The Rainbow Spanning Forest Problem

Francesco Carrabs[a], Carmine Cerrone[c,*], Raffaele Cerulli[a], Selene Silvestri[b]

[a] *Department of Mathematics, University of Salerno Via Giovanni Paolo II n. 132, 84084, Fisciano (SA), Italy.*
[b] *Department of Computer Science, University of Salerno, Via Giovanni Paolo II n. 132, 84084, Fisciano (SA), Italy.*
[c] *Department of Biosciences and Territory, University of Molise, C.da Fonte Lappone, 86090 Pesche (IS), Italy.*

Abstract

Given an undirected and edge colored graph G, a rainbow component of G is a subgraph of G having all the edges with different colors. The Rainbow Spanning Forest Problem consists of finding a spanning forest of G with the minimum number of rainbow components. The problem is known to be NP-hard on general graphs and on trees. In this paper we present an integer linear mathematical formulation and a greedy algorithm to solve it. To further improve the results we applied a multi-start scheme to the greedy algorithm. Computational results are reported on randomly generated instances.

Keywords: Graph theory, edge-colored graph, rainbow components, multi-start scheme, heterochromatic components.

1. Introduction and Problem Description

Let $G = (V, E, L)$ be a connected and undirected graph, where V is the set of n vertices, E is the set of m edges and L is a set of l colors. In addition, let $\ell : E \to L$ be a coloring function that assigns to each edge $e \in E$ a color $\ell(e)$ from the set L. A *rainbow spanning forest* of G is a spanning forest $F = (V_F, E_F, L_F)$ of the graph G, with $V_F = V$, $E_F \subseteq E$ and $L_F \subseteq L$, such that all components are rainbow. A component of the forest is a connected acyclic graph, therefore a *rainbow component* of F is a tree $T = (V_T, E_T, L_T)$, where $V_T \subseteq V$ and $E_T \subseteq E_F$, in which all edges have different colors, i.e. $|L_T| = |\ell(E_T)| = |E_T|$. Note that if F has $\bar{c}$ rainbow components $T_1, \ldots, T_{\bar{c}}$, then $V_F = V = \cup_{i=1}^{\bar{c}} V_{T_i}$, $E_F = \cup_{i=1}^{\bar{c}} E_{T_i}$ and $L_F = \cup_{i=1}^{\bar{c}} L_{T_i}$. The *Rainbow Spanning Forest Problem* (RSFP) consists of finding a rainbow spanning forest with the least number of rainbow trees.

*Corresponding author
Email addresses: fcarrabs@unisa.it (Francesco Carrabs), carmine.cerrone@unimol.it (Carmine Cerrone), raffaele@unisa.it (Raffaele Cerulli), selene.silvestri@gmail.com (Selene Silvestri)

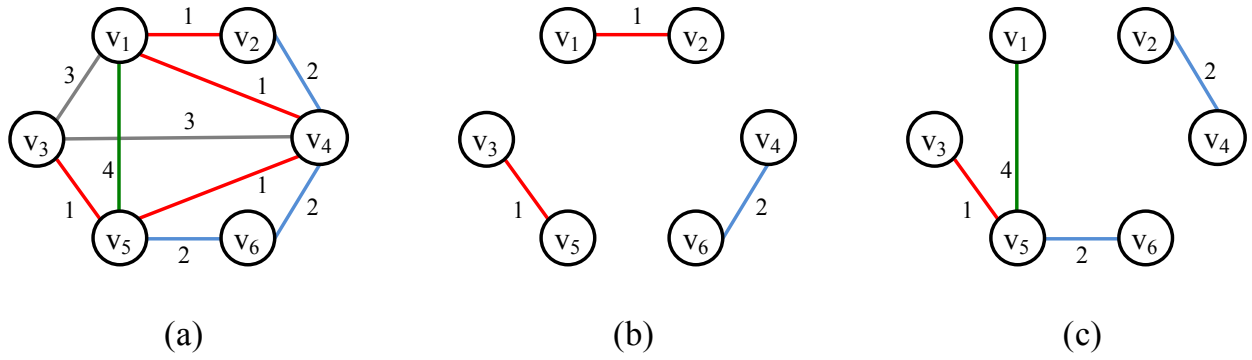

Figure 1: *(a) The colored graph G. (b) A rainbow spanning forest composed by three rainbow trees. (c) A rainbow spanning forest composed by only two rainbow trees. This is the optimal solution for the RSFP on G.*

For instance, given the colored graph G, depicted in Figure 1(a), a feasible solution for the RSFP is shown in Figure 1(b). This is a rainbow spanning forest which value is equal to three because it is composed by three rainbow trees. Notice that in this solution the color 1 is used in two rainbow trees. Finally, the optimal solution for the RSFP on G is shown in Figure 1(c).

Li and Zhang [23] proved that the RSFP is NP-hard on general graphs. Akbari and al. [1] gave necessary and sufficient condition for the existence of a heterochromatic spanning tree in an edge-colored connected graph. Carrabs et al. [6] proved that the problem is NP-hard on trees and that it is polynomially solvable if the optimal solution value is equal to one. More recently Carraher et al. [9] give conditions to calculate a lower bound on the number of edge-disjoint spanning trees rainbow shown in a graph.

The RSFP belongs to a recently studied class of problems, defined on edge-colored graphs. Such a type of graphs may be used to model many real-world situations in which we need to distinguish between different types of connections. For example, in telecommunication networks colors can represent different types of communications media (such as optical fiber, coaxial cable, telephone line), different companies to which the connections belong, or different transmission frequencies.The RSFP generalizes a well-known problem in the context of edge-colored graphs, that is, the Minimum Labeling Spanning Tree Problem (MLSTP). The MLSTP was introduced by Chang and Shing-Jiuan [15], who proved it to be NP-hard and provided an heuristic, the Maximum Vertex Coverage Algorithm (MVCA), as well as an exact A* algorithm. Brualdi et al. [3] give conditions on color distributions of the complete bipartite graph which guarantee the existence of rainbow subgraph, while Suzuki

[25] gives a necessary and sufficient condition for the existence of a rainbow spanning tree in a graph. Many other colored problems have been studied in the literature, like the Minimum Labeling Steiner Problem [17], [18], the Minimum Labeling Spanning Tree Problem [10], [11], [14], [22], the Minimum Labeling Generalized Forest [4], the Colorful Traveling Salesman Problem [13], [21], [28], the Generalized Minimum Label Spanning Tree Problem [16], the Label-Constrained Minimum Spanning Tree Problem [29], the Labeled Maximum Matching Problem [8], the Maximum Labeled Clique Problem [7], the Orderly Colored Longest Path Problem [26], the Optimal Pathway Reconstruction on 3D NMR maps [27], the Proper–Path Colorings Problem [2] and the Rainbow Cycle Cover Problem [24].

In this paper we propose an integer mathematical formulation and a greedy algorithm. In order to identify the local optimal choice, the greedy algorithm solves a matching problems on bipartite graphs. The bipartite graphs are built at each iteration taking into account the current solution. Moreover, to further improve the results we embed the greedy algorithm in a multi-start scheme.

The remainder of the paper is organized as follows. Section 2 contains the integer linear mathematical formulation. The greedy algorithm and the related multi-start scheme applied to the greedy algorithm to improve its results are presented in Section 3. Computational results and concluding remarks are presented in Section 4 and 5, respectively.

2. RSFP: An Integer Linear Mathematical Formulation

In this section we provide an integer linear mathematical formulation for the RSFP. Let α_c be a set of binary variables, for $c = 1, \ldots, \bar{c}$, associated with each component c of a rainbow spanning forest, whose value is equal to 1 if and only if c contains at least one vertex. Let y_v^c be a set of binary variables such that y_v^c is equal to 1 if and only if vertex v belongs to component c, and let x_e^c be a set of binary variables equal to 1 if and only if edge e belongs to component c. Note that the number of variables depends on the number of possible components, so it is useful to have a good upper bound to the optimal value. If we do not know an upper bound we set $\bar{c} = n - 1$, that is equal to the maximum number of rainbow components that we can identify. In order to introduce constraints that can help prevent equivalent solutions, we define an index set $I_q = \{1, \ldots, q\}$, for an integer q, and the vertex set $V = I_n$. The formulation (ILP) is then as follows:

$$\text{minimize } z = \sum_{c=1}^{\bar{c}} \alpha_c \tag{1}$$

subject to

$$\sum_{v \in V} y_v^c \leq (l+1)\,\alpha_c \qquad c = 1, \ldots, \bar{c} \tag{2}$$

$$\alpha_c \leq \sum_{v \in V} y_v^c \qquad c = 1, \ldots, \bar{c} \tag{3}$$

$$\sum_{c=1}^{\bar{c}} y_v^c = 1 \qquad v \in V \tag{4}$$

$$x_e^c \leq y_v^c \qquad v \in V,\ e \in \delta(v) \tag{5}$$

$$\sum_{e \in E_k} x_e^c \leq \alpha_c \qquad c = 1, \ldots, \bar{c},\ k \in L \tag{6}$$

$$\sum_{c=1}^{\bar{c}} \sum_{e \in E(S)} x_e^c \leq |S| - 1, \qquad S \subseteq V,\ |S| \geq 2 \tag{7}$$

$$\sum_{e \in E} x_e^c = \sum_{v \in V} y_v^c - \alpha_c \qquad c = 1, \ldots, \bar{c} \tag{8}$$

$$\alpha_{c+1} \leq \alpha_c \qquad c = 1, \ldots, \bar{c} - 1 \tag{9}$$

$$y_1^1 = 1 \tag{10}$$

$$y_v^c \leq \sum_{w < v} y_w^{c-1} \qquad v \in V \setminus \{1\},\ c = 3, \ldots, \bar{c} \tag{11}$$

$$\alpha_c \in \{0, 1\} \qquad c = 1, \ldots, \bar{c} \tag{12}$$

$$y_v^c \in \{0, 1\} \qquad v \in V,\ c = 1, \ldots, \bar{c} \tag{13}$$

$$x_e^c \in \{0, 1\} \qquad e \in E,\ c = 1, \ldots, \bar{c}, \tag{14}$$

where $\delta(v)$ denotes the set of edges incident to v in G and $E_k = \{e \in E : \ell(e) = k\}$.
The objective function (1) requires the minimization of the number of rainbow components. Constraints (2) and (3) are logical constraints linking the binary variables α_c with the binary variables y_v^c. Note that the maximum number of vertices that can belong to the same component is $l + 1$, since l is the number of different colors of the graph. Constraints (4) ensure that each vertex belongs to exactly one component. Constraints (5) impose that if a vertex is not in the component c, the edge incident on such vertex cannot belong to that tree. Constraints (6) impose that a component cannot contain two edges having the same color, ensuring the rainbow property. Constraints (7) are the subtour elimination constraints, introduced by Dantzig et al. [19], adapted to our variables. Constraints (8) impose that each component is a tree. They guarantee solutions with not more than one

tree associated to each variable α_c. Constraints (9), (10) and (11) help break symmetries. Constraints (9) mean that there will never be a variable α_{c+1} equal to one if α_c is equal to zero, for any c. The constraints (10) and (11) are the symmetry breaking constraints introduced by Fischetti et al. [20] in the context of the Vehicle Routing Problem. Vertex v can belong to a component of index c if and only if at least one vertex w with a lower index belongs to the component of index $c-1$. For the resolution of the model we also use the following constraints:

$$y_v^c \leq \alpha_c \qquad v \in V,\ c = 1, \ldots, \bar{c} \qquad (15)$$

$$\sum_{e \in \delta_k(v)} x_e^c \leq y_v^c \qquad v \in V,\ c = 1, \ldots, \bar{c},\ k \in L \qquad (16)$$

$$\sum_{c=1}^{\bar{c}} \left\{ x_e^c + \sum_{f \in \{\delta_k(u) \cup \delta_k(v)\}} x_f^c \right\} \leq 2 \qquad e = (v,u) \in E,\ k \in L \setminus \{\ell(e)\} \qquad (17)$$

proposed by Silvestri et al. [24] to solve the Rainbow Cycle Cover Problem, and which are valid for the RSFP. Constraints (15) state that if a vertex belongs to a component, then the variable representing that component must be used. The valid inequalities (16) impose that if vertex v belongs to a tree c, then at most one edge having color k and incident to v can be selected. Constraints (17) impose that if edge $e = (v,u)$ is selected, then at most one edge having color $k \neq \ell(e)$ and belonging to the set $\{\delta_k(v) \cup \delta_k(u)\}$ can be selected.

3. The Greedy Algorithm

In this section we propose a greedy algorithm for the Rainbow Spanning Forest Problem. Moreover, to further improve the results we embed it in a multi-start scheme. As described below, the greedy algorithm uses three different selection criteria in order to perform at each iteration the most promising choice. Given a rainbow spanning forest $F = (V, E_F, L_F)$, we denote by $\hat{E}_F$ the set of *feasible edges* in $E \setminus E_F$, that is the set of edges whose endpoints belong to different components of F and such that an edge and the two components to which the endpoints belong to, do not have colors in common. Formally, $\hat{E}_F = \{(u,v) \in E \setminus E_F : u \in T_i,\ v \in T_j,\ i \neq j,\ \{\ell(T_i) \cap \ell(T_j)\} = \emptyset,\ \{\ell(u,v)\} \cap \{(\ell(T_i) \cup \ell(T_j))\} = \emptyset$, where T_i and T_j are two generic components of F.

The greedy algorithm starts with the trivial feasible solution $F_0 = (V, E_{F_0}, L_{F_0})$, with $E_{F_0} = L_{F_0} = \emptyset$, in which each vertex is an acyclic and rainbow component. At iteration k the algorithm selects a feasible edge $(u,v) \in \hat{E}_{F_k}$, with $u \in T_i$ and $v \in T_j$, and builds the new

rainbow spanning forest F_{k+1} by adding the edge (u, v) in F_k, that is $E_{k+1} = E_k \cup \{(u, v)\}$ and $L_{k+1} = L_k \cup \{\ell(u, v)\}$. It is easy to see that the solution F_{k+1} contains a new larger tree, with respect to F_k, obtained by joining T_i and T_j through (u, v). For this reason, at each iteration the number of trees in the rainbow spanning forest decreases by one, while the number of edges increases by one. The set of feasible edges $\hat{E}_{F_{k+1}}$ is obtained by removing from $\hat{E}_{F_k}$ all edges that are no longer feasible due to the insertion of (u, v). The algorithm stops when there are no more feasible edges. Procedure 1 shows the pseudocode of the greedy algorithm.

Algorithm 1: Greedy Algorithm

Input: graph $G = (V, E, L)$.
Output: a rainbow spanning forest F of G.

1 Set the number of components z equal to n and $k = 0$
2 F_k trivial starting solution, $\hat{E}_{F_k} = E$
3 **while** $\hat{E}_{F_k}$ *is not empty* **do**
4 update the weights of feasible edges $e \in \hat{E}_{F_k}$
5 select feasible edge $(u, v) \in \hat{E}_{F_k}$ with the min weight
6 $F_{k+1} \leftarrow F_k \cup \{(u, v)\}$
7 update the set of feasible edges $\hat{E}_{F_{k+1}}$
8 update the number of components $z = z - 1$
9 $k = k + 1$
10 **return** F_k.

A key point of the greedy algorithm is the selection of the feasible edge at each iteration. To this end, we define the function $w : \hat{E}_{F_k} \to R$ that associates a weight to the feasible edges in $\hat{E}_{F_k}$ (line 4). The weight $w(u, v)$ represents an estimate of the number of potential improvements that we lose by adding edge (u, v) at iteration k. For this reason, the algorithm always selects the edge of lowest weight (line 5). The details regarding the computation of the weights are given below.

Suppose that the graph depicted in Figure 2 is the rainbow spanning forest F_k built by the greedy algorithm at iteration k. The dashed edges are the feasible edges connecting the four rainbow trees T_1, T_2, T_3, T_4. Let $\delta_k(T_i)$ be the sets of feasible edges incident to the vertices belonging to T_i. Moreover, let $N_k(T_i)$ be the set of trees that can be connected to T_i in F_k through at least one feasible edge. Formally, $\delta_k(T_i) = \{(u, v) \in \hat{E}_{F_k} : u \in V_{T_i} \text{ or } v \in V_{T_i}\}$ and $N_k(T_i) = \{T_j \in T_{F_k} : \exists\, (u, v) \in \hat{E}_{F_k}, u \in T_i, v \in T_j\}$. For instance, in Figure 2 we have $\delta_k(T_1) = \{(v_1, v_5), (v_2, v_4), (v_2, v_8), (v_3, v_8), (v_3, v_{11})\}$ and $N_k(T_1) = \{T_2, T_3, T_4\}$. The first

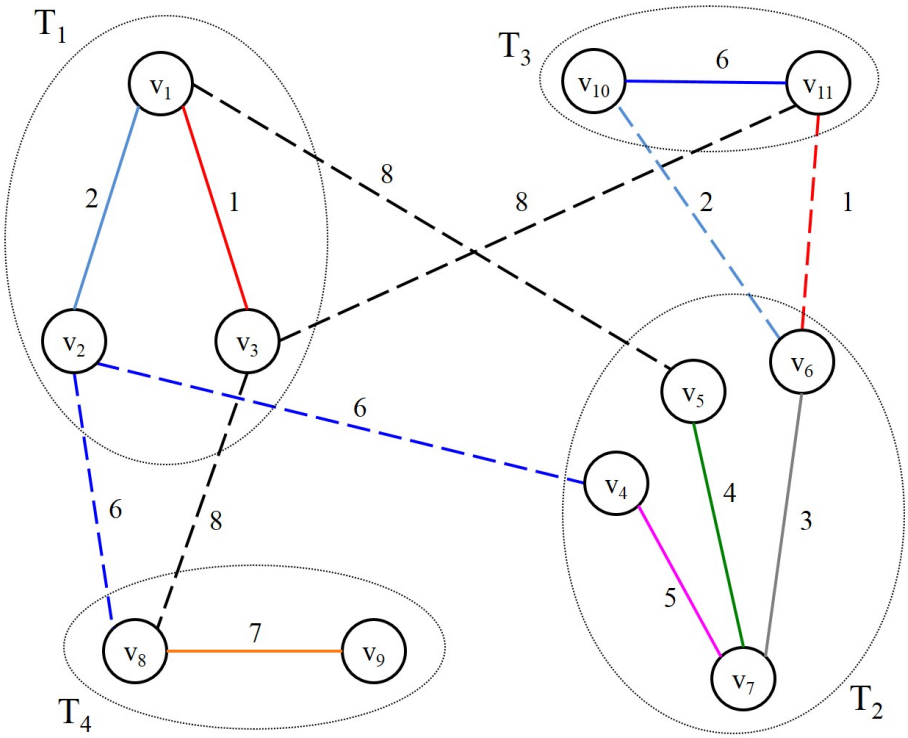

Figure 2: A feasible rainbow spanning forest composed by components T_1, T_2, T_3, T_4. The edges inside a component are depicted as solid lines while the edges among the components are shown as dashed lines. These last ones are the feasible edges candidated to be inserted in the incumbent solution. The numbers associated to the edges represent their colors.

information that we want to compute, for each tree T_i, is the "potential" number of trees in T_{F_k} that can be joined to T_i through feasible edges. We call this number the *joining number* and we denote it by M_{T_i}. It is easy to see that $|N_k(T_i)|$ is a trivial upper bound to the value of M_{T_i}. We compute M_{T_i} by solving the maximum matching problem on a bipartite graph B_i in which we define the two sets of vertices $V^1_{B_i}$ and $V^2_{B_i}$ as follows:

- for each $l \in \ell(\delta_k(T_i))$ we define vertex l i.e. $l \in V^1_{B_i}$;
- for each $T_j \in N_k(T_i)$ we define vertex T_j i.e. $T_j \in V^2_{B_i}$.

Moreover, for each feasible edge $(u, v) \in \delta_k(T_i)$, with $v \in T_j$, a corresponding edge in B_i between vertices $\ell(u, v)$ and T_j is introduced. Figure 3(a) depicts the bipartite graph B_1 associated to the tree T_1 of the Figure 2. Since $\ell(\delta_k(T_1)) = \{6, 8\}$ and $N_k(T_i) = \{T_2, T_3, T_4\}$ then the set of vertices $V^1_{B_1}$ contains only two vertices, $V^1_{B_1} = \{6, 8\}$, while the second set is composed of three vertices, $V^2_{B_1} = \{T_2, T_3, T_4\}$. Regarding the edges in B_i, since the edges

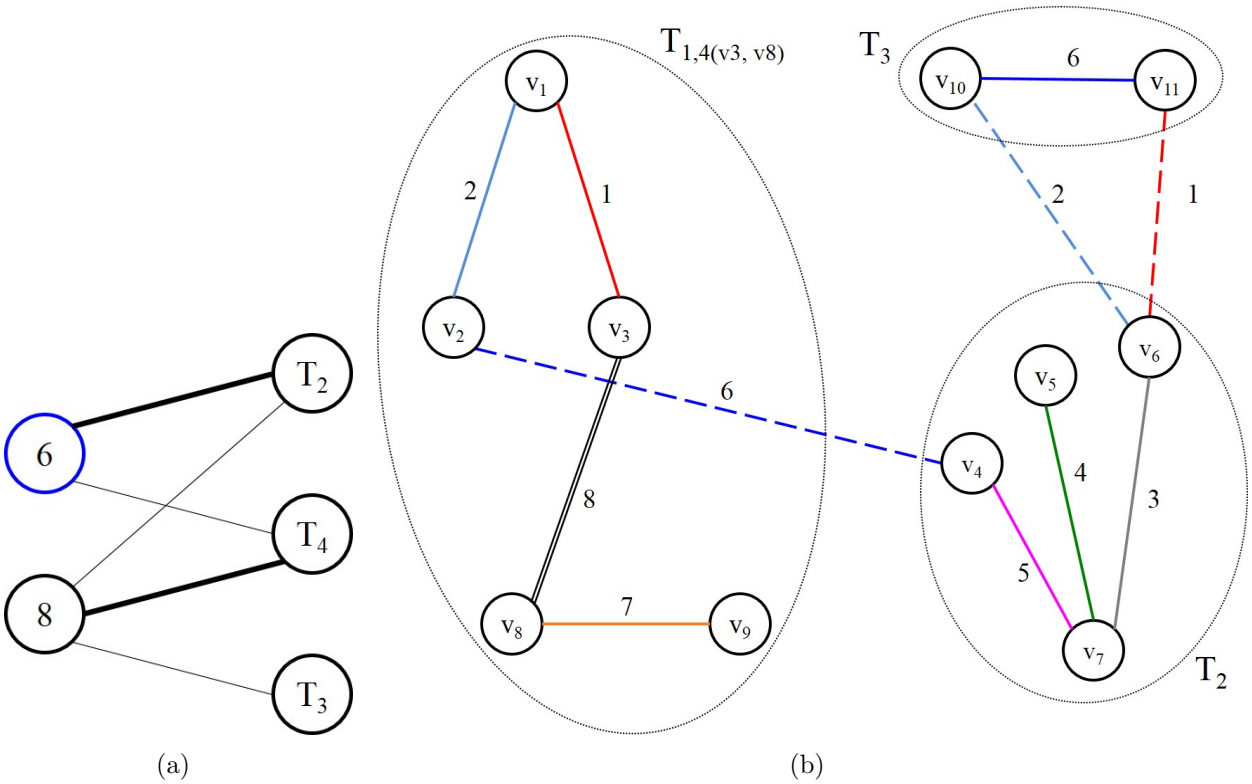

Figure 3: Feasible solution F_{k+1} obtained by adding edge (v_3, v_8).

with color 6 in $\delta_k(T_1)$ connect T_1 to T_2 and to T_4 then in the bipartite graph we connect node 6 with the vertices T_2 and T_4, and so on. The maximum matching of B_1 is equal to two (edges in bold) and then $M_{T_1} = 2$ that is tighter than $|N_k(T_1)| = 3$. This means that T_1 can be joined with, at most, two trees in T_{F_k}. The edges of the maximum matching "suggest" which are the feasible edges to select. For instance, edge $(8, T_4)$ of the maximum matching corresponds to the feasible edge (v_3, v_8) in $\hat{E}_{F_k}$ and by joining the trees T_1 and T_4 through (v_3, v_8), we obtain a larger tree, denoted by $T_{1,4}(v_3, v_8)$, and the new solution F_{k+1} shown in Figure 3(b). From now on, given a feasible edge (u, v), with $u \in T_i$ and $v \in T_j$, we denote by $T_{i,j}(u, v)$ the tree obtained by joining T_i and T_j through edge (u, v). Note that in F_{k+1} it is possible to join the new tree $T_{1,4}(v_3, v_8)$ with T_2, through (v_2, v_4), performing, in this way, the two joinings estimated by M_{T_1}. However, the number of joinings carried out does not always coincide with M_{T_i} because the choice of the feasible edge to select affects the final result, and when there are more maximum matchings in B_i there are also more choices available. For instance, Figure 4(a) shows another maximum matching of B_1. By joining T_1 and T_3 in F_k through the feasible edge (v_3, v_{11}), as shown in Figure 4(b), we obtain the

tree $T_{1,3}(v_3, v_{11})$ with $\delta_{k+1}(T_{1,3}(v_3, v_{11})) = \emptyset$. In this case we carried out a single joining compared to the two estimated joinings.

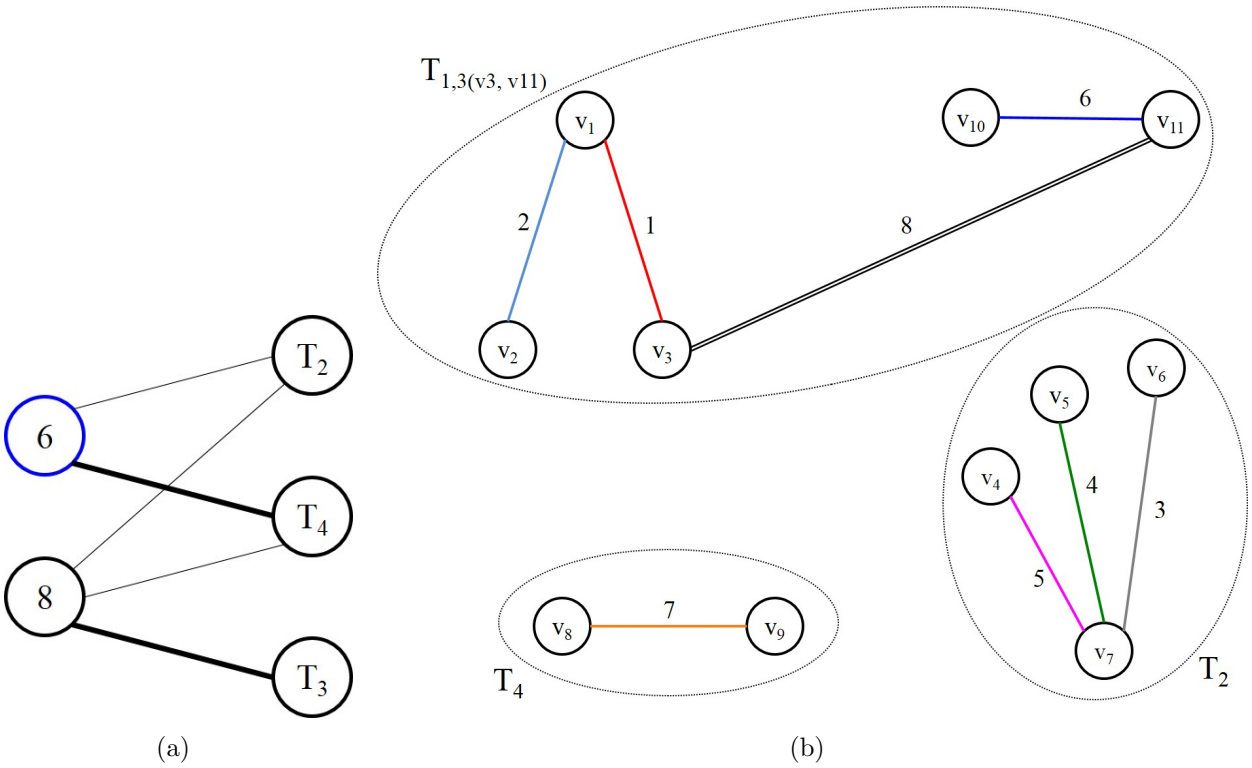

Figure 4: Feasible solution F_{k+1} obtained by adding edge (v_3, v_{11}).

The previous example shows that it is necessary to select the feasible edges in an accurate way. For this reason, we propose to first join the trees with lowest *joining number* (*first criterion*). Indeed, there is a high probability that these trees will remain isolated components if they are not joined as soon as possible because a low value of *joining number* means less joining opportunities. For instance, in the solution F_k, depicted in Figure 2, $M_{T_1} = M_{T_2} = M_{T_3} = 2$ and $M_{T_4} = 1$, hence T_4 is the tree on which carry out the joining operation with one of its neighbor, T_1 in this case. Since this joining operation can be carried out through edges (v_2, v_8) or (v_3, v_8), then we add a second criterion to select the most promising edge. For each edge $(u, v) \in \delta_k(T_4)$, the greedy algorithm computes the *joining number* of the new tree $T_{1,4}(u, v)$, obtained by joining T_1 and T_4 through (u, v), and it selects the edge (u, v) which value $M_{T_{1,4}(u,v)}$ is the maximum one (referred to as *second criterion*). As previously shown, by selecting edge (v_3, v_8) we obtain the new solution reported in Figure 3 with $M_{T_{1,4}(v_3,v_8)} = 1$. Since also $M_{T_{1,4}(v_2,v_8)} = 1$ then we provide a third selection criterion consisting of choosing the edge which color is less frequent in the current solution

(referred to as *third criterion*). In our example, $\ell(v_3, v_8) = 8$ and there are no edges in E_{F_k} with color 8 while the color 6 of edge (v_2, v_8) belongs to $\ell(E_{F_k})$ due to edge (v_{10}, v_{11}). For this reason, the greedy algorithm selects edge (v_3, v_8) and, thanks to this choice, it finds the optimal solution.

Summarizing, to each feasible edge (u, v) of $\hat{E}_k$, with $u \in T_i$ and $v \in T_j$, we associate a weight according to the following formula:

$$w(u, v) = n \times \min(M_{T_i}, M_{T_j}) - M_{T_{i,j}(u,v)} + \frac{|E_{F_k}(\ell(u, v))|}{z}, \tag{18}$$

where $E_{F_k}(c) = \{e \in E_{F_k} : \ell(e) = c\}$. This formula is a linear combination of three terms: $min(M_{T_i}, M_{T_j})$, $M_{T_{i,j(u,v)}}$ and $\frac{|E_{F_k}(\ell(u,v))|}{z}$. Obviously, lower is the weight of an edge higher is its probability to be selected. For this reason, the three terms of the linear combination are multiplied for appropriate coefficients (n, -1 and 1, respectively) to reflect this idea. More in details, the term $min(M_{T_i}, M_{T_j})$ is used to measure the joining number of T_i and T_j. Lower is the joining number lower should be the weight of the edge. The first term represents our primary selection criterion and then it has the priority with respect to the other two terms. For this reason, the first term is multiplied by n, in the linear combination. The second term $M_{T_{i,j(u,v)}}$ is taken into account when the first term value is the same for the two edges. Since higher is the value of the second term lower should be the weight of the edge, its value is subtracted in the linear combination. Finally, the last term represents the average number of the color $l(u, v)$ in F_k and its value is relevant in the formula only when two edges has the same value for the first and second terms. Accordingly, the weights for the feasible edges of the solution in Figure 2 are the following:

$$\begin{aligned}
w(v_1, v_5) &= 10 \times \min(2, 2) - 1 + 0 = 19 \\
w(v_2, v_4) &= 10 \times \min(2, 2) - 1 + \frac{1}{4} = 19.25 \\
w(v_2, v_8) &= 10 \times \min(2, 1) - 1 + \frac{1}{4} = 9.25 \\
w(v_3, v_8) &= 10 \times \min(2, 1) - 1 + 0 = 9 \\
w(v_3, v_{11}) &= 10 \times \min(2, 2) - 0 + 0 = 20 \\
w(v_6, v_{10}) &= 10 \times \min(2, 2) - 0 + \frac{1}{4} = 20.25 \\
w(v_6, v_{11}) &= 10 \times \min(2, 2) - 0 + \frac{1}{4} = 20.25
\end{aligned}$$

3.1. The multi-start scheme

The greedy algorithm starts from the trivial solution F_0 and, at each step k, adds a new edge from the set of feasible edges $\hat{E}_{F_k}$, until $\hat{E}_{F_k}$ is not empty. The selection of edges is carried out according to their weights, that are dynamically updated according to the choices performed in the previous steps. This means that a bad choice carried out in the first steps can heavily affect the quality of the final solution produced. We face this problem by embedding the greedy algorithm in a multi-start scheme which, by performing a deeper exploration of the solution space, finds better solutions. More specifically, the multi-start algorithm invokes the greedy algorithm several times with different starting feasible solutions in which a *first edge* is fixed. This first edge is chosen from a set of edges $\tilde{E}$ that the multi-start algorithm builds at the beginning of the computation. The key point is how the set $\tilde{E}$ is built because i) the effectiveness of the multi-start algorithm depends on the edges in this set, and ii) the performance of the multi-start algorithm depends on the cardinality of $\tilde{E}$, since the greedy algorithm is invoked $|\tilde{E}|$ times. To build $\tilde{E}$, the multi-start algorithm first computes the weight of all edges, according to equation (18), and then, for each vertex $v \in V$, it inserts in $\tilde{E}$ the edge of $\delta(v)$ with the lowest weight. If this edge already belongs to $\tilde{E}$ it is ignored and a new vertex is selected. This construction ensures that the greedy algorithm will build each solution starting from a different vertex. The selection of the most promising edge for each vertex rather than the most promising at all, i.e. the edges with n lowest weights, allows a better exploration of the solution space. Since the cardinality of $\tilde{E}$ affects the performance of the multi-start algorithm, we have to control the growth of this cardinality as the instances size increases. This choice, on the one hand, preserves the performance of the algorithm even on the larger instances and, on the other hand, reduces the quality of solutions found on these instances. For this reason, we bound the cardinality of $\tilde{E}$ as follows:

$$|\tilde{E}| \leq \min \left\{ 10 + \left\lfloor \frac{500}{\sqrt{n}} \right\rfloor, n \right\}. \tag{19}$$

With this formula we ensure that the multi-start algorithm invokes the greedy algorithm at least ten times (being $n \geq 20$ in our instances) and no more that n times. Moreover, the value $\lfloor \frac{500}{\sqrt{n}} \rfloor$ guarantees that $|\tilde{E}|$ decreases as the size of the instances increases. Finally, the multi-start algorithm invokes $|\tilde{E}|$ times the greedy algorithm and returns the best solution among the $|\tilde{E}|$ solutions identified.

4. Computational results

In this section we present computational results obtained by solving the ILP, the greedy algorithm and the multi-start method. The greedy algorithm and the multi-start scheme were coded in Java and computational experiments were performed on a 64-bit GNU/Linux operating system, 96 GB of RAM and one processor Intel Xeon X5675 running at 3.07 GHz. The mathematical model ILP was coded in C and solved using IBM ILOG CPLEX 12.5 on the same machine. We applied the Students t-test on the Tables 1 and 2 obtaining an error lower then the 1%. This result certify the statistical significance of the results obtained. To the best of our knowledge, there are no available benchmark instances for the RSFP and we therefore used randomly generated instances. Some instances considered in the current study were introduced by Silvestri et al. [24] for the Rainbow Cycle Cover Problem. Here we create additional instances according to the procedure used in [24], and described next. Each instance is characterized by the number of vertices n (size), the number of edges m and the number of colors l. Given the number of vertices n, the number of edges is set to $m = \lceil \frac{n(n-1)}{2} \times d + n \rceil$, with $d \in \{0.1, 0.2, 0.3\}$, and the number of colors is set to $\lceil \frac{1}{2} \log(m) \rceil$, $\lceil \log(m) \rceil$ and $\lceil 2 \log(m) \rceil$. Note that the number of colors is always less than n, therefore the optimal solution cannot be a tree and hence the instances are not polynomially solvable. The total number of different scenarios is nine for each size. Each scenario is composed by five different instances having the same number of vertices, edges and colors but the seed, used to initialize the pseudorandom number generator, is different. The results reported in each line of the tables are the average values computed over these five instances. The combination of all these parameters allows us to verify how the effectiveness and performance of our algorithms are affected by the number of vertices, the density of the graph and the number of colors. The scenarios are divided into two groups: the small scenarios, where the value of n ranges from 20 to 50 with a step equal to 10, and the large scenarios, where the value of n ranges from 100 to 400 with a step equal to 100.

Note that the density of the generated instances is low (up to 0.3) in order to obtain meaningful results. It is easy to see that for larger density values the value of the optimal solution almost always coincides with the trivial lower bound $LB = \lceil \frac{n}{l+1} \rceil$. To explain this we need to introduce the definition of *rainbow star*.

Definition 1. *Given an edge-colored tree $T = (V_T, E_T, L_T)$ with $|V_T| = k+1$, T is a rainbow star if and only if*

- *each edge has a different color, i.e. $|L_T| = |E_T| = k$;*

- *k vertices are leaves, i.e. k vertices have degree equal to one in T, and one vertex has degree k.*

Note that, for any vertex $v \in V$, as the density increases the probability that $\delta(v)$ contains a rainbow star increases. Therefore, for larger density, the probability of obtaining a trivial optimal solution with $\lceil \frac{n}{l+1} \rceil$ rainbow stars is larger.

Table 1 reports the results of the mathematical model (ILP), of the Greedy (GA) and of the multi-start (MS) algorithms, on small scenarios. The first four columns report the characteristics of each scenario: scenario ID, the number of vertices (n), the number of edges (m), the number of colors (l), respectively. The columns ILP, GA and MS are divided into two subcolumns (Value and Seconds) reporting the solution value and the computing time in seconds, respectively. We have imposed a time limit of $10,800$ seconds. Whenever α instances of a scenario were not solved to optimality by ILP, within the time limit, we report (α) close to the solution value, therefore the value reported is an upper bound on the optimal solution value. In this cases we say that a ILP *failure* occurs and we refer to the solutions with the symbol (α) as the *best bound* solutions. The last column (GAP) reports the gaps between the solution computed by ILP and the solutions found by GA and by MS, respectively. We mark in bold the gaps equal to zero to highlights the scenarios where the algorithms find an optimal solution.

The results of Table 1 show that ILP finds an optimal solution on 29 out of 36 scenarios and that the hardest scenarios to solve are those with lowest density ($d = 0.1$). Indeed, five of the seven ILP failures occur on scenarios with density 0.1 (ID n° 20, 21, 28, 29 and 30). The remaining two failures occur on scenarios with density 0.2 (ID n° 13 and 32). Finally, on the scenarios with density 0.3 ILP always finds the optimal solution and the CPU time is always lower than six minutes. These results highlight the good performances of our mathematical model, that almost always finds the optimal solution for the instances with a density greater than or equal to 0.2. The worst results are obtained on the scenarios with density 0.1 which from now on we denote as *critical scenarios*. This is probably due to the low number of optimal solutions available. Indeed, as the density decreases, the number of equivalent feasible solutions decreases. Therefore there may be a really small number of optimal solutions, and then it results harder to find one of them.

It is interesting to observe that for the greedy and the multi-start algorithms the critical scenarios are also the hardest to solve. Indeed, the highest gap values occur on these scenarios and on the critical scenario n° 28, the *peak* (the maximum gap value) of GA and MS occurs. This peak is equal to 3.4 for GA and 2.0 for MS.

ID	n	m	l	ILP		GA		MS		GAP	
				Value	Seconds	Value	Seconds	Value	Seconds	GA	MS
1	20	39	3	6.6	2.94	8.0	0.01	7.0	0.10	1.4	0.4
2			6	4.2	3.74	5.0	0.02	4.8	0.10	0.8	0.6
3			11	2.2	1.10	3.0	0.02	2.8	0.13	0.8	0.6
4	20	58	3	5.6	3.90	6.4	0.02	6.4	0.13	0.8	0.8
5			6	3.2	2.14	4.4	0.03	3.6	0.15	1.2	0.4
6			12	2.0	1.21	2.2	0.04	2.0	0.05	0.2	**0.0**
7	20	77	4	4.0	2.30	4.8	0.03	4.4	0.11	0.8	0.4
8			7	3.0	1.63	3.2	0.05	3.0	0.07	0.2	**0.0**
9			13	2.0	1.63	2.0	0.06	2.0	0.06	**0.0**	**0.0**
10	30	74	4	8.4	129.42	8.8	0.03	8.8	0.20	0.4	0.4
11			7	5.0	204.63	6.2	0.03	5.4	0.25	1.2	0.4
12			13	3.0	8.02	3.6	0.06	3.4	0.16	0.6	0.4
13	30	117	4	$6.8^{(1)}$	2199.73	8.4	0.04	7.4	0.33	1.6	0.6
14			7	4.2	16.72	4.8	0.07	4.4	0.27	0.6	0.2
15			14	2.6	51.51	3.2	0.09	3.0	0.49	0.6	0.4
16	30	161	4	6.4	36.88	8.0	0.07	6.4	0.32	1.6	**0.0**
17			8	4.0	17.27	4.6	0.09	4.0	0.12	0.6	**0.0**
18			15	2.0	18.84	3.0	0.12	2.2	0.39	1.0	0.2
19	40	118	4	10.2	2283.43	12.6	0.03	12.0	0.29	2.4	1.8
20			7	$6.4^{(2)}$	4711.31	8.2	0.06	7.4	0.41	1.8	1.0
21			14	$3.4^{(2)}$	4348.36	5.0	0.08	4.0	0.51	1.6	0.6
22	40	196	4	8.8	2336.74	11.0	0.08	9.8	0.48	2.2	1.0
23			8	5.0	121.37	6.2	0.10	5.2	0.43	1.2	0.2
24			16	3.0	82.75	3.8	0.14	3.2	0.33	0.8	0.2
25	40	274	5	7.0	89.99	7.8	0.11	7.0	0.22	0.8	**0.0**
26			9	4.0	86.37	5.0	0.14	5.0	1.00	1.0	1.0
27			17	3.0	139.92	3.2	0.18	3.0	0.21	0.2	**0.0**
28	50	173	4	$12.6^{(4)}$	8994.51	16.0	0.06	14.6	0.35	3.4	2.0
29			8	$9.0^{(4)}$	10652.42	10.4	0.08	9.2	0.46	1.4	0.2
30			15	$4.8^{(3)}$	6536.58	5.4	0.12	5.4	0.54	0.6	0.6
31	50	295	5	9.0	618.57	11.0	0.10	9.8	0.55	2.0	0.8
32			9	$5.2^{(1)}$	2318.95	6.6	0.14	6.0	0.84	1.4	0.8
33			17	3.0	1009.18	4.4	0.17	4.0	1.11	1.4	1.0
34	50	418	5	9.0	294.84	9.4	0.14	9.0	0.59	0.4	**0.0**
35			9	5.0	295.96	6.0	0.20	6.0	1.19	1.0	1.0
36			18	3.0	337.71	4.0	0.28	3.2	0.67	1.0	0.2

Table 1: Test results of ILP model, GA and MS algorithms on the small scenarios.

Regarding the effectiveness of the two algorithms, as expected the results of MS are much better than those of GA. MS finds the optimal solution on eight out of 29 scenarios where the optimal solution is known while GA finds the optimal solution only one time (n° 9). Moreover, the gap value of MS is lower than or equal to one on 34 out of 36 scenarios while for GA this condition holds only 21 times.

From the gap values it is evident that the density of the graph is the main parameter affecting the effectiveness of GA and MS. In order to analyze the trend of these two algorithms on the scenarios with the same density but with an increasing number of vertices, we introduce another measure, the *AvgGap*. This measure represents the average gap values computed on the scenarios with the same number of vertices and edges. For instance, for $n = 20$ and $m = 39$, the AvgGap of MS is equal to $(0.4 + 0.6 + 0.6)/3 = 0.53$.

In Figure 5 the AvgGap of GA (in red) and MS (in blue) are plotted for the small scenarios with $d = 0.1$, $d = 0.2$ and $d = 0.3$, respectively. Here on the x-axis and y-axis the number of vertices and the AvgGap are reported, respectively.

The chart in Figure 5 shows that the two algorithms have a similar growth but the AvgGaps of MS are nearly half of the AvgGaps of GA. Again, the highest values for the AvgGap are obtained when $d = 0.1$ (Figure 5(a)). Despite that, the AvgGap of GA is always lower than two while the AvgGap of MS is always lower than 1.2. On the scenarios with $d = 0.2$ (Figure 5(b)) the growth is more regular for both algorithms, even if it is slower for MS. Indeed, the AvgGap of MS ranges from 0.40 for $n = 20$ to 0.87 for $n = 50$ while the AvgGap of GA ranges from 0.73 for $n = 20$ to 1.60 for $n = 50$. It is interesting to observe that even on these small scenarios the AvgGap between the two algorithms can be significantly different. For instance, for $n = 40$ the AvgGap of GA is three times greater than the AvgGap of MS. Finally, for $d = 0.3$ (Figure 5(c)) the results of both algorithms improve, with a maximum AvgGap equal to 1.07 for GA and 0.40 for MS. Even in this case, the AvgGap of MS is always half of AvgGap of GA and, on the scenarios with $n = 30$, the difference grows up to one. By observing the three charts of Figure 5 it is evident that our algorithms are more effective when the density grows.

Regarding the performance, both the algorithms are very fast with a negligible running time, rarely greater than one second.

Table 2 shows the results obtained by GA and MS algorithms on the large scenarios. Since the solutions of the ILP are not available for these scenarios, we compare the solution values produced by GA and MS and we compute the GAP on these values.

Despite the negligible computational times on the small scenarios, here the performance

ID	n	m	l	GA		MS		GAP
				Value	Seconds	Value	Seconds	
37	100	595	5	26.6	0.18	24.8	0.73	1.8
38			10	12.8	0.25	11.8	1.15	1.0
39			19	8.2	0.33	7.2	1.45	1.0
40	100	1090	6	17.0	0.39	15.6	1.52	1.4
41			11	10.6	0.54	10.2	2.00	0.4
42			21	7.2	0.57	6.2	2.64	1.0
43	100	1585	6	15.8	0.71	15.2	1.27	0.6
44			11	10.0	0.92	9.2	2.11	0.8
45			22	6.0	0.96	5.6	4.50	0.4
46	200	2190	6	46.0	0.77	43.6	3.47	2.4
47			12	20.0	1.09	18.6	5.05	1.4
48			23	13.8	1.39	13.2	7.62	0.6
49	200	4180	7	30.0	2.46	27.0	12.63	3.0
50			13	17.6	2.17	16.6	16.96	1.0
51			25	11.8	2.57	10.4	23.29	1.4
52	200	6170	7	26.2	2.77	26.2	25.26	0.0
53			13	16.0	3.71	15.2	25.50	0.8
54			26	9.4	4.50	9.0	45.37	0.4
55	300	4785	7	58.2	1.72	54.8	12.32	3.4
56			13	27.8	2.23	26.4	19.26	1.4
57			25	19.6	2.97	18.0	28.99	1.6
58	300	9270	7	47.0	4.81	41.8	50.44	5.2
59			14	22.2	7.14	22.0	82.13	0.2
60			27	15.0	8.98	14.2	96.14	0.8
61	300	13755	7	39.4	11.58	39.2	125.62	0.2
62			14	21.4	18.83	21.0	203.72	0.4
63			28	12.4	18.63	11.8	177.91	0.6
64	400	8380	7	86.0	4.37	79.2	43.33	6.8
65			14	33.0	6.51	31.8	69.59	1.2
66			27	25.0	8.78	22.8	97.05	2.2
67	400	16360	7	82.0	16.29	60.6	176.52	21.4
68			14	29.0	25.31	28.4	279.07	0.6
69			28	18.4	27.01	17.0	306.68	1.4
70	400	24340	8	46.6	43.30	45.2	247.19	1.4
71			15	26.4	77.16	26.0	857.59	0.4
72			30	15.2	61.59	14.4	718.85	0.8

Table 2: Test results of GA and MS on the large scenarios.

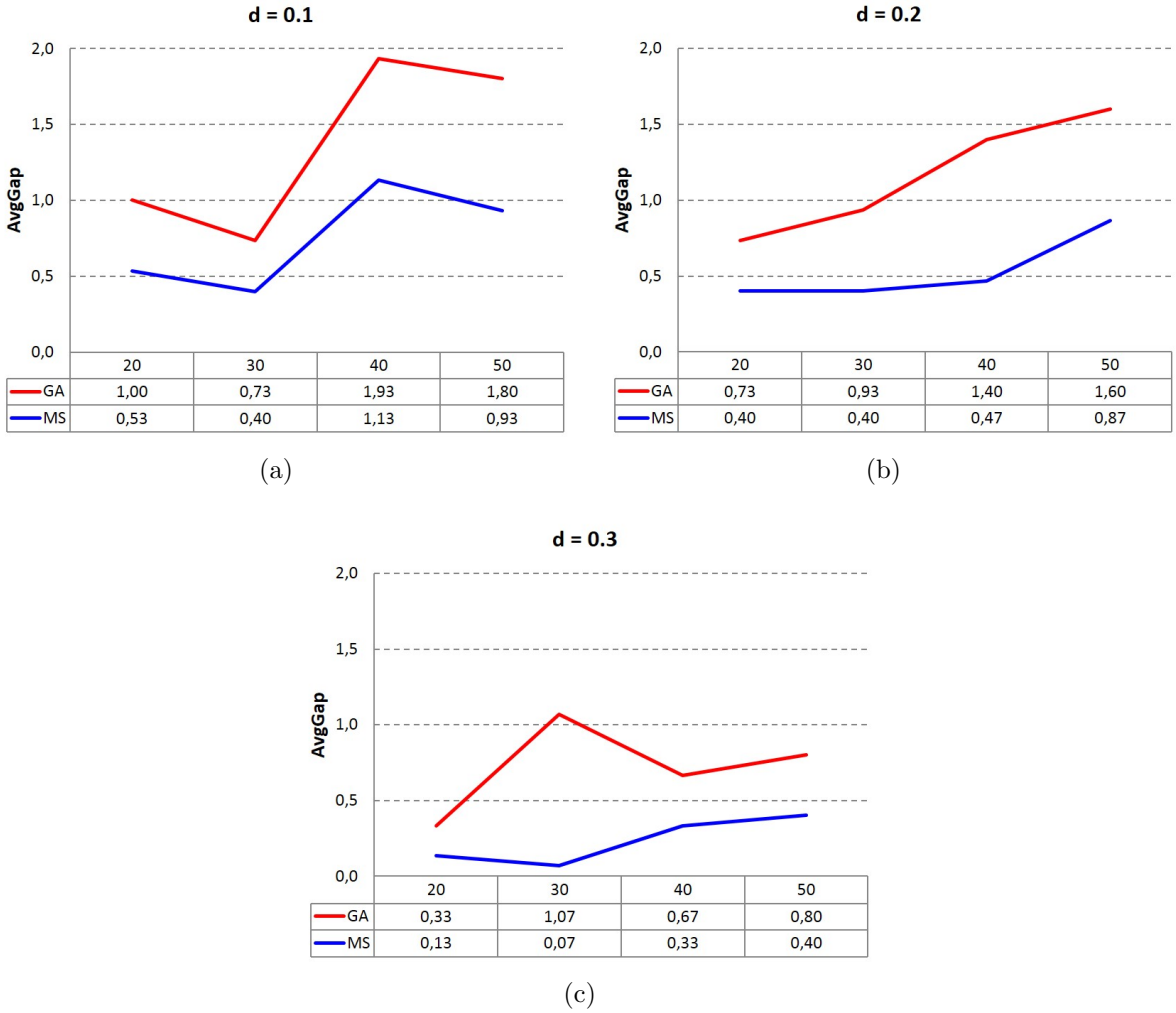

Figure 5: *The AvgGap of GA and MS algorithms on the larger scenarios.*

of the two algorithms are much different. In particular, GA is one order of magnitude faster than MS, often its running time is lower than one minute and, in the worst case ($n°$ 71) it requires 77 seconds. More computational time is required by MS that in the worst case ($n°$ 71) spends 857 seconds ($n°$ 71). Anyway, most of the scenarios are solved by MS in less than five minutes. It is clear that the significant performance difference on the large scenarios is due to the fact that MS invokes several times the GA algorithm. The trend of the running times for both algorithms shows that the performance are mainly affected by the density of the scenarios, as the computation time increases with density.

As expected, MS algorithm is slower than GA but it is more effective as shown by values of GAP column. On all the scenarios, the solutions found by MS are always better than the

solutions found by GA. More in details, on 20 out of 36 scenarios, the GAP value is greater than or equal to one and in six cases this GAP is greater than two. We observe also a GAP equal to 21.4 on the scenario n° 67 where the solution of GA is very poor.

It is interesting to observe that the lowest GAP values occur on the scenarios with $d = 0.3$ on which the two algorithms require more computational time. Our conjecture is that, since these scenarios contains more edges, there are more local minimums with a good solution value. As a consequence, when GA is trapped into a local minimum the value found is good and the GAP from MS low. Instead, the situation changes on the scenarios with fewer edges ($d = 0.1$ and $d = 0.2$) where the number of good local minimums is lower. In this case, there are more chances that GA is trapped into a poor local minimum producing a solution value far from the solution value of MS.

5. Conclusion

In this paper we propose a mathematical formulation and a greedy algorithm for the Rainbow Spanning Forest Problem. Moreover we embedded the greedy algorithm in a multi-start schema to improve the quality of the solutions found. The computational results show that the proposed formulation solves, within the time limit, the 80% of the instances with at most 50 nodes. We observed that the running time of the model decreases as the density increases. Indeed, all the scenarios with density equal to 0.3 are solved in less than 340 seconds while on the scenarios with density 0.1 the time limit is reaches more times. Regarding the MS algorithm, it results very effective, on the small instances, with a gap from the best known solution that is greater than 1% only two times. The negligible running time of MS, on these instances, make it particularly suitable to be embedded in exact approaches where, often, is required to quickly found good solutions. On the large scenarios the MS is slower than GA but its solutions are much better. This justify the idea to use a multi-start approach. A possible direction for future works could be to apply a Carousel Schema [10] to our GA or to replace the MS with more sophisticated metaheuristics like a Tabu Search [5], [12].

Compliance with Ethical Standards:

Conflict of Interest: The first author declares that he has no conflict of interest. The second author declares that he has no conflict of interest. The third author declares that he has no conflict of interest. The fourth author declares that he has no conflict of interest.

Ethical approval: This article does not contain any studies with human participants or

animals performed by any of the authors.

References

[1] S. Akbari and A. Alipour. Multicolored trees in complete graphs. *Journal of Graph Theory*, 54(3):221–232, 2007.

[2] Eric Andrews, Chira Lumduanhom, Elliot Laforge, and Ping Zhang. On proper-path colorings in graphs. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 97:189–207, 2016.

[3] R. A. Brualdi and S. Hollingsworth. Multicolored forests in complete bipartite graphs. *Discrete Mathematics*, 240:239–245, 2001.

[4] R.D. Carr, S. Doddi, G. Konjedov, and M. Marathe. On the red-blue set cover problem. In *11th ACN-SIAM Symposium on Discrete Algorithms*, pages 345–353, 2000.

[5] F. Carrabs, C. Cerrone, and R. Cerulli. A tabu search approach for the circle packing problem. In *2014 17th International Conference on Network-Based Information Systems*, pages 165–171. IEEE, 2014.

[6] F. Carrabs, C. Cerrone, R. Cerulli, and S. Silvestri. On the complexity of rainbow spanning forest problem. Technical Report 14922, Department od Mathematics, University of Salerno, March 2016.

[7] F. Carrabs, R. Cerulli, and P. Dell'Olmo. A mathematical programming approach for the maximum labeled clique problem. *Procedia-Social and Behavioral Sciences*, 108:69–78, 2014.

[8] F. Carrabs, R. Cerulli, and M. Gentili. The labeled maximum matching problem. *Computers & Operations Research*, 36:1859–1871, 2009.

[9] J. M. Carraher, S. G. Hartke, and P. Horn. Edge-disjoint rainbow spanning trees in complete graphs. *European Journal of Combinatorics*, 57:71–84, 2016.

[10] C. Cerrone, R. Cerull, and B. Golden. Carousel greedy: a generalized greedy algorithm with applications in optimization. *University of Maryland (submitted for publication)*.

[11] C. Cerrone, R. Cerulli, and M. Gaudioso. Omega one multi ethnic genetic approach. *Optimization Letters*, 10(2):309–324, 2016.

[12] C Cerrone, R Cerulli, and M Gentili. Vehicle-id sensor location for route flow recognition: Models and algorithms. *European Journal of Operational Research*, 247(2):618–629, 2015.

[13] R. Cerulli, P. Dell'Olmo, M. Gentili, and A. Raiconi. Heuristic approaches for the minimum labelling hamiltonian cycle problem. *Electronic notes in discrete mathematics*, 25:131–138, 2006.

[14] R. Cerulli, A. Fink, M. Gentili, and S. Voß. Metaheuristics comparison for the minimum labelling spanning tree problem. In *The Next Wave in Computing, Optimization, and Decision Technologies*, pages 93–106. Springer, 2005.

[15] R.S. Chang and S.J. Leu. The minimum labeling spanning trees. *Information Processing Letters*, 63:277–282, 1997.

[16] Y. Chen, N. Cornick, A. O. Hall, R. Shajpal, J. Silberholz, I. Yahav, and B. Golden. Comparison of heuristics for solving the gmlst problem. In *Telecommunications Modeling, Policy, and Technology*, pages 191–217. Springer, 2008.

[17] S. Consoli, K. Darby-Dowman, N. Mladenović, and J. A. Moreno-Pérez. Variable neighbourhood search for the minimum labelling steiner tree problem. *Annals of Operations Research*, 172:71–96, 2009.

[18] S. Consoli, J. A. Moreno-Pérez, K. Darby-Dowman, and N. Mladenović. Discrete particle swarm optimization for the minimum labelling steiner tree problem. *Natural Computing*, 9:29–46, 2010.

[19] G. B. Dantzig, D. R. Fulkerson, and S. M. Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2:393–410, 1954.

[20] M. Fischetti, J. J. Salazar González, and P. Toth. Experiments with a multi-commodity formulation for the symmetric capacitated vehicle routing problem. In *Proceedings of the 3rd Meeting of the EURO Working Group on Transportation*, pages 169–173, 1995.

[21] N. Jozefowiez, G. Laporte, and F. Semet. A branch-and-cut algorithm for the minimum labeling hamiltonian cycle problem and two variants. *Computers & Operations Research*, 38:1534–1542, 2011.

[22] S. Krumke and H. Wirth. On the minimum label spanning tree problem. *Information Processing Letters*, 66(2):81–85, 1998.

[23] X. Li and X.Y. Zhang. On the minimum monochromatic or multicolored subgraph partition problems. *Theoretical Computer Science*, 385:1–10, 2007.

[24] S. Silvestri, G. Laporte, and R. Cerulli. The rainbow cycle cover problem. *Networks*, 2016.

[25] K Suzuki. A necessary and sufficient condition for the existence of a heterochromatic spanning tree in a graph. *Graphs and Combinatorics*, 22:261–269, 2006.

[26] M. Szachniuk, M. De Cola, G. Felici, and J. Błażewicz. The orderly colored longest path problem–a survey of applications and new algorithms. *RAIRO-Operations Research*, 48(1):25–51, 2014.

[27] M. Szachniuk, M. De Cola, G Felici, D De Werra, and J. Błażewicz. Optimal pathway reconstruction on 3d nmr maps. *Discrete Applied Mathematics*, 182:134–149, 2015.

[28] Y. Xiong, B. Golden, and E. Wasil. The colorful traveling salesman problem. In *Extending the Horizons: Advances in Computing, Optimization, and Decision Technologies*, pages 115–123. Springer, 2007.

[29] Y. Xiongm, B. Golden, E. Wasil, and S. Chen. The label-constrained minimum spanning tree problem. In *Telecommunications Modeling, Policy, and Technology*, pages 39–58. Springer, 2008.