Università di Pisa

Dipartimento di Informatica

Technical Report

A Binarisation Heuristic for Non-Convex Quadratic Programming with Box Constraints

Laura Galli Adam N. Letchford

February 2018, Revised May 2018

A Binarisation Heuristic for Non-Convex Quadratic Programming with Box Constraints

Laura Galli[*] Adam N. Letchford[†]

February 2018, Revised May 2018

Abstract

Non-convex quadratic programming with box constraints is a fundamental problem in the global optimization literature, being one of the simplest $\mathcal{NP}$-hard nonlinear programs. We present a new heuristic for this problem, which enables one to obtain solutions of excellent quality in reasonable computing times. The heuristic consists of four phases: binarisation, convexification, branch-and-bound, and local optimisation. Some very encouraging computational results are given.

Keywords: Global optimisation, heuristics, integer programming.

1 Introduction

Non-convex quadratic programming with box constraints (QPB) is the problem of minimizing a nonconvex quadratic function subject to lower and upper bounds on the variables. An instance takes the form:

$$\min\left\{x^TQx + c\cdot x:\ \ell \le x_i \le u\right\},$$

where $Q \in \mathbb{Q}^{n\times n}$ and $c, \ell, u \in \mathbb{Q}^n$. As usual in the literature, we assume that (a) the box constraints take the simple form $x \in [0,1]^n$, and (b) Q is symmetric. Any instance not satisfying these properties can be easily transformed into one that does.

When Q is positive semidefinite (psd), QPB can be solved quickly using convex programming techniques. In general, however, it is $\mathcal{NP}$-hard in the strong sense. It has received much attention in the global optimization literature (e.g., [3–6, 9–11, 13, 14, 24, 27, 30, 31, 33]), being one of the simplest $\mathcal{NP}$-hard nonlinear programs. The current leading exact algorithms are the ones in [9, 27].

[*]Dipartimento di Informatica, Università di Pisa, Largo B. Pontecorvo 3, 56127 Pisa, Italy. E-mail: `laura.galli@unipi.it`

[†]Department of Management Science, Lancaster University, Lancaster LA1 4YX, United Kingdom. E-mail: `A.N.Letchford@lancaster.ac.uk`

In this paper, we present a new approach to QPB. It is heuristic in nature, but solves a subproblem by branch-and-bound, and therefore can be viewed as a "matheuristic" (see [28]). The heuristic exploits the fact that there now exist good software packages for solving mixed 0-1 convex quadratic programs. It consists of four phases: binarisation, convexification, branch-and-bound and local optimisation. In our computational experiments, on both standard test instances and new ones, the heuristic consistently found solutions of excellent quality in reasonable computing times.

The structure of the paper is as follows. The literature is reviewed in Section 2, the heuristic is described in Section 3, the computational results are given in Section 4, and some concluding remarks are made in Section 5. Throughout the paper, we let N denote $\{1, \ldots, n\}$.

2 Literature Review

We now review the relevant literature. For the purposes of exposition, we cover QPB itself in Subsection 2.2. Subsections 2.1 and 2.3 are concerned with related problems: unconstrained binary quadratic programming and integer quadratic programming. For brevity, we mention only works of direct relevance.

2.1 Unconstrained binary quadratic programming

Unconstrained binary quadratic programming (UBQP) is like QPB, but the variables are binary instead of continuous. It was shown in [22] that UBQP is equivalent to the well-known *max-cut* problem. Since max-cut is $\mathcal{NP}$-hard in the strong sense [18], so is UBQP. Both max-cut and UBQP have been studied in great depth; see [15].

Fortet [16] proposed the following linearisation approach. For $1 \leq i < j \leq n$, replace the product $x_i x_j$ with a new binary variable y_{ij}, and add the constraints $y_{ij} \leq x_i$, $y_{ij} \leq x_j$ and $y_{ij} \geq x_i + x_j - 1$. $\big($There is no need to define y_{ij} when $i = j$, since $x_i^2 = x_i$.$\big)$ The result is a 0-1 LP with $O\big(n^2\big)$ variables and constraints.

Glover [19] proposed a different linearisation. For $i \in N$, define a new continuous variable, say w_i, representing $x_i \sum_{j \in N} q_{ij} x_j$, and replace the objective function with $\sum_{i \in N} w_i$. For all $i \in N$, compute a lower bound L_i and an upper bound U_i on the value of w_i. (One can use, e.g., $\sum_{j \in N} \min\{0, q_{ij}\}$ and $\sum_{j \in N} \max\{0, q_{ij}\}$, respectively.) Then add the following constraints for $i \in N$:

$$\begin{aligned} L_i x_i \quad &\leq w_i \leq \quad U_i x_i \\ \sum_{j=1}^{n} q_{ij} x_j - U_i(1 - x_i) \quad &\leq w_i \leq \quad \sum_{j=1}^{n} q_{ij} x_j - L_i(1 - x_i). \end{aligned}$$

The resulting mixed 0-1 LP has only $O(n)$ variables and constraints.

A third approach is to make the matrix Q psd, if necessary, by adding terms of the form $\lambda_i(x_i^2 - x_i)$ to the objective function, where $\lambda \in \mathbb{Q}^n$. One can then solve the resulting convex UBQP instance via branch-and-bound with convex QP relaxations. Hammer & Rubin [23] proposed to set all of the λ_i to the minimum Eigenvalue of Q multiplied by -1. A more sophisticated method to determine the λ_i is given in [7].

2.2 Quadratic programming with box constraints

Any UBQP instance can be transformed to a QPB instance, by adding the penalty $M \sum_{i \in N} \left(x_i^2 - x_i\right)$ to the cost function, where M is a large positive constant. This implies that QPB is $\mathcal{NP}$-hard in the strong sense. Good surveys of QPB can be found in [3, 12].

To our knowledge, the first exact algorithm for QPB was due to McCormick [29]. It is a spatial branch-and-bound algorithm based on LP relaxation. The LP relaxation uses a continuous variable y_{ij} for $1 \le i \le j \le n$, representing $x_i x_j$. The relaxation can be strengthened using various valid inequalities; see, e.g., [11, 33].

Hansen *et al.* [24] presented a spatial branch-and-bound algorithm that works in the space of the original x variables. This algorithm has a clever branching rule based on the signs of the partial derivatives of the objective function. It also performs extensive variable fixing, based on the following proposition:

Proposition 1 (Hansen *et al.*, 1993) *Let $N^- = \big\{i \in N : Q_{ii} \le 0\big\}$. There exists an optimal QPB solution in which $x_i \in \{0,1\}$ for all $i \in N^-$.*

We will use this proposition to good effect in Subsection 3.1.

An & Tao [2] presented another spatial branch-and-bound algorithm for QPB, based on *difference-of-convex* (DC) programming (see [25]). This algorithm was recently improved in [27].

A completely different approach was proposed by Vandenbussche & Nemhauser [30, 31]. They used first-order KKT conditions to convert QPB into an LP with complementarity constraints, and use complementarity to branch.

There are also several papers on SDP relaxations of QPB (e,g., [4–6, 13]). Burer & Vandenbussche [13] used SDP relaxation together with complementarity-based branching, as in [31], to obtain a finite branch-and-bound algorithm. An enhanced version was presented in [10].

Finally, we mention Bonami *et al.* [9], which presents an exact algorithm based on spatial branch-and-bound with convex QP relaxations.

2.3 Integer quadratic programming

UBQP is a special case of *integer quadratic programming* (IQP) with bounded variables. Using an approach of Watters [32], one can reduce bounded IQP to 0-1 LP. The approach is in two steps. The first step is what we call "binarisation". Suppose that $x_i \in \mathbb{Z} \cap [0, u_i]$, where u_i is a positive integer. We replace x_i with

$$\sum_{s=0}^{\lfloor \log_2 u_i \rfloor} 2^s \, \tilde{x}_{is},$$

where the $\tilde{x}_{is}$ are new binary variables. Applying binarisation to all variables, we reduce IQP to 0-1 QP. In the second step, we use a method such as that of Fortet [16] to reduce 0-1 QP to 0-1 LP.

A few papers have used binarisation to convert bounded *mixed-integer bilinear* programs into mixed 0-1 LPs [19–21]. In our recent paper [17], we show how to extend the approaches in [19–21,32] to bounded mixed-integer *quadratic* programs (MIQPs). A related paper is Billionnet *et al.* [8], which uses binarisation to convert bounded MIQPs into convex mixed 0-1 QPs.

3 Our Approach

Although QPB and UBQP are both $\mathcal{NP}$-hard, QPB tends to be much more challenging in practice. This led us to consider a heuristic approach to QPB which solves UBQP as a subproblem. We describe this approach in the following four subsections.

3.1 Binarisation

The first step is to use binarisation to create a UBQP instance that "approximates" the QPB instance. From Proposition 1, the variables with $Q_{ii} \leq 0$ can be declared binary immediately. For the remaining variables, we use the following approach. Let p be a positive integer parameter. For $s = 0, \ldots, p-1$, introduce a binary variable z_{is}. Then eliminate x_i using the following identity:

$$x_i = \frac{1}{2^p - 1} \sum_{s=0}^{p-1} 2^{p-1} \cdot z_{is}. \qquad (1)$$

The effect of this is to restrict x_i to take a value that is a multiple of $1/(2^p - 1)$. Of course, this restriction can cause optimal solutions to be lost. Nevertheless, we will see in the next section that it works well in practice.

3.2 Convexification

Now that we have a UBQP instance, the next step is to make it convex. From the results mentioned in Subsection 2.1, there are three options available:

1. Use the approach of Fortet [16], and convert the UBQP into a 0-1 LP. Since the UBQP already has $O(np)$ variables, the resulting 0-1 LP has $O\big(n^2p^2\big)$ variables and constraints.

2. Use the approach of Glover [19] to convert the UBQP into a mixed 0-1 LP. The resulting mixed 0-1 LP has $O(np)$ binary variables, continuous variables and constraints.

3. Use the approach of Hammer & Rubin [23], to convexify the cost function of the UBQP directly. This yields a convex UBQP with $O(np)$ variables.

3.3 Branch-and-bound

At this point, we have a 0-1 LP, mixed 0-1 LP or convex UBQP. To solve the 0-1 LP or mixed 0-1 LP, one can use a standard branch-and-bound (or branch-and-cut) solver. As for the convex UBQP, we found that it was best to convert it into a 0-1 *second order cone program* (SOCP), using the standard transformation (see, e.g., [1]). We could then feed the 0-1 SOCP into the `CPLEX` mixed-integer SOCP solver. We remark that this solver itself provides two algorithmic options. In one, the SOCP relaxations are solved via an interior-point method (IPM). In the other, they are solved via a simplex-based cutting-plane method.

We conducted some preliminary experiments to gain some insight into the relative performance of the four different options (i.e., 0-1 LP, mixed 0-1 LP, 0-1 SOCP with IPM, and 0-1 SOCP with simplex). We found that, in all cases, the IPM option was at least an order of magnitude faster than the other three. This is rather surprising, given that re-optimisation after branching is much harder for IPMs than for the simplex method. We do not have a convincing explanation for this phenomenon.

3.4 Local optimisation

The optimal solution to the convexified problem can be converted into a feasible solution $x^* \in [0,1]^n$ to the original QPB instance, using the mapping (1) where necessary. Along with the feasible solution, of course, we get an upper bound.

Observe that x^* is not guaranteed even to be a local minimum for the original QPB instance. Our fourth and final phase is therefore to move from x^* to a nearby local minimum. To do this, we use the open-source non-linear programming solver `IPOPT` [26]. As we understand it, `IPOPT` is a primal-dual interior-point solver with a logarithmic barrier function, in which a line-search is conducted in each Newton iteration. As in the case of `CPLEX`, we use default parameter settings.

Actually, there is a complication: `IPOPT` needs to start from solutions that are in the strict interior of the feasible region, but almost all of the solutions obtained by the branch-and-bound procedure lay on the boundary. To deal with this, we simply moved the initial x^* a little towards the centre of the unit hypercube, before passing it to `IPOPT`. More precisely, we changed x_i^* to $0.9x_i^* + 0.05$ for $i = 1, \ldots, n$.

4 Computational Results

In order to ascertain the potential of our heuristic, we ran experiments, using the callable library of `CPLEX` 12.8 to solve all 0-1 LPs, mixed 0-1 LPs and 0-1 SOCPs, and `IPOPT` 3.11.8 to perform the local optimisation step. The experiments were performed on a 64-bit 2.3 Ghz AMD Opteron 6376 processor with 16Gb RAM, under the Ubuntu 12.4 operating system. All programs were implemented in C++ and compiled with gcc 4.4.3.

4.1 Vandenbussche-Nemhauser instances

We began with the QPB instances described in Vandenbussche & Nemhauser [31], which we call "`VN`" instances. They were created as follows: given a *density* $\Delta \in (1, 100]$, each entry in Q and c is set to zero with probability $(100 - \Delta)\%$, and to a random integer between -50 and 50 with probability $\Delta\%$. For 30 different combinations of n and $\Delta\%$, there are three random instances. So there are 54 instances in total. These instances, which are all of maximisation type, are available at `http://sburer.github.io/projects.html`. Their optimal solution values were kindly given to us by Kurt Anstreicher.

First, we report some results concerned with the quality of the QPB solutions found by our heuristic. These results are in Table 1. Each row corresponds to a particular combination of n and Δ. For numbers of bits ranging from 1 to 5, we report the average, over the three instances of the given type, of the percentage gap (i.e., the difference between our lower bound and the optimum, expressed as a percentage of the optimum). For interest, we also report the gaps for solutions obtained using `IPOPT` alone, using five different random feasible initial solutions as starting points. Also, in Table 2, we report the gaps obtained by our heuristic when the local optimisation step is switched off.

We were surprised to see that solutions of excellent quality are obtained with $p = 1$, even when local optimisation is switched off. Not only that, but using additional bits is of little benefit. A partial explanation is that, for these instances, the number of positive Q_{ii} values is around $n\Delta/200 \leq n/2$. Then, by Proposition 1, we can expect the majority of the variables to take binary values at the optimum. It is also apparent that our heuristic consistently yields solutions of much higher quality than those found with `IPOPT` alone.

For brevity, we report running times only for the fastest algorithm (called "0-1 SOCP and IPM" in Subsection 3.3). These times are shown in Table 3. All times are in seconds, and each figure is the average over three instances.

		Number of bits (p)					
n	Δ	1	2	3	4	5	IPOPT
20	100	0.00	0.00	0.00	0.00	0.00	1.46
30	60	0.02	0.02	0.02	0.02	0.02	1.14
30	70	0.00	0.00	0.00	0.00	0.00	2.33
30	80	0.02	0.02	0.02	0.02	0.02	0.87
30	90	0.00	0.00	0.00	0.00	0.00	0.00
30	100	0.02	0.00	0.00	0.00	0.00	1.22
40	30	0.00	0.00	0.00	0.00	0.00	0.57
40	40	0.04	0.04	0.04	0.04	0.04	2.55
40	50	0.00	0.00	0.00	0.00	0.00	0.49
40	60	0.00	0.00	0.00	0.00	0.00	0.48
40	70	0.01	0.01	0.01	0.01	0.01	0.01
40	80	0.00	0.00	0.00	0.00	0.00	0.01
40	90	0.00	0.00	0.00	0.00	0.00	0.83
40	100	0.03	0.00	0.00	0.00	0.00	0.00
50	30	0.00	0.00	0.00	0.00	0.00	0.42
50	40	0.00	0.00	0.00	0.00	0.00	0.37
50	50	0.00	0.00	0.00	0.00	0.00	2.53
60	20	0.00	0.00	0.00	0.00	0.00	1.74

Table 1: Average percentage gaps for `VN` instances

As one might expect, the time increases with both p and Δ. We remark that the time taken by the local optimisation step was negligible in all cases.

A natural question is how the running time of our heuristic compares with that of the leading exact algorithms, described in [9, 27]. Due to the different machines used, a precise comparison is difficult. In general, our impression is that the two-bit version of our algorithm is around one order of magnitude faster that the exact algorithms. The four-bit version has a comparable running time.

4.2 Other instances

Burer and Vandenbussche [13] created 36 larger instances using the same scheme that was used to create the `VN` instances. These instances have $n \in \{70, 80, 90, 100\}$ and $\Delta \in \{25, 50, 75\}$. The results obtained with these instances were very similar to those obtained for the `VN` instances. In particular, just one bit, plus local optimisation, was enough to obtain solutions within 0.01% of optimal for all instances apart from one. (For the remaining instance, which was the third instance with $n = 100$ and $\Delta = 75$, `CPLEX` ran into memory difficulties.) By comparison, the average gap for the solutions found by `IPOPT` was around 0.82%.

In a sense, then, the `VN` and `BV` instances are relatively "easy" for our

		Number of bits (p)				
n	Δ	1	2	3	4	5
20	100	0.00	0.00	0.00	0.00	0.00
30	60	0.03	0.03	0.02	0.02	0.02
30	70	0.02	0.01	0.00	0.00	0.00
30	80	0.05	0.03	0.02	0.02	0.02
30	90	0.06	0.00	0.00	0.00	0.00
30	100	0.05	0.01	0.00	0.00	0.00
40	30	0.00	0.00	0.00	0.00	0.00
40	40	0.04	0.04	0.04	0.04	0.04
40	50	0.11	0.01	0.00	0.00	0.00
40	60	0.03	0.00	0.00	0.00	0.00
40	70	0.01	0.01	0.01	0.01	0.01
40	80	0.00	0.00	0.00	0.00	0.00
40	90	0.00	0.00	0.00	0.00	0.00
40	100	0.05	0.00	0.00	0.00	0.00
50	30	0.01	0.01	0.00	0.00	0.00
50	40	0.01	0.01	0.00	0.00	0.00
50	50	0.08	0.00	0.00	0.00	0.00
60	20	0.00	0.00	0.00	0.00	0.00

Table 2: Average percentage gaps for `VN` instances: without local optimisation

		Number of bits (p)				
n	Δ	1	2	3	4	5
20	100	0.26	0.87	3.85	8.34	23.26
30	60	1.12	3.61	8.92	14.33	32.40
30	70	1.01	4.35	10.45	24.27	55.05
30	80	1.07	4.00	7.05	18.62	34.55
30	90	1.81	7.53	21.14	51.41	107.67
30	100	3.67	14.44	43.63	121.65	272.76
40	30	0.10	0.14	0.14	0.36	0.57
40	40	1.41	2.67	4.45	10.72	16.87
40	50	1.65	3.39	5.40	13.47	19.38
40	60	2.35	7.80	23.20	61.86	137.63
40	70	3.95	13.91	49.99	148.18	320.25
40	80	4.70	19.33	57.39	176.89	329.50
40	90	5.70	18.54	57.89	167.35	253.95
40	100	14.51	107.69	424.62	1742.30	4039.57
50	30	0.95	2.16	2.60	2.72	2.97
50	40	4.48	9.39	19.79	21.76	31.39
50	50	16.93	24.49	73.33	228.02	211.74
60	20	1.16	1.46	1.60	1.92	2.58

Table 3: Average running times for VN instances

approach. Following the suggestion of an anonymous referee, we attempted to modify the VN instances to make them harder. Specifically, for each instance, we computed the maximum Eigenvalue of the profit matrix Q, which we denote by λ. We then added

$$\frac{\lambda}{2} \sum_{i \in N} \left(x_i - x_i^2\right)$$

to the profit function. The effect of this modification is to make the instances "closer" to being concave, without actually making them concave. (Intuitively, it also makes the extreme points of the box less "attractive" than the points in the interior.) Surprisingly, we found that, for all but two of the modified instances, both IPOPT and the one-bit version of our heuristic found the optimal solution. The gaps for the two remaining instances are given in Table 4.

5 Conclusion

Non-convex quadratic programming with box constraints is very hard to solve to proven optimality, in both theory and practice. We have presented a heuristic that is easy to understand and (fairly) easy to implement using

		Without local opt.			With local opt.			
n	Δ	$p=1$	$p=2$	$p=3$	$p=1$	$p=2$	$p=3$	IPOPT
30	70	4.81	0.69	0.11	1.32	0.00	0.00	4.63
40	50	2.81	0.26	0.08	0.00	0.00	0.00	0.62

Table 4: Average percentage gaps for the two hard “near-concave” instances

readily available software. The computational results show that our heuristic finds solutions of remarkably good quality, in reasonable computing times.

An interesting topic for future research is to explain, perhaps using probabilistic arguments, why (near-)optimal solutions can often be found at extreme points of the box, even for instances that are neither convex nor concave. It would also be interesting to extend our heuristic to the case of general nonconvex quadratic programming.

Acknowledgement

We thank Kurt Anstreicher and Sam Burer for providing us with the optimal solution values for several benchmark instances.

References

[1] F. Alizadeh & D. Goldfarb (2003) Second-order cone programming. *Math. Program.*, 95, 3–51.

[2] L.T.H. An & P.D. Tao (1998) A branch and bound method via d.c. optimization algorithms and ellipsoidal technique for box constrained nonconvex quadratic problems. *J. Glob. Optim.*, 13, 171–206.

[3] P.L. De Angelis, P.M. Pardalos & G. Toraldo (1997) Quadratic programming with box constraints. In I.M. Bomze, T. Csendes, R. Horst & P.M. Pardalos (eds.) *Developments in Global Optimization.* Dordrecht: Kluwer.

[4] K.M. Anstreicher (2009) Semidefinite programming versus the reformulation-linearization technique for non-convex quadratically constrained quadratic programming. *J. Glob. Optim.*, 43, 471–484.

[5] K.M. Anstreicher (2012) On convex relaxations for quadratically constrained quadratic programming. *Math. Program.*, 136, 233–251.

[6] K.M. Anstreicher & S. Burer (2010) Computable representations for convex hulls of low-dimensional quadratic forms. *Math. Program.*, 124, 33–43.

[7] A. Billionnet & S. Elloumi (2007) Using a mixed-integer quadratic programming solver for the unconstrained quadratic 0–1 Problem. *Math. Program.*, 109, 55–68.

[8] A. Billionnet, S. Elloumi & A. Lambert (2012) Extending the QCR method to general mixed-integer programs. *Math. Program.*, 131, 381–401.

[9] P. Bonami, O. Günlük & J. Linderoth (2017) Solving box-constrained nonconvex quadratic programs. *Working paper*, available at Optimization Online.

[10] S.A. Burer & J. Chen (2012) Globally solving nonconvex quadratic programming problems via completely positive programming. *Math. Program. Comput.*, 4, 33–52.

[11] S.A. Burer & A.N. Letchford (2009) On non-convex quadratic programming with box constraints. *SIAM J. Optim.*, 20, 1073–1089.

[12] S.A. Burer & A.N. Letchford (2012) Non-convex mixed-integer nonlinear programming: a survey. *Surveys in Oper. Res. & Mgmt. Sci.*, 17, 97–106.

[13] S.A. Burer & D. Vandenbussche (2007) Globally solving box-constrained nonconvex quadratic programs with semidefinite-based finite branch-and-bound. *Comput. Optim. Appl.*, 43, 181–195.

[14] T.F. Coleman & L.A. Hulbert (1989) A direct active set algorithm for large sparse quadratic programs with simple bounds. *Math. Program.*, 45, 373–406.

[15] M.M. Deza & M. Laurent (1997) *Geometry of Cuts and Metrics.* Springer: Berlin.

[16] R. Fortet (1959) L'Algèbre de Boole et ses applications en recherche opérationnelle. *Cahiers Centre Etudes Rech. Oper.*, 4, 5–36.

[17] L. Galli & A.N. Letchford (2017) Using bit representation to improve LP relaxations of mixed-integer quadratic programs. *Working paper*, Department of Management Science, Lancaster University.

[18] M.R. Garey, D.S. Johnson & L. Stockmeyer (1976) Some simplified $\mathcal{NP}$-complete graph problems. *Theor. Comp. Sci.*, 1, 237–267.

[19] F. Glover (1975) Improved linear integer programming formulations of nonlinear integer problems. *Mgmt. Sci.*, 22, 455–460.

[20] O. Günlük, J. Lee & J. Leung (2012) A polytope for a product of real linear functions in 0/1 variables. In J. Lee & S. Leyffer (eds) *Mixed Integer Nonlinear Programming*, pp. 513–529. New York: Springer US.

[21] A. Gupte, S. Ahmed, M.S. Cheon & S. Dey (2013) Solving mixed integer bilinear problems using MILP formulations. *SIAM J. Optim.*, 23, 721–744.

[22] P.L. Hammer (1965) Some network flow problems solved with pseudo-Boolean programming. *Oper. Res.*, 13, 388–399.

[23] P.L. Hammer & A.A. Rubin (1970) Some remarks on quadratic programming with 01 variables. *RAIRO*, 3, 67–79.

[24] P. Hansen, B. Jaumard, M. Ruiz & J. Xiong (1993) Global minimization of indefinite quadratic functions subject to box constraints. *Naval Res. Log. Quart.*, 40, 373–392.

[25] R. Horst & N.V. Thoai (1999) DC programming: overview. *J. Optim. Th. Appl.*, 103, 1–43.

[26] `IPOPT` (Interior-Point Optimizer). Managed by A. Wächter & S. Wigerske. Available at `https://projects.coin-or.org/Ipopt`

[27] C. Lu & Z. Deng (2017) DC decomposition based branch-and-bound algorithms for box-constrained quadratic programs. *Optim. Lett.*, to appear.

[28] V. Maniezzo, T. Stützle & S. Stefan (eds.) (2010) *Matheuristics: Hybridizing Metaheuristics and Mathematical Programming.* Annals of Information Systems vol. 10. Springer US.

[29] G.P. McCormick (1976) Computability of global solutions to factorable nonconvex programs: Part I — Convex underestimating problems. *Math. Program.*, 10, 147–175.

[30] D. Vandenbussche & G.L. Nemhauser (2005) A polyhedral study of nonconvex quadratic programs with box constraints. *Math. Program.*, 102, 531–557.

[31] D. Vandenbussche & G.L. Nemhauser (2005) A branch-and-cut algorithm for nonconvex quadratic programs with box constraints. *Math. Program.*, 102, 559–575.

[32] L.J. Watters (1967) Reduction of integer polynomial programming problems to zero-one linear programming problems. *Oper. Res.*, 15, 1171–1174.

[33] Y. Yajima & T. Fujie (1998) A polyhedral approach for nonconvex quadratic programming problems with box constraints. *J. Glob. Optim.*, 13, 151–170.

Supplementary Material

In this supplement, we present some simple results that, under certain conditions, enable one to reduce the domains of some of the variables in a QPB instance. Throughout, we assume that the instance takes the form:

$$\min \left\{ x^T Q x + c \cdot x : \; x \in [0,1]^n \right\},$$

where Q is symmetric. We also let N, N^- and N^+ denote $\{1, \ldots, n\}$, $\big\{ i \in N : Q_{ii} \leq 0 \big\}$ and $N \setminus N^-$, respectively.

We define the following function for each $i \in N$:

$$F(i) = Q_{ii} x_i^2 + c_i x_i + 2 x_i \sum_{j \neq i} Q_{ij} x_j .$$

Note that $F(i)$ is the component of the objective function that involves x_i. We also define the following constants:

$$s_i = 2 \sum_{j \neq i} \min \big\{ 0, Q_{ij} \big\} \text{ and } t_i = 2 \sum_{j \neq i} \max \big\{ 0, Q_{ij} \big\}.$$

We then have the following results.

Proposition 2 *Suppose that $i \in N^-$. If $Q_{ii} + c_i + s_i \geq 0$, we can fix x_i to 0, while retaining at least one optimal solution. If $Q_{ii} + c_i + t_i \leq 0$, we can fix x_i to 1. (If both are true, then we can fix x_i to either value.)*

Proof. From Proposition 1 in Hansen *et al.* (1993), x_i will be binary in any optimal solution. Now, if $x_i = 0$, then $F(i) = 0$. On the other hand, if $x_i = 1$, then

$$F(i) = \big(Q_{ii} + c_i \big) + 2 \sum_{j \neq i} Q_{ij} x_j .$$

Thus, if $x_i = 1$, we have:

$$\big(Q_{ii} + c_i \big) + s_i \leq F(i) \leq \big(Q_{ii} + c_i \big) + t_i .$$

If the term on the left is non-negative, then changing x_i from 0 to 1 can never reduce the cost. Similarly, if the term on the right is non-positive, then changing x_i from 1 to 0 can never reduce the cost. $\square$

Proposition 3 *Suppose that $i \in N^+$. If $c_i + t_i < 0$, we can increase the lower bound on x_i from 0 to*

$$\min \left\{ 1, \frac{-(c_i + t_i)}{2Q_{ii}} \right\},$$

while retaining at least one optimal solution. Moreover, if $2Q_{ii} + c_i + s_i > 0$, we can decrease the upper bound on x_i from 1 to

$$\max \left\{ 0, \frac{-(c_i + s_i)}{2Q_{ii}} \right\}.$$

Proof. Since $i \in N^+$, $F(i)$ is a convex function. Its derivative with respect to x_i is:

$$2Q_{ii}x_i + c_i + 2\sum_{j \neq i} Q_{ij}x_j.$$

Setting the derivative to zero, we find that $F(i)$ is minimised when x_i equals $-\big(c_i + 2\sum_{j\neq i} Q_{ij}x_j\big)/\big(2Q_{ii}\big)$. This quantity lies between $-\big(c_i + s_i\big)/\big(2Q_{ii}\big)$ and $-\big(c_i + t_i\big)/\big(2Q_{ii}\big)$. $\square$