Solving the minimum labelling spanning tree problem using hybrid local search

S. Consoli [a,*], K. Darby-Dowman [a], N. Mladenović [a], and 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

Given a connected, undirected graph whose edges are labelled (or coloured), the minimum labelling spanning tree (MLST) problem seeks a spanning tree whose edges have the smallest number of distinct labels (or colours). In recent work, the MLST problem has been shown to be NP-hard and some effective heuristics (Modified Genetic Algorithm (MGA) and Pilot Method (PILOT)) have been proposed and analyzed. A hybrid local search method, that we call Group-Swap Variable Neighbourhood Search (GS-VNS), is proposed in this paper. It is obtained by combining two classic metaheuristics: Variable Neighbourhood Search (VNS) and Simulated Annealing (SA). Computational experiments show that GS-VNS outperforms MGA and PILOT. 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 heuristic.

Key words: Combinatorial optimization, Graphs and Networks, Local search, Minimum labelling spanning tree.
1991 MSC: 90C27, 90C35, 90C59

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

1. Introduction

In the *minimum labelling spanning tree* (MLST) problem, we are given an undirected graph with labelled edges as input. Each edge has a single label and different edges can have the same label. We can think of each label as a unique colour. The goal is to find a spanning tree with the minimum number of labels.

Such a model can represent many real-world problems. For example, in telecommunications networks, there are many different types of communications media, such as optical fibre, coaxial cable, microwave, and telephone line [1]. 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 may reduce the construction cost and the complexity of the network. Another example is given by multimodal transportation networks [2]. In such problems, it may be desirable to provide a complete service using the minimum number of companies. A multimodal transportation network can be represented by a graph where each edge is assigned a label, denoting a different company managing that link. 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 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 nodes, E is the set of edges, and L is the set of labels. The purpose is to 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, it is easier to work firstly in terms of feasible solutions. 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 first seek a feasible solution with the least number of labels [3].

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 MLST problem was first introduced by Chang and Leu [4]. They also proved that it is an NP-hard problem and provided a polynomial time heuristic, the *Maximum Vertex Covering Algorithm* (MVCA), for the problem. This heuristic was successively improved by Krumke and Wirth [5] as defined in Algorithm 1. The procedure starts with an empty graph. Successively, it adds at random one label from those labels that minimize the number of connected components. The procedure continues until only one connected component is left, i.e. when only a connected subgraph is obtained.

- INSERT FIGURE 2 -

Figure 2 shows how the MVCA by Krumke and Wirth [5] works on the graph of Figure 1. In the initial step, it adds label 1 because it gives the least number of connected components (3 components). In the second step, all the three remaining labels (2, 3 and

Input: A labelled, undirected, connected graph $G = (V, E, L)$, with n vertices, m edges, ℓ labels, and $Q \subseteq V$ basic nodes;
Output: A spanning tree T;
Initialization:
- 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

Algorithm 1: Maximum Vertex Covering Algorithm [5]

4) produce the same number of components (2). In this case, the algorithm selects at random and, for example, adds label 3. At this time, all the nodes of the graph are visited, but the subgraph is still disconnected. Thus, label 2 is added and only one connected component is obtained (equivalently, label 4 could have been added instead of label 2). Summarizing, the final solution is {1; 2; 3}, which is worse than the optimal solution {2; 3} of Figure 1.

Krumke and Wirth [5] also proved that MVCA can yield a solution with a value no greater than $(1+2\log n)$ times optimal, where n is the total number of nodes. Later, Wan et al. [6] obtained a better bound for the greedy algorithm introduced by Krumke and Wirth [5]. The algorithm was shown to be a $(1+\log(n-1))$-approximation for any graph with n nodes ($n > 1$).

Brüggemann et al. [7] 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. [8] derived tighter bounds than those proposed by Wan et al. [6]. 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 that is: $H_b = \sum_{i=1}^{b} \frac{1}{i} = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{b}$.

Subsequently, they constructed a worst-case family of graphs such that the MVCA solution is 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. [6].

The usual rule of Krumke and Wirth [5] to select the label that minimizes the total number of connected components at each step, results in fast and high-quality solutions. However, a difficulty arises when more than one label with same resulting minimum number of connected components is detected, in a specific step. Since, frequently, there are many labels with this minimum number of connected components, the results mainly

depend on the rule chosen to select a candidate from this set of ties. If the initial label from this set is chosen, the results are affected by the sorting of the labels. Therefore, different executions of the algorithm may result in different solutions, with a slightly different number of labels.

Other heuristic approaches to the MLST problem are proposed in the literature. For example, Xiong et al. [3] presented a *Genetic Algorithm* (GA) to solve the MLST problem, outperforming MVCA in most cases.

Subsequently, Cerulli et al. [9] applied the *Pilot Method*, greedy heuristic developed by Duin and Voß [10] and subsequently extended by Voß et al. [11], to the MLST problem. Considering different sets of instances of the MLST problem, Cerulli et al. [9] compared this method with other metaheuristics (Reactive Tabu Search, Simulated Annealing, and 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. [12] implemented modified versions of MVCA focusing on the initial label added. For example, after the labels have been sorted according to their frequencies, from highest to lowest, the modified version tries only the most promising 10% of the labels at the initial step. Afterwards, it runs MVCA to determine the remaining labels and then it selects the best of the $|L|/10$ resulting solutions (where L is the set of possible labels for all edges). Compared with the Pilot Method of Cerulli et al. [9], this version can potentially reduce the computational 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 the Pilot Method. Another modified version by Xiong et al. [12] is similar to the previous one, except that it tries the most promising 30% of the labels at the initial step. Then it runs MVCA to determine the remaining labels. Moreover, Xiong et al. [12] proposed another way to modify MVCA. They consider at each step the three most promising labels, and assign a different probability of selection that is proportional to their frequencies. Then, they randomly select one of these candidates, and add it to the incomplete solution. In addition, Xiong et al. [12] presented a *Modified Genetic Algorithm* (MGA) that was shown to have the best performance for the MLST problem in terms of solution quality and running time.

The structure of the paper is as follows. In the next section, we present the details of the algorithms considered in this paper. Section 3 contains a computational analysis and evaluation. Finally, conclusions are described in Section 4. For a survey on the basic concepts of metaheuristics and combinatorial optimization, the reader is referred to [13, 14].

2. Description of the algorithms

In this section, the details of the algorithms considered in this paper are specified. First, an exact method for the MLST problem is introduced and described. Then, the details of the heuristics, that are reported in the literature to be the best performing for the MLST problem, are considered: the *Modified Genetic Algorithm (MGA)* by Xiong et al. [12] and the *Pilot Method* by Cerulli et al. [9]. We then present a new approach to the problem, obtained by combining two classic metaheuristics: *Variable Neighbourhood Search* (VNS) [15, 16, 17] and *Simulated Annealing* (SA) [18, 19, 20]. We call this hybrid

local search as *Group-Swap Variable Neighbourhood Search* (GS-VNS).

2.1. *Exact Method*

This exact approach to the MLST problem is based on an A* or backtracking procedure to test the subsets of L. It 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 2.

Input: A labelled, undirected, connected graph $G = (V, E, L)$, with n vertices, m edges, ℓ labels, and $Q \subseteq V$ basic nodes;
Output: A spanning tree T;
Initialization:
- 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

Algorithm 2: Exact Method for the MLST problem

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 result in a single 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 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 "acceptable" 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).

2.2. *Modified Genetic Algorithm [12]*

Genetic Algorithms are based on the principle of evolution, operations such as crossover and mutation, and the concept of fitness [21]. 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. [12] the only parameter to tune is the population size.

Input: A labelled, undirected, connected graph $G = (V, E, L)$, with n vertices, m edges, ℓ labels, and $Q \subseteq V$ basic nodes;
Output: A spanning tree T;
Initialization:
- Let $C \leftarrow 0$ be the initially empty set of used labels for each iteration;
- Let $H = (V, E(C))$ be the subgraph of G restricted to V and edges with labels in C, where $E(C) = \{e \in E : L(e) \in C\}$;
- Set the size P_n of the population;

begin
  $(s[0], s[1], \ldots, s[P_n - 1]) \leftarrow$*Initialize-Population*$(G, P_n)$;
  repeat
    for $i = 1$ *to* $P_n/2$ **do**
      for $j = 1$ *to* $P_n/2$ **do**
        $t[1] \leftarrow s[j]$;
        $t[2] \leftarrow s[mod((j + i), P_n)]$;
        $t_{crossovered} \leftarrow$*Crossover*$(t[1], t[2])$;
        $t_{mutated} \leftarrow$*Mutation*$(t_{crossovered})$;
        if $t_{mutated} < t[1]$ **then** $t[1] \leftarrow t_{mutated}$;
      end
    end
  until *termination conditions* ;
  $C =$ *Extract-the-Best* $(s[0], s[1], \ldots, s[P_n - 1])$;
  Update $H = (V, E(C))$;
  $\Rightarrow$ Take any arbitrary spanning tree T of $H = (V, E(C))$.
end

Algorithm 3: The Modified Genetic Algorithm for the MLST problem [12]

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 [5] 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. The details of the Modified Genetic Algorithm for the MLST problem are specified in Algorithm 3.

2.3. *Pilot Method [9]*

The Pilot Method is a metaheuristic proposed by Duin and Voß [10] and Voß et al. [11]. 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 extends 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 the most promising of such elements, i.e. $f(i^*) < f(i)$, $\forall i \notin M$. 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 forms the output of the procedure.

For the MLST problem, the Pilot Method proposed by Cerulli et al. [9] starts from the null solution (an empty set of labels) as master solution, uses the revised MVCA of Krumke and Wirth [5] 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. The details are specified in Algorithm 4.

The method 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

Input: A labelled, undirected, connected graph $G = (V, E, L)$, with n vertices, m edges, ℓ labels, and $Q \subseteq V$ basic nodes;
Output: A spanning tree T;
Initialization:
- Let $M \leftarrow 0$ be the master solution, and $H = (V, E(M))$ the subgraph of G restricted to V and edges with labels in M, where $E(M) = \{e \in E : L(e) \in M\}$;
- Let $Comp(M)$ be the number of connected components of $H = (V, E(M))$;
- Let $M^* \leftarrow L$ be a set of labels, and $H^* = (V, E(M^*))$ the subgraph of G restricted to V and edges with labels in M^*, where $E(M^*) = \{e \in E : L(e) \in M^*\}$;
- Let i^* be the best candidate move;
begin
  while (*termination conditions*) *OR* $(Comp(M) > 1)$ **do**
    foreach $i \in (L - M)$ **do**
      Add label i to the master solution: $M \leftarrow M \cup \{i\}$;
      Update $H = (V, E(M))$ and $Comp(M)$;
      while $Comp(M) > 1$ **do**
        Let $S = \{e \in (L - M) : \min Comp(M \cup \{e\})\}$ be the set of unused labels with the minimum number of connected components;
        Select at random a label $u \in S$;
        Add label u to the solution: $M \leftarrow M \cup \{u\}$;
        Update $M = (V, E(M))$ and $Comp(M)$;
      end
      Local Search(M);
      if $|M| < |M^*|$ **then**
        Update the best candidate move $i^* \leftarrow i$;
        Keep the solution produced by the best move: $M^* \leftarrow M$;
      end
      Delete label i from the master solution: $M \leftarrow M \cup \{i\}$;
      Update $H = (V, E(M))$ and $Comp(M)$;
    end
    Update the master solution with the best move: $M \leftarrow M \cup \{i^*\}$;
  end
  while $Comp(M) > 1$ **do**
    Let $S = \{e \in (L - M) : \min Comp(M \cup \{e\})\}$ be the set of unused labels with the minimum number of connected components;
    Select at random a label $u \in S$;
    Add label u to the solution: $M \leftarrow M \cup \{u\}$;
    Update $M = (V, E(M))$ and $Comp(M)$;
  end
  $\Rightarrow$ Take any arbitrary spanning tree T of $H = (V, E(M))$.
end

Algorithm 4: The Pilot Method for the MLST problem [9]

```
Procedure Local Search(M):
for j = 1 to |M| do
    Delete label j from the set M, i.e. M ← M − {j};
    Update H = (V, E(M)) and Comp(M);
    if Comp(M) > 1 then
        Add label j to the set M, i.e. M ← M ∪ {j};
        Update H = (V, E(M)) and Comp(M);
    end
end
```

Algorithm 5: Procedure Local Search(·)

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. The last master solution represents the output of the method. A local search mechanism is further included at the end of the computation in order to try to greedily drop labels whilst retaining feasibility (see Algorithm 5).

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 $O(\ell^3)$.

2.4. *Hybrid local search*

A current trend in the area of combinatorial optimization is the integration of good characteristics from one or more metaheuristics within the implementation of another "pure" one, in order to improve its performance. Often, this produces new methods that cannot be classified within a defined heuristic class, but are referred to as *hybrid metaheuristics* [14, 22].

For example, a current trend is the integration of trajectory methods within population-based ones. The strength of population-based methods is the concept of recombining solutions. It allows the population-based methods to perform "big" guided steps in the search space, usually larger than the ones performed by trajectory methods. The strength of trajectory methods is based on a local search procedure which is able to strictly explore a promising region in the search space. In this way, the danger of being close to good solutions but "missing" them is not as high as in population-based methods. Summarizing, population-based methods tend to be better at identifying promising areas in the search space, whereas trajectory methods tend to be superior in exploring specific zones of the domain. Thus, hybrid local search methods, combining the advantages of population-based methods with the power of trajectory methods, are often very successful [14].

In many cases, hybrid algorithms are more complex to implement compared to a pure one. Thus, the application of hybrid local search to a combinatorial optimization problem must be justified by establishing its effective performance with respect to that problem.

In this section we introduce a hybrid local search method for the MLST problem, obtained by combining *Variable Neighbourhood Search* (VNS) [15, 16, 17] and *Simulated*

Annealing (SA) [18, 19, 20]. We call this hybrid metaheuristic *Group-Swap Variable Neighbourhood Search* (GS-VNS).

2.4.1. *Variable Neighbourhood Search [15, 16, 17]*

Variable Neighbourhood Search is a relatively new and widely applicable metaheuristic based on dynamically changing neighbourhood structures during the search process [15, 16, 17]. VNS doesn't 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. The basic VNS procedure, is specified in Algorithm 6. At the starting point, it is required to define a suitable neighbourhood structure of size k_{max} (user parameter to be set). The simplest and most common choice is a structure in which the neighbourhoods have increasing cardinality: $|N_1(\cdot)| < |N_2(\cdot)| < ... < |N_{k_{max}}(\cdot)|$. 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.

VNS starts from an initial solution C (e.g. generated at random) with k increasing from 1 up to k_{max} during the progressive execution. The basic idea of VNS to change the neighbourhood structure, when the search is trapped at a local minimum, is implemented by the Shaking phase (*Shaking phase($N_k(C)$)* procedure). It consists of the random selection of a point C' in the neighbourhood $N_k(C)$ of the current solution C. It may provide a better starting point for the successive local search phase (*Local Search((C'))*), which tries to improve, if possible, the current solution (C'). Afterwards, if no improvements are obtained ($|C'| \geq |C|$) in the move phase, the neighbourhood structure is increased ($k = k + 1$) giving a progressive diversification ($|N_1(C)| < |N_2(C)| < ... < |N_{k_{max}}(C)|$). Otherwise, if an improved solution C' is obtained ($|C'| < |C|$), it becomes the best solu-

Initialization:
- Define the neighbourhood structure $N_k(\cdot)$, with $k = 1, 2, ..., k_{max}$. and where k_{max} represents the size of the neighbourhood structure (e.g. increasingly distant neighbourhoods: $|N_1(\cdot)| < ... < |N_{k_{max}}(\cdot)|$);
- Let C be the best solution to date;
- Let C' be a generic solution;

begin
  C = *Generate-Initial-Solution()*;
  repeat
    Set $k = 1$;
    while $k < k_{max}$ **do**
      C' = *Shaking phase($N_k(C)$)*;
      Local Search(C');
      if $|C'| < |C|$ **then**
        Move $C \leftarrow C'$;
        Set $k = 1$;
      else
        Increase the size of the neighbourhood structure: $k = k + 1$;
      end
    end
  until *termination conditions* ;
end

Algorithm 6: Variable Neighbourhood Search [15, 16, 17]

tion to date ($C \leftarrow C'$) and the algorithm restarts from the first neighbourhood ($k = 1$) of the best solution to date ($N_1(C)$). The algorithm proceeds until the user termination conditions (maximum allowed CPU time, maximum number of iterations, or maximum number of iterations between two successive improvements) are satisfied.

2.4.2. *Simulated Annealing [18, 19, 20]*

Simulated Annealing (SA) is possibly the oldest probabilistic local search method for global optimization problems, and surely one of the first to clearly provide a way to escape from local traps. It was independently invented by Kirkpatrick et al. [18], and by Cerny [19]. The SA metaheuristic performs a stochastic search of the neighbourhood space. In the case of a minimization problem, modifications to the current solution that increase the value of the objective function are allowed in SA, in contrast to classical descent methods where only modifications that decrease the objective value are possible.

The name and inspiration of this method come from the process of annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. The heat causes the atoms to become unstuck from their initial positions (a local minimum of the internal energy) and wander randomly through states of higher energy; the slow cooling provides an opportunity to find configurations with lower internal energy than the initial one.

By analogy with this physical process, each step of the SA algorithm replaces the current solution by a random "nearby" solution, chosen with a probability that depends on the difference between the corresponding function values and on a global parameter T (called *temperature*), that is gradually decreased during the process (cooling process).

The dependency is such that the current solution changes arbitrarily in the search domain when T is large, i.e. at the beginning of the algorithm, through uphill moves (or random walks) that saves the method from becoming trapped at a local minimum. Afterwards, the temperature T is gradually decreased, intensifying the search process in the specific promising-zone of the domain (downhill moves).

More precisely, the current solution is always replaced by a new one if this modification reduces the objective value, while a modification increasing the objective value by Δ is only accepted with a probability $\exp(-\Delta/T)$ (Boltzmann function). At a high temperature, the probability of accepting an increase to the objective value is high (uphill moves: high diversification and low intensification). Instead, this probability gets lower as the temperature is decreased (downhill moves: high intensification and low diversification).

The process described is memory-less because it follows a trajectory in the state space in which the successor state is chosen depending only on the incumbent one, without taking into account the history of the search process.

The initial temperature value (T_0), the number of iterations to be performed (k_{max}), the temperature value at each step (T_k), the cooling (reduction) rate of T, and the stopping criterion are determined by a so-called *cooling schedule* (or *cooling law*), generally specified by the following rule:

$$T_{k+1} = func(T_k, k), \tag{1}$$

where, given an iteration k (with $k = 1, ..., k_{max}$), T_{k+1} represents the temperature value in the successive step $k + 1$. Different cooling schedules may be considered, such as a *logarithmic cooling law*:

$$T_{k+1} = \frac{cost}{\lg(k + k_0)}, \tag{2}$$

where *cost* and k_0 are arbitrary constant values that must be set experimentally by the user. Sometimes, the logarithmic cooling law is too slow for practical purposes. Therefore, faster cooling schedule techniques may be adopted, such as a *geometric cooling law*, which is a cooling rule with an exponential decay of the temperature:

$$T_{k+1} = \alpha \cdot T_k, \tag{3}$$

where $\alpha \in [0, 1]$.

Other complex cooling techniques can be used in order to improve the performance of the SA algorithm. For example, to have an optimal balance between diversification and intensification, the cooling rule may be updated during the search process. At the beginning, T can be constant or linearly decreasing to have a high diversification factor for a larger exploration of the domain; after that, T can follow a fast rule, such as the geometric one, to converge quickly to a local optimum.

Simulated Annealing has been applied to several combinatorial problems with success, such as the Quadratic Assignment problem and the Job Shop Scheduling problem [14]. Rather than as a stand-alone algorithm, it is nowadays used as a component in hybrid metaheuristics to improve performance in specific applications, as is our case with the MLST problem.

2.4.3. *Group-Swap Variable Neighbourhood Search*

Variable Neighbourhood Search provides a general framework and many variants have been proposed in the literature to try to improve its performance in some circumstances [17]. For example, Pérez et al. [23] proposed a hybridization between VNS and a path-relinking metaheuristic to solve the p-hub median problem, while Pacheco et al. [24] mixed VNS and Tabu search for the variable selection and the determination of the coefficients for these variables that provide the best linear discrimination function, with the objective of obtaining a high classification success rate.

Although hybridizing a metaheuristic may increase the complexity of the implementation, we consider a more advanced VNS version, with some new features, for the MLST problem, with a view to obtaining improved results.

For our implementation, 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

$$c_i = \begin{cases} 1 & \textit{if label } i \textit{ is in solution } C \\ 0 & \textit{otherwise} \end{cases} \qquad (\forall i = 1, \ldots, |L|). \tag{4}$$

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:

$$\rho(C_1, C_2) = |C_1 - C_2| = \sum_{i=1}^{\ell} \lambda_i \tag{5}$$

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 where k_{max} represents the size of the Shaking phase. In order to construct the neighbourhood of a solution C, the algorithm first proceeds with the deletion of labels from C. In other words, given a solution C, its k^{th} neighbourhood, $N_k(C)$, consists of all the different sets obtained from C by removing k labels, where $k = 1, 2, ..., k_{max}$. In a more formal way, given a solution C, its k^{th} neighbourhood is defined as $N_k(C) = \{S \subset L : (|C| - |S|) = k\}$, where $k = 1, 2, ..., k_{max}$. In particular, we start from an initial feasible solution generated at random and let parameter k_{max} vary during the execution. At each iteration of the Shaking phase, k_{max} is set to the number of labels of the current feasible solution whose neighbourhood is being explored ($k_{max} = |C|$). Since deletion of labels often gives an infeasible solution, additional labels may be added in order to restore feasibility.

The first variant with respect to the basic VNS that we propose consists of introducing a *Group-Swap* operation, which extracts a feasible solution from the *complementary space* of the current solution. The complementary space of a solution C is defined as the set of all the labels that are not contained in C, that is $(L - C)$. To yield the solution, the Group-Swap applies a constructive heuristic, such as the MVCA, to the subgraph of G with labels in the complementary space of the current solution. Then, the Shaking phase is applied to this solution, according to the basic VNS.

In order to illustrate the Group-Swap procedure, consider the example shown in Figure 3. Given an initial random solution X_0, the algorithm searches for new solutions in increasingly distant neighbourhoods of X_0. In this example, no better solutions are detected, and the current solution is still X_0. Now, the Group-Swap procedure is applied to X_0. It consists of extracting a feasible solution from the complementary space of X_0, defined as $(L - X_0)$. Let the new solution be X_0^{swap}. Then, the algorithm searches for new solutions in the neighbourhoods of X_0^{swap}. In this example, a better solution X_1 is found. The algorithm continues with this procedure until the termination conditions are satisfied. In the example, the final solution is denoted by X_2.

- INSERT FIGURE 3 -

We propose this variant in order to improve the diversification of the search process. A VNS implementation with the Group-Swap feature has been compared with the previous algorithms, resulting in good performance. However, in order to seek further improvements, we have introduced another variation. We propose another heuristic to yield solutions from the complementary space of the current solution, in order to further improve the diversification by allowing worse components to be added to incomplete solutions. We call this heuristic *Probabilistic MVCA*. The introduction of a probabilistic element within the Probabilistic MVCA heuristic is inspired by Simulated Annealing (SA). However, the Probabilistic MVCA does not work with complete solutions but with partial solutions created with components added at each step. The resulting algorithm that combines the basic VNS and the Probabilistic MVCA, represents a hybridization between VNS and SA metaheuristics.

The Probabilistic MVCA heuristic could be classified as another version of MVCA, but with a probabilistic choice of the next label. It extends basic local search by allowing moves to worse solutions. Starting from an initial solution, successively a candidate move is randomly selected; this move is accepted if it leads to a solution with a better objective function value than the current solution, otherwise the move is accepted with

a probability that depends on the deterioration Δ of the objective function value.

Following the SA criterion, the acceptance probability is computed according to the Boltzmann function as $\exp(-\Delta/T)$, using the temperature (T) as control parameter. The value of T is initially high, which allows many worse moves to be accepted, and is gradually reduced following a specific cooling schedule. The aim is to allow, with a specified probability, worse components with a higher number of connected components to be added to incomplete solutions.

Probability values assigned to each colour are inversely proportional to the number of components they give. So the colours with a lower number of connected components will have a higher probability of being chosen. Conversely, colours with a higher number of connected components will have a lower probability of being chosen. Thus, the possibility of choosing less promising labels is allowed.

Summarizing, at each step the probabilities of selecting colours giving a smaller number of components will be higher than the probabilities of selecting colours with a higher number of components. Moreover, these differences in probabilities increase step by step as a result of the reduction of the temperature for the cooling schedule. It means that the difference between the probabilities of two colours giving different numbers of components is higher as the algorithm proceeds. The probability of a colour with a high number of components will decrease as the algorithm proceeds and will tend to zero. In this sense, the search becomes MVCA-like.

As an example, consider a graph with four colours a, b, c, and d. Starting from an empty incomplete solution, the first label is added. The numbers of connected components the colours give are evaluated. Suppose they give $a \Rightarrow 8$, $b \Rightarrow 4$, $c \Rightarrow 6$, $d \Rightarrow 2$ components. The smaller number of components is 2 given by d. Call this colour s. To select the next label to add, it is necessary to compute the probabilities for each colour. For a generic candidate colour k, to evaluate the probability of it being added to the current solution C, we need to compute the Boltzmann function $\exp(-\Delta/T)$, that is:

$$\exp\left(-\frac{Comp(C \cup k) - Comp(C \cup s)}{T}\right), \tag{6}$$

where $Comp(C \cup k)$ is the number of connected components given by adding the colour k, while $Comp(C \cup s)$ is the minimum number of connected components, given by adding the colour s.

For simplicity, consider a linear cooling law for the temperature T, that is $T_{|C|} = \frac{1}{|C|+1}$, where C is the current incomplete solution. The temperature T will have value $1/1 = 1$ in the initial step (that is when we need to add the initial colour), $1/2 = 0.5$ in the second step, $1/3 = 0.33$ in the third step, and so on. Therefore, in the initial step the Boltzmann values for each colour are: $a \Rightarrow 0.0024$, $b \Rightarrow$, $c \Rightarrow 0.018$, $d \Rightarrow 1$. After having evaluated the Boltzmann values, they are normalized to lie in the interval $[0, 1]$, giving the probabilities for each colour to be selected. Thus, the probabilities (expressed as percentages) are: $a \Rightarrow 0.2\%$, $b \Rightarrow 11.7\%$, $c \Rightarrow 1.6\%$, $d \Rightarrow 86.5\%$. We select at random one colour according to these probabilities. Suppose colour c is selected.

As the current solution is not a single connected component, we need to add a second colour. In this second step we need to compute again the probabilities, but with a temperature equal to 0.5. Suppose the numbers of connected components that the remaining colours give are: $a \Rightarrow 3$, $b \Rightarrow 2$, $d \Rightarrow 2$. The smaller number of components is 2 given by both b and d. Thus, in this second step ($T = 0.5$), the Boltzmann function for a generic

candidate colour k to add is given by $\exp(-\Delta/T) = \exp\left(-\frac{Comp(C \cup k)-2)}{0.5}\right)$, resulting in the following values: $a \Rightarrow 0.135$, $b \Rightarrow 1$, $d \Rightarrow 1$. They are normalized to lie in the interval $[0, 1]$, and resulting in the probabilities (expressed as percentages): $a \Rightarrow 6.3\%$, $b \Rightarrow 46.8\%$, $d \Rightarrow 46.8\%$. We select at random one colour according to these probabilities, and so on. The algorithm proceeds until we have only one single connected component.

Obviously, in a complex problem such as the MLST problem, the linear cooling law $T_{|C|} = \frac{1}{|C|+1}$ for the temperature is not satisfactory. After having tested different cooling laws, the best performance was obtained by using a geometric cooling schedule: $T_{k+1} = \alpha \cdot T_k = \alpha^k \cdot T_0$, where $\alpha \in [0, 1]$. This cooling law is very fast for the MLST problem, yielding a good balance between intensification and diversification. The initial temperature value T_0 and the value of α need to be evaluated experimentally.

A VNS implementation using the Probabilistic MVCA as a constructive heuristic has been tested. However, the best results were obtained by combining the Group-Swap operation with the Probabilistic MVCA constructive heuristic within the VNS, obtaining the hybrid metaheuristic that we call Group-Swap VNS. The Probabilistic MVCA is applied both to the Group-Swap operation (in order to obtain a solution from the complementary

Input: A labelled, undirected, connected graph $G = (V, E, L)$, with n vertices, m edges, ℓ labels, and $Q \subseteq V$ basic nodes;
Output: A spanning tree T;
Initialization:
- Let $Best_C \leftarrow 0$ be the global set of colours, and $H^{BEST} = (V, E(Best_C))$ be the subgraph of G restricted to V and edges with labels in $Best_C$, where $E(Best_C) = \{e \in E : L(e) \in Best_C\}$;
- Let $C \leftarrow 0$ be the set of used colours, 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 of $H = (V, E(C))$;
- Let C' be a set of colours, 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 $COMPL \leftarrow (L - Best_C)$ the complementary space of the best solution $Best_C$;
begin
    $Best_C$ = *Generate-Initial-Solution-At-Random()*;
    Local-Search($Best_C$*)*;
    repeat
        Perform the swapping of the best solution: C = *Group-Swap(*$Best_C$*)*;
        while $|C| < |Best_C|$ **do** Continue to swap: C = *Group-Swap(*$Best_C$*)*;
        Set $k = 1$ and $k_{max} = |C|$;
        while $k < k_{max}$ **do**
            C' = *Shaking phase(*$N_k(C)$*)*;
            Local-Search(C'*)*;
            if $|C'| < |C|$ **then**
                Move $C \leftarrow C'$;
                Set $k = 1$ and $k_{max} = |C|$;
            else Increase the size of the neighbourhood structure: $k = k + 1$;
        end
        if $|C| < |Best_C|$ **then** Move $Best_C \leftarrow C$;
    until *termination conditions* ;
    Update $H^{BEST} = (V, E(Best_C))$;
    $\Rightarrow$ Take any arbitrary spanning tree T of $H^{BEST} = (V, E(Best_C))$.
end

Algorithm 7: Group-Swap Variable Neighbourhood Search for the MLST problem

Procedure Group-Swap($Best_C$):
Set $C \leftarrow 0$;
while $Comp(C) > 1$ **do**
  foreach $c \in COMPL$ **do**
    Geometric Group-Swap cooling schedule for the temperature:
    $T^{GroupSwap}(|C|+1) = \frac{T^{GroupSwap}(0)}{\alpha^{|C|}}$ where $\begin{cases} T^{GroupSwap}(0) = |Best_C| \\ \alpha = |Best_C| \end{cases}$;
    Calculate the probabilities $P(c)$ for each colour, normalizing the values given by the Boltzmann function: $\exp\left(-\frac{(Comp(C\cup\{c\})-Comp(C\cup\{s\}))}{T^{GroupSwap}(|C|+1)}\right)$ where $s \in COMPL$ is the colour which minimizes $Comp(C \cup \{s\})$;
  end
  Select at random an unused colour $u \in COMPL$ following the probabilities $P(\cdot)$;
  Add label u to the set of used colours: $C \leftarrow C \cup \{u\}$;
  Update $H = (V, E(C))$ and $Comp(C)$;
end

Algorithm 8: Procedure Group-Swap(·)

space of the current solution) and in the Shaking phase (to restore feasibility by adding colours to incomplete solutions).

The details of the implementation of the GS-VNS are specified in Algorithm 7. We start from an initial feasible solution generated at random, denoted by $Best_C$. Then the Group-Swap operation is applied to $Best_C$, as shown in Algorithm 8, obtaining a solution C from the complementary space of $Best_C$ by means the Probabilistic MVCA constructive heuristic. For the geometric schedule in the Group-Swap procedure, computational experiments have shown that $T_0 = |Best_C|$ and $\alpha = |Best_C|$, where $Best_C$ is

Procedure Shaking phase($N_k(C)$):
for $i = 1$ *to* k **do**
  Select at random a colour $c' \in C'$;
  Delete label c' from the set of used colours: $C' \leftarrow C' - \{c'\}$;
  Update $H' = (V, E(C'))$ and $Comp(C')$;
end
Let $s \in (L - C')$ be the colour that minimizes $Comp(C' \cup \{s\})$;
while $Comp(C') > 1$ **do**
  foreach $c \in (L - C')$ **do**
    Geometric Shaking cooling schedule for the temperature:
    $T^{Shaking}_{(|C'|+1)} = \frac{T^{Shaking}_{(0)}}{\alpha^{|C'|}}$ where $\begin{cases} T^{Shaking}(0) = |Best_C|^2 \\ \alpha = |Best_C| \end{cases}$;
    Calculate the probabilities $P(c)$ for each colour, normalizing the values given by the Boltzmann function: $\exp\left(-\frac{(Comp(C'\cup\{c\})-Comp(C'\cup\{s\}))}{T^{Shaking}(|C|+1)}\right)$ where $s \in (L - C')$ is the colour which minimizes $Comp(C' \cup \{s\})$;
  end
  Select at random an unused colour $u \in (L - C')$ following the probabilities $P(\cdot)$;
  Add label u to the set of used colours: $C' \leftarrow C' \cup \{u\}$;
  Update $H' = (V, E(C'))$ and $Comp(C')$;
end

Algorithm 9: Procedure Shaking phase(·)

the current best solution, are values that performed well. So, the resulting cooling law for the Group-Swap procedure is

$$T^{GroupSwap}_{(|C|+1)} = \frac{T^{GroupSwap}_{(0)}}{\alpha^{|C|}} = \frac{1}{|Best_C|^{(|C|-1)}}. \quad (7)$$

Subsequently, the Shaking phase is applied to the resulting solution C (see Algorithm 9). The Shaking phase consists of the random selection of a point C' in the neighbourhood $N_k(C)$ of the current solution C ($N_k(C) = \{S \subset L : (|C| - |S|) = k\}$, where $k = 1, 2, ..., k_{max}$). At each iteration of the Shaking phase, k_{max} is set to the number of colours of the current feasible solution whose neighbourhood is being explored ($k_{max} = |C|$). In order to restore feasibility, the addition of colours at this step is according to the Probabilistic MVCA constructive heuristic. For the geometric schedule in the Shaking phase, computational experiments have shown that $T_0 = |Best_C|^2$ and $\alpha = |Best_C|$, where $Best_C$ is the current best solution, are values that performed well. The corresponding geometric cooling law is

$$T^{Shaking}_{(|C'|+1)} = \frac{T^{Shaking}_{(0)}}{\alpha^{|C'|}} = \frac{1}{|Best_C|^{(|C'|-2)}}. \quad (8)$$

The successive local search is the same as that used in the Pilot Method (see Algorithm 5) which tries to delete colours one by one from the specific solution (C'), whilst maintaining feasibility. Afterwards, if no improvements are obtained ($|C'| > |C|$), the neighbourhood structure is changed ($k = k + 1$) giving a progressive diversification ($|N_1(C)| < |N_2(C)| < ... < |N_{k_{max}}(C)|$). Otherwise (i.e. if $|C'| < |C|$), the algorithm moves to the solution C' ($C \leftarrow C'$) restarting the search with the smallest neighbourhood ($k = 1$). The algorithm proceeds with the same procedure until the user termination conditions (maximum allowed CPU time, maximum number of iterations, or maximum number of iterations between two successive improvements) are satisfied.

3. Computational results

To test the performance and the efficiency of the algorithms presented in this paper, we randomly generate instances of the MLST problem based on the number of nodes (n), the density of the graph (d), and the number of labels (ℓ). In our experiments, we consider 48 different datasets, each one containing 10 instances of the problem (yielding a total of 480 instances), including instances with number of vertices, n, and number of labels, ℓ, from 20 up to 500. The number of edges, m, is obtained indirectly from the density d, whose values are chosen to be 0.8, 0.5, and 0.2. The complexity of the instances increases with the dimension of the graph (i.e. increasing n and/or ℓ), and the reduction in the density of the graph. All the considered data are available from the authors in [25].

For each dataset, solution quality is evaluated as the average objective value among the 10 problem instances. A maximum allowed CPU time, that we call *max-CPU-time*, is chosen as the stopping condition for all the metaheuristics, determined with respect to the dimension of the problem instance. For MGA, we set the number of generations for each instance 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 the metaheuristics with respect to the quality of their solutions.

Our results are reported in Tables 1 - 4. All the computations have been made on a Pentium Centrino microprocessor at 2.0 GHz with 512 MB RAM. In each table, the first three columns show the parameters characterizing the different datasets (n, ℓ, d), while the remaining columns give the computational results of the considered algorithms, which are identified with the abbreviations: EXACT (Exact Method), PILOT (Pilot Method), MGA (Modified Genetic Algorithm), GS-VNS (Group-Swap Variable Neighbourhood Search).

- INSERT TABLE 1, TABLE 2, TABLE 3, TABLE 4 -

All the metaheuristics run for the *max-CPU-time* specified in each table 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, except in the case of the exact method, where the exact solution is reported unless a single instance computes for more than 3 hours of CPU time. In the case that no solution is obtained in *max-CPU-time* by the metaheuristics (in 3 hours by the exact method) a not found status (NF) is reported in the tables. All the reported times have precision of ± 5 ms. The performance of an algorithm can be considered better than another one if either it obtains a smaller average objective value, or an equal average objective value but in a shorter computational running time.

The motivation for a high diversification capability in GS-VNS is to obtain a better performance in large problem instances. Inspection of the tables shows that this aim is achieved. GS-VNS is the best performing algorithm for all the considered datasets. It is interesting to note that in all the problem instances for which the exact method obtains the solution, GS-VNS also yielded the exact solution.

To confirm this evaluation, the ranks of the algorithms for each dataset are evaluated, with a rank of 1 assigned to the best performing algorithm, a rank of 2 to the second best one, and so on. Obviously, if an algorithm records a NF for a dataset, the worst rank is assigned to that method in the specified dataset. The average ranks of the algorithms, among the considered datasets, are shown in Table 5, in which the algorithms are ordered from the best one to the worst one with respect to the average ranks.

- INSERT TABLE 5 -

According to the ranking, GS-VNS is the best performing algorithm, followed respectively by PILOT, MGA, and EXACT. Thus, this evaluation further indicates the superiority of GS-VNS with respect to the other approaches.

To analyse the statistical significance of differences between the evaluated ranks, we make use of the *Friedman Test* [26] and its corresponding *Nemenyi Post-hoc Test* [27]. For more details on the issue of statistical tests for comparison of algorithms over multiple datasets see [28, 29].

According to the Friedman Test, a significant difference between the performance of the metaheuristics, with respect to the evaluated ranks, exists (at the 1% of significance level). Since the equivalence of the algorithms is rejected, the Nemenyi post-hoc test is applied in order to perform pairwise comparisons. It considers the performance of two algorithms significantly different if their corresponding average ranks differ by at least a specific threshold critical difference (CD). In our case, considering a significance level of the Nemenyi test of 1%, this critical difference (CD) is 0.82. The differences between the average ranks of the algorithms are reported in Table 6.

- INSERT TABLE 6 -

From this table, it is possible to identify two groups of algorithms with different performance. The best performing group consists of just GS-VNS, because it obtains the smallest rank which is significantly different from all the other ranks. The remaining group consists of PILOT, MGA, and EXACT, which have comparable performance according to the Nemenyi test (since, in each case, the value of the test statistic is less than the critical difference 0.82), but worse than that of GS-VNS.

Furthermore, some exact solutions reached by the metaheuristics require a greater computational time than required by the Exact Method. (note that the Exact Method obtains the exact solution for all problem instances of 32 datasets out of a total of 48 datasets; for the remaining sets NF is reported). Again, the best performances are obtained by GS-VNS, which requires 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 14 and 16 datasets, respectively. Although PILOT and MGA reach most of the exact solutions, they require high computational effort.

The results demonstrate that GS-VNS is an effective metaheuristic for the MLST problem, producing high quality solutions in short computational running times.

4. Conclusions

In this paper, we have studied algorithms to solve the minimum labelling spanning tree (MLST) problem. An exact method has been proposed (EXACT), along with the best heuristics recommended in the literature: the Modified Genetic Algorithm (MGA) by Xiong et al. [12] and the Pilot Method (PILOT) by Cerulli et al. [9]. A new hybrid metaheuristic for the MLST problem has been proposed. It has been obtained by combining Variable Neighbourhood Search (VNS) and Simulated Annealing (SA) metaheuristics. We call this hybrid approach as Group-Swap Variable Neighbourhood Search (GS-VNS).

Computational experiments were performed using different instances of the MLST problem to evaluate how the algorithms are influenced by the parameters and the structure of the network. Applying the nonparametric statistical tests of Friedman [26] and Nemenyi [27], we concluded that the proposed GS-VNS has significantly better performance than the other algorithms presented with respect to solution quality and computational running time. GS-VNS obtains a large number of optimal or near-optimal solutions, showing an extremely high diversification capability.

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

[1] A. S. Tanenbaum, Computer Networks, Prentice-Hall, Englewood Cliffs, New Jersey, 1989.

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

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

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

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

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

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

[8] Y. Xiong, B. Golden, E. Wasil, Worst case behavior of the mvca heuristic for the minimum labelling spanning tree problem, Operations Research Letters 33 (1) (2005) 77–80.

[9] R. Cerulli, A. Fink, M. Gentili, S. Voß, Metaheuristics comparison for the minimum labelling spanning tree problem, in: B. L. Golden, S. Raghavan, E. A. Wasil (Eds.), The Next Wave on Computing, Optimization, and Decision Technologies, Springer-Verlag, New York, 2005, pp. 93–106.

[10] C. Duin, S. Voß, The Pilot Method: A strategy for heuristic repetition with applications to the Steiner problem in graphs, Wiley InterScience 34 (3) (1999) 181–191.

[11] S. Voß, A. Fink, C. Duin, Looking ahead with the Pilot Method, Annals of Operations Research 136 (2004) 285–302.

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

[13] S. Voß, S. Martello, I. H. Osman, C. Roucairol, Meta-Heuristics. Advanced and Trends Local Search Paradigms for Optimization, Kluwer Academic Publishers, Norwell, MA, 1999.

[14] C. Blum, A. Roli, Metaheuristics in combinatorial optimization: Overview and conceptual comparison, ACM Computing Surveys 35 (3) (2003) 268–308.

[15] P. Hansen, N. Mladenović, Variable neighbourhood search, Computers and Operations Research 24 (1997) 1097–1100.

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

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

[18] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, Optimization by simulated annealing, Science 220 (4598) (1983) 671–680.

[19] V. Cerny, Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm, Journal of Optimization Theory and Applications 45 (1985) 41–51.
[20] E. Aarts, J. Korst, W. Michiels, Simulated annealing, in: E. K. Burke, G. Kendall (Eds.), Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, Springer Verlag, 2005, pp. 187–210.
[21] D. E. Goldberg, K. Deb, B. Korb, Don't worry, be messy, in: Proc. 4th Intern. Conf. on Genetic Algorithms, Morgan-Kaufmann, La Jolla, CA, 1991, pp. 24–30.
[22] F. Glover, G. A. Kochenberger, Handbook of metaheuristics, Kluwer Academic Publishers, Norwell, MA, 2003.
[23] M. P. Pérez, F. A. Rodríguez, J. M. Moreno-Vega, A hybrid VNS-path relinking for the p-hub median problem, IMA Journal of Management Mathematics 18 (2) (2007) 157–171.
[24] J. Pacheco, S. Casado, L. Nuñez, Use of VNS and TS in classification: variable selection and determination of the linear discrimination function coefficients, IMA Journal of Management Mathematics 18 (2) (2007) 191–206.
[25] S. Consoli, Test datasets for the minimum labelling spanning tree problem, [online], http://people.brunel.ac.uk/˜mapgssc/MLSTP.htm (March 2007).
[26] M. Friedman, A comparison of alternative tests of significance for the problem of m rankings, Annals of Mathematical Statistics 11 (1940) 86–92.
[27] P. B. Nemenyi, Distribution-free multiple comparisons, Ph.D. thesis, Princeton University, New Jersey (1963).
[28] J. Demśar, Statistical comparison of classifiers over multiple data sets, Journal of Machine Learning Research 7 (2006) 1–30.
[29] M. Hollander, D. A. Wolfe, Nonparametric statistical methods, 2nd Edition, John Wiley & Sons, New York, 1999.

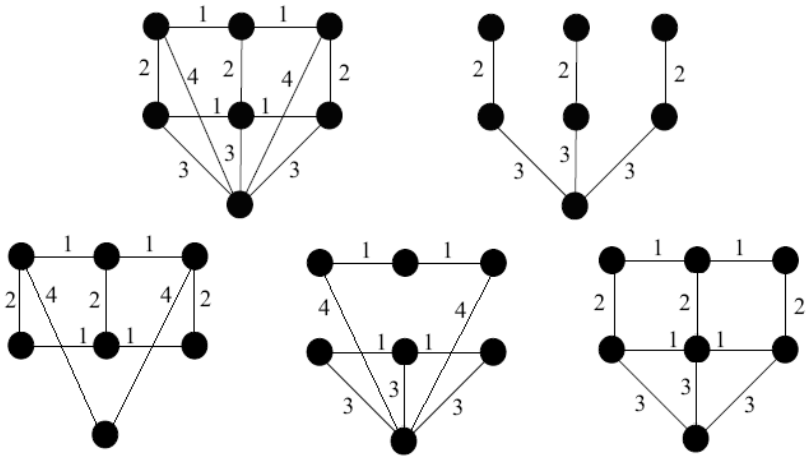

Figure 1. The top two graphs show a sample graph and its optimal solution. The bottom three graphs show some feasible solutions.

1th ITERATION				
LABEL	1	2	3	4
NUMBER OF COMPONENTS	3	4	4	5
2th ITERATION				
LABEL	1	2	3	4
NUMBER OF COMPONENTS	-	2	2	2
3th ITERATION				
LABEL	1	2	3	4
NUMBER OF COMPONENTS	-	1	-	1

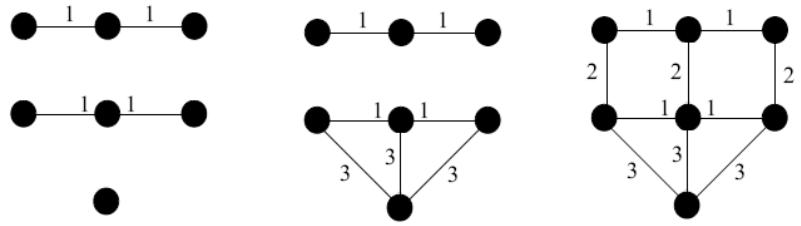

Figure 2. Example illustrating the steps of the Maximum Vertex Covering Algorithm by Krumke and Wirth [5].

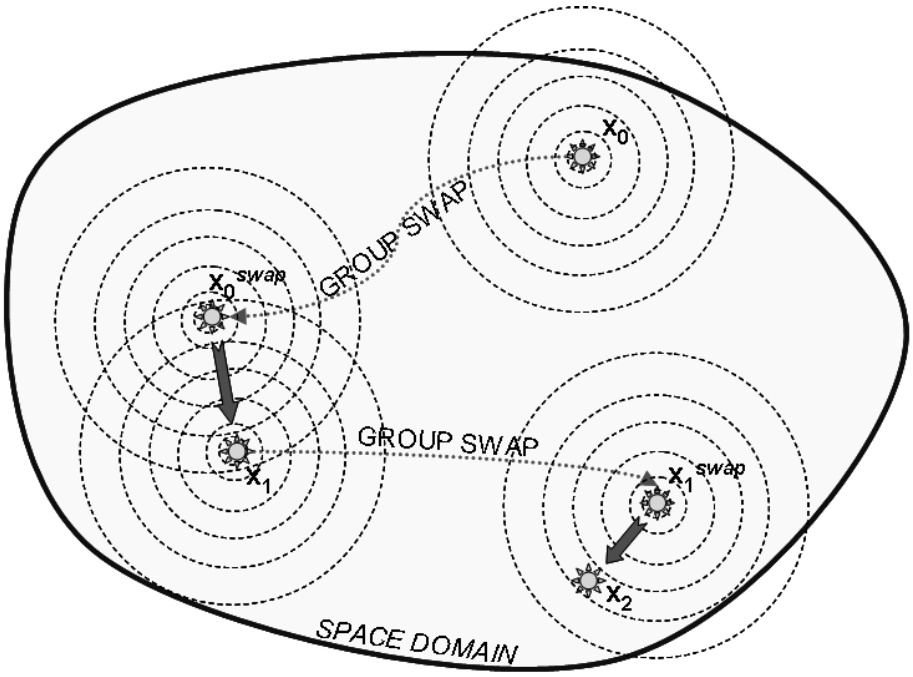

Figure 3. Example illustrating the steps of Group-Swap Variable Neighbourhood Search.

Table 1
Computational results for $n = \ell = 20, 30, 40, 50$ (*max-CPU-time* for heuristics = 1000 ms)

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

Parameters			**Computational times (milliseconds)**			
n	ℓ	d	EXACT	PILOT	MGA	GS-VNS
		0.8	0	0	15.6	0
20	20	0.5	0	1.6	22	0
		0.2	11	3.1	23.4	0
		0.8	0	3	9.4	1.5
30	30	0.5	0	3.1	26.5	0
		0.2	138	4.7	45.4	3.1
		0.8	2	6.3	12.5	1.5
40	40	0.5	3.2	7.9	28.2	3.1
		0.2	$100.2*10^3$	10.8	120.3	6.2
		0.8	3.1	17.1	21.8	3.1
50	50	0.5	21.9	20.2	531.3	6.2
		0.2	$66.3*10^3$	17.2	93.6	8
	TOTAL:		$166.7*10^3$	95	950	32.7

Table 2
Computational results for $n = 100$, $\ell = 0.25n$, $0.5n$, n, $1.25n$ (*max-CPU-time* for heuristics = $20*10^3$ ms)

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

Parameters			**Computational times (milliseconds)**			
n	ℓ	d	EXACT	PILOT	MGA	GS-VNS
		0.8	9.4	4.7	26.5	0
	25	0.5	14	12.6	29.7	4.5
		0.2	34.3	23.2	45.3	4.8
		0.8	17.8	67.3	23.5	12.6
	50	0.5	23.5	90.7	106.2	21.7
100		0.2	$10.2*10^3$	103.2	148.3	26.5
		0.8	142.8	378.1	254.7	146.9
	100	0.5	$2.4*10^3$	376.2	300	75.9
		0.2	NF	399.9	$9.4*10^3$	514
		0.8	496.9	565.7	68.7	20.2
	125	0.5	$179.6*10^3$	576.3	759.4	345.4
		0.2	NF	634.5	$2*10^3$	$1.2*10^3$
TOTAL:			-	$3.2*10^3$	$13.2*10^3$	$2.4*10^3$

Table 3
Computational results for $n = 200$, $\ell = 0.25n$, $0.5n$, n, $1.25n$ (*max-CPU-time* for heuristics = $60*10^3$ ms)

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

Parameters			**Computational times (milliseconds)**			
n	ℓ	d	EXACT	PILOT	MGA	GS-VNS
		0.8	29.7	90.7	26.5	0
	50	0.5	32.7	164.1	68.8	34.4
		0.2	$5.4*10^3$	320.4	326.6	232.8
		0.8	138.6	876.5	139.3	140.8
	100	0.5	807.8	$1.2*10^3$	$1.6*10^3$	159.4
200		0.2	NF	$1.3*10^3$	$2.2*10^3$	$2.9*10^3$
		0.8	$22.5*10^3$	$5.9*10^3$	204.6	79.7
	200	0.5	NF	$5.6*10^3$	$16.1*10^3$	876.1
		0.2	NF	$5*10^3$	$12.7*10^3$	$33.7*10^3$
		0.8	$20.6*10^3$	$9.1*10^3$	$2.2*10^3$	$1.5*10^3$
	250	0.5	NF	$8.4*10^3$	$17.6*10^3$	$2.3*10^3$
		0.2	NF	$8*10^3$	$26.4*10^3$	$1.5*10^3$
	TOTAL:		-	$45.9*10^3$	$79.6*10^3$	$43.4*10^3$

Table 4
Computational results for $n = 500$, $\ell = 0.25n$, $0.5n$, n, $1.25n$ (*max-CPU-time* for heuristics = $300*10^3$ ms)

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

Parameters			**Computational times (milliseconds)**			
n	ℓ	d	EXACT	PILOT	MGA	GS-VNS
		0.8	370	$3.4*10^3$	18	45
	125	0.5	597	$6.6*10^3$	$2.6*10^3$	560
		0.2	NF	$11.9*10^3$	$57.1*10^3$	$3.7*10^3$
		0.8	$5.3*10^3$	$35.4*10^3$	516	490
	250	0.5	NF	$65.3*10^3$	$28*10^3$	$26.9*10^3$
500		0.2	NF	$156.4*10^3$	$181.2*10^3$	$10.2*10^3$
		0.8	NF	$200.5*10^3$	$117.5*10^3$	$8.6*10^3$
	500	0.5	NF	$190.1*10^3$	$170.9*10^3$	$110.2*10^3$
		0.2	NF	$300.6*10^3$	$241.8*10^3$	$50.3*10^3$
		0.8	NF	$184.3*10^3$	$51.9*10^3$	970
	625	0.5	NF	$200.9*10^3$	$222.2*10^3$	$33.9*10^3$
		0.2	NF	$289.9*10^3$	$297.8*10^3$	$60*10^3$
TOTAL:			-	$1645.3*10^3$	$1371.5*10^3$	$395.9*10^3$

Table 5
Average ranks of the algorithms among all the considered datasets

ALGORITHM	GS-VNS	PILOT	MGA	EXACT
average rank	1.24	2.80	2.92	3.04

Table 6
Pairwise differences of the average ranks of the algorithms (Critical difference = 0.82 for a significance level of 1% for the Nemenyi test)

ALGORITHM (rank)	GS-VNS (1.24)	PILOT (2.80)	MGA (2.92)	EXACT (3.04)
GS-VNS (1.24)	-	**1.56**	**1.68**	**1.8**
PILOT (2.80)	-	-	0.12	0.24
MGA (2.92)	-	-	-	0.12
EXACT (3.04)	-	-	-	-