Branch-and-Cut and Branch-and-Cut-and-Price Algorithms for the Adjacent Only Quadratic Minimum Spanning Tree Problem

Dilson Lucas Pereira
CAPES Foundation, Ministry of Education of Brazil, Brasília—DF 70.040-020, Brazil

Michel Gendreau
École Polytechnique de Montréal, Montréal—Quebec, Canada

Alexandre Salles da Cunha
Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, Belo Horizonte—MG, Brazil

The quadratic minimum spanning tree problem (QMSTP) consists of finding a spanning tree of a graph *G* such that a quadratic cost function is minimized. In its adjacent only version (AQMSTP), interaction costs only apply for edges that share an endpoint. Motivated by the weak lower bounds provided by formulations in the literature, we present a new linear integer programming formulation for AQMSTP. In addition to decision variables assigned to the edges, it also makes use of variables assigned to the stars of *G*. In doing so, the model is naturally linear (integer), without the need of implementing usual linearization steps, and its linear programming relaxation better estimates the interaction costs between edges. We also study a reformulation derived from the new model, obtained by projecting out the decision variables associated with the stars. Two exact solution approaches are presented: a branch-and-cut-and-price algorithm, based on the first formulation, and a branch-and-cut algorithm, based on its projection. Our computational results indicate that the lower bounds introduced here are much stronger than previous bounds in the literature. Being designed for the adjacent only case, our duality gaps are one order of magnitude smaller than the Gilmore–Lawler lower bounds for AQMSTP. As a result, the two exact algorithms introduced here outperform the previous exact solution approaches in the literature. In particular, the branch-and-cut method we propose managed to solve AQMSTP instances with as many as 50 vertices to proven optimality. **© 2015 Wiley Periodicals, Inc. NETWORKS, Vol. 65(4), 367–379 2015**

Received September 2013; revised August 2014; accepted August 2014
Correspondence to: D. L. Pereira; e-mail: dilson@dcc.ufmg.br
Contract grant sponsor: CAPES (D. L. P.); Contract grant number: BEX 2418/11-8
Contract grant sponsor: CNPq (A. S. da C.); Contract grant numbers: 471464/2013-9, 305423/2012-6, and 477863/2010-8
Contract grant sponsor: FAPEMIG (A. S. da C.); Contract grant number: PRONEX APQ-01201-09 and PPM-VII-00164-13
DOI 10.1002/net.21580
Published online 2 February 2015 in Wiley Online Library (wileyonlinelibrary.com).

Keywords: quadratic 0–1 programming; spanning trees; branch-and-cut; branch-and-cut-and-price; row generation; column generation; cutting planes

1. INTRODUCTION

Given a connected and undirected graph $G=(V,E)$, with $n=|V|$ vertices and $m=|E|$ edges, and a matrix $Q = (q_{ef} \geq 0)_{e,f \in E}$, the quadratic minimum spanning tree problem (QMSTP) asks for a spanning tree of G whose incidence vector $\mathbf{x} \in \mathbb{B}^m$ minimizes the quadratic expression

$$\sum_{e,f \in E} q_{ef} x_e x_f .$$

According to the QMSTP cost function, whenever edges e and f are simultaneously selected, a cost q_{ef} is incurred. For this reason, we refer to q_{ef} as the interaction cost between e and f. As $x_e x_e = 1$ if and only if $x_e = 1$, q_{ee} can be seen as an interaction cost between e and itself, as a direct cost for edge e, or as a function of both.

In this work, we focus on a particular QMSTP variant, known as the adjacent only QMSTP (AQMSTP). In the AQMSTP case, interaction costs are zero for pairs of edges that do not share an endpoint. Both QMSTP and AQMSTP are NP-Hard [2]. Applications of both are found in the design of telecommunication, transportation, and hydraulic networks, as described in [2].

QMSTP and AQMSTP were first studied by Assad and Xu [2], where an exact algorithm and heuristics were proposed. The exact solution approach in that reference is a branch-and-bound (BB) algorithm based on a dual ascent lower bounding procedure. Three heuristics were proposed: two of them are constructive heuristics, while the third is a Lagrangian heuristic that results from their lower bounding scheme. Instances defined over complete graphs with

12 and 15 vertices were solved for QMSTP and AQMSTP, respectively.

Solution approaches for QMSTP span various types of algorithms. Exact solution methods were investigated by Öncan and Punnen [12], by Cordone and Passeri [5], and by ourselves [15, 16]. QMSTP heuristics based on a variety of approaches, such as tabu search and genetic algorithms, can also be found in [14, 19, 21].

Öncan and Punnen [12] introduced a lower bounding procedure based on Lagrangian relaxation and a local search based heuristic. However, the algorithms in [12] were not embedded into a BB exact framework.

Cordone and Passeri [5] also proposed local search based heuristics, along with some improvements for the QMSTP exact algorithm in [2]. The enhanced BB algorithm in [5] managed to solve instances defined over complete graphs with up to 15 vertices. Larger sparse instances, with up to 20 vertices, could also be solved by that algorithm.

A hierarchy of strong Lagrangian relaxation bounds, together with BB algorithms based on them, were presented in Pereira et al. [16] and in the PhD thesis by Pereira [15]. The algorithms in [15, 16] significantly improved on the size of QMSTP instances that could be solved to proven optimality. More precisely, the best algorithm in [15, 16] managed to solve dense instances with as many as 50 vertices.

A polyhedral investigation for a related polytope, the boolean quadric forest polytope, was conducted by Lee and Leung [8]. Several facet defining inequalities for that polytope were presented in that study.

Recently, a biobjective version of AQMSTP was investigated by Maia et al. [10]. The authors considered $\sum_{e \in E} q_{ee} x_e$ and $\sum_{e,f \in E, e \neq f} q_{ef} x_e x_f$ as two separate objective functions. To generate Pareto optimal solutions, they devised a local search algorithm and an adaptation of the BB algorithm in [2]. The BB algorithm in [10] was able to generate the set of Pareto optimal solutions for instances with up to 20 vertices.

A preliminary version of the present study [17] was presented during the 2013 International Network Optimization Conference. In that work, an AQMSTP linear integer programming formulation was proposed. The motivation for that formulation is the fact that, when applied to AQMSTP, previous QMSTP formulations in the literature provide very poor lower bounds. The formulation we presented in [17] makes use of an extended variable space: in addition to variables assigned to the edges of G, it also uses decision variables assigned to the stars of G. Different from all previous studies for QMSTP and AQMSTP, the lower bounding procedure in [17] is not based on Lagrangian relaxation. To evaluate the linear relaxation lower bounds from the formulation presented in that work, a row and column generation (RCG) algorithm was proposed.

In this article, we improve on our previous results in [17] in many ways. An enhanced version of the RCG lower bounding scheme devised in that work is now the basis of a branch-and-cut-and-price algorithm for AQMSTP. We also propose a second formulation for the problem, obtained from the first, by projecting out the variables assigned to the stars of G. A branch-and-cut algorithm based on the projected reformulation is also introduced here. For the first time, dense AQMSTP instances with as many as 50 vertices were solved to optimality.

As we pointed out before, all the existing exact solution approaches in the QMSTP literature are based on Lagrangian relaxation. To solve the Lagrangian subproblems, they rely on a procedure due to Gilmore [6] and Lawler [7], originally proposed for the quadratic assignment problem [3] and later adapted for many other quadratic 0–1 problems, such as the quadratic 0–1 knapsack problem [18]. This procedure, even when combined with multipliers adjustment, provides very poor bounds for AQMSTP. Reasons for that are discussed in section 2. The new AQMSTP formulation is presented in section 3. A branch-and-cut-and-price algorithm based on that model is discussed next, in section 4. In section 5, we introduce an AQMSTP branch-and-cut algorithm based on a projection of the reformulation. Computational results are presented in section 6. We conclude the article in section 7.

Throughout the article, we use the following notation. Given $U \subseteq V, E(U) = \{\{u,v\} \in E : u,v \in U\}$ denotes the set of edges with both endpoints in U and $\delta(U) = \{\{u,v\} \in E : u \in U, v \in V \backslash U\}$ denotes the set of edges with only one endpoint in U. For simplicity, if U has only one vertex, say v, we use $\delta(v)$ instead of $\delta(\{v\})$. Given an integer programming formulation F for AQMSTP, we denote by $L(F)$ the value of its linear programming relaxation bound.

2. QMSTP FORMULATIONS FROM THE LITERATURE

To formulate QMSTP as a linear integer program, consider binary decision variables $\mathbf{x} = (x_e)_{e \in E}$, which indicate whether $(x_e = 1)$ or not $(x_e = 0)$ an edge $e \in E$ appears in the tree we are looking for. Consider also linearization variables $\mathbf{y} = (y_{ef})_{e,f \in E}$, used to replace the products $x_e x_f$, $e, f \in E$. Denote by $\mathbf{y}_e = (y_{ef})_{f \in E}$ the row of $\mathbf{y}$ indexed by $e \in E$.

The following QMSTP formulation was introduced by Assad and Xu [2]:

$$\min \sum_{e,f \in E} q_{ef} y_{ef} \tag{1}$$

$$\text{s.t.} \quad \mathbf{x} \in X, \tag{2}$$

$$\mathbf{y}_e \in X(x_e), \quad e \in E, \tag{3}$$

$$\sum_{f \in E} y_{fe} = (n-1)x_e, \quad e \in E, \tag{4}$$

$$(\mathbf{x}, \mathbf{y}) \in \mathbb{B}^{m+m^2}, \tag{5}$$

where X denotes the following representation for the spanning tree polytope:

$$\sum_{f \in E} x_f = n - 1, \tag{6}$$

$$\sum_{f \in E(U)} x_f \leq |U| - 1, \quad U \subset V, |U| \geq 2, \tag{7}$$

$$x_f \geq 0, \quad f \in E, \tag{8}$$

and $X(x_e)$, $e \in E$, denotes the set of constraints obtained by multiplying (6)–(8) and $\{x_e = 1\}$ by x_e, and then substituting for variables $\mathbf{y}$:

$$\sum_{f \in E} y_{ef} = (n-1)x_e, \tag{9}$$

$$\sum_{f \in E(U)} y_{ef} \leq (|U|-1)x_e, \quad U \subset V, |U| \geq 2, \tag{10}$$

$$y_{ee} = x_e, \tag{11}$$

$$y_{ef} \geq 0, \quad f \in E. \tag{12}$$

Constraints (2) and (5) state that $\mathbf{x}$ must be the incidence vector of a spanning tree of G. Constraints (3), together with (5), state that, if $x_e = 1$, then $\mathbf{y}_e$ must be the incidence vector of a spanning tree containing $e \in E$. Conversely, if $x_e = 0$, $\mathbf{y}_e$ must be the zero vector. Finally, constraints (4) force the edges in the tree defined by $\mathbf{x}$ to also appear in the trees defined by $\mathbf{y}_e$, for all $e \in E$.

For reasons that will become clear shortly, Lagrangian relaxation became the technique of choice for solving QMSTP [2, 12, 16]. Although these studies make use of different constraints to reinforce the basic QMSTP model (1)–(5), they all use a common procedure for solving Lagrangian subproblems, the Gilmore–Lawler procedure, explained next. Consider a relaxation for the QMSTP model (1)–(5), where constraints (4) are dropped from the formulation. To obtain an optimal solution $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$ to such a relaxation, first solve

$$\overline{q}_e = \min \left\{ \sum_{f \in E} q_{ef} y_{ef} : \mathbf{y}_e \in X \cap \mathbb{B}^m, y_{ee} = 1 \right\}, \tag{13}$$

for each edge $e \in E$, and let $\tilde{\mathbf{y}}_e$ be a solution vector. Then, solve

$$\overline{q}_0 = \min \left\{ \sum_{e \in E} \overline{q}_e x_e : \mathbf{x} \in X \right\},$$

and let $\tilde{\mathbf{x}}$ be an optimal solution. Finally, set $\overline{\mathbf{x}} = \tilde{\mathbf{x}}$ and $\overline{\mathbf{y}}_e = \tilde{x}_e \tilde{\mathbf{y}}_e$, for all $e \in E$. This algorithm is the well known Gilmore–Lawler procedure [9], whose application to QMSTP provides the Gilmore–Lawler bound $\overline{q}_0$.

Assad and Xu [2] devised an algorithm that follows similar guidelines, by relaxing and attaching Lagrangian multipliers to (4). Another contribution in that direction was introduced by Öncan and Punnen [12]. They reinforced the QMSTP model (1)–(5) with valid inequalities

$$\sum_{e \in \delta(v)} y_{ef} \geq x_f, \quad f \in E, v \in V, \tag{14}$$

and implemented a Lagrangian relaxation algorithm, obtained after relaxing (4) and (14). Multipliers adjustment was only implemented for (14). In a recent study for QMSTP [15, 16], we made use of the following stronger constraints, in replacement of (4):

$$y_{ef} = y_{fe}, \quad e < f \in E. \tag{15}$$

A parallel BB algorithm based on a Lagrangian relaxation procedure where constraints (15) are relaxed and dualized in Lagrangian fashion was also introduced in that QMSTP study.

It should be clear that any of the QMSTP lower bounding procedures outlined previously could be applied to AQMSTP. However, Gilmore–Lawler based bounds tend to be much weaker when applied to AQMSTP. We try to give an argument to support this claim in the discussion below.

Consider an AQMSTP problem instance. Assume that G is complete and $Q \geq \mathbf{0}$. Let T be a spanning tree induced by a Hamiltonian path of G. Then, the edges at the extremities of the path interact with exactly one other edge each, while each one of the remaining $(n-3)$ edges interact with exactly two other edges. Thus, a total of $2(n-3)+2 = 2(n-2)$ interactions happen in T. This is the minimum number of interactions that is possible in any spanning tree of G. Consider now the application of the Gilmore–Lawler procedure to this AQMSTP instance. When solving (13) for an edge $e = \{u, v\} \in E$, the solution spanning tree will use only one of the edges adjacent to e, namely, edge

$$f \in \arg\min \left\{ q_{eg} : g \in \delta(u) \cup \delta(v) \right\}.$$

Thus, the Gilmore–Lawler bound will take into account only one interaction for each edge, that is, $n-1$ interactions, which is about half the minimum number of interactions that can happen. Such an underestimation on the number of edge interactions in an AQMSTP solution leads to very weak lower bounds. This motivates the need for an AQMSTP formulation that takes its particular cost structure into account.

3. A FORMULATION FOR AQMSTP BASED ON THE STARS OF *G*

Since interaction costs are incurred only by pairs of edges that share an endpoint, we can consider the stars of G as modeling entities to obtain an integer programming formulation for AQMSTP on an extended variable space. To that aim, let $S^v = \{H \subseteq \delta(v)\}$ be the set of all stars of G centered at $v \in V$ and let $S = \cup_{v \in V} S^v$ be the union of all such sets. Note that according to our definition, a star centered at v is any subset of edges incident to v, including the empty set. Consider a vector of binary decision variables $\mathbf{t} = (t_H)_{H \in S}$, used to indicate whether $(t_H = 1)$ or not $(t_H = 0)$ a star $H \in S$ is included in the AQMSTP solution. Consider also variables $\mathbf{x}$, defined as before. For all $H \in S$, define the cost

$$q_H = \sum_{e,f \in H: e \neq f} q_{ef} + \frac{1}{2} \sum_{e \in H} q_{ee}.$$

A binary integer programming reformulation for AQM-STP is given by

$$(F^*) \quad \min \sum_{H \in S} q_H t_H \tag{16}$$

$$\text{s.t.} \quad \sum_{H \in S^v} t_H = 1, \qquad v \in V, \tag{17}$$

$$\sum_{H \in S^v : e \in H} t_H = x_e, \quad e \in \delta(v), v \in V, \tag{18}$$

$$\mathbf{x} \in X, \tag{19}$$

$$(\mathbf{x}, \mathbf{t}) \in \mathbb{B}^{m+|S|}. \tag{20}$$

Constraints (17) state that, for each $v \in V$, exactly one star centered at v must be selected. Constraints (18) couple variables $\mathbf{x}$ and $\mathbf{t}$, that is, they enforce that x_e assumes value 1 if and only if edge e appears in both stars centered at its endpoints. Constraints (19) impose that the selected edges must imply a spanning tree of G.

Based on the result provided next, we assume throughout the article that (20) is replaced by $\mathbf{x} \in \mathbb{B}^m$ and $\mathbf{t} \in \mathbb{R}_+^{|S|}$.

Proposition 1. *The integrality of* $\mathbf{x}$ *implies the integrality of* $\mathbf{t}$ *in* F^*.

Proof. Consider a solution $(\bar{\mathbf{x}}, \bar{\mathbf{t}})$ for the linear relaxation of F^* in which $\bar{\mathbf{x}}$ is integer. If $\bar{\mathbf{t}}$ is fractional, then there is a vertex v such that $\bar{t}_H$ is fractional for some of the stars H in S^v. This implies that there is an edge e that appears in some, but not all, of the stars with $\bar{t}_H > 0$. As $\bar{x}_e = 1$, $\sum_{H \in S^v : e \in H} \bar{t}_H = 1$, but $\sum_{H \in S^v : e \notin H} \bar{t}_H > 0$, which implies that (17) is violated. Thus, integer $\mathbf{x}$ implies integer $\mathbf{t}$. ∎

4. A BRANCH-AND-CUT-AND-PRICE ALGORITHM FOR AQMSTP

Since F^* uses exponentially many variables $t_H, H \in S$, and subtour elimination constraints (SECs) (7), the direct resolution of its linear programming relaxation is not a practical way to evaluate $L(F^*)$. To get around that, we develop a RCG algorithm, described next. That algorithm is then used as the basis for an AQMSTP branch-and-cut-and-price algorithm, described in section 4.3.

4.1. Row and Column Generation Algorithm for Solving the Linear Programming Relaxation of F^*

Let $\overline{R} \subseteq \{U \subset V : |U| \geq 2\}$ denote a (tiny) subset of the sets for which the SECs are formulated. Likewise, for each $v \in V$, let $\overline{S}^v \subseteq S^v$ denote a (tiny) subset of the stars of G centered at v. For the sake of solving the linear programming relaxation of F^*, our algorithm initially considers only the stars in $\{\overline{S}^v : v \in V\}$ and the SECs formulated by the sets in $\overline{R}$. Let $\overline{F}^*$ denote the resulting linear program. If the optimal solution for $\overline{F}^*$ satisfies all SECs and if all variables $\mathbf{t}$ have non-negative reduced cost, that solution is also an optimal solution for the linear programming relaxation of F^*. Otherwise, we either update $\overline{R}$, by adding to it vertex sets of violated SECs, or update $\{\overline{S}^v : v \in V\}$, with stars whose associated decision variables have negative reduced cost. We then reoptimize the new linear program $\overline{F}^*$. The process is repeated until an optimal solution for the linear programming relaxation of F^* is obtained. The detailed RCG algorithm we implement, denoted by RCG, is presented below.

RCG: Row and Column Generation Algorithm

1. Formulate problem $\overline{F}^*$ in terms of the current sets $\overline{R}$ and $\{\overline{S}^v : v \in V\}$. Let $(\bar{\mathbf{x}}, \bar{\mathbf{t}})$ be an optimal primal solution. Let $\bar{\boldsymbol{\beta}} = (\bar{\beta}^v)_{v \in V}$, and $\bar{\boldsymbol{\gamma}} = (\bar{\gamma}_e^v)_{e \in \delta(v)}, v \in V$, be optimal dual variables respectively assigned to constraints (17) and (18). Denote the row of $\bar{\boldsymbol{\gamma}}$ indexed by $v \in V$ by $\bar{\boldsymbol{\gamma}}^v = (\bar{\gamma}_e^v)_{e \in \delta(v)}$.
2. Solve the separation problem of the SECs
$$\max_{U \subset V : |U| \geq 2} \left\{ \sum_{f \in E(U)} \bar{x}_f - |U| + 1 \right\}. \tag{21}$$
Details on how this problem is solved will be given later. Let $\overline{V}$ be a vertex set attaining the maximum. If the corresponding SEC is violated, that is, the maximum is positive, update $\overline{R} \leftarrow \overline{R} \cup \{\overline{V}\}$ and go back to step 1. Otherwise, proceed to step 3.
3. For each star $H \in S^v, v \in V$, let $\bar{q}_H$ be the reduced cost of t_H, that is,
$$\bar{q}_H = q_H - \bar{\beta}^v - \sum_{e \in H} \bar{\gamma}_e^v.$$
Let $\tilde{S}^v = \{H \in S^v : |H| \leq K_{\text{enum}}, \bar{q}_H < 0\}$ be the set of the stars centered at v with negative reduced cost and at most K_{enum} edges. The value K_{enum} is an implementation parameter, small enough so that the stars in $\tilde{S}^v$ can be enumerated. If $\tilde{S}^v \neq \emptyset$, update $\overline{S}^v \leftarrow \overline{S}^v \cup \tilde{S}^v$ and go back to step 1. Otherwise, proceed to step 4.
4. For each $v \in V$, solve the pricing problem
$$(\text{PP}(v)) \quad \min \left\{ \bar{q}_H : H \in S^v \right\}.$$
Details on how this problem is solved will be given later. Let $\overline{H}$ be a star attaining the minimum and let $r^v = \bar{q}_{\overline{H}}$ be its reduced cost. If $r^v < 0$, update $\overline{S}^v \leftarrow \overline{S}^v \cup \{\overline{H}\}$ and go back to step 1. Otherwise, $(\bar{\mathbf{x}}, \bar{\mathbf{t}})$ is an optimal solution for the linear programming relaxation of F^*.

Before moving on, let us point out that, to obtain a valid lower bound for AQMSTP, we do not need to wait until no negative reduced cost stars are found, in step 4. By adding r^v to each $\bar{\beta}^v$, we obtain a set of dual multipliers for which no star of negative reduced costs exists, that is, we obtain a feasible solution for the dual of the linear programming relaxation of F^*, whose corresponding objective function value is

$$\sum_{v \in V} \left(\sum_{H \in \overline{S}^v} q_H \bar{t}_H + r^v \right). \tag{22}$$

In our implementation of RCG, we initialize the algorithm with $\overline{R} \leftarrow \emptyset$. SECs are then added to $\overline{R}$ only when the separation problem is solved, in step 2. To that aim, we make use of the separation algorithm introduced in Padberg and Wolsey [13], which involves solving $O(n)$ max-flow problems, in conveniently defined networks, with an overall $O(n^4)$ worst case complexity. Besides the most violated SEC, that algorithm might also provide other violated SECs. If that applies, we also add them to $\overline{R}$.

To obtain a linear program $\overline{F}^*$ for which an initial basic feasible solution exists, we initialize the sets $\{\overline{S}^v : v \in V\}$ with the stars in a feasible AQMSTP solution. That solution is obtained by a randomized AQMSTP heuristic, described in section 4. Additional columns are added to the model when the sets $\{\overline{S}^v : v \in V\}$ are updated, in steps 3 and 4. In step 3, the pricing problems are solved heuristically, while, in step 4, they are solved exactly.

Step 4 accounts for most of the CPU time of the algorithm that is why it is the last step to be executed. Step 3 is an attempt to accelerate the method by enumerating and computing the reduced costs of promising stars. More precisely, the algorithm computes the reduced cost for all stars $\{H \in S : |H| \leq K_{\text{enum}}\}$, for a small valued implementation parameter K_{enum}. Such a pricing heuristic is grounded on the fact that the cost q_H of a star $H \in S$ grows very rapidly with the number of edges in the star. Consequently, it is unlikely that stars used at the optimal solution of the linear programming relaxation of F^* (as well as in optimal solutions for AQMSTP) have too many edges. In our implementation, we set $K_{\text{enum}} = 6$. As the pricing problems solved for each $v \in V$ in steps 3 and 4 are independent, they are solved in parallel. We now discuss how the exact pricing problem is solved.

4.2. *Solving the Pricing Problem PP(v)*

Our solution strategy for exactly solving the pricing problem consists of modeling it as a linear integer program, which is then solved by means of a commercial solver. The first step in that direction is to formulate PP(v) as a quadratic binary program. To do that, we consider binary variables $\mathbf{w} = (w_e)_{e \in \delta(v)}$, used to select the edges in a star of G centered at v. More precisely, if an edge $e \in \delta(v)$ is included in the star, $w_e = 1$. Otherwise, $w_e = 0$. We can now formulate PP(v) as the following unconstrained quadratic binary program:

$$\min \left\{ \sum_{e,f \in \delta(v)} \overline{q}_{ef} w_e w_f : \mathbf{w} \in \mathbb{B}^{|\delta(v)|} \right\} - \overline{\beta}^v, \qquad (23)$$

where $\overline{q}_{ee} = q_{ee} - \overline{\gamma}_e^v$ and $\overline{q}_{ef} = q_{ef}$ for $e \neq f \in \delta(v)$.

Although polynomial time algorithms do exist for solving quadratic binary optimization problems in which the diagonal entries of the cost matrix are non-negative and the off-diagonal entries are nonpositive [4], we are not aware of polynomial time algorithms for problems like (23), where the opposite holds. Indeed, the general unconstrained quadratic binary optimization problem is NP-Hard [20].

Therefore, to solve PP(v), we linearize (23) and solve the resulting integer program. To that aim, consider the introduction of linearization variables $\mathbf{z} = (z_{ef})_{e,f \in \delta(v), e<f}$. Let $z_{fe} = z_{ef}$ if $f > e$. A binary integer program for PP(v) is

$$-\overline{\beta}^v + \min \sum_{e,f \in \delta(v): e \neq f} \overline{q}_{ef} z_{ef} + \sum_{e \in \delta(v)} \overline{q}_{ee} w_e \qquad (24)$$

$$\text{s.t.} \quad z_{ef} \leq w_e, \quad e,f \in \delta(v), e < f, \qquad (25)$$

$$z_{ef} \leq w_j, \quad e,f \in \delta(v), e < f, \qquad (26)$$

$$w_e + w_f - 1 \leq z_{ef}, \quad e,f \in \delta(v), e < f, \qquad (27)$$

$$(\mathbf{w}, \mathbf{z}) \in \mathbb{B}^{|\delta(v)| + \frac{1}{2}(|\delta(v)|^2 - |\delta(v)|)}. \qquad (28)$$

To solve (24)–(28), we use a strategy that has two phases. The first is a preprocessing phase. The second is a cut-and-branch phase. In the first phase, we try to fix variables and generate optimality cuts for the problem. We also try to use the fact that the solution space was partially explored by enumeration to avoid entering the second phase, where the integer program (24)–(28) is solved to proven optimality. If the second phase is needed, we carry the optimality cuts and variable fixing information to (24)–(28) and then solve the problem via BB. To implement the second phase we make use of a commercial integer programming solver. The strategies used in the first phase are described next.

4.2.1. Optimality Cuts and Acceleration Strategies

In the following, assume that an upper bound u on the optimal solution value of AQMSTP is known and that no negative reduced cost stars with K_{enum} or fewer edges were found.

Let $K_{\max}$ be the smallest integer k such that $q_H \geq u$ for all $H \in S^v, |H| > k$. Then the optimality cut (29) below states that no star centered at v could have more than $K_{\max}$ edges in an optimal AQMSTP solution:

$$\sum_{e \in \delta(v)} w_e \leq K_{\max}. \qquad (29)$$

This observation is valid since, under non-negative costs, using a star with more than $K_{\max}$ edges would lead to a spanning tree whose cost is at least u. If $K_{\max} \leq K_{\text{enum}}$, we can stop the resolution of the pricing problem for this particular $v \in V$, since all stars with $K_{\max}$ or fewer edges were already dismissed. Note that, different from K_{enum}, $K_{\max}$ is a precomputed value that depends on an upper bound u initially available for the particular instance being solved.

As we assumed $Q \geq \mathbf{0}$, if $\overline{q}_{ee} \geq 0$, we can set $w_e = 0$ without worsening the cost of any solution. For this reason, we assume $\overline{q}_{ee} < 0$ for all $e \in \delta(v)$ in what follows.

Let $(\overline{\mathbf{w}}, \overline{\mathbf{z}})$ be a feasible solution for (24)–(28). Observe that if $\sum_{f \in \delta(v) \setminus \{e\}} \overline{w}_f (\overline{q}_{ef} + \overline{q}_{fe}) \geq |\overline{q}_{ee}|$ for some $e \in \delta(v)$ such that $\overline{w}_e = 1$, a solution not worse than $(\overline{\mathbf{w}}, \overline{\mathbf{z}})$ could be obtained by setting $\overline{w}_e = 0$. The following optimality cut follows directly from this observation:

$$\sum_{f \in \delta(v): f \neq e} (\overline{q}_{ef} + \overline{q}_{fe}) z_{ef} \leq |\overline{q}_{ee}| w_e.$$

The observation above is also helpful in devising the following variable fixing test. Given $e \in \delta(v)$, sort the elements of $\delta(v)\setminus\{e\}$ as $(f_1, ..., f_{|\delta(v)|-1})$ in nondecreasing order of the values $\overline{q}_{ef} + \overline{q}_{fe}$. Let $\overline{k}$ be the smallest k for which $\sum_{l=1}^{k}(\overline{q}_{ef_l} + \overline{q}_{f_l e}) \geq |\overline{q}_{ee}|$. Note that we can set $\overline{w}_e = 0$ in any solution $(\overline{\mathbf{w}}, \overline{\mathbf{z}})$ that uses e along with $\overline{k}$ or more other edges from $\delta(v)$ without worsening the solution. Thus, we are interested in solutions in which e is selected along with at most $\overline{k} - 1$ other edges. If $\overline{k} \leq K_{\text{enum}}$, these solutions were already dismissed and we can set $w_e = 0$. After running the test for all $e \in \delta(v)$, if any variable has been fixed, the process is repeated, considering then only the free variables. When all edges $e \in \delta(v)$ have been tested and no further variables could be fixed, we obtain two optimality cuts:

$$w_e + \sum_{l=1}^{\overline{k}} w_{f_l} \leq \overline{k},$$

and

$$\sum_{f \in \delta(v)\setminus\{e\}} z_{ef} \leq (\overline{k} - 1)w_e.$$

Finally, a lower bound on the reduced cost of a star containing exactly k edges is given by

$$\min_{H \in S^v : |H| = k} \{q_H\} - \overline{\beta}^v - \max_{H \in S^v : |H| = k} \left\{ \sum_{e \in H} \overline{\gamma}_e^v \right\}.$$

Thus, if these bounds are non-negative for all k such that $K_{\text{enum}} < k \leq K_{\max}$, we can stop the exact resolution of the pricing problem for this particular vertex v. Note that the minimization term in the above expression is a precomputed value, while the maximization term can be easily evaluated by simply sorting the edges in $\delta(v)$ in nonincreasing order of the costs $\{\overline{\gamma}_e^v : e \in \delta(v)\}$ and taking the sum over the first k edges.

4.3. *Branch-and-Cut-and-Price Implementation Details*

We combine the lower bounding procedure RCG with BB to develop a branch-and-cut-and-price algorithm, BCP, for AQMSTP. BCP is initialized with a valid upper bound provided by the heuristic described in section 4.4. Such a spanning tree also provides the initial stars $\{\overline{S}^v : v \in V\}$ used to obtain an initial basic feasible solution for RCG. For the remaining branch-and-cut-and-price nodes, the sets $\{\overline{S}^v : v \in V\}$ and $\overline{R}$ used to formulate the very first linear program are those available when the last linear program was solved at the parent node.

To avoid solving too many pricing problems by the exact procedure, RCG is always aborted right after the first round of exact resolution of pricing problems is concluded, no matter which node, the root or its descendants, is being investigated. The lower bound (22) obtained at that moment is taken as a lower bound on the optimum for that node.

If the solution vector $(\overline{\mathbf{x}}, \overline{\mathbf{t}})$ of the linear programming relaxation of a BCP node is fractional, we then enter a variable fixing phase. We first try to fix nonbasic variables $\mathbf{x}$ by means of their reduced costs. After that, a different variable fixing procedure is carried out as follows. We first create a list C of some fractional edges. To that aim, let $g_e = 1/2 - |1/2 - \overline{x}_e|$ and $g^* = \max\{g_e : e \in E\}$. Initialize $C \leftarrow \emptyset$. Then add fractional edges to C according to a probability given by g_e / g^*. After that, for each edge $e \in C$, let h_e^1 (respectively, h_e^0) be the lower bound obtained by RCG when the constraint $x_e = 1$ (respectively, $x_e = 0$) is further imposed to the the current node. If either h_e^0 or h_e^1 is larger than a known valid upper bound for AQMSTP, the corresponding variable is fixed accordingly. After examining all edges in C, if any of them has been fixed, we solve the node again.

When no variable can be fixed by the procedures outlined above and the solution vector remains fractional, we select an edge e for which $\overline{x}_e$ is fractional and create two new subproblems. For one of them we impose $x_e = 1$, and for the other, $x_e = 0$. The edge e on which we branch is selected according to the strong branching technique [1]. Strong branching attempts to reduce the size of the BB tree by actually solving the subproblems that would result if we branched on a variable e, for all e in a subset of the fractional variables. Then, the variable for which the smallest lower bound in the two resulting subproblems is maximum is selected as the branching variable. Note that the information we need to perform strong branching is already made available during our variable fixing tests. Thus, we branch on an edge

$$e \in \arg\max \left\{ \min \left\{ h_f^0, h_f^1 \right\} : f \in C \right\},$$

ties being broken randomly.

To deal with possible infeasibility caused by branching along the enumeration tree, artificial variables with large costs are added to F^*. For example, given a vertex $v \in V$, we can rewrite the constraint (17) defined for v as $\sum_{H \in S^v} t_H + a_1 - a_2 = 1$, where a_1 and a_2 are artificial variables with large cost coefficients in the objective function. This way, the constraint can always be satisfied, but, if either a_1 or a_2 has a positive value in the optimal solution for the node, we know the node is infeasible.

It is important to remark that a large amount of information associated with the columns, rows, and other programming entities used by the linear programming package we use and by our own data structures, must be stored at every BCP node. Attempting to avoid memory issues later on in the enumeration tree, BCP implements a depth-first policy for node selection. Depth-first search generally produces a smaller number of active BB nodes, resulting in reduced computer memory.

4.4. *AQMSTP Heuristic*

The following multistart heuristic is used by our exact solution algorithms to obtain an initial upper bound.

First, we select a spanning tree $T = (V, E_T)$ of G at random. To do that, we assign a weight to each edge of E, selected with uniform probability in $\{1, \ldots, 100\}$. Then, to obtain T, we compute the minimum spanning tree of G under these weights.

Given the current spanning tree T, we select each edge e not in T, one by one. For each one, we identify the best spanning tree T', under the original quadratic cost function, that can be obtained by replacing an edge f in T by e. This is easily accomplished by letting T' be the spanning tree with the smallest quadratic cost that can be obtained by replacing some edge in the cycle of $(V, E_T \cup \{e\})$ by e. If T' is better than T, then T is replaced by T'. The process is repeated until all edges $e \in E \backslash E_T$ have been investigated and no improving spanning tree could be found.

The heuristic above is executed $n_{heu} = 10$ times and the best solution found is provided to BCP as an initial upper bound for AQMSTP. In section 6, we present some computational results supporting our choice for the number of times n_{heu} the multistart heuristic is called.

5. A BRANCH-AND-CUT ALGORITHM FOR AQMSTP

In this section, we use projection to eliminate variables t_H, $H \in S$, from F^*. This yields a reformulation for AQMSTP with the same linear programming relaxation bounds as F^*, on a compact variable space. Thus, solving its linear programming relaxation is an alternative approach to evaluate $L(F^*)$. The projection process generates exponentially many new constraints. Hence, to solve that linear programming relaxation, we implement a cutting plane algorithm that separates two classes of valid inequalities: SECs and projection cuts. The formulation and the cutting plane algorithm are presented in the next two sections. After that, in section 5.3, we discuss a branch-and-cut algorithm based on the cutting plane algorithm. For an exposition on projection and its use in linear and integer programming, we refer the reader to [11].

5.1. *A Projection of F^**

To use projection to eliminate $\mathbf{t}$ from F^*, we first remove those variables from the objective function. To do that, we make use of artificial continuous variables $\mathbf{p} = (p_v)_{v \in V}$. The objective function of F^* is then stated as

$$\min \sum_{v \in V} p_v,$$

and the constraints

$$p_v \geq \sum_{H \in S^v} q_H t_H, \quad v \in V \tag{30}$$

are appended to the formulation.

Assume that dual multipliers $\alpha = (\alpha^v)_{v \in V}$, $\boldsymbol{\beta} = (\beta^v)_{v \in V}$, and $\boldsymbol{\gamma} = (\gamma_e^v)_{e \in \delta(v), v \in V}$, are respectively associated with constraints (30), (17), and (18). Denote by $\boldsymbol{\gamma}^v = (\gamma_e^v)_{e \in \delta(v)}$ the row of $\boldsymbol{\gamma}$ indexed by $v \in V$. Then the projection cone C^v, defined for each $v \in V$, which we use to eliminate the variables t_H, $H \in S^v$, is given by

$$(C^v) \quad -q_H \alpha^v + \beta^v + \sum_{e \in H} \gamma_e^v \leq 0, \quad H \in S^v, \tag{1}$$

$$(\alpha^v, \beta^v, \boldsymbol{\gamma}^v) \in \mathbb{R}_+ \times \mathbb{R} \times \mathbb{R}^{|\delta(v)|}. \tag{31}$$

Using $\{C^v \colon v \in V\}$, we obtain the projection of F^* onto the $(\mathbf{x}, \mathbf{p})$ space:

$$\begin{aligned}
(F^*_{\mathbf{x}}) \quad \min \quad & \sum_{v \in V} p_v \\
\text{s.t.} \quad & \alpha^v p_v \geq \beta^v + \sum_{e \in \delta(v)} \gamma_e^v x_e, \\
& v \in V, (\alpha^v, \beta^v, \boldsymbol{\gamma}^v) \in T(C^v), && (32) \\
& \mathbf{x} \in X, \\
& (\mathbf{x}, \mathbf{p}) \in \mathbb{B}^m \times \mathbb{R}^n, && (33)
\end{aligned}$$

where $T(C^v)$ denotes the set of the extreme rays of the projection cone C^v.

Observe that $(\overline{\mathbf{x}}, \overline{\mathbf{p}})$ is feasible for the linear programming relaxation of $F^*_{\mathbf{x}}$ if and only if there exists $\overline{\mathbf{t}}$ such that $(\overline{\mathbf{x}}, \overline{\mathbf{t}}, \overline{\mathbf{p}})$ is feasible for the linear programming relaxation of F^* (extended with variables $\mathbf{p}$). Consequently, F^* and $F^*_{\mathbf{x}}$ yield the same linear programming relaxation bound.

5.2. *A Cutting Plane Algorithm for Solving the Linear Programming Relaxation of $F^*_{\mathbf{x}}$*

Let $\overline{T}(C^v) \subseteq T(C^v), v \in V$, be a (tiny) subset of the extreme rays of C^v. Once again, consider $\overline{R} \subseteq \{U \subset V : |U| \geq 2\}$. Instead of explicitly taking into account all SECs and constraints (32) implied by $\{T(C^v) \colon v \in V\}$ in the linear programming relaxation of $F^*_{\mathbf{x}}$, we consider only those respectively implied by the subsets $\overline{R}$ and $\{\overline{T}(C^v) : v \in V\}$. The resulting relaxation is denoted $\overline{F}^*_{\mathbf{x}}$. If $\overline{F}^*_{\mathbf{x}}$ is solved and its optimal solution satisfies all SECs and projection constraints (32), the solution vector also solves the linear programming relaxation of $F^*_{\mathbf{x}}$. Otherwise, violated SECs and projection cuts are used to reinforce the relaxation. More precisely, we update $\overline{R}$, with vertex sets of violated SECs, or $\{\overline{T}(C^v) : v \in V\}$, with extreme rays corresponding to violated projection cuts (32). A strengthened relaxation $\overline{F}^*_{\mathbf{x}}$ is then optimized. The process is repeated until an optimal solution for the linear programming relaxation of $F^*_{\mathbf{x}}$ is found. The main details are given in the algorithm below, denoted by CP.

CP: Cutting *Plane Algorithm*

1. Formulate and solve the relaxation $\overline{F}^*_{\mathbf{x}}$ implied by the current restricted sets $\overline{R}$ and $\overline{T}(C^v)$. Let $(\overline{\mathbf{x}}, \overline{\mathbf{p}})$ be an optimal solution.
2. Solve (21), the separation problem for the SECs. Let $\overline{V}$ be a vertex set attaining the maximum. If the corresponding SEC is violated, update $\overline{R} \leftarrow \overline{R} \cup \overline{V}$.

3. For every $v \in V$, solve the separation problem for the projection cuts (32):

$$(SP(v)) \quad \max \quad \beta^v + \sum_{e \in \delta(v)} \gamma_e^v \overline{x}_e - \alpha^v \overline{p}^v \tag{34}$$

$$\text{s.t.} \quad (\alpha^v, \beta^v, \gamma^v) \in C_v, \tag{35}$$

$$\alpha^v = 1. \tag{36}$$

Details on how this problem is solved will be given later. Let $(\overline{\alpha}^v, \overline{\beta}^v, \overline{\boldsymbol{\gamma}}^v)$ be an optimal solution. If (32) is violated by $(\overline{\alpha}^v, \overline{\beta}^v, \overline{\boldsymbol{\gamma}}^v)$, update $\overline{T}(C^v) \leftarrow \overline{T}(C^v) \cup \{(\overline{\alpha}^v, \overline{\beta}^v, \overline{\boldsymbol{\gamma}}^v)\}$.
4. If violated constraints were found in steps 2 or 3, go back to step 1. Otherwise, stop since $(\overline{\mathbf{x}}, \overline{\mathbf{p}})$ solves the linear programming relaxation of $F_{\mathbf{x}}^*$.

Similar to steps 3 and 4 of RCG, the separation of projection cuts, step 3 of CP, is executed in parallel for each $v \in V$, in our implementation. Note that we assume $\alpha^v = 1$ for these cuts. Before discussing how they can be separated, we first show that this assumption makes each SP(v) bounded and does not result in losing any eventually violated constraint (32). To that aim, assume for a moment that $\alpha^v > 0$. In this case, since the other variables can be conveniently scaled, we can assume $\alpha^v = 1$. Consider now the dual of SP(v):

$$(\text{DSP}(v)) \quad \min \quad \sum_{H \in S^v} q_H t_H \tag{37}$$

$$\text{s.t.} \quad \sum_{H \in S^v} t_H = 1, \tag{38}$$

$$\sum_{H \in S^v : e \in H} t_H = \overline{x}_e, \quad e \in \delta(v), \tag{39}$$

$$\mathbf{t} \in \mathbb{R}_+^{|S|}. \tag{40}$$

DSP(v) can be seen as the problem of finding the cheapest way to represent $\overline{\mathbf{x}}$ as a convex combination of stars centered at v. Note that the stars centered at v are simply the vertices of the 0–1 box in $\mathbb{R}^{|\delta(v)|}$. Since the point $\overline{\mathbf{x}}$ to be separated satisfies $\overline{\mathbf{x}} \in [0,1]^m$, DSP($v$) is always feasible, which implies that SP(v) is bounded. Now, assume that there is a violated constraint (32) for which $\alpha^v = 0$, that is, there is some $\tilde{\beta}^v$ and $\tilde{\boldsymbol{\gamma}}^v$ such that $\tilde{\beta}^v + \sum_{e \in \delta(v)} \tilde{\gamma}_e^v \overline{x}_e > 0$. This would be an improving ray for SP(v), contradicting the fact that it is bounded. Thus, we can safely assume $\alpha^v = 1$.

Note that DSP(v) closely resembles F^*. In fact, the reduced cost expression for variables t_H, $H \in S$, is precisely the same as in F^*. Thus, we can apply the same algorithm outlined in section 4.2 to find stars of negative reduced cost. Therefore, instead of solving SP(v), we actually solve DSP(v) by column generation. To that aim, consider the restricted problem $\overline{\text{DSP}}(v)$, which is obtained from DSP(v) by considering only variables t_H implied by stars H in a restricted set $\overline{S}^v \subseteq S^v$. The column generation algorithm, denoted CG, is outlined below.

CG: Column Generation Algorithm

1. Formulate and solve the problem $\overline{\text{DSP}}(v)$ implied by the current set $\overline{S}^v$. Let $\overline{\mathbf{t}}$ be an optimal primal solution. Let $\overline{\beta}^v$ and $\overline{\boldsymbol{\gamma}}^v$ be optimal dual variables, respectively, associated with constraints (38) and (39).
2. Let $\tilde{S}^v = \{H \in S^v : |H| \leq K_{\text{enum}}, \overline{q}_H < 0\}$. If $\tilde{S}^v \neq \emptyset$, update $\overline{S}^v \leftarrow \overline{S}^v \cup \tilde{S}^v$ and go back to step 1. Otherwise, proceed to step 3.
3. Solve $PP(v)$. Let $\overline{H}$ be a star attaining the minimum. If $r^v = \overline{q}_{\overline{H}} < 0$, update $\overline{S}^v \leftarrow \overline{S}^v \cup \{\overline{H}\}$ and return to step 1. Otherwise, $\overline{\mathbf{t}}$ is an optimal solution for DSP(v) and $(1, \overline{\beta}^v, \overline{\boldsymbol{\gamma}}^v)$ is an optimal solution for SP(v).

To ensure that a feasible solution for DSP(v) is readily available at each call to CG, we initialize $\overline{S}^v$ with the stars generated by the following algorithm.

Feasible solution for DSP(v)

1. Let $\tilde{\mathbf{x}} = \overline{\mathbf{x}}, \tilde{\mathbf{t}} = 0$, and consider a set $\tilde{S}^v = \emptyset$.
2. Update $\tilde{S}^v \subseteq S^v$ as follows:
   (a) Let $H = \{e \in \delta(v) : \tilde{x}_e > 0\}$ and add H to $\tilde{S}$.
   (b) Let $\tilde{t}_H = \min\{\tilde{x}_e : e \in \delta(v), \tilde{x}_e > 0\}$.
   (c) Set $\tilde{x}_e = \tilde{x}_e - \tilde{t}_H$ for all $e \in \delta(v)$ with $\tilde{x}_e > 0$.
   (d) If $\tilde{\mathbf{x}} = 0$ go to step 3, otherwise repeat step 2.
3. Let $H_\emptyset$ be the empty star and set $\tilde{t}_{H_\emptyset} = 1 - \sum_{H \in \tilde{S}^v} \tilde{t}_H$.

To see that this algorithm indeed gives a feasible solution for DSP(v), observe that

$$\overline{x}_e = \sum_{H \in \tilde{S}^v : e \in H} \tilde{t}_H, \forall e \in \delta(v) \quad \text{and} \quad \sum_{H \in \tilde{S}^v} \tilde{t}_H = 1.$$

In CP, the most expensive step is the separation of (32) in step 3, since separating those constraints involves the exact resolution of PP(v). To accelerate the convergence of the separation algorithm CG and solve as few exact pricing problems as possible, we keep in $\overline{S}^v$ the stars generated during the previous calls to CG. Another strategy to reduce CPU times is to abort CG as soon as the first exact pricing problem is solved. In that case, a feasible solution for the dual of DSP(v) can be obtained by adding r^v to $\overline{\beta}^v$. Since the dual of DSP(v) is SP(v), this solution yields a valid constraint (32). In fact, we resort to this strategy in our branch-and-cut algorithm.

Each PP(v) in CG is solved as discussed in section 4.2. However, in CG it is possible to develop another strategy for accelerating the resolution of that pricing problem. Basically, all edges e such that $\overline{x}_e = 0$ in the point $\overline{\mathbf{x}}$ to be separated can be forbidden in the resolution of the pricing. Consider the dual variables $\overline{\boldsymbol{\gamma}}^v$ in the optimal solution for the dual of $\overline{\text{DSP}}(v)$, obtained in Step 1 of CG. We know from section 4.2 that if $\overline{\gamma}_e^v \leq 0$ for some $e \in \delta(v)$, we can safely assume that e will not be used in the solution of PP(v). Now, if $\overline{\gamma}_e^v > 0$, setting $\overline{\gamma}_e^v = 0$ still gives multipliers that are feasible for the dual of $\overline{\text{DSP}}(v)$. If $\overline{x}_e = 0$, they are also optimal. Consequently, we can set $\overline{\gamma}_e^v = 0$ for all e such that $\overline{x}_e = 0$ and possibly accelerate the exact resolution of PP(v). While this scheme indeed reduces the overall computational effort of the exact pricing, the convergence of CP is badly affected, since sparser cuts are generated by CG, that is, $\overline{\gamma}_e^v = 0$ for all e such that $\overline{x}_e = 0$.

To overcome this undesired side effect, we seek to reduce $\overline{\gamma}_e^v$ by the smallest possible amount while still ensuring that e will be fixed out of the solution of PP(v). Again, let $(f_1, ..., f_{|\delta(v)|-1})$ be a sorting of $\delta(v)\backslash\{e\}$ in nondecreasing order of $\overline{q}_{ef} + \overline{q}_{fe}$. Assume $\overline{\gamma}_e^v > q_{ee}$ and $\sum_{l=1}^{K_{\text{enum}}}(\overline{q}_{ef_l} + \overline{q}_{f_le}) < |\overline{q}_{ee}| = \overline{\gamma}_e^v - q_{ee}$, since otherwise e would have been ruled out by the methods discussed in section 4.2. We then reduce $\overline{\gamma}_e^v$ to $\sum_{l=1}^{K_{\text{enum}}}(\overline{q}_{ef_l} + \overline{q}_{f_le}) + q_{ee}$. In doing so, we guarantee that e is fixed out of PP(v), while keeping part of the dual information obtained through enumeration. Consequently, the resolution of PP(v) becomes faster without slowing down the overall convergence of CP.

Although in CP we still have to solve the same type of pricing problems solved by RCG, there are some advantages, from a computational perspective, in evaluating $L(F^*)$ by means of CP. We highlight some of them below.

Since the spanning tree polytope is completely described by (6)–(8), any attempt at further strengthening F^* by means of additional valid inequalities should be carried out by adding inequalities involving variables t_H, $H \in S$. Such inequalities would, very likely, affect the structure of the pricing problems PP(v), $v \in V$. Conversely, for the case of $F^*_{\mathbf{x}}$, which has a compact variable space, we can easily apply general purpose procedures for generating valid inequalities (Chvátal-Gomory cuts, for instance) to strengthen the formulation. Inequalities added to any relaxation of $F^*_{\mathbf{x}}$ do not change the structure of the problem of separating projection cuts. Being so, general purpose classes of valid inequalities separated by most commercial solvers can be used to strengthen $F^*_{\mathbf{x}}$ in a branch-and-cut framework, without any change in how the separation procedures are implemented. Since only cutting planes are being generated, the implementation of a branch-and-cut algorithm is straightforward since it suffices to embed CP in a branch-and-cut framework, available in most integer programming commercial packages. A final argument in favor of using projection cuts is that the control of the enumeration tree can be left in charge of the commercial solver, while the combined use of RCG forced us to implement the search tree by ourselves. As a result the branch-and-cut algorithm benefits to a larger extent from the expertise in cut generation, variable fixing and selection, branching, and tree management, available in most commercial packages.

5.3. *Branch-and-Cut Implementation Details*

We combine the lower bounding procedure CP with BB to develop BC, a branch-and-cut algorithm for AQMSTP. To implement BC, we simply embed CP into the branch-and-cut framework of a commercial optimization package. We let the solver manage the search and all configuration parameters are left at their default values.

There are two differences between the implementation of CP used in BC and the one described in the previous section. In BC, for each $v \in V$, the separation routine CG of CP is stopped as soon as the first exact pricing problem is solved. The dual feasible solution obtained by adding r^v to $\overline{\beta}^v$ is then used for generating a projection cut. Additionally, we only add to $\overline{R}$ the projection cuts that are violated by at least $0.01 \times \overline{p}^v$.

As in BCP, the BC search tree is initialized with the best AQMSTP feasible solution obtained with the multistart heuristic described in section 4.4.

6. COMPUTATIONAL EXPERIMENTS

In this section, we describe the computational experiments conducted with BCP and BC. The tests were executed on an Intel XEON E5645 machine with two processors, each processor with six cores, each core running at 2.4 GHz. A total of 32 GB of shared RAM memory was available. Ubuntu 12.04 LTS operating system was used.

The algorithms presented here were implemented with the help of the commercial optimization package IBM ILOG CPLEX 12.5. The CPLEX linear programming solver was used to solve the linear programs in RCG and CG. The CPLEX integer programming solver was used to solve integer programs during the exact resolution of the pricing problems. BC was implemented by embedding CP into the CPLEX branch-and-cut framework. In all cases, CPLEX parameters were left at their default values. The OpenMP API was used to parallelize steps 3 and 4 of RCG and step 3 of CP. All algorithms were coded in C++ and compiled with G++ 4.6.3, optimization flag -O3 turned on.

Since AQMSTP instances in the literature were not available to us, new test instances were generated according to the guidelines in [2]. Instances considered here are defined over complete graphs with $n \in \{15, 20, 30, 40, 50\}$ vertices. For each size n, 10 instances were generated. Diagonal entries of Q are randomly chosen integers in the range [0,100], while off-diagonal entries are integers in [0,20], also chosen with uniform probability.

We compare BCP and BC computational results with those provided by the parallel QMSTP BB algorithm in [15]. Since the BB algorithms in [15] and [16] are the same, in this section, we cite only reference [15] for the results that are also available in [16]. The experiments in [15] were conducted on the same computational environment used here, thus, computational results obtained by that algorithm and those we report for BCP and BC are directly comparable. We also compare our lower bounds with those presented in [2]. To that aim, we implemented their lower bounding procedure as described in that reference.

In Table 1, AQMSTP lower bounds are presented. In the first three columns of the table, we provide instance data: the size n, an integer *id* in the range [1, 10] that stands as an instance identifier, and the optimal cost ub* (as obtained by BCP and BC). For each value of n, we provide three sets of results, each one depicted in one row. For a given n, in the first of the three rows we present results for the instance for which $L(F^*)$ provided the smallest duality gap, defined as $100 \times \frac{\text{ub}^* - L(F^*)}{\text{ub}^*}$. In the second of the three rows, we present results averaged over 10 instances. In the third

TABLE 1. Lower bound comparisons.

Instance			F^*				Pereira [15]			Assad and Xu [2]		
					$t(s)$							
n	id	ub*	$L(F^*)$	gap(%)	RCG	CP	lb	gap(%)	$t(s)$	lb	gap(%)	$t(s)$
$n=15$												
min gap	4	271	271	0	2.9	3.4	162.6	39.9	1.1	111.8	58.7	0
avg. gap	1–10	275.7	270	2	3.0	3.6	171.7	37.9	1.2	125.1	54.7	0
max gap	5	206	196	4.8	2.3	2.8	105.3	48.8	1.1	76.2	63	0
$n=20$												
min gap	5	332	332	0	13	15.6	170.7	48.5	3.4	112.6	66	0
avg. gap	1–10	357	348.2	2.4	14.3	18	213.8	40.2	3.4	153.6	57	0
max gap	9	322	304	5.5	13.6	16	165.4	48.6	3.3	110	65.8	0
$n=30$												
min gap	10	418	415	0.7	111.6	242.8	226.9	45.6	14.5	156.4	62.5	0
avg. gap	1–10	434.9	423.3	2.6	134.5	256.5	223.4	48.6	14.5	134.4	69	0
max gap	3	464	441.9	4.7	149.8	291.1	222.5	52	14.2	125.4	72.9	0
$n=40$												
min gap	2	502	488.7	2.6	621.2	2,199.9	229	54.3	71.6	118.6	76.3	0.1
avg. gap	1–10	486.7	468.5	3.7	654.5	1,868.2	229	52.9	70.8	130.9	73	0.1
max gap	4	505	472.7	6.3	690.3	1,942.6	188.7	62.6	71.8	103.5	79.4	0.1
$n=50$												
min gap	9	498	485.7	2.4	2,653.3	1,1189.5	214.1	56.9	185.6	105.9	78.7	0.2
avg. gap	1–10	547	521.4	4.6	2,893.9	13,311.2	234	57.3	187.3	128.8	76.5	0.3
max gap	3	589	550.1	6.5	3,366.9	15,280.7	262.8	55.3	187.6	139.9	76.2	0.3

row, we present results for the instance where $L(F^*)$ provided the worst duality gap. In the four columns under F^*, we provide the lower bound $L(F^*)$, the implied duality gap (minimum, average and maximum, depending on the row of the table) and the CPU times $t(s)$ (in seconds), needed by RCG and CP to evaluate $L(F^*)$ (recall that RCG and CP imply the same lower bound $L(F^*)$). Note that the computational times we present for RCG and CP take into account all preprocessing that is performed, as indicated in section 4.2, to generate information for the exact resolution of the pricing problems. The next three columns in the table are dedicated to the QMSTP lower bounding scheme in Pereira [15]. In those columns, we respectively present the lower bound lb provided by that formulation, the implied duality gap, and the time $t(s)$ needed to evaluate the bounds. In the last columns of the table, similar results are provided for our implementation of the lower bounding procedure in Assad and Xu [2].

Computational results in Table 1 indicate that the lower bounds $L(F^*)$ are much stronger than its competitors, the Gilmore–Lawler based bounds in [15] and [2]. Taking $n=50$ as a baseline, we see that average duality gaps implied by F^* (4.6%) are less than one tenth of the duality gaps implied by the Gilmore–Lawler bounds in [15] and in [2] (respectively, 57.3% and 76.5%).

Compared to RCG, CP demands a much higher computational effort to evaluate $L(F^*)$, as n grows. The reason being that, every time projection cuts are separated, the CG algorithm needs to be executed. Thus, more exact pricing problems need to be solved exactly by CP rather than by the RCG algorithm.

TABLE 2. Comparison of computational times under different strategies for generating rows and columns.

Strategy	SHE	HSE	HES	SE	ES	(HS)E	H(SE)	(SE)
$t(s)$	15.0	20.04	20.2	1,890.8	1,133.5	15.7	20.3	1,349.5

Some computational experiments were performed to justify the sequence according to which our algorithms are called to generate columns and rows in RCG. Various possible strategies were tested. The instances with $n=40$ were used for these experiments. For each possible strategy considered here, average computational times for solving the 10 instances with $n=40$ are presented in Table 2. In the first row of the table, we present the different possible strategies we tested. Each strategy is represented by a string of characters. In the string, the character H stands for the heuristic pricing resolution step, S stands for the step in which the SECs are separated, and E stands for the step in which the pricing problems are solved exactly. The sequence of the letters H,S,E in the string indicates the order of calling the corresponding steps. Two letters between parenthesis indicate that the two corresponding steps where merged into one single step. Observe that it does not make sense to execute the heuristic pricing algorithm together or after the exact pricing algorithm. Thus, such sequences were not considered here. In the second row of Table 2, we present the average computational times (in seconds) over the 10 instances. These times do not account for the time taken during the preprocessing phase. The reason being the fact that the time taken in that phase largely dominates the time that is actually spent

TABLE 3. Impact of the number of calls n_{heu} of the multistart heuristic on BCP and BC running times.

	BCP			BC		
n_{heu}	ub_{heu}	$t_{heu(s)}$	$t_{total(s)}$	ub_{heu}	$t_{heu(s)}$	$t_{total(s)}$
1	513.5	1.7	328.6	516.0	1.8	185.6
10	472.5	15.2	317.9	471.1	15.6	173.9
20	464.9	30.3	325.5	464.2	30.6	182.6
30	461.1	45.2	339.7	460.5	46.1	195.6
40	458.2	60.5	330.9	458.1	61.2	209.5
50	456.3	75.0	349.6	457.6	75.8	225.7

in RCG. Besides that, when BB is executed, preprocessing is performed only once, at the root node.

In Table 2, we see that the sequence adopted in RCG, SHE, is indeed the one that provides the best computational times, on the average, for the $n = 40$ instances. We can also see that the use of the heuristic for solving the pricing problems greatly improves on the computational times.

Before presenting the results for BCP and BC, we present some results supporting our choice of the $n_{heu} = 10$ calls of the multistart heuristic for obtaining initial upper bounds. These results also give a clue on how BCP and BC behave under different choices for that parameter. The results, presented in Table 3, were obtained for instances with $n = 30$ vertices. We tested the following values for n_{heu}: 1, 10, 20, 30, 40, and 50. For each of these values, BCP and BC were executed 10 times, for each of the 10 instances. Thus, 100 executions were performed for each value of n_{heu}. In Table 3, the first column indicates n_{heu}. The columns under headings BCP and BC present the results for the corresponding algorithm. For each algorithm, we present the initial upper bound ub_{heu}, the total time t_{heu} in seconds taken to obtain that bound, and the total time t_{total} in seconds required for the execution of the algorithm, BCP or BC, to completion. Results were averaged over the 100 executions.

Results in Table 3 suggest that, as one could expect, the quality of the initial upper bounds improve and average times t_{heu} increase as the heuristic is called more times. According to these experiments, the best trade off was indeed obtained for the $n_{heu} = 10$ case. Although more experiments should be carried out to define the best value for n_{heu} from a statistical standpoint, it seemed to us that $n_{heu} = 10$ would be a good value to be used in our experiments with BCP and BC.

In Table 4, we present a comparison of BCP, BC, and the parallel QMSTP BB algorithm of [15]. For each algorithm, we report the best upper (ub) and lower (lb) bounds found during the search, the duality gap ($100 \times \frac{\text{ub}-\text{lb}}{\text{ub}}$) at termination, the total number of nodes investigated during the search (nodes), and the CPU time ($t(s)$, in seconds) to obtain such results. Each algorithm was allowed to run for a time limit of 10 hours, for each instance. A failure to solve the problem within the time limit is indicated with the symbol * in the corresponding computational time cell in the table. Since BCP and BC solved all instances with up to 40 vertices to proven optimality whereas the BB in [15] could not solve a single instance with 30 or more vertices, computational results for that algorithm are not provided for $n \geq 40$.

Computational results in Table 4 suggest that both BCP and BC are much faster than the Gilmore–Lawler based BB algorithm of [15]. Although lower bounds given by the procedure of [15] can be evaluated in CPU times that are at least one order of magnitude smaller than the RCG and CP counterparts, the strength of $L(F^*)$ makes up for the additional computational effort. To validate such a claim, observe that, on the average, BCP and BC solve instances with $n \leq 20$ two orders of magnitude faster than their competitor.

For the instances in our test bed, BC clearly outperformed BCP, not only in terms of CPU times, but also in terms of the size of the instances that could be solved to proven optimality. Whereas BCP solved all instances with up to 40 vertices, and only two instances with 50 vertices, BC was able to solve all instances, including all those with 50 vertices.

It is interesting to see that BC largely outperformed BCP, despite the fact that RCG evaluated the lower bound $L(F^*)$ much faster than CP, in the experiments presented in Table 1. The main reason is that in BC, the resolution of BB nodes is actually faster than what Table 1 indicates. Such an observation applies because, as mentioned earlier, the implementation of CP used in BC stops the separation procedure CG as soon as the first exact pricing problem is solved. In addition, only projection cuts that are violated by more than a threshold value are added to the relaxation. Also, most of the projection cuts are generated by BC at the root node; only few additional cuts are generated at the descendant nodes.

Another aspect that might contribute to BC being faster than BCP is the fact that, for BC, the search tree was managed by CPLEX, whereas, for BCP, the search tree was implemented by ourselves in a depth-first fashion. BC is likely to benefit from several policies for variable fixing, variable selection, and heuristics that being hidden to us, could not be implemented within BCP.

7. CONCLUSIONS AND FUTURE RESEARCH

In this work, we investigated the adjacent only version of the QMSTP. We discussed how Gilmore–Lawler based approaches fail in providing good lower bounds for the problem. As an attempt to obtain better bounds, a reformulation based on the stars of G was introduced. We presented two strategies to compute the linear programming lower bounds provided by that reformulation. The first is a combined RCG procedure while the second is a cutting plane method, based on a projection of the proposed model. Two exact solution approaches were implemented and tested: a branch-and-cut-and-price algorithm based on the first and a branch-and-cut procedure based on the second. Computational experiments conducted here indicate that our lower bounds are much stronger than previous bounds in the literature. As a consequence, our exact methods managed to solve instances with up to $n = 50$ vertices, while the previous algorithms in the literature could not solve instances with more than $n = 20$ vertices.

TABLE 4. BB comparisons.

Instance			BCP					BC					Pereira [15]				
n	id	ub*	ub	lb	gap(%)	nodes	*t(s)*	ub	lb	gap(%)	nodes	*t(s)*	ub	lb	gap(%)	nodes	*t(s)*
$n = 15$																	
min gap	4	271	271	271	0	1	3	271	271	0	0	3.2	271	271	0	385	12
avg. gap	1–10	275.7	275.7	275.7	0	4.4	3.5	275.7	275.7	0	3.2	3.4	275.7	275.7	0	446.2	11.9
max gap	5	206	206	206	0	9	3.2	206	206	0	11	2.8	206	206	0	453	13.4
$n = 20$																	
min gap	5	332	332	332	0	1	13.2	332	332	0	0	13.8	332	332	0	1,9583	1,213
avg. gap	1–10	357	357	357	0	4.6	17.0	357	357	0	14.7	16.2	357	357	0	12,731.3	752.2
max gap	9	322	322	322	0	7	18.8	322	322	0	40	16.4	322	322	0	19,963	1,215.3
$n = 30$																	
min gap	10	418	418	418	0	5	199.3	418	418	0	0	126.7	432	226.4	47.5	18,011	36,000*
avg. gap	1–10	434.9	434.9	434.9	0	11.4	283.1	434.9	434.9	0	43.7	169.9	452.3	223.2	50.6	18,111	36,000*
max gap	3	464	464	464	0	13	333.3	464	464	0	112	194.6	486	222.7	54.1	18,011	36,000*
$n = 40$																	
min gap	2	502	502	502	0	11	3,037.8	502	502	0	46	991	—	—	—	—	—
avg. gap	1–10	486.7	486.7	486.7	0	26.8	5,817.8	486.7	486.7	0	126.9	1,081.5	—	—	—	—	—
max gap	4	505	505	505	0	81	19,105.5	505	505	0	591	1,577	—	—	—	—	—
$n = 50$																	
min gap	9	498	498	498	0	9	10,320.7	498	498	0	34	4,239.2	—	—	—	—	—
avg. gap	1–10	547	556.9	524.5	5.8	27.9	33,013.0	547	547	0	638.7	7,921.6	—	—	—	—	—
max gap	3	589	602	550.1	8.6	39	36,000*	589	589	0	1,042	9,974.1	—	—	—	—	—

The symbol * indicates that the corresponding algorithms could not solve the problem within the specified time limit of 10 hours.

Our computational experiments indicate that increasing n does not worsen the duality gaps as much as it increases the computational times needed to evaluate our linear programming lower bounds. Such an observation suggests that faster algorithms for evaluating these bounds might lead to the resolution of even larger instances. A possible approach we intend to investigate in that direction is the dualization of the projection constraints (32) in a Lagrangian relaxation framework.

REFERENCES

[1] T. Achterberg, T. Koch, and A. Martin, Branching rules revisited, Oper Res Lett 33 (2005), 42–54.

[2] A. Assad and W. Xu, The quadratic minimum spanning tree problem, Naval Res Logist 39 (1992), 399–417.

[3] R.E. Burkard, E. Çela, P.M. Pardalos, and L.S. Pitsoulis, "The quadratic assignment problem," Handbook of combinatorial optimization, volume 3, D.Z. Du and P.M. Pardalos (Editors), Kluwer Academic Publishers, Boston, 1998, pp. 241–337.

[4] A. Caprara, Constrained 0–1 quadratic programming: Basic approaches and extensions, Eur J Oper Res 187 (2008), 1494–1503.

[5] R. Cordone and G. Passeri, Solving the quadratic minimum spanning tree problem, Appl Math Comput 218 (2012), 11597–11612.

[6] P.C. Gilmore, Optimal and suboptimal algorithms for the quadratic assignment problem, J Soc Ind Appl Math 10 (1962), 305–313.

[7] E.L. Lawler, The quadratic assignment problem, Manage Sci 9 (1963), 586–599.

[8] J. Lee and J. Leung, On the Boolean quadric forest polytope, INFOR 42 (2004), 125–141.

[9] Y. Li, P. Pardalos, K. Ramakrishnan, and M. Resende, Lower bounds for the quadratic assignment problem, Ann Oper Res 50 (1994), 387–410.

[10] S.M.D.M. Maia, E.F.G. Goldbarg, and M.C. Goldbarg, On the biobjective adjacent only quadratic spanning tree problem, Electron Notes Discrete Math 41 (2013), 535–542, [Presented at the 2013 International Network Optimization Conference (INOC)].

[11] R.K. Martin, Large scale linear and integer optimization: A unified approach, Kluwer Academic Publishers, Boston, 1999.

[12] T. Öncan and A.P. Punnen, The quadratic minimum spanning tree problem: A lower bounding procedure and an efficient search algorithm, Comput Oper Res 37 (2010), 1762–1773.

[13] M.W. Padberg and L.A. Wolsey, Trees and cuts, Annals of Discrete Mathematics 17 (1983), 511–517.

[14] G. Palubeckis, D. Rubliauskas, and A. Targamadzė, Metaheuristic approaches for the quadratic minimum spanning tree problem, Inf Technol Control 39 (2010), 257–268.

[15] D.L. Pereira, Formulations and algorithms based on linear integer programming for the quadratic minimum spanning

tree problem, Ph.D. Thesis, Universidade Federal de Minas Gerais, Brazil, 2014.

[16] D.L. Pereira, M. Gendreau, and A.S. da Cunha, Lower bounds and exact algorithms for the quadratic minimum spanning tree problem, Comput Oper Res (in press).

[17] D.L. Pereira, M. Gendreau, and A.S. da Cunha, Stronger lower bounds for the quadratic minimum spanning tree problem with adjacency costs, Electron Notes Discrete Math 41 (2013), 229–236, [Presented at the 2013 International Network Optimization Conference (INOC)].

[18] D. Pisinger, The quadratic knapsack problem—a survey, Discrete Appl Math 155 (2007), 623–648.

[19] S. Sundar and A. Singh, A swarm intelligence approach to the quadratic minimum spanning tree problem, Inf Sci 180 (2010), 3182–3191.

[20] D. Wang and R. Kleinberg, Analyzing quadratic unconstrained binary optimization problems via multicommodity flows, Discrete Appl Math 157 (2009), 3746–3753.

[21] G. Zhout and M. Gen, An effective genetic algorithm approach to the quadratic minimum spanning tree problem, Comput Oper Res 25 (1998), 229–237.