European Journal of Operational Research 245 (2015) 680–689

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

Exact and heuristic solutions for the Minimum Number of Branch Vertices Spanning Tree Problem

Alfredo Marín*

Departamento de Estadística e Investigación Operativa, Universidad de Murcia, Spain

ARTICLE INFO

Article history:
Received 25 September 2014
Accepted 6 April 2015
Available online 14 April 2015

Keywords:
Integer programming
Spanning tree
Branch vertices

ABSTRACT

The Minimum Number of Branch Vertices Spanning Tree Problem is to minimize the number of vertices of degree greater than two in a spanning tree. We present a branch-and-cut algorithm based on an enforced Integer Programming formulation, which can solve many more instances than previous methods. Since the problem is NP-hard, very large instances cannot be solved exactly. For such cases, a new heuristic two-stage method that gives very good approximate solutions is developed.

1. Introduction

In an undirected connected graph, a spanning tree (ST) is defined by a connected subgraph, maintaining all the nodes of the graph and with a minimum number of edges. When considering weighted edges, the Minimum Spanning Tree Problem (MSTP) aims to find the spanning tree with the minimum sum of weights. The number of spanning trees in a given graph can be huge, but finding an ST of minimum weight is a simple task, which can be carried out, for instance, using Prim's algorithm (Prim, 1957). The efficiency of MST algorithms makes it unnecessary to design Integer Programming models to solve this problem. Nevertheless, there are many other STs of interest in a graph: to cite only a few, the degree-constrained ST (Narula & Ho, 1980) the ST of Hubs (Contreras, Fernández, & Marín, 2009) and edge-equitable STs (Landete & Marín, 2014). The difficulty of finding these and other optimal trees by means of an efficient ad hoc algorithm has led to the development of different IP formulations. The key point in these formulations is either (i) guaranteeing connectivity, or (ii) avoiding the formation of cycles, since both ensure the tree structure of the resulting subgraph when the number of edges is fixed. A relatively successful approach that we will follow partially in this work is to consider the ST problem as a network design problem where some flow between the nodes of the network is sent. Here, two variables associated to each edge (one per arc) indicate whether or not the arc is available to carry any flow. Thus, an order is imposed on the nodes of the graph and cycles cannot be closed.

At another level, we find ST optimization problems with no weights associated to the edges. In this case the objective is related to the structure of the tree. Some problems within this family are the Minimum Leaf STP (Cerrone, Cerulli, & Raiconi, 2014; Salamon & Wiener, 2008), the Full Degree STP (also called Degree Preserving STP) (Bhatia, Khuller, Pless, & Sussmann, 1999; Gendron, Lucena, Salles da Cunha, & Simonetti, 2013; Khuller, Bhatia, & Pless, 2003), which keeps the maximum number of nodes with the same degree as in the original graph, and the Minimum-Degree STP (Fürer & Raghavachari, 1990), where the maximum degree of the nodes in the tree is to be minimized. Still in this family, there is a sub-family of interesting problems which optimize a measure that depends only on the so-called *branch vertices* (alternatively branching nodes, branches), those nodes in the tree with a degree greater than two. One of these problems is the Minimum Degree Sum of Branch Vertices STP, whose objective function is the sum of the degrees of all the vertices except those of degrees 1 and 2 (Cerrone et al., 2014; Cerulli, Gentili, & Iossa, 2009; Sundar, Singh, & Rossi, 2012). Finally, the Minimum Number of Branch Vertices STP, MBV, looks for the spanning tree with minimum number of vertices of degree greater than two, which is what we are interested in. The optimization problems corresponding to all the mentioned graphs are different and NP-hard (see the cited papers and references therein).

Regarding the problem in question, the MBV, it seems to be have been proposed first in Gargano, Hell, Stacho, and Vaccaro (2002), justified by the design of optical networks, where a light signal can traverse a degree-two node without additional help, but must be split by a *switch* in the branching nodes. These switches must then be located in all the branching nodes and the cost of the network depends on their number. Cerulli et al. (2009) propose an Integer Programming formulation for the MBV and designed different heuristic strategies. Their formulation guarantees connection by sending a unit of flow from a root to any other node of the graph. They also generate many different instances, comparing the heuristic solutions with the optima reported by CPLEX solver running the IP formulation. Optimal

* Tel.: +34 868883627.
E-mail address: amarin@um.es

http://dx.doi.org/10.1016/j.ejor.2015.04.011
0377-2217/

solutions could only be obtained for some small instances. Silva et al. (2011) propose an adaptation of the Iterative Refinement approach to solve MBV, choosing to penalize *violating edges* and replace them with edges with less violation. They report better computational results than those obtained by Cerulli et al. (2009) on a set of small instances. Sundar et al. (2012) design two heuristic algorithms, one that starts with an ST and interchanges one tree edge and one edge outside the tree, keeping the acyclic structure and trying to improve the solution, together with an ant colony heuristic. They use the same IP formulation as in Cerulli et al. (2009), and generate similar random instances to make a comparison with the results in Cerulli et al. (2009). Xpress solver is used here, and the authors concluded that their heuristics work better than previous ones. Merabet, Durand, and Molnar (2012) consider again the same IP formulation as in Cerulli et al. (2009) to approach a variant of the problem. Carrabs, Cerulli, Gaudioso, and Gentili (2013) consider four IP formulations. The first contains an exponential number of constraints. The second is very similar to the formulation previously used in the literature, sending a unit of flow from a root node to any other node, and they also consider a multi-commodity flow formulation. The last uses additional variables and constraints in the style of Miller, Tucker, and Zemlin (1960). Although Carrabs et al. (2013) considered all the formulations to generate bounds (by means of different relaxations), the one that was used to optimally solve some instances was again the single commodity flow formulation. Their instances, available on the web page of the first author, were obtained when trying to produce a significant number of branch vertices in the optimal solution (implying the graphs were very sparse). CPLEX 11 was applied to the formulations. From 250 nodes on, there were instances that could not be optimally solved with any formulation. Silva et al. (2014) checked a different edge-swap heuristic on new sets of instances, and compared it with some instances of the previous authors, concluding that the new algorithm performed better, in general, that the previous approaches. Although they revisited one of the previous IP formulations, it was not used in their study. The last paper on MBV problem is Cerrone et al. (2014). The three smallest IP formulations were again reconsidered and relaxed, and another heuristic method (a memetic algorithm) proposed. Old and new instances were used in their computational study, obtaining relatively good approximate solutions.

There is little in the literature to optimally solve the MBV problem. The formulations used are straightforward adaptations of classical formulations designed for similar spanning tree problems. Here we intend to modify the single commodity flow formulation to (i) adapt it better to the problem in question and (ii) enforce it to produce better bounds. We will also consider several properties that a spanning tree must satisfy, that will allow us to preprocess the instances, some of them very effectively. Moreover, since good heuristic solutions can be used to reduce the time needed to solve the instances, and they are the only way to produce a good solution when an instance is too difficult, we will also devote a lot of attention to the development of a two-stage heuristic method. As will be seen in the computational results section, the conjunction of these strategies in a branch-and-cut algorithm will lead to a spectacular improvement of previous results.

Section 2 introduces the problem and some notations. Section 3 contains the details of a preprocessing phase, and a two-stage heuristic method is presented in Section 4. In Section 5 we present and improve the formulation previously used in the literature, and Section 6 is devoted to obtaining valid inequalities to tighten it. All these tools are combined in a branch-and-cut algorithm in Section 7. Computational results are obtained and compared with previous results in Section 8. An extension of the problem, incorporating weights on the edges and fixed costs associated with the branch vertices is also introduced in Section 9 and computationally studied in Section 10. Some conclusions close the paper.

2. The problem

Let $G'=(V',E')$ be a connected, undirected graph with set of vertices (nodes) $V'=\{1,\ldots,n'\}$ and set of edges $E'\subseteq\{e=\{i,j\}: i\neq j\in V'\}$ with $|E'|=m'$. Let $\mathcal{T}'$ be the set of spanning trees in G'. Let $\delta_j(G')$ be the degree in G' of node j, i.e., $\delta_j(G'):=|\{e\in E': j\in e\}|$. Similarly, given any tree $T=(V',E_T)\in\mathcal{T}'$, let $\delta_j(T)$ be the degree in T of node j, i.e., $\delta_j(T):=|\{e\in E_T: j\in e\}|$.

The Minimum Number of Branch Vertices Problem, MBV, is defined as

$$\min_{T\in\mathcal{T}'}|\{j\in V':\ \delta_j(T)\geq 3\}|.$$

A node $j\in V'$ such that $\delta_j(T)\geq 3$ will be called a *branching vertex*, a *branch node* or simply a *branch* of T. We also define the neighborhood (set of neighbors) of $j\in V'$ in G' as

$$N_{G'}(j):=\{i\in V':\ \{i,j\}\in E'\}.$$

An edge $e\in E'$ such that $(V',E'\backslash\{e\})$ becomes disconnected will be called a *bridge* of G'. A set of two different edges $\{e,e'\}$ such that $(V',E'\backslash\{e,e'\})$ becomes disconnected but both $(V',E'\backslash\{e\})$ and $(V',E'\backslash\{e'\})$ are still connected will be called a *2-cocycle* or a cocycle of size two in G'.

We will also split each edge $\{i,j\}\in E'$ into two arcs (i,j) and (j,i). We call A' this set of arcs. If $\{i,j\}$ is a bridge, we say that (i,j) is an *arc bridge* (or simply a bridge) in G' if all oriented paths from i to n' traverse node j (i.e., if (i,j) is oriented *towards* n'). When $\{e,e'\}$ is a 2-cocycle, we call the set of two arcs obtained orienting e and e' towards n' *2-cocircuit*.

3. Preprocessing

All bridges of a connected graph must be in the set of edges of any spanning tree. When removing all bridges some nodes can become isolated. In the left hand side of Fig. 1 we show a graph with six bridges. The removal of five of them (dotted lines) leaves five nodes isolated, as shown in the right hand side of the figure. After identifying the set of isolated nodes $V'_I\subseteq V'$, we will remove them and the corresponding edges from G', producing the subgraph induced by $V:=V'\setminus V'_I$, called $G=(V,E)$. For the sake of simplicity, we will assume that $V=\{1,\ldots,n\}$ and then $V_I:=\{n+1,\ldots,n'\}$. Since some of the removed nodes may be branches (like node 2 in the figure), we must add the corresponding constant to the objective function. There will be also some nodes in V which have two or more neighbors among the removed nodes (like node 16 in the figure). These other nodes will also be branches. And nodes $j\in V'$ with $\delta_j(G')\leq 2$ will never be branches. To take note of the interesting nodes, we create a new set V'', obtained from V by removing all the nodes which definitely are either branches or not. Bridges whose removal do not produce isolated nodes (like the thick edge of the left hand side of the figure) will be included a priori in the optimum spanning tree. We call the number of deleted nodes among the neighbors of j $\delta^p_j:=|N_{G'}(j)\cap V_I|$.

Identifying bridges in G' can be done efficiently with the algorithm proposed in Schmidt (2013). The method has two stages. In the first, using depth-first search, a spanning arborescence (oriented spanning tree) with all arcs pointing towards root node n' has to be built. The details are given in Fig. 2. In step 1, all nodes are initially unmarked ($m(j):=0$), the level of all nodes except node n' is fixed to -1 ($\ell(j):=-1$) and the level of node n' is fixed to 0. A is going to be the set of arcs of the arborescence, which is initially empty. Step 2 is repeated until all the nodes are marked ($m(j)=1$). An unmarked node of maximum level i is first identified. There are two possibilities: (i) all neighbors of i were previously marked; this means that they have been already included in the arborescence, and then node i is in turn marked, and (ii) there exists at least one unmarked node k in the neighborhood

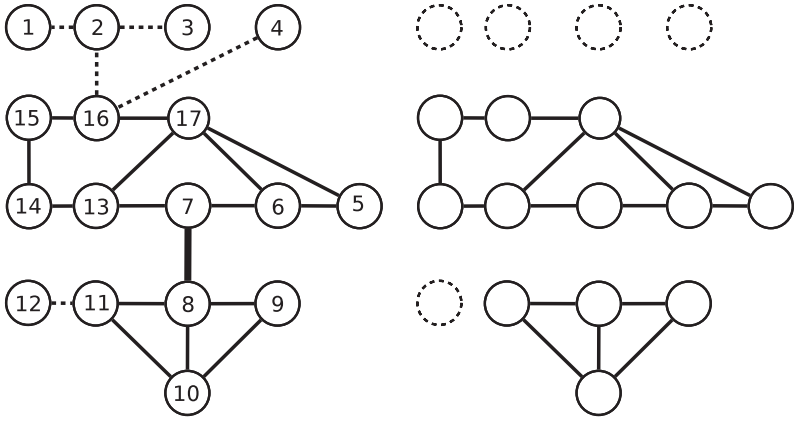

Fig. 1. Size reduction.

Step 1. For all $j \in V'$, set $m(j) := 0$.
For all $j \in V' \setminus \{n'\}$ set $\ell(j) := -1$. Set $\ell(n') := 0$.
Set $A := \emptyset$.
Step 2. If $m(j) = 1$ for all $j \in V'$ then STOP.
Find $i := \arg\max\{\ell(j) : m(j) = 0\}$.
If $\ell(j) \geq 0$ for all $j \in N_{G'}(i)$ then set $m(i) := 1$ and go to Step 2.
Find $k \in N_{G'}(i)$ such that $\ell(k) = -1$.
Set $\ell(k) := \ell(i) + 1$ and $A := A \cup \{(k, i)\}$.
Go to Step 2.

Fig. 2. Algorithm for building the arborescence.

of i; since i has the maximum possible level among the nodes of the arborescence with unmarked neighbors, arc (k, i) is included in A, and k is labeled $\ell(i) + 1$. The process is illustrated in Fig.3, using the same graph as in Fig. 1. The root node $n' = 17$ is gray and the numbers inside the nodes are the values of $\ell(j)$. Starting with $i = 17$, its neighbor $k = 5$ is chosen and the arc (5, 17) is included in A (bold arrow). The level of 5 is now 1, and then 5 is chosen as node i. Its neighbor 6 is then chosen, and so forth until arriving at node 1 and the situation of Fig.3, top-left. Since node 1 has no neighbors with level -1, it is marked and node 2 is chosen as i. There is a neighbor of 2 with level -1, namely node 3, and so 3 is chosen as k and arc (3, 2) is included in A, as seen in the top-right part of the figure. Now 3 and 2 are marked, 16 is chosen as i and 4 is chosen as k, with level 8 as indicated in the down-left part. The final arborescence and corresponding levels are shown in the lower-right part of the figure.

Consider now an edge not in the arborescence. The way in which the previous algorithm works guarantees that both extremes of the

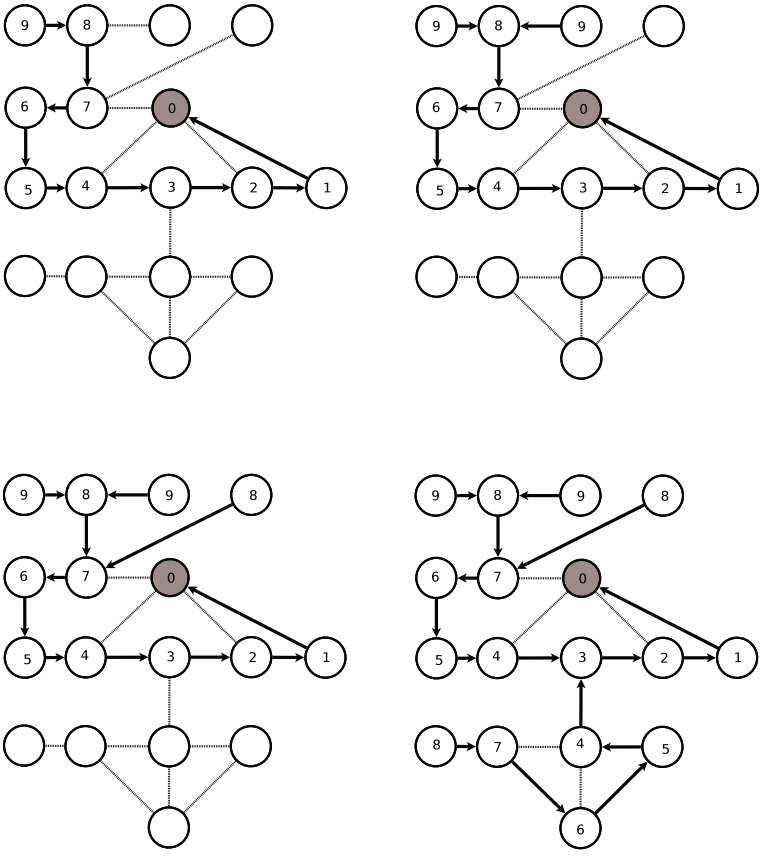

Fig. 3. Building the arborescence.

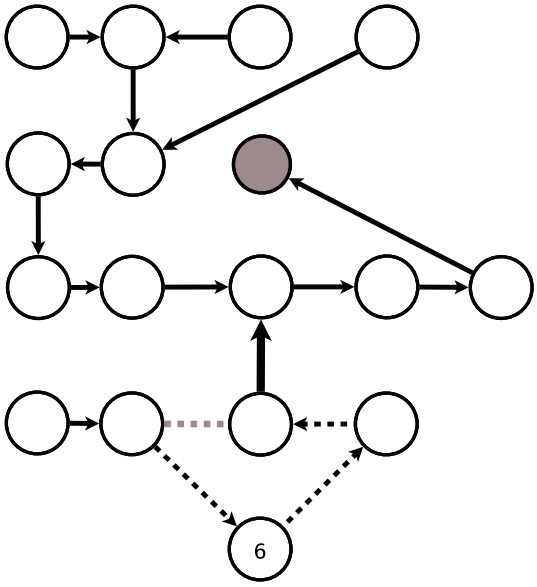

Fig. 4. Looking for bridges.

edge have different levels, and also the existence of an oriented path from the extreme with maximum level to the other extreme. We illustrate this in Fig. 4, choosing edge (8, 11) (dotted edge). The extreme with maximum level is 11, with level 7. The oriented path (inside the arborescence) from 11 to 8 is depicted in the figure with dashed arrows. Since all the edges of this path plus the dotted edge close a cycle, they cannot be bridges. Using this fact, the second stage of the method for identifying bridges, given in Fig. 5, iteratively considers all edges not in the arborescence, goes over the corresponding cycle and discards the edges in the cycle. The remaining edges will be the bridges of the graph. Moreover, a bridge oriented as in the arborescence will be an arc bridge (thick arc in the figure).

Identifying 2-cocycles (and 2-cocircuits) can be done by removing any edge other than a bridge from E (one at a time) and then applying the algorithm to identify the bridges of the resulting subgraph.

4. Heuristic solution

The importance of finding good feasible solutions to MBV is twofold. On the one hand, tight upper bounds will help to solve the problem exactly by reducing the search of any Integer Programming solver. On the other hand, difficult instances which cannot be exactly solved will be approximately solved by means of the heuristic procedure. Several heuristics have been developed in the literature. An edge weighting strategy, a node coloring strategy and a combination of both were the first (Cerulli et al., 2009). In Silva et al. (2011) an Iterative Refinement approach gave better computational results than those of Cerulli et al. (2009) on a set of small instances. In Sundar et al. (2012), an ant colony heuristic was tested, and reported to perform better than the previous ones. The dual ascent based on an IP formulation of Carrabs et al. (2013), the edge-swap heuristic of Silva et al. (2014) and the memetic approach of Cerrone et al. (2014) close the list of approaches that, to the best of our knowledge, are all the methods published to date.

Here we present a heuristic method which starts with a spanning tree built with a dynamically modified version of Prim's algorithm. This tree is then improved with several exchange strategies. Some of the strategies reduce the number of branches of the current tree, but several others only perturb it, without increasing the number of branches. Since the first part of the heuristic uses randomized weights, and given that the overall process is very fast, a multi-start method is used.

The basic structure of the heuristic is given in Fig. 6. The weights of step 1a are only relevant when one of the extremes of $\{i, j\}$, say i, is marked and the other extreme is unmarked. Let τ be the current tree, M a large amount and α a random uniform variable between 0 and 1. Then the weight assigned to $\{i, j\}$ will be

$$\begin{array}{rl} \alpha + \delta_j(G') + \delta_i(\tau) & \text{if } \delta_i(\tau) \geq 3 \text{ or } \delta_i^p \geq 2, \\ \alpha + M + \delta_j(G') & \text{if } \delta_i(\tau) = 1, \\ \alpha + M^2 & \text{otherwise.} \end{array}$$

The effect of these weights is that edges with a marked extreme which is going to be a branch are prioritized, and preferred if their original degree is low, since their incident edges will be more probable in any spanning tree. Edges that are leaves in the current tree are then prioritized and randomly ordered. The rest of the nodes are also randomly ordered.

There are two different improving movements in step 2a, IMP1 and IMP2. IMP1 is first repeated until not possible. How IMP1 works is illustrated in Fig. 7. Edges e from outside the tree are iteratively considered, and selected if none of the extremes have degree two in the current tree (dotted edge in the left hand side). Edges in the tree that close a cycle with e are identified (thick edges). If replacing one of the edges in the cycle by e reduces the degree of any node in the tree from 3 to 2, the replacement is made and the objective value is reduced by one (or perhaps by two). IMP2 is applied only once and then the graph is returned to IMP1. How IMP2 works is illustrated in Fig. 8. Edges e from outside the tree are iteratively considered. Each of these edges closes a cycle when added to the tree. e is selected if (i) one of its extremes has a degree not equal to two in the current tree and (ii) the edge that is incident to the other extreme of e that belongs to the cycle (thick edge) is also incident to a second node j_i (gray node) of degree 3. It can be seen in the figure that on replacing the thick edge by the candidate edge (dotted edge) there is one less branch vertex in the new tree.

When no improvement is possible by applying IMP1 and IMP2, the perturbation phase modifies the tree without improving the objective value but by reducing the degree of some branch in the tree. PER1 is similar to IMP2, but will be applied when the degree of j_i is greater than 3. PER2 is similar to IMP1, and will be applied whenever the degree of some branch in the closed cycle is reduced in case of replacement. Finally, PER3 is similar to PER2, but the exchange is made when the degree in the tree of a branch is reduced from 3 to 2 and another node of size 2 is converted into a branch. To avoid cycling,

Step 1. Build an arborescence using depth-first search.
Step 2. For all $e \in E'$, set $m(e) := 0$.
Step 3. For each edge $e = \{i, j\}$ out of the arborescence such that $\ell(i) > \ell(j)$:
  Set $m(e) := 1$.
  Set $m(e') = 1$ for all edges e' in the path of the arborescence from i to j.
OUTPUT: $\{e \in E' : \ m(e) = 0\}$ is the set of bridges of the graph.

Fig. 5. Algorithm for identifying bridges.

Repeat several times:
Step 1. *Dynamically modified Prim's algorithm.*
Start with a random edge, adding it to the tree. Mark both extremes.
Repeat $n-2$ times steps 1a and 1b.
Step 1a. Give new weights to the edges of E.
Step 1b. Choose the edge with minimum weight with exactly one extreme marked. Add it to the tree and mark its unmarked extreme.
Step 2. Repeat steps 2a and 2b until not possible:
Step 2a. *Improvement phase.* Repeat IMP1 and IMP2 until no improvement is possible.
Step 2b. *Perturbation phase.* Apply PER2. If the tree remains unchanged, apply PER1. If the tree remains unchanged, apply PER3.

Fig. 6. Outline of the heuristic algorithm.

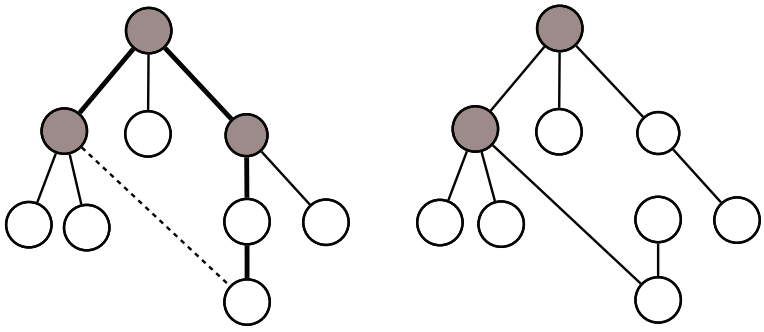

Fig. 7. Illustration of first improvement in the heuristic algorithm.

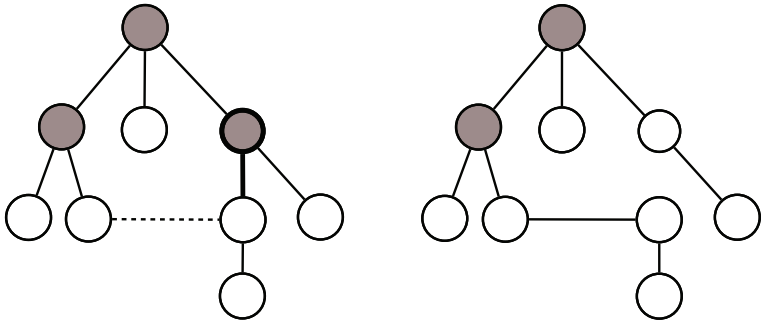

Fig. 8. Illustration of second improvement in the heuristic algorithm.

each candidate edge is considered only once if no new improvement is obtained in step 2a.

5. Improved single commodity flow formulation

The formulation we are going to use starts with the preprocessed graph $G=(V,E)$ with $|V|=n$ and $|E|=m$, where all the vertices that become isolated after removing the bridges from the graph have been eliminated, and there is a constant that has to be added to the optimal value of the problem. We also make use of the corresponding set of arcs A and the set V'' that contains the nodes that could be branches or not. On the other hand, the (basic) formulation we take (developed in Carrabs et al., 2013) and we show first does not use V'' and, for the sake of simplicity, will be explained in graph G (note that, in the literature, it is used directly in graph G' and set of arcs A').

Standard binary decision variables

$$x_{ij}=\begin{cases}1 & \text{if } (i,j) \text{ belongs to } E_T\\ 0 & \text{otherwise}\end{cases}\qquad \forall (i,j)\in A$$

will be used to characterize a spanning tree in G. Non-negative variables z_{ij} associated to the arcs will be used to represent the amount of flow traversing them. Finally, and necessary to calculate the objective value of a solution, binary variables y_j defined as follows are incorporated:

$$y_j=\begin{cases}1 & \text{if } j \text{ has degree at least 3 in } T\\ 0 & \text{otherwise}\end{cases}\qquad \forall j\in V.$$

The formulation commonly used in the literature is

$$\text{(L)}\ \min \sum_{j\in V} y_j$$

$$\text{s.t.} \sum_{i\in V:\ (i,n)\in A} z_{in} - \sum_{i\in V:\ (n,i)\in A} z_{ni} = n-1, \tag{1}$$

$$\sum_{i\in V:\ (j,i)\in A} z_{ji} = \sum_{i\in V:\ (i,j)\in A} z_{ij} + 1 \quad \forall j<n, \tag{2}$$

$$z_{ij} \le (n-1)x_{ij} \quad \forall (i,j)\in A, \tag{3}$$

$$z_{ij} \ge x_{ij} \quad \forall (i,j)\in A, \tag{4}$$

$$\sum_{i\in V:\ (j,i)\in A} x_{ji} = 1 \quad \forall j<n, \tag{5}$$

$$\sum_{i,j\in V:\ (i,j)\in A} x_{ij} = n-1, \tag{6}$$

$$\sum_{i\in V:\ (i,j)\in A} x_{ij} + \sum_{i\in V:\ (j,i)\in A} x_{ji} \le 2 + \delta_j(G)y_j \quad \forall j \in V, \tag{7}$$

$$x_{ij} \in \{0,1\} \quad \forall i,j \in V:\ (i,j) \in A,$$

$$y_j \in \{0,1\} \quad \forall j \in V,$$

$$z_{ij} \ge 0 \quad \forall i,j \in V:\ (i,j) \in A.$$

Note that there is a flow of $n-1$ units towards n (any other node could also be chosen as destination of the flow), according to constraint (1), and one unit of this flow is created in all other nodes, according to constraints (2). Constraints (3) and (4) make x_{ij} take value 1 if and only if there is a positive flow traversing arc (i,j). Along with constraints (5) and (6), which fix the number of arcs with first node j to 1 and to $n-1$ the total number of arcs, the result is that x-variables taking value 1 configure an arborescence (oriented towards n). Due to constraints (7), if the degree of j in the arborescence – left hand side – is at least 3, y_j must take value 1. Otherwise, coefficient $\delta_j(G)$ guarantees that y_j is free and will take value 0 in the optimal solution.

Starting from formulation (L), we have carried out several improvements. First, some nodes must be branches since they are incident to several bridges and their degree will consequently be at least three. The corresponding y-variables are fixed to one. Some other nodes, those $j \in V'$ with $\delta_j(G') \le 2$, will never be branches. Their y-variables are fixed to zero. All this information results in creating y-variables only for $j \in V''$. Note that since we work on G, n' is replaced by n as the root of the arborescence.

We then fix the z- and x-variables that correspond with arcs (n,j) to zero, since no flow will leave node n. We remove constraint (6), since it is obtained as the sum of constraints (5) for all $j < n$. Constraints (4) were also eliminated, since they are not needed, are many, and we did not observe any advantage in using them.

Regarding constraints (3), note that the important effect they produce is to let x_{ij} take value 1 if a flow $z_{ij} \ge 1$ traverses arc (i,j). To improve (3), the upper bound $n-1$ on the total amount of flow that can traverse i should be reduced. To this end, we calculate d_j, the length of the path with fewest edges from node j to node n in G. In the worst case, the flow from j to n will follow this path, incorporating d_j units after leaving j. Therefore, for a given arc (i,j), the associated constraint in (3) can be replaced by

$$z_{ij} \le (n-1-d_j)x_{ij}.$$

Another improvement of (3) we have incorporated is to consider the situation in which $j \in N_G(n)$ and $k \in N_G(j)$, $k \ne n$. Then, constraint

$$z_{kj} \le (n-3)x_{kj} + x_{jn} \tag{8}$$

means that the flow going from k to j through a single arc, which is known to be less than or equal to $n-2$ since $j \ne n$, will be also less than or equal to $n-3$ if arc (j,n) is not in the tree.

Consider the original version of constraints (7) on graph G'. We take into account that, for any $j < n'$,

$$\sum_{i\in V':\ (i,j)\in A'} x_{ij} + \sum_{i\in V':\ (j,i)\in A'} x_{ji} = \sum_{i\in V':\ (i,j)\in A'} x_{ij} + 1 \le \delta_j(G').$$

Then the coefficient of y_j can be reduced to let the right hand side reach this maximum, and (7) is replaced by

$$\sum_{i\in V':\ (i,j)\in A'} x_{ij} \le 1 + (\delta_j(G') - 2)y_j \quad \forall j < n'.$$

In the case $j = n'$ the constraint remains

$$\sum_{i\in V':\ (i,n')\in A'} x_{ij} \le 2 + (\delta_{n'}(G') - 2)y_{n'}.$$

However, since the preprocessing phase will remove some edges from graph G' to produce the reduced graph G and will fix some edges in the tree, we must take this into account to obtain the final version of these constraints:

$$\sum_{i\in V:\ (i,j)\in A} x_{ij} + \delta_j^p(T) \le 1 + (\delta_j(G) + \delta_j^p(T) - 2)y_j \quad \forall j < n,$$

$$\sum_{i\in V:\ (i,j)\in A} x_{in} + \delta_n^p(T) \le 2 + (\delta_n(G) + \delta_n^p(T) - 2)y_n,$$

where $\delta_j^p(T)$ is the number of arcs added to T in the preprocessing phase that is incident to j.

Some bridges can have been identified in the preprocessing phases that are still in G. When $\{i,j\}$ is a bridge in G, the procedure we have followed to identify it will give us the orientation of the arc bridge. The arc must be in the arborescence, and then

$$x_{ij} = 1 \quad \forall (i,j) \text{ arc bridge in } G$$

will also be added to the formulation.

Step 1. Apply preprocessing phase, removing isolated nodes.
Step 2. Identify arc bridges and 2-cocircuits.
Step 3. Apply heuristic procedure. Get upper bound ub.
Step 4. Find the paths with least edges from n to all nodes.
Step 5. Build formulation (MBV).
Step 6. Run branch-and-cut procedure (using ub as initial upper bound).

Fig. 9. Outline of the exact algorithm.

6. Valid inequalities

In this section we develop several valid inequalities for the problem. The first set of inequalities has already been used (see, for example, Akgün & Tansel, 2010). It is given by

$$x_{ij} + x_{ji} \le 1 \quad \forall \{i,j\} \in E. \tag{9}$$

The preprocessing phase provides us with a set of 2-cocircuits

$$C = \{\{(i_1,j_1),(i'_1,j'_1)\},\ldots,\{(i_g,j_g),(i'_g,j'_g)\}\}.$$

At least one arc from each cocircuit must be in the arborescence. Valid inequalities

$$x_{i_kj_k} + x_{i'_kj'_k} \ge 1 \quad \forall \{(i_k,j_k),(i'_k,j'_k)\} \in C, \tag{10}$$

are then generated. Some of them, those associated to nodes of degree 2 in G', will already be in the formulation, while several others will not.

The previous inequalities can be used in any spanning tree optimization problem, since they only make use of the x-variables. Moreover, the amount of inequalities in families (8), (9) and (10) is relatively small, and all of them will be incorporated to the formulation. We have also developed a family of valid inequalities which do depend on the y-variables (i.e., they are specifically oriented to the MBV), whose size is larger:

$$x_{aj} + x_{bj} \le 1 + y_j \quad \forall j < n,\ \forall (a,j),(b,j) \in A:\ a < b. \tag{11}$$

Here, since there will be a third arc pointing from j (note that $j \ne n$), if two arcs pointing node j are in the arborescence, then j will be a branch vertex and y_j must take value one. This is a disaggregated version of constraints (7).

7. Solution algorithm

The outline of the exact method we use to solve MBV is given in Fig. 9. First the graph is reduced and an upper bound obtained as seen in previous sections, carrying out the search 100 times per instance. Bridges and cocircuits found in step 2 are used to build constraints (21) and (19), respectively. In order to build constraints (14), lengths

d_j are obtained in step 4. Then the formulation to be used, putting together all improvements made on (L), is given by

$$\text{(MBV)}\ \min \sum_{j\in V''} y_j$$

$$\text{s.t.} \sum_{j\in V:\ (j,n)\in A} z_{jn} = n-1, \tag{12}$$

$$\sum_{i\in V:\ (j,i)\in A} z_{ji} = \sum_{i\in V:\ (i,j)\in A} z_{ij} + 1 \quad \forall j < n, \tag{13}$$

$$z_{ij} \le (n-1-d_j)x_{ij} \quad \forall (i,j)\in A, \tag{14}$$

$$\sum_{i\in V:\ (j,i)\in A} x_{ji} = 1 \quad \forall j \in V, j < n, \tag{15}$$

$$\sum_{i\in V:\ (i,j)\in A} x_{ij} + \delta_j^p(T) \le 1 + (\delta_j(G) + \delta_j^p(T) - 2)y_j \quad \forall j \in V'', j < n, \tag{16}$$

$$\sum_{i\in V:\ (i,n)\in A} x_{in} + \delta_n^p(T) \le 2 + (\delta_n(G) + \delta_n^p(T) - 2)y_n \quad \text{if } n \in V'', \tag{17}$$

$$x_{ij} + x_{ji} \le 1 \quad \forall \{i,j\} \in E, \tag{18}$$

$$x_{i_k j_k} + x_{i'_k j'_k} \ge 1 \quad \forall \{(i_k,j_k),(i'_k,j'_k)\} \in C, \tag{19}$$

$$z_{kj} \le (n-3)x_{kj} + x_{jn} \quad \forall j \in N_G(n),\ n \ne k \in N_G(j), \tag{20}$$

$$x_{ij} = 1 \quad \forall (i,j) \text{ arc bridge}, \tag{21}$$

$$x_{ij} \in \{0,1\} \quad \forall (i,j)\in A, \tag{22}$$

$$y_j \in \{0,1\} \quad \forall j \in V'', \tag{23}$$

$$z_{ij} \ge 0 \quad \forall (i,j)\in A. \tag{24}$$

Note that the node set V is used to configure the tree but only those nodes in V'' have their corresponding y-variable. Although the number of valid inequalities (11) is polynomial, adding all of them to the formulation makes it unnecessarily large and worsens the computational results. Therefore, (11) are separated and added to the formulation inside a branch-and-cut procedure by explicitly checking all of them. A maximum of 1000 violated inequalities is added to the formulation when solving a given instance. There are several possibilities with the upper bound. If ub = 0, then there exists a Hamiltonian path in the graph (a spanning tree without any branch). In such a case, the procedure stops without additional search, since this solution must be optimal. When the upper bound is positive (and integer), whether the linear relaxation of the formulation gives a value greater than ub-1, it is guaranteed that the optimal solution will take value ub and the search is also stopped. Otherwise, the IP search must be used to find better bounds.

8. Computational experiment

All the results of the experiment were obtained using an Intel Core 2 Quad CPU Q9300, 2.50 gigahertz × 4, with 3 gigabytes of RAM memory, running linux Ubuntu 12.04. The heuristic was coded in pascal using Free Pascal Compiler, version 2.4.4–3.1, without additional parameters. The branch-and-cut algorithm was programmed in Xpress mosel 32-bit v3.4.3, with the default parameters (except those oriented to stop the search when the optimality is guaranteed).

Several instances from the literature were used, mainly to compare the results of our method with the results obtained with the latest (and best) approaches in the literature. Note that dense graphs use to contain a Hamiltonian path. To produce more interesting instances, Carrabs et al. (2013) generated sparse graphs with n between 20 and 500 and different densities. They solved (L) by applying CPLEX, with a maximum running time of 1 hour. Some instances with 250 nodes and most of the instances with $n \ge 400$ could not be solved in this time (solving instances with more than 500 nodes was not tried, but we also use instances with up to 1000 nodes generated by them). Different relaxations and heuristic methods were used to produce bounds for all instances. These data are available on the website of the first author of Carrabs et al. (2013) and are the first group of instances we used in the computational study (detailed results of their study are also available there). The instances use to contain many bridges, most of them producing isolated nodes, and our preprocessing phase benefited from this. We call these *medium instances* (when $n \le 500$) or *large instances* (when $n \ge 600$).

Silva et al. (2014) use more difficult instances, which do not contain bridges and cannot be reduced in the preprocessing phase. We have incorporated several of these instances into our study, grouped in three classes, with all of them available following the links in Silva et al. (2014). Graphs called *dimacs* have 450 nodes and between 5714 and 17 425 edges. The edge-swap heuristic these authors use to obtain solutions gave from 0 to 8 branches per graph, and requires several minutes. Compared to the previous algorithms of Cerulli et al. (2009) (re-programmed in Silva et al., 2014), with upper bounds between 1 and 10, the method can be considered satisfactory. Other instances we take from that paper are *hcp* (four instances with 1000 to 4000 nodes) and *stein* (five instances with 1000 nodes and 5000 edges). In all cases the results in Silva et al. (2014) seem to be much better than those obtained by the same authors using the method from Cerulli et al. (2009).

Results for medium instances are reported in Table 1. We have averaged the results by number of nodes. Each line represents 25 different graphs. Times are in seconds. (L) refers to the results obtained in Carrabs et al. (2013) using formulation (L) to optimally solve the instances. **(L)time** only takes into account the instances that could be optimally solved in 3600 seconds (the actual times can be much larger). The best upper bound **(L)ub** they could produce is shown in the table. The rest of columns refer to our method. The total computational time needed by the heuristic procedure (100 runs on each of the 400 instances) was 239 seconds, i.e., 0.6 seconds per instance and 0.006 seconds per run in average (not considered in the table). **C** is the number of isolated nodes removed in the preprocessing phase plus the nonisolated nodes which are removed from V to produce V'', i.e., the constant to be added to the objective value got using formulation (MBV) on graph G. The lower bound produced by the linear relaxation of (MBV) is given in column **lb**, and column **ub** reports the upper bound produced by our heuristic algorithm. Both are very tight. The number of nodes of the branch-and-bound algorithm (Xpress) is **bbn**, and the number of separated valid inequalities in family (11) is **cuts**. The number of **bridges** (of all types) and **cocy**cles used in the solution method are also reported. Column **time** shows the averaged computational times needed to solve the instances optimally. The results are self-explanatory, since all instances could be optimally solved in extremely short times, much shorter than those previously needed to produce approximate solutions.

Similar results are obtained for large instances (Table 2). The total computational time needed by the heuristic procedure (100 runs on each of the 125 instances) was 264 seconds, i.e., 2.1 seconds per instance. In this case, each entry of the table averages five instances of the same size (n and m). Relative deviation of the lower and upper bound with respect to the optimal value is also shown (**lb percent** and **ub percent**, respectively). Again due to the tightness of the bounds, all instances could be optimally solved in minutes of CPU time. Ten minutes of CPU time is the average for the most difficult and largest instances.

Columns **ES** and **EStime** of Table 3 show, respectively, the best upper bounds and the computational times (100 runs) of the Edge-Swap

Table 1
MBV. Medium instances. Optimal results.

n'	m'	(L)time	(L)ub	C	lb	ub	opt	bbn	time	cuts	bridges	cocy
20	41.8	0.0	1.1	0.0	0.6	0.8	0.8	0.4	0.0	0.6	1.9	2.8
40	70.8	0.2	3.7	0.2	2.2	2.9	2.8	0.6	0.1	0.4	6.6	9.1
60	95.0	1.3	8.2	0.6	5.1	6.6	6.3	63.4	1.4	113.8	14.0	18.4
80	119.8	2.2	11.4	1.0	7.7	9.5	9.2	129.8	2.2	183.3	22.0	26.5
100	144.0	4.7	16.3	1.8	11.7	13.9	13.3	147.5	2.9	148.7	31.1	32.8
120	168.8	8.8	21.3	2.6	15.5	18.0	17.5	89.6	3.7	117.5	40.1	41.7
140	193.0	37.2	25.2	2.9	18.6	21.8	20.9	148.0	4.9	203.4	48.0	53.1
160	217.8	50.1	30.2	3.4	22.7	25.8	25.0	88.5	6.1	136.1	57.0	57.9
180	242.0	173.4	34.9	5.0	26.3	30.3	29.1	111.8	6.8	220.8	67.2	66.0
200	266.8	125.3	39.4	5.6	29.9	33.8	32.6	136.2	7.8	176.4	77.6	70.2
250	321.0	534.4	53.1	8.4	41.6	46.0	44.6	197.4	11.3	279.3	104.6	96.0
300	380.0	777.2	66.9	13.0	53.7	59.0	57.4	210.0	13.6	284.8	135.8	109.0
350	434.8	1771.7	78.9	16.0	63.7	70.3	68.6	349.8	20.3	451.1	161.5	138.4
400	489.0	1919.1	93.8	21.8	77.2	83.8	81.8	403.8	24.2	402.1	195.8	156.0
450	548.0	2241.4	107.2	24.1	88.1	95.7	93.4	309.5	29.7	384.5	221.3	177.3
500	602.8	2132.5	120.5	29.8	101.6	109.4	106.7	346.7	35.3	375.2	256.4	190.5

Table 2
MBV. Large instances. Optimal results.

n'	m'	C	lb	lb percent	ub	ub percent	opt	bbn	time	cuts	bridges	cocy
600	637	112.6	180.3	1.9	184.0	0.1	183.8	407.8	5.1	341.0	493.6	68.8
600	674	75.4	163.8	2.1	168.8	1.0	167.2	179.4	10.9	210.0	437.4	71.6
600	712	51.4	147.2	2.3	154.8	2.8	150.6	267.6	19.9	266.2	394.4	68.6
600	749	38.4	136.1	2.0	144.0	3.8	138.8	139.4	26.7	213.0	363.4	55.6
600	787	27.6	123.9	1.6	132.8	5.6	125.8	83.2	39.2	214.2	333.6	49.4
700	740	130.8	211.0	1.6	215.0	0.3	214.4	151.6	6.5	168.0	576.8	91.4
700	780	92.4	193.6	2.2	199.6	0.8	198.0	806.0	16.8	466.4	518.4	89.2
700	821	64.2	175.7	2.4	184.0	2.2	180.0	1846.4	37.8	843.6	470.2	79.4
700	861	50.6	160.8	2.0	169.2	3.2	164.0	606.6	39.7	587.4	436.6	62.8
700	902	38.2	151.1	2.0	161.8	4.9	154.2	465.2	57.8	462.6	403.2	63.6
800	843	154.8	242.0	1.5	246.6	0.4	245.6	99.8	6.7	124.6	666.8	90.6
800	886	109.4	223.4	1.8	229.6	0.9	227.6	391.6	19.1	390.4	599.4	98.8
800	930	78.6	204.2	2.0	213.0	2.2	208.4	6058.4	85.0	808.8	546.6	89.2
800	973	59.0	189.9	2.2	200.2	3.1	194.2	2175.8	65.6	624.4	505.8	82.0
800	1017	43.4	172.3	2.2	184.0	4.4	176.2	4204.6	167.2	615.2	468.2	71.4
900	944	179.4	275.1	1.6	280.6	0.4	279.6	783.6	10.0	316.6	756.4	105.4
900	989	130.6	253.9	2.0	261.6	0.9	259.2	382.2	23.4	409.4	685.6	110.4
900	1034	101.0	235.6	2.1	245.4	2.0	240.6	1996.8	44.5	784.4	633.0	105.0
900	1079	73.6	218.0	2.3	229.8	3.0	223.2	2944.0	94.0	1001.2	583.6	98.0
900	1124	60.6	202.2	1.9	214.8	4.3	206.0	914.2	81.2	481.8	547.6	83.2
1000	1047	206.6	307.5	1.4	313.4	0.4	312.0	694.0	10.6	422.2	849.6	110.2
1000	1095	148.0	283.0	2.4	292.4	0.8	290.0	5286.4	71.1	850.4	767.0	121.0
1000	1143	110.0	265.3	2.2	275.8	1.7	271.2	8825.0	112.4	1001.6	705.0	121.2
1000	1191	86.6	244.9	2.4	257.8	2.7	251.0	5198.4	150.2	932.2	657.6	109.8
1000	1239	62.8	230.2	2.1	244.6	4.0	235.2	28,116.4	642.9	874.8	609.8	105.6

Table 3
MBV. *dimacs* instances. Optimal results.

Instance	n'	m'	ub	time	ES	EStime
le450_5a.col	450	5714	0	1.5	1	310
le450_5b.col	450	5734	0	0.7	1	300
le450_5c.col	450	9803	0	7.2	0	380
le450_5d.col	450	9757	0	2.5	0	380
le450_15a.col	450	8168	0	0.4	4	470
le450_15b.col	450	8169	0	5.5	3	550
le450_15c.col	450	16,680	0	0.4	0	550
le450_15d.col	450	16,750	0	0.4	0	540
le450_25a.col	450	8260	0	0.2	8	800
le450_25b.col	450	8263	0	0.2	4	610
le450_25c.col	450	17,343	0	0.5	0	720
le450_25d.col	450	17,425	0	0.4	0	680

Table 4
MBV. *hcp* and *stein* instances. Heuristic results.

Instance	n'	m'	ub	time	ES	EStime
alb1000	1000	1998	9	84.0	54	1960
alb2000	2000	3996	19	697.0	121	18,350
alb3000a	3000	5999	29	2467.0	191	71,330
alb4000	4000	7997	39	8783.0	247	178,340
steind11.txt	1000	5000	4	45.1	33	4630
steind12.txt	1000	5000	4	47.8	26	4040
steind13.txt	1000	5000	4	45.0	28	4420
steind14.txt	1000	5000	4	42.0	28	4450
steind15.txt	1000	5000	4	48.7	27	4560

heuristic developed in Silva et al. (2014) when applied to the *dimacs* instances. After several minutes, six instances were optimally solved (with optimal value 0), but up to 8 branches were needed in the best solution of one of the instances. Our heuristic method found an optimal value equal to 0 in 1.7 seconds (average time), most of the times in step 1 of Fig. 5.

No optimal solutions could be obtained in 1 hour of CPU time for any of the instances in the other two groups (Table 4). In this case an average time of 1362 seconds (although very different from one instance to another) was needed to produce upper bounds one order of magnitude better than the bounds previously produced in

Repeat several times:
Step 1. *Dynamically modified Prim's algorithm.*
Start with a random edge, adding it to the tree. Mark both extremes.
Repeat $n-2$ times steps 1a and 1b.
Step 1a. Give new weights to the edges of E.
Step 1b. Choose the edge with minimum weight with exactly one extreme marked. Add it to the tree and mark its unmarked extreme.
Step 2. *Improvement phase.*
Repeat IMP until not possible.

Fig. 10. Outline of the heuristic algorithm for the location problem. .

Table 5
MBVL. Medium instances. Optimal results.

n'	m'	C	lb	ub	opt	bv	bbn	time	cuts
20	41.8	32.6	354.0	367.5	359.9	0.8	1.0	0.1	0.4
40	70.8	275.7	1624.3	1697.5	1655.7	1.8	4.6	0.8	4.2
60	95.0	894.6	3957.4	4142.2	4057.4	2.6	15.5	1.4	13.2
80	119.8	1868.7	7242.3	7609.5	7425.3	3.6	27.6	1.9	23.5
100	144.0	3426.2	11,790.2	12,314.0	12,051.9	4.2	39.0	2.7	24.9
120	168.8	5279.5	17,300.1	18,008.4	17,648.2	4.3	56.7	3.5	38.5
140	193.0	7497.4	24,045.4	25,079.1	24,602.5	5.8	85.6	5.2	70.6
160	217.8	10,143.4	31,932.8	33,212.0	32,514.3	6.6	65.3	5.2	49.0
180	242.0	13,821.4	40,732.8	42,344.2	41,488.5	7.9	102.8	7.0	91.5
200	266.8	17,870.7	50,856.7	52,782.6	51,736.7	6.8	160.4	9.1	140.4
250	321.0	30,232.6	81,318.5	84,274.4	82,597.0	9.3	197.7	13.4	144.5
300	380.0	48,726.6	120,143.0	123,936.8	121,801.8	10.6	346.8	17.9	137.6
350	434.8	68,158.4	165,369.2	170,544.6	167,973.0	14.4	473.1	29.3	292.1
400	489.0	96,152.0	219,538.4	226,203.8	222,740.9	15.3	695.7	32.2	282.6
450	548.0	121,663.7	278,513.5	286,873.8	282,515.0	16.9	507.9	39.1	309.7
500	602.8	157,862.2	348,252.3	357,524.6	352,846.1	17.5	736.2	50.3	328.9

Table 6
MBVL. Large instances. Optimal results.

n'	m'	C	lb	lb percent	ub	ub percent	opt	bv	bbn	time	cuts
600	637	433,450.8	577,582.9	0.5	581,484.6	0.2	580,231.4	11.6	79.6	4.8	30.6
600	674	354,383.0	554,014.5	0.6	561,143.6	0.7	557,418.8	12.0	323.8	15.1	249.4
600	712	302,387.4	538,003.2	0.6	547,587.4	1.1	541,493.4	13.6	296.2	25.9	229.4
600	749	265,158.2	520,532.5	0.5	529,226.2	1.2	523,071.8	7.4	319.2	40.2	191.0
600	787	236,270.4	505,522.8	0.4	518,092.8	2.1	507,503.8	8.4	853.4	73.3	401.6
700	740	592,947.6	791,513.3	0.4	796,667.2	0.2	794,778.0	12.2	67.6	5.4	35.0
700	780	495,864.4	762,172.4	0.6	770,137.4	0.5	766,670.2	12.8	342.0	19.0	229.4
700	821	421,315.2	735,905.1	0.6	748,035.8	1.0	740,423.4	14.4	583.4	37.8	454.2
700	861	375,202.2	705,624.4	0.4	718,706.8	1.4	708,778.2	12.4	427.0	46.6	313.2
700	902	336,766.0	691,102.1	0.5	706,066.6	1.6	694,666.4	11.2	243.4	62.8	248.2
800	843	780,242.8	1,028,538.9	0.4	1,034,361.8	0.2	1,032,444.4	16.0	17.6	5.7	17.4
800	886	662,423.4	1,000,575.3	0.5	1,010,451.6	0.5	1,005,737.4	14.4	1034.8	28.7	381.0
800	930	564,123.0	960,610.4	0.6	974,350.8	0.8	966,330.8	15.2	1581.2	54.6	403.8
800	973	500,982.6	935,092.0	0.6	953,264.4	1.3	940,720.2	15.2	494.4	71.3	375.8
800	1017	447,983.2	915,241.6	0.5	938,374.2	2.0	920,102.6	13.6	628.2	99.7	473.8
900	944	1,009,576.6	1,310,572.4	0.4	1,318,059.0	0.1	1,316,129.2	21.2	130.2	7.9	87.8
900	989	856,848.6	1,264,934.9	0.5	1,277,150.4	0.4	1,271,645.8	19.2	477.6	25.7	297.0
900	1034	757,785.2	1,232,485.2	0.6	1,249,716.8	0.8	1,239,497.4	17.8	1257.4	68.7	530.0
900	1079	660,090.2	1,198,628.9	0.6	1,221,917.6	1.3	1,206,271.2	15.6	2126.6	130.3	784.0
900	1124	602,917.8	1,162,639.6	0.5	1,185,854.2	1.5	1,168,248.6	14.4	483.0	104.8	348.6
1000	1047	1,266,659.8	1,613,699.5	0.4	1,621,852.8	0.1	1,619,455.2	18.4	154.6	8.4	73.6
1000	1095	1,064,281.8	1,563,077.2	0.6	1,578,426.0	0.3	1,573,104.0	22.0	542.8	33.5	396.2
1000	1143	931,434.6	1,527,648.2	0.6	1,547,928.2	0.7	1,536,728.8	19.2	802.2	70.4	560.6
1000	1191	835,911.4	1,478,035.9	0.6	1,504,901.4	1.2	1,486,570.0	18.6	4012.2	179.5	885.4
1000	1239	733,989.4	1,444,851.8	0.5	1,477,552.8	1.7	1,452,799.6	17.8	2531.4	234.0	836.6

the literature in about 32,500 seconds. All lower bounds take value 0, which means that all these instances are probably Hamiltonian but the search procedure cannot find the optimal solution.

9. MBV location problem

We have also considered an extended version of the model where the branches can have different costs depending on the node where they are located, and the edges of the graph are weighted. Let $f_j > 0$ be the fixed cost associated to node $j \in V'$, to be paid when j is a branch and let $c_{ij} > 0$ be the fixed cost (weight) associated to edge $\{i, j\}$ if it is in the tree. We call this version MBV Location Problem, MBVL.

Obviously during the preprocessing phase we must add to the constant of the objective function the cost of nodes not in V'' and the costs of the removed bridges.

The heuristic procedure has to be completely re-designed. Since the number of branches is now secondary, and it is their cost that should be reduced when exchanging edges from outside and inside

Table 7
MBVL. Difficult instances. Heuristic results.

Instance	n'	m'	ub
alb1000	1000	1998	1,690,864
alb2000	2000	3996	6,749,783
alb3000a	3000	5999	15,273,177
alb4000	4000	7997	27,051,074
le450_5a.col	450	5714	172,659
le450_5b.col	450	5734	167,779
le450_5c.col	450	9803	148,684
le450_5d.col	450	9757	151,458
le450_15a.col	450	8168	171,539
le450_15b.col	450	8169	175,080
le450_15c.col	450	16,680	133,173
le450_15d.col	450	16,750	134,086
le450_25a.col	450	8260	202,260
le450_25b.col	450	8263	185,211
le450_25c.col	450	17,343	135,824
le450_25d.col	450	17,425	136,362
steind11.txt	1000	5000	78,181
steind12.txt	1000	5000	72,324
steind13.txt	1000	5000	78,401
steind14.txt	1000	5000	88,385
steind15.txt	1000	5000	80,429

the tree, the procedure becomes a little simpler. It is outlined in Fig. 10. In step 1, the weights we use to dynamically adapt Prim's algorithm depend now on the costs of the model:

$$c_{ij}\delta_j(G') \quad \text{if } \delta_i(\tau) \geq 3,$$
$$c_{ij}\delta_j(G') + \frac{\alpha f_i}{\delta_i(G')} \quad \text{if } \delta_i(\tau) = 1,$$
$$c_{ij} + f_i \quad \text{otherwise},$$

where again i is the marked extreme of $\{i, j\}$. The improvement phase IMP is repeated until not possible and then the procedure stops. IMP works by iteratively considering edges not in the tree $\{i, j\}$, identifying edges in the tree that close a cycle with $\{i, j\}$ and replacing one of the edges in the cycle by $\{i, j\}$ if the cost of the tree is reduced (taking into account the costs of both edges and the possible changes in the set of branches of the tree).

After calculating the constant to be added, only a simple change in the objective function of (MBV) suffices to get a valid formulation for MBVL:

$$\text{(MBVL)} \quad \min \sum_{j\in V''} f_j y_j + \sum_{(i,j)\in A} c_{ij} x_{ij}$$
$$\text{s.t.} \quad (12) - (24).$$

10. Computational results for MBVL

We considered the same instances as in Section 8. Since none of them comes with fixed costs, for easy reproducibility we took

$$f_j := 2n' + \text{Round}(j/10) \quad \forall j \in V'.$$

Instances *stein* come with their own weights. For the rest of the instances, trying to compensate both costs and to produce optimal solutions with several branches, we took

$$c_{ij} := 2n' - i - j \quad \forall \{i, j\} \in E$$

(and consequently both arcs with the same cost in (MBV)).

Again all medium and large instances could be optimally solved in a short time (Tables 5 and 6). Now the total computational time needed by the heuristic procedure was 187 (resp. 217) seconds for the medium (resp. large) graphs, i.e., 0.5 (resp. 1.7) seconds per instance in average. Bounds were again really tight. We have included a column in the table with the number of branching vertices of the optimal solution found by the procedure (**bv**). The times and size of the search tree were slightly greater than the times of MBV, and the number of added cuts was similar.

The rest of the instances were heuristically solved, needing a total time of 822 seconds. The upper bounds produced by the algorithm are shown in Table 7.

11. Conclusions

Tightening several constraints, adding or separating valid inequalities and preprocessing the graphs when possible, instances of the Minimum Number of Branches Problem previously considered too difficult have been optimally solved, and good approximate solutions have been generated by a new heuristic method and give better solutions than the best previously known. Also a new problem (which can be considered as a discrete location problem) has been introduced and solved with a modification of the method. Solution methods for many other spanning tree problems in the literature can benefit from some of the ideas presented in the paper.

Acknowledgments

The author would like to acknowledge that research reported here was supported by *Plan Nacional de Investigación Científica, Desarrollo e Innovación Tecnológica (I+D+I)*, project MTM2012-36163-C06-04.

References

Akgün, I., & Tansel, B. (2010). Min-degree constrained minimum spanning tree problem: New formulation via miller-tucker-zemlin constraints. *Computers and Operations Research, 37*, 72–82.
Bhatia, R., Khuller, S., Pless, R., & Sussmann, Y. J. (1999). The full degree spanning tree problem. *Proceedings of the tenth annual ACM-SIAM symposium on discrete algorithms*, (pp. 864–865).
Carrabs, F., Cerulli, R., Gaudioso, M., & Gentili, M. (2013). Lower and upper bounds for the spanning tree with minimum branch vertices. *Computational Optimization and Applications, 56*, 405–438.
Cerrone, C., Cerulli, R., & Raiconi, A. (2014). Relations, models and a memetic approach for three degree-dependent spanning tree problems. *European Journal of Operational Research, 232*, 442–453.
Cerulli, R., Gentili, M., & Iossa, A. (2009). Bounded-degree spanning tree problems: Models and new algorithms. *Computational Optimization and Applications, 42*, 353–370.
Contreras, I., Fernández, E., & Marín, A. (2009). Tight bounds from a path based formulation for the tree-of-hubs location problem. *Computers and Operations Research, 36*, 3117–3127.
Fürer, M., & Raghavachari, B. (1990). An NC approximation algorithm for the minimum degree spanning tree problem. In: *Proceedings of the 28th annual Allerton conference on communication, control and computing*, (pp. 274–281).
Gargano, L., Hell, P., Stacho, L., & Vaccaro, U. (2002). Spanning trees with bounded number of branch vertices. *Lecture Notes in Computer Science, 2380*, 355–365.
Gendron, B., Lucena, A., Salles da Cunha, A., & Simonetti, L. (2013). The degree preserving spanning tree problem: Valid inequalities and a branch-and-cut method. *Electronic Notes in Discrete Mathematics, 41*, 173–180.
Khuller, S., Bhatia, R., & Pless, R. (2003). On local search and placement of meters in networks. *SIAM Journal on Computing, 32*, 470–487.
Landete, M., & Marín, A. (2014). Looking for edge-equitable spanning trees. *Computers and Operations Research, 41*, 44–52.
Merabet, M., Durand, S., & Molnar, M. (2012). Minimization of branching in the optical trees with constraints on the degree of nodes. In: *ICN 2012, The eleventh international conference on networks* (pp. 235–240).
Miller, C. E., Tucker, A. W., & Zemlin, R. A. (1960). Integer programming formulations and travelling salesman problems. *Journal of the Association of Computing Machinery, 7*, 326–329.
Narula, S. C., & Ho, C. A. (1980). Degree-constrained minimum spanning tree. *Computers and Operations Research, 7*, 239–249.
Prim, R. C. (1957). Shortest connection networks and some generalizations. *Bell Systems Technical Journal, 36*, 1389–1401.
Salamon, G., & Wiener, G. (2008). On finding spanning trees with few leaves. *Information Processing Letters, 105*, 164–169.
Schmidt, J. M. (2013). A simple test on 2-vertes- and 2-edge-connectivity. *Information Processing Letters, 113*, 241–244.
Silva, R. M. A., Silva, D. A., Mateus, G. R., Gonçalves, J. F., Resende, M. G. C., & Festa, P. (2011). An iterative refinement algorithm for the minimum branch vertices problem. *Lecture Notes in Computer Science, 6630*, 421–433.
Silva, R. M. A., Silva, D. M., Resende, M. G. C., Mateus, G. R., Mateus, J. F., & Festa, P. (2014). An edge-swap heuristic for generating spanning trees with minimum number of branch vertices. *Optimization Letters, 8*, 1225–1243.
Sundar, S., Singh, A., & Rossi, A. (2012). New heuristics for two bounded-degree spanning tree problems. *Information Sciences, 195*, 226–240.