



CIRRELT

Centre interuniversitaire de recherche
sur les réseaux d'entreprise, la logistique et le transport

Interuniversity Research Centre
on Enterprise Networks, Logistics and Transportation

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

**Bernard Gendron
Abilio Lucena
Alexandre Salles da Cunha
Luidi Simonetti**

November 2013

CIRRELT-2013-70

Bureaux de Montréal :
Université de Montréal
Pavillon André-Aisenstadt
C.P. 6128, succursale Centre-ville
Montréal (Québec)
Canada H3C 3J7
Téléphone : 514 343-7575
Télécopie : 514 343-7121

Bureaux de Québec :
Université Laval
Pavillon Palasis-Prince
2325, de la Terrasse, bureau 2642
Québec (Québec)
Canada G1V 0A6
Téléphone : 418 656-2073
Télécopie : 418 656-2624

www.cirrelt.ca

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

Bernard Gendron^{1,*}, Abilio Lucena², Alexandre Salles da Cunha³, Luidi Simonetti⁴

¹ Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT) and Université de Montréal, Pavillon André-Aisenstadt, Department of Computer Science and Operations Research, P.O. Box 6128, Station Centre-Ville, Montréal, Canada H3C 3J7

² Departamento de Administração, Universidade Federal do Rio de Janeiro, Av. Pedro Calmon, 550 - Cidade Universitária, Rio de Janeiro - RJ, 21941-901, Brasil

³ Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, Avenida Presidente Antônio Carlos, 6627 - Ventosa, Belo Horizonte - MG, 31270-901, Brasil

⁴ Instituto de Computação, Universidade Federal Fluminense, Niterói, R. Miguel de Frias, 9 - Icaraí, Niterói - RJ, 24220-008, Brasil

Abstract. 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 with 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 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 branch-and-cut are proposed. Extensive computational experiments are conducted with the solution algorithms introduced in this study.

Keywords. Spanning trees, vertex feedback edge set problem, Benders decomposition, branch-and-cut, Lagrangian relaxation

Acknowledgements. Bernard Gendron research is partially funded by the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grants Program. Abilio Lucena is partially funded by CNPq grant 310561/2009-4 and FAPERJ grant E26-110.552/2010. Alexandre Salles da Cunha is partially funded by CNPq grants 305423/2012-6, 477863/2010-8 and FAPEMIG grants PRONEX APQ-01201-09 and PPM-VII-00164-13. Luidi Simonetti is partially funded by FAPERJ and CNPq grants 483.243/2010-8, 304793/2011-6. All sources of financial support are gratefully acknowledged.

Results and views expressed in this publication are the sole responsibility of the authors and do not necessarily reflect those of CIRRELT.

Les résultats et opinions contenus dans cette publication ne reflètent pas nécessairement la position du CIRRELT et n'engagent pas sa responsabilité.

* Corresponding author: Bernard.Gendron@cirrelt.ca

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* 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].

DPSTP has also been investigated in [9] 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. Since 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 [15]. That could be accomplished by simply installing flow meters in all network edges. However, a cheaper alternative is 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 [15], 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 [15]. 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]. 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 [11].

Lewinter [10] 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 heuris-

tics [9, 15]. 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 [1] and the branch-and-cut (BC) algorithm in Gendron et al. [5]. The former algorithm is based on linear programming (LP) upper bounds given by a multi-commodity network flow formulation of the problem. However, virtually no computational results are available for it.

This paper improves on our previous results [5] 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* [6]. These new algorithms are compared here to an improved version of the BC algorithm in [5]. In this process, additional optimality certificates are obtained for the test instances used in [5].

The paper 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 paper 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 $A(S, V \setminus S) = \{(i, j) \in A : i \in S, j \notin S\}$, which denotes the set of arcs pointing from a vertex in S to a vertex in $V \setminus S$. 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 $e \in E \setminus E_T$, 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 non-symmetrical model, i.e., LP upper bounds vary according to the vertex chosen as the arborescence root. Accordingly, we resort to the reformulation by intersection technique [6] 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 to enforce those properties of full degree sets of vertices to be presented next.

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. On the other hand, 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. \square

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:

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

and

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

Notice that cycle cuts (2) can be understood as a stronger particular case of split cuts (1), i.e., 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:

$$y(V') \leq \alpha(V'), \quad (3)$$

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

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

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

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

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:

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

where P_u denotes the intersection of the spanning tree polytope P_{STP} , i.e.,

$$z(E) = n - 1 \quad (7a)$$

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

$$z_e \leq 1, e \in E, \quad (7c)$$

$$z_e \geq 0, e \in E, \quad (7d)$$

with the degree-enforcing inequalities

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

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).

It is not difficult to check, at the LP relaxation of (6), that inequalities (8) must be tight. That remark holds because 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 the following LP:

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

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 $L(\mu) = \max\{\sum_{i \in V} (1 - \mu_i |\delta(i)|)y_i + \sum_{i \in V} \mu_i z(\delta(i)) : z \in P_{STP} \cap \mathbb{B}^m\}$. Given that the Lagrangian subproblem has the integrality property, the Lagrangian dual gives the same bound as the LP relaxation. Hence, optimal multipliers are $\mu_i = \frac{1}{|\delta(i)|}$ for all $i \in V$ and (9) is precisely the (optimal) Lagrangian subproblem.

As our computational results demonstrate, DPSTP upper bounds $w(P_u)$ are weak. However, a quite effective BB algorithm results from them. That remark holds because these bounds are very 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 originally proposed by Fujie [4] for the maximum leaf spanning tree problem. In that case, constraints akin to (8) are used to characterize leaf nodes in spanning trees. Accordingly, LP relaxation bounds for Fujie's formulation are obtained by essentially following the procedure described above.

Formulation P_u can be significantly strengthened with valid inequalities such as $y_i \leq z_{ij}$ and $y_j \leq z_{ij}$ for all $\{i, j\} \in E$. 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 since 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 pre-selected vertex r . Besides the previously defined variables $\{y_i : i \in V\}$, the formulation also involves binary variables $\{x_{ij}^r : (i, j) \in A\}$, to establish whether or not an arc $(i, j) \in A$ appears in the arborescence. In case it does, $x_{ij}^r = 1$ holds; otherwise, $x_{ij}^r = 0$ applies. The formulation is given by

$$w = \max \{y(V) : (x^r, y) \in P_d \cap \mathbb{R}^{2m} \times \mathbb{B}^n\}, \quad (10)$$

where polytope P_d is defined as

$$x^r(A) = n - 1 \quad (11a)$$

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

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

$$y_i - x_{ij}^r - x_{ji}^r \leq 0, \forall i \in V, \{i, j\} \in \delta(i) \quad (11d)$$

$$x_{ij}^r \geq 0, \forall (i, j) \in A, \quad (11e)$$

$$y_i \geq 0, \forall i \in V. \quad (11f)$$

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. Although indegree constraints (11b) are implied by (11a) and (11c), they are kept in the model because, somehow, they make it faster to compute the corresponding LP relaxation bound. Constraints (11d) enforce that i is of full degree only if an arc corresponding to every edge $\{i, j\} \in \delta(i)$, i.e., (i, j) or (j, i) , appears in the arborescence. Finally, notice that variables x^r need not be explicitly restricted to assuming integral values. Indeed, due to (11d), that is naturally implied by $y \in \mathbb{B}^n$.

In addition to x^r and y , the multi-commodity 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) [12]. The formulation in [1] also uses constraints (11d) to define full degree vertices. As a result, P_d and the formulation in [1] produce the same LP bound, $w(P_d)$.

Consider now the following degree enforcing inequalities:

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

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

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

Constraints (12a) impose that, if $i \in V \setminus \{r\}$, at most $|\delta(i)| - 2$ arcs must point outwards of i (resp. $|\delta(i)| - 1$, in case $y_i = 1$). Similar reasoning applies to inequalities (12b). On the other hand, 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_d .

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_d^r , 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_d , the resulting formulation 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. We denote by P_D^r the intersection of P_d with inequalities (12), for a choice of r as the root vertex.

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 3 *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*

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

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

- Case 1: $x_{ij}^r + x_{ji}^r = x_{ik}^r + x_{ki}^r = 0$.

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 $\{i, j\}$, i.e., $x_{ij}^r + x_{ji}^r = 1$. For the other edge, $x_{ik}^r + x_{ki}^r = 0$.

Then, since 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 $y(C \setminus \{i\}) + x_{ij}^r + x_{ji}^r + x_{ik}^r + x_{ki}^r \leq |C| - 2 + 1 = |C| - 1$ holds.

- Case 3: $x_{ij}^r + x_{ji}^r = x_{ik}^r + x_{ki}^r = 1$.

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. Since 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 $y(C \setminus \{i\}) + x_{ij}^r + x_{ji}^r + x_{ik}^r + x_{ki}^r \leq |C| - 3 + 2 = |C| - 1$ holds and the proof is complete. \square

2.4 Reformulation by intersection

One drawback of formulation P_D^r 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 [6]. The procedure was originally introduced for the multi-weighted 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 [13, 14] and the minimum spanning tree problem with a lower bound on the number of leaves [7].

The idea is to reformulate DPSTP by simultaneously taking into account all polytopes P_D^r , 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

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

and DPSTP may thus be reformulated as

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

where polytope P_I is given by the intersection of (14) and $\bigcap_{r=1}^n P_D^r$. An important property associated with formulation P_I is its compactness, i.e., neither subtour breaking constraints (7b) nor directed cutset constraints (11c) are required by it (see [6, 13] 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_D^r reinforced with inequalities (1),(2) and (13), BCD, and, finally, a BB algorithm based on P_I , BBI.

BBI was implemented by means of the XPRESS solver Mose1 programming language. All XPRESS default parameters for heuristics, branching rules and cutting policies were kept for that algorithm. Since 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, on the other hand, 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 $\text{MWTREE}(c)$, 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 enumeration 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, $\text{STAR}(p)$, 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 [16], $\text{STAR}(p)$ 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].

$\text{STAR}(p)$ 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, $\text{STAR}(p)$ 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, 8]. 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, that relies on the fact that full degree spanning trees of G interpolate [10], as mentioned in the Introduction. An obvious consequence of that property is the result that follows.

Corollary 4 *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

$$y(V) = |W| + 1. \quad (16)$$

By Corollary 4, 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 (rhs) 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 rhs 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

In order 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 rhs 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. In order 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 5 For a given $\hat{y} \in \mathbb{B}^n$, $\hat{S} = \{i \in V : \hat{y}_i = 1\}$ 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 carried out 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 $C = \{v_1, \dots, v_{|C|}\}$. Assume as well that the unique cycle in $(V, E_F \cup \{e\})$ is given by the following set of edges: $\{\{v_1, v_2\}, \{v_2, v_3\}, \dots, \{v_{|C|-1}, v_{|C|}\}, \{v_1, v_{|C|}\}\}$. One has two cases to analyse:

– $|C|$ is even.

Partition C into V_1 and V_2 such that $V_1 = \{v_1, v_3, \dots, v_{|C|-1}\}$ and $V_2 = \{v_2, v_4, \dots, v_{|C|}\}$. Since all edges in the cycle $\{\{v_1, v_2\}, \{v_2, v_3\}, \dots, \{v_{|C|-1}, v_{|C|}\}, \{v_1, v_{|C|}\}\}$ have at least one endpoint i such that $\hat{y}_i = 1$, either $y(V_1) \leq |V_1| - 1$ or $y(V_2) \leq |V_2| - 1$ are violated by \hat{y} .

– $|C|$ is odd.

Since $|C|$ is odd and every edge in the cycle either belongs to $\delta(\hat{S})$ or to $E(\hat{S})$, there must exist two consecutive cycle vertices, say v_1 and v_2 , such that $\hat{y}_{v_1} = \hat{y}_{v_2} = 1$ holds. This remark applies since, otherwise, an edge $\{v_1, v_2\}$ would exist in the cycle such that neither v_1 or v_2 belongs to $\Gamma(\hat{S})$. Now take $V' = \{v_1, v_2, v_4, \dots, v_{|C|-1}\} \subset C$ and notice that $\hat{y}(\{v_4, \dots, v_{|C|-1}\}) = \lfloor \frac{|C|-2}{2} \rfloor$, since, otherwise, there would exist a cycle edge that neither belongs to $\delta(\hat{S})$ or to $E(\hat{S})$. Thus $\hat{y}(V') = 2 + \lfloor \frac{|C|-2}{2} \rfloor = \lfloor \frac{|C|}{2} \rfloor + 1 > |V'| - 1 = \lceil \frac{|C|}{2} \rceil - 1$ holds and the split cut defined for V' is violated. \square

Strategies used for adding and lifting split and cycle cuts are the following. Whenever $C \subseteq \hat{S}$ holds, we investigate if the rhs 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 rhs 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.
After solving the Benders subproblem, run heuristic $\text{STAR}(p)$, 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 upon the best feasible solution so far obtained, update T^* and the rhs of (16), accordingly.
 - ii. Otherwise, i.e., 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 rhs 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, the strategy we outlined above led to better computational results. Fewer Benders iterations resulting when several cuts were simultaneously used, as indicated in the proof of Proposition 5.

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, a 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 since 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

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

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

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

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 (resp. S_1) are of incomplete (resp. full) degree. Now, substituting the rhs of (18) for their corresponding y entries in (17),

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

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:

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

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

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

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

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

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

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 Branch-and-bound 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. On the other hand, if $|\delta_T(i)| = |\delta(i)|$, for any $i \in S_0$, the subproblem is pruned by optimality. Validity of such observations results from the fact that M is a valid upper bound on w . If $|\delta_T(i)| < |\delta(i)|$ and $|\delta_T(j)| = |\delta(j)|$ respectively hold, for any $i \in S_0$ and for every $j \in S_1$, the LP relaxation for that node is then 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

BCD firstly 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 $\bar{A} = \{(i, j) \in A : \bar{x}_{ij}^r > 0\}$, is the support graph associated with (\bar{x}^r, \bar{y}) , i.e., 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 $\{\bar{x}_{ij}^r : (i, j) \in \bar{A}\}$. 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 firstly define edge weights $c_{ij} = \bar{x}_{ij}^r + \bar{x}_{ji}^r, \{i, j\} \in E$ 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 $C_{T_1, e} = \{v_1, \dots, v_{|C_{T_1, e}|}\}$. If $|C_{T_1, e}|$ is even, one sets $V_1 = \{v_1, v_3, \dots, v_{|C_{T_1, e}|-1}\}$ and $V_2 = \{v_2, v_4, \dots, v_{|C_{T_1, e}|}\}$ 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 5, i.e., 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 $C_{T_1, e}$, i.e., $v_4, \dots, v_{|C_{T_1, e}|-1}$. 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.

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.

We experimented with a branching strategy where higher priority was given to variables corresponding to vertices of G with higher edge degrees. As was the case for BBU, our expectation was to quickly improve best upper bounds. The reasoning being that a large number of subproblems would be quickly declared infeasible. However, for the BCD case, better computational results were obtained by sticking to XPRESS's default branching policies.

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

4 Computational Results

For each size $n \in \{30, 50, 75, 100, 125, 150\}$, we generated four DPSTP instances without cutnodes nor bridges as follows. We first include in E edges in an Hamiltonian cycle of G : $\{\{1, 2\}, \{2, 3\}, \dots, \{n-1, n\}, \{1, n\}\}$. For the other pairs $i, j \in V$, we include the corresponding edge $\{i, j\}$ into E , according to a probability, defined by a desired graph density. In total, 24 instances were tested, with graph densities in the range from 6% up to 30%.

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 hexa-core machine, running at 2.4GHz, with 24 GB of shared RAM memory. All algorithms were executed with only one core; no multi-threading was allowed. XPRESS MIP package release 23.01.06 was used to implement the CBEN, BCD and BBI.

4.1 LP upper bounds

In Table 1, we present LP upper bounds for each formulation considered here. The first two columns of the table 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 $w(P_d)$. The next column stands for $w(P_d^+)$, the bound implied by P_d 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 table is the one provided by the reformulation by intersection. Finally, 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 P_D^r and P_D^{r+} , we have chosen $r = 1$ as the root arborescence.

As one could expect, the evaluation of $w(P_I)$ is very time consuming. 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 instance 30_4. Nevertheless, the BCD algorithm that separates these constraints outperforms its counterpart that does not. Degree enforcing constraints significantly improved on bounds $w(P_d)$. 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 3 units away from optimal solution values.

4.2 Comparison of exact algorithms

Detailed computational results for BBI, CBEN, BBU and BCD are given in Table 2. 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, 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 table, 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 grounded in 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 19 out of 24 instances, within the imposed time limit. Except for instances 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, since infeasibility could be early detected for many branches in the tree. We tested another BBU

Instance	d (%)	Linear Programming upper bounds						OPT	STAR(p)
		$w(P_u)$	$w(P_d)$	$w(P_d^+)$	$w(P_D^r)$	$w(P_D^{r+})$	$w(P_I)$		
30_1	10.6	21.600	19.477	19.477	17.423	17.423	17.423	17	15
30_2	10.3	21.926	19.405	19.162	17.000	17.000	17.000	17	16
30_3	8.5	25.333	23.337	23.128	22.000	22.000	22.000	22	21
30_4	27.8	8.746	7.195	7.195	5.309	5.285	5.285	4	4
50_1	29.6	8.547	6.776	6.772	4.789	4.789	4.745	3	3
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_4	32.6	7.517	6.140	6.140	4.026	4.026	3.996	2	2
75_1	10.9	22.468	18.597	18.597	13.814	13.814	13.734	11	11
75_2	10.5	24.413	19.592	19.592	15.566	15.566	15.563	13	13
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
100_1	10.1	26.071	20.395	20.395	15.679	15.679	15.663	12	11
100_2	9.5	25.828	21.019	21.019	14.919	14.919	14.919	11	9
100_3	10.6	24.484	19.195	19.195	14.345	14.345	14.326	11	9
100_4	6.0	41.479	33.915	33.915	27.529	27.529	27.502	25	22
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

Table 1: Linear Programming upper bounds and lower bounds for DPSTP. Any LP upper bound entry “-” indicates that the bound could not be evaluated within 1 CPU hour. Whenever the optimal objective function is unknown, the corresponding OPT entry is given together with a (?) indication.

implementation, for which higher branching priorities were given to vertices with the smallest degrees in G . Such an implementation solved only 13 out of the 19 instances originally solved by BBU. Among the BBU unsolved instances, no additional instance was solved by the second implementation. On the other hand, considering only those 13 instances that were solved by both, a 86% 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 and 16, respectively for instances 100_4, 150_1 and 150_2 while BBU did not improve the initial upper bounds of 22, 14 and 13, 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 18 out of 24 instances, being unable to solve 100_2 that could be solved by BBU. For instances that both algorithms 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 and 150_2, while the same did not happen for BBU.

BCD solved only half 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 five cases (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, much slower. For two instances that were not solved by any algorithm in this study, 125_1 and 125_2, the best known upper bounds were provided by BCD.

As a general rule, dense instances seem to be harder to BCD, since 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 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 $\text{STAR}(p)$ is usually smaller for these instances. Being so, the number of updates in the rhs of the feasibility constraint (16) is smaller.

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 in order 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 rhs is computed and only then the cut is appended to the model.

Computational results obtained by HCBEN and HBCD are given in Table 3, 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 Table 2. In Table 3, $t_l(s)$ stands for the total amount of time, in seconds, taken to compute the optimal rhs in the feasibility cuts, by calling BBU

within each algorithm, HCBEN and HBCD.

HBCD managed to solve 75_3 that was 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 951 nodes and about 808 seconds to solve that instance, HBCD needed only 137 nodes and 78 seconds to do the same. However, for sparse instances, the opposite seems to hold. BCD and HBCD needed the same number of nodes to solve 100_4, but HBCD took additional 60 seconds to solve that instance, precisely the additional time used to lift the cuts. 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 rhs 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. Putting all together, the impact of the time taken to lift the cuts in the overall HBCD time tend to be small and, thus, the computation of the optimal lifting had a positive impact in the overall CPU time.

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 reduced 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. On the other hand, 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, in order to testify optimality. On the other hand, the average time involved in solving each Benders feasibility problem increased substantially for HCBEN. That is true since, 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 paper, we provided integer programming formulations and exact solution approaches for the degree preserving spanning tree problem. Classes of valid inequalities were also proposed and used within these algorithms.

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

Despite being grounded on the weakest formulation, the BB algorithm based on the undirected spanning tree representation provided the best computational results. That applies due to conveniently chosen branching and search policies, combined to the fact that its upper bounding procedure runs very fastly. The second best was the Benders,

Instance		BBI				CBEN				BBU				BCD			
name	den(%)	P_I		$t(s)$	P_C		cuts	P_u		nodes	$w(P_D^+)$		$t(s)$	nodes			
		BLB	BUB		BLB	proven ?		$t(s)$	iter		BLB	BUB			BLB	BUB	
30-1	10.6	17	17	1.28	17	yes	0.17	14	70	9331	17	17	0.03	1			
30-2	10.3	17	17	0.90	17	yes	0.62	50	199	9327	17	17	0.03	1			
30-3	8.5	22	22	0.38	22	yes	0.14	15	55	587	22	22	0.03	1			
30-4	27.8	4	4	106.06	4	yes	0.48	48	477	1065	4	4	0.29	2			
50-1	29.6	-	4	-	3	yes	6.80	166	2797	2695	3	3	16.27	110			
50-2	10.1	15	15	2183.80	15	yes	34.27	136	1140	896530	15	15	0.29	2			
50-3	10.4	16	16	23.51	16	yes	8.08	101	980	352207	16	16	0.08	1			
50-4	32.6	-	3	-	2	yes	12.72	357	4108	2377	2	2	29.94	275			
75-1	10.9	-	12	-	11	yes	199.80	224	3430	1741013	11	11	111.38	1073			
75-2	10.5	-	14	-	13	yes	95.58	167	2809	1183469	13	13	41.03	399			
75-3	19.4	-	7	-	4	yes	39.38	346	5024	18343	4	4	-	2106			
75-4	29.4	-	-	-	2	yes	91.34	770	11948	5469	2	2	808.17	951			
100-1	10.1	-	15	-	12	yes	1360.64	462	7432	5587181	12	12	-	2364			
100-2	9.5	-	14	-	11	no	-	795	9763	13183851	11	11	-	9153			
100-3	10.6	-	14	-	11	yes	623.90	388	6538	3397119	11	11	-	4073			
100-4	6.0	-	25	-	25	no	-	557	8944	88773337	25	25	41.68	475			
125-1	6.9	-	-	-	17	no	-	1327	17831	58231725	18	20	-	2188			
125-2	7.6	-	-	-	14	no	-	1354	16780	56657444	15	17	-	6837			
125-3	10.5	-	-	-	10	yes	1296.96	540	10637	4113585	10	12	-	1771			
125-4	19.5	-	-	-	4	yes	249.21	647	21707	21951	4	4	-	1180			
150-1	7.2	-	-	-	16	no	-	1039	19459	42068826	16	19	-	1107			
150-2	7.6	-	-	-	16	no	-	928	18560	41117303	16	18	-	1634			
150-3	10.1	-	-	-	9	yes	1499.14	999	18222	6282171	8	12	-	396			
150-4	19.3	-	-	-	3	yes	947.61	1727	42730	28379	3	3	-	427			

Table 2: Comparison of DPSTP exact solution approaches.

Instance		HCBEN						HBCD				
name	den(%)	P_I						$w(P_D^+)$				
		BLB	proven ?	$t(s)$	$t_l(s)$	iter	cuts	BLB	BUB	$t(s)$	$t_l(s)$	nodes
30_1	10.6	17	yes	0.11	0.01	19	82	17	17	0.02	0.00	1
30_2	10.3	17	yes	0.18	0.12	7	32	17	17	0.02	0.00	1
30_3	8.5	22	yes	0.07	0.01	13	42	22	22	0.02	0.00	2
30_4	27.8	4	yes	0.62	0.05	30	307	4	4	0.38	0.05	3
50_1	29.6	3	yes	3.98	0.69	69	1090	3	3	8.66	0.90	28
50_2	10.1	15	yes	45.02	0.29	126	1139	15	15	0.50	0.22	2
50_3	10.4	16	yes	12.42	0.50	101	928	16	16	0.09	0.01	1
50_4	32.6	2	yes	6.78	0.99	127	1308	2	2	19.07	2.04	82
75_1	10.9	11	yes	198.99	1.11	226	3409	11	11	133.88	23.88	1037
75_2	10.5	13	yes	147.54	0.97	157	2726	13	13	43.61	10.54	137
75_3	19.4	4	yes	37.84	2.19	280	3793	4	4	561.06	69.33	851
75_4	29.4	2	yes	29.55	3.17	220	3202	2	2	78.67	9.9	137
100_1	10.1	12	yes	1664.81	2.41	461	7292	12	13	-	387.56	2448
100_2	9.5	11	no	-	2.81	930	10144	11	12	-	234.65	7086
100_3	10.6	11	yes	593.76	2.98	340	6177	11	12	-	278.66	3313
100_4	6.0	25	no	-	9.11	980	12580	25	25	102.91	60.11	475
125_1	6.9	17	no	-	5.02	1240	17156	18	20	-	487.77	1767
125_2	7.6	15	no	-	10.58	1057	16586	14	17	-	135.20	6019
125_3	10.5	10	yes	1433.62	8.18	484	10260	10	12	-	361.61	1316
125_4	19.5	4	yes	335.32	12.85	316	9689	4	5	-	90.58	417
150_1	7.2	16	no	-	10.31	1084	19690	16	19	-	261.67	1083
150_2	7.6	16	no	-	15.92	712	16832	16	18	-	200.40	1609
150_3	10.1	9	yes	2072.85	15.00	852	16628	8	12	-	70.31	386
150_4	19.3	3	yes	1680.83	32.77	809	16805	2	5	-	117.18	93

Table 3: Computational results for hybrid algorithms, HCBEN and HBCD.

followed by the BC algorithm.

Motivated by the good results obtained by the BB method, we proposed two additional algorithms, a hybrid Benders and a hybrid BC. These two algorithms make calls to the BB method, in order 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 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 BB method was thus used to compute the optimal cuts within the Benders and the BC frameworks. Our computational results indicated that, for dense instances, the hybrid BC 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, since 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 BC algorithm could also benefit from other lifting strategies that operate under more selective and/or restrictive policies for calling the BB algorithm used to lift the cuts.

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.

Acknowledgment Bernard Gendron is partially funded by the Natural Sciences and Engineering Council of Canada (NSERC) Discovery Grants Program. Abilio Lucena is partially funded by CNPq grant 310561/2009-4 and FAPERJ grant E26-110.552/2010. Alexandre Salles da Cunha is partially funded by CNPq grants 305423/2012-6, 477863/2010-8 and FAPEMIG grants PRONEX APQ-01201-09 and PPM-VII-00164-13. Luidi Simonetti is partially funded by FAPERJ and CNPq grants 483.243/2010-8, 304793/2011-6. 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*, 36:203–209, 2000.
- [2] H. Broersma, O. Koppius, H. Tuinstra, A. Huck, T. Kloks, D. Kratsch, and H. Müller. Degree-preserving trees. *Networks*, 35:26–39, 2000.
- [3] G. Codato and M. Fischetti. Combinatorial Benders’ cuts for mixed-integer linear programming. *Operations Research*, 54(4):756–766, 2006.
- [4] T. Fujie. An exact algorithm for the maximum-leaf spanning tree problem. *Computers and Operations Research*, 30:1931–1944, 2003.
- [5] B. Gendron, A. Lucena, A.S. da Cunha, and L. Simonetti. The degree preserving spanning tree problem: Valid inequalities and branch-and-cut method. *Electronic Notes in Discrete Mathematics*, 41:173–180, 2013.
- [6] L. Gouveia and J. Telhada. The multi-weighted Steiner Tree problem: A reformulation by intersection. *Computers and Operations Research*, 35:3599–3611, 2008.
- [7] 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, editor, *Lecture Notes in Computer Science, Proceedings of the 5th International Network Optimization Conference*, volume LNCS 6701, pages 83–91, 2011.
- [8] J. N. Hooker. *Integrated Methods for Optimization*. Springer, 2007.
- [9] S. Khuller, R. Bhatia, and R. Pless. On local search and placement of meters in networks. *SIAM Journal on Computing*, 32:470–484, 2003.
- [10] N. Lewinter. Interpolation theorem for the number of degree-preserving vertices of spanning trees. *IEEE Trans Circ Syst, CAS-34*, 1987.
- [11] D. Lokshtanov, V. Raman, S. Saurabh, and S. Sikdar. On the directed full-degree spanning tree problem. *Discrete Optimization*, 8:97–109, 2011.
- [12] T.L. Magnanti and L. Wolsey. Optimal Trees. In O. Ball et al, editor, *Handbooks in OR and MS*, volume 7, pages 503–615. North-Holland, 1995.
- [13] L.C. Martinez and A.S. da Cunha. The Min-Degree Constrained Minimum Spanning Tree Problem: Formulations and Branch-and-cut algorithm. *Discrete Applied Mathematics*.
- [14] 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., editor, *Lecture Notes in Computer Science*, volume 7422, pages 237–248, 2nd International Symposium on Combinatorial Optimization - Athens, 2012.
- [15] 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.
- [16] R.E. Tarjan. *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, 1983.