Separable Non-convex Underestimators for Binary Quadratic Programming

Christoph Buchheim and Emiliano Traversi

Fakultät für Mathematik, Technische Universität Dortmund
Vogelpothsweg 87, 44227 Dortmund, Germany
{christoph.buchheim,emiliano.traversi}@tu-dortmund.de

Abstract. We present a new approach to constrained quadratic binary programming. Dual bounds are computed by choosing appropriate global underestimators of the objective function that are separable but not necessarily convex. Using the binary constraint on the variables, the minimization of this separable underestimator can be reduced to a linear minimization problem over the same set of feasible vectors. For most combinatorial optimization problems, the linear version is considerably easier than the quadratic version. We explain how to embed this approach into a branch-and-bound algorithm and present experimental results.

1 Introduction

Many combinatorial optimization problems admit natural formulations as binary quadratic optimization problems. Such problems take the form

$$\begin{array}{ll} \min & f(x) := x^\top Q x + L^\top x \\ \text{s.t.} & x \in X \,, \end{array} \tag{1}$$

where $Q \in \mathbb{R}^{n \times n}$ is a symmetric matrix, $L \in \mathbb{R}^n$ is a vector and $X \subseteq \{0,1\}^n$ is the set of feasible binary vectors. In this paper, we consider problems where the linear counterpart of Problem (1),

$$\begin{array}{ll} \min & c^\top x \\ \text{s.t.} & x \in X \,, \end{array} \tag{2}$$

can be solved efficiently for any vector $c \in \mathbb{R}^n$. We do not make any assumptions on how Problem (2) is solved. In particular, any combinatorial algorithm can be used, a compact linear description (or polynomial-time separation algorithm) for $\operatorname{conv}(X)$ is not required.

Even under this assumption, the quadratic problem (1) is usually NP-hard. This is true, e.g., for the unconstrained case $X = \{0,1\}^n$, where Problem (1) is equivalent to unconstrained quadratic binary optimization and hence to the max-cut problem. To give another example, the quadratic spanning tree problem is NP-hard [1], while the linear counterpart can be solved very quickly, e.g., by Kruskal's algorithm.

V. Bonifaci et al. (Eds.): SEA 2013, LNCS 7933, pp. 236–247, 2013.

The standard approach for solving problems of type (1) is based on linearization. In a first step, a new variable y_{ij} representing the product $x_i x_j$ is introduced for each pair i, j. Then the convex hull of feasible solutions in the extended space is usually approximated either by a polyhedral relaxation or by semidefinite programming (SDP) models, or by a combination of both. The main focus lies on enforcing the connection between x- and y-variables. For the unconstrained case, we point the reader to [5] and the reference therein. In the constrained case, most approaches presented in the literature are highly problem-specific.

A different approach to binary optimization is the QCR technique [2]. Instead of linearizing the problem, it is reformulated as an equivalent binary optimization problem with a *convex* quadratic objective function. This allows to apply more powerful software tailored for convex problems. In particular, it is now possible to solve the continuous relaxation of the problem efficiently. The QCR approach is designed such that this relaxation yields as tight lower bounds as possible.

In this paper, we propose a different approach. It is based on computing underestimators g of the quadratic objective function f. A lower bound on Problem (1) can then be computed by minimizing $g(x)$ over $x \in X$. Unlike most other approaches based on underestimators, we however do not use convex functions in general, but separable non-convex functions. The main idea of our approach is to determine a good separable underestimator g of f in the first step; in the second step we can reduce the separable quadratic function to a linear function exploiting the binarity of all variables. The minimization of $g(x)$ over $x \in X$ can thus be performed by solving Problem (2). Convexity is not required for this approach. The resulting lower bounds are embedded into a branch-and-bound scheme for solving Problem (1) to optimality.

Compared with linearization, the advantage of our approach lies in the fact that we do not need to add any additional variables. Moreover, we do not require any polyhedral knowledge about $\operatorname{conv}(X)$ and do not use any LP solver at all. At the same time, any algorithmic knowledge about the linear problem (2) is exploited directly. Compared with the convexification approach, we have a chance to obtain better lower bounds, since we do not require convexity of the underestimator.

An important question in our approach is how to compute the separable underestimator g. We first fix a point $z \in \mathbb{R}^n$ where $f(z) = g(z)$, i.e., where the underestimator touches the original objective function. Reasonable choices discussed in this paper are the stationary point $\bar{x}$ of f, the origin, and the center of the box $\frac{1}{2}\mathbf{1}$. Under this restriction, we compute a separable quadratic function g that is a global underestimator for f and that maximizes the minimum of $g(x)$ over X, i.e., that yields a best possible lower bound. We show that this task can be accomplished efficiently either by solving a semidefinite program or by applying a subgradient method, depending on z and X.

This paper is organized as follows. In the next section, we formalize the main ideas of our approach. In Section 3, we present strategies to determine separable underestimators yielding best possible lower bounds. In Section 4, we discuss how lower bounds can be improved by taking valid linear equations into account.

Details of our branch-and-bound algorithm are given in Section 5. In Section 6, we evaluate our approach computationally, applying it to unconstrained problems and to instances of the quadratic spanning tree problem. It turns out that the new algorithm, though being very general, can solve problems of medium size in reasonable running time.

2 Notation and Basic Idea

We consider Problem (1) and assume that its linear counterpart, Problem (2), can be solved efficiently for any vector $c \in \mathbb{R}^n$. We will use Problem (2) as a black box in the following. Our main idea is to derive a lower bound for Problem (1) by globally underestimating f by a separable but not necessarily convex function g and then using Problem (2) to compute the bound.

For an arbitrary point $z \in \mathbb{R}^n$, we can rewrite $f(x)$ as

$$f(x) = (x-z)^\top Q(x-z) + (L+2Qz)^\top x - z^\top Qz \; . \tag{3}$$

Now define

$$\begin{aligned} g_z^{(t)}(x) &:= (x-z)^\top \mathrm{Diag}(t)(x-z) + (L+2Qz)^\top x - z^\top Qz \\ &= \sum_{i=1}^n t_i x_i^2 + \sum_{i=1}^n (-2z_i t_i + l_i + 2q_i^\top z) x_i + \sum_{i=1}^n z_i^2 t_i - z^\top Qz \end{aligned}$$

for $t \in \mathbb{R}^n$, where q_i denotes the i-th row of Q. Then $g_z^{(t)}(z) = f(z)$, i.e., the function $g_z^{(t)}$ touches f in the point z. By (3), it is easy to see that the function $g_z^{(t)}$ is a global underestimator of f if and only if $Q \succeq \mathrm{Diag}(t)$. In this case, the desired lower bound can be obtained as

$$\min \; g_z^{(t)}(x) \quad \text{s.t. } x \in X \; . \tag{4}$$

As $X \subseteq \{0,1\}^n$, we can replace Problem (4) by the equivalent problem

$$\min \; l_z^{(t)}(x) \quad \text{s.t. } x \in X \tag{5}$$

where the function

$$\begin{aligned} l_z^{(t)}(x) &:= \sum_{i=1}^n t_i x_i + \sum_{i=1}^n (-2z_i t_i + l_i + 2q_i^\top z) x_i + \sum_{i=1}^n z_i^2 t_i - z^\top Qz \\ &= ((\mathbf{1} - 2z) \cdot t + L + 2Qz)^\top x + z^2 \cdot t - z^\top Qz \end{aligned}$$

is bilinear in $x, t \in \mathbb{R}^n$. Here we use $\cdot$ to denote entrywise multiplication and define $z^2 := z \cdot z$. Note that Problem (5) is of type (2) and can thus be solved efficiently by our assumption.

This approach is feasible for each touching point $z \in \mathbb{R}^n$. Throughout this paper, we concentrate on three different choices of z: the origin, the point $\frac{1}{2}\mathbf{1}$,

and the stationary point $\bar{x} := -\frac{1}{2}Q^{-1}L$ of f (if Q is a regular matrix). In the respective special cases, the function $l_z^{(t)}$ can be simplified as follows:

$$l_z^{(t)}(x) = \begin{cases} x^\top t + L^\top x & \text{if } z = 0 \\ \frac{1}{4}\mathbf{1}^\top t + (L + Q\mathbf{1})^\top x - \frac{1}{4}\mathbf{1}^\top Q\mathbf{1} & \text{if } z = \frac{1}{2}\mathbf{1} \\ ((\mathbf{1} - 2\bar{x}) \cdot x + \bar{x}^2)^\top t - \bar{x}^\top Q\bar{x} & \text{if } z = \bar{x}\ . \end{cases}$$

3 Optimal Separable Underestimators

The choice of t is crucial for the strength of the lower bound resulting from (5). As discussed above, this lower bound is valid for each $t \in \mathbb{R}^n$ with $Q \succeq \mathrm{Diag}(t)$. Our objective is to maximize the lower bound induced by t. In other words, our aim is to solve the problem

$$\begin{array}{ll} \max & \min_{x \in X} l_z^{(t)}(x) \\ \text{s.t.} & Q \succeq \mathrm{Diag}(t)\ . \end{array} \tag{6}$$

In the easiest case $X = \{0,1\}^n$, we have

$$\begin{aligned} \min_{x \in \{0,1\}^n} l_z^{(t)}(x) &= \min_{x \in \{0,1\}^n} ((\mathbf{1} - 2z) \cdot t + L + 2Qz)^\top x + z^2 \cdot t - z^\top Qz \\ &= z^2 \cdot t - z^\top Qz + \sum_{i=1}^n \min\{0, (1 - 2z_i)t_i + l_i + 2q_i^\top z\} \end{aligned}$$

so that Problem (6) reduces to solving the semidefinite program

$$\begin{array}{ll} \max & \sum_{i=1}^n z_i^2 t_i + y_i \\ \text{s.t.} & y_i \le 0 \\ & y_i \le (1 - 2z_i)t_i + l_i + 2q_i^\top z \\ & Q \succeq \mathrm{Diag}(t)\ . \end{array}$$

For general X, Problem (6) can be solved by a subgradient approach; this is discussed in Section 3.1. However, if the chosen touching point is $z = \frac{1}{2}\mathbf{1}$, the problem can again be reduced to solving a single semidefinite program, as explained in Section 3.2.

3.1 Subgradient Method

For general $X \subseteq \{0,1\}^n$, Problem (6) can be solved by a subgradient method. For this, we can model the constraint $Q \succeq \mathrm{Diag}(t)$ by a penalty function, and obtain the following problem:

$$\begin{array}{ll} \max & \min_{x \in X} l_z^{(t)}(x) + \mu \min\{0, \lambda_{\min}(Q - \mathrm{Diag}(t))\} \\ \text{s.t.} & t \in \mathbb{R}^n\ , \end{array} \tag{7}$$

where μ is a suitably large non-negative number. The objective function of (7) is concave, so that a subgradient approach can be used to solve the problem efficiently. The supergradient of

$$\min_{x \in X} \; l_z^{(t)}(x)$$

at a given point t^k can be computed by using the black box (2), as $l_z^{(t^k)}(x)$ is a linear function in x. Given the optimal solution $\hat{x}^k$, the desired supergradient is the gradient of $l_z^{(t)}(\hat{x}^k)$, which is easily computed since $l_z^{(t)}(\hat{x}^k)$ is a linear function also in t. If $\lambda_{\min}(Q - \mathrm{Diag}(t^k)) < 0$, the supergradient of the penalty term can be obtained as $-\mu v^2$, where v is a normalized eigenvector corresponding to the eigenvalue $\lambda_{\min}(Q - \mathrm{Diag}(t^k))$.

The resulting subgradient approach is sketched in Algorithm 1. Note that Algorithm 1 can be stopped at any time. Let t^k be the best solution to Problem (7) obtained so far. If $Q - \mathrm{Diag}(t^k) \succeq 0$, then t^k is also feasible for (6). If $\lambda_{\min}(Q - \mathrm{Diag}(t^*)) < 0$, then a new solution $\bar{t}$ can be obtained by

$$\bar{t} := t^k + \lambda_{\min}(Q - \mathrm{Diag}(t^k))\mathbf{1}$$

and $\bar{t}$ is a feasible solution for (6) by construction.

Algorithm 1. computation of optimal underestimator

input : function f, set X, touching point z, penalty parameter μ, procedure for solving Problem (2)
output: a (near-)optimal solution to Problem (6)

$t^0 \leftarrow \lambda_{\min}(Q)\mathbf{1}$;
$k \leftarrow 0$, STOP$\leftarrow$ false;

while *STOP= false* **do**
  solve $\min_{x \in X} l_z^{(t^k)}(x)$, let $\hat{x}^k$ be the optimal inner solution;
  // using black box to solve the inner problem
  $\Delta t^k \leftarrow \big(\nabla_t \, l_z^{(t)}(\hat{x}^k)\big)(t^k)$; $\lambda \leftarrow \lambda_{\min}(Q - \mathrm{Diag}(t^k))$;
  if $\lambda < 0$ **then**
    choose normalized eigenvector v of $Q - \mathrm{Diag}(t^k)$ to eigenvalue λ;
    $\Delta t^k \leftarrow \Delta t^k - \mu v^2$;
  end
  // computing a supergradient
  if $\Delta t^k \approx 0$ **then**
    STOP$\leftarrow$true; *// t^k is (near-)optimal*
  end
  else
    $t^{k+1} \leftarrow t^k + \Delta t^k$; $k \leftarrow k + 1$;
  end
end

3.2 Box Center as Touching Point

If the touching point z is chosen as $\frac{1}{2}\mathbf{1}$, the optimization problem (6) can be solved more efficiently. In this case, the function

$$l_z^{(t)}(x) = \frac{1}{4}\mathbf{1}^\top t + (L + Q\mathbf{1})^\top x - \frac{1}{4}\mathbf{1}^\top Q\mathbf{1}$$

does not contain any product between x and t. Problem (6) can thus be decomposed as follows:

$$\begin{array}{lll} \max & \frac{1}{4}\mathbf{1}^\top t & + \quad \min \quad (L + Q\mathbf{1})^\top x \quad - \quad \frac{1}{4}\mathbf{1}^\top Q\mathbf{1} \\ \text{s.t.} & Q \succeq \mathrm{Diag}(t) & \quad\;\; \text{s.t.} \quad x \in X \end{array}$$

The first problem is an SDP, while the second problem can be solved by calling the oracle (2) once. In particular, the optimal underestimator only depends on Q in this case, but not on L. This fact can be exploited in our branch-and-bound algorithm, as explained in Section 5.1.

4 Taking Valid Equations into Account

So far, we assumed that we can access the set X of feasible solutions only via the linear optimization oracle (2). This oracle is used in the second step of the lower bound computation, the minimization of the underestimator. In particular, this step implicitly exploits full knowledge about X.

On the other hand, the computation of an underestimator does not exploit any properties of X, we require that $g_z^{(t)}$ globally underestimates f. If it is known that the set X satisfies certain linear equations $Ax = b$, this information can be used to improve the lower bounds significantly: it is enough to require that the function $g_z^{(t)}$ is an underestimator of f on the affine subspace given by $Ax = b$. This leads to a weaker condition on t, which can still be handled efficiently.

More precisely, let $H = \{x \in \mathbb{R}^n \mid Ax = b\}$ be nonempty and choose $w \in H$. Let $v_1, \ldots, v_k$ be an orthonormal basis of the kernel of A and set $V = (v_1 | \ldots | v_k)$, so that $H = \{w + Vy \mid y \in \mathbb{R}^k\}$. We first assume that the touching point z belongs to H, e.g., by defining it as the orthogonal projection of $\frac{1}{2}\mathbf{1}$ to H:

$$z := w + VV^\top(\tfrac{1}{2}\mathbf{1} - w)$$

Now $g_z^{(t)}|_H$ is an underestimator of $f|_H$ if and only if

$$\begin{array}{lll} & (x-z)^\top Q(x-z) \geq (x-z)^\top \mathrm{Diag}(t)(x-z) \;\forall x \in H & \\ \Leftrightarrow & y^\top V^\top QVy \geq y^\top V^\top \mathrm{Diag}(t)Vy & \forall y \in \mathbb{R}^k \\ \Leftrightarrow & V^\top QV \succeq V^\top \mathrm{Diag}(t)V\ . & \end{array}$$

The latter constraint can be used to replace the stronger constraint $Q \succeq \mathrm{Diag}(t)$ both in the subgradient approach and in the SDP based computation of t, potentially yielding tighter lower bounds in both approaches. In the former approach, the penalty term can be replaced by $\min\{0, \lambda_{\min}(V^\top(Q - \mathrm{Diag}(t))V)\}$.

The corresponding supergradient is $-(Vv)^2$, where v is a normalized eigenvector of $V^\top(Q - \mathrm{Diag}(t))V$ corresponding to its smallest eigenvalue.

In the latter approach, the constraint $Q - \mathrm{Diag}(t) \succeq 0$ can be replaced by $V^\top(Q - \mathrm{Diag}(t))V \succeq 0$ and the resulting problem remains a semidefinite program. Note that the dimension of this SDP decreases by $n - k = \mathrm{rk}(A)$. In other words, a bigger rank of A implies a smaller number of variables in the semidefinite program.

5 Branch-and-bound Algorithm

In order to solve Problem (1) exactly, we embed the lower bounds derived in Section 3 into a branch-and-bound framework. We thus need to compute the lower bounds as quickly as possible and for many related problems. In the following, we describe how the ideas presented in the previous sections can be adapted to this situation.

5.1 Branching Strategy

In order to compute lower bounds as quickly as possible, we restrict ourselves in two different ways:

1. We determine an order of variables at the beginning and fix variables always in this order. More precisely, if $x_1, \dots, x_n$ is the chosen order, the next variable to be fixed is the free variable with smallest index. The same idea has been used in [3] and [4].
2. We do not call the subgradient method to compute an optimal t in every node, but try to find one fixed t for each level of the enumeration tree that yields strong lower bounds on average. This reduces the number of oracle calls to one per node.

The reason for accepting Restriction 1 is as follows: by this branching strategy, the reduced matrices Q in the nodes of the enumeration tree only depend on the depth of the node but not on the specific subproblem. Consequently, only n such matrices can appear in the enumeration tree, instead of 2^n when applying other branching strategies. All time-consuming computations concerning this matrix can now be performed in a preprocessing phase. In particular, in combination with Restriction 2, we can now determine one feasible t for all nodes on a given depth in the preprocessing.

More precisely, consider a subproblem on depth d of the enumeration tree. This means that variables $x_1, \dots, x_d$ have been fixed to some values $\alpha \in \{0,1\}^d$ and the resulting objective function in the given node becomes

$$f_\alpha \colon \mathbb{R}^{n-d} \to \mathbb{R}, \quad f_\alpha(x) = x^\top Q_\alpha x + L_\alpha^\top x + c_\alpha \ ,$$

where Q_α is obtained from Q by deleting the first d rows and columns and

$$(L_\alpha)_j := L_{d+j} + 2\sum_{i=1}^{d} \alpha_i Q_{d+j,i}, \quad c_\alpha := \sum_{i,j=1}^{d} \alpha_i \alpha_j Q_{i,j} + \sum_{i=1}^{d} \alpha_i L_i \ .$$

As Q_α only depends on the depth d but not on α, we may denote it by Q_d. The coefficients of L_α and c_α can be computed incrementally in $O(n-d)$ time per node using the recursive formulae

$$\begin{aligned}(L_\alpha)_j &= (L_{(\alpha_1,\ldots,\alpha_{d-1})})_{j+1} + 2\alpha_d Q_{d+j,d} \\ c_\alpha &= c_{(\alpha_1,\ldots,\alpha_{d-1})} + \alpha_d^2 Q_{d,d} + \alpha_d L_d \ .\end{aligned}$$

In Section 3 we showed that for the special touching point $\frac{1}{2}\mathbf{1}$ the optimal lower bound can be computed as

$$\begin{array}{lll}\max & \frac{1}{4}\mathbf{1}^\top t & + \quad \min \quad (L+Q\mathbf{1})^\top x \quad - \quad \frac{1}{4}\mathbf{1}^\top Q\mathbf{1} \ . \\ \text{s.t.} & Q \succeq \mathrm{Diag}(t) & \quad \text{s.t.} \quad x \in X\end{array}$$

The first problem is a semidefinite program that does not depend on L_α or c_α. In other words, the optimal t for all nodes on depth d is the same and can be computed in the preprocessing.

In order to accelerate the computation of lower bounds, we can apply this approach for any choice of a touching point z. Any solution of the SDP

$$\begin{array}{ll}\max & \mathbf{1}^\top t \\ \text{s.t.} & Q \succeq \mathrm{Diag}(t)\end{array} \tag{8}$$

yields feasible lower bounds. In general, the resulting lower bounds are weaker than the bounds obtained from the subgradient method presented in Section 3, but in terms of total running time this approach outperforms the subgradient approach, as only one oracle call per node is necessary.

5.2 Incremental Update for Valid Equations

The fixed order of variables can also be exploited to accelerate the computation of data necessary to handle valid equations. It implies that the induced constraint matrix in a given node again only depends on its depth d in the enumeration tree, it results from deleting the first d columns from A; denote the resulting matrix by A_d. Consequently, the kernel vectors V_d of A_d can be computed in a preprocessing phase again, and the same is true for the matrices $V_d^\top Q_d V_d$ needed in the computation of lower bounds.

On contrary, the induced right hand side of the set of valid equations depends on the fixings applied so far, it turns out to be

$$b_\alpha := b - \sum_{i=1}^{d} \alpha_i A_{\bullet,i} \in \mathbb{R}^m \ .$$

This implies that the projection z_α of some touching point $z_d \in \mathbb{R}^{n-d}$ to the subspace given by $A_d x = b_\alpha$ depends on the specific node and cannot be computed in the preprocessing. However, it can be calculated incrementally, thus avoiding to solve a linear system of equations in every node of the enumeration

tree: in the preprocessing phase we determine vectors $w_0 \in \mathbb{R}^n$ and $y_d \in \mathbb{R}^{n-d}$ for $d = 1, \dots, n$ satisfying

$$Aw_0 = b \text{ and } A_{\bullet,d+1\dots n} y_d = A_{\bullet,d} \text{ for all } d = 1, \dots, n \, .$$

When enumerating the branch-and-bound nodes, we incrementally compute a vector w_α satisfying $A_d x = b_\alpha$ as follows: for $d = 0$, we can use $w_\alpha = w_0$. For $d \geq 1$, we set

$$w_\alpha := (w_{(\alpha_1,\dots,\alpha_{d-1})})_{2\dots n-d+1} + ((w_{(\alpha_1,\dots,\alpha_{d-1})})_1 - \alpha_d) y_d \in \mathbb{R}^{n-d} \, .$$

Then

$$\begin{aligned} A_d w_\alpha &= A_d (w_{(\alpha_1,\dots,\alpha_{d-1})})_{2\dots n-d+1} + ((w_{(\alpha_1,\dots,\alpha_{d-1})})_1 - \alpha_d) A_d y_d \\ &= A_{d-1} w_{(\alpha_1,\dots,\alpha_{d-1})} - \alpha_d A_{\bullet,d} \end{aligned}$$

so that by recursion we obtain

$$A_d w_\alpha = A_0 w_0 - \sum_{i=1}^{d} \alpha_i A_{\bullet,i} = b_\alpha$$

as desired. The projected touching point can now be computed using the formula

$$z_\alpha := w_\alpha + V_d V_d^\top (z_d - w_\alpha)$$

given in Section 4, where $V_d V_d^\top$ can again be computed in the preprocessing. The total running time for computing z_α in a node on depth d using this approach is $O((n-d)^2)$. The time spent in the preprocessing is dominated by the time needed to solve the $n+1$ systems of linear equations determining w_0 and $y_1, \dots, y_n$.

5.3 Application of the Subgradient Method

Inside a branch-and-bound framework, the running time of the subgradient method for computing a vector t, as presented in Section 3, can be cut in several ways. As with any subgradient method, a careful tuning of parameters, such as step length, is important for obtaining a decent rate of convergence. Moreover, in a given node on depth $d \geq 1$, we use the best solution $t^*_{(\alpha_1,\dots,\alpha_{d-1})}$ of the parent node for warmstarting. More precisely, we use the last $n-d$ entries of this solution as initial solution for t^0_α and choose an initial step length that is decreasing with increasing depth d in the enumeration tree.

Furthermore, as every feasible iterate in the subgradient method yields a valid lower bound for our primal problem, we can stop Algorithm 1 as soon as the current lower bound given by (5) exceeds the primal bound, i.e., the objective value of the best known solution of Problem (1).

From a practical point of view, a good strategy is to perform a few reoptimization iterations of Algorithm 1 in every node. An even more restricted approach is to determine the best possible t in the root node and then keep the corresponding underestimator throughout the entire branch-and-bound algorithm, with the necessary adaptions. We will compare these choices in the numerical experiments in Section 6.

6 Experiments

The aim of this section is to determine which variant of our approach yields the most effective underestimator t. As benchmark we use two sets of instances:

- *Unconstrained BQP.*
  We generated a set of random binary instances with $n = 20, 30$. Ten possible levels of convexity of Q are tested: from 10% to 100% of negative eigenvalues. For a given concavity, we randomly generate three different instances for a total of 60 instances.

- *Quadratic Spanning Tree Problem.*
  We generated a set of random graphs $G = (V, E)$ and associated linear and quadratic costs L and Q with uniformly distributed random integer entries, with absolute value in the interval $[1, 100]$. A given instance is characterized by (1) the number of nodes $|V| = 15, 20$, (2) the density $d = 25\%, 50\%, 75\%$ of G, and (3) the percentage of positive coefficients $p = 25\%, 50\%, 75\%$; the matrix Q is dense in all instances. For each combination of parameters we randomly generate three different instances for a total of 54 instances.

For all tests, we use an Intel Xeon E5-2670 processor, running at 2.60 GHz with 64 GB of RAM. Running times are stated in CPU seconds.

As first step, we test different touching points z as explained in Section 2. For each candidate, we solve Problem (8) in the preprocessing phase. As test bed we use the *Unconstrained BQP*. In Figure 1 we present the average number of branch-and-bound nodes for a given percentage of negative eigenvalues. In addition to the results obtained by using the optimal t of Problem (8) (*SDP*), we also report those obtained by using the trivial underestimator $t = -\lambda_{\min}(Q)\mathbf{1}$ (*Triv*). For each policy for t we report the results obtained by fixing the touching point to the origin (0), $\frac{1}{2}\mathbf{1}$ (0.5) or $\bar{x}$ (*stat*). It is obvious from these results that the best choice is $z = \frac{1}{2}\mathbf{1}$ (yellow and green columns): the total number of explored nodes is 10 times and 100 times less than the number of nodes needed with touching point $z = 0$ (blue and red columns) and $z = \bar{x}$ (brown and light blue columns) respectively.

As second step we want to test how t is improved by taking valid inequalities into account; see Section 4. The set *Quadratic Spanning Tree Problem* is used and the (only) valid equation is $\sum_{e \in E} x_e = |V| - 1$. In Table 1 we show how even one single equation is improving the behaviour of the corresponding t. Every line is reporting the number of nodes and computing time (corresponing to an average of three instances). The dimension n of the instances is stated in the second column. With Eq we indicate that t is obtained considering equations and with $NoEq$ the opposite, moreover we report the ratio r between these values. Also in this case the answer is clear: considering equations decreases significantly the number of nodes. E.g., for the larger instances this decreases the number of nodes by a factor of 50 and the solution time by a factor of 20. We finally remark that Table 1 only reports results for smaller instances, because only 15 out of 21 large instances were solved within our time limit of four hours by $NoEq$. Also in

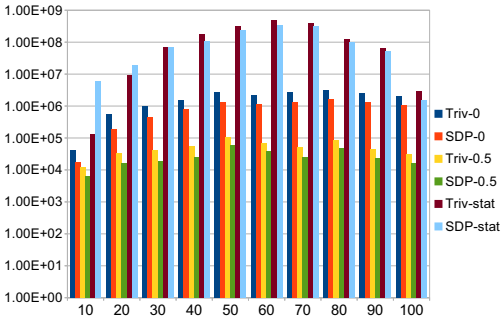


Fig. 1. Touching points comparison

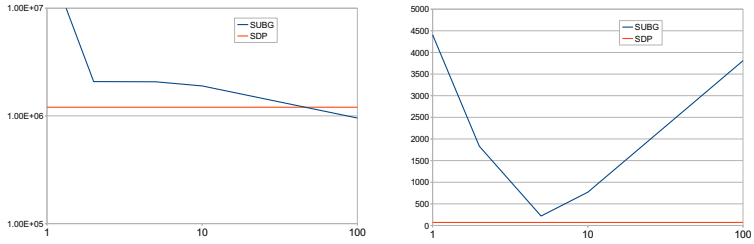
Table 1. Effect of taking equations into account

instances	n	Nodes NoEq	Nodes Eq	Nodes ratio	Time NoEq	Time Eq	Time ratio
qstp_15_25_25	26	2,714.3	2,685.0	1.0	0.4	1.1	0.4
qstp_15_25_50	26	7,514.3	6,747.0	1.1	0.4	1.1	0.4
qstp_15_25_75	26	5,545.7	5,141.7	1.1	0.4	1.0	0.4
qstp_15_50_25	52	456,420.3	138,324.3	3.3	24.0	66.4	0.4
qstp_15_50_50	52	29,846,421.7	1,203,823.0	24.8	163.0	70.3	2.3
qstp_15_50_75	52	983,578,822.3	18,793,411.7	52.3	4452.3	207.0	21.5

this case, using Eq improved the performance, allowing to solve 5 out of the 6 instances unsolved by $NoEq$.

Finally, we test whether updating t during the exploration of the branch-and-bound tree applying Algorithm 1 is better than solving a series of SDPs in the preprocessing. Using the warmstart described in Section 5.3, the values of t in the non-root nodes of the tree are computed by k rounds of Algorithm 1. We tried different settings $k = 1, 2, 5, 10, 100$, but none of them succeeded in improving the overall computation time. In Figure 2, we show how the increase in k affects the total number of nodes and the running time for the set of instances qstp_15_50_50. The red line represents the values obtained by computing t in the preprocessing and the blue line represents the subgradient evolution. As we can see, almost 100 iterations of Algorithm 1 per node are needed in order to improve at least the number of nodes.

The results presented clearly indicate that the best setting is using $\frac{1}{2}\mathbf{1}$ as touching point, fixing the best underestimators t levelwise from the beginning and taking into account valid equations. Obtaining a good solution to Problem (6) is a crucial aspect, additional tests showed also that increasing the tolerance in the SDP solver (and hence allowing worse solutions) provides significantly worse underestimators. This consideration, together with the other results



(a) *nodes* (b) *time*

Fig. 2. Results of the subgradient method for `qstp_15_50_50`

provided in this section, gives an idea about the strong sensitivity of the overall algorithm to the chosen vector t. The importance of valid inequalities makes problems such as the quadratic assignment problem or the quadratic shortest path problem particularly appealing for future applications of our approach.

Acknowledgments. The authors would like to thank Antonio Frangioni for fruitful discussions and suggestions that improved the present paper significantly.

References

1. Assad, A., Xu, W.: The quadratic minimum spanning tree problem. Naval Research Logistics 39(3), 399–417 (1992)
2. Billionnet, A., Elloumi, S., Plateau, M.-C.: 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)
3. Buchheim, C., Caprara, A., Lodi, A.: An effective branch-and-bound algorithm for convex quadratic integer programming. Mathematical Programming (Series A) 135(1-2), 369–395 (2012)
4. Buchheim, C., De Santis, M., Palagi, L., Piacentini, M.: An exact algorithm for quadratic integer minimization using nonconvex relaxations. Technical report, Optimization Online (2012)
5. Palagi, L., Piccialli, V., Rendl, F., Rinaldi, G., Wiegele, A.: Computational approaches to Max-Cut. In: Handbook on Semidefinite, Conic and Polynomial Optimization, pp. 821–849. Springer (2012)