Relations and a Genetic Approach for Three Degree-Constrained Spanning Tree Problems

C. Cerrone[a], R. Cerulli[b], A. Raiconi[b]

[a]*Department of Computer Science, University of Salerno, Via Ponte Don Melillo, 84084 Fisciano (SA) Italy*
[b]*Department of Mathematics, University of Salerno, Via Ponte Don Melillo, 84084 Fisciano (SA) Italy*

Abstract

In this paper we take into account three different degree-constrained spanning tree problems with main application in the field of optical network design. In particular, we propose the classical Minimum Leaves Spanning Tree problem as a relevant problem in this field and show its relations with the Minimum Branch Vertices and the Minimum Degree Sum problems. We present a unified genetic algorithm for the three problems and show its effectiveness on a wide range of test instances.

Keywords: Combinatorial optimization, Genetic algorithms, Spanning Trees, Optical Networks

1. Introduction

Spanning tree problems with degree-related constraints are among the most relevant and widely studied in the field of constrained network design. In such problems, we generally look for spanning trees respecting certain conditions regarding the degree of the vertices, in order to model restrictions or cost factors deriving from the underlying real-world applications.

In the *Minimum Branch Vertices Problem* (or MBV), we look for the spanning tree with the minimum number of vertices (called *branch* vertices) with a degree higher than two.

This problem has great relevance, for instance, in the context of multicast on optical networks. On such networks, the optical signal can be split and therefore sent from a source to multiple destinations by using an appropriate network device (*switch*). Multicast communications can therefore be performed through a spanning tree of the network (light-tree), by placing a switch on each branch vertex. For several

Email addresses: `cerronecarmine@gmail.com` (C. Cerrone), `raffaele@unisa.it` (R. Cerulli), `araiconi@unisa.it` (A. Raiconi)

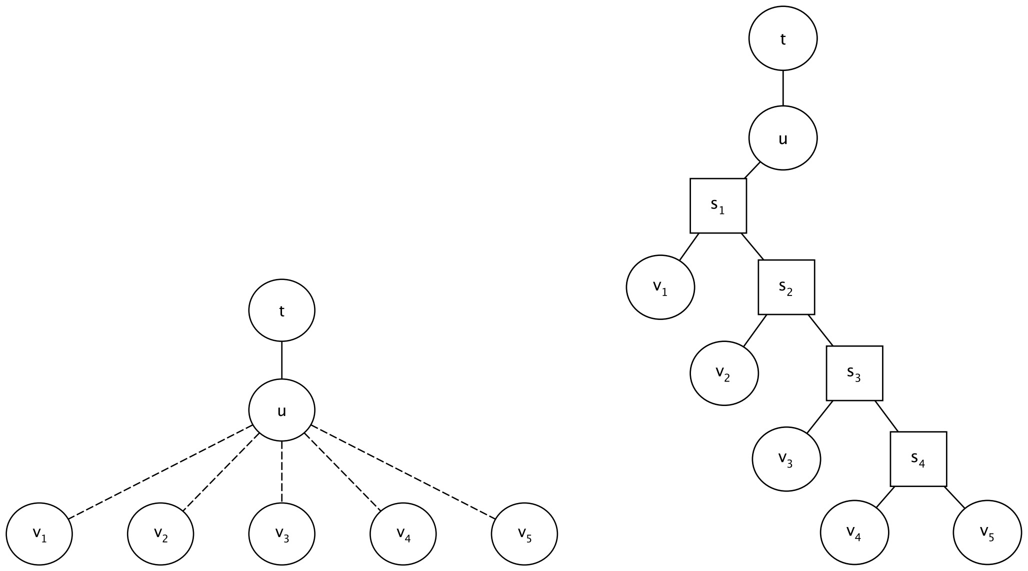


Figure 1: Example subtree. Switches $s_1, \ldots, s_4$ need to be located on node u to transmit information from t to $v_1, \ldots, v_5$.

reasons (such as budget constraints or signal quality preservation, among others) it can be important to determine the spanning tree which requires the minimum number of switches, that is, the optimal solution for the MBV problem.

Indeed, many switch devices can only duplicate laser beams; therefore, the actual number of devices to be located on a branch vertex is related to the degree of the node. For this reason, the problem of minimizing the degree sum of the branch vertices of any spanning tree of the network (*Minimum Degree Sum Problem* or MDS) has been proposed in the literature.

However, as can be noted from Figure 1, if $\delta(u, T)$ is the degree of a branch node that is used to propagate information, the exact number of required devices is $\delta(u, T) - 2$. More in general, in this context, the optimization problem consists in minimizing the degree sum of the branch vertices less the cardinality of the set of branch vertices multiplied by two. In this paper we show for the first time that this problem, which models the considered underlying application more accurately than MDS, is equivalent to the well known *Minimum Leaves Problem* (ML), i.e. the problem of finding the spanning tree with the minimum number of degree-1 vertices. This will be proved in Section 2.

The aim of this paper is therefore to propose ML as a relevant problem in the field of optical network design and to show that it is closely related to MDS and MBV, by demonstrating some theoretical properties linking their three objective functions. These objectives are also pursued by presenting a unified genetic algorithm that makes use of a single set of rules to perform crossover, mutation an local search operations for the three problems. An extensive experimental phase proves the effectivity of the proposed approach.

MBV has been first introduced in [1] where the problem was shown to be $\mathcal{NP}$-Hard. In [2] the authors present four different mathematical formulations and compare the results of different relaxations, solving the lagrangian dual by means of a standard subgradient method and an ad-hoc finite ascent algorithm. MDS has been presented and analyzed in [3], which also contains some mathematical formulations and heuristic procedures for both MBV and MDS. Other heuristic approaches for MBV and MDS have been recently proposed in [4]. The ML problem was proven to be $\mathcal{NP}$-Hard and hard to approximate in [5]. In [6], the authors introduce some approximation algorithms for the related problem of maximizing the number of internal nodes (of course, the two problems have identical optimal solutions).

The sequel of the paper is organized as follows. Section 2 contains the formal definition of the studied problems, and the demonstrations of several relations among them. Section 3 introduces a mathematical formulation for each of the three problems, and Section 4 describes the genetic algorithm that we designed to solve them. Section 5 includes the results of the extensive computational tests we performed to compare our genetic with the mathematical formulations solved by means of the CPLEX solver. Finally, Section 6 presents some final remarks.

2. Problems Definitions and relations

2.1. Notation

Let $G = (V, E)$ be a connected undirected input graph, and $T = (V, E')$ be a subgraph of G. Let $V(T)_i \subseteq V$, $i \geq 0$ be the set of vertices with degree i in T. Moreover, let $V(T)_B = V \setminus \{V(T)_1 \cup V(T)_2\}$; that is, if T is a spanning tree, $V(T)_B$ is the set of its branch vertices.

Moreover, for each $j \in V$, let $\delta(j, T)$ be the degree of j in T. Note that if $j \in V(T)_i$, then $\delta(j, T) = i$. Finally, for each set of vertices $X \subseteq V$, let $\Delta(X, T)$ be the degree sum of the vertices of X in T

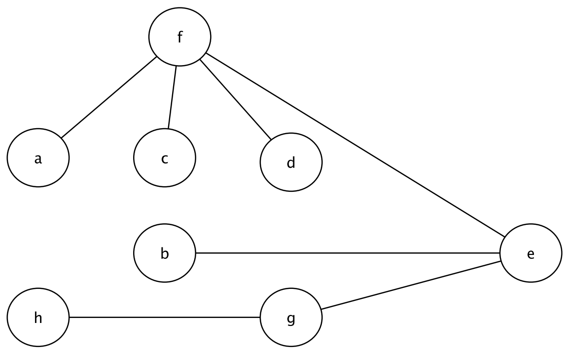


Figure 2: Example tree T

$(\Delta(X,T) = \sum_{j \in X} \delta(j,T))$.

In order to better clarify the introduced notation, consider the tree T in Figure 2. We have that $V(T)_1 = \{a,b,c,d,h\}$, $V(T)_2 = \{g\}$, $V(T)_3 = \{e\}$, $V(T)_4 = \{f\}$, $V(T)_B = \{e,f\}$. Moreover, for example, $\delta(g,T) = 2$, $\Delta(\{a,b,f\},T) = 6$, $\Delta(V(T)_B,T) = \Delta(\{e,f\},T) = 7$.

2.2. Definitions

We can now formally define the three problems studied in this paper.

Minimum Branch Vertices Problem (MBV)

Find a spanning tree T of G such that the number of branch vertices is minimized, that is, such that the set $V(T)_B$ has the minimum cardinality.

Minimum Degree Sum Problem (MDS)

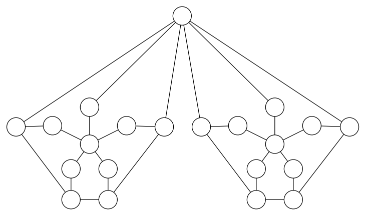
Find a spanning tree T of G such that the degree sum of the branch vertices is minimized, that is, such that $\Delta(V(T)_B,T)$ is minimized.

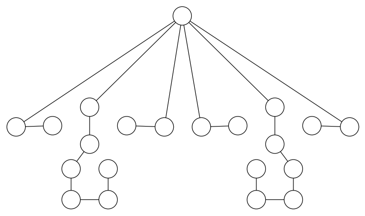
Minimum Leaves Problem (ML)

Find a spanning tree T of G such that the number of degree-1 vertices is minimized, that is, such that the set $V(T)_1$ has the minimum cardinality.

2.3. Relations among the problems

In [3] the authors proved that, though being strictly related, MDS and MBV are not equivalent. This was shown by presenting an example graph admitting two distinct optimal solutions for MBV, one of which is also optimal for MDS, while the other is not. However, it was not proved whether an optimal


(a) Example graph


(b) Optimal solution for both MBV and MDS

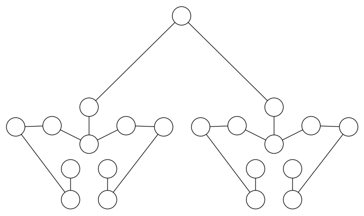
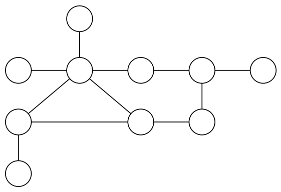
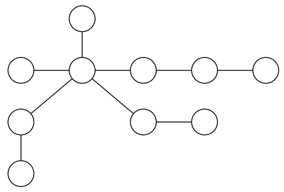

(c) Optimal solution for MDS

Figure 3: MBV and MDS are not equivalent


(a) Example graph


(b) Optimal solution for both MBV and MDS

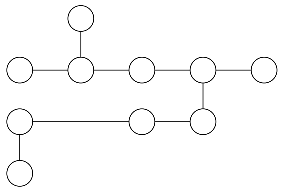

(c) Optimal solution for ML

Figure 4: MBV, MDS and ML are not equivalent

solution for MDS is always optimal for MBV. Here we provide evidence that this is not true, and therefore that optimal solutions for the two problem can be completely distinct.

Consider the example graph in Figure 3(a). The optimal solution for the MBV problem consists in a spanning tree with a single branch vertex, as shown in Figure 3(b). However, the optimal solution value of MDS for this graph is six, that is also the degree sum of the two branch vertices of the tree shown in Figure 3(c), which of course is not an optimal solution for MBV.

It can also be easily shown that the ML problem is not equivalent to neither MBV nor MDS. For example, given the graph in Figure 4(a), the optimal solution value for MBV is one, and the optimal solution value for MDS is five; Figure 4(b) shows an optimal tree for both the problems. This solution has five degree-1 vertices, while the optimal ML solution value for this instance is four; the related optimal solution is shown in Figure 4(c). This solution is not optimal for the other two problems, having two branch vertices with a total degree sum equal to six.

Now, consider the following propositions:

Proposition. *Given a connected undirected graph $G = (V, E)$ and a spanning tree $T = (V, E')$ of G, the following equation holds:*

$$\Delta(V, T) = 2|V| - 2 \tag{1}$$

Proof. Since any spanning tree T of G has exactly $|V|-1$ edges and any edge of T increases the degree of two nodes by one, if follows that

$$\Delta(V,T) = 2|E'| = 2(|V|-1) = 2|V|-2 \tag{2}$$

∎

Proposition. *Given a connected undirected graph $G=(V,E)$ and a spanning tree $T=(V,E')$ of G, the following equation holds:*

$$\Delta(V(T)_B,T) = 2|V|-2-|V(T)_1|-2|V(T)_2| \tag{3}$$

Proof. It is straightforward to observe that

$$\Delta(V(T)_B,T) = \Delta(V,T) - \Delta(V(T)_1,T) - \Delta(V(T)_2,T) \tag{4}$$

$$\Delta(V(T)_1,T) = |V(T)_1| \tag{5}$$

$$\Delta(V(T)_2,T) = 2|V(T)_2| \tag{6}$$

By substituting (1), (5) and (6) inside (4), we obtain (3). ∎

Proposition. *Given a connected undirected graph $G=(V,E)$ and a spanning tree $T=(V,E')$ of G, the following equation holds:*

$$|V(T)_1| = \Delta(V(T)_B,T) - 2|V(T)_B| + 2 \tag{7}$$

Proof. By reformulating (3) we have

$$|V(T)_1| = 2(|V|-|V(T)_2|) - 2 - \Delta(V(T)_B,T) \tag{8}$$

It is also easy to note that

$$|V| - |V(T)_2| = |V(T)_B| + |V(T)_1| \tag{9}$$

By substituting (9) in (8) and reformulating, we obtain (7). ∎

We can use (7) to show that ML is a problem of interest in the field of optical network design. As already said in Section 1, many switch devices can only duplicate light signals. Consider a spanning tree $T=(V,E')$ which is used for multicast communications on a graph $G=(V,E)$. Let u be a given vertex which is reached by the signal; no switch devices are needed in u if $\delta(u,T)=1$ (i.e., u is a leaf) or $\delta(u,T)=2$ (u just propagates the signal coming from its parent in T to its child). If u is a branch vertex, the signal entering in u must be propagated to its $\delta(u,T)-1 \geq 2$ children; without loss of generality let us call them $v_1,\dots,v_{\delta(u,T)-1}$. As exemplified in Figure 1, this can be accomplished using $\delta(u,T)-2$ devices $(s_1,\dots,s_{\delta(u,T)-2})$: the signal is sent from u to s_1 and each s_i propagates it to v_i and to either s_{i+1} if $i<\delta(u,T)-2$ or v_{i+1} if $i=\delta(u,T)-2$. By iterating this reasoning for each branch vertex, we have that the number of required switch devices is $\Delta(V(T)_B,T)-2|V(T)_B|$, which is the right hand side of (7) minus a constant factor of 2. Therefore ML can be used to find the spanning tree which requires the minimum number of switch devices.

2.3.1. Unified objective function

The relations among the three problems can be further highlighted by rewriting their objective functions in terms of a parametric unified objective function, as we will show in the following.

The MBV objective function can be reformulated as

$$\min |V(T)_B| = \min(|V| - |V(T)_1| - |V(T)_2|) \tag{10}$$

Since $|V|$ is a constant value,

$$\min |V(T)_B| = -\max(|V(T)_1| + |V(T)_2|) + |V| \tag{11}$$

Moreover, using (3) the MDS objective function can be reformulated as

$$\min \Delta(V(T)_B, T) = \min(2|V| - 2 - |V(T)_1| - 2|V(T)_2|) \tag{12}$$

Since $2|V| - 2$ is a constant value,

$$\min \Delta(V(T)_B, T) = -\max(|V(T)_1| + 2|V(T)_2|) + 2|V| - 2 \tag{13}$$

Finally, it is also easy to note that the ML objective function can be reformulated as

$$\min |V(T)_1| = -\max -|V(T)_1| \tag{14}$$

We can then use the following objective function to find optimal solutions for the three problems

$$\max(\alpha|V(T)_1| + \beta|V(T)_2|) \tag{15}$$

where

- **for MBV**: $\alpha = 1, \beta = 1$;
- **for MDS**: $\alpha = 1, \beta = 2$;
- **for ML**: $\alpha = -1, \beta = 0$.

3. Mathematical Formulations

In this section, we present a mathematical formulation for each of the three considered problems. The formulations are based on the well-known Miller-Tucker-Zemlin subtour elimination constraints [7] to impose the selection of a spanning tree T of the input graph. The formulations are defined on directed graphs, therefore we consider a directed version of G that contains both arcs (u, v) and (v, u) for each edge $(u, v) \in E$; let $G^d = (V, E^d)$ be this graph. The idea underlying MTZ constraints is to define a labeling function on the nodes of G^d, such that an arbitrary source vertex s has the smallest label and for each arc (u, v) selected to be part of the spanning tree vertex u has a smaller label than v.

The mathematical formulation for MBV is the following:

$$\min \sum_{v \in V} y_v \tag{16}$$

subject to

$$\sum_{(u,v) \in E^d} x_{uv} = 1 \qquad \forall v \in V \setminus \{s\} \tag{17}$$

$$t_v \leq |V| - (|V| - 1)x_{sv} \qquad \forall v \in V \setminus \{s\}, (s, v) \in E^d \tag{18}$$

$$t_v \geq 2 - x_{sv} \qquad \forall v \in V \setminus \{s\}, (s, v) \in E^d \tag{19}$$

$$(|V| - 2)x_{vu} + |V|x_{uv} + t_u \leq t_v + (|V| - 1) \qquad \forall (u, v) \in E^d, u \in V \setminus \{s\}, v \in V \setminus \{s\} \tag{20}$$

$$\sum_{(v,u) \in E^d} x_{vu} + \sum_{(u,v) \in E^d} x_{uv} \leq \delta(v, G)y_v + 2 \qquad \forall v \in V \tag{21}$$

$$x_{uv} \in \{0, 1\} \qquad \forall (u, v) \in E^d \tag{22}$$

$$y_v \in \{0, 1\} \qquad \forall v \in V \tag{23}$$

$$t_v \geq 0 \qquad \forall v \in V \tag{24}$$

Binary variables x_{uv} $\forall (u, v) \in E^d$ and y_v $\forall v \in V$ determine whether (u, v) is part of T and v is a branch vertex, respectively, while variables t_v contain the labeling function values. The objective function (16) minimizes the number of branch vertices. Constraints (17) make sure that each vertex except the source has exactly one parent in T. Constraints (18)-(19) impose that, for each vertex v adjacent to s, $t_v = 1$ if $x_{sv} = 1$, $t_v \geq 2$ otherwise; they are an improvement introduced in [8] of the classical constraints imposing $t_s = 0$ and $t_v \geq 1 \ \forall v \in V \setminus \{s\}$. Constraints (20) ensure that if $x_{uv} = 1$, $u \neq s$ and $v \neq s$, then $t_v = t_u + 1$, and are a lifted version proposed in [9] of the original MTZ constraints. Finally, constraints (21) impose vertex v to be a branch if its degree is greater than two in the tree; note that the value of y_v is unconstrained if $\delta(v, T) \leq 2$, however in this case it will be set to 0 by the objective function.

The considered formulation for the MDS problem is the following:

$$\min \sum_{v \in V} z_v \tag{25}$$

subject to

(17)-(24)

$$\sum_{(v,u) \in E^d} x_{vu} + \sum_{(u,v) \in E^d} x_{uv} \leq z_v + 2 - 2y_v \qquad \forall v \in V \tag{26}$$

$$z_v \geq 0 \qquad \forall v \in V \tag{27}$$

We need an additional set of variables z_v in order to take into account the degree of the branch vertices. The model uses all the constraints of the MBV formulation to build a spanning tree and identify branch vertices; moreover, Constraints (26) impose $z_v \geq \delta(v, T)$ if $y_v = 1$ and therefore v is a branch vertex, leaving it unconstrained if $y_v = 0$. Objective function (25) minimizes the sum of variables z_v and therefore will impose $z_v = \delta(v, T)$ for the branch vertices, 0 otherwise.

Finally, we present a formulation for ML:

$$\min \sum_{v \in V} y_v \tag{28}$$

subject to

(17)-(20),(22)-(24)

$$\sum_{(v,u)\in E^d} x_{vu} + \sum_{(u,v)\in E^d} x_{uv} + y_v \geq 2 \quad \forall v \in V \tag{29}$$

In this formulation, variables y_v are used to represent whether the vertices are leaves or not in T. Again, constraints (17)-(20) are used to define a tree. Constraints (29) ensure that $y_v = 1$ if $\delta(v,T) = 1$. The value of y_v is not constrained by (29) if it is not a leaf, however since the objective function (28) minimizes the sum of y_v variables, it will be set to 0.

4. Genetic Algorithm

Genetic algorithms (GA) are randomized metaheuristic techniques based on the biological process of natural selection that have been successfully applied to many combinatorial optimization problems. A genetic algorithm emulates the evolutionary process on a *population* composed of solutions (*chromosomes*). While starting populations are often created randomly, new individuals are iteratively formed by recombining together two or more older chromosomes, or by perturbing a single one; the best chromosomes have generally better chances of being selected in these steps. Therefore, new solutions are likely to inherit good characteristics from old solutions, and, by repeating this process over a sufficient number of generations, eventually near-optimal solutions can be reached. For an introduction to genetic algorithms, see for example [10].

The main elements that must be provided in order to implement a GA are:

- A representation scheme for each chromosome. Our GA considers only feasible solutions, therefore each chromosome is a spanning tree and is represented internally as a list of its edges.
- A function to evaluate each chromosome (*fitness* function). Our GA evaluates the chromosomes using the parametric objective function (15). Depending on the problem on which we intend to focus, the appropriate values for parameters α and β are used.
- Rules to derive new solutions, by combining two parent solutions (*crossover*) and by perturbing a single individual (*mutation*). Our crossover and mutation operators are described in Sections 4.2 and 4.3 respectively.

- Termination criteria for the procedure. In our case, the algorithm ends when a given number of iterations without improvements has been performed.

A high level outline of the procedure is given in Algorithm 1. As can be seen, our algorithm also includes a local search phase in order to look for further local improvements starting from specific individuals, and can therefore be considered a *memetic* algorithm [11].

Each step reported in Algorithm 1 will be explained in detail in the remaining part of this section. The algorithm description makes use of various input parameters, whose setting during our experimental phase is discussed in Section 5.

Algorithm 1 Genetic Algorithm

```
1: Build a random population $\Pi$
2: while iterations without improvements $\leq$ max-it do
3:     choose two parent chromosomes $T_1$, $T_2$
4:     perform crossover on $T_1$ and $T_2$ obtaining $T_3$
5:     perform a mutation on $T_3$ obtaining $T_3'$
6:     perform a local search starting from $T_3'$, obtaining $T_3''$
7:     insert $T_3''$ in $\Pi$ substituting an older chromosome $T_i$
8: end while
```

4.1. Parents Selection and Child Insertion Policies

As reported in Line 3 of Algorithm 1, at each iteration, two chromosomes are selected for the crossover phase. Our algorithm uses *tournament selection*, which is a scheme commonly used by GAs in order to favor good individuals.

We implemented a simple tournament selection which can be summarized as follows:

- Select randomly h chromosomes from the population Π , let T_1 be the one with the *best* fitness function value among them;

- Select randomly $\frac{h}{2}$ chromosomes from $\Pi \setminus \{T_1\}$, let T_2 be the one with the *best* fitness function value among them;

- Select T_1, T_2 as parents for the crossover phase.

Ties are broken randomly. The idea underlying the implemented tournament scheme is to favor the selection of at least a parent chromosome with a good fitness value, while a higher degree of diversity is left for the selection of the other parent. Of course, the chosen value for parameter h influences both the convergence rate of the algorithm and its capacity to escape from local minima.

The new chromosome T_3'' that will result after the phases reported in Lines 4-6 will replace an older one; therefore the population size $|\Pi|$ is constant throughout every phase of the algorithm. The substituted element is also selected using a tournament mechanism:

- Select randomly k chromosomes from Π , let T_i be the one with the *worst* fitness function value among them;
- Substitute T_i with T_3'' in Π.

4.2. Crossover

The crossover operator builds a new individual from two parents. In classical GAs, crossover is performed by simple recombination of the information contained in the parents. For example, supposing that in a GA chromosomes are represented as lists, two children might be obtained from two parents by swapping between them all the data contained after a given position (*crossover point*). Our crossover, instead, creates new individuals which inherit "good" characteristics from the parents, but are not a direct recombination of them.

More in detail, given the parent chromosomes T_1 and T_2, we start by defining two weight functions w_1 and w_2 on the edges of the input graph $G = (V, E)$; each w_i is based on the characteristics of T_i. Finally, the new spanning tree T_3 is built by finding a Minimum Spanning Tree of the weighted graph $G_w = (V, E, w)$, where

$$w(u,v) = w_1(u,v) + w_2(u,v) \qquad \forall (u,v) \in E$$

The aim of the weight functions is to penalize or favor the selection in T_3 of the edges of E, using three key ideas:

- Strongly favor the selection of chains coming from the parents; this is an obvious choice since we would like to obtain the highest possible number of vertices with degree ≤ 2 in the final solution.

The longer is a chain, the more its edges are favored.

- Favor the selection of edges connected to vertices that are branch in the parents; as the procedure converges, some branch vertices are likely to be perceived as required. Edges actually belonging to the parents are more favored than new edges adjacent to such vertices.

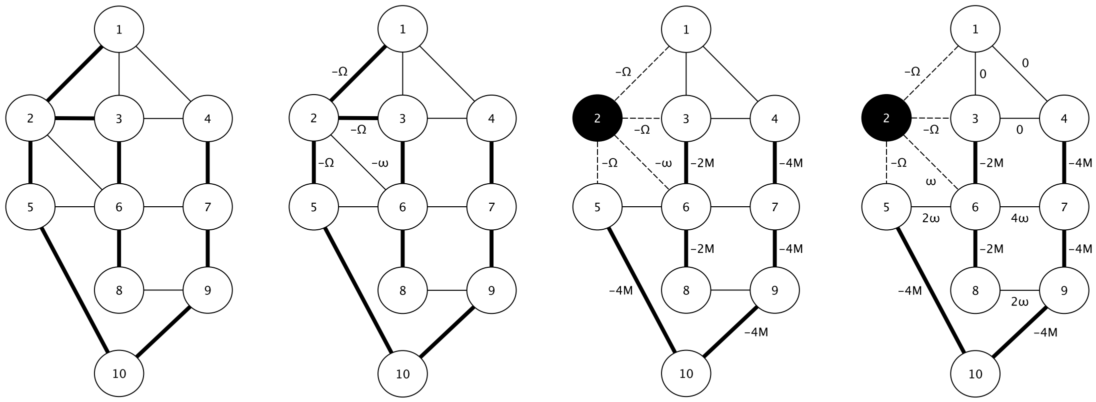
- Penalize the selection of edges adjacent to vertices with degree 2 in the chains, in order to avoid new branch vertices.

These guidelines might suggest to both penalize and favor the same edge. We represent penalizing and favoring weights for each edge (u, v) and parent $T_i = (V, E_i)$ using three positive input parameters M, Ω, ω, such that $M > 4\Omega$ and $\Omega > 4\omega$. More in detail, w_i is computed as follows:

1. Initialize $w_i(u, v) = 0 \ \forall (u, v) \in E$;
2. For each $u \in V(T_i)_B$, add $-\Omega$ to $w_i(u, v) \ \forall (u, v) \in E_i$;
3. For each $u \in V(T_i)_B$, add $-\omega$ to $w_i(u, v) \ \forall (u, v) \in E \setminus E_i$;
4. Remove all edges adjacent to branch vertices from T_i, obtaining the forest $F_i = (V, E_i^{'})$;
5. For each $(u, v) \in E_i^{'}$, let $l(u, v)$ be the length of the chain it belongs to in F_i, and add $-Ml(u, v)$ to $w_i(u, v)$;
6. For each $u \in V(F_i)_2$, add 2ω to $w_i(u, v) \ \forall (u, v) \in E \setminus E_i^{'}$.

A positive value for $w_i(u, v)$ means that the selection of (u, v) is penalized according to parent T_i, while a negative value means that it is considered a useful edge (recall that the weights are used to compute a minimum spanning tree).

The above described steps are illustrated on the tree shown in Figure 5. In particular, Figure 5(b) shows the negative weights applied to favor edges incident to the branch vertex 2. In Figure 5(c) the negative weights associated with chains are added, and finally Figure 5(d) includes the penalizing weights for the edges adjacent to nodes with degree 2 in F_i. It can be noted that each edge with a positive weight would add at least one branch vertex if included in F_i; in particular, the favoring weight assigned to $(2, 6)$ for being adjacent to branch vertex 2 is counterbalanced by its penalization deriving from vertex 6.



(a) Example tree T_i (b) Negative weights assigned to edges adjacent to branch vertex 2 (c) Negative weights assigned to chains (d) Positive weights assigned to edges adjacent to chains

Figure 5: Weighting function example for the crossover operation

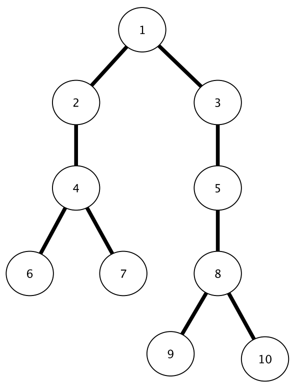
The chosen values for parameters ω, Ω and M are reported in Section 5. By imposing $M > 4\Omega$, we make sure that edges belonging to chains are always favored with respect to other edges (the smallest weight that can be assigned to an edge that is not in a chain is -4Ω, associated to an edge belonging to both parents and adjacent to two branch vertices in each of them). For a similar reasoning, we also impose $\Omega > 4\omega$.

4.3. Mutation

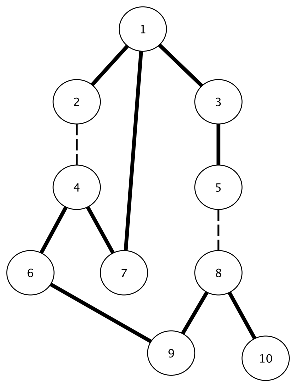
Given the new individual T_3 resulting from the crossover, the mutation phase operates by applying two different mutation operators $\mathbf{M_1}, \mathbf{M_2}$. M_1 is always executed, while M_2 is executed after M_1 with probability p_{m2}. The operators work as follows:

- $\mathbf{M_1}$: A leaf node of T_3 is randomly selected and connected to another random node. The resulting cycle is broken randomly obtaining a new spanning tree;
- $\mathbf{M_2}$: Randomly eliminates two edges, trying to favor the elimination of branch vertices by repeating the random choice up to a fixed number of iterations it_{M_2}. The resulting three components are rejoined by randomly selecting two new edges, obtaining the new tree.

Mutation M_2 is illustrated in Figure 6, where two branch vertices are removed by selection edges $(2, 4)$ and $(5, 8)$, and a new branch vertex is created when edges $(1, 7)$ and $(6, 9)$ (supposing that they exist in



(a) Example tree T_i



(b) Mutated tree obtained by substituting $(2, 4)$ and $(5, 8)$ with $(1, 7)$ and $(6, 9)$

Figure 6: Mutation M_2 example

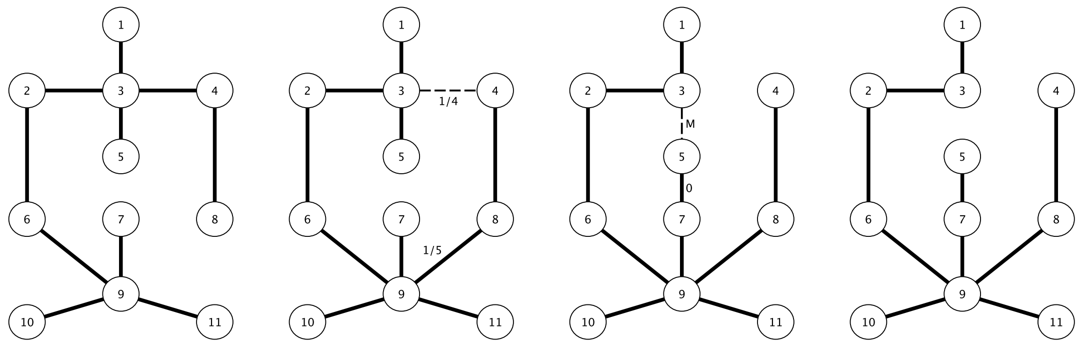
G) are chosen to rejoin the three components.

4.4. Local Search

At the end of each iteration of the genetic algorithm, a local search is performed on the new chromosome T_3' obtained after the mutation phase. Similarly to the M_1 operator, each iteration of the local search tries to switch one of the edges belonging to the current solution with a new one; however, instead of selecting edges randomly, we look for an *improvement* of the original tree. The improvement is evaluated according to a heuristic function that favors the introduction of new nodes whose degree is not greater than two and strongly penalizes the introduction of new branch vertices. Additional edges added to old branch vertices are also penalized, using values that are inversely proportional to the node degree. More in detail, given a subgraph H of G, we define the following functions:

1. Let $im(x, H)$ be a function defined on the nodes of H, such that $im(x, H)$ is equal to: 0 if $\delta(x, H) \leq 2$, M if $\delta(x, H) = 3$, $1/\delta(x, H)$ if $\delta(x, H) > 3$;
2. Let $IM((u, v), H)$ be a function defined on the edges of H such that $IM((u, v), H) = im(u, H) + im(v, H)$.

Let (u', v') be an edge of T_3'; moreover, let $(u'', v'') \notin T_3'$ be an edge of G with an endpoint in each of the two connected components that would be created by removing (u', v') from T_3'. We use the IM function to evaluate if replacing (u', v') with (u'', v'') would be a convenient choice as follows:



(a) Example tree T_i (b) Improved tree obtained by substituting $(3, 4)$ with $(8, 9)$ (c) Improved tree obtained by substituting $(3, 5)$ with $(5, 7)$ (d) Resulting tree

Figure 7: Tree improvements in local search

1. Let $H = T_3' \cup \{(u'', v'')\}$;
2. If $IM((u'', v''), H) < IM((u', v'), H)$ then $T_3'' = H \setminus \{(u', v')\}$ is an *improvement* of T_3'.

Each iteration of the local search looks for an improvement, by considering all the possible edge substitutions that can be used to obtain a new tree. Each iteration accepts the first improvement found, and the local search phase ends when no improvements are available.

Two local search iterations are shown in Figure 7. Starting from the tree with two branch vertices in Figure 7(a), removing $(3, 4)$ and rejoining the components using edge $(8, 9)$ (supposing that it exists in the graph) produces a new tree that is an improvement of the previous one, as shown in Figure 7(b). Note that by effect of the weighting function, it is preferred to lower the degree of vertex 3 from 4 to 3, at the expense of increasing the degree of vertex 9 from 4 to 5. On the resulting tree, assuming that edge $(5, 7)$ also exists in the graph, it allows to decrease again the degree of vertex 3 by removing edge $(3, 5)$, as illustrated in Figure 7(c), producing the tree with a single branch vertex shown in Figure 7(d).

5. Computational Results

In this section, we describe the computational results that we obtained by applying the proposed GA for the three problem variants on a set of test instances. The mathematical formulations introduced in Section 3 were used to obtain optimal solutions. Three single-commodity flow formulations were also considered and implemented, however they resulted to be consistently slower than the MTZ ones and therefore their results are not reported.

5.1. Instances Description

In order to test the performances of our genetic algorithm, we considered a wide set of test instances. The instances, generated according to parameters originally proposed in [2] for the MBV problem, are designed to be sparse in order to require a significant number of branch vertices. We generated instances with 14 different values for the number of nodes n between 150 and 1000. The number of edges is generated according to the following formula: $\lfloor(|V|-1)+i\times 1.5\times\lceil\sqrt{|V|}\rceil\rfloor$ with $i=1,2,3,4,5$, for a total of 70 scenarios. We generated five instances for each scenario, therefore the total number of instances is 350.

5.2. Parameters and Testing Environment

Regarding our genetic algorithm, after a tuning phase we chose the following values for the GA parameters, which seemed to provide a good tradeoff among solution quality and computational time: $|\Pi|=100$, $max\text{-}it=25000$, $h=4$, $k=2$, $M=21$, $\Omega=5$, $\omega=1$, $p_{m2}=0.5$, $it_{m2}=5$. The genetic algorithm has been coded in C++. The IBM ILOG CPLEX 12 solver was used to solve the Miller-Tucker-Zemlin formulations, considering a time limit of 1 hour for each instance. When instances could not be solved to completion in 1 hour, the best solution found is reported. All tests have been executed on an Intel Xeon 2Ghz workstation with 8GB of RAM.

5.3. Results

Tables 1-3 contain average percentage ratios between GA and MTZ solutions for each scenario; each entry is an average on the 5 instances corresponding to a given number of nodes and a given value for the i parameter. It can be noted that, on average, the objective function value of the solutions returned by the GA is never higher than 5% with respect to the one returned by the mathematical models for MDS and ML. For MBV, the gap can grow up to 18.11%, reached on instances with 160 nodes and $i=5$; however, it is higher than 10% only in four out of 70 scenarios. It can be noted that the gap tends to grow on higher densities, especially for MBV; however, this can be easily explained by the fact that instances with higher density allow generally a smaller number of branch vertices in the optimal solutions, and therefore suboptimal solutions have a highest weight in the percentage. It is also worth nothing that the gaps do not deteriorate on instances with a higher number of nodes; while in the case of MBV and MTZ this could partially depend on the fact that for many of the bigger instances the CPLEX solver could not provide a guaranteed optimal solution within the considered time limit, this is not the case for ML. Overall, the GA proves to be an effective metaheuristic for the three problems.

Tables 4-6 report the average computational times for the GA, expressed in seconds. It can be noted that the procedure has reasonable computational times regardless of the considered problem, averaging about 7 seconds for the smallest instances and requiring on average up to 794.22 seconds for MBV, 678.77 seconds for MDS and 408.16 seconds for ML for instances with 1000 nodes.

Finally, Table 7 shows the number of times that the MTZ formulations failed to provide a certified optimal solution for MBV and MTZ on the 30 biggest scenarios (from 350 to 1000 nodes). Recalling that each scenario is composed of 5 instances, an entry with value 5 means that none of its instances were solved to completion. The number of failures grows with the size of the input; in particular, seven out of 25 instances with 1000 nodes could be solved for MBV, and only four of them could be solved for MDS, emphasizing the need for efficient heuristic approaches. While this behavior was less noticeable for ML, since the instances were specifically designed for MBV, the complex and unapproximable nature of the problem suggests the relevance of heuristic resolution methods also for this problem.

6. Conclusions

In this paper, we show that finding a spanning tree with the minimum number of leaves is a relevant problem in the context of multicast communications on optical networks. In particular, we demonstrate that it can be used to model the problem of minimizing the number of required light-splitting devices more accurately than two problems already proposed in the literature to solve the same issue (namely, the Minimum Branch Vertices and the Minimum Degree Sum problems). Moreover, we propose a unified genetic procedure that makes use of a common set of rules to produce accurate solutions for the three problems, as shown by our computational test phase.

Table 1: Genetic/MTZ average percentage gap for MBV

	i **parameter**				
|V|	**1**	**2**	**3**	**4**	**5**
150	0.54	0.71	5.86	7.87	8.22
160	0.49	0.63	3.34	8.29	18.11
170	0.48	1.84	4.21	7.72	11.58
180	0	1.7	3.14	7.75	12.7
190	0	1.54	2.07	5.06	9.17
200	0	1.5	2.62	5.57	11.24
250	0	2.25	3.21	5.81	9.22
300	0.25	1.18	2.97	4.78	7.59
350	0	1.25	2.14	4.93	6.65
400	0.36	1.5	2.8	3.54	6.09
450	0.32	1.49	2	3.11	3.98
500	0.43	1.49	2.27	3.13	5.49
750	0.43	0.94	1.64	2.11	3.29
1000	0.32	0.76	1.11	2.15	3.05

Table 2: Genetic/MTZ average percentage gap for MDS

	i **parameter**				
|V|	**1**	**2**	**3**	**4**	**5**
150	0	0.36	2.02	2.6	4.63
160	0	0	2	3.34	4.82
170	0	0.6	2.14	3.69	3.89
180	0.11	0.55	2.13	1.86	4.97
190	0	0.37	0.46	2.33	4.84
200	0.1	0.6	1.36	2.14	3.9
250	0.07	0.45	0.85	2.79	2.99
300	0.12	0.29	1.13	2.22	3.59
350	0.05	0.48	0.86	1.82	2.66
400	0.18	0.36	0.78	1.09	2.49
450	0.12	0.46	0.89	1.37	2.2
500	0.07	0.32	0.76	1.4	1.93
750	0.21	0.45	0.82	0.77	1.72
1000	0.22	0.37	0.53	1.04	1.44

Future research will be aimed at obtaining a better characterization of the similarities among the three problems, looking for possible dominance relations under particular assumptions. Further efforts will be also spent at producing improved resolution approaches and test instances coming from real-world applications.

Table 3: Genetic/MTZ average percentage gap for ML

	***i* parameter**				
|V|	**1**	**2**	**3**	**4**	**5**
150	1.03	1.7	1.77	2.2	3.3
160	0.99	1.56	4.85	4.09	2.93
170	1.38	0.58	4.7	3.32	4.71
180	1.06	1.38	2.85	2.8	4.86
190	0.58	1.24	1.85	3.45	2.26
200	1.68	0.7	2.41	1.79	3.3
250	0.73	0.18	1.06	1.85	2.24
300	0.6	1.14	1.06	2.6	3.55
350	0.51	0.72	0.73	2.07	4.02
400	0.43	0.71	0.35	1.24	2.54
450	0.55	0.54	0.74	0.84	3.04
500	0.49	0.55	0.37	1.53	3.18
750	0.13	0.26	0.37	0.24	0.36
1000	0.25	0.16	0.3	0.18	0.28

Table 4: Genetic average computational time for MBV

	***i* parameter**				
|V|	**1**	**2**	**3**	**4**	**5**
150	7.35	7.68	7.01	6.9	7.75
160	9.15	8.34	8.19	7.49	7.62
170	9.43	9.41	10.92	8.16	8.38
180	11.5	10.11	10.09	10.22	10.08
190	11.46	10.95	10.53	10.6	9.66
200	12.35	12.14	11.64	16.22	12.31
250	21.42	19.51	22.25	22.94	18.77
300	30.79	29.98	31.86	33.72	29.17
350	40.86	42.27	53	48.19	45.42
400	54.87	61.89	64.31	70.79	55.03
450	72.12	80.19	73.35	95.43	96.44
500	93.13	99.34	103.8	108.46	94
750	231.04	272.59	350.89	361	369.97
1000	503.78	528.75	654.34	794.22	613.52

References

[1] L. Gargano, P. Hell, L. Stacho, U. Vaccaro, Spanning trees with bounded number of branch vertices, in: Automata, Languages and Programming, Vol. 2380 of Lecture Notes in Computer Science, Springer, Berlin / Heidelberg, 2002, pp. 355–365.

[2] F. Carrabs, R. Cerulli, M. Gaudioso, M. Gentili, Lower and upper bounds for the spanning tree with minimum branch vertices, submitted to Computational Optimization and Applications.

Table 5: Genetic average computational time for MDS

	***i* parameter**				
|V|	**1**	**2**	**3**	**4**	**5**
150	7.72	7	6.99	7.76	10.85
160	9.29	8.81	8.14	8.26	10.22
170	10	10.22	9.09	9.05	10.5
180	10.87	12.96	10.46	12.45	9.1
190	12.25	12.9	12.19	13.08	10.19
200	12.49	12.71	13.06	18.56	15.51
250	22.07	18.58	27	21.91	23.1
300	34	39.92	35.01	35.13	39.08
350	42.77	55.08	51.81	53.45	64.37
400	55.63	67.9	70.71	83.66	63.92
450	77.97	77.52	86.39	86.15	103.79
500	94.33	118.57	119.75	143.65	133.01
750	225.61	269.55	318.71	421.73	363.17
1000	423.92	553.14	623.47	678.77	609.98

Table 6: Genetic average computational time for ML

	***i* parameter**				
|V|	**1**	**2**	**3**	**4**	**5**
150	6.91	6.67	6.53	6.35	7.26
160	7.8	8.01	8.73	6.93	6.81
170	8.83	8.8	8.07	9.3	7.97
180	9.94	10.09	9.24	10.1	9.38
190	10.98	10.58	10.86	11.94	9.53
200	12.06	11.89	11.59	11.45	10.51
250	18.93	19.64	20.07	21.49	20.14
300	28.5	28.44	32.98	30.73	31.17
350	39.51	39.62	44.31	43.16	39.89
400	53.56	53.23	52.25	74.4	70.52
450	67.72	69.08	75.09	92.41	99.97
500	87.046	87.09	107.49	103.41	135.54
750	211.62	225.29	222.11	220.1	220.33
1000	387.47	399.91	402.81	405.79	408.16

Table 7: Number of MTZ failures

(a) MBV

	***i* parameter**				
|V|	**1**	**2**	**3**	**4**	**5**
350	0	3	1	2	1
400	0	0	0	1	0
450	0	1	2	0	2
500	0	0	2	1	2
750	0	3	3	3	4
1000	1	3	5	4	5

(b) MDS

	***i* parameter**				
|V|	**1**	**2**	**3**	**4**	**5**
350	0	1	1	1	2
400	0	0	0	0	1
450	0	1	2	1	1
500	0	0	1	1	1
750	0	4	5	4	3
1000	1	5	5	5	5

[3] R. Cerulli, M. Gentili, A. Iossa, Bounded-degree spanning tree problems: models and new algorithms, Computational Optimization and Applications 42 (3) (2009) 353–370.

[4] S. Sundar, A. Singh, A. Rossi, New heuristics for two bounded-degree spanning tree problems, Information Sciences 195 (2012) 226–240.

[5] H.-I. Lu, R. Ravi, The power of local optimization: Approximation algorithms for maximum-leaf spanning tree, Tech. Rep. CS-96-05, Brown University, RI (Jan. 1996).

[6] G. Salamon, G. Wiener, On finding spanning trees with few leaves, Information Processing Letters 105 (5) (2008) 164–169.

[7] C. E. Miller, A. W. Tucker, R. A. Zemlin, Integer programming formulations and travelling salesman problems, Journal of the ACM 7 (4) (1960) 326–329.

[8] L. Gouveia, Using the Miller-Tucker-Zemlin constraints to formulate a minimal spanning tree with hop constraints, Computers & Operations Research 22 (9) (1995) 959–970.

[9] M. Desrochers, G. Laporte, Improvements and extensions to the Miller-Tucker-Zemlin subtour elimination constraints, Operations Research Letters 10 (1) (1991) 27–36.

[10] C. R. Reeves, Genetic algorithms, in: Handbook of Metaheuristics, Vol. 146 of International Series in Operations Research & Management Science, Springer, US, 2010, pp. 109–139.

[11] J.-K. Hao, Memetic algorithms in discrete optimization, in: Handbook of Memetic Algorithms, Vol. 379 of Studies in Computational Intelligence, Springer, Berlin / Heidelberg, 2012, pp. 73–94.