Math. Prog. Comp. (2012) 4:33–52
DOI 10.1007/s12532-011-0033-9

FULL LENGTH PAPER

Globally solving nonconvex quadratic programming problems via completely positive programming

Jieqiu Chen · Samuel Burer

Received: 24 February 2011 / Accepted: 10 October 2011 / Published online: 16 November 2011

Abstract Nonconvex quadratic programming (QP) is an NP-hard problem that optimizes a general quadratic function over linear constraints. This paper introduces a new global optimization algorithm for this problem, which combines two ideas from the literature—finite branching based on the first-order KKT conditions and polyhedral-semidefinite relaxations of completely positive (or copositive) programs. Through a series of computational experiments comparing the new algorithm with existing codes on a diverse set of test instances, we demonstrate that the new algorithm is an attractive method for globally solving nonconvex QP.

Keywords Nonconvex quadratic programming · Global optimization · Branch-and-bound · Semidefinite programming · Copositive programming · Completely positive programming

Mathematics Subject Classification (2000) 90C20 · 90C22 · 90C26

The research of J. Chen and S. Burer was supported in part by NSF Grant CCF-0545514. J. Chen was supported in part by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357.

J. Chen (✉)
Mathematics and Computer Science Division, Argonne National Laboratory,
Argonne, IL 60439, USA
e-mail: jieqchen@mcs.anl.gov

S. Burer
Department of Management Sciences, University of Iowa,
Iowa City, IA 52242-1994, USA
e-mail: samuel-burer@uiowa.edu

1 Introduction

We consider the problem of optimizing a general quadratic function subject to linear and bound constraints:

$$\begin{aligned} \min \quad & \frac{1}{2} x^T H x + f^T x \\ \text{s.t.} \quad & Ax \leq b \\ & A_{eq}\, x = b_{eq} \\ & l \leq x \leq u, \end{aligned} \tag{QP}$$

where $x \in \Re^n$ is the variable and $H \in \Re^{n\times n}$, $f \in \Re^n$, $A \in \Re^{m\times n}$, $b \in \Re^m$, $A_{eq} \in \Re^{m_{eq}\times n}$, $b_{eq} \in \Re^{m_{eq}}$, $l \in \Re^n$, and $u \in \Re^n$ are the data. Without loss of generality, H is symmetric, and we assume H is *not* positive semidefinite, which implies generally that (QP) is nonconvex and NP-hard [21]. Components of the vectors l and u are allowed to be infinite, but we assume the feasible set of (QP) is bounded. We also assume without loss of generality that (QP) has an interior point and A_{eq} has full row rank.

Problem (QP) arises in many application areas, such as inventory theory [14], scheduling [24], and free boundary problems [13]. Moré and Toraldo [17] mention several applications of the box-constrained case, namely, when the only constraints are $l \leq x \leq u$, and make a connection with the linear complementarity problem. Extensions of (QP) that incorporate additional nonlinearities—for example, variants of the trust-region subproblem within trust-region methods [8] and subproblems in robust linear regression [18]—are relevant in practice. In this sense, (QP) is also an important substructure occurring in other optimization problems.

Because (QP) is nonconvex, a local minimum may not be a global minimum. Many methods for (QP) employ nonlinear programming techniques such as active-set or interior-point methods to calculate critical points that satisfy the Karush-Kuhn-Tucker (KKT) conditions with a good objective value. Gould and Toint [12] survey numerous such methods, and the function `quadprog` in Matlab's Optimization Toolbox [16] is a widely available implementation. Various techniques are also available to find a global solution of (QP), and Pardalos [20] surveys several of these global optimization methods. A recent method, which is closely related to this paper, is by Burer and Vandenbussche [6]. Excellent off-the-shelf software packages, such as BARON [22] and Couenne [1], are also available for the solution of (QP).

In this paper, we propose a new method for globally optimizing (QP). Our approach follows that of Burer and Vandenbussche [6] by employing a finite branch-and-bound (B&B) scheme, in which branching is based on the first-order KKT conditions and polyhedral-semidefinite relaxations are solved at each node of the B&B tree. However, our method differs from theirs in that our relaxations have a different structure and are solved with a different algorithm that exploits this structure.

In particular, our relaxations are derived from *completely positive* (CP) and *doubly nonnegative* (DNN) programs as specified by Burer [3,4]. (Completely positive programs are also known as *copositive programs*.) We connect (QP) with CP programs by reformulating (QP) as a quadratic program with linear equality, nonnegativity, and

complementarity constraints. Burer [3] shows that such problems can be formulated as CP programs, which are convex programs that optimize a linear function over the convex cone of completely positive matrices subject to linear constraints. Since even CP programs are NP-hard, however, one must relax them in practice. Relaxing a CP program in a natural way yields a DNN program, whose distinguishing feature is its matrix variable, which is both positive semidefinite and entrywise nonnegative. Furthermore, Burer [4] develops a specialized algorithm for these DNN programs and demonstrates that similar quality bounds can be obtained in less time compared with the type of relaxations employed by Burer and Vandenbussche [6].

The main goal of this paper is to demonstrate that merging finite KKT-based B&B with completely positive programming yields an attractive method for globally solving (QP). To show this, we conduct computational experiments on 243 diverse instances of (QP) and compare our method with the method of Burer and Vandenbussche [6] and with the general-purpose global solver Couenne. The results demonstrate that our algorithm solves (QP) faster than the one by Burer and Vandenbussche on most of the test instances. In addition, our algorithm performs well compared to Couenne. For example, on nearly all instances that took Couenne more than 1,000 s to solve, our algorithm requires about the same or less time. In addition, Couenne runs out of time or memory more often than our algorithm does on the test instances.

A secondary goal of this paper is to provide a convenient global optimization solver for (QP) that can be used by practitioners and researchers alike. Our implementation is based in Matlab, uses the same syntax as the local optimization routine `quadprog`, and requires only an external linear programming solver. In contrast, the method by Burer and Vandenbussche [6] requires a convex quadratic programming solver. In Sect. 5, we illustrate the convenience of our implementation by using it as a subroutine to solve a more general nonlinear program motivated by trust-region methods.

The paper is organized as follows. In Sect. 2, we provide background on the finite KKT-based B&B method of Burer and Vandenbussche [6] and the approach of Burer [3,4] involving CP and DNN programs. In Sect. 3, we describe our reformulation of (QP) as a CP program using the KKT conditions and discuss the important technical issue of bounding the dual variables in the CP formulation. Section 4 combines the ingredients from Sects. 2 and 3 to describe our B&B implementation and conduct the computational experiments. Section 5 details the trust-region example. In Sect. 6, we conclude the paper with a few brief remarks on possible ways to improve our approach.

Notation and terminology. We let $e \in \Re^n$ represent the vector of all ones. For an index i and vector x, we use x_i to denote the ith entry of x. For an index set $F \subseteq \{1, \ldots, n\}$, we define $x_F \in \Re^{|F|}$ as the vector composed of entries of x that are indexed by F. The notation $X \succeq 0$ means that the matrix X is symmetric positive semidefinite, and $\mathrm{diag}(X)$ denotes the main diagonal of X as a vector. The inner product of two matrices $A, B \in \Re^{n \times n}$ is defined as $A \bullet B := \mathrm{trace}(A^T B)$; $\circ$ represents the Hadamard product, that is, the componentwise product.

2 Background

In this section, we briefly review the finite B&B method [6] and the CP and DNN techniques [3,4] mentioned in the introduction.

2.1 The finite branch-and-bound method

The finite B&B method proposed by Burer and Vandenbussche [6] works by first incorporating into (QP) its first-order KKT conditions and then enforcing the first-order KKT conditions through branching. The primary advantage of such an approach is that the B&B tree is finite, unlike other global optimization approaches based on spatial branching. A secondary advantage is that this approach allows one to develop stronger relaxations of (QP), for example, ones that involve the KKT conditions.

For the sake of simplicity, we discuss finite branching with respect to a simplified version of (QP) that has no equality constraints and no explicit lower and upper bounds; in other words, the feasible set is simply $\{x : Ax \leq b\}$.

Specifically, as illustrated by the following program, the authors start with the quadratic program (1)–(2) and incorporate its first-order KKT conditions (3)–(4):

$$\begin{aligned} \min \quad & \frac{1}{2}x^T Hx + f^T x && (1)\\ \text{s.t.} \quad & Ax \leq b && (2)\\ & Hx + f + A^T\gamma = 0, \quad \gamma \geq 0 && (3)\\ & (b - Ax)_i\gamma_i = 0 \ \ \forall\, i = 1, \ldots, m. && (4) \end{aligned}$$

Then, at each node in the B&B tree, the authors introduce index sets $F^{(b-Ax)}$, $F^{\gamma} \subseteq \{1, \ldots, m\}$ and replace (4) by

$$\begin{aligned} (b - Ax)_i &= 0 \quad \forall\, i \in F^{(b-Ax)}\\ \gamma_i &= 0 \quad\ \ \forall\, i \in F^{\gamma}. \end{aligned} \tag{5}$$

Note that (5) enforces the complementarity conditions $(b - Ax)_i\gamma_i = 0$ only for those $i \in F^{(b-Ax)} \cup F^{\gamma}$. At the root node, the authors set $F^{(b-Ax)} = F^{\gamma} = \emptyset$. Then the idea is to branch by adding indices to $F^{(b-Ax)}$ or F^{γ}, progressively enforcing more complementarities at nodes deeper in the tree. In particular, branching on a node involves selecting an index $i \in \{1, \ldots, m\}\backslash(F^{(b-Ax)} \cup F^{\gamma})$ and creating two children by adding i to $F^{(b-Ax)}$ for one child and adding i to F^{γ} for the other. Branching maintains $F^{(b-Ax)} \cap F^{\gamma} = \emptyset$ always, and a leaf node is characterized by $F^{(b-Ax)} \cup F^{\gamma} = \{1, \ldots, m\}$, which enforces (4) fully.

At any node, note that (1)–(3) and (5) still constitute a nonconvex problem because of the quadratic objective. Thus, a convex relaxation is constructed and solved to compute a lower bound for that node. There are many choices for the convex relaxations, and a particular SDP relaxation is constructed by Burer and Vandenbussche [6]. With that SDP relaxation, the authors show that any leaf node can be pruned and thus their B&B scheme is correct and finite. We mention that for our method, presented in Sect. 3, the same types of results can be shown ensuring the finiteness and correctness of our algorithm also.

2.2 Doubly nonnegative programs

Consider a quadratic program having linear equality, nonnegativity, and complementarity constraints:

$$\begin{aligned} \min \quad & \frac{1}{2}\tilde{x}^T \tilde{H}\tilde{x} + \tilde{f}^T\tilde{x} && \text{(NQP)} \\ \text{s.t.} \quad & \tilde{A}\tilde{x} = \tilde{b},\ \tilde{x} \geq 0 \\ & \tilde{x}_i\tilde{x}_j = 0 \quad \forall\,(i,j) \in E, && (6) \end{aligned}$$

where E is a fixed set of pairs. Burer [3] has shown (NQP) is equivalent to a *completely positive program*

$$\begin{aligned} \min \quad & \frac{1}{2}\tilde{H} \bullet \tilde{X} + \tilde{f}^T\tilde{x} && \text{(CPP)} \\ \text{s.t.} \quad & \tilde{A}\tilde{x} = \tilde{b},\ \operatorname{diag}(\tilde{A}\tilde{X}\tilde{A}^T) = \tilde{b} \circ \tilde{b} \\ & \tilde{X}_{ij} = 0 \ \ \forall\,(i,j) \in E \\ & \begin{pmatrix} 1 & \tilde{x}^T \\ \tilde{x} & \tilde{X} \end{pmatrix} \in \mathcal{C}, \end{aligned}$$

where $\mathcal{C}$ is the completely positive cone

$$\mathcal{C} := \left\{ X \in \mathcal{S}_n \mid \exists \text{ integer } k,\ B \in \Re_{n\times k},\ B_{ij} \geq 0,\ X = BB^T \right\}.$$

(CPP) naturally has the following SDP relaxation:

$$\begin{aligned} \min \quad & \frac{1}{2}\tilde{H} \bullet \tilde{X} + \tilde{f}^T\tilde{x} && \text{(DNP)} \\ \text{s.t.} \quad & \tilde{A}\tilde{x} = \tilde{b},\ \operatorname{diag}(\tilde{A}\tilde{X}\tilde{A}^T) = \tilde{b} \circ \tilde{b} \\ & \tilde{X}_{ij} = 0 \ \ \forall\,(i,j) \in E \\ & \begin{pmatrix} 1 & \tilde{x}^T \\ \tilde{x} & \tilde{X} \end{pmatrix} \geq 0,\ (\tilde{x}, \tilde{X}) \geq 0, \end{aligned}$$

where the semidefinite matrix is called *doubly nonnegative* because both its eigenvalues and entries are nonnegative. The major contribution of [4] is an algorithm to solve (DNP) efficiently, yet approximately. The algorithm also produces dual bounds, which makes it appropriate for use in a B&B algorithm. The implementation of the algorithm requires an additional input, namely, finite upper bounds on the variable $\tilde{x}$. For example, assuming (NQP) is bounded, such bounds can be obtained in a preprocessing phase by solving several linear programming (LP) problems. Note that these upper bounds may not appear explicitly in (DNP).

Related to this issue of boundedness, in Sect. 3.2 we discuss how to bound the dual variables arising from the KKT approach of Sect. 2.1, which will ultimately become

components of $\tilde{x}$ in (NQP). A similar issue was faced by Burer and Vandenbussche [6], and we will discuss our approach for dealing with it. Otherwise, we will not review the details of the algorithm for (DNP), since we can treat it as an available subroutine and simply embed it in the finite B&B method.

3 Reformulation and bounding

In this section, we first discuss the steps required to reformulate (QP) as (NQP). In short, (NQP) serves as a kind of standard form, and the reformulation involves techniques such as introducing the KKT system of (QP), representing inequalities as equalities, and shifting and scaling variables. While the transformation is not particularly complicated, it warrants discussion because it is critical to our approach. After the reformulation, we show how to bound the dual variables that arise when formulating the KKT system (see also Sect. 2.2).

We caution the reader that, in order to simplify the notation, we sometimes abuse it. For example, the same variable x may be used before and after a shifting or scaling.

3.1 Reformulation

There are several ways of reformulating (QP) as (NQP). Our fundamental approach is to incorporate the first-order KKT conditions of (QP) into its formulation. This gives rise to the complementarity conditions in (NQP) that will become the basis of branching as discussed in Sect. 2.1. This approach also allows us to construct potentially stronger semidefinite relaxations.

Since we employ the finite B&B framework, it is also desirable to keep the number of complementarity conditions minimal in order to have a small B&B tree. In fact, we would like the number of complementarities in (NQP) to be exactly the same as the number naturally occurring in the KKT system of (QP). For example, suppose x_j is a free variable in (QP) with $(l_j, u_j) = (-\infty, +\infty)$, which is nevertheless bounded by assumption. To put (QP) in the form of (NQP), one could split x_j into the difference of nonnegative variables before constructing the KKT system, but doing so would introduce two more complementarities compared with first constructing the KKT system and then splitting x_j. There is an even better way to handle such x_j, as we describe next.

To describe our explicit transformation—the one that has been implemented exactly in our code—we group the variables x_j of (QP) into categories based on the following index sets:

$$\begin{aligned}
L &:= \{j : -\infty < l_j, u_j = +\infty\} && \text{lower bounds only}\\
U &:= \{j : -\infty = l_j, u_j < +\infty\} && \text{upper bounds only}\\
B &:= \{j : -\infty < l_j, u_j < +\infty\} && \text{both lower and upper bounds}\\
F &:= \{j : -\infty = l_j, u_j = +\infty\} && \text{free.}
\end{aligned}$$

Using the transformation $x_j \to u_j - x_j$, variables x_j with $j \in U$ can easily be transformed into variables with $j \in L$. In other words, we may assume $U = \emptyset$. In a similar

manner, we may also assume $l_j = 0$ for all $j \in L$ and $(l_j, u_j) = (0, 1)$ for all $j \in B$. Then (QP) becomes

$$\begin{aligned} \min \quad & \frac{1}{2} x^T H x + f^T x \\ \text{s.t.} \quad & Ax \le b \\ & A_{eq}\, x = b_{eq} \\ & 0 \le x_L,\ 0 \le x_B \le e,\ x_F \text{ free}. \end{aligned} \tag{QP$'$}$$

To form the KKT conditions, we introduce multipliers $\gamma \ge 0$, y free, $\lambda_L \ge 0$, $\lambda_B \ge 0$, and $\rho_B \ge 0$ for the respective inequality constraints of (QP$'$). These give the Lagrangian

$$\frac{1}{2} x^T H x + f^T x - \gamma^T (b - Ax) - y^T (b_{eq} - A_{eq} x) - \lambda_L^T x_L - \lambda_B^T x_B - \rho_B^T (e - x_B)$$

with optimality conditions

$$\begin{aligned} & Hx + f + A^T \gamma + A_{eq}^T y - \lambda_L - \lambda_B + \rho_B = 0 \\ & Ax \le b,\ A_{eq} x = b_{eq},\ 0 \le x_L,\ 0 \le x_B \le e,\ x_F \text{ free} \\ & \gamma \ge 0,\ y \text{ free},\ \lambda_L \ge 0,\ \lambda_B \ge 0,\ \rho_B \ge 0 \\ & (b - Ax) \circ \gamma = 0,\ x_L \circ \lambda_L = 0,\ x_B \circ \lambda_B = 0,\ (e - x_B) \circ \rho_B = 0, \end{aligned} \tag{7}$$

where, in the first equation, the vectors λ_L, λ_B, and ρ_B are understood to have zeros in their indices that do not belong to L, B, and B, respectively. Note we also have the implied complementarity

$$\lambda_B \circ \rho_B = 0, \tag{8}$$

which is valid because $x_i \ge 0$ and $1 - x_i \ge 0$ for all $i \in B$ cannot be active at the same time. We also enforce (8) when branching in the B&B scheme; see Sect. 4.1 for details.

Since (NQP) allows only equality constraints, we introduce slack variables (s, w_B) to turn inequalities into equalities and arrive at the following:

$$Hx + f + A^T \gamma + A_{eq}^T y - \lambda_L - \lambda_B + \rho_B = 0 \tag{9}$$

$$Ax + s = b,\ A_{eq} x = b_{eq},\ x_B + w_B = e,\ (x_L, x_B, s, w_B) \ge 0,\ x_F \text{ free} \tag{10}$$

$$y \text{ free},\ (\gamma, \lambda_L, \lambda_B, \rho_B) \ge 0$$

$$s \circ \gamma = 0,\ x_L \circ \lambda_L = 0,\ x_B \circ \lambda_B = 0,\ w_B \circ \rho_B = 0,\ \lambda_B \circ \rho_B = 0. \tag{11}$$

This format fits (NQP) except that y and x_F are free. Recall our assumption that the feasible set of (QP) is bounded, and so bounds on x_F can be calculated by solving some LPs; more details are given in Sect. 3.2. We also postpone the discussion of how to bound y to Sect. 3.2.

Suppose now that y and x_F have available lower bounds. Again, we can shift the variables such that their lower bounds are zeros. We thus obtain

$$\begin{aligned} &Hx + f + A^T\gamma + A_{eq}^T y - \lambda_L - \lambda_B + \rho_B = r \\ &Ax + s = b,\ A_{eq}x = b_{eq},\ x_B + w_B = e,\ (x, s, w_B) \geq 0 \\ &(\gamma, y, \lambda_L, \lambda_B, \rho_B) \geq 0 \\ &s \circ \gamma = 0,\ x_L \circ \lambda_L = 0,\ x_B \circ \lambda_B = 0,\ w_B \circ \rho_B = 0,\ \lambda_B \circ \rho_B = 0, \end{aligned} \tag{12}$$

where r is a constant vector resulting from the shift.

Now we are ready to reformulate (QP′) as (NQP) using (12). We stack all the primal variables (x, s, w_B) and the dual multipliers $(\gamma, y, \lambda_L, \lambda_B, \rho_B)$ together to form the new variable $\tilde{x} \geq 0$. It is easy to formulate appropriate $(\tilde{A}, \tilde{b})$ such that $\tilde{A}\tilde{x} = \tilde{b}$ models all the linear equalities in (12), and E is constructed to encapsulate all the complementarity conditions in (12) via the constraints $\tilde{x}_i\tilde{x}_j = 0$ for all $(i, j) \in E$.

3.2 Finite bounds on dual variables

As mentioned in Sect. 2.2, the algorithm we have chosen for solving (DNP) requires finite upper bounds on the nonnegative variables $\tilde{x}_j$. Thus we need explicit finite lower and upper bounds on all the original primal and dual variables, which can easily be translated into upper bounds on the $\tilde{x}_j$'s.

First, we show how to compute bounds on x through LP-based preprocessing. By assumption, the primal feasible set is bounded even though l and u may have nonfinite components. Then, if a lower (upper) bound is needed for x_j, we solve the following LP:

$$\min\ (\max)\ \left\{x_j : Ax \leq b,\ A_{eq}x = b_{eq},\ l \leq x \leq u\right\}.$$

Next we discuss the dual variables. We start by proving that they are indeed bounded. In order to bound y, the KKT conditions alone are not sufficient. We further need

$$x^T Hx + f^T x + b^T\gamma + b_{eq}^T y + e^T\rho_B = 0, \tag{13}$$

which is obtained from (9)–(11). Indeed, multiplying x^T on both sides of (9) yields

$$x^T Hx + x^T f + x^T A^T\gamma + x^T A_{eq}^T y - x_L^T\lambda_L - x_B^T\lambda_B + x_B^T\rho_B = 0,$$

and then (10) and (11) simplify the above equality to (13). Note that $x_B^T\rho_B$ simplifies to $e^T\rho_B$ since $w_B^T\rho_B = 0$ and thus

$$x_B^T\rho_B = (x_B + w_B)^T\rho_B = e^T\rho_B.$$

To simplify the proof, we assume both x and its slack s are bounded below by 0 and above by e; indeed, we can first compute bounds for x and s and then shift and scale. We then define

$$\hat{A} := \begin{pmatrix} A & I \\ A_{eq} & 0 \end{pmatrix}, \quad \hat{b} := \begin{pmatrix} b \\ b_{eq} \end{pmatrix}, \quad \hat{x} := \begin{pmatrix} x \\ s \end{pmatrix},$$

so that the feasible set can be stated as

$$P := \{\hat{x} \in \Re^{n+m} : \ \hat{A}\hat{x} = \hat{b}, \ 0 \le \hat{x} \le e\}.$$

We let $\hat{y}$ be the dual multiplier for $\hat{A}\hat{x} = \hat{b}$ and introduce $\hat{\lambda} \ge 0$ and $\hat{\rho} \ge 0$ as the dual multipliers for $0 \le \hat{x} \le e$. Note that the dual multipliers in (12) are simply subvectors of $(\hat{y}, \hat{\lambda}, \hat{\rho})$, and so the boundedness of $(\hat{y}, \hat{\lambda}, \hat{\rho})$ will imply that of the dual multipliers.

It is helpful to restate the first equation of (12) in terms of $(\hat{y}, \hat{\lambda}, \hat{\rho})$ as follows:

$$\hat{H}\hat{x} + \hat{f} + \hat{A}^T\hat{y} - \hat{\lambda} + \hat{\rho} = 0, \tag{14}$$

where $\hat{H} := \begin{pmatrix} H & 0 \\ 0 & 0 \end{pmatrix}$, $\hat{f} := \begin{pmatrix} f \\ 0 \end{pmatrix}$. Similarly, the equivalent version of (13) is

$$\hat{x}^T\hat{H}\hat{x} + \hat{f}^T\hat{x} + \hat{b}^T\hat{y} + e^T\hat{\rho} = 0. \tag{15}$$

Note that (15) is nonlinear, and so we first linearize it by applying the reformulation-linearization (RLT) technique of Sherali and Adams [23]. In particular, we introduce a new matrix variable $\hat{X}$ to replace $\hat{x}\hat{x}^T$, namely, $\hat{X}_{ij} = \hat{x}_i\hat{x}_j$ componentwise, which allows us to write (15) as

$$\hat{H} \bullet \hat{X} + \hat{f}^T\hat{x} + \hat{b}^T\hat{y} + e^T\hat{\rho} = 0. \tag{16}$$

RLT also produces the following valid inequalities:

$$0 \le \hat{X}_{i,j} \le \min\{\hat{x}_i, \hat{x}_j\}, \quad \forall\, 1 \le i \le j \le n+m \tag{17}$$

$$1 - \hat{x}_i - \hat{x}_j + \hat{X}_{i,j} \ge 0, \quad \forall\, 1 \le i \le j \le n+m, \tag{18}$$

which further tighten the relationship between $\hat{X}$ and $\hat{x}$. In light of (17) and (18), the boundedness of $\hat{x}$ implies that of $\hat{X}$, and the following set is bounded:

$$\hat{P} := \left\{(\hat{x}, \hat{X}) \ : \ \hat{x} \in P, \ (17), (18)\right\}.$$

In other words, the recession cone $\hat{P}^0$ of $\hat{P}$ is trivial, that is, $\hat{P}^0 = \{0\}$.

We are then ready to prove that $(\hat{y}, \hat{\lambda}, \hat{\rho})$ is bounded. We prove this by demonstrating that the recession cone of the set defined collectively by the primal, dual variables and their associated valid inequalities is empty. The recession cone of P, (14), and (16)–(18) is as follows:

$$R^0 := \left\{ \begin{array}{ccc} (\triangle\hat{x}, \triangle\hat{X}) \ge 0 & & (\triangle\hat{x}, \triangle\hat{X}) \in \hat{P}^0 \\ \triangle\hat{y} \text{ free} & : & \hat{H}\triangle\hat{x} + \hat{A}^T\triangle\hat{y} - \triangle\hat{\lambda} + \triangle\hat{\rho} = 0 \\ (\triangle\hat{\lambda}, \triangle\hat{\rho}) \ge 0 & & \hat{H} \bullet \triangle\hat{X} + \hat{f}^T\triangle\hat{x} + \hat{b}^T\triangle\hat{y} + e^T\triangle\hat{\rho} = 0 \end{array} \right\}.$$

Since $\hat{P}^0 = \{0\}$, R^0 simplifies to

$$R^0 = \left\{ \begin{array}{c} (\triangle\hat{x}, \triangle\hat{X}) = (0,0) \\ (\triangle\hat{\lambda}, \triangle\hat{\rho}) \geq 0 \\ \triangle\hat{y} \text{ free} \end{array} : \begin{array}{c} \hat{A}^T\triangle\hat{y} - \triangle\hat{\lambda} + \triangle\hat{\rho} = 0 \\ \hat{b}^T\triangle\hat{y} + e^T\triangle\hat{\rho} = 0 \end{array} \right\}.$$

Now we define

$$R := \left\{ \begin{array}{c} (\triangle\hat{\lambda}, \triangle\hat{\rho}) \geq 0 \\ \triangle\hat{y} \text{ free} \end{array} : \begin{array}{c} \hat{A}^T\triangle\hat{y} - \triangle\hat{\lambda} + \triangle\hat{\rho} = 0 \\ \hat{b}^T\triangle\hat{y} + e^T\triangle\hat{\rho} = 0 \end{array} \right\}.$$

R is the projection of the recession cone of R^0 onto the variables $(\triangle\hat{y}, \triangle\hat{\lambda}, \triangle\hat{\rho})$. Hence, it suffices to show that $R = \{0\}$. We do so by exploiting the assumptions that (QP) has an interior point and A_{eq} has full row rank, which imply that P contains an interior point and $\hat{A}$ has full row rank.

Proposition 3.1 *$R = \{0\}$, and hence the dual variables $(\hat{y}, \hat{\lambda}, \hat{\rho})$ are bounded.*

Proof Consider the LP $\max\{0 : \hat{x} \in P\}$ and its dual

$$\min \left\{ \hat{b}^T\triangle\hat{y} + e^T\triangle\hat{\rho} : \begin{array}{c} \hat{A}^T\triangle\hat{y} - \triangle\hat{\lambda} + \triangle\hat{\rho} = 0 \\ (\triangle\hat{\lambda}, \triangle\hat{\rho}) \geq 0,\ \triangle\hat{y} \text{ free} \end{array} \right\}.$$

Strong duality ensures $\hat{b}^T\triangle\hat{y} + e^T\triangle\hat{\rho} = 0$ at optimality, and so the dual optimal solution set is precisely R.

Now let $(\triangle\hat{y}^*, \triangle\hat{\lambda}^*, \triangle\hat{\lambda}^*) \in R$ be any optimal solution of the dual, and let $\hat{x}^0$ be an interior point of P, which is obviously an optimal solution to the primal problem since the objective function is $f(\hat{x}) := 0$. Complementary slackness implies $(\triangle\hat{\lambda}^*, \triangle\hat{\rho}^*) = (0,0)$ since $0 < \hat{x}^0 < e$. Then the dual linear constraint simplifies to $\hat{A}^T\triangle\hat{y}^* = 0$. Since $\hat{A}$ has full row rank, $\triangle\hat{y}^* = 0$. Hence, $(\triangle\hat{y}^*, \triangle\hat{\lambda}^*, \triangle\hat{\lambda}^*) = (0,0,0)$, which demonstrates $R = \{0\}$. □

Now that we know the dual variables $(\hat{y}, \hat{\lambda}, \hat{\rho})$ are bounded, to actually compute bounds for them, we again solve some LPs. The constraints of the LPs are exactly the valid constraints we derived based on optimality conditions and RLT techniques. For example, to compute the lower and upper bounds for the kth component of $\hat{y}$, we solve the following pair of LPs:

$$\min/\max \left\{ \hat{y}_k : \hat{x} \in P,\ (14),\ (16)\text{–}(18) \right\}. \tag{19}$$

We compute the upper bounds for $\hat{\lambda}$ and $\hat{\rho}$ using the same linear programs but with modified objective functions. These bounds must be computed during preprocessing and reformulation because they are required as input later by the algorithm that solves the relaxation in the form of (DNP) at each node of the B&B scheme.

4 Computational results

In this section, we describe our implementation of the finite B&B algorithm to globally solve (QP), and we compare its performance with that of two other methods on a diverse set of test problems. The code has been made publicly available at http://dollar.biz.uiowa.edu/~sburer under the keyword `QuadprogBB`.

4.1 Implementation details

The implementation consists of two main components: (1) the reformulation of (QP) into (NQP) described in Sect. 3.1, including the calculation of bounds, and (2) the B&B scheme described in Sect. 2.1.

The reformulation includes a preprocessing phase in which we remove fixed variables and check whether the assumptions of the method are satisfied, in other words, whether (QP) has an interior point and A_{eq} has full row rank. After preprocessing, we recast (QP) as (QP′) by flipping some variables and shifting and scaling others. Then we formulate the first-order KKT conditions of (QP′) as detailed in Sect. 3.1. With the KKT system available, we compute bounds for each primal and dual variable by solving LPs, for example, ones of the form (19) for the dual variables. These LPs are the most time-consuming part of the reformulation.[1] We used warm-start to speed up the solution of LPs. Specifically, we save the row and column basis information from the solution of the previous LP, and use the basis information to warm start the next LP solve. The last step of the reformulation puts the problem into the form (NQP) and scales each of the nonnegative variables in (NQP) to have an implicit upper bound of 1, which is done because we found it improved the numerical precision of the overall algorithm. Also, it facilitates our branching rule, as described below.

The B&B algorithm, as with all such algorithms, involves four types of calculations: upper bounds, lower bounds, fathoming, and branching. At each node of the tree, we apply a nonlinear programming method to obtain a local solution of (NQP) without the complementarity condition (6), which provides a currently best global upper bound (GUB). In our implementation, we use the `quadprog` function in Matlab's Optimization Toolbox to solve the QP. `Quadprog` finds a locally optimal solutions to QPs in a fast and reliable way. At any given node, this calculation is initialized with the value of $\tilde{x}$ from the node's relaxation, which increases the diversity of the search for a good GUB throughout the tree. For lower bounds, at each node we solve the DNN relaxation (DNP) of (NQP) tailored only to respect the complementarities enforced at that node. For example, we solve the root relaxation with no complementarity conditions enforced. At any particular node, if the lower bound obtained by

[1] We also experimented with bounding the sum of several nonnegative variables at a time (say, $\sum_k y_k$). The idea was that this would still produce bounds—albeit looser ones—for individual variables but would require the solution of fewer LPs. However, we found that the looser bounds resulted in much longer time spent in the subsequent B&B calculations. So we decided to stick with calculating separate bounds for each variable.

solving the relaxation is within a prespecified tolerance of GUB, then that node is fathomed.

If a node cannot be fathomed, then branching involves the enforcement of the complementarity conditions by setting variables to 0 as discussed in Sect. 2.1. In our implementation, we branch on the following complementarities from (12):

$$\begin{aligned} s \circ \gamma &= 0 \\ x_L \circ \lambda_L &= 0 \\ x_B \circ \lambda_B &= 0 \\ w_B \circ \rho_B &= 0. \end{aligned}$$

A violated complementarity condition, say $s_j\gamma_j > 0$, is selected and $s_j\gamma_j = 0$ is enforced in the two children nodes, where one node enforces $s_j = 0$ and the other node enforces $\gamma_j = 0$. Note that we do not branch on the implied complementarity (8). However, we maintain (8) when we create the children nodes in the B&B tree. Specifically, if the complementarity to branch on is $x_j\lambda_j = 0$, $j \in B$, we will enforce $\rho_j = 0$ in the node $x_j = 0$; similarly, if we branch on $w_j\rho_j = 0$, $j \in B$, we will enforce $\lambda_j = 0$ in the node $w_j = 0$. In both cases, we maintain the implied complementarity $\lambda_j\rho_j = 0$.

More details of the B&B algorithm are as follows:

- As mentioned in Sect. 2.2, we solve the relaxations (DNP) using the specialized decomposition technique proposed by Burer [4], which is efficient and always provides a valid lower bound on the original QP. In particular, the relaxation would be expensive to solve by interior-point methods.
- When selecting the next active node to solve, we choose the one that has the lowest lower bound.
- We use a relative optimality tolerance for fathoming. For a given tolerance ϵ, a node with lower bound v_{lb} is fathomed if $(\text{GUB} - v_{lb})/\max\{|\text{GUB}|, 1\} < \epsilon$. In our implementation, we set $\epsilon = 10^{-6}$.
- When choosing the complementarities to branch on, we employ a maximum violation approach. Given a solution $\tilde{x}$, we select the index (i, j), where $\tilde{x}_i\tilde{x}_j$ is maximum among all violated complementarities. We point out that the products $\tilde{x}_i\tilde{x}_j$ are properly scaled since every component of $\tilde{x}$ has been scaled in $[0, 1]$.

All calculations have been implemented in Matlab (version 7.8.0.347, R2009a). We used CPLEX 12.2 for solving the LPs during preprocessing via CPLEX's Matlab interface. In particular, this interface made it easy to warm-start the LPs as discussed above.

4.2 Experiments

For the computational study, we collected a total of 243 instances. All instances satisfied the assumptions of boundedness and interiority and had $H \not\succeq 0$. These instances are of one of four types:

Table 1 Statistics of the test instances

Type	# Instances	n	$m + m_{eq}$	H density
BoxQP	90	[20, 100]	[0, 0]	[0.19, 0.99]
Globallib	83	[2, 100]	[1, 52]	[0.01, 1]
CUTEr	6	[4, 12]	[0, 13]	[0.08, 1]
RandQP	64	[20, 50]	[14, 35]	[0.23, 1]

- **BoxQP**. We selected 90 instances having $m = m_{eq} = 0$ and $(l, u) = (0, e)$. These so-called BoxQP instances were created and solved in a recent pair of papers [7,25].
- **Globallib**. Globallib [10] instances are standard benchmark instances in global optimization. We included 83 instances that satisfy our assumptions for (QP).
- **CUTEr**. We included 6 instances from the CUTEr test problem set [11]. The CUTEr test problem set is a rich nonlinear programming problem test set. However, most QP instances in the set are convex, that is, they have $H \succeq 0$. We found only 9 instances that satisfy our selection criteria, i.e., $H \not\succeq 0$ and bounded feasible set with interior, but we excluded 3 either because of size or numerical difficulties encountered by each of the methods.
- **RandQP**. We generated 64 instances of QPs with varying size and sparsity using a code (with slight modifications) written by Sven Leyffer for generating random mixed-integer quadratic programming instances. All instances have both inequality and equality constraints. Each element in the data matrices is a random number in a certain range. The bounds for the variables are $(l, u) = (0, e)$.

We summarize the key statistics of these instances in Table 1, including the number of instances, the number of variables, the number of constraints, and the density of H. For the sake of brevity, we present only ranges.

From now on, we refer to our method as QUADPROGBB. We compare QUADPROGBB with the method of [6], referred to as BURVAN. The BURVAN code is not publicly available, but since the second author was a co-author of the BURVAN code, the code was available to us. BURVAN uses CPLEX 9.1, and while we tried to convert it to use CPLEX 12.2 as QUADPROGBB does, we were unsuccessful due to CPLEX run-time errors that we could not fix.

Recall that both methods are based on finite branching via the KKT system, but the key differences between QUADPROGBB and BURVAN are the structure of the relaxations to be solved at each node and the algorithm used to solve them. Specifically, the relaxations employed by BURVAN include the first-order KKT conditions but do not lift the dual variables into a semidefinite matrix as a means to strengthen the relaxation and explicitly model the complementarity conditions.[2] On the other hand, the SDP relaxation utilized in QUADPROGBB is derived from the completely positive representation of QP, which incorporates the dual variables and complementarity conditions in a DNN matrix. For solving the relaxations, BURVAN employs an algorithm that

[2] In fact, Burer and Vandenbussche [6] do investigate relaxations that lift the dual variables into a semidefinite matrix, but the authors conclude that the increased size of the relaxations causes overall larger B&B times. So we employ their simpler relaxation in order to have the best overall times for their method.

is designed to handle Lovász-Schrijver-style SDP relaxations of quadratic programs [5,15], while the algorithm we employ is specifically tailored for (DNP).

At the implementation level, another difference between QUADPROGBB and BURVAN is that BURVAN has been designed to only solve instances of (QP) with no equality constraints, that is, with $m_{eq} = 0$. Moreover, BURVAN requires an interior point just as we do, so one cannot simply split equalities into inequalities and solve the resulting instance with BURVAN. While certainly BURVAN could be adapted to solve instances with $m_{eq} > 0$, we do not make such modifications to their algorithm here and test it only on instances with $m_{eq} = 0$.

We also compare with Couenne 0.3 [1], a state-of-the-art open source global optimization solver.[3] Couenne differs from our methods in three major ways. First, Couenne is a general mixed-integer nonlinear solver and thus is not specialized for solving (QP). Second, Couenne utilizes a reformulation-based spatial B&B technique that implements linearization, branching, heuristics to find feasible solutions, and bound reduction [2]. It is well known that spatial B&B can produce an infinite number of nodes; in comparison, our approach is a finite B&B method. Third, Couenne solves LP relaxations at the nodes, where we solve SDP relaxations. In particular, the computational cost at each node is relatively cheap for Couenne compared with QUADPROGBB. The LP solver used by Couenne is Clp.[4]

All instances with $m_{eq} = 0$ are solved three times, once by each of the above methods. Since BURVAN cannot handle $m_{eq} > 0$, those instances with $m_{eq} > 0$ are solved by QUADPROGBB and Couenne only. Both QUADPROGBB and BURVAN use a relative tolerance of 10^{-6}. We also set Couenne's *allowable_fraction_gap* option to be 10^{-6}, which serves a similar purpose as the relative tolerance in QUADPROGBB. All computations were performed on a Pentium D running at 3.2 GHz under the Linux operating system. Also a time limit of 10 h and memory limit of 1 gigabyte were enforced for each method on each instance.

To compare QUADPROGBB and BURVAN, we present log-log plots of the CPU times in Fig. 1, where the left plot shows Globallib and CUTEr results and the right shows BoxQP results. Each square represents one instance, and its xy-coordinates represent the CPU times of QUADPROGBB and BURVAN, respectively. The diagonal $y = x$ line is also plotted for reference. If a square is located above the diagonal line, QUADPROGBB solves that instance faster. A similar comparison is made between QUADPROGBB and Couenne in Fig. 2, where all four types of instances are solved. We group the results of Globallib, RandQP, and CUTEr instances together and plot them in Fig. 2a while separating the BoxQP results in Fig. 2b.

Figures 1 and 2 depict only instances that were solved by both methods, that is, instances on which neither method ran out of time or memory, and also were nontrivial. Here we regard instances as "nontrivial" if they require more than 10 s to solve by at least one of the three methods. Table 2 summarizes those instances that ran out of time or memory. When a particular method ran out of resources on an instance, we calculated the relative optimality gap at the moment of termination, where the gap is

[3] Specifically, we used Couenne stable version 0.3, revision:738, 03/07/2011. https://projects.coin-or.org/Couenne, accessed 08/10/2011.

[4] Clp—Coin-Or Linear Programming. http://projects.coin-or.org/Clp/, accessed 08/10/2011.

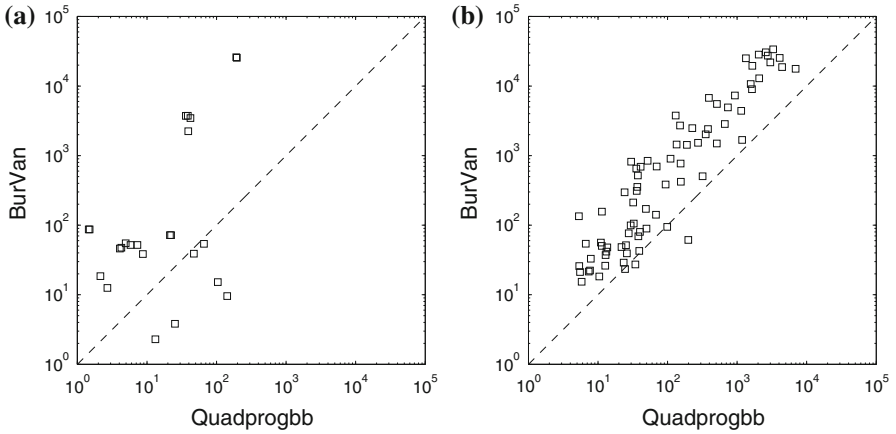

Fig. 1 CPU times—QUADPROGBB versus BURVAN. (**a**) Globallib and CUTEr, (**b**) BoxQP

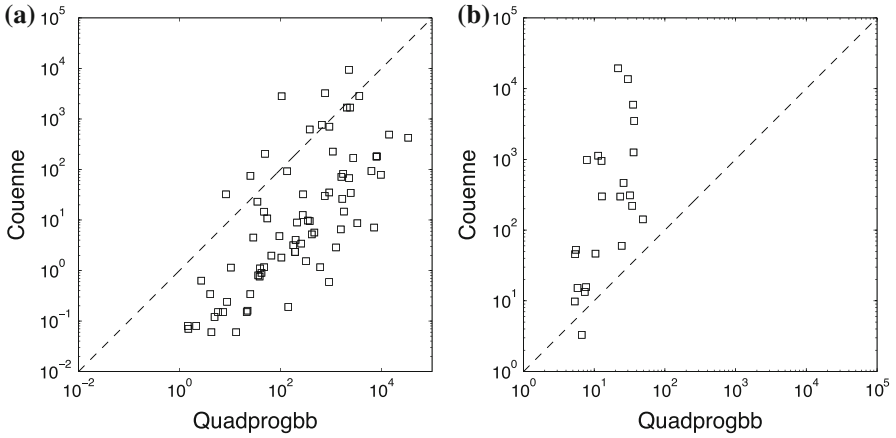

Fig. 2 CPU times—QUADPROGBB versus Couenne. (**a**) Globallib, RandQP, and CUTEr, (**b**) BoxQP

defined as

$$\frac{\text{GUB} - z_{lb}}{\max\{1, |\text{GUB}|\}} \times 100\% \tag{20}$$

GUB denotes the current global upper bound, and z_{lb} represents the worst bound of all the remaining active nodes. In the table, we report a method's average relative optimality gap on a set of nonsolved instances. We mention also that BURVAN failed numerically on one Globallib instance.

We summarize our interpretations of Figs. 1 and 2 and Table 2 as follows:

- On the Globallib and CUTEr instances, QUADPROGBB performed better than BURVAN. In the left plot of Fig. 1, more than two thirds of the instances are above the diagonal line. Several squares are located far above the diagonal line,

Table 2 Comparison of the three methods on failed instances in terms of average optimality gap

	Type	# Of instances	# Out of time	# Out of memory	Avg. Rel. Opt. gap when out of resources (%)
QUADPROGBB	BoxQP	90	5	–	0.18
	Globallib	83	1	–	0.30
	RandQP	64	3	–	9.81
BURVAN	BoxQP	90	12	–	0.87
	Globallib	69	2	–	0.17
Couenne	BoxQP	90	–	66	60.96
	Globallib	83	–	3	21.64
	RandQP	64	–	16	35.72

(The per instance time limit is 10 h, and the memory limit is 1 GB)

indicating that QUADPROGBB performed better than BURVAN on harder instances. From Table 2, QUADPROGBB and BURVAN seem to have similar average optimality gaps on their failed Globallib instances.

- QUADPROGBB outperforms BURVAN on the BoxQP instances. This performance is clear among instances that were solved by both methods (see Fig. 1b) as well as on the failed instances (see Table 2), where QUADPROGBB ran out of resources less often and, even when it did, had a better average optimality gap.
- For most Globallib, RandQP and CUTEr instances, Couenne outperforms QUADPROGBB as shown in Fig. 2a. We mention, however, that QUADPROGBB performs better than Couenne in closing the gaps on the failed instances. In particular, on the Globallib instances, QUADPROGBB only ran out of time on 1 instance (*qp3*), and the gap is 0.30%; the average gap is 21.64% for Couenne; on the RandQP instances, the average gap is 9.81% for QUADPROGBB, while it is 35.72% for Couenne.
- For BoxQP instances, QUADPROGBB significantly outperforms Couenne on instances solved by both methods (see Fig. 2b), and Couenne runs out of memory on the larger instances leaving large optimality gaps (see Table 2). The high memory usage is due to the generation and storage of a very large B&B tree. We attribute the performance differences mainly to the relaxation differences: we use stronger SDP-based relaxations, while Couenne uses weaker LP relaxations.

To compare all three methods and take into account failed instances, we also present their performance profiles in Fig. 3. A performance profile "for a solver is the (cumulative) distribution function for…the ratio of the computing time of the solver versus the best time of all of the solvers" over all instances [9]. Again we separate the instances into BoxQP instances, Fig. 3b, and all other instances, Fig. 3a. Since BURVAN does not apply to RandQP instances, we do not include BURVAN in Fig. 3a. We also exclude from the performance profiles the trivial instances, those instances that can be solved within 10 s by all the comparing methods.

The performance profiles confirm the information presented in Figs. 1, 2 and Table 2. For Globallib, RandQP and CUTEr instances, Couenne is faster than QUADPROGBB on many of them but could not solve as many instances as QUADPROGBB. On BoxQP

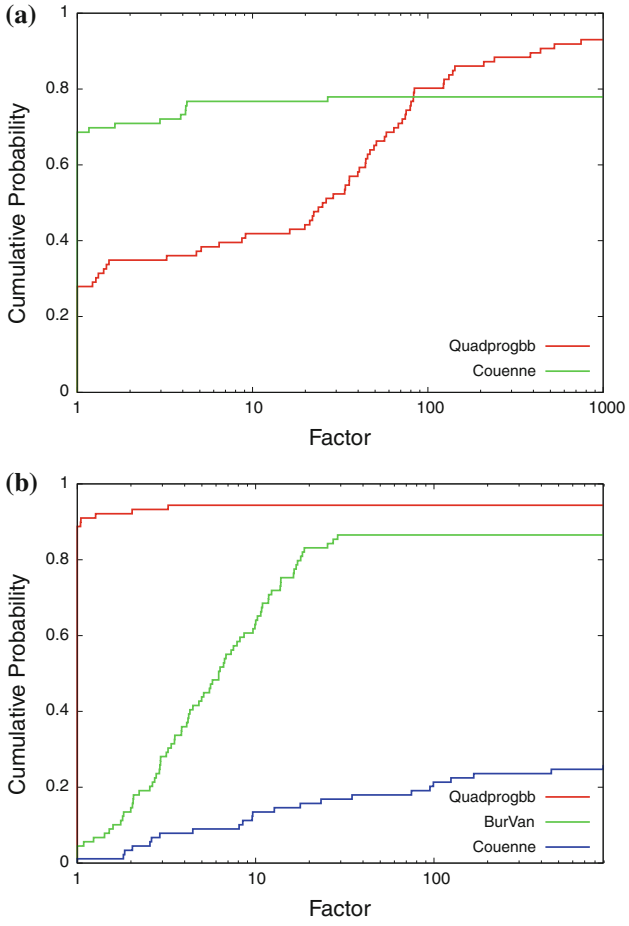

Fig. 3 Performance profiles. (**a**) Globallib, RandQP, and CUTEr, (**b**) BoxQP

instances, QUADPROGBB is clearly the winner among all the three methods: not only it is the fastest, but also it solved most of the problems.

The full details of all runs summarized in this section have been posted as an online appendix at http://dollar.biz.uiowa.edu/~sburer under the keyword `QuadprogBB`.

5 Extension

We have implemented QUADPROGBB in Matlab, and as such, it is convenient to use QUADPROGBB directly or as a subroutine to solve more complex problems. In this subsection, we provide an example to illustrate this approach.

Consider a problem of the form

$$\min_x \left\{ \frac{1}{2} x^T Q x + c^T x \ : \ \begin{array}{c} l \le x \le u \\ \|x\|_2 \le \Delta \end{array} \right\}, \tag{21}$$

which arises, for example, as a subproblem when applying a trust-region method to minimize a function $f(y)$ subject to bound constraints: $\min\{f(y) : \tilde{l} \leq y \leq \tilde{u}\}$; see [19] for an introduction to trust-region methods. The quadratic objective in (21) can be viewed as an approximation of $f(y)$ in the trust-region specified by $\|y - \bar{y}\| \leq \Delta$ around a given point $\bar{y}$ by defining $x := y - \bar{y}$, and the bounds $l \leq x \leq u$ reflect the bounds of the feasible region. Problem (21) would likely be solved approximately in trust-region implementations. Nevertheless, here we are interested in solving it globally.

Using standard Lagrangian duality, one can show that the optimal value of (21) is equal to the optimal value of

$$\max_{\lambda \geq 0} L(\lambda), \quad L(\lambda) := \min \left\{ \frac{1}{2} x^T Q x + c^T x + \lambda(x^T x - 1) \ : \ l \leq x \leq u \right\}, \quad (22)$$

provided that a primal-dual solution (λ^*, x^*) to (22) satisfies $\lambda^*(1 - (x^*)^T x^*) = 0$. If not, then $\lambda^*(1 - (x^*)^T x^*)$ is the associated duality gap. Note that for any given λ, one can evaluate $L(\lambda)$ by solving a box-constrained QP. In addition, $L(\lambda)$ is a concave function in λ. These facts allow us to solve (22) via bisection, for example.

We compare two ways to (approximately) solve (21): (1) with the bisection method for (22) mentioned above, where QUADPROGBB is used to evaluate $L(\lambda)$, and (2) directly using Couenne. We tested the two methods on 33 randomly generated instances. In particular, we use the 33 smallest BoxQP instances ($20 \leq n \leq 40$) for the objective data (Q, c). Additionally, we fix $\Delta = 1$ and generate each element of l uniformly between $[-1, 0]$ and each element of u uniformly between $[0, 1]$.

Couenne could not solve any of these instances to within an acceptable tolerance within 1 h, and Couenne's average relative optimality gap when out of resources was 1068%. Our method, on the other hand, solved 31 of the 33 instances within an acceptable tolerance (the average relative duality gap was 0.026%) within 1 h. The average time for those 31 instances was 233 s.

As evidenced by Couenne's performance, these random instances of (21) are not "easy," and the main goal of this experiment has been to demonstrate that it is possible to solve more general problems using QUADPROGBB in a convenient manner.

6 Conclusion

In this paper, we have shown how to globally solve the nonconvex quadratic programming problem (QP) by combining ideas from finite KKT-based branching and completely positive (or copositive) programming. The resultant algorithm often outperforms existing methods such as Couenne and the one by Burer and Vandenbussche [6]. In addition, the algorithm can be conveniently incorporated in other global optimization approaches.

The algorithm can be improved in several potential ways. For example, we could speed up the algorithm by including a more sophisticated strategy of choosing branching variable and improving the subroutine, which solves the DNN relaxations at each node (and which we have treated as a black box in this paper).

Acknowledgments We thank Stefan Wild for helpful discussions, which led to the variant of the trust-region subproblem solved in Sect. 5. We also thank the three referees for many helpful suggestions that improved the paper significantly.

References

1. Belotti, P.: Couenne: a user's manual. Technical report, Clemson University. http://projects.coin-or.org/Couenne/browser/trunk/Couenne/doc/couenne-user-manual.pdf (2010)
2. Belotti, P., Lee, J., Liberti, L., Margot, F., Wächter, A.: Branching and bounds tightening techniques for non-convex MINLP. Optim. Methods Softw. **24**(4–5), 597–634 (2009). http://www.optimization-online.org/DB_HTML/2008/08/2059.html
3. Burer, S.: On the copositive representation of binary and continuous nonconvex quadratic programs. Math. Program. **120**(2), 479–495 (2009)
4. Burer, S.: Optimizing a polyhedral-semidefinite relaxation of completely positive programs. Math. Prog. Comp. **2**(1), 1–19 (2010)
5. Burer, S., Vandenbussche, D.: Solving lift-and-project relaxations of binary integer programs. SIAM J. Optim. **16**(3), 726–750 (2006)
6. Burer, S., Vandenbussche, D.: A finite branch-and-bound algorithm for nonconvex quadratic programming via semidefinite relaxations. Math. Program. **113**(2, Ser. A), 259–282 (2008)
7. Burer, S., Vandenbussche, D.: Globally solving box-constrained nonconvex quadratic programs with semidefinite-based finite branch-and-bound. Comput. Optim. Appl. **43**(2), 181–195 (2009)
8. Celis, M.R., Dennis, J.E., Tapia, R.A.: A trust region strategy for nonlinear equality constrained optimization. In: Numerical Optimization, 1984 (Boulder, Colo., 1984), pp. 71–82. SIAM, Philadelphia (1985)
9. Dolan, E.D., Moré, J.J.: Benchmarking optimization software with performance profiles. Math. Program. **91**(2, Ser. A), 201–213 (2002)
10. Globallib: Gamsworld global optimization library. http://www.gamsworld.org/global/globallib.htm
11. Gould, N., Orban, D., Toint, P.L.: CUTEr (and SifDec): a constrained and unconstrained testing environment, revisited. ACM Trans. Math. Softw. **29**(4), 373–394 (2003)
12. Gould, N.I.M., Toint, P.L.: Numerical methods for large-scale non-convex quadratic programming. In: Trends in Industrial and Applied Mathematics (Amritsar, 2001). Appl. Optim., vol. 72, pp. 149–179. Kluwer, Dordrecht (2002)
13. Lin, Y., Cryer, C.W.: An alternating direction implicit algorithm for the solution of linear complementarity problems arising from free boundary problems. Appl. Math. Optim. **13**(1), 1–17 (1985)
14. Lootsma, F.A., Pearson, J.D.: An indefinite-quadratic-programming model for a continuous-production problem. Philips Res. Rep. **25**, 244–254 (1970)
15. Lovász, L., Schrijver, A.: Cones of matrices and set-functions and 0–1 optimization. SIAM J. Optim. **1**, 166–190 (1991)
16. MathWorks: MATLAB Reference Guide. The MathWorks Inc., Natick (2010)
17. Moré, J.J., Toraldo, G.: Algorithms for bound constrained quadratic programming problems. Numer. Math. **55**(4), 377–400 (1989)
18. Nguyen, T., Welsch, R.: Outlier detection and least trimmed squares approximation using semi-definite programming. Comput. Stat. Data Anal. **54**, 3212–3226 (2010)
19. Nocedal, J., Wright, S.: Numerical Optimization. Springer, New York (1999)
20. Pardalos, P.: Global optimization algorithms for linearly constrained indefinite quadratic problems. Comput. Math. Appl. **21**, 87–97 (1991)
21. Pardalos, P.M., Vavasis, S.A.: Quadratic programming with one negative eigenvalue is NP-hard. J. Global Optim. **1**(1), 15–22 (1991)
22. Sahinidis, N.V.: BARON: a general purpose global optimization software package. J. Global Optim. **8**(2), 201–205 (1996)

23. Sherali, H.D., Adams, W.P.: A Reformulation-Linearization Technique for Solving Discrete and Continuous Nonconvex Problems. Nonconvex Optimization and its Applications, vol. 31. Kluwer, Dordrecht (1999)
24. Skutella, M.: Convex quadratic and semidefinite programming relaxations in scheduling. J. ACM **48**(2), 206–242 (2001)
25. Vandenbussche, D., Nemhauser, G.: A branch-and-cut algorithm for nonconvex quadratic programs with box constraints. Math. Program. **102**(3), 559–575 (2005)