A computational practicability study of MIQCQP reformulations

Sven Wiese[1]

MOSEK ApS, Fruebjergvej 3, Symbion Science Park, 2100 Copenhagen, Denmark
sven.wiese@mosek.com

Abstract. We report on computational results regarding several reformulation methods for MIQCQPs. The focus is set on the computational practicability of a reformulation, and we include challenging instances in our experiments. We are especially, though not exclusively, interested in reformulations based on SDPs and pursue the question of how such reformulations can be useful in an off-the-shelf solver.

Keywords: MIQCQP · Eienvalue-method · Diagonal perturbation · SDP relaxation

1 Introduction

Most authors use the term Mixed Integer Quadratic Programming (MIQP) for the minimization of a quadratic form over linear constraints imposed on mixed-integer variables. When also the constraints are allowed to contain quadratic terms, we usually talk about Mixed Integer Quadratically-Constrained Quadratic Programming (MIQCQP). The canonical form we use for describing such a problem throughout this paper is

$$\begin{array}{ll} \min & x^T Q^0 x + c^T x \\ \text{s.t.} & x^T Q^k x + a_k^T x \leq b_k, \quad k = 1, \ldots, m \\ & l \leq x \leq u \\ & x \in \mathbb{Z}^p \times \mathbb{R}^{n-p}. \end{array} \tag{P}$$

A decisive question about (P) is whether all involved matrices Q_k, that are w.l.o.g. assumed to be symmetric, are positive semidefinite (p.s.d.) or not. If so, it is well known that the continuous relaxation of (P) is a convex optimization problem, thus solvable in polynomial time to any desired accuracy. For solving (P), we may employ non-linear Branch-and-Bound or Outer-Approximation methods in this case. Although (P) is not a convex optimization problem itself due to the presence of integer variables, we then call it a convex MIQCQP, implicitly referring to its continuous relaxation.

If the latter is non-convex instead, we call it a non-convex MIQCQP. In that case we usually have to resort to spatial branching in order to solve it, increasing the computational burden. Yet, it is sometimes possible to translate the non-convexity encoded in the Q-matrices to the integrality of variables, and

we end up with a reformulation of the problem that is a convex MIQCQP, or even a Mixed Integer Linear Program (MILP). Such reformulations and their implementation, with a special focus on practicability, are the topic of this paper. Note that such reformulations can be useful also when (P) is already convex, as we will shortly discuss further below.

Two recurring themes in the literature for solving (MI)QCQPs are given by the reformulation-linearization (RLT) technique [5], and by the exploitation of Semidefinite Programming (SDP) relaxations. Our study is largely inspired by the line of research in [9, 7, 8, 14], that can be roughly described as a means of finding suitable perturbations of the matrices Q with desirable properties, again based on SDP-techniques. Perturbations that are restricted to the diagonals of the involved matrices are studied in [12, 13]. The recent paper [11] addresses the issue of choosing automatically between some of the reformulations we treat further below by means of machine learning.

Many commercial and non-commercial solvers that allow for mixed integer variables, i.e., Mixed Integer Programming (MIP) solvers, can handle some form of MIQCQP nowadays. Global optimization solvers like ANTIGONE [18], BARON [19], Couenne [6] and SCIP [20] accept both convex and non-convex MIQCQPs. Also CPLEX [3], and more recently Gurobi [2] extended their capability of solving convex MIQCQPs to the non-convex case. [10] provides an overview of the various techniques that contribute to the solution of MIQCQPs in such a software package. FICO Xpress [1] can be used to solve convex MIQCQPs, and non-convex ones are accepted and solved in a heuristic fashion. MOSEK [4] accepts convex MIQCQP in the current release 9.2. The results in Section 3 with non-convex MIQCQPs were obtained using MOSEK and showcase how it will be extended in a future release.

A recently vivid topic has been the conversion of convex quadratic constraints to conic form. For Q p.s.d., it is well-known that we can always find a factorization $Q = F^T F$ with some matrix F, leading to the identity

$$x^T Q x = \|Fx\|_2^2. \tag{1}$$

Replacing all quadratic forms in (P) with the above right-hand side for some appropriate factor F^k will transform (P) into a Mixed Integer Second-Order Cone Program (MISOCP). Some of the above solvers accept second-order constraints, or maybe even perform the conversion internally. A discussion about the advantages or disadvantages of the transformation of quadratic constraints to conic form is, nevertheless, outside the scope of this paper.

The remainder is organized in two sections: In Section 2 we discuss in more detail the various reformulation methods that form the basis of the computational results presented in Section 3. The contributions are twofold:

- We present computational results comparing these approaches in a unified computational environment, with a special focus on practical implementations. We include instances in our experiments that to the best of our knowledge have not been part of previous computational studies on the subject.

– Some of the reformulation methods are based on the solution of SDPs, and to the best of our knowledge there is so far no other MIP solver employing such reformulations. We thus try to answer the question of how useful SPD-based reformulations of MIQCQPs can be in such a software package.

2 Reformulation methods

In order to outline the various reformulation approaches, we largely base our notation on [8]. The authors therein introduce an infinite family of reformulations of (P), parametrized in the matrices $P^0, \dots, P^m$, as follows:

$$\begin{array}{ll} \min & x^T(Q^0 + P^0)x + c^T x - \langle P^0, X\rangle \\ \text{s.t.} & x^T(Q^k + P^k)x + a_k^T x - \langle P^k, X\rangle \le b_k, \quad k = 1, \dots, m \\ & X = xx^T \\ & l \le x \le u \\ & x \in \mathbb{Z}^p \times \mathbb{R}^{n-p}, \end{array} \qquad (\tilde{\mathrm{R}}_{P^0,\dots,P^m})$$

where $X \in \mathbb{R}^{n\times n}$ and $X = xx^T$ encodes the product relations $X_{ij} = x_i x_j$. The idea here is to perturb each Q^k with some matrix P^k, and we are of course interested in those perturbations such that $Q^k + P^k$ are p.s.d. for all k. If so, the new sources of non-convexity in $(\tilde{\mathrm{R}}_{P^0,\dots,P^m})$ are the integrality of variables, and the constraint $X = xx^T$. The latter can, under certain circumstances, again be reformulated so as to translate this non-convexity to the integrality of the variables. The easiest case where this can be seen is when $i = j$ and x_i is a binary variable. Then clearly

$$x_i^2 = x_i, \tag{2}$$

or in other words $X_{ii} = x_i$, which is why it is not even necessary to introduce the variable X_{ii} explicitly. A more difficult case is when still at least one of the two variables in the product $x_i x_j$, say x_i, is binary, and the other has finite bounds $l_j \le x_j \le u_j$. Then $X_{ij} = x_i x_j$ is equivalent to

$$l_j x_i \le X_{ij} \le u_j x_i, \tag{3}$$

$$x_j - u_j(1 - x_i) \le X_{ij} \le x_j - l_j(1 - x_i). \tag{4}$$

If none of the two variables is binary instead, but at least one is integer with finite bounds, say again $l_i \le x_i \le u_i$, it can be written via its binary expansion

$$x_i = \sum_{k=0}^{\lfloor \log(u_i - l_i)\rfloor} 2^k t_{ik} + l_i \tag{5}$$

with auxiliary binary variables t_{ik}. Therewith we can rewrite the product

$$x_i x_j = \sum_{k=0}^{\lfloor \log(u_i - l_i)\rfloor} 2^k t_{ik} x_j + l_i x_j, \tag{6}$$

and subsequently introduce yet more auxiliary variables $z_{jik} = t_{ik}x_j$. Since these products contain a binary variable now, they can be rewritten using the same idea as in (3) - (4), leading to an extended but linear description of $X_{ij} = x_i x_j$.

Denote by $L = \{(i,j) \mid \exists k : p^k_{i,j} \neq 0\}$ the set of all variable pairs that are perturbed with a non-zero coefficient in $(\tilde{\mathrm{R}}_{P^0,\ldots,P^m})$, and by $L_1 \subseteq L$ those pairs that can be linearized by (2) - (6). For ease of exposition, we assume that $L = L_1$ throughout the rest of this section. We denote by $S_{x,X,t,z}$ the mixed-integer set collecting all the relations (2) - (6) for the pairs in L, and can thus write the next step in our reformulation process as

$$\begin{array}{lll} \min & x^T(Q^0 + P^0)x + c^T x - \langle P^0, X\rangle & \\ \text{s.t.} & x^T(Q^k + P^k)x + a_k^T x - \langle P^k, X\rangle \leq b_k, \quad k = 1,\ldots,m & \\ & (x, X, t, z) \in S_{x,X,t,z} & (\mathrm{R}_{P^0,\ldots,P^m}) \\ & l \leq x \leq u & \\ & x \in \mathbb{Z}^p \times \mathbb{R}^{n-p}. & \end{array}$$

$(\mathrm{R}_{P^0,\ldots,P^m})$ is the generic reformulation we work with, and below we list four different choices for the matrices P^k.

Remark 1. It can happen that $L_0 := L \setminus L_1 \neq \emptyset$, for example in the presence of products between two continuous variables, or when required bounds are not finite. One might also resort to a policy of not performing the linearizations (3) - (6) when either $\max(|l_j|, |u_j|)$ or $u_i - l_i$ are larger than specified constants. In Section 3 we allow that $L_0 \neq \emptyset$, provided that $\sum_{i,j \in L_0} q^k_{ij} x_i x_j \geq 0$ for all k. Also, we restrict to reformmulations such that $p^k_{ij} = 0$ for all $i, j \in L_0$ and all k in such a case. In other words, we never perturb the coefficient of a product that is not linearized, and the non-linearized residual of each Q^k is required to be p.s.d. This way we still end up with a valid reformulation $(\mathrm{R}_{P^0,\ldots,P^m})$ that is a convex MIQCQP. For the details, especially regarding Section 2.4, we refer to [8]. The approaches in Sections 2.1 - 2.3 can be extended to the case $L_0 \neq \emptyset$ in straightforward ways.

2.1 Complete linearization

Setting $P^k = -Q^k$ for all k amounts to linearizing the whole program, ending up with a MILP (unless $L_0 \neq \emptyset$, cf. Remark 1). This is particularly interesting from a computational point-of-view. MILP solver technology is much more sophisticated than Mixed Integer Nonlinear Programming (MINLP) technology in general, but also compared to more special paradigms like convex MIQCQP or MISOCP, say. This means that a MILP can on average be solved faster than a MIQCQP of comparable size. This alone shows how such a reformulation can also be useful for convex MIQCQPs.

The average dominance of MILP over the other named paradigms may of course not hold true when looking at two individual problems. That is also the reason why it is not necessarily true that a complete linearization is always preferable to any of the reformulation methods in the sections to follow, although

they usually don't lead to MILPs. Also, depending on the set L, the amount of linearization required to perform this reformulation can vary from instance to instance and make it more or less attractive.

2.2 The eigenvalue-method

The so-called eigenvalue-method has originally been proposed in [16]. Assume for a moment that Q^k is not p.s.d. and let its eigenvalues be denoted by $\lambda_1^k \leq \lambda_2^k \leq \ldots \leq \lambda_m^k$ with $\lambda_1^k < 0$. Setting $P^k = -\lambda_1^k I$ will make sure that the matrix $Q^k + P^k$ is p.s.d. This can be seen by noting that its eigenvalues are given by $0 \leq \lambda_2^k - \lambda_1^k \leq \ldots \leq \lambda_m^k - \lambda_1^k$, i.e., they are all non-negative.

One might ask why exactly we take λ_1^k in order to form the perturbation matrix, as by the same reasoning any $f \leq \lambda_1^k$ would achieve positive semidefiniteness of $Q^k + P^k$. The reason is the somewhat heuristic way of thinking that less convex quadratic forms, achieved through smaller eigenvalues, lead to better dual bounds, i.e. lower bounds obtained from solving the continuous relaxation of $(\mathrm{R}_{P^0,\ldots,P^m})$. Therefore we seek such an f that is maximal. Another way of looking at it is to shift the whole interval of eigenvalues $[\lambda_1^k, \lambda_m^k]$ to the right, but just enough so that its left boundary is the origin.

The dual-bound point-of-view is also well suited for illustrating how this reformulation (and also the ones of the following two sections) can be useful for convex MIQCQPs. If $\lambda_1^k > 0$, perturbing with $P^k = -\lambda_1^k I$ will again shift the interval of eigenvalues (this time leftward) so that its left boundary is precisely the origin. This "less convex" problem is likely to have a better dual bound than the original one. We analyze the dual bounds of the various reformulations computationally in section 3.

Contrary to a complete linearization, the reformulated problem will still be a MIQCQP. Another difference is the amount of linearization that has to be performed: this tends to be lower since the perturbation is restricted to the diagonal of the matrices, while otherwise they have to be linearized entirely. On the other hand we note the following:

Remark 2. It seems quite natural, from the presolving point-of-view, say, to substitute all squares of binary variables in (P) with just the variable itself, and a complete linearization does precisely that via (2). In the eigenvalue-method, if x_i is binary, we are likely to keep x_i^2 explicitly in $(\mathrm{R}_{P^0,\ldots,P^m})$ with a non-zero perturbed coefficient. It might even be that $q_{ii}^k = 0$ in (P), but $p_{ii}^k \neq 0$, and we thus introduce a square where there was none before. The eigenvalue-method (and the two methods to follow) are thus not necessarily able to make use of the simplification $x_i^2 = x_i$ everywhere in $(\mathrm{R}_{P^0,\ldots,P^m})$.

2.3 The diagonal-method

From the dual-bound point-of-view, taking the minimum eigenvalue in the previous section is the best we can do, but only under the restriction that the whole diagonal of Q^k is perturbed by the same value. [13] removes this restriction and

allows for different perturbations in different diagonal entries. So the task is to find a $P^k = -\operatorname{diag}(\mu_1, \ldots, \mu_n)$ such that the resulting perturbed matrix is p.s.d., making the μ_i possibly large. Just as in [13] we choose to maximize their sum in the following. Thus, for each k we solve the SDP

$$\begin{array}{ll} \max & \sum\limits_{i=1}^{n} \mu_i \\ \text{s.t.} & Q^k - \operatorname{diag}(\mu_1, \ldots, \mu_n) \succeq 0. \end{array} \qquad (\mu\text{-SDP})$$

We refer to the resulting reformulation as the diagonal-method. The chosen objective amounts to minimizing the sum of the eigenvalues of the perturbed matrix. This can be seen by the fact that the sum of the eigenvalues of any matrix is equal to its trace, and we have

$$\operatorname{tr}\left(Q^k - \operatorname{diag}(\mu_1, \ldots, \mu_n)\right) = \sum_{i=1}^{n} (q_{ii}^k - \mu_i) = \sum_{i=1}^{n} q_{ii}^k - \sum_{i=1}^{n} \mu_i,$$

where the first sum is constant in (μ-SDP).

We can see that the diagonal-method is in a certain sense more general than the eigenvalue-method: if we added the constraints $\mu_i = \mu_j$ for all $i \neq j$, the optimal solution would be given by a vector composed of n repetitions of the minimum eigenvalue. Therefore one may expect the dual bound of the diagonal-method to be tendentially better. However, performing the reformulation is also likely more expensive, since it involves the solution of a SDP, opposed to just the computation of the minimum eigenvalue.

2.4 Maximizing the dual bound

[8] provides the fourth reformulation method we consider. It is based on the solution of a SPD relaxation of (P) that combines two themes for constructing relaxations of MIQCQPs. First, we have the famous RLT- or McCormick [17] inequalities,

$$X_{ij} - u_j x_i - l_i x_j \leq -u_j l_i, \qquad (7)$$
$$X_{ij} - l_j x_i - u_i x_j \leq -l_j u_i, \qquad (8)$$
$$X_{ij} - u_j x_i - u_i x_j \geq -u_j u_i, \qquad (9)$$
$$X_{ij} - l_j x_i - l_i x_j \geq -l_j l_i. \qquad (10)$$

Note that (3) - (4) are just a special case of these. A feature that seems unique to [8] is the additional use of the equations

$$X_{ii} - x_i \geq 0, \qquad (11)$$
$$X_{ii} + x_i \geq 0 \qquad (12)$$

that represent $x_i^2 \geq |x_i|$, which holds true for any integer variable x_i. We then also have the routinely used semidefinite relaxation of $X = xx^T$, i.e., $X - xx^T \succeq 0$. The SDP in question then reads

$$
\begin{array}{lll}
\min & \langle Q^0, X\rangle + c^T x & \\
\text{s.t.} & \langle Q^k, X\rangle + a_k^T x \le b_k, & k = 1,\dots,m \\
& (7)-(10) & (i,j) \in L \\
& (11),(12) & i = 1,\dots,p \\
& l \le x \le u & \\
& \begin{pmatrix} X & x \\ x^T & 1 \end{pmatrix} \succeq 0. &
\end{array}
\tag{RSDP}
$$

The last constraint can be seen to be equivalent to $X - xx^T \succeq 0$ by the Schur complement lemma. If we wrote out the dual of (RSDP) it would contain a dual $(n+1)\times(n+1)$-matrix variable $\bar{S} \succeq 0$. The reformulation consists in taking $P^0 = S - Q^0$, where $S \succeq 0$ is obtained from $\bar{S}$ by eliminating the last row and column, and $P^k = -Q^k$ for all $k \ge 1$.

The nice theoretical results in [8] show that among all convex reformulations ($\mathrm{R}_{P^0,\dots,P^m}$) with added inequalities (7) - (10) and (11), (12), this one maximizes the optimal value of its continuous relaxation. On the one hand, it can thus be seen as the reformulation that has the desirable property of maximizing the dual bound. On the other hand, we note that it is likely to be more expensive than a complete linearization or the eigenvalue-method, since it involves the solution of a costly SDP. It is likely also more expensive than the diagonal-method, since (RSDP) is generally more difficult than the SDPs required for the diagonal-method. Especially when MIQCQP contains several Q-matrices, the diagonal-method acts on them independently, one at a time, instead of solving a problem that considers all the quadratic structure (and more) at once.

We close this section by noting that the amount of linearization required for this reformulation is in the ballpark area of that of a complete linearization, and thus tends to be larger than that for the eigenvalue- or diagonal-method.

3 Computation

We now report on computational experiments with the above reformulations that we conducted inside MOSEK. The history of MOSEK is tied to convex optimization, and so it accepts convex MIQCQPs and will instead reject non-convex ones in the current release MOSEK 9.2. The results presented in this section allowing to solve also non-convex MIQCQPs are part of the development version of MOSEK and are intended to carry over to a future release. We point out, however, that if a given instance is not amenable to a reformulation because the conditions stated in Remark 1 are not met, MOSEK will continue to reject it. All the sub-problem classes needed in the following, i.e., LPs and QCQPs for the computation of dual bounds, and SPDs, are readily solvable with MOSEK.

Performing the reformulation. We want to asses advantages and disadvantages of the different approaches computationally. We have mentioned the fact

Table 1. Comparison of gaps, (reformulation) times and non-zeroes on instances from [**?**].

	gap		reform. time (sec.)		total time (sec.)		#nz	
	ar.	geo.	ar.	geo.	ar.	geo.	ar.	geo.
(R_Q)	267.74	227.16	0.0038	0.0038	0.1556	0.1480	20674	11489
(R_λ)	54.85	42.90	0.0018	0.0018	0.0599	0.0580	3040	2002
(R_μ)	37.13	21.56	0.0168	0.0166	0.0754	0.0730	3037	2003
(R_S)	20.93	15.26	4.9738	1.9407	6.0085	2.4002	21302	11915
(R_S^c)	20.43	13.70	0.4781	0.3576	0.9778	0.7309	15320	8991
(R_S^i)	11238.70	24.08	0.0237	0.0236	0.2372	0.161	5859	2759

that they lead to different dual bounds, and thus include the latter in our analysis, besides the cost of a reformulation in terms of the time it takes to perform it. When we compute optimality gaps, we mean

$$gap = 100 \cdot \frac{z - z^*}{\max(1, |z|)},$$

where z is the objective value of the optimal or best known solution to a given instance, and z^* is the solution value of the continuous relaxation of the reformulation in question. We also report the total time it takes to compute the dual bound of the reformulated problem (i.e., the reformulation time plus the time for solving the relaxation) and the number of non-zeroes in its constraint matrix. Note that for QCQPs, MOSEK automatically performs the conversion (1) by computing a Cholesky factorization, so that the reported number of non-zeroes captures both the linear and quadratic part of (P), and is thus to give an indication of the problem size that a reformulation leads to.

We apply a moderate amount of MOSEK's presolving before applying a reformulation so as to increase the likelihood of being able to apply one at all, cf. Remark 1. Below we report arithmetic and geometric means of gaps, times and number of non-zeroes. For gaps, the geometric mean is computed by considering the maximum of 1 and the actual gap for each instance, while for times, we use a shift of 1 second. Throughout the tables, we write (R_Q), (R_λ), (R_μ) and (R_S) for referring to the reformulations from Sections 2.1, 2.2, 2.3 and 2.4, respectively.

Smallish instances for illustration purposes. We start by taking the instance set used in [8]. These are 281 randomly generated non-convex instances that cover both the cases $p = n$ and $p < n$. Some instances also have $m = 0$, and we stress that no binary variables appear in any instance. These instances might be of less practical interest, but they are also rather small when compared to the instance set we use further down, and are thus useful for illustrating some concepts that guide us in making amendments later on.

Table 1 shows means for the four methods we introduced. They indicate a clear dominance relationship: the average gaps decrease monotonically when going from (R_Q) over (R_λ) and (R_μ) to (R_S). This gain in the dual bound

comes with a price though: the average reformulation times increase. While (R_Q) and (R_λ) seem equally cheap, (R_μ) is more expensive by about an order of magnitude, and (R_S) by several orders of magnitude. This reflects the cost of solving SDPs, but also shows that (RSDP) in particular can become quite costly to solve. In fact, on the more realistic instance set we use in the next section, it is sometimes completely out-of-reach to solve (RSDP) within the range of hours, or it can cause memory issues due to its size. The bottleneck are usually the RLT-inequalities (7) - (10) when the set L (or better L_1) is large.

Therefore, we refrain from solving (RSDP) in its entirety in the following, but instead start with some relaxation of (RSDP), containing (7) - (10) for only some selected pairs (i, j), and separate more in successive separation rounds. We included the numbers obtained with this strategy in Table 1 in the column (R_S^c). In each round we separated all RLT-inequalities that were violated by a small tolerance, and only aborted this straightforward cut loop if no new inequalities were separated.

We note that the average reformulation time is cut down again by about an order of magnitude. Taking into account that we have to solve several SDPs successively during the cut loop, this is an interesting gain, showing the savings that can be achieved by using only those RLT-inequalities that are "necessary". We stress though that the average reformulation time remains significantly higher than that of the other reformulation methods. We also note that we don't loose anything of the average gap closure here. This seems intuitive as at the end of a cut loop as described above, there are no violated inequalities left that could increase the solution value of (RSDP). The latter is directly connected to the dual bound of the reformulation [8]. The gap closure is even slightly higher for (R_S^c), which is related to the fact that additional inequalities in (RSDP) can alter the dual solution and thus the matrix S used in the reformulation.

It is also interesting to note that when we only use the initial relaxation of (RSDP) without any separation, included in Table 1 in the column (R_S^i), we sometimes are left with huge gaps, explaining the large arithmetic mean. This again highlights the importance that the separation can have in some cases.

Finally we note that the different amount of linearization required for different reformulations is reflected in Table 1. (R_λ) and (R_μ) behave almost identically, as could be expected, and lead to significantly smaller reformulations than (R_Q) and (R_S), who behave quite similar to each other. Note that just as for reformulation times, the separation in (R_S^c) is able to cut down a non-negligible part of the reformulation size compared to (R_S). The difference in size can also have an impact on the time spent solving the relaxation, and thus the total time, as can be seen by comparing (R_Q) with (R_λ) or (R_μ) for example.

QPLIB. We now present results obtained with QPLIB [15]. We first picked a subset of 143 instances by discarding those that are either convex or not amenable to a reformulation according to Remark 1 or don't contain any integer variable. The instances in the QPLIB collection are more realistic and also much larger on average than the ones of the previous section, and as anticipated

Table 2. Comparison of gaps, (reformulation) times and non-zeroes on 128 QPLIB instances.

	gap		reform. time (sec.)		total time (sec.)		#nz	
	ar.	geo.	ar.	geo.	ar.	geo.	ar.	geo.
(R_Q)	1810.56	195.53	0.0345	0.0331	7.6124	3.5416	63958	28164
(R_λ)	5556.73	62.24	0.0859	0.0671	7.2121	1.0821	73959	17601
(R_μ)	3769.75	70.12	3.8893	0.7897	10.7663	1.6356	73978	17641
(R_S^c)	2647.31	28.96	41.3180	4.7273	49.4606	6.1242	79564	21776
(R_S^i)	3578.91	41.25	9.4072	1.8879	16.0606	2.9770	77736	20285

solving (RSDP) is sometimes completely out-of-reach for excessive memory or time consumption. Even the initial relaxation of (RSDP), used for (R_S^c) and (R_S^i), can be so large that it is prohibitive to solve. Also the solution of (μ-SDP) or the computation of a minimum eigenvalue can be prohibitive for very large matrices. We therefore made it possible in our implementation to impose several work limits when performing a reformulation, in particular on:

1. the dimension of any involved matrix when performing (R_λ) or (R_μ)
2. the size of the initial relaxation of (RSDP) when performing (R_S^c) or (R_S^i)
3. the total computational effort spent in all solved SDPs
4. the amount of work spent in separation, in terms of
   - the number of separation rounds
   - the number of inequalities added per round
   - the minimum required improvement of the optimal value of the (RSDP) from one round to the next

1. and 2. can also mean that an instance is not amenable to a certain reformulation. In order to obtain the following results, we set the work limits such that no memory issues occur, and such that the reformulation time never exceeds 30 minutes. This leaves us with 128 instances on which all four reformulations can be performed. Table 2 shows the results for (R_Q), (R_λ), (R_μ), (R_S^c) and (R_S^i).

The picture about the gap closure on this more realistic instance set is somewhat different to the previous section. No such clear dominance relation can be seen here, some methods having lower geometric but higher arithmetic means than others. Out of 128 instances, it happens 39 times that (R_Q) leads to a better bound than any of the other methods. Note that (R_S) is dominant in gap over (R_Q) by construction, but in these 39 cases the separation-based approach (R_S^c) is not able to identify the right RLT-inequalities within a reasonable amount of effort. This difference to the previous section can in part be explained by the presence of binary variables in many instances, see also Remark 2. We stress though that the sheer presence of binary variables is not a sufficient indicator for explaining the observed phenomenon, as there are pure binary instances where a complete linearization has the worst bound of all methods. Providing such an indicator, one that divides instances into those where a complete linearization is likely to lead to a better bound than other methods that keep quadratic structure in the reformulation, is outside the scope of this paper. We mention [11] as a possible direction here.

We still note that in two thirds of the cases, we can achieve a better dual bound by not performing a complete linearizaton, and among these methods, we still see a dominance relation in average gap closure of (R_S^c) over (R_μ) over (R_λ). The dominance of (R_μ) over (R_λ) is not as clear as in the previous section. This is mostly due though to a series of problems with an optimal objective of zero on which (R_λ) performs better, and on which a small absolute gap results in large relative gaps. (R_μ) still wins in 101 cases over (R_λ) in the sense that its gap closure is smaller, while the latter wins 27 times.

(R_S^i) is included again in order to demonstrate the importance that the separation of RLT-inequalities can have. The trends on reformulation times persist, and the cost of solving SDPs becomes more evident also when looking at (R_μ). We then note again that the resulting problem sizes somewhat divide (R_λ) and (R_μ) from (R_Q) and (R_S^c), although it can happen that (R_Q) leads to a much smaller reformulation than the other methods. The nevertheless larger geometric mean in the non-zeroes of (R_Q) is also reflected in comparing its geometric mean of the total times (including LP-time) with those of (R_λ) and (R_μ), just as in the previous section.

Conclusions

We set out to shed light on the question of how useful the reformulations, especially SDP-based ones, can be in an off-the-shelf solver when solving instances to optimality or to some predefined optimality gap. We have seen that performing some of the reformulation methods, for example at the root of a Branch-and-Bound tree, can lead to excessive time or memory consumption, which one usually would like to avoid in such a setting. However, we have also seen that effective work limits can be imposed to prevent this from happening. In MOSEK, we have implemented yet more restrictive work limits than those used in the previous section, so that performing a reformulation should never exceed a few seconds, even on very large instances. The subset of 128 instances that are amenable to all four reformulations from the previous section, for example, is therewith further reduced, but only by roughly 20%. All of the four methods thus represent a value that can be added to a solver by exposing the choice of the reformulation method, if amenable, through a user parameter. We stress particularly that we do observe instance classes where the SDP-based method (R_S^c) can systematically outperform all other ones. Also (R_μ) often gives an advantage over (R_λ) due to a better dual bound with almost identical reformulation sizes, albeit paid with a slightly higher reformulation time.

It would be desirable to have an automatic choice of the best reformulation method for a given instance. We have implemented some heuristic rules that try to take into account the characteristics of each reformulation discussed in the previous section. This heuristic choice is of course far from perfect, and more data-driven methods like the one in [11] are likely to be of value here. We point out, however, that on the initial set of 143 QPLIB instances from the previous section, each of the methods (R_Q), (R_λ), (R_μ) and (R_S^c) is chosen at least once.

References

1. FICO Xpress Solver. https://www.fico.com/en/products/fico-xpress-solver, accessed October-23-2020
2. Gurobi Optimizer. https://www.gurobi.com/products/gurobi-optimizer/, accessed October-23-2020
3. IBM ILOG CPLEX. https://www.ibm.com/products/ilog-cplex-optimization-studio, accessed October-23-2020
4. MOSEK. https://www.mosek.com/products/mosek/, accessed October-23-2020
5. Adams, W., Sherali, H.: A tight linearization and an algorithm for zero-one quadratic programming problems. Management Science **32**, 1274–1290 (1986)
6. Belotti, P.: Couenne: A users manual. Tech. rep., Lehigh University (2009)
7. Billionnet, A., Elloumi, S., Lambert, A.: Extending the QCR method to general mixed-integer programs. Math. Programming **131**, 381–401 (2012)
8. Billionnet, A., Elloumi, S., Lambert, A.: Exact quadratic convex reformulations of mixed-integer quadratically constrained problems. Math. Programming **158**, 235–266 (2016)
9. Billionnet, A., Elloumi, S., Plateau, M.: Improving the performance of standard solvers for quadratic 0-1 programs by a tight convex reformulation: The QCR method. Discrete Applied Mathematics **157**(6), 1185–1197 (2009)
10. Bliek, 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. 171–180 (2014)
11. Bonami, P., Lodi, A., Zarpellon, G.: Learning a classification of mixed-integer quadratic programming problems. In: van Hoeve, W.J. (ed.) Integration of Constraint Programming, Artificial Intelligence, and Operations Research. CPAIOR 2018. pp. 595–604. Springer (2018)
12. Dong, H.: Relaxing nonconvex quadratic functions by multiple adaptive diagonal perturbations. SIAM J. on Optim. **26**(3), 1962–1985 (2016)
13. Dong, H., Lou, Y.: Compact disjunctive approximations to nonconvex quadratically constrained programs. Tech. rep. (2018)
14. Elloumi, S., Lambert, A.: Global solution of non-convex quadratically constrained quadratic programs. Optimization Methods and Software **34**(1), 98–114 (2019)
15. Furini, F., Traversi, E., Belotti, P., Frangioni, A., Gleixner, A., Gould, N., Liberti, L., Lodi, A., Misener, R., Mittelmann, H., Sahinidis, N., Vigerske, S., Wiegele, A.: QPLIB: A library of quadratic programming instances. Mathematical Programming Computation (11), 237–265 (2018)
16. Hammer, P.L., Rubin, A.: Some remarks on quadratic programming with 0-1 variables. RAIRO **3**, 67–79 (1970)
17. McCormick, G.P.: Computability of global solutions to factorable nonconvex programs: Part I Convex underestimating problems. Math. Programming **10**(1), 147–175 (1976)
18. Misener, R., Floudas, C.A.: ANTIGONE: algorithms for continuous/integer global optimization of nonlinear equations. Journal of Global Optimization **59**(2–3), 503–526 (2014)
19. Sahinidis, N.: BARON: A general purpose global optimization software package. Journal of Global Optimization **8**, 201–205 (1996)
20. Vigerske, S., Gleixner, A.: SCIP: Global Optimization of Mixed-Integer Nonlinear Programs in a Branch-and-Cut Framework. Optimization Methods and Software **33**(3), 563–593 (2018)