



HAL
open science

Using quadratic cuts to iteratively strengthen convexifications of box quadratic programs

Amélie Lambert, Daniel Cosmin Porumbel

► **To cite this version:**

Amélie Lambert, Daniel Cosmin Porumbel. Using quadratic cuts to iteratively strengthen convexifications of box quadratic programs. 2024. hal-04582848

HAL Id: hal-04582848

<https://hal.science/hal-04582848v1>

Preprint submitted on 22 May 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Using quadratic cuts to iteratively strengthen convexifications of box quadratic programs

Amélie Lambert¹ and Daniel Porumbel²

^{1,2}Cedric-Cnam, 292, rue saint-Martin 75141 Paris Cedex 03 France.

Contributing authors: amelie.lambert@cnam.fr;
daniel.porumbel@cnam.fr;

Abstract

We seek the global minimum of a quadratic function f with box constrained variables. For this goal, we underestimate f by a convex piecewise-quadratic function defined as the maximum of $\mathbf{p} \geq \mathbf{1}$ convex quadratic functions (\mathbf{p} underestimators). We show that when $\mathbf{p} \rightarrow \infty$ the optimal solution of this relaxation converges to an optimal solution of the strong "Shor plus RLT" semi-definite relaxation of the initial problem. To compute the new relaxation, we introduce an iterative algorithm that adds convex quadratic cuts (or cutting-quadratics) one by one in cutting plane fashion. The resulting convexification is tighter than the one produced by previous related methods that use $\mathbf{p} = \mathbf{1}$, *i.e.*, using multiple underestimators leads to a stronger convexification than using a unique one (as in past work). Its integration into a spatial branch-and-bound algorithm brings a second advantage: compared to previous work, we can refine the lower bound at each node of the branching tree. This is because we are able to compute underestimators that act specifically on any particular node of the branching tree. Numerical results show that even a small value of $\mathbf{p} \in \{\mathbf{2}, \mathbf{3}\}$ can often be enough to reduce the branching tree size by half compared to sticking to $\mathbf{p} = \mathbf{1}$. The resulting algorithm is also competitive in terms of CPU time compared to well-established solvers that rely on other techniques.

Keywords: Quadratic Programming, piecewise-quadratic underestimator, cutting-quadratics algorithm

1 Introduction and literature review

Our goal is to find the exact solution of the following non-convex Quadratic Box-constrained Program (*QBP*):

$$(QBP) \begin{cases} \min f(x) \equiv \langle Q, xx^\top \rangle + c^\top x \\ \ell_i \leq x_i \leq u_i \end{cases} \quad \forall i \in \mathcal{I} \quad (1)$$

where $\langle A, B \rangle = \sum_{i=1}^n \sum_{j=1}^n a_{ij} b_{ij}$, $\mathcal{I} = \{1, \dots, n\}$, $(Q, c, \ell, u) \in \mathcal{S}_n \times \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^n$, and \mathcal{S}_n is the set of real symmetric matrices of order n . Without loss of generality, we assume that the box constraints (1) take the form $x_i \in [0, 1]$ and that the feasible domain of (*QBP*) is non-empty.

(*QBP*) is a fundamental \mathcal{NP} -hard global optimization problem [1]. Although (1) is the simplest non-convex quadratic optimization program, finding its global optimum remains very challenging even for medium sized instances. Since f is not a convex function, many local optima may not be global; standard approaches for solving (*QBP*) to global optimality include spatial branch-and-bound algorithms [2–4] combined with convex relaxations to determine lower bounds (see for instance [5, 6]). These convex relaxations are typically either linear, quadratic convex or semi-definite. The solution space is partitioned by branching along the spatial branch-and-bound execution, so as to tighten the convex relaxation as the nodes cover a smaller and smaller feasible area. This family of methods also apply to more general classes of quadratic problems involving quadratic constraints or mixed-integer variables (see for instance [6–9]).

1.1 Main relaxations from the literature

Many relaxations from the literature express the quadratic function in an extended space of variables, introducing new variables Y_{ij} that are meant to satisfy $Y_{ij} = x_i x_j$, for all $(i, j) \in \mathcal{I}^2$ where \mathcal{I}^2 is the cartesian product of set \mathcal{I} . This standard approach was first used for linearizing f , obtaining the following reformulation of (*QBP*):

$$(LP) \begin{cases} \min f_L(x, Y) \equiv \langle Q, Y \rangle + c^\top x \\ \ell_i \leq x_i \leq u_i \\ Y = xx^\top \end{cases} \quad \forall i \in \mathcal{I} \quad (2a)$$

$$(2b)$$

Problem (*LP*) is equivalent to problem (*QBP*), since when (x^*, Y^*) is a feasible solution to (*LP*), x^* is also feasible for (*QBP*), and both problems have the same objective value (i.e. $f_L(x^*, Y^*) = f(x^*)$). Thus, (*QBP*) can be solved by a spatial branch-and-bound based on the linear relaxation obtained by relaxing the non-convex feasible set (2a)–(2b) by its convex hull. Since it may be hard to completely describe this convex hull (see [10–12]), it is preferable to construct an outer approximation based on the

McCormick’s envelopes [13] captured by the following set \mathcal{M} :

$$\mathcal{M} := (x, Y) \in \mathbb{R}^n \times \mathcal{S}_n : \begin{cases} Y_{ij} \leq u_j x_i + \ell_i x_j - \ell_i u_j & (i, j) \in \mathcal{I}^2 & (3a) \\ Y_{ij} \leq \ell_j x_i + u_i x_j - u_i \ell_j & (i, j) \in \mathcal{I}^2 & (3b) \\ Y_{ij} \geq u_j x_i + u_i x_j - u_i u_j & (i, j) \in \mathcal{I}^2 & (3c) \\ Y_{ij} \geq \ell_j x_i + \ell_i x_j - \ell_i \ell_j & (i, j) \in \mathcal{I}^2 & (3d) \\ \ell_i \leq x_i \leq u_i & \forall i \in \mathcal{I} & (3e) \end{cases}$$

In the resulting linear relaxation called (\overline{LP}) , inequalities (3a)–(3e) involve the lower and upper bounds (ℓ and u) on the original variables x . Then, since the branching rules update the interval $[\ell, u]$ at each node of the branching tree, these inequalities become tighter and tighter in the course of the spatial branch-and-bound; this improves the value of the relaxation along the search, as deeper and deeper sub-nodes with shorter and shorter intervals $[\ell, u]$ are generated. This approach is used by several authors (see for instance [13–15]). In order to tighten (\overline{LP}) , several families of valid linear inequalities were introduced (see for instance [5, 16]) and added to the formulation by use of a cutting-plane approach. This idea is used by most software implementing the methods described above, see, *e.g.*, Baron ([17]), GloMIQO ([8, 18, 19]), or Gurobi ([20]). Although the evaluation of a linear relaxation is fast, the associated bound is often too weak, and the use of (\overline{LP}) or its extensions in a spatial branch-and-bound often fails to solve medium-sized problems to global optimality.

In order to get tighter relaxations of (QBP) , using semi-definite relaxations within branch-and-bound frameworks was also widely studied ([3, 4, 6, 21–23]). A semi-definite relaxation of (QBP) can be obtained by lifting x to a symmetric matrix $X = xx^T$ where these non-convex constraints are relaxed to $X - xx^T \succeq 0$, where $M \succeq 0$ means that M is positive semidefinite. By using the Schur complement, $X - xx^T \succeq 0$ is equivalent to (4e) below. After linking variables X and x with the McCormick constraints (4a)–(4d) and linearizing the objective function, we obtain the following model (introduced in [21]) referred to as the “Shor’s plus RLT” relaxation of (QBP) . The associated bound is very tight, but solving (SDP) in practice may be prohibitively slow even for medium-sized programs; this makes its direct and full integration into a branch and bound framework rather impractical.

$$(SDP) \left\{ \begin{array}{ll} \min f(X, x) \equiv \langle Q, X \rangle + c^T x & \\ X_{ij} \leq u_j x_i + \ell_i x_j - \ell_i u_j & (i, j) \in \mathcal{I}^2 \quad (4a) \\ X_{ij} \leq \ell_j x_i + u_i x_j - u_i \ell_j & (i, j) \in \mathcal{I}^2 \quad (4b) \\ X_{ij} \geq u_j x_i + u_i x_j - u_i u_j & (i, j) \in \mathcal{I}^2 \quad (4c) \\ X_{ij} \geq \ell_j x_i + \ell_i x_j - \ell_i \ell_j & (i, j) \in \mathcal{I}^2 \quad (4d) \\ \begin{pmatrix} 1 & x^T \\ x & X \end{pmatrix} \succeq 0 & (4e) \\ x \in \mathbb{R}^n \quad X \in \mathcal{S}_n & (4f) \end{array} \right.$$

A third family of classical approaches relies on convex quadratic relaxations, including in particular methods MIQCR (Mixed Integer Quadratic Convex

Reformulation) and extensions ([24–26]). In these approaches, a quadratic convex relaxation of (QBP) , referred to as $(\overline{P}_{S_0^*})$ and formally defined below, is calculated by exploiting the above (SDP) relaxation. The strength of this approach is that the optimal value of $(\overline{P}_{S_0^*})$ is equal to that of (SDP) . In fact, any SDP matrix $S_0 \succeq \mathbf{0}$ can produce a relaxation (\overline{P}_{S_0}) , but the optimal one is obtained by constructing matrix S_0^* from the optimal dual solution of (SDP) . We need to solve (SDP) only once for this very purpose.

Once the optimal S_0^* is determined, the original problem is then solved by a branch-and-bound based on the relaxation $(\overline{P}_{S_0^*})$. The problem is also expressed in the extended (x, Y) space; the reformulated objective can be seen as a quadratic surface (quadric) described by:

$$f_{S_0}(x, Y) = \langle S_0, xx^\top \rangle + c^\top x + \langle Q - S_0, Y \rangle, \quad (5)$$

where $S_0 \succeq \mathbf{0}$ can be any SDP matrix. It is easy to check that for any $S_0 \succeq \mathbf{0}$, the function f_{S_0} is convex; we also have $f_{S_0}(x, Y) = f(x)$ if $Y = xx^\top$. Note moreover that if S_0 is the positive semi-definite null matrix, we have $f_{S_0}(x, Y) = f_L(x, Y)$, which corresponds to simply linearising f . In other words, the reformulated objective function above includes the linear relaxation as special case (when taking $S_0 = \mathbf{0}$).

Let us now formally define model (P_{S_0}) . Considering all above information together, the following model is equivalent to (QBP) and the objective is convex for any $S_0 \succeq \mathbf{0}$:

$$(P_{S_0}) \left\{ \begin{array}{ll} \min f_{S_0}(x, Y) \equiv \langle S_0, xx^\top \rangle + c^\top x + \langle Q - S_0, Y \rangle & (6a) \\ Y = xx^\top & (6b) \\ \ell_i \leq x_i \leq u_i & i \in \mathcal{I} \quad (6c) \\ Y \in \mathcal{S}_n & (6d) \end{array} \right.$$

Then, as for the linear relaxation, the non-convex feasible set can be relaxed with the outer approximation based on the set \mathcal{M} of McCormick envelopes, leading to the following quadratic convex relaxation:

$$(\overline{P}_{S_0}) \left\{ \begin{array}{ll} \min f_{S_0}(x, Y) \equiv \langle S_0, xx^\top \rangle + c^\top x + \langle Q - S_0, Y \rangle & (7a) \\ (x, Y) \in \mathcal{M} & (7b) \end{array} \right.$$

One strength of this method is that (SDP) is solved only once at the root node of the spatial branch-and-bound leading to a quadratic convex relaxation with the same value as the optimum of (SDP) . The equivalence between both problems holds at the root node, i.e. for the initial lower and upper bounds ℓ and u only. In the course of the spatial branch-and-bound of MIQCR, we use the same S_0^* ; the same non-SDP convex program $(\overline{P}_{S_0^*})$ is solved at each sub-node of the branching tree, even if the interval $[\ell, u]$ is updated along the execution. This is significantly faster than solving a semi-definite optimization problem at each sub-node of the tree, i.e., for each $[\ell, u]$. However, for a given sub-node (i.e., when ℓ and u are updated), the optimal value of $(P_{S_0^*})$ is no longer equal to that of the (SDP) program associated to the updated ℓ and u .

1.2 Our contributions

Our goal is to design an approach that aims at reaching the value of (SDP) for any interval $[\ell, u]$ corresponding to any sub-node of the branching tree. We can strengthen the convexification at each sub-node by enriching $(P_{S_0^*})$ with quadratic convex cuts (hyper-surfaces) specifically tailored to the current $[\ell, u]$ (corresponding to the current sub-node). As mentioned previously problem $(P_{S_0^*})$ captures the tightness of (SDP) at the root node (i.e. for the initial values of ℓ and u), but not for each sub-node (i.e. with updated values of ℓ and u). We aim at improving the value of $(P_{S_0^*})$ at each sub-node by strengthening it with specifically tailored quadratic cuts, to make the convexification at the local sub-node for the current $[\ell, u]$ tend to the associated value of (SDP) .

For this purpose, our main idea is to replace the unique function f_{S_0} with multiple functions $f_{S_k}, S_k \geq 0$, with $k = 0, 1, 2, \dots, p$ and p a given integer. We then minimize, over all $(x, Y) \in \mathcal{M}$, the following function:

$$f^*(x, Y) = \max_{k=\{1, \dots, p\}} f_{S_k}(x, Y)$$

Let us focus on Figure 1. Since each function f_{S_k} is quadratic and convex, notice we obtain a piecewise-quadratic convex underestimator f^* . In our main algorithm, the idea is to generate the functions f_k one by one as in a cutting-planes approach, since each function f_{k+1} aims at cutting the current optimal solution (x^k, Y^k) at iteration k by making it sub-optimal. In our new general scheme, the relaxation (P_{S_0}) corresponds to the case $p = 1$. Figure 1 also illustrates how this approach with multiple quadratic cuts ($p > 1$) may generate a tighter more refined convexification than MIQCR ($p = 1$) even without necessarily improving the value of the general lower bound.

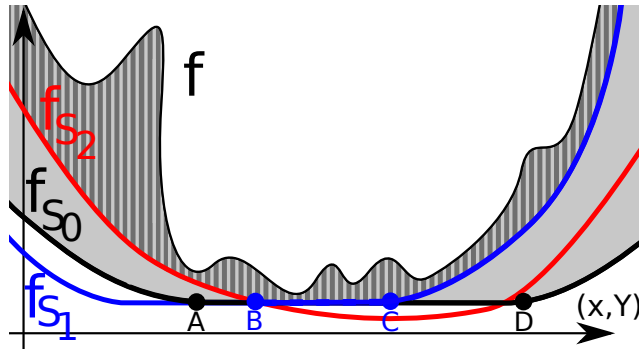


Fig. 1: Convex function f_{S_0} in black may reach its minimum over many solutions of the (x, Y) space (see the long flat segment $[A, D]$), because S_0 may have a large null space, often of dimension close to $\frac{n}{2}$. The resulting convexification in light-gray is weaker than the piecewise-quadratic convexification from the striped area (max of f_{S_0} , f_{S_1} and f_{S_2}) that has the same minimum; this latter convex function reaches its minimum over a segment $[B, C]$ shorter than $[A, D]$.

The above piece-wise quadratic function f^* is convex, but it is particularly computationally demanding to optimize it repeatedly. Each time f^* integrates a new quadric (at each iteration), we have to call a convex QCQP (quadratically constrained quadratic programming) solver to determine its new optimal solution. This is the *main computational bottleneck* of the overall approach and it is very important to accelerate this solver in practice. We thus propose the following *speed-up technique*: identify the variables that do not change too much from one QCQP call to another and strongly limit their variation at each new QCQP call. The effect is similar to reducing the number of variables. If the resulting optimal solution stays strictly inside the box associated to the imposed variation limit, we considered the speed-up technique was successful. Otherwise, the technique failed and we may have to remove the box and call the full solver; this is described in Section 4.2.

The road-map of the paper is as follows. Section 2 introduces $(\bar{P}_{\mathcal{K}})$, a parameterized family of piecewise-quadratic and convex relaxations of (QBP) . We then show that for any $[\ell, u]$, the associated value of (SDP) is equal to the optimal value of $(\bar{P}_{\mathcal{K}})$, *i.e.*, it is equal to the best relaxation within this family. Section 3 introduces an iterative **Cutting Quadratics Algorithm (CQA)** that iteratively refines convexification $(\bar{P}_{\mathcal{K}})$ by adding convex quadratic cuts one by one. This algorithm proceeds in cutting-plane fashion, using quadratic hyper-surfaces (or quadrics) instead of hyper-planes to iteratively tighten the relaxation. In Section 4, we integrate Algorithm CQA within a branch-and-bound scheme to solve (QBP) to global optimality. Finally, Section 5 presents experimental results on the *boxqp* instances, suggesting that our new approach is faster than state-of-the-art solvers, and is able to significantly reduce the number of nodes in comparison to the basic MIQCR algorithm.

2 A family of convex piecewise-quadratic relaxations

Given a set $\mathcal{K} = \{S_k \succeq \mathbf{0}, k = 0, 1, \dots, p\}$ of SDP matrices, the multi-cut version of (\bar{P}_{S_0}) from (6a)–(6d) takes the form below, forming a family of equivalent formulations of (QBP) indexed by set \mathcal{K} :

$$(P_{\mathcal{K}}) \begin{cases} \min t & \\ t \geq \langle S_k, xx^\top \rangle + c^\top x + \langle Q - S_k, Y \rangle & S_k \in \mathcal{K} & (8a) \\ Y = xx^\top & & (8b) \\ \ell_i \leq x_i \leq u_i & i \in \mathcal{I} & (8c) \\ Y \in \mathcal{S}_n, t \in \mathbb{R} & & (8d) \end{cases}$$

Like in the mono-cut version (6a)–(6d), the only non-convexity of $(P_{\mathcal{K}})$ comes from constraints (8b) that we can classically relax using the set \mathcal{M} of McCormick envelopes (3a)–(3d). We thus obtain $(\bar{P}_{\mathcal{K}})$ a family of convex relaxations of (QBP) indexed by the same set \mathcal{K} as above:

$$(\bar{P}_{\mathcal{K}}) \begin{cases} \min t & \\ t \geq \langle S_k, xx^\top \rangle + c^\top x + \langle Q - S_k, Y \rangle, S_k \in \mathcal{K} & (9a) \\ (x, Y) \in \mathcal{M} & (9b) \\ Y \in \mathcal{S}_n, t \in \mathbb{R} & (9c) \end{cases}$$

Clearly, for any set \mathcal{K} , the problem $(\overline{P}_{\mathcal{K}})$ is a relaxation of (QBP) , since for any solution \bar{x} of (QBP) of value t , the solution $(\bar{x}, \bar{x}\bar{x}^\top, t)$ is feasible for $(\overline{P}_{\mathcal{K}})$ with the same value t . Moreover, since all matrices $S_k \in \mathcal{K}$ are positive semi-definite, each constraint (9a) define a quadratic convex set, and so, $(\overline{P}_{\mathcal{K}})$ is a convex problem. Now, given an integer p , we consider the problem (LB_p) of determining the best set of matrices $\mathcal{K}^* = \{S_0, S_1, \dots, S_p\}$, in the sense of leading to the tightest lower bound of (QBP) :

$$(LB_p) \left\{ \begin{array}{l} \max \\ S_0, S_1, \dots, S_p \succeq \mathbf{0} \end{array} v(\overline{P}_{\mathcal{K}=\{S_0, S_1, \dots, S_p\}}) \right.$$

where $v(P)$ stands for the optimal value of problem (P) .

This connects our work with previous convex relaxations [24–26], since when we restrict $(\overline{P}_{\mathcal{K}})$ to $p = 1$ (i.e. $\mathcal{K} = \{S_0\}$), we obtain the original MIQCR method from [25]. It is proven in [26], that, the optimal solution of (LB_1) can be derived from the optimal dual solution of (SDP) from (4a)–(4f). It is, moreover, proven in [26] that, if strong duality holds for (SDP) , the optimal value of (LB_1) equals the optimal value of (SDP) which is always the case in a quadratic box-constrained problem. We now state Proposition 1

Proposition 1. $v(LB_\infty) = v(LB_p) = \dots = v(LB_1) = v(SDP)$.

Proof. The last equality (for $p = 1$) follows from [26, Theorem 3.1]. Moreover, by construction, we obviously have $v(LB_\infty) \geq v(LB_p) \geq \dots \geq v(LB_1) = v(SDP)$. We now show $v(LB_\infty) \leq v(SDP)$. Take a feasible solution (\bar{x}, \overline{X}) of (SDP) of objective value $f(\overline{X}, \bar{x}) = \langle Q, \overline{X} \rangle + c^\top \bar{x}$. The solution $(Y = \overline{X}, x = \bar{x})$ is feasible for $(\overline{P}_{\mathcal{K}})$ with a value of

$$\max_{k \in \mathcal{K}} \langle S_k, \bar{x}\bar{x}^\top - \overline{X} \rangle + c^\top \bar{x} + \langle Q, \overline{X} \rangle \leq f(\overline{X}, \bar{x})$$

since $\langle S_k, \bar{x}\bar{x}^\top - \overline{X} \rangle \leq 0$, which is true for any $S_k \succeq 0$ given that $\bar{x}\bar{x}^\top - \overline{X} \preceq 0$ by virtue of (4e). \square

Our idea in the rest of the paper is to use Proposition 1 to reinforce at each sub-node of the spatial branch-and-bound the value of the lower bound, if possible up to reaching the optimum (SDP) . Starting from any initial set \mathcal{K} , we introduce a cutting quadrics algorithm, which adds convex quadratic hyper-surfaces (which amounts to add positive semi-definite matrices to set \mathcal{K}) at each iteration. We will next present the Cutting Quadrics Algorithm (CQA) algorithm as well as its proof of convergence.

3 An algorithm for computing a tight quadratic convex relaxation

We now introduce our Cutting Quadrics Algorithm (CQA) that aims at solving (LB_∞) by extending the cutting-planes idea to the case of quadratic convex hyper-surfaces. Our idea is to start from an initial relaxation $(\overline{P}_{\mathcal{K}_0})$ associated to an initial set \mathcal{K}_0 of positive semi-definite matrices. We consider the assumption that the set \mathcal{K}_0 contains at least the null matrix (denoted by $\mathbf{0}_n$), but it can also be composed of any

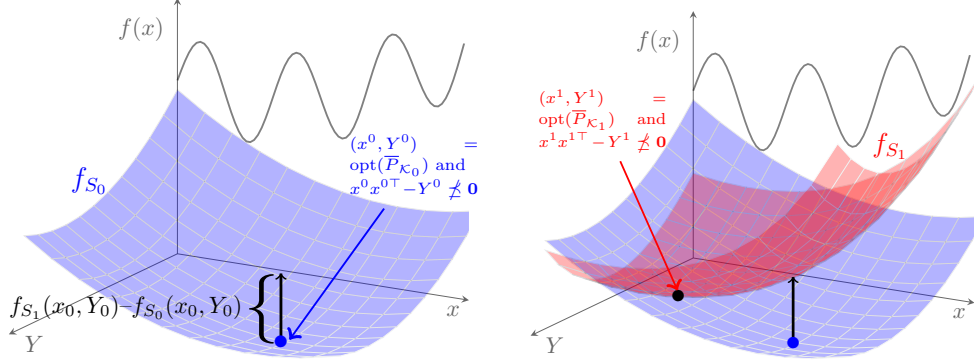


Fig. 2: The Cutting Quadratics Algorithm (CQA) starts from an initial convex function f_{S_0} (blue surface) of problem $(\bar{P}_{\mathcal{K}_0})$ whose optimal solution is (x^0, Y^0) , see the small blue disk. Then it generates matrix S_1 to make point (x^0, Y^0) reach a penalized higher objective value of $f_{S_1}(x^0, Y^0)$, as marked by the black vertical arrow in both figures. Thus, after adding function f_{S_1} (red surface) to $(\bar{P}_{\mathcal{K}_0})$, the optimal solution of $(\bar{P}_{\mathcal{K}_1})$ moves to (x^1, Y^1) .

other positive semi-definite matrices like for instance the best matrix S_0^* determined by MIQCR. At each iteration k , CQA calls a convex QCQP (quadratically constrained quadratic programming) solver to determine the optimal solution (x^k, Y^k, t^k) of $(\bar{P}_{\mathcal{K}_k})$, where t^k is its optimal value. It then constructs a matrix $S_{k+1} \succeq 0$ such that solution (x^k, Y^k, t^k) becomes sub-optimal for $(\bar{P}_{\mathcal{K}_{k+1}})$, *i.e.*, for the new program (9a)-(9c) enriched with the following convex quadric:

$$t \geq f_{S_{k+1}} = \langle S_{k+1}, xx^\top \rangle + c^\top x + \langle Q - S_{k+1}, Y \rangle$$

Let us focus on Figure 2 please. Each new iteration $k + 1$ aims at cutting solution (x^k, Y^k) when solving $(\bar{P}_{\mathcal{K}_{k+1}})$. We can say (x^k, Y^k) is separated in a cutting-planes fashion, but we use convex quadratic cuts instead of separating hyperplanes.

The key idea is to determine a new matrix S_{k+1} such that the additional quadric penalizes (very heavily) the optimal solution from the previous iteration (x^k, Y^k) . For a given $r^k > 0$, we take $S_{k+1} = r^k \cdot v_{max} v_{max}^\top$, where v_{max} is the eigenvector of matrix $(x^k x^{k\top} - Y^k)$ of maximum eigenvalue. With an appropriate setting of the parameter r^k , this choice ensures that solution (x^k, Y^k) will become sub-optimal in the new program $(\bar{P}_{\mathcal{K}_{k+1}})$, *i.e.*, defining S_{k+1} as above with a very large r^k will induce a prohibitively large penalty $\langle S_{k+1}, x^k x^{k\top} - Y^k \rangle$ on point (x^k, Y^k) . The choice of matrix S_{k+1} is not unique, and in practice, we can also define $S_{k+1} = r^k \sum v_i v_i^\top$, where the sum is carried out over all eigenvectors v_i having a positive eigenvalue. This will enable the new quadric to penalize larger areas of the (x, Y) space, *i.e.*, more solutions (x, Y) such that $\langle S_{k+1}, xx^\top - Y \rangle > 0$.

The overall solution method is summed up in Algorithm 1, and we prove its convergence in Theorem 1. Note that, as previously mentioned, we start with the assumption

that set \mathcal{K}_0 will at least integrate the null matrix $\mathbf{0}_n$, that corresponds to the standard linearization of f .

Algorithm 1: The Cutting-Quadratics Algorithm (CQA)

Input : variable bounds ℓ and u , initial penalty parameter r^0 , precision parameter δ , (optional) initial matrices \mathcal{K}_0

Output: The best lower bound on (LB_∞)

$\mathcal{K}_0 \leftarrow \mathcal{K}_0 \cup \{\mathbf{0}_n\}$ // \mathcal{K}_0 may contain the optimal matrix used by MIQCR

$(x^0, Y^0, t^0) \leftarrow \text{Solve}(\overline{P}_{\mathcal{K}_0})$ // variable t^0 is the optimum obj. value

$k \leftarrow 0$

while $(x^k x^{k\top} - Y^k \not\leq 0)$ // In practice we use $\lambda_{\max}(x^k x^{k\top} - Y^k) \geq \delta$

do

$r^k = r^0 + k$ // A penalty parameter;

$v_k \leftarrow$ the eigenvector of $x^k x^{k\top} - Y^k$ of maximum eigenvalue

$S_{k+1} \leftarrow r^k \cdot v_k v_k^\top$ // Or $r^k \cdot \sum v_i v_i^\top$, where the sum is carried out over all eigenvectors v_i having a positive eigenvalue

$\mathcal{K}_{k+1} = \mathcal{K}_k \cup S_{k+1}$

$(x^{k+1}, Y^{k+1}, t^{k+1}) \leftarrow \text{Solve}(\overline{P}_{\mathcal{K}_{k+1}})$

$k \leftarrow k + 1$

return t^k as the optimal solution of (QBP) if $Y^k = x^k x^{k\top}$, or as a lower bound otherwise

We now state Theorem 1 ensuring that our algorithm stops when t^k reaches the optimal value of (SDP).

Theorem 1. *When $k \rightarrow \infty$, the value of the solutions t^k generated by Algorithm 1 converge to the optimal value of (SDP).*

Proof. In order to prove that CQA converges to the optimal value of (SDP), i.e. that $t^{k^*} = v(\text{SDP})$, we decompose the proof in three steps:

- i) we first show that each intermediate optimal solution (x^k, Y^k, t^k) such that $\lambda_{\max}(x^k x^{k\top} - Y^k) > 0$ will be separated by CQA;
- ii) we prove that $\lambda_{\max}(x^k x^{k\top} - Y^k)$ converges to 0 as $k \rightarrow \infty$;
- iii) we finally prove that t^k reaches the optimal value of (SDP) when $k \rightarrow \infty$.

(i) Recall (x^k, Y^k, t^k) is the optimal solution to $(\overline{P}_{\mathcal{K}_k})$ of iteration k of objective value t^k . We first prove that the new quadric $f_{S_{k+1}}$ forces CQA to change the current optimal solution (x^k, Y^k, t^k) , or equivalently that the value of $(\overline{P}_{\mathcal{K}_{k+1}})$ at point (x^k, Y^k) is always greater or equal than t^k . Let v_k be the eigenvector of matrix $(x^k x^{k\top} - Y^k)$ of maximum eigenvalue λ_{\max} . By definition, we have $S_{k+1} = r^k \cdot v_k v_k^\top$ and we can develop:

$$\begin{aligned}
f_{S_{k+1}}(x^k, Y^k) - f_{S_k}(x^k, Y^k) &= \langle S_{k+1}, x^k x^{k\top} \rangle + c^\top x^k + \langle Q - S_{k+1}, Y^k \rangle \\
&\quad - \langle S_k, x^k x^{k\top} \rangle - c^\top x^k - \langle Q - S_k, Y^k \rangle \\
&= \langle S_{k+1}, x^k x^{k\top} - Y^k \rangle - \langle S_k, x^k x^{k\top} - Y^k \rangle \\
&= \langle r^k \cdot v_k v_k^\top, x^k x^{k\top} - Y^k \rangle - \langle r^{k-1} \cdot v_{k-1} v_{k-1}^\top, x^k x^{k\top} - Y^k \rangle \\
&= r^k \lambda_{max} - \langle r^{k-1} \cdot v_{k-1} v_{k-1}^\top, x^k x^{k\top} - Y^k \rangle \\
&> r^k \lambda_{max} - r^{k-1} \lambda_{max},
\end{aligned}$$

where the last inequality holds for any S_k of the form $r^{k-1} v_{k-1} v_{k-1}^\top$, based on the following well-known property [27, §3.2]:

$$\lambda_{max} = \max_{u \in \mathbb{R}^n, \|u\|=1} \langle uu^\top, x^k x^{k\top} - Y^k \rangle$$

This proves $f_{S_{k+1}}(x^k, Y^k) > f_{S_k}(x^k, Y^k) = t^k$, because $r^k > r^{k-1}$. Recall the first line of the **while** loop constructs $r^k = r^{k-1} + 1$ (and after another i iterations r^{k+i} is even higher) so that the penalty imposed on (x^k, Y^k) by the use of S_{k+1} forces CQA to move from (x^k, Y^k, t^k) to another solution.

We still need to show that $f_{S_{k+1}}(x^k, Y^k) > f_{S_k}(x^k, Y^k)$ also holds for the very first step, when $k = 0$ and S_k no longer has a form $r^{k-1} v_{k-1} v_{k-1}^\top$. The initial quadratics belong to a set \mathcal{K}_0 provided by the user. The inequality remains true using a high enough r_0 since

$$\begin{aligned}
f_{\mathcal{K}_0}(x^0, Y^0) &= \max_{S_i \in \mathcal{K}_0} \{ \langle S_i, x^0 x^{0\top} \rangle + c^\top x^0 + \langle Q - S_i, Y^0 \rangle \} \\
f_{\mathcal{K}_0}(x^0, Y^0) &< \langle r^0 \cdot v_0 v_0^\top, x^0 x^{0\top} \rangle + c^\top x^0 + \langle Q - r^0 \cdot v_0 v_0^\top, Y^0 \rangle, \tag{10}
\end{aligned}$$

where in the last inequality we used $\langle v_0 v_0^\top, x^0 x^{0\top} - Y^0 \rangle > 0$ which is true because vector v_0 corresponds to the maximum eigenvalue of $x^0 x^{0\top} - Y^0$ which is considered positive here. Still, depending on the initial set \mathcal{K}_0 , we may need a very large initial r^0 (in theory).

(ii) We will now prove that for any $\delta > 0$ no matter how small, there exist some \bar{k} such that $\lambda_{max}(x^k x^{k\top} - Y^k) < \delta$ for any $k \geq \bar{k}$.

Suppose for the sake of contradiction that this is not the case. This means that the algorithm generates an infinite number of matrices $(x^k x^{k\top} - Y^k) \in \mathcal{S}_n$ that all satisfy $\lambda_{max}(x^k x^{k\top} - Y^k) \geq \delta$. The Bolzano-Weierstrass theorem states that any infinite sequence in a bounded set (recall that (x, Y) belong to the unit hypercube) contains a convergent subsequence. This means that there exists a subsequence (k_i) such that $(x^{k_i} x^{k_i\top} - Y^{k_i})$ converges to some fixed point $\bar{x} \bar{x}^\top - \bar{Y}$ when $i \rightarrow \infty$. By assumption, this convergence point satisfies $\lambda_{max}(\bar{x} \bar{x}^\top - \bar{Y}) \geq \delta$.

Let us study how the above subsequence (k_i) stays in a box very close to (\bar{x}, \bar{X}) . For any infinitesimal ϵ , there exists some k_ϵ such that for any $k_i > k_\epsilon$ in the above subsequence, any element in matrix $(x^{k_i} x^{k_i\top} - Y^{k_i})$ is at a distance smaller than ϵ from

the corresponding element of $(\bar{x}\bar{x}^\top - \bar{Y})$, *i.e.*, it stays in a neighborhood N_ϵ of $(\bar{x}\bar{x}^\top - \bar{Y})$ of ∞ -norm below ϵ . Moreover, since λ_{max} is a continuous function, we also have for a sufficiently small ϵ , $\lambda_{max}(xx^\top - Y) > \delta' > 0 \forall (x, Y) \in N_\epsilon$ for some δ' arbitrarily close to δ . All points $k_i > k_\epsilon$ in above subsequence will satisfy $\lambda_{max}(x^{k_i}x^{k_i^\top} - Y^{k_i}) > \delta'$.

But here comes the contradiction : for a sufficiently large \hat{k}_i , we have $r^{\hat{k}_i} = r^0 + \hat{k}_i$ so that $f_{S_{\hat{k}_i}}(x, Y)$ will include a penalty of at least $r^{\hat{k}_i}\delta'$ that can become large enough to make any $(x, Y) \in N_\epsilon$ sub-optimal. In other words, at some iteration \hat{k}_i , the weight $r^{\hat{k}_i}$ will become large-enough to cut the whole box, since the associated penalty $r^{\hat{k}_i}\lambda_{max}(x^{\hat{k}_i}x^{\hat{k}_i^\top} - Y^{\hat{k}_i}) > r^{\hat{k}_i}\delta'$ is very large. Thus, no iteration after \hat{k}_i of above subsequence can stay in N_ϵ and this is a contradiction.

(iii) We now prove that $t^k \rightarrow v(SDP)$.

Let us first take any convergence point $(\bar{x}, \bar{Y}, \bar{t})$ of Algorithm 1 that has to satisfy $\lambda_{max}(\bar{x}\bar{x}^\top - \bar{Y}) \leq 0$ according to point (ii) above. This means $\bar{x}\bar{x}^\top - \bar{Y} \preceq 0$. We will use this \bar{x} and \bar{Y} to build a feasible solution of (SDP) such that $\bar{t} \geq v(SDP)$. Since $(\bar{x}\bar{x}^\top - \bar{Y}) \preceq 0$, the solution $(x = \bar{x}, X = \bar{Y})$ is feasible for (SDP) with a value of $c^\top \bar{x} + \langle Q, \bar{Y} \rangle \geq v(SDP)$. The optimal value \bar{t} can be written as below (regardless of how exactly \mathcal{K} was constructed):

$$\bar{t} = \max_{S \in \mathcal{K}} \langle S, \bar{x}\bar{x}^\top - \bar{Y} \rangle + c^\top \bar{x} + \langle Q, \bar{Y} \rangle = c^\top \bar{x} + \langle Q, \bar{Y} \rangle$$

The last equality holds since $\max_{S \in \mathcal{K}} \langle S, \bar{x}\bar{x}^\top - \bar{Y} \rangle = \langle \mathbf{0}_n, \bar{x}\bar{x}^\top - \bar{Y} \rangle = 0$ which is true because $\mathbf{0}_0 \in \mathcal{K}_0$ and because $\langle S, \bar{x}\bar{x}^\top - \bar{Y} \rangle \leq 0$ for any positive semi-definite matrix $S \in \mathcal{K}$.

Since (\bar{x}, \bar{Y}) is feasible for (SDP) , this proves that $\bar{t} \geq v(SDP)$.

We now show $\bar{t} \leq v(SDP)$. Take the optimal solution (\tilde{x}, \tilde{Y}) of (SDP) and notice that it is feasible for $(\bar{P}_{\mathcal{K}_k})$, *i.e.*, for program (9a)-(9c) using any quadratics $\mathcal{K} = \mathcal{K}_k$ constructed by the algorithm. The value of \bar{t} has to be lower or equal to the objective value of (\tilde{x}, \tilde{Y}) in $(\bar{P}_{\mathcal{K}_k})$ which is

$$\max_{S \in \mathcal{K}_k} \langle S, \tilde{x}\tilde{x}^\top - \tilde{Y} \rangle + c^\top \tilde{x} + \langle Q, \tilde{Y} \rangle = \max_{S \in \mathcal{K}_k} \langle S, \tilde{x}\tilde{x}^\top - \tilde{Y} \rangle + v(SDP) \leq v(SDP),$$

where we simply used $\langle S, \tilde{x}\tilde{x}^\top - \tilde{Y} \rangle \leq 0$. This follows from $S \succeq \mathbf{0}$ and $\tilde{x}\tilde{x}^\top - \tilde{Y} \preceq \mathbf{0}$, which holds by virtue of (4e). \square

The proof of Theorem 1 implies the following Corollary.

Corollary 1. *Algorithm 1 solves problem (SDP) . In particular, if $(\bar{x}, \bar{Y}, \bar{t})$ is a convergence point of Algorithm 1, (\bar{Y}, \bar{x}) is an optimal solution to (SDP) with a value of $\bar{t} = v(SDP)$.*

Theorem 1 thus states that Algorithm 1 computes a quadratic convex relaxation that reaches the value of (SDP) . Within a branch and bound framework, we obtain the same lower bound as MIQCR at the root node but we may obtain improved lower bounds at the subnodes discovered by branching. Moreover, as mentioned previously,

the additional quadrics may tighten the convexification by reducing the set of optimal equivalent solutions at each sub-node (like in Figure 1). This increases the potential of the new convexification when integrated within a spatial branch-and-bound algorithm. The numerical experiments will show that the total number of nodes may be halved even if Algorithm 1 only produces very few quadrics (few iterations of the **while** loop).

Recall that Algorithm 1 can start from any set \mathcal{K}_0 . To improve its convergence, we describe hereafter several ways to populate \mathcal{K}_0 , whose size is not limited. The first one consists of adding to \mathcal{K}_0 the optimal matrix calculated by MIQCR, which is $S_0^* = Q + \Phi^1 + \Phi^2 - \Phi^3 - \Phi^4$, where $\Phi^1, \Phi^2, \Phi^3, \Phi^4$ are the symmetric matrices built from the optimal dual variables associated with the McCormick constraints (4a)–(4d) of (SDP). While this implies solving a large SDP problem at the root node, it can reduce the number of CQA iterations. Another choice relies on extracting the convex part of Q , by constructing the matrix

$$S_0^+ = \sum \lambda_i v_i v_i^T \quad (11)$$

where the sum is carried over all non-negative eigenvalues λ_i of Q associated to eigenvectors v_i . Preliminary tests show that integrating these constraints is a good choice for all CQA variants.

4 The spatial branch-and-bound to optimally solve (QBP)

We are now interested in computing the optimal solution of (QBP). A classical way is to use a branch-and-bound algorithm (see [28] for a complete description), where the lower bound is computed at each node using the new CQA algorithm. We proved in Section 3 that CQA converges to an optimal solution of (SDP), but it is important that each iteration of CQA provides a valid lower bound of (QBP). Moreover, since each CQA iteration solves a computationally-expensive convex optimization problem ($\overline{P}_{\mathcal{K}_k}$), we need to find the best trade-off between the tightness of the convexification and its computation time. Thus, in our numerical experiments, we will only consider CQA variants that perform relatively few iterations at each branch-and-bound node.

We first provide in Section 4.1 below the general framework of the branch-and-bound. We then present in Section 4.2 two techniques to speed up the solution of ($\overline{P}_{\mathcal{K}}$), because the calls to the convex QCQP solver that optimizes ($\overline{P}_{\mathcal{K}}$) represent the main computational bottleneck of the overall method. Finally, we describe our upper bounding procedure in Section 4.3.

4.1 The dynamics of the branch-and-bound tree construction

We consider a branching tree that contains only fully evaluated nodes, meaning that the lower and upper bounds of each node are known at all times. Each lower bound is determined by calling CQA and each upper bound is determined using the heuristic

described in Section 4.3. A branching decision is automatically taken at the end of the node evaluation, unless the node is pruned. However, there are two types of nodes:

- A node is closed if the branching decision was implemented and the node produced two evaluated child nodes.
- A node is open or childless if, although evaluated, its branching decision was not yet implemented to produce child nodes.

Each branch-and-bound step performs the following: (i) seek an open node; (ii) implement the branching decisions determined when the parent node was evaluated (at a previous moment) and (iii) evaluate the two resulting child nodes. The selected parent node is thus no longer childless and is marked closed. Its two child nodes are considered open. After evaluating the child nodes, we take right away a branching decision that can be implemented later.

4.1.1 How many iterations per node

Each evaluation of the lower bound is performed from scratch, starting from the same \mathcal{K}_0 . We did try to enable certain nodes to inherit quadratic cuts from some ancestor nodes, but the observed speed-up is not large enough to warrant complicating the overall branch-and-bound. The most important decision for making CQA reach its full potential in practice is related to the number of iterations that should be executed on each branch-and-bound node. More iterations means fewer nodes in the general branching tree, but this comes at the cost of needing more computational work per node.

We thus consider two different stopping criteria for CQA. Since the algorithm is supposed to stop when matrix $(x^k x^{k\top} - Y^k) \preceq 0$, we replace this condition with $\lambda_{\max}(x^k x^{k\top} - Y^k) > \delta$. Thus, by varying the value δ , we can choose to make an additional iteration only if matrix $(x^k x^{k\top} - Y^k)$ is far to be SDN, *i.e.*, if its largest eigenvalue can be considered large enough. Our second stopping criteria is simply a bound on the number of iterations of the **while** loop in Algorithm 1.

4.1.2 The branching decision

The variable selection strategy is as follows. Let $\epsilon_b > 0$ be a very small precision parameter for considering an imperfect equality as satisfied. Denoting the solution of $(\overline{P}_{\mathcal{K}_k})$ at the current node by (x^k, Y^k) , two cases are possible:

1. If $-\epsilon_b \leq (Y - xx^\top)_{ij} \leq \epsilon_b$ for all $(i, j) \in \mathcal{I}^2$, then (x, Y) is the optimal solution of the considered branch.
2. Else, we determine i^* by only looking for the diagonal values so that $x_i^2 \neq Y_{ii}$. We first restrict to variables i with a decent range $u_i - \ell_i$ above a given threshold γ fixed to 0.1 in the beginning. We thus solve

$$i^* = \operatorname{argmax}_{u_i - \ell_i > \gamma} |x_i^2 - Y_{ii}|$$

If this scanning finds no satisfactory solution, we divide the above parameter γ by 3. And the process is repeated until we find a satisfactory variable i to be split.

The feasible interval $[\ell_{i^*}, u_{i^*}]$ of the selected variable x_{i^*} is always split in half: the two child nodes are $x_{i^*} \in \left[\ell_{i^*}, \frac{\ell_{i^*} + u_{i^*}}{2} \right]$ and $x_{i^*} \in \left[\frac{\ell_{i^*} + u_{i^*}}{2}, u_{i^*} \right]$.

Regarding the selection of the next sub-problem to solve, we use the “best-first” strategy, in the sense that we select the node with the highest evaluated lower bound. If this lower bound is higher than the current best-known upper bound, the node is pruned. Otherwise, we implement the branching decision and evaluate the child nodes as described in Section 4.1.

4.2 Using boxes to speed-up the QCQP solver that iteratively optimizes the programs $(\overline{P}_{\mathcal{K}_k})$

We now present a technique that may speed-up the convex solver that is called iteratively at each sub-node to optimize $(\overline{P}_{\mathcal{K}_k})$, because this operation is the main computational bottleneck of the overall method. The solvers we are aware of can not take advantage of the fact that program $(\overline{P}_{\mathcal{K}_{k+1}})$ is simply program $(\overline{P}_{\mathcal{K}_k})$ enriched with a new quadratic constraint. The Simplex algorithm may simply perform a unique pivoting step when a new linear constraint is added, but – far from such a re-optimization potential – our QCQP solver restarts from scratch when a new constraint is added. Any future research or improvement in QCQP re-optimization may speed-up our algorithm.

Generally, our speed-up idea relies on restricting the allowed variation of certain variables, by putting a box around them. If the optimal solution of the resulting overly-restricted convex program is strictly inside the box, then this is also the optimal solution of the initial relaxation. Otherwise, we remove the artificial box and the convex solver may have to be called a second time.

4.2.1 Boxes in relation to father and ancestor nodes

The optimal solution in a child node is sometimes not fundamentally different from that of the father node, in the sense that many variables stay fixed in both the current and the father nodes. To speed up the first call to the convex solver at each node, we restrict the decision variables as follows. We first identify the variables that never changed in any ancestor node on the genealogical lineage upwards for β levels, where β is a parameter. We first solve the node by strongly limiting the variation of these variables (we almost fix them) compared to their value in the father node. If the optimal solution is within the box we succeed in speeding up this optimization stage. Otherwise, we failed because we have to call the convex solver a second time after removing the box. In many cases, there are multiple optimal solutions. If at least some of them stay inside the box, this technique will succeed.

4.2.2 Boxes acting on the iterations of CQA

While the above technique is applied before starting the **while** loop of Algorithm 1, we now present a technique to be applied inside this **while** loop. We build the following

box. First, we compute the maximum variation x^{Max} (or Y^{Max} respectively) over all optimal x (or Y , resp.) values observed during the convex solver calls from previous iterations (including at the QCQP call before the **while** loop). We then limit the variation of variables x and Y within a box of radius x^{Max} and Y^{Max} , respectively. If the obtained optimal solution stays inside the box, we succeeded in speeding-up the QCQP convex solver. Otherwise, we failed and we do not consider the reported solution and node lower bound reliable. In the latter case, we simply do not update the evaluated lower bound of the considered node; yet, we do not call the convex solver a second time.

4.3 A heuristic for computing upper bounds

We use a rather basic coordinate descent heuristic to determine upper bounds. We start from the solution x^k of $(\overline{P}_{\mathcal{K}_k})$ at the current node (ignoring all Y^k components), that is also feasible for the original (QBP). Our coordinate descent begins by iterating over all $i \in [1..n]$; for each i , we fix all variables to their current values except for x_i^k . We then solve an optimization sub-problem with only one decision variable, namely x_i^k . This requires minimizing a simple quadratic function in dimension one that can be determined using elementary mathematics, regardless of its convexity status. We decided to keep x_i in its range $[l_i, u_i]$, but this could be relaxed. After scanning all variables once, we can repeat the process as long as there is at least one variable i for which we detect a possible improvement.

5 Numerical results

To evaluate our new solution method hereafter called **Cutting-Quadric Branch-and-Bound** (CQBB), we use two variants of CQA for computing the lower bound at each node of the branch-and-bound tree, leading to methods CQBB-1 and CQBB-2.

- CQBB-1: the stopping criteria of CQA is $\lambda_{\max}(x^k x^{k\top} - Y^k) > \delta$, with $\delta = 0.8$. Additionally, we limit for each node the maximum number of iterations of the **while** loop to 3.
- CQBB-2: We perform exactly one iteration of the **while** loop of CQA, taking $\delta = 0$.

We will compare these two methods with the solvers **Cplex 12.9** [29], **Baron 21.1.13** [17], and **Gurobi 9.1.1** [20], as well as with the original algorithm **MIQCR** [26]. In fact, we will use a new implementation of **MIQCR**, with a new branching strategy, a different upper bound heuristic and with many other updated parameters.

We run our experiments on the *boxqp* instances [3, 4], that consist in minimizing a continuous quadratic function within box constraints. The sizes of the instances vary from $n = 20$ to 125 and the densities (% of non-zero elements) of matrix Q from 20% to 100%.

Both CQBB-1 and CQBB-2 start from the initial set $\mathcal{K}_0 = \{S_0^+, S_0^*, \mathbf{0}\}$, where S_0^+ is given by (11) and S_0^* is the optimal matrix of method **MIQCR**. It is obtained by solving (*SDP*) heuristically by calling the solver **Mosek** [30] together with the **Conic Bundle**

library [31] within a Lagrangian duality framework as described in [32]. We only use $r^0 = 1000$ to compute all S_k with $k \geq 1$, a value that proved more than sufficient to make inequality (10) hold. The QCQP solver for optimizing $(\overline{P}_{\mathcal{K}_k})$ at each CQA iteration calls the Mosek solver using the C interface. We define $S_{k+1} = r^k \sum v_i v_i^T$, summing over all eigenvectors v_i associated to an eigenvalue no smaller than 0.01. We computed all eigenvalues and eigenvectors using the C++ Eigen library.

Regarding the parameters of the boxes in relation to ancestor nodes (described in Sections 4.2.1), we use $\beta = 3$ for instances with up to 90 variables (*i.e.*, we restrict the variables that did not change for the father, grandfather or great-grandfather node), and $\beta = 12$ for the instances with $n \geq 100$.

We set the time limit to 1 hour for the instances with up to 90 variables and to 3 hours for the largest ones (for all compared methods); the relative optimality gap of the branch-and-bound to $\epsilon_b = 10^{-4}$.

We describe hereafter the settings of the compared solvers:

- For MIQCR, we solve (*SDP*) heuristically using the Conic Bundle library as described above for algorithm CQBB.
- For the solvers Cplex 12.9 [29] and Gurobi 9.1.1 [20], we use the AMPL [33] interface and the default parameters.
- For the solver Baron 21.1.13 [17], we use the Gams [34] interface and the default parameters.

We run our experiences on a cluster under Open Suze Linux with 2 CPU Intel Xeon of 2.3 GHz, each of them having 32 threads.

5.1 Comparing CQBB-1 and CQBB-2 with the starting point MIQCR

Tables 1-3 report the results of methods MIQCR, CQBB-1 and CQBB-2 on small instances ($20 \leq n \leq 40$), medium-sized instances ($50 \leq n \leq 90$) and respectively large instances ($n \geq 100$). For each of these three tables, the column “*Instance*” represents the instance under the name $n-d-k$, where n is the number of variables, d is the density in percents, and k is a numeration label. The column “*Nodes*” report the total number of nodes needed by the branch-and-bound; Column “*CPU*” provides the CPU time (in seconds) required for solving the instance to global optimality; a “_” symbol means that the instance was not solved within the indicated time limit. Finally, the columns “*#calls*” indicate the total number of calls to the QCQP solver.

The general rankings of the 3 algorithms are the following:

number of nodes: 1) CQBB-2; 2) CQBB-1; 3) MIQCR.

cpu time: 1) CQBB-1; 2) CQBB-2; 3) MIQCR.

More precisely, we observe that MIQCR develops 34% more nodes than CQBB-1 on average over all instances (solved within the time limit), and the number of nodes is further significantly reduced by method CQBB-2 since MIQCR requires 74% more nodes than CQBB-2 on average. As such, CQBB-2 was able to reduce the number of nodes by half (or more) for almost a third of the small and medium instances. The

Table 1: The main results of the two considered CQA versions against the original MIQCR on the smallest instances

instance	MIQCR			CQBB-1			CQBB-2		
	nodes	/time	#calls	nodes	/time	#calls	nodes	/time	#calls
020-100-1	9	/0.2	9	3	/0.0	3	3	/0.0	4
020-100-2	13	/0.2	13	9	/0.1	9	9	/0.1	13
020-100-3	5	/0.1	5	3	/0.0	3	3	/0.0	4
030-060-1	33	/1.0	33	31	/0.5	31	27	/1.2	48
030-060-2	1	/0.0	1	1	/0.0	1	1	/0.0	1
030-060-3	29	/0.6	29	19	/0.3	19	19	/0.6	32
030-070-1	71	/1.9	71	63	/1.0	66	65	/2.2	105
030-070-2	5	/0.1	5	3	/0.1	3	3	/0.1	4
030-070-3	41	/0.9	41	37	/0.6	41	17	/0.5	27
030-080-1	69	/2.0	69	67	/1.4	100	73	/3.0	155
030-080-2	3	/0.1	3	5	/0.1	5	5	/0.2	7
030-080-3	13	/0.3	13	7	/0.1	7	9	/0.3	14
030-090-1	7	/0.2	7	3	/0.1	3	3	/0.1	4
030-090-2	13	/0.3	13	11	/0.3	17	3	/0.1	5
030-090-3	3	/0.1	3	3	/0.1	3	3	/0.1	4
030-100-1	11	/0.3	11	9	/0.2	9	7	/0.2	11
030-100-2	9	/0.3	9	9	/0.2	9	7	/0.3	11
030-100-3	27	/0.6	27	21	/0.4	23	13	/0.4	20
040-030-1	7	/0.3	7	3	/0.1	3	3	/0.1	4
040-030-2	13	/0.5	13	5	/0.2	5	5	/0.3	7
040-030-3	5	/0.2	5	3	/0.1	3	3	/0.2	4
040-040-1	445	/15	445	263	/7	275	249	/12	416
040-040-2	7	/0.3	7	5	/0.2	5	5	/0.3	7
040-040-3	27	/1.0	27	23	/0.7	25	23	/1.3	38
040-050-1	29	/1.1	29	25	/0.7	25	25	/1.3	38
040-050-2	43	/1.6	43	35	/1.2	43	25	/1.5	42
040-050-3	25	/1.0	25	17	/0.6	21	11	/0.6	17
040-060-1	279	/11	279	219	/7	251	209	/11	373
040-060-2	21	/0.9	21	13	/0.6	19	9	/0.5	15
040-060-3	7	/0.4	7	5	/0.2	5	5	/0.4	8
040-070-1	7	/0.3	7	5	/0.2	5	5	/0.3	7
040-070-2	7	/0.3	7	7	/0.2	7	5	/0.4	8
040-070-3	9	/0.4	9	5	/0.2	5	5	/0.4	8
040-080-1	5	/0.2	5	5	/0.2	5	5	/0.3	7
040-080-2	5	/0.3	5	7	/0.3	9	5	/0.4	8
040-080-3	13	/0.5	13	15	/0.5	15	11	/0.7	18
040-090-1	9	/0.4	9	5	/0.2	5	5	/0.3	7
040-090-2	9	/0.4	9	9	/0.3	9	9	/0.5	13
040-090-3	7	/0.3	7	5	/0.2	5	5	/0.3	7
040-100-1	13	/0.6	13	7	/0.2	7	7	/0.4	13
040-100-2	21	/1.8	21	17	/0.5	17	17	/1.0	27
040-100-3	221	/18	221	207	/6.3	245	177	/10	323

most spectacular improvement is visible on instance 080-25-1: CQBB-2 reduced the number of nodes from 391 to 23.

Regarding the total CPU time, we observe that MIQCR is 56% slower than CQBB-1 on average over all the instances (solved within the time limit), and 15% slower than

Table 2: The main results of the two considered CQA variants against the original MIQCR on medium-size instances

instance	MIQCR			CQBB-1			CQBB-2		
	nodes	/ time	#calls	nodes	/ time	#calls	nodes	/ time	#calls
050-030-1	13	/ 0.8	13	9	/ 0.5	11	7	/ 0.8	12
050-030-2	19	/ 1.1	19	17	/ 0.9	19	15	/ 1.4	24
050-030-3	31	/ 1.6	31	27	/ 1.4	35	13	/ 1.2	20
050-040-1	9	/ 0.7	9	9	/ 0.6	12	5	/ 0.6	8
050-040-2	43	/ 2.5	43	31	/ 1.3	31	25	/ 2.1	40
050-040-3	7	/ 0.5	7	7	/ 0.4	7	5	/ 0.6	8
050-050-1	5207	/ 381	5207	4549	/ 218	5088	4447	/ 455	8286
050-050-2	65	/ 3.7	65	67	/ 3.2	77	53	/ 4.5	88
050-050-3	155	/ 8	155	121	/ 7	180	63	/ 5.8	114
060-020-1	13	/ 1.2	13	5	/ 0.5	5	5	/ 0.8	7
060-020-2	11	/ 1.0	11	5	/ 0.5	5	5	/ 0.8	7
060-020-3	61	/ 4.7	61	33	/ 2.1	33	33	/ 4.2	54
070-025-1	77	/ 8	77	41	/ 3.7	41	33	/ 6.5	57
070-025-2	199	/ 22	199	129	/ 11	133	109	/ 21	209
070-025-3	225	/ 25	225	145	/ 15	168	105	/ 17	174
070-050-1	183	/ 21	183	159	/ 17	184	145	/ 26	255
070-050-2	29	/ 3.6	29	25	/ 3.0	31	23	/ 4.7	40
070-050-3	9	/ 1.4	9	11	/ 1.6	15	7	/ 1.7	11
070-075-1	65	/ 7	65	59	/ 6.1	63	55	/ 11	108
070-075-2	1735	/ 203	1735	1435	/ 161	1750	1401	/ 263	2592
070-075-3	819	/ 96	819	729	/ 86	990	585	/ 113	1148
080-025-1	391	/ 61	391	227	/ 41	412	23	/ 8	53
080-025-2	559	/ 88	559	433	/ 62	487	405	/ 114	796
080-025-3	161	/ 25	161	95	/ 12	99	77	/ 19	131
080-050-1	21543	/ 3532	21543	17953	/ 2631	21255	12597	/ timeout	0
080-050-2	65	/ 10	65	77	/ 15	137	21	/ 6.8	43
080-050-3	327	/ 56	327	293	/ 48	361	279	/ 87	578
080-075-1	113	/ 18	113	101	/ 16	127	103	/ 30	214
080-075-2	333	/ 52	333	309	/ 51	390	301	/ 82	579
080-075-3	1149	/ 183	1149	1049	/ 168	1301	1075	/ 294	2115
090-025-1	3243	/ 668	3243	2141	/ 418	2552	2149	/ 728	3985
090-025-2	2683	/ 561	2683	1703	/ 350	2038	1567	/ 516	2780
090-025-3	623	/ 126	623	405	/ 77	472	391	/ 124	684
090-050-1	1707	/ 387	1707	1413	/ 321	1783	1419	/ 550	2885
090-050-2	21	/ 5.3	21	19	/ 4.2	22	15	/ 7.0	27
090-050-3	493	/ 113	493	389	/ 88	495	393	/ 154	806
090-075-1	6037	/ 1209	6037	5919	/ 1347	7746	6387	/ 2510	13509
090-075-2	5947	/ 1234	5947	5451	/ 1206	6831	5371	/ 2005	10611
090-075-3	1353	/ 282	1353	1251	/ 298	1677	1231	/ 487	2605

CQBB-2. In fact, despite a reduced number of nodes, the time spent at each node penalizes the overall branch-and-bound of algorithm CQBB-2. Clearly, the stopping criteria of CQBB-1 that rely on the accuracy of the positivity of the maximum eigenvalue seems to be a good trade-off between bound quality and CPU time.

Finally, it may be interesting to comment on the number of calls to the QCQP solver (Column *#calls*). As expected, method MIQCR performs exactly one call per node. The number of calls for our new approaches is not fixed and depends on the value of the solution (x^0, Y^0) computed before the **while** loop of CQA (Algorithm 1).

Table 3: The main results of the two considered CQA variants against the original MIQCR on the largest instances

instance	MIQCR			CQBB-1			CQBB-2					
	nodes	time	#calls	nodes	time	#calls	nodes	time	#calls			
100-025-1	665	/	182	665	505	/	121	518	429	/	209	776
100-025-2	315	/	85	315	261	/	65	292	191	/	85	323
100-025-3	269	/	75	269	191	/	45	195	199	/	90	343
100-050-1	63553	/	timeout	0	60229	/	timeout	0	33709	/	timeout	0
100-050-2	25421	/	7347	25421	23855	/	6657	27153	20433	/	timeout	0
100-050-3	811	/	235	811	755	/	190	813	663	/	309	1225
100-075-1	12001	/	3480	12001	11929	/	3094	13170	11367	/	5092	20731
100-075-2	9479	/	2712	9479	9517	/	2557	10694	9077	/	4302	17313
100-075-3	8867	/	2509	8867	8829	/	2269	9575	8521	/	3913	16017
125-025-1	36504	/	timeout	0	32189	/	timeout	0	18271	/	timeout	0
125-025-2	7249	/	3745	7249	5629	/	2570	6144	4769	/	4033	8372
125-025-3	1733	/	895	1733	1365	/	625	1419	1275	/	1024	2088
125-050-1	38318	/	timeout	0	30221	/	timeout	0	17973	/	timeout	0
125-050-2	21909	/	timeout	0	17530	/	timeout	0	10269	/	timeout	0
125-050-3	21173	/	timeout	0	17723	/	timeout	0	10129	/	timeout	0
125-075-1	4841	/	2483	4841	4757	/	2949	7293	4763	/	4508	10748
125-075-2	20787	/	timeout	0	21021	/	timeout	0	12333	/	timeout	0
125-075-3	20605	/	timeout	0	16834	/	timeout	0	10265	/	timeout	0

Simply computing this (x^0, Y^0) requires a first call to the QCQP solver. CQBB-1 can call the QCQP solver up to 3 additional times because it performs at most 3 iterations of the **while** loop. CQBB-2 always performs exactly one iteration, leading to 2 calls to the QCQP solver (one before the **while** and one after). However, sometimes the **while** loop may be skipped completely, either because $\lambda_{\max}(x^0 x^{0\top} - Y^0) \leq \delta$, or because the lower bound associated to (x^0, Y^0) is good enough to prune the node. As such, we do not systematically have 2 QCQP solver calls per iteration in the case of CQBB-2.

5.2 General comparison with existing solvers

We now compare methods CQBB-1, CQBB-2 and MIQCR [26] with the standard solvers Cplex 12.9 [29], Baron 21.1.13 [17], and Gurobi 9.1.1 [20]. For this, we use a performance profile of the CPU times (see [35] for a complete description). The basic idea is the following: for each instance i and each solver s , we denote by t_{is} the time for solving instance i by solver s , and we define the *performance ratio* as $r_{is} = \frac{t_{is}}{\min_s t_{is}}$. Let N be the total number of instances considered;

an overall assessment of the performance of solver s for a given τ is given by: $P(r_{is} \leq \tau) = \frac{1}{N} * \text{number of instances } i \text{ such that } r_{is} \leq \tau$.

Thus, in a performance profile of CPU times, each curve corresponds to a solver, where each point of a curve gives, for a given factor τ , the percentage of instances whose CPU time was at most τ times greater than the minimal CPU time reported by any of all the solvers. In particular, for $\tau = 1$, we have the proportion of instances for which the considered solver was the fastest method.

We present the performance profiles of the CPU times for all compared algorithms on the *boxqp* instances with up to 90 variables in Figure 3, and on the largest instances

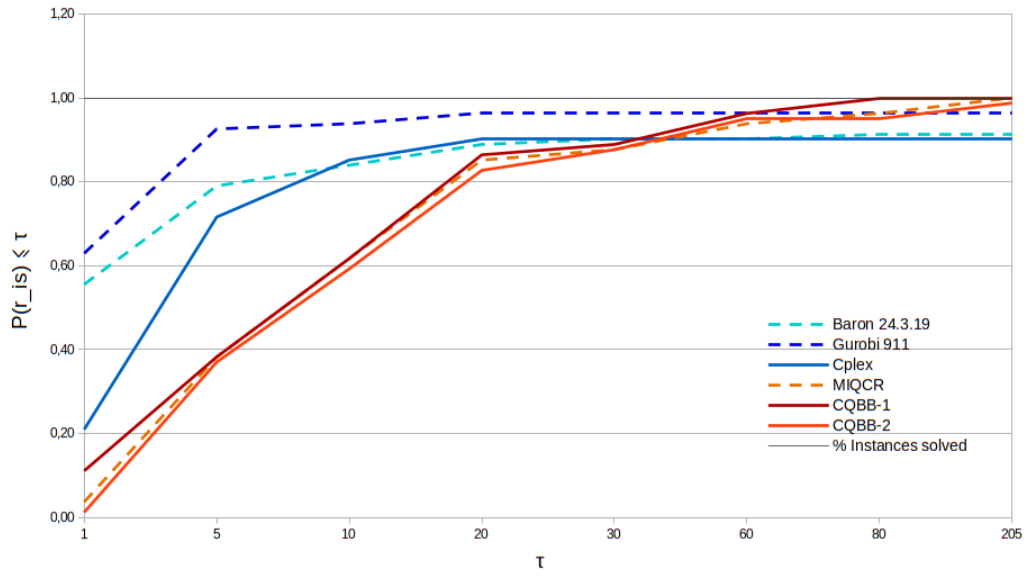


Fig. 3: Performance profile of the total CPU time for the *boxqp* instances with $n = 20$ to 90 within a time limit of 1 hour.

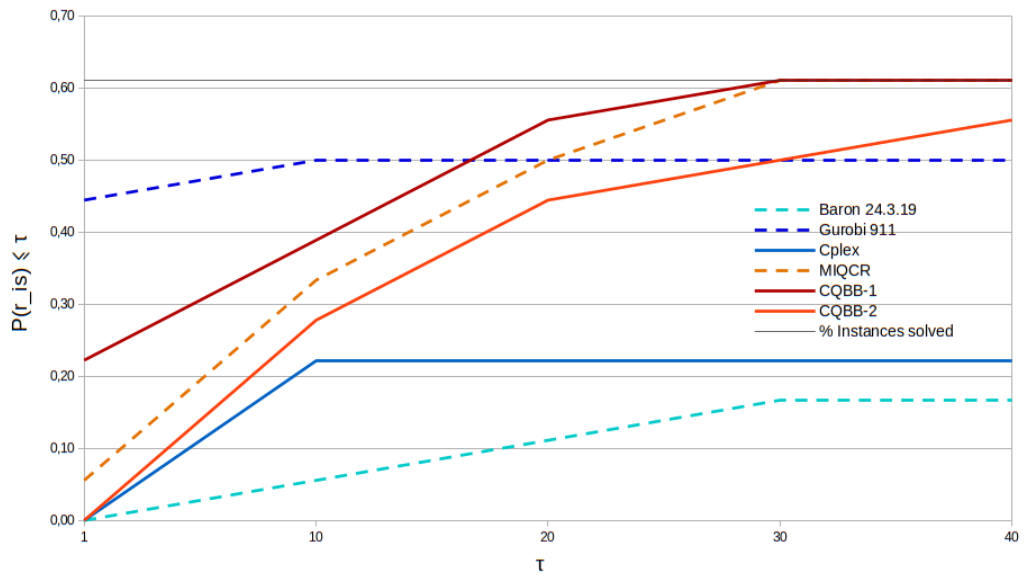


Fig. 4: Performance profile of the total CPU time for the *boxqp* instances with $n \geq 100$ within a time limit of 3 hours

(with $n \geq 100$) in Figure 4. We observe that our new approach **CQBB-1** compares well with **MIQCR** and all standard solvers in terms of the number of instances solved (maximum τ). For the small and medium-sized instances (Figure 3), **Cplex** solves 73 instances, **Baron** 74 solves instances, **Gurobi** solves 78 instances, **CQBB-1** solves 80 instances, and **MIQCR** and **CQBB-1** solve 81 instances out of 81 (within the time limit of 1 hour). However, note that the performance profile starts for $\tau = 1$ with a smaller value $P(r_{is} \leq \tau)$ for **CQBB-1** than for the commercial solvers. This is because the smallest instances are solved more rapidly by **Gurobi**, **Cplex** or **Baron**. However, the focus of our algorithm is on the largest and denser instances, and, with regards to them, Figure 4 shows that the (red) curve of **CQBB-1** is above all other curves except over the segment $\tau < 15$ where it only dominated by **Gurobi**. On the long run ($\tau > 15$), the winner is **CQBB-1** followed by **MIQCR**.

Considering the maximum time limit of three hours (maximum τ), **CQBB-1** and **MIQCR** both solve 11 instances but the curve of **MIQCR** is globally above that of **MIQCR** in Figure 4. For the same time limit, **CQBB-2** solves 10 instances, **Gurobi** 9 instances, **Cplex** 4 instances and **Baron** 3 instances.

We also noticed that the average final gap of **CQBB-1** over the larger instances unsolved within the time limit is roughly 1%, while it is around 33% for **Gurobi**, 15% for **Cplex** and 10% for **Baron**. This can be explained by the tightness of the SDP convexification computed at the root node by **CQBB-1**.

6 Conclusions

We have presented a generic approach to solve box-constrained quadratic programs to global optimality. The main idea is to combine the strength of quadratic convex relaxations with the cutting-planes logic. Indeed, instead of considering a unique convex quadric to underestimate the objective function, we propose a family of convexifications indexed by an arbitrary number of quadrics.

We have proposed the idea to add these cutting-quadrics one by one, in a cutting-planes fashion. The quadric generated at each iteration is actually a quadratic cut that separates the current optimal solution, acting similarly to a hyper-plane in a cutting-planes method. We proved that this iterative algorithm converges to the optimal value of a tight semidefinite relaxation of (QBP).

This iterative algorithm was integrated into a spatial branch-and-bound method; the lower bound of each branch-and-bound node is determined using the new multi-quadric convexification. By generating quadrics specifically-tailored for each node, we almost systematically reduced the total number of nodes compared to existing methods that convexify using a unique quadric. The main computational bottleneck comes from the convex QCQP solver that has to be called after adding each new quadric; we managed to accelerate this step by using artificial tentative boxes over the decision variables (that are lifted only if necessary). Numerical results suggest it is more practical to use relatively few quadrics per node ($p \leq 3$); the resulting methods compete well not only with existing solvers based on other convexifications, but also with other solvers that use completely different techniques.

References

- [1] Horst, R., Pardalos, P.M., Thoai, N.V.: Introduction to global optimization. Kluwer, Dordrecht (2000)
- [2] An, L.T.H., Tao, P.D.: A branch and bound method via d.c. optimization algorithms and ellipsoidal technique for box constrained nonconvex quadratic problems. *Journal of Global Optimization* **13**(2), 171–206 (1998) <https://doi.org/10.1023/A:1008240227198>
- [3] Burer, S., Vandenbussche, D.: Globally solving box-constrained nonconvex quadratic programs with semidefinite-based finite branch-and-bound. *Comput Optim Appl* **43**, 181–195 (2009)
- [4] Vandenbussche, D., Nemhauser, G.: A branch-and-cut algorithm for nonconvex quadratic programs with box constraints. *Mathematical Programming* **102**(3), 259–275 (2005)
- [5] Bonami, P., Günlük, O., Linderoth, J.: Globally solving nonconvex quadratic programming problems with box constraints via integer programming methods. *Mathematical Programming Computation* **10**(3), 333–382 (2018)
- [6] Chen, J., Burer, S.: Globally solving nonconvex quadratic programming problems via completely positive programming. *Mathematical Programming Computation* **4**(1), 33–52 (2012)
- [7] Blik, C., Bonami, P., Lodi, A.: Solving Mixed-Integer Quadratic Programming problems with IBM-CPLEX: a progress report. In: Proceedings of the Twenty-Sixth RAMP Symposium, Hosei University, Tokyo, October 16-17 (2014)
- [8] Misener, R., Smadbeck, J.B., Floudas, C.A.: Dynamically generated cutting planes for mixed-integer quadratically constrained quadratic programs and their incorporation into GloMIQO 2. *Optimization Methods and Software* **30**(1), 215–249 (2015)
- [9] Tawarmalani, M., Sahinidis, N.V.: Global optimization of mixed-integer nonlinear programs: A theoretical and computational study. *Mathematical programming* **99**(3), 563–591 (2004)
- [10] Anstreicher, K.M., Burer, S.: Computable representations for convex hulls of low-dimensional quadratic forms. *Mathematical Programming* **124**(1), 33–43 (2010) <https://doi.org/10.1007/s10107-010-0355-9>
- [11] Burer, S., Letchford, A.N.: On nonconvex quadratic programming with box constraints. *SIAM Journal on Optimization* **20**(2), 1073–1089 (2009) <https://doi.org/10.1137/080729529>

- [12] Dong, H., Linderoth, J.: On valid inequalities for quadratic programming with continuous variables and binary indicators. In: Goemans, M., Correa, J. (eds.) *Integer Programming and Combinatorial Optimization*, pp. 169–180. Springer, Berlin, Heidelberg (2013)
- [13] McCormick, G.P.: Computability of global solutions to factorable non-convex programs: Part i - convex underestimating problems. *Mathematical Programming* **10**(1), 147–175 (1976)
- [14] Serali, H.D., Adams, W.P.: *A Reformulation-linearization Technique for Solving Discrete and Continuous Nonconvex Problems* vol. 31. Springer, ??? (2013)
- [15] Yajima, Y., Fujie, T.: A polyhedral approach for nonconvex quadratic programming problems with box constraints. *Journal of Global Optimization* **13**(2), 151–170 (1998)
- [16] Lambert, A.: Using general triangle inequalities within quadratic convex reformulation method. *Optimization Methods and Software* **38**(3), 626–653 (2023) <https://doi.org/10.1080/10556788.2022.2157002>
- [17] Sahinidis, N.V., Tawarmalani, M.: *Baron 9.0.4: Global optimization of mixed-integer nonlinear programs. User’s Manual* (2010)
- [18] Misener, R., Floudas, C.A.: Global optimization of mixed-integer quadratically-constrained quadratic programs (MIQCQP) through piecewise-linear and edge-concave relaxations. *Math. Program. B* **136**(1), 155–182 (2012). http://www.optimization-online.org/DB_HTML/2011/11/3240.html
- [19] Misener, R., Floudas, C.A.: GloMIQO: Global mixed-integer quadratic optimizer. *Journal of Global Optimization* **57**(1), 3–50 (2013) <https://doi.org/10.1007/s10898-012-9874-7>
- [20] GUROBI Optimization: GUROBI 9.1 (2021). <https://www.gurobi.com/>
- [21] Anstreicher, K.M.: Semidefinite programming versus the reformulation-linearization technique for nonconvex quadratically constrained quadratic programming. *Journal of Global Optimization* **43**(2), 471–484 (2009) <https://doi.org/10.1007/s10898-008-9372-0>
- [22] Burer, S., Vandenbussche, D.: A finite branch-and-bound algorithm for nonconvex quadratic programming via semidefinite relaxations. *Mathematical Programming* **113**(2), 259–282 (2008) <https://doi.org/10.1007/s10107-006-0080-6>
- [23] Vandenbussche, D., Nemhauser, G.L.: A polyhedral study of nonconvex quadratic programs with box constraints. *Mathematical Programming* **102**(3), 531–557 (2005)

- [24] Billionnet, A., Elloumi, S., Lambert, A.: Extending the QCR method to the case of general mixed integer program. *Mathematical Programming* **131**(1), 381–401 (2012)
- [25] Billionnet, A., Elloumi, S., Lambert, A.: Exact quadratic convex reformulations of mixed-integer quadratically constrained problems. *Mathematical Programming* **158**(1), 235–266 (2016) <https://doi.org/10.1007/s10107-015-0921-2>
- [26] Elloumi, S., Lambert, A.: Global solution of non-convex quadratically constrained quadratic programs. *Optimization Methods and Software* **34**(1), 98–114 (2019) <https://doi.org/10.1080/10556788.2017.1350675>
- [27] Porumbel, D.: Demystifying the characterization of SDP matrices in mathematical programming. arXiv preprint nr. 2210.13072 (2022)
- [28] Belotti, P., Kirches, C., Leyffer, S., Linderoth, J., Luedtke, J., Mahajan, A.: Mixed-integer nonlinear optimization. *Acta Numerica* **22**, 1–131 (2013)
- [29] IBM: IBM CPLEX 12.9 Reference Manual. "<https://www.ibm.com/docs/en/icos/12.9.0?topic=cplex-callable-library-c-api-reference-manual>" (2020)
- [30] ApS, M.: The MOSEK Optimization Toolbox for MATLAB Manual. Version 9.2. (2019). <http://docs.mosek.com/9.0/toolbox/index.html>
- [31] Helmberg, C.: Conic Bundle V0.3.10. (2011). <http://www-user.tu-chemnitz.de/helmberg/ConicBundle/>
- [32] Billionnet, A., Elloumi, S., Lambert, A., Wiegele, A.: Using a Conic Bundle method to accelerate both phases of a Quadratic Convex Reformulation. *INFORMS Journal on Computing* **29**(2), 318–331 (2017) <https://doi.org/10.1287/ijoc.2016.0731>
- [33] Fourer, R., Gay, D.M., Kernighan, B.W.: *AMPL: A Modeling Language for Mathematical Programming*, p. 351. The Scientific Press (now an imprint of Boyd & Fraser Publishing Co.), Danvers, MA, USA (1993)
- [34] GAMS Development Corporation. Fairfax, U. VA: General Algebraic Modeling System (GAMS) Release 36.1.0. (2021). <https://www.gams.com/download/>
- [35] Dolan, D., Moré, J.: Benchmarking optimization software with performance profiles. *Mathematical Programming* **91**, 201–213 (1986)