Available online at www.sciencedirect.com

ScienceDirect

Procedia - Social and Behavioral Sciences 108 (2014) 153 – 163

Procedia
Social and Behavioral Sciences

The *k*-labeled Spanning Forest Problem

R. Cerulli[a,1], A. Fink[b,2], M. Gentili[a,1], A. Raiconi[a,1,*]

[a]*Department of Mathematics, University of Salerno*
[b]*Faculty of Economics and Social Sciences, Helmut-Schmidt-University*

Abstract

In the *k*-labeled Spanning Forest Problem (kLSF), given a graph G with a label (color) assigned to each edge, and an integer positive value k_{max} we look for the minimum number of connected components that can be obtained by using at most k_{max} different labels. The problem is strictly related to the Minimum Labelling Spanning Tree Problem (MLST), since a spanning tree of the graph (i.e. a single connected component) would obviously be an optimal solution for the kLSF, if it can be obtained without violating the bound on k_{max}. In this work we present heuristic and exact approaches to solve this new problem.

Keywords: Spanning Forest, Edge-Labeled Graphs, Metaheuristics

1. Introduction

In this work, we introduce a new combinatorial optimization problem, namely the *k*-Labeled Spanning Forest Problem, and present some heuristics as well as an exact method for its resolution. The *k*-labeled Spanning Forest (kLSF) Problem belongs to a recently studied class of problems, defined on edge-labeled graphs; that is, for each edge of the graph, a different label is assigned to it. Such a type of graphs may be used to model many real-world situations in which we want to distinguish between different "types" of connections; for example, in telecommunication networks, there can be different types of communications media (such as optical fiber, coaxial cable, telephone line) different companies to which the connections belong, or different transmission frequencies. It is then clear that we may be interested in optimizing this factor when we are considering the edges to be included in the solution of the problem that we are going to solve.

In the kLSF Problem, we are interested in finding the minimum number of connected components that can be found in an undirected edge-labeled graph when we have a constraint on the maximum number k_{max} of labels that can be used. This type of constraint is meaningful with respect to many real-world situations in which there is a limit that cannot be violated. For example, with the emergence of wireless communication technologies (such as WiFi and cellular networks), in the last years we have been surrounded by electromagnetic fields (EMFs) of many frequencies. The overall effect of this exposition on human beings

*Corresponding author
[1]Email: {raffaele,mgentili,araiconi}@unisa.it
[2]Email: andreas.fink@hsu-hamburg.de

doi:10.1016/j.sbspro.2013.12.828

and environment has risen concern in the scientific community (see for example [1]). In this scenario it could be desirable to build a telecommunication network with hard constraints on the maximum number of used transmission frequencies. In the case in which such constraints would not allow to obtain a single connected network, a terminal hub for each connected component could be used to connect it with the others using older, less effective technology (such as cable networks). Minimizing the number of connected components would then mean to minimize the impact due to the performances or cost of older technology in the context of a sustainable wireless network.

The kLSF generalizes a well-known problem in the context of edge-labeled graphs, that is, the problem of finding the Spanning Tree of the graph that uses the minimum number of labels (Minimum Labeling Spanning Tree, or MLST). In fact, a spanning tree would represent the optimum if we can find one that uses at most the number of labels that we allow in a solution (note that an optimal single-component solution for the kLSF may contain cycles, but then we could arbitrarily break each of them in polynomial time).

The MLST was introduced by Chang and Leu [2]. They proved it to be $\mathcal{NP}$-complete and provided an heuristic, the Maximum Vertex Coverage Algorithm (MVCA), as well as an exact A* algorithm. Krumke and Wirth [3] proposed an improved version of MVCA. They also proved an approximation ratio of $(1 + 2 \log n)$ for the algorithm with respect to the optimal solution, where n is the total number of nodes. Wan et al. [4] proved a tighter bound for the approximation ratio of MVCA, that is, $1 + \log(n - 1))$ if $n > 1$. Xiong et al. [5] furtherly improved the bound by Wan et al., showing that the worst-case bound of MVCA is the bth-harmonic number $H_b = \sum_{i=1}^{b} 1/i$, where b is the bound on the maximum frequency of any label in the graph. This is an improvement on the previous bound because $H_b < (1 + log(n - 1))$ and $b \leq (n - 1)$. They also proved this new bound to be tight

In [6], the authors proved that a mixed integer linear relaxation of the problem always has an optimal solution value that is equal to the one of the original problem.

Several metaheuristic approaches have been proposed for MLST. Xiong et al. [7] presented a Genetic Algorithm (GA) that generally outperforms MVCA. Cerulli et al. [8] applied different metaheuristics to the problem (Reactive Tabu Search, Simulated Annealing, Variable Neighbourhood Search, Pilot Method). The Pilot Method obtains the best results for most of the instances, but generally requires a larger amount of running time. Xiong et al. [9] implemented both a simplified version of the Pilot Method and a Modified Genetic Algorithm (MGA). Consoli et al. [10] presented a Greedy Randomized Adaptive Search (GRASP) and a Variable Neighborhood Search (VNS) algorithm.

Other addressed problems in the same field include the Minimum Labelling Steiner Problem [11], [12], [13]), the Minimum Labelling Generalized Forest [14], the Colorful Traveling Salesman Problem ([15], [16], [17], [18]), the Generalized Minimum Label Spanning Tree Problem [19], the Label-Constrained Minimum Spanning Tree Problem [20] and the Labeled Maximum Matching Problem [21].

The remainder of the paper is organized as follows. In section 2 the problem is formally defined and it is shown to be $\mathcal{NP}$-complete. Heuristic and metaheuristic appoaches are discussed in section 3. A Branch and Bound exact algorithm is presented in section 4. The results of an experimental phase conducted on our algorithms is presented in section 5. Section 6 contains conclusions and some final remarks.

2. Problem Description

Let $G = (V, E)$ be an undirected edge-labeled graph with V being the set of nodes and E the set of edges. Let $L = \{l_1, l_2, \ldots, l_c\}$ the set of all the labels. We denote with $L(e) \in L$ the label assigned to each edge $e \in E$. Any subgraph H of G has associated the set of its labels $L(H)$; analogously, let $E(L')$ be the subset of edges with a label belonging to $L' \subseteq L$. Given an integer parameter k_{max} such that $1 \leq k_{max} \leq |L|$, we look for a spanning forest of G, $H = \{H_1, ..., H_w\}$ such that each H_i is connected, the cardinality of H is minimized and the cardinality of the set of its labels $L(H) = \bigcup_{i=1...w}(L(H_i))$ is at most k_{max}. In the following, let $n(H)$ be the number of connected components for a given subgraph H of G.

Now, let us consider the decision versions of both MLST and kLSF. Let w be a positive integer:
MLST. Given $k_{max} \leq |L|$, is there a connected subgraph H of G containing edges with up to k_{max} different labels?

kLSF. Given $k_{max} \leq |L|$ and $w \leq |V|$, is there a subgraph H of G containing edges with up to k_{max} different labels, and such that $n(H) \leq w$?

kLSF belongs to the $\mathcal{NP}$ class. Indeed, given a feasible solution, it is possible to verify in polynomial time whether it contains no more than w connected components. It can be noticed that $MLST$ is a special case of kLSF with $w = 1$. Therefore, since $MLST$ is $\mathcal{NP}$-Complete, kLSF is $\mathcal{NP}$-Complete as well.

3. Heuristic Approaches

3.1. Modified MVCA

The first heuristic we developed is a variant of the Maximum Vertex Covering Algorithm for the Minimum Labelling Spanning Tree problem, presented in [2] by Chang and Leu and enhanced by Krumke and Wirth in [3]. Recalling that the Minimum Labeling Spanning Tree Problem consists in finding the spanning tree with the minimum number of different labels, this heuristic adds iteratively labels reducing the number of connected components by as many as possible, until one connected component is left. Our variant works in quite the same way, with the major difference that the algorithm ends if the k_{max} limit is reached. More formally, given an initially empty set of labels C, the algorithm iteratively selects label $l \in L \backslash C$ such that subgraph $H = (V, E(C \bigcup \{l\}))$ has the minimum number of connected components and adds it to C, until either H is connected or $|C| = k_{max}$. The algorithm is highly efficient in terms of computational time. For this reason we decided to develop more complex metaheuristic approaches, where we use Modified MVCA as a subroutine.

3.2. Pilot Method

The Pilot Method [22] is a metaheuristic based on the idea of using a basic heuristic algorithm (such as the Modified MVCA) as a look-ahead mechanism by means of iterated executions. In each Pilot Method iteration we evaluate every local choice available from a so-called *master solution* (i.e, we move to all the neighbors of this solution), we run our basic heuristic on each of these choices and record the best result obtained; we will move to its corresponding move for the next Pilot Method iteration.

We use the Pilot Method as a constructive scheme that starts from an empty solution (i.e., no selected labels) and operates on a solution space made up of feasible solutions (i.e., no more than k_{max} selected labels). Our neighborhood structure is based on the addition of labels (the neighbors of a given solution x will be all the solutions that have all the labels of x selected, plus a new one). Therefore, at each step of our algorithm we will add every unused label to our master solution, and perform the Modified MVCA for each of this choices, moving for the next iteration on the neighbor that led to the best solution, until either k_{max} labels have been selected or a single component has been obtained.

3.3. Local Search Based Metaheuristics

While the algorithms described in Sections 3.1 and 3.2 behave in a constructive manner (that is, they start from an empty solution and add new labels to it), the Tabu Search and Simulated Annealing procedures (described in Sections 3.3.1 and 3.3.2, respectively) start from an input solution and iteratively try to improve it. For this reason, a new type of neighborhood structure is required.

The solution space for our Tabu Search and Simulated Annealing procedures is composed of solutions where k_{max} labels are selected. Indeed, these two procedures are initialized using Modified MVCA, which always returns a solution composed of k_{max} labels, unless it contains a single component that does not need to be improved.

Moreover, it is obvious that an optimal solution is always reachable when k_{max} labels are selected, since adding new edges to an optimal solution with $k < k_{max}$ labels would preserve the same number of components.

Let $l_1, ..., l_{kmax}$ be the chosen labels in our current solution x, and $h_1, ..., h_{k'}$ be the unchosen ones (please note that $k' = |L| - k_{max}$). For each $i = 1, ..., k_{max}$ and for each $j = 1, ..., k'$ there will be a neighbor x_{ij} of x, whose list of selected labels is $l_1, , l_{(i-1)}, h_j, l_{(i+1)}, , l_{kmax}$ (that is, h_j substitutes l_i).

Therefore, each neighborhood size is $k_{max}(|L| - k_{max})$.

3.3.1. Reactive Tabu Search

The basic paradigm of Tabu Search is to use information about the search history to guide local search approaches to overcome local optimality (see [23] for a survey on Tabu Search). In general, this is done by a dynamic transformation of the local neighborhood. Based on some sort of memory certain moves may be forbidden, we say they are flagged as tabu (and appropriate move attributes such as a certain index indicating a specific label put into a list, the so-called tabu list). The search may imply acceptance deteriorating moves when no improving moves exist or all improving moves of the current neighborhood are flagged as tabu. At each iteration a best admissible neighbor may be selected. A neighbor, respectively a corresponding move, is called admissible, if it is not tabu.

Reactive Tabu Search (RTS) aims at the automatic adaptation of the tabu list length [24]: if the tabu memory indicates that the search is revisiting formerly evaluated solutions, then the tabu list size is increased. A possible specification can be described as follows: starting with a tabu list length ls of 1 it is increased to $\min\{\max\{ls + 2, ls \times 1.2\}, b_u\}$ every time a solution has been repeated, taking into account an appropriate upper bound b_u (to guarantee at least one admissible move). If there has been no repetition for some iterations, we decrease it to $\max\{\min\{ls - 2, ls/1.2\}, 1\}$. To accomplish the detection of a repetition of a solution, one may apply a trajectory based memory using hash codes.

For RTS, it is appropriate to include means for diversifying moves whenever the tabu memory indicates that we are trapped in a certain region of the search space. As a trigger mechanism one may use, e.g., the combination of at least three solutions each having been evaluated three times. A very simple escape strategy is to perform randomly a number of moves (depending on the average of the number of iterations between solution repetitions).

The main termination criterion is given by the individuation of a feasible solution composed of a single connected component; alternatively, the procedure ends when a given time limit is reached.

3.3.2. Simulated Annealing

Like the Tabu Search, Simulated Annealing (SA) extends basic local search. The basic concept of SA may be described as follows: starting from an initial solution, a candidate move in the current neighborhood is accepted if it leads to a solution with a better objective function value than the current solution, otherwise the move is accepted with a probability that depends on the deterioration Δ of the objective function value. The acceptance probability is computed according to the Boltzmann function as $e^{-\Delta/T}$, using a temperature T as control parameter. Various authors describe robust realizations of this general SA concept. Following [25], the value of T is initially high, which allows many worse moves to be accepted, and is gradually reduced through multiplication by a parameter $coolingFactor$ according to a geometric cooling schedule. Given a parameter $sizeFactor$, $sizeFactor \cdot n_{size}$ candidate moves are tested (n_{size} denotes the neighborhood size) before the temperature is reduced. The starting temperature is determined as follows: Given a parameter $initialAcceptanceFraction$ and based on an abbreviated trial run, the starting temperature is set so that the fraction of accepted moves is approximately $initialAcceptanceFraction$. A further parameter, $frozenAcceptanceFraction$ is used to decide whether the annealing process is frozen and should be terminated. Every time a temperature is completed with less than $frozenAcceptanceFraction$ of the candidate moves accepted, a counter is increased by one, while this counter is re-set to 0 each time a new best solution has been obtained. The whole procedure is terminated when either a solution composed of a single component is found or the counter reaches a parameter $frozenLimit$.

3.4. Genetic Algorithm

Genetic Algorithms (GAs) rely on analogies to the natural processes of evolution and natural selection of individuals belonging to a population. Each individual represents an element of the solution space, with the hope to evolve (after a certain number of *generations*) to a near-optimal solution. The elements that must be provided in order to implement a GA are:

1. An abstract representation for each member of the population (generally called its *chromosome*); in our algorithm the chromosome is simply the list of labels selected in the solution. As for the local search based metaheuristics, we only consider solutions where exactly k_{max} labels are selected.

2. A function to evaluate each individual (the *fitness* function); our fitness function is given by the number of connected components.
3. Mechanisms to derive new solutions, either from two "parent" solutions (*crossover*) or from a perturbation of a single individual (*mutation*).
   Given two distinct feasible solutions S' and S'', each composed of k_{max} labels, the crossover operator in our proposed GA starts by considering the union U of their labels (such a set will contain at least $(k_{max} + 1)$ elements, since the two solutions are different). At this point, the algorithm creates a new solution by applying the Modified MVCA algorithm on this set of labels (that is, only labels belonging to U will be considered at each step of the procedure), until k_{max} of them are selected.
   The mutation mechanism, instead, operates by adding a random unselected label to a given feasible solution S. At this point, an unfeasible $(k_{max} + 1)$-labeled individual has been created, and therefore one label must be removed. The algorithm performs this task by removing the label that gives the best value in terms of fitness function, i.e., such that the increase in the number of connected components is minimized. Only the original labels of S are considered for removal (i.e. we will not obtain a solution identical to S), since the aim of the mutation operator is to diversify the population. Finally, a set of rules that describe how the above presented mechanisms are applied and how new individuals (new *generations*) substitute the older ones must be defined.
   Our proposed GA is derived from the one developed by Xiong, Golden and Wasil in [7]. This procedure is designed to be extremely simple. Indeed, while other genetic procedures rely on a series of parameters and random choices to determine events such as the verification of a mutation or the selection of individuals that will generate a new solution, this scheme depends on a single user-defined parameter, that represents both the population size and the number of iterations of the procedure. Let p be the value of this parameter; moreover, let $\{S_0, \ldots, S_{(p-1)}\}$ be the initial population.
   The Genetic Algorithm creates the generic i-th population from the $(i-1)$-th as follows:

   for each j from 0 to $(p-1)$, do:

   - T_j=crossover(S_j,$S_{((j+i) \mod p)}$)
   - T_j=mutation(T_j)
   - S_j= the solution with the better fitness value between T_j and S_j

   Our GA follows all the above described steps. The initial population is created randomly. The solution with the best fitness value created during the execution is always stored as incumbent optimum. The procedure ends before the last generation if a solution composed of a single component is found.

4. Branch and Bound Approach

We developed an exhaustive search algorithm for the k-Labeled Spanning Forest problem.

Some fundamental notation used in this section is now introduced. During the execution of the algorithm, let $L_{opt} \subseteq L$ be the subset of labels corresponding to the best solution found so far, that is, such that $n(H_{opt})$ is the minimum value found, where $H_{opt} = (V, E(L_{opt}))$. L_{opt} is initialized with the empty set (corresponding to the trivial upper bound $n(H_{opt}) = |V|$). Any given node of the Branch and Bound search tree represents a solution $H_{curr} = (V, E(L_{curr}))$, $L_{curr} \subseteq L$, where L_{curr} represents a decision about which labels to include from a subset of labels $\{l_1, \ldots, l_d\}$, $d \leq |L|$. If $d < |L|$, labels $l_{(d+1)}$,...,$l_{|L|}$ could be added to L_{curr} in the solutions represented by the nodes belonging to the subtree rooted at the current node; such labels are called *undecided*.

A proposition is now introduced and proven. The proposition gives an upper bound on the improvement in terms of objective function that can be obtained by adding a subset of undecided labels M' to a given solution, using the contributions provided by the individual labels of M'. Such results are used by one of the pruning operators of the Branch and Bound algorithm, as will be discussed next.

Proposition 1. *Let $G = (V, E)$ be an input labeled graph, and let L be the set of its associated labels. Let M and M' be two nonempty subsets of L such that $M \cap M' = \emptyset$ and $|M'| \geq 1$. Let $H = (V, E(M))$, $H' = (V, E(M) \cup E(M'))$ and $H_l = (V, E(M) \cup E(\{l\}))$ for a given label $l \in L$. The following inequality holds:*

$$n(H) - n(H') \leq \sum_{l \in M'} (n(H) - n(H_l)) \tag{1}$$

Proof. Proposition 1 is now proved by induction on the cardinality of M'. Suppose that $|M'| = 1$; in this case, M' is composed by a single label l, and both the left-hand side and the right-hand side of Inequality (1) assume the same value $n(H) - n(H_l)$. Therefore, the proposition is proved for the base of induction.

Now, consider the case $|M'| > 1$. Let l' be an arbitrary element of M', M''=$M' \setminus \{l'\}$, and $H'' = (V, E(M) \cup E(M''))$. By induction, suppose that $n(H) - n(H'') \leq \sum_{l \in M''} (n(H) - n(H_l))$. Since adding edges can not disconnect previously connected components, then $n(H'') - n(H') \geq 0$. The proof is now carried on distinctly for the two cases $n(H'') - n(H') = 0$ and $n(H'') - n(H') > 0$.

If $n(H'') - n(H') = 0$ (that is, adding the edges labeled with l' to H'' does not decrease the number of components), then obviously $n(H) - n(H'') = n(H) - n(H')$. Moreover, for the same reasons expressed above, it can also be noted that $n(H) - n(H_{l'}) \geq 0$. Therefore, it follows that $n(H) - n(H') \leq \sum_{l \in M''} (n(H) - n(H_l)) + (n(H) - n(H_{l'}))$, and Proposition 1 is proved.

Otherwise, if $n(H'') - n(H') = \delta > 0$, then δ distinct couples of disconnected components in H'' get connected by one or more edges when label l' is added. For each couple of newly connected components, select randomly one of the edges labeled with l' connecting them. For each of such edges, it can be noted that its endpoints belong to different components in H; indeed, they belong to different components in H'', and as previously noted, components can not be disconnected by adding new edges. It follows that $n(H) - n(H_{l'}) \geq \delta$, and therefore $n(H'') - n(H') \leq n(H) - n(H_{l'})$. It can then be noted that $n(H) - n(H'') + n(H'') - n(H') \leq \sum_{l \in M''} (n(H) - n(H_l)) + (n(H) - n(H_{l'}))$, which again proves Proposition 1. □

Our Branch and Bound algorithm can now be introduced.

As previously introduced, each solution explored can be considered as a node of a binary tree, and each level of the tree represents a binary choice on the selection of a specific label. That is, the root of the tree represents the starting partial solution where no choice has been made. The subtree rooted in the left child of the root will contain all the solutions where the first label has been selected (and therefore $l_1 \in L_{curr}$ in their representation), while in the one rooted in the right child there will be all the ones in which this label has been excluded (i.e. $l_1 \notin L_{curr}$). Similarly, for instance, the subtree rooted in the left child of the left child of the root will contain all the solutions in which the first two labels have been selected ($\{l_1, l_2\} \in L_{curr}$). This search tree is explored with a steepest descent approach. Note that the value of parameter d for any node depends on its level in the tree, more in detail $d = 0$ for the root node, $d = 1$ for its two child nodes, and so on.

We define an ordering on the labels $l_1, \ldots, l_{|L|}$ based on their frequencies in the graph in non-increasing order, and visit them in the tree according to it. Since the labels with higher frequencies are more likely to be part of an optimal solution, this choice increases the chance of finding such a solution earlier. Moreover, one of the pruning techniques, which measures the potential contribution of the undecided labels, is more effective when most or all the remaining undecided labels have few edges.

The following pseudocode **BranchAndBound**(H_{curr}, d, H_{opt}) describes the behavior of the procedure for any node of the search tree:

0. Input: $H_{curr} = (V, E(L_{curr}))$, $d \leq |L|$, $H_{opt} = (V, E(L_{opt}))$
1. if $|L_{curr}| = k_{max}$
   (a) if $n(H_{curr}) < n(H_{opt})$
      i. $H_{opt} \leftarrow H_{curr}$
      ii. if $n(H_{opt}) = 1$ return H_{opt}, end algorithm
   (b) return H_{opt}
2. if $(|L| - d < k_{max} - |L_{curr}|)$ return H_{opt}

3. evaluate *max_reduce*
4. if $(n(H_{curr}) - max\_reduce) \geq n(H_{opt})$ return H_{opt}
5. $L_{curr} \leftarrow L_{curr} \cup \{l_{(d+1)}\}$
6. $H_{opt} \leftarrow BranchAndBound(H_{curr}, (d+1), H_{opt})$
7. $L_{curr} \leftarrow L_{curr} \setminus \{l_{(d+1)}\}$
8. $H_{opt} \leftarrow BranchAndBound(H_{curr}, (d+1), H_{opt})$
9. return H_{opt}

The procedure takes as input the subgraph representing the current solution H_{curr}, as well as its related d value and the subgraph corresponding to the incumbent optimal solution H_{opt}.

As already discussed, an optimal solution will be in one or more solutions where k_{max} labels are selected. Therefore, if such a solution has been reached (as checked in Line 1), $n(H_{curr})$ is evaluated and compared with $n(H_{opt})$ to check if a better solution was found and therefore if H_{opt} needs to be updated. If that is the case, and the new optimum has a single connected component, the whole procedure ends and the new H_{opt} is returned (see Lines 1(a)). In the case $|L_{curr}| = k_{max}$ and $n(H_{opt}) > 1$, regardless of the value of $n(H_{curr})$, since adding new labels would lead to an infeasible solution, the procedure backtracks to the parent of the current node, discarding its subtree and returning the current incumbent optimum (Line 1(b)). Otherwise, if $|L_{curr}| < k_{max}$, in order to check whether the current subtree can be discarded, two different pruning techniques are used. More in detail:

1. If $|L| - d < k_{max} - |L_{curr}|$, as checked in Line 2, we cannot obtain a solution with k_{max} labels in the subtree rooted at the current node (note that the left-hand side is the current number of undecided labels). Since we are interested in such solutions, we can avoid visiting this region, and therefore the procedure backtracks.
2. Otherwise, consider the current undecided labels, $l_{(d+1)}, , l_{|L|}$. For each of them, let $H_{(d+i)} = (V, E(L_{curr}) \cup E(\{l_{(d+i)}\}))$. Let us assume without loss of generality that $n(H_{(d+1)}) \leq n(H_{(d+2)}) \leq \ldots \leq n(H_{|L|})$, and let $max\_reduce = \sum_{i=1}^{(k_{max}-|L_{curr}|)} n(H_{curr}) - n(H_{(d+i)})$. By Proposition 1, this is an upper bound on the number of components that can be reduced by adding to the current solution any subset of $k_{max} - |L_{curr}|$ undecided labels. Therefore if $n(H_{curr}) - max\_reduce \geq n(H_{opt})$, (Line 4), no solution containing k_{max} labels is better than the incumbent optimum, and again the subtree rooted at the current node can be discarded. Implementation details regarding the evaluation of *max_reduce* are given in Section 4.1.

If the current subtree was not pruned, the procedure is recursively invoked on the two children of the current node (Lines 6 and 8, respectively), where the decision on including $l_{(d+1)}$ is taken. Note that, as previously said, the child containing the new label is explored first.

Finally, the incumbent optimum is either returned to the parent of the current node, or returned as final solution when the procedure backtracks to the root node.

4.1. Computing max_reduce

For efficiency matters, instead of evaluating $n(H_{(d+i)})$ for each undecided label with a different graph visit, we implicitly evaluate $n(H_{curr}) - n(H_{(d+i)})$ by considering the connected components of the current solution and the edges with label $l_{(d+i)}$. For each of these edges, the procedure checks if they connect different components; some additional data structures (the variables **parent** and **level** defined on each connected component) and the auxiliary procedure **root** are used to keep track of components that have already been merged by some other edges.

0. Input: $H_1, \ldots, H_{n(H_{curr})}$ connected components of H_{curr}, undecided label $l_{(d+i)}$
1. $\forall i = 1, ..., n(H_{curr})$, let $parent_i \leftarrow i$, $level_i \leftarrow 0$
2. let $(n(H_{curr}) - n(H_{(d+i)}) \leftarrow 0$
3. $\forall (i, j) \in E(\{l_{(d+1)}\})$, such that i and j belong to different components H_a and H_b, if $root(a) \neq root(b)$, then
   (a) if $level_{root(a)} > level_{root(b)}$, then $parent_{root(b)} \leftarrow root(a)$

(b) else
  i. if $level_{root(a)} = level_{root(b)}$ then $level_{root(b)} \leftarrow level_{root(b)} + 1$
  ii. $parent_{root(a)} \leftarrow root(b)$
(c) $(n(H_{curr}) - n(H_{(d+i)}) \leftarrow (n(H_{curr}) - n(H_{(d+i)}) + 1$
4. return $n(H_{curr}) - n(H_{(d+i)})$

The pseudocode of the procedure **root** is the following:

0. Input: $i \in \{1, ..., n(H_{curr})\}$
1. $current \leftarrow i$
2. while $parent_{current} \neq current$, do
   (a) $current \leftarrow parent_{current}$
3. return $current$

The value returned by **root**, accessible through **parent** pointers, makes sure that for each group of merged components a single element is considered. The **level** values are used to determine this single element. Reaching the root element for a given component takes $O(log(n(H_{curr})))$ time. We show an example of the procedure in Figure 1.

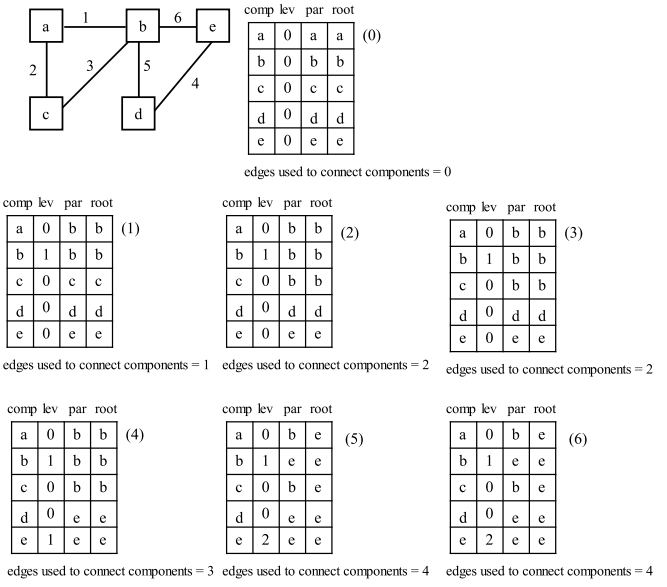

Fig. 1. Computing how many components can be merged by using a given new label. The five square nodes represent the connected components of a given partial solution. Only the edges of the considered new label linking different components are shown. The number assigned to each edge represents the order in which they will be visited by the procedure. Table (0) shows the initialization value for each variable of each component and the initial values returned by the root procedure. The other tables show the updated values after that the related edge has been visited. Note that no changes occur when the third and the sixth edge are considered, since they connect components with the same root (i.e., already connected before).

5. Experimental results

In this section, we collect the results of a series of experiments conducted on our metaheuristic approaches (Simulated Annealing, Reactive Tabu Search, Pilot Method and Genetic Algorithm) to prove their effectiveness. Being Modified MVCA a simple and fast constructive heuristic, it was not compared with the metaheuristics, but was used as a subroutine for them as described in Section 3. We also tested our Branch and Bound procedure on instances that proved to be solvable using such an approach.

5.1. Testing environment, instances and parameters settings

Computational experiments were performed on an HP xw8000 Workstation with 1 GB RAM and a 2.8 GHz Intel Xeon processor. The metaheuristic framework HotFrame [26] was used for the implementations of SA, RTS and Pilot Method, while GA and the Branch and Bound were coded from scratch. All the algorithms were implemented using the C++ language.

A dataset of 200 randomly generated instances was used. These instances are derived from the ones presented in [7] and [8] for the minimum label spanning tree problem.

For these instances, we used $|V| = 100, 150, 200, 400, 500$ and $|L| = \lfloor |V|/4 \rfloor, |V|/2, |V|, \lfloor 1.25|V| \rfloor$.

For each of these values, we generated datasets with a number of edges equal to $|E| = \frac{(|V|-1)|V|}{10}$. For each combination of $|V|$ and $|L|$, 10 different instances are considered.

In order to be used in our problem, the additional parameter k_{max} had to be defined for each instance. This is a factor of crucial relevance to create meaningful instances. In fact, if the chosen value is too large (in particular, greater than the number of labels needed to obtain a MLST), it could be easy for the heuristics to detect a single connected component as solution in a small amount of time. On the other hand, if the value of k_{max} is trivially small, the problem could be equally easy since the search space could be quickly covered.

Therefore, for each value of $|V|$ and $|L|$ we experimentally determined the value k_{max} as $\lfloor |V|/2^i \rfloor$, where i is the smallest value such that the generated instances did not report a solution value of 1 when solved with Modified MVCA. The actual k_{max} values used are reported in Tables 1-2.

For the SA procedure, we follow the parameters setting indicated in [25], which was reported to be robust for different problems. Namely, we use $\alpha = 0.95$, $initialAcceptanceFraction = 0.4$, $frozenAcceptanceFraction = 0.02$, $sizeFactor = 16$ and $frozenLimit = 5$. The RTS algorithm uses a time limit as termination parameter; therefore, for each instance, we used the execution time of the related SA execution. Since the procedure checks at the end of each iteration whether the time limit was reached, some of the running times reported for RTS in Tables 1-2 are slightly higher than the related SA times. For the single parameter p used by our proposed GA, after a preliminary testing phase we empirically chose the value 50. The Pilot Method does not depend on specific parameters.

5.2. Comments on the results

Results as well as computational times, expressed in seconds, are reported in Tables 1-2. Each entry in the tables is an average over the 10 instances of the related scenario. All of the instances with 100 nodes, as well as the ones with 150 nodes and $|L| = \lfloor |V|/4 \rfloor, |V|/2, |V|$ could be solved exactly within one hour of computational time. Therefore in Table 1 we compare heuristic and exact solutions on such instances, while in Table 2 we compare the heuristics to investigate runtime growth and comparitive solution quality on bigger scenarios.

On instances with $|V| = 100$ and 150, the Branch and Bound procedure is actually on average the fastest procedure in 3 scenarios, and the average lifetime per instance is less than one second for 5 scenarios. Overall, the pruning techniques prove to be very effective on instances of this size.

For those instances where the exact solution is known, all the heuristics show a very good performance. In these 7 scenarios, Simulated Annealing and Reactive Tabu Seach always find the optimal solution in 5 cases, the Genetic Algorithm in 6 cases and the Pilot Method in 3 cases.

Conversely, on the larger instances in Table 2, the Pilot Method emerges overall as the most accurate heuristic, obtaining on average the best solutions in 8 out of 12 scenarios. SA and RTS obtain the best results in 5 and 6 cases, respectively, while GA performs slightly worse than the other algorithms in all cases except one.

Regarding computational times, Reactive Tabu Search performs in some cases faster than Simulated Annealing. This shows that RTS tends to have a faster convergence (recall that for each RTS instance the execution time of SA was used as time limit). The Genetic Algorithm is generally slower than both of them, while the Pilot Method is the slowest heuristic, especially when the size of the problem grows and its performances improve with respect to the other heuristics.

Dataset			SA		RTS		GA		Pilot		B&B	
$\|V\|$	$\|L\|$	k_{max}	sol.	time	sol.	time	sol.	time	sol.	time	sol.	time
100	25	3	6.3	0.54	6.3	0.54	6.3	1.77	6.3	0.03	6.3	0.05
100	50	6	2.7	1.17	2.9	0.81	2.6	2.97	2.7	0.58	2.6	4.59
100	100	6	15.0	2.72	15.0	2.75	15.0	4.75	15.6	2.21	15.0	0.25
100	125	7	15.7	3.79	15.7	3.83	15.7	6.18	15.8	4.19	15.7	0.57
150	37	4	3.5	1.35	3.5	0.94	3.5	3.70	3.5	0.25	3.5	0.85
150	75	4	22.3	2.81	22.3	2.83	22.3	5.69	22.3	0.94	22.3	0.76
150	150	9	7.7	6.92	7.3	5.96	8.1	20.82	8.3	9.31	7.1	1518.00
150	187	11	6.1	9.34	6.2	7.80	6.2	23.32	6.1	13.22	N/A	TL

Table 1. Algorithms comparisons for $|V| = 100, 150$

Dataset			SA		RTS		GA		Pilot	
$\|V\|$	$\|L\|$	k_{max}	sol.	time	sol.	time	sol.	time	sol.	time
200	50	3	17.0	2.64	17.0	2.65	17.2	6.62	17.0	0.35
200	100	6	9.3	5.94	9.3	5.99	9.6	18.20	9.6	6.22
200	200	12	2.6	11.10	3.1	7.60	3.1	35.79	3.0	19.34
200	250	15	1.8	11.78	1.6	8.52	1.5	34.61	1.2	38.03
400	100	3	35.6	36.59	35.6	36.78	35.6	31.75	35.6	9.80
400	200	6	24.4	80.01	23.7	80.67	25.1	87.56	24.0	117.64
400	400	12	12.1	180.72	11.4	154.01	12.5	266.40	11.1	581.79
400	500	15	8.7	243.29	8.2	151.04	8.9	347.84	7.7	1159.22
500	125	4	22.7	93.24	22.7	93.72	22.9	75.79	22.7	59.30
500	250	7	22.4	195.84	21.5	197.28	22.8	175.97	21.9	235.35
500	500	15	6.3	335.78	5.3	298.95	6.6	537.67	5.0	1790.50
500	625	19	3.4	374.48	3.0	274.16	3.4	789.43	2.1	1522.14

Table 2. Algorithms comparisons for $|V| = 200, 400, 500$

6. Conclusions

In this paper we addressed an optimization problem that generalizes the Minimum Labeling Spanning Tree problem, namely the k-Labeled Spanning Forest Problem. We defined the problem and designed various metaheuristic approaches, as well as a Branch and Bound exact procedure. The promising performances of the algorithms have been shown experimentally on a set of 200 instances. Future research will be focused in defining new metaheuristics and exact methods, such as a Branch and Cut approach, in order to extend the classes of instances that can be solved efficiently to optimality.

References

[1] Electromagnetic fields (EMF) special issue, Pathophysiology 16 (2–3) (2009) 67–250.
[2] R.-S. Chang, S.-J. Leu, The minimum labeling spanning trees, Information Processing Letters 63 (1997) 277–282.
[3] S. Krumke, H.-C.Wirth, On the minimum label spanning tree problem, Information Processing Letters 66 (1998) 81–85.
[4] Y. Wan, G. Chen, Y. Xu, A note on the minimum label spanning tree, Information Processing Letters 84 (2002) 99–101.
[5] Y. Xiong, B. Golden, E. Wasil, Worst-case behavior of the MVCA heuristic for the minimum labeling spanning tree problem, Operations Research Letters 33 (2005) 77–80.
[6] M. E. Captivo, J. C. N. Climaco, M. M. B. Pascoal, A mixed integer linear formulation for the minimum label spanning tree problem, Computers & Operations Research 36 (11) (2009) 3082–3085.
[7] Y. Xiong, B. Golden, E. Wasil, A one-parameter genetic algorithm for the minimum labeling spanning tree problem, Tech. rep., University of Maryland (2003).
[8] R. Cerulli, A. Fink, M. Gentili, S. Voß, Metaheuristics comparison for the minimum labelling spanning tree problem, in: B. Golden, S. Raghavan, E. Wasil (Eds.), The Next Wave on Computing, Optimization, and Decision Technologies, Springer, New York, 2005, pp. 93–106.
[9] Y. Xiong, B. Golden, E. Wasil, Improved heuristics for the minimum label spanning tree problem, IEEE Transactions on Evolutionary Computation 10 (6) (2006) 700–703.
[10] S. Consoli, K. Darby-Dowman, N. Mladenović, J. Moreno-Perez, Greedy randomized adaptive search and variable neighbourhood search for the minimum labelling spanning tree problem, European Journal of Operational Research 196 (2) (2009) 440–449.
[11] R. Cerulli, A. Fink, M. Gentili, S. Voß, Extensions of the minimum labelling spanning tree problem, Journal of Telecommunications and Information Technology (4) (2006) 39–45.
[12] S. Consoli, K. Darby-Dowman, N. Mladenović, J. Moreno-Perez, Variable neighbourhood search for the minimum labelling steiner tree problem, Annals of Operations Research 172 (1) (2009) 71–96.
[13] S. Consoli, J. Moreno-Perez, K. Darby-Dowman, N. Mladenović, Discrete particle swarm optimization for the minimum labelling steiner tree problem, Natural Computing 9 (1) (2010) 29–46.
[14] R. D. Carr, S. Doddi, G. Konjedov, M. Marathe, On the red-blue set cover problem, in: 11th ACN-SIAM Symposium on Discrete Algorithms, 2000, pp. 345–353.
[15] R. Cerulli, P. Dell'Olmo, M. Gentili, A. Raiconi, Heuristic approaches for the minimum labelling hamiltonian cycle problem., Electronic Notes in Discrete Mathematics 25 (1) (2006) 131–138.
[16] Y. Xiong, B. Golden, E. Wasil, The colorful traveling salesman problem, in: Extending the Horizons: Advances in Computing, Optimization, and Decision Technologies, 2007, pp. 115–123.
[17] N. Jozefowiez, G. Laporte, F. Semet, A branch-and-cut algorithm for the minimum labeling hamiltonian cycle problem and two variants, Computers & Operations Research 38 (11) (2011) 1534–1542.
[18] J. Silberholz, A. Raiconi, R. Cerulli, M. Gentili, B. Golden, S. Chen, Comparison of heuristics for the colorful traveling salesman problem, International Journal of Metaheuristics 2 (2) (2013) 141–173.
[19] Y. Chen, N. Cornick, A. O. Hall, R. Shajpal, J. Silberholz, I. Yahav, B. Golden, Comparison of heuristics for solving the gmlst problem, in: Telecommunications Modeling, Policy, and Technology, Vol. 44, Springer US, 2008, pp. 191–217.
[20] Y. Xiong, B. Golden, E. Wasil, S. Chen, The label-constrained minimum spanning tree problem, in: Telecommunications Modeling, Policy, and Technology, Vol. 44, Springer US, 2008, pp. 39–58.
[21] F. Carrabs, R. Cerulli, M. Gentili, The labeled maximum matching problem, Computers & Operations Research 36 (6) (2009) 1859–1871.
[22] C. Duin, S. Voß, The pilot method: A strategy for heuristic repetition with application to the Steiner problem in graphs, Networks 34 (1999) 181–191.
[23] F. W. Glover, M. Laguna, Tabu Search, Kluwer Academic, 1997.
[24] R. Battiti, Reactive search: Toward self-tuning heuristics, in: V. Rayward-Smith, I. Osman, C. Reeves, G. Smith (Eds.), Modern Heuristic Search Methods, Wiley, Chichester, 1996, pp. 61–83.
[25] D. S. Johnson, C. R. Aragon, L. A. McGeoch, C. Schevon, Optimization by simulated annealing: An experimental evaluation; part I, graph partitioning, Operations Research 37 (1989) 865–892.
[26] A. Fink, S. Voß, HotFrame: A heuristic optimization framework, in: S. Voß, D. Woodruff (Eds.), Optimization Software Class Libraries, Kluwer, Boston, 2002, pp. 81–154.