 Some Advanced Algorithmic Enhancements

We conclude with several key optional OA enhancements we implemented in Pajarito. First, Pajarito by default uses an *extended formulation* for each second-order cone constraint. Vielma et al. [2017] demonstrate on a testset of mixed-integer second-order cone (MISOCP) problems that OA algorithms tend to converge much faster when using this extended representation for each second-

[28]Note the user is responsible for setting the desired relative optimality gap tolerance on both the MIP solver and on Pajarito directly.

order cone constraint.[29] In Appendix B, we describe how to lift a $\mathcal{K}^*$ cut for the second-order cone into $\mathcal{K}^*$ cuts for the extended formulation. This technique also allows us to describe a much more economical set of initial fixed $\mathcal{K}^*$ cuts for the second-order cone.

Second, Pajarito can optionally use a MISOCP OA model instead of a MILP OA model. There exist several powerful MISOCP solvers that can be used, or Pajarito itself may be used. Since a second-order cone constraint can imply an infinite number of $\mathcal{K}^*$ cuts, Pajarito can achieve tighter relaxations of the conic constraint in the OA model. Of course, this can only make practical sense for certain types of problems that aren't pure MISOCP. One potential use case is where the MI-conic problem has second-order cones as well as exponential and/or positive semidefinite cones, but the only stable and efficient continuous conic solvers we have access to are for SOCP problems. In this case, we can use Pajarito with the SOCP solver as an MISOCP solver, and pass this into a second Pajarito solver that uses a conic solver for mixed-cone problems. This arrangement helps minimize the number of calls to the less-effective conic solver. Another use case is for problems involving positive semidefinite cone constraints. We demonstrate in Appendix C how to strengthen $\mathcal{K}^*$ cuts for PSD constraints to rotated-second-order cone constraints.[30]

6 Computational Experiments

Our computational experiments demonstrate the speed and robustness of our open source MI-conic solver Pajarito. As we emphasize in Section 5.3, our algorithmic implementations differ from the description of Algorithm 1, because of our practical decision to use branch-and-cut MILP solvers through a limited, solver-independent interface. In Section 6.1, we summarize our metrics for comparing the practical performance of different MI-conic solvers and describe our presentation of tables and performance profile plots. In Section 6.2, we benchmark Pajarito (version 0.5.1) and several open source and commercial mixed-integer second-order cone (MISOCP) solver packages accessible through MathProgBase on a MISOCP library, and conclude that Pajarito is the fastest and most-reliable open source solver for MISOCP. In Section 6.3, we compare the performance of several of Pajarito's algorithmic variants on MI-conic instances involving mixtures of positive semidefinite, second-order, and exponential cones, demonstrating practical advantages of the methodological extensions we describe in Sections 3 and 4 and Appendix A. The scripts and data we use to run our experiments are available in the supplement github.com/chriscoey/PajaritoSupplement.

[29] DCP modeling software implementations such as Convex.jl do not perform this transformation because they are simply designed to access conic solvers. Pajarito keeps the original second-order cone formulation in the conic subproblems because conic solvers are likely to perform better with this representation than with the higher-dimensional extended formulation.

[30] For the MSD method, since most MISOCP solvers don't currently allow adding lazy quadratic constraints, only the initial fixed cuts can be strengthened in this way.

6.1 Presentation Of Results

We define a 'solver' as a MathProgBase solver object given a particular complete set of algorithmic options. Each solver we test is deterministic, i.e. it performs consistently across different runs on a particular dedicated system. We define an 'instance' as a particular MI-conic problem (stored in CBF format; see Section 5.2) that is known to be feasible and bounded (but an optimal solution or the optimal objective value is not necessarily known). For a particular instance, a solver may return a 'solution', which is a vector of real floating point numbers representing an assignment of the variables of the instance (not necessarily feasible for the constraints).

First, we compare the performances of a group of MI-conic solvers on a particular testset of instances by counting the number of instances for which each solver returns and apparently proves 'approximate optimality' of a solution. To be more precise, we use the following four categories to characterize a solver's apparent success or failure on an instance.

ex (exclude) means either the solver incorrectly claims the instance is infeasible or unbounded, or the solver returns a solution it claims is approximately-optimal but we detect one of the following inconsistencies.

- The solution significantly violates at least one primitive cone constraint or integrality constraint. We calculate the absolute violation of a primitive cone constraint as the worst violation of the inequalities defining the standard cone (see Appendix A), and our tolerances are 10^{-6} for linear cones, 10^{-5} for second-order and exponential cones, and 10^{-4} for positive semidefinite cones. We calculate the variable-wise integrality violation as the distance to the nearest integer, and our tolerance is 10^{-6}.
- The relative objective gap condition (equation (17)) for optimality is significantly violated. Our optimality condition (17) matches that used by most MIP solvers. We set the constant $\theta = 10^{-5}$ (to avoid division by zero) and use the tolerance $\epsilon = 10^{-5}$. We ensure we do not exclude in the case that the gap we calculate is sensitive to a solver's different value of θ.
- The objective value or objective bound significantly differs from that of a preponderance of other solvers [[moved from footnote to text:]] (we assess this semi-manually from the output of our scripts).

co (converge) means the solver returns a solution that it claims is (approximately) optimal (and it is not excluded for the reasons above).

li (reach limit) means the solver does not terminate before the time limit, or (rarely) the solver reaches a memory limit and is forced to terminate.

er (error) means the solver crashes or terminates with an error message.

Second, we compare aggregate quantitative measures of solver performance. We define the shifted geometric mean $\tilde{g}$ of L positive values $p_1, \dots, p_L$ as:

$$\tilde{g}(\boldsymbol{p}, q) = \prod_{l \in [\![L]\!]} (p_l + q)^{\frac{1}{L}} - q, \tag{27}$$

where $q > 0$ is the shift [Achterberg, 2009]. Unlike the standard geometric mean $\tilde{g}(\boldsymbol{p}, 0)$, the shifted geometric mean decreases the relative influence of smaller values in $\boldsymbol{p}$, thus giving less weight to very 'easy' instances (small values are preferable for all of our metrics). We shift by $q = 10$ seconds for execution times, $q = 1$ iterations for iteration counts, and $q = 10$ nodes for MIP-solver-reported node counts. For comparing a particular group of solvers $S_1, \dots, S_n$ on a particular performance metric (such as execution time), we calculate for each solver S_i the following three shifted geomeans, each over a different subset of the testset.

aco (all solvers converge) is calculated over the instances for which $S_1, \dots, S_n$ all have a 'co' status.

tco (this solver converges) is calculated over the instances for which S_i has a 'co' status.

all (all instances) is calculated over all instances. Missing execution times are set to the time limit, and missing iteration/node counts are ignored.

Finally, we employ 'performance profiles', described by Dolan and Moré [2002], Gould and Scott [2016], to visually compare the relative execution times and iteration or node counts of pairs of solvers. Again, we decrease the relative influence of very easy instances by shifting the metrics by the same shift values q we use for shifted geomeans. A performance profile is a plot that should be interpreted as follows: for a fixed factor F on the horizontal axis (a linear scale from 1 to the value at the bottom right of the plot), the level of solver S_i on the vertical axis (a linear scale from 0 to 1) represents the proportion P_i of instances (out of the instances for which at least one of the pair of solvers has a 'co' status) for which S_i has a 'co' status and a (shifted) performance metric that is within a factor of F of per-instance best achieved by either solver. So, at $F = 1$ (i.e. on the left vertical axis), P_i is the fraction of solved instances on which solver S_i has the best performance. As F increases, we can infer that solver S_i has reliably better performance than solver S_j if P_i remains above P_j.

6.2 MISOCP Solver Performance Comparisons

Our open source Pajarito solvers, 'Iter-GLPK' and 'Iter-CBC', use the iterative method (see Section 5.3.2) with ECOS [Domahidi et al., 2013] for continuous conic subproblems and CBC or GLPK for MILPs.[31] Our two restricted-license

[31]We do not test the Pajarito's MIP-solver-driven method with CBC or GLPK MIP solvers because their support for MathProgBase callbacks is limited.

Pajarito solvers, 'Iter-CPLEX' (using the iterative method) and 'MSD-CPLEX' (using the MIP-solver-driven method; see Section 5.3.3), call MOSEK's continuous conic solver and CPLEX's MILP solver.

The open source Bonmin solver package is described in detail by Bonami et al. [2008] and uses CBC to manage branching and Ipopt to solve continuous NLP (derivative-based nonlinear programming) subproblems.[32] Our 'Bonmin-BB' solver uses the nonlinear B&B method (no polyhedral approximation), 'Bonmin-OA' uses the B&B OA method, and 'Bonmin-OA-D' is equivalent to the 'Bonmin-OA' solver but applied to transformed instances that use the second-order cone extended formulation we describe in Section 5.4. Our two restricted-license MISOCP solvers are 'SCIP' and 'CPLEX'. Unlike Bonmin, these MISOCP solvers use the second-order cone extended formulation internally.[33]

These nine MISOCP solvers are each given a relative optimality gap tolerance of 10^{-5}. The 'SCIP' and 'CPLEX' solvers are given an absolute linear-constraint-wise feasibility tolerance of 10^{-8}, and 'CPLEX' is given an integrality tolerance of 10^{-9}. The MILP solvers used by Pajarito are given an absolute linear-constraint-wise feasibility tolerance of 10^{-8}, an integrality tolerance of 10^{-9}, and a relative optimality gap tolerance of 0 for 'Iter-GLPK', 'Iter-CBC', and 'Iter-CPLEX' and 10^{-5} for 'MSD-CPLEX'. Due to limited resources, we set a one hour time limit for each run of a solver on an instance, and run all solvers (including the MILP and conic solvers called by Pajarito) in single-threaded mode.

We use a testset of 120 MISOCP instances drawn from the larger CBLIB library, recently compiled by Friberg [2016]. The testset contains randomly selected subsets of most of the major families of models in CBLIB. We exclude instances that are not bounded and feasible, or are solved in under 5 seconds by all solvers, or are unable to be solved by all solvers in under an hour. Our computations are performed on the Amazon EC2 cloud computing platform with 'm4.xlarge' computing nodes having 16GB of RAM.[34] As the computing nodes are virtual machines, timing results on EC2 are subject to random variability. However, by repeatedly running some of the experiments, we verified that this variability is sufficiently small to avoid impacting our conclusions. The nodes run Ubuntu 16.04 with Julia version 0.6.0. Version information for the Julia packages can be obtained from the supplement.

Table 1 summarizes the status counts and shifted geomeans of performance metrics on instance subsets (explained in Section 6.1) for the nine MISOCP solvers on the 120 MISOCP instances. The Bonmin solvers fail on most instances, and overall solve significantly fewer instances than the open source Pajarito solvers. Pajarito tends perform faster using CBC rather than GLPK.[35] Figure 5a

[32]We are unaware of any mainstream open source solvers designed for MISOCP. The functional representation of the second-order cone has points of nondifferentiability that may cause Bonmin to crash or suffer numerical issues.

[33]CPLEX is available under an academic or commercial licence, and SCIP is an academic solver that is not released under an OSI-approved open source license. We use CPLEX version 12.7.0 and SCIP version 4.0.0.

[34]See aws.amazon.com/ec2/instance-types.

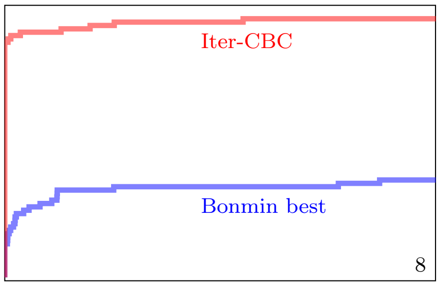
[35]However, for most of the 9 excluded instances from 'Iter-CBC', we verify that CBC is

is a performance profile (explained in Section 6.1) comparing the execution times of the open source Pajarito (with CBC) solver and the instance-wise best of the three Bonmin solvers. From these results, we claim that Pajarito with ECOS and CBC is the fastest and most reliable open source MISOCP solver.

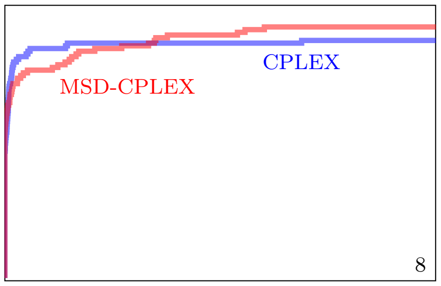
Using CPLEX, Pajarito's MSD method is significantly faster and more reliable than its iterative method.[36] The performance profile Figure 5b compares the execution times of Pajarito's MSD method using CPLEX's MILP solver against CPLEX's specialized MISOCP solver. The execution time comparisons between 'CPLEX' and 'MSD-CPLEX' are ambiguous, however we argue that, at least by our metrics, Pajarito is a more reliable MISOCP solver. This may be due largely to Pajarito's use of conic certificate $\mathcal{K}^*$ cuts.

		statuses				time (s)		
	solver	co	li	er	ex	aco	tco	all
open source	Bonmin-BB	34	44	11	31	38.0	83.8	463
	Bonmin-OA	25	53	29	13	64.2	64.5	726
	Bonmin-OA-D	30	48	29	13	15.1	61.6	610
	Iter-GLPK	56	60	3	1	2.0	29.7	377
	Iter-CBC	78	30	3	9	1.6	50.3	163
restricted	SCIP	74	35	8	3	3.2	41.5	160
	CPLEX	90	16	5	9	0.9	16.1	50
	Iter-CPLEX	86	26	0	8	0.4	37.0	106
	MSD-CPLEX	97	20	2	1	0.4	18.2	56

Table 1: MISOCP solver performance summary.



(a) Open source Bonmin (instance-wise best of 3) and Pajarito iterative solvers.



(b) CPLEX MISOCP and Pajarito MSD solvers.

Figure 5: MISOCP solver execution time performance profiles.

responsible for the significant integrality violations that result in exclusion.

[36]For 'MSD-CPLEX', the two errors occur where Pajarito claims a solution is suboptimal and has an objective gap no worse than 1.04×10^{-5}, and the one exclusion occurs where Pajarito's solution violates a linear constraint by 9.78×10^{-6}.

6.3 Comparative Testing Of Algorithmic Variants

To compare the performance of several of Pajarito's algorithmic variants, we use a testset of 95 MI-conic instances involving mixtures of positive semidefinite (PSD), second-order, and exponential cones. Formulations for the instances we generated can be found at github.com/JuliaOpt/Pajarito.jl/tree/master/examples. These instances are all bounded and feasible and come from the following four sources.

Discrete experiment design (14 instances). Recall from Section 5.2 that Boyd and Vandenberghe [2004, Ch. 7.5] describes MI-convex experiment design problems. We generate 'A-optimal' and 'E-optimal' instances that include PSD cones, and 'D-optimal' instances that include PSD and exponential cones.

Portfolios with mixed risk constraints (16 instances). We formulate a portfolio problem that maximizes expected returns subject to some combinatorial constraints on stocks and three types of convex risk constraints on subsets of stocks with known covariances. Each instance includes multiple exponential cones from entropy risk constraints, second-order cones from norm risk constraints, and PSD cones from robust norm risk constraints.

Retrofit-synthesis of process networks (32 instances). We select a representative subset of the two CBLIB families 'syn' and 'rsyn'. Each instance includes exponential cones.

A subset of the MISOCP testset (33 instances). We select a representative subset of the CBLIB families 'estein', 'ccknapsack', 'sssd', 'uflquad', and 'portfoliocard'.

We use Pajarito with Gurobi (version 7.5.2) as the MILP solver and MOSEK (version 9.0.0.29-alpha) as the continuous conic solver.[37] Pajarito is given a relative optimality gap tolerance of 10^{-5}. Gurobi is given an absolute linear-constraint-wise feasibility tolerance of 10^{-8}, an integrality tolerance of 10^{-9}, and a relative optimality gap tolerance of 0 when the iterative method is used and 10^{-5} when the MSD method is used. We set a one hour time limit for each run of a solver on an instance, and limit Gurobi and MOSEK to 8 threads. We run the computations on dedicated hardware with two Intel Xeon E5-2650 CPUs and 64GB of RAM. The machine runs Ubuntu 17.10 and Julia 0.6.2. Version information for the Julia packages can be obtained from the supplement. By repeatedly running some of the experiments, we again verified that, as expected, any random variability of timings within these experiments is sufficiently small to avoid impacting our conclusions.

6.3.1 Initial Fixed Cuts, Certificate Cuts, And Separation Cuts

Recall from Section 5.3 that Pajarito by default uses three different types of $\mathcal{K}^*$ cuts: initial fixed cuts, certificate cuts, and separation cuts. For both the

[37]MOSEK 9 is the first version to recognize exponential cones.

iterative and MSD methods, we compare the following four important algorithmic variants of OA that use different combinations of these three cut types.

c means initial fixed cuts on linear primitive cones only, and certificate cuts on nonpolyhedral primitive cones.

cs means initial fixed cuts on linear primitive cones only, and certificate cuts on nonpolyhedral primitive cones, and separation cuts when apparently needed for convergence. The separation cuts allow us to cut off significantly infeasible OA solutions, so Pajarito can also obtain (approximately) feasible solutions from OA solutions found by the MILP solver.

ics means initial fixed cuts on all primitive cones, certificate cuts on nonpolyhedral primitive cones, and separation cuts when apparently needed for convergence. This is Pajarito's default approach, as described in Section 5.3.

is means initial fixed cuts on all primitive cones, and separation cuts only. No conic solver is used, hence all (approximately) feasible solutions found are OA solutions.

Table 2 summarizes the status counts and shifted geomeans of performance metrics on instance subsets. Although the MSD method is significantly faster than the iterative method, we see similar relative performances for the four types of cuts under iterative versus MSD. When using certificate cuts only ('c'), Pajarito often failed to converge to the desired optimality gap (though it typically came very close), likely due to the inexactness of the certificates from the numerical continuous conic solver. By also using separation cuts on repeated integer sub-solutions and accepting (approximately) conic feasible OA solutions as incumbents, Pajarito is able to converge on many more instances. Starting with initial fixed cuts ('ics') further increases Pajarito's robustness, particularly for the MSD method. Comparing the 'ics' variant with the separation-based variant with initial fixed cuts ('is'), we see significantly faster overall performance and fewer iterations or nodes when using the continuous conic solver and adding certificate cuts. The performance profiles Figures 6a to 6d compare the execution times or iteration/node counts for the 'ics' and 'is' solvers, unambiguously demonstrating superiority of Pajarito's default 'ics' method.

6.3.2 Extreme Ray Disaggregation

To test the efficacy of the $\mathcal{K}^*$ extreme ray disaggregation technique we describe in Section 4.1 and Appendix A, we run Pajarito using only certificate cuts (the 'c' variant described in Section 6.3.1), with and without disaggregation. We do not run the default 'ics' variant because disabling disaggregation disables use of the second-order cone extended formulation (which has no benefit without disaggregation), making it impossible to use a polynomial number of initial cuts to achieve the same initial fixed outer approximation for the second-order cone (see Appendix A.2.1).

	cuts	statuses				time (s)			subproblems			iters or nodes		
		co	li	er	ex	aco	tco	all	aco	tco	all	aco	tco	all
Iter	c	72	1	21	1	5.59	6.47	7.17	5.31	5.36	4.23	5.48	5.53	4.46
	cs	88	1	3	3	5.56	12.72	14.38	5.31	6.97	6.36	5.48	7.22	6.84
	ics	89	2	0	4	4.73	11.57	14.77	4.17	5.93	6.03	4.32	6.15	6.32
	is	84	1	0	10	8.35	14.53	22.08	-	-	-	13.41	16.52	18.07
MSD	c	76	0	18	1	2.37	3.40	3.50	12.63	15.80	12.70	223	438	348
	cs	88	0	5	2	3.33	6.47	7.76	18.96	26.77	24.87	295	843	815
	ics	92	0	1	2	2.20	6.31	6.52	15.62	24.58	24.95	273	796	857
	is	84	1	0	10	3.12	5.29	7.49	-	-	-	522	932	1345

Table 2: $\mathcal{K}^*$ cut types performance summary.

Table 3 summarizes the status counts and shifted geomeans of performance metrics on instance subsets, and the performance profiles Figures 7a to 7d compare the execution times or iteration/node counts. For both the iterative and MSD methods, disaggregation improves performance on nearly every solved instance. For the iterative method, it enables the pure-certificate-based variant to converge on more than double the number of instances, and it more than halves the execution time and iteration count. Without disaggregation, the MSD method manages to converge on many more instances than the iterative method. Disaggregation greatly improves the performance of the MSD method, though the comparison is not quite as striking as for the iterative method.

	disag	statuses				time (s)			subproblems			iters or nodes		
		co	li	er	ex	aco	tco	all	aco	tco	all	aco	tco	all
Iter	off	33	10	52	0	10.95	11.11	17.20	11.63	12.41	11.82	11.95	12.74	12.51
	on	72	1	21	1	4.47	6.47	7.17	4.04	5.36	4.23	4.25	5.53	4.46
MSD	off	51	3	41	0	1.74	6.18	6.71	15.51	50.51	27.98	70	613	261
	on	76	0	18	1	1.06	3.40	3.50	7.57	15.80	12.70	36	438	348

Table 3: $\mathcal{K}^*$ cut disaggregation performance summary.

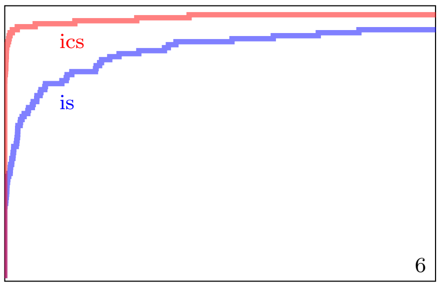
6.3.3 Certificate-Based Scaling

To test the efficacy of the $\mathcal{K}^*$ certificate cut scaling technique for an LP solver with a feasibility tolerance we describe in Section 3.2, we run Pajarito using only certificate cuts (the 'c' variant described in Section 6.3.1), with and without scaling. We do not run the default 'ics' variant because certificate cut scaling does not apply to initial fixed cuts and separation cuts; we want to isolate the effect of the scaling methodology on the reliability of convergence, and the certificate cuts are the only types of cuts yielding convergence guarantees. The results were similar for 'ics' though slightly less pronounced, as initial and separation cuts enable us to converge on nearly all problems, at the cost of more cuts needed. We set a larger feasibility tolerance on these four Pajarito solvers ($\delta = 10^{-6}$ instead of 10^{-8}, which we used for all other tests), to reduce the chance that any observed effects are caused by numerical issues near machine epsilon.

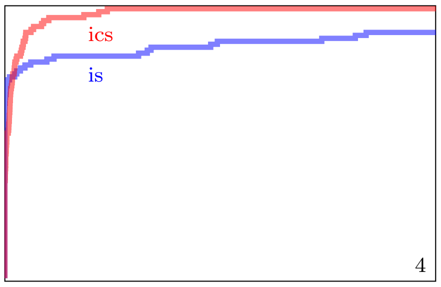
Table 4 summarizes the status counts and shifted geomeans of performance metrics on instance subsets, and the performance profiles Figures 8a to 8d compare the execution times or iteration/node counts. For both the iterative and MSD methods, using scaling improves the robustness of the pure-certificate-based variant, allowing us to converge on 6 or 7 additional instances. On the subset of instances solved by all four solvers (the 'aco' columns), scaling slightly reduces conic subproblem counts and iteration or node counts, but has small and ambiguous effects on the execution times.

	scale	statuses				time (s)			subproblems			iters or nodes		
		co	li	er	ex	aco	tco	all	aco	tco	all	aco	tco	all
Iter	off	63	1	28	3	4.54	4.41	6.59	5.15	5.03	4.23	5.18	5.06	4.40
	on	69	1	22	3	4.35	5.20	6.73	4.90	4.92	3.88	4.99	4.99	4.04
MSD	off	60	0	30	5	2.68	2.77	3.15	12.44	14.48	12.78	193	240	366
	on	67	0	26	2	2.92	4.02	3.86	11.88	15.77	12.07	188	392	393

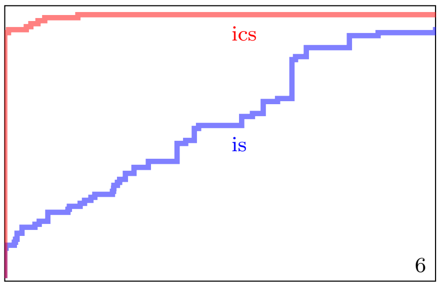
Table 4: $\mathcal{K}^*$ certificate cut scaling performance summary (larger δ).



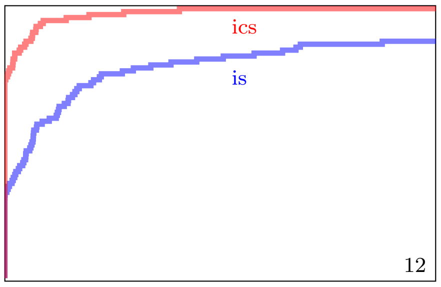
(a) Iter - execution time.



(b) MSD - execution time.

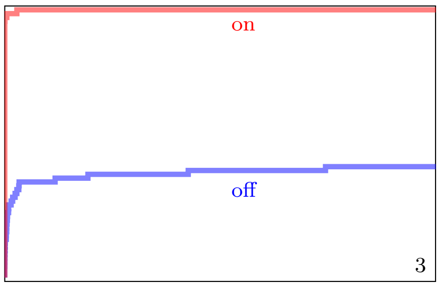


(c) Iter - iteration count.

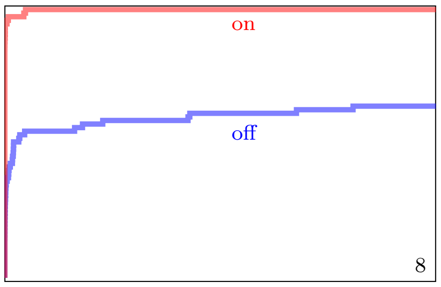


(d) MSD - node count.

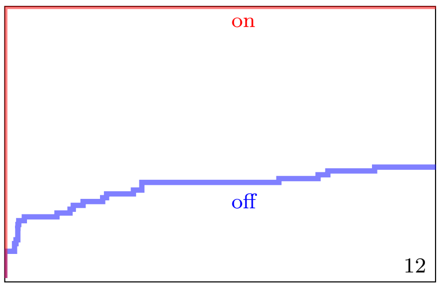
Figure 6: $\mathcal{K}^*$ cut types performance profiles.



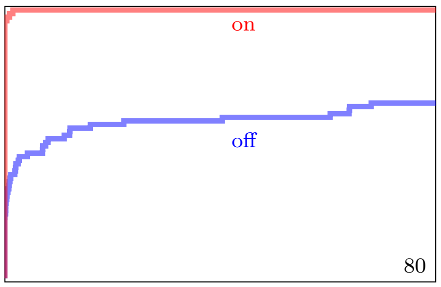
(a) Iter - execution time.



(b) MSD - execution time.

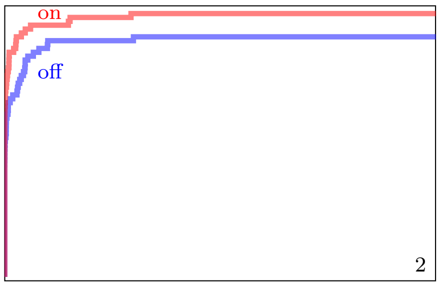


(c) Iter - iteration count.

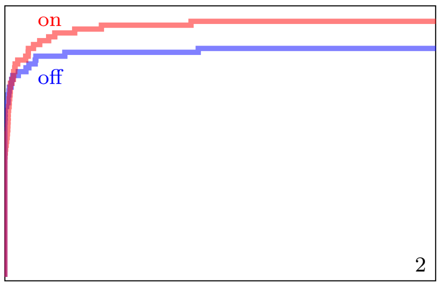


(d) MSD - node count.

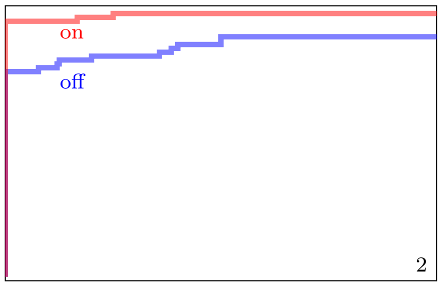
Figure 7: $\mathcal{K}^*$ cut disaggregation performance profiles.



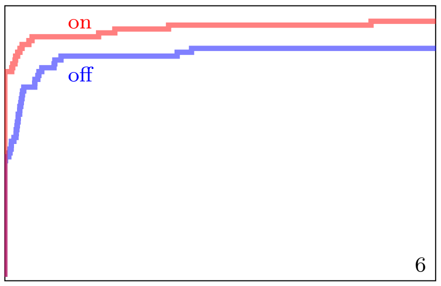
(a) Iter - execution time.



(b) MSD - execution time.



(c) Iter - iteration count.



(d) MSD - node count.

Figure 8: $\mathcal{K}^*$ certificate cut scaling performance profiles.

A The Standard Primitive Nonpolyhedral Cones

As we discussed in Section 5.2, Pajarito recognizes three standard primitive nonpolyhedral cones defined by MathProgBase: exponential, second-order, and positive semidefinite cones. Here, we tailor the general techniques for tightening OAs from Section 4 to a primitive cone constraint involving one of these three cones. In particular, we describe the initial fixed OAs (see Section 4.2), extreme ray disaggregations (see Section 4.1), and separation procedures (see Section 4.3) implemented in Pajarito.

These ideas could be adapted to other primitive nonpolyhedral cones if the user desires. For example, consider a convex constraint (in NLP form) $f(\boldsymbol{t}) \leq r$, where $f : \mathbb{R}^n \to \mathbb{R}$ is a non-homogeneous convex function. If we define a closed convex cone using the closure of the epigraph of the perspective of f:

$$\mathcal{K}_f = \mathrm{cl}\{(r, s, \boldsymbol{t}) \in \mathbb{R}^{2+n} : s > 0, r \geq s f(\boldsymbol{t}/s)\}, \tag{28}$$

then the equivalent conic constraint is $(r, 1, \boldsymbol{t}) \in \mathcal{K}_f$. From Zhang [2014], the dual cone of $\mathcal{K}_f$ is the closure of the perspective of the epigraph of the convex conjugate of f, $f^*(\boldsymbol{w}) = \sup\{-\boldsymbol{w}^T \boldsymbol{t} - f(\boldsymbol{t}) : \boldsymbol{t} \in \mathrm{dom}\, f\}$:

$$\mathcal{K}_f^* = \mathrm{cl}\{(u, v, \boldsymbol{w}) \in \mathbb{R}^{2+n} : u > 0, v \geq u f^*(\boldsymbol{w}/u)\}. \tag{29}$$

Note $\mathcal{K}_f^* \neq \mathcal{K}_{f^*}$ because of the permuting of the first two indices.

A.1 Exponential Cone

The exponential cone $\mathcal{E}$ is defined from the convex univariate exponential function $f(t) = \exp(t)$ in equation (28):

$$\begin{aligned} \mathcal{E} &= \mathrm{cl}\{(r,s,t) \in \mathbb{R}^3 : s > 0, r \geq s\exp(t/s)\} \\ &= \{(r,0,t) : r \geq 0, t \leq 0\} \cup \{(r,s,t) : s > 0, r \geq s\exp(t/s)\}. \end{aligned} \tag{30}$$

The convex conjugate is $f^*(w) = w - w\log(-w)$, so by equation (29), the dual cone of the exponential cone is:

$$\begin{aligned} \mathcal{E}^* &= \mathrm{cl}\{(u,v,w) \in \mathbb{R}^3 : u > 0, w < 0, v \geq w - w\log(-w/u)\} \\ &= \{(u,v,0) : u, v \geq 0\} \cup \{(u,v,w) : u > 0, w < 0, v \geq w - w\log(-w/u)\}. \end{aligned} \tag{31}$$

A.1.1 Initial Fixed Polyhedral Relaxation

Suppose we have a primitive cone constraint $(r,s,t) \in \mathcal{E}$. We use the two $\mathcal{E}^*$ extreme rays $(1,0,0), (0,1,0)$ to impose the simple bound constraints $r, s \geq 0$. We use more $\mathcal{E}^*$ extreme rays of the form $(1, w - w\log(-w), w)$ by picking several different values $w < 0$. Note the corresponding cuts separate any point $(0,0,t)$ satisfying $t > 0$.

A.1.2 Extreme Ray Disaggregation

Suppose we have the $\mathcal{E}^*$ point (u,v,w). If $w = 0$, the point is already a nonnegative combination of the initial fixed $\mathcal{E}^*$ points from Appendix A.1.1, so we discard it. If $w < 0$, we use the $\mathcal{E}^*$ extreme ray $(u, w - w\log(-w/u), w)$, which when added to some nonnegative multiple of $(0,1,0)$, gives (u,v,w).[38]

A.1.3 Separation Of An Infeasible Point

Suppose we want to separate a point $(r,s,t) \notin \mathcal{E}$ that satisfies the initial fixed cuts. Then $r, s \geq 0$ and if $r = s = 0$ then $t \leq 0$. If $s = 0$, then $t > 0$ and $r > 0$, and we use the $\mathcal{E}^*$ extreme ray $(t/r, -2 + 2\log(2r/t), -2)$. If $s > 0$, then $r < s\exp(t/s)$, and we use the $\mathcal{E}^*$ extreme ray $(1, (t/s - 1)\exp(t/s), -\exp(t/s))$.

A.2 Second-Order Cone

For $n \geq 2$, the second-order cone is the epigraph of the ℓ_2-norm, which is convex and homogeneous:

$$\mathcal{L}^{1+n} = \{(r, \boldsymbol{t}) \in \mathbb{R}^{1+n} : r \geq \|\boldsymbol{t}\|_2\}. \tag{32}$$

We sometimes drop the dimension $1 + n$ when implied by context. This cone is self-dual ($\mathcal{L}^* = \mathcal{L}$). We also define the (self-dual) rotated second-order cone:

$$\mathcal{V}^{2+n} = \{(r, s, \boldsymbol{t}) \in \mathbb{R}^{2+n} : r, s \geq 0, 2rs \geq \|\boldsymbol{t}\|_2^2\}. \tag{33}$$

[38]This also projects $(u,v,w) \notin \mathcal{E}^*$ with $u > 0, w < 0$ onto $\mathcal{E}^*$.

Note that $\mathcal{V}$ is an invertible linear transformation of $\mathcal{L}$, since $(r, s, \boldsymbol{t}) \in \mathcal{V}^{2+n}$ if and only if $(r+s, r-s, \sqrt{2}t_1, \ldots, \sqrt{2}t_n) \in \mathcal{L}^{2+n}$, so for simplicity we restrict attention to $\mathcal{L}$.[39]

A.2.1 Initial Fixed Polyhedral Relaxation

Suppose we have a primitive cone constraint $(r, \boldsymbol{t}) \in \mathcal{L}^{1+n}$. First, we note that the ℓ_∞-norm lower-bounds the ℓ_2-norm, since for any $\boldsymbol{t} \in \mathbb{R}^n$ we have:

$$\|\boldsymbol{t}\|_\infty = \max_{i \in [\![n]\!]} |t_i| \leq \|\boldsymbol{t}\|_2. \tag{34}$$

Let $\boldsymbol{e}(i) \in \mathbb{R}^n$ be the ith unit vector in n dimensions. We use the $2n$ $\mathcal{L}^*$ extreme rays $(1, \pm\boldsymbol{e}(i)), \forall i \in [\![n]\!]$, which imply the conditions $r \geq |t_i|, \forall i \in [\![n]\!]$, equivalent to the homogenized box relaxation $r \geq \|\boldsymbol{t}\|_\infty$. Second, we note that the ℓ_1-norm also provides a lower bound for the ℓ_2-norm, since for any $\boldsymbol{t} \in \mathbb{R}^n$ we have:

$$\|\boldsymbol{t}\|_1 = \sum_{i \in [\![n]\!]} |t_i| \leq \sqrt{n}\|\boldsymbol{t}\|_2. \tag{35}$$

We use the 2^n $\mathcal{L}^*$ extreme rays $(1, \boldsymbol{\sigma}/\sqrt{n}), \forall \boldsymbol{\sigma} \in \{-1, 1\}^n$, which imply the homogenized diamond relaxation $r \geq \|\boldsymbol{t}\|_1/\sqrt{n}$. Although the number of initial fixed cuts is exponential in the dimension n, in Appendix B we describe how to use an extended formulation introduced by Vielma et al. [2017] with n auxiliary variables to imply an initial fixed OA that is no weaker but uses only a polynomial number of cuts. Note that the $\mathcal{L}^*$ point $(1, \boldsymbol{0})$, which corresponds to the simple variable bound $r \geq 0$, is a nontrivial conic combination of these initial fixed $\mathcal{L}^*$ extreme rays.

A.2.2 Extreme Ray Disaggregation

Suppose we have the $\mathcal{L}^*$ point $(u, \boldsymbol{w})$. If $\boldsymbol{w} = \boldsymbol{0}$, the point is already a nonnegative multiple of the $\mathcal{L}^*$ point $(1, \boldsymbol{0})$, so we discard it. Otherwise, we use the $\mathcal{L}^*$ extreme ray $(\|\boldsymbol{w}\|_2, \boldsymbol{w})$, which when added to some nonnegative multiple of $(1, \boldsymbol{0})$, gives the original point $(u, \boldsymbol{w})$.[40]

A.2.3 Separation Of An Infeasible Point

Suppose we want to separate a point $(r, \boldsymbol{t}) \notin \mathcal{L}$ that satisfies the initial fixed cuts. Then $r \geq 0$ and so $\boldsymbol{t} \neq \boldsymbol{0}$, and we use the $\mathcal{L}^*$ extreme ray $(1, -\boldsymbol{t}/\|\boldsymbol{t}\|_2)$.

A.3 Positive Semidefinite Cone

For $n \geq 2$, we define the $n \times n$-dimensional positive semidefinite (PSD) matrix cone $\mathbb{S}^n_+$ as a subset of the symmetric matrices $\mathbb{S}^n = \{\boldsymbol{T} \in \mathbb{R}^{n \times n} : \boldsymbol{T} = \boldsymbol{T}^T\}$,

[39] As noted in Section 5.2, Pajarito transforms any $\mathcal{V}$ constraints to equivalent $\mathcal{L}$ constraints during preprocessing.

[40] This also projects $(u, \boldsymbol{w}) \notin \mathcal{L}^*$ with $u < \|\boldsymbol{w}\|_2$ onto $\mathcal{L}^*$.

avoiding the need to enforce symmetry constraints. From the minimum eigenvalue function $\lambda_{\min} : \mathbb{S}^n \to \mathbb{R}$, we have:

$$\mathbb{S}^n_+ = \{\boldsymbol{T} \in \mathbb{S}^n : \lambda_{\min}(\boldsymbol{T}) \geq 0\}. \tag{36}$$

For $\boldsymbol{W}, \boldsymbol{T} \in \mathbb{S}^n$, we use the trace inner product $\langle \boldsymbol{W}, \boldsymbol{T} \rangle = \sum_{i,j \in [\![n]\!]} W_{i,j} T_{i,j}$. $\mathbb{S}^n_+$ is self-dual and its extreme rays are the rank-1 PSD matrices [Ben-Tal and Nemirovski, 2001a], i.e. any $\boldsymbol{\omega}\boldsymbol{\omega}^T$ for $\boldsymbol{\omega} \in \mathbb{R}^n$. An extreme ray $(\mathbb{S}^n_+)^*$ cut has the form:

$$\langle \boldsymbol{\omega}\boldsymbol{\omega}^T, \boldsymbol{T} \rangle = \boldsymbol{\omega}^T \boldsymbol{T} \boldsymbol{\omega} \geq 0. \tag{37}$$

In Appendix C, we describe how to strengthen extreme ray $\mathbb{S}^*_+$ cuts to rotated second-order cone constraints (for MISOCP OA; see Section 5.4).

Recall that our MI-conic form $\mathfrak{M}$ uses vector cone definitions, as does MathProgBase. Mosek ApS [2016] refers to the matrix cone $\mathbb{S}^n_+ \subset \mathbb{S}^n$ as the *smat PSD cone*, and to its equivalent vectorized definition $\mathcal{S}^{n(n+1)/2} \subset \mathbb{R}^{n(n+1)/2}$ as the *svec PSD cone*. In svec space, we use the usual vector inner product, and $\mathcal{S}$ is also self-dual. The invertible linear transformations for an smat-space point $\boldsymbol{T} \in \mathbb{S}^n$ and an svec-space point $\boldsymbol{t} \in \mathbb{R}^{n(n+1)/2}$ are:

$$\operatorname{svec}(\boldsymbol{T}) = (T_{1,1}, \sqrt{2}T_{2,1}, \dots, \sqrt{2}T_{n,1}, T_{2,2}, \sqrt{2}T_{3,2}, \dots, T_{n,n}), \tag{38a}$$

$$\operatorname{smat}(\boldsymbol{t}) = \begin{bmatrix} t_1 & t_1/\sqrt{2} & \cdots & t_n/\sqrt{2} \\ t_2/\sqrt{2} & t_{n+1} & \cdots & t_{2n-1}/\sqrt{2} \\ \vdots & \vdots & \ddots & \vdots \\ t_n/\sqrt{2} & t_{n-1}/\sqrt{2} & \cdots & t_{n(n+1)/2} \end{bmatrix}. \tag{38b}$$

A.3.1 Initial Fixed Polyhedral Relaxation

Suppose we have a primitive cone constraint $\boldsymbol{T} \in \mathbb{S}^n_+$. Let $\boldsymbol{e}(i) \in \mathbb{R}^n$ be the ith unit vector in n dimensions. For each $i \in [\![n]\!]$, we let $\boldsymbol{\omega} = \boldsymbol{e}(i)$ in the extreme ray $\mathbb{S}^*_+$ cut (37), which imposes the diagonal nonnegativity condition $T_{i,i} \geq 0$ necessary for PSDness. For each $i, j \in [\![n]\!] : i > j$, we let $\boldsymbol{\omega} = \boldsymbol{e}(i) \pm \boldsymbol{e}(j)$ in (37), which enforces the condition $T_{i,i} + T_{j,j} \geq 2|T_{i,j}|$ necessary for PSDness. Ahmadi and Hall [2015] discuss an LP inner approximation of the PSD cone called the cone of diagonally dominant (DD) matrices. Our initial fixed $\mathbb{S}^*_+$ points $\boldsymbol{\omega}\boldsymbol{\omega}^T$ are exactly the extreme rays of the DD cone, so our initial fixed OA is the dual cone of the DD cone.

A.3.2 Extreme Ray Disaggregation

Suppose we have the $\mathbb{S}^*_+$ point $\boldsymbol{W}$, not necessarily and extreme ray of $\mathbb{S}^*_+$. We perform an eigendecomposition $\boldsymbol{W} = \sum_{i \in [\![n]\!]} \lambda_i \tilde{\boldsymbol{\omega}}_i \tilde{\boldsymbol{\omega}}_i^T$, where for all $i \in [\![n]\!]$, λ_i is the ith eigenvalue and $\tilde{\boldsymbol{\omega}}_i$ is its corresponding eigenvector.[41] Since $\boldsymbol{W}$ is PSD,

[41] Note that for real symmetric matrices, all eigenvalues are real. We select the eigenvectors to be orthonormal.

every eigenvalue is nonnegative, and there are $\operatorname{rank}(\boldsymbol{W}) \leq n$ positive eigenvalues. For each $i \in [\![n]\!] : \lambda_i > 0$, we let $\boldsymbol{\omega} = \sqrt{\lambda_i}\tilde{\boldsymbol{\omega}}_i$ in (37).[42] These extreme ray $\mathbb{S}_+^*$ cuts aggregate to imply the original $\mathbb{S}_+^*$ cut $\langle \boldsymbol{W}, \boldsymbol{T} \rangle \geq 0$.

A.3.3 Separation Of An Infeasible Point

Suppose we want to separate a point $\boldsymbol{T} \in \mathbb{S}^n \backslash \mathbb{S}_+^n$. We perform an eigendecomposition $\boldsymbol{T} = \sum_{i \in [\![n]\!]} \lambda_i \boldsymbol{\tau}_i \boldsymbol{\tau}_i^T$, for which at least one eigenvalue is negative. For each $i \in [\![n]\!] : \lambda_i < 0$, we let $\boldsymbol{\omega} = \boldsymbol{\tau}_i$ in (37) (note $\langle \boldsymbol{\tau}_i \boldsymbol{\tau}_i^T, \boldsymbol{T} \rangle = \boldsymbol{\tau}_i^T \boldsymbol{T} \boldsymbol{\tau}_i = \lambda_i < 0$).

B The Second-Order Cone Extended Formulation

Recall the definitions of the second-order cone $\mathcal{L}$ and the rotated-second-order cone $\mathcal{V}$ in Appendix A.2. Both $\mathcal{L}$ and $\mathcal{V}$ are self-dual. As discussed in Section 5.4, Pajarito can optionally use an *extended formulation* (EF) for second-order cone constraints, leading to tighter polyhedral relaxations. Vielma et al. [2017] show that the constraint $(r, \boldsymbol{t}) \in \mathcal{L}^{1+n}$ is equivalent to the following $1+n$ constraints on r, $\boldsymbol{t}$, and the auxiliary variables $\boldsymbol{\pi} \in \mathbb{R}^n$:

$$\sum_{i \in [\![n]\!]} 2\pi_i \leq r \tag{39a}$$

$$(r, \pi_i, t_i) \in \mathcal{V}^3 \qquad \forall i \in [\![n]\!]. \tag{39b}$$

By projecting out the $\boldsymbol{\pi}$ variables, the equivalence is obvious. Constraints (39b) imply $r \geq 0$ and $\pi_i \geq 0$ and $2r\pi_i \geq t_i^2$ for all $i \in [\![n]\!]$. Aggregating the latter conditions and using the linear inequality (39a), we see $r^2 \geq \sum_{i \in [\![n]\!]} 2r\pi_i \geq \sum_{i \in [\![n]\!]} t_i^2$, which is equivalent to the original constraint (for $r \geq 0$). We only use $\mathcal{L}^{1+n}$ and $\mathcal{V}^3$ here, so for convenience we drop the dimensions.[43]

Suppose $(u, \boldsymbol{w})$ is a $\mathcal{L}^*$ extreme ray, so from Appendix A.2.2, we have $\boldsymbol{w} \neq \boldsymbol{0}$ and $u = \|\boldsymbol{w}\|_2 > 0$. Then $ur + \boldsymbol{w}^T\boldsymbol{t} \geq 0$ is a $\mathcal{L}^*$ cut. Note that the linear constraint (39a) in the EF implies:

$$ur + \boldsymbol{w}^T\boldsymbol{t} \geq \frac{ur}{2} + u\sum_{i \in [\![n]\!]} \pi_i + \boldsymbol{w}^T\boldsymbol{t} = \sum_{i \in [\![n]\!]} \left(\frac{w_i^2 r}{2u} + u\pi_i + w_i t_i \right). \tag{40}$$

For each $i \in [\![n]\!]$, consider the $\mathcal{V}^*$ extreme ray $(w_i^2/2u, u, w_i)$, which implies a $\mathcal{V}^*$ cut for the ith constraint (39b) in the EF. The RHS of (40) is an aggregation of these n $\mathcal{V}^*$ cuts, which means the $\mathcal{V}^*$ cuts imply the $\mathcal{L}^*$ cut condition $ur + \boldsymbol{w}^T\boldsymbol{t} \geq 0$. Therefore, there is no loss of strength in the polyhedral relaxations, and we maintain the certificate $\mathcal{K}^*$ cut guarantees from Section 3.1.[44]

[42]By dropping any $i \in [\![n]\!] : \lambda_i < 0$, this projects $\boldsymbol{W} \in \mathbb{S} \backslash \mathbb{S}_+^*$ onto $\mathbb{S}_+^*$.

[43]Ben-Tal and Nemirovski [2001b] introduced an alternative extended formulation for the $\mathcal{L}^{1+n}$. See Vielma et al. [2017] for a discussion and computational comparison of various $\mathcal{L}^{1+n}$ extended formulations in the context of a separation-based B&B-OA algorithm for MISOCP.

[44]Without the ability to rescale the linear constraint (39a), we cannot recover the guarantees under an LP solver with a feasibility tolerance from Section 3.2. However, Pajarito heuristically scales up each $\mathcal{V}^*$ point by a factor n.

We now apply this lifting procedure to the initial fixed $\mathcal{L}^*$ points described in Appendix A.2.1. The $\mathcal{L}^*$ points for the ℓ_∞-norm relaxation are $(1, \pm\boldsymbol{e}(i)), \forall i \in [\![n]\!]$; for each $i \in [\![n]\!]$, we get three unique $\mathcal{V}^*$ extreme rays $(0,1,0)$ (for $w_i = 0$) and $(1/2, 1, \pm 1)$ (for $w_i = \pm 1$). The $\mathcal{L}^*$ points for the ℓ_1-norm relaxation are $(1, \boldsymbol{\sigma}/\sqrt{n}), \forall \boldsymbol{\sigma} \in \{-1,1\}^n$; for each $i \in [\![n]\!]$, we get two unique $\mathcal{V}^*$ extreme rays $(1/2n, 1, \pm 1/\sqrt{n})$ (for $w_i = \pm 1/\sqrt{n}$). The polyhedral relaxation implied by these $5n$ $\mathcal{V}^*$ points in the EF (39a) and (39b) is at least as strong as that implied by the $2n + 2^n$ $\mathcal{L}^*$ points from Appendix A.2.1, so our initial fixed OA can be imposed much more economically with the EF.

C SOCP Outer Approximation For PSD Cones

Recall the definitions of the self-dual smat-space PSD cone $\mathbb{S}_+$ in Appendix A.3 and the self-dual rotated-second-order cone $\mathcal{V}$ in Appendix A.2.[45] For a primitive cone constraint $\boldsymbol{T} \in \mathbb{S}^n_+$, we demonstrate how to strengthen an $(\mathbb{S}^n_+)^*$ extreme ray cut $\langle \boldsymbol{\omega}\boldsymbol{\omega}^T, \boldsymbol{T} \rangle \geq 0$ to up to n different $\mathcal{V}^3$ constraints. As discussed in Section 5.4, Pajarito can optionally solve an MISOCP OA model including these $\mathcal{V}^3$ constraints, leading to tighter relaxations of a challenging $\mathbb{S}^n_+$ constraint.

Fix the index $i \in [\![n]\!]$. Let $\underline{\omega} = \omega_i$ be the ith element of $\boldsymbol{\omega}$, and $\underline{\boldsymbol{\omega}} = (\omega_j)_{j \in [\![n]\!] \setminus \{i\}} \in \mathbb{R}^{n-1}$ be the (column) subvector of $\boldsymbol{\omega}$ with the ith element removed. Similarly, let $\underline{t} = T_{i,i}$ and $\underline{\boldsymbol{t}} = (T_{i,j})_{j \in [\![n]\!] \setminus \{i\}} \in \mathbb{R}^{n-1}$, and let $\underline{\boldsymbol{T}} = (T_{k,j})_{k,j \in [\![n]\!] \setminus \{i\}} \in \mathbb{S}^{n-1}$ be the submatrix of $\boldsymbol{T}$ with the ith column and row removed. Kim et al. [2003] prove a variant of the standard Schur-complement result that $\boldsymbol{T} \in \mathbb{S}^n_+$ if and only if $\underline{t} \geq 0$ and:

$$\underline{\boldsymbol{T}} \in \mathbb{S}^{n-1}_+ \tag{41a}$$

$$\underline{t}\underline{\boldsymbol{T}} - \underline{\boldsymbol{t}}\underline{\boldsymbol{t}}^T \in \mathbb{S}^{n-1}_+. \tag{41b}$$

Consider the 3-dimensional rotated second-order cone constraint:

$$(\underline{t}, \underline{\boldsymbol{\omega}}^T \underline{\boldsymbol{T}} \underline{\boldsymbol{\omega}}, \sqrt{2}\underline{\boldsymbol{\omega}}^T \underline{\boldsymbol{t}}) \in \mathcal{V}^3. \tag{42}$$

By the definition of $\mathcal{V}$, constraint (42) is equivalent to the conditions $\underline{t} \geq 0$ and:

$$\underline{\boldsymbol{\omega}}^T \underline{\boldsymbol{T}} \underline{\boldsymbol{\omega}} \geq 0 \tag{43a}$$

$$\underline{\boldsymbol{\omega}}^T (\underline{t}\underline{\boldsymbol{T}}) \underline{\boldsymbol{\omega}} \geq (\underline{\boldsymbol{\omega}}^T \underline{\boldsymbol{t}})^2. \tag{43b}$$

Condition (41a) implies condition (43a), by the dual cone definition (5). Since $(\underline{\boldsymbol{\omega}}^T \underline{\boldsymbol{t}})^2 = \underline{\boldsymbol{\omega}}^T \underline{\boldsymbol{t}}\underline{\boldsymbol{t}}^T \underline{\boldsymbol{\omega}}$, condition (43b) is equivalent to $\underline{\boldsymbol{\omega}}^T (\underline{t}\underline{\boldsymbol{T}} - \underline{\boldsymbol{t}}\underline{\boldsymbol{t}}^T) \underline{\boldsymbol{\omega}} \geq 0$. Thus, by the dual cone definition again, condition (41b) implies condition (43b). Therefore, constraint (42) is a valid relaxation of the PSD constraint $\boldsymbol{T} \in \mathbb{S}^n_+$. Furthermore, from Theorem 3.3 of Kim et al. [2003], constraint (42) holds if and only if:

$$\langle \boldsymbol{W}, \boldsymbol{T} \rangle \geq 0 \qquad \forall \boldsymbol{W} \in \mathbb{S}^n_+ : (W_{k,j} = \omega_k \omega_j, \forall k, j \in [\![n]\!] \setminus \{i\}). \tag{44}$$

[45] $\mathcal{V}^3$ is in fact a simple linear transformation of $\mathbb{S}^2_+$.

Thus constraint (42) potentially implies an infinite family of $(\mathbb{S}^n_+)^*$ cuts, including the original $(\mathbb{S}^n_+)^*$ cut $\langle \boldsymbol{\omega}\boldsymbol{\omega}^T, \boldsymbol{T} \rangle \geq 0$.[46] Note that the choice of $i \in [\![n]\!]$ is arbitrary, so we can derive n different $\mathcal{V}$ constraints of the form (42).[47]

We now apply this strengthening procedure to the initial fixed $(\mathbb{S}^n_+)^*$ extreme rays described in Appendix A.3.1. Letting $\boldsymbol{\omega} = \boldsymbol{e}(i) \pm \boldsymbol{e}(j)$ for each $i, j \in [\![n]\!] : j > i$ in constraint (42), we get the ${}^{n(n-1)}\!/_2$ initial fixed $\mathcal{V}$ constraints:

$$(T_{i,i}, T_{j,j}, \sqrt{2} T_{i,j}) \in \mathcal{V}^3 \qquad \forall i, j \in [\![n]\!] : j > i. \tag{45}$$

These constraints enforce that every 2×2 principal matrix of $\boldsymbol{T}$ is PSD, a necessary but insufficient condition for $\boldsymbol{T} \in \mathbb{S}^n_+$. Ahmadi and Hall [2015] discuss an SOCP inner approximation of the PSD cone called the cone of scaled diagonally dominant (SDD) matrices. Our initial fixed SOCP OA is the dual SDD matrix cone, a strict subset of the polyhedral dual DD matrix cone that corresponds to our initial fixed LP OA from Appendix A.3.1.

[46] We do not explore how to scale these $\mathcal{V}$ constraints to recover the guarantees from Section 3.2 for an SOCP solver with an absolute feasibility tolerance.

[47] For strengthening separation or certificate $\mathcal{K}^*$ cuts, Pajarito heuristically picks one of the n possible $\mathcal{V}$ constraints by choosing i as the coordinate of the largest absolute value in $\boldsymbol{\omega}$.

Bibliography

T. Achterberg. SCIP: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.

A. A. Ahmadi and G. Hall. Sum of squares basis pursuit with linear and second order cone programming. *arXiv preprint arXiv:1510.01597*, 2015.

P. Belotti, C. Kirches, S. Leyffer, J. Linderoth, J. Luedtke, and A. Mahajan. Mixed-integer nonlinear optimization. *Acta Numerica*, 22:1–131, 2013. ISSN 1474-0508.

A. Ben-Tal and A. Nemirovski. *Lectures on Modern Convex Optimization.* Society for Industrial and Applied Mathematics, 2001a.

A. Ben-Tal and A. Nemirovski. On polyhedral approximations of the second-order cone. *Mathematics of Operations Research*, 26(2):193–205, 2001b.

J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017.

P. Bonami, L. T. Biegler, A. R. Conn, G. Cornuéjols, I. E. Grossmann, C. D. Laird, J. Lee, A. Lodi, F. Margot, N. Sawaya, and A. Wächter. An algorithmic framework for convex mixed integer nonlinear programs. *Discrete Optimization*, 5(2):186–204, 2008. ISSN 1572-5286.

P. Bonami, M. Kılınç, and J. Linderoth. Algorithms and software for convex mixed integer nonlinear programs. In J. Lee and S. Leyffer, editors, *Mixed Integer Nonlinear Programming*, volume 154 of *The IMA Volumes in Mathematics and its Applications*, pages 1–39. Springer New York, 2012. ISBN 978-1-4614-1926-6.

B. Borchers. Csdp, ac library for semidefinite programming. *Optimization methods and Software*, 11(1-4):613–623, 1999.

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

S. Diamond and S. Boyd. CVXPY: A python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002. ISSN 0025-5610.

A. Domahidi, E. Chu, and S. Boyd. ECOS: An SOCP solver for embedded systems. In *European Control Conference (ECC)*, pages 3071–3076, 2013.

S. Drewes and S. Ulbrich. Subgradient based outer approximation for mixed integer second order cone programming. In J. Lee and S. Leyffer, editors, *Mixed Integer Nonlinear Programming*, volume 154 of *The IMA Volumes in Mathematics and its Applications*, pages 41–59. Springer New York, 2012. ISBN 978-1-4614-1926-6. doi: 10.1007/978-1-4614-1927-3_2.

I. Dunning, J. Huchette, and M. Lubin. JuMP: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320, 2017. doi: 10.1137/15M1020575.

H. A. Friberg. CBLIB 2014: a benchmark library for conic mixed-integer and continuous optimization. *Mathematical Programming Computation*, 8(2):191–214, 2016. ISSN 1867-2957.

T. Gally, M. E. Pfetsch, and S. Ulbrich. A framework for solving mixed-integer semidefinite programs. *Optimization Methods and Software*, 33(3):594–632, 2018. doi: 10.1080/10556788.2017.1322081.

N. Gould and J. Scott. A note on performance profiles for benchmarking software. *ACM Transactions on Mathematical Software*, 43(2), 2016. ISSN 0098-3500.

M. Grant and S. Boyd. Cvx: Matlab software for disciplined convex programming, version 2.1. cvxr.com/cvx, 2014.

M. Grant, S. Boyd, and Y. Ye. Disciplined convex programming. In L. Liberti and N. Maculan, editors, *Global Optimization*, volume 84 of *Nonconvex Optimization and Its Applications*, pages 155–210. Springer US, 2006. ISBN 978-0-387-28260-2.

O. Günlük and J. Linderoth. Perspective reformulation and applications. In J. Lee and S. Leyffer, editors, *Mixed Integer Nonlinear Programming*, volume 154 of *The IMA Volumes in Mathematics and its Applications*, pages 61–89. Springer New York, 2012. ISBN 978-1-4614-1926-6.

S. Kim, M. Kojima, and M. Yamashita. Second order cone programming relaxation of a positive semidefinite constraint. *Optimization Methods and Software*, 18(5):535–541, 2003.

O. Kröger, C. Coffrin, H. Hijazi, and H. Nagarajan. Juniper: An open-source nonlinear branch-and-bound solver in julia, 2018.

S. Leyffer. *Deterministic Methods for Mixed Integer Nonlinear Programming*. PhD thesis, University of Dundee, 12 1993.

M. Lubin and I. Dunning. Computing in operations research using Julia. *INFORMS Journal on Computing*, 27(2):238–248, 2015.

M. Lubin, E. Yamangil, R. Bent, and J. P. Vielma. Extended formulations in mixed-integer convex programming. In Q. Louveaux and M. Skutella, editors, *Integer Programming and Combinatorial Optimization: 18th International Conference, IPCO 2016, Liège, Belgium, June 1-3, 2016, Proceedings*, pages 102–113. Springer International Publishing, 2016. ISBN 978-3-319-33461-5. doi: 10.1007/978-3-319-33461-5_9.

M. Lubin, I. Zadik, and J. P. Vielma. Mixed-integer convex representability. In F. Eisenbrand and J. Könemann, editors, *Integer Programming and Combinatorial Optimization - 19th International Conference, IPCO 2017, Waterloo, ON, Canada, June 26-28, 2017, Proceedings*, volume 10328 of *Lecture Notes in Computer Science*, pages 392–404. Springer, 2017a. ISBN 978-3-319-59249-7. doi: 10.1007/978-3-319-59250-3_32. URL `https://doi.org/10.1007/978-3-319-59250-3_32`.

M. Lubin, I. Zadik, and J. P. Vielma. Regularity in mixed-integer convex representability. *arXiv preprint arXiv:1706.05135*, 2017b.

M. Lubin, E. Yamangil, R. Bent, and J. P. Vielma. Polyhedral approximation in mixed-integer convex optimization. *Mathematical Programming*, 172:139–168, 2018.

Mosek ApS. Modeling Cookbook revision 2.0.1. docs.mosek.com/MOSEKModeling Cookbook-letter.pdf accessed 2017-04-12, 2016.

H. Nagarajan, M. Lu, S. Wang, R. Bent, and K. Sundar. An adaptive, multivariate partitioning algorithm for global optimization of nonconvex programs. *arXiv preprint arXiv:1707.02514*, 2017.

B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd. Conic optimization via operator splitting and homogeneous self-dual embedding. *Journal of Optimization Theory and Applications*, 169(3):1042–1068, 2016. ISSN 1573-2878. doi: 10.1007/s10957-016-0892-3. URL `http://dx.doi.org/10.1007/s10957-016-0892-3`.

P. A. Parrilo. Semidefinite programming relaxations for semialgebraic problems. *Mathematical programming*, 96(2):293–320, 2003.

F. Permenter, H. A. Friberg, and E. D. Andersen. Solving conic optimization problems via self-dual embedding and facial reduction: a unified approach. *Optimization Online, September*, 2015.

I. Quesada and I. Grossmann. An LP/NLP based branch and bound algorithm for convex MINLP optimization problems. *Computers & Chemical Engineering*, 16(10):937–947, 1992. ISSN 0098-1354. doi: 10.1016/0098-1354(92)80028-8. URL `www.sciencedirect.com/science/article/pii/0098135492800288`.

M. Saltzman, L. Ladáanyi, and T. Ralphs. The COIN-OR Open Solver Interface: Technology overview. Presented at CORS/INFORMS Banff., 5 2004. URL `https://www.coin-or.org/Presentations/CORS2004-OSI.pdf`.

S. A. Serrano. *Algorithms for unsymmetric cone optimization and an implementation for problems with the exponential cone.* PhD thesis, Stanford University, 2015.

M. Udell, K. Mohan, D. Zeng, J. Hong, S. Diamond, and S. Boyd. Convex optimization in Julia. In *Proceedings of HPTCDL '14*, pages 18–28, Piscataway, NJ, USA, 2014. IEEE Press. ISBN 978-1-4799-7020-9.

J. P. Vielma. Small and strong formulations for unions of convex sets from the cayley embedding. *Mathematical Programming*, Mar 2018. ISSN 1436-4646. doi: 10.1007/s10107-018-1258-4. URL `https://doi.org/10.1007/s10107-018-1258-4`.

J. P. Vielma, I. Dunning, J. Huchette, and M. Lubin. Extended formulations in mixed integer conic quadratic programming. *Mathematical Programming Computation*, 9(3):369–418, Sep 2017. ISSN 1867-2957. doi: 10.1007/s12532-016-0113-y. URL `https://doi.org/10.1007/s12532-016-0113-y`.

S. Vigerske. MINLPLIB2 library. www.gamsworld.org/minlp/minlplib2/html/ accessed 2016-05-13, 2018.

J. Witzig, T. Berthold, and S. Heinz. Experiments with conflict analysis in mixed integer programming. In *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 211–220. Springer, 2017.

S. Zhang. Lecture 4: The dual cone and dual problem. University Lecture, 2014. URL `http://www.isye.umn.edu/courses/ie8534/pdf/Lecture-4.pdf`.