Math. Program., Ser. A (2008) 113:259–282
DOI 10.1007/s10107-006-0080-6

FULL LENGTH PAPER

A finite branch-and-bound algorithm for nonconvex quadratic programming via semidefinite relaxations

Samuel Burer · Dieter Vandenbussche

Received: 27 June 2005 / Accepted: 30 October 2006 / Published online: 12 December 2006

Abstract Existing global optimization techniques for nonconvex quadratic programming (QP) branch by recursively partitioning the convex feasible set and thus generate an infinite number of branch-and-bound nodes. An open question of theoretical interest is how to develop a finite branch-and-bound algorithm for nonconvex QP. One idea, which guarantees a finite number of branching decisions, is to enforce the first-order Karush-Kuhn-Tucker (KKT) conditions through branching. In addition, such an approach naturally yields linear programming (LP) relaxations at each node. However, the LP relaxations are unbounded, a fact that precludes their use. In this paper, we propose and study semidefinite programming relaxations, which are bounded and hence suitable for use with finite KKT-branching. Computational results demonstrate the practical effectiveness of the method, with a particular highlight being that only a small number of nodes are required.

Keywords Nonconcave quadratic maximization · Nonconvex quadratic programming · Branch-and-bound · Lift-and-project relaxations · Semidefinite programming

This author was supported in part by NSF Grants CCR-0203426 and CCF-0545514.

S. Burer (✉)
Department of Management Sciences,
University of Iowa, Iowa City, IA 52242-1000, USA
e-mail: samuel-burer@uiowa.edu

D. Vandenbussche
Axioma, Inc., Marietta, GA 30068, USA

1 Introduction

This paper studies the problem of maximizing a quadratic objective subject to linear constraints:

$$\begin{array}{ll} \max & \frac{1}{2} x^T Q x + c^T x \\ \text{s.t.} & Ax \leq b \\ & x \geq 0 \end{array} \tag{QP}$$

where $x \in \mathbb{R}^n$ is the variable and $(Q, A, b, c) \in \mathbb{R}^{n \times n} \times \mathbb{R}^{m \times n} \times \mathbb{R}^m \times \mathbb{R}^n$ are the data. We will refer to the feasible set of (QP) as

$$P := \{x \in \mathbb{R}^n : Ax \leq b, x \geq 0\}.$$

Any quadratic program (or "QP") over linear constraints can be put in the above standard form, and without loss of generality, we assume Q is symmetric. If Q is negative semidefinite, then (QP) is solvable in polynomial time [14]. Here, however, we consider that Q is indefinite or positive semidefinite, in which case (QP) is an $\mathcal{NP}$-hard problem [19]. Nevertheless, our goal is to obtain a global maximizer.

The difficulty in optimizing (QP) is that it harbors many local maxima. Much of the literature on nonconvex quadratic programming has focused on obtaining first-order or (weak) second-order Karush–Kuhn–Tucker (KKT) critical points, which have a good objective value, using nonlinear programming techniques such as active-set or interior-point methods. The survey paper by Gould and Toint [9] provides information on such methods. On the other hand, a variety of global optimization techniques can solve (QP) exactly. For example, Sherali and Tuncbilek [23] developed a Reformulation–Linearization-Technique (RLT) for solving nonconvex QPs. For a review of several global optimization methods for nonconvex QP, see the work by Pardalos [18]. Floudas and Visweswaran [5] also present a general survey, including local and global optimization methods, various applications, and special cases. Off-the-shelf general–purpose global optimization software, such as BARON [20], is also available for solving (QP).

Existing global optimization techniques for (QP) work by recursively partitioning the polyhedron P. In this way, each node of the branch-and-bound tree essentially considers the quadratic optimization restricted to just a portion of P. To calculate a bound at each node, convex relaxations (typically linear programs) are employed. This basic framework allows for various branching and bounding strategies, but due to the convex nature of P, it may theoretically be necessary to subdivide P infinitely (i.e., to generate an infinite branch-and-bound tree) in order to verify global optimality in exact arithmetic. The following statement from Sherali and Tuncbilek [22] provides an illustration of a typical convergence result for existing global optimization approaches for (QP): "The algorithm either terminates finitely with an optimal solution, or

else an infinite sequence of nodes is generated such that, along any infinite branch of the branch-and-bound tree, any accumulation point of the sequence of solutions of the convex relaxations is optimal." We are unaware of any methods that can verify optimality in a finite number of nodes (using exact arithmetic).

It is important to point out that these same methods can verify "ε-optimality" in a finite number of nodes, though an a priori bound on the number of nodes required is still unknown. Also, in practice, concerns of an infinite tree are quite irrelevant due to the inexact nature of floating-point arithmetic.

For the box-constrained case of (QP) when $(A, b) = (I, e)$ and e is the all-ones vector, there has been a considerable amount of effort towards solving (QP) exactly. In fact, it is possible in this case to develop a finite branch-and-bound algorithm, as has been shown in the recent work of Vandenbussche and Nemhauser [25,26]. The key idea in their approach—developed from ideas of Hansen et al. [10]—is to explicitly account for the Lagrange multipliers in addition to the original x. This approach allows for logical branching on the first-order KKT conditions, which in turn enables the implicit enumeration and comparison of all first-order KKT points to obtain a global maximum. The natural convex relaxations at each node are linear programs (or "LPs"), which are actually unbounded. However, by exploiting the specific structure of the box-constrained case, it is possible to bound the LPs easily, completing the specification of a full, finite branch-and-bound algorithm for this case. Vandenbussche and Nemhauser further enhanced this approach by incorporating cuts for the convex hull of first-order KKT points to develop a highly efficient branch-and-cut implementation.

It is straightforward to extend the finite KKT-branching for the box-constrained case to (QP) generally. We use this as the basic idea for finite branching in this paper. In doing so, however, one still encounters LP relaxations that are unbounded. Unfortunately, there is no obvious way to bound the relaxations as Vandenbussche and Nemhauser have done in the box-constrained case. Thus, we will propose and investigate the alternative use of semidefinite programming (or "SDP") relaxations.

SDP relaxations of quadratically constrained quadratic programs (QCQPs), of which (QP) is a special case, have been studied by a number of researchers. One of the first efforts in this direction was made by Lovász and Schrijver [16], who developed a hierarchy of SDP relaxations for 0–1 linear integer programs, where one can think of the constraint $x_i \in \{0, 1\}$ as the quadratic equality $x_i^2 = x_i$. Later, approximation guarantees using SDP relaxations were obtained for specific problems, the premiere example being the Goemans and Williamson [8] analysis of the maximum cut problem on graphs; see also [17,27]. In particular, Ye [27] establishes a 4/7-approximation guarantee for the box-constrained case of (QP). Recent work by Bomze and de Klerk [3] has studied strong SDP relaxations of nonconvex QP over the simplex. Sherali and Fraticelli [21] discuss the use of semidefinite cuts to improve RLT relaxations of QPs, and Anstreicher [2] gives insight into how including semidefiniteness improves RLT relaxations. Probably one of the most general approaches for relaxing

QCQPs has been proposed by Kojima and Tunçel [12], who provide an extension of the Lovász–Schrijver approach to any optimization problem that can be expressed by a quadratic objective and (a possibly infinite number of) quadratic constraints. The result is a recursively defined sequence of convex relaxations that can provide a global optimum in a finite number of applications. In a follow-up paper, Kojima and Tunçel [13] also develop several different SDP relaxations for linear complementarity problems, a special class of QCQPs. To our knowledge, no one has studied in detail the specific SDP relaxations that we consider in this paper.

A first goal of this paper is to demonstrate how SDP relaxations, when employed in branch-and-bound with finite KKT-branching, do not suffer the same drawbacks as LP relaxations and consequently yield the first finite branch-and-bound algorithm for globally solving (QP). We begin in Sect. 2 by considering the so-called "KKT" reformulation of (QP). This leads naturally to the idea of finite branching, which we explain in detail. We also discuss the associated LP relaxations and their unboundedness. Then, in Sect. 3, we construct SDP relaxations and show how they address the issue of unboundedness. We also establish the finite branch-and-bound result. An important aspect is establishing the correctness of the branch-and-bound algorithm, which turns out to be a non-trivial task. We continue in Sect. 4 by discussing ways of strengthening the relaxations. Together with the results of Sect. 3, this paper presents two different SDP relaxations of the first-order KKT conditions of (QP). The relaxations are based heavily on ideas from the work of Lovász and Schrijver [16] mentioned above, but we incorporate important elements that reflect and exploit the specific structure of the problem at hand.

A second goal of this paper is to investigate the practical aspects of our method. In Sect. 5, we describe how the SDP relaxations are incorporated into an implementation of the branch-and-bound algorithm. We present detailed computational results, which demonstrate the effectiveness of our branch-and-bound algorithm on a number of different problem classes. One highlight is that the algorithm consistently requires only a small number of nodes in the branch-and-bound tree.

We remark that the current study has been motivated to a large extent by a recent work of the authors (Burer and Vandenbussche [4]), which provides efficient computational techniques for solving the Lovász–Schrijver SDP relaxations of 0–1 integer programs. Such SDPs—including the related ones introduced in this paper—are too large to be solved with conventional SDP algorithms, and so the ability to solve these types of SDPs is an indispensable component of this paper.

Finally, we would like to provide caution that the global optimization techniques described in this paper are not the best choice if one wishes only to compute a good, but not necessarily global, solution of (QP). For such a purpose, quicker, more scalable methods exist—such as the approximation algorithm of Ye [27] or the nonlinear optimization techniques of Absil and Tits [1].

1.1 Terminology and notation

In this section, we introduce some of the notation that will be used throughout the paper. $\mathbb{R}^n$ refers to n-dimensional Euclidean space. The norm of a vector $x \in \mathbb{R}^n$ is denoted by $\|x\| := \sqrt{x^T x}$. We let $e_i \in \mathbb{R}^n$ represent the i-th unit vector. $\mathbb{R}^{n\times n}$ is the set of real, $n \times n$ matrices; $\mathcal{S}^n$ is the set of symmetric matrices in $\mathbb{R}^{n\times n}$; and $\mathcal{S}^n_+$ is the set of positive semidefinite symmetric matrices. The special notations $\mathbb{R}^{1+n}$ and $\mathcal{S}^{1+n}$ are used to denote the spaces $\mathbb{R}^n$ and $\mathcal{S}^n$ with an additional "0-th" entry prefixed or an additional 0-th row and 0-th column prefixed, respectively. Given a matrix $X \in \mathbb{R}^{n\times n}$, we denote by $X_{\cdot j}$ and $X_{i\cdot}$ the j-th column and i-th row of X, respectively. The inner product of two matrices $A, B \in \mathbb{R}^{n\times n}$ is defined as $A \bullet B := \text{trace}(A^T B)$, where trace$(\cdot)$ denotes the sum of the diagonal entries of a matrix. Given two vectors $x, z \in \mathbb{R}^n$, we denote their Hadamard product by $x \circ z \in \mathbb{R}^n$, where $[x \circ z]_j = x_j z_j$. Finally, diag(A) is defined as the vector with the diagonal of the matrix A as its entries.

2 Finite branching and unbounded multipliers

In this section, we review a general method for reformulating (QP) as an LP with complementarity constraints (see [6]) and detail how this reformulation leads to a finite-branching scheme to solve (QP). We then discuss the unboundedness of the natural LP relaxations in this scheme, and Sect. 3 will discuss our approach for handling the unboundedness.

2.1 The KKT reformulation and finite branching

By introducing nonnegative multipliers y and z for the constraints $Ax \le b$ and $x \ge 0$ in P, respectively, any locally optimal solution x of (QP) must have the property that the sets

$$G_x := \left\{(y, z) \ge 0 : A^T y - z = Qx + c\right\}$$
$$C_x := \{(y, z) \ge 0 : (b - Ax) \circ y = 0,\ \ x \circ z = 0\}$$

satisfy $G_x \cap C_x \neq \emptyset$. In words, G_x is the set of multipliers where the gradient of the Lagrangian vanishes, and C_x consists of those multipliers satisfying complementarity with x. The condition $G_x \cap C_x \neq \emptyset$ specifies that x is a first-order KKT point.

One can show the following property of KKT points.

Proposition 2.1 (Giannessi and Tomasin [6]) *Suppose $x \in P$ and $(y, z) \in G_x \cap C_x$. Then $x^T Qx + c^T x = b^T y$.*

Proof We first note that $(y, z) \in C_x$ implies $(Ax)^T y = b^T y$ and $x^T z = 0$. Next, pre-multiplying the equality $A^T y - z = Qx + c$ by x^T yields $x^T Qx + c^T x = (Ax)^T y - x^T z = b^T y$, as desired.

This shows that (QP) may be reformulated as the following linear program with complementarity constraints:

$$\begin{array}{ll} \max & \frac{1}{2} b^T y + \frac{1}{2} c^T x \\ \text{s.t.} & x \in P \quad (y,z) \in G_x \cap C_x. \end{array} \tag{KKT}$$

By dropping $(y,z) \in C_x$, we obtain a natural LP relaxation:

$$\begin{array}{ll} \max & \frac{1}{2} b^T y + \frac{1}{2} c^T x \\ \text{s.t.} & x \in P \quad (y,z) \in G_x. \end{array} \tag{RKKT}$$

The reformulation (KKT) suggests a finite branch-and-bound approach, where complementarity is recursively enforced using linear equations. A particular node of the tree is specified by four index sets $F^x, F^z \subseteq \{1,\dots,n\}$ and $F^{b-Ax}, F^y \subseteq \{1,\dots,m\}$, which satisfy $F^x \cap F^z = \emptyset$ and $F^{b-Ax} \cap F^y = \emptyset$. These index sets correspond to complementarities that are enforced via linear equations in the following restriction of (KKT):

$$\begin{array}{ll} \max & \frac{1}{2} b^T y + \frac{1}{2} c^T x \\ \text{s.t.} & x \in P, \quad (y,z) \in G_x \cap C_x \\ & x_j = 0, \quad j \in F^x \\ & z_j = 0, \quad j \in F^z \\ & A_{i\cdot} x = b_i, \quad i \in F^{b-Ax} \\ & y_i = 0, \quad i \in F^y. \end{array} \tag{1}$$

At a given node, the branch-and-bound algorithm will solve a convex relaxation of (1). Due to the presence of the linear equations, one can relax the nonconvex quadratic constraint $(y,z) \in C_x$ and yet still maintain (partial) complementarity via the linear equations: we have $x_j z_j = 0$ for all $j \in F^x \cup F^z$, and $(b_i - A_{i\cdot} x) y_i = 0$ for all $i \in F^{b-Ax} \cup F^y$.

Branching on a node involves selecting some $j \in \{1,\dots,n\} \setminus (F^x \cup F^z)$ or some $i \in \{1,\dots,m\} \setminus (F^{b-Ax} \cup F^y)$ and then creating two children, one with $j \in F^x$ and one with $j \in F^z$ (if a j was selected), or one with $i \in F^{b-Ax}$ and one with $i \in F^y$ (if an i was selected). We remark that the root node in the tree has all four index sets empty, and a leaf node is specified by sets satisfying $F^x \cup F^z = \{1,\dots,n\}$ and $F^{b-Ax} \cup F^y = \{1,\dots,m\}$.

2.2 The issue of unboundedness

Of course, any branch-and-bound approach depends heavily on the relaxations solved at each node. The most natural relaxation of (1) is the LP gotten by simply dropping $(y,z) \in C_x$. (Other relaxations may certainly be possible, e.g., the

SDP relaxations described in later sections.) However, there is a fundamental problem with using linear programing relaxations, namely that (RKKT), which is the relaxation at the root node, has an unbounded objective (under mild assumptions), as we detail in the next proposition and corollary.

Proposition 2.2 *If P is nonempty and bounded, then the set*

$$R := \left\{(\Delta y, \Delta z) \geq 0 : A^T \Delta y - \Delta z = 0\right\}$$

contains nonzero points. Moreover:

- $b^T \Delta y \geq 0$ *for all* $(\Delta y, \Delta z) \in R$*; and*
- *if P has an interior, then $b^T \Delta y > 0$ for all nonzero $(\Delta y, \Delta z) \in R$.*

Proof To prove both parts of the proposition, we consider the primal LP $\max \left\{d^T x : x \in P\right\}$ and its dual $\min \left\{b^T y : A^T y - z = d, (y, z) \geq 0\right\}$ for a specific choice of d.

Let $d = e$. Because P is nonempty and bounded, the dual has a feasible point (y, z). It follows that $(\Delta y, \Delta z) := (y, z + e)$ is a nontrivial element of R.

Next let $d = 0$, and note that the dual feasible set equals R, which immediately implies the second statement of the proposition.

To prove the third statement, keep $d = 0$ and let x^0 be an interior-point of P. Note that x^0 is optimal for the primal. Complementary slackness thus implies that $(0, 0)$ is the unique optimal solution of the dual, which proves the result.

Corollary 2.3 *If P is bounded with interior, then* (*RKKT*) *has unbounded objective value.*

Proof We first note that R defined in Proposition 2.2 is the recession cone of G_x (for arbitrary x). Hence, (RKKT) contains a nontrivial direction of recession $(\Delta y, \Delta z)$ in the variables (y, z) such that $b^T \Delta y > 0$, which proves that it has an unbounded objective value.

In the remainder of the paper, we make the following mild assumptions, which are in line with Corollary 2.3:

Assumption 2.4 *P is bounded. In particular, P is contained in the unit box* $\{x \in \mathbb{R}^n : 0 \leq x \leq e\}$.

Assumption 2.5 *P contains an interior point, i.e., there exists $x \in P$ such that $Ax < b$ and $x > 0$.*

Note that, if P is bounded but not inside the unit box, then Assumption 2.4 can be enforced by a simple variable scaling. Likewise, if P does not satisfy Assumption 2.5, then dimension-reduction techniques can be used to transform P into an equivalent polyhedron, which has an interior.

We remark that, although Corollary 2.3 only establishes the unboundedness of the root-node LP relaxation (RKKT), it easy to see that many of

the nodes in the branch-and-bound tree will have unbounded LP relaxations, again stemming from the unboundedness of (y, z). One cannot expect the LP relaxations to be bounded until many complementarities have been fixed by branching.

3 Finite branch-and-bound via SDP

We have discussed in Sect. 2 how bounding the Lagrange multipliers (y, z) becomes an issue when attempting to execute finite KKT-branching. We now show how this obstacle can be overcome by an application of semidefinite programming, and as a result, we establish the first finite branch-and-bound algorithm for (QP).

3.1 An SDP relaxation of the KKT formulation

We first introduce a basic SDP relaxation of (QP). Our motivation comes from the SDP relaxations of 0–1 integer programs introduced by Lovász and Schrijver [16]. Consider a matrix variable Y, which is related to $x \in P$ by the following quadratic equation:

$$Y = \begin{pmatrix} 1 \\ x \end{pmatrix} \begin{pmatrix} 1 \\ x \end{pmatrix}^T = \begin{pmatrix} 1 & x^T \\ x & xx^T \end{pmatrix}. \tag{2}$$

From (2), we observe the following:

- Y is symmetric and positive semidefinite, i.e., $Y \in \mathcal{S}^{1+n}_+$ (or simply $Y \succeq 0$);
- if we multiply the constraints $Ax \leq b$ of P by some x_i, we obtain the quadratic inequalities $b\,x_i - Ax\,x_i \geq 0$, which are valid for P; these inequalities can be written in terms of Y as

$$\big(b \big| -A\big)\, Ye_i \geq 0 \quad \forall i = 1, \ldots, n;$$

- the objective function of (QP) can be modeled in terms of Y as

$$\frac{1}{2} x^T Q x + c^T x = \frac{1}{2} \begin{pmatrix} 0 & c^T \\ c & Q \end{pmatrix} \bullet \begin{pmatrix} 1 & x^T \\ x & xx^T \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 0 & c^T \\ c & Q \end{pmatrix} \bullet Y.$$

For convenience, we define

$$K := \left\{ (x_0, x) \in \mathbb{R}^{1+n} \;:\; Ax \leq x_0\, b,\ (x_0, x) \geq 0 \right\}$$

and

$$\tilde{Q} := \frac{1}{2} \begin{pmatrix} 0 & c^T \\ c & Q \end{pmatrix},$$

which allow us to express the second and third properties above more simply as $Ye_i \in K$ and $x^T Qx/2 + c^T x = \tilde{Q} \bullet Y$. In addition, we let M_+ denote the set of all Y satisfying the first and second properties:

$$M_+ := \{Y \succeq 0 : Ye_i \in K \ \ \forall i = 1, \ldots, n\}.$$

Then we have the following equivalent formulation of (QP):

$$\begin{array}{ll} \max & \tilde{Q} \bullet Y \\ \text{s.t.} & Y = \begin{pmatrix} 1 & x^T \\ x & xx^T \end{pmatrix} \in M_+ \\ & x \in P. \end{array}$$

Finally, by dropping the last n columns of the equation (2), we arrive at the following linear SDP relaxation of (QP):

$$\begin{array}{lll} \max & \tilde{Q} \bullet Y & \text{(SDP}_0\text{)} \\ \text{s.t.} & Y \in M_+, \quad Ye_0 = (1; x) & \\ & x \in P. & \end{array}$$

We next combine this basic SDP relaxation with the LP relaxation (RKKT) introduced in Section 2 to obtain an SDP relaxation of (KKT). More specifically, we consider the following optimization over the combined variables (Y, x, y, z) in which the two objectives are equated using an additional constraint:

$$\begin{array}{lll} \max & \tilde{Q} \bullet Y & \text{(SDP}_1\text{)} \\ \text{s.t.} & Y \in M_+, \quad Ye_0 = (1; x) & \\ & x \in P, \quad (y, z) \in G_x & \\ & \tilde{Q} \bullet Y = (b^T y + c^T x)/2. & (3) \end{array}$$

This optimization problem is a valid relaxation of (KKT) if the constraint (3) is valid for all points feasible for (KKT), which indeed holds as follows: let (x, y, z) be such a point, and define Y according to (2); then (Y, x, y, z) satisfies the first four constraints of (SDP$_1$) by construction and the constraint (3) is satisfied due to Proposition 2.1 and (2).

3.2 Properties of the SDP relaxation

In this short subsection, we establish two key properties of the SDP relaxation (SDP$_1$)—ones which will allow us to establish the finite branch-and-bound procedure in the next subsection.

First, unlike the LP relaxation (RKKT), the SDP relaxation (SDP$_1$) is bounded.

Proposition 3.1 *The feasible set of* (SDP_1) *is bounded.*

Proof We first argue that the feasible set of (SDP_0) is bounded. By Assumption 2.4, the variable x is bounded, and hence Ye_0 is as well. Because Y is symmetric, this in turn implies that the 0-th row of Y is bounded. Now consider $Ye_i \in K$ for $i \geq 1$. The recession cone corresponding to the constraint $Ye_i \in K$ is

$$\left\{(r_0, r) \in \mathbb{R}^{1+n}_+ \mid Ar - br_0 \leq 0\right\} = \left\{(0, r) \in \mathbb{R}^{1+n}_+ \mid Ar \leq 0\right\},$$

where the equality follows from the fact that the 0-th component of Ye_i is bounded. Note that for any $(0, r)$ in the cone, we know r is in the recession cone of P, which is $\{0\}$ by Assumption 2.4. Hence, we may conclude that Ye_i is bounded.

We now show that the recession cone of the feasible set of (SDP_1) is trivial. Using the definition of R in Proposition 2.2 as well as the result of the previous paragraph, the recession cone is

$$\left\{(\Delta Y, \Delta x, \Delta y, \Delta z) : (\Delta Y, \Delta x) = (0, 0),\ (\Delta y, \Delta z) \in R,\ b^T \Delta y = 0\right\}.$$

However, Assumption 2.5 and Proposition 2.2 imply $b^T \Delta y > 0$ for all nontrivial $(\Delta y, \Delta z) \in R$. So the recession cone is trivial.

Second, we prove another important property of (SDP_1), namely that if its optimal solution yields a first-order KKT point $(\bar{x}, \bar{y}, \bar{z})$, then $\bar{x}$ is optimal for (QP).

Proposition 3.2 *Suppose* $(\bar{x}, \bar{y}, \bar{z})$ *is part of an optimal solution for* (SDP_1). *If* $(\bar{y}, \bar{z}) \in C_{\bar{x}}$, *then* $\bar{x}$ *is an optimal solution of* (QP).

Proof Let $(\bar{Y}, \bar{x}, \bar{y}, \bar{z})$ be an optimal solution of (SDP_1). In particular, (3) holds, i.e.,

$$\tilde{Q} \bullet \bar{Y} = \frac{1}{2} b^T \bar{y} + \frac{1}{2} c^T \bar{x}.$$

Because $(\bar{y}, \bar{z}) \in G_{\bar{x}} \cap C_{\bar{x}}$, Proposition 2.1 implies $(b^T \bar{y} + c^T \bar{x})/2 = \bar{x}^T Q \bar{x}/2 + c^T \bar{x}$. So $\tilde{Q} \bullet \bar{Y}$ equals the function value at $\bar{x}$. Since $\tilde{Q} \bullet \bar{Y}$ is an upper bound on the optimal value of (QP), it follows that $\bar{x}$ is an optimal solution.

3.3 Finite branch-and-bound

In Sect. 2.1, we have described the basic approach for constructing a branch-and-bound algorithm for (QP) by recursively enforcing complementarities through branching. We now incorporate (SDP_1) and establish the finite branch-and-bound algorithm.

We first point out that, although the SDP relaxation (SDP_1) has been constructed as a relaxation of (KKT), it is easy to extend the same ideas to yield an SDP relaxation of the restricted KKT subproblem (1) associated with each node. The only modification is the inclusion of the linear equalities represented by F^x, F^z, F^{b-Ax}, and F^y into the definitions of P, G_x, and K in the natural way. For example, at any node, we define

$$K := \left\{ (x_0, x) \in \mathbb{R}^{1+n} \; : \; \begin{array}{ll} Ax \leq x_0\, b, & (x_0, x) \geq 0 \\ A_i.x = x_0\, b_i & \forall\, i \in F^{b-Ax} \\ x_j = 0 & \forall\, j \in F^x \end{array} \right\}.$$

Moreover, the properties outlined in Propositions 3.1 and 3.2 also hold for the SDP relaxations at the nodes: (a) each relaxation is bounded; and (b) if the solution of a particular relaxation yields a KKT point $(\bar{x}, \bar{y}, \bar{z})$, then $\bar{x}$ is an optimal solution of (1). We also point out a simple observation: (c) if the relaxation is infeasible, then (1) is infeasible.

Overall, the branch-and-bound algorithm starts at the root node, and begins evaluating nodes, adding or removing nodes from the tree at each stage. Evaluating a node involves solving the SDP relaxation and then fathoming the node (if possible) or branching on the node (if fathoming is not possible and if the node is not a leaf). The algorithm finishes when all nodes generated by the algorithm have been fathomed. In order to prove that our algorithm does indeed finish, it is necessary to establish that all leaf nodes will be fathomed. Otherwise, they cannot be eliminated from the tree since they cannot be branched on. This we term the *correctness* of the algorithm.

We explain fathoming in a bit more detail. Fathoming a node (i.e., eliminating it from further consideration) is only allowable if we can guarantee that its associated subproblem (1) contains no optimal solutions of (QP), other than possibly the solution $(\bar{x}, \bar{y}, \bar{z})$ obtained from the relaxation. Such a guarantee can be obtained in two ways. First, if the relaxation is infeasible, then (1) is infeasible so that it contains no optimal solutions. Second, we can fathom if the relaxed objective value at $(\bar{x}, \bar{y}, \bar{z})$ is equal to or less than the (QP) objective of some $x \in P$. In particular, we compare the relaxed value with the (QP) value of $\bar{x}$ itself as well as that of other solutions encountered at other nodes. Because the (QP) value of $\bar{x}$ cannot be greater than the relaxed value, fathoming occurs due to $\bar{x}$ only when the two values are equal, i.e., when the relaxation has no gap.

Accordingly, to have a correct branch-and-bound algorithm, it must be the case that (when feasible) the SDP relaxation has no gap at a leaf node. This is established by the property (ii) above because the solution $(\bar{x}, \bar{y}, \bar{z})$ at a leaf node is a KKT point by construction.

Thus, we have established the key theoretical result of the paper, which is stated carefully in the theorem and proof below. We stress that the theorem is concerned with the solution of (QP) in exact arithmetic.

Theorem 3.3 *The branch-and-bound approach for (QP), which employs KKT-branching and SDP relaxations of the type (SDP$_1$), is both correct and finite.*

Proof We prove the theorem under two different models of computation. The first proof, which assumes exact arithmetic, shows the essence of the result, while the second establishes the result under the more realistic bit model.

Under the assumption of exact arithmetic, we are able to obtain the exact solution of the SDP relaxations. Then the branch-and-bound approach is well-defined because the SDP relaxations are bounded (refer to property (i) above); finiteness is ensured by KKT branching; and correctness is guaranteed by the discussion above the theorem.

Under the bit model of computation, we are only able to obtain an approximate solution of the SDP relaxation (since the solution may have irrational entries, for example). However, it is easy to see that the SDP relaxation at a leaf node is equivalent to the natural LP relaxation of (1) in the sense that both types of relaxations yield the same solution $(\bar{x}, \bar{y}, \bar{z})$. Thus, at the leaf nodes, the SDPs can be solved exactly under the bit model because the LPs can be solved exactly. Combined with well-definition and finiteness as above, this establishes the correctness of algorithm.

4 SDP relaxations for quadratic programs

In this section, we investigate SDP relaxations of KKT further, a particular goal being techniques for strengthening the relaxation (SDP$_1$). As a result we propose a strengthened relaxation called (SDP$_2$).

In Sect. 3.1, we introduced SDP relaxations (SDP$_0$) and (SDP$_1$). Although (SDP$_0$) was not appropriate for use with KKT-branching because the multipliers (y, z) do not appear explicitly, a natural question arises: how much tighter is (SDP$_1$) than (SDP$_0$)? Although we have been unable to obtain a precise answer to this question, some simple comparisons of the two relaxations reveal interesting relationships, as we describe next.

For each $x \in P$ and any H_x such that $G_x \cap C_x \subseteq H_x \subseteq G_x$, define

$$\delta(x, H_x) := \frac{1}{2} \min \left\{ b^T y : (y, z) \in H_x \right\} + \frac{1}{2} c^T x.$$

We first point out that Proposition 2.1 implies $\delta(x, G_x \cap C_x) = (b^T y + c^T x)/2$ for all $(y, z) \in G_x \cap C_x$. Hence, one can actually reformulate (KKT) as

$$\max \{\delta(x, G_x \cap C_x) : x \in P\}.$$

Thus, for an approximation H_x of $G_x \cap C_x$, $\delta(x, H_x)$ can be interpreted as a conservative under-approximation of $\delta(x, G_x \cap C_x)$. Since relaxations of (KKT) depend at least in part on how closely $G_x \cap C_x$ is approximated by some H_x

(given either explicitly or implicitly), it turns out that $\delta(x, H_x)$ is a surrogate for understanding the tightness of H_x. Note also that $H_x \subseteq H'_x$ implies $\delta(x, H_x) \geq \delta(x, H'_x)$.

In particular with $H_x := G_x$, we can characterize those (Y, x), which are feasible for (SDP$_0$) but not for (SDP$_1$), i.e., ones with no $(y, z) \in G_x$ such that (Y, x, y, z) is feasible for (SDP$_1$). We say the point is *cut off* by (SDP$_1$).

Proposition 4.1 *Let (Y, x) be feasible for (SDP$_0$). Then (Y, x) is cut off by (SDP$_1$) if and only if $\delta(x, G_x) > \tilde{Q} \bullet Y$.*

Proof Suppose $\tilde{Q} \bullet Y < \delta(x, G_x)$, which implies $\tilde{Q} \bullet Y < (b^T y + c^T x)/2$ for all $(y, z) \in G_x$. Hence, (Y, x) cannot be feasible for (SDP$_1$) because it contains the constraint $\tilde{Q} \bullet Y = (b^T y + c^T x)/2$.

When $\tilde{Q} \bullet Y \geq \delta(x, G_x)$, we know that $\tilde{Q} \bullet Y \geq (b^T y + c^T x)/2$ for at least one $(y, z) \in G_x$, namely a solution defining $\delta(x, G_x)$. Using Proposition 2.2 along with Assumptions 2.4 and 2.5, we can adjust (y, z) so that $b^T y$ increases to satisfy (3), while maintaining $(y, z) \in G_x$. This yields (Y, x, y, z) feasible for (SDP$_1$).

This proposition essentially demonstrates that the added variables and constraints, which differentiate (SDP$_1$) from (SDP$_0$), are effective in tightening the feasible set when $\delta(x, G_x)$ is greater than $\tilde{Q} \bullet Y$ for a large portion of (Y, x).

From a practical standpoint, we have actually never observed a situation in our test problems for which the optimal value of (SDP$_1$) is strictly lower than that of (SDP$_0$). We believe this is evidence that the specific function $\delta(x, G_x)$ is rather weak, i.e., it is often less than or equal to $\tilde{Q} \bullet Y$. Nevertheless, the perspective provided by Proposition 4.1 gives insight into how one can tighten the SDP relaxations. Roughly speaking, the proposition tells us there are two basic ways to tighten:

(i) lower $\tilde{Q} \bullet Y$, that is, tighten the basic SDP relaxation (SDP$_0$) so that (Y, x) better approximates (2);
(ii) raise $\delta(x, G_x)$, that is, replace G_x in the KKT relaxation (SDP$_1$) with an H_x that better approximates $G_x \cap C_x$.

We will see in the next section that enforcing complementarities in branch-and-bound serves to accomplish both (i) and (ii), and this branching significantly tightens the SDP relaxations as evidenced by a small number of nodes in the branch-and-bound tree. In addition, we will introduce below a second SDP relaxation of (KKT) that is constructed in the spirit of (i) and (ii).

Before we describe the next relaxation, however, we would like to make a comment about the specific constraint (3) in (SDP$_1$), which equates the objectives of (SDP$_0$) and (RKKT). Though this constraint is key in bounding the feasible set of (SDP$_1$) and in developing the perspective given by Proposition 4.1, we have found in practice that it does not tighten the SDP relaxation unless many complementarity constraints have been fixed through branching. Moreover, SDPs that do not include (3) can often be solved much more quickly than ones that do. So, while (3) is an important constraint theoretically (we will discuss additional properties below and in Sect. 5), we have found that it can

be sometimes practically advantageous to avoid. We should remark, however, that, to maintain boundedness, dropping (3) does require one to include valid bounds for (y, z)—bounds that should be computed in a preprocessing phase that explicitly incorporates (3). We will discuss this in more detail in the next section.

The SDP that we introduce next relaxes (KKT) by handling the quadratic constraints $(y, z) \in C_x$ directly. We follow the discussion at the beginning of Sect. 2, except this time we focus on the variables (x, y, z) instead of just x. Let $\hat{Y}$ be related to (x, y, z), which is a feasible solution of (KKT), by the following quadratic equation:

$$\hat{Y} = \begin{pmatrix} 1 \\ x \\ y \\ z \end{pmatrix} \begin{pmatrix} 1 \\ x \\ y \\ z \end{pmatrix}^T = \begin{pmatrix} 1 & x^T & y^T & z^T \\ x & xx^T & xy^T & xz^T \\ y & yx^T & yy^T & yz^T \\ z & zx^T & zy^T & zz^T \end{pmatrix}. \tag{4}$$

Defining $\hat{n} := n + m + n$,

$$\hat{K} := \left\{ (x_0, x, y, z) \in \mathbb{R}^{1+\hat{n}} \ : \ Ax \leq x_0 b,\ A^T y - z = Qx + x_0 c,\ (x_0, x, y, z) \geq 0 \right\},$$

and

$$\hat{Q} := \frac{1}{2} \begin{pmatrix} 0 & c^T & 0 & 0 \\ c & Q & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

and letting $\hat{Y}_{xy}$ and $\hat{Y}_{xz}$ denote the xy- and xz-blocks of $\hat{Y}$, respectively, we observe the following:

- $\hat{Y} \in \mathcal{S}_+^{1+\hat{n}}$ (or simply $\hat{Y} \succeq 0$);
- $\hat{Y} e_i \in \hat{K}$ for all $i = 1, \ldots, \hat{n}$;
- using Proposition 2.1, the objective of (KKT) can be expressed as $\hat{Q} \bullet \hat{Y}$;
- similar to (SDP_1), the equation $\hat{Q} \bullet \hat{Y} = (b^T y + c^T x)/2$ is satisfied;
- the condition $(y, z) \in C_x$ is equivalent to $\operatorname{diag}(A \hat{Y}_{xy}) = b \circ y$ and $\operatorname{diag}(\hat{Y}_{xz}) = 0$.

We let $\hat{M}_+$ denote the set of all $\hat{Y}$ satisfying the first and second properties:

$$\hat{M}_+ := \left\{ \hat{Y} \succeq 0 : \hat{Y} e_i \in \hat{K} \quad \forall i = 1, \ldots, \hat{n} \right\}.$$

By dropping the last $\hat{n}$ columns of the equation (4), we arrive at the following relaxation:

$$
\begin{aligned}
\max \quad & \hat{Q} \bullet \hat{Y} && (\mathrm{SDP}_2)\\
\text{s.t.} \quad & \hat{Y} \in M_+, \quad \hat{Y}e_0 = (1; x; y; z)\\
& x \in P, \quad (y, z) \in G_x\\
& \hat{Q} \bullet \hat{Y} = (b^T y + c^T x)/2\\
& \operatorname{diag}(A\hat{Y}_{xy}) = b \circ y, \quad \operatorname{diag}(\hat{Y}_{xz}) = 0.
\end{aligned}
$$

We remark that, similar to (SDP_1), (SDP_2) can be tailored to derive a related SDP relaxation for each each node in the branch-and-bound tree.

Clearly (SDP_2) is at least as strong as (SDP_1), though it is much bigger. We can see its potential for strengthening (SDP_1) through the perspective of (i) and (ii) above. By lifting and projecting with respect to (x, y, z) and the linear constraints $x \in P$, $(y, z) \in G_x$, we accomplish (i), namely we tighten the relationship between the original variables and the positive semidefinite variable. By incorporating the "diag" constraints, which represent complementarity, we are indirectly tightening the constraint $(y, z) \in G_x$ by reducing it to a smaller one that better approximates $G_x \cap C_x$. The computational results will confirm the strength of (SDP_2) over (SDP_1).

5 The branch-and-bound implementation

In this section, we describe our computational experience with the branch-and-bound algorithm for (QP) using both SDP relaxations (SDP_1) and (SDP_2).

5.1 Implementation details

One of the most fundamental decisions for any branch-and-bound algorithm is the method employed for solving the relaxations at the nodes of the tree. As mentioned in the introduction, we have chosen to use the method proposed by Burer and Vandenbussche [4] for solving the SDP relaxations because of its applicability and scalability for SDPs of the type (SDP_1) and (SDP_2). For the sake of brevity, we only describe the features of the method that are relevant to our discussion, since a number of the method's features directly affect key implementation decisions.

The algorithm uses a Lagrangian constraint-relaxation approach, governed by an augmented Lagrangian scheme to ensure convergence, that focuses on obtaining subproblems that only require the solution of convex quadratic programs over the constraint set K or $\hat{K}$, which are solved using CPLEX by ILOG, Inc. [11]. In particular, constraints such as (3) are relaxed with explicit dual variables. By the nature of the method, a valid upper bound on the optimal value of the relaxation is available at all times, which makes the method a reasonable choice within branch-and-bound even if the relaxations are not solved to high accuracy. For convenience, we will refer to this method as the *AL method*.

Recall that the boundedness of (y, z) in (SDP$_1$) and (SDP$_2$) relies on the equality constraint (3). So, in principle, relaxing this constraint with an unrestricted multiplier λ can cause problems for the AL method because its subproblems may have unbounded objective value corresponding to $b^T y \to \infty$. (This behavior was actually observed in practice.) Hence to ensure that the subproblems in the AL method remain bounded, one must restrict $\lambda \leq 0$ so that the term $\lambda\, b^T y$ appears in the subproblem objective. In fact, it is not difficult to see that this is a valid restriction on λ.

Early computational experience demonstrated that (3) often slows down the solution of the SDP subproblems, mainly due to the fact that it induces certain dense linear algebra computations in the AL subproblems. On the other hand, removing this constraint completely from (SDP$_1$) and (SDP$_2$) had little negative effect on the quality of the relaxations at the top levels of the branch-and-bound tree. As pointed out in Sect. 4, (3) is unlikely to have any effect until many complementarities have been fixed, that is, until H_x gets close to $G_x \cap C_x$. Of course, it is theoretically unwise to ignore (3) because both the boundedness of (y, z) and the correctness of branch-and-bound depend on it. (We were able to generate an example of a leaf node that would not be fathomed if (3) was left out.)

It is nevertheless possible to drop (3) in most of the branch-and-bound relaxations and still manage the boundedness and correctness issues:

- In a preprocessing phase, we include (3) and compute bounds on (y, z). To compute these bounds, we solve two SDPs, with objectives $e^T y$ and $e^T z$, respectively, which yield valid upper bounds for y_i and z_j since $y, z \geq 0$. We solve these to a loose accuracy. The resulting bounds for y and z are then carried throughout all branch-and-bound calculations.
- At the leaf nodes, instead of (SDP$_1$) or (SDP$_2$), we substitute the natural LP relaxation of (1), which correctly fathoms leaf nodes. (In our computations branch-and-bound completed before reaching the leaf nodes, and so we actually never had to invoke this LP relaxation.)

So we chose to remove (3) from all calculations except the preprocessing for upper bounds on (y, z). Excluding the constraint usually did not result in an increase in the number of nodes in the branch-and-bound tree, and so we were able to realize a significant reduction in overall computation time.

To further expedite the branch-and-bound process, we also attempted to tighten (SDP$_1$) by adding constraints of the form $Y(e_0 - e_i) \in K$, which are valid due to Assumption 2.4. Note that, while the constraint $x \leq e$ may be redundant for P, the constraints $Y(e_0 - e_i) \in K$ are generally not redundant for the SDP. Although these additional constraints do increase the computational cost of the AL method, the impact is small due to the decomposed nature of the AL method. Moreover, the benefit to the overall branch-and-bound performance is dramatic due to strengthened bounds coming from the relaxations.

In the case of (SDP$_2$), we can add similar constraints. Assuming that bounds for (y, z) have already been computed as described above, we can then rescale these variables so that they, like x, are also contained in the unit box. We now

may add $\hat{Y}(e_0 - e_i) \in \hat{K}$ to tighten (SDP$_2$). (Note that we chose to scale (y, z) in the case of (SDP$_1$) as well; see first item in the next paragraph.)

Some final details of the branch-and-bound implementation are:

- Complementarities to branch on are chosen using a maximum normalized violation approach. Given a primal solution $(\bar{x}, \bar{y}, \bar{z})$ and slack variables $\bar{s} := b - A\bar{x}$ obtained from a relaxation, we compute $\text{argmax}_j\{\bar{x}_j\bar{z}_j\}$ and $\text{argmax}_i\{\bar{s}_i\bar{y}_i/\hat{s}_i\}$, where $\hat{s}_i$ is an upper bound on the slack variable that has been computed *a priori* in a preprocessing phase (in the same way as bounds for (y, z) are computed). Recall that x, y, and z are already scaled to be in the unit interval and hence the violations computed are appropriately normalized. We branch on the resulting index (i or j) corresponding to the highest normalized violation.
- After solving a relaxation, we use $\bar{x}$ as a starting point for a QP solver based on nonlinear programming techniques to obtain a locally optimal solution to (QP). In this way, we obtain good lower bounds that can be used to fathom nodes in the tree.
- We use a best bound strategy for selecting the next node to solve in the branch-and-bound tree.
- Upon solution of each relaxation of the branch-and-bound tree, a set of dual variables is available for those constraints of the SDP relaxation that were relaxed in the AL method. These dual variables are then used as the initial dual variables for the solution of the next subproblem, in effect performing a crude warm-start, which proved extremely helpful in practice.
- We use a relative optimality tolerance for fathoming. In other words, for a given tolerance ε, a node with upper bound z_{ub} is fathomed if $(z_{ub} - z_{lb})/z_{lb} < \varepsilon$, where z_{lb} is the value of the best primal solution found thus far. In our computations, we experiment with $\varepsilon = 10^{-6}$ and $\varepsilon = 10^{-2}$, which we refer to as *default* and *1%* optimality tolerances, respectively.

5.2 Computational results

We tested our SDP-based branch-and-bound on several types of instances. Our primary interest is the performance of (SDP$_1$) compared to that of (SDP$_2$), but we also consider the effect of different optimality tolerances (default versus 1%). In total, each instance was solved four times, representing the various choices of relaxation type and optimality tolerance. All computations were performed on a Pentium 4 running at 2.4 GHz under the Linux operating system. A summary of our conclusions is as follows:

- In all cases, the number of nodes required is small, and when a 1% optimality tolerance is used, the vast majority of problems solve in one node. This illustrates the strength of the SDP relaxations.
- Runs with (SDP$_2$) often require significantly fewer nodes than those with (SDP$_1$), indicating the strength of (SDP$_2$) over (SDP$_1$). In runs where (SDP$_1$) takes fewer nodes, numerical inaccuracy when solving

(SDP$_2$)—a consequence of the size and complexity of (SDP$_2$)—appears to be an important contributing factor.
- For all instances tested, runs with (SDP$_1$) outperform those with (SDP$_2$) in terms of CPU time.
- In absolute terms, the CPU times of the runs range from quite reasonable (a few seconds on the smallest instances using (SDP$_1$) and a 1% tolerance) to quite large (a few days on the largest instances using (SDP$_2$) and the default tolerance). The bottleneck is the solution of the SDPs. The small number of nodes indicates that improved SDP solution techniques will have a significant impact on overall CPU times.

Related to the last point, we should note that it is possible to tweak the implementation so that the subproblems are not solved to a high level of accuracy, especially at nodes with small depth in the branch-and-bound tree. This way you may enumerate a few more nodes but may save in total CPU time.

5.2.1 Box-constrained instances

We tested our methodology on the 54 instances of box-constrained QPs introduced by Vandenbussche and Nemhauser [26]. These instances range in size from 20 to 60 variables and have varying densities for the matrix Q. To compare (SDP$_1$) and (SDP$_2$), we present log–log plots of the number of nodes and CPU times in Figs. 1 and 2 for both default and 1% optimality tolerances. Observe that the number of nodes required for any instance is quite small and that (SDP$_2$) requires fewer nodes than (SDP$_1$). This reflects the fact that (SDP$_2$) is at least as tight as (SDP$_1$). Also, the overall time required by (SDP$_1$) is significantly smaller than that for (SDP$_2$). (One comment on the graphics: the plots in Fig. 1 may appear to have fewer than 54 points, but in actuality there are many overlapping points.)

5.2.2 Applications: inventory and scheduling models

We also tested our method on problems motivated by models in inventory theory and scheduling.

The inventory model, which we briefly describe here, has been presented by Lootsma and Pearson [15]. Suppose there are q products, having sales rates s_i and production rates p_i $\forall i = 1, \ldots, q$. The products are to be produced on one machine according to a sequence $\sigma \in \mathbb{Z}^{2n}$, where $0 \leq \sigma_j \leq q$ indicates that product σ_j will be produced in the j-th production period. If $\sigma_j = 0$, then the machine is idle in period j. We assume that $\sigma_{2j} = 0$ $\forall j = 1, \ldots, n$ and $\sigma_{2j+1} \neq 0$ $\forall j = 0, \ldots, n-1$. Using this data, we can define an activity matrix a_{ij} $\forall i = 1, \ldots, q$, $j = 1, \ldots, 2n$ with

$$a_{ij} = \begin{cases} p_i - s_i & \text{if } i = \sigma_j \\ -s_i & \text{otherwise.} \end{cases}$$

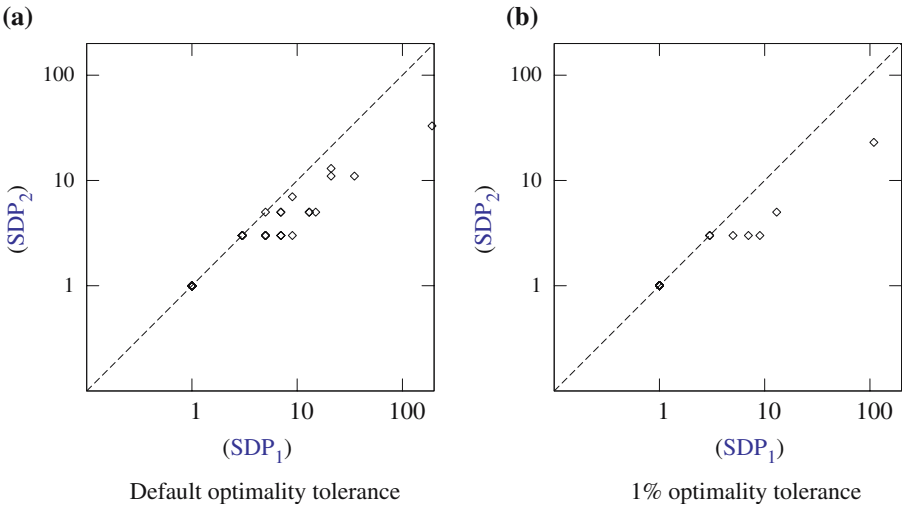

Fig. 1 Nodes required for (SDP$_1$) and (SDP$_2$) on box-constrained instances (both default and 1% optimality tolerances)

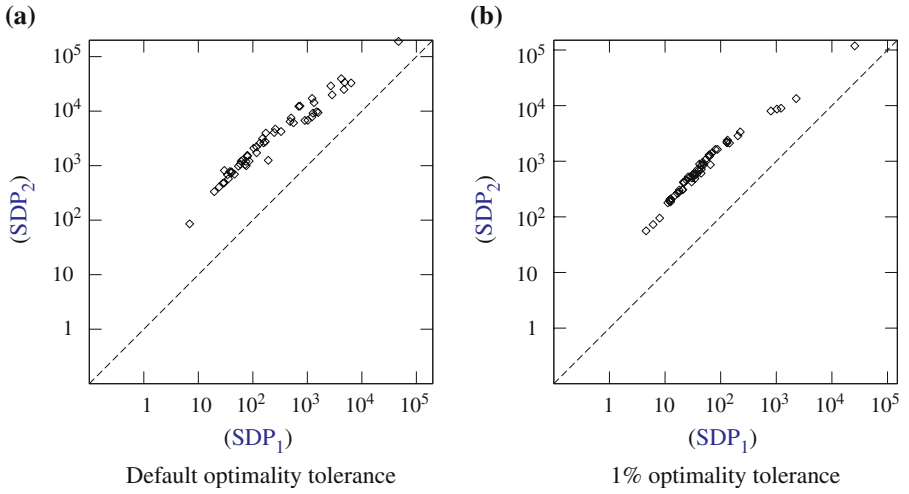

Fig. 2 CPU times required for (SDP$_1$) and (SDP$_2$) on box-constrained instances (both default and 1% optimality tolerances)

Finally, denote c_i as the inventory holding cost for product i. We want to find initial inventories v_i for each product and the length of each production period t_j $\forall j = 1, \ldots, 2n$. The objective is to minimize holding costs over one production cycle of a given length T and to make the production cycle repeatable. Lootsma and Pearson [15] showed that this problem may be modeled as a nonconvex QP:

$$
\begin{aligned}
\min \quad & \sum_{i=1}^{q} c_i \sum_{j=1}^{2n} \left[\left(v_i + \sum_{k=1}^{j-1} a_{ik} t_j \right) t_j + \frac{1}{2} a_{ij} t_j^2 \right] \\
\text{s.t.} \quad & v_m + \sum_{k=1}^{2j} a_{mk} t_k \geq 0, \quad \forall j = 1, \ldots, n-1,\ m = \sigma_{2j+1} \\
& \sum_{j=1}^{2n} t_j = T \\
& \sum_{j=1}^{2n} a_{ij} t_j \geq 0 \quad \forall i = 1, \ldots, q \\
& v_i \geq 0\ \forall i, \quad t_j \geq 0\ \forall j
\end{aligned}
$$

It is easy to see that we may replace the equality with $\sum_{j=1}^{2n} t_j \geq T$; we may also assume $T = 1$. In addition, any optimal solution will have $v_i \leq s_i$, and hence we may rescale the initial inventory variables to be contained in the unit interval. The resulting QP, which fits our standard form, has $2n + q$ nonnegative variables and $n + q$ inequality constraints. We randomly generated three instances for each parameter pair $(n, q) \in \{10, 20\} \times \{4, 6\}$ for a total of 12 instances. Production rates p are integers uniformly generated from the interval $[40, 50]$, while the sales rates are integers ranging uniformly over $[1, 5]$. Lastly, the σ vector was produced by first assigning each product to a random time period (this guarantees that each product will have at least some production time), and then randomly assigning products to the remaining time periods.

The QP derived from a scheduling problem is developed by Skutella [24] and solves the scheduling problem $Rm||\sum_{j=1}^{n} w_j C_j$, i.e. scheduling n jobs on m parallel unrelated machines to minimize the weighted sum of completion times. The data for this problem consist of weights w_j for each job $j = 1, \ldots, n$ and processing times p_{ij} for job j on machine i. For each machine i, an ordering $\prec_i$ of the jobs is defined by setting $j \prec_i k$ if $w_j/p_{ij} > w_k/p_{ik}$ or $w_j/p_{ij} = w_k/p_{ik}$ and $j < k$.

Defining binary variables a_{ij} to indicate if job j is assigned to machine i, a discrete model for this scheduling problem is

$$
\begin{aligned}
\min \quad & \sum_{j=1}^{n} w_j \left[\sum_{i=1}^{m} a_{ij} \left(p_{ij} + \sum_{k \prec_i j} a_{ik} p_{ik} \right) \right] \\
\text{st} \quad & \sum_{i=1}^{m} a_{ij} = 1\ \forall j \\
& a_{ij} \in \{0, 1\}\ \forall i, j
\end{aligned}
\tag{IQP}
$$

Skutella [24] showed that the optimal objective value does not decrease by relaxing the integrality restrictions and moreover that one can construct an

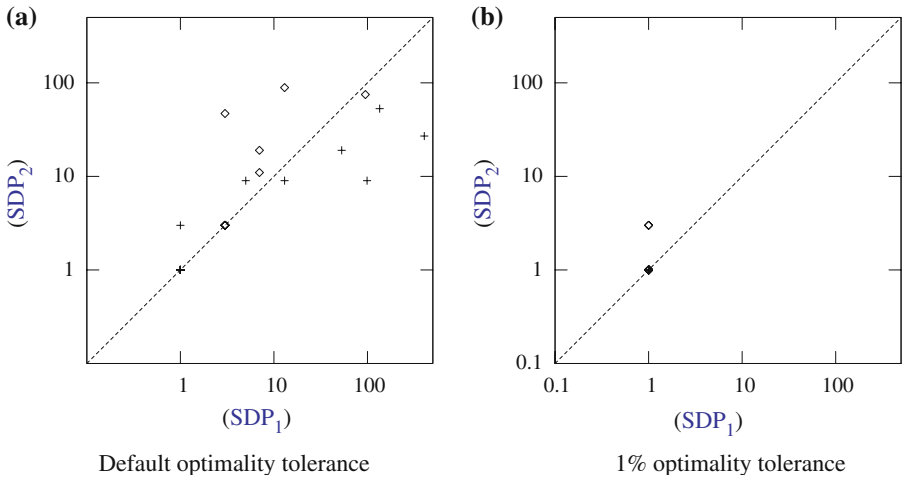

Fig. 3 Nodes required for (SDP_1) and (SDP_2) on scheduling (+) and inventory ($\diamond$) instances (both default and 1% optimality tolerances)

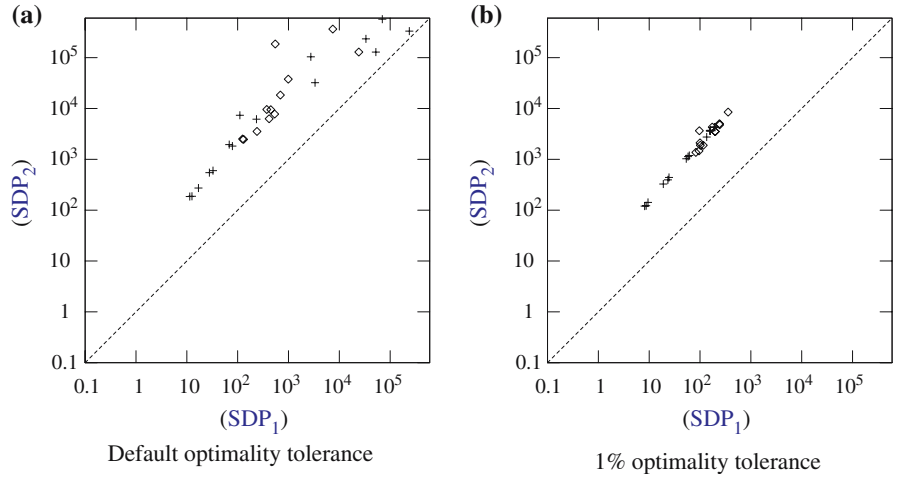

Fig. 4 CPU times required for (SDP_1) and (SDP_2) on scheduling (+) and inventory ($\diamond$) instances (both default and 1% optimality tolerances)

optimal integer solution from an optimal fractional solution. By further replacing each constraint $\sum_i a_{ij} = 1$ with $\sum_i a_{ij} \geq 1$, which clearly does not affect the problem, we can bring the scheduling problem into our standard form. We randomly generated three scheduling instances for each parameter pair $(n, m) \in \{(10, 2), (10, 3), (20, 2), (20, 3), (30, 2)\}$ for a total of 15 instances. Processing times were integers varying uniformly over the interval $[10, 50]$, while the weights, w_j, ranged from 1 to 10.

We compare the number of nodes required by (SDP_1) and (SDP_2) in Fig. 3. Unlike with the box-constrained instances, (SDP_2) is not the clear winner.

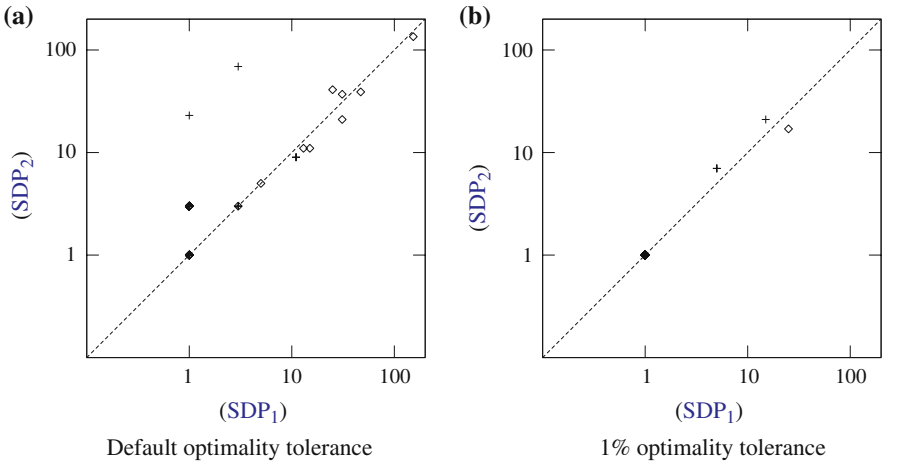

Fig. 5 Nodes required for (SDP_1) and (SDP_2)on Globallib (+) and random ($\diamond$) instances (both default and 1% optimality tolerances)

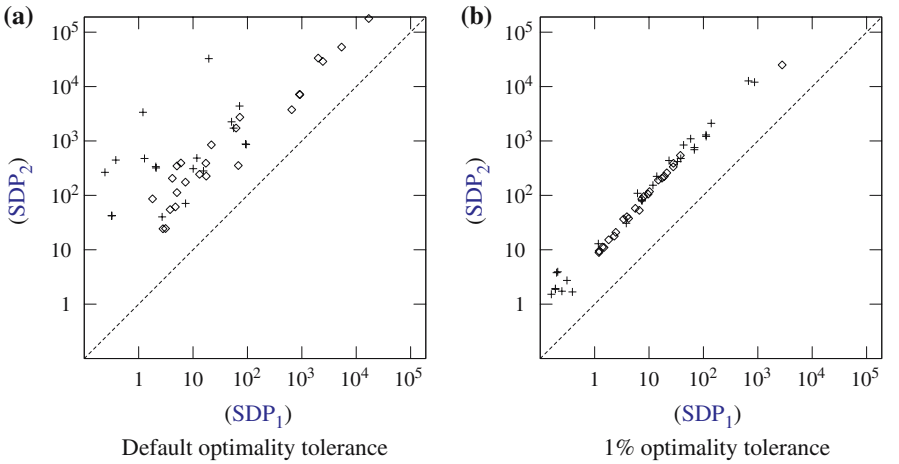

Fig. 6 CPU times required for (SDP_1) and (SDP_2)on Globallib (+) and random ($\diamond$) instances (both default and 1% optimality tolerances)

Upon closer inspection, we believe that instances where (SDP_2) requires mores nodes can be attributed to the inaccurate solution of the much larger SDP relaxations for (SDP_2). In either case, however, very few nodes were required. In particular, when a 1% optimality tolerance is used, almost all instances can be solved at the root, which indicates the strength of the proposed SDP relaxations. In Fig. 4, we compare CPU times for both types of relaxations in log-log plots. Once we again we see that runs with (SDP_1) take significantly less time.

5.2.3 Globallib and random instances

To further test our algorithm, we used QP instances from Globallib [7], a collection of nonlinear programming problems, many of which are nonconvex. In particular, we selected any linearly constrained, bounded, nonconvex QP whose number of variables ranged between 10 and 50. In order to use these instances within our framework, we scaled the variables to be restricted to the unit interval. However, this rescaling introduced some occasional numerical failures in CPLEX when solving convex QP subproblems in the AL method. We have omitted instances where such issues where encountered. Overall, we report on approximately 20 Globallib instances.

We also randomly generated 24 interior-feasible QPs with general constraints $Ax \leq b$ as well as the constraints $0 \leq x \leq e$ to ensure boundedness. These instances range in size from 10 to 20 variables and 1 to 20 general constraints and have fully dense Q and A matrices. The entries of A are uniformly generated integers over the range $[-5, 5]$ and the entries of Q and c range from -50 to 50. The right hand side, b, was obtained by rounding up each entry of $\frac{1}{2}Ae$.

In Figs. 5 and 6, we compare the number of nodes required for both relaxation types in a log-log plot. Similar conclusions hold as for runs previously mentioned.

Acknowledgments The authors would like to thank the editors and anonymous referees for numerous suggestions that have improved the paper immensely.

References

1. Absil, P.-A., Tits, A.: Newton-KKT interior-point methods for indefinite quadratic programming. Manuscript, Department of Electrical and Computer Engineering, University of Maryland, College Park, MD, USA (2006). To appear in Computational Optimization and Applications
2. Anstreicher, K.M.: Combining RLT and SDP for nonconvex QCQP. Talk given at the Workshop on Integer Programming and Continuous Optimization, Chemnitz University of Technology, November 7–9 (2004)
3. Bomze, I.M., de Klerk, E.: Solving standard quadratic optimization problems via linear, semidefinite and copositive programming. J. Global Optim. **24**(2), 163–185 (2002). Dedicated to Professor Naum Z. Shor on his 65th birthday
4. Burer, S., Vandenbussche, D.: Solving lift-and-project relaxations of binary integer programs. SIAM J. Optim. **16**(3), 726–750 (2006)
5. Floudas, C., Visweswaran, V.: Quadratic optimization. In: Horst, R., Pardalos, P. (eds.) Handbook of Global Optimization, pp. 217–269. Kluwer Academic Publishers (1995)
6. Giannessi, F., Tomasin, E.: Nonconvex quadratic programs, linear complementarity problems, and integer linear programs. In: Fifth Conference on Optimization Techniques (Rome, 1973), Part I, pp. 437–449. Lecture Notes in Computer Science, vol. 3. Springer, Berlin Heidelberg New York (1973)
7. Globallib: http://www.gamsworld.org/global/globallib.htm
8. Goemans, M., Williamson, D.: Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. J. ACM **42**, 1115–1145 (1995)
9. 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), vol. 72 of Appl. Optim., pp. 149–179. Kluwer Academic Publishers, Dordrecht (2002)

10. Hansen, P., Jaumard, B., Ruiz, M., Xiong, J.: Global minimization of indefinite quadratic functions subject to box constraints. Naval Res. Logist. **40**(3), 373–392 (1993)
11. ILOG, Inc.: ILOG CPLEX 9.0, User Manual (2003).
12. Kojima, M., Tunçel, L.: Cones of matrices and successive convex relaxations of nonconvex sets. SIAM J. Optim. **10**(3), 750–778 (2000)
13. Kojima, M., Tunçel, L.: Some fundamental properties of successive convex relaxation methods on LCP and related problems. J. Global Optim. **24**(3), 333–348 (2002)
14. Kozlov, M.K., Tarasov, S.P., Khachiyan, L.G.: Polynomial solvability of convex quadratic programming. Dokl. Akad. Nauk SSSR **248**(5), 1049–1051 (1979)
15. Lootsma, F.A., Pearson, J.D.: An indefinite-quadratic-programming model for a continuous-production problem. Philips Res. Rep. **25**, 244–254 (1970)
16. Lovász, L., Schrijver, A.: Cones of matrices and set-functions and 0–1 optimization. SIAM J. Optim. **1**, 166–190 (1991)
17. Nesterov, Y.: Semidefinite relaxation and nonconvex quadratic optimization. Optim. Methods Softw. **9**, 141–160 (1998)
18. Pardalos, P.: Global optimization algorithms for linearly constrained indefinite quadratic problems. Comput. Math. Appl. **21**, 87–97 (1991)
19. Pardalos, P.M., Vavasis, S.A.: Quadratic programming with one negative eigenvalue is NP-hard. J. Global Optim. **1**(1), 15–22 (1991)
20. Sahinidis, N.V.: BARON a general purpose global optimization software package. J. Glob. Optim. **8**, 201–205 (1996)
21. Sherali, H.D., Fraticelli, B.M.P.: Enhancing RLT relaxations via a new class of semidefinite cuts. J. Global Optim. **22**, 233–261 (2002)
22. Sherali, H.D., Tuncbilek, C.H.: A global optimization algorithm for polynomial programming problems using a reformulation-linearization technique. J. Global Optim. **2**(1), 101–112 (1992) Conference on Computational Methods in Global Optimization, I (Princeton, NJ, 1991).
23. Sherali, H.D., Tuncbilek, C.H.: A reformulation-convexification approach for solving nonconvex quadratic programming problems. J. Global Optim. **7**, 1–31 (1995)
24. Skutella, M.: Convex quadratic and semidefinite programming relaxations in scheduling. J. ACM **48**(2), 206–242 (2001)
25. Vandenbussche, D., Nemhauser, G.: A polyhedral study of nonconvex quadratic programs with box constraints. Math. Program. **102**(3), 531–557 (2005)
26. Vandenbussche, D., Nemhauser, G.: A branch-and-cut algorithm for nonconvex quadratic programs with box constraints. Math. Program. **102**(3), 559–575 (2005)
27. Ye, Y.: Approximating quadratic programming with bound and quadratic constraints. Math. Program. **84**(2, Ser. A), 219–226 (1999)