Thinning out Steiner trees: A node-based model for uniform edge costs

Matteo Fischetti [*1], Markus Leitner[†2], Ivana Ljubić [‡3], Michele Monaci [§1], Martin Luipersbeck[¶2], Max Resch [‖4], Domenico Salvagnin [**1], and Markus Sinnl[††2]

[1]*DEI, University of Padua, Italy.*
[2]*ISOR, University of Vienna, Austria.*
[3]*ESSEC Business School of Paris, France.*
[4]*Vienna University of Technology, Austria.*

Abstract

The Steiner tree problem is a challenging NP-hard problem. Many hard instances of this problem are publicly available, that are still unsolved by state-of-the-art branch-and-cut codes. A typical strategy to attack these instances is to enrich the polyhedral description of the problem, and/or to implement more and more sophisticated separation procedures and branching strategies. In this paper we investigate the opposite viewpoint, and try to make the solution method as simple as possible while working on the modeling side. Our working hypothesis is that the extreme hardness of some classes of instances mainly comes from over-modeling, and that some instances can become quite easy to solve when a simpler model is considered. In other words, we aim at "thinning out" the usual models for the sake of getting a more agile framework. In particular, we focus on a model that only involves node variables, which is rather appealing for the "uniform" cases where all edges have the same cost.

In our computational study, we first show that this new model allows one to quickly produce very good (sometimes proven optimal) solutions for notoriously hard instances from the literature. In some cases, our approach takes just few seconds to prove optimality for instances never solved (even after days of computation) by the standard methods. Moreover, we report improved solutions for several SteinLib instances, including the (in)famous hypercube ones. We also demonstrate how to build a unified solver on top of the new node-based model and the previous state-of-the-art model (defined in the space of arc and node variables). The solver relies on local branching, initialization heuristics, preprocessing and local search procedures. A filtering mechanism is applied to automatically select the best algorithmic ingredients for each instance individually.

1 Introduction

The Steiner tree problem (STP), in any of its various versions, is a challenging NP-hard problem that involves two related decisions: choosing the nodes to cover, and then covering them at minimum cost. Once the first decision has been taken, the second one is just trivial as it amounts to solving a minimum-cost tree spanning the selected nodes.

In this paper we introduce a new Mixed-Integer Linear Programming (MIP) approach for solving hard instances of the Steiner tree problem (close) to optimality. Instead of modeling graph connectivity using edge or arc variables

[*]matteo.fischetti@unipd.it
[†]markus.leitner@univie.ac.at
[‡]ivana.ljubic@essec.edu
[§]michele.monaci@unipd.it
[¶]martin.luipersbeck@univie.ac.at
[‖]max.resch@alumni.tuwien.ac.at
[**]domenico.salvagnin@unipd.it
[††]markus.sinnl@univie.ac.at

(where many of them can exist), we propose to model it by using node variables only. Our model is particularly suited for "uniform" cases where all edges have the same cost. Besides the fact that these node-based models contain significantly less variables, they also avoid equivalences induced by uniform edge weights. For very dense graphs, or those containing a lot of symmetries, this strategy significantly outperforms the standard models where connectivity is modeled by using edge variables. We then demonstrate how to build a unified solver on top of the new node-based model and the previous state-of-the-art model from [21] (defined in the space of arc and node variables). The solver relies on local branching, initialization heuristics, preprocessing and local search procedures. A filtering mechanism is applied to automatically select the best algorithmic ingredients for each instance individually.

Our approach works for different variants of the Steiner tree problem, including the (rooted) prize-collecting STP (PCSTP), the node-weighted STP (NWSTP), the maximum-weight connected subgraph problem (MWCS), and also the degree-constrained STP (DCSTP). To have a unified framework, in the following we will therefore focus on a slightly more general variant of the PCSTP, with a (potentially empty) set of terminal nodes. This general problem definition covers both the classical Steiner tree problem in graphs and its prize-collecting counterpart, as special cases. Necessary adaptations for the remaining problems will be explained below. It is worth mentioning that our code (with four variants submitted under the names `mozartballs`, `staynerd`, `hedgekiller` and `mozartduet`) was the winner of the DIMACS Challenge on Steiner trees [1] in most of the categories (see [11] for more details). This article contains a summary of the main ingredients of this implementation.

Definition 1.1 (The prize-collecting Steiner tree problem (PCSTP))**.** *Given an undirected graph $G = (V,E)$ with a (possibly empty) set of* real terminals *$T_r \subset V$, edge costs $c : E \mapsto \mathbb{R}^+$ and node revenues $p : V \mapsto \mathbb{R}^+$, the goal is to find a subtree $\mathscr{T} = (V[\mathscr{T}], E[\mathscr{T}])$ that spans all real terminals and such that the cost*

$$c(\mathscr{T}) = \sum_{e \in E[\mathscr{T}]} c_e + \sum_{i \notin V[\mathscr{T}]} p_i$$

is minimized.

In the classical PCSTP version studied in the previous literature, $T_r = \emptyset$, and the problem can be equivalently stated as searching for a subtree that maximizes the difference between the collected node revenues ($\sum_{i \in V[\mathscr{T}]} p_i$) and the costs for establishing the links of that tree ($\sum_{e \in E[\mathscr{T}]} c_e$). One objective value can be transformed into another by subtracting $c(\mathscr{T})$ from the sum of all node revenues ($P = \sum_{i \in V} p_i$).

In general, each node in $V \setminus T_r$ is considered as a *Steiner node*, i.e., it can be used as an intermediate node to connect real terminals, or those with positive revenues.

Observe that there always exists an optimal PCSTP solution in which non-terminal nodes with zero revenue are not leaves. The same holds for each node $i \in V$ such that $p_i > 0$ and $\min_{\{i,j\} \in E} c_{ij} > p_i$. Note that we impose strict inequality in the latter condition. Hence, besides real terminals only a specific subset of nodes in the PCSTP can be leaves of an optimal solution. We will refer to those nodes as *potential terminals*.

Definition 1.2 (Potential terminals)**.** *Among the nodes $i \in V \setminus T_r$, only those with revenues $p_i > 0$ such that at least one adjacent edge is strictly cheaper than p_i are considered as potential leaves. These nodes are referred to as* potential terminals*, and the associated set is denoted by T_p:*

$$T_p = \{v \in V \setminus T_r \mid \exists \{u,v\} \text{ s.t. } c_{uv} < p_v\}.$$

In the following, we will call the set $T = T_r \cup T_p$, the set of *terminal nodes*. Our general problem definition covers the Steiner tree problem in graphs, since in this case all node revenues are equal to zero ($p_v = 0$, for all $v \in V$) and the set of real terminals is nonempty, i.e., $\emptyset \neq T = T_r \subset V$.

Previous work: A MIP model for the PCSTP using both node and arc variables has been proposed in [21]. The formulation is based on a transformation of the PCSTP to a directed instance. A similar cut-based model has also been applied to the STP in [18]. In our work the model of [21] has been extended to the general problem definition of the PCSTP presented in this article which covers both problems. Given an instance $I = (G(V,E), c, p, T_r)$ of the PCSTP, an instance $I' = (G'(V',A'), c', p, T_r, r)$ is constructed as follows: An artificial root node r is added, i.e., $V' = V \cup \{r\}$. For each edge $\{i,j\} \in E$, A' contains two anti-parallel arcs (i,j) and (j,i) with $c'_{ij} = c_{ij} - p_j$. Furthermore, r is connected by arcs (r, j) with $c'_{rj} = -p_j$ to each potential terminal $j \in T_p$. The set of terminals and revenues are left unchanged. For $S \subset V$, let $\delta^-(S)$ be defined as the set $\{(i, j) \in A' : i \notin S, j \in S\}$. Subsequently, a MIP model for the PCSTP is formulated as follows:

$$(\text{PCSTP}_{xy}) \qquad \min \sum_{(i,j)\in A'} c'_{ij} x_{ij} + \sum_{v\in V} p_v \tag{1}$$

$$\sum_{(i,v)\in A'} x_{iv} = y_v \qquad \forall v \in V \tag{2}$$

$$x(\delta^-(S)) \geq y_v \qquad \forall v \in (S \cap T), r \notin S, \forall S \subset V' \tag{3}$$

$$\sum_{(r,i)\in A'} x_{ri} = 1 \tag{4}$$

$$y_v = 1 \qquad \forall v \in T_r \tag{5}$$

$$x_{ij}, y_v \in \{0,1\} \qquad \forall (i, j) \in A, \forall v \in V \tag{6}$$

Binary arc variables $x_{ij}, \forall (i, j) \in A'$, are set to one if arc (i, j) is part of the solution. Similiarly, binary node variables $y_v, \forall v \in V' \setminus \{r\}$, are set to one if node v is part of the solution. Constraints (2) ensure that each node selected has exactly one incoming arc, and link node and arc variables. Cut constraints (3) guarantee that the solution is connected. Constraint (4) ensures that exactly one artificial root arc is chosen.

Constraints (5) have been added to allow the incorporation of real terminals T_r into the model, which have not been considered in [21]. Note that if $|T_r| \geq 1$, instead of adding an artificial root and its associated arcs, it is sufficient to choose an arbitrary real terminal $v \in T_r$ as root. In this scenario constraint (4) must be excluded from the model.

The formulation may be augmented with further valid or strengthening inequalities, e.g., flow-balance inequalities, root asymmetry constraints, and generalized subtour elimination constraints (GSEC) of size two (cf. [21] for details). The advantage of this formulation is that it makes little assumptions on instance structure, however the number of arc variables may be computationally prohibitive on dense graphs. Throughout this article, we will refer to this formulation as (x, y)-model.

2 A node-based MIP model

Node-based models for solving the maximum-weight connected subgraph problem have been compared, both theoretically and computationally, in a recent publication [2]. Since there are no edge-costs involved in the objective of the MWCS, a natural MIP modeling approach is to derive a formulation in the space of node variables only. The first node-based model for the MWCS has been proposed in [5], and has been shown to computationally outperform extended formulations (involving both edge and node variables) on a benchmark set of instances from bioinformatics applications. However, as demonstrated in [2], the cycle-elimination model of [5] provides arbitrarily bad lower bounds and can be computationally improved by considering the notion of node separators whose definition is provided below. For brevity, in the rest of the article a node separator will be referred to simply as *separator*.

For STP/PCSTP instances, uniform edge costs can be embedded into node revenues (as shown below), so that using node-based MIPs appears natural in this case.

Definition 2.1 (Node separators). *For two distinct nodes k and ℓ from V, a subset of nodes $N \subseteq V \setminus \{k, \ell\}$ is called (k, ℓ)-separator if and only if after eliminating N from V there is no (k, ℓ) path in G. A separator N is minimal if $N \setminus \{i\}$ is not a (k, ℓ)-separator, for any $i \in N$. Let $\mathcal{N}(k, \ell)$ denote the family of all (k, ℓ)-separators.*

Note that in order to make sure that a subset of chosen nodes is connected, it is sufficient to impose connectivity between the pairs of terminals (due to the minimization of the objective function). Therefore, we are mainly interested in separators between pairs of terminals.

Let $\mathcal{N}_\ell = \cup_{k\in T, k\neq\ell} \mathcal{N}(k, \ell)$ be the family of all such separators with respect to a node $\ell \in T$. We will refer to elements from $\mathcal{N}_\ell$ as ℓ-separators. Finally, let $\mathcal{N} = \cup_{\ell\in T} \mathcal{N}_\ell$ be the set of all node subsets that separate two arbitrary terminals.

Let us assume that we are dealing with an undirected graph G with node revenues p_v and uniform edge costs $c_e = c$ for all $e \in E$.

In order to derive a node-based model, we will first shift edge costs into node costs as follows:

$$\tilde{c}_v = c - p_v, \quad \forall v \in V.$$

Let $P = \sum_{v \in V} p_v$ be the sum of all node revenues in G. Binary node-variables y_v, $\forall v \in V$, will be set to one if node v is part of the solution, and the node-based MIP model can be derived as follows:

$$(\mathrm{PCSTP}_y) \qquad \min \sum_{v \in V} \tilde{c}_v y_v + (P - c) \tag{7}$$

$$y(N) \geq y_i + y_j - 1 \qquad \forall i, j \in T, i \neq j, \quad \forall N \in \mathcal{N}(i,j) \tag{8}$$

$$y_v = 1 \qquad \forall v \in T_r \tag{9}$$

$$y_v \in \{0,1\} \qquad \forall v \in V \setminus T_r \tag{10}$$

where $y(N) = \sum_{v \in N} y_v$.

Connectivity constraints (8) are used to ensure connectivity of the underlying solution. Basically, whenever two distinct terminals $i, j \in T$ are part of a solution, at least one node from any separator $N \in \mathcal{N}(i,j)$ has to be chosen as well, in order to ensure that there exists a path between i and j.

There is a difference between our model and the one considered in [2], where a more general MWCS variant on digraphs has been studied. In this latter variant, a root node needs to be established, i.e., a node r with in-degree zero and such that there is a directed path from r to any node j being part of the solution. Consequently, an additional set of node variables was needed to locate the root, separators were defined on digraphs, and connectivity constraints have been lifted with respect to root variables. Since our input graphs are undirected, we rely on the undirected model to keep the number of variables as small as possible. Only very recently, connectivity constraints (8) were given more attention in the literature concerning polyhedral studies. In particular, [25] studies the connected subgraph polytope, involving node variables only, and show that (8) define facets if and only if N is a minimal separator separating i and j.

Node-degree inequalities. The following node-degree inequalities are also valid for our model:

$$y(A_i) \geq \begin{cases} y_i, & \text{if } i \in T \\ 2y_i, & \text{otherwise} \end{cases} \tag{11}$$

where $A_i = \{v \in V \mid \exists \{v,i\} \in E\}$ is the set of all neighboring nodes of i. For terminals, these constraints ensure that at least one of their neighbors is part of the solution (assuming $\sum_{v \in V} y_v \geq 2$, which can be safely assumed after preprocessing single-node solutions). Clearly, they are just a special case of the inequalities (8), but can be used to initialize the model for a branch-and-cut approach. For the remaining nodes, constraints (11) make sure that each such node that belongs to a solution will be used as an intermediate node, i.e., at least two of its neighbors have to be included in the solution as well. These constraints are not implied by (8), in fact they can improve the quality of lower bounds of the original model.

It is sufficient to consider only minimal separators in inequalities (8) (since they dominate the remaining ones). In order to derive minimal separators associated to node-degree constraints, we observe that nodes from $V \setminus T$ that are only adjacent to i and other nodes from A_i do not play a role in connecting i to the remaining terminals. Let J be the set of all such neighbors of a given $i \in T$, i.e., $J = \{j \in A_i \setminus T \mid A_j \subseteq A_i \cup \{i\}\}$. If $V \setminus A_i$ contains other terminals, then $A_i \setminus J$ is a minimal i-separator, and we can correspondingly strengthen the node-degree inequalities (11).

Finally, observe that for potential terminals $i \in T_p$, the node-degree inequality can be lifted as follows:

$$2 \sum_{v \in A_i : c_{vi} < p_i} y_v + \sum_{v \in A_i : c_{vi} \geq p_i} y_v \geq 2y_i \quad \forall i \in T_p.$$

2-Cycle inequalities. Observe the following: if node $i \in V$ is adjacent to a node $j \in T_p$, so that $c_{ij} < p_j$, then if i is part of the optimal solution, j has to be included as well, i.e.,

$$y_i \leq y_j \quad i \in V, j \in T_p, c_{ij} < p_j \tag{12}$$

2.1 Separation of connectivity constraints

Whenever we want to cut off a fractional solution $\tilde{y}$, we can separate the connectivity cuts (8) by applying a maximum flow algorithm. For each pair of distinct terminals (i, j) such that $\tilde{y}_i + \tilde{y}_j - 1 > 0$, one would need to find a minimum (i, j)-cut in a support digraph D which is constructed as follows. First, each edge $e \in E$ is replaced by two directed arcs. Then, each node $v \in V \setminus \{i, j\}$ is replaced by an arc (v', v'') whose capacity is defined as $\tilde{y}_v$, all arcs entering v are now directed into v', and all arcs leaving v are now directed out of v''. Capacities of these arcs are set to ∞. Since all arcs except the node-arcs are of infinite capacity, the maximum (i, j)-flow returns the desired violated connectivity constraint.

According to our computational experience, however, the above procedure is rather time consuming (all terminal pairs need to be considered, and for each pair, the maximum flow is calculated). As there is always a certain trade-off between the quality of lower bounds obtained by separating fractional points and the associated computational effort, we refrain from the separation of fractional points in our default implementation.

Consequently, to ensure the validity of our branch-and-cut approach, we need to cut off infeasible integer points enumerated during the branching procedure (or detected by the heuristics of the MIP solver, given that the solver was not provided a complete information about the structure of the problem). Infeasible points are cut off by means of a `LazyCutCallback` in our setting based on the commercial MIP solver IBM ILOG CPLEX. For a given pair of distinct terminal nodes $i, j \in T$ such that $\tilde{y}_i = \tilde{y}_j = 1$, our separation procedure runs in linear time (with respect to $|E|$) and works as outlined below.

To derive our algorithm, we use the following well-known property of node separators (see, e.g., [17]):

Lemma 2.1. *Let $N \in \mathcal{N}(i, j)$ be an (i, j)-separator for $i, j \in T$, $i \neq j$, and let C_i and C_j be connected components of $G - N$ such that $i \in C_i$, $j \in C_j$. Then N is a minimal (i, j) node separator iff every node in N is adjacent to at least one node in C_i and to at least one node in C_j.*

Let $\tilde{y}$ be an integer solution, and let $G_{\tilde{y}} = (V, E_{\tilde{y}})$ denote the support graph induced by $\tilde{y}$, where

$$E_{\tilde{y}} = \{\{i, j\} \in E \mid \tilde{y}_i = \tilde{y}_j = 1\}$$

If $\tilde{y}$ is infeasible, then $G_{\tilde{y}}$ contains at least two connected components, say C_i and C_j, with $i \in C_i$, $j \in C_j$, and $\tilde{y}_i = \tilde{y}_j = 1$. Let $A(C_i)$ be the set of neighboring nodes of C_i in G, i.e., $A(C_i) = \{v \in V \setminus C_i \mid \exists \{u, v\} \in E, u \in C_i\}$. Clearly, $\{i, j\} \notin E$ and hence $A(C_i) \in \mathcal{N}_i$. However, $A(C_i)$ is not necessarily a minimal (i, j)-separator, and Algorithm 1 below describes how to derive a minimal separator starting from $A(C_i)$.

Algorithm 1: A linear time algorithm for detecting a minimal separator between two components C_i and C_j in $G_{\tilde{y}}$.

```
Data: Infeasible solution defined by a vector $\tilde{y} \in \{0,1\}^n$ with $\tilde{y}_i = \tilde{y}_j = 1$, $C_i$ being the connected component of
      $G_{\tilde{y}}$ containing $i$, and $j \notin C_i$.
Result: A minimal separator $N$ that violates inequality (8) with respect to $i$, $j$.
Delete all edges in $E[C_i \cup A(C_i)]$ from $G$
Find the set $R_j$ of nodes that can be reached from $j$
Return $N = A(C_i) \cap R_j$
```

In practice, set R_j can be found by just running a standard Breadth-First Search (BFS) on the original graph G, starting from node j, with the additional rule that nodes in $A(C_i)$ are never put in the BFS queue.

Proposition 2.1. *Algorithm 1 returns a minimal separator $N \in \mathcal{N}(i, j)$ in time $O(|E|)$.*

Proof. By definition of N, i and j are not connected in $G - N$. To see that N is a minimal (i, j)-separator, consider $G - N$ and let C_i' and C_j' be two connected components, containing i and j, respectively. Clearly, $C_i \subset C_i'$ and $C_j' = R_j \setminus N$. Hence, by Lemma 2.1, it follows that N is minimal. □ □

In case $p > 2$ connected components exist in $G_{\tilde{y}}$ (each of them containing at least one terminal), one can repeat the procedure described in Algorithm 1 for each pair of distinct components C_i and C_j.

We conclude this section by observing that, for the pure STP case with real terminals only, connectivity constraints translate into

$$y(N) \geq 1, \quad \forall N \in \mathscr{N}$$

where $\mathscr{N}$ is the family of all separators between arbitrary real terminal pairs. Our model can therefore be interpreted as set covering problem with an exponential number of elements to be covered. As demonstrated by our computational experiments, this property can be exploited to derive specialized set covering heuristics for pure STP instances with uniform costs, with a significant performance boost.

3 Algorithmic framework

The proposed node-based model can be solved by means of a branch-and-cut (B&C) algorithm, to be initialized with a high-quality feasible solution and with a suitable set of relevant connectivity constraints.

Our initial MIP model, called the *basic model* in what follows, is made by (7), (9), (10), plus the node-degree inequalities (11) and the 2-cycle inequalities (12).

The overall algorithmic framework is shown in Algorithm 2. In an *initialization phase*, an initial solution pool $\mathscr{S}_{\text{init}}$ is generated by means of some problem-dependent heuristics. In a subsequent *local branching phase*, multiple calls of the B&C algorithm are used to explore the neighborhood of starting solutions chosen from the solution pool $\mathscr{S}_{\text{init}}$. All connectivity constraints separated during this phase are globally valid, hence they are stored in a global cut pool *CutPool*, and added to the initial MIP model before each B&C re-execution. The incumbent solution (denoted by *Sol*) is updated correspondingly. A detailed description of the `LocalBranching` procedure and of its parameters is given in Section 3.1.

The local branching phase implements a multiple restart policy (with different seed values and a maximum number of iterations *LBMaxRestarts*), intended to gather relevant information about the problem at hand, namely good primal solutions and a relevant set of connectivity constraints; see e.g. [15] for a recent application of multiple restarts to MIPs with uncertainty. The availability of such information at the root node of each B&C re-execution turns out to be very important, as it triggers a more powerful preprocessing as well as the generation of a bunch of useful general-purpose cuts (in particular, $\{0, 1/2\}$-cuts [3, 6]) based on the problem formulation explicitly available on input.

The algorithm terminates after proving the optimality, or after reaching the given time limit.

3.1 Local branching

Local branching (LB) has been proposed in [14] as a solution approach that uses the power of a general-purpose MIP solver as a black box to strategically explore promising solution subspaces. LB is in the spirit of large-neighborhood search metaheuristics, with the main difference that the underlying local search black-box algorithm is a MIP with a specific local branching constraint that restricts the search for an optimal solution within a certain neighborhood of a given reference solution.

The LB framework is built on top of our B&C solver. As already mentioned, our solver deals with two sets of inequalities: those in the *basic model*, that are always part of the model, and connectivity constraints (8) that are dynamically separated and stored in a global *CutPool* to be used in every subsequent B&C call.

Given a reference solution *Sol*, let $W_1 = \{v \in V \mid v \in Sol\}$ and $W_0 = V \setminus W_1$. The *symmetric* local branching constraint makes sure that the new solution is within a given radius r from the solution *Sol* with respect to the Hamming distance between the two solutions, i.e.

$$\sum_{v \in W_0} y_v + \sum_{v \in W_1} (1 - y_v) \leq r$$

Alternatively, one may consider an *asymmetric* local branching constraint, requiring that the new solution contains at least $|Sol| - r$ nodes from *Sol*:

$$\sum_{v \in W_1} (1 - y_v) \leq r. \tag{13}$$

Algorithm 2: Proposed algorithmic framework.

```
Data: Instance $I$ of the STP/PCSTP/NWSTP/MWCS, restart limit $LBMaxRestarts$, time limit $TimeLim$, local
      branching parameters ($r_{min}$, $r_{max}$, $r_{delta}$, $LBMaxIter$, $LBSolLim$, $LBTimeLim$).
Result: A (sub)-optimal solution $Sol$.
$\mathscr{S}_{\text{init}} = \texttt{InitializationHeuristics}()$
$Sol = argmin_{Sol' \in \mathscr{S}_{\text{init}}} cost(Sol')$
$k = 1$, $CutPool = \emptyset$, $SolLim = \infty$
while ($k \leq LBMaxRestarts$) and (time limit not exceeded) do
    Choose $Sol'$ from the solution pool $\mathscr{S}_{\text{init}}$.
    $(Sol', CutPool') = \texttt{LocalBranching}(I, Sol', CutPool, seed, r_{min}, r_{max}, r_{delta},$
                                    $LBMaxIter, LBTimeLim, LBSolLim)$
    $CutPool = CutPool \cup CutPool'$
    if $cost(Sol') < cost(Sol)$ then
        $Sol = Sol'$
    end
    $k = k + 1$, change $seed$.
end
$Sol = \texttt{BranchAndCut}(I, Sol, CutPool, TimeLim, SolLim)$
return $Sol$
```

Notice that, for a fixed radius, the neighborhood of the asymmetric version is larger and leads to potentially better solutions—though it is more time consuming to explore. For example, for $r = 0$ the asymmetric version is equivalent to fixing to 1 all the variables from *Sol*, so that many feasible solutions are still available even in the 0-neighborhood around *Sol*. After some preliminary tests, we decided to use the asymmetric LB constraint (13) in our implementation, with a small radius r ranging from 10 to 30. Working with a small radius is indeed crucial for the success of proximity methods such as LB, as recently pointed out in [16].

The LB framework is shown in Algorithm 3. Since the goal of the B&C in this context is to quickly find high-quality solutions, we do not necessarily search for an optimal solution within the given neighborhood, but we rather impose limits on the number of incumbent solutions found (*LBSolLim*), and a time limit per iteration (*LBTimeLim*).

The neighborhood is systematically explored by starting with an initial radius r_{min}, and increasing it by r_{delta} each time the subproblem could not provide an improved solution. Each time an improved solution is found, the neighborhood radius is reset to r_{min}. The whole framework is executed until a given number of iterations (*LBMaxIter*) is reached, or the radius exceeds r_{max}. Note that the radius limit r_{max}, in combination with a consistent increase of the current radius, implicitly imposes a limit on the overall number of iterations without any improvement.

Algorithm 3: Local Branching.

```
Data: Instance $I$ of the STP/PCSTP/NWSTP/MWCS, starting solution $Sol$, $CutPool$, $seed$, lower and upper
      bound for radius ($r_{min}, r_{max}$), radius step ($r_{delta}$), maximum n. of iterations $LBMaxIter$, time limit
      $LBTimeLim$, solution limit $LBSolLim$.
Result: Improved solution $Sol$, enlarged cut pool $CutPool$.
$r = r_{min}$, $k = 1$
while ($k \leq LBMaxIter$) and ($r \leq r_{max}$) do
    Add LB constraint (13) centered on $Sol$ with radius $r$
    $(Sol', CutPool') = \texttt{BranchAndCut}(I, Sol, CutPool, LBTimeLim, LBSolLim)$
    Remove the LB constraint from the current model
    if $cost(Sol') < cost(Sol)$ then
        $Sol = Sol'$, $r = r_{min}$
    else
        $r = r + r_{delta}$
    end
    $CutPool = CutPool \cup CutPool'$
    $k = k + 1$
end
return (Sol, CutPool)
```

3.2 Benders-like (Set Covering) heuristic

Local branching has a primal nature, in the sense that it produces a sequence of feasible solutions of improved cost. In addition, it needs a starting feasible solution, that in some cases can be time consuming to construct. As a matter of fact, for some very large/hard classes of instances we found that a dual approach is preferable, that produces a sequence of infeasible (typically, disconnected) solutions and tries to repair them to enforce feasibility. Algorithm 4 illustrates a general dual scheme that can be viewed as a heuristic version of the well-known Benders' exact solution approach to general MIPs.

Algorithm 4: A conceptual Benders-like heuristic.

```
Data: Instance $I$ of the STP, time/iteration limits.
Result: Feasible solution $Sol$, cut pool $CutPool$.
$Sol$ = dummy solution of very large cost
$CutPool = \emptyset$
while (time/iteration limit not exceeded) do
    Heuristically solve a set covering relaxation of the current model (including all cuts in $CutPool$) and let $Sol^R$
    be the (possibly disconnected) solution found
    Add the LB constraint (13) centered on $Sol^R$ to the unrelaxed model
    $(Sol', CutPool') = \texttt{BranchAndCut}(I, Sol, CutPool, TimeLim, SolLim)$
    Remove the LB constraint from the unrelaxed model
    if $cost(Sol') < cost(Sol)$ then
        $Sol = Sol'$
    end
    $CutPool = CutPool \cup CutPool'$
end
return (Sol, CutPool)
```

In our experiments, we found that the above approach works very well for uniform STP instances of very large size, for which the standard MIP approach seems not very appropriate as even the LP relaxation of the model takes an exceedingly large computing time to be solved. Our *set-covering based heuristic* is an implementation of Algorithm 4 for these hard instances, and is based on their set covering interpretation. Indeed, as already observed, the basic model turns out to be a compact set covering problem where columns correspond to Steiner nodes, rows to real terminals not adjacent to any other real terminal, and column j covers row i iff $\{i, j\} \in E$. The approach can be outlined as follows.

At each iteration of the while loop, the relaxation to be heuristically solved is constructed through a procedure that automatically extracts a set covering relaxation from the current model. This is done by simply (1) projecting all fixed variables (including y variables for hard terminals) out of the model, and (2) skipping all constraints, if any, that are not of type $y(S) \geq 1$ for some node set S.

We then proceed by heuristically solving the set covering relaxation through an implementation of the Caprara-Fischetti-Toth (CFT) heuristic [7, 8]. This is a very efficient set covering heuristic based on Lagrangian relaxation, that is specifically designed for hard/large instances.

Given a hopefully good set covering solution Sol^R, we *repair* it in a very aggressive way by introducing a local branching constraint in asymmetric form with radius $r = 0$, and then by applying our B&C solver (with its ad-hoc connectivity cut separation) with a short time/node limit. As already observed, this local branching constraint in fact corresponds to fixing $y_j = 1$ for all j such that $Sol^R_j = 1$. As a result, the size/difficulty of the MIP model after fixing is greatly reduced, hence the node throughput of the B&C solver becomes acceptable even for large instances—while setting a larger radius would not result in a comparable speedup.

All violated connectivity cuts generated by the B&C procedure are added to *CutPool* and hence to the current model. This makes solution Sol^R (if disconnected) infeasible even for the next set covering relaxation and thus the procedure can be iterated until an overall time/iteration limit is reached.

To improve diversification, our implementation uses the following two mechanisms:

- the procedure that extracts the set covering model makes a random shuffle of the rows/columns, so as to affect in a random way the performance of the CFT heuristic;
- before the repairing phase, we randomly skip (with a uniform probability of 20%) some variable fixings, meaning that approximately 80% of the variables y_j's with $Sol^R_j = 1$ are actually fixed.

As such, the performance of our final heuristic (though deterministic) is affected by the initial random seed, a property that can be very useful to produce different solutions in a multi-start scheme.

Finally, we observe that our current CFT implementation is sequential and cannot exploit multiple processors. We therefore decided to also run the refining B&C in single-thread mode, thus obtaining an overall sequential code that can be run in parallel and with different random seeds on each single core (in the multi-thread mode). The best found solution is finally returned.

This Benders-like heuristic is embedded in the overall algorithmic framework shown in Algorithm 2 as `InitializationHeuristics` for uniform STP instances on bipartite graphs with a large percentage of terminals. Indeed, these kinds of graphs are very regular and the basic model likely gives a reasonable approximation of the STP problem—in the sense that most connectivity constraints are automatically satisfied. In addition, the while-loop in Algorithm 2 that would apply standard local branching after the set-covering based heuristic is unlikely to be effective for these graphs, so we set $LBMaxIter = 0$ and skip it in this case.

We conclude this section by observing that the "relaxation" to be heuristically solved in the Benders-like scheme is not intended to produce valid dual bounds, as its purpose is to feed the refining procedure with good (possibly disconnected) solutions. As a matter of fact, one can think of the relaxation as a "blurred" version of the original problem, which retains some of its main features but is not bothered by too many details (namely, connectivity conditions) that would overload the model. Following this general idea, we implemented the following variant of our set-covering based heuristic, which is intended for non-uniform instances on bipartite graphs with a large percentage of terminals.

Given a non-uniform instance of the STP, it is transformed to a uniform instance by adapting its revenue and edge costs as follows: For each non-terminal $v \in V \setminus T_r$, its node cost $\tilde{c}_v$ is set to the average cost of its incident edges, i.e., $\tilde{c}_v = \frac{1}{|\delta(v)|} \sum_{e \in \delta(v)} c_e$. Next, all edge costs and node revenues are set to zero. Intuitively, edge cost information is moved into node costs, so as to get a "blurred" uniform instance on the same underlying graph. Note that these modified costs/revenues are only used within the CFT heuristic, while the original ones are used in the refining phase.

3.3 A unified solver and automatic parameter adjustments

To achieve the best performance over all different types of problem instances, we have implemented a unified solver that switches between the node-based model (in the following referred to as y-model) and the (x,y)-model presented in the introduction, depending on the instance properties. The overall algorithmic framework given in Algorithm 2 remains the same, with the main difference being the MIP model for the underlying B&C procedure. To this end, our solver contains a *filter* that analyzes the structure and costs of the input graph. According to these properties, the algorithm decides the actual MIP model, as well as the kind of initialization heuristics and preprocessing to apply. We note that in the proposed algorithm these rules are only used for the problem types STP, PCSTP and MWCS (which are transformed to their PCSTP representation). DC-STP and RPCSTP instances are solved solely by B&C on the (x,y)-model without any sophisticated settings. Table 1 shows all types of instance properties identified by the filter.

Table 2 lists general filter rules for model and parameter selection. By default, the y-model is applied to instances with uniform edge costs, while the (x,y)-model is applied to all others. For sparse, uniform graphs with a relatively low number of terminals, we switch to the (x,y)-model, as preliminary experiments have shown that the y-model is less efficient for this specific class. For the y-model, the MIP at the root node is restarted two times in order to generate more general purpose cutting planes, since the cutting plane generation procedures of CPLEX for such cuts are triggered again after a restart and produce additional cuts.

For the (x,y)-model a tailing-off bound is specified, which defines the allowed ratio between the lower bounds of two consecutive B&C iterations in the root node. In addition, a tailing-off tolerance value is defined, which specifies the number of times this bound is allowed to be violated in a row. If the number of violations exceeds the specified tolerance-parameter, the algorithm switches from separation of fractional solutions to branching. The two parameters are chosen based on graph density. Instances are roughly divided into two classes, sparse ($|E|/|V| \leq 3$) and dense ($|E|/|V| > 3$) graphs. The tailing-off bound is only activated for dense graphs, while for sparse graphs we avoid branching as long as possible. In addition, we identify very dense ($|E|/|V| > 5$) instances, in which we set a lower tailing-off tolerance than in the regular case.

Table 1: Instance properties identified by the filter procedure.

Property	Description
`uniform`	all arc weights have the same weight
`sparse`	edge-to-arc ratio $|E|/|V| \leq 3$
`dense`	edge-to-arc ratio $|E|/|V| > 3$
`verydense`	edge-to-arc ratio $|E|/|V| > 5$
`ratioT`	terminal ratio $|T|/|V|$
`big`	number of nodes $|V| > 10000$
`small`	number of nodes $|V| < 1000$
`hypercube`	all nodes have the same degree
`stp`	problem instance is of type STP
`xy-model`	the (x,y)-model has been selected by the filter
`bipartite`	the instance graph G is bipartite w.r.t. the node sets $V \setminus T$ and T

Table 2: General filter rules.

Rule	Applied settings
`uniform`	$\rightarrow$ y-model
`uniform` $\wedge$ `sparse` $\wedge$ `ratioT` < 0.1	$\rightarrow$ (x,y)-model
$\neg$`uniform`	$\rightarrow$ (x,y)-model
`dense`	$\rightarrow$ use tailing-off bound, high tolerance
`verydense`	$\rightarrow$ use tailing-off bound, low tolerance

Table 3 lists filter rules that select problem-specific settings and algorithms for STP instances. Given that the (x,y)-model has been selected by the filter, the model may be initialized by a set of connectivity cuts generated by a dual ascent algorithm [26], and with a starting solution generated by a parallel implementation of a partitioning heuristic (see [20]). Based on the terminal ratio, the dual ascent connectivity cuts are either added at the beginning of the B&C or separated. Similarly, flow-balance constraints and generalized subtour elimination constraints (GSEC) of size two may be chosen to be separated if the graph is large, sparse and has a low terminal ratio. Dense STP instances are instead preprocessed by using the special distance test [12].

Table 3: STP-specific rules.

Rule				Applied settings
`xy-model`	$\wedge$ `ratioT` < 0.01			$\rightarrow$ separate dual ascent cuts
`xy-model`	$\wedge$ `ratioT` ≥ 0.01			$\rightarrow$ add all dual ascent cuts
`xy-model`	$\wedge$ `ratioT` < 0.1	$\wedge$ `sparse`	$\wedge$ `big`	$\rightarrow$ separate flow-balance constr., GSECs of size 2
`xy-model`	$\wedge$ `big`	$\wedge$ `sparse`		$\rightarrow$ partition-based construction heuristic
`verydense`				$\rightarrow$ preprocessing (special distance test)

Table 4 lists additional rules for the selection of initialization heuristics. In the case of bipartite graphs, the set-covering heuristic is applied for both uniform and non-uniform instances to generate high-quality starting solutions. For non-uniform instances, each node v is assigned the weight $\frac{1}{|\delta(v)|}\sum_{e\in\delta(v)} c_e$ ("blurred" set-covering heuristic). We note that also in the case of PCSTP instances we choose to apply the blurred set-covering heuristic for the large non-uniform instances. A significant slowdown of the (x,y)-model has been observed for larger instances, at which point the blurred set-covering heuristic performs better. By default, the following settings are also applied when executing local branching: The primal heuristic is executed in every branch-and-bound node for the (x,y)-model and in each 1000 nodes for the y-model. The GSECs of size two in the (x,y)-model are separated and stored in the cut pool. The local branching time limit per iteration is set to 120 seconds. The neighborhood radius is initialized with 10 and increased by 10 per iteration up to a maximum of 30.

Table 4: Initialization heuristic rules.

Rule				Applied settings
`bipartite` $\wedge$	`uniform`			$\rightarrow$ set-covering heuristic
`bipartite` $\wedge$	$\neg$`uniform` $\wedge$	`stp`		$\rightarrow$ blurred set-covering heuristic
`hypercube` $\wedge$	$\neg$`uniform` $\wedge$	$\neg$`small` $\wedge$	`pcstp`	$\rightarrow$ blurred set-covering heuristic
$\neg$`bipartite`				$\rightarrow$ local branching

4 Computational results

Our algorithms are applied to the following problems from the DIMACS challenge: STP, (rooted) PCSTP, MWCS, and also degree-constrained STP (DCSTP). We next summarize the results we obtained on the set of hard (unsolved) cases of the SteinLib [19] instance library. Additionally, we consider non-trivial MWCS instances posted at the website of the DIMACS challenge. Detailed computational results, covering all nontrivial instances from the challenge are provided in the Appendix. Among these nontrivial instances, we distinguish between *easy* and *difficult* ones.

For this purpose, we first run all considered instances using a *pure B&C* approach, for which initialization heuristics and local branching are deactivated, effectively restricting Algorithm 2 to its final step. The filter rules listed in Tables 2 and 3 are applied and a time limit of one hour (with a fixed seed value) is imposed. All instances that remained unsolved by this approach are considered as *difficult*.

Our heuristic framework (consisting of the initialization and local branching phase, see Algorithm 2) is then applied to all *difficult* instances with a time limit of one hour (10 independent runs with different seeds).

The experiments were performed on a cluster of computers, each consisting of 20 cores (Intel E5-2670v2 2.5 GHz) and with 64GB RAM available for the 20 cores. Reported computing times are in wall-clock seconds. To limit the overall time needed to complete our experiments, we decided to allow up to five simultaneous 4-core runs on the same computer, which however implies a significant slowdown due to shared memory.

All algorithms have been implemented in C++ and compiled with GCC 4.9.2. For data structures we used OGDF [23] and the dtree library [13]. CPLEX 12.6 was used as MIP-solver with an imposed memory limit of 12GB RAM.

4.1 Implementation details of the (x,y)-model

The model described in Section 1 has been implemented together with several algorithmic enhancements, which are detailed in the following paragraphs.

Initialization heuristic: A pool of initial feasible solutions is constructed as follows. Several terminals are chosen as root nodes, for each of which a solution is calculated by applying the shortest path Steiner tree heuristic (see, e.g., [4]). In the PCSTP case, for a small number of iterations the set $T_p \cup T_r$ is perturbed and subsets of terminals of different size are considered as fixed terminals T'. Then for each chosen set T', the same construction heuristic as for the STP is applied. Each solution is also improved through a local search (see below).

For sparse, (almost) planar non-uniform instances of the STP, our framework computes an additional, enhanced initial solution by applying a parallel variant of the partitioning heuristic described in [20]. Based on a randomly chosen solution from the pool, the input graph is partitioned into a set of smaller subgraphs (containing terminals and their closest Steiner nodes). The STP is then solved to optimality (or with a small time limit) on each of these subgraphs independently. The obtained disconnected (and thus, infeasible) solution is then repaired by a shortest-path like heuristic, followed by a local search. When run in a multi-thread mode, solving the STP on each of the subgraphs is assigned to a single thread.

Primal heuristic: We apply a few rounds of rounding on the set of y variables, with different threshold values. Rounded up variables again define a set of fixed terminals T' on which we run the shortest path Steiner tree heuristic (with a modified cost function that reflects the LP-values at the current branch-and-bound node) and prune Steiner leaves.

Local search heuristics: Solutions obtained by the initialization or the primal heuristic are further improved by applying several local search procedures: Key-Path-Exchange, (Key-)Node-Removal and Node-Insertion. For further implementation details about these heuristics, see [24]. We only accept strictly improving moves, except for uniform and almost uniform instances, for which (due to existence of many symmetric solutions) all moves with non-decreasing objective values are performed. We consider an instance as almost uniform if the absolute difference between the minimum and maximum edge weight is less than ten.

Additional valid inequalities: In [21] the authors observe that the (x,y)-model for PCSTP contains a lot of symmetries, and propose to get rid of them by fixing the terminal node with the smallest index (among those taken in the solution) to be the root node. This is imposed by adding additional asymmetry constraints. The latter constraints added by [21] were given in a disaggregated form, whereas in our implementation we add their stronger, aggregated variant: $\sum_{i>j} x_{ri} \leq 1 - y_j$, for all $j \in T$. We also add 2-cycle inequalities (12) to our starting model.

Separation algorithms: For fractional solutions, connectivity constraints are separated by the calculation of the maximum flow (implemented through `UserCutCallback` of CPLEX). Algorithm (5) shows the separation procedure for the (x,y)-model. First, a small value ε is added to the fractional LP solution x^* of arc variables, which are used as capacities for the maximum flow algorithm. This approach ensures that the separated cuts are of minimum cardinality, and is known to reduce the number of required cutting plane iterations [18, 21]. Subsequently, the maximum flow is computed between the root and each potential terminal $i \in T_p$. Among the two cut sets (S_r and S_i, $r \in S_r$, $i \in S_i$) returned by the maximum flow algorithm of [9], we choose the one closer to the terminal, i.e., S_i (cut sets closer to the root typically involve similar edges, and hence may imply many redundant cuts). A technique referred to as nested/orthogonal cuts [18, 22] is applied to generate more diverse cuts in each iteration. Whenever a violated cut is identified, the capacities of the associated arcs are set to one, which ensures that the intersection between violated cuts separated within the same cutting plane iteration is empty.

Algorithm 5: Separation procedure for the (x,y)-model.

```
Data: Fractional LP solution $(x^*, y^*)$, upper bound $UB$ on the objective value.
Result: A set of violated connectivity cuts $C$.
$x' = x + \varepsilon$
Find the set of potential terminals $W \subset T_p$ reachable from $r$.
for $i \in T_p \setminus W, y_i \geq 0.5$ do
    $f = \texttt{MaxFlow}(G, x', r, i, S_r, S_i)$
    Detect cut $\delta^-(S_i)$ such that $f = x'(\delta^+(S_i)), i \in S_i$.
    if $f < y_i$ then
        if $revenue(S_i) > UB$ and problem type is PCSTP then
            Add the violated cut $x(\delta^-(S_i)) \geq 1$ to $C$.
        else
            Add the violated cut $x(\delta^-(S_i)) \geq y_i$ to $C$.
        end
        Add all $T_p$ reachable from $i$ to $W$.
    end
    $x'_{ij} = 1, \quad \forall (i,j) \in \delta^-(S_i)$
end
return $C$
```

To further speed up separation, we skip the maximum flow calculation for terminals that are already reachable from the root in the support graph (node set W in the algorithm). We do not separate cuts recursively [21, 22], but instead separate at most one connectivity constraint for each terminal per iteration. Avoiding the separation of fractional solutions while branching and adding only one cut per terminal seemed to perform best on average in preliminary experiments.

For the PCSTP we only separate cuts between the root and those terminals i such that $y_i \geq 0.5$ in the given fractional solution to separate. Furthermore, instead of adding a connectivity constraint (3) as in [21], we may replace the right-hand side with one if the revenue on the sink side of the cut is larger than the objective of the incumbent solution. This implies that in an optimal solution, at least one more node on the sink side has to be connected to the root.

Infeasible integer solutions are separated by searching for connected components in the support graph. For each subset S inducing a connected component of an infeasible solution, a connectivity constraint $x(\delta^-(S)) \geq y_i$ is added to the model, for $i \in S$ with the highest revenue.

4.2 Results for uniform STP instances GAPS and SP

For the subgroup “skutella” (s3-s5) of the artificially generated uniform instance set GAPS for the STP, LP-gaps of the standard connectivity-based (x,y)-model are large. Standard MIP approaches for these instances have difficulties in closing the integrality gap. Table 5 reports our results obtained on instances s1 to s5 from GAPS, and clearly demonstrates the power of our y-model.

Additionally, for two previously unsolved instances from the set SP (with uniform edge costs as well), namely w13c29 and w23c23, we provide optimal values. For these two latter instances, both models were able to prove the optimality, with significant speed-ups achieved by the y-model. Table 6 reports results using ten different seeds. The y-model outperforms the (x,y)-model with respect to the best and the average running times for both instances, the average speed-up ranging between one and two orders of magnitude. For both models, the running times vary greatly

Table 5: Uniform STP instances from the GAPS dataset. Proven optimal solutions in boldface. Column Time gives the computing time for proving the optimality (or, the time limit, otherwise). Columns UB and LB show upper and lower bounds obtained by the (x,y)-model, within the time limit of two hours, respectively. (*) Reached memory limit of 12GB within the specified time limit.

				y-model		(x,y)-model			
Instance	$\|V\|$	$\|E\|$	$\|T\|$	OPT	Time (s.)	UB	LB	Gap (%)	Time (s.)
s1	64	192	32	**10**	0.0	10	10	0.00	0.0
s2	106	399	50	**73**	0.0	73	73	0.00	1.4
s3	743	2947	344	**514**	0.2	514	505	1.78	1090.6*
s4	5202	20783	2402	**3601**	1.3	3601	3523	2.21	3444.8*
s5	36415	145635	16808	**25210**	22.3	25210	24056	4.80	7200.0

depending on the chosen seed value. This is due to the fact that solving the instance to optimality is highly dependent on finding an optimal solution.

Table 6: Uniform STP instances from the SP dataset. Proven optimal solutions in boldface. Previous best known solutions given in brackets. Runs have been performed using ten seeds. The Time columns give the best, average and the standard deviation for running time. For the y-model, no primal heuristics have been used besides the ones by CPLEX. For the (x,y)-model, we use local search (cf. Section 4.1).

					y-model Time (s.)			(x,y)-model Time (s.)		
Instance	$\|V\|$	$\|E\|$	$\|T\|$	OPT	BEST	AVG	STD	BEST	AVG	STD
w13c29	783	2262	406	**507** (508)	0.3	0.9	0.5	14.5	38.3	30.0
w23c23	1081	3174	552	**689** (694)	43.9	132.6	60.0	183.9	2600.2	1362.6

4.3 Results for MWCS instances

The MWCS can be transformed into the PCSTP with uniform edge costs (see [2]). We have tested both y- and (x,y)-model on the MWCS dataset, and the obtained computational results (for the most challenging instances) are reported in Table 7. Results on the JMPALMK dataset are available in the Appendix - those instances have all been solved within one second.

The results show that the y-model outperforms the (x,y)-model on all instances with relatively dense graphs by roughly an order of magnitude. In contrast, on the sparser metabol_expr_mice instances, the performance is extremely erratic, ranging from one second to over an hour, while the very same instances are always solved to optimality within a few seconds by the (x,y)-model. A closer inspection of the test run data shows that for instance metabol_expr_mice_1, the y-model enumerates more than four million branch-and-bound nodes, while the optimal solution is found after one hour of computation. Additional test runs have been performed with an optimal solution provided from the start, however the runtime did not decrease significantly. A likely explanation is thus that at least for this instance the bounds provided by the y-model are simply too weak to be competitive to the (x,y)-model.

4.4 Results for STP and PCSTP instances

Table 8 lists all instances from the PUC dataset solved within an hour by the pure B&C (without the `InitializationHeuristics` and the local branching phase, basically, by invoking only the last line of Algorithm 2) with parameters chosen by the filtering procedure. These instances were therefore classified as "easy" for our framework. Using the same configuration, most benchmark instances from the SteinLib could also be solved to optimality.

Table 7: ACTMODPC instances from the DIMACS website. The instances have been transfromed from the MWCS to the PCSTP. For these runs no primal heuristics have been used besides the ones by CPLEX.

					y-model		(x,y)-model	
Instance	$\|V\|$	$\|E\|$	$\|T\|$	OPT	Time (s.)	Gap (%)	Time (s.)	Gap (%)
drosophila001	5226	93394	5226	8273.982630	8.0	**0.00**	86.1	**0.00**
drosophila005	5226	93394	5226	8121.313578	9.5	**0.00**	76.3	**0.00**
drosophila0075	5226	93394	5226	8039.859460	7.5	**0.00**	68.5	**0.00**
HCMV	3863	29293	3863	7371.536373	1.0	**0.00**	6.1	**0.00**
lymphoma	2034	7756	2034	3341.890237	0.3	**0.00**	1.2	**0.00**
metabol_expr_mice_1	3523	4345	3523	11346.927189	5965.8	**0.00**	1.1	**0.00**
metabol_expr_mice_2	3514	4332	3514	16250.235191	1.2	**0.00**	1.6	**0.00**
metabol_expr_mice_3	2853	3335	2853	16919.620407	4.0	**0.00**	0.9	**0.00**

Due to space reasons the other results are not reported here, but are available in the Appendix.

Table 8: STP PUC instances solved by B&C with parameters chosen automatically by the filter procedure (y-model for hc6u, hc7u, bip42u, bipe2u; (x,y)-model for all others). Columns 'Nodes', 'Time' and 'Time-t' list the number of B&B nodes, computation time of the best solution, and total runtime, respectively.

Instance	$\|V\|$	$\|E\|$	$\|T\|$	LB	UB	Gap (%)	Nodes	Time (s.)	Time-t (s.)
bip42u	1200	3982	200	236	236	0.00	1603599	466.5	1038.0
bipe2p	550	5013	50	5616	5616	0.00	4333	59.1	76.0
bipe2u	550	5013	50	54	54	0.00	123	0.1	0.5
cc3-4p	64	288	8	2338	2338	0.00	45113	1.0	43.9
cc3-4u	64	288	8	23	23	0.00	4224	0.0	7.4
cc5-3u	243	1215	27	71	71	0.00	220216	0.6	1940.3
cc6-2p	64	192	12	3271	3271	0.00	694	2.6	13.8
cc6-2u	64	192	12	32	32	0.00	108	0.0	17.0
cc6-3u	729	4368	76	197	197	0.00	10814	371.1	389.9
hc6p	64	192	32	4003	4003	0.00	4004	0.3	2.1
hc6u	64	192	32	39	39	0.00	1142	0.1	0.2
hc7u	128	448	64	77	77	0.00	816615	0.1	1372.7

For the remaining, more difficult instances, Tables 11 to 16 list the most important results after applying the set-covering heuristic or local branching. These datasets have been selected from the set of instances unsolved after one hour by B&C (detailed results are available in the Appendix). Each run has been computed by starting the algorithm with ten different seeds, each with one hour time limit. The choice between the local branching and set-covering heuristic is performed by the filter as described in Section 3.3. The final B&C step of the framework is always skipped.

Each table is structured as follows: The first four columns list the instance name, the number of nodes, edges and terminals. The next pair of columns (BEST) shows objective value and time of computation for the best found solution. The following pairs (AVG and STD) list the average and standard deviation of these two values over all ten runs. For tests on previously known STP datasets (PUC, I640) the column 'Impr.' lists the improvement w.r.t. the best known published solution (by August 1st, 2014) according to the DIMACS challenge website [10]. In all other cases, the improvement is given w.r.t. the best primal solutions produced during the exact runs after one hour. As is to be expected, given the same time limit, the heuristic procedures manage to outperform the pure B&C with respect to

Table 9: Results for instances where the best known upper bounds have been improved by increasing the number of parallel threads and/or the time limit. For instance hc12u2, no upper bounds have been reported before the challenge.

Problem	Instance	Best UB	Previous UB	Time (s.)	#Threads
STP	hc11u	**1144**	1154	474	8
STP	hc12u	**2256**	2275	4817	8
STP	hc12p	**236158**	236949	4411	4
PCSTP	hc12u2	**1492**	–	632	16

Table 10: Results for previously unsolved instances that have been solved to optimality by increasing the number of parallel threads, time limit and memory limit.

Problem	Instance	OPT	Time (s.)	#Threads	#Memory (GB)
STP	i640-313	**35535**	16012	20	32
PCSTP	i640-313	**32401**	15358	20	32

the computed upper bounds. However, the comparison on notoriously difficult instances for the STP (PUC and I640) shows the heuristics' effectiveness, as several upper bounds could be improved. The most noticable and consistent improvement can be observed for the hypercube instances, to which the set-covering heuristic has been applied.

Note that the tables report the results obtained within the time limit of one hour only. Table 9 shows that by extending the time limit, and/or by using more than four threads in parallel, the obtained values can further be improved. Table 10 shows that we were able to prove optimality for the following difficult instances, where according to the SteinLib website [19] the instance i640-313 has previously been unsolved (for the PCSTP version of i640-313, we report similar performance).

5 Conclusions

We have presented a simple model for the Steiner tree problem, involving only node variables. Besides drastically reducing the number of the required variables, the removal of edge variables avoids a number of issues related to equivalent (possibly symmetrical) trees spanning a same node set. In this view, we are "thinning out" the usual edge-based model with the aim of getting a more agile framework. Our model is mainly intended for instances with uniform edge costs, but one could use it to derive a heuristic for tackling the general case (left for future studies). Computational results show that our approach can dramatically improve the performance of an exact solver, and in some cases converts very hard problems into trivial-to-solve ones. At the recent DIMACS challenge on Steiner trees (see [11]), the proposed algorithmic framework, which switches intelligently between the model involving only node-variables and a second model that uses both node and edge variables, won most of the categories of both, exact and heuristic challenge, for the STP, PCSTP, MWCS and DCSTP.

Table 11: PUC STP instances. Column 'Impr.' shows the improvement w.r.t. the previous best known values published on the DIMACS challenge website [10]. Results computed by the set-covering heuristic in the case of hypercube (hc) and bipartite (bip) instances, otherwise through (x, y)-model-based local branching. Improved solutions given in boldface. Time limits set to one hour.

				BEST		AVG		STD		
Instance	$\|V\|$	$\|E\|$	$\|T\|$	UB	Time (s.)	UB	Time (s.)	UB	Time (s.)	Impr.
bip42p	1200	3982	200	24657	38.4	24660.8	664.6	2.0	1125.2	0
bip52p	2200	7997	200	24549	805.6	24566.9	1403.0	13.9	1357.4	-14
bip52u	2200	7997	200	**233**	1390.1	233.8	287.9	0.4	598.0	1
bip62p	1200	10002	200	22906	3.6	22907.0	55.4	1.1	80.8	-36
bip62u	1200	10002	200	**219**	6.2	219.0	12.3	0.0	5.0	1
bipa2p	3300	18073	300	**35355**	547.2	35360.9	1342.9	4.4	879.6	24
bipa2u	3300	18073	300	**337**	185.1	337.0	310.9	0.0	215.2	4
cc10-2p	1024	5120	135	**35257**	875.4	35353.2	704.9	75.1	705.2	122
cc10-2u	1024	5120	135	342	206.3	342.6	818.0	0.5	1078.2	0
cc11-2p	2048	11263	244	**63680**	744.3	63895.7	976.4	103.4	726.6	146
cc11-2u	2048	11263	244	615	1388.7	616.9	1203.8	1.0	951.6	-1
cc12-2p	4096	24574	473	122166	1884.1	123096.0	1912.6	468.0	799.1	-1060
cc12-2u	4096	24574	473	1183	1559.5	1186.3	1937.0	1.8	804.2	-4
cc3-10p	1000	13500	50	**12784**	3471.2	12826.2	1801.6	43.5	1139.7	76
cc3-10u	1000	13500	50	125	61.9	125.0	615.8	0.0	683.5	0
cc3-11p	1331	19965	61	**15599**	458.9	15633.3	812.1	35.4	965.1	10
cc3-11u	1331	19965	61	153	29.7	153.0	269.3	0.0	580.9	0
cc3-12p	1728	28512	74	18879	1290.1	18936.6	1771.1	31.5	1139.8	-41
cc3-12u	1728	28512	74	**185**	59.7	185.0	900.5	0.0	985.4	1
cc3-5p	125	750	13	3661	0.8	3661.0	10.5	0.0	13.2	0
cc3-5u	125	750	13	36	0.0	36.0	0.0	0.0	0.0	0
cc5-3p	243	1215	27	7299	16.4	7299.0	238.2	0.0	208.6	0
cc6-3p	729	4368	76	**20340**	1266.8	20395.9	1544.0	46.0	984.0	116
cc7-3p	2187	15308	222	**57080**	1385.5	57328.7	1197.7	153.9	888.0	8
cc7-3u	2187	15308	222	**551**	383.8	554.1	1267.2	1.5	1078.5	1
cc9-2p	512	2304	64	**17202**	1603.4	17274.4	1579.8	28.5	984.4	94
cc9-2u	512	2304	64	167	15.0	167.3	753.1	0.5	1018.6	0
hc7p	128	448	64	7905	2480.0	7915.8	875.6	6.0	746.9	0
hc8p	256	1024	128	15337	2494.8	15349.5	1057.5	7.5	1128.9	-15
hc8u	256	1024	128	148	0.0	148.0	0.1	0.0	0.0	0
hc9p	512	2304	256	30319	1232.0	30342.3	1824.9	14.1	777.7	-61
hc9u	512	2304	256	292	0.3	292.0	0.4	0.0	0.1	0
hc10p	1024	5120	512	**59981**	267.5	60041.3	1013.5	33.4	817.0	513
hc10u	1024	5120	512	**575**	11.2	575.0	87.0	0.0	85.9	6
hc11p	2048	11264	1024	**119500**	3327.8	119533.0	1708.9	35.1	1129.1	279
hc11u	2048	11264	1024	**1145**	663.3	1145.4	1319.2	0.5	873.1	9
hc12p	4096	24576	2048	**236267**	2782.9	236347.1	2514.0	55.4	565.3	682
hc12u	4096	24576	2048	**2261**	2756.8	2262.5	2805.2	1.3	747.0	14

Table 12: I640 STP instances. Column 'Impr.' shows the improvement w.r.t. the previous best known values published on the DIMACS challenge website [10]. Results computed by local branching using the (x,y)-model. Improved solutions given in boldface. Time limits set to one hour.

				BEST		AVG		STD		
Instance	$\|V\|$	$\|E\|$	$\|T\|$	UB	Time (s.)	UB	Time (s.)	UB	Time (s.)	Impr.
i640-311	640	4135	160	35766	117.6	35779.0	1521.0	21.7	1219.7	0
i640-312	640	4135	160	**35768**	1410.3	35793.2	1478.5	25.4	1104.3	3
i640-313	640	4135	160	35535	292.6	35538.2	923.7	4.1	921.4	0
i640-314	640	4135	160	**35533**	1610.0	35547.0	1673.7	12.5	679.5	5
i640-315	640	4135	160	**35720**	156.2	35733.5	866.8	21.9	695.9	21

Table 13: PUCN STP instances (uniform version of the PUC code-coverage dataset). Column 'Impr.' shows the improvement with respect to the best solution computed by B&C. Results computed by local branching on the y-model. Time limits set to one hour.

				BEST		AVG		STD		
Instance	$\|V\|$	$\|E\|$	$\|T\|$	UB	Time (s.)	UB	Time (s.)	UB	Time (s.)	Impr.
cc10-2n	1024	5120	135	180	89.5	181.0	690.1	0.7	952.4	2
cc11-2n	2048	11263	244	327	39.8	328.0	658.6	0.7	904.0	4
cc12-2n	4096	24574	473	617	930.4	621.8	933.1	2.6	639.2	9
cc3-10n	1000	13500	50	75	0.3	75.0	1.2	0.0	1.0	0
cc3-11n	1331	19965	61	92	0.5	92.0	1.2	0.0	0.8	0
cc3-12n	1728	28512	74	111	2.1	111.0	6.6	0.0	3.8	0
cc7-3n	2187	15308	222	290	51.4	290.7	449.6	0.8	410.3	3
cc9-2n	512	2304	64	98	73.5	98.9	598.7	0.6	575.5	2

Table 14: PUCNU PCSTP instances (uniform PCSTP version of the PUC dataset). Column 'Impr.' shows the improvement with respect to the best solution computed by B&C. Results computed by local branching on the y-model. Time limits set to one hour.

				BEST		AVG		STD		
Instance	$\|V\|$	$\|E\|$	$\|T\|$	UB	Time (s.)	UB	Time (s.)	UB	Time (s.)	Impr.
bipa2nu	3300	18073	300	324	18.0	324.0	534.2	0.0	580.8	1
cc10-2nu	1024	5120	135	168	253.0	169.2	1059.5	0.6	1140.6	3
cc11-2nu	2048	11263	244	305	158.5	306.5	612.7	1.3	563.0	7
cc12-2nu	4096	24574	473	568	1427.2	571.0	921.5	1.5	893.8	13
cc3-10nu	1000	13500	50	61	0.5	61.0	4.2	0.0	5.6	0
cc3-11nu	1331	19965	61	79	11.6	79.3	365.4	0.5	571.8	1
cc3-12nu	1728	28512	74	95	12.2	95.0	412.3	0.0	538.5	1
cc7-3nu	2187	15308	222	271	565.0	274.1	639.5	1.3	751.4	9

Table 15: H and H2 PCSTP instances. Column ‘Impr.’ shows the improvement with respect to the best solution computed by B&C. Results computed by local branching on the y-model for uniform instances (hc*u(2)) and on the (x, y)-model for all non-uniform instances (hc*p(2)). For large non-uniform hypercubes (starting from hc10p(2)), the blurred version of the set-covering heuristic is applied instead. Time limits set to one hour.

				BEST		AVG		STD		
Instance	$\|V\|$	$\|E\|$	$\|T\|$	UB	Time (s.)	UB	Time (s.)	UB	Time (s.)	Impr.
hc8p	256	1024	256	15206	537.3	15228.5	1514.8	15.7	1234.1	58
hc9p	512	2304	512	30043	3062.5	30084.0	1967.6	25.2	1073.5	209
hc10p	1024	5120	1024	59866	919.1	59965.2	1078.3	46.4	922.7	728
hc10u	1024	5120	1024	559	2349.4	559.9	773.6	0.3	1081.3	3
hc11p	2048	11264	2048	119191	3600.0	119377.6	1851.1	96.4	1114.0	1806
hc11u	2048	11264	2048	1116	2284.5	1117.2	1568.6	0.8	1193.9	4
hc12p	4096	24576	4096	235860	2542.6	236103.4	2417.0	147.2	685.5	4621
hc12u	4096	24576	4096	2221	310.5	2223.1	1078.5	1.4	1035.8	5
hc8p2	256	1024	256	15231	178.3	15255.0	1271.7	17.8	996.9	60
hc9u2	512	2304	512	190	10.0	190.0	15.4	0.0	3.0	0
hc10p2	1024	5120	1024	59930	2119.8	59966.4	1849.7	21.5	983.5	601
hc10u2	1024	5120	1024	380	47.7	380.3	1287.8	0.5	1119.2	2
hc11p2	2048	11264	2048	119236	330.5	119381.8	1106.5	89.7	704.0	1762
hc11u2	2048	11264	2048	750	2009.5	751.4	1596.3	0.7	1131.7	11
hc12p2	4096	24576	4096	235687	3172.9	235985.0	2099.7	217.3	1076.0	4793
hc12u2	4096	24576	4096	1494	101.9	1494.1	671.8	0.3	1031.3	13

Table 16: I640 PCSTP instances. Column ‘Impr.’ shows the improvement with respect to the best solution computed by B&C. Results computed by local branching on the (x, y)-model. Time limits set to one hour.

				BEST		AVG		STD		
Instance	$\|V\|$	$\|E\|$	$\|T\|$	UB	Time (s.)	UB	Time (s.)	UB	Time (s.)	Impr.
i640-311	640	4135	160	33503	1062.2	33518.8	2045.0	20.4	1093.3	44
i640-312	640	4135	160	32721	152.9	32721.0	1846.1	0.0	1147.9	69
i640-313	640	4135	160	32401	200.1	32403.0	1302.3	6.3	906.8	0
i640-314	640	4135	160	32871	1572.9	32893.1	1590.9	20.4	1147.9	88
i640-315	640	4135	160	32616	491.3	32631.3	1621.2	9.3	1059.4	50
i640-321	640	204480	160	28787	309.3	28788.3	1121.0	1.6	1234.3	16
i640-322	640	204480	160	28458	943.7	28461.2	989.0	3.9	828.9	9
i640-323	640	204480	160	28153	9.2	28153.5	667.8	0.5	687.5	4
i640-324	640	204480	160	28746	44.5	28747.5	864.6	1.0	1116.1	14
i640-325	640	204480	160	28385	31.3	28386.0	1588.6	0.9	1589.0	1
i640-341	640	40896	160	29702	1585.8	29720.5	1679.1	14.2	984.2	37
i640-342	640	40896	160	29806	1591.9	29828.8	1889.1	16.8	915.5	32
i640-343	640	40896	160	30056	307.3	30059.9	1093.9	8.2	691.1	0
i640-344	640	40896	160	29921	365.2	29943.8	1370.6	12.8	913.1	21
i640-345	640	40896	160	30004	2580.5	30029.6	2289.4	18.7	1141.9	78

Acknowledgements

The research of the Padova's group was supported by the University of Padova (Progetto di Ateneo "Exploiting randomness in Mixed Integer Linear Programming"), and by MiUR, Italy (PRIN project "Mixed-Integer Nonlinear Optimization: Approaches and Applications"). The research of the Vienna's group was supported by the Austrian Research Fund (FWF, Projects P 26755-N19 and I 892-N23). The work of the last author was supported by a STSM Grant from COST Action TD1207; the authors would like to acknowledge networking support by the same COST action.

References

[1] 11th DIMACS Implementation Challenge in Collaboration with ICERM: Steiner tree problems (2014). URL `http://dimacs11.cs.princeton.edu/home.html`

[2] Álvarez-Miranda, E., Ljubić, I., Mutzel, P.: The maximum weight connected subgraph problem. In: Facets of Combinatorial Optimization: Festschrift for Martin Grötschel, pp. 245–270. Springer (2013)

[3] Andreello, G., Caprara, A., Fischetti, M.: Embedding {0, 1/2}-cuts in a branch-and-cut framework: A computational study. INFORMS Journal on Computing **19**(2), 229–238 (2007)

[4] de Aragão, M.P., Werneck, R.F.F.: On the implementation of MST-based heuristics for the Steiner problem in graphs. In: D.M. Mount, C. Stein (eds.) ALENEX, *Lecture Notes in Computer Science*, vol. 2409, pp. 1–15. Springer (2002)

[5] Backes, C., Rurainski, A., Klau, G.W., Müller, O., Stöckel, D., Gerasch, A., Küntzer, J., Maisel, D., Ludwig, N., Hein, M., Keller, A., Burtscher, H., Kaufmann, M., Meese, E., Lenhof, H.P.: An integer linear programming approach for finding deregulated subgraphs in regulatory networks. Nucleic Acids Research **40**, 1–13 (2012)

[6] Caprara, A., Fischetti, M.: {0, 1/2}-Chvátal-Gomory cuts. Mathematical Programming **74**(3), 221–235 (1996)

[7] Caprara, A., Fischetti, M., Toth, P.: A heuristic algorithm for the set covering problem. In: Integer Programming and Combinatorial Optimization, pp. 72–84. Springer Berlin Heidelberg (1996)

[8] Caprara, A., Fischetti, M., Toth, P.: A heuristic method for the set covering problem. Operations Research **47**(5), 730–743 (1999)

[9] Cherkassky, B.V., Goldberg, A.V.: On implementing push-relabel method for the maximum flow problem **19**, 390–410 (1994.)

[10] Best bounds as of August 1, 2014 for SteinLib instances (2014). `http://dimacs11.cs.princeton.edu/instances/bounds20140801.txt`

[11] Results of the 11th DIMACS Competition on Steiner Tree Problems (2015). `http://dimacs11.cs.princeton.edu/contest/results/results.html`

[12] Duin, C.: Preprocessing the Steiner Problem in Graphs. In: D.Z. Du, J. Smith, J. Rubinstein (eds.) Advances in Steiner Trees, *Combinatorial Optimization*, vol. 6, pp. 175–233. Springer US (2000)

[13] Eisenstat, D.: dtree: dynamic trees à la carte (2014). `http://www.davideisenstat.com/dtree/`

[14] Fischetti, M., Lodi, A.: Local branching. Mathematical Programming **98**(1-3), 23–47 (2003)

[15] Fischetti, M., Monaci, M.: Cutting plane versus compact formulations for uncertain (integer) linear programs. Mathematical Programming Computation **4**(3), 239–273 (2012)

[16] Fischetti, M., Monaci, M.: Proximity search for 0-1 mixed-integer convex programming. Journal of Heuristics **20**(6), 709–731 (2014)

[17] Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs (Annals of Discrete Mathematics, Vol 57). North-Holland Publishing Co., Amsterdam, The Netherlands (2004)

[18] Koch, T., Martin, A.: Solving Steiner tree problems in graphs to optimality. Networks **32**(3), 207–232 (1998). DOI 10.1002/(SICI)1097-0037(199810)32:3⟨207::AID-NET5⟩3.0.CO;2-O. URL `http://dblp.uni-trier.de/db/journals/networks/networks32.html#KochM98`

[19] Koch, T., Martin, A., Voß, S.: SteinLib: An updated library on Steiner tree problems in graphs. Tech. Rep. ZIB-Report 00-37, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Takustr. 7, Berlin (2000). URL `http://elib.zib.de/steinlib`

[20] Leitner, M., Ljubić, I., Luipersbeck, M., Resch, M.: A Partition-Based Heuristic for the Steiner Tree Problem in Large Graphs. In: M.J. Blesa, C. Blum, S. Voß (eds.) Hybrid Metaheuristics - Proceedings, *Lecture Notes in Computer Science*, vol. 8457, pp. 56–70. Springer (2014)

[21] Ljubić, I., Weiskircher, R., Pferschy, U., Klau, G.W., Mutzel, P., Fischetti, M.: An algorithmic framework for the exact solution of the prize-collecting Steiner tree problem. Mathematical Programming **105**(2-3), 427–449 (2006)

[22] Lucena, A., Resende, M.G.: Strong lower bounds for the prize collecting Steiner problem in graphs. Discrete Applied Mathematics **141**(1), 277–294 (2004)

[23] OGDF: The Open Graph Drawing Framework (2015). `http://www.ogdf.net/`

[24] Uchoa, E., Werneck, R.F.: Fast local search for Steiner trees in graphs. In: G.E. Blelloch, D. Halperin (eds.) ALENEX, pp. 1–10. SIAM (2010)

[25] Wang, Y., Buchanan, A., Butenko, S.: On imposing connectivity constraints in integer programs (2015). Submitted. Available at `www.optimization-online.org/DB_HTML/2015/02/4768.html`

[26] Wong, R.T.: A dual ascent approach for Steiner tree problems on a directed graph. Mathematical Programming **28**(3), 271–287 (1984). DOI 10.1007/BF02612335