A RELAX-AND-CUT ALGORITHM FOR THE PRIZE-COLLECTING STEINER PROBLEM IN GRAPHS

A.S. DA CUNHA, A. LUCENA, N. MACULAN, AND M.G.C. RESENDE

Abstract. Given an undirected graph G with penalties associated with its vertices and costs associated with its edges, a Prize Collecting Steiner (PCS) tree is either an isolated vertex of G or else any tree of G, be it spanning or not. The weight of a PCS tree equals the sum of the costs for its edges plus the sum of the penalties for the vertices of G not spanned by the PCS tree. Accordingly, the Prize Collecting Steiner Problem in Graphs (PCSPG) is to find a PCS tree with the lowest weight. In this paper, after reformulating and re-interpreting a given PCSPG formulation, we use a Lagrangian Non Delayed Relax and Cut (NDRC) algorithm to generate primal and dual bounds to the problem. The algorithm was capable of adequately dealing with the exponentially many candidate inequalities to dualize. It incorporates ingredients such as a new PCSPG reduction test, an effective Lagrangian heuristic and a modification in the NDRC framework that allowed duality gaps to be further reduced. The Lagrangian heuristic suggested here dominates their PCSPG counterparts in the literature. The NDRC PCSPG lower bounds, most of the time, nearly matched corresponding Linear Programming relaxation bounds.

1. Introduction

Given an undirected graph $G = (V, E)$ with a set V of vertices and a set E of edges, associate real-valued *penalties* $\{d_i \geq 0 : \forall i \in V\}$ with the vertices of G and real-valued *costs* $\{c_e \geq 0 : \forall e \in E\}$ with its edges. A *Prize Collecting Steiner* (PCS) *tree* is either an isolated vertex of G or else any tree of G, be it spanning or not. The *weight* of a PCS tree equals the sum of the costs for its edges plus the sum of the penalties for the vertices of G not spanned by the PCS tree. The Prize Collecting Steiner Problem in Graphs (PCSPG) is to find a PCS tree with the lowest weight.

PCSPG is closely related with both the Steiner Problem in Graphs (SPG) [3] and the Steiner Tree Problem in Graphs (STP) [16]. For the first of these problems, two or more *terminal vertices* must be connected together while minimizing the sum of the costs for the edges used. As such, feasible SPG solutions are implied by connected components of the problem input graph containing all terminal vertices. However, if one restricts the edges of the input graph to only assume nonnegative costs, as it is the case for every single SPG instance found in the literature, an additional *artificial* constraint is implicitly imposed on the problem. That is, provided the instance is feasible, an optimal SPG solution exits with the topology of a tree

Date: January 18, 2007. Revised January 16, 2008.

Key words and phrases. Lagrangian relaxation, non-delayed relax and cut, prize-collecting Steiner problem in graphs.

AT&T Labs Research Technical Report TD-7AWSRU.

spanning all terminal vertices plus, possibly, some nonterminal *Steiner vertices* as well. One should notice, however, that this kind of topology is just a particular case of the more general structures the definition of the problem allows for. Contrary to SPG, for STP a tree topology is explicitly required for feasible solutions and this is the only difference between the two problems. Furthermore, it is normally assumed, either for SPG or STP, that edge costs are unrestricted in sign. However, when these costs are limited to be strictly positive, an optimal solution to STP must also be optimal to SPG, and vice versa.

In the literature, SPG and STP are frequently mixed up. It is not uncommon to find SPG formulations being referred to as STP formulations and vice-versa. For PCSPG, it appears that we are fast following the same track. Indeed, in tune with the discussion above, it would certainly help if one distinguishes PCSPG from a *Prize Collecting Steiner Tree Problem in Graphs* (PCSTP). In this paper we will use a PCSPG formulation that explicitly enforces a PCS tree topology on feasible solutions. Furthermore, it originates from a STP formulation. Thus, it would probably be more adequate to call that formulation a PCSTP formulation, although, in this paper, we refrain from doing so.

SPG instances with nonnegative edge costs may be recast as PCSPGs. This is attained by associating a *sufficiently large* positive penalty to every terminal vertex and zero valued penalties to nonterminals. Thus, optimal PCS trees are guaranteed to contain all terminal vertices, as required in SPG. Given that SPG was proven to be NP-hard in [19], no matter if the edge costs of the input graph are nonnegative or not, and since, under nonnegative edge costs, SPG is a particular case of PCSPG, then PCSPG must also be NP-hard.

For the past few years, PCSPG has been the focus of considerable attention in the literature. A possible explanation for that, apart from the inherent interest the problem attracts, is the fact that *Prize-Collecting* appears to be a realistic model for various applications in network design. An example is mentioned in [7, 18, 27] where one wants to build a fiber-optics network to provide broadband connections to business and residential customers. As such, for that application, G represents a local street map, V corresponds to potential client premises and street intersections, and E represents street segments. Another application, reported in [21], deals with the planning of the expansion of electricity and gas facilities. For that application, vertex penalties are discounted cash flows associated with the estimated heating demands to be supplied. Edges costs are associated with laying (or expanding) pipes and/or electrical cables.

The term *Prize-Collecting* was coined in Balas [1], in the context of the Traveling Salesman Problem. Since then it is has been used to define various combinatorial optimization problems. A common tract for all these problems is a clear trade-off between paying an edge cost to include a vertex into a feasible solution or else incurring a penalty for leaving it out.

References to additional problems closely related to PCSPG go back to Segev [33]. In that reference, *Node-Weighted Steiner Tree Problem* (NWSTP) is used to describe a variant of SPG where, in addition to the usual edge costs, vertex penalties are also present. Solution algorithms for a single terminal vertex version of NWSTP are proposed and tested in [33].

Another exact solution algorithm for NWSTP is proposed in [13]. This algorithm is based on a formulation which is similar to the SPG formulation found in [23]. In

turn, the SPG formulation in [23] could be seen as a strengthened version of the SPG formulation in [4]. Likewise, it could also be seen as a reformulation of the SPG formulation suggested in [14, 22, 28] (see [26] or [25], for details). Additionally, NWSTP lower bounds in [13] are generated as suggested in [23]. Thus, like the algorithm in [23], the Lagrangian relaxation algorithm in [13] could be seen as a Non Delayed Relax-and-Cut algorithm (see [25] for details).

PCSPG, in the minimization form described here, could be polynomially approximated [6, 15]. In Bienstock [6], an algorithm with an approximation factor of 3 was proposed for the problem. Later on, Goemans and Williamson [15] proposed a general primal-dual approximation framework for a family of NP-hard problems, PCSPG included. Specializing that framework to PCSPG resulted in an algorithm with an approximation factor of 2 and complexity $O(|V|^2\log(|V|))$. The algorithm is geared to a rooted version of PCSPG, where a pre-specified vertex is required to be part of any feasible solution. Thus, for the general case, where no root vertex exists, the algorithm requires $|V|$ runs, with every vertex in V playing, in turn, the role of root. In doing so, the approximation factor of 2 is maintained, albeit at a $O(|V|^3\log(|V|)$ complexity. More recently, Minkoff [29] improved upon the Goemans and Williamson algorithm (GWA) by dropping the root vertex requirement, preserving the asymptotic approximation factor, and reducing the worst case complexity to $O(|V|^2\log(|V|))$. An additional contribution found in [29] is the development of a *pruning*, or post processing strategy, based on Dynamic Programming. This strategy dominates the one suggested in [15].

Lucena and Resende [27] specialized the STP formulation in [14, 23, 28] to PCSPG and used a cutting plane algorithm to generate Linear Programming (LP) relaxation bounds. The algorithm was tested on instances from the literature and proved to be capable of returning *strong* PCSPG lower bounds. Still in [27], some pre-processing tests, adapted from the SPG literature, were used to reduce instance input size.

Another PCSPG formulation is found in Ljubic et al. [21]. It relies on a digraph induced by G and involves an additional, artificial, root vertex. A branch-and-cut algorithm based on this formulation was implemented in [21]. The algorithm underwent extensive computational testing and proved to be *very effective*. Indeed, it managed to solve to proven optimality all test instances from the literature. Additionally, optimality certificates were also obtained for some new *hard-to-solve* instances, proposed in [21].

Recently, Haouari and Siala [17] studied the *Quota* PCSPG (QPCSPG). For that variant of PCSPG, instead of vertex penalties, vertex *prizes* are used. Accordingly, for any feasible QPCSPG solution, solution vertices qualify for a prize. Furthermore, for a tree of G to be feasible to QPCSPG it is required to collect at least a given minimum amount in prizes. As such, QPCSPG is more in tune than PCSPG with the idea of *prize collecting*, as originally introduced in [1]. A hybrid solution approach for QPCSPG, involving a genetic algorithm and Lagrangian relaxation, was proposed in [17]. The approach was tested on instances with up to 500 vertices and 5000 edges.

A metaheuristic based heuristic was suggested to PCSPG in Canuto et. al. [7]. It combines a multi-start implementation of GWA (operating under randomly perturbed input costs), Local Search (LS), Variable Neighborhood Search (VNS), and Path-Relinking (PR). The extensive computational experiments in [7] indicate that

the proposed heuristic is capable of attaining near optimal solutions in *acceptable* CPU times.

In this paper, we investigate primal and dual bounds for PCSPG. These originate from Lagrangian relaxations to the PCSPG formulation in [27]. Since, at first sight, that formulation does not seem amenable to be decomposed in a Lagrangian fashion, we reformulate it with a new set of variables. The resulting formulation is then given a more convenient graph theoretical interpretation, allowing it to be easily decomposed in a Lagrangian fashion. In doing so, a Non Delayed Relax-and-Cut (NDRC) algorithm [25] is then applied. In association, an effective new rule for discarding *inactive* dualized inequalities is also proposed and tested here. Additionally, operating under the proposed NDRC framework, a Lagrangian heuristic is implemented to find PCSPG feasible solutions. One of the features of this heuristic is the use of Lagrangian dual information to generate feasible integral solutions to PCSPG. It also uses Local Search to attempt to improve the feasible solutions thus obtained. These combined ingredients are repeatedly used throughout the Relax-and-Cut algorithm. Preprocessing and variable fixing tests, that proved effective in reducing instance input size, are also used in our NDRC algorithm.

This paper is organized as follows. In Section 2, the PCSPG formulation in [27] is reformulated and re-interpreted to make it amenable to the use of Lagrangian relaxation. In Section 3, the basic features of a general NDRC algorithm are described. In Section 4, that general framework is tailored to PCSPG. Reduction tests for the problem and the pricing out of suboptimal variables are discussed in Section 5. In Section 6, the NDRC algorithm is computationally tested. Finally, the paper is closed in Section 7 with some conclusions and directions for future work.

2. An Integer Programming Formulation for PCSPG

The PCSPG formulation used in this paper was suggested in [27] and involves two different sets of variables. Namely, variables $\{y_i \in \{0,1\} : i \in V\}$ to select the vertices to appear in the PCS tree and variables $\{x_e \geq 0 : e \in E\}$ to connect these vertices. We denote by $E(S) \subseteq E$ the set of edges with both endpoints in $S \subseteq V$. Accordingly, $x(E(S)) := \sum_{e \in E(S)} x_e$ represents the sum of the variables associated with the edges in $E(S)$. Likewise, $y(S) := \sum_{i \in S} y_i$ represents the sum of the variables associated with the vertices in S. Using this notation, the PCSPG formulation in [27] is then given by

$$\min \left\{ \sum_{e \in E} c_e x_e + \sum_{v \in V} d_v (1 - y_v) : (x, y) \in \mathcal{R}_0 \cap (\mathbb{R}_+^{|E|}, \mathbb{B}^{|V|}) \right\}, \tag{1}$$

where $\mathbb{B}^{|V|}$ stands for $\{0,1\}^{|V|}$ and the polyhedral region $\mathcal{R}_0$ is defined as

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

$$x(E(S)) \leq y(S \setminus \{j\}), \forall j \in S, \forall S \subseteq V, \tag{3}$$

$$0 \leq x_e \leq 1, \forall e \in E, \tag{4}$$

$$0 \leq y_i \leq 1, \forall i \in V. \tag{5}$$

For the formulation above, for any feasible solution, constraint (2) imposes that the number of edges involved must equal the number of vertices minus one, very much as one would expect from a PCS tree. Constraints (3) generalize the Subtour Elimination Constraints (SECs) of Dantzig, Fulkerson, and Johnson [10] and guarantee that the resulting solution is cycle free. Finally, inequalities (4) and (5) define valid lower and upper bounds for the variables involved. Thus, after introducing necessary integrality constraints on the y variables, it then follows that the set of feasible solutions to (1) imply all PCS trees of G.

Clearly, single vertex solutions to PCSPG could be efficiently computed through explicit enumeration. Bearing that in mind, Lucena and Resende [27] only concentrated on feasible PCSPG solutions involving one or more edges. Such a restricted version of the problem follows from (1) and is given by

$$\min \left\{ \sum_{e \in E} c_e x_e + \sum_{v \in V} d_v (1 - y_v) : (x, y) \in \mathcal{R}_1 \cap (\mathbb{R}_+^{|E|}, \mathbb{B}^{|V|}) \right\}, \tag{6}$$

where polyhedral region $\mathcal{R}_1$ is defined by the set of constraints in $\mathcal{R}_0$ plus

$$x(\delta(i)) \geq y_i, \forall i \in V \text{ with } d_i > 0, \tag{7}$$

and

$$x(\delta(i)) \geq 2y_i, \forall i \in V \text{ with } d_i = 0. \tag{8}$$

Indeed, to exclude single vertex solutions from (1), it suffices to append inequalities

$$x(\delta(i)) \geq y_i, \forall i \in V \tag{9}$$

to (1). However, given the nonnegative edge costs and vertex penalties in PCSPG, we use the stronger inequalities (7), for positive penalty vertices. It should be noticed that (8) explicitly imposes that no zero penalty vertex may be a leaf in an optimal PCS tree. Validity of this condition follows from the fact that a PCS tree of lower weight would otherwise be obtained after eliminating leaves for zero valued penalty vertices (thus contradicting any optimality assumption).

2.1. **Exchanging variables and uncovering structure.** Typically, for the use of Lagrangian relaxation, one looks for an *easy to solve* problem, obtained after dropping a set of *complicating* constraints from the formulation in hand. Ideally, such an easy problem should be capable of returning, for our specific application, a *good quality* bound on (6). In principle, $\mathcal{R}_1$ does not appear to contain a structure meeting these requirements. However, as we shall see next, such a structure is actually hidden in $\mathcal{R}_1$ and could be uncovered by following a two-step procedure. Firstly, binary $0-1$ variables $\{y_i : i \in V\}$ should be replaced by their complements in 1. Then, the new variables should be re-interpreted in terms of a graph which expands $G = (V, E)$ with the introduction of an artificial vertex together with some edges incident to that vertex.

To implement the first of the two steps indicated above, let $\{z_i = 1 - y_i : i \in V\}$ be the set of variables to replace $\{y_i : i \in V\}$ in (6). Exchanging variables results in a polyhedral region $\mathcal{R}_2$, in a one-to-one correspondence with $\mathcal{R}_1$, given by

$$x(E) + z(V) = |V| - 1, \tag{10}$$

$$x(\delta(i)) + z_i \geq 1, \forall i \in V \text{ with } d_i > 0, \tag{11}$$

$$x(\delta(i)) + 2z_i \geq 2, \forall i \in V \text{ with } d_i = 0, \tag{12}$$

$$x(E(S)) + z(S \setminus \{j\}) \leq |S| - 1, \forall j \in S, \forall S \subseteq V, \tag{13}$$

$$0 \leq x_e \leq 1, \forall e \in E, \tag{14}$$

$$0 \leq z_i \leq 1, \forall i \in V. \tag{15}$$

Re-written as above, GSECs (13) now appear very clearly as a lifting of ordinary SECs. A reformulation of (6) is thus given by

$$\min \left\{ \sum_{e \in E} c_e x_e + \sum_{i \in V} d_i z_i : (x, z) \in \mathcal{R}_2 \cap (\mathbb{R}_+^{|E|}, \mathbb{B}^{|V|}) \right\}. \tag{16}$$

At this point, let us give an alternative interpretation to the meaning of variables $\{z_i : i \in V\}$. Assume that an artificial vertex $(n+1)$, where $n = |V|$, has been introduced into $G = (V, E)$ and that every variable z_i, $i \in V$, represents an edge of cost d_i directly linking i to $(n+1)$. Denoting by $G' = (V', E')$ the graph that results from this expansion of G, then $V' = V \cup \{n+1\}$ and $E' = E \cup \{(i, n+1) : i \in V\}$.

Notice that $|V'| - 2$ (or, alternatively, $|V| - 1$) edges of G' must appear at any feasible solution to (10)–(15). Notice as well that such a solution must violate no GSECs. It is thus not difficult to check that any feasible solution to (10)-(15) corresponds to a certain GSEC restricted spanning forest of G' with exactly two connected components. One of these components must either be a *star* centered at vertex $(n+1)$ (i.e., a set of one or more edges of G', all incident to vertex $(n+1)$) or else vertex $(n+1)$ in isolation. GSECs $x_e + z_i \leq 1$ and $x_e + z_j \leq 1$, defined for a set $S = \{i, j\} \in V$, where $e = (i, j) \in E$, imply the topology of the first component. The other component must be a PCS tree involving at least one edge of G and having no vertex $i \in V$ with $d_i = 0$ as a leaf (as enforced by inequalities (11) and (12)). A typical example of such a solution is depicted in Figure 1.

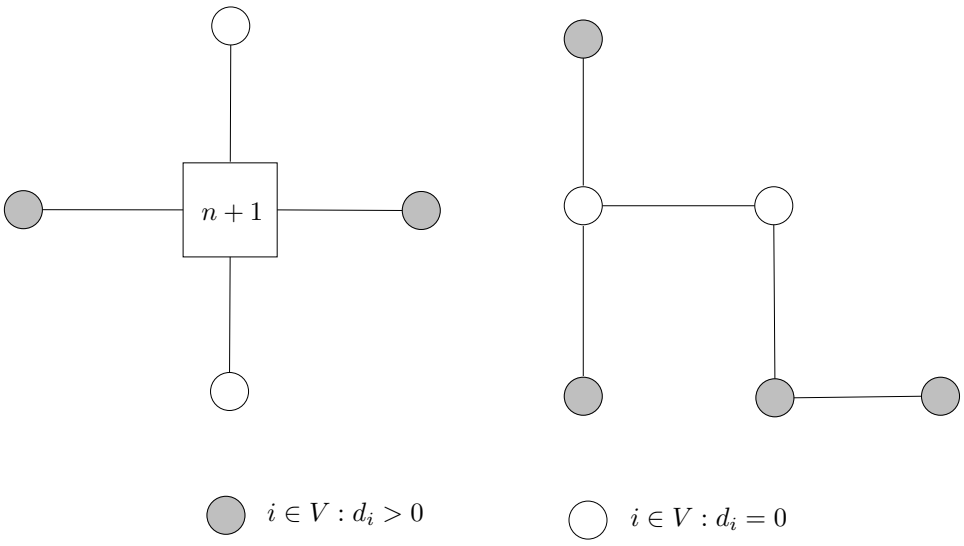

FIGURE 1. A feasible PCSPG solution under the expanded graph G'.

Within a Lagrangian framework, the remarks above provide us with an attractive structure to work with: an unrestricted spanning forest of G' with exactly $|V'| - 2$

edges. Provided such a spanning forest does not violate degree constraints (11) and (12) nor GSECs (13), it must imply a feasible PCSPG solution involving two or more edges. In accordance with that, consider now a polyhedral region $\mathcal{R}_3$ given by

$$x(E) + z(V) = |V'| - 2 \tag{17}$$

$$x(E(S)) \leq |S| - 1, \forall S \subseteq V' \tag{18}$$

$$0 \leq x_e \leq 1, \forall e \in E \tag{19}$$

$$0 \leq z_i \leq 1, \forall i \in V \tag{20}$$

where (18) are ordinary SECs. A forest of G' with exactly $|V'| - 2$ edges must then be associated with any point in $\mathcal{R}_3$ where $z \in \mathbb{B}^{|V|}$. Additionally, if such a point does not violate (11), (12) and (13), it must imply a PCS tree, i.e., the second of the two structures discussed above. Therefore, if one attaches nonnegative multipliers to inequalities (11), (12) and (13) and brings them to the objective function in (16), optimizing the resulting Lagrangian modified objective function over $(x, z) \in \mathcal{R}_3 \cap (\mathbb{R}_+^{|E|}, \mathbb{B}^{|V|})$ returns a valid PCSPG lower bound.

Since exponentially many inequalities exist in (13), dualizing them in a Lagrangian fashion is not as straightforward as it would otherwise be for (11) and (12). Thus, in Section 3, a description is given of NDRC algorithms. As mentioned before, NDRC allows one to deal with the nonstandard Lagrangian relaxation application suggested above.

3. Non Delayed Relax and Cut

The NDRC algorithm in [23, 24] is based upon the use of Subgradient Method (SM) and, throughout this paper, we follow [23, 24, 25] in using SM to describe and test NDRC. The material presented in this section essentially follows from [25].

Assume that a formulation for a NP-hard combinatorial optimization problem is given. Assume as well that exponentially many inequalities are involved in it. Such a formulation is generically described as

$$w = \min\{cx : \quad Ax \leq b, \quad x \in X\}, \tag{21}$$

where, for simplicity, x denotes binary $0-1$ variables, i.e., $x \in \mathbb{B}^p$ for an integral valued $p > 0$. Accordingly, for an integral valued $m > 0$, we have $c \in \mathbb{R}^p$, $b \in \mathbb{R}^m$, $A \in \mathbb{R}^{m \times p}$ and $X \subseteq \mathbb{B}^p$. Assume, as it is customary for Lagrangian relaxation, that

$$\min\{cx : \quad x \in X\} \tag{22}$$

is an *easy to solve* problem. On the other hand, in what is unusual for the application of Lagrangian relaxation, assume that m is an exponential function of p, i.e. (21) contains exponentially many inequalities. Assume as well that one dualizes

$$\{a_i x \leq b_i : \quad i = 1, 2, \ldots, m\} \tag{23}$$

in a Lagrangian fashion, regardless of the difficulties associated with the dualization of exponentially many inequalities. Denote by $\lambda \in \mathbb{R}_+^m$ the corresponding vector of Lagrangian multipliers. A valid lower bound on (21) is thus obtained by solving the *Lagrangian Relaxation Problem* ($LRP(\lambda)$)

$$w_\lambda = \min\{(c + \lambda A)x - \lambda b : \quad x \in X\}. \tag{24}$$

To attain the best possible Lagrangian bound (24), Subgradient Optimization (SO) could be used to solve the corresponding *Lagrangian Dual Problem* (LDP)

$$w_d = \max_{\lambda \in \mathbb{R}^m_+} \{w_\lambda\}. \tag{25}$$

Optimization is typically conducted here in an interactive way with multipliers being updated so that w_d is obtained. For the sake of completeness, let us briefly review SM, as implemented in [5]. That implementation is precisely the one adapted in this paper to produce the computational results in Section 6.

3.1. **A brief description of the Subgradient Method.** At iteration k of SM, for a feasible vector λ^k of Lagrangian multipliers, let $\overline{x}^k$ be an optimal solution to $LRP(\lambda^k)$, with value w_{λ^k}, and $\overline{w}$ be a known upper bound on (21). Additionally, let $g^k \in \mathbb{R}^m$ be a subgradient associated with the relaxed constraints at $\overline{x}$. Corresponding entries for g^k are

$$g_i^k = (b_i - a_i \overline{x}^k), \quad i = 1, 2, \ldots, m. \tag{26}$$

In the literature (see [5], for instance), to update Lagrangian multipliers, one initially generates a *step size*

$$\theta^k = \frac{\alpha[\overline{w} - w_{\lambda^k}]}{\sum\limits_{i=1,\ldots,m} (g_i^k)^2}, \tag{27}$$

where $\overline{w}$ is a valid upper bound on w and α is a real number assuming values in $(0, 2]$. Having done that, one then updates multipliers as

$$\lambda_i^{k+1} \equiv \max\{0; \lambda_i^k - \theta^k g_i^k\}, \quad i = 1, \ldots, m \tag{28}$$

and moves on to iteration $k + 1$ of SM.

Under the conditions imposed here, the straightforward use of updating formulas (27)–(28) is not as simple as it might appear. The reason being the exceedingly large number of inequalities that one would typically have to deal with.

3.2. **NDRC modifications to the Subgradient Method.** Inequalities in (23), at iteration k of SM, may be classified into three sets. The first contains inequalities that are violated at $\overline{x}^k$. The second is for those inequalities that have nonzero multipliers currently associated with them. Notice that an inequality may simultaneously be in the two sets just defined. Finally, the third set contains the remaining inequalities. Following [25], we will refer to the three sets of inequalities just described respectively as the *Currently Violated Active* set, the *Previously Violated Active* set, and the *Currently Inactive* set. Accordingly, they are respectively denoted by $CA(k)$, $PA(k)$, and $CI(k)$.

For the traditional use of Lagrangian relaxation, say when m is a polynomial function of p, Beasley [5] reported *good practical convergence* of SM to (25), while, arbitrarily setting $g_i^k = 0$ whenever $g_i^k > 0$ and $\lambda_i^k = 0$, for $i \in \{1, \ldots, m\}$. In our context, all subgradient entries that are candidates to that modification belong to $CI(k)$.

In spite of the exponentially many inequalities one is faced with, we follow Beasley's advice. The reasoning for doing that comes from two observations. The first one is that, irrespective of the suggested changes, from (28), multipliers for $CI(k)$ inequalities would not change their present null values at the end of the

current SM iteration. As such, at the current SM iteration, clearly, $CI(k)$ inequalities would not directly contribute to Lagrangian costs. On the other hand, they would play a decisive role in determining the value of θ^k and this fact brings us to the second observation. Typically, for the application being described, the number of strictly positive subgradient entries associated with $CI(k)$ inequalities, tends to be huge. If they are all explicitly used in (27), θ^k would result extremely small, leaving multiplier values virtually unchanged from iteration to iteration and SM convergence problems should be expected.

By following Beasley's suggestion, we are capable of dealing adequately, within a SM framework, with the exceedingly large number of inequalities in $CI(k)$. However, we may still face problems arising from a potentially *large* number of inequalities in $(CA(k) \setminus PA(k))$. These, as one may recall, are the inequalities that will become *effectively* dualized, i.e., that will have a nonzero multiplier associated with them at the end of SM iteration k.

Assume now that a *large* number of inequalities exist in $(CA(k) \setminus PA(k))$. These inequalities must therefore be violated at the solution to $LRP(\lambda^k)$ and have zero valued Lagrangian multipliers currently associated with them. Typically, such inequalities may be partitioned into subsets associated, for instance, with a partitioning of the set of vertices in a given associated graph, if that applies. Then, according to some associated criteria, a maximal inequality would exist for each of these subsets. In order to avoid repeatedly penalizing the same variables, again and again, we only dualize one maximal inequality per subset of inequalities. Excluding these inequalities, remaining inequalities in $(CA(k) \setminus PA(k))$ will have their subgradient entries arbitrarily set to 0, thus becoming, in effect, $CI(k)$ inequalities.

One should notice that, under the classification proposed above, inequalities may change groups from one SM iteration to another. It should also be noticed that the only multipliers that will have directly contributed to Lagrangian costs $(c+\lambda^{k+1}A)$, at the end of SM iteration k, are the ones associated with *active* inequalities, i.e., inequalities in $(CA(k) \cup PA(k))$.

An important step in the dynamic scheme outlined above, is the identification of inequalities violated at $\overline{x}^k$. This problem must be solved at every iteration of SM and is equivalent to the separation problems found in Branch-and-Cut algorithms. However, NDRC separation problems typically involve lower complexity algorithms than their Branch-and-Cut counterparts. This follows from the fact that $LRP(\lambda)$ is normally formulated so that separation is conducted over integral structures.

3.3. **Extending the life of dynamically dualized inequalities.** For the NDRC algorithm outlined above, assume that a given inequality is dynamically dualized at iteration k of SM. Accordingly, assume, as previously suggested, that this inequality is dropped from updating formula (27) as soon as it becomes inactive. This would specifically occur at the very first SM iteration $k_1 > k$ for which, simultaneously, the inequality is not violated at the solution $\overline{x}^{k_1}$ to $LRP(\lambda^{k_1})$ and its corresponding Lagrangian multiplier drops to zero.

For the computational experiments in this study, we tested an alternative to the rule above. Namely, we *extend the life* or, better say, the use of a dualized inactive inequality in updating formula (27), past iteration k_1. Accordingly, denote the *age* of a dynamically dualized inequality, the number of consecutive SM iterations past k_1 where the inequality is not violated by corresponding $LRP(\lambda^k)$ solutions. Under this new rule, such an inequality remains dualized and is allowed to be used in (27)

for as long as its age is less than a given parameter EXTRA ≥ 1. Clearly, in doing so, whenever an inequality aged over 1 is violated at a $LRP(\lambda^{k_2})$ solution $\overline{x}^{k_2}$, where $(k_2 - k_1) \leq$ EXTRA, the inequality will leave *probation* and enter set $CA(k_2)$. As a result, no need would exist, for the time being, to keep track of it. Obviously, this situation would prevail until eventually the inequality becomes, yet again, aged 1 and monitoring of it becomes, once more, mandatory.

The use of the alternative rule proposed above proved quite effective. In fact, for some of the instances tested, gaps between NDRC upper and lower bounds were closed by as much as 30%, after setting EXTRA to a value larger than 1.

4. NDRC bounds to PCSPG

We implemented a NDRC algorithm to PCSPG where GSECs (13) are dynamically dualized, as suggested in the previous section. Degree-inequalities (11) and (12), however, which are small in number, were dualized in a traditional Lagrangian fashion. Thus, these inequalities remain dualized throughout SM, irrespective of being active or not.

At iteration k of SM, for a conformable value $q > 0$, assume that Lagrangian multipliers $\lambda^k \in \mathbb{R}^q_+$ are associated with the dualized inequalities. Following the discussion in Section 3, a valid lower bound on (16) is thus given by the solution to $LRP(\lambda^k)$, formulated as

$$(29)\quad w_{\lambda^k} = \min \left\{ \sum_{e \in E} c^k_e x_e + \sum_{i \in V} d^k_i z_i + \text{const}(\lambda^k) : (x, z) \in \mathcal{R}_3 \cap (\mathbb{Z}^{|E|}, \mathbb{B}^{|V|}) \right\},$$

where $\{c^k_e : e \in E\}$ and $\{d^k_i : i \in V\}$ are respectively Lagrangian modified edge costs and vertex penalties and const(λ^k) is a constant implied by λ^k.

Notice that an optimal solution to (29), i.e. a minimum cost forest of G' with exactly $(|V'|-2)$ edges, is easy to obtain. To do so, among the alternatives available, it suffices to adapt Kruskal's algorithm [20] to stop immediately after the first $(|V'| - 2)$ edges are selected.

Let us now concentrate on solutions $(\overline{x}^k, \overline{z}^k)$ to $LRP(\lambda^k)$, which are infeasible to PCSPG. Each of these solutions gives rise to a *support graph*, that is a subgraph of G' induced by the nonzero entries in $(\overline{x}^k, \overline{z}^k)$. A typical support graph is depicted in Figure 2. From previous arguments, it is clear that feasible solutions to (29) that happen to be feasible to (16) as well, must either contain vertex $(n + 1)$ appearing in isolation or else have that vertex as the center of a star, as previously defined. For the solution in Figure 2, vertex $i \in V$ contains two edges incident to it, namely $(i, n+1)$ and (i, l). Thus, that solution must be infeasible to PCSPG and, as such, must imply violated GSECs. Examples of such inequalities are the four maximal GSECs induced by the set of five encircled vertices in the figure. For each of these inequalities, in turn, a different vertex $j \in S \setminus \{i\}$ is singled out in (13).

For a given optimal $LRP(\lambda^k)$ solution $(\overline{x}^k, \overline{z}^k)$, the identification of maximal violated GSECs could be carried out efficiently. This is attained by investigating the support graph associated with $(\overline{x}^k, \overline{z}^k)$. In doing so, assume that edge $(i, n+1)$, for $i \in V$, is contained in that graph. In addition, denote by S_i the set of support graph vertices, i itself included, that could be reached from i without crossing edge $(i, n + 1)$. Whenever $|S_i|$ is larger or equal to 2, violated GSECs must necessarily be associated with S_i. Vertices in S_i could be identified in $O(n)$ time. To do so it

suffices to eliminate $(i, n+1)$ from the support graph and conveniently adapt any available shortest path algorithm to enumerate all vertices reachable from i.

After some computational experiments, it proved advantageous to dualize, in a traditional Lagrangian fashion, in addition to (11) and (12), all GSECs with $|S| = 2$. Only $2|E|$ such inequalities exist and, among all GSECs available, they are the ones that contribute the most to the Lagrangian bound. Apart from these simple GSECs, we only dualize maximal GSECs associated with sets S_i of cardinality larger than 2. Furthermore, from our experiments, for S_i as just described, it proved advantageous to only dualize one out of the $|S_i| - 1$ corresponding maximal violated GSECs. We thus only dualize that inequality in (13) for which $S = S_i$ and vertex $j \in S_i$ is chosen as $j = \arg\{\max_{l \in (S \setminus \{j\})}\{d_l^k\}\}$. Ties are broken arbitrarily.

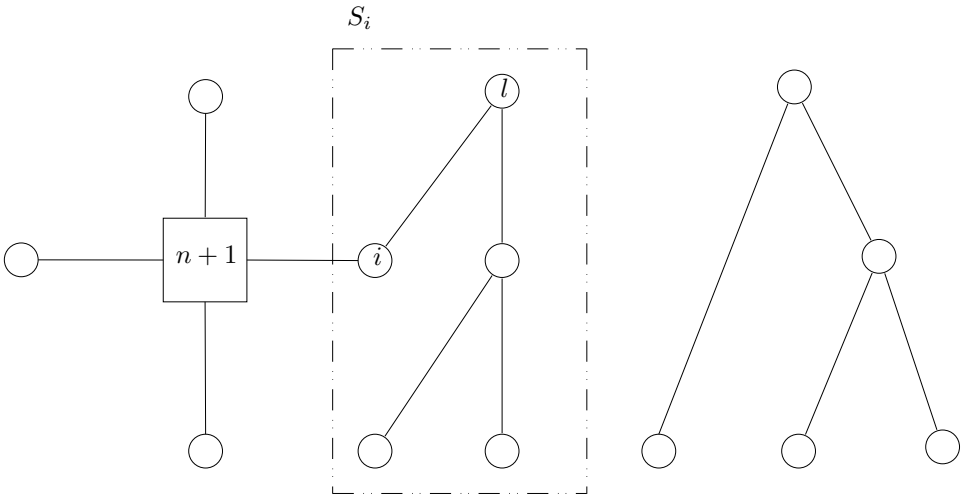

FIGURE 2. A $(\overline{x}^k, \overline{z}^k)$ solution infeasible to (16).

4.1. **PCSPG upper bounds.** Our upper bounding strategy attempts to use Lagrangian dual information in a procedure to generate feasible integral solutions to PCSPG. The basic motivation behind this approach is the intuitive idea, validated by primal-dual algorithms, that dual solutions (respectively Lagrangian dual solutions, for this application) must carry relevant information for generating good quality primal solutions. Operating within a Lagrangian relaxation framework, our implementation of this basic idea involves two main components. The first is an algorithm based on the variant of GWA proposed in [29]. We call it the *Minkoff Algorithm* (MA). The second is a Local Search (LS) procedure that attempts to improve feasible PCSPG solutions returned by MA.

GWA and MA are primal-dual based factor of 2 approximation schemes for PCSPG. They both rely on a constructive algorithm (where a forest of G is greedily built) followed by *pruning* (where one attempts to construct a PCS tree from the available forest components). One difference between GWA and MA is that, for the former, a given pre-specified root vertex $r \in V$ must be passed as an input data to the constructive algorithm. Furthermore, only that connected component containing r may be subjected to pruning in GWA. As a result, $|V|$ runs are required for GWA to attain a factor of 2 approximation. Contrary to that, the

constructive phase in MA, called `UnrootedGrowthPhase`, involves no root vertex. Additionally, all connected components resulting from it may be submitted to pruning. Due to these features, MA has a better run time complexity than GWA, i.e., $O(n^2\log(n))$ against $O(n^3\log(n))$. Furthermore, the pruning algorithm used in MA, called `BestSubTree`, is based on Dynamic Programming and dominates the corresponding algorithm in GW (`BestSubTree` returns the best possible PCS tree for the subgraph of G induced by the connected component under inspection).

Due to the advantages quoted above, MA was selected to be used within our NDRC framework. Essentially, at an iteration k of SM, *complementary costs* $\{(1-\overline{x}_e^k)c_e : e \in E\}$ and *complementary penalties* $\{\overline{z}_i^k d_i : i \in V\}$ are computed and used as input data to `UnrootedGrowthPhase`, instead of the original edge costs and vertex penalties. In doing so, one attempts to make it more attractive for MA to select as many edges in the support graph of $(\overline{x}^k, \overline{z}^k)$ as possible. Accordingly, we thus use dual information to guide the construction of primal feasible solutions. This overall Lagrangian heuristic, MA included, is only run for SM iterations where w_{λ^k} improves upon the best Lagrangian relaxation bound previously generated at SM. For every such run, pruning algorithm `BestSubTree` is then used under the original edge costs and vertex penalties (instead of using corresponding complementary costs and penalties, as for the constructive algorithm). Solutions thus obtained are then subject to Local Search, which is explained next.

4.2. **Local search.** Given a PCS tree $\mathcal{T} = (V_\mathcal{T}, E_\mathcal{T})$, we are essentially interested in comparing $\mathcal{T}$ with PCS trees that result from $\mathcal{T}$ after a single vertex inclusion or vertex exclusion operation. For the first operation, say the inclusion of vertex $i \in V$ into $\mathcal{T}$, the most effective procedure to accomplish that task may involve the insertion into $\mathcal{T}$ of some additional vertices. Accordingly, the same applies for the operation of excluding a vertex $j \in V_\mathcal{T}$ from $\mathcal{T}$. The search neighborhood we aim for is thus formed by those PCS trees that result from an *optimized* inclusion (resp. exclusion) of a vertex into (resp. from) $\mathcal{T}$.

On implementing the Local Search (LS) procedure suggested above, a vertex inclusion (resp. exclusion) move is conducted in two steps. First, a minimal cost tree spanning the enlarged (resp. contracted) vertex set is computed. Then, at a second step, `BestSubTree` is applied to the resulting PCS tree. Only after this second step is carried out, one may evaluate potential vertex inclusion (resp. exclusion) benefits. Another key feature of our LS procedure is the effort to keep run time low and allow LS to be applied whenever MA is used. In order to do so, several dominance tests are performed to hopefully avoid having to evaluate every possible non-profitable move. The time required by the overall scheme is bounded from above by the time to compute, from scratch, a Minimum Spanning Tree (MST) of G, i.e. $O(m\log(n))$. To avoid paying that price, we implemented the dominance tests used in the SPG Tabu Search heuristic of Ribeiro and Souza [30]. As we shall see next, these tests could be easily adapted to PCSPG.

4.3. **Dominance tests.** Given a PCS tree $\mathcal{T}$, consider all different PCS subtrees contained in it. Clearly, the weight of each of these subtrees may differ from that of $\mathcal{T}$. Throughout our upper bounding algorithm, however, we enforce that no PCS tree $\mathcal{T}$ contains a subtree with a lower weight. We call that the *Optimality Condition* (OC) for $\mathcal{T}$ over the graph induced by $\mathcal{T}$ itself. As such, if $\mathcal{T}$ is passed to `BestSubTree` as an input, $\mathcal{T}$ itself must be returned as an output.

Let us first concentrate on dominance tests for insertion moves. Consider a vertex $i \notin V_{\mathcal{T}}$ and the set of edges connecting i to $V_{\mathcal{T}}$, i.e., $\delta_{\mathcal{T}}(i) := \{e \in E : e \in \delta(i) \cap \delta(V_{\mathcal{T}})\}$. Assume that the edges in $\delta_{\mathcal{T}}(i)$, i.e. $\{e_1, e_2, ..., e_{|\delta_{\mathcal{T}}(i)|}\}$, are ordered in nondecreasing value of their edge costs. The following tests are then used to evaluate the benefits of inserting vertex $i \in V$ into $\mathcal{T}$:

(1) If $|\delta_{\mathcal{T}}(i)| = 0$, no PCS tree exists spanning $V_{\mathcal{T}} \cup \{i\}$.
(2) If $|\delta_{\mathcal{T}}(i)| = 1$, inserting i into $\mathcal{T}$ is profitable iff $d_i > c_{e_1}$. In this case, since the original tree satisfies OC, the new one must also satisfy that condition.
(3) If $|\delta_{\mathcal{T}}(i)| = 2$, two cases must be considered:
   (a) If $d_i > c_{e_1}$, inserting i into $\mathcal{T}$ is profitable and the resulting tree satisfies OC.
   (b) If $d_i \leq c_{e_1}$, inserting i into $\mathcal{T}$ might still be profitable. To see why, let us investigate two examples. In the first one, let us assume that $e_1 = (i, k), e_2 = (i, j)$ and that f is the maximum cost edge in the unique path in $\mathcal{T}$ connecting k and j. It is clear that whenever $d_i + c_f - c_{e_1} - c_{e_2} > 0$ the insertion of i is profitable. Now, look at the tree $\mathcal{T}$ indicated in Figure 3. Note that if we remove the path $\mathcal{P} = \{(k, z_1), (z_1, z_2), (z_2, z_3)\}$ (as well as all its internal nodes) and add edges (i, k) and (i, j) to $\mathcal{T}$, the resulting structure is a cost improving tree. Following Duin [12], we call both the edge f in the first example and the path $\mathcal{P}$ in the second as a *key-path* between j and k. More precisely, a key-path between two vertices j and k in a tree $\mathcal{T}$ is either an edge in the unique path connecting them or, else, any subpath between k and j such that all its internal nodes have degree two. Let us now define the *net weight* of a key-path as the sum of its edges costs minus the sum of the penalties for its internal vertices. For example, the net weight of $\mathcal{P}$ in Figure 3 is $w(P) = c_{(k,z_1)} + c_{(z_1,z_2)} + c_{(z_2,z_3)} - d_{z_2} = 14$. Thus, all we have to do when evaluating the inclusion of i is to find a maximal-weight key-path $\mathcal{P}^*$ between j and k and check whether $\mathcal{P}^*$ is profitable, i.e., $d_i + w(\mathcal{P}^*) - c_{e_1} - c_{e_2} > 0$. In positive case, as the current PCS tree satisfies OC, the tree obtained after removing $\mathcal{P}^*$ and including e_1, e_2 also does. Thus, the new (cost-improving) tree replaces the previous one and the search goes on. To find a maximal weight key-path, we implemented a Dynamic Programming procedure that runs at $O(|V_{\mathcal{T}}|)$ time.
(4) If $|\delta_{\mathcal{T}}(i)| \geq 3$, one should first introduce edge $e_1 = (i, k)$ into $\mathcal{T}$. Denote by $\mathcal{T}_i$ the resulting PCS tree. In the sequel, one should add another edge $e_p = (i, j) \in \delta_{\mathcal{T}}(i) \backslash \{e_1\}$ into $\mathcal{T}_i$. Then one finds the largest cost edge f in the unique path of $\mathcal{T}_i$ connecting i and j and compute the gain $d_i + c_f - c_{e_1} - c_{e_p}$. After evaluating, in turn, the gain provided by the inclusion in $\mathcal{T}_i$ of every edge $e_p \in \{e_2, ..., e_{|\delta_{\mathcal{T}}(i)|}\}$, as described above, the least cost PCS tree thus obtained should be submitted to `BestSubTree`, no matter if its gain is positive or not. Denote by $\mathcal{T}_{\delta(i)}$ the PCS tree thus obtained. Provided $\mathcal{T}_{\delta(i)}$ has less weight than $\mathcal{T}$, $\mathcal{T}_{\delta(i)}$ should then be re-labelled $\mathcal{T}$.

Our LS procedure is initiated with the insertion moves described above. In case they fail, exclusion moves, which are computationally more expensive, should then be attempted. However, prior to describing exclusion moves, we will first describe some dominance tests associated with them.

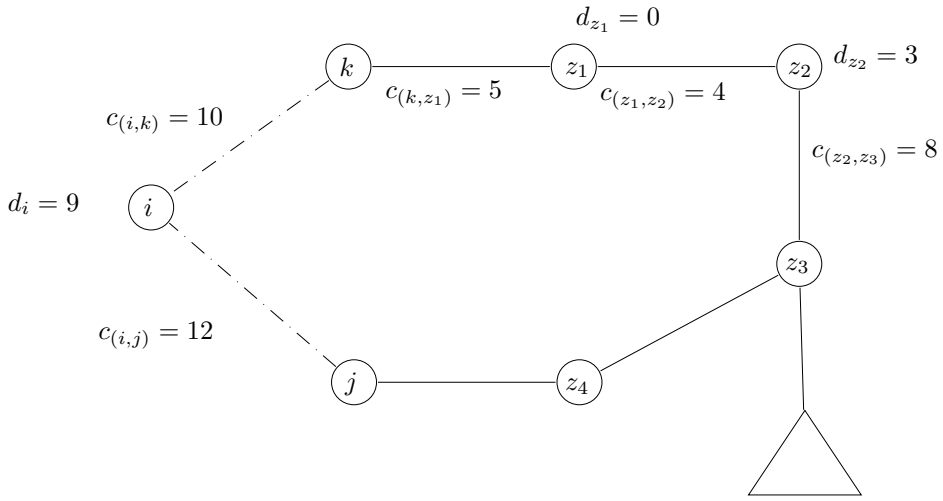

FIGURE 3. An example of a cost improving key-path involving more than one edge.

First of all, let us focus on the analysis carried out in [30] for the exclusion of a Steiner vertex, say vertex j, from a Steiner tree $\mathcal{S} = (V_{\mathcal{S}}, E_{\mathcal{S}})$. In connection with that, assume that $\{(i_1, j), ..., (i_k, j)\}$ are the $k \geq 1$ edges of $\mathcal{S}$ incident to j. Accordingly, if vertex j is removed from $\mathcal{S}$, k trees denoted by $\{\mathcal{S}_l : l = 1, \ldots, k\}$ would result. Let i_l, i_v be any pair of adjacent vertices to j in the current Steiner tree. Assume that j is indeed removed from $\mathcal{S}$ and define (p, q) as the minimum weight edge among all those connecting $\mathcal{S}_l$ to the other $k - 2$ trees $\mathcal{S}_t : t \neq l$, as defined above. It has been proved, see Theorem 1 in [30], that $\eta(j) := c_{(p,q)} - c_{(i_l,j)} - c_{(i_v,j)}$ gives a lower bound on the additional cost of a Steiner tree obtained after eliminating vertex j from $\mathcal{S}$. Clearly, whenever $\eta(j) \geq 0$, the cost of such a Steiner tree is larger than that of $\mathcal{S}$.

It is quite straightforward to adapt the result above to PCSPG. To that order, consider a PCS tree $\mathcal{T}$, a vertex $j \in V_{\mathcal{T}}$, and the corresponding associated trees $\{\mathcal{T}_l : l = 1, \ldots, k\}$, obtained after eliminating j from $\mathcal{T}$. Additionally, redefine $\eta(j)$ as $c_{(p,q)} - c_{(i_l,j)} - c_{(i_v,j)} + d_j$, where, once again, vertices i_l and i_v are any two adjacent vertices to j in $\mathcal{T}$ and (p, q) is the least cost edge connecting $\mathcal{T}_l$ to the other $k - 2$ trees $\mathcal{T}_t : t \neq l$. In doing so, $\eta(j)$ now gives a lower bound on the additional weight of a PCS tree obtained after eliminating vertex j from $\mathcal{T}$. Based on the arguments above, prior to embarking on an analysis to attempt to exclude vertex j from $\mathcal{T}$, one should first determine $\eta(j)$. To do that, one may consider the same choices suggested in [30] for vertices i_l and i_v. Accordingly, i_v must be chosen as the immediate predecessor of j in the path of $\mathcal{T}$ that links j to the root of that tree. Likewise, i_l is taken as the end vertex, other than i_v, of the maximum cost edge of $\mathcal{T}$ incident to j.

5. Reduction tests and the pricing out of suboptimal variables

Prior to using the PCSPG NDRC lower and upper bound procedures in Section 4, a few tests are applied to reduce problem input size. Additionally, throughout

SM, tests which attempt to price out suboptimal edges are also used. Details of these two different types of tests are presented next.

5.1. **Reduction tests.** A reduction test for PCSPG attempts to find vertices and edges of G that are guaranteed not to be in any optimal solution to the problem. The tests that follow were adapted from SPG reduction tests found in the literature (see Duin [11], for instance). In the order they are presented, the first four tests come from [27], the fifth was suggested in [21] while the last is introduced in this study.

5.1.1. *Shortest path test.* The test is only applied if $c_e > 0$ for all $e \in E$. Let $\text{dist}(i, j)$ be the length of the shortest path linking vertices i and j, for $i, j, \in V$. If $\text{dist}(i, j) < c_e$, where $e = (i, j) \in E$, then edge e is suboptimal and could be eliminated from G.

5.1.2. *Cardinality-one test.* Assume that a given vertex, say vertex $i \in V$, has an edge cardinality of one, i.e., $|\delta(i)| = 1$. Denote by e the only edge incident to i. If $c_e > d_i$, then vertex i and, consequently, edge e are suboptimal and could be eliminated from G.

5.1.3. *Cardinality-two test.* Assume that the edge cardinality of $i \in V$ equals two and denote respectively by $e_1 = (i, i_1)$ and $e_2 = (i, i_2)$ the two edges of G that are incident to i. If $c_{e_1} > 0$, $c_{e_2} > 0$ and $d_i = 0$, either e_1 and e_2 must simultaneously appear at an optimal PCS tree or else neither of these edges may be part of such a tree. The reasoning behind this test, as explained before, follows from the suboptimality of any PCS tree containing vertex i as a leaf. Recall, in that case, that the only edge incident to i may be eliminated from the tree and a lesser cost PCS tree would then result.

Provided the conditions set above are met, vertex i could be *pseudo eliminated* by replacing the two edges incident to it by a single edge (i_1, i_2) of cost $c_{(i,i_1)}+c_{(i,i_2)}$. Whenever two edges (i_1, i_2) result from this operation, only the edge with the least cost should be kept.

5.1.4. *Cardinality-larger-than-two test.* Provided certain conditions are met, pseudo elimination could also be extended to vertices with edge degrees larger than 2. Assume, for instance, that vertex $i \in V$ has $|\delta(i)| = 3$, all edges incident to i are positive valued and $d_i = 0$. Assume as well that it was somehow established that no optimal PCS tree exists with 3 edges incident to i. Therefore, under these conditions, either vertex i is part of no optimal PCS tree or else it appears at such a tree with an edge degree of 2 (recall that i could not have an edge degree of 1 at an optimal PCS tree). Vertex i could thus be pseudo eliminated by joining together into single edges each of the 3 possible combinations of two edges incident to i. Clearly, in doing so, no optimal PCS tree would be eliminated from the original solution space.

For a vertex $i \in V$ with edge degree $k \geq 4$, testing for pseudo elimination is considerably more demanding than the situation described above for $k = 3$. For $k \geq 4$, pseudo elimination may only be carried out if it could be established that no optimal PCS tree exists containing exactly l edges incident to i, for $3 \leq l \leq k$. However, due to the combinatorial explosion implied by checking that condition, the test should be restricted, in practice, to vertices where k is not *very large*.

Following the outline suggested above, we will now formally describe the test. For convenience, it will be split in two cases. The first is for vertices $i \in V$ with edge degree $k = 3$. The second is for those vertices $i \in V$ with $k \geq 4$. In either case, the cost of all edges incident to i must be non negative and $d_i = 0$.

Assume first that $|\delta(i)| = 3$ and let i_1, i_2 and i_3 be the three vertices of G sharing an edge with i. Denote respectively by $e_1 = (i, i_1), e_2 = (i, i_2)$, and $e_3 = (i, i_3)$, these edges. Then, see [11] for details, if

$$\begin{aligned}(30)\quad \min\{&\mathrm{dist}(i_1, i_2) + \mathrm{dist}(i_1, i_3), \mathrm{dist}(i_2, i_1) + \mathrm{dist}(i_2, i_3),\\ &\mathrm{dist}(i_3, i_1) + \mathrm{dist}(i_3, i_2)\} \leq c_{e_1} + c_{e_2} + c_{e_3},\end{aligned}$$

no optimal PCS tree exists involving 3 edges incident to i. As such, i could be pseudo eliminated from G. As explained before, that is accomplished by replacing every combination of two distinct edges incident to i by an associated, conveniently defined, single edge.

Let us know extend the test for vertices $i \in V$ with $|\delta(i)| = k$, where $k \geq 4$. To understand this generalization, one should notice that the left hand side of (30) equals the cost of the MST for the *distance subgraph* of G implied by vertices i_1, i_2 and i_3. Thus, in general terms, one should first compute a MST for the distance subgraph of G associated with the k vertices that share and edge with i. Having done that, one should then compare the cost of that MST against the sum of the costs for the k edges incident to i. In case the MST has the smallest cost, a guarantee is obtained that no optimal PCS tree exists involving exactly k edges incident to i. However, at that point, to allow i to be pseudo eliminated, a similar test must also be successful for every distinct combination involving l of the k edges incident to i, for $3 \leq l \leq (k-1)$.

5.1.5. *Minimum adjacency test.* Assume that an edge $(i, j) \in E$ exists linking two positive penalty vertices $i, j \in V$. If $\min\{d_i, d_j\} - c_{(i,j)} > 0$ and $c_{(i,j)} = \min\{c_{(i,u)} : (i, u) \in E\}$, then vertices i and j may be shrunk into a single vertex of penalty $d_i + d_j - c_{(i,j)}$. As a result of this shrinking, whenever edges (i, v) and (j, v) belong to E, two *parallel* edges linking v to the new vertex will result. These should then be merged into a single edge of cost $\min\{c_{(i,v)}, c_{(j,v)}\}$.

5.1.6. *Net weight gain cardinality two path test.* For three given vertices of V, say i, j, and k, assume that edges (i, j), (i, k), and (j, k) belong to E. Assume as well that $c_{(i,k)} + c_{(j,k)} - d_k < c_{(i,j)}$ and $c_{(i,j)} \geq \min\{c_{(i,k)}, c_{(j,k)}\}$ apply. Then, in this Net Weight Gain Cardinality Two Path Test (NWGC2), edge (i, j) must be suboptimal since the path formed by edges (i, k) and (j, k) offers an alternative to spanning vertices i and j through edge (i, j), at a positive net weight gain of $(c_{(i,j)} - c_{(i,k)} - c_{(j,k)} + d_k)$. It should be noticed that the NWGC2 improves on the minimum distance test. That applies since NWGC2 may eventually succeed in proving that an edge $(i, j) \in E$ for which $\mathrm{dist}(i, j) = c_{(i,j)}$, is suboptimal.

5.2. **Variable fixing tests.** As indicated before, feasible solutions to $LRP(\lambda^k)$, defined as suggested in Section 4, imply forests of G' with exactly $(|V'| - 2)$ edges. For this type of structure, LP reduced costs are quite straightforward to compute. Indeed, this task could be accomplished by performing some simple, conveniently defined, edge exchanges. These exchanges, in turn, directly follow from exchanges previously suggested for computing LP reduced costs for spanning trees [32].

For our particular application, assume that the k-th iteration of SM is being implemented and let $(\overline{x}^k, \overline{z}^k) \in \mathbb{B}^{|E|+|V|}$ be an optimal solution to $LRP(\lambda^k)$, formulated at that iteration. Accordingly, the $|E|$ components in $\overline{x}^k$ are associated with the edges of E. Likewise, the $|V|$ components in $\overline{z}^k$ are associated with the edges $\{(n+1, i) : i \in V\}$ of the expanded graph $G' = (V', E')$. As one may recall, edges $\{(n+1, i) : i \in V\}$ are part of our reformulation and reinterpretation of PCSPG in terms of G'.

For $(\overline{x}^k, \overline{z}^k)$, as defined above, assume that $\overline{c}_e^k$, for $e = (i, j) \in E'$, is the corresponding LP reduced cost for the variables involved. Then, $\overline{c}_e^k$ is computed as

$$\overline{c}_e^k = c_e^k - c_{e_0}^k, \tag{31}$$

where e_0 is an edge that is dependent on the forest topology that $(\overline{x}^k, \overline{z}^k)$ implies on G'. Assume first that i and j share a same component in that forest. Then a unique path must exist linking i and j in that component and e_0 should be taken as the largest Lagrangian edge cost for this path. Otherwise, if i and j appear in different components, e_0 should be taken as the largest overall Lagrangian cost for an edge in the solution forest.

Denote by w_{λ^k} the value of solution $(\overline{x}^k, \overline{z}^k)$ to $LRP(\lambda^k)$ and by $\overline{w}$ a known valid PCSPG upper bound. Then, if $(\overline{c}_e^k + w_{\lambda^k}) > \overline{w}$, the variable associated with e is guaranteed not to appear at an optimal PCSPG solution. As such, that variable (respectively edge e) could thus be eliminated from the formulation (respectively from G'). The Lagrangian variable fixing test we have just described became standard in the literature and could be traced back at least to [34, 32].

6. Computational experiments

Computational tests were carried out for a total of 168 PCSPG instances taken from the literature. These instances belong to the following sets:

- Sets P and K, respectively with 11 and 23 instances, as proposed in [18].
- Sets C and D with 40 instances each, as proposed in [7]. These instances originate from the OR-Library SPG test sets C and D [2].
- Set E with 40 instances, as proposed in [21]. Instances in this set also originate from the OR-Library [2] and were generated exactly as sets C and D, above (see [7], for details).
- Set H with 14 instances, as proposed in [21]. These instances originate from the SPG hypercube instances introduced in [31].

All algorithms used in our computational experiments were coded in `C`. Experiments were carried out on a Pentium IV machine running at 3GHz and having 512 Mb of RAM memory. Linux was the operational system used and code was compiled under GNU `gcc` compiler, with flag `-O3` activated.

Due to the large volume of data involved, for convenience, we have chosen to present detailed computational results in Tables 4 - 8 of Appendix A. Condensed aggregate results, that indicate more general trends, are presented in the main text body. However, whenever necessary, specific Appendix A results will also be quoted throughout the text.

6.1. **Pre-processing results.**

Table 1 presents a summary of pre-processing results for each test set considered. Entries in that table, under the headings "nb-v"

and "nb-e", respectively give the percentage of vertices and edges remaining after the pre-processing tests of Section 5 were performed. Results are associated with two different types of experiments. One where all pre-processing tests are carried out and another where, among all available tests, only NWGC2 is omitted. In doing so, benefits from using pre-processing test NWGC2 become more evident. Average CPU times are presented for each type of experiment conducted.

Benefits from using test NWGC2 are more pronounced for test set K where, on the average, 24% more edges ended up being eliminated. For instances in test sets C, D, and E, benefits from using NWGC2 increase with the increase in the proportion of non zero penalty vertices. For this group of instances, the average times quoted in Table 1 were positively influenced by the vigorous performance of NWGC2 for some of the largest instances in the group, namely, C20A, C20B, D20A, and D20B. Preprocessing tests totally failed for set H instances, where not a single edge could be eliminated.

TABLE 1. Summary of preprocessing results

	NWGC2 in			NWGC2 out		
	nb-v	nb-e	t(s)	nb-v	nb-e	t(s)
P	0.84	0.79	0.17	0.84	0.79	0.14
K	0.44	0.38	0.23	0.47	0.50	0.27
C	0.70	0.39	0.75	0.71	0.42	1.17
D	0.72	0.44	3.04	0.72	0.46	4.31
E	0.72	0.20	33.0	0.72	0.21	31.98

6.2. **NDRC lower and upper bounds.** Table 2 presents a summary of the lower and upper bounds obtained after allowing up to `MAXITER`=2000 SM iterations to be performed. For these experiments, parameter `EXTRA` was set to 1 while step-size parameter α, see (27), was initially set to 2, being progressively halved after $\xi = 100$ consecutive SM iterations without an overall improvement on the best Lagrangian lower bound.

The first column in Table 2 identifies the corresponding test set. Columns that follow respectively give the number of instances involved, the number of instances for which NDRC was able to present optimality certificates, i.e., $w_d = \overline{w}$, the number of instances where NDRC upper bounds matched known optimal solution values, i.e., $\overline{w} = w$, the percentage average gaps between upper and lower bounds, and the average CPU times.

For the same parameter settings above, detailed computational results are presented in Tables 4 - 8 of Appendix A. For each of these tables, the first column identifies the instance under investigation. The second indicates the best NDRC lower bound attained, w_d. That is followed by the best upper bound, $\overline{w}$, in column three, the corresponding duality gap, $(\overline{w} - w_d)/w_d$, in column four, and the CPU time (in seconds) to either perform `MAXITER` SM iterations or else to exhibit an optimality certificate, in column five. For Tables 4–7, the last column, w, is the optimal solution value. In Table 8, however, we indicate the best known upper bound. Furthermore, for that table, in the last column, headings "*" indicate that

TABLE 2. Average Relax-and-Cut results - MAXITER=2000, $\xi = 100$, EXTRA = 1

	Number of instances			Average	
	in the set	$w_d = \overline{w}$	$\overline{w} = w$	% dual gap	t(s)
P	11	5	11	0.324	3.053
K	23	11	22	3.829	1.540
C	40	17	40	0.916	5.914
D	40	9	40	1.256	34.832
E	40	6	36	1.480	427.775
H ($d \leq 10$)	10	-	-	6.746	21.630
H ($d = 11, 12$)	4	-	-	10.46	1603.480

the corresponding upper bound was proved to be optimal. Headings "+" indicate that the best upper bound available was found in this study.

As can be appreciated from the computational results presented here, for instances P, K, C, D, and E, NDRC upper bounds are always very sharp. Indeed, for 149 of the 154 instances involved, they matched corresponding optimal solution values. For the remaining 5 instances, the gaps between NDRC upper bounds and corresponding optimal solution values were smaller than 1.5%.

NDRC upper bounds compare favorably with the results reported in [7] in terms of solution quality for test sets P, K, C, and D (test sets E and H were not available at the time [7] was published). Out of the 114 instances involved, NDRC managed to find optimal solutions for 113 of them. Even when pre-processing tests were not used, NDRC upper bounds matched optimal solution values for 109 instances. By comparison, the algorithm in [7] only succeeded in finding optimal solutions for 91 of these instances. When pre-processing is switched off our CPU times increase by an order of magnitude. We refrain from comparing our CPU times with those quoted in [7] since quite different machines were used in the two experiments (a 3.0 GHz Pentium IV processor, for our application, and a 400 MHz Pentium II processor in [7]). However, it is clear that one should expect CPU times in [7] to drop considerably if our pre-processor tests were used.

Lower bounds attained by our NDRC algorithm were, in general, *quite good*. The only exception being the lower bounds for instances in set K. As previously mentioned in a conference version of this paper [8], these instances exhibit a high degree of symmetry and, possibly, this is to blame for the poor NDRC performance.

One should notice that, from a theoretical point of view, the best NDRC lower bounds capable of being attained are those implied by the LP relaxation of (16). From the computational experiments in [27], LP relaxation bounds for (16) are known for instances in test sets C, D, K, and P. For all instances in sets K and P and for almost all instances in sets C and D, LP relaxations in [27] turned out to be naturally integral. Thus, for these instances, LP relaxations correspond to optimal PCSPG solutions. Comparing the NDRC lower bounds in this study with the LP relaxation bounds in [27], it is clear that our bounds are not that far away from the best values they could possibly attain. However, clearly, there is still room for improvements. This is particularly true for instance K400 where

larger than expected duality gaps were obtained. It is possible that such a gap resulted from the Subgradient Method inadequacy to deal with highly symmetrical PCSPG instances. However, one could not be completely sure about that since we also faced some difficulties in choosing the GSECs to dualize (see discussion in Section 4). Recall, from Section 4, that we only dualize at most one GSEC for each candidate vertex set S_i. Recall as well that when violations indeed occur, $|S_i| - 1$ maximal violated GSECs are associated with S_i. For that reason, we have then decided to investigate alternative dualization strategies in an attempt to improve NDRC lower bounds for set K instances. In particular, the following alternatives were tested for every SM iteration where violations occur for vertex set S_i:

- Dualize all associated $|S_i| - 1$ maximal GSECs.
- Dualize a *surrogate* of the associated $|S_i| - 1$ maximal GSECs.

None of these strategies resulted in a consistent lower bound improvement over the strategy we had before. However, another approach was devised that allowed us to partially bridge the gaps between the previously obtained NDRC lower bounds and corresponding best theoretical values. Namely, we have *extended the life* of a dynamically dualized GSEC, as described in Section 3.3. Under this strategy, additional experiments were carried out for the K200 and K400 instances, under different values for parameter `EXTRA`. Corresponding results appear in Table 3 and show that NDRC lower bounds improved, on the average, by 2.32% and 2.54% over their previous values, after `EXTRA` was respectively set to 5 and 10. Motivated by these results, we performed similar experiments for those instances in sets K, P, C, D, and E set for which a duality gap greater or equal to 1% was previously attained. However, for this experiment, the following parameters were used: `MAXITER`=5000, $\xi = 250$ and `EXTRA` $= 5$. In this experiment, four additional optimality certificates were obtained. Furthermore, average duality gaps were reduced by one third.

TABLE 3. Experiments for K200 and K400 under different values for `EXTRA`: `MAXITER` $= 2000$, $\xi = 100$

	`EXTRA`= 1		`EXTRA`= 5			`EXTRA`= 10		
	w_d	t(s)	w_d	t(s)	w_d gain [%]	w_d	t(s)	w_d gain [%]
K200	316673.9170	0.74	325021.7152	0.87	2.64	323684.0345	1.16	2.21
K400	326913.4753	2.73	335354.6354	3.42	2.58	335059.1583	4.03	2.49
K400.1	448931.0463	2.75	456311.7096	4.00	1.64	463199.9214	6.18	3.18
K400.2	426979.2349	5.57	441538.5219	6.87	3.41	443030.7666	9.88	3.76
K400.3	391085.1170	2.39	401146.9746	2.78	2.57	399468.6341	3.95	2.14
K400.4	366707.8591	2.07	373036.3271	2.84	1.73	375097.5467	3.90	2.29
K400.5	495385.8162	2.33	502802.1897	3.45	1.50	505274.0306	5.33	2.00
K400.6	352497.1554	3.34	358232.1295	3.93	1.63	358363.4084	5.35	1.66
K400.7	441518.7576	3.07	451524.8872	4.44	2.27	453673.8149	7.01	2.75
K400.8	398891.5238	3.42	404459.2169	4.26	1.40	404816.8895	6.13	1.49
K400.9	357336.0506	3.00	371505.2503	4.16	3.97	371384.5287	6.44	3.93
K400.10	350741.3267	2.76	359434.9675	3.77	2.48	359876.6518	6.13	2.60

Among PCSPG instances in the literature, those in set H are undoubtedly the hardest to solve to proven optimality. In [21] computational results are presented for

set H instances where $d \leq 10$. These results were obtained under a 1800 seconds CPU time limit imposed on the Branch and Cut algorithm used. For a few of these instances, optimality certificates were obtained in [21]. However, results for the $d = 11$ and $d = 12$ set H instances were not quoted in [21]. Using results in [21] as a basis for comparison, one should notice, from the results in Table 8 (see also Appendix A), that, for the $d \leq 10$ set H instances, *good quality* NDRC upper bounds were obtained under *acceptable* CPU times. In particular, for three of these instances, new best upper bounds were attained. For the remaining instances in that set, i.e. for the $d = 11$ and $d = 12$ instances, NDRC duality gaps were attained with a magnitude comparable to those quoted in [21] for the $d = 10$ instances. Furthermore, these gaps were obtained under CPU times comparable to those quoted in [21] for the, smaller, $d = 10$ instances.

For instances in sets C, D, and E, we have have also compared NDRC CPU times with those given in [21]. Average CPU times quoted in that reference are for a 2.8 Ghz Intel Pentium IV based machine with 2 Gb of RAM memory. The average CPU time required by the Branch and Cut algorithm in [21] to solve all instances in sets C, D, and E, was respectively 4.9, 22.3, and 253.4 seconds. Comparing these results with those required by NDRC to attain the duality gaps we quote here, it is estimated that the algorithm in [21] is already about 1.7 times faster than ours. This result clearly indicates the efficiency of the algorithm in [21] (we believe that LP relaxation bounds for the two formulations should not be that different). Although our NDRC algorithm compares unfavorably with the algorithm in [21] for instance sets C, D, and E, that is not the case for instances in set H, the hardest in the literature. As a general rule, for instances in set H, our heuristic managed to find better solutions in CPU times that were up to 2 orders of magnitude less than those reported in [21].

7. Conclusions

Algorithms to generate primal and dual PCSPG bounds were proposed in this paper. These algorithms originate from a Lagrangian NDRC based approach and incorporate ingredients such as a new PCSPG reduction test, an effective Local Search procedure and a modification in the NDRC framework which allowed additional reductions in duality gaps to be attained.

NDRC upper bounds for PCSPG turned out very sharp for almost all instances tested. In particular, optimal solutions were generated for 149 out of the 154 instances tested in sets K, P, C, D, and E. The Lagrangian heuristic that produced these bounds thus appear to dominate the best PCSPG heuristic available in the literature [7]. On the other hand, in terms of CPU time and lower bound quality, for test sets C, D, and E, NDRC was easily outperformed by the Branch-and-Cut algorithm in [21], which is the best exact solution algorithm available for PCSPG. However, for test set H, the hardest to solve to proven optimality, NDRC outperformed the algorithm in [21]. In particular, while requiring less CPU time than that quoted in [21], NDRC generated new best upper bounds for seven set H instances.

A possible extension of the work presented here is to use the PCSPG NDRC algorithm simultaneously as a pre-processor and a warm start to a corresponding Branch-and-Cut algorithm. In doing so, one would be able to reduce instance input size and also benefit from *good quality* PCSPG upper bounds. Most important of all, one would additionally be able to carry over to Branch-and-Cut some *attractive*

GSECs dualized at the NDRC algorithm. One would thus be able to match, and sometimes even improve upon, at the very first Linear Programming relaxation solved, the best lower bounds attained at NDRC. An example of the scheme just outlined, tailored to the Degree-Constrained Minimum Spanning Tree Problem, could be found in [9].

References

[1] E. Balas. "The prize collecting traveling salesman problem". *Networks*, 19:621–636, 1989.

[2] J.E. Beasley. "OR-Library: Operations Research Library". http://people.brunel.ac.uk/ mastjjb/jeb/info.html.

[3] J.E. Beasley. "An Algorithm for the Steiner Problem in Graphs". *Networks*, 14:147–160, 1984.

[4] J.E. Beasley. "An SST-based Algorithm for the Steiner Problem in Graphs". *Networks*, 19:1–16, 1989.

[5] J.E. Beasley. "Lagrangean Relaxation". In Collin Reeves, editor, *Modern Heuristic Techniques*. Blackwell Scientific Press, Oxford, 1993.

[6] D. Bienstock, M. Goemans, D. Simchi-Levi, and D.P. Williamson. "A note on the prize collecting travelling salesman problem". *Mathematical Programming*, 59:413–420, 1993.

[7] S.A. Canuto, M.G.C. Resende, and C.C. Ribeiro. "Local search with perturbations for the prize collecting Steiner tree problem in graphs". *Networks*, 38:50–58, 2001.

[8] A. S. Cunha, A. Lucena, N. Maculan, and M.G.C Resende. "A Relax-and-cut algorithm for the Prize Collecing Steiner Problem in Graphs". In *Mathematical Programming in Rio: a conference in honour of Nelson Maculan*, pages 72–78, Buzios, Rio de Janeiro, November 9-12, 2003.

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

[10] G. B. Dantzig, R. Fulkerson, and S. Johnson. "Solution of a Large Scale Traveling Salesman Problem". *Operations Research*, 2:393–410, 1954.

[11] C. Duin. *Steiner's problem in Graphs: Approximation, reduction, variation*. PhD thesis, University of Amsterdam, 1994.

[12] C. Duin and S. Voss. "Efficient Path and Vertex Exchange in Steiner Tree Algorithms". *Networks*, 29:89–105, 1997.

[13] S. Engewall, M. Gothe-Lundgren, and P. Varbrand. "A Strong Lower Bound for the Node Weighted Steiner Tree Problem". *Networks*, 31:11–17, 1998.

[14] M. Goemans. "The Steiner tree polytope and related polyhedra". *Mathematical Programming*, 63:157–182, 1994.

[15] M. Goemans and D.P. Williamson. "The primal dual method for approximation algorithms and its applications to network design problems". In Dorit S. Houchbaum, editor, *Approximation algorithms for NP-hard problems*, pages 144–191. P.W.S Publishing Co., 1996.

[16] S.L. Hakimi. "Steiner's problem in graphs". *Networks*, 1:113–133, 1971.

[17] M. Haouari and J.C. Siala. "A hybrid Lagrangian genetic algorithm for the prize collecting Steiner tree problem". *Computers and Operations Research*, 33:1274–1288, 2006.

[18] D.S. Johnson, M. Minkoff, and S. Phillips. "The prize collecting Steiner tree problem: Theory and Practice". In *Proc. 11th ACM-SIAM Symp. on Discrete Algorithms*, 2000.

[19] R.M. Karp. "Reductibility among combinatorial problems". In E. Miller and J.W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[20] 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.

[21] I. Ljubic, R. Weiskircher, U. Pferschy, G. Klau, P. Mutzel, and M. Fischetti. Solving the Prize-Collecting Steiner Problem to Optimality. Technical report, Technische Universitat Wien, Institut fur Computergraphik und Algorithmen, 2004.

[22] 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.

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

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

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

[26] A. Lucena and J.E. Beasley. "A Branch and Cut Algorithm for the Steiner Problem in Graphs". *Networks*, 31:39–59, 1998.

[27] A. Lucena and M.G.C. Resende. "Strong lower bounds for the prize-collecting Steiner problem in Graphs". *Discrete Applied Mathematics*, 141:277–294, 2004.

[28] F. Margot, A. Prodon, and Th.M. Liebling. "Tree polytope on 2-trees". *Mathematical Programming*, 63:183–192, 1994.
[29] M. Minkoff. The Prize Collecting Steiner Tree Problem. Master's thesis, Departament of Electrical Engineering and Computer Science, Massachusetts Insitute of Technology, 2000.
[30] C.C. Ribeiro and M.C. de Souza. "Tabu Search for the Steiner Problem in Graphs". *Networks*, 36(2):138–146, 2000.
[31] I. Rosseti, M. Poggi de Aragão, C.C. Ribeiro, E. Uchoa, and R.F. Werneck. "New benchmark instances for the Steiner problem in graphs". In *Extended Abstracts of the 4th Metaheuristics International Conference*, pages 557–561, 2001.
[32] M. Savelsbergh and A. Volgenant. "Edge exchanges in the degree constrained minimum spanning tree problem". *Computers and Operations Research*, 12(4):341–348, 1985.
[33] A. Segev. "The Node-Weighted Steiner Tree Problem". *Networks*, 17:1–17, 1987.
[34] 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.

Appendix A. Detailed computational results

Table 4. Relax-and-Cut results - Sets K,P - `MAXITER`=2000, ξ = 100, `EXTRA` = 1

Instance	w_d	$\overline{w}$	gap [%]	t(s)	w
P100	803300.0000	803300	Opt	0.25	803300
P100.1	926238.0000	926238	Opt	0.32	926238
P100.2	401641.0000	401641	Opt	0.31	401641
P100.3	659644.0000	659644	Opt	0.23	659644
P100.4	827419.0000	827419	Opt	0.18	827419
P200	1311970.4538	1317874	0.450	1.58	1317874
P400	2450885.8128	2459904	0.368	5.74	2459904
P400.1	2776739.1010	2808440	1.142	7.52	2808440
P400.2	2512006.9007	2518577	0.262	5.18	2518577
P400.3	2938688.2638	2951725	0.444	5.99	2951725
P400.4	2827694.8332	2852956	0.893	6.28	2852956
K100	135511.0000	135511	Opt	0.11	135511
K100.1	124108.0000	124108	Opt	0.09	124108
K100.2	200262.0000	200262	Opt	0.18	200262
K100.3	115953.0000	115953	Opt	0.25	115953
K100.4	87498.0000	87498	Opt	0.04	87498
K100.5	119078.0000	119078	Opt	0.06	119078
K100.6	132886.0000	132886	Opt	0.02	132886
K100.7	172457.0000	172457	Opt	0.10	172457
K100.8	210869.0000	210869	Opt	0.34	210869
K100.9	122917.0000	122917	Opt	0.02	122917
K100.10	133567.0000	133567	Opt	0.03	133567
K200	316673.9170	329211	3.959	0.74	329211
K400	326913.4753	350093	7.090	2.73	350093
K400.1	448931.0463	490771	9.320	2.75	490771
K400.2	426979.2349	477073	11.732	5.57	477073
K400.3	391085.1170	415328	6.199	2.39	415328
K400.4	366707.8591	389451	6.202	2.07	389451
K400.5	495385.8162	519526	4.873	2.33	519526
K400.6	352497.1554	374849	6.341	3.34	374849
K400.7	441518.7576	474466	7.462	3.07	474466
K400.8	398891.5238	418614	4.944	3.42	418614
K400.9	357336.0506	383105	7.211	3.00	383105
K400.10	350741.3267	395413	12.736	2.76	394191

TABLE 5. Relax-and-Cut results - Set C - MAXITER=2000, $\xi = 100$, EXTRA = 1

Instance	w_d	$\overline{w}$	gap [%]	t(s)	w
c1A	18.0000	18	Opt	0.18	18
c1B	83.0874	85	2.302	0.84	85
c2A	49.4865	50	Opt	0.18	50
c2B	139.4014	141	1.147	0.85	141
c3A	413.1276	414	Opt	0.50	414
c3B	733.5273	737	0.473	2.16	737
c4A	616.9955	618	0.163	1.60	618
c4B	1054.3364	1063	0.822	3.49	1063
c5A	1079.0071	1080	Opt	1.48	1080
c5B	1525.6234	1528	0.156	2.56	1528
c6A	17.5147	18	Opt	0.21	18
c6B	50.4103	55	9.105	3.28	55
c7A	49.2453	50	Opt	0.38	50
c7B	101.0692	102	Opt	2.36	102
c8A	359.7694	361	0.342	4.78	361
c8B	496.5290	500	0.699	6.52	500
c9A	530.0124	533	0.564	6.57	533
c9B	689.9805	694	0.583	10.28	694
c10A	856.9452	859	0.240	5.90	859
c10B	1067.1803	1069	0.171	6.67	1069
c11A	17.0926	18	Opt	1.00	18
c11B	29.7161	32	7.686	5.50	32
c12A	37.0331	38	Opt	2.24	38
c12B	43.8827	46	4.825	6.38	46
c13A	234.7131	236	0.548	11.08	236
c13B	255.9738	258	0.792	14.71	258
c14A	290.1036	293	0.998	13.20	293
c14B	315.1460	318	0.906	13.69	318
c15A	498.1160	501	0.579	16.49	501
c15B	549.0151	551	0.362	14.89	551
c16A	10.0490	11	Opt	5.09	11
c16B	10.5539	11	Opt	9.38	11
c17A	17.0040	18	Opt	7.84	18
c17B	17.0590	18	Opt	7.80	18
c18A	109.3350	111	1.523	15.52	111
c18B	111.1436	113	1.670	16.75	113
c19A	145.0291	146	Opt	5.58	146
c19B	145.0766	146	Opt	4.33	146
c20A	265.0154	266	Opt	2.88	266
c20B	266.0868	267	Opt	1.41	267

TABLE 6. Relax-and-Cut results - Set D - MAXITER=2000, $\xi = 100$, EXTRA = 1

Instance	w_d	$\overline{w}$	gap [%]	t(s)	w
d1A	18.0000	18	Opt	0.75	18
d1B	104.0882	106	1.837	2.34	106
d2A	49.2103	50	Opt	0.74	50
d2B	217.0053	218	Opt	1.98	218
d3A	806.0212	807	Opt	2.75	807
d3B	1504.1409	1509	0.323	8.21	1509
d4A	1200.0942	1203	0.242	5.10	1203
d4B	1878.0088	1881	0.159	7.46	1881
d5A	2155.0896	2157	0.089	8.07	2157
d5B	3124.1323	3135	0.348	10.16	3135
d6A	17.2068	18	Opt	0.93	18
d6B	61.6590	67	8.662	11.40	67
d7A	49.2524	50	Opt	1.78	50
d7B	100.7147	103	2.269	11.11	103
d8A	751.9670	755	0.403	17.84	755
d8B	1031.7074	1036	0.416	33.46	1036
d9A	1065.6080	1070	0.412	22.52	1070
d9B	1415.4595	1420	0.321	47.73	1420
d10A	1668.1245	1671	0.172	37.11	1671
d10B	2074.9530	2079	0.195	43.25	2079
d11A	17.2934	18	Opt	4.82	18
d11B	25.9028	29	11.957	20.58	29
d12A	40.4884	42	3.733	28.64	42
d12B	40.6799	42	3.245	23.98	42
d13A	442.7795	445	0.501	58.28	445
d13B	483.9459	486	0.424	81.47	486
d14A	598.2902	602	0.620	86.95	602
d14B	662.3140	665	0.406	121.85	665
d15A	1038.9896	1042	0.290	105.53	1042
d15B	1105.9666	1108	0.184	92.06	1108
d16A	12.0134	13	Opt	22.83	13
d16B	12.0453	13	Opt	23.47	13
d17A	21.8721	23	5.157	43.90	23
d17B	21.8639	23	5.196	47.58	23
d18A	216.1264	218	0.867	76.55	218
d18B	221.5386	223	0.660	71.48	223
d19A	304.9376	306	0.348	89.42	306
d19B	308.6635	310	0.433	71.23	310
d20A	534.9921	536	0.188	38.75	536
d20B	535.9886	537	0.189	9.21	537

TABLE 7. Relax-and-Cut results - Set E - MAXITER=2000, $\xi = 100$, EXTRA = 1

Instance	w_d	$\overline{w}$	gap [%]	t(s)	w
e01A	13.0000	13	Opt	5.25	13
e01B	105.1852	109	3.627	13.81	109
e02A	30.0000	30	Opt	5.25	30
e02B	164.3261	170	3.453	16.25	170
e03A	2227.8590	2231	0.141	24.81	2231
e03B	3796.3260	3806	0.255	82.52	3806
e04A	3146.5045	3151	0.143	35.57	3151
e04B	4876.7883	4888	0.230	94.24	4888
e05A	5651.8605	5657	0.091	57.24	5657
e05B	7990.8468	7998	0.090	103.73	7998
e06A	18.3809	19	Opt	5.79	19
e06B	67.3600	70	3.919	66.35	70
e07A	39.3128	40	Opt	8.34	40
e07B	130.2630	136	4.404	70.47	136
e08A	1872.8424	1878	0.275	152.55	1878
e08B	2547.5657	2555	0.292	424.04	2555
e09A	2781.8277	2787	0.186	196.88	2787
e09B	3535.8632	3541	0.145	761.16	3541
e10A	4580.8285	4586	0.113	439.38	4586
e10B	5496.8508	5502	0.094	696.69	5502
e11A	20.2059	21	Opt	67.74	21
e11B	31.6056	34	7.576	190.08	34
e12A	48.0288	49	Opt	164.91	49
e12B	64.1235	68	6.045	214.32	67
e13A	1167.0608	1169	0.166	933.14	1169
e13B	1266.1192	1270	0.307	1075.74	1269
e14A	1575.6258	1579	0.214	875.29	1579
e14B	1712.9595	1716	0.177	1732.24	1716
e15A	2607.9719	2610	0.078	1355.50	2610
e15B	2764.9737	2767	0.073	1315.38	2767
e16A	13.9277	15	7.699	330.50	15
e16B	13.9762	15	7.325	341.25	15
e17A	23.7220	25	5.387	371.07	25
e17B	23.8987	25	4.608	373.75	25
e18A	552.5118	557	0.812	1264.09	555
e18B	562.0238	566	0.707	1274.03	564
e19A	745.6350	747	0.183	908.66	747
e19B	756.1145	758	0.249	528.91	758
e20A	1329.9984	1331	0.075	429.85	1331
e20B	1340.9730	1342	0.077	104.21	1342

TABLE 8. NDRC results - Set H

Instance	w_d	$\overline{w}$	gap [%]	t(s)	best known upper bound
hc6p	3822.49	3985	4.25	0.50	3908 (*)
hc6u	35.46	37	4.33	0.48	36 (*)
hc7p	7551.41	8134	7.72	1.49	7739
hc7u	70.48	73	3.57	1.17	72 (*)
hc8p	14955.75	16023	7.13	5.20	15274
hc8u	140.43	151	7.53	4.14	150
hc9p	29598.42	32151	8.62	16.06	32151 (+)
hc9u	278.78	296	6.18	13.13	296 (+)
hc10p	58865.31	64974	10.38	114.36	64974 (+)
hc10u	551.28	594	7.75	59.77	594
hc11p	116751.68	130252	11.56	629.92	130252 (+)
hc11u	1099.82	1199	9.02	360.61	1199 (+)
hc12p	231727.31	257833	11.27	3507.70	257833 (+)
hc12u	2184.48	2403	10.00	1915.72	2403 (+)

(Alexandre S. da Cunha) Departamento de Ciência da Computação, Universidade Federal do Minas Gerais, Belo Horizonte, MG, Brazil
E-mail address: acunha@dcc.ufmg.br

(Abilio Lucena) Departamento de Administração, Universidade Federal do Rio de Janeiro, Av. Pasteur 250, 22290-240 Rio de Janeiro, RJ, Brazil
E-mail address: abiliolucena@globo.com

(Nelson Maculan) Programa de Engenharia de Sistemas e Computação, Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ, Brazil
E-mail address: maculan@cos.ufrj.br

(Mauricio G. C. Resende) Internet and Network Systems Research, AT&T Labs Research, 180 Park Avenue, Room C241, Florham Park, NJ 07932 USA.
E-mail address: mgcr@research.att.com