BCOL RESEARCH REPORT 17.02

Industrial Engineering & Operations Research
University of California, Berkeley, CA 94720–1777

SIMPLEX QP-BASED METHODS FOR MINIMIZING A CONIC QUADRATIC OBJECTIVE OVER POLYHEDRA

ALPER ATAMTÜRK AND ANDRÉS GÓMEZ

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 its objective. 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.

Keywords: Simplex method, conic quadratic optimization, quadratic programming, warm starts, value-at-risk minimization, portfolio optimization, robust optimization.

May 2017

A. Atamtürk: Industrial Engineering & Operations Research, University of California, Berkeley, CA 94720-1777. `atamturk@berkeley.edu`
A. Gómez: Industrial Engineering & Operations Research, University of California, Berkeley, CA 94720-1777. `a.gomez@ieor.berkeley.edu` .

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 (El Ghaoui et al., 2003), portfolio optimization (Atamtürk and Jeon, 2017), and robust counterparts of linear programs with an ellipsoidal objective uncertainty set (Ben-Tal and Nemirovski, 1998, 1999; Ben-Tal et al., 2009).

Note that CO includes linear programming (LP) and convex quadratic programming (QP) as special cases. The simplex method (Dantzig et al., 1955; Wolfe, 1959; Van de Panne and Whinston, 1964) is still the most widely used algorithm for LP and QP, despite the fact that polynomial interior point algorithms (Karmarkar, 1984; Nesterov and Nemirovski, 1994; Nemirovski and Scheinberg, 1996) 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. Megiddo, 1991). 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 (Yildirim and Wright, 2002; Çay et al., 2017). 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 (Lobo et al., 1998; Alizadeh and Goldfarb, 2003), which can be solved by polynomial-time interior points algorithms (Alizadeh, 1995; Nesterov and Todd, 1998; Ben-Tal and Nemirovski, 2001). 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 orders of magnitude 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. (1981) give a polynomial time for optimization over spanning trees; Bertsimas and Sim (2004) propose an approximation algorithm that solves series of linear integer programs; Atamtürk and Narayanan (2008) give a cutting plane algorithm utilizing the submodularity of the objective for the binary case; Atamtürk and Goméz (2016) give nonlinear cuts for the mixed 0-1 case; Atamtürk and Narayanan (2009) use parametric linear programming for the binary case with a cardinality constraint.

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 (1994) and Leyffer (2001) 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 (2005), is to use extended formulations. Atamtürk and Narayanan (2007) describe mixed-integer rounding inequalities in an extended formulation for conic quadratic integer programming. Vielma et al. (2015) 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. (2013) for an excellent survey of the solution methods for mixed-integer nonlinear optimization.

In this paper, we reformulate CO through the perspective of its objective function 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 Section 2 we give an alternative formulation for CO using the perspective function of the objective. In Section 3 we present coordinate descent and accelerated bisection algorithms that solve a sequence of QPs. In Section 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 its objective. 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. Hiriart-Urruty and Lemaréchal, 2013, 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 equation (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. □

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

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

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:     $x_{i+1} \leftarrow \arg\min_{x \in X} \left\{ c'x + \frac{\Omega}{2t_i} x'Qx + \frac{\Omega}{2} t_i \right\}$          ▷ solve QP
5:     $t_{i+1} \leftarrow \arg\min_{t \geq 0} \left\{ c'x_{i+1} + \frac{\Omega}{2t} {x_{i+1}}'Qx_{i+1} + \frac{\Omega}{2} t \right\}$    ▷ $t_{i+1} = \sqrt{{x_{i+1}}'Qx_{i+1}}$
6:     $i \leftarrow i + 1$
7: until stopping condition is met
8: return $x_i$
```

First observe that the sequence of objective values $\left\{ c'x_i + \frac{\Omega}{2t_i} x_i' Q x_i + \frac{\Omega}{2} t_i \right\}_{i \in \mathbb{N}}$ 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 $f(x) = \Omega \sqrt{x'Qx}$ 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*

$$\left\| -c' - \Omega \frac{x'_{i+1}Q}{\sqrt{{x_{i+1}}'Qx_{i+1}}} - \lambda' A + \mu \right\| \leq \epsilon + \frac{|\Delta_i|}{t_i} \cdot \|\nabla f(x_{i+1})\|$$

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 ${x_{i+1}}'Qx_{i+1} = t_{i+1}^2 \leq {t^*}^2 = {x^*}'Qx^*$, 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 $\frac{1}{t_i}\left\|\nabla f(x_{i+1})\right\| < M$ for all $i \in \mathbb{N}$, and we find that $\frac{|\Delta_i|}{t_i} \cdot \left\|\nabla f(x_{i+1})\right\| \xrightarrow{\Delta_i \to 0} 0$.

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 $\frac{1}{t_i}\left\|\nabla f(x_{i+1})\right\| < M$ for all $i \in \mathbb{N}$, and we find that $\frac{|\Delta_i|}{t_i} \cdot \left\|\nabla f(x_{i+1})\right\| \xrightarrow{\Delta_i \to 0} 0$.

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 $\{t_i\}_{i \in \mathbb{N}}$ 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.

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.

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 $\epsilon + \frac{|\Delta_i|}{t_i} \cdot \left\|\nabla f(x_{i+1})\right\| < \delta$. Alternatively, if $\exists k$ s.t. $\max_{x \in X} \left\|\nabla f(x)\right\| \leq k < \infty$, 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.

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%.

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

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 $\hat{x}$
```

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 $\frac{|t_1-t_0|}{t_0} \cdot \Omega \left\| \frac{x_0{}'Q}{\sqrt{x_0{}'Qx_0}} \right\| < \delta - \epsilon$ 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 Sections 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 COwith 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

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

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 Section 3.3.1.

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 Section 3. The algorithms are implemented with CPLEX Java API. We use the simplex and barrier solvers of CPLEX version 12.6.2 for the computational experiments. All experiments are conducted on a workstation with a 2.93GHz Intel®Core$^{\text{TM}}$ 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., $X = \{x \in \mathbb{R}^n : \sum_{i=1}^n x_i = b,\ \mathbf{0} \leq x \leq \mathbf{1}\}$ with $b = n/5$. For the second data set the feasible region consists of the path polytope of an acyclic grid network. 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.

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 for convex conic quadratic problems).

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

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.

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

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 the 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.

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.

Table 2. The effect of nonlinearity (cardinality instances).

				$\Omega = 1$			$\Omega = 2$			$\Omega = 3$	
r	α	Method	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
		ALG2	**0.8**	**41**	**14**	**0.9**	**95**	**15**	**0.9**	**150**	**15**
		BAR	4.6	16	-	4.9	24	-	5.2	26	-
100	0.5	ALG1	1.1	33	23	1.1	69	24	1.5	144	37
		ALG2	**0.8**	**60**	**14**	**0.9**	**125**	**15**	**0.9**	**200**	**15**
		BAR	4.5	21	-	5.1	25	-	5.8	29	-
200	0.1	ALG1	0.9	33	19	1.1	73	25	1.2	110	25
		ALG2	**0.8**	**49**	**14**	**0.9**	**126**	**14**	**0.9**	**172**	**14**
		BAR	4.7	22	-	4.5	22	-	5.1	25	-
200	0.5	ALG1	1.0	48	22	1.1	99	22	1.2	151	25
		ALG2	**0.9**	**94**	**14**	**0.9**	**179**	**14**	**1.0**	**233**	**15**
		BAR	4.4	21	-	4.9	24	-	5.2	26	-
avg		ALG1	1.0	34	21	1.1	73	24	1.3	127	29
		ALG2	**0.8**	**61**	**14**	**0.9**	**131**	**15**	**0.9**	**189**	**15**
		BAR	4.3	20	-	4.9	24	-	5.3	27	-

Table 3. The effect of nonlinearity (path instances).

				$\Omega = 1$			$\Omega = 2$			$\Omega = 3$	
r	α	Method	time	#iter	#QP	time	#iter	#QP	time	#iter	#QP
100	0.1	ALG1	**4.4**	**940**	**12**	**6.4**	**1,307**	**16**	**7.5**	**1,505**	**18**
		ALG2	4.8	1,283	11	**6.4**	**1,637**	**13**	**7.5**	**1,865**	**14**
		BAR	68.4	26	-	56.7	21	-	46.3	16	-
100	0.5	ALG1	**4.7**	**902**	**14**	7.4	1,191	21	8.3	1,391	21
		ALG2	4.9	1,148	12	**6.5**	**1,474**	**13**	**7.7**	**1,772**	**14**
		BAR	54.3	19	-	48.8	16	-	47.4	16	-
200	0.1	ALG1	4.5	836	14	**5.7**	**1,053**	**15**	**7.3**	**1,220**	**18**
		ALG2	**4.4**	**932**	**12**	6.0	1,377	13	7.4	1,671	13
		BAR	63.7	25	-	49.8	18	-	54.5	20	-
200	0.5	ALG1	**4.1**	**858**	**12**	**5.5**	**1,048**	**15**	**6.8**	**1,237**	**16**
		ALG2	4.4	978	12	6.0	1,363	13	7.5	1,626	13
		BAR	70.5	26	-	60.2	21	-	52.4	18	-
avg		ALG1	**4.4**	**884**	**13**	**6.2**	**1,150**	**17**	**7.5**	**1,338**	**18**
		ALG2	4.6	1,086	12	**6.2**	**1,463**	**13**	**7.5**	**1,734**	**13**
		BAR	64.2	24	-	53.9	19	-	50.2	17	-

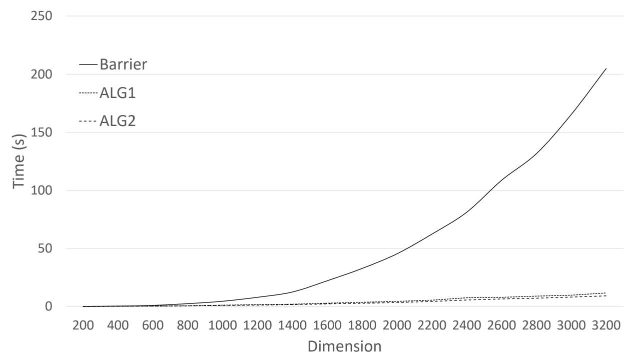
First observe that in both data sets the barrier algorithm is the slowest: it is 3.5 and 6 times slower than the simplex QP-based methods for the cardinality instances, and is up to 15 times slower for the path instances. The barrier algorithm does not appear to be too sensitive to the nonlinearity parameter 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 consecutive QPs changes by a larger amount.

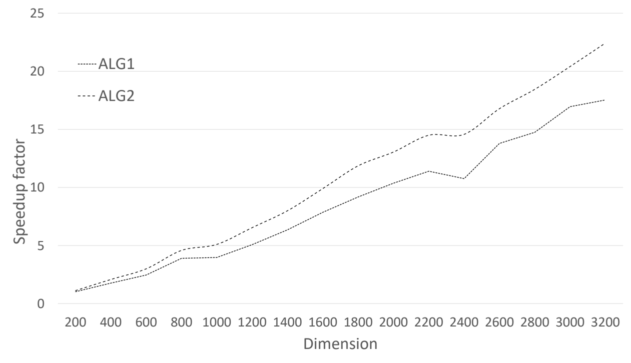
Effect of the dimension. Table 4 presents a comparison of the algorithms for the convex cardinality instances with sizes 400, 800, 1600, and 3200. Each row represents the average over five instances, as before, generated with parameters $r = 200$, $\alpha = 0.1$, and $\Omega = 2$. Additionally, Figure 1 shows the solution time for each algorithm and the speed-up factor of the simplex QP-based algorithms compared to the barrier algorithm as a function of the dimension (n).

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	-



(A) Solution time as a function of dimension.



(B) Speed-up as a function of dimension.

FIGURE 1. Barrier vs the simplex QP-based algorithms.

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 algorithm, often by many factors. For $n = 3200$, the fastest simplex-based algorithm ALG2 is more than 20 times faster than the 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.

4.3. **Discrete instances.** In this section we describe our experiments with the discrete counterpart CDO. As of version 12.6.2 of CPLEX, it is not possible to employ a user-defined convex solver such as Algorithm 1 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 A. 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 A 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 Section 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.

BBBR: Branch-and-bound algorithm in Appendix A, 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 uses extended formulations described in Vielma et al. (2015). Besides presolve, all other parameters are set as in CXLP.

CXD: CPLEX default branch-and-bound algorithm with LP outer approximations.

In all cases the time limit is set to two hours.

Table 5 presents the results for discrete cardinality instances with 200 variables and Table 6 for the discrete path instances with 1,740 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.

Table 5. Comparison for discrete cardinality instances.

		Method	$\Omega = 1$				$\Omega = 2$				$\Omega = 3$			
r	α		time	nodes	egap	#s	time	nodes	egap	#s	time	nodes	egap	#s
100	0.1	BBA1	**1**	**156**	**0.0**	**5**	**26**	**3,271**	**0.0**	**5**	652	68,318	0.0	5
		BBBR	16	156	0.0	5	349	3,270	0.0	5	4,664	43,695	0.1	3
		CXBR	35	276	0.0	5	513	3,497	0.0	5	5,260	32,782	0.2	2
		CXLP	34	9,562	0.0	5	7,200	209,576	0.7	0	7,200	244,911	2.2	0
		CXLPE	2	374	0.0	5	81	7,640	0.0	5	2,629	111,293	0.0	5
		CXD	3	368	0.0	5	37	5,152	0.0	5	**604**	**58,076**	**0.0**	**5**
100	0.5	BBA1	**1**	**87**	**0.0**	**5**	**51**	**6,274**	**0.0**	**5**	**1,323**	**140,874**	**0.0**	**5**
		BBBR	10	87	0.0	5	686	6,274	0.0	5	6,134	56,394	0.3	1
		CXBR	24	183	0.0	5	1,027	6,734	0.0	5	6,399	39,710	0.4	1
		CXLP	294	26,957	0.0	5	7,200	229,641	0.8	0	7,200	263,810	2.3	0
		CXLPE	2	349	0.0	5	191	14,737	0.0	5	4,967	215,292	0.1	3
		CXD	3	373	0.0	5	144	16,070	0.0	5	3,300	245,251	0.0	5
200	0.1	BBA1	**1**	**247**	**0.0**	**5**	**20**	**3,259**	**0.0**	**5**	**388**	**55,248**	**0.0**	**5**
		BBBR	24	247	0.0	5	321	3,259	0.0	5	4,573	39,647	0.1	3
		CXBR	52	460	0.0	5	540	3,711	0.0	5	5,295	34,090	0.2	2
		CXLP	221	17,205	0.0	5	7,200	208,874	0.6	0	7,200	230,304	2.0	0
		CXLPE	4	473	0.0	5	122	6,064	0.0	5	3,376	111,205	0.0	4
		CXD	4	360	0.0	5	52	6,413	0.0	5	1,062	67,577	0.0	5
200	0.5	BBA1	**4**	**674**	**0.0**	**5**	**170**	**24,636**	**0.0**	**5**	**1,140**	**156,632**	**0.0**	**5**
		BBBR	77	674	0.0	5	2,106	17,743	0.0	4	5,590	47,725	0.2	2
		CXBR	104	680	0.0	5	2,452	15,816	0.0	4	6,127	38,973	0.3	1
		CXLP	3,514	120,007	0.1	4	7,200	212,082	1.0	0	7,200	240,445	2.3	0
		CXLPE	18	1,461	0.0	5	1,722	61,593	0.0	4	5,020	198,891	0.2	2
		CXD	16	1,612	0.0	5	1,068	75,098	0.0	5	4,647	299,723	0.1	4
avg		BBA1	**2**	**291**	**0.0**	**20**	**67**	**9,360**	**0.0**	**20**	**876**	**105,268**	**0.0**	**20**
		BBBR	32	291	0.0	20	865	7,637	0.0	19	5,240	46,865	0.2	9
		CXBR	54	400	0.0	20	1,133	7,440	0.0	19	5,770	36,389	0.3	6
		CXLP	1,016	43,433	0.0	19	7,200	215,043	0.8	0	7,200	244,867	2.2	0
		CXLPE	7	664	0.0	20	529	22,508	0.0	19	3,998	159,170	0.1	14
		CXD	7	678	0.0	20	325	25,683	0.0	20	2,403	167,657	0.0	19

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.

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

TABLE 6. Comparison for discrete path instances.

			$\Omega = 1$				$\Omega = 2$				$\Omega = 3$			
r	α	`Method`	`time`	`nodes`	`egap`	`#s`	`time`	`nodes`	`egap`	`#s`	`time`	`nodes`	`egap`	`#s`
100	0.1	BBA1	**256**	**145**	**0.0**	**5**	3,616	1,774	0.0	5	**7,200**	**2,988**	**5.4**	**0**
		BBBR	3,577	91	0.2	4	7,200	184	4.9	0	7,200	236	11.7	0
		CXBR	7,200	67	20.8	0	7,200	79	∞	0	7,200	129	∞	0
		CXLP	533	2,428	0.0	5	7,200	24,776	4.5	0	7,200	20,099	15.0	0
		CXLPE	713	315	0.0	5	6,337	2,432	1.9	1	7,200	2,837	23.2	0
		CXD	1,309	164	0.0	5	**3,361**	**1,176**	**0.0**	**5**	7,200	2,644	6.4	0
100	0.5	BBA1	**589**	**353**	**0.0**	**5**	4,799	2,317	0.4	3	**6,472**	**2,698**	**4.6**	**1**
		BBBR	6,071	134	0.7	2	7,200	175	4.9	0	7,200	162	11.5	0
		CXBR	7,200	24	∞	0	7,200	56	∞	0	7,200	70	∞	0
		CXLP	1,132	6,187	0.0	5	7,200	23,671	4.4	0	7,200	16,851	12.8	0
		CXLPE	903	607	0.0	5	6,207	2,906	2.4	1	7,200	3,128	16.1	0
		CXD	1,532	267	0.0	5	5,189	2,123	0.3	4	7,200	2,645	5.6	0
200	0.1	BBA1	**149**	**77**	**0.0**	**5**	**1823**	**1,075**	**0.0**	**5**	**6,411**	**3,401**	**2.5**	**1**
		BBBR	3,245	76	0.0	5	7,200	171	2.4	0	7,200	180	10.5	0
		CXBR	7,200	34	∞	0	7,200	45	∞	0	7,200	68	∞	0
		CXLP	436	1,548	0.0	5	7,200	30,265	2.9	0	7,200	20,579	12.4	0
		CXLPE	487	188	0.0	5	4,565	1,681	1.0	3	7,200	2,402	17.7	0
		CXD	1,965	106	0.0	5	4,481	1,453	0.4	4	7,200	2,532	4.2	0
200	0.5	BBA1	**292**	**196**	**0.0**	**5**	**3,862**	**2,337**	**0.3**	**4**	**7,200**	**3,703**	**3.7**	**0**
		BBBR	4,826	113	0.2	3	7,200	173	3.9	0	7,200	176	12.7	0
		CXBR	7,200	20	∞	0	7,200	51	∞	0	7,200	89	∞	0
		CXLP	859	4,989	0.0	5	7,200	28,007	4.4	0	7,200	18,873	13.3	0
		CXLPE	923	399	0.0	5	5,730	2,363	1.7	3	7,200	2,691	17.9	0
		CXD	2,028	177	0.0	5	4,752	1,899	0.3	4	7,200	2,775	6.3	0
avg		BBA1	**322**	**193**	**0.0**	**20**	**3,525**	**1,876**	**0.2**	**17**	**6,821**	**3,198**	**4.1**	**2**
		BBBR	4,430	103	0.3	14	7,200	176	4.0	0	7,200	189	11.6	0
		CXBR	7,200	36	∞	0	7,200	58	∞	0	7,200	89	∞	0
		CXLP	740	3,788	0.0	20	7,200	26,680	4.1	0	7,200	19,101	13.4	0
		CXLPE	757	377	0.0	20	5,710	2,346	1.8	8	7,200	2,765	18.7	0
		CXD	1,708	178	0.0	20	4,446	1663	0.3	17	7,200	2,650	5.6	0

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.

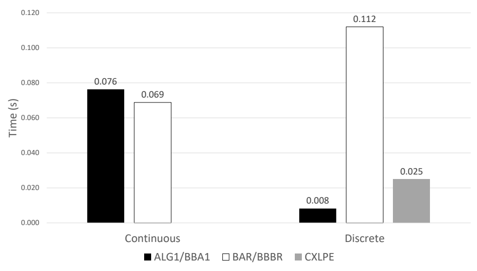
The barrier-based branch-and-bound CXBR, in general, performs poorly. For the cardinality instances, it outperforms CXLP but is slower than the other algorithms. For the path instances it has the worst performance, often struggling to find even a single feasible solution (resulting in infinite end gaps).

Performance of BBA1. Note that BBA1 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 considerably faster than the simplest CPLEX branch-and-bound algorithms CXBR and CXLP.

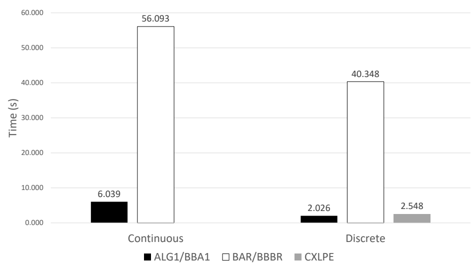
We see that BBA1 outperforms CXLPE (which uses presolve and extended formulations) in all instances. Observe that in the cardinality instances with $\Omega = 1, 2$ and path instances with $\Omega = 1$, BBA1 requires half the number of nodes (or less) compared to CXLPE to solve the instances to optimality (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 is the better choice in 21 of the instance classes considered, while CXD is better in only 2. Moreover, in the instances where CXD is better the difference between the algorithms is small (around 10% difference in solution times), while in the other instances BBA1 is often faster by many factors. We observe that CXD is comparatively better for the instances with a low factor rank ($r = 100$), and BBA1 is comparatively better for the instances with a high factor rank ($r = 200$).

Warm starts. Algorithm BBA1 is faster than BBBR in part due to a faster convex solver (as observed in Section 4.2), and in part due to node warm starts. To quantify the impact of warm starts, we plot in Figure 2 the *time per node* (computed as solution time divided by the number of branch-and-bound nodes) for BBA1, BBBR and CXLPE, and also plot the solution time for the corresponding convex instances with solvers ALG1 and BAR[1].



(A) Cardinality instances



(B) Path instances

FIGURE 2. Time per node.

For the small cardinality instances with 200 variables, Algorithm 1 is slightly worse than the barrier algorithm to solve the convex relaxations; however, it is 15 times faster than barrier when used in branch-and-bound due to the node warm starts from dual feasible solutions. For the larger path instances with 1,740 variables, Algorithm 1 is 10 times faster than the barrier algorithm to solve the convex relaxations, and is about 20 times faster for the discrete instances. Thus node warm starts make the algorithm twice as fast. Finally, observe that the solve time

[1]The time per node is similar for all combinations of parameters Ω, r and α, and thus we plot the average over all parameters.

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.

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.

Acknowledgement

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.

References

Alizadeh, F. (1995). Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM Journal on Optimization*, 5:13–51.

Alizadeh, F. and Goldfarb, D. (2003). Second-order cone programming. *Mathematical programming*, 95:3–51.

Atamtürk, A. and Goméz, A. (2016). Polymatroid inequalities for p-order conic mixed 0-1 optimization. BCOL Research Report 16.02, UC Berkeley.

Atamtürk, A. and Jeon, H. (2017). Lifted polymatroid for mean-risk optimization with indicator variables. BCOL Research Report 17.01, UC Berkeley.

Atamtürk, A. and Narayanan, V. (2007). Cuts for conic mixed-integer programming. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 16–29. Springer Berlin Heidelberg.

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

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

Belotti, P., Kirches, C., Leyffer, S., Linderoth, J., Luedtke, J., and Mahajan, A. (2013). Mixed-integer nonlinear optimization. *Acta Numerica*, 22:1131.

Ben-Tal, A., El Ghaoui, L., and Nemirovski, A. (2009). *Robust optimization.* Princeton University Press.

Ben-Tal, A. and Nemirovski, A. (1998). Robust convex optimization. *Mathematics of Operations Research*, 23:769–805.

Ben-Tal, A. and Nemirovski, A. (1999). Robust solutions of uncertain linear programs. *Operations Research Letters*, 25:1–13.

Ben-Tal, A. and Nemirovski, A. (2001). *Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications.* MPS-SIAM Series on Optimization. SIAM, Philadelphia.

Bertsimas, D. and Sim, M. (2004). Robust discrete optimization under ellipsoidal uncertainty sets.

Borchers, B. and Mitchell, J. E. (1994). An improved branch and bound algorithm for mixed integer nonlinear programs. *Computers & Operations Research*, 21:359–367.

Çay, S. B., Pólik, I., and Terlaky, T. (2017). Warm-start of interior point methods for second order cone optimization via rounding over optimal Jordan frames. ISE Technical Report 17T-006, Lehigh University.

Dantzig, G. B., Orden, A., and Wolfe, P. (1955). The generalized simplex method for minimizing a linear form under linear inequality restraints. *Pacific Journal of Mathematics*, 5:183–196.

El Ghaoui, L., Oks, M., and Oustry, F. (2003). Worst-case value-at-risk and robust portfolio optimization: A conic programming approach. *Operations Research*, 51:543–556.

Hiriart-Urruty, J.-B. and Lemaréchal, C. (2013). *Convex Analysis and Minimization Algorithms I: Fundamentals*, volume 305. Springer Science & Business Media.

Ishii, H., Shiode, S., Nishida, T., and Namasuya, Y. (1981). Stochastic spanning tree problem. *Discrete Applied Mathematics*, 3:263–273.

Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, pages 302–311. ACM.

Leyffer, S. (2001). Integrating SQP and branch-and-bound for mixed integer nonlinear programming. *Computational Optimization & Applications*, 18:295–309.

Lobo, M. S., Vandenberghe, L., Boyd, S., and Lebret, H. (1998). Applications of second-order cone programming. *Linear Algebra & its Applications*, 284:193–228.

Megiddo, N. (1991). On finding primal- and dual-optimal bases. *INFORMS Journal on Computing*, 3:63–65.

Nemirovski, A. and Scheinberg, K. (1996). Extension of Karmarkar's algorithm onto convex quadratically constrained quadratic problems. *Mathematical Programming*, 72:273–289.

Nesterov, Y. and Nemirovski, A. (1994). *Interior-Point Polynomial Algorithms in Convex Programming.* Society for Industrial and Applied Mathematics.

Nesterov, Y. E. and Todd, M. J. (1998). Primal-dual interior-point methods for self-scaled cones. *SIAM Journal on Optimization*, 8:324–364.

Tawarmalani, M. and Sahinidis, N. V. (2005). A polyhedral branch-and-cut approach to global optimization. *Mathematical Programming*, 103(2):225–249.

Van de Panne, C. and Whinston, A. (1964). Simplicial methods for quadratic programming. *Naval Research Logistics Quarterly*, 11(3-4):273–302.

Vielma, J. P., Dunning, I., Huchette, J., and Lubin, M. (2015). Extended formulations in mixed integer conic quadratic programming. *Mathematical Programming Computation.* Forthcoming. doi:10.1007/s12532-016-0113-y.

Wolfe, P. (1959). The simplex method for quadratic programming. *Econometrica: Journal of the Econometric Society*, pages 382–398.

Yildirim, E. A. and Wright, S. J. (2002). Warm-start strategies in interior-point methods for linear programming. *SIAM Journal on Optimization*, 12:782–810.

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).

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:     $(S, B, lb) \leftarrow$ PULL$(L)$                   ▷ select and remove one element from $L$
 6:     if $lb \geq ub$ then
 7:         go to line 4
 8:     end if
 9:     $(x, \bar{B}, z) \leftarrow$ 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:         $(S_{\leq}, S_{\geq}) \leftarrow$ BRANCH$(x)$   ▷ create two subproblems
18:         $L \leftarrow L \cup \{(S_{\leq}, \bar{B}, z), (S_{\geq}, \bar{B}, z)\}$   ▷ add the subproblems to $L$
19:     end if
20: end while
21: return $x^*$
```

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