Greedy randomized adaptive search and variable neighbourhood search for the minimum labelling spanning tree problem

S. Consoli [a,*], K. Darby-Dowman [a], N. Mladenović [a], J. A. Moreno Pérez [b]

[a] *CARISMA and NET-ACE, School of Information Systems, Computing and Mathematics, Brunel University, Uxbridge, Middlesex, UB8 3PH, United Kingdom*

[b] *DEIOC, IUDR, Universidad de La Laguna, Facultad de Matemáticas, 4a planta Astrofisico Francisco Sánchez s/n, 38271, Santa Cruz de Tenerife, Spain*

Abstract

This paper studies heuristics for the minimum labelling spanning tree (MLST) problem. The purpose is to find a spanning tree using edges that are as similar as possible. Given an undirected labelled connected graph, the minimum labelling spanning tree problem seeks a spanning tree whose edges have the smallest number of distinct labels. This problem has been shown to be NP-hard. A Greedy Randomized Adaptive Search Procedure (GRASP) and a Variable Neighbourhood Search (VNS) are proposed in this paper. They are compared with other algorithms recommended in the literature: the Modified Genetic Algorithm and the Pilot Method. Nonparametric statistical tests show that the heuristics based on GRASP and VNS outperform the other algorithms tested. Furthermore, a comparison with the results provided by an exact approach shows that we may quickly obtain optimal or near-optimal solutions with the proposed heuristics.

Key words: Metaheuristics, Combinatorial optimisation, Minimum labelling spanning tree, Variable Neighbourhood Search, Greedy Randomized Adaptive Search Procedure.

1. Introduction

Many combinatorial optimisation problems can be formulated on a graph where the possible solutions are spanning trees. These problems consist of finding spanning trees

* Corresponding author.
☎ +44 (0)1895 266820; fax: +44 (0)1895 269732
✉ sergio.consoli@brunel.ac.uk

that are optimal with respect to some measure and have been extensively studied in graph theory (Avis et al., 2005). Typical measures include the total length or the diameter of the tree. Many real-life combinatorial optimisation 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 are NP-hard. Thus, it is not possible to guarantee that an exact solution to the problem can be found within an acceptable timeframe and one has to settle for heuristics and approximate solution approaches with performance guarantees.

The *minimum labelling spanning tree* (MLST) problem is an NP-hard problem in which, given a graph with labelled edges, one seeks a spanning tree with the least number of labels. Such a model can represent many real-world problems in telecommunications networks, power networks, and multimodal transportation networks. For example, in telecommunications networks, there are many different types of communications media, such as optical fibre, coaxial cable, microwave, and telephone line (Tanenbaum, 2003). A communications node may communicate with different nodes by choosing different types of communications media. Given a set of communications network nodes, the problem is to find a spanning tree (a connected communications network) that uses as few communications types 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 labelled connected undirected graph $G = (V, E, L)$, where V is the set of n nodes, E is the set of m edges, and L is the set of ℓ labels. In the telecommunications example (Tanenbaum, 2003), the vertices represent communications nodes, the edges communications links, and the labels communications types. Each edge in E has a label in a finite set L that identifies the communications type. The objective is to find a spanning tree that 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 labelling 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.

Another example is given by multimodal transportation networks (Van-Nes, 2002). In such problems, it is desirable to provide a complete service using the minimum number of companies. The multimodal transportation network is represented by a graph where each edge is assigned a label, denoting a different company managing that edge. The aim is to find a spanning tree of the graph using the minimum number of labels. The interpretation is that all terminal nodes are connected without cycles, using the minimum number of companies.

The minimum labelling spanning tree problem is formally defined as follows:

MLST problem: Given a labelled graph $G = (V, E, L)$, where V is the set of n nodes, E is the set of m edges, and L is the set of ℓ labels, find a spanning tree T of G such that $|L_T|$ is minimized, where L_T is the set of labels used in T.

Although a solution to the MLST problem is a spanning tree, we first consider connected subgraphs. A feasible solution is defined as a set of labels $C \subseteq L$, such that all the edges with labels in C represent a connected subgraph of G and span all the nodes in G. If C is a feasible solution, then any spanning tree of C has at most $|C|$ labels. Moreover, if C is an optimal solution, then any spanning tree of C is a minimum labelling spanning

tree. Thus, in order to solve the MLST problem we seek a feasible solution with the least number of labels (Xiong et al., 2005a).

The upper left graph of Figure 1 is an example of an input graph with the optimal solution shown on the upper right. The lower part of Figure 1 shows examples of feasible solutions.

- INSERT FIGURE 1 -

The rest of the paper is organised as follows. In the next section, we review the literature of the problem. In section three we present the details of the heuristics considered in this paper: ones recommended in the literature (the *Modified Genetic Algorithm* of Xiong et al. (2006), and the *Pilot Method* of Cerulli et al. (2005)), and some new approaches to the MLST problem (a *Greedy Randomized Adaptive Search Procedure*, and a basic *Variable Neighbourhood Search*). Section four includes the experimental analysis of the comparison of these metaheuristics, and the paper ends with some conclusions. For a survey on the basic concepts of metaheuristics and combinatorial optimisation, the reader is referred to (Voß et al., 1999; Glover and Kochenberger, 2003).

2. Literature Review

In communications network design, it is often desirable to obtain a tree that is "most uniform" in some specified sense. Motivated by this observation, Chang and Leu (1997) introduced the minimum labelling spanning tree problem. They also proved that it is an NP-hard problem and provided a polynomial time heuristic, the *maximum vertex covering algorithm* (MVCA), to find (possibly sub-optimal) solutions. This heuristic begins with an empty graph. It then adds the label whose edges cover as many unvisited nodes as possible until all the nodes are covered. The heuristic solution is an arbitrary spanning tree of the resulting graph. However, with this version of MVCA, it is possible that although all the nodes of the graph are visited, it does not yield a connected graph and thus fails.

Krumke and Wirth (1998) proposed a corrected version of MVCA, depicted in Algorithm 1. This begins with an empty graph. Successively, it adds at random one label from those labels that minimize the number of connected components. The procedure contin-

Algorithm 1: Revised MVCA (Krumke and Wirth, 1998)

Input: A labelled, undirected, connected graph $G = (V, E, L)$ with n vertices, m edges, ℓ labels;
Output: A spanning tree T;
Initialisation:
- Let $C \leftarrow 0$ be the initially empty set of used labels;
- Let $H = (V, E(C))$ be the subgraph of G restricted to V and edges with labels in C, where $E(C) = \{e \in E : L(e) \in C\}$;
- Let $Comp(C)$ be the number of connected components of $H = (V, E(C))$;

begin
  while $Comp(C) > 1$ **do**
    Select the unused label $c \in (L - C)$ that minimizes $Comp(C \cup \{c\})$;
    Add label c to the set of used labels: $C \leftarrow C \cup \{c\}$;
    Update $H = (V, E(C))$ and $Comp(C)$;
  end
  $\Rightarrow$ Take any arbitrary spanning tree T of $H = (V, E(C))$.
end

ues until only one connected component is left, i.e. when only a connected subgraph is obtained.

Krumke and Wirth (1998) proved 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 et al. (2002) obtained a better bound for the greedy algorithm introduced by Krumke and Wirth (1998). The algorithm was shown to be a $(1 + \log(n-1))$-approximation for any graph with n nodes ($n > 1$).

Brüggemann et al. (2003) used a different approach; they applied local search techniques based on the concept of j-switch neighbourhoods 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.

Xiong et al. (2005b) derived tighter bounds than those proposed by Wan et al. (2002). For any graph with label frequency bounded by b, they showed that the worst-case bound of MVCA is the b^{th}-harmonic number $H_b = \sum_{i=1}^{b} \frac{1}{i} = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{b}$;

Later, they constructed a worst-case family of graphs such that the MVCA solution is exactly H_b times the optimal solution. Since $H_b < (1 + \log(n-1))$ 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), the tight bound H_b obtained is, therefore, an improvement on the previously known performance bound of $(1 + \log(n-1))$ given by Wan et al. (2002).

Other heuristic approaches to the MLST problem are proposed in the literature. For example, Xiong et al. (2005a) presented a *Genetic Algorithm* (GA) to solve the MLST problem, outperforming MVCA in most cases.

Subsequently, Cerulli et al. (2005) applied the *Pilot Method*, a greedy heuristic developed by Duin and Voß (1999) and subsequently extended in (Voß et al., 2005), to the MLST problem. Considering different sets of instances of the MLST problem, Cerulli et al. (2005) compared this method with other metaheuristics (Reactive Tabu Search, Simulated Annealing, and an ad-hoc implementation of Variable Neighbourhood Search). Their Pilot Method obtained the best results in most of the cases. It generates high-quality solutions to the MLST problem, but running times are quite large (especially if the number of labels is high).

Xiong et al. (2006) implemented simplified versions of the Pilot Method of Cerulli et al. (2005), along with a *Modified Genetic Algorithm* (MGA) which obtained the best performance for the MLST problem in terms of solution quality and running time.

3. Exploited metaheuristics

In this section, the details of the heuristics considered in this paper are specified. First, those that are reported in the literature to be the best performing are considered, followed by some new approaches.

Xiong et al. (2005a) presented two slightly different Genetic Algorithms to solve the MLST problem. They both were shown to be simple, fast, and effective. In most cases, they also outperformed MVCA, the most popular MLST heuristic in the literature at that time. Later, a *Modified Genetic Algorithm (MGA)* was proposed in (Xiong et al., 2006). It outperformed the first two Genetic Algorithms with respect to solution quality

and running time. MGA is the first metaheuristic that we consider.

Cerulli et al. (2005) applied the *Pilot Method* to the MLST problem. Comparing it with some other metaheuristic implementations (Reactive Tabu Search, Simulated Annealing, and an ad-hoc implementation of Variable Neighbourhood Search), it was the best performing in most of the test problems. The Pilot Method of Cerulli et al. (2005) is the second metaheuristic considered in this paper.

We then present some new approaches to the problem. We propose a new heuristic for the problem based on *GRASP: Greedy Randomized Adaptive Search Procedure*. Basically, GRASP is a metaheuristic combining the power of greedy local search with randomisation. For a survey on GRASP, the reader is referred to (Feo and Resende, 1995; Resende and Ribeiro, 2003).

The other algorithm that we propose is a basic *Variable Neighbourhood Search (VNS)*. Variable Neighbourhood Search is a recently exploited metaheuristic based on dynamically changing neighbourhood structures during the search process. For more details see (Mladenović and Hansen, 1997; Hansen and Mladenović, 2001, 2003).

3.1. *Modified Genetic Algorithm (Xiong et al., 2006)*

Genetic Algorithms are based on the principle of evolution, operations such as crossover and mutation, and the concept of fitness (Goldberg et al., 1991).

In the MLST problem, fitness is defined as the number of distinct labels in the candidate solution. After a number of generations, the algorithm converges and the best individual, hopefully, represents a near-optimal solution.

An individual (or a chromosome) in a population is a feasible solution. Each label in a feasible solution can be viewed as a gene. The initial population is generated by adding labels randomly to empty sets, until feasible solutions emerge. Crossover and mutation operations are then applied in order to build one generation from the previous one. Crossover and mutation probability values are set to 100%. The overall number of generations is chosen to be half of the initial population value. Therefore, in the Genetic Algorithm of Xiong et al. (2006) the only parameter to tune is the population size.

The crossover operation builds one offspring from two parents, which are feasible solutions. Given the parents $P_1 \subset L$ and $P_2 \subset L$, it begins by forming their union $P = P_1 \cup P_2$. Then it adds labels from the subgraph P to the initially empty offspring until a feasible solution is obtained, by applying the revised MVCA of Krumke and Wirth (1998) to the subgraph with labels in P, node set V, and the edge set associated with P. On the other hand, the mutation operation consists of adding a new label at random, and next trying to remove the labels (i.e., the associated edges), from the least frequently occurring label to the most frequently occurring one, whilst retaining feasibility.

3.2. *Pilot Method (Cerulli et al., 2005)*

The Pilot Method is a metaheuristic proposed by Duin and Voß (1999) and Voß et al. (2005). It uses a *basic heuristic* as a building block or application process, and then it tentatively performs iterations of the application process with respect to a so-called *master solution*. The iterations of the basic heuristic are performed until all the possible local choices (or moves) with respect to the master solution are evaluated. At the end of

all the iterations, the new master solution is obtained by extending the current master solution with the move that corresponds to the best result produced.

Considering a master solution M, for each element $i \notin M$, the Pilot Method is to extend tentatively a copy of M to a (fully grown) solution including i, built through the application of the basic heuristic. Let $f(i)$ denote the objective function value of the solution obtained by including each element $i \notin M$, and let i^* be a most promising of such elements, i.e. $f(i^*) \leq f(i)$, $\forall i \notin M$. The element i^*, representing the best local move with respect to M, is included in the master solution by changing it in a minimal fashion, leading to a new master solution $M = M \cup \{i^*\}$. On the basis of this new master solution M, new iterations of the Pilot Method are started $\forall i \notin M$, providing a new solution element i^*, and so on. This *look-ahead* mechanism is repeated for all the successive stages of the Pilot Method, until no further moves need to be added to the master solution. Alternatively, some user termination conditions, such as the maximum allowed CPU time or the maximum number of iterations, may be imposed in order to stop the algorithm when these conditions are satisfied. The last master solution corresponds to the best solution to date and it is produced as the output of the procedure.

For the MLST problem, the Pilot Method proposed by Cerulli et al. (2005) starts from the null solution (an empty set of labels) as master solution, uses the revised MVCA of Krumke and Wirth (1998) as the application process, and evaluates the quality of a feasible solution by choosing the number of labels included in the solution as the objective function. It then computes all the possible local choices from the master solution, performing a series of iterations of the application process to the master solution. This means that, at each step, it alternatively tries to add to the master solution each label not yet included, and then applies MVCA in a greedy fashion from then on (i.e. by adding at each successive step the label that minimizes the total number of connected components), stopping when the resulting subgraph is connected (note that, when the MVCA heuristic is applied to complete a partial solution, in case of ties in the minimum number of connected components, a label is selected at random within the set of labels producing the minimum number of components).

The Pilot Method successively chooses the best local move, that is the label that, if included to the current master solution, produces the feasible solution with the minimum objective function value (number of labels). In case of ties, it selects one label at random within the set of labels with the minimum objective function value. This label is then included in the master solution, leading to a new master solution. If the new master solution is still infeasible, the Pilot Method proceeds with the same strategy in this new step, by alternatively adding to the master solution each label not yet included, and then applying the MVCA heuristic to produce feasible solutions for each of these candidate labels. Again, the best move is selected to be added to the master solution, producing a new master solution, and so on. The procedure continues with the same mechanism until a feasible master solution is produced, that is one representing a connected subgraph, or until the user termination conditions are satisfied. At the end of the computation it may be beneficial to greedily drop labels from the master solution while retaining feasibility. The last master solution represents the output of the method.

Since up to ℓ master solutions can be considered by this procedure, and up to ℓ local choices can be evaluated for each master solution, the overall computational running time of the Pilot Method is $O(\ell^2)$ times the computational time of the application process (i.e. the MVCA heuristic), leading to an overall complexity $O(\ell^3)$.

3.3. *Greedy Randomized Adaptive Search Procedure*

The difficulty with the classical version of MVCA is when it finds more than one label with the same number of connected components. A question arises on the label to be chosen. To find the best MVCA solution, we should alternatively add each of these labels, continuing the same strategy in successive steps. In this way, every possible local choice is computed, because all the solutions that MVCA can produce are visited. But the execution time increases dramatically, especially for low-density graphs with a high number of nodes and labels.

In this paper we propose a Greedy Randomized Adaptive Search Procedure (GRASP) to the MLST problem, trying to unify multiple repetitions of the MVCA heuristic with the Pilot Method strategy in order to obtain an optimal balance between intensification and diversification capabilities.

GRASP is a recently exploited method combining the power of greedy heuristics, randomisation, and local search. It is a multi-start two-phase metaheuristic for combinatorial optimisation proposed by Feo and Resende (1995), basically consisting of a construction phase and a local search improvement phase.

The *solution construction* mechanism builds an initial solution using a greedy randomized procedure, whose randomness allows solutions in different areas of the solution space to be obtained. Each solution is randomly produced step-by-step by uniformly adding one new element from a candidate list (RCL_α: restricted candidate list of length α) to the current solution. Subsequently, a *local search* phase is applied (such as Simulated Annealing, Tabu Search) to try to improve the current best solution. This two-phase process is iterative, continuing until the user termination condition such as the maximum allowed CPU time, the maximum number of iterations, or the maximum number of iterations between two successive improvements, is reached.

Several new components have extended the scheme of GRASP (reactive GRASP, parameter variations, bias functions, memory and learning, improved local search, path relinking, hybrids,...). These components are presented and discussed in (Resende and Ribeiro, 2003).

Our GRASP implementation for the MLST problem is specified in Algorithm 2. The greedy criterion of the construction phase of GRASP (*Construction-Phase()* procedure) is based on the number of connected components produced by the labels, and a *value-based* restricted candidate list is used (Resende and Ribeiro, 2003). This involves placing in the list only the candidate labels having a greedy value (number of connected components) not greater than a user-defined threshold, α, whose values can vary dynamically during the search process. The value of the threshold α and its tuning during the iterations of the algorithm need to be chosen in an appropriate way. Indeed, a small value of α results in few labels in the restricted candidate list, giving a large intensification capability and a small diversification capability. This means that the resulting algorithm is very fast, but it can easily become trapped at a local optimum. Conversely, a large value of α produces an algorithm with a large diversification capability, but a short intensification capability, because many candidate labels are included in the restricted candidate list. In our implementation, we found an adequate trade-off between intensification and diversification capabilities by considering the following scheme. In order to fill the restricted candidate list we fix the threshold as the minimum number of connected components

Algorithm 2: Greedy Randomized Adaptive Search Procedure for the MLST problem

Input: A labelled, undirected, connected graph $G = (V, E, L)$ with n vertices, m edges, ℓ labels;
Output: A spanning tree T;
Initialisation:
- Let $C \leftarrow 0$ be the set of used labels, and $H = (V, E(C))$ the subgraph of G restricted to V and edges with labels in C, where $E(C) = \{e \in E : L(e) \in C\}$;
- Let $Comp(C)$ be the number of connected components of $H = (V, E(C))$;
- Let $C' \leftarrow L$ be the global set of used labels, and $H' = (V, E(C'))$ the subgraph of G restricted to V and edges with labels in C', where $E(C') = \{e \in E : L(e) \in C'\}$;

begin
  repeat
    Set $C \leftarrow 0$ and update $H = (V, E(C))$;
    Construction-Phase(C);
    Local-Search(C);
    if $|C| < |C'|$ **then**
      Move $C' \leftarrow C$ and update $H' = (V, E(C'))$;
    end
  until *termination conditions* ;
  $\Rightarrow$ Take any arbitrary spanning tree T of $H' = (V, E(C'))$.
end

Procedure Construction-Phase(C):
Let $RCL_\alpha \leftarrow 0$ be the restricted candidate list of length α;
if *Number of iterations* > 2 **then**
  Set $RCL_\alpha \leftarrow L$ and $\alpha = \ell$;
  Select at random a label $c \in RCL_\alpha$;
  Add label c to the set of used labels: $C \leftarrow C \cup \{c\}$;
  Update $H = (V, E(C))$ and $Comp(C)$;
end
while $Comp(C) > 1$ **do**
  Set $RCL_\alpha \leftarrow \{\forall c \in L /$ minimizes $Comp(C \cup \{c\})\}$;
  Select at random a label $c \in RCL_\alpha$;
  Add label c to the set of used labels: $C \leftarrow C \cup \{c\}$;
  Update $H = (V, E(C))$ and $Comp(C)$;
end

Procedure Local-Search(C):
for $i = 1$ *to* $|C|$ **do**
  Delete label i from the set C, i.e. $C \leftarrow C - \{i\}$;
  Update $H = (V, E(C))$ and $Comp(C)$;
  if $Comp(C) > 1$ **then** Add label i to the set C, i.e. $C \leftarrow C \cup \{i\}$;
  Update $H = (V, E(C))$ and $Comp(C)$;
end

produced by the candidate labels. This means that only the labels producing the least number of connected components constitute the restricted candidate list. Furthermore, after the two first iterations, complete randomisation is used to choose the initial label to add, taking inspiration from the Pilot Method. This corresponds to setting the threshold to $+\infty$, and all the labels of the graph are present within the restricted candidate list (length α= total number of labels, ℓ). To intensify the search for the remaining labels to add, the list is filled considering only the labels leading to the minimum total number of connected components, as in the previous iterations.

At the end of the construction phase, a local search phase is included (*Local-Search(C)* procedure). It simply consists of trying to drop some labels, one by one, from the current

solution C while retaining feasibility. Local search gives a further improvement to the intensification phase of the algorithm.

The entire algorithm proceeds until the user termination conditions are satisfied.

3.4. *Variable Neighbourhood Search*

Variable Neighbourhood Search (VNS) is a new and widely applicable metaheuristic based on dynamically changing neighbourhood structures during the search process. VNS does not follow a trajectory, but it searches for new solutions in increasingly distant neighbourhoods of the current solution, jumping only if a better solution than the current best solution is found (Mladenović and Hansen, 1997; Hansen and Mladenović, 2001, 2003).

At the starting point, it is required to define a suitable neighbourhood structure. The simplest and most common choice is a structure in which the neighbourhoods have increasing cardinality: $|N_1(C)| < |N_2(C)| < ... < |N_{k_{max}}(C)|$. The process of changing neighbourhoods when no improvement occurs diversifies the search. In particular the choice of neighbourhoods of increasing cardinality yields a progressive diversification. The VNS approach can be summarized as: *"One Operator, One Landscape"*, meaning that promising zones of the search space given by a specific neighbourhood may not be promising for other neighbourhoods (landscape). A local optimum with respect to a given neighbourhood may not be locally optimal with respect to another neighbourhood.

VNS provides a general framework and many variants exist for specific requirements. Our implementation for the MLST is described in Algorithm 3. Given a labelled graph $G = (V, E, L)$ with n vertices, m edges, and ℓ labels, each solution is encoded by a binary string, i.e. $C = (c_1, c_2, \ldots, c_\ell)$ where

$$c_i = \begin{cases} 1 & \textit{if label } i \textit{ is in solution } C \\ 0 & \textit{otherwise} \end{cases} \qquad (\forall i = 1, \ldots, \ell). \tag{1}$$

The algorithm starts from an initial feasible solution C generated at random and lets parameter k vary during the execution. The successive shaking phase (*Shaking-Phase($N_k(C)$)* procedure) represents the core idea of VNS: it changes the neighbourhood structure when the local search is trapped at a local minimum. This is implemented by the random selection of a point C' within the neighbourhood $N_k(C)$ of the current solution C. The random point C' is generated in order to avoid cycling, which might occur if a deterministic rule is used.

In the shaking phase, in order to impose a neighbourhood structure on the solution space S, comprising all possible solutions, we define the distance between any two such solutions $C_1, C_2 \in S$, as the Hamming distance:

$$\rho(C_1, C_2) = |C_1 - C_2| = \sum_{i=1}^{\ell} \lambda_i \tag{2}$$

where $\lambda_i = 1$ if label i is included in one of the solutions but not in the other, and 0 otherwise, $\forall i = 1, ..., \ell$. Then, given a solution C, we consider its k^{th} neighbourhood, $N_k(C)$, as all the different sets having a Hamming distance from C equal to k labels, where $k = 1, 2, \ldots, k_{max}$, and k_{max} represents the size of the shaking. In a more formal

Algorithm 3: Variable Neighbourhood Search for the MLST problem

Input: A labelled, undirected, connected graph $G = (V, E, L)$ with n vertices, m edges, ℓ labels;
Output: A spanning tree T;
Initialisation:
- Let $C \leftarrow 0$ be the global set of used labels, and $H = (V, E(C))$ the subgraph of G restricted to V and edges with labels in C, where $E(C) = \{e \in E : L(e) \in C\}$;
- Let C' be a set of labels, and $H' = (V, E(C'))$ the subgraph of G restricted to V and edges with labels in C', where $E(C') = \{e \in E : L(e) \in C'\}$;
- Let $Comp(C')$ be the number of connected components of $H' = (V, E(C'))$;
begin
  C = *Generate-Initial-Solution-At-Random()*;
  repeat
    Set $k = 1$ and $k_{max} = (|C| + |C|/3)$;
    while $k < k_{max}$ **do**
      C' = *Shaking-Phase*$(N_k(C))$;
      Local-Search(C');
      if $|C'| < |C|$ **then** Move $C \leftarrow C'$, and set $k = 1$ and $k_{max} = (|C| + |C|/3)$;
      else Increase the size of the neighbourhood structure: $k = k + 1$;
    end
  until *termination conditions* ;
  Update $H = (V, E(C))$;
  $\Rightarrow$ Take any arbitrary spanning tree T of $H = (V, E(C))$.
end

Procedure Shaking-Phase$(N_k(C))$***:***
Set $C' \leftarrow C$;
for $i = 1$ *to* k **do**
  Select at random a number between 0 and 1: $rnd = random(0, 1)$;
  if $rnd \leq 0.5$ **then** Delete at random a label $c' \in C'$ from C', i.e. $C' \leftarrow C' - \{c'\}$;
  else Add at random a label $c' \in (L - C)$ to C', i.e. $C' \leftarrow C' \cup \{c'\}$;
  Update $H' = (V, E(C'))$ and $Comp(C')$;
end

Procedure Local-Search(C):
while $Comp(C') > 1$ **do**
  Let S be the set of unused labels which minimize the number of connected components, i.e. $S = \{e \in (L - C') : \min Comp(C' \cup \{e\})\}$;
  Select at random a label $u \in S$;
  Add label u to the set of used labels: $C' \leftarrow C' \cup \{u\}$;
  Update $H' = (V, E(C'))$ and $Comp(C')$;
end
for $i = 1$ *to* $|C'|$ **do**
  Delete label i from the set C', i.e. $C' \leftarrow C' - \{i\}$;
  Update $H' = (V, E(C'))$ and $Comp(C')$;
  if $Comp(C') > 1$ **then** Add label i to the set C', i.e. $C' \leftarrow C' \cup \{i\}$;
  Update $H' = (V, E(C'))$ and $Comp(C')$;
end

way, the k^{th} neighbourhood of a solution C is defined as $N_k(C) = \{S \subset L : (\rho(C, S)) = k\}$, where $k = 1, ..., k_{max}$.

The value of k_{max} is an important parameter to tune in order to obtain an optimal balance between intensification and diversification capabilities. Choosing a small value for k_{max} produces a high intensification capability and a small diversification capability, resulting in a fast algorithm, but with a large probability of being trapped at a local minimum. Conversely, a large value for k_{max} decreases the intensification capability and

increases the diversification capability, resulting in a slower algorithm, but able to escape from local minima. According to our experience, the value $k_{max} = (|C| + |C|/3)$ gives a good trade-off between these two factors.

In our implementation, in order to select a solution in the k-th neighbourhood of a solution C, the algorithm randomly adds further labels to C, or remove labels from C, until the resulting solution has a Hamming distance equal to k with respect to C. Addition and deletion of labels at this stage have the same probability of being chosen. For this purpose, a random number is selected between 0 and 1 ($rnd = random(0, 1)$). If this number is smaller than 0.5, the algorithm proceeds with the deletion of a label from C. Otherwise, an additional label is included at random in C from the set of unused labels $(L - C)$. The procedure is repeated until the number of addition/deletion operations is exactly equal to k.

The successive local search (*Local-Search(C')* procedure) consists of two steps. In the first step, since deletion of labels often gives an infeasible incomplete solution, additional labels may be added in order to restore feasibility. In this case, addition of labels follows the MVCA criterion of adding the label with the minimum number of connected components. Note that in case of ties in the minimum number of connected components, a label not yet included in the partial solution is chosen at random within the set of labels producing the minimum number of components (i.e. $u \in S$ where $S = \{e \in (L - C') : \min Comp(C' \cup \{e\})\}$). Then, the second step of the local search tries to delete labels one by one from the specific solution, whilst maintaining feasibility.

After the local search phase, if no improvements are obtained ($|C'| \geq |C|$), the neighbourhood structure is increased ($k = k+1$) giving a progressive diversification ($|N_1(C)| < |N_2(C)| < ... < |N_{k_{max}}(C)|$). Otherwise, the algorithm moves to the improved solution ($C \leftarrow C'$) and sets the first neighbourhood structure ($k = 1$). Then the procedure restarts with the shaking and the local search phases, continuing iteratively until the user termination conditions (maximum allowed CPU time, maximum number of iterations, or maximum number of iterations between two successive improvements) are satisfied.

4. Computational results

In this section, the metaheuristics are compared in terms of solution quality and computational running time. We identify the metaheuristics with the abbreviations: PILOT (Pilot Method), MGA (Modified Genetic Algorithm), GRASP (Greedy Randomized Adaptive Search Procedure), VNS (Variable Neighbourhood Search).

Different sets of instances of the problem have been generated 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 the number of edges of the graph (m), the number of nodes of the graph (n), and the number of labels assigned to the edges (ℓ).

We thank the authors of (Cerulli et al., 2005), who kindly provided data for use in our experiments. In our computations, run on a Pentium Centrino microprocessor at 2.0 GHz with 512 MB RAM, we consider different datasets, each one containing 10 instances of the problem with the same set of values for the parameters n, ℓ, and m. For each dataset, solution quality is evaluated as the average objective value for the 10 problem instances. A maximum allowed CPU time (*max-CPU-time*), determined with respect to the dimension

of the problem instance, is chosen as the stopping condition for all the metaheuristics. For MGA, we use a variable number of iterations for each instance, determined such that the computations take approximately *max-CPU-time* for the specific dataset. Selection of the maximum allowed CPU time as the stopping criterion is made in order to have a direct comparison of all the metaheuristics with respect to the quality of their solutions.

All the heuristic methods run for *max-CPU-time* and, in each case, the best solution is recorded. The computational times reported in the tables are the times at which the best solutions are obtained. The reported times have precision of ± 5 ms. Where possible, the results of the metaheuristics are compared with the exact solution, identified with the label EXACT.

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 attempts to find a better solution from the current incomplete solution. The main program that solves the MLST problem calls the *Test* procedure with an empty set of labels. The details are specified in Algorithm 4.

In order to reduce the number of test sets, it is more convenient to use a good approximate solution for C^* in the initial step, instead of considering all the labels. Another improvement that avoids the examination of a large number of incomplete solutions consists of rejecting every incomplete solution that cannot be completed to get only one connected component. Note that if we are evaluating an incomplete solution C' with a number of labels $|C'| = |C^*|-2$, we should try to add the labels one by one to check if it is possible to find a better solution for C^* with a smaller dimension, that is $|C'| = |C^*|-1$. To complete this solution C', we need to add a label with a frequency at least equal to

Algorithm 4: Exact Method for the MLST problem

Input: A labelled, undirected, connected graph $G = (V, E, L)$ with n vertices, m edges, ℓ labels;
Output: A spanning tree T;
Initialisation:
- Let $C \leftarrow 0$ be the initially empty set of used labels;
- Let $H = (V, E(C))$ be the subgraph of G restricted to V and edges with labels in C, where $E(C) = \{e \in E : L(e) \in C\}$;
- Let $C^* \leftarrow L$ be the global set of used labels;
- Let $H^* = (V, E(C^*))$ be the subgraph of G restricted to V and edges with labels in C^*, where $E(C^*) = \{e \in E : L(e) \in C^*\}$;
- Let $Comp(C)$ be the number of connected components of $H = (V, E(C))$;

begin
    Call $Test(C)$;
    $\Rightarrow$ Take any arbitrary spanning tree T of $H^* = (V, E(C^*))$.
end

Procedure Test(C):
if $|C| < |C^*|$ **then**
    Update $Comp(C)$;
    if $Comp(C) = 1$ **then**
        Move $C^* \leftarrow C$;
    else if $|C| < |C^*| - 1$ **then**
        foreach $c \in (L - C)$ **do**
            Try to add label $c : Test(C \cup \{c\})$;
        end
    end
end

the actual number of connected components minus 1. If this requirement is not satisfied, the incomplete solution can be rejected, speeding up the search process.

The running time of this Exact Method grows exponentially, but if either the problem size is small or the optimal objective function value is small, the running time is reasonable and the method obtains the exact solution. The complexity of the instances increases with the dimension of the graph (number of nodes and labels), and the reduction in the density of the graph. In our tests, the optimal solution is reported unless a single instance requires more than 3 hours of CPU time. In such a case, we report not found (NF).

4.1. *Experimental analysis*

In our computations we have considered two different groups of datasets, including instances with a number of vertices, n, and a number of labels, ℓ, from 20 up to 500. All these instances are available from the authors (Consoli, 2007). The number of edges, m, is obtained indirectly from the density d of edges whose values are chosen to be 0.8, 0.5, and 0.2. Analysing the performance of the considered algorithms, for a single dataset a metaheuristic should be considered *worse* than another one if either it obtains a larger average objective value, or an equal average objective value but in a greater computational time.

Group 1 examines small instances with the number of vertices equal to the number of labels. These values are chosen to be between 20 and 50 in steps of 10. Thus, the considered datasets are $n = \ell = 20$, 30, 40, 50, and $d = 0.8$, 0.5, 0.2, for a total of 12 datasets (120 instances). Computational results are presented in Table 1, which reports the average objective function values found by the heuristics for the datasets of Group 1, and the corresponding average computational times, with a *max-CPU-time* of 1 second.

- INSERT TABLE 1 -

Looking at this table, all the heuristics performed well and faster than the Exact Method for the Group 1 instances. However, MGA is considerably slower than the other metaheuristics, as a result of a poor intensification capability and an excessive diversification capability for these instances. PILOT is faster than MGA but it produces slightly worse solutions with respect to solution quality. It exhibits an opposite behaviour to that of MGA, being characterised by a limited diversification capability which sometimes does not allow the search process to escape from local optima. The performance of GRASP and VNS are both comparable for these trivial instances of the problem. They are able to obtain all the exact solutions in very short running times and are the best performing heuristics for the Group 1 in terms of solution quality and computational running time.

Group 2 considers larger instances of the MLST problem with a fixed number of vertices, and a number of labels $\ell = 0.25 \cdot n$, $0.5 \cdot n$, n, $1.25 \cdot n$. Thus, the datasets of Group 2 are $n = 100$, 200, 500 vertices, $\ell = 0.25 \cdot n$, $0.5 \cdot n$, n, $1.25 \cdot n$, and $d = 0.8$, 0.5, 0.2, for a total of 36 datasets (360 instances). Furthermore, we have considered a *max-CPU-time* of 20 seconds for Group 2 with $n = 100$; of 60 seconds for Group 2 with $n = 200$; and of 300 seconds for Group 2 with $n = 500$. Average objective function values and the corresponding average computational times are reported in Tables 2 - 3 - 4 respectively.

- INSERT TABLE 2, TABLE 3, TABLE 4-

For all the Group 2 instances with $n = 100$, looking at Table 2, the best performance is obtained by VNS which produces the solutions with the best solution quality and the shortest running times. GRASP also performs well, obtaining the same solutions as VNS, with the exception of the instance $[n = \ell = 100,\ d = 0.2]$. As in Group 1, PILOT and MGA obtain the worse solutions and they confirm their defects: excessive diversification and poor intensification capabilities for MGA and, conversely, excessive intensification and poor diversification capabilities for PILOT.

Table 3 and Table 4 with larger instances of the problem (Group2 with $n = 200$, and Group 2 with $n = 500$) show the same relative behaviour for all the considered metaheuristics. VNS and GRASP are always the best performing methods, indicating an optimal tuning between intensification and diversification of the search process, which evidently is not obtained by PILOT and MGA that obtain the worse solution in terms of solution quality and computational running time. VNS always obtains the solutions with the best quality, but it loses a lot, sometimes, in terms of computational running time (see for example the instances $[n = \ell = 200,\ d = 0.2]$, $[n = \ell = 500,\ d = 0.2]$, and $[n = 500, \ell = 625,\ d = 0.2]$). From this analysis, perhaps GRASP slightly defects in terms of exploration of the search space with respect to the VNS approach.

Considering only solution quality, the average values of the objective function of the metaheuristics among all the considered datasets are: PILOT = 5.66, MGA = 5.68, GRASP = 5.61, VNS = 5.59. Thus, the best ranking with respect to the solution quality (from the best to the worst) is: VNS, GRASP, PILOT, MGA.

4.2. *Statistical analysis of the results*

Computing only the average objective values of the metaheuristics over multiple data does not provide a full comparison between them. Averages are susceptible to outliers: they can allow excellent performance on some datasets to compensate for an overall bad performance. There may be situations in which such behaviour is desired. However, in general we prefer algorithms that behave well on as many problems as possible.

We have carried out tests to determine the statistical significance of differences between the performances of the metaheuristics (Hollander and Wolfe, 1999). The issue of statistical tests for comparison of algorithms on multiple datasets was theoretically and empirically reviewed by Demšar (2006). The null-hypothesis being tested is that the metaheuristics have equal mean performance and the observed differences are merely random. The alternative hypothesis is that the algorithms have different mean performances of statistical significance.

The most common statistical method for testing differences between more than two algorithms is Analysis of Variance (ANOVA) (see (Hollander and Wolfe, 1999; Demšar, 2006) for more details). Since ANOVA is based on assumptions that are violated in this context, we make use of the *Friedman Test* (Friedman, 1940), that is the non-parametric equivalent of ANOVA, and its corresponding *Nemenyi Post-hoc Test* (Nemenyi, 1963).

According to the Friedman test, the statistical significance of differences between the metaheuristics is examined by testing whether the measured average ranks are significantly different from the overall mean rank. In particular, we use the version of the Friedman test developed by Iman and Davenport (1980), which considers a powerful test statistic F_F (Appendix A). If the equivalence of the algorithms is rejected, the Nemenyi

post-hoc test is applied in order to perform pairwise comparisons.

To perform the Friedman and Nemenyi tests, the ranks of the algorithms for each dataset are evaluated, with a rank of 1 assigned to the best performing algorithm, rank 2 to the second best one, and so on. The average ranks for each metaheuristic among the 48 datasets are: PILOT = 3.30, MGA = 3.48, GRASP = 1.76, VNS = 1.46. According to the ranking, VNS is the best performing algorithm, immediately followed by GRASP, then PILOT and MGA achieving the worst results.

Now, we analyse the statistical significance of differences between these ranks. Consider the Iman and Davenport (1980) version of the Friedman test for $k = 4$ algorithms and $N = 48$ datasets. The value of the F_F test statistic, which is distributed according to the F-distribution with $(k-1, (k-1)(N-1)) = (3, 141)$ degrees of freedom, is computed. This value is 124.2, which is greater than the critical value (3.92 for $\alpha = 1\%$, where α is the significance level of the test expressed as percentage). Thus, a significant difference between the performance of the metaheuristics exists, according to the Friedman test.

As the equivalence of the algorithms is rejected, we proceed with the Nemenyi post-hoc test. Considering a significance level $\alpha = 1\%$, the critical value is $q_{0.01} \cong 3.11$. The critical difference (CD) for the Nemenyi test is

$$CD = 3.11 \cdot \sqrt{\frac{4 \cdot 5}{6 \cdot 48}} \cong 0.82; \tag{3}$$

The differences between the average ranks of the metaheuristics are reported in Table 5.

- INSERT TABLE 5 -

From this table, we can identify two groups of metaheuristics. The first group includes VNS and GRASP, while the second group includes PILOT and MGA. Considering a significance level $\alpha = 1\%$, the algorithms within each group have comparable performance according to the Nemenyi test since, in each case, the value of the test statistic is less than the critical difference. Conversely, two algorithms belonging to different groups have significantly different performance according to the Nemenyi test.

Summarizing, from the Friedman and Nemenyi statistical tests, VNS and GRASP have comparable performance, and they are the best performing algorithms. On the other hand, PILOT and MGA have comparable performance, but worse than VNS and GRASP.

Another way to compare the performance of the algorithms is to count the number of times they generate the optimal solution. In particular, counting the overall number of exact solutions obtained is a good approach to estimate the diversification capability of each metaheuristic. The Exact Method obtains the exact solution for all problem instances of 32 datasets, among the overall 48 datasets; for the remaining sets NF is reported. Therefore, the total number of instances having the exact solution is: $32 \times 10 = 320$.

The percentages of the number of optimal solutions obtained by the metaheuristics among the 320 instances are (ranking from the best to the worst algorithm): VNS = 100, GRASP = 99.7, MGA = 99.7, PILOT = 97.5.

VNS obtains all the optimal solutions, underlying a high exploration capability even for complex instances. In the same way, GRASP and MGA offer very good results, missing only 1 solution out of 320, although MGA is extremely time consuming. With 8 cases (out of 320), PILOT fails to find the global optimum and becomes trapped at a local optimum.

Furthermore, some optima reached by the metaheuristics require a greater computational time than required by the Exact Method, thus nullifying the purpose of the metaheuristics. In this sense the best performances are obtained by VNS and GRASP, all of which require less computational time than the Exact Method among the 32 datasets. In contrast, PILOT and MGA obtain the optimal solution but in a time that exceeds that of the Exact Method in 11 and 18 datasets, respectively. Although MGA reaches more exact solutions than PILOT, it is computationally more burdensome.

From this further analysis, the results reinforce the conclusion that VNS and GRASP are effective metaheuristics for the MLST problem. They are particularly recommended for the proposed problem thanks to the following features: ease of implementations, user-friendly codes, high quality of the solutions, and shorter computational running times.

5. Conclusions and further research

In this paper, we have studied metaheuristics for the minimum labelling spanning tree (MLST) problem. In particular, we have examined and implemented the metaheuristics recommended in the literature: the Modified Genetic Algorithm (MGA) of Xiong et al. (2006) and the Pilot Method (PILOT) of Cerulli et al. (2005). Furthermore, some new implementations for the MLST problem have been proposed: a Greedy Randomized Adaptive Search Procedure (GRASP) and a basic Variable Neighbourhood Search (VNS).

Computational experiments were performed using different instances of the MLST problem to evaluate how the algorithms are influenced by the parameters, the structure of the network, and the distribution of the labels on the edges. Applying the nonparametric statistical tests of Friedman (1940) and Nemenyi (1963), we concluded that VNS and GRASP have significantly better performance than the other methods recommended in the literature with respect to solution quality and running time. Furthermore, this result has been reinforced by comparing the metaheuristics with an exact approach. VNS and GRASP obtain a large number of optimal or near-optimal solutions, showing an enhanced diversification capability.

All the results allow us to state that VNS and GRASP are fast and extremely effective metaheuristics for the MLST problem. They are particularly recommended for the proposed problem because of their simplicity and their ability to obtain high-quality solutions in short computational running times.

Future research will consist of trying to further improve the performance of these procedures (for example through hybridization with other metaheuristics) particularly for large instances of the problem. For this purpose, an algorithm based on Ant Colony Optimisation (ACO) is currently under study in order to try to obtain a larger diversification capability by extending the current greedy MVCA local search. Indeed, a proper ACO implementation may allow moves to worse solutions by providing an alternative probabilistic solution construction mechanism.

Appendix A. Statistical tests

Friedman Test (Friedman, 1940): The Friedman test is a non-parametric statistical test that examines the existence of significant differences between the performances of multiple algorithms over different datasets. Given k algorithms and N datasets, it ranks

the algorithms for each dataset separately, and tests whether the measured average ranks are significantly different from the mean rank. The statistic used by Friedman (1940) is

$$\chi_F^2 = \frac{12 \cdot N}{k \cdot (k+1)} \cdot \left[\sum_j R_j^2 - \frac{k \cdot (k+1)}{4} \right], \tag{4}$$

which follows a Chi-Square distribution with $(k-1)$ degrees of freedom.

Iman and Davenport (1980) developed a more powerful version of the Friedman test by considering the following statistic:

$$F_F^2 = \frac{(N-1) \cdot \chi_F^2}{N \cdot (k-1) - \chi_F^2} \ , \tag{5}$$

which is distributed according to the F-distribution with $(k-1)$ and $(k-1) \cdot (N-1)$ degrees of freedom. For more details, see (Demšar, 2006).

Nemenyi Test (Nemenyi, 1963): The Nemenyi test is used to perform pairwise comparisons of multiple algorithms over different datasets (Nemenyi, 1963). The performance of two algorithms is considered significantly different if the corresponding average ranks differ by at least the critical difference (CD):

$$CD = q_\alpha \cdot \sqrt{\frac{k \cdot (k+1)}{6 \cdot N}} \ , \tag{6}$$

where k is the number of the metaheuristics, N the number of datasets, q_α the critical value, and α the significance level of the statistical test. For more details, see (Demšar, 2006).

Acknowledgments

Sergio Consoli was supported by an E.U. Marie Curie Fellowship for Early Stage Researcher Training (EST-FP6) under grant number MEST-CT-2004-006724 at Brunel University (project NET-ACE).

José Andrés Moreno Pérez was supported by the projects TIN2005-08404-C04-03 of the Spanish Government (with financial support from the European Union under the FEDER project) and PI042005/044 of the Canary Government.

We gratefully acknowledge this support.

References

Avis, D., Hertz, A., Marcotte, O., 2005. Graph theory and combinatorial optimization. Springer-Verlag, New York.

Brüggemann, T., Monnot, J., Woeginger, G. J., 2003. Local search for the minimum label spanning tree problem with bounded colour classes. Operations Research Letters 31, 195–201.

Cerulli, R., Fink, A., Gentili, M., Voß, S., 2005. Metaheuristics comparison for the minimum labelling spanning tree problem. In: Golden, B., Raghavan, S., Wasil, E. (Eds.), The next wave in computing, optimization, and decision technologies. Springer-Verlag, Berlin, pp. 93–106.

Chang, R. S., Leu, S. J., 1997. The minimum labeling spanning trees. Information Processing Letters 63 (5), 277–282.

Consoli, S., March 2007. Test datasets for the minimum labelling spanning tree problem. [online], http://people.brunel.ac.uk/~mapgssc/MLSTP.htm.

Demšar, J., 2006. Statistical comparison of classifiers over multiple data sets. Journal of Machine Learning Research 7, 1–30.

Duin, C., Voß, S., 1999. The pilot method: A strategy for heuristic repetition with applications to the Steiner problem in graphs. Networks 34 (3), 181–191.

Feo, T. A., Resende, M. G. C., 1995. Greedy randomized adaptive search procedures. Journal of Global Optimization 6 (2), 109–133.

Friedman, M., 1940. A comparison of alternative tests of significance for the problem of m rankings. The Annals of Mathematical Statistics 11 (1), 86–92.

Glover, F., Kochenberger, G. A., 2003. Handbook of metaheuristics (International series in Operations Research & Management Science). Kluwer Academic Publishers, Norwell, MA.

Goldberg, D. E., Deb, K., Korb, B., 1991. Don't worry, be messy. In: Proceedings of the 4th International Conference on Genetic Algorithms. Morgan-Kaufmann, La Jolla, CA, pp. 24–30.

Hansen, P., Mladenović, N., 2001. Variable neighbourhood search: Principles and applications. European Journal of Operational Research 130, 449–467.

Hansen, P., Mladenović, N., 2003. Variable neighbourhood search. In: Glover, F., Kochenberger, G. A. (Eds.), Handbook of metaheuristics. Kluwer Academic Publishers, Norwell, MA, Ch. 6, pp. 145–184.

Hollander, M., Wolfe, D. A., 1999. Nonparametric statistical methods, 2nd Edition. John Wiley & Sons, New York.

Iman, R. L., Davenport, J. M., 1980. Approximations of the critical region of the Friedman statistic. Communications in Statistics - Theory and Methods 9, 571–595.

Krumke, S. O., Wirth, H. C., 1998. On the minimum label spanning tree problem. Information Processing Letters 66 (2), 81–85.

Mladenović, N., Hansen, P., 1997. Variable neighbourhood search. Computers & Operations Research 24, 1097–1100.

Nemenyi, P. B., 1963. Distribution-free multiple comparisons. Ph.D. thesis, Princeton University, New Jersey.

Resende, M. G. C., Ribeiro, C. C., 2003. Greedy randomized adaptive search procedure. In: Glover, F., Kochenberger, G. A. (Eds.), Handbook of metaheuristics. Kluwer Academic Publishers, Norwell, MA, pp. 219–249.

Tanenbaum, A. S., 2003. Computer networks, 4th Edition. Prentice Hall, Englewood Cliffs, New Jersey.

Van-Nes, R., 2002. Design of multimodal transport networks: A hierarchical approach. Delft University Press, Delft.

Voß, S., Fink, A., Duin, C., 2005. Looking ahead with the pilot method. Annals of Operations Research 136, 285–302.

Voß, S., Martello, S., Osman, I. H., Roucairol, C., 1999. Meta-heuristics: Advances and trends in local search paradigms for optimization. Kluwer Academic Publishers, Norwell, MA.

Wan, Y., Chen, G., Xu, Y., 2002. A note on the minimum label spanning tree. Information Processing Letters 84, 99–101.

Xiong, Y., Golden, B., Wasil, E., 2005a. A one-parameter genetic algorithm for the minimum labeling spanning tree problem. IEEE Transactions on Evolutionary Computation 9 (1), 55–60.

Xiong, Y., Golden, B., Wasil, E., 2005b. Worst-case behavior of the MVCA heuristic for the minimum labeling spanning tree problem. Operations Research Letters 33 (1), 77–80.

Xiong, Y., Golden, B., Wasil, E., 2006. Improved heuristics for the minimum label spanning tree problem. IEEE Transactions on Evolutionary Computation 10 (6), 700–703.

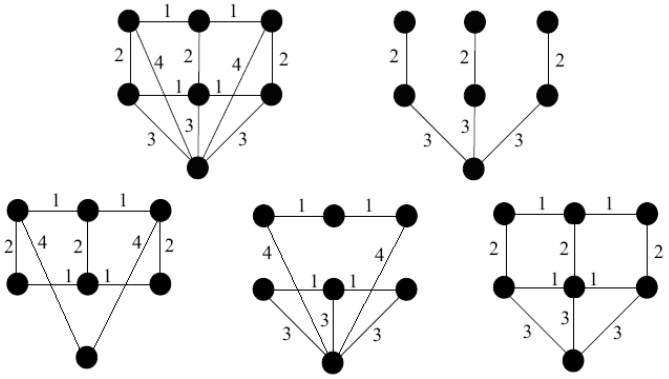

Figure 1. The top two graphs show a sample graph and its optimal solution. The bottom three graphs show some feasible solutions.

Table 1
Computational results for Group 1 (*max-CPU-time* for heuristics = 1000 ms)

Parameters			**Average objective function values**				
n	ℓ	d	EXACT	PILOT	MGA	GRASP	VNS
		0.8	*2.4*	2.4	2.4	2.4	2.4
20	20	0.5	*3.1*	3.2	3.1	3.1	3.1
		0.2	*6.7*	6.7	6.7	6.7	6.7
		0.8	*2.8*	2.8	2.8	2.8	2.8
30	30	0.5	*3.7*	3.7	3.7	3.7	3.7
		0.2	*7.4*	7.4	7.4	7.4	7.4
		0.8	*2.9*	2.9	2.9	2.9	2.9
40	40	0.5	*3.7*	3.7	3.7	3.7	3.7
		0.2	*7.4*	7.6	7.4	7.4	7.4
		0.8	*3*	3	3	3	3
50	50	0.5	*4*	4	4.1	4	4
		0.2	*8.6*	8.6	8.6	8.6	8.6
	TOTAL:		*55.7*	56	55.8	55.7	55.7

Parameters			**Computational times (milliseconds)**				
n	ℓ	d	EXACT	PILOT	MGA	GRASP	VNS
		0.8	*0*	0	15.6	1.6	0
20	20	0.5	*0*	1.6	22	0	0
		0.2	*11*	3.1	23.4	0	1.6
		0.8	*0*	3	9.4	1.6	0
30	30	0.5	*0*	3.1	26.5	0	0
		0.2	*138*	4.7	45.4	1.5	5.2
		0.8	*2*	6.3	12.5	1.5	0
40	40	0.5	*3.2*	7.9	28.2	1.5	3.1
		0.2	*100.2*$\cdot 10^3$	10.8	120.3	15.6	9.6
		0.8	*3.1*	17.1	21.8	3	0
50	50	0.5	*21.9*	20.2	531.3	9.4	4.1
		0.2	*66.3*$\cdot 10^3$	17.2	93.6	3.2	11.9
	TOTAL:		*166.7*$\cdot 10^3$	95	950	38.9	35.5

Table 2
Computational results for Group 2 with $n = 100$ (*max-CPU-time* for heuristics = $20{\cdot}10^3$ ms)

Parameters			**Average objective function values**				
n	ℓ	d	EXACT	PILOT	MGA	GRASP	VNS
		0.8	*1.8*	1.8	1.8	1.8	1.8
	25	0.5	*2*	2	2	2	2
		0.2	*4.5*	4.5	4.5	4.5	4.5
		0.8	*2*	2	2	2	2
	50	0.5	*3*	3.1	3	3	3
100		0.2	*6.7*	6.9	6.7	6.7	6.7
		0.8	*3*	3	3	3	3
	100	0.5	*4.7*	4.7	4.7	4.7	4.7
		0.2	*NF*	10.1	9.9	9.8	9.7
		0.8	*4*	4	4	4	4
	125	0.5	*5.2*	5.4	5.2	5.2	5.2
		0.2	*NF*	11.2	11.1	11	11
TOTAL:			-	58.7	57.9	57.7	57.6

Parameters			**Computational times (milliseconds)**				
n	ℓ	d	EXACT	PILOT	MGA	GRASP	VNS
		0.8	*9.4*	4.7	26.5	0	0
	25	0.5	*14*	12.6	29.7	4.6	0
		0.2	*34.3*	23.2	45.3	9.3	3.1
		0.8	*17.8*	67.3	23.5	6.4	7.7
	50	0.5	*23.5*	90.7	106.2	51.6	42.4
100		0.2	*10.2*$\cdot 10^3$	103.2	148.3	57.8	49.7
		0.8	*142.8*	378.1	254.7	61	215
	100	0.5	*2.4*$\cdot 10^3$	376.2	300	28.2	114.7
		0.2	*NF*	399.9	$9.4{\cdot}10^3$	$1.2{\cdot}10^3$	414.8
		0.8	*496.9*	565.7	68.7	9.4	10.1
	125	0.5	*179.6*$\cdot 10^3$	576.3	759.4	595.4	551.1
		0.2	*NF*	634.5	$2{\cdot}10^3$	562.9	420.4
TOTAL:			-	$3.2{\cdot}10^3$	$13.2{\cdot}10^3$	$2.6{\cdot}10^3$	$1.8{\cdot}10^3$

Table 3
Computational results for Group 2 with $n = 200$ (*max-CPU-time* for heuristics = $60{\cdot}10^3$ ms)

Parameters			**Average objective function values**				
n	ℓ	d	EXACT	PILOT	MGA	GRASP	VNS
		0.8	*2*	2	2	2	2
	50	0.5	*2.2*	2.2	2.2	2.2	2.2
		0.2	*5.2*	5.2	5.2	5.2	5.2
		0.8	*2.6*	2.6	2.6	2.6	2.6
	100	0.5	*3.4*	3.4	3.4	3.4	3.4
200		0.2	*NF*	8.3	8.3	8.1	7.9
		0.8	*4*	4	4	4	4
	200	0.5	*NF*	5.5	5.4	5.4	5.4
		0.2	*NF*	12.4	12.4	12.2	12
		0.8	*4*	4	4	4.1	4
	250	0.5	*NF*	6.3	6.3	6.3	6.3
		0.2	*NF*	13.9	14	13.9	13.9
	TOTAL:		-	69.8	69.8	69.4	68.9

Parameters			**Computational times (milliseconds)**				
n	ℓ	d	EXACT	PILOT	MGA	GRASP	VNS
		0.8	*29.7*	90.7	26.5	20.5	0
	50	0.5	*32.7*	164.1	68.8	14.2	17.2
		0.2	*5.4*$\cdot 10^3$	320.4	326.6	37.5	241.3
		0.8	*138.6*	876.5	139.3	45.3	123.2
	100	0.5	*807.8*	$1.2{\cdot}10^3$	$1.6{\cdot}10^3$	176.6	151.1
200		0.2	*NF*	$1.3{\cdot}10^3$	$2.2{\cdot}10^3$	667.2	$1.7{\cdot}10^3$
		0.8	*22.5*$\cdot 10^3$	$5.9{\cdot}10^3$	204.6	43.6	32
	200	0.5	*NF*	$5.6{\cdot}10^3$	$16.1{\cdot}10^3$	885.6	971.9
		0.2	*NF*	$5{\cdot}10^3$	$12.7{\cdot}10^3$	$9.4{\cdot}10^3$	$12.8{\cdot}10^3$
		0.8	*20.6*$\cdot 10^3$	$9.1{\cdot}10^3$	$2.2{\cdot}10^3$	$4.9{\cdot}10^3$	$1.1{\cdot}10^3$
	250	0.5	*NF*	$8.4{\cdot}10^3$	$17.6{\cdot}10^3$	506	$3.4{\cdot}10^3$
		0.2	*NF*	$8{\cdot}10^3$	$26.4{\cdot}10^3$	$1.4{\cdot}10^3$	$3.2{\cdot}10^3$
	TOTAL:		-	$45.9{\cdot}10^3$	$79.6{\cdot}10^3$	$18.1{\cdot}10^3$	$23.7{\cdot}10^3$

Table 4
Computational results for Group 2 with $n = 500$ (*max-CPU-time* for heuristics = $300 \cdot 10^3$ ms)

Parameters			**Average objective function values**				
n	ℓ	d	EXACT	PILOT	MGA	GRASP	VNS
		0.8	*2*	2	2	2	2
	125	0.5	*2.6*	2.6	2.6	2.6	2.6
		0.2	*NF*	6.3	6.2	6.2	6.2
		0.8	*3*	3	3	3	3
	250	0.5	*NF*	4.2	4.3	4.2	4.1
500		0.2	*NF*	9.9	10.1	9.9	9.9
		0.8	*NF*	4.8	4.7	4.7	4.7
	500	0.5	*NF*	6.7	7.1	6.5	6.5
		0.2	*NF*	15.9	16.6	15.9	15.8
		0.8	*NF*	5.1	5.4	5.1	5.1
	625	0.5	*NF*	8.1	8.3	7.9	7.9
		0.2	*NF*	18.5	19.1	18.4	18.3
TOTAL:			-	87.1	89.4	86.4	86.1

Parameters			**Computational times (milliseconds)**				
n	ℓ	d	EXACT	PILOT	MGA	GRASP	VNS
		0.8	*370*	$3.4 \cdot 10^3$	18	152	17.1
	125	0.5	*597*	$6.6 \cdot 10^3$	$2.6 \cdot 10^3$	455	$1.1 \cdot 10^3$
		0.2	*NF*	$11.9 \cdot 10^3$	$57.1 \cdot 10^3$	$4 \cdot 10^3$	$3.9 \cdot 10^3$
		0.8	$5.3 \cdot 10^3$	$35.49 \cdot 10^3$	516	248	142.3
	250	0.5	*NF*	$65.3 \cdot 10^3$	$28 \cdot 10^3$	583	$84 \cdot 10^3$
500		0.2	*NF*	$156.4 \cdot 10^3$	$181.2 \cdot 10^3$	$3.3 \cdot 10^3$	$5.1 \cdot 10^3$
		0.8	*NF*	$200.5 \cdot 10^3$	$117.5 \cdot 10^3$	$28.1 \cdot 10^3$	$22.3 \cdot 10^3$
	500	0.5	*NF*	$190.1 \cdot 10^3$	$170.9 \cdot 10^3$	$90.9 \cdot 10^3$	$32.3 \cdot 10^3$
		0.2	*NF*	$300.6 \cdot 10^3$	$241.8 \cdot 10^3$	$20.2 \cdot 10^3$	$139.7 \cdot 10^3$
		0.8	*NF*	$184.3 \cdot 10^3$	$51.9 \cdot 10^3$	$4.9 \cdot 10^3$	$16.1 \cdot 10^3$
	625	0.5	*NF*	$200.9 \cdot 10^3$	$222.2 \cdot 10^3$	$35.7 \cdot 10^3$	$44.7 \cdot 10^3$
		0.2	*NF*	$289.9 \cdot 10^3$	$297.8 \cdot 10^3$	$53.1 \cdot 10^3$	$155.5 \cdot 10^3$
TOTAL:			-	$1645.3 \cdot 10^3$	$1371.5 \cdot 10^3$	$213.8 \cdot 10^3$	$504.9 \cdot 10^3$

Table 5
Pairwise differences of the average ranks of the algorithms (Critical difference = 0.82 for a significance level of 1% for the Nemenyi test)

ALGORITHM (average rank)	VNS (1.46)	GRASP (1.76)	PILOT (3.30)	MGA (3.48)
VNS (1.46)	-	0.3	**1.84**	**2.02**
GRASP (1.76)	-	-	**1.54**	**1.72**
PILOT (3.30)	-	-	-	0.18
MGA (3.48)	-	-	-	-