A Hybrid Relax-and-Cut/Branch and Cut Algorithm for the Degree-Constrained Minimum Spanning Tree Problem [*]

Alexandre Salles da Cunha
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais, Brasil
Email: acunha@dcc.ufmg.br

Abilio Lucena
Departamento de Administração
Universidade Federal do Rio de Janeiro, Brasil
Email: abiliolucena@globo.com

March 2008

Abstract

A new exact solution algorithm is proposed for the Degree-Constrained Minimum Spanning Tree Problem. The algorithm involves two combined phases. The first one contains a Lagrangian Relax-and-Cut procedure while the second implements a Branch-and-Cut algorithm. Both phases rely on a standard formulation for the problem, reinforced with Blossom Inequalities. An important feature of the proposed algorithm is that, whenever optimality is not proven by Relax-and-Cut alone, an attractive set of valid inequalities is identified and carried over, in a warm start, from Lagrangian Relaxation to Branch and Cut. In doing so, without the need of running any separation algorithm, one aims at obtaining an initial Linear Programming relaxation bound at least as good as the best Lagrangian bound previously obtained. Computational results indicate that our hybrid algorithm dominates similar hybrid algorithms based on the standard formulation alone. Likewise, it also dominates pure Branch and Cut algorithms based on either the standard formulation or its Blossom Inequalities reinforced version.

Keywords: Branch-and-Cut, Relax-and-Cut, Lagrangian warm start to cutting plane algorithms, Degree-Constrained Minimum Spanning Tree.

[*]This research was partially funded by CAPES grant BEX1543/04-0, for the first author, and grants 309401/2006-2 and E26/71.906/00, respectively from CNPq and FAPERJ, for the second author.

1 Introduction

Let $G = (V, E)$ be a connected undirected graph with a set V of vertices and a set E of edges. Assume that real valued costs $\{c_e : e \in E\}$ are associated with the edges of G. Assume as well that a spanning tree of G is called *degree-constrained* if edge degrees for its vertices do not exceed given integral valued bounds $\{1 \leq b_i \leq |V| - 1\ : i \in V\}$. The Degree Constrained Minimum Spanning Tree Problem (DCMSTP) is to find a least cost degree constrained spanning tree of G. Practical applications of DCMSTP typically arise in the design of electrical, computer, and transportation networks.

A new exact solution algorithm for DCMSTP is proposed in this paper. The algorithm involves two combined phases. The first one contains a Lagrangian Relax-and-Cut [41] procedure while the second implements a Branch-and-Cut [48] algorithm. The two phases are very closely integrated and both rely on a standard formulation for the problem, reinforced with Blossom Inequalities [19]. As such, if optimality is not proven at the Lagrangian phase of the algorithm, an attractive set of Subtour Elimination Constraints [14] and Blossom Inequalities are carried over to Branch-and-Cut. In doing so, without the need of running any separation algorithm, one aims at an initial Linear Programming relaxation bound at least as good as the best previously obtained Lagrangian bound. This step essentially defines a warm start to Branch-and-Cut and could be extended to other problems where Lagrangian relaxations are defined over matroids (see [21], for details). Computational results indicate that our hybrid algorithm dominates similar hybrid algorithms based on the standard formulation alone. Likewise, it also dominates pure Branch-and-Cut algorithms based on either the standard formulation or its Blossom Inequalities reinforced version.

To avoid unnecessary duplication of information, we refer the reader to [12] for an updated literature review. That reference also introduces the first phase of our hybrid algorithm, i.e. the one containing the Relax-and-Cut algorithm. Likewise, it also hints at the use of a hybrid algorithm, akin to the one described here. Finally, the Relax-and-Cut algorithm suggested in [12] was capable of generating the best heuristic solutions in the DCMSTP literature, quite often matched with corresponding optimality certificates. An additional recent reference to the problem is that in [26] where a DCMSTP variant involving vertex as well as edge costs is discussed.

This paper is organized as follows. In Section 2, the integer programming formulation used throughout the paper is described. For the sake of completeness, in Section 3, we briefly review the main aspects of our DCMSTP Relax-and-Cut algorithm, originally introduced in [12]. In Section 4, the motivation behind our proposed warm start to Branch-and-Cut is discussed. This is followed, in Section 5, by a detailed discussion of the Branch-and-Cut algorithm. Next, in Section 6, computational experiments are described. Finally, in Section 7, the paper is closed with some conclusions drawn from the computational results.

2 A Strengthened DCMSTP Formulation

Associate binary 0-1 variables $\{x_e : e \in E\}$ with the edges of G and denote by $E(S)$ the set of edges with both endpoints in $S \subset V$. Additionally, denote by $\delta(S)$ those edges with only one endpoint in S and let

$x(M)$ be the sum of the decision variables associated with the edges in $M \subseteq E$. A standard DCMSTP formulation is then given by

$$z = \min \left\{ \sum_{e \in E} c_e x_e : x \in \mathcal{R}_1 \cap \mathbb{Z}^{|E|} \right\}, \tag{1}$$

where $\mathcal{R}_1$ is the polyhedral region defined by

$$x(E) = |V| - 1, \tag{2}$$

$$x(E(S)) \leq |S| - 1, \, \forall S \subset V, \; S \neq \emptyset, \tag{3}$$

$$x(\delta(i)) \leq b_i, \; \forall \, i \; \in \; V, \tag{4}$$

$$x_e \geq 0, \; \forall e \; \in \; E. \tag{5}$$

Inequalities (2), (3) and (5), to be denoted here polyhedral region $\mathcal{R}_0$, fully describe the convex hull of the incidence vectors for spanning trees of G [18] while the *b-Matching Inequalities* (b-MIs) (4) restrict admissible trees to those satisfying degree constraints. Formulation (1) could be strengthened with valid inequalities for the b-Matching Polytope, as suggested in [11].

A b-matching of G is a collection M of the edges of E, possibly containing multiple edge copies. Accordingly, M must be such that no more than b_i edges (edge copies) are incident on vertex $i \in V$. If the maximum number of copies of $e \in E$ is restricted in M to a nonnegative integer value u_e, the problem is known as the u-Capacitated b-Matching Problem (u-CbMP). If $b_i = 1, \forall i \in V$, the problem is known, simply, as the Matching Problem.

A complete description exists, through linear constraints, to the u-Capacitated b-Matching (u-CbM) Polytope [19]. Likewise, a polynomial time algorithm is available to optimize a linear function over that polytope. For the particular case where $u_e = 1$, $\forall e \in E$, a (non minimal) complete linear description of the 1-CbM Polytope is given by (4), (5) and Blossom Inequalities (BIs)

$$x(E(H)) + x(T) \leq \left\lfloor \frac{1}{2}(b(H) + |T|) \right\rfloor, \forall H \subseteq V, \, \forall T \subseteq \delta(H), \tag{6}$$

where $b(H)$ stands for $\sum_{i \in H} b_i$ and $b(H) + |T|$ is odd.

Inequalities (6) are clearly valid to DCMSTP. Indeed, the set of degree constrained trees of G are the common vertices of two integer polytopes: the Spanning Tree (ST) Polytope [18] and the 1-CbM Polytope [19]. In the literature, few cases exist where the intersection of two integer polyhedra remains integer (see [23, 29], for instance). For our specific case, the set of vertices of the polyhedron given by the intersection of the ST and the 1-CbM Polytopes may have extreme points other than those found in either of the formers. In spite of that, inequalities (6) could be used to strengthen lower bounds given by the Linear Programming (LP) relaxaton of (1). Other inequalities such as Combs and Clique Trees, respectively proposed for the Travelling Salesman Problem (TSP) in [27, 28], could also be generalized to DCMSTP [10] and thus be used to further strengthen the formulation above. In this study, however, we restrict ourselves to DCMSTP lower bounds strengthened only by BIs. This is carried out, firstly, in the Lagrangian Relax-and-Cut algorithm that follows.

3 A Relax-and-Cut algorithm for DCMSTP

Assume that multipliers $\lambda \in \mathbb{R}_+^{|V|}$ are used to dualize, in a Lagrangian fashion, inequalities (4) in DCMSTP formulation (1). A valid lower bound for the problem is thus given by the Lagrangian Relaxation Problem (LRP)

$$z_\lambda = \min \left\{ \sum_{e \in E} c_e x_e + \sum_{i \in V} \lambda_i (x(\delta(i)) - b_i) : x \in \mathcal{R}_0 \right\}, \tag{7}$$

having an unconstrained spanning tree of G as an optimal solution. The best possible bound attainable in (7) is obtained by solving the Lagrangian Dual Problem (LDP)

$$z_d = \max \; \{ z_\lambda : \lambda \geq 0 \} . \tag{8}$$

Bound z_d could be obtained (approximated), for instance, through the use of the Subgradient Method (SM) [31]. It could also be obtained through more robust methods like the Volume Algorithm [3, 2] or Bundle Methods [6]. We use SM for the computational experiments in this study. However, Relax-and-Cut implementations based on [2] or [6] naturally follow along the same lines discussed here.

At least two different approaches exist in the literature to attempt to go, within a Lagrangian relaxation framework, over the z_d bound (see details in [41]). The first one, suggested by Escudero, Guignard and Malik [20], was called Delayed Relax-and-Cut (DRC) in [41] (the term *Relax-and-Cut* was actually coined in [20]). The second approach, Non Delayed Relax-and-Cut (NDRC), was suggested in [39, 40, 41].

A DRC algorithm for DCMSTP is initiated by computing the z_d bound in (8). Having done that, valid inequalities, BIs for instance, violated at the solution to (8), are used to reinforce (2)–(5). That gives rise to a reinforced formulation of DCMSTP and to new, associated, LRPs and LDP. As a result, a new, reinforced, z_d bound could then be computed and the overall scheme repeated until a stopping criterion is met. This variant of Relax-and-Cut has been applied to the Sequential Ordering Problem in [20].

Differently from DRC, a NDRC algorithm would not *delay* the identification and use of violated BIs until LDP (8) is solved. Indeed, violated inequalities are attempted to be identified (and may be dualized) for every LRP eventually solved. The approach was originally devised for the Steiner Problem in Graphs [38, 39] and was used later on for the Edge-Weighted Clique Problem [32], the Quadratic Knapsack Problem [15], the Traveling Salesman Problem [4], the Vehicle Routing Problem [43], the Linear Ordering Problem [5], the Rectangular Partition Problem [8], the Capacitated Minimum Spanning Tree Problem [13], and the Set Partitioning Problem [9]. In experiments conducted in [41], a NDRC algorithm produced sharper bounds than a corresponding DRC algorithm. However, the experiment was quite limited and more general conclusions should not be drawn from it.

Irrespective of the Relax-and-Cut algorithm used for DCMSTP, separation problems must be solved to identify violated BIs. Typically, since Relax-and-Cut solutions are integral valued, these problems are computationally easier to solve than their LP based counterparts.

For the first phase of our hybrid algorithm, we use NDRC and SM to generate lower bounds to DCMSTP. Likewise, a NDRC based Lagrangian heuristic is also used in that phase to obtain feasible integral solutions to the problem.

3.1 NDRC lower bounds

Specific details on how a NDRC algorithm is implemented could be found, for instance, in [8, 41, 43]. However, it suffices to point out here that, contrary to all previously proposed Lagrangian relaxation algorithms to DCMSTP [24, 52, 7, 1], ours is based on the dynamic dualization of BIs (in addition to the *static* dualization of inequalities (4), as conducted in other algorithms). For any SM iteration, apart from the very first one, a LRP may involve, in addition to the inequalities in (4), dualized BIs, as well. Differently from the inequalities in (4), a BI would only be dualized *on the fly*, as it becomes violated at the solution to a LRP of value $z_{\lambda,\beta}$, where, as before, $\lambda \in \mathbb{R}_+^{|V|}$ are the multipliers associated with inequalities in (4) while multipliers β, where the dimension of β may vary along the SM iterations, are associated with dualized BIs.

A crucial step for NDRC (and for DRC, as well), is the identification, at a given Lagrangian Relaxation Problem solution, of violated constraints that are candidates to dualization. Denote such a solution by $\overline{x}^k$, where k identifies the current SM iteration. Algorithms available to separate BIs (see [37, 46], for instance), assume that (4) is not violated at the solution under investigation. Typically this is not the case for $\overline{x}^k$. However, $\overline{x}^k$ has a convenient structure (it is a spanning tree of G) and that could be used to advantage. Based on that structure, we developed a simple greedy heuristic in an attempt to find BIs violated at $\overline{x}^k$.

Since $\overline{x}^k$ corresponds to a spanning tree of G, we describe our separation algorithm in terms of the vertices and edges in that tree. The algorithm is initialized with a set H containing a single vertex for which degree constraints (4) are violated at $\overline{x}^k$. Then, one attempts to identify, by inspection, a conveniently defined accompanying edge set T for which $(b(H) + |T|)$ is odd. Having done that, set H is attempted to be extended, one vertex at a time. That is accomplished while considering as candidate vertices to enter H, spanning tree vertices that are neighbors to vertices already introduced into H. As before, this step must be complemented with the adequate choice of an associated set T. Among candidate vertices, the one selected to enter H, say vertex $j \in V$, is such that $H \cup \{j\}$ implies, for the current step, a most violated BI. Ties are broken arbitrarily. The procedure stops when no more violated BIs could be obtained for larger cardinality sets H.

Since many vertices may simultaneously violate degree constraints (4) at $\overline{x}^k$, potentially many violated BIs could be found through the procedure suggested above. Among these, we only dualize those violated BIs implying largest overall violation, provided they are not already currently dualized with a strictly positive multiplier. Dualized inequalities for which multipliers eventually drop to zero, at any given SM iteration, become *inactive*. As such, they are not explicitly considered for the update of multiplier values, in the SM iterations that follow. This, however, is changed if the inequality becomes yet again violated at the solution to a future LRP, thus becoming, once again, *active*.

As an example of the use of our separation algorithm, assume that the tree depicted in Figure 1 is implied by $\overline{x}^k$ and notice that the two filled circles (vertices) are associated with violated b-MIs. Vertices delimited by each different type of dotted line indicate the possible successive sets H_1, H_2 and H_3, obtained while enlarging the inequality from the left most dotted vertex. It should be noticed that two successive sets H_i and H_{i+1} differ by only one vertex, which must belong to the set of tree vertices that are neighbors to vertices in H_i.

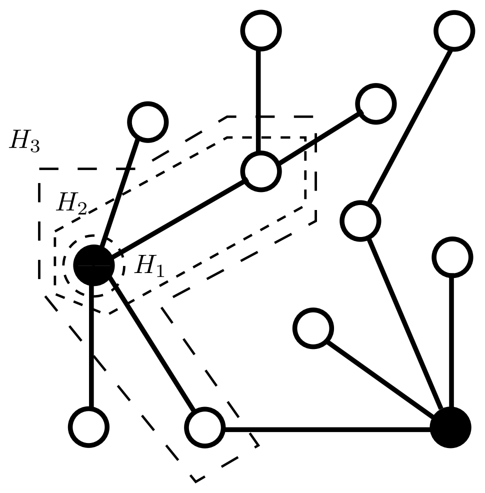

Figure 1: Schematic Relax-and-Cut separation of BIs.

3.2 NDRC based upper bounds

Our upper bounding strategy attempts to use Lagrangian dual information in a heuristic to generate feasible integral solutions to DCMSTP. The basic motivation behind such a strategy is the intuitive idea, validated by primal-dual algorithms, that dual solutions must carry relevant information for generating good quality primal ones. We implemented the suggested scheme within a DCMSTP NDRC framework.

Operating within a Lagrangian relaxation framework, our upper bounding algorithm involves two main steps. The first one is a constructive Kruskal like algorithm [36]. This procedure incorporates a *look ahead mechanism*, introduced for the DCMSTP in [1]. Selected edges of E enter the expanding forest $\mathcal{F}$, one at a time, in non-decreasing order of their costs. A candidate edge $e = (i, j)$ is accepted to enter $\mathcal{F}$ whenever the three conditions that follow are satisfied:

- $\mathcal{F} \cup (i, j)$ implies no circuits;
- $|\delta(i) \cap \mathcal{F}| \leq b_i - 1$ and $|\delta(j) \cap \mathcal{F}| \leq b_j - 1$;
- if $|\mathcal{F}| \neq |V| - 2$, $|\delta(i) \cap \mathcal{F}| + |\delta(j) \cap \mathcal{F}| > 2$.

The second of these three conditions enforces, at the current iteration of the procedure, the feasibility of the DCMSTP solution under construction. The third condition, i.e., the look-ahead mechanism, tries to preserve *vertex degree capacity* to allow the future inclusion of additional vertices into $\mathcal{F}$.

The procedure is called at every SM iteration under *complementary costs* $\{(1 - \overline{x}_e^k)c_e : e \in E\}$ where, as before, $\overline{x}^k$ is the solution to the LRP formulated at iteration k of SM. It should be noticed that, through the use of complementary costs, one tries to force the heuristic to select as many edges appearing in the Lagrangian dual solution as possible.

The second part of the algorithm is a Local Search (LS) procedure that takes as input a degree constrained spanning tree $\mathcal{T} = (V, E_{\mathcal{T}})$, generated by the Kruskal like algorithm above. For the LS procedure, we resort back to the original edge costs $\{c_e : e \in E\}$. At LS, one investigates a *1-Neighborhood* of $\mathcal{T}$. Specifically one investigates degree constrained spanning trees of G that differ from $\mathcal{T}$ by exactly one edge. Whenever a feasible spanning tree, cheaper than $\mathcal{T}$, exists in that neighborhood, $\mathcal{T}$ should be updated accordingly and LS be applied once again. Otherwise, degree-constrained spanning tree $\mathcal{T}$ represents a local optimum.

Typically, when implementing LS, one may choose to accept the first improving solution found (i.e., the *first improvement approach*) or else to take the best overall solution found after searching throughout the complete neighborhood (i.e., the *best improvement approach*). After some experimentation, we decided for the first improvement approach that produced same quality bounds in less CPU time.

In order to allow the use of the heuristic at every SM iteration, without incurring into excessive CPU time demands, we implemented some dominance tests, most of which were originally suggested in [49, 50]. These tests proved very effective. To use them, at the initial stage of LS, given the input solution $\mathcal{T}$, the following parameters are computed:

- cost $p(r, i)$, for all $i \in V$, of the most expensive edge in the unique path between i and a selected root node r of $\mathcal{T}$, where $i \neq r$. By definition $p(r, r) = 0$;
- $a(i) = \max \{c_e : e \in \delta(i) \cap E_{\mathcal{T}}\}, \forall i \in V$, i.e., the cost of the most expensive edge incident to i in $\mathcal{T}$;
- c^*, the cost of the most expensive edge in $E_{\mathcal{T}}$.

Due to the search neighborhood selected, any candidate edge for improving the current solution should have a cost less or equal to c^*. Therefore, any edges that do not satisfy this condition can be left out of an initial list of candidate edges. Suppose now that candidate edge $e = (i, j)$ is being investigated. If that edge has non saturated endpoints, i.e., if $|\delta(i) \cap E_{\mathcal{T}}| \leq b_i - 1$ and $|\delta(j) \cap E_{\mathcal{T}}| \leq b_j - 1$, inclusion of it in $\mathcal{T}$ and elimination of any $\mathcal{T}$ edge in the path between i and j leads to a feasible degree-constrained solution. However, prior to identifying the largest cost edge in that path, one checks if either $c_{(i,j)} < p(i, r)$ or $c_{(i,j)} < p(j, r)$. It is clear that if both conditions are not satisfied, the maximum cost of an edge in the path between i and j, in $\mathcal{T}$, can not exceed $c_{(i,j)}$. In order to improve test efficiency even further, we do not choose r arbitrarily. Ideally, r should be a non leaf vertex of $\mathcal{T}$ with maximun distance to a leaf vertex. Computing (and updating) such a vertex may be expensive. We therefore randomly choose r as a maximal edge degree vertex in $\mathcal{T}$. Finally, suppose that one of the endpoints of e, say i, is not saturated but the other, j, is, i.e., suppose that $|\delta(i) \cap E_{\mathcal{T}}| \leq b_i - 1$ and $|\delta(j) \cap E_{\mathcal{T}}| = b_j$ applies. It is thus clear that updating $\mathcal{T}$ with e would only be advantageous if $c_{(i,j)} < a(j)$. Whenever tree $\mathcal{T}$ is updated, all values involved in the dominance tests above could be efficiently updated through the data structures suggested in [25].

Our primal heuristic is clearly inspired by the DCMSTP heuristic introduced in [1]. Basic differences between the two relate to the constructive algorithms used in either case and the refinement details discussed above. However, the reinforced DCMSTP lower bounds obtained through NDRC contain better dual information for a multi-start Lagrangian heuristic than the lower bounds used in [1]. As a result, our upper bounds ended up being stronger than those in [1].

3.3 Variable fixing tests and Subgradient Optimization cycles

Whenever $|V| > 2$ and $b_i = b_j = 1$, for any $i, j \in V$, edge $e = \{i, j\}$ could be eliminated from E since no feasible DCMSTP solution may contain e. Additionally, at every SM iteration, Lagrangian relaxation solutions, together with their associated DCMSTP lower bounds, could be combined with valid DCMSTP upper bounds, in an attempt to price optimal edges in and suboptimal edges out of an optimal solution. That is accomplished through the use of LP *reduced costs* associated with STs.

For the particular case of STs, LP reduced costs could be obtained through conveniently defined exchanges of tree edges. Such an approach was first suggested, in connection with the TSP, by Volgenant and Jonker [53]. It follows from Gabow [22] and Katoh, Ibaraki and Mine [34], where edge exchanges are used to generate STs, ordered by cost. The approach was first used for DCMSTP in [51] and was later embedded within a Lagrangian relaxation for that problem in Volgenant [52]. It then became standard practice in the literature (see, for instance, [39, 40] for an application to the Steiner Problem in Graphs (SPG) and [1, 7] for additional applications to DCMSTP). The approach is explained next.

Suppose a Minimum Spanning Tree (MST) of G provides a valid lower bound on the optimal solution value of a given problem, defined over G. The LP reduced cost of an edge $e \in E$ (variable x_e), for that spanning tree, is given by the difference between the cost of e and the cost of the largest cost edge in the unique spanning tree path between the end vertices of e. Whenever an edge $e \in E$ has a reduced cost that is larger than the difference between a valid upper bound for our problem and the corresponding lower bound implied by the MST, that edge is guaranteed to be suboptimal. In this process, clearly, if all but one edge incident to vertex i is not priced out, that edge could be removed from the graph since a guarantee exists that it is optimal.

After applying SM to a Lagrangian relaxation of DCMSTP, assume that optimality of the best upper bound in hand could not be proven. Assume as well that a *significant number* of variables were eliminated through the variable fixation tests above. We would then keep applying new SM rounds, again and again, as suggested in [39, 40], for as long as optimality of the upper bound is not proven and, at end of the current SM round, a *significant number* of edges had been eliminated from G.

The idea of preprocessing instances of a problem prior to the application of an exact solution approach is clearly very attractive and is standard practice in the literature. Examples of that could be found, among others, in [17] (tests based on structural properties of the problem), in [7] (tests based on the LP reduced costs above) and in [8] (tests based on structural properties of the problem and on associated LP reduced costs).

4 NDRC warm start to a DCMSTP cutting plane algorithm

Without any need of running separation algorithms, Relax-and-Cut allows Lagrangian bounds to be directly carried over to a cutting plane algorithm. An example of that could be found in [8], where a NDRC algorithm, based on the dualization *on the fly* of Clique Inequalities, was proposed for a Cardinality Constrained Set Partitioning Problem. The integration between these two bounding approaches is made possible because Relax-and-Cut, in addition to dualizing inequalities as traditionally done in Lagrangian relaxation (i.e., in a static way with a relatively small number of inequalities being dualized), is also capable of dynamically dualizing inequalities (i.e., inequalities may move in and out of relaxations, according to some pre-defined dual strengthening criteria). This feature allows Relax-and-Cut to adequately accommodate exponentially many inequalities within a Lagrangian relaxation framework (while explicitly dualizing only a fraction of them). In that respect, Relax-and-Cut could then be understood (see [41], for details) as a Lagrangian relaxation analog to a polyhedral cutting-plane algorithm.

Consider our hybrid algorithm for DCMSTP, where NDRC is used as a preprocessor to Branch-and-Cut. For that algorithm, as explained before, NDRC is used, for instance, in attempts to price variables in and out of an optimal solution. Thus, in principle, our Lagrangian algorithm plays a role similar to that played by the Lagrangian algorithm in [7] (instead of a NDRC algorithm, a dual ascent procedure is used in [7]). However, differently from [7], we also manage to carry over to a cutting plane algorithm, in a warm start, the, hopefully sharp, lower bounds obtained at NDRC.

To implement the warm start suggested above, we use z_{λ^*,β^*} (i.e., the overall best Lagrangian lower bound obtained throughout SM), the Lagrangian Relaxation Problem associated with z_{λ^*,β^*}, and the solution x^* implied by z_{λ^*,β^*}. Consider as well, those BIs explicitly dualized, together with inequalities (4), at the SM iteration where z_{λ^*,β^*} was attained. Clearly, all these BIs should be included into the initial LP relaxation we are aiming for. Additionally, one would also wish to identify and include in that relaxation, a minimal set of SECs necessary to characterize, together with (2) and (5), the spanning tree of G implied by x^* (i.e., one wishes to identify hyperplanes associated with constraints (2), (3) and (5), whose intersection results in x^*). If an initial LP relaxation of DCMSTP contains constraints (2), (4), (5) and those BIs and SECs mentioned above, it is guaranteed to return a bound at least as good as z_{λ^*,β^*}. A procedure that attains that is explained next.

For the SM iteration associated with z_{λ^*,β^*}, assume that the edges in E are ordered in increasing value of their Lagrangian modified costs (ties are broken arbitrarily). A MST of G, under these costs, may then be generated, for instance, through the Kruskal algorithm [36]. In that case, ordered edges are included, one at a time, into the tree being constructed (provided they do not form a cycle with previously selected edges). After $m = |V| - 1$ edge inclusion operations, a MST of G is guaranteed to be obtained. Assume that such a tree is implied by edges $e_{k_1}, \ldots, e_{k_m}$, given here in the order they are selected by the Kruskal algorithm. Clearly, each of these edges, at the time they enter the expanding Kruskal tree, gives rise to a connected component that eventually contains only two vertices. We call it a *greedy set* S_l, where $S_l \subseteq V$, the set of vertices in the connected component resulting from the selection of e_{k_l}, where $l = 1, \ldots, m$. Greedy sets imply some beautiful properties in relation with the ST Polytope or, more generally, the Matroid Polytope. For instance, an optimality proof of the Kruskal algorithm may be obtained by concentrating only on the dual variables associated with inequality (2) and the m greedy set SECs (all other exponentially many dual variables being set to 0). Indeed, in doing so, it is possible to obtain an easy to compute dual feasible solution to MSTP, with the same value as the Kruskal tree (see [42] for details).

Motivated by the validity proof above, we bring over to our initial DCMSTP LP relaxation these SECs associated with $\{S_l : \; l = 1, \ldots, m\}$. This greedy set choice may also be used for additional problems associated with matroids. This suggestion applies since the proof of validity in [42] is nothing else but a particular case of the proof of validity of the greedy algorithm for combinatorial optimization problems defined over matroids (see Proposition 3.3, p. 669, in Nemhauser and Wolsey [44] for further details).

In a previous paper [12] we stated that, to the best of our knowledge at the time, the scheme outlined above had never been suggested before in attempts to identify *attractive* valid inequalities to be carried over from Lagrangian Relaxation to a cutting plane algorithm. However, we recently came across a paper by Fischetti et. al [21], published in 1997, where the same ideas involving the use of greedy sets were proposed for the Generalized Travelling Salesman Problem (GTSP).

Finally, we conclude this Section by pointing out that our warm start is substantially different from the DCMSTP Lagrangian pre-processing scheme used in [7]. In spite of that, one might also consider that scheme, through the problem input size reduction tests it implements, a warm start to a cutting plane algorithm.

5 The Branch-and-Cut algorithm

Implementation details of our DCMSTP Branch-and-Cut algorithm, follow. We specifically indicate the families of valid inequalities used, the cut-pool management being enforced and certain details that are specific to our implementation.

5.1 Valid inequalities and separation procedures

A valid LP relaxation to DCMSTP is given by

$$z_2 = \min \left\{ \sum_{e \in E} c_e x_e : x \in \mathbb{R}^{|E|} \cap \mathcal{R}_2 \right\}, \tag{9}$$

where polyhedral region $\mathcal{R}_2$ is defined by constraints (2), (4) and (5). Let $\overline{x}$ be the optimal solution to (9) and $\overline{G} = (V, \overline{E})$ be the support graph implied by $\overline{x}$, where $\overline{E} = \{e \in E : 0 < \overline{x}_e \leq 1\}$. If $\overline{x}$ is integer and $\overline{G}$ is connected, $\overline{x}$ implies an optimal solution to (1). Otherwise, prior to branching, bound z_2 is attempted to be strengthened by appending violated valid inequalities to the relaxation.

Similarly to [21], but differently from most Branch-and-Cut algorithms in the literature (see [7], for instance), we first look for violated inequalities among those carried over from NDRC. Clearly, this is implemented without the need of solving any separation problem. Let I_1 and I_2 be, respectively, the set of greedy SECs and BIs associated with z_{λ^*,β^*}. All of these inequalities are initially stored in a cut-pool for the problem. Only after we fail to find violated inequalities in the cut-pool, we resort to separation algorithms.

Before moving on to describe our separation algorithms, it is important to mention that the number of inequalities in I_1 and I_2 is typically very small. For instance, $|I_1|$ is limited from above by the number of iterations m, for the Kruskal algorithm. Similarly, in our NDRC experiments, the number of explicitly dualized BIs always remained between forty and sixty percent of the number of vertices in V.

5.1.1 Cut-set inequalities

It is well known that a directed spanning tree formulation, based on cut-set inequalities, is weaker than its SECs based undirected counterpart (see [42], for instance). However, separating cut-sets in a Branch-and-Cut algorithm for a NP-Hard *generalized spanning tree* problem may prove computationally attractive. A reason being that cut-sets are separated, exactly, in $O(|V|^3)$ time, while the exact separation of SEC takes $O(|V|^4)$, in the worst case.

A cut-set separation algorithm attempts to identify, provided one exists, a set of vertices $S \subset V$ such that the capacity of the cut $C(S, V \setminus S)$ in $\overline{G}$, i.e., $\sum_{(i,j)\in\overline{E}:i\in S,j\notin S} \overline{x}_{(i,j)}$, is less than one. If such a set is found, a violated cut-set inequality is implied. Separation of cut-set inequalities is thus equivalent to identifying a min-cut in a capacitated network originating from $\overline{G}$ [45].

To speed up the separation of violated cut-set inequalities, we have used procedures similar to those in [45]. Taking $\overline{G}$ as an input, these procedures output a reduced graph $\overline{G}_r = (\overline{V}_r, \overline{E}_r)$. They are based on the idea of contracting into a single vertex the endpoints of edges (i, j), such that $\overline{x}_{(i,j)} = 1$. Whenever i and j satisfy the contraction condition, no violated cut-set inequality, with i and j on different shores, may possibly exist in $\overline{G}$. This applies since a cut capacity of less than 1 is required to imply violation. As such, i and j must necessarily belong to the same shore of a violated cut-set inequality, provided one exists. Suppose now that i and j share a common vertex l in $\overline{G}$. In that case, contracting i and j into a single vertex $\{i, j\}$ also implies the merging of edges (i, l) and (j, l) into a single new edge $(\{i, j\}, l)$. In doing so, it is necessary to set $\overline{x}_{(\{i,j\},l)} = \overline{x}_{(i,l)} + \overline{x}_{(j,l)}$. After a contraction operation, as one updates edges and their corresponding values, a one-to-one relationship is maintained between minimum capacity cuts in $\overline{G}$ and minimum capacity cuts in $\overline{G}_r$. The contraction procedure is recursively applied until no edge e exists with $\overline{x}_e \geq 1$. For our computational experiments, we have used the $O(|V|^3)$ min-cut algorithm of Hao and Orlin [30] to separate cut-set inequalities.

One should notice that edge contraction offers, as a by product, a heuristic to separate SECs. Accordingly, let $\{j_1, \dots, j_p\} \subset V$ be a set of nodes that were merged throughout the application of the schrinking procedure described above. If $\overline{x}(E(\{j_1, ..., j_p\})) > p - 1$, a violated SEC is implied. For the computational results we present, we have used this heuristic to separate SECs.

It eventually may be the case that graph $\overline{G}$ is not connected. Faced with this situation, we apply the contraction procedure followed by the algorithm of Hao and Orlin, to each connected component in $\overline{G}$. Additionally, for non connected graphs, a violated cut-set inequality is naturally implied by any pair of disjoint connected components.

For our computational experiments, we found it advantageous to only reinforce LP relaxations with inequalities implying violations of at least 10^{-2}.

5.1.2 Subtour elimination constraints

Contraction operations may mislead the search for violated SECs. For that reason, for situations to be explained next, we also rely on the Padberg-Wolsey algorithm [47] for the exact separation of SECs in $\overline{G}$. Among all violated inequalities returned by that algorithm, we only retain the most violated one.

In our computational experiments, we observed that introducing at most one violated SEC per separation round, produced better results than other strategies commonly used in the literature. This specifically applied to introducing the $k > 1$ most violated SECs per round [48] or introducing orthogonal cuts as in [35].

We close this Subsection by pointing out that a SEC defined by $S \subset V$, where $b_i = 1$ for a given vertex $i \in S$, can not define a facet of the DCMSTP Polytope. This occurs because that inequality is dominated by the inequality obtained after adding together the SEC associated with $S \setminus \{i\}$ and the degree constraint for i. Accordingly, while building the network required by the Padberg-Wolsey algorithm, we do not take into account a vertex $i \in V$ for which $b_i = 1$. Likewise, we do not consider these vertices in the previously defined greedy sets S_j.

5.1.3 Blossom inequalities

The algorithm of Letchford et al. [37] solves exactly, in $O(|V|^4)$ time, the separation problem for BIs. However, we have opted for not including it in our Branch-and-Cut algorithm. We have used instead a very simple BI separation heuristic. In our computational experiments, this heuristic, under very acceptable CPU times, was capable of almost entirely close the gaps between z_{SEC} and z_{BLO}, i.e., respectively the LP relaxation bound for the standard DCMSTP formulation (1) and that for the standard formulation reinforced with BIs (6). Prior to describing the heuristic, it should be noticed that BIs could be conveniently written in the following *closed form*:

$$x(\delta(H) \setminus T) + \sum_{e \in T \subseteq \delta(H)} (1 - x_e) + \sum_{i \in H} (b_i - x(\delta(i))) \geq 1. \tag{10}$$

In an initial step, all edges $\{e \in \overline{E} : \overline{x}_e = 1\}$ are eliminated from the support graph of $\overline{x}$. One then investigates every connected component in the resulting graph G' for candidate handles, i.e., vertex sets H in (10). Assume that G' has $p > 1$ connected components, denoted here $\{G'_i = (V'_i, E'_i) : i = 1, \ldots, p\}$. Assume as well that the slack $\sum_{i \in V'_i} (b_i - \overline{x}(\delta(i)))$, associated with G'_i, is less than 1 and that G'_i is odd, i.e., $\sum_{i \in V'_i} (b_i - |\{e \in \delta(i) : \overline{x}_e = 1\}|)$ is odd. Then, the BI defined by $H = V'_i$ and $T = \{e \in \delta(i) : \overline{x}_e = 1\}$ is guaranteed to be violated. Violated BIs, identified in this way, are used to reinforce the current LP relaxation and are also sent to the cut-pool.

Suppose now that a violated BI was found for handle H and teeth T, where $j \notin H$ and $|\delta(j) \cap T| = b_j$. This inequality is dominated by another BI implied when H is replaced by $H \cup \{j\}$ and edges $\delta(j)$ are removed from T. Thus, in this case, only the stronger BI is used.

5.2 Cut-pool rules

In our Branch-and-Cut algorithm, three different families of valid inequalities were used to strengthen LP relaxation bounds. After some computational testing, a hierarchy was established for the separation of these inequalities. This was done taking into account not only their bound improvement capabilities but also their associated computational separation cost. Inequalities were then separated, at every cutting plane round, in the following order: cut-set inequalities, SECs, and BIs. More elaborate and costly separation algorithms were only applied after lower bound improvement was otherwise considered *poor* or else when simpler separation algorithms failed. Hierarchically higher families of inequalities were only investigated after no hierarchically lower violated inequality could be found. The only exceptions to this rule apply to the inequalities in the cut-pool. These were tested for violation, prior to running any separation algorithm, indistinctively of the families they belonged to.

For every node of the enumeration tree, for every LP relaxation solution in hand, we first attempted to find violated inequalities belonging to the cut-pool. If no such inequality was found, or else, if dual bound increase was less than MINIMP, i.e., an empirically established threshold value, we would then resort to separation routines, in the order indicated above. However, these procedures were only actually used if, for at least one of the last MAXITER consecutive iterations, lower bound increase, measured against the immediately previous lower bound, was greater than MINIMP. Otherwise, we would then proceed to branching at the node. An exception to this rule applied when the support graph of $\overline{x}$ was disconnected. In that case, we would resort to the cut-set separation routines, irrespective of other considerations.

Nonbiding cutting plane inequalities were dropped from LP relaxations whenever a lower bound increase of at least MINIMP, measured against the immediately previous lower bound, was observed. Additionally, given that separating cut-set inequalities is computationally cheap, we dropped these inequalities from the cut-pool whenever we decided to branch from the tree node in hand. Contrary to that, SECs and BIs were never dropped from the cut-pool. Following this procedure, the cut-pool was kept reasonably small, without incurring into a high computational cost as a result.

5.3 Some additional implementation details

In an attempt to obtain stronger DCMSTP upper bounds, we also used, within the Branch-and-Cut algorithm, the upper bound procedures in Section 3.2. This was done under complementary costs $\{c_e(1-\overline{x}_e) : e \in E\}$. As before, Local Search, under the original edge costs, was applied right after running the constructive heuristic. Given that the computational demands involved were low, upper bounds were attempted to be improved right after every cutting plane generation round. Some additional features of our Branch-and-Cut algorithm are:

- `XPRESS-MP` release 16 was used with all heuristic and preprocessing keys, except for dual pricing, turned off.
- A depth-search approach, branching on variables under stardard XPRESS-MP rules, was used. For earlier versions of the algorithm, we also experimented with the *best bound* node selection rule. However, no clear advantage over the depth-first search rule was observed.
- After some computational experiments, we settled for MINIMPR = 0.01% and MAXITER = 15.
- A CPU time limit of 4 hours was imposed on the Branch-and-Cut phase of the hybrid algorithm. No CPU time limit was imposed on its NDRC phase.

6 Computational experiments

The hybrid algorithm was implemented in `C` and was tested on four different sets of instances, under the Linux operational system. GNU compiler `gcc` with optimization flag `O3` activated, was used in the experiments. Four test sets, namely, R, ANDIST, DE and DR were considered in our study. For test set R, experiments were performed on a AMD ATHLON 2000 machine with 1 Gb of RAM memory. For all the other test sets, i.e., ANDINST, DE and DR, experiments were conducted on a 1 Gb RAM memory Pentium IV machine, running at 2.4 Ghz.

ANDINST instances were originally proposed in [1]. For these instances, vertices in G are associated with randomly generated points in the Euclidean plane. Corresponding edge costs are taken as the rounded down Euclidean distances between corresponding endpoints (vertices). For every vertex (point), Euclidean plane coordinates are randomly drawn from the uniform distribution, in the range $[1, 1000]$. Additionally, for every vertex $i \in V$, b_i values are randomly drawn from {1,2,3,4}. The number of vertices having $b_i = 1$ was restricted to no more than 10% of the vertices in V. Furthermore, for every ANDINST instance, underlying graphs are always complete and their associated number of vertices is clearly indicated in the name corresponding to the instance. The number of vertices for the ANDINST instances ranged from 100 up to 2000.

For the second test set, DE, instances are also generated as suggested in [1]. Since b_i values for these instances are randomly drawn from the more restricted set $\{1, 2, 3\}$, under uniform probability, they are meant to be harder to solve than the ANDINST instances. Three different DE instances were generated for every $|V| \in \{100, 200, 300, 400, 500, 600\}$.

For the third test set, DR, instances originate from set DE and are obtained after dropping Euclidian costs in favor of random costs drawn from the uniform distribution in the range $[1, 1000]$. All other parameters involved remained unchanged.

Finally, the fourth test set, R, contains random cost k-DCMSTP instances, i.e., DCMST instances where $b_i \leq k$ for all $i \in V$. These are generated exactly as suggested in [7] and have $|V| \in \{100, 500, 800\}$. Set R instances are thus representative of the instances used in [7], where $|V| = 100$ for the smallest dimension instances and $|V| = 800$ for the largest dimension ones. We only quote results for instances with $k = 3$ and $k = 4$ since, by comparison, instances with $k = 5$ and $k = 6$ are far too easy to solve to proven optimality. For every available combination of k and graph density p, where $p \in \{0.05, 0.25, 0.5, 0.75, 1.0\}$, 30 set R instances were generated for every $|V| \in \{100, ..., 800\}$. Edge costs are drawn from the uniform distribution in the range $[1, 1000]$, for all instances considered.

6.1 Computational results

Computational results are presented next, in separate, for each of the two phases of the hybrid algorithm.

6.1.1 NDRC results

For all test sets, with the only exception of the ANDINST instances, computational results for our NDRC algorithm could be found in [12]. That reference also contains a more detailed description of the algorithm. For that reason, in this Section, we will concentrate our analysis on the so far unpublished comparison between the NDRC algorithm and two additional algorithms from the literature. Namely, the VNS procedure recently introduced in [16] and the Lagrangian algorithm in [1]. These two algorithms have been previously tested on the ANDINST instances and also on a set of Hamiltonian path instances, i.e., instances where $b_i = 2, \forall i \in V$. Here we restrict comparisons to the ANDINST instances. This was done because Hamiltonian path instances could be much more effectively dealt with through Travelling Salesman Problem algorithms such as [33, 48]. Finally, we also highlight in this Subsection the benefits a NDRC pre-processing/warm start procedure brings to our DCMSTP Branch-and-Cut algorithm.

For each instance tested, entries in Table 1 respectively give best NDRC lower bound z_n, best NDRC upper bound $\overline{z_n}$, the duality gap implied by these bounds, the CPU time $t(s)$ to attain them, in seconds, and two LP NDRC warm start lower bounds, LB_1 and LB_2. Lower bound LB_1 stands for the LP bound obtained after implementing the NDRC warm start for constraints (2), (4), (5), and the inequalities in I_1. For LB_2, the NDRC warm start was implemented for the same set of constraints as in LB_1, plus inequalities in I_2. Results for the Lagrangian algorithm in [1] and the VNS approach in [16], follow. Accordingly, best Lagrangian lower and upper bounds z_l and $\overline{z_l}$, VNS upper bounds $\overline{z_v}$ and VNS CPU times $t(s)$, in seconds, are sequentially given. Finally, the last entries in Table 1 give the corresponding optimal solution values z obtained by the hybrid algorithm.

Figures in Table 1 indicate that NDRC lower and upper bounds are quite good. Indeed, for only 8 out of 26 instances, rounded up lower bounds did not match optimal solution values. Even for larger instances with up to 2000 vertices, minute duality gaps were obtained, typically in less than one CPU hour. In addition, for all instances tested, warm start LP lower bound LB_2 matched best NDRC lower bound z_n, as one would have hoped for. One should also notice that lower bound LB_2 improved upon lower bound LB_1 for all but one test instance. This indicates that the BIs dualized throughout the application of the NDRC algorithm proved relevant in strengthening LP relaxation lower bounds.

Results in Table 1 confirmed our expectations over the use of BIs. In particular, they indicate that the

Table 1: NDRC comparative results with the literature for set ANDINST.

	Fist phase - NDRC						Lagrangian heuristic in [1]		VNS in [16]		z
	z_n	$\overline{z_n}$	gap (%)	t(s)	LB_1	LB_2	z_l	$\overline{z_l}$	$\overline{z_v}$	t(s)	
100_1	3790.000	3790	-	0.07	-	-	3789.709	3790	3790	8.66	3790
100_2	3829.000	3829	-	0.12	-	-	3829.000	3829	3829	9.34	3829
100_3	3915.417	3916	-	0.29	-	-	3915.118	3916	3916	6.03	3916
200_1	5316.000	5316	-	0.77	-	-	5316.000	5316	5316	65.41	5316
200_2	5646.250	5647	-	2.70	-	-	5645.826	5647	5651	33.25	5647
200_3	5698.000	5698	-	1.79	-	-	5697.289	5698	5703	25.95	5698
300_1	6476.011	6477	-	2.42	-	-	6474.982	6477	6477	198.78	6477
300_2	6804.500	6808	0.05	31.08	6802.500	6804.500	6802.463	6829	6809	91.77	6807
300_3	6429.052	6430	-	4.25	-	-	6429.865	6431	6435	119.77	6430
400_1	7413.627	7414	-	2.33	-	-	7413.968	7416	7414	469.10	7414
400_2	7779.000	7782	0.04	32.07	7775.000	7779.000	7774.779	7797	7793	40.32	7782
400_3	7602.500	7604	0.02	38.42	7601.500	7602.500	7601.189	7609	7614	160.25	7604
500_1	8271.146	8272	-	6.31	-	-	8269.777	8279	8272	802.01	8272
500_2	8391.047	8392	-	18.17	-	-	8391.328	8397	8407	335.18	8392
500_3	8501.250	8502	-	26.38	-	-	8501.562	8502	8506	332.85	8502
600_1	9036.335	9037	-	9.58	-	-	9035.000	9038	9037	619.33	9037
700_1	9785.806	9786	-	13.91	-	-	9752.602	9787	9786	997.99	9786
800_1	10332.276	10333	-	25.45	-	-	10331.113	10341	10333	1761.27	10333
900_1	10917.550	10918	-	29.75	-	-	10917.991	10923	10918	2160.96	10918
1000_1	11407.000	11409	0.02	165.74	11407.000	11407.000	11406.127	11423	11408	2165.89	11408
2000_1	15669.361	15670	-	272.63	-	-	15669.940	15718	15670	10850.35	15670
2000_2	16241.667	16244	0.01	2647.83	16239.000	16241.667	16238.792	16320	16255	12486.61	16242
2000_3	16669.000	16678	0.03	3854.200	16668.000	16669.000	16666.885	16752	16687	13529.58	16670
2000_4	16369.000	16375	0.06	4106.250	16368.000	16369.000	16367.061	16496	16445	2076.86	16370
2000_5	16520.000	16522	0.02	2827.96	16519.500	16520.000	16519.270	16598	16532	11607.89	16521

use of these inequalities, within NDRC, leads to improvements over the Lagrangian bounds in [1]. For 17 of the 26 ANDINST instances (notably, the largest instances in set ANDINST), NDRC upper bounds were stronger than those in [1]. For the remaining instances, same value upper bounds were obtained by the two approaches. Focusing now on the dual bounds, the best lower bounds in [1] are weaker than LB_1, which, in turn, is typically weaker than LB_2.

VNS is also dominated by NDRC and for 12 out of the 25 ANDINST instances, NDRC produced strictly stronger upper bounds. For 12 out of the 13 remaining instances, same vale upper bounds were generated by the two algorithms. Therefore, for only one instance tested, the VNS upper bound came out stronger than its NDRC counterpart. If one now focus on CPU times, quite often NDRC, in a loose comparison, was up to two orders of magnitude faster than VNS. This difference in CPU time could probably be explained by our use of the variable fixation tests in Section 3.3. Given that these tests require primal and dual bounds, they can not be directly implemented in the VNS algorithm.

NDRC average results, for every different test set used, are presented in Table 2. Sets are clearly identified in the first column of that table. For the remaining columns, average results are only presented for those instances NDRC, alone, could not prove optimality. Column two presents NDRC duality gaps, in percentage values. Columns three and four respectively give the ratios $\frac{LB_1}{z_n}$ and $\frac{LB_2}{z_n}$. Results in Table 2 suggest, as one might have expected, that the smaller the bound degree b_i is, the harder the instance is for NDRC to solve it. That applies, irrespective of the costs being Euclidean or not. As it could be appreciated from that table, LB_2 bounds always come out at least as strong as their target values, z_n.

Table 2: Summary of NDRC bounds

Test set	gap (%)	$\frac{LB_1}{z_n}$(%)	$\frac{LB_2}{z_n}$(%)
R	< 0.01	100.000	100.000
ANDINST	0.03	99.985	100.015
DE	0.94	99.944	100.112
DR	1.52	99.992	100.008

Detailed NDRC results for instances in each of the test sets considered, are presented in Tables 5-7. Entries for these tables have a similar meaning to those in Table 1. For Tables 5-7, we also additionally present the computational results attained by our Branch-and-cut algorithm. These results are discussed next.

6.1.2 Branch-and-cut results

In order to better evaluate potential benefits arising from the use of NDRC as a pre-processor to Branch-and-Cut, four different versions of the Branch-and-Cut algorithm were implemented. They are respectively denoted by BC1, BC2, BC3 and BC4. For any of these algorithms, input data is given by a graph that has been pre-processed by NDRC. Differences between the algorithms originate from the different classes of inequalities used in each case. They are also implied by the use or not of the proposed NDRC warm start. Table 3 indicates the main characteristics of each Branch-and-Cut algorithm variant considered while Table 4 summarizes the average computational results obtained by them.

For the simplest Branch-and-Cut variant, BC1, we only separate cut-set inequalities and SECs. Likewise, for this variant, the cut-pool is not initialized with the inequalities in I_1 and I_2, i.e., the inequalities carried

Table 3: Branch-and-cut variants

	BC1	BC2	BC3	BC4
NDRC preprocessing	yes	yes	yes	yes
Separation of SEC and cutsets	yes	yes	yes	yes
Heuristic separation of Blossom inequalities	no	no	yes	yes
Warm start - greedy SEC	no	yes	no	yes
Warm start - Blossom	no	no	no	yes

Table 4: Branch-and-cut results

	BC1		BC2		BC3		BC4	
	Nodes	t(s)	Nodes	t(s)	Nodes	t(s)	Nodes	t(s)
R	11	67.75	7	15.18	11	67.96	7	15.58
DE	548	1033.67	528	555.58	107	485.56	74	271.00
DR	112	1476.13	78	919.15	78	1312.31	71	741.54
ANDINST	58	248.52	42	217.68	9	58.91	6	44.90

over from NDRC. Algorithm BC2 differs from BC1 in that the cut-pool is initialized with the inequalities in I_1. Algorithm BC3 differs from BC1 in that it incorporates the separation of BIs. Finally, the most general Branch-and-Cut algorithm tested, BC4, separates all classes of valid inequalities discussed in this study. Additionally, for BC4, the cut-pool is initialized with the inequalities in I_1 and I_2. Before moving on to comparisons, it should be pointed out that, to some extent, all Branch-and-Cut algorithms quoted above benefit from the use of BIs at NDRC (stronger Lagrangian bounds and more effective variable fixation tests).

Benefits arising from the use of the proposed warm start and also from the separation of BIs, could be evaluated after comparing BC1 and BC4. Algorithm BC4 may be seen as a stronger version of the Branch-

and-Cut algorithm proposed in [7]. This applies since the latter algorithm is based on the standard DCMST formulation (1) and neither uses BIs, at its corresponding Lagrangian phase, nor it implements a warm start for its initial LP relaxation. Furthermore, in our view, a direct comparison of the Branch-and-Cut algorithm in [7] with BC4 would be of limited use since the algorithm in [7] was only tested for k-DCMSTP instances. Aditionally, our NDRC algorithm alone, with little computational effort, managed to solve to proven optimality all but 13 of the almost 900 k-DCMSTP instances in set R, generated exactly as suggested in [7] (see [12] for detailed results). Bearing in mind the issues raised above, for our purposes, comparing BC4 with BC1 would imply, in our view, a fair comparison between BC4 and the algorithm in [7].

Table 4 only presents Branch-and-Cut results for those test instances Relax-and-Cut alone could not find proven optimal solutions. For every Branch-and-Cut variant tested, the CPU times quoted do not include the time spent at the Relax-and-Cut phase of the hybrid algorithm. For every individual test set, average number of enumeration tree nodes and average CPU time (in seconds) are quoted in Table 4.

A comparison between BC2 and BC1 indicates that, for test set R, implementing a warm start that only carries over greedy SECs to Branch-and-Cut (and no BIs) manages to reduce the average CPU times, over the same algorithm without the warm start, by a factor of more than 4. Likewise, it reduced the number of enumeration tree nodes by about 30%. For some of these instances, BC2 was up to 10 times faster than BC1. For the ANDINST test set, BC2 CPU time figures were about 13% lower than corresponding figures for BC1. Similarly, for test sets DR and DE, significant reductions in CPU times, respectively by 47% and 38%, were also attained.

Algorithms BC3 and BC4, that separate SECs and BIs, managed to solve all 15 DE instances within the imposed CPU time limit. Notice that contrary to BC3, where the cutpool is initially empty, for BC4 the cutpool is initialized with I_1 and I_2. Differently from BC3 and BC4, algorithms BC1 and BC2 failed to solve instances de_500_3, de_600_1 and de_600_2, within the imposed time limit. Accordingly, the average CPU time for instances DE, quoted in in Table 4, only include those instances that were solved to proven optimality, within the specified CPU time limit, by all Branch-and-Cut variants tested. As it could be appreciated from these results, BC4 turned out to be 3.8 times faster than BC1. This reduction in CPU time was attained from the combined use of BIs (notice that BC3 is already 2 times faster than BC1) and warm start (BC4 is 1.8 times faster than BC3). Likewise, the number of enumeration tree nodes was significantly reduced, by nearly 5 times, through the use of BIs. For test set R, however, CPU times for BC4 and BC3 resulted quite similar to those quoted for BC2 and BC1. Therefore, for set R, the use of BIs did not bring any CPU time advantage. For test set DR, BC4 was about 2.8 times faster than BC1. In addition, the separation of BIs allowed CPU times to be cut down by about 12.5%. Comparisons between pairs BC2-BC1 and BC4-BC3, indicate that the use of warm start BIs do contribute to a CPU time cut. However, it appears clear that the initialization of the cutpool with greedy set SECs was the main responsible for the overall warm start gains. Finally, for those test sets where LB_2 dominated LB_1, notably for ANDINST and DE, the use of BIs, either through the separation heuristic or else directly from set I_2, significantly contributed to cut down CPU times.

Based on our empirical results, one could come out with the following general suggestion for solving DCM-STP instances. Provided the instances have bound degrees larger or equal to 3, no significant improvements should be expected from using BIs. The standard DCMST formulation (1) should then be used instead of the BI reinforced one. However, as bound degrees become tighter, benefits from using the reinforced formulation are likely to pay off. Separating BIs in a cutting plane algorithm would thus be advantageous. Finally, irrespective of the type of instance considered, the use of the warm start procedure suggested here helps cutting down CPU times.

As far as test sets are concerned, from the computational results in Tables 5-7, instances in test set DE appear to be the hardest to solve to proven optimality while the ANDINST instances appear to be the easiest ones.

We close this Section by remarking that the NDRC pre-processing phase of the hybrid algorithm appears highly important for the overall performance of the algorithm. This assertion is backed by a comparison between the CPU times quoted for the cutting plane algorithm in [11] and those quoted here for the hybrid algorithm. Indeed, for instances in test set DE, CPU times taken for only computing the BI reinforced LP bounds for formulation (1) was 2 to 3 times larger than that required by the hybrid algorithm to solve these instances to proven optimality.

Table 5: Results for set ANDINST

	NDRC Results						BC4 Results		NDRC + BC4	
	z_n	$\overline{z_n}$	gap (%)	t(s)	LB_1	LB_2	BB nodes	t(s)	z	t(s)
100_1	3790.000	3790	-	0.07	-	-	-	-	3790	0.07
100_2	3829.000	3829	-	0.12	-	-	-	-	3829	0.12
100_3	3915.417	3916	-	0.29	-	-	-	-	3916	0.29
200_1	5316.000	5316	-	0.77	-	-	-	-	5316	0.77
200_2	5646.250	5647	-	2.70	-	-	-	-	5647	2.70
200_3	5698.000	5698	-	1.79	-	-	-	-	5698	1.79
300_1	6476.011	6477	-	2.42	-	-	-	-	6477	2.42
300_2	6804.500	6808	0.05	31.08	6802.500	6804.500	4	0.18	6807	31.26
300_3	6429.052	6430	-	4.25	-	-	-	-	6430	4.25
400_1	7413.627	7414	-	2.33	-	-	-	-	7414	2.33
400_2	7779.000	7782	0.04	32.07	7775.500	7779.000	3	0.40	7782	32.47
400_3	7602.500	7604	0.02	38.42	7601.500	7602.500	3	0.15	7604	38.57
500_1	8271.146	8272	-	6.31	-	-	-	-	8272	6.31
500_2	8391.047	8392	-	18.17	-	-	-	-	8392	18.17
500_3	8501.250	8502	-	26.38	-	-	-	-	8502	26.38
600_1	9036.335	9037	-	9.58	-	-	-	-	9037	9.58
700_1	9785.806	9786	-	13.91	-	-	-	-	9786	13.91
800_1	10332.276	10333	-	25.45	-	-	-	-	103330	25.45
900_1	10917.550	10918	-	29.75	-	-	-	-	10918	29.75
1000_1	11407.000	11409	0.02	165.74	11407.000	11407.000	3	3.00	11408	168.74
2000_1	15669.361	15670	-	272.63	-	-	-	-	15670	272.63
2000_2	16241.667	16244	0.01	2647.83	16239.000	16241.667	10	37.56	16242	2712.39
2000_3	16669.000	16678	0.03	3854.200	16668.000	16669.000	10	282.35	16670	4136.55
2000_4	16369.000	16375	0.06	4106.250	16368.000	16369.000	5	125.75	16370	4232.00
2000_5	16520.000	16522	0.02	2827.96	16519.500	16520.000	6	49.88	16521	2877.84

7 Conclusions

In this paper, a hybrid exact solution algorithm was proposed for the Degree-Constrained Minimum Spanning Tree Problem. The algorithm involves two combined phases. Namely, a Lagrangian Non Delayed Relax-and-Cut phase and a Branch-and-Cut one. The Lagrangian phase acts as a pre-processor and a warm starter to Branch-and-Cut.

Our hybrid algorithm incorporates some ingredients that were not previously used for DCMSTP. For instance, the standard DCMSTP formulation was strengthened with Blossom Inequalities. Furthermore,

Table 6: Results for set DE

	NDRC Results						BC4 Results		NDRC + BC4	
	z_n	$\overline{z_n}$	gap (%)	t(s)	LB_1	LB_2	BB nodes	t(s)	z	t(s)
de_100_1	8779.000	8779	-	5.56	-	-	-	-	8779	5.56
de_100_2	9629.000	9629	-	4.14	-	-	-	-	9629	4.14
de_100_3	10783.499	10846	0.58	18.04	10783.500	10783.500	10	0.92	10798	18.96
de_200_1	12028.998	12046	0.14	54.02	12022.000	12029.000	3	0.43	12031	54.45
de_200_2	12616.751	12752	1.07	92.24	12611.500	12617.875	407	101.08	12645	193.52
de_200_3	12209.617	12292	0.67	95.61	12208.500	12210.500	37	6.18	12229	101.79
de_300_1	14769.225	14902	0.89	212.43	14758.500	14770.000	75	48.13	14792	260.56
de_300_2	13920.747	13931	0.07	88.79	13906.000	13921.000	1	0.60	13931	89.39
de_300_3	14940.565	15093	1.00	208.81	14920.500	14947.000	25	110.49	14997	310.30
de_400_1	19232.153	19388	0.81	900.47	19212.250	19234.500	1	88.71	19239	989.18
de_400_2	17384.443	17620	1.36	839.27	17373.750	17386.750	282	740.74	17419	1580.01
de_400_3	16071.866	16100	0.18	342.77	16062.500	16072.000	43	241.80	16091	584.57
de_500_1	19344.868	19416	0.37	754.60	19332.167	19346.250	14	40.37	19357	794.97
de_500_2	22389.598	22749	1.58	2888.06	22389.500	22395.000	8	670.51	22400	3558.57
de_500_3	19373.352	19743	1.87	1711.62	19365.500	19380.250	628	6100.26	19411	7811.88
de_600_1	21906.379	22248	1.54	2556.79	21900.370	21907.333	315	5207.16	21930	7763.95
de_600_2	21408.757	21679	1.25	2622.46	21390.375	21409.675	419	9343.67	21449	11966.13
de_600_3	22221.693	22610	1.71	3632.08	22217.000	22226.000	28	1671.44	22243	5303.52

the resulting strengthened formulation was used not only in Branch-and-Cut but in Non Delayed Relax-and-Cut as well. Additionally, inequalities brought over from Lagrangian relaxation were used to warm start the Branch-and-Cut algorithm. In doing so, without the need of solving any separation problem, initial Linear Programming relaxation bounds were always at least as good as best Lagrangian relaxation bounds. Additionally, as demonstrated in extensive computational testing, significant CPU time speed ups were attained over algorithms that do not use the ingredients considered here. As a result, instances larger than previously attempted in the literature were solved to proven optimality. Likewise, 5 new optimality certificates were obtained for test instances from the literature.

Out of our computational experience, the use of Blossom Inequalities for DCMSTP appears attractive for instances with tight upper bounds on vertex degrees. Otherwise, for example, for k-DCMSTP instances with large values of k, the impact of their use appears very marginal. In any case, irrespective of the type of instance tested, warm starting Branch-and-Cut, as described here, proved very effective in reducing CPU times.

Finally, we would like to apologize to Matteo Fischetti, Juan José Salazar and Paolo Toth for not previously giving due credit to them for the warm start procedure described in [12]. The procedure we

Table 7: Results for set DR

	NDRC Results						BC4 Results		NDRC + BC4	
	z_n	$\overline{z_n}$	gap (%)	t(s)	LB_1	LB_2	BB nodes	t(s)	z	t
dr_100_1	2176.731	2197	0.92	8.79	2174.500	2178.000	8	0.20	2186	8.79
dr_100_2	2673.971	2674	-	6.04	-	-	-	-	2674	6.04
dr_100_3	2688.963	2689	-	8.34	-	-	-	-	2689	8.34
dr_200_1	2138.999	2148	0.42	55.92	2139.000	2139.000	31	5.20	2141	61.12
dr_200_2	2469.847	2504	1.36	73.43	2469.851	2470.000	7	1.00	2471	74.43
dr_200_3	2402.656	2409	0.26	58.99	2402.750	2402.750	25	3.89	2405	62.88
dr_300_1	2459.933	2476	0.65	180.11	2460.000	2460.000	11	9.70	2462	189.81
dr_300_2	2311.972	2312	-	107.09	2312.000	2312.000	-	-	2312	107.09
dr_300_3	2583.862	2621	1.42	276.14	2583.700	2583.889	143	147.18	2585	423.32
dr_400_1	2814.097	2934	4.09	920.99	2813.500	2814.219	105	474.89	2817	1395.88
dr_400_2	2440.337	2478	1.52	588.41	2440.357	2440.818	49	125.16	2443	713.57
dr_400_3	2386.948	2387	-	161.26	2387.000	2387.000	-	-	2387	161.26
dr_500_1	2595.587	2604	0.32	726.65	2595.625	2595.625	47	280.44	2597	1007.09
dr_500_2	2650.000	2847	6.92	4616.21	2650.000	2650.000	40	617.02	2652	5233.23
dr_500_3	2591.712	2607	0.59	824.06	2591.776	2591.776	37	274.21	2593	1098.27
dr_600_1	2819.456	2823	0.13	1603.73	2819.500	2819.500	21	557.18	2820	2160.91
dr_600_2	2659.647	2699	1.46	2076.92	2659.667	2659.667	97	6179.38	2662	8256.30
dr_600_3	2798.296	2832	1.19	1590.49	2798.408	2798.408	367	1706.08	2800	3296.57

suggested in [12] is identical to the one suggested in [21] for a Travelling Salesman Problem. However, only after the publication of [12], we became unaware of the work in [21].

References

[1] R. Andrade, A. Lucena, and N. Maculan. “Using Lagrangian dual information to generate degree constrained spanning trees”. *Discrete Applied Mathematics*, 154(5):703–717, 2006.

[2] L. Bahiense, N. Maculan, and C. Sagastizábal. “The volume algorithm revisited: relation with bundle methods”. *Mathematical Programming*, 94:41–70, 2002.

[3] F. Barahona and R. Anbil. “The volume algorithm: producing primal solutions with subgradient method”. *Mathematical Programming*, 87:385–399, 2000.

[4] A. Belloni and A. Lucena. “A Relax and Cut Algorithm for the Traveling Salesman Problem”. In *17th International Symposium on Mathematical Programming*, 2000.

[5] A. Belloni and A. Lucena. “A Lagrangian Heuristic for the Linear Ordering Problem”. In M.G.C. Resende and J. Pinho de Sousa, editors, *Metaheuristics: Computer Decision-Making*, pages 123–151. Kluwer Academic, Norwell, MA, 2003.

[6] A. Belloni and C. Sagastizábal. Dynamic Bundle Methods. Technical Report A 2004/296, Instituto de Matemática Pura e Aplicada, IMPA, Rio de Janeiro, 2004.

[7] L. Caccetta and S.P. Hill. “A Branch and cut Method for the Degree-Constrained Minimum Spanning Tree Problem”. *Networks*, 37(2):74–83, 2001.

[8] F. Calheiros, A. Lucena, and C. de Souza. “Optimal Rectangular Partitions”. *Networks*, 41:51–67, 2003.

[9] V.F. Cavalcante, C.C. de Souza, and A. Lucena. “A Relax-and-Cut algorithm for the set partitioning problem”. *Computers and Operations Research*, 35(6):1963–1981, 2008.

[10] A.S. Cunha. *Árvores ótimas em grafos: modelos, algoritmos e aplicações*. PhD thesis, Programa de Engenharia de Sistemas e Computação, Universidade Federal do Rio de Janeiro, 2006.

[11] A.S. Cunha and A. Lucena. “lower and upper bounds for the degree-constrained minimal spanning tree problem”. In L. Gouveia and C. Mour ao, editors, *INOC 2005 Proceedings*, volume 1, pages 186–194, Lisbon, March 2005. Faculty of Science, University of Lisbon.

[12] A.S. Cunha and A. Lucena. “Lower and Upper Bounds for the Degree-Constrained Minimal Spanning Tree Problem”. *Networks*, 50:55–66, 2007.

[13] J.B.C. da Silva. Uma Heuristica Lagrangeana para o Problema da Árvore Capacitada de Custo Mínimo. Master’s thesis, Programa de Engenharia de Sistemas e Computação, COPPE, Universidade Federal do Rio de Janeiro, 2002.

[14] G.B. Dantzig, D.R. Fulkerson, and S.M. Johnson. “Solution of a large scale traveling salesman problem”. *Operations Research*, 2:393–410, 1954.

[15] M. de Moraes Palmeira, A. Lucena, and O. Porto. A relax and cut algorithm for the quadratic knapsack problem. Technical report, Departamento de Administração, Universidade Federal do Rio de Janeiro, 1999.

[16] M.C. de Souza and P. Martins. " Skewed VNS Enclosing Second Order Algorithm for the Degree Constrained Minimum Spanning Tree Problem". *European Journal of Operational Research*, 2007. accepted for publication.

[17] C. Duin. *Steiner's Problem in Graphs.* PhD thesis, University of Amsterdam, 1993.

[18] J. Edmonds. "Matroids and the Greedy Algorithm". *Mathematical Programming*, 1:127–136, 1971.

[19] J. Edmonds and E.L. Johnson. "Matching: A Well-Solved Class of Integer Linear Programs". In R.K. Guy et al., editor, *Proceedings of the Calgary International Conference on Combinatorial Structures and their Applications*, pages 89–92. Gordon and Breach, 1970.

[20] L. Escudero, M. Guignard, and K. Malik. "A Lagrangian relax and cut approach for the sequential ordering with precedence constraints". *Annals of Operations Research*, 50:219–237, 1994.

[21] M. Fischetti and J.J.S. González. " A Branch-anbd-cut algorithm for the Symmetric Generalized Traveling Salesman Problem". *Operations Research*, 45, 1997.

[22] H.N. Gabow. "Two Algorithms for Generating Weighted Spanning Trees in Order". *Networks*, 8:201–208, 1978.

[23] A.B. Gamble and W.R. Pulleyblank. "Forest covers and a polyhedral intersection theorem". *Mathematical Programming*, 45:49–58, 1989.

[24] B. Gavish. "Topological design of Centralized Computer Networks - formulations and algorithms". *Networks*, 12:355–377, 1982.

[25] F. Glover and D. Klingman. "Recent developments in computer implementation technology for network flow algorithms". *INFOR*, 20:433–452, 1982.

[26] L. Gouveia and P. Moura. "Models for the Degree-Constrained Minimum Spanning Tree Problem with Node-Degree Dependent Costs". Technical report, 2006.

[27] M. Grotschel and M. Padberg. "On the Symmetric Traveling Salesman Problem I: Inequalities". *Mathematical Programming*, 16:265–280, 1979.

[28] M. Grotschel and W.R. Pulleyblank. "Clique Tree Inequalities and the Symmetric Traveling Salesman Problem". *Mathematics of Operations Research*, 11:537–569, 1986.

[29] L. Hall and T.L. Magnanti. "A polyhedral intersection theorem for capacitated spanning trees". *Mathematics of Operations Research*, 17(2):398–410, 1982.

[30] J. Hao and J. B. Orlin. “A Faster Algorithm for Finding the Minimum Cut in a Directed Graph”. *J. Algorithms*, 17:424–446, 1994.

[31] M.H. Held, P. Wolfe, and H.D Crowder. “Validation of Subgradient Optimization”. *Mathematical Programming*, 6:62–88, 1974.

[32] M. Hunting, U. Faigle, and W. Kern. “A Lagrangian relaxation approach to the edge-weighted clique problem”. *European Journal of Operational Research*, 131 (1):119–131, 2001.

[33] M. Jnger, G. Reinelt, and G. Rinaldi. “The Traveling Salesman Problem”. In M.O. Ball, editor, *Handbooks in OR and MS, Vol. 7*, pages 225–330. Elsevier Science Publishers, 1995.

[34] N. Katoh, T. Ibaraki, and H. Mine. “An Algorithm for Finding K Minimum Spanning trees”. *SIAM Journal on Computing*, 10:247–255, 1981.

[35] T. Koch and A. Martin. “Solving Steiner Tree Problems in Graphs to Optimality”. *Networks*, 32:207–232, 1998.

[36] J.B. Kruskal. “On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem”. *Proceedings of the American Mathematical Society*, 7:48–50, 1956.

[37] A. Letchford, G. Reinelt, and D.O. Theis. “A faster exact separation algorithm for blossom inequalities”. In G.L. Nemhauser and D. Bienstock, editors, *Integer Programming and Combinatorial Optimization 10. Lecture Notes in Computer Science*, volume 3064. Springer Verlag, Berlin, 2004.

[38] A. Lucena. Tight bounds for the Steiner problem in graphs. Talk given at the TIMS-XXX - SOBRAPO XXIII Joint International Meeting, Rio de Janeiro, July 15-17 1991.

[39] A. Lucena. “Steiner Problem in Graphs: Lagrangean Relaxation and Cutting Planes”. *COAL Bulletin*, 21:2–8, 1992.

[40] A. Lucena. Tight bounds for the Steiner problem in graphs. Technical Report TR-21/93., Dipartimento di Informatica. Univesità degli Studi di Pisa, Pisa, Italy, 1993.

[41] A. Lucena. “Non Delayed Relax-and-Cut Algorithms”. *Annals of Operations Research*, 140:375–410, 2005.

[42] 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, Amsterdam, 1995.

[43] C. Martinhon, A. Lucena, and N. Maculan. “Stronger K-tree relaxations for the vehicle routing problem”. *European Journal of Operational Research*, 158:56–71, 2004.

[44] G.L. Nemhauser and L. Wolsey. *Integer and Combinatorial Optimization.* Wiley Interscience, 1988.

[45] M. Padberg and S. Hong. “On the symmetric traveling salesman problem: A computational study”. *Mathematical Programming Study*, 12:78–107, 1980.

[46] M. Padberg and M.R. Rao. “Odd minimum cut-sets and b-matchings”. *Mathematics of Operations Research*, 7:67–80, 1982.

[47] M. Padberg and L. Wolsey. “Trees and cuts”. *Annals of Discrete Mathematics*, 17:511–517, 1983.

[48] M.W. Padberg and G. Rinaldi. “A Branch-and-Cut algorithm for resolution of large scale of Symmetric Traveling Salesman Problem”. *SIAM Review*, 33:60–100, 1991.

[49] C.C. Ribeiro and M.C. de Souza. “Tabu Search for the Steiner Problem in Graphs”. *Networks*, 36(2):138–146, 2000.

[50] C.C. Ribeiro and M.C. de Souza. “Variable Neighborhood Search for the Degree-Constrained Minimum Spanning Tree Problem”. *Discrete Applied Mathematics*, 118:43–54, 2002.

[51] M. Savelsbergh and A. Volgenant. “Edge exchanges in the degree constrained minimum spanning tree problem”. *Computers and Operations Research*, 12(4):341–348, 1985.

[52] A. Volgenant. “A Lagrangean approach to the degree-constrained minimum spanning tree problem”. *European Journal of Operational Research*, 39:325–331, 1989.

[53] A. Volgenant and R. Jonker. “The symmetric traveling salesman problem and edge exchanges in minimal 1-trees”. *European Journal of Operational Research*, 12:394–403, 1983.