Global solution of Quadratic Problems by Interval Methods and Convex Relaxations

Sourour Elloumi, Amélie Lambert, Bertrand Neveu, Gilles Trombettoni

Global solution of Quadratic Problems by Interval Methods and Convex Relaxations

Sourour Elloumi[1,2], Amélie Lambert[2], Bertrand Neveu[3] and Gilles Trombettoni[4]

[1*]UMA, ENSTA Paris, Institut Polytechnique de Paris, Palaiseau, 91120, France.
[2]CEDRIC, Cnam, Paris, 75003, France.
[3]LIGM, Ecole des Ponts ParisTech, Univ. Gustave Eiffel, CNRS, France.
[4]LIRMM, University of Montpellier, CNRS, France.

Contributing authors: sourour.elloumi@ensta-paris.fr; amelie.lambert@cnam.fr; bertrand.neveu@enpc.fr; Gilles.Trombettoni@lirmm.fr;

Abstract

Interval branch-and-bound solvers provide reliable algorithms for handling non-convex optimization problems by ensuring the feasibility and the optimality of the computed solutions, i.e. independently from the floating-point rounding errors. Moreover, these solvers deal with a wide variety of mathematical operators. However, these solvers are not dedicated to quadratic optimization and do not exploit nonlinear convex relaxations in their framework. We present an interval branch-and-bound method that can efficiently solve quadratic optimization problems. At each node explored by the algorithm, our solver uses a quadratic convex relaxation which is as strong as a semi-definite programming relaxation, and a variable selection strategy dedicated to quadratic problems. The interval features can then propagate efficiently this information for contracting all variable domains. We also propose to make our algorithm rigorous by certifying firstly the convexity of the objective function of our relaxation, and secondly the validity of the lower bound calculated at each node. In the non-rigorous case, our experiments show significant speedups on general integer quadratic instances, and when reliability is required, our first results show that we are able to handle medium-sized instances in a reasonable running time.

Keywords: MINLP, Quadratic Programming, Interval methods, Semi-definite programming, Reliable solver, Experiments.

1 Introduction

We consider the Mixed Integer Quadratically Constrained Quadratic Programs (MIQCQPs) of the following form:

$$(P)\begin{cases} \min f(x) \equiv \langle Q_0, xx^T \rangle + c_0^T x & \\ \quad g_r(x) \equiv \langle Q_r, xx^T \rangle + c_r^T x \leq e_r & r \in \mathcal{R} \\ \quad \ell_i \leq x_i \leq u_i & i \in \mathcal{I} \\ \quad x_i \in \mathbb{N} & i \in \mathcal{J} \subseteq \mathcal{I} \end{cases}$$

where $\mathcal{R}$ is the set of inequality indices, $\mathcal{I}$ is the set of variable indices, and $\mathcal{J} \subseteq \mathcal{I}$ is the subset of integer variables. Each variable x_i lies in the interval $[\ell_i, u_i]$ where ℓ_i and u_i are real scalars. This interval is often called the domain of variable x_i. The quadratic forms f and g_r use symmetric real matrices Q_0 and Q_r and real vectors c_0 and c_r. The notation $\langle A_1, A_2 \rangle$ denotes a dot product between two matrices A_1, A_2 of the same dimensions.

Many optimization problems can be modelled as MIQCQPs. A few examples are optimization problems in graphs like max-clique, graph partitioning, or in industrial applications like district heating networks [1], optimal power flow [2]. MIQCQPs can also model any optimization problem with binary variables [3].

MIQCQPs are a particular case of Mixed Integer Non-Linear Problems (MINLP) [4] which are known to be very hard in general. Global solution methods exist however and are implemented in software solutions like `Baron` [5]. Specialized methods for MIQCQPs are implemented in other solvers like `GloMIQO` [6], or in standard solvers like `Gurobi` [7] or `CPLEX` [8].

Methods for solving (P) globally are based on partial enumeration of the search space implemented by a branch-and-bound algorithm. The two main ingredients are bounding and branching. Branching subdivides the set of feasible solutions in order to work on smaller subsets and aims at getting rid of non-feasible solutions as soon as possible. Bounding, which can for example be based on relaxations, finds a lower bound on the optimal value of (P) in order to eliminate subsets that can then be proven to contain no optimal solution. One of the key points to be addressed when using those methods is to find a good compromise between the quality of the bound and the time needed for its computation. Semi-definite programming (SDP) is known to provide strong bounds that need a large computation time. The Quadratic Convex Reformulation for Mixed Integer quadratic problems (`MIQCR`) approach [9][10] gets around this difficulty by capturing the bound quality of an SDP relaxation

within a convex quadratic reformulation of (P). A convex quadratic reformulation is an equivalent problem to (P) that has a trivial quadratic convex relaxation. Hence `MIQCR` is composed of two steps, the first one solves the SDP relaxation and builds the equivalent reformulation. The second step performs a branch-and-bound based on bounding by quadratic convex relaxation.

Interval methods are different solution methods that are still based on partial enumeration but focus on variable domain contraction. They consider the objective function as a particular variable and aim at improving the bounds on all variables by reasoning about all or part of the problem constraints [11–14]. `IbexOpt` [15] is a state-of-the-art constrained nonlinear optimization solver that has been used to build the `QIBEX` solver proposed in this paper. It uses rigorous interval algorithmic operators [15, 16].

Interval methods provide two main advantages: first, the guarantee of the solution obtained despite rounding problems on floating numbers; second, the possibility of defining the constraints and the objective function based on a wide variety of mathematical operators including arithmetic, trigonometric, exponential, logarithm, and even non-differentiable operators such as the absolute value.

Our objective in this paper is to introduce a cooperation between interval methods and quadratic convex reformulation methods. We aim to design an efficient and rigorous global method for problem (P). Our work leads to a new quadratic solver named `QIBEX` that is an variant of `IbexOpt` specialized to quadratic problems and improved by the quadratic convex reformulation framework. At each node, the hybrid solver `QIBEX` uses a quadratic convex relaxation that is built thanks to semi-definite programming (Section 2), together with a variable selection strategy dedicated to quadratic optimization. The interval features (Section 3) can then efficiently propagate this information for reducing/contracting all variable domains (Section 4). We then focus on the rigor issue (Section 5). Our experiments show significant speedups on general integer quadratic instances and emphasize the influence of introducing rigor (Section 6).

2 Convex quadratic reformulation

Quadratic convex reformulation first introduces a new variable X_{ij} that models the product of variables x_i and x_j, for each pair $(i, j) \in \mathcal{I}^2$. Then, it builds the following program, parameterized by the positive semi-definite matrices S_0

and $S_r \ \forall\, r \in \mathcal{R}$:

$$(PC)\begin{cases} \min F(x,X) \equiv \langle S_0, xx^T\rangle + \langle Q_0 - S_0, X\rangle + c_0^T x & & \\ \quad G_r(x) \equiv \langle S_r, xx^T\rangle + \langle Q_r - S_r, X\rangle + c_r^T x \le e_r & r \in \mathcal{R} & \\ \quad X_{ii} \ge x_i & i \in \mathcal{J} & (1)\\ \quad X_{ij} \ge u_j x_i + u_i x_j - u_j u_i & i,j \in \mathcal{I}^2 & (2)\\ \quad X_{ij} \ge \ell_j x_i + \ell_i x_j - \ell_j \ell_i & i,j \in \mathcal{I}^2 & (3)\\ \quad X_{ij} \le u_j x_i + \ell_i x_j - u_j \ell_i & i,j \in \mathcal{I}^2 & (4)\\ \quad X_{ij} \le \ell_j x_i + u_i x_j - \ell_j u_i & i,j \in \mathcal{I}^2 & (5)\\ \quad X_{ij} = x_i x_j & i,j \in \mathcal{I}^2 & (6)\\ \quad x_i \in \mathbb{N} & i \in \mathcal{J} \subseteq \mathcal{I} & (7) \end{cases}$$

Thanks to equalities (6), it can be easily checked that problems (P) and (PC) are equivalent for any parameters S_0, S_r $(r \in \mathcal{R})$. Moreover, since the latter matrices are positive semi-definite, the quadratic forms F and G_r are convex, and the only non-convexities in (PC) come from Constraints (6) and (7). By dropping these constraints from (PC) we obtain Problem $(\overline{PC})$ a convex quadratically constrained quadratic relaxation to (PC), and obviously to (P). Observe that the McCormick's envelopes (1)-(5) are necessary for the purpose of the branch-and-bound algorithm since they tighten the bound at each node of the branch-and-bound with the current domains of the variables.

Now, an important issue is the choice of matrices S_0 and S_r. The criterion adopted in [9] is to choose the matrices such that relaxation $(\overline{PC})$ is as tight as possible. We then consider the problem of finding a best set of positive semi-definite matrices $S_0^*, \ldots, S_m^*$, in the sense that the optimal solution value of $(\overline{PC})$ is as large as possible. It was proved in [10] that the best choice can be deduced from a dual optimal solution of the "Shor plus RLT" semi-definite programming relaxation of (P).

$$(SDP)\begin{cases} \min\langle Q_0, X\rangle + c_0^T x & & \\ \text{s.t.} & & \\ \langle Q_r, X\rangle + c_r^T x \le b_r & r \in \mathcal{R} & (8)\\ X_{ii} \ge x_i & i \in \mathcal{J} & (9)\\ X_{ij} \le u_j x_i + \ell_i x_j - u_j \ell_i & (i,j) \in \mathcal{I}^2 & (10)\\ X_{ij} \le u_i x_j + \ell_j x_i - u_i \ell_j & (i,j) \in \mathcal{I}^2 & (11)\\ X_{ij} \ge u_j x_i + u_i x_j - u_i u_j & (i,j) \in \mathcal{I}^2 & (12)\\ X_{ij} \ge \ell_j x_i + \ell_i x_j - \ell_i \ell_j & (i,j) \in \mathcal{I}^2 & (13)\\ \begin{pmatrix} 1 & x^T \\ x & X \end{pmatrix} \succeq 0 & & \\ x \in \mathbb{R}^n \quad X \in \mathcal{S}_n & & \end{cases}$$

The best matrices $(S_0^*, \ldots, S_m^*)$ can be built as follows:

i) $\forall r \in \mathcal{R}$, $S_r^* = \mathbf{0}_n$ (i.e. we linearize the initial quadratic constraints)

ii) $S_0^* = Q_0 + \sum_{r=1}^{m} \alpha_r^* Q_r + \Phi^*$ where:

- $\diamond$ α^* is the vector of optimal dual variables associated with Constraints (8),
- $\diamond$ matrix $\Phi^* = \Phi^{1*} + \Phi^{2*} - \Phi^{3*} - \Phi^{4*} - diag(\varphi^*)$, where φ^* is the vector of dual variables associated with Constraints (9), and Φ^{1*}, Φ^{2*}, Φ^{3*}, and Φ^{4*} are the symmetric matrices built from the optimal dual variables associated with Constraints (10) (13) respectively.

To sum up, we can solve the dual of problem (SDP) and deduce a positive semi-definite matrix S_0^* that will allow us to build the following tightest convex quadratic relaxation of (P):

$$(PC^*) \begin{cases} \min \quad \langle S_0^*, xx^T \rangle - \langle Q_0 - S_0^*, X \rangle + c_0^T x \\ \quad \langle Q_r, X \rangle + c_r^T x \leq e_r \qquad r \in \mathcal{R} \\ \quad (1) - (5) \end{cases}$$

Problem (PC^*) is a convex quadratic problem with linear constraints. It is also proved in [10] that it has the same optimal value as (SDP). It can be used to compute a tight lower bound to the optimal value of (P) and it can also be used within a spatial branch-and-bound to globally solve (P) by enforcing (6) and (7).

3 Interval branch & bounds

Several interval B&Bs for global optimization have been proposed, including `Numerica` [12], `GlobSol` [11], `Icos` [13], `IBBA` [14] and `IbexOpt` [15]. Let us summarize the principles behind `IbexOpt` that have been used to build our new quadratic solver `QIBEX` that is described in Section 4.

3.1 Outline of interval branch-and-bounds

An interval $[x_i] = [\underline{x_i}, \overline{x_i}]$ defines the set of reals x_i s.t. $\underline{x_i} \leq x_i \leq \overline{x_i}$, and we call a *box*, denoted by $[x]$, a vector of intervals, i.e. the Cartesian product of intervals $[x_1] \times ... \times [x_i] \times ... \times [x_n]$.

`IbexOpt` deals with continuous global optimization under inequality constraints defined by:

$$\min_{x \in [x]} f(x) \quad s.t. \quad g(x) \leq 0$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is the real-valued objective (non convex) function and $g : \mathbb{R}^n \to \mathbb{R}^m$ is a vector-valued (non convex) function. We present in Algorithm 1 the main features of the `IbexOpt` interval branch-and-bound algorithm.

Algorithm 1 is launched with an objective function f, a vector of constraints g, and with $[\ell, u]$ the input domain/box initializing the list q of open nodes.

Algorithm `IntervalBranch&Bound` $(f,\ g,\ [\ell,u],\ \epsilon_{obj},\ \epsilon_{sol})$

$f_{min} \leftarrow -\infty$ /* Lower bound */
$\widetilde{f} \leftarrow +\infty$ /* Upper bound */
$q \leftarrow \{[\ell,u]\}$ /* List of open nodes */
$x_{\widetilde{f}} \leftarrow \bot$ /* Best feasible point */
while $q \neq \emptyset$ **and** $\widetilde{f} - f_{min} > \epsilon_{obj}$ **and** $\frac{\widetilde{f}-f_{min}}{|\widetilde{f}|} > \epsilon_{obj}$ **do**
  $no \leftarrow$ `SelectNode`(q) ; $q \leftarrow q \setminus \{no\}$ /* node selection */
  $\tilde{i} \leftarrow$ `SelectVariable`$(no,\ \epsilon_{sol})$; /* variable selection */
  $(n_L, n_R) \leftarrow$ `Bisect` $(no,\ \tilde{i})$ /* separation/bisection step */
  $(n_L,\ \widetilde{f}) \leftarrow$ `Contract&Bound` $(n_L,\ q,\ \widetilde{f},\ x_{\widetilde{f}},\ f,\ g,\ \epsilon_{obj})$
  $(n_R,\ \widetilde{f}) \leftarrow$ `Contract&Bound` $(n_R,\ q,\ \widetilde{f},\ x_{\widetilde{f}},\ f,\ g,\ \epsilon_{obj})$
  $q \leftarrow$ `UpdateNodes` $(n_L,\ n_R,\ \epsilon_{sol},\ q)$
  $f_{min} \leftarrow \min_q \quad \ell_{obj}$
return $(x_{\widetilde{f}},\ \widetilde{f}\ )$

Algorithm 1: Outline of Interval-based branch-and-bound

Two accuracy parameters are also required as input: ϵ_{obj} that is the absolute or relative precision required on the objective function value and is used in the stopping criterion, and ϵ_{sol} that is the absolute accuracy of a domain.

Therefore, the algorithm computes a feasible point $x_{\widetilde{f}}$ of cost $\widetilde{f}$ such that no other solution exists with a cost lower than $\widetilde{f} - \epsilon_{obj}$. A variable x_{obj} is added to the vector x of variables, corresponding to the objective function value, along with a constraint $f(x) = x_{obj}$.

The B&B algorithm maintains an upper and a lower bound during the main loop:

- $\widetilde{f}$: the value of the best feasible point found so far,
- f_{min}: the minimal value of the lower bounds $\underline{x_{obj}}$ of the nodes q to explore, i.e. the minimal $\underline{x_{obj}}$ in the list of open nodes.
  In other terms, in every node no, there is a guarantee that no feasible point exists with an objective function value lower than $\underline{x_{obj}}$.

The procedure `SelectNode` selects the next node to handle, the one that has the minimal lower bound $\underline{x_{obj}}$ estimated for the objective function, hence performing a best-first search.[1] Once an open node has been selected, its domain/box is split into two sub-boxes along one dimension, that is selected by the variable selection strategy `SmearSumRel` through the procedure `Bisect`. Both sub-boxes are then handled by the `Contract&Bound` procedure (see Algorithm 2).

[1] Note that another node selection strategy can also be used by `IbexOpt` : the `FeasibleDiving` strategy, described in [17], did not appear to be the best option for our new QP solver `QIBEX` .

Algorithm `Contract&Bound` $(no,\ q,\ \widetilde{f},\ x_{\widetilde{f}},\ f,\ g,\ \epsilon_{obj})$

$n \leftarrow$ `Contraction` $(no, g \cup \{f(x) = x_{obj}\} \cup \{x_{obj} \leq \widetilde{f} - \epsilon_{obj}\})$

if $no \neq \emptyset$ **then**

$(xc,\ cost) \leftarrow$ `FeasibleSearch` $(no,\ f,\ g,\ \epsilon_{obj})$

if $cost < \widetilde{f}$ **then**

$\widetilde{f} \leftarrow cost$

$x_{\widetilde{f}} \leftarrow xc$

`FilterOpenNodes`$(q, \widetilde{f} - \epsilon_{obj})$

return $(no,\ q,\ \widetilde{f},\ x_{\widetilde{f}})$

Algorithm 2: The `Contract&Bound` procedure run at each node

A constraint $x_{obj} \leq \widetilde{f} - \epsilon_{obj}$ is first added to the problem for decreasing the upper bound of the objective function in the box. Imposing $\widetilde{f} - \epsilon_{obj}$ as a new upper bound (and not only $\widetilde{f}$) aims at finding a solution *significantly* better than the current best feasible point. Then, the procedure `Contraction` reduces the handled box without loss of feasible part. In other words, some infeasible parts at the limits of the domain are discarded by constraint programming and convexification techniques. This contraction is applied on the extended box that includes the variable x_{obj} modelling the objective function value, and thus improves the value of the lower bound l_{obj}.

The last part of the procedure carries out upper bounding. `FeasibleSearch` calls one or several algorithms searching for a feasible point $x_{\widetilde{f}}$ that improves the best cost $\widetilde{f}$ found so far. If the upper bound of the objective value is improved, the `FilterOpenNodes` procedure performs a type of garbage collector on all the open nodes by removing from q all the nodes having $\underline{x_{obj}} > \widetilde{f} - \epsilon_{obj}$.

Finally, after the calls to procedures `Contract&Bound`, the main B&B algorithm (Algorithm 1) push the two sub-boxes in the set q of open nodes. However, if the size of a box reaches the precision ϵ_{sol}, the box will be no more bisected, and only the minimal value of the objective function of these small discarded boxes is updated.

3.2 Main algorithms used by `IbexOpt`

Most of the algorithmic ingredients used by `IbexOpt` are called in Algorithm 2 by the `Contraction` and `FeasibleSearch` procedures.

In `IbexOpt`, `FeasibleSearch`, that implements upper bounding, does not use any local search (e.g., gradient descent) method. Instead, interval branch-and-bounds called algorithms `InHC4` and `InXTaylor` that are able to extract entirely feasible box or polytopes from the feasible domain [16].

The `Contraction` procedure, that removes non feasible parts of the domain, first includes contraction algorithms issued from constraint programming techniques, such as the state-of-the-art HC4 constraint propagation algorithm[18, 19] that works on each constraint individually, or stronger consistency algorithms (i.e., 3BCID(HC4) [20] or ACID(HC4) [21]). It also includes interval polyhedral relaxation of the feasible space using:

- `XTaylor` [22], a specific interval first order Taylor linear form of each inequality constraint, and/or
- `ART` [14, 19], an algorithm producing an affine form of the constraints based on affine arithmetic.

4 Improving an interval B&B using quadratic convex reformulation : QIBEX

Our hybrid algorithm `QIBEX` is an improvement of the solver `IbexOpt` [15, 16]. The steps of `IbexOpt` algorithm that were modified for designing `QIBEX` are surrounded in the pseudo-code of Algorithm 3 and 4.

We describe in Algorithm 3, the main procedure of our interval B&B. It starts from an initial node with domain/box $[x] = [\ell, u]$. An auxiliary variable x_{obj} represents the objective function value, and is added to the system along with the constraint $f(x) = x_{obj}$. First, the initial box is *contracted*, i.e. the bounds are improved without loss of feasible point. Basically, this `Contraction` procedure applies a sequence of contraction algorithms offered by `IbexOpt` and described in Section 3.2. Then, before performing the tree search, `QIBEX` calls procedure `QuadraticConvexReformulation` that computes the positive semi-definite matrix S_0^* and produces the quadratic convex relaxation (PC^*). Finally, the B&B is described in the while loop, and works in best-first node order. Once a node and a variable are selected, the domain is split into two parts by the `Bisect` separation procedure, and both sub-nodes (n_L, n_R) are handled by the `Cont&BoundQ` procedure before being added into the list of nodes $\mathcal{N}$ by `UpdateNodes`.

Algorithm `Qibex` $(f,\ g,\ x,\ [\ell,u],\ \epsilon_{obj},\ \epsilon_{sol})$

$f_{min} \leftarrow -\infty$ /* Lower bound*/
$\widetilde{f} \leftarrow +\infty$ /* Upper bound */
$x_{\widetilde{f}} \leftarrow \perp$ /* Best feasible point */
$no \leftarrow$ `createNode`$([\ell,u])$
$no \leftarrow$ `Contraction`$(no,\ g)$
$\boxed{PC^* \leftarrow \texttt{QuadraticConvexReformulation}(f,\ g,\ x,\ [\ell,u])}$
$q \leftarrow \{no\}$
while $q \neq \emptyset$ **and** $\widetilde{f} - f_{min} > \epsilon_{obj}$ **and** $\frac{\widetilde{f}-f_{min}}{|\widetilde{f}|} > \epsilon_{obj}$ **do**
  $no \leftarrow$ `SelectNode`(q) ; $q \leftarrow q \setminus \{no\}$ /* node selection */
  $\boxed{\tilde{i} \leftarrow \texttt{SelectVariableQ}(no,\ \epsilon_{sol})}$; /* variable selection */
  $(n_L, n_R) \leftarrow$ `Bisect` $(no,\ \tilde{i})$ /* separation/bisection step */
  $(n_L,\ q,\ x_{\widetilde{f}},\ \widetilde{f}) \leftarrow$ `Contract&BoundQ`$(n_L,\ q,\ f,\ g,\ x, \epsilon_{obj},\ x_{\widetilde{f}},\ \widetilde{f},$ $\boxed{PC^*})$
  $(n_R,\ q,\ x_{\widetilde{f}},\ \widetilde{f}) \leftarrow$ `Contract&BoundQ` $(n_R,\ q,\ f,\ g,\ x, \epsilon_{obj},\ x_{\widetilde{f}},\ \widetilde{f},$ $\boxed{PC^*})$
  $q \leftarrow$ `UpdateNodes` $(n_L,\ n_R, \epsilon_{sol},\ q)$
  $f_{min} \leftarrow \min_q \ \ell_{obj}$
return $(x_{\widetilde{f}},\ \widetilde{f})$

Algorithm 3: The `QIBEX` interval-based branch-and-bound for global solution of quadratic optimization problems

Another new feature of `QIBEX` compared to `IbexOpt` is the variable selection strategy `SelectVariable` that we adapt in a new strategy named `SelectVariableQ` from the strategy dedicated to quadratic programming of [23]. Indeed, the initial `SmearSumRel` variable selection strategy of `IbexOpt` does not take advantage of the nature of the relaxation (PC^*). More precisely, since Constraints (6) (i.e. $X_{ij} = x_i x_j$) are relaxed in (PC^*), it is more relevant to select a variable that maximizes the violation of these constraints. More formally, we first select the variable $x_{\tilde{i}}$ with

$$\tilde{i} = \underset{\substack{i \in \mathcal{I} \\ |\ u_i - \ell_i\ | > \epsilon_{sol} \\ x_i \in]\ell_i, u_i[}}{\arg\max} \ |\ X_{ii} - x_i^2\ |$$

if no violation is detected, we then select the variable $x_{\tilde{i}}$ with

$$\tilde{i} = \underset{\substack{(i,j) \in \mathcal{I}^2 \\ |\ u_i - \ell_i\ | > \epsilon_{sol} \\ x_i \in]\ell_i, u_i[, x_j \in]\ell_j, u_j[}}{\arg\max} \ |\ X_{ij} - x_i x_j\ |$$

To summarize, QIBEX first uses the variable selection strategy described above, if the strategy selects no variable, we resort to the SmearSumRel bisection strategy available in IbexOpt [15].

Algorithm Contract&BoundQ$(no,\ q,\ f,\ g,\ x,\ \epsilon_{obj},\ x_{\widetilde{f}},\ \widetilde{f},\ \boxed{PC^*})$

$no \leftarrow$ Contraction $(no,\ g \cup \{f(x) = x_{obj}\} \cup \{x_{obj} \leq \widetilde{f} - \epsilon_{obj}\})$

if *node no is still feasible* **then**

$\boxed{(x_{pc*},\ cost_{pc*}) \leftarrow \texttt{ConvexOptimize}(PC^*,\ no.[\ell, u])}$

$(xc,\ cost) \leftarrow$ FeasibleSearch $(no,\ f,\ g,\ \epsilon_{obj},\ \boxed{x_{pc*}})$ /* Upper bounding */

if $cost < \widetilde{f}$ **then**

$\widetilde{f} \leftarrow cost$

$x_{\widetilde{f}} \leftarrow xc$

$q \leftarrow$ FilterOpenNodes$(q,\ \widetilde{f} - \epsilon_{obj})$

$\boxed{no \leftarrow \texttt{Contraction}(no,\ g \cup \{cost_{pc*} \leq x_{obj} \leq \widetilde{f} - \epsilon_{obj}\} \cup \{f(x) = x_{obj}\})}$

return $(no,\ q,\ x_{\widetilde{f}},\ \widetilde{f})$

Algorithm 4: The Contract&BoundQ procedure called at each node of the QIBEX branch-and-bound algorithm

The other main improvement relates to the Contract&BoundQ procedure and is described in Algorithm 4. First, the standard Contraction procedure implemented in IbexOpt is called. If it leads to an empty box, it proves the absence of solution in this domain, and we are done. Otherwise, the ConvexOptimize procedure evaluates (PC^*) with the updated domains of variables $[\ell, u]$, whose optimal solution is called x_{pc*}. The four subsequent instructions carry out the upper bounding phase: FeasibleSearch tries to find a feasible point using several techniques proposed in IbexOpt, and as a new feature this procedure tests whether the original variables of x_{pc*} are feasible for the initial problem (P). If such a point xc is found and if its cost improves $\widetilde{f}$, the $x_{\widetilde{f}}$ and $\widetilde{f}$ are updated and the open nodes are filtered by FilterOpenNodes to remove those with a lower bound l_{obj} greater than $\widetilde{f} - \epsilon_{obj}$. A last call to the Contraction procedure is useful either if $cost_{pc*}$ improves the lower bound l_{obj} or if a better upper bound has been found.

An additional contribution is the handling of integer variables in QIBEX which was not provided in IbexOpt. For this, rounding to integer operations enforcing the integrality constraints are launched after the contraction operations and during feasible search computations.

5 A reliable implementation of QIBEX

Contrarily to existing quadratic program solvers, we can ensure that QIBEX is rigorous, reliable, whatever numeric tool is used to solve (SDP) and the (PC^*) relaxation.

Proposition 1. *Let us consider the QIBEX algorithm described in Algorithm 3 that applies to* (P) *with a user-defined precision* ε_{obj} *on the objective function value. Let* x^* *be the theoretical real feasible point minimizing the objective function* f*.*

Then, QIBEX can compute $x_{\tilde{f}}$ *a feasible point to* (P)*, and a lower bound* f_{min} *such that* $f_{min} \leq f(x^*) \leq f(x_{\tilde{f}})$ *and* $f(x_{\tilde{f}}) - f(x^*) \leq \varepsilon_{obj}$*.*

Let us recall that QIBEX is built on the top of IbexOpt and benefits from its interval computations. In particular, the feasibility of the sequence of points found by the interval branch-and-bound algorithm is ensured by interval arithmetic. In addition, at each node, the Contraction procedure called by Contract&BoundQ in the QIBEX algorithm is a slight extension of the Contraction procedure of IbexOpt that takes into account one additional constraint $cost_{pc*} \leq x_{obj}$, where $cost_{pc*}$ is the optimal value of (PC^*) at the current node. Obviously, it cannot lose any feasible point due to underlying interval methods.

Hence, to prove Proposition 1, we have to prove that the features added by QIBEX are also reliable, i.e.:

- The objective function of (PC^*) is a convex function, i.e. S_0^* is a positive semi-definite matrix (see Lemma 1 in Section 5.1).
- The optimal value of (PC^*), on any domain, provides a strict underestimate of the real-valued objective function value of (P) (see Lemma 2 in Section 5.2).

5.1 Guarantee of convexity of (PC^*)

The aim of the ConvexQuadraticRelaxation procedure is to compute the program (PC^*). To do this, we solve (SDP), and from its dual optimal solution, we build S_0^*. In practice, solving a semi-definite optimization problem can be very expensive. We know from [9] that from any feasible dual solution of (SDP), we are able to build a positive semi-definite matrix S_0. Hence, we do not need a solution that is proven optimal for (SDP) and its dual problem to ensure convexity of $F(x, X)$.

However, for numerical reasons the computed matrix S_0^* can be indefinite with eigenvalues very close to 0. To make the Hessian matrix of F positive semi-definite, a simple idea is to replace S_0^* by a positive matrix $S_0^* + |\delta| I_n$, i.e. add a positive value δ on the whole diagonal of S_0^*.

A non guaranteed way to achieve this goal is to compute the smallest eigenvalue λ of S_0^*, and take as corrective value $\delta = |\lambda|$.

We present in Algorithm 5 an iterative method that computes a corrective floating-point value δ that guarantees that $S_0 + |\delta| I_n \succeq 0$, and thus certifies the convexity of $F(x, X)$, the objective function of (PC^*).

Algorithm `CertifyConvexity` $(\lambda,\ \mathcal{P},\ k)$
- **for** $i \leftarrow 1$ *to* n **do**
  - $\epsilon_i \leftarrow 10^{-8}$
  - **while** $0 \notin \left[\mathcal{P}\right]_N([\lambda_i - \epsilon_i, \lambda_i + \epsilon_i])$ **do**
    - $\epsilon_i \leftarrow \epsilon_i * k$
- $\lambda_{\min} \leftarrow \min_i(\lambda_i - \epsilon_i)$
- $\delta \leftarrow \min\{0, \lambda_{\min}\}$
- **return** $(-\ \delta)$

Algorithm 5: An iterative algorithm that computes an underestimator of the smallest eigenvalue of a matrix.

The `CertifyConvexity` procedure is called with the characteristic polynomial $\mathcal{P}$ of a matrix (in our case S_0^*) and the vector λ of its eigenvalues computed by a numerical tool. The `for` loop iterates on each element λ_i in λ. The `while` loop computes a "small" interval around λ_i (of increasing size obtained by iteratively multiplying ϵ_i by a user-defined positive value k, e.g. $k = 2$ or $k = 10$) until $0 \in \left[\mathcal{P}\right]_N([\lambda_i - \epsilon_i, \lambda_i + \epsilon_i])$. We recall that $\left[\mathcal{P}\right]_N$ computes an interval $[a, b]$, by interval evaluation of the characteristic polynomial on the interval $[\lambda_i - \epsilon_i, \lambda_i + \epsilon_i]$. The fact that $0 \in [a, b]$ guarantees that $\lambda_i - \epsilon_i$ is a lower bound of the true real eigenvalue.

Finally, at the end of the `for` loop, $\lambda_{\min}$ stores the smallest lower bound value of any λ_i.[2] If the numerical tool used for computing the eigenvalues provides one or several negative values (due to round-off errors on floating point numbers), δ reflects the worst error and the procedure returns its absolute value. Otherwise, the procedure returns 0.

Lemma 1. *Let M be a matrix with floating-point coefficients. Assuming that a numerical tool can compute the whole set of eigenvalues of M, then the* `CertifyConvexity` *procedure computes a corrective value δ such that:*

$$M + |\delta| I_n \succeq 0$$

[2] The computation $\min_i(\lambda_i - \epsilon_i)$ is in fact achieved on intervals, i.e. $\min_i([\lambda_i, \lambda_i] - [\epsilon_i, \epsilon_i])$.

5.2 A reliable lower bound of (PC^*)

The second important ingredient for proving Proposition 1 concerns the rigor of the value computed when solving (PC^*) at each node of our interval branch-and-bound `QIBEX` .

Lemma 2. *Let us consider the `QIBEX` algorithm described in Algorithm 3 that applies to* (P)*. At each node of the branch-and-bound , `QIBEX` calls a convex QP solver to solve* (PC^*) *in the current domain* d*. This computation provides a value* f^d_{min}*.*

Then, there exists $f^{d'}_{min}$ *a corrective value of* f^d_{min} *that is a lower bound of the objective function value of* (PC^*) *in* d*, i.e. for all feasible point* x *in* d*,* $f^{d'}_{min} \leq f(x)$*.*

The lemma is a direct application of a theorem of interval analysis obtained in [24] and [25]. The result holds on any optimization problem $(\mathcal{CP})$ of a convex function constrained by p convex inequality constraints and q linear equality ones.

Theorem 1. *([24, 25]) Consider the following constrained convex optimization problem:*

$$(\mathcal{CP}) \begin{cases} min_{x \in \mathbb{R}^n} \ f(x) \\ s.c. \\ \qquad\qquad g(x) \leq 0 \\ \qquad\qquad Ax = b \\ \qquad\qquad \ell_i \leq x_i \leq u_i, \forall i \in \{1, \cdots, n\} \end{cases}$$

where f *and* g *are convex functions defined over* $\mathbb{R}^n$ *and* g *has its value in* $\mathbb{R}^p$*.* A *is a real matrix of size* $q \times n$ *and* b *is a real vector of* q *components. Let us denote by* x^* *the global solution of problem* $(\mathcal{CP})$ *and* $\mathcal{L}(x, \mu, \lambda) := f(x) + \mu^T.g(x) + \lambda^T.(Ax - b)$ *the Lagrangian function with* $\lambda \in \mathbb{R}^q$ *and* $\mu \in (\mathbb{R}^p)^+$ *its corresponding multipliers.*

For all $(\tilde{x}, \tilde{\mu}, \tilde{\lambda}) \in ([\ell, u], (\mathbb{R}^p)^+, \mathbb{R}^q)$*, one obtains:*

$$f(x^*) \geq \mathcal{L}(\tilde{x}, \tilde{\mu}, \tilde{\lambda}) + (z - \tilde{x})^T.\nabla_x \mathcal{L}(\tilde{x}, \tilde{\mu}, \tilde{\lambda})$$

where $z_i := u_i$ *if* $\frac{\partial \mathcal{L}}{\partial x_i}(\tilde{x}, \tilde{\mu}, \tilde{\lambda}) \leq 0$ *and* $z_i := \ell_i$ *otherwise.*

In other terms, for any vector $(\tilde{x}, \tilde{\mu}, \tilde{\lambda})$, the corrective value $(z - \tilde{x})^T.\nabla_x \mathcal{L}(\tilde{x}, \tilde{\mu}, \tilde{\lambda})$ added to $\mathcal{L}(\tilde{x}, \tilde{\mu}, \tilde{\lambda})$ allows us to correct the value of $\mathcal{L}(\tilde{x}, \tilde{\mu}, \tilde{\lambda})$ such that that we obtain a reliable lower bound of $f(x^*)$.

Hence, by applying Theorem 1 to the Lagrangian function of $F(x, X)$ we are sure to compute a lower bound of $f(x^*)$, the optimal value of (P). We

now apply Theorem 1 in the pure continuous case (i.e., by dropping Constraints (1) that apply to integer variables), but the reasoning is the same in presence of integer variables. With each constraint $g_r(x, X) \leq 0$, and Constraints (2), (3), (4), and (5), we associate a non-negative Lagrange multiplier μ_r and ν^1_{ij}, ν^2_{ij}, ν^3_{ij}, ν^4_{ij} respectively. We obtain the Lagrangian function:

$$\begin{aligned}\mathcal{L}(x, X, \mu, \nu^1, \nu^2, \nu^3, \nu^4) = \langle S^*_0, xx^T\rangle + c_0^T x - \langle Q_0 - S^*_0, X\rangle \\ + \sum_{r\in\mathcal{R}} \mu_r(\langle Q_r, X\rangle + c_r^T x - e_r) \\ + \sum_{(i,j)\in\mathcal{I}^2} \nu^1_{ij}(-X_{ij} + u_j x_i + u_i x_j - u_j u_i) \\ + \sum_{(i,j)\in\mathcal{I}^2} \nu^2_{ij}(-X_{ij} + \ell_j x_i + \ell_i x_j - \ell_j \ell_i) \\ + \sum_{(i,j)\in\mathcal{I}^2} \nu^3_{ij}(X_{ij} - u_j x_i - \ell_i x_j + u_j \ell_i) \\ + \sum_{(i,j)\in\mathcal{I}^2} \nu^4_{ij}(X_{ij} - \ell_j x_i - u_i x_j + \ell_j u_i)\end{aligned}$$

The first two components of the gradient of $\mathcal{L}(x, X, \mu, \nu^1, \nu^2, \nu^3, \nu^4)$ are:

$$\left(\begin{array}{c}\frac{\partial L}{\partial x_i} = c_{0i} + \sum_{j=1}^{n} S^*_{0ij} x_j + \sum_{r\in\mathcal{R}} \mu_r c_{ri} + \sum_{j=1}^{n} (\nu^1_{ij} u_j + \nu^2_{ij} \ell_j - \nu^3_{ij} u_j - \nu^4_{ij} \ell_j) \\ \frac{\partial L}{\partial X_{ij}} = Q_{0ij} - S^*_{0ij} + \sum_{r\in\mathcal{R}} \mu_r Q_{rij} - \nu^1_{ij} - \nu^2_{ij} + \nu^3_{ij} + \nu^4_{ij}\end{array}\right)$$

Note that when we use a numerical convex solver to solve (PC^*), we expect the computation of a good solution implying a quasi-null gradient. The other factor of the corrective value (i.e., $(z-\tilde{x})^T$) is bounded by the size of the current domain. However, the experiments on three significant benchmarks (see next section) highlight that the overall corrective value is very small and does not lead to the exploration of additional nodes in the branch-and-bound .

6 Experiments

We evaluate our algorithms `QIBEX` and `QIBEX-R` on two sets of instances. The first set is composed of 100 instances of quadratically constrained quadratic programs of [9] available at [26]. For those, we consider two classes of instances: the class QCP_5 of pure-continuous instances and the class $IQCP_5$ of pure-integer instances. Each QCP_5 ($IQCP_5$, resp.) instance consists in minimizing a quadratic function of n continuous (general integer, resp.) variable subject to 5 quadratic inequality constraints. For the considered instances, n varies from 10 to 50, and the variables belong to the interval $[0, 20]$. In the second

set, we consider the 135 instances of quadratically constrained quadratic programs from [27] called *unitbox*. Each *unitbox* instance consists in minimizing a quadratic function of n continuous variables in the interval $[0, 1]$, subject to m quadratic inequalities. For the considered instances, n varies from 8 to 50, and m from 8 to 100.

For solving the semi-definite programs (SDP) of methods `QIBEX` and `QIBEX-R`, we use the algorithm described in [28] with the solver `MOSEK` [29] together with the Conic Bundle Library [30]. For solving the quadratic convex relaxations (PC^*) at each node of the search tree, we use the `AMPL` [31] interface of the solver `Cplex 12.6.3` [32]. Finally, for our reliable version `QIBEX-R`, we implement Algorithm 5 in `IbexOpt` and Lemma 2 with the modelling language `AMPL`.

We set the parameters of `QIBEX` and `QIBEX-R` as follows:

- Node selection strategy : we use the best node first search.
- Contraction algorithms: we use `HC4` and `XTaylor` for $IQCP_5$ and QCP_5 instances, and `HC4` only for *unitbox* instances.
- Feasible search algorithm: for $IQCP_5$ and QCP_5 instances, we use `InHC4`, `InXTaylor` methods, and we use x_{pc^*}, the optimal solution of (PC^*), if it is feasible for (P). For the *unitbox* instances, we only use the last strategy.
- Accuracy parameters
  - Relative mipgap of the branch-and-bound: $\epsilon_{obj} = 10^{-5}$ for classes QCP_5, $IQCP_5$, and $\epsilon_{obj} = 10^{-4}$ for class *unitbox*.
  - Absolute accuracy of the constraints violation: $\epsilon_{const} = 10^{-4}$.
  - Absolute accuracy of the domain $[\ell, u]$: $\epsilon_{sol} = 10^{-6}$ for QCP_5 instances, $\epsilon_{sol} = 10^{-5}$ for *unitbox* instances, and $\epsilon_{sol} = 1$ for the integer instances $IQCP_5$.

6.1 Comparison of `QIBEX` with other solvers

We evaluate our new solver `QIBEX` on the 3 classes of problems $IQCP_5$, QCP_5, and *unitbox* (described at the beginning of the section), and compare its performances with that of 3 state-of-the art solvers: `IbexOpt`, `Baron 24.3.19` [5] and `Gurobi 911` [7]. For this, we use performance profiles (see [33]) of the CPU times. 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 $r_{is} = \frac{t_{is}}{\min\limits_{s} t_{is}}$. Let N be the total number of instances considered, an overall assessment of the performance P of solver s for a given τ is given by $P(r_{is} \leq \tau) = \frac{1}{N} *$ number of instances i such that $r_{is} \leq \tau$.

In Figures 1, 2, and 3, we present the performance profiles of the CPU times for methods `QIBEX` and the solvers `IbexOpt`, `Baron 19.3.24`, and `Gurobi 911` for the instances of the 3 classes of problems that we consider $IQCP_5$,

QCP_5, and *unitbox*, respectively. In our experiments, the time limit is set to 2 hours, and for `QIBEX`, the CPU time includes the pre-processing time necessary for solving (SDP). We observe that `QIBEX` outperforms the compared solvers both in terms of CPU times and number of instances solved for the 3 classes of instances. More precisely, `IbexOpt` solves 66 instances, `Baron` 151, `Gurobi` 187, and `QIBEX` 212 out of 235 instances within the time limit. The fact that `QIBEX` is slower than `Gurobi` on the "easiest" problems essentially comes from the CPU time of the pre-processing phase used to solve (SDP) for building (PC^*).

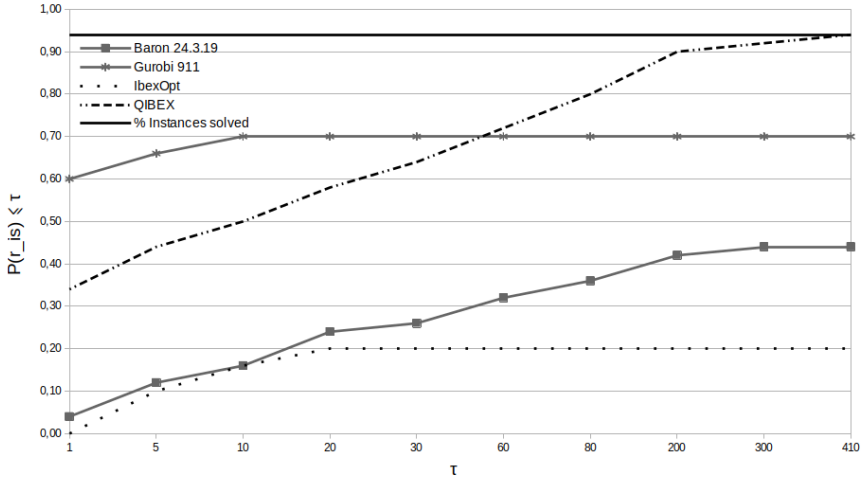


Fig. 1 Performance profile of the CPU times of `QIBEX` and the solvers `IbexOpt`, `Baron`, `Gurobi`, for instances $IQCP_5$ with $n = 10$ to 50. Time limit: 2 hours.

6.2 Impact of the new variable selection strategy

In this section, we start by a comparison of two variants of our new algorithm `QIBEX` : `QIBEX-S` that uses the initial `IbexOpt` variable selection strategy, and `QIBEX` that uses our tailored variable selection strategy `SelectVariableQ` described in Section 4. The two algorithms include the lower bound provided by (PC^*). In Figure 4, we compare the number of nodes required by the 2 methods on instances of size 20 to 50 that were solved by both algorithms within the time limit of 2 hours. We observe that the number of nodes is always reduced by using the new variable selection strategy, with an average factor of about 2. Note that the total CPU time is also reduced by a factor 2 on average.

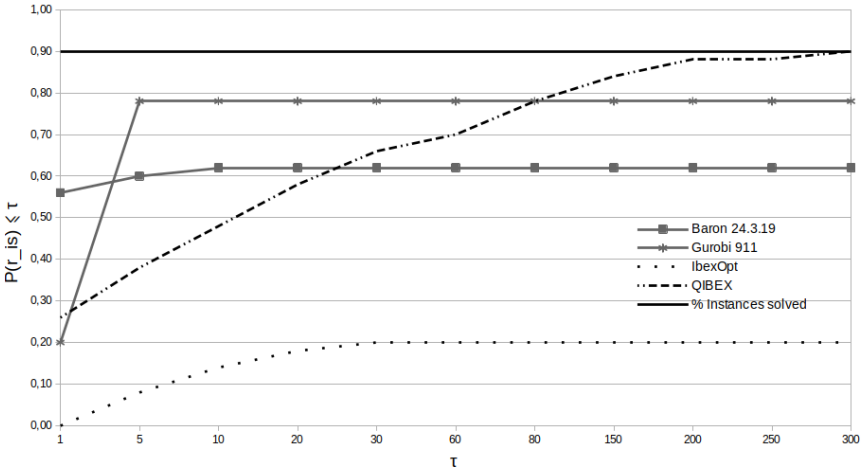


Fig. 2 Performance profile of the CPU times of `QIBEX` and the solvers `IbexOpt`, `Baron`, `Gurobi`, for instances QCP_5 with $n = 10$ to 50. Time limit: 2 hours.

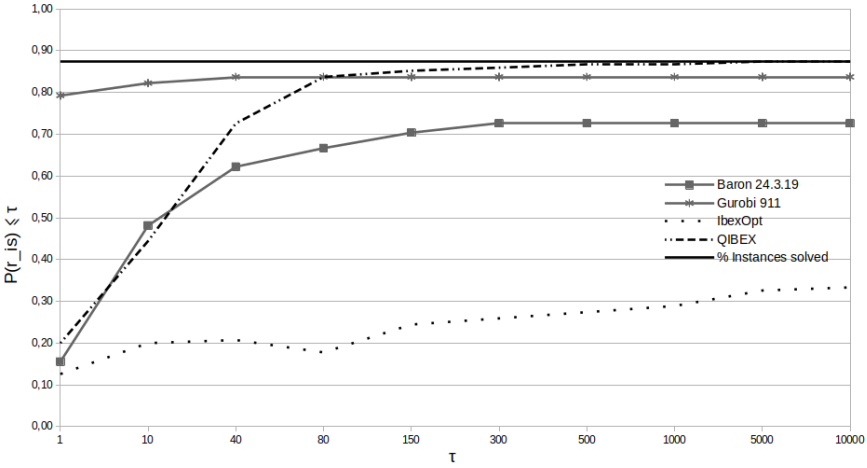
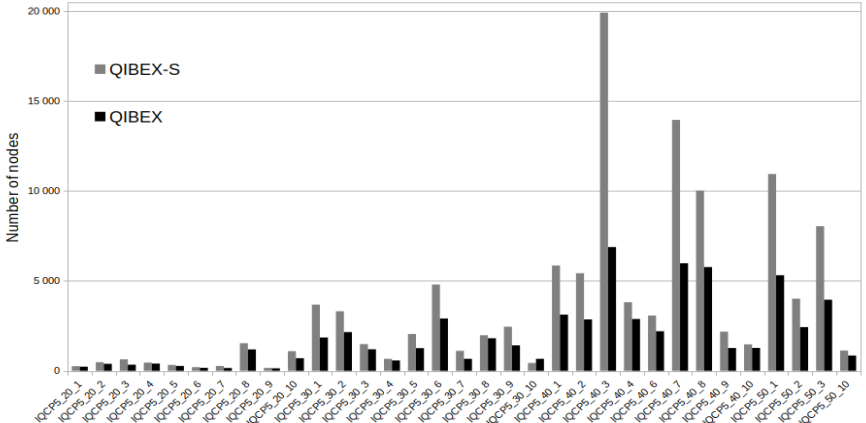


Fig. 3 Performance profile of the CPU times of `QIBEX` and the solvers `IbexOpt`, `Baron`, `Gurobi`, for instances *unitbox* with $n = 8$ to 50. Time limit: 2 hours.



6.3 Impact of the reliable implementation of QIBEX-R

In this section, we evaluate QIBEX-R, our reliable implementation of QIBEX, where we essentially discuss the impact of certifying two steps of QIBEX: the convexity of the objective function of (PC^*), and the validity of the lower bound provided by the optimal solution value of (PC^*). A first observation is that for the 3 families of instances $IQCP_5$, QCP_5 and *unitbox*, the number of nodes is identical on almost all the instances. This is due to the fact that the relative deviation between the optimal value of (PC^*) and the value calculated by Lemma 2 is always lower than 10^{-6}. Concerning the validity of the convexity of the objective function of (PC^*), we observe that the smallest eigenvalue of S_0^* is also almost systematically correct. Note that the time of this last verification is negligible compared to the total CPU time.

Finally, the main impact of the reliable implementation of QIBEX is the CPU time required for certifying the lower bound at each node of the branch-and-bound. Since the CPU times for solving (SDP) is the same for both algorithms, we only compare the CPU times of the branch-and-bound. We present in Figures 5, 6, and 7, the CPU times of the branch-and-bound of each instance solved by both solvers QIBEX, and QIBEX-R within the time limit of 2 hours, for the classes of instances $IQCP_5$, QCP_5, and *unitbox*, respectively. We observe that this certification significantly slows down the resolution for the 3 families of instances. More precisely, this time is increased by an average factor 8, 7 and 4, for the classes $IQCP_5$, QCP_5, and *unitbox*, respectively. This is in part due to the implementation of this certification by an AMPL script and should therefore be improved in a more sophisticated implementation.

Let us finally note that the reliable solver IbexOpt is more efficient on instances with $n \leq 10$ than QIBEX-R because of the time necessary to solve (SDP). However, IbexOpt is not able to solve any instance of classes $IQCP_5$ and QCP_5 with $n > 10$, and of class *unitbox* with $n > 28$, within the time limit of 2 hours, whereas QIBEX-R allows us to solve instances of significantly larger sizes (with $n \geq 40$ for the 3 classes).

7 Conclusion and future research

We show how ideas from quadratic convex relaxation and from interval methods can be mixed to get an efficient and rigorous global solution method for general quadratic optimization problems. Our numerical experiments show that rigor comes with a significant increase in the running time but has a very slight effect on the quality of the computed lower bounds and on the number of required nodes in the branching tree.

Future potential extensions may handle more general non-convex problems. A first idea would be to use a Taylor approximation of order 2 to approximate general non-linear functions by quadratic functions and come back to the quadratic convex relaxation framework.

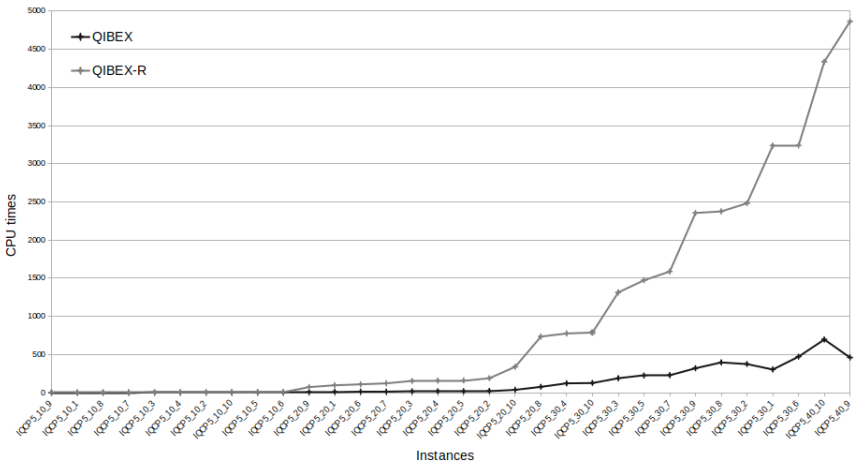


Fig. 5 CPU times per instance of methods `QIBEX` and `QIBEX-R` for instances IQCP5 with 10 to 40 variables

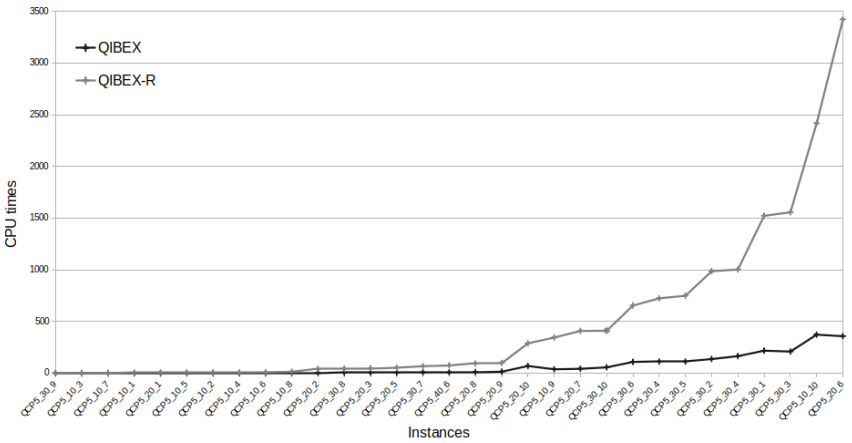


Fig. 6 CPU times per instance of methods `QIBEX` and `QIBEX-R` for instances QCP5 with 10 to 40 variables

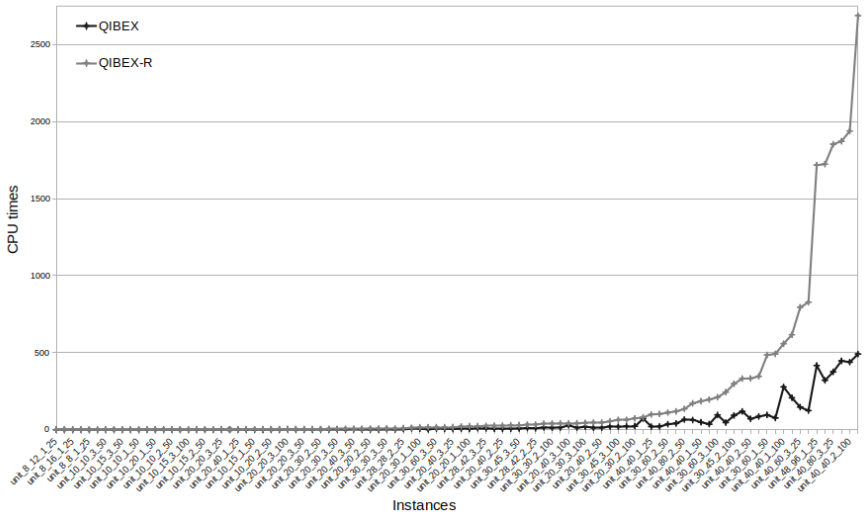


Fig. 7 CPU times per instance of methods `QIBEX` and `QIBEX-R` for instances unitbox with 8 to 48 variables

Aknowlegements

The datasets generated during and/or analysed during the current study are available from the corresponding author on reasonable request.
This research benefited from the support of the FMJH Program Gaspard Monge for optimization and operations research and their interactions with data science.

References

[1] Hering, D., Xhonneux, A., Müller, D.: Design optimization of a heating network with multiple heat pumps using mixed integer quadratically constrained programming. Energy **226**, 120384 (2021). https://doi.org/10.1016/j.energy.2021.120384

[2] Javadi, M.S., Gouveia, C.S., Carvalho, L.M.: A multi-temporal optimal power flow model for normal and contingent operation of microgrids. In: 2022 IEEE International Conference on Environment and Electrical Engineering and 2022 IEEE Industrial and Commercial Power Systems Europe (EEEIC/I&CPS Europe), pp. 1–6 (2022). IEEE

[3] Vandenberghe, L., Boyd, S.: Semidefinite programming. SIAM review **38**(1), 49–95 (1996)

[4] Belotti, P., Kirches, C., Leyffer, S., Linderoth, J., Luedtke, J., Mahajan, A.: Mixed-integer nonlinear optimization. Acta Numerica **22**, 1–131 (2013)

[5] Sahinidis, N.V., Tawarmalani, M.: Baron 24.3.19: Global optimization of mixed-integer nonlinear programs. User's Manual (2021)

[6] Misener, R., Floudas, C.A.: Glomiqo: Global mixed-integer quadratic optimizer. Journal of Global Optimization **57**(1), 3–50 (2013)

[7] Gurobi Optimization, L.: Gurobi Optimizer Reference Manual 911 (2021). http://www.gurobi.com

[8] Bliek1ú, 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, pp. 16–17 (2014)

[9] 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

[10] Elloumi, S., Lambert, A.: Global solution of non-convex quadratically constrained quadratic programs. Optimization Methods and Software **34**(1), 98–114 (2019)

[11] Kearfott, R.B.: Rigorous Global Search: Continuous Problems. Kluwer Academic Publishers, ??? (1996)

[12] Van Hentenryck, P., Michel, L., Deville, Y.: Numerica : A Modeling Language for Global Optimization. MIT Press, ??? (1997)

[13] Lebbah, Y., Michel, C., Rueher, M.: An Efficient and Safe Framework for Solving Optimization Problems. J. Computing and Applied Mathematics **199**, 372–377 (2007)

[14] Ninin, J., Messine, F., Hansen, P.: A Reliable Affine Relaxation Method for Global Optimization. 4OR **13**(3), 247–277 (2015)

[15] Trombettoni, G., Araya, I., Neveu, B., Chabert, G.: Inner regions and interval linearizations for global optimization. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 25 (2011)

[16] Araya, I., Trombettoni, G., Neveu, B., Chabert, G.: Upper Bounding in Inner Regions for Global Optimization under Inequality Constraints. J. Global Optimization (JOGO) **60**(2), 145–164 (2014)

[17] Neveu, B., Trombettoni, G., Araya, I.: Node Selection Strategies in Interval Branch and Bound Algorithms. Journal of Global Optimization **64**(2), 289–304 (2016)

[18] Benhamou, F., Goualard, F., Granvilliers, L., Puget, J.-F.: Revising Hull and Box Consistency. In: Proc. of International Conference on Logic Programming (ICLP), pp. 230–244 (1999)

[19] Messine, F.: Méthodes d'optimisation globale basées sur l'analyse d'intervalle pour la résolution des problèmes avec contraintes. PhD thesis, LIMA-IRIT-ENSEEIHT-INPT, Toulouse (1997)

[20] Trombettoni, G., Chabert, G.: Constructive Interval Disjunction. In: Proc. CP, Constraint Programming, LNCS 4741, pp. 635–650. Springer, ??? (2007)

[21] Neveu, B., Trombettoni, G., Araya, I.: Adaptive Constructive Interval Disjunction: Algorithms and Experiments. Constraints Journal **20**(4), 452–467 (2015)

[22] Araya, I., Trombettoni, G., Neveu, B.: A Contractor Based on Convex Interval Taylor. In: Proc. CPAIOR, LNCS 7298, pp. 1–16 (2012)

[23] Elloumi, S., Lambert, A., Lazare, A.: Solving unconstrained 0-1 polynomial programs through quadratic convex reformulation (2019)

[24] Jansson, C.: A Rigourous Lower Bound for the Optimal Value of Convex Optimization Problems. Journal of Global Optimization **28**(1), 121–137 (2004)

[25] Messine, F., Trombettoni, G.: Reliable Bounds for Convex Relaxation in Interval Global Optimization Codes. AIP Conference Proceedings **2070**(1), 020050 (2019). https://doi.org/10.1063/1.5090017

[26] Lambert, A.: IQCP/MIQCP: Library of Integer and Mixed-Integer Quadratically Constrained Programs. https://github.com/amelie-lambert/IQCP-MIQCP

[27] Bao, X., Sahinidis, N.v., Tawarmalani, M.: Multiterm polyhedral relaxations for nonconvex, quadratically constrained quadratic programs. Optimization Methods Software **24**(4-5), 485–504 (2009)

[28] 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

[29] MOSEK ApS: The MOSEK Optimization Toolbox for MATLAB Manual. Version 9.2. (2019). http://docs.mosek.com/9.0/toolbox/index.html

[30] Helmberg, C.: Conic Bundle V0.3.10

[31] Fourer, R., Gay, D.M., Kernighan, B.W.: Ampl. a modeling language for mathematical programming (2003)

[32] IBM-ILOG: IBM ILOG CPLEX 12.9 Reference Manual

[33] Dolan, D., Moré, J.: Benchmarking optimization software with performance profiles. Mathematical Programming **91**, 201–213 (1986)