Computational Optimization and Applications
https://doi.org/10.1007/s10589-018-0027-x

An exact and heuristic approach for the *d*-minimum branch vertices problem

Jorge Moreno[1] · Yuri Frota[1] · Simone Martins[1]

Received: 12 May 2017

Abstract
Given a connected graph $G = (V, E)$, the d-Minimum Branch Vertices (d-MBV) problem consists in finding a spanning tree of G with the minimum number of vertices with degree strictly greater than d. We developed a Miller–Tucker–Zemlin based formulation with valid inequalities for this problem. The results obtained for different values of d show the effectiveness of the proposed method, which has solved several instances faster than previous methods. Also, an heuristic is proposed for this problem, that was tested on several instances of the Minimum Branch Vertices problem, which is the d-MBV problem, when $d = 2$.

Keywords Spanning tree · Branch vertice · Integer programming · Metaheuristic

1 Introduction

Optimization problems related to finding a spanning tree of an undirected graph have been extensively studied in the literature [1,3,4,9,11]. The criterion for choosing this tree depends on the particular problem and may be associated with properties of the vertices, edges, or both.

The *Minimum Branch Vertices* (MBV) problem is associated to the degree of vertices. The goal of this problem is to find a spanning tree with the lowest number of vertices of degree greater than 2. This problem was introduced by Gargano et al. [6] to help the design of optical networks. In Cerrulli et al. [4], a mixed integer modeling and three heuristics for this problem were developed. Other approaches in the literature,

✉ Jorge Moreno
jmoreno@ic.uff.br

Yuri Frota
yuri@ic.uff.br

Simone Martins
simone@ic.uff.br

1 Institute of Computing, Universidade Federal Fluminense, Rio de Janeiro, Brazil

Published online: 21 August 2018

besides proposing mathematical models, presented heuristics or metaheuristics [18] to solve this problem.

In Silva et al. [15] a heuristic was developed based on an exchange of edges, in which edges of higher weight are replaced by edges of lower weight. The weights of the edges were determined by the degrees of the end vertices of the edge. A refinement of this heuristic was proposed by Silva et al. [16]. Other two heuristics were proposed by Carrabas et al. [3], and tested on a wide set of instances. In Marín [9], a new model was presented, as well as a heuristic with the best average results for the instances used in Carrabas et al. [3].

Silvestri et al. [17] developed a hybrid formulation containing undirected and directed variables. This formulation was solved by a branch-and-cut algorithm, improving the results obtained by Marín [9]. Finally, Melo et al. [11] proposed an effective constructive heuristic, which takes into consideration the problem structure in order to obtain good feasible solutions. Also, a decomposition approach based on bridges and cut vertices of the graph was developed, reducing the size of the subproblems to solve.

Recently, a generalization of the MBV problem was proposed by Merabet et al. [12]. This problem uses the concept of k-branch, which is a vertex with degree strictly greater than $k + 2$. The value of k is considered as a tolerance parameter for the design of optical networks, since if a light signal is splitted into k copies, the signal power of one copy will be reduced with, at least, a factor of $1/k$ of the original signal power. The k-Minimum Branch Vertices problem (k-MBV) consists in searching for a spanning tree with the minimum number of k-branch vertices. Merabet et al. [12] proved that this problem is NP-hard whatever the value of k. Also, an ILP based on a single flow formulation was developed and applied on sparse graphs for different values of the parameter k.

To simplify the notation, we introduce a parameter $d = k + 2$ and call a node in the graph with degree strictly greater than d, ($d \geq 2$) of a d-branch vertex. According to this definition the d-MBV problem is defined as:

Problem (*d-MBV problem*) Given an undirected graph $G = (V, E)$ with $n = |V|$ vertices, the *d-Minimum Branch Vertices* (d-MBV) problem consists in finding a spanning tree of G with the minimum number of vertices with degree greater than a fixed integer value d, ($d \geq 2$).

The remainder of this paper is organized as follows: Section 2 presents a mathematical formulation for this problem and the strategies developed to solve it using exact methods. Section 3 describes the proposed heuristic algorithm. Section 4 presents the results obtained for the heuristic and exact method. Finally, the conclusions are discussed in Sect. 5.

2 Mathematical formulation

First, a mathematical formulation for the problem based on the Miller–Tucker–Zemlin formulation is presented. Then, the strategies used to reduce the time to solve the problem using exact methods based on this formulation are discussed.

2.1 Miller–Tucker–Zemlin based formulation

Given an undirected graph $G = (V, E)$, where V denotes the set of vertices ($|V| = n$) and E the set of edges, a neighborhood in G of vertex $v \in V$ is defined as $N_G(v) = \{u \in V \mid \{u, v\} \in E\}$ where $d_G(v) = |N_G(v)|$ is denoted degree of vertex v. Similarly, the neighborhood of a subset $S \subseteq V$ is defined as $N_G(S) = \bigcup\limits_{u \in S} N_G(u)$.

A usual way to model the MBV problem is through formulations based on creating an arborescence with root $r \in V$ and the use of a set of arcs A, such that for all $\{i, j\} \in E$, then $(i, j), (j, i) \in A$. The main difference between these formulations is how to avoid cycles.

In this work, the **Miller–Tucker–Zemlin** (MTZ) constraints [13] are used to avoid cycles. The idea to solve the problem is finding an arborescence with vertex source r. We define variable y_i equal to 1 if the node $i \in V$ is a d-branch vertex, otherwise it is equal to 0. We also define variable x_{ij}, which will take the value 1 if and only if arc $(i, j) \in A$ belongs to the solution. Variable z_i represents the level of vertex $i \in V$ on the arborescence (the root is at level 0). These variables are used to prevent cycles (e.g. if arc x_{ij} belongs to the solution then $z_j > z_i$). The following formulation for the d-MBV problem is shown below.

$$\min \sum_{i=1}^{n} y_i \tag{1}$$

Subject to:

$$\sum_{j \in V:(j,i) \in A} x_{ji} = 1, \quad \forall i \in V, i \neq r \tag{2}$$

$$\sum_{(i,j) \in A} x_{ij} = n - 1, \tag{3}$$

$$x_{ij} + x_{ji} \leq 1, \quad \forall \{i, j\} \in E \tag{4}$$

$$\sum_{j \in V, j \neq r:(i,j) \in A} x_{ij} \leq (d_G(i) - d) y_i + d - 1, \quad \forall i \in V, i \neq r \tag{5}$$

$$\sum_{j \in V:(j,r) \in A} x_{rj} \leq (d_G(r) - d) y_r + d \tag{6}$$

$$\sum_{(j,r) \in A} x_{jr} = 0 \tag{7}$$

$$z_r = 0 \tag{8}$$

$$z_j \geq z_i + x_{ij} n + x_{ji}(n-2) - (n-1), \quad \forall (i, j) \in A, j \neq r \tag{9}$$

$$y_i \in \{0, 1\}, \quad \forall i \in V \tag{10}$$

$$x_{ij} \in \{0, 1\}, \quad \forall (i, j) \in A \tag{11}$$

$$z_i \in \mathbb{Z}, z_i \in [0, n-1], \quad \forall i \in V \tag{12}$$

The objective function (1) requires minimizing the number of d-branch vertices in the tree. Constraints (2) indicate that there must be exactly one edge entering each vertex, with the exception of the root vertex. Constraints (3) forces that the arborescence contains exactly $n - 1$ edges. Constraints (4) require that there is at most one arc between any pair of vertices. Moreover, constraints (5) ensure that a vertex (different from the root) is d-branch if more than $d - 1$ arcs leave it, while constraints (6) assure that the root vertex is d-branch if at least $d + 1$ arcs leave it. Constraints (7) impose that no arcs enter the root vertex. Constraints (8) define that the level of the root vertex is 0, and Constraints (9) determine that the level of each vertex j is greater than the level of the vertex i, if the arc (i, j) belongs to the arborescence. Finally, constraints (10) to constraints (12) ensure the integrality requirements on the variables.

2.2 Graph decomposition

A graph decomposition approach was developed for the MBV problem by Melo et al. [11] and also recently by Landete et al. [7]. The basic idea is to analyze the problem by decomposing the graph into subgraphs that are easier to solve, and then recombine the solutions of these subgraphs to generate a solution to the problem. To implement this decomposition, they detect and analyze bridges and articulation points (or cut vertices) of the graph.

2.2.1 Decomposition based on bridges

A bridge in a graph G is an edge that when removed from the graph increases the number of connected components of G. Finding the set B_G of bridges in the original graph G allows to characterize some of its vertices as obligatorily d-branch vertices.

Let $\beta(v)$ be the number of bridges incident on the vertex v, then we define the following sets:

- $O_D = \{u \in V \,|\, d_G(u) > d \wedge \beta(u) \geq d\}$
- $N_D = \{u \in V \,|\, d_G(u) \leq d\}$

Note that set O_D only contains vertices that will be d-branch in the optimal solution. If the number of adjacent bridges is greater than d, then the vertex has to be d-branch in the optimal solution. Moreover, suppose that vertex v has exactly d adjacent bridges. In this case, if these bridges were removed, the vertex v would belong to a component consisting of more than one vertex ($d_G(v) > d$) and therefore, for any spanning tree on this component, it would be necessary at least one edge incident to vertex v. As a consequence v have at least $d + 1$ incident edges and therefore has to be a d-branch.

On the other hand, the set N_D contains the vertices that cannot be d-branch in any solution, since none of these vertices have enough incident edges to become a d-branch.

As in Melo et al. [11], a decomposition approach can be applied to solve smaller subgraphs using the previous formulation and combining the solutions. The idea is to eliminate each bridge and in its place incorporate the parameter $l(v)$ associated with each vertex that indicates the number of bridges incident to vertex v. If k is the number of resultant connected components, then the optimal solution of the problem is $f = \sum_{i=1}^{k} f_k$, where f_k is the optimal value obtained for the k-th connected component $G_k = (V_k, E_k)$, with $|V_k| = n_k$. The formulation for the d-MBV problem in G_k is presented below.

$$\min \sum_{i=1}^{n_k} y_i \tag{13}$$

Subject to:
(2)–(4), (9)–(12); (replacing n by n_k)

$$\sum_{j \in V, j \neq r : (i,j) \in A} x_{ij} + l(i) \leq (d_{G_k}(i) + l(i) - d) y_i + (d-1), \quad \forall i \in V, i \neq r \tag{14}$$

$$\sum_{j \in V : (r,j) \in A} x_{rj} + l(r) \leq (d_{G_k}(r) + l(r) - d) y_r + d \tag{15}$$

$$y_i = 1, \quad \forall i \in O_D \tag{16}$$

$$y_i = 0, \quad \forall i \in N_D \tag{17}$$

All bridges must be in any spanning tree of the graph. Since every vertex v has associated the value $l(v)$ (i.e. the number of incident bridges), constraints (14), (15) are sufficient to determine whether or not a vertex will be a d-branch vertex in the solution. Note that, except for the root, all the vertices have an incident arc. Therefore, if the number of arcs coming out of the vertex plus the number of adjacent bridges (obligatory edges) is greater than or equal to d, the vertex will be d-branch. For the root the same principle applies: the root vertex will be d-branch if the number of arcs leaving it plus the number of bridges adjacent to it is greater than or equal to $d + 1$. Finally, constraints (16) and (17) are related to the preprocessing phase and are used to help speed up the problem solving process.

2.2.2 Decomposition based on articulation point

Melo et al. [11] showed that it is possible to further decompose the graph considering the articulation points (cutting vertices) which increase the number of connected components of the graph when removed.

Let $G_k = (V_k, E_k)$ be one of the connected graphs obtained after removing all bridges as explained before such that each vertex $v \in V_k$ has associated the number $l(v)$ of adjacent bridges. Let $c_{G_k}(v)$ be the number of connected components by eliminating v of G_k and suppose $c_{G_k}(v) > 1$. Note that the only way to connect these $c_{G_k}(v)$ components is using an edge incident to v. Therefore, $c_{G_k}(v)$ represents a number of obligatory incident edges to v. Since $l(v)$ also represents a number of obligatory incident edges to v, then the vertex v will be a d-branch if $c_{G_k}(v) + l(v) > d$.

The Algorithm 1 named SOLVEGRAPH was developed to solve the d-MBV problem for the subgraphs G' obtained by eliminating the bridges from G. It receives as input the subgraph $G' = (V', E')$, the values $l(v')$ associated to each vertex $v' \in V'$ and the set $Cut_D = \{v' \in V' \mid c_{G'}(v') > 1 \wedge c_{G'}(v') + l(v') > d\}$ which contains all cut vertices that will necessarily be d-branch vertices in the optimal solution.

The procedure SolveModel in line 3 is used to solve the proposed model presented in the Sect. 2.2.1 for the subcomponents G_k that are subgraphs of G' after removing all cut vertices $v \in Cut_D$. In each step, the method delete a vertex $v' \in Cut_D$ (line 5) and create $c_{G'}$ connected components by duplicating vertex v' in each component $k = 1, \ldots, c_{G'}(v')$ (lines 10 and 11). The neighborhood of the new vertex (denoted as v'_k) is defined as the set of vertices to which v' is adjacent in the component k (line 12). Furthermore, the value $l(v'_k)$ will not necessarily hold the same value after this process, since $l(v'_k)$ is incremented by the number of edges incident to v' that do not belong to the component k. Note that vertex v'_k will be classified as a d-branch vertex in each of the connected components, so the value $c_{G'}(v') - 1$ must be subtracted from the number of vertices of the solution in G' (line 13).

2.3 Analyzing the 2-cocycles

Let $G = (V, E)$ be a connected graph. A 2-cocycle is defined as a set of two edges $\{e, f\} \subseteq E$ such that $(V, E \setminus \{e, f\})$ is not connected, while $(V, E \setminus \{e\})$ and $(V, E \setminus \{f\})$ are connected. So, at least one of the edges of a 2-cocyle has to belong to any feasible solution and the following constraints may be added [9,17]:

$$x_e + x_f \geq 1, \ \ \forall \{e, f\} \in C \tag{18}$$

Algorithm 1: SolveGraph

```
Input: A graph without bridges $G' = (V', E')$ and the value $l(v')$ associated to each vertex $v' \in V'$
       and the set $Cut_D$.
Output: The optimal solution for the $d$-MBV problem
1  $s \leftarrow 0$
2  if $Cut_D = \emptyset$ then
3      $s \leftarrow$ SolveModel($G'$)
4  else
5      Choose any $v' \in Cut_D$ and delete it from $G'$
6      Obtain the connected components $G_k = (V_k, E_k)\ k = 1, \ldots, c_{G'}(v')$
7      Let $Cut_D^{(k)} = Cut_D \cap V_k,\ k = 1, \ldots, c_{G'}(v')$
8      for $k \leftarrow 1$ to $c_{G'}(v')$ do
9          /* create a copy of vertex $v'$ in each component $k$ */
10         $V_k \leftarrow V_k \cup \{v'_k\}$
11         $E_k := E_k \cup \{\{u', v'_k\} \mid u' \in V_k \wedge \{u', v'_k\} \in E'\}$
12         $l(v'_k) \leftarrow l(v'_k) + d_{G'}(v') - d_{G_k}(v'_k)$
13         $s \leftarrow s$ + SolveGraph($G_k$,$Cut_D^{(k)}$)
14     $s \leftarrow s - c_{G'}(v') + 1$
15 return $s$
```

where C denotes the set of 2-cocycles of G. We also consider some important properties related to 2-cocycles. Note that the following equivalent definition of 2-cocycle highlights an interesting property:

Definition 1 The pair of edges $\{e, f\}$ is a 2-cocycles, if edge e becomes a bridge when edge f is removed.

Therefore, when removing the edges e and f, the graph will be divided into exactly two connected components. This definition leads to the following propositions:

Proposition 1 *If the pair of edges $\{e, f\}$ is a 2-cocycle, then there is a simple cycle in G that contains both edges.*

Proof Consider the edges $\{e, f\} \in C$, where $e = \{u_1, u_2\}$ and $f = \{v_1, v_2\}$. If the edges $\{e, f\}$ are removed, the graph G is split into two connected components. Note that there will be a vertex of each edge in each component. Suppose that vertices u_1 and v_1 belong to component 1, and vertices u_2 and v_2 belong to the component 2. As they are connected components, there must be simple paths $p(u_1, v_1)$ and $p(u_2, v_2)$ that connect these vertices in each component, then $\{\ e, p(u_2, v_2), f, p(v_1, u_1)\ \}$ is a simple cycle in the original graph. □

Proposition 2 *If the pair of edges $\{e, f\}$ is a 2-cocycle, then they belong to the same set of simple cycles.*

Proof Consider the edges $\{e, f\} \in C$, where $e = \{u_1, u_2\}$ and $f = \{v_1, v_2\}$. As demonstrated in Proposition 1, there is a common cycle c_1 in G that contains e and f. Suppose that e belongs to other simple cycle c_2, while f does not belong to c_2. Note that, if edge f is removed from G, the graph $G - \{f\}$ continues connected (because f belong to cycle c_1). On the other hand, if edge e is removed from $G - \{f\}$, the graph $G - \{f, e\}$ also remains connected (because e belong to cycle c_2). This would be a contradiction, as $\{e, f\} \in C$, so the only possibility is that they belong to the same set of simple cycles. □

Proposition 3 *If the pair of edges e and f belong to the same (not empty) set of simple cycles, then $\{e, f\} \in C$.*

Proof If the edges e and f belong to the same (not empty) set of simple cycles, then, they are not bridges. Moreover, if edge e is removed, then all cycles containing edge f are eliminated, converting f into a bridge and therefore $\{e, f\} \in C$. □

An important implication of the above propositions is the following theorem:

Theorem 1 *The edges $\{e, f\} \in C$, if and only if they belong to the same (not empty) set of simple cycles.*

Now we define the binary relation $\sim$ as $e \sim f$, if and only if $\{e, f\} \in C$. The following proposition is offered as the 1st step towards a new class of valid inequalities:

Proposition 4 *The relation $e \sim f$ is transitive.*

Proof Consider that $e \sim f$ and $f \sim g$. Then by Theorem 1, e and f belong to the same set of simple cycles $S_1 \neq \emptyset$. On the other hand, f and g belong to the same set of simple cycles $S_2 \neq \emptyset$. So, $S_1 = S_2$ and edges e and g belong to the same set of simple cycles. This means that (by Theorem 1) $\{e, g\} \in C$, and consequently $e \sim g$. □

Based on Proposition 4, we define the concept of co-class of an edge.

Definition 2 The co-class of an edge e, denoted as C_e, is defined as: $C_e = \{f \in E \mid e \sim f\} \cup \{e\}$

By the transitivity (and symmetry) of $\sim$, if $f \in C_e$, then $C_f = C_e$. Note that each edge that belongs to a co-class C_e also belongs to a 2-cocycle structure with each other edge from C_e. Therefore, any two edges from C_e cannot be outside the d-MBV solution at the same time (i.e. the solution would be a disconnected graph). For this reason, the following stronger constraints are used in the model instead those used by Marín [9] and Silvestri et al. [17]:

$$\sum_{f \in C_e} x_f \geq |C_e| - 1, \;\; \forall C_e \neq \emptyset, \text{ with } e \in E \tag{19}$$

In this work, we use the same method used in Marín [9] for the detection of 2-cocycles. The method consists of removing a non-bridge edge and applying the bridge detection algorithm proposed by Schmidt [14]. Note that if an edge belongs to a calculated co-class, it is not necessary to perform the procedure to find the associated 2-cocycles, decreasing the required computational time needed.

2.4 Other valid inequalities

In addition to the constraints presented for the d-MBV model, the following inequalities are useful to strengthen the formulation.

$$\sum_{j \in V, j \neq r : (i,j) \in A} x_{ij} + l(i) \geq d\, y_i, \;\forall i \in V, i \neq r \tag{20}$$

$$\sum_{j \in V : (j,r) \in A} x_{rj} + l(r) \geq d\, y_r + 1 \tag{21}$$

Furthermore, a new family of valid inequalities were developed to the d-MBV problem by extending the valid inequalities proposed in Silvestri et al. [17] for the 2-MBV problem.

Proposition 5 *For all $v \in V \setminus \{r\}$, $S \subset N_G^+(v)$ with $|S| \geq d - l(v)$ and $l(v) < d$:*

$$\sum_{(v,u) \in S} x_{vu} + l(v) + 1 \leq (|S| + l(v) + 1 - d) y_v + d \tag{22}$$

where $N_G^+(v) = \{u \in V \mid (v, u) \in A\}$.

Proof For any subset $S \subset N_G^+(v)$, if the sum of the arcs leaving v (i.e. $\sum_{(v,u)\in S} x_{vu}$) with the number of adjacent leaves (i.e. $l(v)$) and the incident arc on v (i.e. $+\ 1$) exceeds the value of d in the solution, then vertex v has to be a d-branch. $\square$

The separation of inequalities in Proposition 5 is pretty straight forward. Let $(\overline{x},\overline{y})$ be a feasible solution for the linear programming relaxation and $\overline{x}_v \subseteq \overline{x}$ be the set of all values of variables $\overline{x}_{vu}, \forall u \in N_G^+(v)$. For each vertice $v \backslash \{r\}$ where $d_G(v) > d > l(v)$, we first sort in descending order the values of $\overline{x}_v$. Then, search for the minimal k for the set $S_v^k = \{u \in N_G^+(v) \mid \overline{x}_{vu}$ is one of the first k elements of $\overline{x}_v\}$ where

$$\sum_{u\in S_v^k} \overline{x}_{vu} + l(v) + 1 > (k + l(v) + 1 - d)\overline{y}_v + d$$

if we find such inequality, then the following cut is added to the formulation

$$\sum_{u\in S_v^k} \overline{x}_{vu} + l(v) + 1 > (k + l(v) + 1 - d)\overline{y}_v + d$$

3 ILS heuristic for the D-MBV problem

The metaheuristic Iterated Local Search (ILS) [8] has been used in several optimization problems and has obtained good quality results [2,5,19].

The pseudocode for the metaheuristic ILS is presented in Algorithm 2.

Algorithm 2: ITERATED LOCAL SEARCH

```
  Input: The graph $G$.
  Output: A valid solution $T$.
1 $x_0 \leftarrow$ Initial_Solution($G$)
2 $x^* \leftarrow$ Local_Search($G$)
3 repeat
4     $x' \leftarrow$ Perturbation($x^*$)
5     $x^{*'} \leftarrow$ Local_Search($x'$)
6     $x^* \leftarrow$ AcceptanceCriterion($x^*, x^{*'}, history$)
7 until termination condition met
8 return $x^*$
```

First, an initial solution is generated for the problem (line 1) and a local search is applied in this solution to improve the quality of the constructed solution (line 2). Between lines 3 and 7, iterations are performed until a stopping criterion is reached. In each iteration, a perturbation in the current solution is done trying to escape of local optimum and then a local search is performed. In line 6, a criterion is used to decide if the current solution will be replaced by the new generated solution.

We propose an ILS heuristic aiming to provide a good quality solution to be used as an upper bound for the previous proposed formulation. The metaheuristic will be applied to each of the connected subgraphs $G' = (V', E')$ resulting from the decomposition process (Sect. 2.2), where G' is a graph without bridges. The solution (upper bound) for the original graph G will be achieved by merging the solutions obtained for each subgraph, using Algorithm 1, replacing method SolveModel(G') (line 13) with the metaheuristic ILS(G').

Procedures for generating the initial solution, performing local search and perturbation were developed. The acceptance criterion updates the current solution if the solution obtained in the local search has less number of d-branch vertices than the best solution found in previous iterations. The termination condition is met when perturbation can no longer be applied. More details are provided in Sect. 3.3.

3.1 Building an initial solution

A heuristic based on the selection of edges of the undirected graph G that are not already in the tree was developed to generate an initial solution.

The following weights w_{sOD} and w_{aOD} of an edge $\{u, v\}$ are defined as:

$$w_{sOD} = d_G(u) + l(u) + d_G(v) + l(v) - f_D(u) \cdot n - f_D(v) \cdot n$$

and

$$w_{aOD} = d_G(u) + l(u) + d_G(v) + l(v) + f_D(u) \cdot n + f_D(v) \cdot n$$

respectively. Moreover, the function $f_D(v)$ defines if a vertex v belongs to the set O_D and is defined as:

$$f_D(v) = \begin{cases} 1 & \text{if } v \in O_D \\ 0 & \text{if } v \notin O_D \end{cases}$$

Algorithm 3 contains the pseudo-code of the strategy used for the initial construction of the solution. A tree T is initialized with all vertices of G without any edges. In this algorithm, the set of arcs A of the graph (as defined in Sect. 2.1) is explored to select the edges that will be in T. Between lines 3 and 6, the arc (u, v) with associated edge $\{u, v\}$ of minimum w_{sOD} value is selected if (i) vertex u has no incident edges in T, (ii) vertex v is incident to T and (iii) the sum of the degree of v in T with its associated $l(v)$ value is different from d. For this selected arc, the associated edge will be added in T. This added edge will not create neither cycles nor new d-branch vertices in T. Also, selecting arcs with minimum w_{sOD} values for the associated edges prioritizes those arcs whose vertices are d-branch vertices and have a small degree in the graph G. In this way, the obligatory vertices will have more added edges.

Between lines 7 and 9, arcs are selected according to the following ordered criteria. First, in criterion (a) an arc (u, v) is selected if vertex u has a degree less than d and if vertex v is a d-branch vertex, so inserting the associated edge will not generate new d-branch vertices in T. This criterion prioritizes arcs whose vertices have to be d-branch vertices in the final solution and have large degree in G. If no vertices are found in criterion a, then an arc with maximum w_{aOD} value in T for the associated edge and

Algorithm 3: INITIAL SOLUTION

```
Input: A graph $G = (V, E)$ without bridges, the value $l(v)$ associated to each vertex, the set $O_D$
       and the set of arcs $A$ such that for all $\{i, j\} \in E$, then $(i, j), (j, i) \in A$.
Output: A spanning tree $T$ of the graph
1  $T \leftarrow (V, \emptyset)$
2  $m \leftarrow 0$
3  repeat
4      Find the arc $(u, v) \in A$ such that $d_T(u) = 0$ and $d_T(v) + l(v) \neq d$ and whose associated edge
       $\{u, v\}$ has minimum $w_{sOD}$ value; then, add the edge $\{u, v\}$ in $T$.
5      $m \leftarrow m + 1$
6  until There is no arc that satisfies this condition
7  repeat
8      Consider criteria (a)-(f), whose priorities are in descending order ((a) has the highest priority).
       Find an arc $(u, v)$ in $A$ with associated edge $\{u, v\}$, such that $T \cup \{u, v\}$ is a tree and no other arc
       satisfies a higher priority criterion.
           (a) arc $(u, v)$ has maximum value $d_G(v) + l(v) + n \cdot f_D(v)$ such that
               $d_T(u) + l(u) < d$ , $d_T(v) + l(v) > d$.
           (b) edge $\{u, v\}$ has maximum $w_{aOD}$ in $T$, and arc $(u, v)$ with
               $d_T(u) + l(u) > d$ , $d_T(v) + l(v) > d$.
           (c) edge $\{u, v\}$ has minimum $w_{sOD}$ in $T$ and arc $(u, v)$ with
               $d_T(u) + l(u) < d$ , $d_T(v) + l(v) < d$.
           (d) edge $\{u, v\}$ has maximum $w_{aOD}$ in $T$ and arc $(u, v)$ with
               $d_T(u) + l(u) > d$ , $d_T(v) + l(v) = d$.
           (e) edge $\{u, v\}$ has maximum $w_{aOD}$ in $T$ and arc $(u, v)$ with
               $d_T(u) + l(u) < d$ , $d_T(v) + l(v) = d$.
           (f) edge $\{u, v\}$ has maximum $w_{aOD}$ in $T$ and arc $(u, v)$ with
               $d_T(u) + l(u) = d$ , $d_T(v) + l(v) = d$.

       Add $\{u, v\}$ in $T$
       $m \leftarrow m + 1$
9  until $m < |V| - 1$
10 return $T$
```

whose both vertices are already d-branch vertices is selected (criterion (b)), so that the number of d-branch vertices is not incremented. This criterion also prioritizes arcs whose vertices have to be d-branch vertices and have large degree. Again, if no vertices are found in criterion (b), an arc is selected if both vertices are not already d-branch vertices in T in such a way that when an edge is inserted in T none of them turns to be a d-branch vertex (criterion (c)). This criterion prioritizes arcs whose vertices have small degrees, letting the vertices with larger degrees to be analyzed later. Moreover, criteria (d), (e) and (f) choose arcs whose associated edges will create new d-branch vertices, (one when using (d) and (e) and two when using (f)). These criteria select arcs whose vertices have large degrees, so when a vertex turns to be a d-branch vertex, there is a chance that new edges will be chosen incident to this vertex in the next selections. The edge associated to the arc selected in line 8 will be inserted in the tree T.

3.2 Local search

There is no guarantee that the initial solution returns a locally optimal solution with respect to some neighborhood. Therefore, the tree obtained by Algorithm 3 may be improved by the local search procedure described in Algorithm 4. In line 1, the best current solution T_{best} is initialized with the solution T_{curr}, and the procedure FirstBest_Neighbor (line 4), described in Algorithm 5, is executed while there is improvement in the current solution.

```
Algorithm 4: LOCAL SEARCH
  Input: The current solution $T_{curr}$.
  Output: The best solution $T_{best}$.
1 $T_{best} \leftarrow T_{curr}$
2 repeat
3     $improvement \leftarrow 0$
4     $T \leftarrow$FirstBest_Neighbor($G$,$T_{best}$, $O_D$)
5     if $T \neq T_{best}$ then
6         $T_{best} \leftarrow T$
7         $improvement \leftarrow 1$
8 until $improvement = 0$
9 return $T_{best}$
```

```
Algorithm 5: FIRSTBEST_NEIGHBOR
  Input: A graph $G = (V, E)$ without bridges, the current solution $T_{curr}$ and the set $O_D$ of
         obligatory $d$-branch vertices.
  Output: The solution $T$.
1  $T \leftarrow T_{curr}$
2  $L_D \leftarrow O_D$
3  $DB \leftarrow \{v \in V \setminus O_D \mid d_T(v) + l(v) > d\}$
4  Sort $DB$ in ascending order by value $(d_T(v) + l(v))$
5  foreach $v \in DB$ do
6      foreach $u \in N_T(v)$ do
7          Remove the edge $\{u, v\}$ from $T$
8          $e \leftarrow$Find_Edge($u, v, T, L_D$)
9          if $e \neq \emptyset$ then
10             Insert the edge $e$ in $T$
11         else
12             Insert the edge $\{u, v\}$ in $T$
13
14         /* $v$ or $u$ are $d-$branch vertices no more */
15         if $d_T(v) + l(v) \leq d$ or $((d_T(u) + l(u) \leq d) \wedge (u \in DB))$ then
16             return $T$
17     $L_D \leftarrow L_D \cup \{v\}$
18 return $T$
```

The FirstBest_Neighbor method (Algorithm 5) looks for a neighbor solution (i.e. a solution obtained by swapping some edges) with less d-branch vertices than the current solution. The aim of this procedure is to remove edges from a d-branch vertex until its degree is equal or less than d. In line 3 the list of candidate vertices DB

Algorithm 6: FIND EDGE

```
Input: Vertices $u$ and $v$, a forest $T$, where $T_u = (V_u, E_u)$ and $T_v = (V_v, E_v)$ are the connected
       subtrees containing $u$ and $v$ respectively, and the set $L_D$ of $d$-branch vertices.
Output: An edge $e \neq \{u, v\}$ between $T_u$ and $T_v$ or $\emptyset$ if not exists.
1  $U_1 \leftarrow L_D \cap V_u$
2  $U_2 \leftarrow \{w \in V_u \mid d_T(w) + l(w) < d\}$
3  $U_3 \leftarrow \{w \in V_u \setminus L_D \mid d_T(w) + l(w) > d\}$
4
5  $V_1 \leftarrow L_D \cap V_v$
6  $V_2 \leftarrow \{w \in V_v \mid d_T(w) + l(w) < d\}$
7  $V_3 \leftarrow \{w \in V_v \setminus L_D \mid d_T(w) + l(w) > d\}$
8
9  $e \leftarrow \{u', v'\}$ such that $\{u', v'\} \neq \{u, v\}$, $u' \in U_i$, $v' \in V_j$ and $(i, j)$ is lexicographically smaller than
   any other valid pair $(i, j \in \{1, 2, 3\})$.
10 return $e$
```

is initialized with vertices that are not obligatory and whose degree is greater than d. The d-branch vertices in DB with smaller degree should be easier to process, so DB is sorted in ascending order related to the degree of vertices in T. Then, for each $v \in DB$, each one of its neighbors u is analyzed. In lines 5 to 17, the procedure tries to find another edge (different from the one that connects u and v in T) that does not create new d-branch vertices. If a new tree is obtained with fewer d-branch vertices the procedure returns it in line 16. Otherwise, vertex v is inserted in the list L_D of d-branch vertices.

Algorithm 6 shows the procedure that tries to find an edge different from $\{u, v\}$ that connects the subtrees T_u and T_v so that T has fewer d-branch vertices. Subtree T_u is the subtree obtained from T when $\{u, v\}$ is removed and contains vertex u, while subtree T_v is the subtree that contains vertex v. The following sets of vertices are defined for each subtree T_u and T_v :

- U_1,V_1: contain vertices that are obligatory d-branch vertices or have already been processed by Algorithm 5 and are considered d-branch vertices.
- U_2,V_2: contain vertices that are not d-branch vertices and if an edge incident to them is inserted, they do not turn to be d-branch vertices.
- U_3,V_3: contain vertices that are d-branch vertices but are not obligatory d-branch vertices or have not yet been considered d-branch vertices by the Algorithm 5.

The search of a new edge is performed by looking for edges $\{u', v'\}$ (or $\{v', u'\}$) such that $u' \in U_i$, $v' \in V_j$ with $i, j \in \{1, 2, 3\}$, in the following order:

1. $u' \in U_1$ and $v' \in V_1$ and the edge $\{u', v'\}$ has the smallest value $d_G(u') + d_G(v') - n \cdot f_D(u') - n \cdot f_D(v')$. This prioritizes edges whose vertices are obligatory d-branch vertices and have small degree.
2. $u' \in U_1$ and $v' \in V_2$ and the edge $\{u', v'\}$ has the smallest value $d_G(u') - n \cdot f_D(u') + d_G(v')$. In this case, one of the vertices of the edge is a obligatory d-branch and the other vertex has a small degree in the graph. The aim of this criterion is to choose an edge that has one obligatory d-branch vertex and the other one has small degree so that a vertex with small degree is chosen to be part of the solution.

3. $u' \in U_1$ and $v' \in V_3$ and the edge $\{u', v'\}$ has the smallest value $d_G(u') - n \cdot f_D(u') - d_T(v')$. In this case, the objective is to look for an edge with a obligatory d-branch and a non-obligatory d-branch with a large degree, which probably is a d-branch vertex in the optimal solution.
4. $u' \in U_2$ and $v' \in V_2$ and the edge $\{u', v'\}$ has the smallest value $d_G(u') + d_G(v')$. The vertices of these edges are not d-branch vertices and will not turn to be d-branch vertices if an edge incident to them is inserted. The objective is to choose vertices with small degrees.
5. $u' \in U_2$ and $v' \in V_3$ and the edge $\{u', v'\}$ has the smallest value $d_G(u') - d_T(v')$. The objective is to find an edge with one vertex (non d-branch) with a small degree in the graph, and the other vertex is a non-obligatory d-branch vertex with large degree in the tree.
6. $u' \in U_3$ and $v' \in V_3$ and the edge $\{u', v'\}$ has the highest value $d_T(v') + d_T(u')$. The objective is to find an edge whose vertices are non-obligatory d-branch vertices with large degree. This type of vertices are probably in the optimal solution.

In all described cases, the inserted edge will not generate a tree T with new d-branch vertices.

3.3 Perturbation

The perturbation method should enable the algorithm to escape from local optima and provide diversification to the ILS. The method attempts to replace an edge $\{u, v\}$ of T, which has at least one non-obligatory d-branch vertex, by another edge $\{u', v'\} \neq \{u, v\}$, with $u' \in T_u$ and $v' \in T_v$, creating another d-branch vertex (u' or v'). Algorithm 7 presents the pseudocode of the implemented perturbation movement.

Algorithm 7: PERTURBATION

```
Input: A graph $G = (V, E)$ without bridges, the current solution $T_{curr}$ and the set $O_D$ of
       obligatory $d$-branch vertices.
Output: The solution $T$.
1  $T \leftarrow T_{curr}$
2  $DB \leftarrow \{v \in V \setminus O_D \mid d_T(v) + l(v) > d\}$
3  Sort $DB$ in ascending order by value $(d_G(v) + l(v))$
4  foreach $v \in DB$ do
5      foreach $u \in N_T(v)$ do
6          Remove the edge $\{u, v\}$ from $T$
7          Let $\{u', v'\} = argmin_{\{i,j\}}\{d_G(i) + d_G(j) \mid i \in T_u,$
           $j \in T_v$ such $d_T(i) = d$ or $d_T(j) = d,$ but not both $\}$ and $\{u', v'\}$ is not forbidden
8          if $\{u', v'\} \neq \emptyset$ then
9              Insert the edge $\{u', v'\}$ in $T$ and mark this edge as forbidden.
10             return $T$
11         else
12             Insert the edge $\{u, v\}$ in $T$
13 return $T$
```

Each vertex v is analyzed according to the value $d_G(v) + l(v)$ in ascending order. Among all possible edges to be added, the chosen edge $\{u', v'\}$ that will reconnect the tree and create (exactly) one d-branch vertex (i.e. $d_T(u') = d$ or $d_T(v') = d$), should be the one that minimizes:

$$\{u', v'\} = argmin_{\{i,j\}}\{d_G(i) + d_G(j)\}$$

This criterion selects edges whose vertices have small degree in G. In this way, when inserting the new edge, the other vertices with larger degree in G will have more chance to "steal" edges from the d-branch vertices during the local search.

The inserted edges are marked as forbidden and cannot be manipulated by the perturbation process until the method finds a better solution. This rule was established with the purpose of not creating cycles of moves by adding and deleting the same edge without having an improvement over the best value found. The ILS method halts when is not possible to find any unmarked edges to proceed with the perturbation.

4 Computer experiments

To validate the effectiveness of the proposed method, several computational experiments were performed. First, experiments were executed with the most used instances in the literature for the 2-MBV problem: 500 instances proposed by Carrabas et al. [3], which contain sparse graphs with different densities, and 21 instances proposed by Silva et al. [16], which are based on graphs that have a Hamiltonian path and therefore it is possible to find spanning trees without branch vertices. Second, we conducted experiments to investigate the impact of different values of d in a set of random instances. The experiments were developed on an Intel (R) Core i5-4460S CPU @ 2.90 GHz, with 6 Mb of cache and 8 Gb of RAM using Linux and all methods were programmed in C++ language using the gcc compiler.

4.1 Heuristic results

2-MBV results
Tables 1 and 2 show the results obtained by the developed heuristic for the group of instances proposed by Carrabas et al. [3] classified as Medium Instances (Table 1) and Large Instances (Table 2) for the MBV problem ($d = 2$). These instances were also studied in Melo et al. [11], but the authors only presented the results of a subset of instances. In addition, the results presented in Melo et al. [11] did not improve those obtained by Marín [9] in that group of instances and for that reason were not considered in Tables 1 and 2.

Each row represents a group of 25 graphs in Table 1 and a group of 5 graphs in Table 2. The first two columns represent the number of vertices and the average number of edges of each group. Columns 3 and 4 represent the number of vertices and edges respectively (after decomposition). The **opt** column shows the optimal value of each group of instances. The column **ubM** presents the results obtained by the

Table 1 Results of the heuristic for Medium Instances for the MBV problem ($d = 2$)

n’	m’	n_p	m_p	Opt	ubM	gapM	ILS	Gap	Time
20	41.8	18.1	39.9	0.8	0.8	0.0	0.8	**0.0**	0.00
40	70.8	33.4	64.2	2.8	2.9	3.6	3.0	7.1	0.00
60	95.0	46.0	81.0	6.3	6.6	4.8	6.7	6.3	0.00
80	119.8	58.1	97.8	9.2	9.5	3.3	9.6	4.3	0.00
100	144.0	69.0	112.9	13.3	13.9	4.5	13.8	3.8	0.00
120	168.8	80.0	128.7	17.5	18.0	2.9	18.2	4.0	0.01
140	193.0	92.1	145.0	20.9	21.8	4.3	21.6	**3.3**	0.01
160	217.8	103.1	160.8	25.0	25.8	3.2	25.9	3.6	0.02
180	242.0	112.9	174.8	29.1	30.3	4.1	30.2	**3.8**	0.02
200	266.8	122.4	189.2	32.6	33.8	3.7	33.8	**3.7**	0.02
250	321.0	145.4	216.4	44.6	46.0	3.1	45.7	**2.5**	0.03
300	380.0	164.2	244.2	57.4	59.0	2.8	58.7	**2.3**	0.05
350	434.8	188.7	273.3	68.6	70.3	2.5	70.1	**2.2**	0.08
400	489.0	204.3	293.2	81.8	83.8	2.4	83.5	**2.1**	0.11
450	548.0	228.9	326.7	93.4	95.7	2.5	95.3	**2.0**	0.15
500	602.8	243.7	346.4	106.7	109.4	2.5	108.6	**1.8**	0.19

heuristic developed by Marín [9] and the column **gapM** shows the gap obtained by his heuristic in relation to optimum value. The last three columns show the value obtained by the **ILS** heuristic, the **gap** in relation to the optimum value, and the average **time** to process the group of instances in seconds. The gap shows the percentage difference of the value obtained by the heuristic method in relation to the optimum value, using the following equation: $gap = \frac{heur-opt}{opt} \times 100$. Moreover, if a **gap** value is better (or equal) than **gapM**, it is represented in boldface.

Marín [9] presented the best result achieved after executing 100 times each instance using his proposed heuristic (**ubM**). The execution times for Medium and Large instances were 239 s and 264 s respectively leading to a total of 503 s using an Intel Core 2 Quad CPU Q9300, 2.50 GHz $\times$ 4, with 3 Gb of RAM memory and running on Linux. The heuristic developed in this work consumed 17.5 s and 28.7 s to solve the Medium and Large instances respectively with a total time of 46.2 s.

In these Tables 1 and 2 we can see that the gap value decreases as the instance size is increased, which indicates that the proposed heuristic maintains a small absolute difference with respect to the optimum. On the other hand, as the instances become more complex, the proposed heuristic begins to perform better than the one proposed by Marín [9]. For the 25 Large instances, the ILS heuristic obtained 19 better results and 2 ties, while Marín heuristic obtained 4 better values. For the 41 groups of instances, the proposed heuristic obtained strictly better results in 65.8% of the instances and equal results in 9.7% of the instances. The average gap for the heuristic proposed in Marín [9] was 2.6, while the average gap obtained by the ILS heuristic was 2.0.

Table 3 shows the results obtained for 21 instances proposed in Silva et al. [16], also for the MBV problem ($d = 2$), where each line represents a graph. The first three

Table 2 Results of the heuristic for Large Instances for the MBV problem ($d = 2$)

n'	m'	n_p	m_p	Opt	ubM	gapM	ILS	Gap	Time
600	637	106.4	143.4	183.8	184.0	0.1	185.0	0.7	0.0
600	674	162.6	236.6	167.2	168.8	1.0	168.6	**0.8**	0.1
600	712	205.6	317.6	150.6	154.8	2.8	153.0	**1.6**	0.1
600	749	236.6	385.6	138.8	144.0	3.7	140.4	**1.2**	0.1
600	787	266.4	453.2	125.8	132.8	5.6	128.8	**2.4**	0.2
700	740	123.2	163.2	214.4	215.0	0.3	215.4	0.5	0.0
700	780	181.6	261.6	198.0	199.6	0.8	199.8	0.9	0.1
700	821	229.8	350.8	180.0	184.0	2.2	182.4	**1.3**	0.2
700	861	263.4	424.4	164.0	169.2	3.2	167.4	**2.1**	0.2
700	902	296.8	498.8	154.2	161.8	4.9	157.0	**1.8**	0.2
800	843	133.2	176.2	245.6	246.6	0.4	246.6	**0.4**	0.0
800	886	200.6	286.6	227.6	229.6	0.9	229.8	1.0	0.1
800	930	253.4	383.4	208.4	213.0	2.2	211.4	**1.4**	0.1
800	973	294.2	467.2	194.2	200.2	3.1	197.2	**1.5**	0.3
800	1017	331.8	548.8	176.2	184.0	4.4	179.6	**1.9**	0.4
900	944	143.6	187.6	279.6	280.6	0.4	280.6	**0.4**	0.1
900	989	214.4	303.4	259.2	261.6	0.9	261.0	**0.7**	0.2
900	1034	267.0	401.0	240.6	245.4	2.0	243.6	**1.2**	0.3
900	1079	316.4	495.4	223.2	229.8	3.0	226.6	**1.5**	0.5
900	1124	352.4	576.4	206.0	214.8	4.3	209.4	**1.7**	0.4
1000	1047	150.4	197.4	312.0	313.4	0.4	313.2	**0.4**	0.1
1000	1095	233.0	328.0	290.0	292.4	0.8	292.2	**0.8**	0.3
1000	1143	295.0	438.0	271.2	275.8	1.7	275.0	**1.4**	0.4
1000	1191	342.4	533.4	251.0	257.8	2.7	254.8	**1.5**	0.5
1000	1239	390.2	629.2	235.2	244.6	4.0	238.6	**1.4**	0.9

columns of the tables present the file name, the number of vertices n and the number of edges m of the graph. Unfortunately, the decomposition process does not bring any benefits to these instances, so, we omit columns n_p and m_p. Furthermore, columns 4, 6 and 7 show, respectively, the results for these instances obtained by the heuristics of Marín [9] (**ubM**), Melo et al. [11] (**MPE**) and the proposed heuristic in this work (**ILS**). Finally, columns 5 and 8 show the times (in seconds) used by the heuristics **ubM** and **ILS** respectively. In the case of the heuristic **MPE**, the times used to obtain these results are not presented by the authors. Furthermore, if a **ILS** solution (column 7) is better (or equal) than the previous bounds (**ubM** and **MPE**), it is represented in boldface.

The proposed ILS heuristic presents superior performance in time and quality of the results. The optimal value was reached in all instances of the first two groups (*dimacs* and *stein*). For *tcp* instances, the heuristic obtained better values in much less time than required by other heuristics. The value **ubM** is the minimum value obtained in 100 executions, while heuristics **MEP** and **ILS** were executed only once.

Table 3 Heuristics results for *dimacs*, *stein* and *tcp* instances for the MBV problem ($d = 2$)

Instance	n	m	ubM	ubM-time	MPE	ILS	ILS-time
le450_15a	450	5714	0	1.5	0	**0**	0.2
le450_15b	450	5734	0	0.7	0	**0**	0.2
le450_15c	450	9803	0	7.2	0	**0**	0.7
le450_15d	450	9757	0	2.5	0	**0**	0.8
le450_25a	450	8168	0	0.4	0	**0**	0.2
le450_25b	450	8169	0	5.5	0	**0**	0.2
le450_25c	450	16680	0	0.4	0	**0**	0.8
le450_25d	450	16750	0	0.4	1	**0**	0.8
le450_5a	450	8260	0	0.2	0	**0**	0.1
le450_5b	450	8263	0	0.2	0	**0**	0.1
le450_5c	450	17343	0	0.5	0	**0**	0.3
le450_5d	450	17425	0	0.4	0	**0**	0.3
steind11	1000	5000	4	45.1	0	**0**	0.3
steind12	1000	5000	4	47.8	1	**0**	0.3
steind13	1000	5000	4	45.0	0	**0**	0.2
steind14	1000	5000	4	42.0	1	**0**	0.3
steind15	1000	5000	4	48.7	1	**0**	0.3
alb1000	1000	1998	9	84.0	16	**1**	1.0
alb2000	2000	3996	19	697.0	28	**2**	7.8
alb3000a	3000	5999	29	2467.0	43	**4**	35.5
alb4000	4000	7997	39	8783.0	58	**4**	67.5

d-MBV results

In Merabet et al. [12] the authors presented results for the d-MBV problem over a set of random instances with different values of d. Their instances have d-branch vertices for high values of d. Unfortunately, we were not able to use these instances to compare with our method because they were not available. The authors informed that the set of instances was disposed after the experimental analysis, but they provided the generator used by them to create the graphs. So, we used this generator to create new instances which are similar to the instances used by them.

As in Merabet et al. [12] we consider 9 values for the number of vertices $|V| \in \{50, 100, 200, 300, 400, 500, 600, 700, 800\}$ and, for the number of edges m, we used the same equation:

$$m = \left\lfloor |V| - 1 + i \times 1.5 \times \left\lceil \sqrt{|V|} \right\rceil \right\rfloor$$

with $i \in \{1, 2, 3\}$. We have generated 30 instances for each pair $(|V|, i)$ and performed experiments for several values of d. Tables 4 and 5 show the results obtained for values of $d \in \{2, 3, 4, 5\}$ and $d \in \{6, 7, 8, 9\}$. Each row represents a group of 30 graphs. First column indicates the number of vertices of the group. Columns **ILS** and **time** show

Table 4 Heuristic results for $d \in \{2, 3, 4, 5\}$ on random instances

$\|V\|$	$i = 1$			$i = 2$			$i = 3$		
	ILS	Time	Gap	ILS	Time	Gap	ILS	Time	Gap
$d = 2$									
50	6.60	0.000	0.5	3.80	0.001	10.7	2.13	2.134	25.5
100	17.17	0.001	0.6	12.43	0.004	4.5	8.87	0.006	12.2
200	38.00	0.005	0.9	28.93	0.018	3.0	22.30	0.032	8.1
300	60.07	0.011	0.4	47.87	0.045	1.6	37.97	0.089	4.4
400	82.47	0.017	0.2	68.80	0.073	1.2	56.97	0.163	2.6
500	105.20	0.025	0.2	87.73	0.108	1.2	74.93	0.325	2.7
600	128.23	0.039	0.2	107.93	0.188	0.9	93.67	0.428	2.2
700	151.00	0.044	0.2	130.43	0.232	0.7	111.23	0.630	2.0
800	174.77	0.056	0.0	151.47	0.347	0.6	131.23	0.879	1.8
$d = 3$									
50	1.03	0.000	0.0	0.23	0.000	0.0	0.10	0.100	0.0
100	4.77	0.001	1.4	1.37	0.000	0.0	0.37	0.000	0.0
200	11.47	0.002	2.1	5.30	0.002	1.3	1.53	0.001	0.0
300	20.50	0.005	0.7	10.30	0.008	1.3	5.03	0.004	2.7
400	29.87	0.010	0.8	18.27	0.020	1.9	9.87	0.011	3.1
500	38.60	0.015	0.4	25.40	0.034	2.1	13.87	0.024	4.0
600	50.17	0.020	0.5	32.03	0.061	1.4	19.70	0.046	2.1
700	60.43	0.031	0.3	39.67	0.084	1.5	24.97	0.078	2.3
800	69.67	0.038	0.3	47.43	0.139	1.6	32.63	0.154	3.2
$d = 4$									
50	0.13	0.000	0.0	0.03	0.000	0.0	0.00	0.000	0.0
100	1.07	0.000	0.0	0.10	0.000	0.0	0.07	0.000	0.0
200	2.57	0.000	0.0	0.43	0.001	0.0	0.10	0.001	0.0
300	5.47	0.001	0.0	1.37	0.001	0.0	0.47	0.002	0.0
400	8.93	0.003	0.8	3.07	0.002	0.0	1.77	0.002	0.0
500	11.93	0.004	0.8	4.80	0.003	0.0	1.83	0.003	0.0
600	15.80	0.006	0.2	6.73	0.004	0.0	2.93	0.005	0.0
700	19.57	0.010	0.3	9.23	0.007	0.0	4.17	0.006	0.0
800	24.57	0.013	0.7	10.17	0.009	1.3	5.33	0.007	0.0
$d = 5$									
50	0.00	0.000	0.0	0.00	0.000	0.0	0.00	0.000	0.0
100	0.20	0.000	0.0	0.00	0.000	0.0	0.00	0.000	0.0
200	0.47	0.000	0.0	0.03	0.000	0.0	0.03	0.000	0.0
300	1.40	0.000	0.0	0.23	0.000	0.0	0.10	0.000	0.0
400	1.90	0.000	0.0	0.60	0.000	0.0	0.27	0.000	0.0
500	3.60	0.000	0.9	0.93	0.000	0.0	0.30	0.000	0.0
600	3.87	0.000	0.0	1.77	0.000	0.0	0.23	0.000	0.0
700	4.83	0.000	0.0	1.93	0.000	0.0	0.80	0.000	0.0
800	6.77	0.000	0.0	1.87	0.000	0.0	1.10	0.000	0.0

Table 5 Heuristic results for $d \in \{6, 7, 8, 9\}$ on random instances

$\|V\|$	$i = 1$			$i = 2$			$i = 3$		
	ILS	Time	Gap	ILS	Time	Gap	ILS	Time	Gap
$d = 6$									
50	0.00	0.000	0.0	0.00	0.000	0.0	0.00	0.000	0.0
100	0.00	0.000	0.0	0.00	0.000	0.0	0.00	0.000	0.0
200	0.07	0.001	0.0	0.00	0.001	0.0	0.00	0.001	0.0
300	0.33	0.001	0.0	0.00	0.001	0.0	0.00	0.001	0.0
400	0.57	0.001	0.0	0.03	0.001	0.0	0.00	0.002	0.0
500	0.90	0.002	0.0	0.17	0.002	0.0	0.03	0.003	0.0
600	1.00	0.002	0.0	0.53	0.003	0.0	0.07	0.003	0.0
700	1.27	0.003	0.0	0.30	0.004	0.0	0.17	0.004	0.0
800	1.60	0.003	0.0	0.40	0.004	0.0	0.17	0.006	0.0
$d = 7$									
50	0.00	0.000	0.0	0.00	0.000	0.0	0.00	0.000	0.0
100	0.00	0.000	0.0	0.00	0.000	0.0	0.00	0.000	0.0
200	0.03	0.000	0.0	0.00	0.001	0.0	0.00	0.001	0.0
300	0.07	0.001	0.0	0.00	0.001	0.0	0.00	0.001	0.0
400	0.10	0.001	0.0	0.00	0.001	0.0	0.00	0.002	0.0
500	0.10	0.001	0.0	0.00	0.002	0.0	0.00	0.003	0.0
600	0.20	0.002	0.0	0.10	0.003	0.0	0.00	0.003	0.0
700	0.17	0.003	0.0	0.07	0.003	0.0	0.03	0.004	0.0
800	0.37	0.003	0.0	0.07	0.004	0.0	0.03	0.005	0.0
$d = 8$									
50	0.00	0.000	0.0	0.00	0.000	0.0	0.00	0.000	0.0
100	0.00	0.000	0.0	0.00	0.000	0.0	0.00	0.000	0.0
200	0.00	0.000	0.0	0.00	0.001	0.0	0.00	0.001	0.0
300	0.03	0.001	0.0	0.00	0.001	0.0	0.00	0.001	0.0
400	0.03	0.001	0.0	0.00	0.001	0.0	0.00	0.002	0.0
500	0.03	0.001	0.0	0.00	0.002	0.0	0.00	0.002	0.0
600	0.00	0.002	0.0	0.00	0.003	0.0	0.00	0.003	0.0
700	0.03	0.002	0.0	0.00	0.003	0.0	0.00	0.004	0.0
800	0.00	0.003	0.0	0.03	0.004	0.0	0.00	0.005	0.0
$d = 9$									
50	0.00	0.000	0.0	0.00	0.000	0.0	0.00	0.000	0.0
100	0.00	0.000	0.0	0.00	0.000	0.0	0.00	0.000	0.0
200	0.00	0.000	0.0	0.00	0.000	0.0	0.00	0.000	0.0
300	0.00	0.000	0.0	0.00	0.000	0.0	0.00	0.001	0.0
400	0.00	0.001	0.0	0.00	0.001	0.0	0.00	0.002	0.0
500	0.00	0.001	0.0	0.00	0.002	0.0	0.00	0.002	0.0
600	0.00	0.002	0.0	0.00	0.002	0.0	0.00	0.003	0.0
700	0.00	0.002	0.0	0.00	0.003	0.0	0.00	0.004	0.0
800	0.00	0.003	0.0	0.00	0.003	0.0	0.00	0.005	0.0

the upper bound and computational time obtained by the proposed heuristic. Column **gap** shows the gap relative to the optimum. It can be observed that the number of d-branch vertices decreases rapidly while increasing the d value. Our heuristic again shows a good performance, presenting a very small gap relative to the optimum within low computational time. As shown in Tables 4 and 5, our heuristic yielded the optimal value several times, mainly for $d \geq 4$.

4.2 Exact results

2-MBV results
Tables 6 and 7 show the computational time required to obtain exact solutions for the MBV problem ($d = 2$). These optimal values were obtained by solving the model developed in Sect. 2 using Constraints (2) and (21). If the obtained solution was not an integer solution then a search for violated Constraints (22) was performed, and the violated cuts were added to the model.

The first two columns represent the number of vertices and the average number of edges for each group of instances. Columns 3 and 4 show the average number of co-classes and optimal value for each group. Columns 5 and 6 correspond to the times in seconds to find the optimal value by the methods proposed in Marín [9] and Silvestri et al. [17] respectively. Silvestri et al. [17] used one Intel Xeon X5675 running at 3.07 GHz with 96 GB of RAM and a 64-bit Linux operating system. Melo et al. [11] method had a poor performance in this group of instance so we chose to compare our results only with Marín [9] and Silvestri et al. [17].

Table 6 Exact results for Medium Instances for the MBV problem ($d = 2$)

n’	m’	Co-class	Opt	timeM	timeS	Time	Cuts	Nodes
20	41.8	2.4	0.8	0.0	0.0	**0.0**	0.4	0.0
40	70.8	7.5	2.8	0.1	0.1	**0.0**	3.0	2.7
60	95.0	12.2	6.3	1.4	0.5	**0.1**	14.7	17.9
80	119.8	16.4	9.2	2.2	0.7	**0.1**	18.0	4.5
100	144.0	20.2	13.3	2.9	1.0	**0.2**	23.4	23.4
120	168.8	24.7	17.5	3.7	1.1	**0.4**	28.8	28.0
140	193.0	28.7	20.9	4.9	2.0	**0.6**	35.4	71.4
160	217.8	31.9	25.0	6.1	1.9	**0.7**	38.0	36.4
180	242.0	35.6	29.1	6.8	2.5	**1.0**	43.0	86.8
200	266.8	38.4	32.6	7.8	3.1	**0.8**	43.1	41.4
250	321.0	46.1	44.6	11.3	3.1	**1.3**	48.1	101.8
300	380.0	51.8	57.4	13.6	4.2	**1.5**	51.9	79.7
350	434.8	60.1	68.6	20.3	6.9	**3.1**	68.7	221.7
400	489.0	64.9	81.8	24.2	9.1	**2.5**	67.3	181.8
450	548.0	72.3	93.4	29.7	9.5	**3.9**	70.1	335.3
500	602.8	79.0	106.7	35.3	9.8	**3.3**	70.0	206.2

Table 7 Exact results for Large Instances for the MBV problem ($d = 2$)

n'	m'	Co-class	Opt	timeM	timeS	Time	Cuts	Nodes
600	637	37.6	183.8	5.1	3.2	**0.3**	30.0	18.2
600	674	52.8	167.2	10.9	8.7	**1.4**	45.6	94.0
600	712	56.8	150.6	19.9	10.3	**1.9**	64.4	30.0
600	749	50.0	138.8	26.7	17.6	**2.3**	70.2	85.2
600	787	47.8	125.8	39.2	16.2	**3.9**	62.8	197.4
700	740	42.0	214.4	6.5	8.7	**0.5**	37.0	20.0
700	780	58.8	198.0	16.8	11.0	**2.0**	57.8	170.0
700	821	64.6	180.0	37.8	12.5	20.0	74.8	1591.2
700	861	58.4	164.0	39.7	17.4	**7.5**	75.4	904.8
700	902	59.2	154.2	57.8	14.7	**7.5**	93.2	812.0
800	843	45.8	245.6	6.7	10.3	**0.4**	39.4	7.6
800	886	67.4	227.6	19.1	11.2	**1.8**	62.8	178.0
800	930	74.4	208.4	85.0	22.7	**5.8**	90.6	365.4
800	973	74.0	194.2	65.6	48.8	**9.8**	88.0	695.8
800	1017	69.4	176.2	167.2	37.1	**11.3**	103.8	568.6
900	944	51.0	279.6	10.0	12.6	**0.8**	44.6	46.0
900	989	69.8	259.2	23.4	66.2	**2.8**	63.2	268.8
900	1034	79.0	240.6	44.5	30.2	**19.4**	92.6	1121.2
900	1079	86.0	223.2	94.0	90.5	**9.0**	105	406.8
900	1124	77.2	206.0	81.2	30.7	**9.1**	94.6	322.0
1000	1047	52.4	312.0	10.6	26.2	**1.1**	42.2	62.6
1000	1095	78.8	290.0	71.1	17.0	**4.0**	86.0	172.0
1000	1143	91.4	271.2	112.4	57.1	**7.8**	99.4	458.0
1000	1191	91.6	251.0	150.2	75.4	**16.1**	109.4	1138.6
1000	1239	96.4	235.2	642.9	62.6	**31.0**	110.0	1920.8

Column 7 shows the execution time obtained by the proposed model in this work using the IBM ILOG CPLEX 12.6 solver. We use the CPLEX default settings and configure it to run over a single thread of execution and a time limit of one hour. The value reported in this column includes the time used to execute all pre-processing operations described in Sect. 2 and the heuristic described in Sect. 3 to define an upper bound. Finally, the last two columns contain the number of the user's cuts (constraints 19) and nodes. Moreover, the times obtained by our exact method (column 7) that are better (or equal) than the ones obtained in [9] and [17] (columns 5 and 6) are highlighted in boldface.

Martinez et al. [10] showed that the time to solve a based Miller–Tucker–Zemlin formulation may be influenced by the vertex that is chosen as the arborescence root. Moreover, Akgun et al. [1] proposed a methodology to select this root node. In this context, we developed the following criterion to select the root node:

$$r = argmax_{u \in V} \left\{ d_G(u) - (d+1) \cdot l(u) + n \cdot f_D(u) + \sum_{v \in N_G(u)} d_G(v) \cdot (1 + f_D(v)) \right\}$$

Table 8 Exact results for *tcp* instances for the MBV problem ($d = 2$)

Instance	n	m	Opt	timeMelo	Time	Cuts	Nodes
alb1000	1000	1998	0	30.13	284.3	0	875
alb2000	2000	3996	0	180.18	1856.7	0	1094
alb3000a	3000	5999	0	74.70	3600.0	0	40
alb4000	4000	7997	0	1778.87	3600.0	0	0

Table 9 Exact results for values of $d \in \{2, 3, 4\}$ on random instances

$\|V\|$	$i = 1$			$i = 2$			$i = 3$		
	Opt	Time	Nodes	Opt	Time	Nodes	Opt	Time	Nodes
$d = 2$									
50	6.57	0.0	0.1	3.43	0.0	3.8	1.70	1.8	20.7
100	17.07	0.0	1.4	11.90	0.1	3.0	7.90	0.3	31.5
200	37.67	0.0	1.5	28.10	0.1	1.3	20.63	0.7	31.1
300	59.83	0.1	16.3	47.10	0.3	46.5	36.37	0.8	68.9
400	82.33	0.1	0.0	68.00	0.2	0.4	55.53	1.0	34.3
500	105.00	0.1	3.1	86.70	0.4	51.5	72.93	1.5	108.0
600	127.97	0.1	4.8	107.00	0.5	10.5	91.63	1.4	6.5
700	150.70	0.1	0.0	129.53	0.5	11.3	109.10	2.6	114.5
800	174.70	0.1	0.0	150.57	0.8	30.3	128.93	2.8	401.2
$d = 3$									
50	1.03	0.0	13.0	0.23	0.0	0.6	0.10	0.1	2.8
100	4.70	0.0	5.7	1.37	0.1	255.6	0.37	0.0	2.5
200	11.23	0.1	52.3	5.23	0.6	806.9	1.53	0.1	30.3
300	20.37	0.1	2.3	10.17	0.5	294.6	4.90	0.2	35.1
400	29.63	0.1	0.0	17.93	0.2	24.3	9.57	0.2	16.5
500	38.43	0.1	6.6	24.87	0.2	11.0	13.33	0.5	33.3
600	49.90	0.1	2.5	31.60	0.5	24.1	19.30	0.6	37.1
700	60.27	0.1	3.6	39.07	0.5	33.5	24.40	0.9	58.8
800	69.47	0.2	5.2	46.67	0.7	27.9	31.63	6.6	1503.1
$d = 4$									
50	0.13	0.0	0.1	0.03	0.0	0.3	0.00	0.0	0.0
100	1.07	0.0	0.0	0.10	0.0	1.0	0.07	0.0	1.7
200	2.57	0.0	0.9	0.43	0.0	4.7	0.10	0.1	6.4
300	5.47	0.0	3.3	1.37	0.1	10.8	0.47	0.1	13.4
400	8.87	0.1	47.5	3.07	0.1	4.5	1.77	0.2	24.6
500	11.83	0.1	1.3	4.80	0.2	38.5	1.83	0.2	10.6
600	15.77	0.4	302.1	6.73	0.2	38.0	2.93	0.3	26.6
700	19.50	0.1	6.6	9.23	0.3	40.4	4.17	0.3	22.8
800	24.40	0.3	104.3	10.03	0.3	38.0	5.33	0.4	35.8

Table 10 Exact results for values of $d \in \{5, 6, 7, 8, 9\}$ on random instances

$\|V\|$	$i = 1$			$i = 2$			$i = 3$		
	Opt	Time	Nodes	Opt	Time	Nodes	Opt	Time	Nodes
$d = 5$									
50	0.00	0.0	0.0	0.00	0.0	0.0	0.00	0.0	0.0
100	0.20	0.0	0.0	0.00	0.0	1.1	0.00	0.0	0.2
200	0.47	0.0	0.7	0.03	0.0	0.0	0.03	0.0	1.9
300	1.40	0.0	2.5	0.23	0.0	3.3	0.10	0.1	6.3
400	1.90	0.0	0.7	0.60	0.1	9.2	0.27	0.1	10.8
500	3.57	0.1	5.1	0.93	0.1	2.1	0.30	0.2	8.8
600	3.87	0.2	211.3	1.77	0.1	5.8	0.23	0.2	10.6
700	4.83	0.1	37.3	1.93	0.2	42.0	0.80	0.3	42.9
800	6.77	0.1	9.7	1.87	0.2	27.2	1.10	0.3	34.2
$d = 6$									
50	0.00	0.0	0.0	0.00	0.0	1.2	0.00	0.0	0.1
100	0.00	0.0	0.0	0.00	0.0	0.1	0.00	0.0	1.5
200	0.07	0.0	0.0	0.00	0.0	0.1	0.00	0.0	0.0
300	0.33	0.0	3.6	0.00	0.0	2.5	0.00	0.1	4.3
400	0.57	0.0	5.3	0.03	0.1	4.4	0.00	0.1	2.3
500	0.90	0.0	10.3	0.17	0.1	1.2	0.03	0.2	10.8
600	1.00	0.0	1.9	0.53	0.1	26.8	0.07	0.2	2.8
700	1.27	0.1	6.6	0.30	0.1	9.2	0.17	0.2	3.7
800	1.60	0.1	4.7	0.40	0.2	17.5	0.17	0.3	16.4
$d = 7$									
50	0.00	0.0	0.0	0.00	0.0	0.0	0.00	0.0	0.0
100	0.00	0.0	0.0	0.00	0.0	0.0	0.00	0.0	0.8
200	0.03	0.0	0.0	0.00	0.0	0.0	0.00	0.0	0.3
300	0.07	0.0	1.4	0.00	0.0	2.7	0.00	0.1	2.2
400	0.10	0.0	0.9	0.00	0.0	0.2	0.00	0.1	1.0
500	0.10	0.0	1.4	0.00	0.1	1.1	0.00	0.1	2.2
600	0.20	0.0	4.3	0.10	0.1	2.6	0.00	0.2	6.1
700	0.17	0.0	2.6	0.07	0.1	2.3	0.03	0.2	5.5
800	0.37	0.1	1.1	0.07	0.1	1.6	0.03	0.2	3.3
$d = 8$									
50	0.00	0.0	0.0	0.00	0.0	0.0	0.00	0.0	0.0
100	0.00	0.0	0.0	0.00	0.0	0.0	0.00	0.0	0.0
200	0.00	0.0	0.0	0.00	0.0	0.0	0.00	0.0	0.0
300	0.03	0.0	0.0	0.00	0.0	0.0	0.00	0.1	0.1
400	0.03	0.0	0.8	0.00	0.0	0.7	0.00	0.1	1.0
500	0.03	0.0	0.6	0.00	0.1	2.3	0.00	0.1	4.4
600	0.00	0.0	0.3	0.00	0.1	2.0	0.00	0.1	0.4

Table 10 continued

$\|V\|$	$i = 1$			$i = 2$			$i = 3$		
	Opt	Time	Nodes	Opt	Time	Nodes	Opt	Time	Nodes
700	0.03	0.0	0.0	0.00	0.1	0.4	0.00	0.2	3.2
800	0.00	0.1	0.5	0.03	0.1	4.7	0.00	0.2	2.8
$d = 9$									
50	0.00	0.0	0.0	0.00	0.0	0.0	0.00	0.0	0.0
100	0.00	0.0	0.0	0.00	0.0	0.0	0.00	0.0	0.0
200	0.00	0.0	0.0	0.00	0.0	0.0	0.00	0.0	0.0
300	0.00	0.0	0.0	0.00	0.0	0.0	0.00	0.0	0.0
400	0.00	0.0	0.0	0.00	0.0	0.0	0.00	0.1	0.0
500	0.00	0.0	1.0	0.00	0.0	3.0	0.00	0.1	0.9
600	0.00	0.0	0.3	0.00	0.1	1.0	0.00	0.1	4.6
700	0.00	0.0	1.0	0.00	0.1	0.7	0.00	0.1	1.2
800	0.00	0.0	0.6	0.00	0.1	0.5	0.00	0.2	1.6

The first term $(d_G(u) - (d+1) \cdot l(u) + n \cdot f_D(u))$ considers the particular characteristics of vertex u. Vertices that are obligatorily d-branch vertices ($f_D(u) = 1$), with large degree and small $l(u)$ value (adjacent leaves) have a greater chance of being chosen as root. The second term of the expression $(\sum_{v \in N_G(u)} d_G(v) \cdot (1 + f_D(v)))$ benefits those vertices that are in a dense zone of the graph and with adjacent vertices that are obligatorily d -branch vertices. We believe that a good root will be in a dense zone of the graph, reducing the height of the resulting arborescence.

The results show the effectiveness of the proposed exact method. All groups of instances in Table 6 were solved faster than the previous exact methods. The results presented in Table 7 shows that for all instances, excepting the group of graphs with 700 vertices and 821 edges, the optimal values were reached using the proposed formulation in shorter times than previous works. We believe this behavior were obtained due to the preprocessing method, which made it possible to obtain graphs with smaller dimensions, and also allowed to fix the value of several variables by using Constraints (16) and (17), resulting in a lighter formulation.

Table 8 shows the results only for the *tcp* instances, since instances *dimacs* and *stein* were exactly solved by the ILS (the method found a 0 bound). As we stated before, the decomposition process does not bring any benefits to these instances because none of them have cut vertices that split the graph into three or more components and only instance le450_15b has two bridges. For this reason, two (out of four) of these instances were not solved by the proposed exact method into the time limit. On the other hand, Melo et al. [11] method seems to perform particularly well in these instances. As we can see in Table 8, their method could solve the 4 instances into the time limit with a faster machine (Intel Core i7-4790K (4.00 GHz) CPU and 16 GB of RAM).

d-MBV results

The results for the exact approach are presented in Tables 9 and 10, for $d \in \{2, 3, 4, 5, 6, 7, 8, 9\}$.

In Merabet et al. [12] the results obtained by using an exact method with their formulation are shown for different values of d. Their results have shown that the problem becomes more difficult to solve as the parameter d increases. So, we have tested our approach to solve similar instances using exact methods and we came to a different conclusion.

We notice that instances with greater value of d are "easier" to solve because less computational time has been spent to solve them. We believe that the inclusion of the constraints (16) and (17) in our formulation is the main reason for this difference. With the increment of the parameter d it is easier to classify a vertex as non obligatory d-branch. On the other hand, it is also more difficult to classify a vertex as obligatory d-branch.

5 Conclusion

In this work, methods were developed for obtaining exact and heuristic solutions for the d-MBV problem. A decomposition scheme was applied based on the bridges and articulation points of the graph. In the computational experiments, the average number of vertices in relation to the original graph was reduced in 42.3% and the average number of edges was reduced in 52.7%.

An ILS heuristic was developed and obtained good quality results for different d values. Particularly, for the $2-$MBV problem, the heuristic provided better results in 65.8% of the instances and equal results in 9.7% for the instances of Carrabas et al. [3]. For the instances used in Silva et al. [16] the heuristic obtained better results in all instances which previous works have not reached the optimal value.

A based Miller–Tucker–Zemlin formulation and some new valid inequalities were proposed for the problem. The computational results show the effectiveness of the proposed method, since 97.6% of the instances of Carrabas et al. [3] were solved faster than previous works (for the $2-$MBV problem).

Moreover, the experiments on the analyzed instances created based on [12] have shown that the number of solved instances increases and the computational time decreaes as the d value rises, since the number of non-obligatorily vertices is increased.

References

1. Akgün, İ., Tansel, B.Ç.: Min-degree constrained minimum spanning tree problem: new formulation via Miller–Tucker–Zemlin constraints. Comput. Oper. Res. **37**(1), 72–82 (2010)
2. Bastos, L., Ochi, L.S., Protti, F., Subramanian, A., Martins, I., Pinheiro, R.G.S.: Efficient algorithms for cluster editing. J. Comb. Optim. **31**(1), 347–371 (2016)
3. Carrabs, F., Cerulli, R., Gaudioso, M., Gentili, M.: Lower and upper bounds for the spanning tree with minimum branch vertices. Comput. Optim. Appl. **56**(2), 405–438 (2013)
4. Cerulli, R., Gentili, M., Iossa, A.: Bounded-degree spanning tree problems: models and new algorithms. Comput. Optim. Appl. **42**(3), 353–370 (2009)
5. Coelho, V.N., Grasas, A., Ramalhinho, H., Coelho, I.M., Souza, M.J.F., Cruz, R.C.: An ILS-based algorithm to solve a large-scale real heterogeneous fleet VRP with multi-trips and docking constraints. Eur. J. Oper. Res. **250**(2), 367–376 (2016)

6. Gargano, L., Hell, P., Stacho, L., Vaccaro, U.: Spanning trees with bounded number of branch vertices. In: Widmayer, P., Eidenbenz, S., Triguero, F., Morales, R., Conejo, R., Hennessy, M. (eds.) Automata, Languages and Programming. ICALP 2002. Lecture Notes in Computer Science, vol. 2380. Springer, Berlin, Heidelberg (2002)
7. Landete, M., Marín, A., Sainz-Pardo, J.L.: Decomposition methods based on articulation vertices for degree-dependent spanning tree problems. Comput. Optim. Appl. **68**(3), 749–773 (2017)
8. Lourenço, H.R., Martin, O.C., Stützle, T.: Iterated local search: framework and applications. In: Gendreau, M., Potvin, J.Y. (eds.) Handbook of Metaheuristics. International Series in Operations Research & Management Science, vol. 146. Springer, Boston, MA (2010)
9. Marín, A.: Exact and heuristic solutions for the minimum number of branch vertices spanning tree problem. Eur. J. Oper. Res. **245**(3), 680–689 (2015)
10. Martinez, L.C., Da Cunha, A.S.: The min-degree constrained minimum spanning tree problem: formulations and branch-and-cut algorithm. Discret. Appl. Math. **163**, 210–224 (2014)
11. Melo, R.A., Samer, P., Urrutia, S.: An effective decomposition approach and heuristics to generate spanning trees with a small number of branch vertices. Comput. Optim. Appl. **65**(3), 821–844 (2016)
12. Merabet, M., Molnar, M.: Generalization of the Minimum Branch Vertices Spanning Tree Problem. PhD thesis, Nanyang Technological University, Singapore (2016)
13. Miller, C.E., Tucker, A.W., Zemlin, R.A.: Integer programming formulation of traveling salesman problems. J. ACM (JACM) **7**(4), 326–329 (1960)
14. Schmidt, J.M.: A simple test on 2-vertex-and 2-edge-connectivity. Inf. Process. Lett. **113**(7), 241–244 (2013)
15. Silva, D.M., Silva, R.M.A., Mateus, G.R., Gonçalves, J.F., Resende, M.G.C., Festa, P.: An iterative refinement algorithm for the minimum branch vertices problem. In: International Symposium on Experimental Algorithms, pp. 421–433. Springer, Heidelberg (2011)
16. Silva, R.M.A., Silva, D.M., Mateus, G.R., Gonçalves, J.F., Resende, M.G.C., Festa, P.: An edge-swap heuristic for generating spanning trees with minimum number of branch vertices. Optim. Lett. **8**(4), 1225–1243 (2014)
17. Silvestri, S., Laporte, G., Cerulli, R.: A branch-and-cut algorithm for the minimum branch vertices spanning tree problem. Comput. Oper. Res. **81**, 322–332 (2017)
18. Sundar, S., Singh, A., Rossi, A.: New heuristics for two bounded-degree spanning tree problems. Inf. Sci. **195**, 226–240 (2015)
19. Vilar, J.V., Arroyo, J.E.C.: ILS Heuristics for the single-machine scheduling problem with sequence-dependent family setup times to minimize total tardiness. J. Appl. Math. **1**(1), 1–15 (2016)