Formulations and Exact Solution Approaches for the Degree Preserving Spanning Tree Problem

Alexandre Salles da Cunha
Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, Brasil

Luidi Simonetti
Instituto de Computação, Universidade Federal Fluminense, Niteroi, Brasil

Abilio Lucena
Departamento de Administração and Programa de Engenharia de Sistemas e Computação, Universidade Federal do Rio de Janeiro, Brasil

Bernard Gendron
Département d'informatique et de recherche opérationnelle, Université de Montréal, Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT), Montréal, Québec, Canada

Given a connected and undirected graph *G*, the degree preserving spanning tree problem (DPSTP) asks for a spanning tree of *G* with the maximum number of vertices having the same degree in the tree and in *G*. These are called full degree vertices. We introduce integer programming formulations, valid inequalities and four exact solution approaches based on different formulations. Two branch-and-bound procedures, a branch-and-cut (BC) algorithm and an iterative probing combinatorial Benders decomposition method are introduced here. The problem of optimally lifting one of the classes of valid inequalities proposed here is equivalent to solving a DPSTP instance, for a conveniently defined subgraph of *G*. We thus apply one of the proposed methods to optimally lift these cuts, within the other solution methods. In doing so, two additional algorithms, a hybrid Benders decomposition and a hybrid BC are proposed. Extensive computational experiments are conducted with the solution algorithms introduced in this study. **NETWORKS, Vol. 65(4), 329–343 2015**

Received September 2013; accepted November 2014
Correspondence to: A. Salles da Cunha; e-mail: acunha@dcc.ufmg.br
Contract grant sponsor: CNPq; Contract grant number: 471464/2013-9, 305423/2012-6, 477863/2010-8 (A.S.d.C.)
Contract grant sponsor: FAPEMIG; Contract grant number: PRONEX APQ-01201-09 and PPM-VII-00164-13 (A.S.d.C.)
Contract grant sponsor: FAPERJ and CNPq; Contract grant number: 483.243/2010-8, 304793/2011-6 (L.S.)
Contract grant sponsor: CNPq; Contract grant number: 310561/2009-4 (A.L.)
Contract grant sponsor: FAPERJ; Contract grant number: E26-110.552/2010 (A.L.)
Contract grant sponsor: Natural Sciences and Engineering Council of Canada (NSERC) Discovery Grants Program (B.G.)
DOI 10.1002/net.21590
Published online 11 March 2015 in Wiley Online Library (wileyonlinelibrary.com).

Keywords: spanning trees; vertex feedback edge set problem; benders decomposition; branch-and-cut; Lagrangian relaxation

1. INTRODUCTION

Assume, we are given a connected undirected graph $G = (V, E)$ with $n = |V|$ vertices and $m = |E|$ edges, and let $G' = (V, E')$ be a subgraph of G. Denoting by $\delta_G(i)$ and $\delta_{G'}(i)$, the sets of edges incident to vertex i respectively in G and G', whenever $|\delta_{G'}(i)| = |\delta_G(i)|$, i is of full degree (or degree preserving) in G'. Otherwise, if $|\delta_{G'}(i)| < |\delta_G(i)|$, i is of incomplete degree in that subgraph. The degree preserving spanning tree problem (DPSTP) asks for a spanning tree of G with as many full degree vertices as possible. The problem is also known in the literature as the full degree spanning tree problem [1]. For the input graph given on the left of Figure 1, we provide two spanning trees with different numbers of full degree vertices. The spanning tree on the right has two full degree vertices and the one in the middle has only one.

DPSTP has also been investigated in [11] under a complementary problem named the vertex feedback edge set problem (VFESP). The VFESP looks for a cotree (the complement of a tree) of G incident to the minimum number of vertices. As the vertices that are not incident to the edges of a cotree are exactly those with full degree in its corresponding tree, a solution to one problem directly leads to a solution to the other.

The most widely known application of DPSTP originates from a VFESP context: one is given a water distribution network where flows in the edges (pipes) are to be measured [19]. That could be accomplished by simply installing flow meters in all network edges. However, a cheaper alternative is

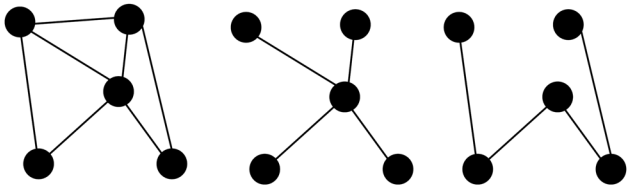

FIG. 1. For a given input graph on the left, two spanning trees with one and two full degree vertices.

to install flow meters only in the edges of a cotree of the network. Flow for the remaining edges could then be inferred by assuming that conservation laws apply. In doing so, precisely $m - n + 1$ flow meters would be required. Another attractive alternative is to install pressure meters in the vertices of the network and then use the pressure drop in the endpoints of a pipe to calculate its flow. According to [19], it suffices to install pressure meters at vertices that are incident to the edges of a cotree of the network. An advantage of the latter approach is that pressure meters are much cheaper than flow meters [19]. As a result, minimizing the number of vertices incident to the cotree leads to the most economical choice. Finally, as long as Kirchoff's conservation laws apply, such a DPSTP/VFESP application could be found in other types of networks (electrical, acoustical, or thermal, for example), where one wishes to measure different types of flows [2].

On the complexity side, DPSTP was proven to be NP-complete, even if G is a split or bipartite planar graph [1, 2]. If G is planar, DPSTP admits a linear time approximation algorithm. The problem is polynomially solvable when G has a bounded treewidth [2], when G is strongly chordal [13] or a directed path graph [13]. Additional polynomial time algorithms for DPSTP were suggested in [2] for cocomparability graphs and for graphs with bounded asteroidal number. The directed version of DPSTP was investigated in [14].

Lewinter [12] proved that the number of full degree vertices in spanning trees of G interpolates, meaning that if there exist two spanning trees T_1 and T_2 with, respectively t_1 and t_2 full degree vertices, there must exist a spanning tree T with exactly t full degree vertices, for $t_1 < t < t_2$. Such a property is of key importance for the development of one of the exact solution approaches introduced here.

To date, solution approaches to DPSTP are mostly restricted to approximation algorithms [2] and heuristics [11, 19]. Bhatia et al. [1] propose a polynomial time approximation algorithm with an $O(\sqrt{n})$ approximation factor. To the best of our knowledge, the only existing exact solution approaches to DPSTP are the branch-and-bound (BB) algorithm in Bhatia et al. [1] and the branch-and-cut (BC) algorithm in Gendron et al. [6]. The former algorithm is based on linear programming (LP) upper bounds given by a multicommodity network flow formulation of the problem. However, virtually no computational results are available for it.

This paper improves on our previous results [6] in various ways. Specifically, we introduce additional families of valid inequalities for DPSTP and three new exact solution approaches for the problem, a combinatorial Benders decomposition method and two BB algorithms. One of the BB algorithms indirectly relies on Lagrangian relaxation upper bounds, while the other is based on a technique known as reformulation by intersection [8]. These new algorithms are compared here to an improved version of the BC algorithm in [6]. In this process, additional optimality certificates are obtained for the test instances used in [6].

The article is organized as follows. In Section 2, we describe four integer programming formulations of DPSTP and some families of valid inequalities for the problem. Each formulation gives rise to a different exact solution algorithm, described in Section 3. Computational experiments are reported in Section 4 and the article is closed in Section 5, with suggestions for future research.

2. INTEGER PROGRAMMING FORMULATIONS

Let $D = (V, A)$ be the directed graph originating from G after taking $A = \{(i,j) \cup (j,i) : \{i,j\} \in E\}$. For any $S \subseteq V$, denote by $\Gamma(S)$ the closed neighborhood $S \cup \{j \in V : \{i,j\} \in \delta_G(i), i \in S\}$ of S. Additionally, let $E(S) = \{\{i,j\} \in E : i,j \in S\}$ and $\delta_G(S) = \{\{i,j\} \in E : i \in S, j \notin S\}$ denote, respectively, the subset of edges of E with both endpoints in S and the subset formed by the edges of E with only one endpoint in S. Corresponding notation for D is $A(S, V\backslash S) = \{(i,j) \in A : i \in S, j \notin S\}$, which denotes the set of arcs pointing from a vertex in S to a vertex in $V\backslash S$. We assume that $\mathcal{T}$ denotes the set of all spanning trees of G, while $\mathcal{F}$ denotes the collection of all cycle-free subgraphs of G. For a given spanning tree $T = (V, E_T)$ of G and an edge $e \in E\backslash E_T$, we use $C_{T,e}$ for the set of vertices in the unique cycle of subgraph $(V, E_T \cup \{e\})$. Given any real valued function f defined over a finite set Q and a subset $Q' \subseteq Q$, $f(Q') = \sum_{q \in Q'} f_q$ applies.

For simplicity, from now on, for a given $S \subseteq V$, whenever we refer to edges of the input graph G and not to those of a subgraph of G, we will be using $\delta(S)$ instead of $\delta_G(S)$. Additionally, for any formulation P of DPSTP, the LP upper bound implied by it will be denoted by $w(P)$. Finally, if S contains a single element, say i, $\delta(i)$ will be used instead of $\delta(\{i\})$.

Optimization problems that ask for optimal trees with side constraints can be formulated in many different ways, depending on how solution connectivity is enforced. In this investigation, a formulation that only involves decision variables associated with choosing full degree vertices in cycle free subgraphs is initially presented. Then, we describe an undirected formulation for which LP upper bounds, although very weak, can be computed very efficiently. A third formulation is a directed model that is reinforced with additional valid inequalities. Some of these inequalities turn the formulation into a nonsymmetrical model, that is, LP upper bounds vary according to the vertex chosen as the arborescence root. Accordingly, we resort to the reformulation by intersection technique [8] to obtain a fourth DPSTP model,

originating from the previous one, for which LP upper bounds are symmetrical.

No matter how and if connectivity conditions are explicitly imposed, all formulations above contain inequalities that exploit the following properties of full degree sets of vertices.

Proposition 1. *Given $V' \subseteq V$, there exists a spanning tree $T = (V, E_T)$ of G such that $|\delta_T(i)| = |\delta(i)|$ for all $i \in V'$, if and only if subgraph $(\Gamma(V'), \delta(V') \cup E(V'))$ belongs to $\mathcal{F}$.*

Proof. If subgraph $(\Gamma(V'), \delta(V') \cup E(V'))$ contains a cycle, at least one of its edges must not be a spanning tree edge. Accordingly, a spanning tree of G where all vertices in V' are full degree would not exist. Conversely, if $(\Gamma(V'), \delta(V') \cup E(V'))$ is cycle free, then there must be a spanning tree (V, E_T) such that $(\delta(V') \cup E(V')) \subseteq E_T$, otherwise G would not be connected. ∎

Corollary 2. *If $G_{V'} = (V', E(V')) \notin \mathcal{F}$, for $V' \subseteq V$, then at least two vertices of V' must not be of full degree. A particular case of these subgraphs occurs when there exists a simple cycle of $G_{V'}$ visiting all vertices of V'.*

The formulations that follow use binary 0-1 variables $\{y_i : i \in V\}$ to select full degree vertices. Accordingly, $y_i = 1$ if i is of full degree and $y_i = 0$, otherwise.

2.1. Formulation Based only on the y Variables

Proposition 1 and Corollary 2, respectively, translate into the split cuts (1) and the cycle cuts (2) that follow:

$$y(V') \leq |V'| - 1, \text{for } V' \subset V \text{ such that } (\Gamma(V'), \delta(V') \cup E(V')) \notin \mathcal{F} \tag{1}$$

and

$$y(V') \leq |V'| - 2, \text{ for } V' \subset V \text{ such that } (V', E(V')) \notin \mathcal{F}. \tag{2}$$

Notice that cycle cuts (2) can be understood as a stronger particular case of split cuts (1), that is, when there exists at least one edge in $E(V')$ that is surely not in any spanning tree of G. Notice as well that both (1) and (2) may be lifted to:

$$y(V') \leq \alpha(V'), \tag{3}$$

where $\alpha(V')$ is the optimal objective function value of the following combinatorial optimization problem:

$$\alpha(V') = \max\{|\bar{V}| : \bar{V} \subseteq V', (\Gamma(\bar{V}), \delta(\bar{V}) \cup E(\bar{V})) \in \mathcal{F}\}. \tag{4}$$

Obtaining optimal values $\alpha(V')$ is as hard as solving the original problem. However, when some particular structures apply, the picture changes for the better. For instance, if $(V', E(V'))$ defines a clique of G, the right hand side of (2) could be decreased to 1. Likewise, if $|E(V')| \geq |V'|+1$ holds, at least two edges of $E(V')$ may not be used in a degree preserving spanning tree. Therefore, the right hand side of cycle cuts (2) could be lifted to $|V'| - 3$. Obviously, when no alternative is available, valid upper bounds on $\alpha(V')$ may be used to produce weaker versions of the cuts.

Finally, if P_C represents the intersection of all constraints (1) and (2), a formulation of DPSTP is given by

$$w = \max\{y(V) : y \in P_C \cap \mathbb{B}^n\}. \tag{5}$$

2.2. Undirected Formulation of DPSTP

Binary 0-1 variables $\{z_{ij} : \{i,j\} \in E\}$ are used to select spanning tree edges. Accordingly, $z_{ij} = 1$ holds if edge i, j is selected while $z_{ij} = 0$ applies, otherwise. DPSTP is then formulated as:

$$w = \max\{y(V) : (z, y) \in P_u \cap \mathbb{B}^{m+n}\}, \tag{6}$$

where P_u denotes the intersection of the spanning tree polytope P_{STP}, that is,

$$z(E) = n - 1 \tag{7a}$$

$$z(E(S)) \leq |S| - 1, S \subset V, S \neq \emptyset, \tag{7b}$$

$$z_e \leq 1, e \in E, \tag{7c}$$

$$z_e \geq 0, e \in E, \tag{7d}$$

with the degree-enforcing inequalities

$$y_i|\delta(i)| \leq z(\delta(i)), \forall i \in V. \tag{8}$$

Constraints (8) guarantee that all edges incident to i must be selected, whenever $y_i = 1$ holds. However, they do not explicitly forbid selection of any of these edges when $y_i = 0$ holds, instead. In any case, irrespective of that, due to the objective function used, the latter type of solution would never be optimal. Therefore, although (6) is not a DPSTP formulation in the strict sense, it may be used to find optimal solutions for DPSTP. Likewise, valid DPSTP upper bounds are given by the LP relaxation of (6). One important property about the LP relaxation of (6) is summarized in the following result.

Proposition 3. *The value of the LP relaxation of (6) can be obtained by solving the following LP:*

$$w(P_u) = \max\left\{\sum_{\{i,j\} \in E} (1/|\delta(i)| + 1/|\delta(j)|) z_{ij} : z \in P_{STP}\right\}. \tag{9}$$

In other words, (6) and (9) are LP equivalent.

Proof. It is not difficult to verify, for the LP relaxation of (6), that inequalities (8) must be tight. Indeed, inequalities (8) involve only one entry of y at a time and it would thus be suboptimal to leave any constraint (8) slack. Therefore,

replacing y_i in the objective function by $\frac{1}{|\delta(i)|}z(\delta(i))$, for all $i \in V$, and dropping inequalities (8), DPSTP upper bounds can be obtained by solving (9). ∎

The LP (9) can be linked to a Lagrangian relaxation of model (6). If constraints (8) are relaxed and dualized in a Lagrangian fashion, with multipliers $\{\mu_i \geq 0 : i \in V\}$ attached to them, the corresponding Lagrangian subproblem is $L(\mu) = \max\{\sum_{i\in V}(1 - \mu_i|\delta(i)|)y_i + \sum_{i\in V}\mu_i z(\delta(i)) : z \in P_{\text{STP}} \cap \mathbb{B}^m\}$. Given that the Lagrangian subproblem has the integrality property, the Lagrangian dual gives the same bound as the LP relaxation. Hence, optimal multipliers are $\mu_i = \frac{1}{|\delta(i)|}$ forall $i \in V$ and (9) is precisely the (optimal) Lagrangian subproblem.

As our computational results demonstrate, DPSTP upper bounds $w(P_u)$ are weak. However, an effective BB algorithm results from them, because these bounds are efficiently obtained by generating maximum weight spanning trees of G, under the conveniently defined weight function that results from (9).

A similar formulation and exact solution algorithm were proposed by Fujie [5] for the maximum leaf spanning tree problem (MLSTP). In that case, constraints akin to (8) are used to characterize leaf nodes in spanning trees. Bearing in mind that those constraints would always be tight at LP relaxations for MLSTP, Fujie used them to project out leaf defining variables from the objective function in the MLSTP formulation. As a consequence, the LP relaxation of that formulation could be evaluated by solving a maximum weight spanning tree problem, akin to (9), with other conveniently defined weights. Therefore, the LP relaxation bounds provided by (9) are obtained essentially following the guidelines proposed by Fujie for MLSTP.

Formulation P_u can be significantly strengthened with valid inequalities such as $y_i \leq z_{ij}$ and $y_j \leq z_{ij}$ forall $\{i,j\} \in E$. These inequalities imply that a vertex cannot be of full degree if an edge incident to it is missing from the spanning tree. We do not use them in P_u, as the efficient bounding procedure outlined above would not apply anymore. Directed versions of these inequalities are going to be considered next, in a directed formulation of the problem.

2.3. *Directed Formulation for DPSTP*

The topology behind the DPSTP formulation to be described next is that of a spanning arborescence of digraph $D = (V, A)$, rooted at a preselected vertex r. Besides the previously defined variables $\{y_i : i \in V\}$, the formulation also involves binary variables $\{x^r_{ij} : (i,j) \in A\}$, to establish whether or not an arc $(i,j) \in A$ appears in the arborescence. In case it does, $x^r_{ij} = 1$ holds; otherwise, $x^r_{ij} = 0$ applies. The formulation is given by

$$w = \max\left\{y(V) : (x^r, y) \in P^r_d \cap \mathbb{B}^{2m+n}\right\}, \tag{10}$$

where polytope P^r_d is defined as

$$x^r(A) = n - 1 \tag{11a}$$

$$x^r(A(V \setminus \{i\}, \{i\})) = 1, \forall i \in V \setminus \{r\} \tag{11b}$$

$$x^r(A(V \setminus S, S)) \geq 1, \forall S \subset V \setminus \{r\}, S \neq \emptyset \tag{11c}$$

$$y_i - x^r_{ij} - x^r_{ji} \leq 0, \forall i \in V, \{i,j\} \in \delta(i) \tag{11d}$$

$$x^r_{ij} \geq 0, \forall (i,j) \in A, \tag{11e}$$

$$y_i \geq 0, \forall i \in V. \tag{11f}$$

Constraints (11d) enforce that i is of full degree only if an arc corresponding to every edge $\{i,j\} \in \delta(i)$, that is, (i, j) or (j, i), appears in the arborescence. Constraints (11a) enforce that as many arcs as it is necessary to obtain a spanning arborescence of D are selected. Directed cutset inequalities (11c) guarantee that there must exist a path connecting r to every other vertex in the solution. Indegree constraints (11b) are implied by (11a) and (11c). To verify that, take $S = \{i\}$ for $i \neq r$ and write $x^r(A(V \setminus \{i\}, \{i\})) \geq 1$. Summing over $i \in V \setminus \{r\}$, one obtains $\sum_{i\in V\setminus\{r\}} x^r(A(V \setminus \{i\}, \{i\})) \geq n - 1$. As for any feasible solution $x^r(A(V \setminus \{r\}, \{r\})) = 0$ holds, one has $\sum_{i\in V\setminus\{r\}} x^r(A(V \setminus \{i\}, \{i\})) = x^r(A) - x^r(A(V \setminus \{r\}, \{r\})) = x^r(A) = n - 1$. Thus, as $x^r_{ij} \geq 0$ for every $(i,j) \in A$, $x^r(A(V \setminus \{i\}, \{i\})) \geq 1$ must hold tight and (11b) follows. Despite that, constraints (11b) are included in the model, as our computational experience indicates that they help in computing the bound $w(P^r_d)$ at lower CPU times.

In addition to x^r and y, the multicommodity flow DPSTP formulation in [1] relies on real valued flow variables and flow balance constraints to ensure solution connectivity. The projection of that formulation into the x^r space is precisely the intersection of (11a), (11b), (11c), and (11e) [16]. The formulation in [1] also uses constraints (11d) to define full degree vertices. As a result, P^r_d and the formulation in [1] produce the same LP bound, $w(P^r_d)$. Consider now the following degree enforcing inequalities:

$$x^r(A(\{i\}, V \setminus \{i\})) - y_i \leq |\delta(i)| - 2, \quad \forall i \in V \setminus \{r\} \tag{12a}$$

$$x^r(A(\{r\}, V \setminus \{r\})) - y_r \leq |\delta(r)| - 1 \tag{12b}$$

$$x^r(A(\{i\}, V \setminus \{i\})) - (|\delta(i)| - 1)y_i \geq 0 \quad \forall i \in V \setminus \{r\}. \tag{12c}$$

Constraints (12a) impose that, if $i \in V \setminus \{r\}$ and i is not of full degree, at most $|\delta(i)| - 2$ arcs must point outward of i ($|\delta(i)| - 1$, in case $y_i = 1$). A similar reasoning applies to inequalities (12b). Conversely, constraints (12c) guarantee that, if $y_i = 1$, exactly $|\delta(i)| - 1$ arcs must leave vertex i. Notice that, summing up constraints (11d) for all $\{i,j\} \in \delta(i)$ and using (11b), one ends up with $x(A(\{i\}, V \setminus \{i\})) \geq |\delta(i)|y_i - 1$, which is weaker than (12c). Thus, stronger LP upper bounds are likely to be obtained if the three families of inequalities indicated above are appended to P^r_d. Assume thus that P^r_D denotes the intersection of polytope P^r_d with constraints (12).

Constraints akin to (12) appear in directed models for other combinatorial optimization problems defined over spanning trees, where the degree of the vertices play a role either in the feasibility of the tree or in the objective cost function. That is, the case of the MLSTP [4, 15] and the min-degree constrained minimum spanning tree problem [17], for instance.

Although for our test bed instances, no bound improvements were observed after appending (12a) and (12b) to the formulation, it is not difficult to establish that these inequalities are not redundant. To that aim, one can simply pick any vertex i and optimize the linear function $x^r(A(\{i\}, V \setminus \{i\})) - y_i$ over P^r_d, together with (12c). In doing that, it could be shown that the objective function of that LP exceeds the right-hand-side of (12a). In addition, our computational experiments indicate that fewer BC nodes result when these two inequalities are kept in the model.

After appending inequalities (12) to P^r_d, the resulting formulation P^r_D may become non symmetrical with respect to the root choice. In other words, depending on which vertex r is used as the arborescence root, different DPSTP LP relaxation upper bounds may result. As the next result indicates, formulation P^r_d (and P^r_D) are stronger than P_u.

Proposition 4. $w(P^r_d) \leq w(P_u)$

Proof. Let $(\bar{x}^r, \bar{y}) \in P^r_d$. For every $i \in V$, sum constraints (11d) over $\{i,j\} \in \delta(i)$ to obtain $|\delta(i)|\bar{y}_i \leq \sum_{\{i,j\}\in\delta(i)}(\bar{x}^r_{ij} + \bar{x}^r_{ji})$. Define $\hat{y} = \bar{y}$ and $\hat{z}_{ij} = \bar{x}^r_{ij} + \bar{x}^r_{ji}$ for every $\{i,j\} \in E$. Thus, constraints (8) are satisfied by $(\hat{x}, \hat{y})$. As $\hat{z}$ must be a point in P_{STP} (see [16] for details), $(\hat{z}, \hat{y}) \in P_u$ and the result follows. ∎

2.3.1. Additional Valid Inequalities Under the spanning arborescence representation of feasible DPSTP solutions, the following inequalities, derived from cycle cuts (2), are valid.

Proposition 5. *Assume that a set of vertices $C \subset V$ is given where, $G_C = (C, E(C)) \notin \mathcal{F}$, $|\delta_{G_C}(i)| \geq 2$, $i \in C$, G_C is connected, and there exists a simple cycle of G that visits every vertex in C. Let j and k be the neighbor vertices to i in that simple cycle. A valid inequality for DPSTP is then given by*

$$y(C \setminus \{i\}) + x^r_{ij} + x^r_{ji} + x^r_{ik} + x^r_{ki} \leq |C| - 1, \{i,j,k\} \subset C, \{i,j\}, \{i,k\} \in E(C), G_C \notin \mathcal{F}. \quad (13)$$

Proof. We have three cases to consider for a feasible solution (x^r, y):

- Case 1: $x^r_{ij} + x^r_{ji} = x^r_{ik} + x^r_{ki} = 0$.
  From (2) and $y_i \geq 0$, one has: $y(C \setminus \{i\}) \leq y(C) \leq |C| - 2 \leq |C| - 1$.
- Case 2: Only one edge incident to i exists in the cycle, say i, j, that is, $x^r_{ij} + x^r_{ji} = 1$. For the other edge, $x^r_{ik} + x^r_{ki} = 0$.
  Then, as neither (i, k) nor (k, i) is included in the arborescence, the maximum number of vertices in $C \setminus \{i\}$ having full degree is $|C|-2$. Thus $y(C \setminus \{i\}) + x^r_{ij} + x^r_{ji} + x^r_{ik} + x^r_{ki} \leq |C| - 2 + 1 = |C| - 1$ holds.
- Case 3: $x^r_{ij} + x^r_{ji} = x^r_{ik} + x^r_{ki} = 1$.
  Then at least one edge in the remaining cycle edges $(E(C) \setminus \{\{i,j\}, \{i,k\}\})$ cannot have arcs corresponding to it in the arborescence. As the endpoints of that edge cannot be of full degree, the maximum number of full degree vertices in $C \setminus \{i\}$ is $|C|-3$. Thus $y(C \setminus \{i\}) + x^r_{ij} + x^r_{ji} + x^r_{ik} + x^r_{ki} \leq |C| - 3 + 2 = |C| - 1$ holds and the proof is complete. ∎

2.4. Reformulation by Intersection

One drawback of formulation P^r_D is that we cannot anticipate, beforehand, a root vertex leading to the strongest LP relaxation bound. Attempting to overcome that limitation and obtain a symmetric formulation that hopefully provides stronger LP relaxation bounds, we make use of a technique known as reformulation by intersection, proposed by Gouveia and Telhada [8]. The procedure was originally introduced for the multiweighted Steiner tree problem and was later on used to derive stronger formulations for other combinatorial optimization problems like the min-degree constrained minimum spanning tree problem [17, 18] and the minimum spanning tree problem with a lower bound on the number of leaves [9].

The idea is to reformulate DPSTP by simultaneously taking into account all polytopes P^r_D, for $r \in V$, and then impose that the individual arborescences resulting from each of them originate from a same set of edges of E. That condition is imposed by intersection constraints

$$z_{ij} = x^r_{ij} + x^r_{ji}, \{i,j\} \in E, r \in V, \quad (14)$$

and DPSTP may thus be reformulated as

$$w = \max\{y(V) : (x^1, \ldots, x^n, z, y) \in P_I \cap (\mathbb{R}^{2mn} \times \mathbb{R}^m \times \mathbb{B}^n)\}, \quad (15)$$

where polytope P_I is given by the intersection of (14) and $\cap^n_{r=1} P^r_D$. An important property associated with formulation P_I is its compactness, that is, neither subtour breaking constraints (7b) nor directed cutset constraints (11c) are required by it (see [8, 17] for details). Indeed, once the intersection constraints (14) are imposed, (7b) and (11c) become redundant.

3. ALGORITHMS

In this section, we introduce four exact solution procedures for DPSTP. Namely, a combinatorial Benders decomposition algorithm based on formulation P_C, CBEN, a BB algorithm based on P_u, BBU, a BC algorithm based on P^r_D reinforced with inequalities (1), (2), and (13), BCD, and, finally, a BB algorithm based on P_I, BBI.

BBU is a depth-first BB algorithm that uses (9) to obtain valid upper bounds for DPSTP. It also uses the spanning tree of G that solves (9) to derive valid lower bounds for the problem. BBI was implemented by means of the XPRESS solver Mosel programming language. All XPRESS default parameters for heuristics, branching rules and cutting policies were kept for that algorithm. As P_I is compact, other than loading the corresponding DPSTP model into XPRESS, no significant implementation work for BBI was actually conducted by us. Therefore, no further details are provided on that algorithm. BCD and CBEN, conversely, were implemented with calls to the XPRESS MIP libraries. Except for BBI, all remaining exact solution algorithms considered here make use of the primal DPSTP heuristics to be described next.

3.1. Heuristics

The first heuristic, called MWTREE(c), associates weights $\{c_{ij} : \{i,j\} \in E\}$ to the edges of E and then computes a maximum weight spanning tree of G, according to these edge weights. The number of full degree vertices in such a tree is taken as a lower bound on w. The heuristic is used not only to initialize CBEN and BCD, but also throughout these algorithms. That strategy has two objectives in mind. The first one is to attempt to improve the primal bounds obtained at the root node of the enumaration tree. The second, as we shall see, is to characterize violated inequalities (1) and (2), as well as (13) for the case of BCD. As it will be explained later, that is done by assigning different weights to the edges of G. These weights, in turn, originate from the different LP relaxations at hand. For the initialization of CBEN and BCD, weights $c_{ij} = \frac{1}{|\delta(i)|} + \frac{1}{|\delta(j)|}, \{i,j\} \in E$ are applied.

The second heuristic, STAR(p), is the Greedy Star Insertion Algorithm proposed in Bhatia et al. [1]. The heuristic is used for algorithms BCD, BBU, and CBEN. Using the union-find data structure [20], STAR(p) runs in time $O(m\alpha(m,n))$, where $\alpha(m,n)$ denotes the inverse Ackermann function. It is proven to produce a spanning tree with at least $\frac{w}{2\sqrt{n}+1}$ full degree vertices [1].

STAR(p) is a Kruskal-like algorithm where one attempts to build a forest (V, E_F) by adding stars $\{\delta(i) : i \in V\}$ of G to E_F (initially $E_F = \emptyset$). Instead of selecting one edge at a time, as imposed by Kruskal's algorithm, STAR(p) selects all edges in $\delta(i) \setminus E_F$, for a given chosen $i \in V$, provided none of these edges induce a cycle with previously selected edges. If at least one edge in $\delta(i) \setminus E_F$, say e, is such that $(V, E_F \cup \{e\}) \notin \mathcal{F}$, no edge in $\delta(i) \setminus E_F$ is selected. Then, the edges incident to another (uninvestigated) vertex are considered. When no vertex remains to be investigated, whenever necessary, additional edges are inserted to the forest, to make it a spanning tree of G. In the original implementation in [1], the sequence under which stars are investigated is the lower the degree of a vertex, the sooner it is investigated (ties are broken arbitrarily). Given that our aim is to call the heuristic several times throughout our exact solution algorithms, a priority vector $p \in \mathbb{R}^n$ is used in an attempt to bring dual information into the picture and ultimately defining the order in which vertices are to be investigated. For the first call of the heuristic, the priority vector is such that vertices scanned earlier have smaller degrees. That condition is ensured, for instance, by taking $p_i = \frac{1}{|\delta(i)|}, i \in V$.

3.2. Combinatorial Benders Decomposition Algorithm

Formulation (5) may be used to design a so-called combinatorial Benders decomposition algorithm [3, 10]. In that case, split cuts (1) and cycle cuts (2) would play the role of feasibility cuts. Such an algorithm would explore the fact that the separation problem associated with these inequalities is solvable in polynomial time, for a given $\hat{y} \in \mathbb{B}^n$. One may then attempt to solve DPSTP by solving integer programming relaxations of (5) that include only a tiny portion of feasibility cuts (1) and (2). Remaining inequalities (1) and (2) would be treated as cutting planes, in a Benders-like fashion.

The algorithm we ended up implementing does explore the ideas above. However, it operates under an iterative probing framework (as in a similar approach for solving the minimum connected dominating set problem [7]), that relies on the fact that full degree spanning trees of G interpolate [12], as mentioned in the Introduction. An obvious consequence of that property is the result that follows.

Corollary 6. *Assume, for a given $W \subset V$, that $(\Gamma(W), \delta(W) \cup E(W)) \in \mathcal{F}$ holds and, consequently, that $|W|$ is a valid lower bound on w. Then, if no spanning tree of G exists with $|W|+1$ full degree vertices, no such trees exist with a larger number of full degree vertices.*

Therefore, given a set W such that $(\Gamma(W), \delta(W) \cup E(W)) \in \mathcal{F}$ holds, the idea of CBEN is to iteratively solve a feasibility problem that includes a tiny portion of cuts (1) and (2), together with constraint

$$y(V) = |W| + 1. \tag{16}$$

By Corollary 6, if the feasibility problem is infeasible, $|W|$ must be the maximum number of full degree vertices in a spanning tree of G. However, if $\hat{y} \in \mathbb{B}^n$ is a solution for that problem, let us define $\hat{S} = \{i \in V : \hat{y}_i = 1\}$. If $(\Gamma(\hat{S}), \delta(\hat{S}) \cup E(\hat{S})) \in \mathcal{F}$ holds, the algorithm updates $W \leftarrow \hat{S}$, increases by one unity the right-hand-side of (16) and then solves the resulting feasibility problem. Otherwise, if $(\Gamma(W), \delta(W) \cup E(W)) \notin \mathcal{F}$ applies, the algorithm separates cuts (1) and (2). Violated cuts are added to a new reinforced feasibility problem, where the right-hand-side of constraint (16) is kept fixed. This iterative probing procedure is repeated until the feasibility problem being investigated turns out to be infeasible.

3.2.1. Solving the Benders Subproblem To describe how Benders subproblems are solved, assume that an initial feasible solution for DPSTP is available and let W be the set of full degree vertices in it. Assume as well that the feasibility problem formulated for W, possibly involving only (16), and no feasibility cuts (1) and (2), is solved. Denote by $\hat{y}$ the corresponding solution and define $\hat{S} = \{i \in V : \hat{y}_i = 1\}$.

The Benders subproblem consists of checking whether or not $(\Gamma(\hat{S}), \delta(\hat{S}) \cup E(\hat{S})) \in \mathcal{F}$. For a positive answer, there must exist a tree, (V, E_T), for which $(\delta(\hat{S}) \cup E(\hat{S})) \subseteq E_T$ holds. Consequently, the right-hand-side of constraint (16) should be increased by one unit. Otherwise, as it will be shown next, there must exist a split-cut (1) or a cycle cut (2), violated by $\hat{y}$.

The separation algorithm, in a Kruskal-like style, attempts to insert into a forest $F = (V, E_F)$ of G, initialized as $E_F = \emptyset$, as many edges of $\delta(\hat{S}) \cup E(\hat{S})$ as possible. To do that, it scans every edge $e \in \delta(\hat{S}) \cup E(\hat{S})$, accepting those that do not form a cycle with previously selected edges. The algorithm is based on the result that follows.

Proposition 7. *For a given $\hat{y} \in \mathbb{B}^n$, let $\hat{S} = \{i \in V : \hat{y}_i = 1\}$ and $E_F \subset \delta(\hat{S}) \cup E(\hat{S})$. Then, if $(V, E_F \cup \{e\}) \notin \mathcal{F}$, for $e \in \delta(\hat{S}) \cup E(\hat{S})$, at least one feasibility cut (1)–(2) is violated by $\hat{y}$. Identification of such a cut is performed in polynomial time.*

Proof. Assume that C denotes the set of vertices in the single cycle of $(V, E_F \cup \{e\})$. Two possibilities are there to consider:

- If $C \subseteq \hat{S}$, $\hat{y}(C) = |C|$ and a cycle cut (2) is violated by $\hat{y}$.
- If $C \not\subseteq \hat{S}$, as it will be demonstrated, at least one split cut (1), defined for a subset of the vertices of C, is violated. To that aim, assume that $C = \{v_1, \ldots, v_{|C|}\}$. Assume as well that the unique cycle in $(V, E_F \cup \{e\})$ is given by the following set of edges: $\{\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{|C|-1}, v_{|C|}\}, \{v_1, v_{|C|}\}\}$. As all edges in the cycle belong to $\delta(\hat{S}) \cup E(\hat{S})$, at least one vertex incident to every edge in the cycle must be in $\hat{S}$. Thus, we have that $|\hat{S} \cap C| \geq \frac{|C|}{2}$. Then, one has two cases to analyse:
  - $|\hat{S} \cap C| = \frac{|C|}{2}$.
    Note that in this case, $|C|$ must be even. So partition C into V_1 and V_2 such that $V_1 = \{v_1, v_3, \ldots, v_{|C|-1}\}$ and $V_2 = \{v_2, v_4, \ldots, v_{|C|}\}$ (i.e., V_1 and V_2 get alternate vertices in the cycle). Thus, either $V_1 = \hat{S} \cap C$ or $V_2 = \hat{S} \cap C$. Assuming that the former holds true, $\hat{y}(V_1) = |\hat{S} \cap C| = |V_1|$ and a split cut (1) is violated.
  - $|\hat{S} \cap C| > \frac{|C|}{2}$.
    As every edge in the cycle must be incident to at least one vertex in $\hat{S}$, the cycle has at least one path with at least two consecutive vertices in $\hat{S}$. Assume that consecutive vertices in $C \cap \hat{S}$ are merged into super-nodes. Assume as well that, after the merging operation, cycle C is now represented in terms of a cycle that has vertices $\bar{C} = \{\bar{v}_1, \bar{v}_2, \ldots, \bar{v}_{|\bar{C}|}\}$. Note that, under this notation, some vertices of $\bar{C}$ represent single vertices of C, while others represent two or more consecutive vertices of $C \cap \hat{S}$. Observe that, whenever one super-node $\bar{v}_i \in \bar{C}$ includes (at least) two nodes of C, say v_i, v_j, we can only have $\hat{y}_{v_i} = \hat{y}_{v_j} = 1$ (we do not merge vertices with different values of $\hat{y}$'s into the same super-node). Note as well that after the merging operation, we cannot have $|\bar{C}|$ odd. That applies as, otherwise, we would either have two consecutive vertices $v_i, v_j \in C \cap \hat{S}$ that were not merged into the same super-node or, else, we would have two consecutive vertices $v_i, v_j \in C \setminus \hat{S}$.
    Thus, as in the case above, we can partition $\bar{C} = \bar{V}_1 \cup \bar{V}_2$, such that $\bar{V}_1 = \{\bar{v}_1, \ldots, \bar{v}_{|\bar{C}|-1}\}$ and $\bar{V}_2 = \{\bar{v}_2, \ldots, \bar{v}_{|\bar{C}|}\}$. Define V_1 as the vertices of C that belong to the super-nodes in $\bar{V}_1$ (V_2 is defined similarlly). Then, either $\hat{y}(V_1) = |\hat{S} \cap C|$ and $\hat{y}(V_2) = 0$ or else $\hat{y}(V_1) = 0$ and $\hat{y}(V_2) = |\hat{S} \cap C|$. Assuming that the former holds true, $\hat{y}(V_1) = |\hat{S} \cap C| = |V_1|$ and a split cut is violated.

∎

Strategies used for adding and lifting split and cycle cuts are the following. Whenever $C \subseteq \hat{S}$ holds, we investigate if the right-hand-side of (2) could be easily lifted from $|C| - 2$, and add the resulting cut into a new, stronger, feasibility problem. For the lifting, we check if $(C, E(C))$ defines a clique of G. For a negative answer, we simply verify if $|E(C)| \geq |C| + 1$. For a positive one, the right-hand-side of (2) is set to $|C| - 3$.

The following algorithm summarizes the Benders decomposition algorithm:

- (Initialization) Run heuristics MWTREE(c) and STAR(p). Let $T^* = (V, E_{T^*})$ denote the best solution obtained by these two algorithms and let W be the set of full degree vertices in T^*. Take constraint (16), written in terms of W, and proceed as follows. Append to the first feasibility problem, cycle cuts (2), defined by the set of vertices in the unique cycle in $(V, E_{T^*} \cup \{e\})$, implied by every $e \in E \setminus E_{T^*}$.
- Repeat until a convergence criterion is met.
  1. Solve the feasibility problem at hand.
     (a) If it is infeasible, stop. The best known solution T^* solves DPSTP.
     (b) Otherwise, let $\hat{y} \in \mathbb{B}^n$ be the solution obtained for that problem and set $\hat{S} := \{i \in V : \hat{y}_i = 1\}$:
        i. If subgraph $(\Gamma(\hat{S}), \delta(\hat{S}) \cup E(\hat{S}))$ is not cycle free, there exists at least one feasibility cut violated by $\hat{y}$. Add that cut to a new feasibility problem.
        ii. After solving the Benders subproblem, run heuristic STAR(p), with vector p now given by $p_i = \frac{1}{(1.1 - \hat{y}_i)|\delta(i)|}, i \in V$. If the solution thus obtained improves on the best feasible solution so far obtained, update T^* and the right-hand-side of (16), accordingly.
        iii. Otherwise, that is, if $(\Gamma(\hat{S}), \delta(\hat{S}) \cup E(\hat{S}))$ is cycle free, update T^* as any spanning tree of G such that every vertex in $\hat{S}$ has full degree in T^*. Update as well the right-hand-side of (16).

Notice that the distance between the initial lower bound and the optimal number of full degree vertices is finite. Notice as well that the number of feasibility cuts (1) and (2) is equally finite. Therefore, the iterative probing Benders decomposition algorithm terminates.

In the case $(\Gamma(\hat{S}), \delta(\hat{S}) \cup E(\hat{S})) \notin \mathcal{F}$, it suffices to add only one cut $y(\hat{S}) \leq |\hat{S}| - 1$, or a stronger version of it, say $y(\hat{S}) \leq \alpha(\hat{S})$, to the new Benders feasibility problem. However, several cuts are added at each iteration, one for each cycle $(V, E_F \cup \{e\})$ identified when checking for the feasibility of set $\hat{S}$.

3.3. BB Algorithm Based on the Undirected Formulation

BBU is defined in terms of a list $\mathcal{L}$ of nodes/subproblems where, for each subproblem, an LP similar to (9) must be solved. In doing that, upper bounds are generated for DPSTP.

For the root node, all vertices are free to be either of incomplete or full degree. The algorithm then computes a maximum weight spanning tree of G, as implied by (9), through, for instance, Kruskal's algorithm. If the number of full degree vertices in the best spanning tree so far obtained is at least $\lfloor w(P_u) \rfloor$, the algorithm stops as that tree solves DPSTP. Such a tree might even be the one corresponding to (9). Otherwise, if the condition is not met, the algorithm branches and adds new subproblems to $\mathcal{L}$.

Every subproblem tackled by BBU is defined in terms of two subsets of vertices of V, namely S_0 and S_1; S_0 corresponding to vertices that are set to be of incomplete degree; S_1 corresponding to those set to be of full degree. At the root node of the enumeration tree, $S_0 = S_1 = \emptyset$. Additionally, M is taken as any valid upper bound on w, say $M = n$, for example.

The LP to be solved at each subproblem is then given by

$$\bar{w}(S_0, S_1) = -M|S_1| + \max \left\{ (M+1) \sum_{i \in S_1} y_i + \sum_{i \in V \setminus (S_0 \cup S_1)} y_i : (z, y) \in P_u \right\}, \tag{17}$$

where S_0 and S_1 are the subsets of vertices defining the subproblem and

$$y_i = \frac{1}{|\delta(i)|} z(\delta(i)), \quad i \in V \setminus S_0. \tag{18}$$

Notice that constraints (8) are not forced to be tight for the vertices in S_0. Therefore, at least one edge incident to these vertices must not be in the solution to (17). Then, for the sake of reformulating the objective function of (17) in terms of a weighted function of the edges of E, degree enforcing constraints are only imposed to be tight for vertices in $V \setminus S_0$. Notice as well that the contribution of $\{y_i : i \in S_1\}$ to $\bar{w}(S_0, S_1)$ is precisely $|S_1|$, as long as the solution to (17) is such that all vertices in S_0 (respectively, S_1) are of incomplete (respectively, full) degree. Now, substituting the right-hand-side of (18) for their corresponding y entries in (17),

$$\bar{w}(S_0, S_1) = -M|S_1| + \max \left\{ \sum_{\{i,j\} \in E} l_{ij} z_{ij} : z \in P_{\text{STP}} \right\} \tag{19}$$

is obtained, where the weights l_{ij} depend on the subsets, S_0 or S_1, i and j belong to. These weights are then given by:

$$l_{ij} = 0, \text{ if } i, j \in S_0, \tag{20a}$$

$$l_{ij} = \frac{M+1}{|\delta(i)|}, \text{ if } i \in S_1 \text{ and } j \in S_0, \tag{20b}$$

$$l_{ij} = \frac{1}{|\delta(i)|}, \text{ if } i \in V \setminus (S_0 \cup S_1) \text{ and } j \in S_0, \tag{20c}$$

$$l_{ij} = \frac{1}{|\delta(i)|} + \frac{M+1}{|\delta(j)|}, \text{ if } i \in V \setminus (S_0 \cup S_1) \text{ and } j \in S_1, \tag{20d}$$

$$l_{ij} = \frac{1}{|\delta(i)|} + \frac{1}{|\delta(j)|}, \text{ if } i, j \in V \setminus (S_0 \cup S_1), \tag{20e}$$

$$l_{ij} = \frac{M+1}{|\delta(i)|} + \frac{M+1}{|\delta(j)|}, \text{ if } i \in S_1 \text{ and } j \in S_1. \tag{20f}$$

To solve (19), one would first check if $(\Gamma(S_1), \delta(S_1) \cup E(S_1)) \in \mathcal{F}$ applies. If the subgraph is not cycle free, the corresponding BB node would be pruned by infeasibility. However, the procedure that is actually used to solve (19) relies on Kruskal's algorithm, to find a maximum weight spanning tree of G, under the weights $\{l_{ij} : \{i,j\} \in E\}$ defined above. Let $T = (V, E_T)$ be the tree thus obtained. If $|\delta_T(i)| < |\delta(i)|$, for any $i \in S_1$, the subproblem is infeasible. Conversely, if $|\delta_T(i)| = |\delta(i)|$, for any $i \in S_0$, the subproblem is pruned by optimality. The validity of such observations results from the fact that M is a valid upper bound on w. If $|\delta_T(i)| < |\delta(i)|$ and $|\delta_T(j)| = |\delta(j)|$ hold, for any $i \in S_0$ and for every $j \in S_1$, the LP relaxation for that node is given by $\bar{w}(S_0, S_1) = \sum_{\{i,j\} \in E_T} l_{ij} - M|S_1|$. The algorithm then counts the number of full degree vertices in T, to eventually update the best lower bound w^* at hand. If $\lfloor \bar{w}(S_0, S_1) \rfloor \leq w^*$ holds, the node is pruned by optimality. Otherwise, branching on vertex variables y is carried out as follows. Define $j = \arg\max\{|\delta(i)| : i \in V \setminus (S_0 \cup S_1)\}$ (ties are broken arbitrarily). Two new subproblems, $(S_0, S_1 \cup \{j\})$ and $(S_0 \cup \{j\}, S_1)$, are then added to the list.

The algorithm implements a depth-first search enumeration where subproblem $(S_0, S_1 \cup \{j\})$ is always investigated before $(S_0 \cup \{j\}, S_1)$. Whenever $\mathcal{L} = \emptyset$, the algorithm stops. The spanning tree with the largest number of full degree vertices identified by it, solves DPSTP.

3.4. BC Algorithm based on the Strengthened Cutset Formulation

Algorithm BCD first solves relaxation $\max\{y(V) : (x^r, y) \in \bar{P}\}$, where $\bar{P}$ is implied by (11a), (11b), (11d)–(11f), and (12). If the solution thus obtained, $(\bar{x}^r, \bar{y})$, is integer and corresponds to an arborescence of D, it implies an optimal solution to DPSTP. Otherwise, we look for violated inequalities (11c), (1), (2), and (13) to reinforce $\bar{P}$.

Cutset inequalities (11c) are separated in $O(n^4)$ time, as follows. Assume that $\bar{D} = (V, \bar{A})$, for $\bar{A} = \{(i,j) \in A : \bar{x}^r_{ij} > 0\}$, is the support graph associated with $(\bar{x}^r, \bar{y})$, that is, the LP relaxation solution at hand. For each $i \in V \setminus \{r\}$, compute a minimum cut $A(V \setminus S, S)$, where $r \in V \setminus S$ and $i \in S$, separating r and i in the network defined by $\bar{D}$ and arc capacities $\{\bar{x}^r_{ij} : (i,j) \in \bar{A}\}$. If $\bar{x}^r(A(V \setminus S, S)) < 1$, a cutset (11c) is violated and is appended to the relaxation.

For the separation of (1), (2), and (13), we first define edge weights $c_{ij} = \bar{x}^r_{ij} + \bar{x}^r_{ji}, \{i,j\} \in E$ and run heuristic MWTREE(c) for them. Assume that a spanning tree $T_1 = (V, E_{T_1})$ is thus obtained. T_1 not only provides a valid lower bound on w. It is also used in an attempt to identify candidate sets of vertices implying violated inequalities (1), (2), and (13).

More precisely, for every edge $e \in E \setminus E_{T_1}$, we identify the set of vertices $C_{T_1,e}$, that define the unique cycle of subgraph $(V, E_{T_1} \cup \{e\})$. After checking if the right hand side of (2) could be easily lifted (as described in Section 2.1), we append the cut to a new, strengthened, LP relaxation of DPTSP, in case of violation. For every set $C_{T_1,e}$, we also attempt to reinforce the relaxation by identifying violated inequalities (13). That is conducted by checking, for all possible vertices $i \in C_{T_1,e}$, if the cut (13) being implied is violated.

Given $C_{T_1,e}$, violated split cuts (1) are attempted to be identified, as follows. Let $C_{T_1,e} = \{v_1, \ldots, v_{|C_{T_1,e}|}\}$. If $|C_{T_1}, e|$ is even, one sets $V_1 = \{v_1, v_3, \ldots, v_{|C_{T_1,e}|-1}\}$ and $V_2 = \{v_2, v_4, \ldots, v_{|C_{T_1,e}|}\}$ and checks if inequality (1), written for V_1 or V_2, is violated. If $|C_{T_1}, e|$ is odd, one defines V' as in the proof of Proposition 7, that is, adds to V' any two consecutive vertices in $C_{T_1,e}$, say, v_1 and v_2, and complements V' with other alternate vertices in $C_{T_1,e}$, that is, $v_4, \ldots, v_{|C_{T_1,e}|-1}$. One then checks violation of inequalities (1), for every possible pair of consecutive cycle vertices, v_1 and v_2.

Given $(\bar{x}^r, \bar{y})$, we also run heuristic STAR(p), after taking $p_i = \frac{1}{(1.1-\bar{y}_i)|\delta(i)|}$, $i \in V$ as the input priorities. Assuming that a spanning tree $T_2 = (V, E_{T_2})$ is returned by the procedure, one then proceeds, as previously suggested for T_1. Namely, one attempts to identify additional inequalities (1), (2), and (13) that are violated. As before, that is carried out by investigating the cycles of G resulting from adding to T_2 one edge of $E \setminus E_{T_2}$ a time.

In conclusion, for every LP relaxation solution $(\bar{x}^r, \bar{y})$, found at any enumeration tree node, we separate (11c), find T_1 and T_2 and proceed with the separation of (1), (2), and (13), as described above. The separation of valid inequalities is carried out until no more violated cuts are found. Then, if the LP relaxation solution is not integral valued, BCD branches on variables.

In our implementation, preference is given to branching first on x^r and then on y. We experimented with other branching strategies, namely: (1) branching first on y and then on x^r, (2) branching first on y and then on the special ordered sets implied by (11b), and (3) branching only on y. For the latter strategy, we did not prove that $y \in \mathbb{B}^n$ implies $x^r \in \mathbb{B}^{2m}$. Nevertheless, for the instances in our test bed, we never had to branch on x^r if preference was given to branching first on y. The other three branching strategies led to worse CPU times, if compared to the strategy of branching first on x^r.

BCD implements a *best-first* search. Apart from the heuristics, cut generation and preprocessing procedures that were turned off, all other default XPRESS settings were used.

4. COMPUTATIONAL RESULTS

Instances in our computational experiments have $n \in [25, 175]$ and different ranges for the graph density, for different values of n. Some instances considered in the current study were introduced in [6]. Additional ones are introduced here according to the procedure used in [6], described next.

For a given size n, we generated DPSTP instances without cutnodes nor bridges as follows. We first include in E edges in an Hamiltonian cycle of G: $\{\{1,2\}, \{2,3\}, \ldots, \{n-1, n\}, \{1, n\}\}$. For the other pairs $i, j \in V$, we include the corresponding edge i, j into E, according to a probability, defined by a target graph density. Depending on the size of the instances, the target density and the observed density (defined as $100\frac{2m}{n(n-1)}$) may differ significantly. That applies especially for instances with small values of n, when the maximum amount ($\frac{n(n-1)}{2} - n$) of edges to be randomly picked (after the Hamiltonian cycle edges were included in E) is small. When that is the case, just one or two additional edges that are included (or not) in E represent a large proportion of the maximum number $\frac{n(n-1)}{2}$ of edges in a complete graph. To illustrate, take instance 25_8. For that instance, $n = 25$ and the target density used was 25%, while the measured density is 27%.

In total, 42 instances were tested here. For $n \in \{30, 50, 75, 100, 125, 150\}$, we generated four instances with graph densities in the range from 6 to 30%, with a nonregular pattern of graph density. For $n = 25$ and $n = 175$, a more regular pattern of graph density was considered. For $n = 25$, we generated two instances for each target density in the set $\{10\%, 15\%, 20\%, 25\%\}$. For $n = 175$, we also generated two instances for a target density of 5%. Instances with $n = 25$ and target graph density of 5% were not considered here as they include, in addition to the Hamiltonian cycle edges above, only one or two more edges, being thus very easy to solve. Our aim in including small ($n = 25$) and larger ($n = 175$) instances with a more regular pattern for the target density is to validate our empirical observation that the scale of increase or decrease in difficulty for solving the instances varies with different values of n.

All algorithms were implemented in C,C++ and compiled with g++. All computational results reported here were obtained with an Intel XEON E5645Core TMi7-980 hexacore machine, running at 2.4 GHz, with 24 GB of shared RAM memory. All algorithms were executed with only one core; no multithreading was allowed. XPRESS MIP package release 23.01.06 was used to implement CBEN, BCD, and BBI.

4.1. LP Upper Bounds

In Tables 1 and 2, we present LP upper bounds for each formulation considered here, respectively for $25 \leq n \leq 75$ and $100 \leq n \leq 175$. The first two columns of the tables provide the instance name, followed by its density, defined as $100\frac{2m}{n(n-1)}$. The next two columns, respectively, indicate bounds $w(P_u)$ and $w(P_d^r)$. The next column stands for $w(P_d^{r+})$, the bound implied by P_d^r when strengthened by inequalities (1), (2), and (13). For the computation of that upper bound, the same separation heuristics for inequalities (1), (2), and (13) used by BCD were considered. The next two LP bounds are $w(P_D^r)$ and its strengthened version $w(P_D^{r+})$. Similarly, $w(P_D^{r+})$ stands for the bound $w(P_D^r)$ after the addition of those inequalities (1), (2), and (13) found violated by our separation heuristics. The last upper bound in the tables is the one provided by the reformulation by intersection. Finally,

TABLE 1. Linear Programming upper bounds and lower bounds for DPSTP, $25 \leq n \leq 75$

		Linear programming upper bounds							
Instance	d (%)	$w(P_u)$	$w(P_d^r)$	$w(P_d^{r+})$	$w(P_D^r)$	$w(P_D^{r+})$	$w(P_I)$	OPT	STAR(p)
25_1	10.3	20.750	19.375	19.306	18.000	18.000	18.000	17	17
25_2	11.7	19.367	17.508	17.508	16.333	16.333	16.333	15	15
25_3	12.0	17.667	16.667	16.667	14.000	14.000	14.000	12	12
25_4	14.0	16.317	14.333	14.333	12.403	12.403	12.283	11	11
25_5	17.7	13.386	11.368	11.368	9.243	9.243	9.243	8	8
25_6	18.3	12.969	10.960	10.960	8.775	8.775	8.775	8	8
25_7	25.7	9.862	8.012	8.012	6.169	6.169	6.163	5	4
25_8	27.0	8.890	7.407	7.407	5.457	5.453	5.346	4	4
30_3	8.5	25.333	23.667	23.128	22.000	22.000	22.000	22	21
30_2	10.3	21.926	19.405	19.162	17.000	17.000	17.000	17	16
30_1	10.6	21.600	19.477	19.477	17.423	17.423	17.423	17	15
30_4	27.8	8.746	7.195	7.195	5.309	5.285	5.285	4	4
50_2	10.1	24.617	20.074	20.074	16.646	16.646	16.646	15	14
50_3	10.4	23.601	19.451	19.432	16.162	16.162	16.161	16	13
50_1	29.6	8.547	6.776	6.772	4.789	4.789	4.745	3	3
50_4	32.6	7.517	6.140	6.140	4.026	4.026	3.996	2	2
75_2	10.5	24.413	19.592	19.592	15.566	15.566	15.563	13	13
75_1	10.9	22.468	18.597	18.597	13.814	13.814	13.734	11	11
75_3	19.4	12.972	10.335	10.335	7.142	7.142	7.126	4	3
75_4	29.4	8.382	6.793	6.793	4.510	4.510	—	2	2

Any LP upper bound entry "—" indicates that the bound could not be evaluated within 1 CPU hour.

TABLE 2. Linear programming upper bounds and lower bounds for DPSTP, $100 \leq n \leq 175$

		Linear Programming upper bounds							
Instance	d (%)	$w(P_u)$	$w(P_d^r)$	$w(P_d^{r+})$	$w(P_D^r)$	$w(P_D^{r+})$	$w(P_I)$	OPT	STAR(p)
100_4	6.0	41.479	33.915	33.915	27.529	27.529	27.502	25	22
100_2	9.5	25.828	21.019	21.019	14.919	14.919	14.919	11	9
100_1	10.1	26.071	20.395	20.395	15.679	15.679	15.663	12	11
100_3	10.6	24.484	19.195	19.195	14.345	14.345	14.326	11	9
125_1	6.9	37.316	29.887	29.887	22.503	22.503	—	(?) 18	17
125_2	7.6	32.987	26.479	29.479	19.381	19.381	—	(?) 15	13
125_3	10.5	24.474	19.195	19.195	13.897	13.897	—	10	8
125_4	19.5	12.562	10.258	10.258	6.786	6.786	—	4	4
150_1	7.2	35.903	28.291	28.291	21.170	21.170	—	(?) 16	15
150_2	7.6	34.241	27.049	27.049	20.460	20.460	—	(?) 16	15
150_3	10.1	24.458	19.622	19.622	13.715	13.715	—	9	7
150_4	19.3	12.310	10.338	10.228	6.500	6.500	—	3	2
175_1	4.8	54.363	41.952	41.952	33.086	33.086	—	(?) 27	25
175_2	5.1	50.951	40.146	40.146	31.610	31.610	—	(?) 25	23
175_3	10.0	24.822	20.007	20.007	13.612	13.612	—	8	6
175_4	10.1	24.572	19.894	19.883	13.546	13.546	—	8	7
175_5	14.7	16.482	13.612	13.612	8.773	8.773	—	5	3
175_6	14.9	16.652	13.402	13.402	8.887	8.887	—	4	4
175_7	19.1	12.422	10.453	10.453	6.485	6.485	—	3	2
175_8	19.7	12.071	10.147	10.147	6.334	6.334	—	2	2
175_9	25.0	9.311	8.003	8.003	4.792	4.792	—	1	1
175_10	25.3	9.266	7.915	7.915	4.802	4.802	—	1	1

Any LP upper bound entry "—" indicates that the bound could not be evaluated within1 CPU hour. Whenever the optimal objective function is unknown, the corresponding OPT entry is given together with a (?) indication.

the last two columns provide the optimal objective function value (OPT) and the lower bound provided by STAR(p), the approximation algorithm in [1]. Whenever the optimal solution is unknown, the best known lower bound is indicated also in columns under headings OPT. In such cases, the lower bound is presented with an accompanying symbol "?." Whenever an upper bound could not be evaluated within a time limit of 3600 CPU seconds, an indication "-" is provided in the corresponding column. For all directed formulations considered here, we have chosen $r = 1$ as the root arborescence.

Each instance is indicated by a word n_id, where n denotes the size of the instance and id is used to characterize that particular instance among the others having the same n. For example, instance 175_1 is an instance with $n = 175$ whose id is 1. Instances in all tables are presented in an increasing order of n. For a given fixed n, instances are sorted in a nondecreasing order of graph density. To allow comparisons with results in [6], we keep the ids used in [6]. Therefore, for some values of n, the tables do not present the instances in increasing order of their id's.

As one could expect, the evaluation of $w(P_I)$ is very time consuming when n and the graph density increase. For many cases, such bounds could not be computed within the imposed time limit. For the remaining instances, P_I provided the strongest upper bounds among all formulations tested here. Despite the fact that many violated constraints (1), (2), and (13) have been found violated in the course of the cutting plane algorithm, the bounds given by P_D^{r+} and P_D^r do not change except for instances 25_8 and 30_4. Nevertheless, the BCD algorithm that separates these constraints outperforms its counterpart that does not. Degree enforcing constraints significantly improved on bounds $w(P_d^r)$. Among all models considered here, P_u is the weakest. Considering only those instances whose optimal solutions are known, STAR(p) provided lower bounds that are not more than three units away from optimal solution values.

Finally, one important observation regarding the upper bounds discussed here is the following. If one considers the instances for which bounds $w(P_I)$ could be evaluated, quite often, formulations P_D^r, P_D^{r+}, and P_I essentially provided the same upper bounds in practice. As the objective function involves only integer coefficients, the floor of the bounds provided by these formulations are also valid DPSTP upper bounds. Thus, quite often, the bounds provided by formulations P_D^r, P_D^{r+} and P_I become the same when fractional bounds $w(P_D^r), w(P_D^{r+})$, and $w(P_I)$ are rounded down.

4.2. *Comparison of Exact Algorithms*

Detailed computational results for BBI, CBEN, BBU, and BCD are given in Tables 3 and 4, respectively for $25 \leq n \leq 75$ and $100 \leq n \leq 175$. For BBI, BBU, and BCD, we indicate, for each instance, the best lower (BLB) and upper (BUB) bounds and the CPU time ($t(s)$, in seconds) to obtain them. For CBEN, we provide the best lower bound and an indication (proven ?) about whether (yes) or not (no) BLB was proven to be the optimal objective value. The next columns indicate the CPU time, the total number of feasibility problems (iter) that were solved and the total number of feasibility cuts (cuts) added along the search. For BBU and BCD, we provide the number of nodes investigated in the BB tree.

As it could be appreciated from results in the tables, BBI is capable of solving only instances with $n \leq 50$. For larger instances, it never managed to find a feasible solution for the problem. Although BBI is based on the strongest DPTSP formulation, poor computational results were obtained mainly because of the large number of constraints and variables [$O(nm)$] explicitly handled by the algorithm.

BBU provided the highest success rate among all algorithms, being able to solve 35 out of 42 instances, within the imposed time limit. Except for instances 25_3, 30_1, 30_2, 50_2, 50_3, and 100_4, BBU also outperformed the other algorithms in terms of CPU times, despite the fact that it is based on the weakest DPSTP upper bounds considered here. The main reasons for that seem to be the availability of good initial DPSTP lower bounds, a very fast lower bounding procedure, combined with effective branching and search policies. The fact that BBU branches first on vertices with high degrees and investigates first the $y_i = 1$ branch was of crucial importance to reduce the depth of the search tree, as infeasibility could be early detected for many branches in the tree. We tested another BBU implementation, for which higher branching priorities were given to vertices with the smallest degrees in G. Such an implementation solved only 27 out of the 35 instances originally solved by BBU. Among the BBU unsolved instances, no additional instance was solved by the second implementation. Conversely, considering only those 27 instances that were solved by both, a 95% CPU time increase was observed when the branching policy was changed. For the instances that were not solved by both, the algorithm that branches first on low degree vertices improves on the initial lower bounds for more cases. That implementation provided the lower bounds of 24, 16, 16, and 24, respectively for instances 100_4, 150_1, 150_2, and 175_2, while BBU did not improve the initial upper bounds of 22, 14, 13, and 23 for the same instances.

Measured by the number of instances solved to proven optimality, CBEN was the second best algorithm in our study. It solved 29 out of 42 instances. For instances that BBU and CBEN could solve within the imposed time limit, CBEN CPU times were always larger than BBU counterparts. Sometimes, CBEN took two orders of magnitude more running time than BBU. Nevertheless, computational results lean in favor of CBEN for those instances that are left unsolved by both. At the end of the time limit, CBEN improved on the initial heuristic lower bounds for instances 100_4, 150_1, 150_2, and 175_2, while the same did not happen for BBU.

BCD solved only 20 of the instances to optimality, the second lowest success rate in this study. Nevertheless, compared to CBEN, BBU, and BBI, it was the only algorithm capable of solving instance 100_4. Although, BCD CPU times were larger than BBU counterparts for most of the instances, in six cases (25_3, 30_1, 30_2, 50_2, 50_3, and 100_4), BCD was the fastest algorithm. Compared to CBEN, there is also no dominance. For some instances it is faster, while for others, it is much slower. For four instances that were not solved by any algorithm in this study, 125_1, 125_2, 175_1, and 175_2, the best known upper bounds were provided by BCD.

As a general rule, dense instances seem to be harder to BCD, as the CPU times needed to evaluate the LP relaxations of the directed model tend to be larger. For CBEN and BBU, it seems that the opposite holds. To validate such a claim, observe that for a given n, sparse instances usually demand

TABLE 3. Comparison of DPSTP exact solution approaches, $25 \leq n \leq 75$

Instance		BBI			CBEN					BBU				BCD			
		P_I			P_C					P_u				P_D^{r+}			
Name	Den (%)	BLB	BUB	$t(s)$	BLB	Proven ?	$t(s)$	iter	Cuts	BLB	BUB	$t(s)$	Nodes	BLB	BUB	$t(s)$	Nodes
25_1	10.3	17	17	0.44	17	Yes	0.59	63	190	17	17	0.00	1811	17	17	0.03	2
25_2	11.7	15	15	0.81	15	Yes	0.13	20	53	15	15	0.00	801	15	15	0.04	2
25_3	12.0	12	12	31.39	12	Yes	6.49	161	426	12	12	0.15	24607	12	12	0.07	51
25_4	14.0	11	11	1.47	11	Yes	0.46	10	167	11	11	0.03	3451	11	11	0.06	4
25_5	17.7	8	8	1,72	8	Yes	0.31	32	163	8	8	0.01	1899	8	8	0.08	2
25_6	18.3	8	8	3.20	8	Yes	0.38	38	211	8	8	0.01	1399	8	8	0.04	1
25_7	25.7	5	5	1.73	5	Yes	0.22	30	198	5	5	0.00	847	5	5	0.11	5
25_8	27.0	4	4	345.85	4	Yes	0.24	28	235	4	4	0.00	867	4	4	0.23	5
30_3	8.5	22	22	0.38	22	Yes	0.14	15	55	22	22	0.00	587	22	2	0.03	1
30_2	10.3	17	17	0.90	17	Yes	0.62	50	199	17	17	0.07	9327	17	17	0.02	1
30_1	10.6	17	17	1.28	17	Yes	0.17	14	70	17	17	0.06	9331	17	17	0.03	1
30_4	27.8	4	4	106.06	4	Yes	0.48	48	477	4	4	0.02	1065	4	4	0.29	5
50_2	10.1	15	15	2183.80	15	Yes	34.27	136	1140	15	15	15.28	896530	15	15	0.30	2
50_3	10.4	16	16	23.51	16	Yes	8.08	101	980	16	16	6.36	352207	16	16	0.08	1
50_1	29.6	—	4	—	3	Yes	6.80	166	2797	3	3	0.08	2695	3	3	18.07	99
50_4	32.6	—	3	—	2	Yes	12.72	357	4108	2	2	0.08	2377	2	2	26.94	295
75_2	10.5	—	14	—	13	Yes	95.58	167	2809	13	13	38.08	1183469	13	13	43.81	351
75_1	10.9	—	12	—	11	Yes	199.80	224	3430	11	11	55.99	1741013	11	11	246.90	1526
75_3	19.4	—	7	—	4	Yes	39.38	346	5024	4	4	0.86	18343	4	5	—	1829
75_4	29.4	—	—	—	2	Yes	91.34	770	11948	2	2	0.32	5469	2	2	754.26	937

TABLE 4. Comparison of DPSTP exact solution approaches, $100 \leq n \leq 175$

Instance		BBI			CBEN					BBU				BCD			
		P_I			P_C					P_u				P_D^{r+}			
Name	Den (%)	BLB	BUB	$t(s)$	BLB	Proven ?	$t(s)$	iter	Cuts	BLB	BUB	$t(s)$	Nodes	BLB	BUB	$t(s)$	Nodes
100_4	6.0	—	25	—	25	No	—	557	8944	22	41	—	88773337	25	25	30.96	373
100_2	9.5	—	14	—	11	No	—	795	9763	11	11	633.31	13183851	11	12	—	6576
100_1	10.1	—	15	—	12	Yes	1360.64	462	7432	12	12	273.46	5587181	12	14	—	1739
100_3	10.6	—	14	—	11	Yes	623.90	388	6538	11	11	170.34	3397119	11	12	—	2884
125_1	6.9	—	—	—	17	No	—	1327	17831	17	37	—	58231725	18	20	—	1601
125_2	7.6	—	—	—	14	No	—	1354	16780	14	32	—	56657444	15	19	—	4546
125_3	10.5	—	—	—	10	Yes	1296.96	540	10637	10	10	301.62	4113585	10	12	—	1904
125_4	19.5	—	—	—	4	Yes	249.21	647	21707	4	4	2.52	21951	4	5	—	576
150_1	7.2	—	—	—	16	No	—	1039	19459	15	35	—	42068826	16	19	—	671
150_2	7.6	—	—	—	16	No	—	928	18560	15	33	—	41117303	16	18	—	1252
150_3	10.1	—	—	—	9	Yes	1499.14	999	18222	9	9	619.46	6282171	8	12	—	823
150_4	19.3	—	—	—	3	Yes	947.61	1727	42730	3	3	4.3	28379	3	5	—	433
175_1	4.8	—	—	—	25	No	—	1045	22661	25	54	—	37525072	27	31	—	564
175_2	5.1	—	—	—	24	No	—	1036	23160	23	50	—	37246265	25	30	—	680
175_3	10.0	—	—	—	8	No	—	1094	25047	8	8	697.8	5493395	8	12	—	131
175_4	10.1	—	—	—	8	No	—	1223	24974	8	8	601.1	4662623	8	12	—	463
175_5	14.7	—	—	—	5	Yes	887.18	1206	35581	5	5	19.4	120987	5	7	—	290
175_6	14.9	—	—	—	4	Yes	1256.91	1601	40802	4	4	17.7	109985	4	8	—	143
175_7	19.1	—	—	—	3	Yes	1989.23	2568	67454	3	3	6.76	34567	3	5	—	111
175_8	19.7	—	—	—	2	No	—	4293	71997	2	2	6.85	34443	2	5	—	139
175_9	25.0	—	—	—	1	No	—	4760	48157	1	1	7.1	30449	1	4	—	133
175_10	25.3	—	—	—	1	No	—	5739	57263	1	1	7.1	30449	1	4	—	61

several times more BBU nodes than denser instances to be solved, which can be explained by the fact that, under the branching rule we implemented, BBU detects infeasibility very early in the search tree. Although the picture is less clear in the iterative probing Benders case, it also seems that CBEN works better for dense instances. That seems to apply because the difference between optimal solution values and the initial lower bounds provided by STAR(p) is usually smaller for these instances. Being so, the number of updates in the right-hand-side of the feasibility constraint (16) is smaller.

Our computational results suggest that the same graph density may lead to different levels of difficulty for solving instances with different values of n. In other words, an instance of density d may be considered hard to solve for a given value of n (compared to instances having the same size n and other values of density) but relatively easy for $\bar{n} >> n$. To substantiate such observation, take instances with $n = 25$ and $n = 175$ as examples. BBU solves all $n = 25$ instances in more or less the same magnitude of CPU time, including those with density close to 25%. However, the same does not apply for $n = 175$. Instances with $n = 175$ and 25% density are solved in about 7 seconds, while $n = 175$ instances with 10% density are not solved in 1 hour. In summary, the scale of relative difficulty for the density is not the same for different values of n.

4.2.1. Hybrid Algorithms Motivated by the good computational results obtained by BBU, we implemented two variants of CBEN and BCD, that make calls to BBU to obtain the optimal cuts (3). These hybrid algorithms are denoted HCBEN and HBCD, respectively for the Benders and the BCD variants. Whenever a given set V' is such that $(\Gamma(V'), \delta(V') \cup E(V')) \notin \mathcal{F}$, we call BBU to define $\alpha(V')$, for the feasibility cut implied by V'. In the Benders case, these sets are found when solving the Benders subproblem. For the BCD, these sets are the fundamental cycles of the spanning trees T_1 and T_2 devised by the separation heuristics outlined in Section 3.4. Before checking for violation, the optimal right-hand-side is computed and only then the cut is appended to the model.

Computational results obtained by HCBEN and HBCD are given in Table 5, under the same time limit of 3600 seconds. Apart from $t_l(s)$, all other entries in the table have the same meaning as in Tables 3 and 4. In Table 5, $t_l(s)$ stands for the total amount of time, in seconds, taken to compute the optimal right-hand-side in the feasibility cuts, by calling BBU within each algorithm, HCBEN and HBCD.

HBCD managed to solve 75_3 and 125_4 that were not solved by BCD. In general, HBCD outperforms BCD for denser instances in our test bed. Take instance 75_4 as an example. While BCD needed 937 nodes and about 754 seconds to solve that instance, HBCD needed only 149 nodes and 99 seconds to do the same. In a certain way, that goes along with what one would expect. For dense instances, values of α should be significantly smaller than those values in the right-hand-side of (1) and (2) and those values obtained by simple liftings we described before. In addition, dense instances are the hardest for BCD and the easiest for BBU. All together, the time taken to lift the cuts, as compared to overall HBCD time, tend to be small and, thus, the computation of the optimal lifting had a positive impact on the overall CPU time. For sparse instances, the effect of the hybridization is less important, as these are the hardest instances for BBU and because lifted coefficients are likely to improve less. Indeed, more often BCD dominates HBCD for sparse instances. For instance 100_4, for example, both BCD and HBCD explore more or less the same number of nodes in the search tree. However, HBCD takes about 35 additional seconds for solving the problem, corresponding to the total time spent in attempting to lift the cuts.

Except for instances 175_9 and 175_10 that were not solved by CBEN, HCBEN and CBEN solved the same set of instances to optimality. While for HBCD, the total time involved in lifting the feasibility cuts could be significant, this time is smaller for HCBEN. That happens because HBCD calls BBU more frequently and because the sizes of the sets/subgraphs of G for which the liftings are computed are typically small under the Benders framework. Conversely, for HBCD, the subgraphs can be almost as large as G, depending on the size of the fundamental cycles of T_1 and T_2 that are characterized during our separation heuristics.

The fact that stronger feasibility cuts are being used within HCBEN as opposed to CBEN translates into fewer feasibility problems to be solved, to verify optimality. Conversely, the average time involved in solving each Benders feasibility problem increased substantially for HCBEN. That is true as, compared to CBEN, the overall CPU times for HCBEN sometimes increased substantially, despite the fact that HCBEN usually needed fewer feasibility problems and cuts to solve the problem. Note that the times involved in computing the liftings were small, usually much smaller than the increase in CPU time observed from CBEN to HCBEN. As an example, consider instance 150_4. HCBEN needs about one half of the iterations implemented by CBEN. Nevertheless, HCBEN takes 70% more time to solve that instance. The calls of BBU within HCBEN, though, are responsible for only 33 of these 730 additional CPU seconds.

5. CONCLUSIONS AND FUTURE RESEARCH

In this article, we provided integer programming formulations and exact solution approaches for the DPSTP. Classes of valid inequalities were also proposed and used within these algorithms.

In total, four algorithms were proposed. BBU, a BB based on an undirected spanning tree representation model, BCD, a BC algorithm based on a spanning arborescence formulation, CBEN, an iterative probing combinatorial Benders decomposition and, finally, BBI, a BB based on the application of the reformulation by intersection technique to the directed model.

Despite being based on the weakest formulation, the BB algorithm based on the undirected spanning tree representation provided the best computational results, due to conveniently chosen branching and search policies, combined to the fact that its upper bounding procedure runs very fast. The second best was CBEN, followed by the BCD algorithm.

Motivated by the good results obtained by the BBU method, we proposed two additional algorithms, a hybrid Benders and a hybrid BCD. These two algorithms make calls to the BBU method, to lift a particular type of valid inequalities that were proposed here. These inequalities, denoted feasibility cuts, impose that the maximum number of full degree vertices in a cycle-free subgraph of G must not exceed

TABLE 5. Computational results for hybrid algorithms, HCBEN and HBCD

Instance		HCBEN						HBCD				
		P_C						P_D^{r+}				
Name	Den (%)	BLB	Proven ?	$t(s)$	$t_l(s)$	iter	Cuts	BLB	BUB	$t(s)$	$t_l(s)$	Nodes
25_1	10.3	17	Yes	0.53	0.02	56	175	17	17	0.04	0.00	2
25_2	11.7	15	Yes	0.07	0.01	14	38	15	15	0.03	0.00	2
25_3	12.0	12	Yes	9.94	0.02	169	454	12	12	0.09	0.00	51
25_4	14.0	11	Yes	0.60	0.01	38	153	11	11	0.07	0.00	3
25_5	17.7	8	Yes	0.09	0.07	1	1	8	8	0.08	0.00	2
25_6	18.3	8	Yes	0.26	0.03	19	102	8	8	0.03	0.00	1
25_7	25.7	5	Yes	0.28	0.01	22	139	5	5	0.09	0.00	3
25_8	27.0	4	Yes	0.38	0.02	29	219	4	4	0.22	0.00	5
30_3	8.5	22	Yes	0.07	0.01	13	42	22	22	0.02	0.00	2
30_2	10.3	17	Yes	0.18	0.12	7	32	17	17	0.02	0.00	1
30_1	10.6	17	Yes	0.11	0.01	19	82	17	17	0.02	0.00	1
30_4	27.8	4	Yes	0.62	0.05	30	307	4	4	0.38	0.05	3
50_2	10.1	15	Yes	45.02	0.29	126	1139	15	15	0.49	0.21	2
50_3	10.4	16	Yes	12.42	0.50	101	928	16	16	0.07	0.00	1
50_1	29.6	3	Yes	3.98	0.69	69	1090	3	3	9.21	1.10	49
50_4	32.6	2	Yes	6.78	0.99	127	1308	2	2	18.95	2.62	75
75_2	10.5	13	Yes	147.54	0.97	157	2726	13	13	48.54	10.24	452
75_1	10.9	11	Yes	198.99	1.11	226	3409	11	11	319.72	49.00	1468
75_3	19.4	4	Yes	37.84	2.19	280	3793	4	4	423.14	50.69	991
75_4	29.4	2	Yes	29.55	3.17	220	3202	2	2	99.12	9.45	149
100_4	6.0	25	No	—	9.11	980	12580	25	25	66.61	34.22	387
100_2	9.5	11	No	—	2.81	930	10144	11	12	—	327.43	7978
100_1	10.1	12	Yes	1664.81	2.41	461	7292	12	13	—	372.16	1689
100_3	10.6	11	Yes	593.76	2.98	340	6177	11	12	—	312.92	3238
125_1	6.9	17	No	—	5.02	1240	17156	18	20	—	450.55	1512
125_2	7.6	15	No	—	10.58	1057	16586	14	17	—	115.41	4848
125_3	10.5	10	Yes	1433.62	8.18	484	10260	10	12	—	215.47	1753
125_4	19.5	4	Yes	335.32	12.85	316	9689	4	4	3191.64	118.91	853
150_1	7.2	16	No	—	10.31	1084	19690	16	19	—	231.96	610
150_2	7.6	16	No	—	15.92	712	16832	16	18	—	82.32	1225
150_3	10.1	9	Yes	2072.85	15.00	852	16628	9	12	—	217.07	1827
150_4	19.3	3	Yes	1680.83	32.77	809	16805	2	5	—	202.76	403
175_1	4.8	25	No	—	7.85	956	21761	27	31	—	104.84	540
175_2	5.1	24	No	—	7.90	1181	24377	25	30	—	225.17	638
175_3	10.0	8	No	—	28.06	1127	24362	8	12	—	99.80	87
175_4	10.1	8	No	—	16.00	1191	24062	8	12	—	88.77	402
175_5	14.7	5	Yes	1036.93	16.73	795	25446	5	7	—	121.20	170
175_6	14.9	4	Yes	1740.47	14,60	1270	29743	4	8	—	57.21	68
175_7	19.1	3	Yes	1856.66	20.34	768	20364	3	5	—	122.85	190
175_8	19.7	2	No	—	22.86	1804	26980	2	5	—	61.86	74
175_9	25.0	1	Yes	577.09	40.84	1063	9843	1	3	—	136.90	227
175_10	25.3	1	Yes	861.56	41.16	1175	10972	1	3	—	120.91	187

the number of full degree vertices in an optimal solution for a DPSTP instance, conveniently defined for that subgraph. As such, the computation of the optimal feasibility cuts involves solving a DPSTP for a subgraph of G. The BBU method was thus used to compute the optimal cuts within the Benders and the BCD frameworks. Our computational results indicated that, for dense instances, the hybrid BCD method outperformed its original version that uses suboptimal feasibility cuts. The hybrid Benders had its total number of Benders iterations reduced as a consequence of the optimal cuts. However, the overall CPU times were not reduced accordingly, as each of these Benders problems became much harder to solve.

One interesting possible future research consists of investigating, from a theoretical point of view, optimal liftings of the feasibility cuts for certain classes of graphs. That would allow us to use these cuts, with optimal coefficients, without the need of calling an optimization procedure to compute them, as we did here for general graphs.

We believe that the hybrid algorithms we introduced here are promising and deserve further investigation. For the hybrid Benders method, one interesting future research is to devise specific algorithms to solve each Benders feasibility problem, instead of using the black-box MIP tool we used here. For example, a constraint programming approach may be suitable for dealing with each Benders feasibility

problem. The hybrid BCD algorithm could also benefit from other lifting strategies that operate under more selective and/or restrictive policies for calling the BBU algorithm used to lift the cuts. Another algorithm that may be worth investigating is a BC method based on the y formulation, that only separates split and cycle cuts for integer valued points.

Finally, once the reformulation by intersection provided the strongest upper bounds in our study, other algorithms based on that model might be worth investigating. Among them, we could list decomposition approaches like classic Benders decomposition and Lagrangian relaxation, to handle the large number of constraints and variables in that formulation.

Acknowledgments

All sources of financial support are gratefully acknowledged.

REFERENCES

[1] R. Bhatia, S. Khuller, R. Pless, and Y.S. Sussmann. The full-degree spanning tree problem. Networks 36 (2000), 203–209.

[2] H. Broersma, O. Koppius, H. Tuinstra, A. Huck, T. Kloks, D. Kratsch, and H. Müller, Degree-preserving trees, Networks 35 (2000), 26–39.

[3] G. Codato and M. Fischetti, Combinatorial Benders' cuts for mixed-integer linear programming, Oper Res 54 (2006), 756–766.

[4] M.L. Fernandes and L. Gouveia, Minimal spanning trees with a constraint on the number of leaves, Eur J Oper Res 104 (1998), 250–261.

[5] T. Fujie, An exact algorithm for the maximum-leaf spanning tree problem, Comput Oper Res 30 (2003), 1931–1944.

[6] B. Gendron, A. Lucena, A.S. da Cunha, and L. Simonetti, The degree preserving spanning tree problem: Valid inequalities and branch-and-cut method, Electron Notes Discrete Math 41 (2013), 173–180.

[7] B. Gendron, A. Lucena, A.S. da Cunha, and L. Simonetti, Benders decomposition, branch-and-cut, and hybrid algorithms for the minimum connected dominating set problem, INFORMS J Comput 26 (2014), 645–657.

[8] L. Gouveia and J. Telhada, The multi-weighted steiner tree problem: A reformulation by intersection, Comput Oper Res 35 (2008), 3599–3611.

[9] L. Gouveia and J. Telhada, Reformulation by intersection method on the MST problem with lower bound on the number of leaves, In J. Pahl, R. Torsten, S. Voß (Editors), Lecture Notes in Computer Science, Proceedings of the 5th International Network Optimization Conference, Vol. LNCS 6701, Berlin, 2011, pp. 83–91.

[10] J.N. Hooker, Integrated methods for optimization, Springer, New York, 2007.

[11] S. Khuller, R. Bhatia, and R. Pless, On local search and placement of meters in networks, SIAM J Comput 32 (2003), 470–484.

[12] N. Lewinter, Interpolation theorem for the number of degree-preserving vertices of spanning trees, IEEE Trans Circ Syst 34 (1987), 205.

[13] C. Lin, G.J. Chang, and G. Chen, The degree-preserving spanning tree problem in strongly chordal and directed path graphs, Networks 56 (2010), 183–187.

[14] D. Lokshtanov, V. Raman, S. Saurabh, and S. Sikdar, On the directed full-degree spanning tree problem, Discrete Optim 8 (2011), 97–109.

[15] A. Lucena, N. Maculan, and L. Simonetti, Reformulation and solution algorithms for the maximum leaf spanning tree problem, Comput Manag Sci 7 (2010), 289–311.

[16] T.L. Magnanti and L. Wolsey, Optimal trees, In O. Ball et al. (Editors), Handbooks in OR and MS, Vol. 7, North-Holland, Amsterdam, 1995, pp. 503–615.

[17] L.C. Martinez and A.S. da Cunha, The min-degree constrained minimum spanning tree problem: Formulations and branch-and-cut algorithm, Discrete App Math 164 (2014), 210–224.

[18] L.C. Martinez and A.S. da Cunha, A parallel lagrangian relaxation algorithm for the min-degree constrained minimum spanning tree problem, In A.R. Mahjoub et al. (Editors), Lecture Notes in Computer Science, Vol. 7422, 2nd International Symposium on Combinatorial Optimization, Athens, 2012, pp. 237–248.

[19] I.W.M. Pothof and J. Schut, Graph-theoretic approach to identifiability in water distribution network. Memorandum 1283, Faculty of Applied Mathematics, University of Twente, Enschede, The Netherlands, 1995.

[20] R.E. Tarjan, Data structures and network algorithms, Society for Industrial and Applied Mathematics, Philadelphia, 1983.