Optim Eng (2014) 15:855–887
DOI 10.1007/s11081-013-9219-5

Cutting-plane-based algorithms for two branch vertices related spanning tree problems

André Rossi · Alok Singh · Shyam Sundar

Received: 23 September 2011 / Accepted: 17 March 2013 / Published online: 7 June 2013

Abstract A branch vertex is a vertex with degree larger than or equal to three. This paper addresses two spanning tree problems in an undirected, simple graph. The first one is to find a spanning tree that minimizes the number of branch vertices (MBV), and the second one is to find a spanning tree that minimizes the degree sum of branch vertices (MDS). These two problems arise in the design of wavelength-division networks (WDN), when the cost of equipments for enabling multicast communication is to be minimized. After investigating the relations of MBV and MDS with the problem of minimizing the number of leaves in a spanning tree, new formulations based on ILP are proposed for MBV and MDS, along with two cutting plane algorithms for addressing them. A repair function is also introduced for deriving feasible solutions from the candidate trees returned at each iteration of the cutting plane algorithm, as well as a Tabu search procedure for further quality improvement. The resulting hybrid approach is shown to outperform pure ILP formulations and heuristic approaches published earlier.

Keywords Spanning tree · Branch vertex · Cutting plane algorithm · Matheuristic · Tabu search · Wavelength-division

A. Rossi (✉)
Lab-STICC, CNRS, Université de Bretagne-Sud, B.P. 92116, 56321 Lorient Cedex, France
e-mail: andre.rossi@univ-ubs.fr

A. Singh · S. Sundar
Department of Computer and Information Sciences, University of Hyderabad, Hyderabad 500046, India

A. Singh
e-mail: alokcs@uohyd.ernet.in

S. Sundar
e-mail: mc08pc17@uohyd.ernet.in

1 Introduction

The use of fiber optic networks has undergone a rapid and massive spread during the last decade, allowing the unprecedented development of the Internet (Jajszczyk 2005). In addition to the large bandwidth capacity they offer, fiber optic networks can take advantage of the latest advances in electronic and optic technologies without having to be completely replaced. For example, wavelength-division multiplexing allows to expand existing networks bandwidth without having to lay additional fiber, as multiplexers only have to be updated (Zhou and Poo 2005).

One of these technologies is wavelength-division networks (WDN), that allows for different signals to be sent in the same link at the same time, provided they use different wavelengths. WDN are available since the late seventies, and have focused a lot of attention since then (Tomlinson and Lin 1978), as wavelength-division allows for increasing the maximum bandwidth at which fiber optic networks can be operated.

More specifically, the problem of operating WDN at low cost can be introduced as follows. Given a set of communication paths, the minimum number of wavelength to be used in the network is referred to as the maximal link load (Jia et al. 2003), its value is equal to the maximum number of communication paths that use the same link. Using that minimum number of wavelengths could easily be achieved by installing wavelength converters at every node, but this would result in prohibitive costs, as wavelength converters remain expensive devices (Elmirghani and Mouftah 2000). Thus, a classical problem is to minimize the number of wavelength converters (Kleinberg and Kumar 2001) while using as few different wavelengths as possible. This problem is shown to be $\mathcal{NP}$-hard in the strong sense in Wilfong and Winkler (1998).

With the development of the Internet and the rise of multicast technology (Lee and Kim 2007), fiber optic networks tend to be used in a different way. Each node must now be able to broadcast data to all the other nodes in the network. Typical examples of such usages of multicast communication are high definition television, video-on-demand and video conferencing. In such a context, switches that allow replication of signals by splitting light are required (Gargano et al. 2002). Thus, multicast communication is achieved through a so called light-tree, that spans over all the nodes involved in the communication. The nodes with degree larger than or equal to 3 are called branch vertices, and a switch is required at each of these nodes of the light-tree. Hence, finding a spanning tree with a minimum number of branch vertices allows to use multicast communication in the network, at minimum cost.

More formally, let G be an undirected, unweighted, simple graph. It is well known that deciding whether G has an Hamiltonian path or not is a $\mathcal{NP}$-hard problem. If G is proved Hamiltonian, then a spanning tree without branch vertex can be found, leading to a network with no switch.

This paper addresses two closely related problems arising in the design of WDN, and that can be seen as generalizations of the Hamiltonian path problem. The first one is to minimize the number of branch vertices in a spanning tree, and is referred to as MBV (Arora and Subramaniam 2002; Gargano et al. 2002). In the context of WDN, MBV seeks to minimize the number of switches for allowing multicast communication from any node. The second problem is to minimize the degree sum of branch

vertices in a spanning tree and is referred to as MDS. This problem is introduced in Cerulli et al. (2009) as a more realistic version of MBV, where the WDN cost does not only depends on the number of branch vertices, but rather on the degree sum of branch vertices. Thus, it is assumed that the cost of the device to be installed at a branch node is proportional to the degree of that node in the tree, i.e., proportional to the number of times an input signal needs to be replicated. Naturally, if a graph has a Hamiltonian path, then both MBV and MDS have a zero optimal objective value.

MBV and MDS are proved $\mathcal{NP}$-hard in Gargano et al. (2002) and Cerulli et al. (2009) respectively, and are also shown to be $\mathcal{NP}$-hard to approximate. Exact ILP formulations are proposed in Cerulli et al. (2009) along with some heuristic approaches for MBV and MDS. The present paper aims at improving these results by proposing cutting plane algorithms hybridized with Tabu search, so as to either return a proven optimal solution, or a good one if an optimal solution cannot be found within one hour. In that case, the cutting plane algorithm returns a lower bound allowing to assess the maximum gap to optimality for the best solution found so far.

This paper is organized as follows. Section 2 highlights the relations between MBV, MDS and the problem of minimizing the number of leaves in a spanning tree. The cutting plane scheme for addressing MBV is introduced in Sect. 3, along with the hybridization with a Tabu search for producing high quality solutions. Section 4 is devoted to approaches for MDS, and, as these approaches are similar to their counterparts for MBV, only the differences with respective MBV approaches are mentioned. In Sect. 5, the proposed approaches are compared to the exact formulations and heuristics proposed in Cerulli et al. (2009). Section 6 concludes the paper.

2 Properties

2.1 Minimization of the number of switches

Along with MBV and MDS, the authors in Cerulli et al. (2009) mention a third problem related to optical network design. Each branch node must be equipped with switches for replicating and transmitting the signals to their adjacent vertices. The number of switches required to equip a branch vertex is equal to its degree minus two. The problem of finding a spanning tree with a minimum number of switches is denoted by MSW. It is now shown to be equivalent to finding a spanning tree in which the number of leaves is minimized.

Lemma 1 *Minimizing the number of switches in a spanning tree is equivalent to minimizing the number of its leaves.*

Proof Let f_L be the number of leaves and f_{MSW} be the number of switches in a spanning tree T.

For all v in V, the degree of v in T can be written as:

$$\delta(v) = \min\big(2, \delta(v)\big) + \max\big(0, \delta(v) - 2\big)$$

The second term of the right-hand side is the number of switches required at this vertex: it is zero if $\delta(v) \leq 2$. Summing up $\delta(v)$ over V yields:

$$2(n-1) = \sum_{v \in V} \min\big(2, \delta(v)\big) + f_{MSW}$$

It can be noticed that $\min(2, \delta(v))$ is equal to 2 for all vertices except leaves for which it is 1. Since there is no isolated node in a tree, the sum over V in the last equality can be replaced with $2n - f_L$, so

$$f_L = 2 + f_{MSW} \qquad \square$$

We now prove a tight relation between the degree sum of branch vertices denoted by f_{MDS}, the number of branch vertices denoted by f_{MBV} and the number of leaves.

Lemma 2 *The following equality holds for any spanning tree*

$$f_{MDS} = 2 f_{MBV} + f_L - 2$$

Proof The degree sum in a spanning tree can be split into two terms as follows

$$\sum_{v \in V} \delta(v) = 2(n-1) = \sum_{\substack{v \in V \\ \delta(v) \geq 3}} \delta(v) + \sum_{\substack{v \in V \\ \delta(v) \leq 2}} \delta(v) = f_{MDS} + \sum_{\substack{v \in V \\ \delta(v) \leq 2}} \delta(v)$$

Then, the last sum is also split into three terms

$$\sum_{\substack{v \in V \\ \delta(v) \leq 2}} \delta(v) = 2n - 2 \sum_{v \in V} b_v - \sum_{\substack{v \in V \\ \delta(v) = 1}} \delta(v)$$

Where b_v is a binary variable that is set to one if and only if vertex v is a branch vertex. Indeed, the left-hand side can be written as two times the number of vertices minus the contribution of branch vertices to that sum, minus the contribution of the leaves.

Replacing $\sum_{\substack{v \in V \\ \delta(v) \leq 2}} \delta(v)$ in the degree sum expression yields

$$2(n-1) = f_{MDS} + 2n - 2 f_{MBV} - f_L \qquad \square$$

Lemma 2 suggests that MBV, MDS and MSW are closely related problems as they have the same constraints (i.e., the solution must be a spanning tree) and they only differ by their objective function, which are linked with an equality. This raises the question of their independence, that is addressed in the next subsection.

2.2 Independence of MBV, MDS and MSW

MBV, MDS and MSW are closely related problems, as they only differ by their objective function. If an Hamiltonian path exists in a graph, it is obviously an optimal

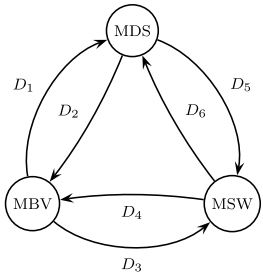

Fig. 1 The six potential dominance relations between MBV, MDS and MSW

solution to the all three problems. Moreover, the early tests conducted on these problems suggested to hypothesize that any optimal solution to MDS is also optimal for MBV. More precisely, two problems $\mathcal{P}_1$ and $\mathcal{P}_2$ are said to be equivalent if and only if any optimal solution to $\mathcal{P}_1$ is optimal to $\mathcal{P}_2$, and any optimal solution to $\mathcal{P}_2$ is optimal to $\mathcal{P}_1$. Besides equivalence, the dominance relation is defined as follows: problem $\mathcal{P}_1$ dominates problem $\mathcal{P}_2$ if any optimal solution to $\mathcal{P}_1$ is also optimal to $\mathcal{P}_2$ (when the converse is also true, $\mathcal{P}_1$ and $\mathcal{P}_2$ are equivalent). An example of such a relation arises with the two single-machine scheduling problems denoted by $1\|L_{max}$ and $1\|T_{max}$ using the three-field notation (Graham et al. 1979). It is recalled that the maximum lateness L_{max} and the maximum tardiness T_{max} of a collection of n jobs are defined as

$$L_{max} = \max_{j\in\{1,\ldots,n\}} (c_j - d_j), \qquad T_{max} = \max(0, L_{max})$$

Where d_j is the due date of job j and c_j is the completion time of job j, for all j in $\{1, \ldots, n\}$. If a solution is optimal for $1\|L_{max}$ with $L_{max} \geq 0$, then it is also optimal for $1\|T_{max}$ as $T_{max} = \max(0, L_{max}) = L_{max}$ in that case. Otherwise, if the optimal solution to $1\|L_{max}$ is such that $L_{max} < 0$, then $T_{max} = 0$ and the solution is also optimal for $1\|T_{max}$ as $T_{max} \geq 0$ by definition. Thus, $1\|L_{max}$ dominates $1\|T_{max}$ but both problems are not equivalent, as can be shown very easily with a counterexample in which $L_{max} < 0$. Consequently, the Earliest Due Date (EDD) algorithm (Pinedo 2007) that solves $1\|L_{max}$ to optimality can also be used for solving $1\|T_{max}$, so there is no need for devising two different algorithms when a dominance relation exists between two problems. When no dominance relation exists, the two problems are said to be independent and the search for two different algorithms is fully justified. In this section, two examples are used for showing that MBV, MDS and MSW are independent.

This section exhibits two counterexample graphs G' and G'' (see Figs. 2 and 3 respectively) for showing that despite the equality of Lemma 2, none of the six potential dominance relations denoted by $D_1, D_2, \ldots, D_6$ (see Fig. 1) hold.

First, MBV and MSW are shown to be independent with the connected, undirected graph G' in Fig. 2.

Solution $\mathcal{T}_1'$ is shown in Fig. 4, it is optimal for both MDS and MBV on G'. Solution $\mathcal{T}_2'$ is an optimal solution to MSW on G', it is given in Fig. 5. Branch vertices have been shown in gray.

Solution $\mathcal{T}_1''$ is shown in Fig. 6, it is optimal for both MDS and MBV on G''. Solution $\mathcal{T}_2''$ is an optimal solution to MDS on G'', it is given in Fig. 7.

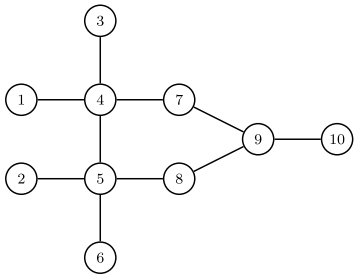

Fig. 2 Counterexample graph G'

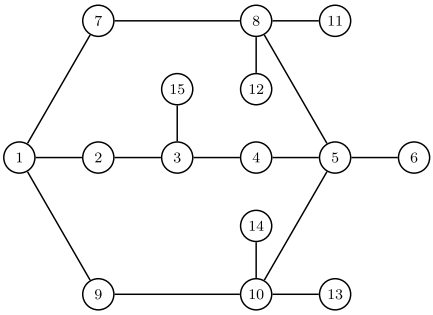

Fig. 3 Counterexample graph G''

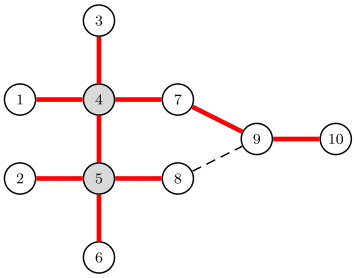

Fig. 4 $\mathcal{T}_1'$, an optimal solution to both MBV and MDS on G'

Table 1 Showing that MSW is not equivalent to MBV nor to MDS

Solution	$\mathcal{T}_1'$	$\mathcal{T}_2'$
Number of branch vertices	**2**	3
Degree sum of branch vertices	**8**	9
Number of switches	4	**3**

When a solution is optimal to a problem, the corresponding objective value is displayed in bold font in Table 1. This table shows that relations D_3, D_4, D_5 and D_6 do not hold, which also proves that MSW is neither equivalent to MBV, nor to MDS.

Second, it is shown that MBV and MDS are independent. A counterexample is used in Cerulli et al. (2009) for showing that relation D_1 does not hold, i.e., any optimal solution to MBV is not optimal to MDS. Now, a counterexample is provided for showing that an optimal solution to MDS is not optimal to MBV.

Table 2 shows that any optimal solution to MDS is not optimal to MBV.

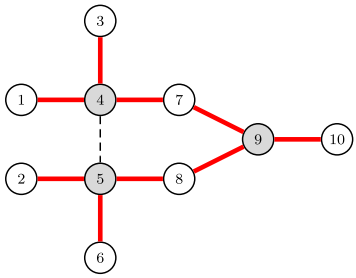

Fig. 5 $\mathcal{T}_2'$, an optimal solution to MSW on G'

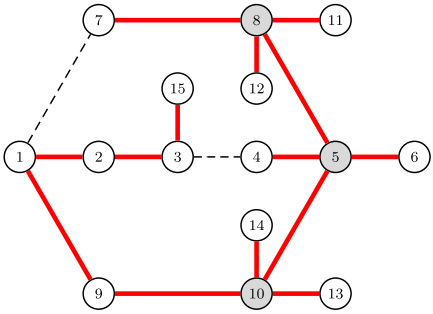

Fig. 6 $\mathcal{T}_1''$, an optimal solution to MBV on G''

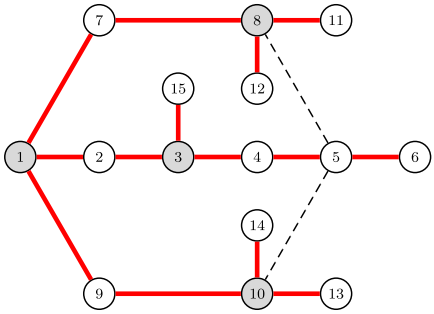

Fig. 7 $\mathcal{T}_2''$, an optimal solution to MDS on G''

Table 2 Showing that an optimal solution to MDS is not optimal to MBV

Solution	$\mathcal{T}_1''$	$\mathcal{T}_2''$
Number of branch vertices	**3**	4
Degree sum of branch vertices	12	**12**

As a result, MBV, MDS and MSW are three independent problems. This justifies the development of specific algorithms for each of these problems. The problem of minimizing the number of leaves in a spanning tree is well-known, so it is not addressed in this paper. The reader is referred to Fernandes and Gouveia (1998), Salamon and Wiener (2008) for exact approaches and approximation schemes to MSW. The rest of this paper is focused on devising efficient cutting plane algorithms for MBV and MDS.

3 Cutting plane schemes for MBV

Section 3.1 introduces the cutting plane algorithm along with two different types of cuts, and a procedure for generating them. A repair function is proposed in Sect. 3.2 for deriving feasible solutions at each iteration of the cutting plane algorithm. Section 3.3 is concerned with the generation of cuts at low computational cost. This is achieved by allocating a variable time limit for solving the master problem at each iteration, and by enforcing a new constraint on the objective function value that aims at taking advantage of the best solution found so far. The first version of the cutting plane algorithm, denoted by CPA_{MBV}, is endowed with all the aforementioned features. The second version, denoted by CPA^{TS}_{MBV}, additionally performs a Tabu search beginning with each feasible solution returned by the repair function for further improving the solution quality. The Tabu search procedure is described in Sect. 3.4.

Such a cutting-plane based approach has been successfully used in Montemanni and Gambardella (2005) for addressing the minimum power symmetry connectivity problem, and in Leggieri et al. (2008) for the minimum power multicasting problem in wireless networks. These two problems, MBV and MDS share the following features: the objective function is solely related to vertices, whereas difficult constraints (i.e., related to connectivity) have to be enforced on edges. In this paper, we follow an iterative approach similar to those used in Montemanni and Gambardella (2005) and Leggieri et al. (2008) and hybridized it with a repair function and a Tabu search.

3.1 The cutting plane algorithm

An integer linear programming approach for MBV, based on a flow formulation, can be found in Cerulli et al. (2009). The proposed cutting plane algorithm for addressing MBV serves as a foundation for CPA_{MBV} and CPA^{TS}_{MBV}. It relies on two sets of binary variables: for all e in E, x_e is set to one, if edge e is part of the solution (it is zero otherwise). For all v in V, y_v is set to one, if vertex v is a branch vertex (it is zero otherwise).

In this paper, $I(v)$ denotes the set of incident edges to vertex v, so the degree of vertex v in the connected, simple, undirected graph G is defined by $\delta_G(v) = |I(v)| \geq 1$ for all $v \in V$.

MBV can be stated as the following integer program denoted by IP_{MBV}:

$$\min \sum_{v \in V} y_v \tag{1}$$

$$\sum_{e \in I(v)} x_e \geq 1 \quad \forall v \in V \tag{2}$$

$$\sum_{e \in E} x_e = n - 1 \tag{3}$$

$$y_v = 0 \quad \forall v \in V, \left|I(v)\right| \leq 2 \tag{4}$$

$$\sum_{e \in I(v)} x_e - 2 \leq \left(\left|I(v)\right| - 2\right) y_v \quad \forall v \in V \tag{5}$$

$$\sum_{e \in CB(S)} x_e \geq 1 \quad \forall S \subset V \tag{6}$$

$$x_e \in \{0, 1\} \quad \forall e \in E \tag{7}$$

$$y_v \in \{0, 1\} \quad \forall v \in V \tag{8}$$

Equation (1) is the objective function. Equation (2) prevents isolated vertices to appear in the solution. Equation (3) ensures that $n - 1$ edges are part of the solution. Constraints (4) and (5) ensure that y_v is set to one whenever the degree of vertex v in the solution is greater than or equal to 3. Equation (6) is the set of all connectivity constraints, which ensures that the solution is a connected subgraph of G. More precisely, S is any strict subset of V, and $CB(S)$ is the *coboundary* of S (Naadef 2004), i.e., the set of all edges that have an endpoint in S and an endpoint in $V \setminus S$, so there is an exponential number of such constraints.

As the number of constraints in (6) is exponential, IP_{MBV} cannot be solved even for modest size instances. Consequently, the *master problem* is initially defined by Eqs. (1)–(5) and (7)–(8), i.e., it is a relaxation of IP_{MBV}, in which connectivity constraints are dropped. A solution to the master problem is called a *candidate tree* as non-connectivity (and cycles) may affect it.

After solving the master problem, the candidate tree is tested for connectivity. If it is not connected, then cuts are introduced in the master problem as follows: Using the proper ILP terminology (Wolsey 1998), a *valid inequality* is said to be a *cut* with respect to a particular solution, if this solution violates the considered valid inequality. That is the reason why the master problem is enriched with cuts, and not constraints.

Alternatively, since a spanning tree is a minimally connected graph and also a maximal cycle-free graph, it is possible to replace connectivity constraints in IP_{MBV} with the following *cycle constraints*:

$$\sum_{e \in CY} x_e \leq |CY| - 1 \quad \forall CY \in \mathcal{C}_G \tag{9}$$

Where $\mathcal{C}_G$ is the set of all cycles in G, and CY is a cycle. Once again, whereas the number of these constraints is exponential, the cutting plane algorithm can introduce cycle cuts in the master problem when necessary, i.e., whenever a cycle is found in the candidate tree.

More precisely, the number of cycles ν in a graph is upper and lower bounded as follows (Volkmann 1996):

$$\mu \leq \nu \leq 2^{\mu} - 1$$

Where μ is the cyclomatic number (or circuit rank), i.e., the minimum number of edges to remove from a graph that makes it cycle-free. It is defined by:

$$\mu = m - n + \kappa$$

Where κ is the number of connected components in the graph. As any solution to the master problem has $n - 1$ edges, $\mu = \kappa - 1$, so the number of cycles in any candidate tree is bounded as follows:

$$\kappa - 1 \leq \nu \leq 2^{\kappa - 1} - 1$$

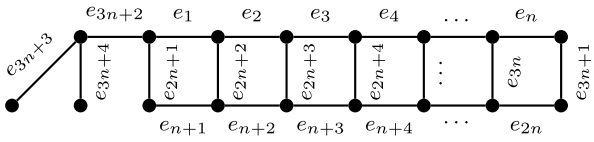

Fig. 8 A graph for showing the use of cycle cuts alone

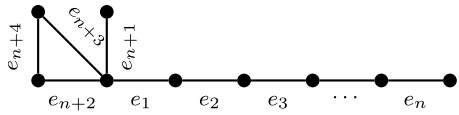

Fig. 9 A graph for showing the use of connectivity cuts alone

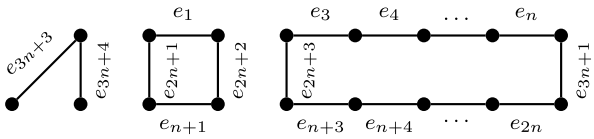

Fig. 10 One of the $n-2$ pairs of cycles

$\kappa - 1$ cycles are enumerated in the candidate tree for generating the same number of cycle cuts with the concern to avoid introducing *dense cuts* (Diaz 2006) in the master problem. To do so, minimum-cardinality cycles are computed on the candidate tree by using a simple heuristic based on a shortest path computation (see Algorithm 2).

While a cutting plane algorithm solely based on connectivity cuts (or solely based on cycle cuts) is sufficient for converging toward an optimal solution to MBV, we have observed that the joint generation of these two types of cuts, implemented in CPA_{MBV} and CPA^{TS}_{MBV}, is computationally more efficient. The two graphs in Figs. 8 and 9 are introduced for illustrating this point.

First, the cutting plane algorithm is run on the graph in Fig. 8, and cycle cuts only are generated. As can be seen in Fig. 10, two cycles are affecting the optimal solution of the master problem. However, enforcing cycle cuts only may not be efficient as at most $n-2$ pairs of cycles may be affecting the master problem solutions before finding an optimal solution. More formally, for all $p \in \{1, \ldots, n-2\}$, pair of cycles p is defined by $(\{e_1, \ldots, e_p, e_{2n+1}, e_{2n+p+1}, e_{n+1}, \ldots, e_{n+p}\}, \{e_{p+2}, \ldots, e_n, e_{2n+p+2}, e_{3n+1}, e_{n+p+2}, \ldots, e_{2n}\})$. Figure 10 shows the pair of cycles for $p = 1$. It can be seen that enforcing the connectivity cut associated with the component made of the endpoints of e_{3n+3} and e_{3n+4} leads to one branch vertex, that the master problem tries to avoid by creating cycles. Hence, enforcing a connectivity cut in that example can save up to $n-2$ iterations compared to the situation in which cycle cuts only are generated.

Second, the cutting plane algorithm is run on the graph in Fig. 9, and connectivity cuts only are now generated. As can be seen in Fig. 11, two connected components are affecting the optimal solution of the master problem. However, enforcing connectivity cuts only may not be efficient as at most $n-2$ pairs of connected components may be affecting the master problem solutions before finding an optimal solution. More formally, for all $p \in \{1, \ldots, n-2\}$, pair of connected components p is defined by the graph where edge e_{p+1} is removed. Figure 11 shows the pair of connected components for $p = 1$. It can be seen that enforcing the cycle cut associated with

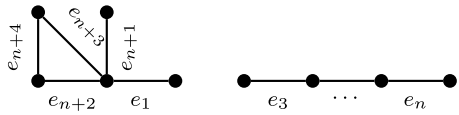

Fig. 11 One of the $n-2$ pairs of connected components

$\{e_{n+2}, e_{n+3}, e_{n+4}\}$ immediately leads to the optimal solution. Hence, enforcing a cycle cut in that example can save up to $n-2$ iterations compared to the situation in which connectivity cuts only are generated.

The cutting plane algorithm is presented in more detail in Algorithm 1. First, the number of branch vertices of the best feasible solution found so far, denoted by f_{UB}, is initialized to n, so that it is updated to a smaller value when the first feasible solution is found. The cutting plane algorithm main loop is run until an optimal solution is found (the Boolean variable *stop* will then be set to `true`). The master problem is then solved, yielding the candidate tree $\mathcal{T}$, i.e., a collection of $n-1$ edges spanning the graph, along with its associated objective value f.

The candidate tree $\mathcal{T}$ is then searched for cycles and connected components with function *CCCD*, where CCCD stands for Cycles and Connected Components Detection. This function is presented in detail in Algorithm 2.

CCCD returns the four following outputs:

- κ, the number of connected components in $\mathcal{T}$,
- $(S_k)_{k\in\{1,\ldots,\kappa\}}$, the collection of κ connected components,
- $(CB_k)_{k\in\{1,\ldots,\kappa-1\}}$, the collection of the associated $\kappa-1$ coboundaries,
- $(CY_j)_{j\in\{1,\ldots,\kappa-1\}}$, a collection of $\kappa-1$ different cycles found in $\mathcal{T}$.

If $\mathcal{T}$ is connected, then it is a spanning tree, i.e., it is optimal for MBV, and the algorithm stops. If $\mathcal{T}$ is not connected, a repair function denoted by $Repair_{MBV}$ is called for generating a feasible solution $\mathcal{T}_k$ from $\mathcal{T}$, taking the connected component S_k as a starting point, for all k in $\{1,\ldots,\kappa\}$. The function $Repair_{MBV}$ is presented in more detail in Algorithm 3. The best solution found so far (denoted by $\mathcal{T}_{UB}$) as well as its objective value f_{UB} are updated with $\mathcal{T}_k$ and f_k, whenever $f_k < f_{UB}$. If f_{UB} is equal to the objective value of the master problem, then $\mathcal{T}_{UB}$ is optimal and the cutting plane algorithm returns it along with the objective value.

Finally, $\kappa-1$ connectivity cuts and $\kappa-1$ cycle cuts are added to the master problem, which is to be solved again until an optimal solution to MBV is found.

Given a solution $\mathcal{T}$ to the master problem, *CCCD* (see Algorithm 2) initializes the number of connected components, κ, to one, the number of cycles, σ, to zero, and the set of vertices to explore, denoted by V_S, to V. This function aims at building S_κ, the current connected component of $\mathcal{T}$. A vertex v_0 is picked in V_S for initializing S_κ. Vertex v_0 that can be chosen arbitrarily in V_S as all connected components of $\mathcal{T}$ are to be enumerated. The set of explored edges of $\mathcal{T}$, denoted by E_S, is initially empty. The Boolean variable *stable* is set to `false` as long as the connected component S_κ can grow. Whenever an unexplored edge of $\mathcal{T}$ with exactly one endpoint in S_κ is found, its other endpoint is added to S_κ, and the considered edge is added to the explored edges set E_S. If an unexplored edge with both endpoints in S_κ is found, then this means that there exists a cycle in E_S. The minimum cardinality cycle containing that edge is found by computing the shortest path between the two endpoints of this edge

```
   /* Initialization                                                    */
1  The master problem is initially defined by Eqs. (1)–(5) and (7)–(8);
2  stop ⟵ false;
3  f_UB ⟵ n;
4  while stop = false do
5      (T, f) ⟵ Solve the master problem;
6      /* Search solution T for cycles and connected
          components                                                    */
7      (κ, (S_k)_{k∈{1,...,κ}}, (CB_k)_{k∈{1,...,κ−1}}, (CY_j)_{j∈{1,...,κ−1}}) ⟵ CCCD(T);
8      if κ = 1 then
9          /* Solution T is a tree                                     */
10         T_UB ⟵ T;
11         f_UB ⟵ f;
12         stop ⟵ true;
13     else
14         /* Solution T is not a tree: it is not connected            */
15         /* Repairing T                                              */
16         k ⟵ 1;
17         while k < κ and stop = false do
18             (T_k, f_k) ⟵ Repair_MBV(T, S_k);
19             if f_k < f_UB then
20                 T_UB ⟵ T_k;
21                 f_UB ⟵ f_k;
22             end
23             if f_UB = f then
24                 stop ⟵ true;
25             end
26             k ⟵ k + 1;
27         end
28         /* Adding cuts to the master problem                        */
29         Add the following set of connectivity cuts to the master problem;
30         ∑_{e∈CB_k} x_e ≥ 1 ∀k ∈ {1, ..., κ − 1};
31         Add the following set of cycle cuts to the master problem;
32         ∑_{e∈CY_j} x_e ≤ |CY_j| − 1 ∀j ∈ {1, ..., κ − 1};
33     end
34 end
35 return (T_UB, f_UB);
```

Algorithm 1: The cutting plane algorithm for MBV

in the graph $G_S = (S_\kappa, E_S)$. The shortest path computation assumes that every edge weight is one. The cycle counter, σ, is incremented by one and the cycle is stored in CY_σ. The edge whose both endpoints are in S_κ is then added to E_S.

All the edges of $\mathcal{T} \backslash E_S$ are tested again until variable *stable* is equal to `true`. V_S is then updated to $V_S \backslash S_\kappa$. If V_S is empty, the algorithm stops and returns its results as all the edges in $\mathcal{T}$ have been explored. Otherwise the vertices in S_κ are not connected to those in $V_S \backslash S_\kappa$, so a connectivity constraints will have to be

Input: $\mathcal{T}$ the solution of the master problem
1 /* Initialization */
2 $\kappa \longleftarrow 1$;
3 $\sigma \longleftarrow 0$;
4 $V_S \longleftarrow V$;
5 **while** $|V_S| > 0$ **do**
6 /* Searching for S_κ, a connected component in V_S */
7 Let v_0 be a vertex in V_S;
8 $S_\kappa \longleftarrow \{v_0\}$;
9 $E_S \longleftarrow \{\emptyset\}$;
10 *stable* $\longleftarrow$ `false`;
11 **while** *stable* = `false` **do**
12 *stable* $\longleftarrow$ `true` ;
13 **for** $(u, v) \in \mathcal{T} \backslash E_S$ **do**
14 **if** $u \in S_\kappa$ and $v \notin S_\kappa$ **then**
15 $S_\kappa \longleftarrow S_\kappa \cup \{v\}$;
16 $E_S \longleftarrow E_S \cup \{(u, v)\}$;
17 *stable* $\longleftarrow$ `false`;
18 **else if** $u \in S_\kappa$ and $v \in S_\kappa$ **then**
19 /* A cycle is detected in S_κ */
20 Find $\mathcal{P}$, a shortest path from u to v in the graph $G_S = (S_\kappa, E_S)$;
21 $\sigma \longleftarrow \sigma + 1$;
22 $CY_\sigma \longleftarrow \mathcal{P} \cup \{(u, v)\}$;
23 $E_S \longleftarrow E_S \cup \{(u, v)\}$;
24 **end**
25 **end**
26 **end**
27 **if** $|S_\kappa| = n$ **then**
28 /* The solution is connected, no coboundary. */
29 $CB_\kappa \longleftarrow \{\emptyset\}$;
30 **else**
31 /* A connectivity constraint is required for connecting S_κ to $V_S \backslash S_\kappa$ */
32 /* Computation of CB_κ */
33 $CB_\kappa \longleftarrow \{\emptyset\}$;
34 **for** $(u, v) \in E \backslash E_S$ **do**
35 **if** $u \in S_\kappa$ and $v \notin S_\kappa$ **then**
36 $CB_\kappa \longleftarrow CB_\kappa \cup \{(u, v)\}$;
37 **end**
38 **end**
39 $\kappa \longleftarrow \kappa + 1$;
40 **end**
41 $V_S \longleftarrow V_S \backslash S_\kappa$;
42 **end**
43 **return** $(\kappa, (S_k)_{k \in \{1,\ldots,\kappa\}}, (CB_k)_{k \in \{1,\ldots,\kappa-1\}}, (CY_j)_{j \in \{1,\ldots,\kappa-1\}})$;

Algorithm 2: *CCCD*: cycles and connected components detection

Input: $\mathcal{T}$ the solution of the master problem and S_k, a connected component of $\mathcal{T}$

/* Initialization */
$\delta(v) \longleftarrow 0 \; \forall v \in V$;
Let v_0 be a vertex in S_k;
$S \longleftarrow \{v_0\}$;
$\mathcal{T}_k \longleftarrow \{\emptyset\}$; $V_S \longleftarrow V$;
while $|V_S| > 0$ **do**
  /* Computing S */
  stable $\longleftarrow$ `false`;
  while *stable* = `false` **do**
    stable $\longleftarrow$ `true`;
    for $(u, v) \in \mathcal{T} \backslash \mathcal{T}_k$ **do**
      if $u \in S$ and $v \notin S$ **then**
        $S \longleftarrow S \cup \{v\}$;
        $\mathcal{T}_k \longleftarrow \mathcal{T}_k \cup \{(u, v)\}$; $\delta(u) \longleftarrow \delta(u) + 1$; $\delta(v) \longleftarrow \delta(v) + 1$;
        stable $\longleftarrow$ `false`;
      else if $u \in S$ and $v \in S$ **then**
        /* A cycle is detected */
        Find $\mathcal{P}$, a shortest path from u to v in the graph $G_S = (S, \mathcal{T}_k)$;
        $CY \longleftarrow \mathcal{P} \cup \{(u, v)\}$;
        /* Selecting the best edge for removal from $\mathcal{T}_k$ */
        $(u_{rem}, v_{rem}) \longleftarrow (0, 0)$; $dec\_max \longleftarrow 0$;
        for $(u', v') \in CY$ **do**
          $dec \longleftarrow 0$;
          if $\delta(u') = 3$ **then** $dec \longleftarrow dec + 1$;
          if $\delta(v') = 3$ **then** $dec \longleftarrow dec + 1$;
          if $dec > dec\_max$ **then** $dec\_max \longleftarrow dec$; $(u_{rem}, v_{rem}) \longleftarrow (u', v')$;
        end
        /* Removing edge (u_{rem}, v_{rem}) from $\mathcal{T}_k$ */
        $\mathcal{T}_k \longleftarrow \mathcal{T}_k \backslash \{(u_{rem}, v_{rem})\}$; $\delta(u_{rem}) \longleftarrow \delta(u_{rem}) - 1$;
        $\delta(v_{rem}) \longleftarrow \delta(v_{rem}) - 1$;
      end
    end
  end
  $V_S \longleftarrow V_S \backslash S$;
  if $|V_S| > 0$ **then**
    /* Selecting the best edge for addition to $\mathcal{T}_k$ */
    $(u_{add}, v_{add}) \longleftarrow (0, 0)$; $inc\_min \longleftarrow 3$;
    for $(u, v) \in E \backslash \mathcal{T}_k$ **do**
      if $u \in S$ and $v \notin S$ **then**
        $inc \longleftarrow 0$;
        if $\delta(u) = 2$ **then** $inc \longleftarrow inc + 1$;
        if $\delta(v) = 2$ **then** $inc \longleftarrow inc + 1$;
        if $inc < inc\_min$ **then** $inc\_min \longleftarrow inc$; $(u_{add}, v_{add}) \longleftarrow (u, v)$;
      end
    end
    /* Adding edge (u_{add}, v_{add}) to $\mathcal{T}_k$ */
    $\mathcal{T}_k \longleftarrow \mathcal{T}_k \cup \{(u_{add}, v_{add})\}$; $\delta(u_{add}) \longleftarrow \delta(u_{add}) + 1$; $\delta(v_{add}) \longleftarrow \delta(v_{add}) + 1$;
  end
end
$f_k \longleftarrow |\{v \in V : \delta(v) \geq 3\}|$;
return $(\mathcal{T}_k, f_k)$;

Algorithm 3: $Repair_{MBV}$: the repair function for MBV

added to the master problem. To this end, the coboundary of S_κ, CB_κ is computed as the set of all edges in that have exactly one endpoint in S_κ. Since at least another connected component exists in that case, κ is incremented by one and S_κ is again initialized with a vertex arbitrarily chosen in V_S. The whole process is repeated until V_S is empty. Then, *CCCD* returns κ, $(S_k)_{k\in\{1,\ldots,\kappa\}}$, $(CB_k)_{k\in\{1,\ldots,\kappa-1\}}$, and $(CY_j)_{j\in\{1,\ldots,\kappa-1\}}$.

It may not be obvious from Algorithm 2 that *CCCD* returns exactly $\kappa - 1$ different cycles from any solution to the master problem. This result is stated and proved in the following lemma.

Lemma 3 *CCCD returns* $\kappa - 1$ *different cycles.*

Proof *CCCD* takes $\mathcal{T}$ as input, i.e., a graph having n vertices and $n - 1$ edges. This graph has κ connected components, so $\kappa - 1$ edges are missing for connecting these components together. Then, these $\kappa - 1$ edges are connecting pairs of vertices inside the κ connected components, where they are responsible for cycles. For all k in $\{1, \ldots, \kappa\}$, let S_k be a connected component of $\mathcal{T}$, and let μ_k be the cyclomatic number of S_k. By definition of the cyclomatic number, $\mu_k = m_k - n_k$ where m_k is the number of edges of $\mathcal{T}$ whose both endpoints are in S_k, and $n_k = |S_k|$. The sum of μ_k over k in $\{1, \ldots, \kappa\}$ is equal to $\kappa - 1$, as any edge missing for connecting $\mathcal{T}$ corresponds to an extra edge that creates at least one cycle in a connected component. As the connected components such that $\mu_k = 0$ are cycle-free, let us consider the connected components S_k such that $\mu_k \geq 1$. In the course of building S_k (lines 11 to 26 in Algorithm 2), each edge with at least one endpoint in S_k is used exactly once, either for connecting a new vertex to S_k (line 14 to 17), or is found to have two endpoints in S_k (line 18 to 23). Because S_k is connected, $n_k - 1$ edges will be used for connecting new vertices, and μ_k edges will be used for enumerating a cycle. It can easily be checked that each edge e among these μ_k edges is used to build a cycle in S_k, which is distinct from all the cycles enumerated so far. Indeed, every time a cycle is enumerated, it contains edge e, which is used for the very first time. This is enforced by the fact that in line 13, unexplored edges only are considered for building S_k. As a result, each connected component S_k for which $\mu_k \geq 1$ is used for enumerating μ_k different cycles. Since the sum of μ_k over $k \in \{1, \ldots, \kappa\}$ is equal to $\kappa - 1$, this proves that *CCCD* returns $\kappa - 1$ different cycles. □

3.2 The repair function

A repair function $Repair_{MBV}$ is called at each iteration of the cutting plane algorithm to generate a feasible solution $\mathcal{T}_k$ as well as its objective value f_k from a solution $\mathcal{T}$ to the master problem (i.e., a candidate tree) and one of its connected component S_k. This repair function is presented in detail in Fig. 3 and is based on the following idea. Suppose that the solution $\mathcal{T}$ to the master problem has $\kappa > 1$ connected components. Then, a feasible solution to MBV can be computed from $\mathcal{T}$ by adding $\kappa - 1$ edges for ensuring connectivity, and by removing $\kappa - 1$ edges so as to make it cycle-free. Edge additions and removals are performed in a greedy way, so as to

maximize the decrease of the number of branch vertices (when removing an edge) and so as to minimize the increase of the number of branch vertices (when adding an edge).

More precisely, $\delta(v)$ is initialized to 0 for all v in V, and $\mathcal{T}_k$ is initialized to the empty set. An arbitrary vertex v_0 in S_k is used for initializing S, the set of vertices covered by the solution $\mathcal{T}_k$ under construction. As in function *CCCD*, V_S is initialized to V and denotes the set of unexplored vertices, so the algorithm runs until V_S is empty. The Boolean variable *stable* is set to `false` as long as set S can grow. If an edge in $\mathcal{T}\backslash\mathcal{T}_k$ has exactly one endpoint in S, then the other endpoint is added to S, the considered edge is added to $\mathcal{T}_k$, and the degree of both its endpoints is incremented by one. If an edge in $\mathcal{T}\backslash\mathcal{T}_k$ has both its endpoints in S, then there is a cycle in $\mathcal{T}_k$. This cycle is denoted by CY and is computed as in *CCCD*. The edge denoted by $(u_{rem}, v_{rem}) \in CY$ has to be removed from $\mathcal{T}_k$ for removing this cycle. This edge is selected as the one whose removal maximizes the decrease in the number of branch vertices. More precisely, if an edge has an endpoint with degree three, then removing it will cause the number of branch vertices to decrease by one. Once (u_{rem}, v_{rem}) has been identified, it is removed from $\mathcal{T}_k$ and the degree of both its endpoints is decremented by one.

Once S is stable, V_S is updated to $V_S\backslash S$. If V_S is empty, the solution is complete and the function returns $\mathcal{T}_k$ and its objective value f_k. Otherwise, an edge which is not in $\mathcal{T}$ must be added to $\mathcal{T}_k$ for connecting it to the vertices in V_S. The edge to add is denoted by (u_{add}, v_{add}), it is selected among the edges that have exactly one endpoint in S, and is such that its addition minimizes the increase in the number of branch vertices in $\mathcal{T}_k$. More precisely, if such an edge has an endpoint with degree two, then adding it will create a new branch vertex. Once (u_{add}, v_{add}) has been identified, it is added to $\mathcal{T}_k$ and the degree of both its endpoints is incremented by one.

This function repairs a solution in a greedy way, so it can be called κ times at every iteration of the cutting plane algorithm. Its second argument S_k allows to set the initial vertex of S in a different connected component of $\mathcal{T}$ at every call, leading to potentially different feasible solutions to MBV. This repair procedure is also useful for saving iterations of the cutting plane algorithm as an optimal solution may often be found quickly by invoking it, especially for dense graphs that are likely to be Hamiltonian as the Dirac density condition (Dirac 1952) may hold.

3.3 Generating cuts at low computational cost

The cutting plane algorithm presented in the Sect. 3.1 is improved based on the following two observations.

1. The proposed cutting plane algorithm can be viewed as a series of ILP solution processes. Except the last one, the main result yielded at each iteration is a collection of cuts that will be integrated in the problem formulation to be solved at the next iteration. Thus, each single iteration is likely to require a lot of computational time and effort.

2. Sparsest graphs are the most difficult instances as can be seen from Sect. 5. But while still practically too large for being introduced in the problem formulation, the number of connectivity constraints (6) and cycle constraints (9) drops significantly as sparsity increases.

These observations suggest generating cuts at low computational cost, i.e., without solving the problem formulation to optimality at each iteration. In a regular cutting plane algorithm, cuts are implicitly defined with respect to the optimal solution found at each iteration. Now, cutting planes are generated with respect to the best integer solution found so far at the current iteration. Thus, this may result in generating valid inequalities (and not *cuts* with respect to an optimal solution to the current formulation). Nevertheless, even if those valid inequalities cannot be proved to be cuts for an optimal solution, the idea is that adding them to the formulation can help for addressing difficult instances. From a pragmatic point of view, solving the problem for a few iterations with cheap valid inequalities yields better lower and upper bounds than attempting to solve the problem formulation to optimality at each iteration. In the sequel, the word *cut* means *cut with respect to the best integer solution found so far in the current iteration*.

Naturally, if cuts are generated from a solution that is very far from optimality, these cuts are bound to be useless as no good solution to the problem (and its subsequent versions in the next iterations) is likely to share the same cycles and connected components. This raises the question of the trade-off between cuts computational cost and quality. To this respect, a regular cutting plane algorithm generates maximum quality cuts as each problem formulation is solved to optimality, but these cuts may be computationally expensive to obtain if the problem to be solved is $\mathcal{NP}$-hard.

This trade-off is empirically addressed as follows.

- Computational cost control
  An adjustable time limit is imposed on the solver for addressing the current problem formulation. However, the time limit is enforced only after the first integer solution is found. Moreover, every α iterations, extra time is given to the solver, letting it the opportunity to reach higher quality solutions by taking advantage of the cheap cuts generated so far.
- Quality enforcement
  The following constraint is added to the master problem:

$$\sum_{v \in V} y_v \leq f_Q \tag{10}$$

  Where f_Q is the maximum number of branch vertices in the solution. It is updated at each iteration as:

$$f_Q \longleftarrow \left\lceil \frac{f_{LB} + f_{UB}}{2} \right\rceil$$

  Where f_{LB} is the maximum lower bound on the number of branch vertices and f_{UB} is the best feasible solution objective value found so far by the repair procedure.

Because of (10), the problem may now be infeasible if f_Q is too low. In that case, f_{LB} is updated to $f_Q + 1$, and f_Q is itself updated using the last formula. Enforcing tighter quality requirement has appeared to be counter-productive as deciding infeasibility is often very time consuming, especially when f_Q is close to the optimal value for *MBV*.

Implementing these features requires a more complex control structure, mostly for setting time parameters for the solver, handling infeasibility, and also because finding a solution with a single connected component is no longer sufficient for proving its optimality. Hence, the following four modifications have to be performed on Algorithm 1.

1. The initialization phase (lines 1 to 3) has to be replaced with

```
The master problem is initially defined by Eqs. (1)–(5), (7)–(8) and (10);
stop ⟵ false;
it ⟵ 0;
add_more_time ⟵ 0;
f_LB ⟵ 0;
f_UB ⟵ n;
```

   Where *it* is an iteration counter, and *add_more_time* is an integer that allows to give more time to the solver for solving the problem if the current solution is a tree, but is not optimal (or has not been proved optimal yet).
2. The call to the solver (line 5) has to be replaced with

```
f_Q ⟵ ⌈(f_LB + f_UB)/2⌉;
Update the RHS of constraint Σ_{v∈V} y_v ≤ f_Q;
if it mod α ≠ 0 then
    actual_time ⟵ timelimit;
else
    actual_time ⟵ ⌈timelimit + 60 × it/α⌉;
end
Set the solver so as it stops actual_time seconds after the first integer solution is
found;
(T, f) ⟵ Solve the master problem;
if the master problem is infeasible then
    f_LB ⟵ f_Q + 1;
else
    it ⟵ it + 1;
    if f > f_LB then
        f_LB ⟵ f;
    end
end
```

   The solver is allocated *actual_time* seconds to find an optimal solution to the master problem after the first integer solution is found. The regular value for *actual_time* is *timelimit*, however, every α iterations, the solver is allocated more time in order to take advantage of the cuts that have been generated so far. The

right hand side (RHS) of the quality enforcement constraint (10) is also adjusted if the master problem is unfeasible, otherwise the best lower bound is updated provided that $f > f_{LB}$.

3. The actions to perform when $\mathcal{T}$ is a tree (lines 10 to 12) must be updated to

```
if f_LB = f_UB then
    T_UB ⟵ T;
    f_UB ⟵ f;
    stop ⟵ true;
else
    add_more_time ⟵ add_more_time + 1;
end
```

4. The following line should be inserted between lines 16 and 17

```
add_more_time ⟵ 0;
```

Indeed, lines 16 to 27 in the algorithm are executed only if solution $\mathcal{T}$ is not connected, so at this point there is no longer a need for giving extra time to the solver because the current solution $\mathcal{T}$ cannot be a tree, so *add_more_time* is reset to zero.

In the sequel, CPA_{MBV} refers to the implementation of the cutting plane algorithm along with the aforementioned features.

3.4 A Tabu search procedure for improving feasible solutions

The repair function is appended with a Tabu search (Glover et al. 1997) for improving the quality of upper bounds. This Tabu search procedure is described here. The idea is that even if the candidate trees returned by the master problem are not feasible, they may have desirable features and share structural similarities with an optimal solution, especially after some iterations. By searching around these promising regions of the search space, the Tabu search is expected to find good quality solutions to *MBV*.

The main challenge in designing a Tabu search procedure for *MBV* is its efficiency as it is supposed to be used on every repaired solution. If it is slow, it will impact the overall efficiency of the resulting algorithm, therefore, with the focus on efficiency, we have implemented only minimal features in our Tabu search for MBV. The Tabu search procedure uses two Tabu lists and performs search at two levels. The first level is concerned with intensification, whereas the second level is concerned with diversification. The first level begins by creating a list of branch vertices, which is divided into two parts. All branch vertices with degree up to a maximum degree, denoted by D_{max}, are sorted according to non decreasing order of their degrees and form the first part of the list. All other branch vertices are in the latter part of the list. Starting from the first vertex in the sorted part of the list, the first level iteratively tries to reduce the degree of each branch vertex in this part by deleting one-by-one all edges incident to it and inserting in the place of the deleted edge, an edge that can connect the two components and whose insertion will not result into creation of any new branch vertex. The first possible such an exchange is performed and both parts

of the list of branch vertices is updated to reflect any changes and the search starts afresh in the next iteration from the first vertex in the sorted part of the updated list. If no such exchange is possible then the degree of the branch vertex in consideration cannot be reduced and the next vertex in the sorted part is considered in the next iteration. To prevent indefinite cycling in a computationally efficient way, a counter is associated with each vertex and initialized to zero, whenever a new edge is inserted, the counters associated with its endpoints are incremented by one. If the counter for any vertex exceeds a limit denoted by lim_s, then no attempt is made to reduce the degree of the corresponding vertex, even if it belongs to the sorted part of the list of branch vertices. Therefore, all branch vertices belonging to the sorted part with counter value greater than lim_s forms a sort of Tabu list. This Tabu list is referred to as first Tabu list. If at any point during the search the number of branch vertices reduces to zero then the Tabu search procedure terminates immediately. The whole process is repeated until it fails to reduce the degree of any branch vertex in the sorted part of the list. At this point, the second level of search starts.

The second level of search tries to convert a branch vertex into a non branch vertex even at the cost of creating some new branch vertices. This level of search also starts from the first vertex in the sorted part of the list, but it can continue in the second part of the list, if it fails to convert any branch vertex in the first part into a non branch vertex, i.e., it considers all branch vertices. The second level tries to reduce the degree of the branch vertex in consideration by deleting one-by-one all edges incident to it and inserting in the place of the deleted edge, an edge that can connect the two components and whose insertion neither increases the number of branch vertices by more than 1 (note, not to forget the possibility that other endpoint of the deleted edge can be a branch vertex with original degree 3) nor converts a vertex in the second Tabu list (to be described later in this paragraph) into a branch vertex. The first such edge found is inserted. If there is no such edge then the deleted edge is inserted back. After this, the next edge incident to the branch vertex in consideration is deleted and again an alternate edge is searched. If this process fails to transform the branch vertex in consideration into a non branch vertex then the next branch vertex is considered. This is repeated until either a branch vertex has been converted into a non branch vertex or all branch vertices have been considered. In the latter case Tabu search procedure terminates. If a branch vertex has been converted into a non branch vertex, then this vertex is put into a Tabu list, which is referred to as second Tabu list. The vertices in the second Tabu list cannot be again made branch vertices for the duration determined by Tabu tenure. Now, there are two options available with respect to the newly converted non branch vertex. Either try to further reduce its degree to 1 or to leave it at degree 2. With probability p_{d1} the first option is followed, otherwise the second option is followed. After this the first Tabu list is flushed, i.e., the counter associated with each vertex is reinitialized to zero, and Tabu search procedure again switches to the search at level 1.

Whenever Tabu search improves the best solution (to be precise, the best solution from the beginning of the current run of the Tabu search, and not the overall best solution of the combined algorithm), both Tabu lists are flushed to intensify the search around this improved solution. The Tabu search procedure terminates, if the best solution fails to improve over it_{max} iterations of first level. As stated above, it can

also terminate if the number of branch vertices becomes zero or the second level fails to convert any branch vertex to a non branch vertex.

The cutting plane algorithm version that integrates Tabu search is denoted by CPA^{TS}_{MBV}.

4 Cutting plane schemes for MDS

An integer linear programming formulation of MDS can be found in Cerulli et al. (2009). The two proposed cutting plane algorithms for addressing MDS are denoted by CPA_{MDS} and CPA^{TS}_{MDS}, they are very similar to CPA_{MBV} and CPA^{TS}_{MBV} respectively. An additional set of integer variables is introduced for modeling the degree of branch vertices: for all $v \in V$, z_v is equal to the degree of vertex v if it is a branch vertex, otherwise it is set to zero.

An integer programming formulation denoted by IP_{MDS} is defined by the following equations.

$$\min \sum_{v \in V} z_v \tag{11}$$

$$\sum_{e \in I(v)} x_e \geq 1 \quad \forall v \in V$$

$$\sum_{e \in E} x_e = n - 1$$

$$y_v = 0 \quad \forall v \in V, \left|I(v)\right| \leq 2$$

$$\sum_{e \in I(v)} x_e - 2 \leq \left(\left|I(v)\right| - 2\right) y_v \quad \forall v \in V$$

$$\sum_{e \in I(v)} x_e - 2 + 2y_v \leq z_v \quad \forall v \in V \tag{12}$$

$$x_e \in \{0, 1\} \quad \forall e \in E$$

$$y_v \in \{0, 1\} \quad \forall v \in V$$

$$z_v \in \mathbb{N} \quad \forall v \in V \tag{13}$$

Equation (11) is MDS objective function, unnumbered constraints are the same in IP_{MBV} and in IP_{MDS}. Constraints (12) and (13) ensure that z_v is equal to the degree of vertex v, if v is a branch in the solution.

As for *MBV*, the master problem for *MDS* is built from IP_{MDS} by dropping connectivity constraints. The joint generation of both cuts types is implemented for speeding up convergence. The cutting plane algorithm for MDS is identical to the one proposed in Sect. 3, except for the following four points:

1. In Algorithm 1, line 1 is to be replaced with

> The master problem is initially defined by Eqs. (2)–(5), (7)–(8) and (11)–(14);

2. In Algorithms 1 and 3, f_{UB} now refers to the degree sum of the best repaired solution found so far.
3. The quality enforcement constraint is now

$$\sum_{v \in V} z_v \leq f_Q \tag{14}$$

Where f_Q is the maximum degree sum of branch vertices in the solution. It is updated at each iteration as:

$$f_Q \longleftarrow \left\lceil \frac{f_{LB} + f_{UB}}{2} \right\rceil$$

Where f_{LB} is the maximum lower bound on the degree sum of branch vertices and f_{UB} is the best feasible solution objective function found so far by the repair procedure $Repair_{MDS}$.

4. The repair procedure $Repair_{MBV}$ must be updated. A new repair procedure for MDS called $Repair_{MDS}$ is created by copying Algorithm 3 and performing the following changes

– Lines 24 and 25 must be replaced with

if $\delta(u') = 3$ **then** $dec \longleftarrow dec + 3$;
else if $\delta(u') > 3$ **then** $dec \longleftarrow dec + 1$;
if $\delta(v') = 3$ **then** $dec \longleftarrow dec + 3$;
else if $\delta(v') > 3$ **then** $dec \longleftarrow dec + 1$;

When removing an edge, the decrease in the degree sum of branch vertices due to one of its endpoint is three, if the degree of that node was originally three. In that case, the removal of the considered edge deletes a branch vertex. If the degree of that vertex was originally greater than three, then the degree sum simply decreases by one when the considered edge is removed. The contribution of the other endpoint follows the same rule.

– Line 36 must be replaced with

$(u_{add}, v_{add}) \longleftarrow (0, 0)$; $inc\_min \longleftarrow 7$;

Indeed, when adding an edge, the degree sun of branch vertices can increase up to 6, if both the endpoints of the edge to be added were originally having degree two. By initializing *inc_min* to 7, we make sure that any candidate edge for addition will lead to a lower increase.

– Lines 40 and 41 must be replaced with

if $\delta(u) = 2$ **then** $inc \longleftarrow inc + 3$;
else if $\delta(u) > 2$ **then** $inc \longleftarrow inc + 1$;
if $\delta(v) = 2$ **then** $inc \longleftarrow inc + 3$;
else if $\delta(v) > 2$ **then** $inc \longleftarrow inc + 1$;

When adding an edge, the increase in the degree sum of branch vertices due to one of its endpoint is three, if the degree of that node was originally two. In that case, the addition of the considered edge creates a new branch vertex. If the degree of that vertex was originally greater than two, then the degree sum simply increases by one when the considered edge is added. The contribution of the other endpoint follows the same rule.

– The Tabu search procedure is same as for *MBV*. It also uses two levels of search, two Tabu lists and have exactly the same termination and Tabu list flushing criteria. The second level of search is exactly same as in MBV. However, the first level has the following differences with the first level of search for *MBV* to account for the differences between the two problems. The whole list of branch vertices is sorted here. The first level search considers all the branch vertices and not only those with degree less than or equal to D_{max}. Here, starting from the first vertex in the sorted list, the first level iteratively tries to reduce the degree of each branch vertex by deleting one-by-one all edges incident to it and inserting in the place of the deleted edge, an edge whose insertion does not lead to creation of any new branch vertex and can connect the two components, and additionally have one of the following properties in decreasing order of search (priority):
  (a) An edge none of whose endpoints can be a branch vertex,
  (b) If the other endpoint of the deleted edge is also a branch vertex, then an edge whose exactly one endpoint can be a branch vertex,
  (c) If the branch vertex in consideration has degree less than or equal to D_{max}, then, an edge is searched like in *MBV*.

  The first such edge found is inserted in place of the deleted edge.

5 Computational results

5.1 Computational protocol

The original ILP model and the best heuristic proposed in Cerulli et al. (2009) are compared to the cutting plane algorithm introduced in Sect. 3.3 with and without Tabu search for both solving MBV in Sect. 5.2, and MDS in Sect. 5.3 (with the corresponding modifications mentioned in Sect. 4). All these approaches are coded in C and executed on an Intel Pentium IV Processor at 3.2 GHz with 1 GByte RAM running Microsoft Windows XP. The solver used for addressing integer programs is XPRESS-MP through its XPRESS-BCL library functions (FICO 2009).

The same three graph generators as in Cerulli et al. (2009), namely *Genmax*, *Netgen* and *Random* have been used for generating the instances used in this section. All these generators are available online from DIMACS (http://dimacs.rutgers.edu/). *Genmax* and *Netgen* are network flow problem instance generators, which return problem instances with random edges and uniform capacity. The edge capacity of the generated instances has been discarded and the resulting directed graphs have been transformed into undirected graphs. For each generator, 8 different values for $|V|$ have been considered, i.e., $\{20, 30, 40, 50, 100, 300, 500, 1000\}$. For each value

of $|V|$, the following 5 different density d have been considered $\{1.5, 2, 4, 10, 15\}$, where 'density' refers to the ratio of the number of edges to the number of vertices. Due to some internal restrictions in the generator program, *Random* failed to return any instance for $\{|V| = 20, d = 10\}$, $\{|V| = 20, d = 15\}$ and $\{|V| = 30, d = 15\}$. Thus, 40 different combinations of $|V|$ and d for each generator have been considered, except for *Random* for which only 37 combinations are available. For each combination, a set of 5 instances has been generated leading to a grand total of 585 instances. The results are presented through average computations over those 5 instance sets in order to identify trends, and to highlight the features that account for their computational difficulty. Instance names are of the form `typ_XXXX_YYYYY`, where `typ` is either `gen` for *Genmax*, `net` for *Netgen* and `ran` for *Random*. `XXXX` is the number of vertices in the graph and `YYYYY` is the number of edges.

The generation of cuts at low computational cost relies on two parameters, α and *timelimit*. The chosen values are the following: $\alpha = 5$, and *timelimit* is set to $\lceil \frac{n}{25} \rceil$ provided that the number of edges is less than $2.2n$ and $n \geq 300$, otherwise *timelimit* is set to $+\infty$ (i.e., the master problem is solved to optimality). As shown below, the most difficult instances are sparse graphs having more than 300 vertices. Thus, the amount of time dedicated to solve the master problem is limited for these difficult instances, allowing the generation of cuts at low computational cost. However, in order to reach a good balance between the number of cuts and their quality, *timelimit* is proportional to the number of vertices. Thus, the numerical value for α and the formula for *timelimit* depend on the computer and on the solver used, so as in a metaheuristic, these values have been set on an empirical basis.

As far as parameter settings for Tabu search procedure is concerned, we have used $D_{max} = 5$, $lim_s = 4$, $p_{d1} = 0.75$, $it_{max} = \max(2n, 200)$. The Tabu tenure is $\max(3N_{br}, 50)$ iterations of the first level search, where N_{br} is the number of branch vertices in the current solution. All these parameter values are set empirically. These parameter values provide good results, though these values may not be optimal for all instances.

The overall process is stopped if no optimal solution is found after one hour, and the best lower and upper bound are returned.

5.2 Computational results for MBV

Tables 3, 4 and 5 report the results for MBV, on the sets of instances generated by *Genmax*, *Netgen* and *Random* respectively. The first column of these tables is the instance name where instances are listed by increasing number of vertices and edges. The next three columns are dedicated to assessing solution quality by displaying the maximum gap in percentage between the lower and upper bound returned by ILP_{MBV} which refers to the integer linear program of Cerulli et al. (2009) (ΔILP_{MBV}), the cutting plane algorithm with the repair function (ΔCPA_{MBV}), and with the Tabu search procedure (ΔCPA^{TS}_{MBV}), respectively. Moreover, the number of proven optimal solutions found is given in parenthesis. For example, 0.00 (5) indicates that the corresponding approach has found the optimal solution for all the 5 instances of type `typ` having `XXXX` vertices and `YYYYY` edges. The column with title ΔCA assesses the improvement provided by CPA^{TS}_{MBV}, in percentage, over the Combined Approach (CA),

Table 3 MBV results on *Genmax* instances

Instance	Gap between LB and UB, in percent			ΔCA	Average CPU time in s			
	ΔILP_{MBV}	ΔCPA_{MBV}	ΔCPA^{TS}_{MBV}		ILP_{MBV}	CPA_{MBV}	CPA^{TS}_{MBV}	CA
gen_0020_00030	0.00 (5)	0.00 (5)	0.00 (5)	48.33	0.09	0.56	0.21	0.00
gen_0020_00040	0.00 (5)	0.00 (5)	0.00 (5)	73.33	0.10	0.23	0.13	0.00
gen_0020_00080	0.00 (5)	0.00 (5)	0.00 (5)	80.00	0.10	0.20	0.12	0.00
gen_0020_00200	0.00 (5)	0.00 (5)	0.00 (5)	0.00	0.13	0.17	0.14	0.00
gen_0020_00300	0.00 (5)	0.00 (5)	0.00 (5)	0.00	0.27	0.18	0.14	0.00
gen_0030_00045	0.00 (5)	0.00 (5)	0.00 (5)	51.76	0.18	0.23	0.18	0.00
gen_0030_00060	0.00 (5)	0.00 (5)	0.00 (5)	84.33	0.40	0.20	0.13	0.00
gen_0030_00120	0.00 (5)	0.00 (5)	0.00 (5)	89.33	0.41	0.19	0.13	0.00
gen_0030_00300	0.00 (5)	0.00 (5)	0.00 (5)	40.00	0.83	0.15	0.14	0.00
gen_0030_00450	0.00 (5)	0.00 (5)	0.00 (5)	0.00	0.51	0.16	0.14	0.00
gen_0040_00060	0.00 (5)	0.00 (5)	0.00 (5)	61.71	0.31	0.28	0.18	0.00
gen_0040_00080	0.00 (5)	0.00 (5)	0.00 (5)	87.67	0.47	0.17	0.13	0.00
gen_0040_00160	0.00 (5)	0.00 (5)	0.00 (5)	100.00	0.59	0.15	0.14	0.00
gen_0040_00400	0.00 (5)	0.00 (5)	0.00 (5)	60.00	0.88	0.16	0.14	0.00
gen_0040_00600	0.00 (5)	0.00 (5)	0.00 (5)	60.00	1.60	0.17	0.14	0.00
gen_0050_00075	0.00 (5)	0.00 (5)	0.00 (5)	58.92	0.70	0.51	0.34	0.00
gen_0050_00100	0.00 (5)	0.00 (5)	0.00 (5)	86.31	0.71	0.31	0.14	0.00
gen_0050_00200	0.00 (5)	0.00 (5)	0.00 (5)	100.00	0.83	0.16	0.14	0.00
gen_0050_00500	0.00 (5)	0.00 (5)	0.00 (5)	100.00	1.51	0.23	0.15	0.00
gen_0050_00750	0.00 (5)	0.00 (5)	0.00 (5)	40.00	1.68	0.17	0.15	0.00
gen_0100_00150	0.00 (5)	0.00 (5)	0.00 (5)	61.70	12.11	5.54	2.53	0.00
gen_0100_00200	0.00 (5)	0.00 (5)	0.00 (5)	84.39	11.22	6.32	0.45	0.00
gen_0100_00400	0.00 (5)	0.00 (5)	0.00 (5)	100.00	4.78	0.21	0.15	0.00
gen_0100_01000	0.00 (5)	0.00 (5)	0.00 (5)	100.00	14.42	0.17	0.18	0.00
gen_0100_01500	0.00 (5)	0.00 (5)	0.00 (5)	100.00	13.66	0.20	0.18	0.01
gen_0300_00450	32.84 (0)	1.90 (4)	1.00 (4)	59.35	–	1720.69	1023.95	0.00
gen_0300_00600	63.03 (0)	2.86 (4)	0.00 (5)	85.01	–	1783.08	215.92	0.01
gen_0300_01200	0.00 (5)	0.00 (5)	0.00 (5)	100.00	92.09	12.82	1.06	0.03
gen_0300_03000	0.00 (5)	0.00 (5)	0.00 (5)	100.00	554.34	0.40	0.37	0.11
gen_0300_04500	0.00 (5)	0.00 (5)	0.00 (5)	100.00	1345.46	0.87	0.48	0.13
gen_0500_00750	42.39 (0)	8.05 (0)	5.19 (0)	60.07	–	–	–	0.02
gen_0500_01000	71.86 (0)	20.14 (0)	8.94 (1)	85.30	–	–	1810.50	0.06
gen_0500_02000	10.00 (4)	0.00 (5)	0.00 (5)	99.20	548.22	7.42	2.06	0.08
gen_0500_05000	40.00 (3)	0.00 (5)	0.00 (5)	100.00	1809.49	2.12	1.37	0.24
gen_0500_07500	60.00 (2)	0.00 (5)	0.00 (5)	100.00	1375.68	1.12	0.87	0.37
gen_1000_01500	68.94 (0)	18.21 (0)	14.69 (0)	55.24	–	–	–	0.13
gen_1000_02000	80.98 (0)	31.32 (0)	10.56 (0)	83.50	–	–	–	0.18
gen_1000_04000	76.98 (1)	0.00 (5)	0.00 (5)	99.01	2558.30	274.66	4.84	0.40
gen_1000_10000	100.00 (0)	0.00 (5)	0.00 (5)	100.00	–	13.41	2.88	1.02
gen_1000_15000	100.00 (0)	0.00 (5)	0.00 (5)	100.00	–	14.84	5.95	1.66
Proved opt. sol.	33.33 %	70.67 %	73.33 %		915.62	295.35	128.82	0.30

Table 4 MBV results on *Netgen* instances

Instance	Gap between LB and UB, in percent			ΔCA	Average CPU time in s			
	ΔILP_{MBV}	ΔCPA_{MBV}	ΔCPA^{TS}_{MBV}		ILP_{MBV}	CPA_{MBV}	CPA^{TS}_{MBV}	CA
net_0020_00030	0.00 (5)	0.00 (5)	0.00 (5)	100.00	0.05	0.12	0.12	0.00
net_0020_00040	0.00 (5)	0.00 (5)	0.00 (5)	80.00	0.11	0.12	0.12	0.00
net_0020_00080	0.00 (5)	0.00 (5)	0.00 (5)	20.00	0.16	0.11	0.12	0.00
net_0020_00200	0.00 (5)	0.00 (5)	0.00 (5)	20.00	0.20	0.12	0.12	0.00
net_0020_00300	0.00 (5)	0.00 (5)	0.00 (5)	20.00	0.32	0.12	0.12	0.00
net_0030_00045	0.00 (5)	0.00 (5)	0.00 (5)	100.00	0.09	0.12	0.12	0.00
net_0030_00060	0.00 (5)	0.00 (5)	0.00 (5)	100.00	0.17	0.13	0.12	0.00
net_0030_00120	0.00 (5)	0.00 (5)	0.00 (5)	100.00	0.42	0.13	0.12	0.00
net_0030_00300	0.00 (5)	0.00 (5)	0.00 (5)	80.00	0.67	0.13	0.13	0.00
net_0030_00450	0.00 (5)	0.00 (5)	0.00 (5)	60.00	0.70	0.14	0.13	0.00
net_0040_00060	0.00 (5)	0.00 (5)	0.00 (5)	100.00	0.20	0.14	0.12	0.00
net_0040_00080	0.00 (5)	0.00 (5)	0.00 (5)	100.00	0.40	0.12	0.12	0.00
net_0040_00160	0.00 (5)	0.00 (5)	0.00 (5)	100.00	0.68	0.12	0.12	0.00
net_0040_00400	0.00 (5)	0.00 (5)	0.00 (5)	80.00	1.03	0.13	0.13	0.00
net_0040_00600	0.00 (5)	0.00 (5)	0.00 (5)	100.00	1.22	0.14	0.13	0.00
net_0050_00075	0.00 (5)	0.00 (5)	0.00 (5)	100.00	0.19	0.13	0.12	0.00
net_0050_00100	0.00 (5)	0.00 (5)	0.00 (5)	100.00	0.56	0.13	0.12	0.00
net_0050_00200	0.00 (5)	0.00 (5)	0.00 (5)	100.00	1.09	0.12	0.13	0.00
net_0050_00500	0.00 (5)	0.00 (5)	0.00 (5)	100.00	1.47	0.13	0.15	0.00
net_0050_00750	0.00 (5)	0.00 (5)	0.00 (5)	100.00	1.24	0.15	0.15	0.00
net_0100_00150	0.00 (5)	0.00 (5)	0.00 (5)	100.00	1.09	0.18	0.17	0.00
net_0100_00200	0.00 (5)	0.00 (5)	0.00 (5)	100.00	2.75	0.18	0.13	0.00
net_0100_00400	0.00 (5)	0.00 (5)	0.00 (5)	100.00	4.19	0.15	0.15	0.00
net_0100_01000	0.00 (5)	0.00 (5)	0.00 (5)	100.00	8.95	0.20	0.19	0.00
net_0100_01500	0.00 (5)	0.00 (5)	0.00 (5)	100.00	11.53	0.18	0.18	0.00
net_0300_00450	0.00 (5)	0.00 (5)	0.00 (5)	100.00	8.24	0.38	0.25	0.00
net_0300_00600	0.00 (5)	0.00 (5)	0.00 (5)	100.00	21.96	0.58	0.27	0.00
net_0300_01200	0.00 (5)	0.00 (5)	0.00 (5)	100.00	104.53	1.56	0.73	0.03
net_0300_03000	0.00 (5)	0.00 (5)	0.00 (5)	100.00	504.14	0.33	0.33	0.10
net_0300_04500	20.00 (4)	0.00 (5)	0.00 (5)	100.00	1188.95	0.61	0.47	0.14
net_0500_00750	0.00 (5)	0.00 (5)	0.00 (5)	100.00	22.10	0.78	0.58	0.03
net_0500_01000	0.00 (5)	0.00 (5)	0.00 (5)	100.00	100.59	3.83	0.57	0.06
net_0500_02000	0.00 (5)	0.00 (5)	0.00 (5)	100.00	646.94	3.23	1.85	0.11
net_0500_05000	20.00 (4)	0.00 (5)	0.00 (5)	100.00	1721.58	4.43	1.00	0.23
net_0500_07500	60.00 (2)	0.00 (5)	0.00 (5)	100.00	3001.29	2.64	1.30	0.38
net_1000_01500	0.00 (5)	0.00 (5)	0.00 (5)	100.00	121.11	6.62	8.11	0.09
net_1000_02000	0.00 (5)	0.00 (5)	0.00 (5)	100.00	934.20	12.53	4.92	0.17
net_1000_04000	80.00 (1)	0.00 (5)	0.00 (5)	100.00	2385.70	8.53	4.86	0.38
net_1000_10000	100.00 (0)	0.00 (5)	0.00 (5)	100.00	–	13.76	3.03	1.06
net_1000_15000	100.00 (0)	0.00 (5)	0.00 (5)	100.00	–	7.80	5.47	1.56
Proved opt. sol.	74.67 %	100.00 %	100.00 %		577.67	4.51	2.25	0.29

Table 5 MBV results on *Random* instances

Instance	Gap between LB and UB, in percent			ΔCA	Average CPU time in s			
	ΔILP_{MBV}	ΔCPA_{MBV}	ΔCPA^{TS}_{MBV}		ILP_{MBV}	CPA_{MBV}	CPA^{TS}_{MBV}	CA
ran_0020_00030	0.00 (5)	0.00 (5)	0.00 (5)	56.67	0.12	0.15	0.13	0.00
ran_0020_00040	0.00 (5)	0.00 (5)	0.00 (5)	100.00	0.13	0.11	0.11	0.00
ran_0020_00080	0.00 (5)	0.00 (5)	0.00 (5)	60.00	0.16	0.11	0.11	0.00
ran_0030_00045	0.00 (5)	0.00 (5)	0.00 (5)	47.00	0.33	0.26	0.27	0.00
ran_0030_00060	0.00 (5)	0.00 (5)	0.00 (5)	95.00	0.13	0.12	0.11	0.00
ran_0030_00120	0.00 (5)	0.00 (5)	0.00 (5)	100.00	0.48	0.12	0.12	0.00
ran_0030_00300	0.00 (5)	0.00 (5)	0.00 (5)	20.00	1.05	0.13	0.13	0.00
ran_0040_00060	0.00 (5)	0.00 (5)	0.00 (5)	47.56	0.45	0.28	0.28	0.00
ran_0040_00080	0.00 (5)	0.00 (5)	0.00 (5)	69.67	0.76	0.43	0.17	0.00
ran_0040_00160	0.00 (5)	0.00 (5)	0.00 (5)	100.00	0.70	0.12	0.13	0.00
ran_0040_00400	0.00 (5)	0.00 (5)	0.00 (5)	40.00	0.99	0.13	0.13	0.00
ran_0040_00600	0.00 (5)	0.00 (5)	0.00 (5)	0.00	2.58	0.15	0.15	0.00
ran_0050_00075	0.00 (5)	0.00 (5)	0.00 (5)	60.16	0.60	0.39	0.24	0.00
ran_0050_00100	0.00 (5)	0.00 (5)	0.00 (5)	78.00	1.26	0.28	0.12	0.00
ran_0050_00200	0.00 (5)	0.00 (5)	0.00 (5)	100.00	0.90	0.12	0.12	0.00
ran_0050_00500	0.00 (5)	0.00 (5)	0.00 (5)	80.00	2.27	0.14	0.14	0.00
ran_0050_00750	0.00 (5)	0.00 (5)	0.00 (5)	40.00	3.11	0.14	0.14	0.00
ran_0100_00150	0.00 (5)	0.00 (5)	0.00 (5)	59.15	11.41	8.10	1.94	0.00
ran_0100_00200	0.00 (5)	0.00 (5)	0.00 (5)	81.52	69.32	5.91	2.01	0.00
ran_0100_00400	0.00 (5)	0.00 (5)	0.00 (5)	100.00	7.03	0.17	0.14	0.00
ran_0100_01000	0.00 (5)	0.00 (5)	0.00 (5)	100.00	13.60	0.17	0.16	0.00
ran_0100_01500	0.00 (5)	0.00 (5)	0.00 (5)	80.00	25.65	0.17	0.17	0.01
ran_0300_00450	27.39 (0)	1.18 (4)	2.05 (3)	60.84	–	1465.08	435.72	0.00
ran_0300_00600	45.93 (1)	2.86 (4)	0.00 (5)	85.36	2544.13	1269.44	79.70	0.01
ran_0300_01200	0.00 (5)	0.00 (5)	0.00 (5)	100.00	99.72	1.96	1.02	0.03
ran_0300_03000	0.00 (5)	0.00 (5)	0.00 (5)	100.00	1104.37	0.46	0.37	0.08
ran_0300_04500	0.00 (5)	0.00 (5)	0.00 (5)	100.00	1307.45	0.53	0.46	0.14
ran_0500_00750	44.08 (0)	10.16 (0)	6.15 (0)	59.63	–	–	–	0.02
ran_0500_01000	71.13 (0)	20.94 (0)	10.48 (0)	83.37	–	–	–	0.03
ran_0500_02000	10.00 (4)	0.00 (5)	0.00 (5)	99.31	1032.66	22.94	1.35	0.11
ran_0500_05000	60.00 (2)	0.00 (5)	0.00 (5)	100.00	2613.19	1.23	0.95	0.27
ran_0500_07500	80.00 (1)	0.00 (5)	0.00 (5)	100.00	12.41	1.87	1.89	0.41
ran_1000_01500	66.92 (0)	18.12 (0)	15.36 (0)	53.52	–	–	–	0.11
ran_1000_02000	83.53 (0)	33.12 (0)	11.48 (0)	84.71	–	–	–	0.17
ran_1000_04000	59.70 (2)	0.00 (5)	0.00 (5)	99.64	3184.16	25.34	4.08	0.37
ran_1000_10000	100.00 (0)	0.00 (5)	0.00 (5)	100.00	–	9.82	3.50	1.24
ran_1000_15000	100.00 (0)	0.00 (5)	0.00 (5)	100.00	–	11.16	3.20	2.09
Proved opt. sol.	33.33 %	70.67 %	70.67 %		1233.58	213.48	33.77	0.34

which is shown in Cerulli et al. (2009) to be the best heuristic for MBV. The improvement reaches 100 % when the Combined Approach returns a nonzero number of branch vertices, whereas a Hamiltonian path is found by our approach. The next three columns provide the average CPU time in seconds, over the instances that have been solved to optimality. If none of the five instances has been solved to optimality within one hour, '–' is displayed in the table. The last column is the average CPU time for the Combined Approach over the five instances.

The last line of the tables aims at summarizing the results on the instances with more than 300 vertices as follows: columns 2 to 4 display how many instances are solved to optimality, in percentage. Column 6 to 8 provide the average CPU time required for solving an instance to optimality (the calculation ignores the instances for which no optimal solution is found), and the last column is the average CPU time required by *CA*.

It can be seen from Tables 3, 4 and 5 that all exact approaches are able to find the optimal solution for $|V| \leq 100$. That is the reason why only instances with at least 300 vertices are taken into account for providing global comparisons at the last line of the tables. However, it can be observed that the cutting plane based approaches can be up to 100 times faster that ILP_{MBV} on these small instances, regardless of the their type. The Tabu search does not provide a clear advantage for these easy instances, though.

For the instances with at least 300 vertices, both cutting plane-based approaches have a significant advantage over ILP_{MBV}, especially for high density instances. This is because a Hamiltonian path is very likely to exist in that case, and when such a path is found by the repair procedure (or by the Tabu search using a repaired solution as input), it is immediately identified as optimal. ILP_{MBV} is unable to take advantage of high density, and has a running time which increases drastically with the problem size. As can be seen with largest instances and $d \geq 10$, ILP_{MBV} is not able to return even a feasible solution in one hour, while CPA^{TS}_{MBV} finds an optimal solution (a Hamiltonian path) in less than 6 seconds. The Tabu search accelerates convergence as the CPU time is cut by up to 80 % compared to CPA_{MBV} for `gen_1000_10000`. The results on the instances generated by *Netgen* clearly show the benefits of cutting-plane algorithm-based approaches over ILP_{MBV}, as all these instances are solved to optimality. Furthermore, the Combined Approach, while requiring very modest CPU time, returns poor quality solutions, even for dense instances, where a Hamiltonian path exists. This shows that the Combined Approach cannot be used for proving that a Hamiltonian path exists.

Thus, the most difficult instances for MBV are those with a large number of vertices, and low density, especially the ones generated by *Random*. Even when all exact approaches fail to return an optimal solution in one hour, cutting plane-based approaches significantly outperform ILP_{MBV} in terms of solution quality, and CPA^{TS}_{MBV} emerges as the most efficient approach, as the maximum gap to optimality of the best solution found can be up to 3 times less when Tabu search is used. Whereas the number of solutions solved to optimality is the very close (for *Genmax*) or the same (for *Random*) with and without Tabu search, the gap to optimality is around 2 times smaller then using Tabu search.

The comparison in terms of computational time is also in favor of CPA^{TS}_{MBV}, that is up to 6 times faster than CPA_{MBV} for returning the same number of optimal solutions

on *Random* instances. The average CPU time of ILP^{TS}_{MBV} for computing an optimal solution is even 36 times less than ILP_{MBV}, allowing CPA^{TS}_{MBV} to solve nearly twice more instances than ILP_{MBV} in one hour.

5.3 Computational results for MDS

Tables 6, 7 and 8 report the results for MDS. The conclusions are quite similar to those drawn for MBV, with the following differences.

The best heuristic approach in Cerulli et al. (2009) is the Edge Weight algorithm, which is denoted by *EW* in the tables. The number of instances having at least 300 vertices that can be solved to optimality is larger with MDS, which suggests that MDS is easier than MBV on *Genmax* and *Netgen* instances. Moreover, the advantage provided by Tabu search, while being still noticeable, is a bit less significant for MDS. However, the last line of Table 8 regarding CPA_{MDS} and CPA^{TS}_{MDS} must be analyzed carefully. Indeed, it appears that the average CPU time required for solving *Random* instances to optimality is less with CPA_{MDS} than with CPA^{TS}_{MDS}, but this is due to the fact that CPA^{TS}_{MDS} solves more instances than CPA_{MDS}, and those additional instances require a long amount of time to solve. This happens in particular with `ran_0500_01000`.

Finally, as for MBV, the proposed cutting-plane approaches provide significant advantages over the ILP and heuristics proposed in Cerulli et al. (2009).

6 Conclusions

This paper addresses two closely related (but independent) problems that arise when minimizing the cost of devices to add to an existing fiber optic network that uses wavelength-division for implementing multicast communication. The proposed cutting plane approaches appear to be very efficient in terms of solution quality and computation time. There are two reasons for this. First, allocating a limited amount of time to the ILP solver for addressing the master problem prevents the search from stalling and allows for generating cuts at low computational cost that are useful for the upcoming iterations. This contributes in improving the lower bound. Second, the repair function takes advantage of the candidate trees returned by the solver for finding good solutions. The quality of these solutions is improved further by a Tabu search procedure. This contributes in improving the upper bound. Finally, these two processes collaborate together as follows: a performance inequality based on the upper bound is introduced in the master problem formulation, and the repair function and the Tabu search directly benefit from the results of the cutting plane algorithm. This mutually beneficial collaboration of an exact approach and a metaheuristic highlights the potential of *matheuristic* for addressing combinatorial optimization problems, and goes beyond the "metaheuristic plus lower bound" scheme. To this respect, the comparison with the results in Cerulli et al. (2009), where ILP and heuristics are proposed independently, is clearly in favor of the hybrid approach proposed in the present paper. Such a hybrid approach is expected to perform well for addressing other difficult variations of the minimum spanning tree problem, like the dominating tree problem, or other degree constrained spanning tree problems.

Table 6 MDS results on *Genmax* instances

Instance	Gap between LB and UB, in percent			ΔEW	Average CPU time in s			
	ΔILP_{MDS}	ΔCPA_{MDS}	ΔCPA^{TS}_{MDS}		ILP_{MDS}	CPA_{MDS}	CPA^{TS}_{MDS}	EW
gen_0020_00030	0.00 (5)	0.00 (5)	0.00 (5)	44.89	0.06	0.14	0.13	0.00
gen_0020_00040	0.00 (5)	0.00 (5)	0.00 (5)	61.11	0.20	0.16	0.15	0.00
gen_0020_00080	0.00 (5)	0.00 (5)	0.00 (5)	80.00	0.16	0.13	0.12	0.00
gen_0020_00200	0.00 (5)	0.00 (5)	0.00 (5)	100.00	0.21	0.14	0.13	0.00
gen_0020_00300	0.00 (5)	0.00 (5)	0.00 (5)	100.00	0.32	0.13	0.12	0.00
gen_0030_00045	0.00 (5)	0.00 (5)	0.00 (5)	42.37	0.27	0.23	0.22	0.00
gen_0030_00060	0.00 (5)	0.00 (5)	0.00 (5)	73.33	0.35	0.36	0.23	0.00
gen_0030_00120	0.00 (5)	0.00 (5)	0.00 (5)	80.00	0.23	0.37	0.23	0.00
gen_0030_00300	0.00 (5)	0.00 (5)	0.00 (5)	100.00	0.64	0.14	0.13	0.00
gen_0030_00450	0.00 (5)	0.00 (5)	0.00 (5)	100.00	0.65	0.14	0.14	0.00
gen_0040_00060	0.00 (5)	0.00 (5)	0.00 (5)	55.71	0.25	0.47	0.28	0.00
gen_0040_00080	0.00 (5)	0.00 (5)	0.00 (5)	81.67	25.73	0.17	0.13	0.00
gen_0040_00160	0.00 (5)	0.00 (5)	0.00 (5)	100.00	0.56	0.15	0.14	0.00
gen_0040_00400	0.00 (5)	0.00 (5)	0.00 (5)	100.00	1.05	0.17	0.16	0.00
gen_0040_00600	0.00 (5)	0.00 (5)	0.00 (5)	100.00	1.68	0.16	0.15	0.00
gen_0050_00075	0.00 (5)	0.00 (5)	0.00 (5)	43.68	0.87	0.96	0.67	0.00
gen_0050_00100	0.00 (5)	0.00 (5)	0.00 (5)	74.44	19.17	0.37	0.16	0.00
gen_0050_00200	0.00 (5)	0.00 (5)	0.00 (5)	100.00	0.88	0.14	0.14	0.00
gen_0050_00500	0.00 (5)	0.00 (5)	0.00 (5)	100.00	1.77	0.17	0.16	0.00
gen_0050_00750	0.00 (5)	0.00 (5)	0.00 (5)	100.00	1.79	0.17	0.17	0.00
gen_0100_00150	0.00 (5)	0.00 (5)	0.00 (5)	47.63	27.25	3.82	3.38	0.00
gen_010_00200	1.33 (4)	0.00 (5)	0.00 (5)	63.08	12.34	2.25	0.52	0.00
gen_0100_00400	0.00 (5)	0.00 (5)	0.00 (5)	100.00	3.33	0.21	0.17	0.00
gen_0100_01000	0.00 (5)	0.00 (5)	0.00 (5)	100.00	9.05	0.22	0.22	0.01
gen_0100_01500	0.00 (5)	0.00 (5)	0.00 (5)	100.00	6.60	0.19	0.18	0.03
gen_0300_00450	16.03 (0)	0.00 (5)	0.00 (5)	44.81	–	1653.98	1242.78	0.01
gen_0300_00600	34.73 (0)	3.70 (3)	0.93 (4)	64.50	–	470.74	307.90	0.01
gen_0300_01200	0.00 (5)	0.00 (5)	0.00 (5)	100.00	23.91	5.32	0.69	0.05
gen_0300_03000	0.00 (5)	0.00 (5)	0.00 (5)	100.00	110.05	1.22	0.72	0.15
gen_0300_04500	0.00 (5)	0.00 (5)	0.00 (5)	100.00	276.30	2.25	2.07	0.21
gen_0500_00750	98.46 (0)	1.69 (0)	1.16 (0)	45.55	–	–	–	0.03
gen_0500_01000	81.05 (0)	8.15 (0)	2.20 (1)	63.77	–	–	2144.59	0.06
gen_0500_02000	4.00 (4)	0.00 (5)	0.00 (5)	98.92	228.84	41.89	3.33	0.14
gen_0500_05000	40.00 (3)	0.00 (5)	0.00 (5)	100.00	194.66	1.38	1.35	0.39
gen_0500_07500	0.00 (5)	0.00 (5)	0.00 (5)	100.00	1198.72	3.86	2.82	0.65
gen_1000_01500	96.79 (0)	5.47 (0)	4.71 (0)	43.76	–	–	–	0.15
gen_1000_02000	98.72 (0)	10.06 (0)	5.23 (0)	61.80	–	–	–	0.22
gen_1000_04000	89.16 (0)	0.00 (5)	0.00 (5)	98.54	–	336.88	70.46	0.49
gen_1000_10000	40.00 (3)	0.00 (5)	0.00 (5)	100.00	952.77	19.27	4.43	1.78
gen_1000_15000	100.00 (0)	0.00 (5)	0.00 (5)	100.00	–	15.50	5.59	2.92
Proved opt. sol.	40.00 %	70.67 %	73.33 %		413.42	223.02	182.68	0.49

Table 7 MDS results on *Netgen* instances

Instance	Gap between LB and UB, in percent			ΔEW	Average CPU time in s			
	ΔILP_{MDS}	ΔCPA_{MDS}	ΔCPA^{TS}_{MDS}		ILP_{MDS}	CPA_{MDS}	CPA^{TS}_{MDS}	EW
net_0020_00030	0.00 (5)	0.00 (5)	0.00 (5)	60.00	0.02	0.13	0.12	0.00
net_0020_00040	0.00 (5)	0.00 (5)	0.00 (5)	80.00	0.09	0.12	0.13	0.00
net_0020_00080	0.00 (5)	0.00 (5)	0.00 (5)	60.00	0.06	0.12	0.12	0.00
net_0020_00200	0.00 (5)	0.00 (5)	0.00 (5)	80.00	0.29	0.12	0.12	0.00
net_0020_00300	0.00 (5)	0.00 (5)	0.00 (5)	100.00	0.25	0.13	0.13	0.00
net_0030_00045	0.00 (5)	0.00 (5)	0.00 (5)	100.00	0.10	0.13	0.12	0.00
net_0030_00060	0.00 (5)	0.00 (5)	0.00 (5)	100.00	0.20	0.12	0.12	0.00
net_0030_00120	0.00 (5)	0.00 (5)	0.00 (5)	80.00	0.28	0.13	0.13	0.00
net_0030_00300	0.00 (5)	0.00 (5)	0.00 (5)	100.00	0.75	0.14	0.14	0.00
net_0030_00450	0.00 (5)	0.00 (5)	0.00 (5)	100.00	0.39	0.14	0.15	0.00
net_0040_00060	0.00 (5)	0.00 (5)	0.00 (5)	100.00	0.12	0.13	0.13	0.00
net_0040_00080	0.00 (5)	0.00 (5)	0.00 (5)	100.00	0.42	0.13	0.11	0.00
net_0040_00160	0.00 (5)	0.00 (5)	0.00 (5)	100.00	0.44	0.13	0.12	0.00
net_0040_00400	0.00 (5)	0.00 (5)	0.00 (5)	100.00	1.07	0.15	0.15	0.00
net_0040_00600	0.00 (5)	0.00 (5)	0.00 (5)	100.00	1.78	0.15	0.15	0.00
net_0050_00075	0.00 (5)	0.00 (5)	0.00 (5)	100.00	0.15	0.13	0.12	0.00
net_0050_00100	0.00 (5)	0.00 (5)	0.00 (5)	100.00	0.46	0.12	0.12	0.00
net_0050_00200	0.00 (5)	0.00 (5)	0.00 (5)	100.00	0.89	0.15	0.14	0.00
net_0050_00500	0.00 (5)	0.00 (5)	0.00 (5)	100.00	2.22	0.16	0.16	0.00
net_0050_00750	0.00 (5)	0.00 (5)	0.00 (5)	100.00	1.13	0.16	0.15	0.00
net_0100_00150	0.00 (5)	0.00 (5)	0.00 (5)	100.00	0.85	0.16	0.13	0.00
net_0100_00200	0.00 (5)	0.00 (5)	0.00 (5)	100.00	1.79	0.21	0.13	0.00
net_0100_00400	0.00 (5)	0.00 (5)	0.00 (5)	100.00	3.37	0.22	0.16	0.00
net_0100_01000	0.00 (5)	0.00 (5)	0.00 (5)	100.00	9.00	0.17	0.17	0.00
net_0100_01500	0.00 (5)	0.00 (5)	0.00 (5)	100.00	22.35	0.22	0.21	0.01
net_0300_00450	0.00 (5)	0.00 (5)	0.00 (5)	100.00	9.35	0.81	0.40	0.00
net_0300_00600	0.00 (5)	0.00 (5)	0.00 (5)	100.00	39.55	0.57	0.25	0.00
net_0300_01200	0.00 (5)	0.00 (5)	0.00 (5)	100.00	73.84	1.79	1.09	0.03
net_0300_03000	0.00 (5)	0.00 (5)	0.00 (5)	100.00	47.79	1.16	1.12	0.15
net_0300_04500	0.00 (5)	0.00 (5)	0.00 (5)	100.00	187.73	0.89	0.89	0.20
net_0500_00750	0.00 (5)	0.00 (5)	0.00 (5)	100.00	18.55	2.08	2.13	0.03
net_0500_01000	0.00 (5)	0.00 (5)	0.00 (5)	100.00	252.68	3.85	0.82	0.06
net_0500_02000	0.00 (5)	0.00 (5)	0.00 (5)	100.00	210.41	7.67	2.50	0.14
net_0500_05000	40.00 (3)	0.00 (5)	0.00 (5)	100.00	154.18	4.83	4.01	0.39
net_0500_07500	20.00 (4)	0.00 (5)	0.00 (5)	100.00	196.77	3.50	2.48	0.56
net_1000_01500	0.00 (5)	0.00 (5)	0.00 (5)	100.00	233.06	11.76	12.42	0.14
net_1000_02000	0.00 (5)	0.00 (5)	0.00 (5)	100.00	2021.92	17.52	7.81	0.22
net_1000_04000	60.00 (2)	0.00 (5)	0.00 (5)	100.00	1478.23	19.30	3.76	0.52
net_1000_10000	0.00 (5)	0.00 (5)	0.00 (5)	100.00	905.48	34.36	8.16	1.72
net_1000_15000	80.00 (1)	0.00 (5)	0.00 (5)	100.00	974.10	14.38	10.74	2.82
Proved opt. sol.	86.67 %	100.00 %	100.00 %		387.42	8.30	3.91	0.47

Table 8 MDS results on *Random* instances

Instance	Gap between LB and UB, in percent			ΔEW	Average CPU time in s			
	ΔILP_{MDS}	ΔCPA_{MDS}	ΔCPA^{TS}_{MDS}		ILP_{MDS}	CPA_{MDS}	CPA^{TS}_{MDS}	EW
ran_0020_00030	0.00 (5)	0.00 (5)	0.00 (5)	50.00	0.10	0.15	0.13	0.00
ran_0020_00040	0.00 (5)	0.00 (5)	0.00 (5)	60.00	0.07	0.13	0.12	0.00
ran_0020_00080	0.00 (5)	0.00 (5)	0.00 (5)	60.00	0.06	0.12	0.12	0.00
ran_0030_00045	0.00 (5)	0.00 (5)	0.00 (5)	36.31	0.32	0.32	0.30	0.00
ran_0030_00060	0.00 (5)	0.00 (5)	0.00 (5)	95.00	0.54	0.12	0.12	0.00
ran_0030_00120	0.00 (5)	0.00 (5)	0.00 (5)	100.00	0.22	0.12	0.12	0.00
ran_0030_00300	0.00 (5)	0.00 (5)	0.00 (5)	100.00	0.93	0.13	0.13	0.00
ran_0040_00060	0.00 (5)	0.00 (5)	0.00 (5)	45.53	0.68	0.33	0.34	0.00
ran_0040_00080	0.00 (5)	0.00 (5)	0.00 (5)	66.83	36.44	0.31	0.32	0.00
ran_0040_00160	0.00 (5)	0.00 (5)	0.00 (5)	100.00	0.60	0.13	0.13	0.00
ran_0040_00400	0.00 (5)	0.00 (5)	0.00 (5)	100.00	1.00	0.13	0.14	0.00
ran_0040_00600	0.00 (5)	0.00 (5)	0.00 (5)	100.00	2.73	0.15	0.15	0.00
ran_0050_00075	0.00 (5)	0.00 (5)	0.00 (5)	48.09	1.31	0.41	0.33	0.00
ran_0050_00100	0.00 (5)	0.00 (5)	0.00 (5)	76.52	2.13	0.34	0.21	0.00
ran_0050_00200	0.00 (5)	0.00 (5)	0.00 (5)	100.00	0.60	0.14	0.13	0.00
ran_0050_00500	0.00 (5)	0.00 (5)	0.00 (5)	100.00	2.05	0.15	0.14	0.00
ran_0050_00750	0.00 (5)	0.00 (5)	0.00 (5)	100.00	2.38	0.16	0.16	0.00
ran_0100_00150	0.00 (5)	0.00 (5)	0.00 (5)	47.86	110.73	3.82	2.49	0.00
ran_0100_00200	2.00 (4)	0.00 (5)	0.00 (5)	68.94	115.86	5.10	4.63	0.00
ran_0100_00400	0.00 (5)	0.00 (5)	0.00 (5)	100.00	3.67	0.20	0.17	0.00
ran_0100_01000	0.00 (5)	0.00 (5)	0.00 (5)	100.00	7.76	0.22	0.22	0.01
ran_0100_01500	0.00 (5)	0.00 (5)	0.00 (5)	100.00	13.12	1.15	1.15	0.03
ran_0300_00450	14.69 (0)	0.00 (5)	0.00 (5)	45.79	–	653.22	294.31	0.01
ran_0300_00600	16.50 (0)	0.59 (4)	0.00 (5)	63.83	–	456.20	789.24	0.02
ran_0300_01200	0.00 (5)	0.00 (5)	0.00 (5)	100.00	37.83	4.96	1.36	0.06
ran_0300_03000	0.00 (5)	0.00 (5)	0.00 (5)	100.00	50.23	3.92	1.50	0.15
ran_0300_04500	0.00 (5)	0.00 (5)	0.00 (5)	100.00	471.89	0.99	0.47	0.23
ran_0500_00750	98.48 (0)	1.81 (1)	1.78 (1)	45.79	–	1484.69	1500.73	0.03
ran_0500_01000	88.56 (0)	8.27 (0)	2.64 (1)	62.33	–	–	1311.19	0.04
ran_0500_02000	6.67 (4)	0.00 (5)	0.00 (5)	99.13	406.83	19.53	1.52	0.14
ran_0500_05000	40.00 (3)	0.00 (5)	0.00 (5)	100.00	618.27	6.63	2.83	0.40
ran_0500_07500	20.00 (4)	0.00 (5)	0.00 (5)	100.00	545.82	2.55	2.54	0.62
ran_1000_01500	96.90 (0)	5.80 (0)	5.20 (0)	43.07	–	–	–	0.15
ran_1000_02000	98.81 (0)	14.38 (0)	6.27 (0)	62.00	–	–	–	0.22
ran_1000_04000	100.00 (0)	0.00 (5)	0.00 (5)	99.68	–	108.74	6.43	0.49
ran_1000_10000	60.00 (2)	0.00 (5)	0.00 (5)	100.00	910.21	12.49	6.40	1.80
ran_1000_15000	40.00 (3)	0.00 (5)	0.00 (5)	100.00	1266.90	4.76	3.83	3.19
Proved opt. sol.	41.33 %	73.33 %	76.00 %		454.39	134.52	146.74	0.51

References

Arora A, Subramaniam S (2002) Wavelength conversion placement in WDM mesh optical networks. Photonic Netw Commun 4(2):167–177

Cerulli R, Gentili M, Iossa A (2009) Bounded-degree spanning tree problems: models and new algorithms. Comput Optim Appl 42:353–370

Diaz I (2006) A branch-and-cut algorithm for graph coloring. Discrete Appl Math 154(5):826–847

Dirac G (1952) Some theorems on abstract graphs. Proc Lond Math Soc (3) 2:69–81

Elmirghani J, Mouftah H (2000) All-optical wavelength conversion technologies and applications in DWDM networks. IEEE Commun Mag 38(3):86–92

Fernandes L, Gouveia L (1998) Minimal spanning trees with a constraint on the number of leaves. Eur J Oper Res 101:250–261

FICO (2009) Xpress-MP. http://www.dashoptimization.com/

Gargano L, Hell P, Stacho L, Vaccaro U (2002) Spanning trees with bounded number of branch vertices. In: Lecture notes in computer science, vol 2380. Springer, Berlin, pp 355–363

Glover F, Laguna M (1997) Tabu search. Kluwer Academic, Norwell

Graham R, Lawler E, Lenstra J, Rinnooy Kan A (1979) Optimization and approximation in deterministic sequencing and scheduling: a survey. Ann Discrete Math 4:287–326

Jajszczyk A (2005) Optical networks—the electro-optic reality. Opt Switch Netw 1(1):3–18

Jia X, Du D, Hu X, Huang H, Li D (2003) On the optimal placement of wavelength converters in WDM networks. Comput Commun 26:986–995

Kleinberg J, Kumar A (2001) Wavelength conversion in optical networks. J Algorithms 38(1):25–50

Lee C, Kim H (2007) Reliable overlay multicast trees for private Internet broadcasting with multiple sessions. Comput Oper Res 34(9):2849–2864

Leggieri V, Nobili P, Triki C (2008) Minimum power multicasting problem in wireless networks. Math Methods Oper Res 68:295–311

Montemanni R, Gambardella LM (2005) Exact algorithms for the minimum power symmetric connectivity problem in wireless networks. Comput Oper Res 32:2891–2904

Naadef D (2004) Polyhedral theory and branch-and-cut algorithm for the symmetric TSP. In: Gutin G, Punnen A (eds) The traveling salesman problem and its variations. Kluwer Academic, Norwell. Chapter 2

Pinedo M (2007) Planning and scheduling in manufacturing and services. Springer, New York

Salamon G, Wiener G (2008) On finding spanning trees with few leaves. Inf Process Lett 105:164–169

Tomlinson W, Lin C (1978) Optical wavelength-division multiplexer for the 1–1.4-micron spectral region. Electron Lett 14:345–347

Volkmann L (1996) Estimation for the number of cycles in a graph. Period Math Hung 33(2):153–161

Wilfong G, Winkler P (1998) Ring routing and wavelength transmission. In: Proceedings of the ninth annual ACM-SIAM symposium on discrete algorithms (SODA), pp 333–341

Wolsey L (1998) Integer programming. Wiley-Interscience, New York

Zhou Y, Poo G (2005) Optical multicast over wavelength-routed WDM networks: a survey. Opt Switch Netw 2(3):176–197