Mathematical Programming Computation (2019) 11:311–340
https://doi.org/10.1007/s12532-018-0152-7

FULL LENGTH PAPER

Simplex QP-based methods for minimizing a conic quadratic objective over polyhedra

Alper Atamtürk[1] · **Andrés Gómez**[2]

Received: 17 June 2017 / Accepted: 28 September 2018 / Published online: 3 December 2018

Abstract

We consider minimizing a conic quadratic objective over a polyhedron. Such problems arise in parametric value-at-risk minimization, portfolio optimization, and robust optimization with ellipsoidal objective uncertainty; and they can be solved by polynomial interior point algorithms for conic quadratic optimization. However, interior point algorithms are not well-suited for branch-and-bound algorithms for the discrete counterparts of these problems due to the lack of effective warm starts necessary for the efficient solution of convex relaxations repeatedly at the nodes of the search tree. In order to overcome this shortcoming, we reformulate the problem using the perspective of the quadratic function. The perspective reformulation lends itself to simple coordinate descent and bisection algorithms utilizing the simplex method for quadratic programming, which makes the solution methods amenable to warm starts and suitable for branch-and-bound algorithms. We test the simplex-based quadratic programming algorithms to solve convex as well as discrete instances and compare them with the state-of-the-art approaches. The computational experiments indicate that the proposed algorithms scale much better than interior point algorithms and return higher precision solutions. In our experiments, for large convex instances, they provide up to 22x speed-up. For smaller discrete instances, the speed-up is about 13x over a barrier-based branch-and-bound algorithm and 6x over the LP-based branch-and-bound algorithm with extended formulations. The software that was reviewed as part of this submission was given the Digital Object identifier https://doi.org/10.5281/zenodo.1489153.

✉ Alper Atamtürk
atamturk@berkeley.edu

Andrés Gómez
agomez@pitt.edu

[1] Industrial Engineering & Operations Research, University of California, Berkeley, CA 94720-1777, USA

[2] Department of Industrial Engineering, Swanson School of Engineering, University of Pittsburgh, Pittsburgh, PA 15261-3077, USA

Keywords Simplex method · Conic quadratic optimization · Quadratic programming · Warm starts · Value-at-risk minimization · Portfolio optimization · Robust optimization

Mathematics Subject Classification 90C20 · 90C49 · 90C11

1 Introduction

Consider the minimization of a conic quadratic function over a polyhedron, i.e.,

$$\text{(CO)} \quad \min_{x\in\mathbb{R}^n} \left\{ c'x + \Omega\sqrt{x'Qx} : x \in X \right\},$$

where $c \in \mathbb{R}^n$, $Q \in \mathbb{R}^{n\times n}$ is a symmetric positive semidefinite matrix, $\Omega > 0$, and $X \subseteq \mathbb{R}^n$ is a rational polyhedron. We denote by CDO the discrete counterpart of CO with integrality restrictions: $X \cap \mathbb{Z}^n$. CO and CDO are frequently used to model utility with uncertain objectives as in parametric value-at-risk minimization [25], portfolio optimization [5], and robust counterparts of linear programs with an ellipsoidal objective uncertainty set [13,14,16].

Note that CO includes linear programming (LP) and convex quadratic programming (QP) as special cases. The simplex method [22,39,41] is still the most widely used algorithm for LP and QP, despite the fact that polynomial interior point algorithms [28,32,34] are competitive with the simplex method in many large-scale instances. Even though non-polynomial, the simplex method has some distinct advantages over interior point methods. Since the simplex method iterates over bases, it is possible to carry out the computations with high accuracy and little cost, while interior point methods come with a trade-off between precision and efficiency. Moreover, an optimal basis returned by the simplex method is useful for sensitivity analysis, while interior point methods do not produce such a basis unless an additional "crashing" procedure is performed [e.g. 31]. Finally, if the parameters of the problem change, re-optimization can often be done very fast with the simplex method starting from a primal or dual feasible basis, whereas warm starts with interior point methods have limitations [21, 42]. In particular, fast re-optimization with the dual simplex method is crucial when solving discrete optimization problems with a branch-and-bound algorithm.

CO is a special case of conic quadratic optimization [3,30], which can be solved by polynomial-time interior points algorithms [2,15,35]. Although CO can be solved by a general conic quadratic solver, we show in this paper that iterative QP algorithms scale much better. In particular, simplex-based QP algorithms allowing warm starts perform much faster than interior point methods for CO.

For the discrete counterpart CDO, a number of different approaches are available for the special case with a diagonal Q matrix: Ishii et al. [27] give a polynomial time for optimization over spanning trees; Bertsimas and Sim [17] propose an approximation algorithm that solves series of linear integer programs; Atamtürk and Narayanan [7] give a cutting plane algorithm utilizing the submodularity of the objective for the binary case; Atamtürk and Goméz [4] give nonlinear cuts for the mixed 0-1 case; Atamtürk and Narayanan [8] give a parametric $O(n^3)$ algorithm for the binary case

with a cardinality constraint. Maximization of the same objective over the binaries is $\mathcal{NP}$-hard [1].

The aforementioned approaches do not extend to the non-diagonal case or to general feasible regions, which are obviously $\mathcal{NP}$-hard as quadratic and linear integer optimization are special cases. The branch-and-bound algorithm is the method of choice for general CDO. However, branch-and-bound algorithms that repeatedly employ a nonlinear programming (NLP) solver at the nodes of the search tree are typically hampered by the lack of effective warm starts. Borchers and Mitchell [20] and Leyffer [29] describe NLP-based branch-and-bound algorithms, and they give methods that branch without solving the NLPs to optimality, reducing the computational burden for the node relaxations. On the other hand, LP-based branch-and-bound approaches employ linear outer approximations of the nonlinear terms. This generally results in weaker relaxations at the nodes, compared to the NLP approaches, but allows one to utilize warm starts with the simplex method. Therefore, one is faced with a trade-off between the strength of the node relaxations and the solve time per node. A key idea to strengthen the node relaxations, as noted by Tawarmalani and Sahinidis [37], is to use extended formulations. Atamtürk and Narayanan [6] describe mixed-integer rounding inequalities in an extended formulation for conic quadratic integer programming. Vielma et al. [40] use an extended formulation for conic quadratic optimization that can be refined during branch-and-bound, and show that an LP-based branch-and-bound using the extended formulations typically outperforms the NLP-based branch-and-bound algorithms. The reader is referred to Belotti et al. [12] for an excellent survey of the solution methods for mixed-integer nonlinear optimization.

In this paper, we reformulate CO through the perspective of the quadratic term and give algorithms that solve a sequence of closely related QPs. Utilizing the simplex method, the solution to each QP is used to warm start the next one in the sequence, resulting in a small number of simplex iterations and fast solution times. Moreover, we show how to incorporate the proposed approach in a branch-and-bound algorithm, efficiently solving the continuous relaxations to optimality at each node and employing warm starts with the dual simplex method. Our computational experiments indicate that the proposed approach outperforms the state-of-the-art algorithms for convex as well as discrete cases.

The rest of the paper is organized as follows. In Sect. 2 we give an alternative formulation for CO using the perspective function of the quadratic function. In Sect. 3 we present coordinate descent and accelerated bisection algorithms that solve a sequence of QPs. In Sect. 4 we provide computational experiments, comparing the proposed methods with state-of-the-art barrier and other algorithms.

2 Formulation

In this section we present a reformulation of CO using the perspective function of the quadratic term. Let $X = \{x \in \mathbb{R}^n : Ax = b,\ x \geq 0\}$ be the feasible region of problem CO. For convex quadratic $q(x) = x'Qx$, consider the function $h : \mathbb{R}^{n+1} \to \mathbb{R}_+ \cup \{\infty\}$ defined as

$$h(x,t) = \begin{cases} \frac{x'Qx}{t} & \text{if } t > 0, \\ 0 & \text{if } x'Qx = 0, t = 0, \\ +\infty & \text{otherwise.} \end{cases}$$

Observe that

$$\begin{aligned} &\min \left\{ c'x + \Omega\sqrt{x'Qx} : x \in X \right\} \\ &= \min \left\{ c'x + \frac{\Omega}{2}h(x,t) + \frac{\Omega}{2}t : x \in X,\ t = \sqrt{x'Qx} \right\} \\ &\geq \zeta, \end{aligned}$$

where

$$\text{(PO)} \quad \zeta = \min \left\{ c'x + \frac{\Omega}{2}h(x,t) + \frac{\Omega}{2}t : x \in X,\ t \geq 0 \right\}.$$

We will show that problems CO and PO have, in fact, the same optimal objective value and that there is a one-to-one correspondence between the optimal primal-dual pairs of both problems.

Proposition 1 *Problem PO is a convex optimization problem.*

Proof It suffices to observe that h is the closure of the *perspective function* $tq(x/t)$ of the convex quadratic function $q(x)$, and is therefore convex (e.g. [26, p.160]). Since all other objective terms and constraints of PO are linear, PO is a convex optimization problem. □

Proposition 2 *Problems CO and PO are equivalent.*

Proof If $t > 0$, the objective function of problem PO is continuous and differentiable, and since the feasible region is a polyhedron and the problem is convex, its KKT points are equivalent to its optimal solutions. The KKT conditions of PO are

$$\begin{aligned} Ax &= b,\ x \geq 0,\ t \geq 0 \\ -c' - \frac{\Omega}{t}x'Q &= \lambda'A - \mu \qquad (1) \\ \frac{\Omega}{2t^2}x'Qx - \frac{\Omega}{2} &= 0 \qquad (2) \\ \mu &\geq 0 \\ \mu'x &= 0, \end{aligned}$$

where λ and μ are the dual variables associated with constraints $Ax = b$ and $x \geq 0$, respectively. Note that $t > 0$ and (2) imply that $t = \sqrt{x'Qx}$. Substituting $t = \sqrt{x'Qx}$ in (1), one arrives at the equivalent conditions

$$
\begin{aligned}
Ax &= b,\ x \geq 0 \\
-c' - \frac{\Omega}{\sqrt{x'Qx}} x'Q &= \lambda' A - \mu \qquad (3) \\
t &= \sqrt{x'Qx} \qquad (4) \\
\mu &\geq 0 \\
\mu' x &= 0.
\end{aligned}
$$

Ignoring the redundant variable t and Eq. (4), we see that these are the KKT conditions of problem CO. Therefore, any optimal primal-dual pair for PO with $t > 0$ is an optimal primal-dual pair for CO. Similarly, we see that any optimal primal-dual pair of problem CO with $x'Qx > 0$ gives an optimal primal-dual pair of problem PO by setting $t = \sqrt{x'Qx}$. In both cases, the objective values match.

On the other hand, if $t = 0$, then PO reduces to problem

$$\min_{x \in \mathbb{R}^n} \left\{ c'x : Ax = b, x \geq 0, x'Qx = 0 \right\},$$

which corresponds to CO with $x'Qx = 0$, and hence they are equivalent. □

Proposition 2 indeed holds for more general problems; it is not necessary to have a polyhedral feasible set [9]. Since they are equivalent optimization problems, we can use PO to solve CO. In particular, we exploit the fact that, for a fixed value of t, PO reduces to a QP.

3 Algorithms

For simplicity, assume that PO has an optimal solution; hence, X is nonempty and may be assumed to be bounded. Consider the one-dimensional optimal value function

$$g(t) = \min_{x \in X} c'x + \frac{\Omega}{2} h(x, t) + \frac{\Omega}{2} t. \qquad (5)$$

As X is nonempty and bounded, g is real-valued and, by Proposition 1, it is convex. Throughout, $x(t)$ denotes an optimal solution to (5).

In this section we describe two algorithms for PO that utilize a QP oracle. The first one is a coordinate descent approach, whereas, the second one is an accelerated bisection search algorithm on the function g. Finally, we discuss how to exploit the warm starts with the simplex method to solve convex as well as discrete cases.

3.1 Coordinate descent algorithm

Algorithm 1 successively optimizes over x for a fixed value of t, and then optimizes over t for a fixed value of x. Observe that the optimization problem in line 4 over x

is a QP, and the optimization in line 5 over t has a closed form solution: by simply setting the derivative to zero, we find that $t_{i+1} = \sqrt{x_{i+1}{}' Q x_{i+1}}$.

Algorithm 1 Coordinate descent.

Input: X polyhedron; Q psd matrix; c cost vector; $\Omega > 0$
Output: Optimal solution x^*
1: **Initialize** $t_0 > 0$ ▷ e.g. $t_0 = 1$
2: $i \leftarrow 0$ ▷ iteration counter
3: **repeat**
4: $\quad x_{i+1} \leftarrow \arg\min_{x \in X} \left\{ c'x + \frac{\Omega}{2t_i} x' Q x + \frac{\Omega}{2} t_i \right\}$ ▷ solve QP
5: $\quad t_{i+1} \leftarrow \arg\min_{t \geq 0} \left\{ c' x_{i+1} + \frac{\Omega}{2t} x_{i+1}{}' Q x_{i+1} + \frac{\Omega}{2} t \right\}$ ▷ $t_{i+1} = \sqrt{x_{i+1}{}' Q x_{i+1}}$
6: $\quad i \leftarrow i + 1$
7: **until** stopping condition is met
8: **return** x_i

First observe that the sequence of objective values $\{c'x_i + \frac{\Omega}{2t_i} x_i' Q x_i + \frac{\Omega}{2} t_i\}_{i \in \mathbb{N}}$ is non-increasing. Moreover, the dual feasibility KKT conditions for the QPs in line 4 are of the form

$$-c' - \frac{\Omega}{t_i} x_{i+1}{}' Q = \lambda' A - \mu. \tag{6}$$

Let $\|\cdot\|$ be a norm and suppose that the QP oracle finds feasible primal-dual pairs with $\epsilon > 0$ tolerance with respect to $\|\cdot\|$. In particular x_{i+1} in line 4 violates (6) by at most ϵ, i.e.,

$$\left\| -c' - \frac{\Omega}{t_i} x_{i+1}{}' Q - \lambda' A + \mu \right\| \leq \epsilon.$$

Proposition 3 below states that, at each iteration of Algorithm 1, we can bound the violation of the dual feasibility condition (3) corresponding to the original problem CO. The bound depends only on the precision of the QP oracle ϵ, the relative change of t in the last iteration $\frac{\Delta_i}{t_i}$, where $\Delta_i = t_{i+1} - t_i$, and the gradient of the function $f(x) = \Omega\sqrt{x' Q x}$ evaluated at the new point x_{i+1}.

Proposition 3 *(Dual feasibility bound) A pair* (x_{i+1}, t_{i+1}) *in Algorithm 1 satisfies* $\| -c' - \Omega \frac{x_{i+1}' Q}{\sqrt{x_{i+1}{}' Q x_{i+1}}} - \lambda' A + \mu \| \leq \epsilon + \frac{|\Delta_i|}{t_i} \cdot \|\nabla f(x_{i+1})\|$

Proof

$$
\begin{aligned}
&\left\| -c' - \Omega \frac{x_{i+1}{}'Q}{\sqrt{x_{i+1}{}'Qx_{i+1}}} - \lambda' A + \mu \right\| \\
&= \left\| -c' - \Omega \frac{x_{i+1}{}'Q}{t_i + \Delta_i} - \lambda' A + \mu \right\| \\
&= \left\| -c' - \Omega \frac{x_{i+1}{}'Q}{t_i} - \Omega x_{i+1}{}'Q \left(\frac{1}{t_i + \Delta_i} - \frac{1}{t_i} \right) - \lambda' A + \mu \right\| \\
&= \left\| -c' - \Omega \frac{x_{i+1}{}'Q}{t_i} - \lambda' A + \mu + \Omega \left(\frac{\Delta_i}{t_i \cdot t_{i+1}} \right) x_{i+1}{}'Q \right\| \\
&\leq \epsilon + \left\| \Omega \frac{\Delta_i}{t_i} \cdot \frac{x_{i+1}{}'Q}{t_{i+1}} \right\| = \epsilon + \Omega \frac{|\Delta_i|}{t_i} \cdot \left\| \frac{x_{i+1}{}'Q}{\sqrt{x_{i+1}{}'Qx_{i+1}}} \right\|.
\end{aligned}
$$

□

Let t^* be a minimizer of g on $\mathbb{R}_+$. We now show that the sequence of values of t produced by Algorithm 1, $\{t_i\}_{i\in\mathbb{N}}$, is monotone and bounded by t^*.

Proposition 4 *(Monotonicity) If $t_i \leq t^*$, then $t_{i+1} = \sqrt{x_{i+1}{}'Qx_{i+1}}$ satisfies $t_i \leq t_{i+1} \leq t^*$. Similarly, if $t_i \geq t^*$, then $t_i \geq t_{i+1} \geq t^*$.*

Proof If $t_i \leq t^*$, then $\frac{\Omega}{2t_i} \geq \frac{\Omega}{2t^*}$. It follows that $x(t_{i+1})$ is a minimizer of an optimization problem with a larger coefficient for the quadratic term than $x(t^*)$, and therefore $x_{i+1}{}'Qx_{i+1} = t_{i+1}^2 \leq t^{*2} = x^{*\prime}Qx^*$, and $t_{i+1} \leq t^*$. Moreover, the inequality $t_i \leq t_{i+1}$ follows from the convexity of the one-dimensional function g and the fact that function g is minimized at t^*, and that $g(t_{i+1}) \leq g(t_i)$. The case $t_i \geq t^*$ is similar. □

Since the sequence $\{t_i\}_{i\in\mathbb{N}}$ is bounded and monotone, it converges to a supremum or infimum. Thus $\{t_i\}_{i\in\mathbb{N}}$ is a Cauchy sequence, and $\lim_{i\to\infty} \Delta_i = 0$. Corollaries 1 and 2 below state that Algorithm 1 converges to an optimal solution. The cases where there exists a KKT point for PO (i.e., there exists an optimal solution with $t^* > 0$) and where there are no KKT points are handled separately.

Corollary 1 (Convergence to a KKT point) *If PO has a KKT point, then Algorithm 1 converges to a KKT point.*

Proof By convexity, the set of optimal solutions to (5) is an interval, $[t_\ell, t_u]$. Since by assumption there exists a KKT point, we have that $t_u > 0$. The proof is by cases, depending on the value of t_0 in line 1 of Algorithm 1.

Case $t_\ell \leq t_0 \leq t_u$ Since t_0 is optimal, we have by Proposition 4 that $t_1 = t_0$. Since $\Delta_0 = 0$ and $t_0 = \sqrt{x'_{i+1}Qx_{i+1}} > 0$, we have that $\|\nabla f(x_{i+1})\| < \infty$ in Proposition 3, and $\frac{|\Delta_i|}{t_i} \cdot \|\nabla f(x_{i+1})\| = 0$.

Case $t_0 < t_\ell$ We have by Proposition 4 than for all $i \in \mathbb{N}$, $t_i = \sqrt{x_i' Q x_i} \geq t_0 > 0$. Therefore, there exists a number M such that $\frac{1}{t_i} \|\nabla f(x_{i+1})\| < M$ for all $i \in \mathbb{N}$, and we find that $\frac{|\Delta_i|}{t_i} \cdot \|\nabla f(x_{i+1})\| \xrightarrow{\Delta_i \to 0} 0$.

Case $t_0 > t_u$ We have by Proposition 4 than for all $i \in \mathbb{N}$, $t_i = \sqrt{x_i' Q x_i} \geq t_u > 0$. Therefore, there exists a number M such that $\frac{1}{t_i} \|\nabla f(x_{i+1})\| < M$ for all $i \in \mathbb{N}$, and we find that $\frac{|\Delta_i|}{t_i} \cdot \|\nabla f(x_{i+1})\| \xrightarrow{\Delta_i \to 0} 0$.

Therefore, in all cases, Algorithm 1 convergences to a KKT point by Proposition 3. □

Corollary 2 (Convergence to 0) *If $t^* = 0$ is the unique optimal solution to $\min\{g(t) : t \in \mathbb{R}_+\}$, then for any $\xi > 0$ Algorithm 1 finds a solution $(\bar{x}, \bar{t})$, where $\bar{t} < \xi$ and $\bar{x} \in \arg\min \left\{c'x : \sqrt{x'Qx} = \bar{t}, x \in X\right\}$.*

Proof The sequence $\{t_i\}_{i \in \mathcal{N}}$ converges to 0 (otherwise, by Corollary 1, it would converge to a KKT point). Thus, $\lim_{i \to \infty} \sqrt{x_i' Q x_i} = 0$ and all points obtained in line 4 of Algorithm 1 satisfy $x_{i+1} \in \arg\min \left\{c'x : \sqrt{x'Qx} = t_{i+1}, x \in X\right\}$. □

We now discuss how to initialize and terminate Algorithm 1, corresponding to lines 1 and 7, respectively.

3.1.1 Initialization

The algorithm may be initialized by an arbitrary $t_0 > 0$. Nevertheless, when a good initial guess on the value of t^* is available, t_0 should be set to that value. Moreover, observe that setting $t_0 = \infty$ results in a fast computation of x_1 by solving an LP.

3.1.2 Stopping condition

Proposition 3 suggests a good stopping condition for Algorithm 1. Given a desired dual feasibility tolerance of $\delta > \epsilon$, we can stop when $\epsilon + \frac{|\Delta_i|}{t_i} \cdot \|\nabla f(x_{i+1})\| < \delta$. Alternatively, if $\exists k$ s.t. $\max_{x \in X} \|\nabla f(x)\| \leq k < \infty$, then the simpler $\left|\frac{\Delta_i}{t_i}\right| \leq \frac{\delta - \epsilon}{k}$ is another stopping condition. For instance, a crude upper bound on $\|\nabla f(x)\| = \Omega \left\| \frac{x'Q}{\sqrt{x'Qx}} \right\|$ can be found by maximizing/minimizing the numerator $x'Q$ over X and minimizing $x'Qx$ over X. The latter minimization is guaranteed to have a nonzero optimal value if $0 \notin X$ and Q is positive definite.

Remark 1 Observe that the stopping condition above may fail if $t^* = 0$ is the unique optimal solution of $\min_{t \geq 0} g(t)$ (Corollary 2). This case happens only if Q is positive semi-definite (but not positive definite), or if $x^* = 0$ is the unique optimal solution of CO. Such situations rarely arise in practice and can often be ruled out *a priori*. Nonetheless, stopping Algorithm 1 also when $t_i \leq \xi$ for some small $\xi > 0$ ensures that the algorithm terminates (as specified in Corollary 2) even when $t^* = 0$.

3.2 Bisection algorithm

Algorithm 2 is an accelerated bisection approach to solve PO. The algorithm maintains lower and upper bounds, $t_{\min}$ and $t_{\max}$, on t^* and, at each iteration, reduces the interval $[t_{\min}, t_{\max}]$ by at least half. The algorithm differs from the traditional bisection search algorithm in lines 7–11, where it uses an acceleration step to reduce the interval by a larger amount: by Proposition 4, if $t_0 \leq t_1$ (line 7), then $t_0 \leq t_1 \leq t^*$, and therefore t_1 is a higher lower bound on t^* (line 8); similarly, if $t_0 \geq t_1$, then t_1 is an lower upper bound on t^* (lines 9 and 10). Intuitively, the algorithm takes a "coordinate descent" step as in Algorithm 1 after each bisection step. Preliminary computations show that the acceleration step reduces the number of steps as well as the overall solution time for the bisection algorithm by about 50%.

Algorithm 2 Accelerated bisection.

Input: X polyhedron; Q psd matrix; c cost vector; $\Omega > 0$
Output: Optimal solution x^*
1: **Initialize** $t_{\min}$ and $t_{\max}$ ▷ ensure $t_{\min} \leq t^* \leq t_{\max}$
2: $\hat{z} \leftarrow \infty$ ▷ best objective value found
3: **repeat**
4: $t_0 \leftarrow \frac{t_{\min}+t_{\max}}{2}$
5: $x_0 \leftarrow \arg\min_{x \in X} \left\{ c'x + \frac{\Omega}{2t_0} x'Qx + \frac{\Omega}{2} t_0 \right\}$ ▷ solve QP
6: $t_1 \leftarrow \sqrt{x_0{}'Qx_0}$
7: **if** $t_0 \leq t_1$ **then** ▷ accelerate bisection
8: $t_{\min} \leftarrow t_1$
9: **else**
10: $t_{\max} \leftarrow t_1$
11: **end if**
12: **if** $c'x_0 + \Omega\sqrt{x_0{}'Qx_0} \leq \hat{z}$ **then** ▷ update the incumbent solution
13: $\hat{z} \leftarrow c'x_0 + \Omega\sqrt{x_0{}'Qx_0}$
14: $\hat{x} \leftarrow x_0$
15: **end if**
16: **until** stopping condition is met
17: **return** $\hat{x}$

3.2.1 Initialization.

In line 1, $t_{\min}$ can be initialized to zero and $t_{\max}$ to ${x_{LP}}'Q x_{LP}$, where x_{LP} is an optimal solution to the LP relaxation $\min_{x \in X} c'x$.

3.2.2 Stopping condition.

There are different possibilities for the stopping criterion in line 16. Note that if we have numbers t_m and t_M such that $t_m \leq t^* \leq t_M$, then $c'x(t_M) + \Omega\sqrt{x(t_m)'Qx(t_m)}$ is a lower bound on the optimal objective value $c'x^* + \Omega\sqrt{x^{*\prime}Qx^*}$. Therefore, in line 5, a lower bound z_l on the objective function can be computed, and the algorithm can be stopped when the gap between $\hat{z}$ and z_l is smaller than a given threshold.

Alternatively, stopping when $\frac{|t_1-t_0|}{t_0} \cdot \Omega \left\| \frac{x_0' Q}{\sqrt{x_0' Q x_0}} \right\| < \delta - \epsilon$ provides a guarantee on the dual infeasibility as in Proposition 3.

3.3 Warm starts

Although any QP solver can be used to run the coordinate descent and bisection algorithms described in Sects. 3.1 and 3.2, simplex methods for QP are particularly effective as they allow warm starts for small changes in the model parameters in iterative applications. This is the main motivation for the QP based algorithms presented above.

3.3.1 Warm starts with primal simplex for convex optimization

All QPs solved in Algorithms 1, 2 have the same feasible region and only the objective function changes in each iteration. Therefore, an optimal basis for a QP is primal feasible for the next QP solved in the sequence, and can be used to warm start a primal simplex QP solver.

3.3.2 Warm starts with dual simplex for discrete optimization

When solving discrete counterparts of CO with a branch-and-bound algorithm one is particularly interested in utilizing warm starts in solving convex relaxations at the nodes of the search tree. In a branch-and-bound algorithm, children nodes typically have a single additional bound constraint compared to the parent node.

For this purpose, it is also possible to warm start Algorithm 1 from a dual feasible basis. Let (x^*, t^*) be an optimal solution to PO and B^* be an optimal basis. Consider a new problem

$$\min \left\{ c'x + \frac{\Omega}{2t} x' Q x + \frac{\Omega}{2} t : x \in \bar{X},\ t \geq 0 \right\}, \tag{7}$$

where the feasible set $\bar{X}$ is obtained from X by adding new constraints.

Note that B^* is a dual feasible basis for (7) when $t = t^*$. Therefore, Algorithm 1 to solve problem (7) can be warm started by initializing $t_0 = t^*$ and using B^* as the initial basis to compute x_1 with a dual simplex algorithm. The subsequent QPs can be solved using the primal simplex algorithm as noted in Sect. 3.3.1.

3.4 Special cases

The simplex method is a general algorithm that can be used with any polyhedron X and, as mentioned in Sect. 3.3, is well suited for solving the sequence of QPs. Nevertheless, for particular feasible regions, other specialized algorithms may be preferable. For instance, in the trivial unbounded case ($X = \mathbb{R}^n$), the QPs can be solved in closed

form. Another, more interesting, case is the problem

$$\min_{x\in\mathbb{R}^n} \sqrt{(x-y)'Q(x-y)} + \beta\|x\|_1, \tag{8}$$

where $y \in \mathbb{R}^n$ is fixed, $\|\cdot\|_1$ denotes the ℓ_1-norm and $\beta > 0$ is a regularization parameter. Note that (8) is a special case of CO with the usual linearization of the ℓ_1-norm. Problem (8) arises in compressed sensing [10] and sparse linear regression [11]), and is solved fast using first-order methods [33]. Note if Algorithm 1 or 2 is used instead, then every QP subproblem $\min_{x\in\mathbb{R}^n} \frac{(x-y)'Q(x-y)}{2t} + \beta\|x\|_1$ corresponds to the well-studied Lasso problem [38], for which efficient special algorithms exist. In particular, problem (8) can be solved with a *single* call to an algorithm that computes the regularization path (i.e., solves the problem for all β), such as Least Angle Regression [24].

4 Computational experiments

In this section we report on computational experiments with solving convex CO and its discrete counterpart CDO with the algorithms described in Sect. 3. The algorithms are implemented with CPLEX Java API. We use the simplex and barrier solvers of CPLEX version 12.6.2, as well as the barrier solver of MOSEK version 8.1.0 for the computational experiments. All experiments are conducted on a workstation with a 2.93GHz Intel®Core™ i7 CPU and 8 GB main memory using a single thread.

4.1 Test problems

We test the algorithms on two types of data sets. For the first set the feasible region is described by a cardinality constraint and bounds, i.e., $X = \left\{x \in \mathbb{R}^n : \sum_{i=1}^n x_i = b,\ \mathbf{0} \le x \le \mathbf{1}\right\}$ with $b = n/5$; problems with a cardinality constraint are common in finance [19] and statistics [18]. For the second data set the feasible region consists of the path polytope of an acyclic grid network; conic quadratic optimization over paths has been studied in [17,36], and similar substructures arise in more complex problems such as vehicle routing [23]. For discrete optimization problems we additionally enforce the binary restrictions $x \in \mathbb{B}^n$.

For both data sets the objective function $q(x) = c'x + \Omega\sqrt{x'Qx}$ is generated as follows: Given a rank parameter r and density parameter α, Q is the sum of a low rank factor matrix and a full rank diagonal matrix; that is, $Q = F\Sigma F' + D$, where

- D is an $n \times n$ diagonal matrix with entries drawn from Uniform(0, 1).
- $\Sigma = HH'$ where H is an $r \times r$ matrix with entries drawn from Uniform(-1, 1).
- F is an $n \times r$ matrix in which each entry is 0 with probability $1 - \alpha$ and drawn from Uniform(-1, 1) with probability α.

Note that the construction of matrix Q is consistent with factor models often used in finance. In particular, F is the factor exposure matrix, Σ is the factor covariance

matrix and D is the matrix of the residual variances. Each linear coefficient c_i is drawn from Uniform$(-2\sqrt{Q_{ii}}, 0)$.

4.2 Experiments with convex problems

In this section we present the computational results for convex instances. We compare the following algorithms:

ALG1 Algorithm 1.
ALG2 Algorithm 2.
BAR CPLEX barrier algorithm (the default solver in CPLEX for convex conic quadratic problems).
MOS MOSEK barrier algorithm.

For algorithms ALG1 and ALG2 we use CPLEX primal simplex algorithm as the QP solver.

4.2.1 Optimality tolerance

As the speed of the interior point methods crucially depends on the chosen optimality tolerance, it is prudent to first compare the speed vs the quality of the solutions for the algorithms tested. Here we study the impact of the optimality tolerance in the solution time and the quality of the solutions for CPLEX barrier algorithm BAR and simplex QP-based algorithm ALG1. The optimality tolerance of the barrier algorithm is controlled by the QCP convergence tolerance parameter ("BarQCPEpComp"), and in Algorithm 1, by the stopping condition $\frac{|\Delta_i|}{t} \leq \delta$.

In both cases, a smaller optimality tolerance corresponds to a higher quality solution. We evaluate the quality of a solution as $\texttt{optgap} = |(z_{\min} - z)/z_{\min}|$, where z is the objective value of the solution found by an algorithm with a given tolerance parameter and $z_{\min}$ is the objective value of the solution found by the barrier algorithm with tolerance 10^{-12} (minimum tolerance value allowed by CPLEX). Table 1 presents the results for different tolerance values for a 30×30 convex grid instance with $r = 200$, $\alpha = 0.1$, and $\Omega = 1$.

The table shows, for varying tolerance values and for each algorithm, the quality of the solution, the solution time in seconds, the number of iterations, and QPs solved (for ALG1). We highlight in bold the default tolerance used for the rest of the experiments presented in the paper. The tolerance value 10^{-7} for the barrier algorithm corresponds to the default parameter in CPLEX.

First observe that the solution time increases with reduced optimality tolerance for both algorithms. With lower tolerance, while the barrier algorithm performs more iterations, ALG1 solves more QPs; however, the total number of simplex iterations barely increases. For ALG1 the changes in the value of t are very small between QPs, and the optimal bases of the QPs are thus the same. Therefore, using warm starts, the simplex method is able to find high precision solutions inexpensively. ALG1 achieves much higher precision an order of magnitude faster than CPLEX barrier algorithm. For the default tolerance parameters used in our computational experiments, Algorithm 1 is several orders of magnitude more precise than the barrier algorithm.

Table 1 The effect of optimality tolerance

Tolerance	BAR			ALG1			
	optgap	time	#iter	optgap	time	#iter	#QP
10^{-1}	8.65×10^{-2}	29.9	10	5.48×10^{-5}	3.2	835	4
10^{-2}	8.77×10^{-3}	41.5	15	3.24×10^{-7}	4.2	844	6
10^{-3}	6.98×10^{-4}	54.6	23	2.60×10^{-9}	4.3	844	8
10^{-4}	5.52×10^{-5}	62.9	27	2.12×10^{-11}	4.7	844	10
10^{-5}	3.72×10^{-6}	66.8	29	$\mathbf{6.80 \times 10^{-13}}$	**5.2**	**844**	**12**
10^{-6}	7.12×10^{-7}	69.6	30	5.32×10^{-13}	5.4	844	13
10^{-7}	$\mathbf{2.04 \times 10^{-8}}$	**72.0**	**32**	5.15×10^{-13}	6.0	844	15
10^{-8}	2.65×10^{-9}	74.0	33	5.15×10^{-13}	6.2	844	17
10^{-9}	2.42×10^{-10}	75.9	34	5.15×10^{-13}	6.6	844	19
10^{-10}	1.97×10^{-11}	78.7	35	5.15×10^{-13}	7.0	844	21
10^{-11}	9.61×10^{-12}	79.6	36	5.15×10^{-13}	7.4	844	23
10^{-12}	0	89.6	39	5.15×10^{-13}	7.8	844	25

4.2.2 Effect of the nonlinearity parameter Ω.

We now study the effect of changing the nonlinearity parameter Ω. Tables 2 and 3 show the total solution time in seconds, the total number of simplex or barrier iterations, and the number of QPs solved in cardinality (1000 variables) and path instances (1760 variables), respectively. Each row represents the average over five instances for a rank (r) and density (α) configuration and algorithm used. For each parameter choice the fastest algorithm is highlighted in bold. Figure 1 also shows the total number of instances solved within the given time limit for each instance class.

Observe that, compared to CPLEX barrier algorithm, the simplex QP-based methods are 3.5 and 6 times faster for the cardinality instances and up to 15 times faster for the path instances. Additionally, the simplex QP-based methods are two to three times faster than MOSEK barrier algorithm for the cardinality instances, and up to four times faster for the path instances. Figure 1 shows that the simplex QP-based methods solve most of the instances well within the time required for MOSEK barrier algorithm to solve the easiest instance.

The barrier algorithms do not appear to be too sensitive to the nonlinearity parameter Ω, whereas the simplex QP-based methods are faster for smaller Ω. The number of simplex iterations in ALG1 increases with the nonlinearity parameter Ω. Indeed, the initial problem solved by ALG1 is an LP (corresponding to $\Omega = 0$), so as Ω increases the initial problem becomes a worse approximation, and more work is needed to converge to an optimal solution. Also note that Algorithm 2 requires fewer QPs to be solved, but as a result it benefits less from warm starts (it requires more simplex iterations per QP than ALG1). Indeed, in ALG2 the value of t changes by a larger amount at each iteration (with respect to ALG1), so the objective function of two

Table 2 The effect of nonlinearity (cardinality instances)

r	α	Method	$\Omega = 1$			$\Omega = 2$			$\Omega = 3$			$\Omega = 4$		
			time	#iter	#QP	time	#iter	#QP	time	#iter	#QP	time	#iter	#QP
100	0.1	ALG1	1.0	22	20	1.1	53	24	1.3	104	29	1.4	123	26
		ALG2	**0.8**	**41**	**14**	**0.9**	**95**	**15**	**0.9**	**150**	**15**	**1.1**	**219**	**16**
		BAR	4.6	16	–	4.9	24	–	5.2	26	–	5.1	25	–
		MOS	2.1	10	–	2.4	11	–	2.2	10	–	2.4	11	–
100	0.5	ALG1	1.1	33	23	1.1	69	24	1.5	144	37	1.6	192	30
		ALG2	**0.8**	**60**	**14**	**0.9**	**125**	**15**	**0.9**	**200**	**15**	**1.1**	**251**	**16**
		BAR	4.5	21	–	5.1	25	–	5.8	29	–	5.7	29	–
		MOS	2.2	10	–	2.2	10	–	2.4	10	–	2.4	11	–
200	0.1	ALG1	0.9	33	19	1.1	73	25	1.2	110	25	1.4	157	26
		ALG2	**0.8**	**49**	**14**	**0.9**	**126**	**14**	**0.9**	**172**	**14**	**1.1**	**259**	**15**
		BAR	4.7	22	–	4.5	22	–	5.1	25	–	5.3	27	–
		MOS	2.4	11	–	2.6	12	–	2.5	11	–	2.4	11	–
200	0.5	ALG1	1.0	48	22	1.1	99	22	1.2	151	25	1.6	218	24
		ALG2	**0.9**	**94**	**14**	**0.9**	**179**	**14**	**1.0**	**233**	**15**	**1.3**	**326**	**15**
		BAR	4.4	21	–	4.9	24	–	5.2	26	–	5.8	31	–
		MOS	2.3	10	–	2.4	10	–	2.4	11	–	2.6	11	–
Avg		ALG1	1.0	34	21	1.1	73	24	1.3	127	29	1.5	173	27
		ALG2	**0.8**	**61**	**14**	**0.9**	**131**	**15**	**0.9**	**189**	**15**	**1.2**	**264**	**15**
		BAR	4.3	20	–	4.9	24	–	5.3	27	–	5.5	28	–
		MOS	2.3	10	–	2.4	11	–	2.4	11	–	2.4	11	–

Table 3 The effect of nonlinearity (path instances)

r	α	Method	$\Omega = 1$			$\Omega = 2$			$\Omega = 3$			$\Omega = 4$		
			time	#iter	#QP	time	#iter	#QP	time	#iter	#QP	time	#iter	#QP
100	0.1	ALG1	**4.9**	**940**	**12**	**7.0**	**1307**	**16**	8.5	1505	18	10.5	1756	21
		ALG2	5.4	1283	11	**7.0**	**1637**	**13**	**8.4**	**1865**	**14**	**9.7**	**2375**	**13**
		BAR	81.1	26	–	64.3	21	–	55.3	16	–	56.8	16	–
		MOS	19.4	19	–	16.8	17	–	15.6	18	–	15.8	19	–
100	0.5	ALG1	**5.2**	**902**	**14**	8.2	1191	21	9.3	1391	21	11.0	1641	21
		ALG2	5.5	1148	12	**7.2**	**1474**	**13**	**8.6**	**1772**	**14**	**9.6**	**2020**	**14**
		BAR	62.7	19	–	56.0	16	–	57.1	16	–	57.7	16	–
		MOS	17.4	17	–	17.8	18	–	15.7	19	–	14.9	17	–
200	0.1	ALG1	**4.9**	**836**	**14**	**6.3**	**1053**	**15**	**8.3**	**1220**	**18**	14.4	1429	17
		ALG2	**4.9**	**932**	**12**	6.8	1377	13	8.4	1671	13	**12.4**	**1833**	**13**
		BAR	76.8	25	–	60.1	18	–	66.0	20	–	128.3	21	–
		MOS	16.2	17	–	16.1	18	–	15.5	18	–	15.2	18	–
200	0.5	ALG1	**4.5**	**858**	**12**	**6.2**	**1048**	**15**	**7.6**	**1237**	**16**	**12.7**	**1387**	**18**
		ALG2	4.9	978	12	6.8	1363	13	8.5	1626	13	15.9	1794	14
		BAR	83.1	26	–	72.7	21	–	64.6	18	–	101.2	16	–
		MOS	18.1	19	–	16.8	20	–	16.9	20	–	16.2	19	–
avg		ALG1	**4.9**	**884**	**13**	**6.9**	**1150**	**17**	**8.4**	**1338**	**18**	12.1	1553	20
		ALG2	5.2	1086	12	**6.9**	**1463**	**13**	8.5	1734	13	**11.9**	**2005**	**14**
		BAR	75.9	24	–	63.2	19	–	60.7	17	–	86.0	17	–
		MOS	17.8	18	–	16.8	18	–	15.9	19	–	15.5	18	–

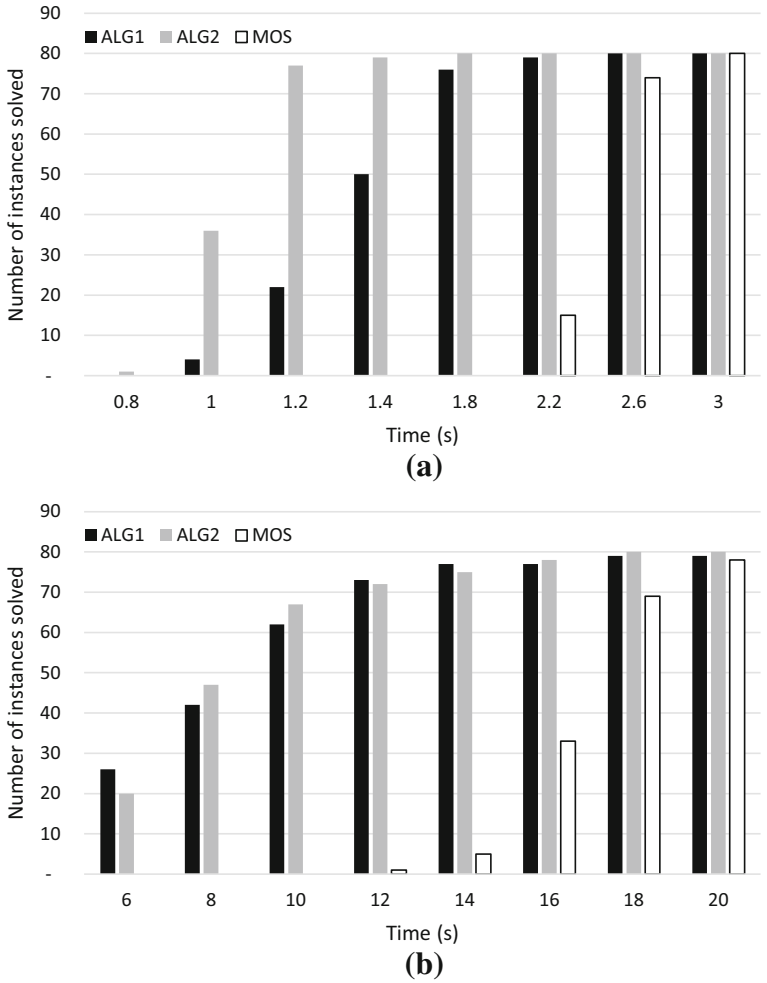

Fig. 1 Number of convex instances solved within a time limit for algorithms ALG1, ALG2 and MOS. **a** Cardinality instances, **b** path instances

consecutive QPs changes by a larger amount. Finally, note that although their runtime is very close, the performance of ALG2 is slightly better than ALG1 overall.

4.2.3 Effect of the dimension

Table 4 presents a comparison of the algorithms for the convex cardinality instances with sizes 400, 800, 1600, and 3200. Each row represents the average over five instances, as before, generated with parameters $r = 200$, $\alpha = 0.1$, and $\Omega = 2$. Additionally, Fig. 2 shows the solution time for ALG1, ALG2 and MOS as a function of the dimension (n).

Observe in Table 4 that the number of QPs solved with the simplex-based algorithms does not depend on the dimension. The number of simplex iterations, however, increases with the dimension. For $n = 400$ all algorithms perform similarly and the problems are solved very fast. However, as the dimension increases, the simplex-based algorithms outperform the barrier algorithms, often by many factors. For $n = 3200$, the fastest simplex-based algorithm ALG2 is more than 20 times faster than CPLEX barrier algorithm, and more than five times faster than MOSEK barrier algorithm. Similar results are obtained for other parameter choices and for the path instances as well. In summary, the simplex-based algorithms scale better with the dimension, and are faster by orders of magnitude for large instances. As in Sect. 4.2.2, ALG2 slightly outperforms ALG1 for the instances considered.

4.3 Discrete instances

In this section we describe our experiments with the discrete counterpart CDO. To the best of our knowledge, as of version 12.6.2 of CPLEX, there is no documented way to embed a user-defined convex solver such as Algorithms 1 or 2 at the nodes of the CPLEX branch-and-bound algorithm. Therefore, in order to test the proposed approach for CDO, we implement a rudimentary branch-and-bound algorithm described in "Appendix 1". The algorithm uses a maximum infeasibility rule for branching, and does not employ presolve, cutting planes, or heuristics. We test the following configurations:

BBA1 Branch-and-bound algorithm in "Appendix 1" using Algorithm 1 as the convex solver. The first QP at each node (except the root node) is solved with CPLEX dual simplex method using the parent dual feasible basis as a warm start (as mentioned in Sect. 3.3) and all other QPs are solved with CPLEX primal simplex method using the basis from the parent node QP as a warm start.

BBA2 Branch-and-bound algorithm in "Appendix 1" using Algorithm 2 as the convex solver. Algorithm 2 resulted in the best performance in the continuous instances; however, unlike Algorithm 1, it cannot be naturally warm-started. Thus, in this configuration, each convex subproblem is solved without exploiting the solution from the parent node.

BBBR Branch-and-bound algorithm in "Appendix 1", using CPLEX barrier algorithm as the convex solver. This configuration does not use warm starts.

CXBR CPLEX branch-and-bound algorithm with barrier solver, setting the branching rule to maximum infeasibility, the node selection rule to best bound, and disabling presolve, cuts and heuristics. In this setting CPLEX branch-and-bound algorithm is as close as possible to our branch-and-bound algorithm.

CXLP CPLEX branch-and-bound algorithm with LP outer approximations, setting the branching rule to maximum infeasibility, the node selection rule to best bound, and disabling presolve, cuts and heuristics. In this setting CPLEX branch-and-bound algorithm is as close as possible to our branch-and-bound algorithm.

CXLPE CPLEX branch-and-bound algorithm with LP outer approximations, setting the branching rule to maximum infeasibility, the node selection rule to best bound, and disabling cuts and heuristic. Since presolve is activated, CPLEX

Table 4 The effect of dimension (cardinality instances)

Method	$n = 400$			$n = 800$			$n = 1600$			$n = 3200$		
	time	#iter	#QP	time	#iter	#QP	time	#iter	#QP	time	#iter	#QP
ALG1	**0.2**	**43**	**20**	0.6	65	19	2.8	75	25	11.7	104	25
ALG2	**0.2**	**73**	**14**	**0.5**	**116**	**14**	**2.2**	**129**	**15**	**9.1**	**175**	**15**
BAR	0.3	21	–	2.4	22	–	22.1	27	–	204.9	30	–
MOS	**0.2**	**9**	–	1.2	11	–	7.3	12	–	50.4	12	–

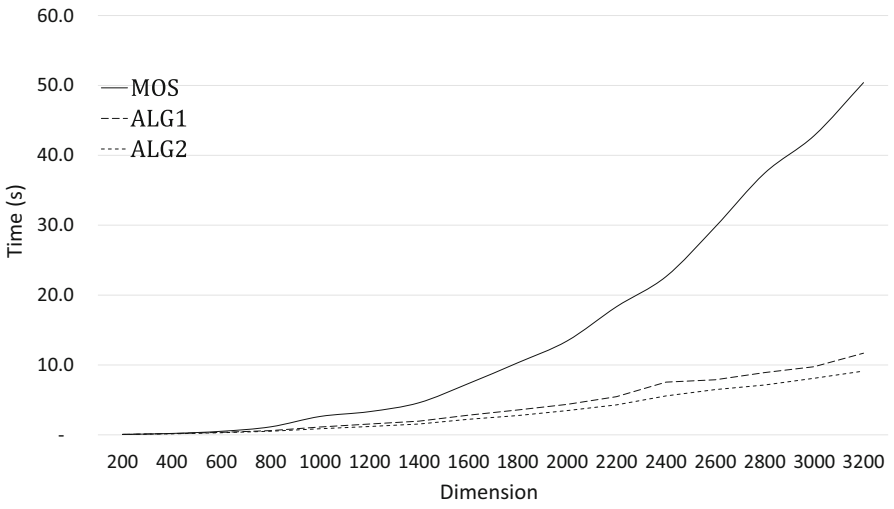

Fig. 2 Solution time as a function of dimension

uses extended formulations described in [40]. Besides presolve, all other parameters are set as in CXLP.

CXD CPLEX default branch-and-bound algorithm with LP outer approximations. This algorithm utilizes all sophisticated features of CPLEX, such as presolver, cutting planes, heuristics, advanced branching and node selection rules.

The time limit is set to two hours for each algorithm.

Table 5 presents the results for discrete cardinality instances with 200 variables and Table 6 for the discrete path instances with 1740 variables (30×30 grid). Each row represents the average over five instances with varying rank and density parameters, and algorithm. The tables show the solution time in seconds, the number of nodes explored in the branch-and-bound tree, the end gap after two hours as percentage, and the number of instances that are solved to optimality for varying values of Ω. For each instance class we highlight in bold the algorithm with the best performance. Figure 3 shows, for each instance class, the total number of instances solved within given time limits for BBA1, CXLPE and CXD.

First of all, observe that the difficulty of the instances increases considerably for higher values of Ω due to higher integrality gap. The problems corresponding to high values of the density parameter α are also more challenging.

4.3.1 Performance of CPLEX branch-and-bound

Among CPLEX branch-and-bound algorithms, CXD is the best choice when $\Omega \geq 2$. Configuration CXD is much more sophisticated than the other configurations, so a better performance is expected. However, note that for $\Omega = 1$ configuration CXD is not necessarily the best. In particular in the path instances (Table 6) CXLP and CXLPE are 2.3 times faster than CXD. This result suggests that in simple instances

Table 5 Comparison for discrete cardinality instances

r	α	Method	$\Omega = 1$				$\Omega = 2$				$\Omega = 3$				$\Omega = 4$			
			time	nodes	egap	#s	time	nodes	egap	#s	time	nodes	egap	#s	time	nodes	egap	#s
100	0.1	BBA1	**1**	**156**	**0.0**	**5**	**29**	**3271**	**0.0**	**5**	685	68,318	0.0	5	**3644**	**272,527**	**0.1**	**4**
		BBA2	3	156	0.0	5	83	3271	0.0	5	1823	68,317	0.0	5	6218	172,489	0.3	2
		BBBR	16	156	0.0	5	349	3270	0.0	5	4664	43695	0.1	3	7200	23,324	1.2	0
		CXBR	35	276	0.0	5	513	3497	0.0	5	5260	32,782	0.2	2	7200	17,169	1.3	0
		CXLP	34	9562	0.0	5	7200	209,576	0.7	0	7200	244,911	2.2	0	7200	265,485	3.8	0
		CXLPE	2	374	0.0	5	91	7640	0.0	5	2788	111,293	0.0	5	6983	191,065	0.6	1
		CXD	4	368	0.0	5	42	5152	0.0	5	**640**	**58,076**	**0.0**	**5**	3778	183,816	0.1	4
100	0.5	BBA1	**1**	**87**	**0.0**	**5**	**59**	**6274**	**0.0**	**5**	**1469**	**140,874**	**0.0**	**5**	**6328**	**447,989**	**0.4**	**2**
		BBA2	2	87	0.0	5	160	6274	0.0	5	4024	139,154	0.0	4	7200	223,921	0.7	0
		BBBR	10	87	0.0	5	686	6274	0.0	5	6134	56,394	0.3	1	7200	21,111	1.7	0
		CXBR	24	183	0.0	5	1027	6734	0.0	5	6399	39,710	0.4	1	7200	20,101	2.0	0
		CXLP	294	26,957	0.0	5	7200	229,641	0.8	0	7200	263,810	2.3	0	7200	244,863	4.7	0
		CXLPE	2	349	0.0	5	218	14,737	0.0	5	5116	215,292	0.1	2	7200	170,710	1.1	0
		CXD	3	373	0.0	5	164	16,070	0.0	5	3643	245,251	0.0	4	7042	336,814	0.8	1
200	0.1	BBA1	**1**	**247**	**0.0**	**5**	**23**	**3259**	**0.0**	**5**	**637**	**55,248**	**0.0**	**5**	**3761**	**344,662**	**0.2**	**4**
		BBA2	5	247	0.0	5	79	3259	0.0	5	2083	55,248	0.0	5	6975	242,548	0.4	2
		BBBR	24	247	0.0	5	321	3259	0.0	5	4573	39,647	0.1	3	7200	17,017	1.4	0
		CXBR	52	460	0.0	5	540	3711	0.0	5	5295	34,090	0.2	2	7200	13,490	1.5	0
		CXLP	221	17,205	0.0	5	7200	208,874	0.6	0	7200	230,304	2.0	0	7200	186,490	4.2	0
		CXLPE	4	473	0.0	5	139	6064	0.0	5	4073	111,205	0.1	3	7200	158,866	0.9	0
		CXD	5	360	0.0	5	60	6413	0.0	5	1410	67,577	0.0	5	7044	349,653	0.5	1

Table 5 continued

r	α	Method	$\Omega = 1$				$\Omega = 2$				$\Omega = 3$				$\Omega = 4$			
			time	nodes	egap	#s	time	nodes	egap	#s	time	nodes	egap	#s	time	nodes	egap	#s
200	0.5	BBA1	**4**	**674**	**0.0**	**5**	**194**	**24,636**	**0.0**	**5**	**1674**	**156,632**	**0.0**	**5**	**5778**	**526,215**	**0.3**	**2**
		BBA2	15	674	0.0	5	633	24,635	0.0	5	3446	98,028	0.1	4	7200	259,040	0.6	0
		BBBR	77	674	0.0	5	2106	17,743	0.0	4	5590	47,725	0.2	2	7200	20,422	1.5	0
		CXBR	104	680	0.0	5	2452	15,816	0.0	4	6127	38,973	0.3	1	7200	14,955	1.7	0
		CXLP	3514	120,007	0.1	4	7200	212,082	1.0	0	7200	240,445	2.3	0	7200	195,841	4.8	0
		CXLPE	21	1461	0.0	5	1739	61,593	0.0	4	5435	163,105	0.2	2	7200	197,287	1.1	0
		CXD	18	1612	0.0	5	1211	75,098	0.0	5	5017	245,412	0.2	2	7200	319,645	1.0	0
avg		BBA1	**2**	**291**	**0.0**	**20**	**76**	**9360**	**0.0**	**20**	**1116**	**105,268**	**0.0**	**20**	**4878**	**397,848**	**0.3**	**12**
		BBA2	6	291	0.0	20	239	9360	0.0	20	2844	90,187	0.0	18	6898	224,500	0.5	4
		BBBR	32	291	0.0	20	865	7637	0.0	19	5240	46,865	0.2	9	7200	20,469	1.4	0
		CXBR	54	400	0.0	20	1133	7440	0.0	19	5770	36,389	0.3	6	7200	16,429	1.6	0
		CXLP	1016	43,433	0.0	19	7200	215,043	0.8	0	7200	244,867	2.2	0	7200	223,170	4.4	0
		CXLPE	7	664	0.0	20	547	20,800	0.0	19	4353	139,632	0.1	12	7146	179,482	0.9	1
		CXD	7	678	0.0	20	369	25,683	0.0	20	2677	151,588	0.1	16	6267	297,482	0.6	6

Table 6 Comparison for discrete path instances

r	α	Method	$\Omega = 1$				$\Omega = 2$				$\Omega = 3$				$\Omega = 4$			
			time	nodes	egap	#s	time	nodes	egap	#s	time	nodes	egap	#s	time	nodes	egap	#s
100	0.1	BBA1	**287**	**145**	**0.0**	**5**	**4511**	**1774**	**0.0**	**5**	**7200**	**2720**	**5.9**	**0**	**7200**	**2360**	**13.9**	**0**
		BBA2	619	142	0.0	5	6871	1105	1.1	1	7200	1278	7.9	0	7200	974	17.2	0
		BBBR	3577	91	0.2	4	7200	184	4.9	0	7200	236	11.7	0	7200	61	37.2	0
		CXBR	7200	67	20.8	0	7200	79	∞	0	7200	129	∞	0	7200	13	∞	0
		CXLP	533	2428	0.0	5	7200	24,776	4.5	0	7200	20,099	15.0	0	7200	8,971	30.2	0
		CXLPE	802	315	0.0	5	6726	1967	2.9	1	7200	2585	23.5	0	7200	2,377	45.1	0
		CXD	1466	164	0.0	5	4655	1176	0.0	5	7200	2428	7.4	0	7200	2,047	16.6	0
100	0.5	BBA1	**625**	**353**	**0.0**	**5**	**5424**	**1904**	**0.6**	**2**	**6512**	**2725**	**4.8**	**1**	**7200**	**2833**	**12.5**	**0**
		BBA2	1457	362	0.0	5	6286	971	1.6	1	7200	1369	7.3	0	7200	1296	16.2	0
		BBBR	6071	134	0.7	2	7200	175	4.9	0	7200	162	11.5	0	7200	28	∞	0
		CXBR	7200	24	∞	0	7200	56	∞	0	7200	70	∞	0	7200	13	∞	0
		CXLP	1132	6187	0.0	5	7200	23,671	4.4	0	7200	16,851	12.8	0	7200	8,180	23.8	0
		CXLPE	967	607	0.0	5	6420	2077	3.4	1	7200	3070	16.2	0	7200	3036	42.3	0
		CXD	1645	267	0.0	5	**6421**	**1931**	**0.5**	**3**	7200	2659	5.6	0	7200	4166	12.8	0
200	0.1	BBA1	**155**	**77**	**0.0**	**5**	**2392**	**1075**	**0.0**	**5**	**6434**	**3380**	**2.6**	**1**	**7200**	**3245**	**10.0**	**0**
		BBA2	306	79	0.0	5	4324	823	0.3	3	7152	1469	4.6	1	7200	1195	12.7	0
		BBBR	3245	76	0.0	5	7200	171	2.4	0	7200	180	10.5	0	7200	33	∞	0
		CXBR	7200	34	∞	0	7200	45	∞	0	7200	68	∞	0	7200	11	∞	0
		CXLP	436	1548	0.0	5	7200	30,265	2.9	0	7200	20,579	12.4	0	7200	7274	26.7	0
		CXLPE	524	188	0.0	5	5156	1437	1.4	3	7200	2420	17.7	0	7200	2,157	46.7	0
		CXD	2059	106	0.0	5	5715	1251	0.5	4	7200	2568	4.1	0	7200	1996	12.9	0

Table 6 continued

r	α	Method	$\Omega = 1$				$\Omega = 2$				$\Omega = 3$				$\Omega = 4$			
			time	nodes	egap	#s	time	nodes	egap	#s	time	nodes	egap	#s	time	nodes	egap	#s
200	0.5	BBA1	**321**	**196**	**0.0**	**5**	**3953**	**2286**	**0.3**	**4**	**7200**	**3194**	**4.3**	**0**	**7200**	**2486**	**13.2**	**0**
		BBA2	808	201	0.0	5	5517	1204	0.9	3	7200	1284	6.5	0	7200	1009	16.6	0
		BBBR	4826	113	0.2	3	7200	173	3.9	0	7200	176	12.7	0	7200	54	43.6	0
		CXBR	7200	20	∞	0	7200	51	∞	0	7200	89	∞	0	7200	10	∞	0
		CXLP	859	4989	0.0	5	7200	28,007	4.4	0	7200	18,873	13.3	0	7200	10,313	28.9	0
		CXLPE	1046	399	0.0	5	5948	2212	1.8	2	7200	2305	21.2	0	7200	2,048	51.2	0
		CXD	2281	177	0.0	5	4975	1873	0.4	4	7200	2233	6.9	0	7200	1325	16.0	0
avg		BBA1	**347**	**193**	**0.0**	**20**	**4070**	**1760**	**0.2**	**16**	**6837**	**3005**	**4.4**	**2**	**7200**	**2731**	**12.4**	**0**
		BBA2	798	196	0.0	20	5750	1026	1.0	8	7189	1350	6.6	1	7200	1119	15.7	0
		BBBR	4430	103	0.3	14	7200	176	4.0	0	7200	189	11.6	0	7200	44	∞	0
		CXBR	7200	36	∞	0	7200	58	∞	0	7200	89	∞	0	7200	12	∞	0
		CXLP	740	3788	0.0	20	7200	26,680	4.1	0	7200	19,101	13.4	0	7200	8,684	27.4	0
		CXLPE	835	377	0.0	20	6063	1923	2.4	7	7200	2595	19.7	0	7200	2,404	46.3	0
		CXD	1863	178	0.0	20	5441	1558	0.3	16	7200	2472	6.0	0	7200	2,383	14.6	0

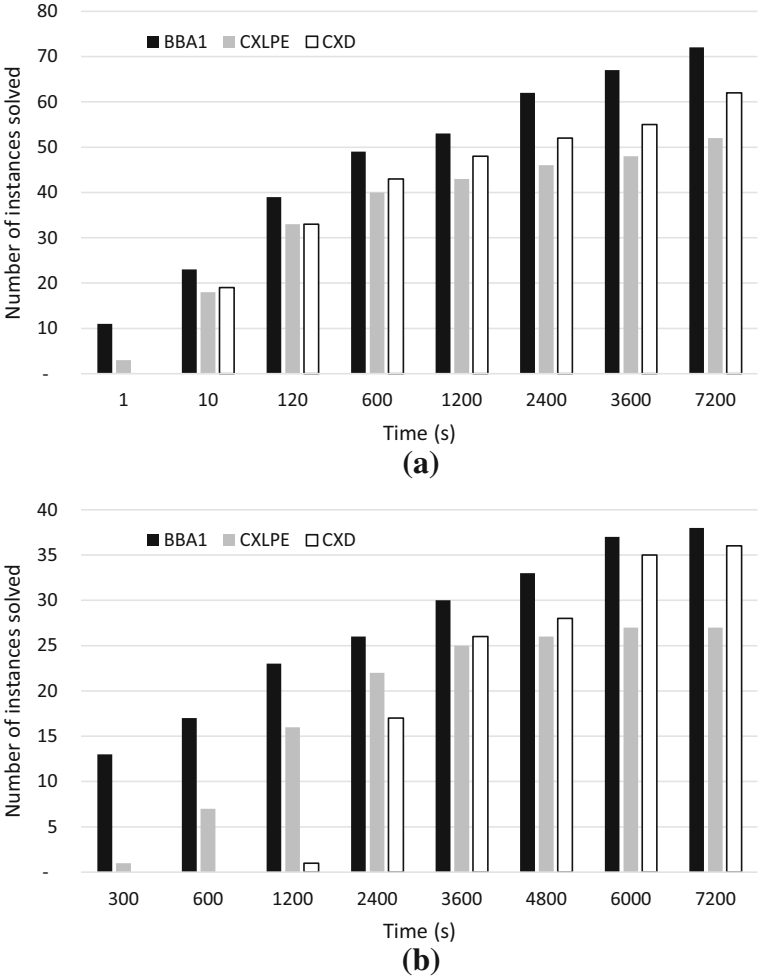

Fig. 3 Number of discrete instances solved within a time limit. **a** Cardinality instances, **b** path instances

the additional features used by CXD (e.g. cutting planes and heuristics) may be hurting the performance.

The extended formulations result in much stronger relaxations in LP based branch-and-bound and, consequently, the number of branch-and-bound nodes required with CXLPE is only a small fraction of the number of nodes required with CXLP. However, CXLPE requires more time to solve each branch-and-bound node, due to the higher number of variables and the additional effort needed to refine the LP outer approximations. For the cardinality instances, CXLPE is definitely the better choice and is faster by orders of magnitude. For the path instances, however, CXLP is not necessarily inferior: when $\Omega = 1$ CXLP is competitive with CXLPE, and when $\Omega = 3$ CXLP performs better.

The barrier-based branch-and-bound CXBR, in general, performs poorly. For the cardinality instances, it outperforms CXLP but is slower than the other algorithms. For the path instances it has the worst performance, often struggling to find even a single feasible solution (resulting in infinite end gaps).

4.3.2 Performance of BBA1

Note that BBA1, BBA2 and BBBR are very simple and differ only by the convex node solver. BBA1 is faster than BBBR by an order of magnitude. BBA1 is also two to three times faster than BBA2, despite the fact that Algorithm 2 is faster for convex problems. This improvement is due to the warm start capabilities of Algorithm 1. BBA1 is considerably faster than the simplest CPLEX branch-and-bound algorithms CXBR and CXLP.

We see that BBA1 consistently outperforms CXLPE (which uses presolve and extended formulations), often by many factors. In fact, BBA1 resulted in better performance (faster solution times or lower end gaps) than CXLPE in every instance. Observe that in the cardinality instances with $\Omega = 1, 2$ and path instances with $\Omega = 1$, BBA1 requires half the number of nodes (or less) compared to CXLPE to solve the instances to optimality (since the relaxations solved at each node are stronger), which translates into faster overall solution times. In the more difficult instances BBA1 is able to solve more instances to optimality, and the end gaps are smaller.

Despite the fact that BBA1 is a rudimentary branch-and-bound implementation, it is faster than default CPLEX in most of the cases. Indeed, BBA1 outperforms CXD in 143 out of 160 instances tested. Figure 3 clearly shows that BBA1 solves more instances faster compared to CXLPE and CXD.

4.3.3 Warm starts

To quantify the impact of warm starts, we plot in Fig. 4 the *time per node* (computed as solution time divided by the number of branch-and-bound nodes) for BBA1, BBA2, BBBR and CXLPE, and also plot the solution time for the corresponding convex instances with solvers ALG1, ALG2, and BAR.[1]

For the small cardinality instances with 200 variables, all three algorithms perform similarly for the convex instances; however, Algorithm 1 is 15 times faster than barrier and more than three times faster than Algorithm 2 when used in branch-and-bound due to the node warm starts from dual feasible solutions. For the larger path instances with 1740 variables, Algorithm 1 is again close to three times faster than Algorithm 2 and almost 20 times faster than barrier in discrete instances.

Finally, observe that the solve time per node for BBA1 is smaller compared to CXLPE: the proposed simplex-based algorithm is thus as effective as the simplex method for extended formulations in exploiting warm starts. Moreover, it solves the nonlinear convex relaxations at each node to optimality, whereas CXLPE solves its LP relaxation. The improved lower bounds lead to significantly small search trees.

[1] The time per node is similar for all combinations of parameters Ω, r and α. We plot the average for all instances with $\Omega = 2$.

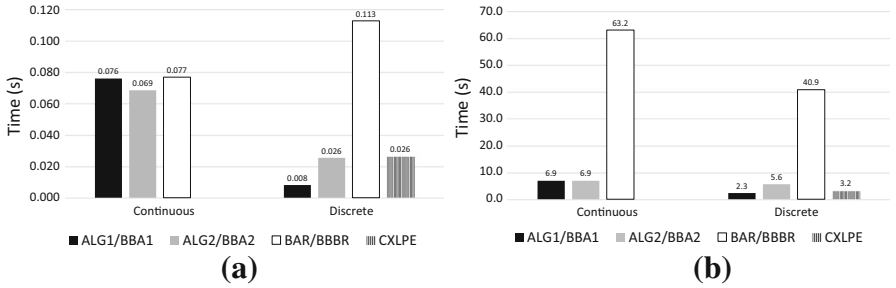

Fig. 4 Time per node. **a** Cardinality instances, **b** path instances

We conclude that Algorithm 1 is indeed suitable for branch-and-bound algorithms since it benefits from node warms starts from the parent nodes, resulting in a significant improvement in solution times.

5 Conclusions

We consider minimization problems with a conic quadratic objective and linear constraints, which are natural generalizations of linear programming and quadratic programming. Using the perspective function we reformulate the objective and propose simplex QP-based algorithms that solve a quadratic program at each iteration. Computational experiments indicate that the proposed algorithms are faster than interior point methods by orders of magnitude, scale better with the dimension of the problem, return higher precision solutions, and, most importantly, are amenable to warm starts. Therefore, they can be embedded in branch-and-bound algorithms quite effectively.

Acknowledgements This research is supported, in part, by grant FA9550-10-1-0168 from the Office of the Assistant Secretary of Defense for Research and Engineering.

Appendix A. Branch-and-bound algorithm

Algorithm 3 describes the branch-and-bound algorithm used in computations. Throughout the algorithm, we maintain a list L of the nodes to be processed. Each node is a tuple (S, B, lb), where S is the subproblem, B is a basis for warm starting the continuous solver and lb is a lower bound on the objective value of S. In line 3 list L is initialized with the root node. For each node, the algorithm calls a continuous solver (line 9) which returns a tuple $(x, \bar{B}, z)$, where x is an optimal solution of S, $\bar{B}$ is the corresponding optimal basis and z is the optimal objective value (or ∞ if S is infeasible). The algorithm then checks whether the node can be pruned (lines 10–11), x is integer (lines 12–15), or it further branching is needed (lines 16–18).

We now describe the specific implementations of the different subroutines. For branching (line 17) we use the maximum infeasibility rule, which chooses the variable

Algorithm 3 Branch-and-bound algorithm

Input: P, discrete minimization problem
Output: Optimal solution x^*

1: $ub \leftarrow \infty$ ▷ Upper bound
2: $x^* \leftarrow \emptyset$ ▷ Best solution found
3: $L \leftarrow \{(P, \emptyset, -\infty)\}$ ▷ list of nodes L initialized with the original problem
4: **while** $L \neq \emptyset$ **do**
5: $(S, B, lb) \leftarrow \texttt{PULL}(L)$ ▷ select and remove one element from L
6: **if** $lb \geq ub$ **then**
7: **go to** line 4
8: **end if**
9: $(x, \bar{B}, z) \leftarrow \texttt{SOLVE}(S, B)$ ▷ solve continuous relaxation
10: **if** $z \geq ub$ **then** ▷ if S is infeasible then $z = \infty$
11: **go to** line 4 ▷ prune by infeasibility or bounds
12: **else if** x is integer **then**
13: $ub \leftarrow z$ ▷ update incumbent solution
14: $x^* \leftarrow x$
15: **go to** line 4 ▷ prune by integer feasibility
16: **else**
17: $(S_{\leq}, S_{\geq}) \leftarrow \texttt{BRANCH}(x)$ ▷ create two subproblems
18: $L \leftarrow L \cup \left\{(S_{\leq}, \bar{B}, z), (S_{\geq}, \bar{B}, z)\right\}$ ▷ add the subproblems to L
19: **end if**
20: **end while**
21: **return** x^*

x_i with value v_i furtherest from an integer (ties broken arbitrarily). The subproblems $S_{\leq}$ and $S_{\geq}$ in line 18 are created by imposing the constraints $x_i \leq \lfloor v_i \rfloor$ and $x_i \geq \lceil v_i \rceil$, respectively. The PULL routine in line 5 chooses, when possible, the child of the previous node which violates the bound constraint by the least amount, and chooses the node with the smallest lower bound when the previous node has no child nodes. The list L is thus implemented as a sorted list ordered by the bounds, so that the PULL operation is done in $O(1)$ and the insertion is done in $O(\log |L|)$ (note that in line 18 we only add to the list the node that is not to be processed immediately). A solution x is assumed to be integer (line 12) when the values of all variables are within 10^{-5} of an integer. Finally, the algorithm is terminated when $\frac{ub - lb_{best}}{|lb_{best} + 10^{-10}|} \leq 10^{-4}$, where lb_{best} is the minimum lower bound among all the nodes in the tree.

The maximum infeasibility rule is chosen due to its simplicity. The other rules and parameters correspond to the ones used in CPLEX branch-and-bound algorithm in default configuration.

Appendix B. Code

We provide a Java implementation of the proposed algorithms in "Simplex QP-based methods for minimizing a conic quadratic objective over polyhedra," identified with DOI https://www.doi.org/10.5281/zenodo.1489153. Using the code requires having Java and CPLEX software installed on the computer. We now provide a brief description of the contents of the software, as well as a quick guide on how to run the experiments described in the paper.

Contents

The main directory contains two scripts, `generateScript.bat` and `generateScript.sh`, for running the code in Windows and Linux, respectively. The rest of the key directories are:

`dist`	Contains the executable jar file, `LagrangeanConic.jar`, for running the code. Also contains a `lib` directory with required libraries.
`doc`	Javadoc.
`results`	Directory where the results are stored.
`src`	Directory with the source code. Subdirectory `src/cplex` contains the codes corresponding to the proposed methodology. Specifically,

`QuadraticSolver.java`	Code for Algorithm 1.
`QuadraticSolverCoordinateLP.java`	Algorithm 1 with $t_0 = \infty$.
`QuadraticSolverBisection.java`	Code for Algorithm 2.
`ConicSolver.java`	Code for solving with CPLEX default algorithms.
`BranchAndBound.java`	Code for the branch-and-bound algorithm.

Subdirectory `src/instances` contains code pertaining to the generation of the instances, subdirectory `src/results` contains code for processing the results, and subdirectory `src/interfaces` contains auxiliary code.

Running the experiments

The file `README.txt` in the main directory contains detailed instructions for running experiments. We now present a quick summary on how to run the experiments.

The files `generateScript.bat` and `generateScript.sh` contain two commands:

(1) java -cp ./dist/LagrangeanConic.jar instances.FileGeneratorLinux **# path**
(2) ./run.bat (or ./run.sh)

In command (1), **#** should be replaced with a number between 1 and 6, and **path** should be replaced with the path to CPLEX .dll file. When executing command (1), another script called `run.bat` or `run.sh` will be created for running all the experiments corresponding to Table **#** in the paper. For example, if **#** is replaced with 1, then the command will create a file that, when executed, runs the code 24 times, one for each precision and algorithm, and stores the results in the directory `results`. Command (2) simply executes the newly created file.

References

1. Ahmed, S., Atamtürk, A.: Maximizing a class of submodular utility functions. Math. Program. **128**, 149–169 (2011)

2. Alizadeh, F.: Interior point methods in semidefinite programming with applications to combinatorial optimization. SIAM J. Optim. **5**, 13–51 (1995)
3. Alizadeh, F., Goldfarb, D.: Second-order cone programming. Math. Program. **95**, 3–51 (2003)
4. Atamtürk, A., Goméz, A.,: Submodularity in conic quadratic mixed 0–1 optimization. arXiv preprint arXiv:1705.05918, (2016). BCOL Research Report 16.02, UC Berkeley
5. Atamtürk, A., Jeon, H.: Lifted polymatroid inequalities for mean-risk optimization with indicator variables. arXiv preprint arXiv:1705.05915, (2017). BCOL Research Report 17.01, UC Berkeley
6. Atamtürk, A., Narayanan, V.: Cuts for conic mixed-integer programming. In: Fischetti, M., Williamson, D.P. (eds.) Integer Programming and Combinatorial Optimization, pp. 16–29. Springer, Berlin (2007) ISBN 978-3-540-72792-7
7. Atamtürk, A., Narayanan, V.: Polymatroids and risk minimization in discrete optimization. Oper. Res. Lett. **36**, 618–622 (2008)
8. Atamtürk, A., Narayanan, V.: The submodular 0–1 knapsack polytope. Discrete Optim. **6**, 333–344 (2009)
9. Atamtürk, A., Deck, C., Jeon, H.: Successive quadratic upper-bounding for discrete mean-risk minimization and network interdiction. arXiv preprint arXiv:1708.02371, (2017). BCOL Reseach Report 17.05, UC Berkeley. Forthcoming in INFORMS Journal on Computing
10. Aybat, N.S., Iyengar, G.: A first-order smoothed penalty method for compressed sensing. SIAM J. Optim. **21**, 287–313 (2011)
11. Belloni, A., Chernozhukov, V., Wang, L.: Square-root lasso: pivotal recovery of sparse signals via conic programming. Biometrika **98**, 791–806 (2011)
12. Belotti, P., Kirches, C., Leyffer, S., Linderoth, J., Luedtke, J., Mahajan, A.: Mixed-integer nonlinear optimization. Acta Numerica **22**, 1131 (2013)
13. Ben-Tal, A., Nemirovski, A.: Robust convex optimization. Math. Oper. Res. **23**, 769–805 (1998)
14. Ben-Tal, A., Nemirovski, A.: Robust solutions of uncertain linear programs. Oper. Res. Lett. **25**, 1–13 (1999)
15. Ben-Tal, A., Nemirovski, A.: Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications. MPS-SIAM Series on Optimization. SIAM, Philadelphia (2001)
16. Ben-Tal, A., El Ghaoui, L., Nemirovski, A.: Robust Optimization. Princeton University Press, Princeton (2009)
17. Bertsimas, D., Sim, M.: Robust discrete optimization under ellipsoidal uncertainty sets (2004) **(unpublished)**
18. Bertsimas, D., King, A., Mazumder, R., et al.: Best subset selection via a modern optimization lens. Ann. Stat. **44**, 813–852 (2016)
19. Bienstock, D.: Computational study of a family of mixed-integer quadratic programming problems. Math. Program. **74**, 121–140 (1996)
20. Borchers, B., Mitchell, J.E.: An improved branch and bound algorithm for mixed integer nonlinear programs. Comput. Oper. Res. **21**, 359–367 (1994)
21. Çay, S.B., Pólik, I., Terlaky, T.: Warm-start of interior point methods for second order cone optimization via rounding over optimal Jordan frames, May 2017. ISE Technical Report 17T-006, Lehigh University
22. Dantzig, G.B., Orden, A., Wolfe, P.: The generalized simplex method for minimizing a linear form under linear inequality restraints. Pac. J. Math. **5**, 183–196 (1955)
23. Dinh, T., Fukasawa, R., Luedtke, J.: Exact Algorithms for the Chance-Constrained Vehicle Routing Problem, pp. 89–101. Springer, New York (2016)
24. Efron, B., Hastie, T., Johnstone, I., Tibshirani, R., et al.: Least angle regression. Ann. Stat. **32**, 407–499 (2004)
25. El Ghaoui, L., Oks, M., Oustry, F.: Worst-case value-at-risk and robust portfolio optimization: a conic programming approach. Oper. Res. **51**, 543–556 (2003)
26. Hiriart-Urruty, J.-B., Lemaréchal, C.: Convex Analysis and Minimization Algorithms I: Fundamentals. Springer, New York (2013)
27. Ishii, H., Shiode, S., Nishida, T., Namasuya, Y.: Stochastic spanning tree problem. Discrete Appl. Math. **3**, 263–273 (1981)
28. Karmarkar, N.: A new polynomial-time algorithm for linear programming. In: Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing, pp. 302–311. ACM (1984)
29. Leyffer, S.: Integrating SQP and branch-and-bound for mixed integer nonlinear programming. Comput. Optim. Appl. **18**, 295–309 (2001)

30. Lobo, M.S., Vandenberghe, L., Boyd, S., Lebret, H.: Applications of second-order cone programming. Linear Algebra Appl **284**, 193–228 (1998)
31. Megiddo, N.: On finding primal- and dual-optimal bases. INFORMS J. Comput. **3**, 63–65 (1991)
32. Nemirovski, A., Scheinberg, K.: Extension of Karmarkar's algorithm onto convex quadratically constrained quadratic problems. Math. Program. **72**, 273–289 (1996)
33. Nesterov, Y.: Smooth minimization of non-smooth functions. Math. Program. **103**, 127–152 (2005)
34. Nesterov, Y., Nemirovski, A.: Interior-Point Polynomial Algorithms in Convex Programming. Society for Industrial and Applied Mathematics (1994). URL http://epubs.siam.org/doi/abs/10.1137/1.9781611970791
35. Nesterov, Y.E., Todd, M.J.: Primal-dual interior-point methods for self-scaled cones. SIAM J. Optim. **8**, 324–364 (1998)
36. Nikolova, E., Kelner, J.A., Brand, M., Mitzenmacher, M.: Stochastic shortest paths via quasi-convex maximization. In: European Symposium on Algorithms, pp. 552–563. Springer, New York (2006)
37. Tawarmalani, M., Sahinidis, N.V.: A polyhedral branch-and-cut approach to global optimization. Math. Program. **103**(2), 225–249 (2005)
38. Tibshirani, R.: Regression shrinkage and selection via the lasso. J. R. Stat. Soc Ser. B (Methodological) **58**, 267–288 (1996)
39. Van de Panne, C., Whinston, A.: Simplicial methods for quadratic programming. Naval Res. Log. Q. **11**(3–4), 273–302 (1964)
40. Vielma, J.P., Dunning, I., Huchette, J., Lubin, M.: Extended formulations in mixed integer conic quadratic programming. Math. Program. Comput. (2015). https://doi.org/10.1007/s125312-016-0113-y
41. Wolfe, P.: The simplex method for quadratic programming. Economet. J. Economet. Soc. **27**, 382–398 (1959)
42. Yildirim, E.A., Wright, S.J.: Warm-start strategies in interior-point methods for linear programming. SIAM J. Optim. **12**, 782–810 (2002)