A simplicial decomposition-based branch and price for convex quadratic mixed binary problems

Enrico Bettiol[1], Lucas Létocart[2], Francesco Rinaldi[3], Emiliano Traversi[2]

Abstract

Simplicial Decomposition is a column generation method for solving convex continuous optimization problems. In this work, we show that such a method can be efficiently embedded in a branch-and-price algorithm to solve quadratic convex mixed binary problems. This allows to efficiently reuse the variables generated in one node of the branch-and-bound tree, thus providing a speed-up in terms of computing time. We also analyze the interaction of additional techniques for improving the overall performances of the framework, in particular we introduce a warmstart based on projection of columns. We finally report computational results on quadratic minimum spanning tree problems and quadratic shortest path problems, which show the efficiency of the proposed method, in particular when compared to the commercial solver *Cplex*.

Keywords: Convex Quadratic Programming, Mixed Binary Quadratic Optimization, Simplicial Decomposition, Column Generation, Branch-and-Bound

[1]Fakultät für Mathematik, TU Dortmund, Vogelpothsweg 87, Dortmund, Germany

[2]LIPN, UMR CNRS 7030, Université Sorbonne Paris Nord, 99, avenue Jean-Baptiste Clément, 93430 Villetaneuse, France

[3]Università di Padova, Dipartimento di Matematica, Via Trieste, 63, 35121 Padova, Italy

1. Introduction

Many real-world applications can be modelled as *Mixed Binary Quadratic Problems* (MBQPs). The problems we deal with have the following structure:

$$\begin{aligned} \min\ & f(x) = x^\top Q x + c^\top x \\ \text{s. t.}\quad & Ax \geq b \\ & Cx = d \\ & l \leq x \leq u \\ & x_i \in \{0,1\} \quad \forall i \in I \subseteq \{1,\dots,n\} \end{aligned} \tag{1}$$

with $Q \in \mathbb{R}^{n\times n}$, $c, l, u \in \mathbb{R}^n$, $A \in \mathbb{R}^{m_1\times n}$, $b \in \mathbb{R}^{m_1}$, $C \in \mathbb{R}^{m_2\times n}$, $d \in \mathbb{R}^{m_2}$, $n, m_1, m_2 \in \mathbb{N}$.

We focus on convex problems, so we assume that the Hessian matrix Q is positive semidefinite. Moreover, we assume that the set

$$X = \{x \in \mathbb{R}^n :\ Ax \ \geq\ b,\ \ Cx \ =\ d,\ \ l \leq\ x \leq u\}$$

is non-empty and bounded. Furthermore, among all possible problems of type (1), we are particularly interested in those in which the matrix Q is dense and with the following additional property: there exists an efficient method for minimizing a linear function over the feasible set X, i.e., there exists an efficient linear minimization oracle that, for a given $y \in \mathbb{R}^n$, solves the problem:

$$\min_{x\in X} y^\top x.$$

We present an algorithm for tackling this class of NP-Hard problems that combines a classic Branch and Bound (B&B) strategy with a *Simplicial Decomposition* (SD) type approach (used to handle the continuous relaxation of the problem). We introduce ad-hoc techniques that exploit the combination of SD in a B&B framework and in particular we show how columns generated in one node can be efficiently projected to produce a warmstart for the child nodes and improve the performance of the algorithm. In [1], it is showed that SD can easily handle quadratic programming problems with the aforementioned structure. This is another motivation for our choice to use it to solve the continuous relaxation at a given node of the B&B tree. However, it is worth noticing that the proposed algorithm can handle any convex problem of type (1) and can be easily modified in order to deal with general mixed integer convex optimization problems. We further exploit the

SD structure by implementing several warm-up techniques to quickly solve the nodes in the B&B tree. Within this framework we obtain promising results compared with the state-of-the-art solver *Cplex* on some specific classes of instances.

The paper is organized as follows: in Section 2 we present a state of the art related to binary quadratic problems and decomposition techniques. In Section 3 we introduce Simplicial Decomposition and in Section 4 we show how to efficiently use Simplicial Decomposition in a B&B framework. Finally, in Section 5 we show the performances of our method and Section 6 concludes the paper.

2. State of the art

Classical approaches to solve exactly binary quadratic problems are mainly based on semidefinite relaxations or linearization techniques. A branch-and-bound method for solving binary quadratic problems using semidefinite relaxations is proposed in [2]. In [3], a recent theoretical and computational study of the classical linearization techniques for binary quadratic problems is developed. In [4], the authors present a new linear reformulation to convert a binary quadratically constrained quadratic program into a 0–1 mixed integer linear program. In order to strengthen the relaxation, one could also exploit polyhedral aspects, as in [5], where strong convex valid inequalities are described for conic quadratic mixed 0–1 optimization. For convex problems, [6] provides a complete characterization of the sets that appear in the feasibility version of mixed binary convex quadratic optimization problems. In [7], the authors review convex mixed-integer quadratic programming approaches to deal with single-objective single-period mean-variance portfolio selection problems under real-world financial constraints. Interior point methods can also be used to solve convex quadratic problems, as in [8].

Quadratic Convex Reformulation approaches are another way to tackle mixed integer nonlinear problems (MINLP). These methods have been applied to binary quadratic problems [9], general mixed integer quadratic problems [10] and, recently, to unconstrained binary polynomial problems [11]. Other convexification approaches have been proposed, as in [12], where the methodology relies on a convex reformulation of the proposed MINLP and a branch-and-cut algorithm based on outer approximation cuts, or in [13], where the authors employ quadratic surrogate functions and convexify all the

quadratic inequality constraints to construct quadratic convex reformulation for nonconvex binary quadratically constrained quadratic problems.

Dantzig-Wolfe Reformulation (DWR) is a well known technique used to obtain tight bounds for MILPs (see for example [14, 15, 16]). Its principle is to replace the feasibility region corresponding to a subset of the constraints of a model by the convex hull of its extreme points through an inner representation. A recent research trend is indeed investigating on how to embed automatic decomposition techniques into general purpose MILP solvers [17, 18].

DWR can be applied in principle also to nonlinear mathematical programming models, provided a subset of constraints exists whose corresponding feasible region can be represented as a polyhedron. In fact, the extension of DWR to nonlinear problems has been analyzed in several theoretical papers in the past years (see for example [19, 20, 21]). However, whether or not its application to MINLP may yield successful computational methods is still an open research question.

One of the main issues is the following: the application of DWR leads to a formulation with an exponential number of variables. It can be solved via iterative procedures like Column Generation (we refer the reader to [15] for an extensive review of such a method), but additional conditions have to be fulfilled to ensure convergence.

In this paper, computational experiments will focus on the Quadratic Minimum Spanning Tree (QMST) Problem and on the Quadratic Shortest Path (QSP) Problem. Specific works in order to solve these problems have been proposed in literature. For the QMST problem, [22] firstly introduced the problem and proved NP hardness. Exact algorithms are proposed in [22, 23, 24] and recently in [25]. Lower bounds for QMST are described in [26] and DWR is used in the special case of adjacent only QMST problem [27]. For the QSP problem, one can refer to [28] and [29].

3. Simplicial Decomposition Methods

SD represents a class of methods used for dealing with convex problems. It was first introduced by Holloway in [30] and then further studied in other papers (see, e.g., [31, 32, 33]). A complete overview of this kind of methods can be found in [34].

This method has been developed as an extension of the Frank-Wolfe algorithm, but it can also be seen as an application of Column Generation

to a particular DWR of nonlinear problems. In fact, it is based on the Caratheodory theorem, stating that any point x in the convex hull of a set $X \in \mathbb{R}^n$ can be represented as a convex combination of at most as many elements of X as its dimension (notice that such a number is lower or equal than $n+1$). The method works as follows: it alternates between the exact solution of a reduced problem (*master problem*) that uses an inner approximation of the feasible set (based on the convex hull of a subset of extreme points/columns), and the generation of a new extreme point/column that improves the inner description. Such an extreme point is obtained by minimizing a linearization of the original function (calculated w.r.t. the optimum of the reduced problem) over the original feasible set (*pricing problem*). More specifically, we consider a minimization problem of the following form:

$$\begin{aligned} &\min f(x) \\ &\text{s. t.} \quad x \in X \end{aligned} \tag{2}$$

where f is a continuous, convex function and $X \subset \mathbb{R}^n$ is a convex and compact set.

The column generation is done in the following way: starting from a single point, the domain of a master program at a given iteration is the convex hull of a finite set of affinely independent points, i.e. a simplex, and these points are the solution of the pricing problems solved so far.

In practice, the feasible set X is approximated with the convex hull of an ever expanding finite set $X_k = \{\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_m\}$ where $\tilde{x}_i$, $i = 1, \ldots, m$ are extreme points of X. We denote this set with $conv(X_k)$:

$$conv(X_k) = \{x \mid x = \sum_{i=1}^{m} \lambda_i \tilde{x}_i, \sum_{i=1}^{m} \lambda_i = 1, \lambda_i \geq 0\} \tag{3}$$

At each iteration, it is possible to add new extreme points to X_k in such a way that a function reduction is guaranteed when minimizing the objective function over the convex hull of the new (enlarged) set of extreme points. If the algorithm does not find at least one new point, the solution is optimal and the algorithm terminates.

The use of the proposed method is particularly indicated when the following two conditions are satisfied:

1. Minimizing a linear function over X is much simpler than solving the original nonlinear problem;

2. Minimizing the original objective function over the convex hull of a relatively small set of extreme points is much simpler than solving the original nonlinear problem (i.e. tailored algorithms can be used for tackling the specific problem in our case).

The first condition is needed due to the way a new extreme point is generated. Indeed, this new point is the solution of the following linear programming problem

$$\begin{array}{ll} \min & \nabla f(x_k)^\top (x - x_k) \\ \text{s.t.} & x \in X \end{array} \tag{4}$$

where a linear approximation calculated at the last iterate x_k (i.e. the solution obtained by minimizing f over $conv(X_k)$) is minimized over the original feasible set X.

Below, we report the detailed scheme related to the classical simplicial decomposition algorithm [31, 34, 35] (see Algorithm 1). At a generic iteration k of the simplicial decomposition algorithm, given the set of extreme points X_k, we first minimize f over the set $conv(X_k)$ (Step 1), thus obtaining the new iterate x_k then, at Step 2, we generate an extreme point $\tilde{x}_k$ by solving the linear program (6). Finally, at Step 3, we update X_k.

Algorithm 1 Simplicial Decomposition Algorithm

Initialization: Choose a starting set of extreme points X_0.

For $k = 1, 2, \ldots$

Step 1) Generate iterate x_k by solving the **master problem**

$$\begin{array}{ll} \min & f(x) \\ \text{s.t.} & x \in conv(X_k) \end{array} \tag{5}$$

Step 2) Generate an extreme point $\tilde{x}_k$ by solving the **subproblem**

$$\begin{array}{ll} \min & \nabla f(x_k)^\top (x - x_k) \\ \text{s.t.} & x \in X \end{array} \tag{6}$$

Step 3) If $\nabla f(x_k)^\top (\tilde{x} - x_k) \geq 0$, **Stop**. Otherwise Set $X_{k+1} = X_k \cup \{\tilde{x}_k\}$

End For

Finite convergence of the method is stated in the following Proposition (see, e.g., [31, 35]):

Proposition 3.1. *Simplicial Decomposition algorithm obtains a solution of Problem* (2) *in a finite number of iterations, if* X *is polyhedral.*

As already written in [31], a *vertex dropping rule* is also used to remove those vertices in X_k whose weight is zero in the representation of the master solution x_k. This dropping phase does not modify the formulation of the pricing, so it does not change the steps of the algorithm. However, it can guarantee significant savings in terms of CPU time, since it keeps the dimensions of the master problem small, thus justifying the name of the algorithm (the domain of the master is always a simplex).

The most significant advantage of Simplicial Decomposition is that the dimensions of the master programs are always small: in practice, the maximal dimension of the master problems is often much smaller than the dimension of the original problem. The overall computing time can hence be reduced thanks to this decomposition. Another really important feature of this technique is the fact that all the constraints are in the pricing, so, since the problem is convex, all the master programs are feasible and no dual information is necessary.

It is worth noticing that the minimization of the linearized function in the pricing problem, thanks to the convexity of the objective function, always gives a valid lower bound for the original problem. This specific feature will be exploited when dealing with mixed integer problems.

4. Embedding SD in a Branch and Bound Scheme

In our B&B algorithm, at each node we solve the continuous relaxation with SD; in particular, to solve the master problems we use the *ACDM* algorithm proposed in [1], which is very efficient in this case. Embedding the SD algorithm in this structure is useful for several reasons:

- SD has a good performance in terms of computational time with respect to *Cplex* when solving convex quadratic problems, so it can quickly solve the node relaxation.
- From a theoretical point of view, the extreme points given by a SD algorithm are always feasible, and if they are also binary it is possible to show that every node of the B&B is feasible.
- From an algorithmic point of view, it is possible to use an advanced *warm start* technique called *column projection* to solve more efficiently the SD during the exploration of the B&B.

In the following subsections we will show more in detail all the mentioned improvements.

4.1. SD and valid dual bounds

The SD pricing problem, at each cycle, gives a valid lower bound on the solution: indeed, it solves a linear underestimator of the original objective function. Alternatively, it can be seen as the dual bound given by the Dantzig-Wolfe decomposition method. Such a lower bound can be exploited for pruning the nodes before the end of the SD procedure. This allows to avoid performing the complete column generation algorithm at each node of the B&B tree and therefore to obtain a significant speedup.

4.2. Branching rule and the shape of the B&B tree

The basic branching rule implemented in the B&B branches on variables that are fractional in the continuous relaxation by means of the canonical disjunction. More precisely, at each branching, we fix to 1 the variable with the fractional part closer to one. This choice generally allows us to keep several columns feasible in the left child node. Moreover, since the solutions of our problems are often sparse, this choice also allows us to quickly find a good upper bound and keep the height of the B&B tree limited.

Regardless of how we select the fractional variable to branch on, the following result provides an interesting properties for SD-based B&B:

Theorem 4.1. *Assume that the following two conditions hold at a given node of the B&B tree:*

- *the SD-based B&B algorithm branches on fractional variables of the continuous relaxation;*
- *the SD algorithm only generates integer extreme points in the solution of the continuous relaxation.*

Then both child nodes are feasible.

Proof. Let $X_k = \{\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_k\}$ be the set of extreme points of X generated during the solution of the continuous relaxation associated to a B&B node, let (λ^*) be the obtained optimal solution: the values of the corresponding original variables can be computed as $x^* = \sum_{i=1}^{k} \lambda_i \tilde{x}_i^*$. Let $[x]_j$ be the j-th element of the vector x. Let $\hat{\jmath}$ be the index of the branching variable selected.

The branching rule implies that $0 < [x^*]_{\hat{\jmath}} < 1$. All the extreme points in X_k are integer by definition of SD, this implies that, in order to have $[\tilde{x}^*]_{\hat{\jmath}}$ fractional, we must have at least two extreme points, say $\tilde{x}^*_h$ and $\tilde{x}^*_l$ where $[\tilde{x}^*_h]_{\hat{\jmath}} = 0$ and $[\tilde{x}^*_l]_{\hat{\jmath}} = 1$. This implies that $\tilde{x}^*_h$ and $\tilde{x}^*_l$ are feasible solutions for the partial branching associated to each of the child nodes, respectively, and therefore both branching nodes are feasible. □

4.3. Tree exploration and column projection

We use the *depth first search* in the exploration of the tree. More specifically, after each branching, we selected the left child as the next node to explore. Such a choice allows to keep the number of open nodes very small. Indeed, at any step of the algorithm, at most $n+1$ nodes are opened. Moreover, it allows to quickly find a feasible solution (and hence an upper bound).

Another useful enhancement of the SD algorithm is that all the columns generated at each node can be reused in the children nodes. Indeed, when at a given node a fractional solution is found for the continuous relaxation, and branching is performed for the i^{th} variable, then all the columns with i^{th} component equal to 1 or 0 can be stored either for the left child node or for the right child node. In this way, we can have an initial set of extreme columns for every node of the B&B tree. This enables us to *warmstart* the SD algorithm at every node. We note that this procedure is cheap in terms of memory when we adopt a depth first search strategy, because only two sets of columns are needed to be stored at each node (those with a component fixed to 0 and to 1), and the number of open nodes is small enough. A different strategy, like the breadth first search for instance, where the number of open nodes can be much higher, would be much more memory consuming.

The idea of reusing columns coming from the father node can be pushed further via a specific *column projection*. The basic idea of such a technique is to warmstart, at a given node, the SD method with a set of columns obtained by suitably modifying the columns of the father node.

Let X^t be the feasible region at a node t (X^t is the original feasible set X with some fixed variables) and suppose we branch on variable j. We introduce two projections p^0_j and p^1_j as follow: $p^0_j : X^t \to \{x \in X^t : [x]_j = 0\}$ and $p^1_j : X^t \to \{x \in X^t : [x]_j = 1\}$. For ease of notation we omit the node index t. For each extreme column of the father node $\tilde{x} \in X^t$, $p^0_j(\tilde{x})$ and $p^1_j(\tilde{x})$ will be feasible for one of the child nodes respectively and as close as possible to $\tilde{x}$.

Let $X_k^F = \{\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_k\}$ be the set of extreme points of X generated during the solution of the continuous relaxation associated to the father of a given node of the B&B tree. We propose to use a suitable subset of projected columns $\{p_j^1(\tilde{x}_1), p_j^1(\tilde{x}_2), \ldots, p_j^1(\tilde{x}_k)\}$ as a starting set of feasible columns for the child node obtained after fixing $[x]_j = 1$ (and an analogous procedure when solving the child node obtained after fixing $[x]_j = 0$). The idea behind this procedure is to accelerate the solution of a node by mimicking the solution of its father node. Moreover, we want to add columns only if they provide a descent direction: in this way, indeed, we assure that the new columns will improve the solution, and there is no redundancy in the set. Below we report the pseudocode of the warmstart procedure obtained when fixing $[x]_j$ to 0 or 1 and $1 \leq j \leq n$.

Algorithm 2 Warmstart by projection on $[x]_j = i$ $(i = 0, 1)$

Initialization: Set $\bar{x}_1 = p_j^i(\tilde{x}_1)$, $P = \{\bar{x}_1\}$, $x_{opt} = \bar{x}_1$, and $g_{opt} = \nabla f(x_{opt})$.

For $k = 2, 3, \ldots m$:

$\bar{x}_k = p_j^i(\tilde{x}_k)$. If $g_{opt}^\top(\bar{x}_k - x_{opt}) < 0$ set: $P = P \cup \{\bar{x}_k\}$, $x_{opt} = \arg\min_{x \in conv(P)} f(x)$, $g_{opt} = \nabla f(x_{opt})$.

End For

An essential requirement for such a column projection to be successful is being able to quickly execute $p()$. In Section 5.2, we give a detailed description of the column projection used for each class of instances solved.

5. Computational Results

In this section, we first present the instances used in the tests. We consider instances of combinatorial problems and in particular, we focus on the quadratic minimum spanning tree (QMST) problem and the quadratic shortest path (QSP) problem. Then we describe, for each class of instances studied, the related column projection we used. Finally, we report extensive computational results obtained with the SD based B&B algorithm we presented in the previous sections.

5.1. Instance descriptions

5.1.1. Quadratic Minimum Spanning Tree (QMST) Problem

Given an undirected graph G with n nodes and m edges, the QMST problem is to find the subtree of G spanning all the vertices of G with minimum

cost, where the cost is given by the total weight of the edges and the sum of interaction costs over all pairs of edges on the tree. We generate graphs of QMST in two ways. Firstly, we consider squared grid graphs, of three different sizes. Secondly we use random graphs, of three different sizes, with three different densities: $1/3, 2/3$, or 1 (that is complete graphs). For each class of graphs, we generate quadratic costs on the edges as in [1] and the linear terms are randomly chosen in three different intervals. In each case we generate data with three different seeds, thus obtaining 9 instances for every choice of n and m. We thus have 18 QMST instances on grid graphs, with 40 to 84 edges and 25 to 49 nodes, and 54 instances of random graphs, with 10 to 20 nodes and with 30 to 105 edges. (Indeed we exclude the instances with 10 nodes and $d = 1/3$ since they are too small).

5.1.2. Quadratic Shortest Path (QSP) Problem

Given a directed graph, with one source (s) and one destination (t) node, the QSP problem is to find the path between s and t with minimum cost, where the cost is given by the total weight of the arcs and the sum of interaction costs over all pairs of arcs on the path.

We use two types of QSP instances: Quadratic Grid Shortest Path (QGSP) instances, that is graphs represented by a squared grid, and Quadratic Random Shortest Path (QRSP) instances, that is randomly generated graphs (obtained by the generator ch9-1-1 used in the 9th DIMACS implementation challenge [36]). For grid graphs, we generate quadratic costs on the arcs as in [1] and the linear terms are randomly chosen in two different intervals. Thus we obtain 2 instances for every choice of n, and we have 6 different values for n. For randomly generated graphs we generate instances combining three different choices for the number of arcs, two values for the number of nodes, three different choices of linear coefficients and two seeds; we generate quadratic costs on the arcs as described in [1] with two different seeds.

The benchmark consists of 12 QGSP instances (with 180 to 420 variables, and 100 to 225 constraints) and 72 QRSP instances (1000, 3000 or 5000 variables, 100 to 300 constraints).

5.2. Tailored column projections

In this section, we describe the projections devised for each class of instances. We highlight that in both cases of QMST and QSP, the problem is on a undirected or directed graph, respectively, where all columns $\tilde{x}$ are

binary and correspond to a subset of edges (arcs). Hence, projecting a column can be seen as adding and removing edges (arcs). In the rest of this section we indicate with $\tilde{x}$ the column to be projected, with j the variable to branch on, and with g the gradient $\nabla f(x_{opt})$ of the function in the optimal point according to Algorithm 2. We are then interested in defining $p_j^0(\tilde{x})$ and $p_j^1(\tilde{x})$.

5.2.1. Column projection for the QMST problem

In the QMST problem, the columns $\tilde{x}$ represent feasible spanning trees of the graph. Before describing each projection, we start with a proposition.

Proposition 5.1. *Both projections $p_j^0(\tilde{x})$ and $p_j^1(\tilde{x})$ of a column $\tilde{x}$, at each node of the B&B, can be obtained by either keeping the column $\tilde{x}$ as it is, or switching the value of exactly two components of $\tilde{x}$.*

Proof. In both projecting $\tilde{x}$ to $[x]_j = 0$ or $[x]_j = 1$, either $\tilde{x}$ is already feasible, and it is kept unchanged, or the edge j must be respectively removed or added. If it is removed, then $\tilde{x}$ has now two connected components, each of which is a tree. We shall add another edge to obtain a feasible spanning tree. From Theorem 4.1, we know that the branching is feasible, so there must be at least one edge connecting the two connected components, different from i, which is not fixed to 0, and can be added to get a feasible projection. Similarly, if the edge j is added, a cycle is obtained and another edge of the cycle must be removed. Again, from Theorem 4.1, we know that not all other edges of the cycle are fixed to 1, thus at least one of them can be removed, obtaining a feasible projected spanning tree. □

The projections $p_j^0(\tilde{x})$ and $p_j^1(\tilde{x})$ for the QMST problem are defined as follows.

If $[\tilde{x}]_j = 0$, then $p_j^0(\tilde{x}) = \tilde{x}$; instead, $p_j^1(\tilde{x})$ is obtained by the following steps: firstly we add the edge j to $\tilde{x}$, then we detect the cycle C that is generated and remove from it the edge h of maximal component in g among the edges of C, different from j, which are not fixed to 1. Indeed, such an edge exists for Proposition 5.1 and this clearly provides the best candidate projection to satisfy the condition $g_{opt}^\top(\bar{x}_k - x_{opt}) < 0$ in Algorithm 2.

If on the other hand $[\tilde{x}]_j = 1$, then similarly $p_j^1(\tilde{x}) = \tilde{x}$; $p_j^0(\tilde{x})$ is calculated by removing from $\tilde{x}$ the edge j and finding the two connected sub-trees of $\tilde{x}$; then adding the edge h that connects the two sub-trees, which has minimal component in g, among the edges which are not fixed to 0. Such an edge exists and is the best candidate for the same reasons as in the previous case.

5.2.2. Column projection for the QSP problem

In the QSP problem, the columns $\tilde{x}$ represent directed paths form the source to the termination nodes. Although a projection can be defined in any graph, for the sake of clarity we restrict ourselves to the class of grid graphs (QGSP); moreover, these graphs are more challenging than the random graphs, as will be clear from the computational results.

We consider grid graphs in which the source and the destination are respectively as the top-left and the bottom-right node. In order to define projections $p_j^0(\tilde{x})$ and $p_j^1(\tilde{x})$ for the QGSP problem we need a little more notation. We indicate with o and d respectively the origin and destination nodes of the arc j. Moreover, since we consider grids, every node can be described by two coordinates u and v in the grid, where the origin is the source node. Let u_o and v_o be the coordinates of o, and u_d and v_d be the coordinates of d.

If $\tilde{x}_j = 0$ then $p_j^0(\tilde{x}) = \tilde{x}$; instead, the path $p_j^1(\tilde{x})$ is obtained by making the slightest change to the original path $\tilde{x}$ as follows. From o we add arcs "backward" (if they are not fixed to 0) keeping one of the coordinates u_o and v_o fixed (and decreasing the other one) until we get a node n_0 traversed by $\tilde{x}$, or the origin. Similarly, from d we add arcs forward (if they are not fixed to 0) fixing one of the coordinates u_d and v_d (and increasing the other one) until we get a node n_1 of $\tilde{x}$ or the destination. Now, the arcs of $p_j^1(\tilde{x})$ coincide with those of $\tilde{x}$ from s to n_0 and from n_1 to t and are given by the added arcs between n_0 and n_1. This is done if none of the arcs of $\tilde{x}$ from n_0 to n_1 are fixed to 1.

If on the other hand $\tilde{x}_j = 1$, then similarly $p_j^1(\tilde{x}) = \tilde{x}$; in order to obtain $p_j^0(\tilde{x})$, we consider the arcs h and ℓ obtained by shifting j by one position in parallel in the grid (we notice that one of them may not exist, if j is on the boundary of the grid); then we compute $y = p_h^1(\tilde{x})$ and $z = p_\ell^1(\tilde{x})$ and we define $p_j^0(\tilde{x})$ as one between y and z, if they exist; if both exist, $p_j^0(\tilde{x})$ will be the one between y and z for which the scalar product with g is minimal.

5.3. Numerical results

In this section, we collect the computational results obtained for the considered problems. In particular, we propose two versions of our framework: in the first one, called *SDBB-basic*, we embed SD in the B&B with no warm-start; then we present an advanced version, *SDBB-advanced*, in which we also introduce the projection of the columns from the father to the child nodes. In order to get the most efficient setting, we do not project all columns at

every node, but only those with larger weight in the final simplex of the father node: indeed, as we will see, this is enough to get a good improvement, without overloading the master problems with too many columns.

5.3.1. QMST problems

In this section, we present the results of our tests on the QMST instances.

In Table 1, we can see, for all instances with the same number of nodes (n) and edges (m), the average CPU time in seconds, together with the average number of nodes and the average number of SD iterations generated in our framework, for the basic and advanced settings. In the last line, we report the overall average. We notice that 5 out of the 9 instances with 105 edges are not solved within the time limit of 3 hours, and are not considered in the table.

n	m	SDBB-basic			SDBB-advanced		
		T(s)	#nodes	#its	T(s)	#nodes	#its
10	30	0.1	776	6526	0.1	777	3932
10	45	2.3	9143	102233	2.0	9139	66368
15	35	0.4	2183	19411	0.3	2190	11119
15	70	285.9	603770	9751849	241.9	603836	6481872
15	105	2001.6	3087969	55444482	1719.3	3087306	36940699
20	64	73.9	149602	2365146	60.1	149529	1448596
25	40	0.3	1474	12959	0.2	1476	6383
36	60	60.0	180790	1879460	37.2	180177	827665
49	84	5353.3	9175201	126114913	3175.3	9144096	58401499
Average		653.0	1125163	16418334	431.3	1122283	8586314

Table 1: Results for QMST instances.

These results show that the CPU time can vary a lot depending on the instance type, but the projections always guarantee significant performance improvements. We can also notice a reduction in the number of generated columns, when switching from the basic to the advanced version, while the number of nodes stays the same, as expected.

5.3.2. QSP problems

For the QSP problems, which have a compact formulation, we can compare our performance with the commercial solver Cplex. The results are

performed with the version 12.8, but tests on newer versions provide similar outcome. It is worth noticing that the linearization option of the solver Cplex makes an important role for our instances. By default, indeed, the solver firstly linearizes the problem before solving it, but without this linearization, the solution is always much faster. We hence consider only this second option for Cplex.

We notice that the projection is implemented on QGSP problems only. In Table 2, we present in detail the average results for this class of problems (we use the same notation as the one in the previous table); here we also report the results of CPU time and number of nodes of Cplex for comparison.

n	m	Cplex		SDBB-basic			SDBB-advanced		
		T(s)	#nodes	T(s)	#nodes	#its	T(s)	#nodes	#its
100	180	5.7	5295	7.4	7596	43674	4.1	7644	21393
121	220	20.7	13039	27.8	23047	144249	15.4	23020	69217
144	264	62.4	29081	76.0	45546	346718	43.4	45533	177583
169	312	349.1	117044	364.8	188983	1466319	208.9	189083	737974
196	364	1152.1	281713	1177.2	494343	4094175	687.9	493437	2076179
225	420	2829.7	483860	2823.6	847296	8776642	1667.3	847058	4587965
Averages		736.6	155005	746.1	267802	2478629	437.8	267629	1278385

Table 2: Results for QGSP instances.

We observe that all algorithms solve the given instances to optimality within the imposed timelimit of three hours. We see that the basic setting give slightly worse CPU time performances than Cplex, while the advanced setting outperforms both. As for the QMST problems, this is mainly due to the generated columns that seem to boost the solver. We notice that the number of nodes is similar in the two version of our algorithm, and is always larger than the number of nodes for Cplex.

Devising an efficient projection for general random graphs is not straightforward; nevertheless, even a simplified projection can be improving in the end. For the QRSP problems, we introduce the following strategy in the SDBB-advanced: we just keep the columns (from the father node) that are feasible in the child node.

We finally present the average results on all instances of QRSP problems in Table 3, together with the average results for the QGSP problem. As before, we show the CPU time (in seconds) and number of nodes to solve

the instances for *Cplex*; then, for our algorithm, we report the CPU time in seconds, the number of B&B nodes, and the number of columns generated in the two cases without (*BBSD-basic*) and with projection (*BBSD-advanced*).

type	Cplex		SDBB-basic			SDBB-advanced		
	T(s)	#nodes	T(s)	#nodes	#its	T(s)	#nodes	#its
QGSP	736.6	155005	746.1	267802	2478629	437.8	267629	1278385
QRSP	75.0	47.7	41.7	829.4	9173.7	13.2	829.5	2372.2

Table 3: Quadratic Shortest Path Problem.

These results show that the QRSP problem is easier than the QGSP problem. The solutions are generally much sparser and hence less nodes are needed to get the solution. The proposed algorithm can fully exploit this sparsity and is much faster than *Cplex* in the QRSP instances. Our algorithm generates a larger number of nodes on average than *Cplex* in all QSP instances: since the overall time is shorter, it means that SD is much faster in solving the relaxation at the node. It can be noticed that the column projection is effective: while the number of nodes is almost unchanged, projecting columns allows significant savings in terms of CPU time.

6. Conclusions

We presented a new framework for mixed binary convex quadratic problems that efficiently combines a tailored SD algorithm with a B&B scheme. The method exploits all the good features of the SD algorithm (that is the lower bound and the warmstart given by the structure of the simplices) in an efficient way. We showed, through a numerical experience, that our algorithm performs better than *Cplex* when dealing with Quadratic Shortest Path instances with a dense Hessian matrix. We also provided results for the Quadratic Minimum Spanning Tree problem. In conclusion, we showed how the SD algorithm, originally designed for continuous problems, can be profitably embedded in a framework for mixed binary quadratic problems.

References

[1] E. Bettiol, L. Létocart, F. Rinaldi, E. Traversi, A conjugate direction based simplicial decomposition framework for solving a specific class

of dense convex quadratic programs, Computational Optimization and Applications (2019) 1–40.

[2] N. Krislock, J. Malick, F. Roupin, Biqcrunch: A semidefinite branch-and-bound method for solving binary quadratic problems, ACM Transactions on Mathematical Software (TOMS) 43 (4) (2017) 32.

[3] F. Furini, E. Traversi, Theoretical and computational study of several linearisation techniques for binary quadratic problems, Annals of Operations Research 279 (1) (2019) 387–411. doi:10.1007/s10479-018-3118-2.
URL https://doi.org/10.1007/s10479-018-3118-2

[4] S. Jiang, S.-C. Fang, T. Nie, Q. An, Structured linear reformulation of binary quadratically constrained quadratic programs, Optimization Letters 14 (3) (2020) 611–636. doi:10.1007/s11590-018-1361-8.
URL https://doi.org/10.1007/s11590-018-1361-8

[5] A. Atamturk, , A. Gomez, Submodularity in conic quadratic mixed 0–1 optimization, Operations Research 68 (2) (2020) 609–630.

[6] A. Del Pia, J. Poskin, Characterizations of mixed binary convex quadratic representable sets, Mathematical Programming 177 (1) (2019) 371–394. doi:10.1007/s10107-018-1274-4.
URL https://doi.org/10.1007/s10107-018-1274-4

[7] L. Mencarelli, C. D'Ambrosio, Complex portfolio selection via convex mixed-integer quadratic programming: a survey, International Transactions in Operational Research 26 (2) (2019) 389–414. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/itor.12541, doi:https://doi.org/10.1111/itor.12541.
URL https://onlinelibrary.wiley.com/doi/abs/10.1111/itor.12541

[8] S. Pougkakiotis, J. Gondzio, An interior point-proximal method of multipliers for convex quadratic programming, Computational Optimization and Applications 78 (2) (2021) 307–351.

[9] A. Billionnet, S. Elloumi, M.-C. Plateau, Improving the performance of standard solvers for quadratic 0-1 programs by a tight convex reformu-

lation: The qcr method, Discrete Applied Mathematics 157 (6) (2009) 1185–1197.

[10] A. Billionnet, S. Elloumi, A. Lambert, Extending the qcr method to general mixed-integer programs, Mathematical programming 131 (1-2) (2012) 381–401.

[11] S. Elloumi, A. Lambert, A. Lazare, Solving unconstrained 0-1 polynomial programs through quadratic convex reformulation, Journal of Global Optimization (2021). doi:10.1007/s10898-020-00972-2.
URL https://doi.org/10.1007/s10898-020-00972-2

[12] B. Rostami, F. Errico, A. Lodi, A convex reformulation and an outer approximation for a class of binary quadratic program, Tech. rep., Technical reports, DATA SCIENCE FOR REAL-TIME DECISION-MAKING, Polytechnique Montréal, Canada (2018).

[13] X. Zheng, Y. Pan, X. Cui, Quadratic convex reformulation for nonconvex binary quadratically constrained quadratic programming via surrogate constraint, Journal of Global Optimization 70 (4) (2018) 719–735. doi:10.1007/s10898-017-0591-0.
URL https://doi.org/10.1007/s10898-017-0591-0

[14] G. Dantzig, P. Wolfe, Decomposition principle for linear programs, Operations Research 8 (1960) 101–111.

[15] G. Desaulniers, J. Desrosiers, M. Solomon, Column generation, Vol. 5, Springer Science & Business Media, 2006.

[16] F. Vanderbeck, M. Savelsbergh, A generic view of Dantzig-Wolfe decomposition in mixed integer programming, Oper. Res. Lett. 34 (3) (2006) 296–306.

[17] M. Bergner, A. Caprara, A. Ceselli, F. Furini, M. Lübbecke, E. Malaguti, E. Traversi, Automatic dantzig-wolfe reformulation of mixed integer programs, Math. Program. 149 (1-2) (2015) 391–424.

[18] G. Gamrath, M. Lübbecke, Experiments with a generic Dantzig-Wolfe decomposition for integer programs, in: P. Festa (Ed.), Experimental Algorithms, Springer Berlin Heidelberg, 2010, pp. 239–252.

[19] M. Aganagic, S. Mokhtari, Security constrained economic dispatch using nonlinear Dantzig-Wolfe decomposition, IEEE Transactions on Power Systems 12 (1) (1997) 105–112.

[20] K. Holmberg, Minlp: Generalized cross decomposition., in: Encyclopedia of Optimization, Springer, 2009, pp. 2148–2155.

[21] S. Lawphongpanich, Simplicial with truncated Dantzig-Wolfe decomposition for nonlinear multicommodity network flow problems with side constraints, Operations Research Letters 26 (1) (2000) 33–41.

[22] A. Assad, W. Xu, The quadratic minimum spanning tree problem, Naval Research Logistics (NRL) 39 (3) (1992) 399–417.

[23] R. Cordone, G. Passeri, Solving the quadratic minimum spanning tree problem, Applied Mathematics and Computation 218 (23) (2012) 11597–11612.

[24] D. L. Pereira, M. Gendreau, A. S. da Cunha, Lower bounds and exact algorithms for the quadratic minimum spanning tree problem, Computers & Operations Research 63 (2015) 149–160.

[25] D. A. Guimarães, A. S. da Cunha, D. L. Pereira, Semidefinite programming lower bounds and branch-and-bound algorithms for the quadratic minimum spanning tree problem, European Journal of Operational Research 280 (1) (2020) 46–58.

[26] B. Rostami, F. Malucelli, Lower bounds for the quadratic minimum spanning tree problem based on reduced cost computation, Computers & Operations Research 64 (2015) 178–188.

[27] D. L. Pereira, A. S. da Cunha, Dynamic intersection of multiple implicit dantzig–wolfe decompositions applied to the adjacent only quadratic minimum spanning tree problem, European Journal of Operational Research 284 (2) (2020) 413–426.

[28] B. Rostami, F. Malucelli, D. Frey, C. Buchheim, On the quadratic shortest path problem, in: International Symposium on Experimental Algorithms, Springer, 2015, pp. 379–390.

[29] B. Rostami, A. Chassein, M. Hopf, D. Frey, C. Buchheim, F. Malucelli, M. Goerigk, The quadratic shortest path problem: complexity, approximability, and solution methods, European Journal of Operational Research 268 (2) (2018) 473–485.

[30] C. A. Holloway, An extension of the frank and wolfe method of feasible directions, Mathematical Programming 6 (1) (1974) 14–27.

[31] B. Von Hohenbalken, Simplicial decomposition in nonlinear programming algorithms, Mathematical Programming 13 (1) (1977) 49–68.

[32] D. W. Hearn, S. Lawphongpanich, J. A. Ventura, Restricted simplicial decomposition: Computation and extensions, Computation Mathematical Programming (1987) 99–118.

[33] J. A. Ventura, D. W. Hearn, Restricted simplicial decomposition for convex constrained problems, Mathematical Programming 59 (1) (1993) 71–85.

[34] M. Patriksson, The traffic assignment problem: models and methods, Courier Dover Publications, 2015.

[35] D. P. Bertsekas, A. Scientific, Convex optimization algorithms, Athena Scientific Belmont, 2015.

[36] C. Demetrescu, A. V. Goldberg, D. S. Johnson, The Shortest Path Problem: Ninth DIMACS Implementation Challenge, Vol. 74, American Mathematical Soc., 2009.