Constructive Heuristics for the Minimum Labelling Spanning Tree Problem: a preliminary comparison*

Consoli S.(1), Moreno José A.(2), Mladenovic N.(1), and Darby-Dowman K.(1)

(1) CARISMA, NET-ACE, Brunel University, London (UK)
(2) DEIOC, IUDR. Universidad de La Laguna, (Spain)

DEIOC Technical Report. September 2006

ABSTRACT: This report studies constructive heuristics for the minimum labelling spanning tree (MLST) problem. The purpose is to find a spanning tree that uses edges that are as similar as possible. Given an undirected labeled connected graph (i.e., with a label or color for each edge), the minimum labeling spanning tree problem seeks a spanning tree whose edges have the smallest possible number of distinct labels. The model can represent many real-world problems in telecommunication networks, electric networks, and multimodal transportation networks, among others, and the problem has been shown to be NP-complete even for complete graphs. A primary heuristic, named the maximum vertex covering algorithm has been proposed. Several versions of this constructive heuristic have been proposed to improve its efficiency. Here we describe the problem, review the literature and compare some variants of this algorithm.

1. INTRODUCTION

Many combinatorial optimization problems can be formulated on a graph where the possible solutions are spanning trees. These problems consist of finding spanning trees optimizing some measure and have been extensively studied in graph theory. Typical measures include the total length or the diameter of the tree. The *minimum labeling spanning tree* (MLST) problem is one of these problems. Many real-life combinatorial optimization problems belong to this class of problems and consequently there is a large and growing interest in both theoretical and practical aspects. For some of these problems there are polynomial-time algorithms, while most of them are NP-hard. Mainly, it means that it is not possible to guarantee that an exact solution to the problem can be found and one has to settle for heuristics and approximate solution approaches with performance guarantees.

In the MLST problem, we are given a graph with colored edges, and one seeks a spanning tree with the least number of colors. Such a model can represent many real-world problems in telecommunication networks, electric networks, and multimodal transportation networks, among others. For example, in communication networks, there are many different types of communication media, such as optic fiber, cable, microwave, telephone line and so on (Tanenbeum A.S., 1989). A communication node may communicate with different nodes by choosing different types of communication medium. Given a set of communication network nodes, the problem is to find a spanning tree (a connected communication network)

* This research has been partially supported by projects TIN2005-08404-C04-03 (70% of which are FEDER funds) and PI042005/044

that uses as few types of communication lines as possible. This spanning tree will reduce the construction cost and the complexity of the network.

The MLST problem can be formulated as a network or graph problem. We are given a labeled connected undirected graph $G = (V,E,L)$. The vertices represent communication nodes, the edges communication links and the labels communication type. Each edge in E is labeled or colored by a label or color in a finite set L that identifies the type of communication; different edges in the graph may have the same label. The objective is to find a spanning tree which uses the smallest number of different types of edges. Define L_T to be the set of different labels of the edges in a spanning tree T. The labeling can be represented by a function f_L: $E \rightarrow L$ for all edges $e \in E$ or by a partition P_L of the edge set; the sets of the partitions are those consisting of the edges with the same label or color.

For example, for telecommunication networks (and, more generally, any type of communication networks), managed by different and competing companies, the aim of each company is to ensure the service to each terminal node of the network whilst minimizing the cost (represented by connections managed by other companies). The telecommunication network is represented by a graph where each edge is assigned a set of colors and each color denotes a different company managing the edge (Voss S., Cerulli R., Fink A. and Gentili M., 2005). The aim is to find a spanning tree of the graph using the minimum number of colors, which means that all the terminal nodes are required to be covered avoiding cycles and minimizing the overall number of different companies.

The minimum labeling spanning tree problem (MLSTP) is formally defined as follows:

> **MLST Problem**. Given a labeled graph $G = (V,E,L)$, find a spanning tree T of G such that $|L_T|$ is minimized, where L_T is the set of labels used in T.

Figure 1 shows an input graph and its optimal MLST solution (Xiong Y., 2005). In contrast to the traditional minimum spanning tree (MST) problem, the MLST problem focuses on the uniqueness of a graph instead of the cost of a graph.

The rest of the report is organized as follows. In the next section, we review the literature on the problem. In section three we present several versions of the *maximum vertex covering algorithm* (MVCA). Section four includes the experimental analysis and the report ends with some brief conclusions.

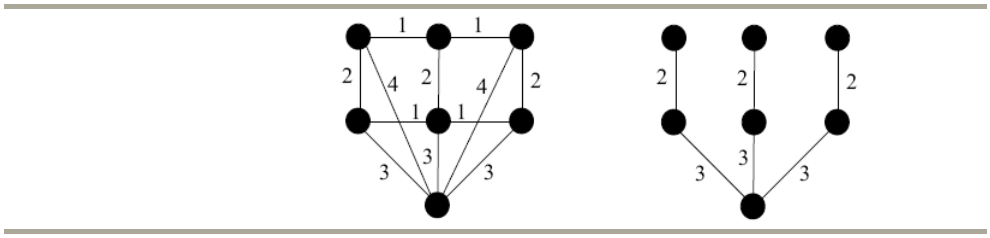

Figure 1: A sample graph and its optimal solution for the MLST problem.

2. LITERATURE REVIEW

In communication network design, it is often desirable to obtain a tree that is "most uniform" in some specified sense. Motivated by this observation, Chang R.S. and Leu S.J. (1997) introduced the minimum label spanning tree problem. They also proved that it is a NP-hard problem and provided an exponential time algorithm, the *maximum vertex covering algorithm* (MVCA), to find, in polynomial time, (possibly sub-optimal) solutions. This heuristic begins with an empty graph: $H = (V,\varnothing)$. Then it successively adds the label whose edges cover as many uncovered nodes as possible until all the nodes are covered. The heuristic solution is an arbitrary spanning tree of the resulting graph. The maximum vertex covering algorithm is given in Figure 2.

Original MVCA Algorithm

Input: A labeled graph $G = (V,E,L)$, with n vertices, m edges, and l labels.
Output: A spanning tree.

1. Let $C \leftarrow \varnothing$ be the initially empty set of used colors.
2. Let $H=(V,E(C))$ be the subgraph of G restricted to V and the edges with labels in C, where $E(C) = \{e \in E: L(e) \in C\}$.
3. *While H* has unvisited nodes:
   3.1. Find the unused label $c \in (L–C)$ such that edges with label c cover as many uncovered vertices as possible. If there is more than one candidate, select one randomly.
   3.2. Add label c to the set of used colors C: $C \leftarrow C \cup \{c\}$.
   3.3. Update H - add all the edges of label c: $E_H \leftarrow E_H \cup E(\{c\})$.
   end while.
4. Take an arbitrary spanning tree T of H.

Algorithm 1. Original MVCA

However, this version of MVCA has a major error. Sometimes, it does not yield a connected graph after all nodes are visited and thus fails, as can be seen in the example shown in Figure 2. This figure shows how the MVCA version of Chang R.S. and Leu S.J. (1997) works on the graph G of Figure 1.

LABEL	1	2	3	4
NUMBER OF VISITED NODES	6	6	4	3

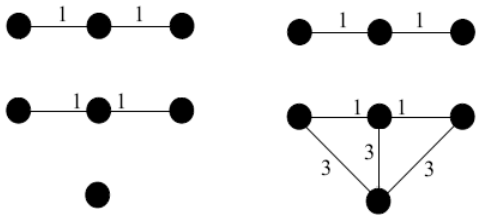

Figure 2: Original MVCA solution of the graph presented in Figure 1.

MVCA begins with an empty graph and first adds the label covering the highest number of unvisited nodes. In this example, label 1 and label 2 cover the same highest number of vertices (6). Suppose MVCA selects one of them randomly. So it adds label 1. Then it adds label 3 (4 vertices). At this time, all the nodes of the graph are visited and so MVCA halts, but the subgraph is still disconnected!

A corrected version of MVCA was proposed by Krumke S.O. and Wirth H.C. (1998). This version of MVCA begins with an empty graph. It successively adds labels by reducing the number of connected components by as many as possible until only one connected component is left, i.e. when only a connected subgraph is obtained. The details of this version of MVCA are described as follows:

Revised MVCA Algorithm:

Input: A labeled graph $G = (V,E,L)$, with n vertices, m edges, and l labels.
Output: A spanning tree T.

1. Let $C \leftarrow \varnothing$ be the initially empty set of used colors.
2. Let $H=(V,E(C))$ be the subgraph of G restricted to V and the edges with labels in C, where $E(C) = \{e \in E: L(e) \in C\}$.
3. Let $Comp(C)$ be the number of connected components of $H=(V,E(C))$.
4. *While* H has more than one connected component:
   4.1. Select the unused color $c \in (L–C)$ that maximizes $Comp(C \cup \{c\})$.
   4.2. Add label c to the set of used colors C: $C \leftarrow C \cup \{c\}$.
   4.3. Update H - add all the edges of label c: $E_H \leftarrow E_H \cup E(\{c\})$.
   end while.
5. Take an arbitrary spanning tree T of H.

Algorithm 2. Revised MVCA

1st ITERATION:				
LABEL	**1**	**2**	**3**	**4**
NUMBER OF COMPONENTS	3	4	4	5
2st ITERATION:				
LABEL	**1**	**2**	**3**	**4**
NUMBER OF COMPONENTS	-	2	2	2
3st ITERATION:				
LABEL	**1**	**2**	**3**	**4**
NUMBER OF COMPONENTS	-	1	-	1

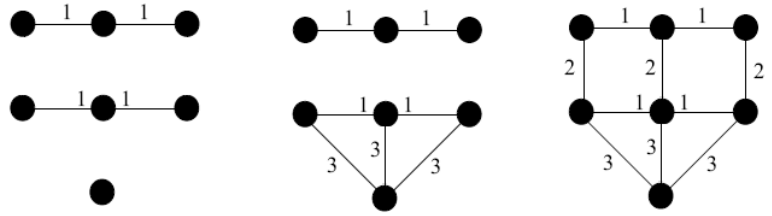

Figure 3: How the revised MVCA version works.

Figure 3 shows how this version of MVCA works on the graph of Figure 1. In the first step, it adds label 1 because it gives the least number of connected components (3 components). In the second step, all the three remaining colors (2, 3 and 4) produce the same number of components (2). In this case, the algorithm proceeds completely at random and it adds label 3. At this time, all the nodes of the graph are visited, but the subgraph is still disconnected. The old version of Chang R.S. and Leu S.J. (1997) stopped here, resulting in an error because the subgraph is still disconnected. However, the MVCA version of Krumke S.O. and Wirth H.C. (1998) finally adds label 2 to get only one connected component (even if it could add label 4 instead label 2; as in the second step, the selection criterion chooses

arbitrarily). Summarizing, the final solution is {1*;* 2*;* 3}, which is worse than the optimal solution {2*;* 3} of Figure 1.

Another feature of the MVCA heuristic is its worst-case behavior. Krumke S.O. and Wirth H.C. (1998) proved also that MVCA can yield a solution no greater than $(1 + 2 \log n)$ times optimal, where n is the total number of nodes. Later, Wan Y., Chen G. and Xu Y. (2002) obtained a better bound for the greedy algorithm introduced by Krumke S.O. and Wirth H.C. (1998). The algorithm was shown to be a $(1 + \log\cdot(n-1))$-approximation for any graph with n nodes ($n > 1$), improving the known performance guarantee $(1 + 2\cdot\log n)$.

Brüggemann T., Monnot J. and Woeginger G. J. (2003) used a different approach; they applied local search techniques based on the concept of *j*-switch neighborhoods to a restricted version of the MLST problem. In addition, they proved a number of complexity results and showed that if each label appears at most twice in the input graph, the MLST problem is solvable in polynomial time.

The MVCA bounds obtained in Wan Y., Chen G. and Xu Y. (2002) and Krumke S.O. and Wirth H.C. (1998) are not tight bounds. In fact, these bounds can never be attained. Xiong Y., Golden B. and Wasil E. (2005) obtained a tight result. For any graph with label frequency bounded by b, they showed that the worst-case bound of MVCA is b^{th}-harmonic number H_b, that is:

$$H_b = \sum_{i=1}^{b} \frac{1}{i}$$

Later, they constructed a worst-case family of graphs such that the MVCA solution is exactly H_b times the optimal solution. Since $H_b < (1 + \log\cdot(n - 1))$ and $b \leq (n - 1)$ (since otherwise the subgraph induced by the labels of maximum frequency contains a cycle and one can safely remove edges from the cycle without changing the problem), the tight bound H_b obtained is, therefore, an improvement on the previously known performance bound of $(1 + \log(n-1))$ given by Wan Y., Chen G. and Xu Y. (2002).

Xiong Y., Golden B. and Wasil E. (2005) presented a genetic algorithm (GA) to solve the MLST problem, outperforming MVCA in most cases.

Subsequently, Voss S., Cerulli R., Fink A. and Gentili M. (2005) applied their *pilot* method (originally developed by Duin C. and Voss S. (1999) and subsequently extended by Voss S. (2005)) to the MLST problem. Their pilot method is a greedy heuristic with a limited look-ahead capability. This modified version of MVCA tries each label at the first step and then applies MVCA in a greedy fashion from there on. Then it selects the best of the resulting solutions. This method generates high-quality solutions to the MLST problem, but running times are quite large (especially if the number of edges is high).

Xiong Y. (2005) implemented the pilot method of Voss S., Cerulli R., Fink A. and Gentili M. (2005), along with faster simplified versions. Also, these modified versions of MVCA focus on the first label added. In particular, after all of the labels have been sorted according to their frequencies from highest to the lowest, the first modified version tries only the most promising 10% of the labels at the first step. Afterwards, it runs MVCA to determine the remaining labels and then it selects the best of the *1* / 10 resulting solutions to go on

(where $l=|L|$ is the total number of colors and L is the set of possible labels for all edges). Compared with the original MVCA, this version can potentially reduce running time by about 90%. However, since a higher frequency label may not always be the best place to start, it may not perform as well as MVCA.

A second modified version of Xiong Y. (2005) is similar to the previous one, except that it tries the most promising 30% of the labels at the first step. Then it runs MVCA to determine the remaining labels. This version has been shown to be between MVCA and the first modified version of Xiong Y. (2005) with respect to accuracy and running time. Moreover, Xiong Y. (2005) presented a modified version of GA, which competes with the best of the pilot methods (the first modified version) in terms of solution quality and nearly comparable to the fastest of the pilot methods in terms of running time, offering a good compromise with respect to accuracy and running time.

The exact method is an A* or backtracking procedure to test the subsets of L. This search method performs a branch and prune procedure in the partial solution space based on a recursive procedure *Test* that test the possibilities of finding a better solution from the current partial solution. The main program that solves the MLST problem needs to call the *Test* procedure to an empty set of labels.

Procedure *Backtracking*:

1. Let $C \leftarrow \varnothing$ be the initially empty set of used colors.
2. Let $C^* \leftarrow L$ be the global set of used colors.
3. *Test(C).*
4. Take an arbitrary spanning tree T of $H^* = (V, E(C^*))$.

Procedure *Test*(*C*):

if $|C| < |C^*|$ *then*
  *if Comp(*C) = 1 *then*
    $C^* \leftarrow$ C.
  else if $|C| < |C^*|-1$ *then*
    for each $c \in (L-C)$ *do*
      Test$(C \cup \{c\})$.
    end for.

Algorithm 3. Exact algorithm for the MLST

Using an initial step that gets a good approximate solution C^* (for instance, a solution found from any version of the MVCA) will improve the performance by reducing the number of tested sets. Another improvement that avoids the examination of a large number of partial solutions consists of rejecting every partial solution that can not be completed to get only one connected component.

Note that if we are evaluating a partial solution C' with a number of colors $|C'|=|C^*|-2$, we should try to add one by one all the colors to check if it is possible to find a better solution of C^* with a smaller dimension, that is $|C'|=|C^*|-1$. To complete a partial solution C' with a number of colors $|C'|=|C^*|-2$ and get a better solution $|C'|=|C^*|-1$, we need to add a color with a frequency, at least, the actual number of connected components minus 1.

So, if the frequency of the color to add is not greater than the actual number of connected components minus 1, the partial solution can be rejected, speeding up the search process.

3. CONSTRUCTIVE ALGORITHMS

In this section we propose several versions that combine in several ways the main ideas used in constructive heuristics for the MLST; the use of the number of connected components, the frequency of each labels and the pilot strategies.

1. Basic MVCA: This algorithm is the MVCA algorithm of Krumke S.O. and Wirth H.C. (1998) and takes into account the number of components when adding each label.

Basic MVCA Algorithm:

Input: A labeled graph $G = (V,E,L)$, with n vertices, m edges, and l labels.
Output: A spanning tree T.

1. Let $C \leftarrow \emptyset$ be the initially empty set of used colors.
2. Let $H=(V,E(C))$ be the subgraph of G restricted to V and the edges with labels in C, where $E(C) = \{e \in E: L(e) \in C\}$.
3. Let $Comp(C)$ be the number of connected components of $H=(V,E(C))$.
4. *While* H has more than one connected component:
   4.1. Select the unused color $c \in (L–C)$ that maximizes $Comp(C \cup \{c\})$.
   4.2. Add label c to the set of used colors C: $C \leftarrow C \cup \{c\}$.
   4.3. Update H - add all the edges of label c: $E_H \leftarrow E_H \cup E(\{c\})$.
   end while.
5. Take an arbitrary spanning tree T of H.

Algorithm 4. **Basic MVCA**

2. QuickSort MVCA: This algorithm is the MVCA algorithm of Krumke S.O. and Wirth H.C. (1998) but it adds a sorting of the colors with respect to the general frequency of the colors. The general frequency of a color c is defined as the number of appearances, which is the number of edges with color c in the graph. The sorting of the frequencies is carried out using the efficient Quick Sort. This sorting algorithm, due to C.A.R. Hoare (1961), is a divide-and-conquer and massively recursive procedure (see, for instance Sedgewick R. (2001)). The algorithm consists of four steps:

1. If there is one or fewer elements in the array to be sorted, return immediately;
2. Pick an element in the array to serve as a "pivot" point. (the left-most element in the array is used);
3. Split the array into two parts - one with elements larger than the pivot and the other with elements smaller than the pivot;
4. Recursively repeat the algorithm for both halves of the original array.

The worst-case computational time of the Quick Sort $O(n^2)$ but it is $O(n \log n)$ in average-case with a very good practical performance.

QuickSort MVCA Algorithm:

Input: A labeled graph $G = (V,E,L)$, with n vertices, m edges, and l labels.
Output: A spanning tree T.

1. Let $C \leftarrow \emptyset$ be the empty set of used colors.
2. Let $H=(V,E(C))$ be the subgraph of G restricted to V and the edges with labels in C, where $E(C) = \{e \in E: L(e) \in C\}$.
3. Let $Comp(C)$ be the number of connected components of $H=(V,E(C))$.
4. Sort the colors in L in decreasing order with respect to the general frequency, using the Quick-Sort algorithm.
5. *While H* has more than one connected component:
   5.1. Select an unused color $c \in (L–C)$ as follows:
      5.1.1. Let $Comps^* = Comp(C)$.
      5.1.2. *for each* $i \in (L–C)$ *in the sorted order do*
         if $Comp(C \cup \{i\}) < Comps^*$ *then*
         then $c = i$; $Comps^* = Comp(C \cup \{c\})$.
      end for.
   5.2. Add label c to the set of used colors C: $C \leftarrow C \cup \{c\}$.
   5.3. Update H - add all the edges with label c: $E_H \leftarrow E_H \cup E(\{c\})$.
   end while.
6. Take an arbitrary spanning tree T of H.

Algorithm 5. QuickSort MVCA

3. Frequency MVCA: This algorithm is a MVCA version that does not perform any initial ordering of the colors with respect to the general frequency, as the *Basic MVCA*. As with the *QuickSort MVCA* algorithm, it proceeds, by adding at each step, a color minimizing the total number of connected components in the subgraph. The difference is that when more colors give the same number of connected components, it will add the one with the highest general frequency value.

Frequency MVCA Algorithm:

Input: A labeled graph $G = (V,E,L)$, with n vertices, m edges, and l labels.
Output: A spanning tree T.

1. Let $C \leftarrow \emptyset$ be the empty set of used colors.
2. Let $H=(V,E(C))$ be the subgraph of G restricted to V and the edges with labels in C, where $E(C) = \{e \in E: L(e) \in C\}$.
3. Let $Comp(C)$ be the number of connected components of $H=(V,E(C))$.
4. Get the general frequency of every color i by $fr(i)$, for all $i \in L$.
5. *While H* has more than one connected component:
   5.1. Select an unused color $c \in (L–C)$ as follows:
      5.1.1. Let $Comps^* = Comp(C)$, $fr^* = 0$.
      5.1.2. *for each* $i \in (L–C)$ *do*
         if $(Comp(C \cup \{i\}) < Comps^*)$ *then*
            $c = i$; $Comps^* = Comp(C \cup \{c\})$; $fr^* = fr(c)$.
         if $(Comp(C \cup \{i\}) = Comps^*)$ and $(fr(i) > fr^*)$ *then*
            $c = i$; $fr^* = fr(c)$.
      end for.

5.2. Add label c to the set of used colors C: $C \leftarrow C \cup \{c\}$.
5.3. Update H - add all the edges with label c: $E_H \leftarrow E_H \cup E(\{c\})$.
end while.
6. Take an arbitrary spanning tree T of H.

Algorithm 6. Frequency MVCA

4. Random MVCA: This is a MVCA version that also does not perform any initial ordering of the colors with respect to the general frequency, as the *Basic MVCA*. As usual, the algorithm proceeds, by adding at each step, the color that minimizes the total number of connected components in the subgraph. The difference is that when more than one color gives the same number of connected components, it will add one of them randomly.

Random MVCA Algorithm:

Input: A labeled graph $G = (V,E,L)$, with n vertices, m edges, and l labels.
Output: A spanning tree T.
1. Let $C \leftarrow \emptyset$ be the initially empty set of used colors.
2. Let $H=(V,E(C))$ be the subgraph of G restricted to V and the edges with labels in C, where $E(C) = \{e \in E: L(e) \in C\}$.
3. Let $Comp(C)$ be the number of connected components of $H=(V,E(C))$.
4. *While* H has more than one connected component:
   4.1. Select an unused color $c \in (L–C)$ as follows:
      4.1.1. Let $Comps^* = Comp(C)$ and $C^* \leftarrow \emptyset$.
      4.1.2. *for each* $i \in (L–C)$ *do*
         if $(Comp(C \cup \{i\}) = Comps^*)$ *then*
            $C^* \leftarrow C^* \cup \{i\}$.
         if $(Comp(C \cup \{i\}) < Comps^*)$ *then*
            $C^* \leftarrow \{i\}$; $Comps^* = Comp(C \cup \{i\})$.
         end for.
      4.1.3. Choose at random a color c in C^*.
   4.2. Add the label c to the set of used colors C: $C \leftarrow C \cup \{c\}$.
   4.3. Update H - add all the edges with label c: $E_H \leftarrow E_H \cup E(\{c\})$.
   end while.
5. Take an arbitrary spanning tree T of H.

Algorithm 7. Random MVCA

5. Multiple-Random MVCA: As in *Random MVCA*, at each step the method randomly adds the color that minimizes the total number of connected components in the subgraph. The difference is that the method runs N (N = 50 in our examples) times for each instance and keeps the best found solution (C^*).

Multiple-Random MVCA Algorithm:

Input: A labeled graph $G = (V,E,L)$, with n vertices, m edges, and l labels.
Output: A spanning tree T.
1. Let $C' \leftarrow L$ be the global set of used colors.

2. Let $H'=(V,E(C'))$ be the subgraph of G restricted to V and the edges with labels in C', where $E(C') = \{e \in E: L(e) \in C'\}$.
3. *Repeat N* times steps 4 to 8:
4. Let $C \leftarrow \varnothing$ be the initially empty set of used colors for each repetition.
5. Let $H=(V,E(C))$ be the subgraph of G restricted to V and the edges with labels in C, where $E(C) = \{e \in E: L(e) \in C\}$.
6. Let $Comp(C)$ be the number of connected components of $H=(V,E(C))$.
7. *While H* has more than one connected component:
   7.1. Select an unused color $c \in (L–C)$ as follows:
      7.1.1. Let $Comps^* = Comp(C)$ and $C^* \leftarrow \varnothing$.
      7.1.2. *for each* $i \in (L–C)$ *do*
         if $(Comp(C \cup \{i\}) = Comps^*)$ *then*
            $C^* \leftarrow C^* \cup \{i\}$.
         if $(Comp(C \cup \{i\}) < Comps^*)$ *then*
            $C^* \leftarrow \{i\}$; $Comps^* = Comp(C \cup \{i\})$.
         end for.
      7.1.3. Choose at random a color c in C^*.
   7.2. Add the label c to the set of used colors C: $C \leftarrow C \cup \{c\}$.
   7.3. Update H - add all the edges with label c: $E_H \leftarrow E_H \cup E(\{c\})$.
   end while.
8. *if* $|C| < |C'|$ *then*
   $C' \leftarrow C$; $H' \leftarrow H$.
9. Take an arbitrary spanning tree T of $H' = (V,E(C'))$.

Algorithm 8. Multiple-Random MVCA

6. Random-Frequency MVCA. In this version, at each step the algorithm adds the color which minimizes the total number of connected components in the subgraph. If more than one color gives the same number of connected components, it will add the one with the highest general frequency value. In the case of more than one color producing an equal number of connected components and with the same general frequency value, the algorithm will add one of them randomly.

Random-Frequency MVCA Algorithm:

Input: A labeled graph $G = (V,E,L)$, with n vertices, m edges, and l labels.
Output: A spanning tree T.

1. Let $C \leftarrow \varnothing$ be the initially empty set of used colors.
2. Let $H=(V,E(C))$ be the subgraph of G restricted to V and the edges with labels in C, where $E(C) = \{e \in E: L(e) \in C\}$.
3. Let $Comp(C)$ be the number of connected components of $H=(V,E(C))$.
4. Get the general frequency of every color i by $fr(i)$, for all $i \in L$.
5. *While H* has more than one connected component:
   5.1. Select an unused color $c \in (L–C)$ as follows:

5.1.1. Let $Comps^* = Comp(C)$, $C^* \leftarrow \varnothing$ and $fr^* = 0$.
5.1.2. *for each* $i \in (L–C)$ *do*
if $(Comp(C \cup \{i\}) < Comps^*)$ *then*
$C^* \leftarrow \{i\}$; $Comps^* = Comp(C \cup \{i\})$; $fr^* = fr(i)$.
if $(Comp(C \cup \{c\}) = Comps^*)$ and $(fr(i) > fr^*)$ *then*
$C^* \leftarrow \{i\}$; $Comps^* = Comp(C \cup \{i\})$; $fr^* = fr(i)$.
if $(Comp(C \cup \{c\}) = Comps^*)$ and $(fr(i) = fr^*)$ *then*
$C^* \leftarrow C^* \cup \{i\}$.
end for.
5.1.3. Choose at random a color c in C^*.
5.2. Add the label c to the set of used colors C: $C \leftarrow C \cup \{c\}$.
5.3. Update H - add all the edges with label c: $E_H \leftarrow E_H \cup E(\{c\})$.
end while.
6. Take an arbitrary spanning tree T of H.

Algorithm 9. Random-Frequency MVCA

7. Pilot-First MVCA (Voss S., Cerulli R., Fink A. and Gentili M., 2005): This modified version of MVCA focuses on the first label to add. It tries each label at the first step and then applies MVCA in a greedy fashion from then on, i.e. adding at each step the color that minimizes the total number of connected components and stopping when the resulting graph is connected. Then it selects the best of the $|L| = l$ resulting solutions as the output of the method. If the number of labels is large, we expect this algorithm be quite time-consuming.

Pilot-First MVCA Algorithm:
Input: A labeled graph $G = (V,E,L)$, with n vertices, m edges, and l labels.
Output: A spanning tree T.
1. Let $C' \leftarrow L$ be the global set of used colors.
2. Let $H'=(V,E(C))$ be the subgraph of G restricted to V and the edges with labels in C', where $E(C') = \{e \in E: L(e) \in C'\}$.
3. *for each* $i \in L$ *do*
3.1. Let $C \leftarrow \{i\}$ be the initial set of used colors for each iteration.
3.2. Let $H=(V,E(C))$ be the subgraph of G restricted to V and the edges with labels in C, where $E(C) = \{e \in E: L(e) \in C\}$.
3.3. Let $Comp(C)$ be the number of connected components of $H=(V,E(C))$.
3.4. *While* H has more than one connected component:
3.4.1. Select the unused color $c \in (L–C)$ that maximizes $Comp(C \cup \{c\})$.
3.4.2. Add label c to the set of used colors C: $C \leftarrow C \cup \{c\}$.
3.4.3. Update H - add all the edges of label c; $E_H \leftarrow E_H \cup E(\{c\})$.
end while.
3.5. *if* $|C| < |C'|$ *then*
$C' \leftarrow C$; $H' \leftarrow H$.
end for.
4. Take an arbitrary spanning tree T of $H' = (V,E(C'))$.

Algorithm 10. Pilot-First MVCA

8. Pilot-Each-Step MVCA: The difficulty with the classical version of MVCA is when it finds more than one color with same resulting number of connected components. A question arises on which of these colors should be chosen. To find the best possible solution that MVCA can give, we alternatively add each of these possible colors, continuing after the same strategy to the resulting subgraph. Every possible local choice is computed in this way and all the possible solutions MVCA may give are evaluated. The best result is selected. So no other MVCA version can give a better result than this variation of MVCA, because it visits all the solutions that every MVCA version can produce. Obviously, it is the slowest MVCA method and, if the number of labels is large, it will be quite time-consuming.

This method can be implemented as an A* or backtracking algorithm constrained to the colors that reach the same minimal number of connected components at each step. This search can be implemented using a modified version of the recursive procedure *Test* used in the exact A* or Backtracking algorithm. This modified version of the procedure *Test* just tries to add each of the colors with the same minimum number of connected components. Moreover, the main program that implements the *Pilot-Each-Step MVCA* also applies the *Test* procedure to the empty set.

Procedure *Pilot-Each-Step MVCA*:

1. Let $C \leftarrow \emptyset$ be the initially empty set of used colors.
2. Let $C^* \leftarrow L$ be the global set of used colors.
3. *Test(C).*
4. Take an arbitrary spanning tree T of $H^* = (V, E(C^*))$.

Procedure *Test(C)*:

if $|C| < |C^*|$ *then*
  *if Comp(*C) = 1 *then*
    $C^* \leftarrow$ C.
  else if $|C| < |C^*|-1$ *then*
    $C^* \leftarrow \emptyset$; $Comps^* = n$.
    for each $c \in (L–C)$ *do*
      if Comp(C $\cup\{c\}$) = *Comps* then*
        $C^* \leftarrow C^* \cup \{c\}$.
      if Comp(C $\cup\{$c$\}$) < *Comps* then*
        $C^* \leftarrow \{c\}$; $Comps^* = Comp($C $\cup\{c\}$).
    end for.
    for each $c \in C^*$ *do*
      Test($C \cup \{c\}$).
    end for.

Algorithm 11. Pilot-Each-Step MVCA

9. Pilot-Total MVCA: This method is the union of the previous two algorithms: *Pilot-First MVCA* and *Pilot-Each-Step MVCA*. It tries each label at the first step and then applies the *Pilot-Each-Step MVCA* method from there on, i.e. adding at each step alternatively all of the colors having the same minimum number of connected components and stopping when the resulting graph is connected. In this way, for each color added as the first one, every possible local choice is evaluated. The best result is recorded as output of the *Pilot-Each-Step MVCA*

method and it represents the best possible solution MVCA can produce given the first color chosen. Afterwards, this procedure is repeated for a different color added in the first position, until all the possibilities are exhausted. Then the method will select the best of the $|L| = l$ resulting solutions as the output of the method. Obviously, it is the more time-consuming method compared to the others analyzed so far.

This method can also be implemented as an A* or backtracking algorithm where the color included, except for the first color, is constrained to the colors that reach the minimal number of connected components. This search can be implemented using the same recursive procedure *Test* used in the *Pilot-Each-Step MVCA* algorithm, but with a main program that starts with every unitary set of colors instead of the empty set. This main program that implements the *Pilot-Total MVCA* heuristic is shown below on Algorithm 12.

Procedure *Pilot-Total MVCA*:

1. Let $C \leftarrow \varnothing$ be the initially empty set of used colors.
2. Let $C^* \leftarrow L$ be the global set of used colors.
3. *for each* $c \in L$ *do*
   3.1. $C \leftarrow \{c\}$.
   3.2. *Test*(C).
4. *end for.*
5. Take an arbitrary spanning tree T of $H^* = (V, E(C^*))$.

Algorithm 12. Pilot-Total MVCA

4. EXPERIMENTAL TESTS

In this section, the results of several heuristics are compared to the exact solution found via backtrack search.

Different sets of instances of the problem have been used in order to evaluate how the algorithms are influenced by the parameters, the structure of the network and the distribution of the labels on the edges. The parameters considered are:

m: total number of edges of the graph;
n: total number of nodes of the graph;
l: total number of colors assigned to the edges of the graph.

The number of vertices and colors are chosen between 20 and 50 with step 10. The number of edges is obtained indirectly from the density d of edges whose values are chosen to be 0.8, 0.5, 0.2. The density d is the number of edges in the graph divided by the number of possible edges, i.e. that of the complete graph is $n(n–1)/2$. The number of edges from the number of vertices and the density is obtained by $m = d \cdot n(n–1)/2$. The considered scenarios are $n = l = 20, 30, 40, 50$, and $d = 0.8, 0.5, 0.2$, for a total of 12 scenarios. For each scenario we use 10 different randomly generated instances. We thank authors of (Cerulli R., Fink A., Gentili M. and Voss S., 2005), who kindly provided data for use in our experiments. Are results are therefore directly comparable with their results.

Tables 1 and Table 2 show the results obtained with the nine proposed versions of the MVCA, identified with the labels: 1) Basic MVCA (B-MVCA), 2) QuickSort MVCA (Q-MVCA), 3) Frequency MVCA (F-MVCA), 4) Random MVCA (R-MVCA), 5) Multiple-Random MVCA (MR-MVCA), 6) Random-Frequency MVCA (RF-MVCA), 7) Pilot-First MVCA (P1-MVCA), 8) Pilot-Each-Step MVCA (PS- MVCA) and 9) Pilot-Total MVCA (PT-MVCA) Table 1 includes the average objective values among the 10 instance for each combination of the parameter and each algorithm. In the first row we find the optimal value obtained with the backtracking. Table 2 includes the corresponding average values of the computational time in milliseconds.

	***n* = *l* = 20**			***n* = *l* = 30**			***n* = *l* = 40**			***n* = *l* = 50**		
Density:	**0.8**	**0.5**	**0.2**	**0.8**	**0.5**	**0.2**	**0.8**	**0.5**	**0.2**	**0.8**	**0.5**	**0.2**
Exact Method	***2.4***	***3.1***	***6.7***	***2.8***	***3.7***	***7.4***	***2.9***	***3.7***	***7.4***	***3.0***	***4.0***	***8.6***
1. B-MVCA	2.6	3.5	7.1	2.8	3.8	7.8	2.9	4.1	8.6	3.0	4.2	9.4
2. Q-MVCA	2.6	3.5	6.9	2.8	4.0	7.9	2.9	3.9	8.2	3.0	4.4	9.6
3. F-MVCA	2.6	3.5	7.0	2.8	3.9	7.9	2.9	3.9	8.1	3.0	4.4	9.5
4. R-MVCA	2.5	3.6	6.9	2.8	3.9	7.8	2.9	4.2	8.4	3.0	4.3	9.4
5. MR-MVCA	2.5	3.5	6.7	2.8	3.7	7.4	2.9	3.9	7.7	3.0	4.2	8.6
6. RF-MVCA	2.5	3.6	6.9	2.8	4.0	7.8	2.9	3.9	8.0	3.0	4.4	9.5
7. P1-MVCA	2.4	3.2	6.7	2.8	3.7	7.4	2.9	3.7	7.6	3.0	4.1	8.6
8. PS-MVCA	2.5	3.5	6.7	2.8	3.7	7.4	2.9	3.9	7.7	3.0	4.2	8.6
9. PT-MVCA	2.4	3.1	6.7	2.8	3.7	7.4	2.9	3.7	7.5	3.0	4.1	8.6

Table 1. Average objective Values

	***n* = *l* = 20**			***n* = *l* = 30**			***n* = *l* = 40**			***n* = *l* = 50**		
Density:	**0.8**	**0.5**	**0.2**	**0.8**	**05**	**0.2**	**0.8**	**0.5**	**0.2**	**0.8**	**0.5**	**0.2**
Exact Method	***0***	***0***	***0.011***	***0***	***0***	***0.138***	***0.002***	***0.003***	***1.002***	***0.003***	***0.019***	***62.728***
1. B-MVCA	0.0	0.0	1.5	0.0	0.0	1.6	0.0	0.0	0.0	3.0	1.6	0.0
2. Q-MVCA	0.0	0.0	0.0	0.0	0.0	0.0	3.1	1.6	0.0	4.7	0.0	0.0
3. F-MVCA	0.0	0.0	0.0	0.0	1.5	0.0	0.0	1.6	0.0	0.0	0.0	1.5
4. R-MVCA	0.0	1.6	3.1	1.5	0.0	7.8	1.5	4.6	3.2	3.1	0.0	4.7
5. MR-MVCA	7.9	6.3	6.2	15.6	12.4	11.0	29.7	18.8	15.7	34.5	37.5	28.1
6. RF-MVCA	0.0	0.0	0.0	1.6	0.0	0.0	0.0	1.5	3.1	1.6	0.0	0.0
7. P1-MVCA	0.0	1.6	1.6	7.8	7.7	4.8	17.3	1.7	14.1	32.9	31.2	25.0
8. PS-MVCA	0.0	0.0	4.8	0.0	1.6	6.2	1.6	4.9	20.3	3.1	0.0	25.0
9. PT-MVCA	0.0	0.0	3.6	4.8	17.2	65.7	15.6	14.0	137.5	3.1	26.6	321.0

Table 2. Computational times (milliseconds)

5. CONCLUSIONS

We have considered the design of constructive heuristics for the minimum labelling spanning tree problems. The *maximum vertex covering algorithm* (MVCA) is wrong but several corrected versions can reach very good solutions in a very short computational time. Among the different strategies implemented the best effectiveness is shown by the P1-MVCA and PT-MVCA based on the Pilot Method but with a high time consuming. In the other side, the F-MVCA and RF-MVCA has also good, but not so optimal, effectiveness with much less

computational time. In an intermediate situation is PS-MVCA. The future research will consist in improving the procedures, testing with larger instances, and testing their use to generate initial solutions for other metaheuristics as VNS.

REFERENCES

[1] Brüggemann T., Monnot J. and Woeginger G. J., (2003). 'Local search for the minimum label spanning tree problem with bounded color classes', *Operations Research Letters*, vol. 31, pp. 195–201.

[2] Cerulli R., Fink A., Gentili M. and Voss S., (2005). 'Metaheuristics comparison for the minimum labelling spanning tree problem'. In Golden B.L, Raghavan S. and Wasil E.A. (eds.), *The Next Wave on Computing, Optimization, and Decision Technologies*, New York: in Springer, 93 - 106.

[3] Chang R.S. and Leu S.J., (1997). 'The minimum labeling spanning trees', *Information Processing Letters*, vol. 63, no. 5, pp. 277–282.

[4] Duin C. and Voss S., (1999). 'The pilot method: A strategy for heuristic repetition with applications to the Steiner problem in graphs', *Wiley InterScience*, Vol. 34, Issue 3, pag. 181-191.

[5] Golden B., Raghavan S. and Stanojević, (2004). 'Heuristic search for the generalized minimum spanning tree problem'. *INFORMS J. Computation XXX*

[6] Hoare, C. A. R. (1961) 'Partition: Algorithm 63', 'Quicksort: Algorithm 64', and 'Find: Algorithm 65'. Comm. ACM Vol. 4, pp. 321-322.

[7] Krumke S.O. and Wirth H.C., (1998). 'On the minimum label spanning tree problem', *Information Processing Letters*, vol. 66, no. 2, pp. 81–85.

[8] Sedgewick R., (2002). *'Algorithm in C – Part 5: Graph algorithms'*, Addison-Wesley.

[9] Voss S., Cerulli R., Fink A. and Gentili M., (2005). 'Applications of the pilot method to hard modifications of the minimum spanning tree problem', Presented at the *18th MINI EURO Conference on VNS*, Tenerife, Spain.

[10] Wan Y., Chen G. and Xu Y., (2002). 'A note on the minimum label spanning tree', *Information Processing Letters*, vol. 84, pp. 99–101.

[11] Sedgewick, R. (2001). *'Algorithms in C. Parts 1-5 (Bundle): Fundamentals, Data Structures, Sorting, Searching, and Graph Algorithms'*, Addison-Wesley

[12] Tanenbeum A.S., (1989). *'Computer Networks',* Englewood Cliffs, NJ: in Prentice-Hall.

[13] Xiong Y., Golden B. and Wasil E., (2005). 'Worst case behavior of the MVCA heuristic for the minimum labeling spanning tree problem', *Operations Research Letters*, vol. 33, no. 1, pp. 77–80.

[14] Xiong Y., Golden B. and Wasil E., (2005). 'A One-Parameter Genetic Algorithm for the Minimum Labeling Spanning Tree Problem', *IEEE Transactions on Evolutionary Computation.*, vol. 9, no. 1.

[15] Xiong Y., (2005).*'The Minimum Labeling Spanning Tree Problem and some variants',* Ph.D. Thesis, graduate School of the University of Maryland, USA.