Heuristics based on 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], 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

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-complete. A Greedy Randomized Adaptive Search Procedure (GRASP) and different versions of Variable Neighbourhood Search (VNS) are proposed. 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.

Keywords: Metaheuristics, Combinatorial optimization, Minimum labelling spanning tree, Variable Neighbourhood Search, Greedy Randomized Adaptive Search Procedure.

* Corresponding author.
Tel: +44 (0)1895 266820; fax: +44 (0)1895 269732
E-mail address: **sergio.consoli@brunel.ac.uk**

1. Introduction

Many combinatorial optimization problems can be formulated on a graph where the possible solutions are spanning trees. These problems consist of finding spanning trees 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 optimization problems belong to this class of problems and consequently there is a large and growing interest in both theoretical and practical aspects. For some of these problems there are polynomial-time algorithms, while most 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 coloured edges, one seeks a spanning tree with the least number of colours. Such a model can represent many real-world problems in telecommunications networks, power networks, and multimodal transportation networks, among others. For example, in telecommunications networks, there are many different types of communications media, such as optical fibre, coaxial cable, microwave, and telephone line (Tanenbaum, 1989). 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 nodes, E is the set of edges, and L is the set of labels. In the telecommunications example (Tanenbaum, 1989), the vertices represent communications nodes, the edges communications links, and the labels communications types. Each edge in E is labelled or coloured by a label or colour 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 or colour.

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 colour, denoting a different company managing that edge. The aim is to find a spanning tree of the graph using the minimum number of colours. 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 nodes, E is the set of edges, and L is the set of 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 colours $C \subseteq L$, 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 organized 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*, a *Basic Variable Neighbourhood Search*, and a different version that we have called *Group-Swap 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 optimization, the reader is referred to (Consoli and Darby-Dowman, 2006), (Blum and Roli, 2003), (Glover and Kochenberger, 2003), and (Voss et al., 1999).

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: $H = (V,\varnothing)$. It then successively 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. This begins with an empty graph. Successively it adds labels by reducing the number of connected components by as much as possible until only one connected component is left, i.e. when only a connected subgraph is obtained. The details of this version of MVCA are described in the following Algorithm 1.

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

1. *While* H has more than one connected component *do*
   1.1. Select the unused colour $c \in (L–C)$ that minimizes $Comp(C \cup \{c\})$.
   1.2. Add label c to the set of used colours C: $C \leftarrow C \cup \{c\}$.
   1.3. *Update* H: add all the edges of label c to the subgraph H: $E_H \leftarrow E_H \cup E(\{c\})$.
2. *end while*.
3. Take an arbitrary spanning tree T of H.

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

Figure 2 shows how this version of MVCA 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 colours (2, 3 and 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. The old version of Chang and Leu (1997) would stop here, resulting in an error. However, the MVCA version of Krumke and Wirth (1998) finally adds label 2 to get only one connected component (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.

- INSERT FIGURE 2 -

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 that is:

$$H_b = \sum_{i=1}^{b} 1/i = 1 + 1/2 + 1/3 + \ldots + 1/b;$$

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).

The usual rule of Krumke and Wirth (1998) to select the colour that minimizes the total number of connected components at each step, results in fast and high-quality solutions. The problem with this classic approach occurs when more than one colour with same resulting minimum number of connected components is detected, in a specific step. Since, frequently, there are many colours reaching this minimum value, the results mainly depend on the rule chosen to select a candidate from this set of ties. If the initial colour encountered 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. (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* (developed by Duin and Voss (1999) and subsequently extended in (Voss et al., 2004)) to the MLST problem. Their Pilot Method is a greedy heuristic with a limited look-ahead capability. This modified version of MVCA tries each label at the initial step and then applies MVCA in a greedy fashion from then on. Then it selects the best of the resulting solutions. Moreover, Cerulli et al. (2005) compared this method with other metaheuristics (Reactive Tabú Search, Simulated Annealing, and an ad-hoc implementation of Variable Neighbourhood Search) on different test problems. 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 colours is high).

Xiong et al. (2006) implemented the Pilot Method of Cerulli et al. (2005), along with faster simplified versions (see also Xiong (2005)). They are modified versions of MVCA focusing on the initial label added, as is the case with the Pilot Method. For example, after the colours have been sorted according to their

frequencies, from highest to lowest, the initial 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. (2005), this version can potentially reduce running time by about 90%. However, since a higher frequency label may not always be the best place to start, it may not perform as well as the Pilot Method. Another modified version of Xiong et al. (2006) 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. (2006) proposed another way to modify MVCA. They consider at each step the three most promising colours, 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. (2006) presented a *Modified Genetic Algorithm* (*MGA*) that has 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) and Xiong (2005). It outperformed the first two Genetic Algorithms with respect to solution quality and running time. MGA is the first metaheuristic considered here, using the C++ implementation specified in Xiong (2005).

Cerulli et al. (2005) applied the *Pilot Method* to the MLST problem. Comparing it with some other metaheuristic implementations (Reactive Tabú 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 we analyse in this paper.

We then report 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 randomization. For a survey on GRASP, the reader is referred to (Pitsoulis and Resende, 2002) and (Resende and Ribeiro, 2002).

The remaining algorithms are two versions of *VNS: Variable Neighbourhood Search*. These are a *Basic VNS*, and an advanced version that we call *Group-Swap VNS*. Variable Neighbourhood Search is a recently exploited metaheuristic based on dynamically changing neighbourhood structures during the search

process. For more details see (Hansen and Mladenović, 1997, 2001, 2003).

3.1 Modified Genetic Algorithm (Xiong et al., 2006) (Xiong, 2005):

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

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 colour 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. The details of the Modified Genetic Algorithm for the MLST problem are specified in the following Algorithm 2.

Modified Genetic Algorithm:
Input: A labelled graph $G = (V,E,L)$, with n vertices, m edges, and ℓ labels.
Output: A spanning tree T.
Initialization:
- Let $C \leftarrow \varnothing$ be the set of colours.
- Let $H=(V,E(C))$ be the subgraph of G restricted to V and the edges with labels in C, where $E(C) = \{e \in E: L(e) \in C\}$.
- Fix P_n, the population size.

1. (s[0], s[1], …s[P_n-1]) ← *Initialize-Population* (G, P_n).
2. *for each i from 1 to* $P_n/2$ *do*
    2.1. *for each j from 1 to* P_n *do*
        2.1.1. t[1] ← j.
        2.1.2. t[2] ← mod((j+i), Pn).
        2.1.3. t-crossovered ← *Crossover* (t[1], t[2]).
        2.1.4. t-mutated ← *Mutation* (t-crossovered).
        2.1.5. *if* (t-mutated < t[1]) *then*
            2.1.5.1. t[1] ← t-mutated.
        2.1.6. *end if.*
    2.2. *end for*.

3. *end for*.
4. C = *Extract-the-Best* (s[0], s[1], …s[P_n-1]).
5. *Update* $H=(V,E(C))$, where $E(C) = \{e \in E: L(e) \in C\}$.
6. Take an arbitrary spanning tree T of H.

Algorithm 2. The Modified Genetic Algorithm for the MLST problem (Xiong, 2005), (Xiong et al., 2006).

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

The Pilot Method is a metaheuristic proposed by (Duin and Voss, 1999) and (Voss et al., 2004). 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 with respect to the master solution are evaluated, and the best solution to date becomes the new master solution. The algorithm continues until the user termination conditions are reached.

For the MLST problem, the Pilot Method proposed by Cerulli et al. (2005) focuses on the initial label to add. It uses the revised MVCA of Krumke and Wirth (1998) as the application process, and the null solution (an empty set of colours) as the master solution. 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. The details are specified in Algorithm 3.

It tries each label at the initial step and then applies MVCA in a greedy fashion from then on, i.e. by adding at each step the colour that minimizes the total number of connected components and stopping when the resulting graph is connected. Then it selects the best of the $|L| = \ell$ resulting solutions as the output of the method. At the end of the search, a local search mechanism is included in order to greedily drop colours while retaining feasibility, if possible (*Apply-Local-Search(C)* function).

Since up to ℓ local choices are evaluated from the master solution, the overall computational time is $O(\ell)$ times the MVCA computational time, leading to an overall complexity $O(\ell^2)$.

Pilot Method:
Input: A labelled graph $G = (V,E,L)$, with n vertices, m edges, and ℓ labels.
Output: A spanning tree T.
Initialization:
- Let $C \leftarrow \varnothing$ be the initial set of used colours for each iteration.
- Let $H=(V,E(C))$ be the subgraph of G restricted to V and the edges with labels in C, where $E(C) = \{e \in E: L(e) \in C\}$.
- Let *Comp*(C) be the number of connected components of $H=(V,E(C))$.
- Let $C' \leftarrow L$ be the global set of used colours.
- Let $H'=(V,E(C'))$ be the subgraph of G restricted to V and edges with labels in C', where $E(C') = \{e \in E: L(e) \in C'\}$.

1. *for each* $i \in L$ *do*
   1.1. Set $C \leftarrow \{i\}$.
   1.2. *Update* $H=(V,E(C))$, where $E(C) = \{e \in E: L(e) \in C\}$.
   1.3. *Update Comp*(C).
   1.4. *While* H has more than one connected component *do*
      1.4.1. Select the unused colour $c \in (L–C)$ that minimizes *Comp*($C \cup \{c\}$).
      1.4.2. Add label c to the set of used colours C: $C \leftarrow C \cup \{c\}$.

1.4.3. *Update H*: add all the edges of label c: $E_H \leftarrow E_H \cup E(\{c\})$.
1.5. *end while*.
1.6. *Apply- Local-Search(C)*.
1.7. *if* $|C| < |C'|$ *then*
1.7.1. $C' \leftarrow C$;
1.7.2. $H' \leftarrow H$.
1.8. *end if*.
2. *end for*.
3. Take an arbitrary spanning tree T of $H' = (V,E(C'))$.

Algorithm 3. The Pilot Method for the MLST problem (Cerulli et al., 2005).

3.3 Greedy Randomised Adaptive Search Procedure:

The difficulty with the classical version of MVCA is when it finds more than one colour with the same number of connected components. A question arises on the colour to be chosen. To find the best MVCA solution, we should alternatively add each of these colours, continuing the same strategy in the 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 colours.

The Pilot Method is able to achieve a greater diversification of the search process than the MVCA approach. Instead, by performing multiple repetitions of the MVCA, it is possible to increase the intensification of the basic MVCA. So, in this paper, we propose a Greedy Randomized Adaptive Search Procedure (GRASP) to the MLST problem, trying to unify the diversification improvement given by the Pilot strategy with the intensification improvement from a multi-start MVCA.

GRASP is a recently exploited method combining the power of greedy heuristics, randomization, and local search. It is a multi-start two-phase metaheuristic for combinatorial optimization proposed by Pitsoulis and Resende (2002). It mainly consists 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, Tabú 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 (2002).

Our GRASP implementation for the MLST problem is a multi-start MVCA, and the details are specified in Algorithm 4.

In the initial iterations, the colour minimizing the total number of connected components is added as initial colour (our experiments indicate that two iterations at this stage produce the best results). Subsequently, initial colour at each iteration is selected at random, taking inspiration from the Pilot Method. This represents the construction phase of GRASP (*Construct-Greedy-Randomized-Solution* function). Experimentally, it gives a good trade off between the diversification characteristic of the Pilot Method with the intensification characteristic of a multi-start MVCA.

The local search phase of GRASP for the MLST problem simply consists of trying to drop some labels at the end of the search (*Apply-Local-Search()* function). Local search gives a further improvement to the intensification phase of the algorithm.

The greedy criterion of the construction phase is based on the number of connected components produced by the colours, and a *value-based* restricted candidate list is used (Resende and Ribeiro, 2002). This involves placing in the list only the candidate colours having a greedy value (number of connected components) not greater than a user-defined threshold. In particular, in order to fill the restricted candidate list we fix the threshold as the minimum number of connected components produced by the candidate colours. This means that only the colours producing the least number of connected components constitute the restricted candidate list. Furthermore, after the two first iterations, complete randomization is used to choose the initial colour to add. This corresponds to setting the threshold to $+\infty$, and all the colours of the graph are present in the restricted candidate list (length α = total number of colours, ℓ). To intensify the search for the remaining colours to add, the list is filled considering only the colours leading to the minimum total number of connected components, as in the previous iterations.

Greedy Randomized Adaptive Search Procedure:
Input: A labelled graph $G = (V,E,L)$, with n vertices, m edges, and ℓ labels.
Output: A spanning tree T.
Initialization:
- Let $C \leftarrow \varnothing$ be the set of colours used at each iteration.
- Let $H=(V,E(C))$ be the subgraph of G restricted to V and the edges with labels in C, where $E(C) = \{e \in E: L(e) \in C\}$.
- Let $Comp(C)$ be the number of connected components of $H=(V,E(C))$.
- Let $C' \leftarrow L$ be the global set of used colours.
- Let $H'=(V,E(C'))$ be the subgraph of G restricted to V and edges with labels in C', where $E(C') = \{e \in E: L(e) \in C'\}$.

1. *While* termination condition not met *do*
2. Set $C \leftarrow \varnothing$ and update $H=(V,E(C))$.
3. C = *Construct-Greedy-Randomized-Solution( )*.
4. *Apply-Local-Search(C)*.
5. *if* $|C| < |C'|$ *then*
   5.1. $C' \leftarrow C$.
   5.2. $H' \leftarrow H$.
6. *end if.*

7. *end while.*
8. Take an arbitrary spanning tree T of $H' = (V,E(C'))$.

Function *Construct-Greedy-Randomized-Solution*:
Output: A set C of used colours.
Initialization:
- Let $C \leftarrow \emptyset$.
- Let $H=(V,E(C))$ be the subgraph restricted to V and edges with labels in C, where $E(C) = \{e \in E: L(e) \in C\}$.
- Let $RCL_\alpha \leftarrow \emptyset$ be the restricted candidate list of α length.

1. *if* Number-of-Iterations > 2 *then*
   1.1. Set $RCL_\alpha = L$ and $\alpha = \ell$.
   1.2. Select a colour $c \in RCL_\alpha$.
   1.3. Add label c to the set of used colours $C \leftarrow C \cup \{c\}$.
   1.4. *Update H*: add all the edges of label c: $E_H \leftarrow E_H \cup E(\{c\})$.
2. *end if.*
3. *Update Comp*(C).
4. *While H* has more than one connected component *do*
   4.1. Set $RCL_\alpha = \{\forall c \in L$ / minimizes $Comp(C \cup \{c\})\}$.
   4.2. Select a colour $c \in RCL_\alpha$.
   4.3. Add label c to the set of used colours C: $C \leftarrow C \cup \{c\}$.
   4.4. *Update H*: add all the edges of label c: $E_H \leftarrow E_H \cup E(\{c\})$.
5. *end while*.

Algorithm 4. GRASP for the MLST problem.

3.4 Basic Variable Neighbourhood Search:

Variable Neighbourhood Search is a new and widely applicable metaheuristic based on dynamically changing neighbourhood structures during the search process (Hansen and Mladenović, 1997, 2001, 2003).

VNS provides a general framework and many variants exist for specific requirements. 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.

At the starting point, it is required to define arbitrarily the neighbourhood structure. The simplest and most common choice is a structure in which the neighbourhoods have increasing cardinality: $|N_1(x)| < |N_2(x)| < \ldots < |N_{max}(x)|$. 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. The Basic VNS algorithm, applied to solve the MLST, is described in Algorithm 5.

The basic idea of the VNS is to change the neighbourhood structure when the local search is trapped on a local minimum. This is implemented by the shaking phase which 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

local search phase. The random point C' is generated in order to avoid cycling, which might occur if a deterministic rule is used.

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

$$c_i = \begin{cases} 1 & \textit{if colours } i \textit{ is in solution } C, \\ 0 & \textit{otherwise}, \end{cases}$$

$\forall i \in [1, |L|]$.

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:

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

where $\lambda_i = 1$ if colour i is included in one of the solution but not in the other, and 0 otherwise, $\forall\ i = 1, \ldots, \ell$. Then, given a solution C, we consider its kth neighbourhood, $N_k(C)$, as all the different sets having an Hamming distance from C equal to k colours, where $k = 1, 2, \ldots, k_{max}$, and k_{max} represents the size of the shaking (the value $k_{max} = (|C| + |C|/3)$ is the best choice according to our experiments). In a more formal way, the kth neighbourhood of a solution C is defined as $N_k(C) = \{S \subset L : \rho(C, S) = k\}$, where $k = 1, 2, \ldots, k_{max}$.

In order to construct the neighbourhood of a solution C, the algorithm firstly proceeds with the deletion of colours from C. After all the colours are removed, additional colours are included at random in C, from the set of unused colours ($L - C$), if a further expansion of the neighbourhood structure is required (case $k > |C|$).

The algorithm starts from an initial feasible solution generated at random, sets parameter k_{max} to a value greater than the number of colours of the current solution ($k_{max} > |C|$), and lets k vary during the execution. Since deletion of colours often gives an infeasible incomplete solution, additional colours may be added at this stage in order to restore feasibility. In this case, addition of colours follows the MVCA criterion of adding the colour with the minimum number of connected components.

The successive local search is not restricted to $N_k(C)$ but considers all the domain space. It consists of a post-optimization phase that tries to delete colours one by one from the specific solution, whilst maintaining feasibility. Afterwards, if no improvements are obtained ($|C'| > |C|$) in the move phase, the neighbourhood structure is changed ($k = k + 1$) giving a progressive diversification ($|N_1(x)| < |N_2(x)| < \ldots < |N_{max}(x)|$).

The algorithm proceeds until the stopping conditions such as the maximum allowed CPU time, the maximum number of iterations, or the maximum number of iterations between two successive improvements, are reached.

Basic Variable Neighbourhood Search:
Input: A labelled graph $G = (V,E,L)$, with n vertices, m edges, and ℓ labels.
Output: A spanning tree T.
Initialization:
- Let $C \leftarrow \varnothing$ be the global set of used colours.
- Let $H=(V,E(C))$ be the subgraph of G restricted to V and the edges with labels in C, where $E(C) = \{e \in E: L(e) \in C\}$.
- Let C' be a set of colours.

1. C = *Generate-Initial-Solution-At-Random ( )*.
2. *While* termination condition not met *do*
   2.1. Fix $k = 1$ and $k_{max} > |C|$.
   2.2. *While* ($k < k_{max}$) *do*
      2.2.1. C' = *Shaking-Phase*($N_k(C)$).
      2.2.2. *Apply-Local-Search*(C').
      2.2.3. *if* $|C'| < |C|$ *then*
         2.2.3.1. Move $C \leftarrow C'$.
         2.2.3.2. Fix $k = 1$and $k_{max} > |C|$.
      2.2.4. *else*
         2.2.4.1. $k = k + 1$.
      2.2.5. *end if.*
   2.3. *end while.*
3. *end while.*
4. *Update* H: add all the edges of label $c \in C$: $E_H \leftarrow E_H \cup E(\{c\})$.
5. Take an arbitrary spanning tree T of $H = (V,E(C))$.

Function *Shaking-Phase*($N_K(C)$):
Output: A set C' of used colours.
Initialization:
- Let $C' \leftarrow C$.
- Let $H'=(V,E(C'))$ be the subgraph of G restricted to V and the edges with labels in C', where $E(C') = \{e \in E: L(e) \in C'\}$.
- Let *Comp*(C') be the number of connected components of $H'=(V,E(C'))$.

1. *for i=1 to k do*
   1.1. *if* $i \leq |C|$ *then*
      1.1.1. Select a colour $c' \in C'$.
      1.1.2. Delete label c' from the set of used colours C': $C' \leftarrow C' - \{c'\}$.
   1.2. else
      1.2.1. Select an unused colour c', i.e. $c' \in (L - C)$.
      1.2.2. Add label c' to the set of used colours C': $C' \leftarrow C' + \{c'\}$.
   1.3. *end if*.
   1.4. *Update* $H'(V,E(C'))$: where $E(C') = \{e \in E: L(e) \in C'\}$.
2. *end for*.
3. *Update Comp*(C').
4. *While* H' has more than one connected component *do*
   4.1. Select the unused colour $u \in (L–C')$ that minimizes *Comp*($C' \cup \{u\}$).
   4.2. Add label u to the set of used colours C': $C' \leftarrow C' \cup \{u\}$.
   4.3. *Update* H': add all the edges of label u: $E_{H'} \leftarrow E_{H'} \cup E(\{u\})$.
5. *end while*.

Algorithm 5. The Basic VNS for the MLST problem.

3.5 Group-Swap Variable Neighbourhood Search:

Experimentally, VNS performance can be improved in some circumstances. Numerous variants have been proposed in the literature to achieve this goal (Hansen and Mladenović, 1997, 2001, 2003). For the MLST problem, we report on some new features to the Basic VNS.

The first variant is to introduce a *Group-Swap* operation. The Group-Swap consists of extracting 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 colours 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 VNS.

For our implementation, given a solution C, we consider its kth neighbourhood, $N_k(C)$, as all the different sets obtained from C by removing k colours, where $k = 1, 2, \ldots, k_{max}$. In a more formal way, given a solution C, its kth neighbourhood is defined as $N_k(C) = \{S \subset L : (|C| - |S|) = k\}$, where $k = 1, 2, \ldots, k_{max}$. In particular, we start from an initial feasible solution generated at random and let parameter k_{max} vary during the execution. At each shaking, k_{max} is set to the number of colours of the current feasible solution whose neighbourhood is being explored ($k_{max} = |C|$). Since deletion of colours often gives an infeasible incomplete solution, additional colours may be added in order to restore feasibility. Addition of colours at this step follows the MVCA criterion of adding the colour with the minimum number of connected components.

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}. Now, 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 version 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 is inspired by the Simulated Annealing (SA) metaheuristic (Kirkpatrick et al., 1983). However, the Probabilistic MVCA does not work with complete solutions but with partial solutions created with components added at each step.

The resulting algorithm combines VNS and the Probabilistic MVCA, representing 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 a *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* (or *cooling law*). 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 that give 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 towards zero. In this case 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:

$$\exp\left(-\left(Comp(C \cup k) - Comp(C \cup s)\right)/T\right),$$

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 $T_{|C|} = 1/(|C|+1)$, 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, T=1/3=0.33 in the third step, and so on. Therefore, in the initial step the Boltzmann values for each colour are: $a \Rightarrow 0.0024$, $b \Rightarrow 0.135$, $c \Rightarrow 0.018$, $d \Rightarrow 1$. 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 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:

$$\exp(-\Delta / T) = \exp(-(Comp(C \cup k) - 2)/0.5),$$

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 $T_{|C|} = 1/(|C|+1)$ for the temperature is not satisfactory. After having tested different cooling laws, the best performance has been obtained by using a geometric cooling schedule, $T_{k+1} = T_k / \alpha = T_0 / \alpha^k$. 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 version using the Probabilistic MVCA as constructive heuristic has been tested. However, the best results have been obtained by combining together the Group-Swap operation and the Probabilistic MVCA constructive heuristic inside the VNS. We call this hybrid metaheuristic Group-Swap VNS. The Probabilistic MVCA is applied both to the Group-Swap operation, in order to obtain a solution from the complementary space of the current solution, and in the Shaking Phase, when we need to reach feasibility by adding colours to incomplete solutions.

For the geometric schedule in the Group-Swap, experimentally we have found $T_0 = |Best\_C|$ and $\alpha = |Best\_C|$ to be values that performed well, where $Best\_C$ is the current best solution. So, the resulting

cooling law for the Group-Swap procedure is

$$T\_GroupSwap_{(|C|+1)} = \frac{T\_GroupSwap_{(0)}}{\alpha^{|C|}}.$$

In the same way we have found $T_0 = |Best\_C|^2$ and $\alpha = |Best\_C|$ as good values for the geometric cooling law in the Shaking Phase. The corresponding geometric cooling law is

$$T\_shaking_{(|C'|+1)} = \frac{T\_shaking_{(0)}}{\alpha^{|C'|}} \quad .$$

The details of the overall algorithm implemented are described below (Algorithm 6). We start from an initial feasible solution generated at random. Then the Group-Swap operation is applied, and the shaking phase is applied to the resulting solution, denoted by *C*. The shaking phase consists of the random selection of a point *C'* in the neighbourhood $N_k(C)$ of the current solution *C*.

For our implementation, given a solution *C*, we consider its neighbourhood $N_k(C)$ as all the different sets of colours that are possible to get from *C* by removing *k* colours, where $k = 1, 2, \ldots, k_{max}$. 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 $k = 1, 2, \ldots, k_{max}$. At each shaking, 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, addition of colours at this step follows the Probabilistic MVCA as constructive heuristic.

The successive local search tries to delete colours one by one from the specific solution, whilst maintaining feasibility. Afterwards, if no improvements are obtained ($|C'| > |C|$) in the move phase, the neighbourhood structure is changed ($k = k + 1$) giving a progressive diversification ($|N_1(x)| < |N_2(x)| < \ldots < |N_{max}(x)|$). After the local search, the Group-Swap operation is applied to the actual best solution and the algorithm starts again, shaking the resulting solution and letting parameter k_{max} vary during the execution.

Group-Swap Variable Neighbourhood Search:
Input: A labelled graph $G = (V,E,L)$, with *n* vertices, *m* edges, and ℓ labels.
Output: A spanning tree *T*.
Initialization:
- Let $Best\_C \leftarrow \emptyset$, $C \leftarrow \emptyset$, $C' \leftarrow \emptyset$ be sets of colours.
- Let $H=(V,E(Best\_C))$ be the subgraph of *G* restricted to *V* and the edges with labels in *Best_C*, where $E(Best\_C) = \{e \in E: L(e) \in Best\_C\}$.

1. *Best_C = Generate-Initial-Solution-At-Random( )*.
2. *Apply-Local-Search(Best_C)*.
3. *While* termination condition not met *do*
   3.1. Perform the swapping of the best solution: *C = Group-Swap(Best_C)*.
   3.2. *While* ($|C| < |Best\_C|$) *do*
      3.2.1. Continue to swap: *C = Group-Swap(Best_C)*.
   3.3. *end while*.
   3.4. Set $C' \leftarrow C$.
   3.5. Fix $k = 1$ and $k_{max} = |C|$.
   3.6. *While* ($k < k_{max}$) *do*
      3.6.1. *C' = Shaking-Phase($N_k(C')$)*.

3.6.2. *Apply-Local-Search(C')*.
3.6.3. *if* $|C'| < |C|$ *then*
3.6.3.1. Move $C \leftarrow C'$.
3.6.3.2. Fix $k = 1$ and $k_{max} = |C|$.
3.6.4. *else*
3.6.4.1. $k = k + 1$.
3.6.5. *end if.*
3.7. *end while.*
3.8. *if* $|C| < |Best\_C|$ *then*
3.8.1. Move $Best\_C \leftarrow C$.
3.9. *end if.*
4. *end while.*
5. *Update H*: add all the edges of label $c \in Best\_C$: $E_H \leftarrow E_H \cup E(\{c\})$.
6. Take an arbitrary spanning tree T of $H = (V, E(Best\_C))$.

Function *Group-Swap(Best_C)*:
Output: the set C of colours.
Initialization:
- Set $C \leftarrow \varnothing$.
- Let $H=(V,E(C))$ be the subgraph of G restricted to V and the edges with labels in C, where $E(C) = \{e \in E: L(e) \in C\}$.
- Let $Comp(C)$ be the number of connected components of $H=(V,E(C))$.
- Let $C\_compl \leftarrow (L - Best\_C)$ the complementary space of C.
- Let $s \in C\_compl$ be the colour that minimizes $Comp(C \cup \{s\})$.

1. *While H* has more than one connected component *do*
1.1. *for each* $c \in (C\_compl)$ *do*
1.1.1. Geometric Group-Swap cooling schedule for the temperature:

$$T\_GroupSwap\left(|C|+1\right) = \frac{T\_GroupSwap(0)}{\alpha^{|C|}},$$

$$\text{where} \begin{cases} T\_GroupSwap(0) = |Best\_C|; \\ \alpha = |Best\_C|. \end{cases}$$

1.1.2. Calculate the probabilities $P(c)$ for each colour, normalizing the values given by the Boltzmann function:

$$\exp\left(-\frac{\left(Comp(C \cup c) - Comp(C \cup s)\right)}{T\_GroupSwap}\right).$$

1.2. *end for*.
1.3. *Roulette selection*: select an unused colour $u \in (C\_compl)$ following the probabilities values $P(c)$, $\forall c \in (C\_compl)$.
1.4. Add label u to the set of used colours C: $C \leftarrow C \cup \{u\}$.
1.5. *Update H*: add all the edges of label u: $E_H \leftarrow E_H \cup E(\{u\})$.
1.6. *end for*.
2. *end while*.

Function *Shaking-Phase(N_K(C'))*:
Initialization:
- Let $H'=(V,E(C'))$ be the subgraph of G restricted to V and the edges with labels in C', where $E(C') = \{e \in E: L(e) \in C'\}$.
- Let $Comp(C')$ be the number of connected components of $H'=(V,E(C'))$.

1. *for* $i=1$ *to k do*
1.1. Select a colour $c' \in C'$.
1.2. Delete label c' from the set of used colours C': $C' \leftarrow C' - \{c'\}$.
1.3. *Update H'*: delete all the edges of label c': $E_{H'} \leftarrow E_{H'} - E(\{c'\})$.
2. *end for*.
3. *Update* $Comp(C')$.
4. Let $s \in (L - C')$ be the colour that minimizes $Comp(C' \cup \{s\})$.
5. *While H* has more than one connected component *do*
5.1. *for each* $c \in (L - C')$ *do*
5.1.1. Geometric shaking cooling schedule for the temperature:

$$T\_shaking\left(|C'|+1\right)=\frac{T\_shaking(0)}{\alpha^{|C'|}},$$

$$\text{where}\begin{cases} T\_shaking(0)=|Best\_C|^2; \\ \alpha=|Best\_C|. \end{cases}$$

5.1.2. Calculate the probabilities $P(c)$ for each colour, normalizing the values given by the Boltzmann function:

$$\exp\left(-\frac{\left(Comp(C\cup c)-Comp(C\cup s)\right)}{T\_GroupSwap}\right).$$

5.2. *end for*.

5.3. *Roulette selection*: select an unused colour $u\in(L-C')$ following the probabilities values $P(c)$, $\forall c \in (L-C')$.

5.4. Add label u to the set of used colours C': $C' \leftarrow C' \cup \{u\}$.

5.5. *Update H'*: add all the edges of label u: $E_{H'} \leftarrow E_{H'} \cup E(\{u\})$.

6. *end while*.

Algorithm 6. The Group-Swap VNS for the MLST problem.

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), B-VNS (Basic Variable Neighbourhood Search), GS-VNS (Group-Swap 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:

m: total number of edges of the graph;

n: total number of nodes of the graph;

ℓ: total number of colours assigned to the edges of the graph.

We thank the authors of (Cerulli et al., 2005), who kindly provided data for use in our experiments.

In our experiments, 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, for each combination of the parameters n, ℓ, and d. A maximum allowed CPU time, that we call *max-CPU-time*, is chosen as the stopping condition for all the metaheuristics, determined experimentally with respect to the dimension of the problem instance. For MGA, we use a variable number of iterations for each instance, experimentally determined such that the computations take approximately *max-CPU-time* for the specific dataset. Such a stopping condition is not applied to the Pilot Method since, for each instance, it considers every label as the initial colour to add. *Max-CPU-time* is evaluated in order to allow the Pilot Method to finish. 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.

When possible, the results of the metaheuristics are compared with the exact solution, identified with the label EXACT.

4.1 Exact Method:

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 the following Algorithm 7.

Procedure *Backtracking*:
Initialization:
- Let $C \leftarrow \varnothing$ be the initially empty set of used colours.
- Let $C^* \leftarrow L$ be the global set of used colours.
- Let $H=(V,E(C))$ be the subgraph of G restricted to V and the edges with labels in C, where $E(C) = \{e \in E: L(e) \in C\}$.
- Let $H^*=(V,E(C^*))$ be the subgraph of G restricted to V and the edges with labels in C^*, where $E(C^*) = \{e \in E: L(e) \in C^*\}$.
- Let *Comp*(C) be the number of connected components of $H=(V,E(C))$.

1. *Test(C)*.
2. Take an arbitrary spanning tree T of $H^* = (V, E(C^*))$.

Procedure *Test(C)*:
1. *if* $|C| < |C^*|$ *then*
    1.1. *Update Comp*(C).
    1.2. *if Comp(C)* = 1 *then*
        1.2.1. $C^* \leftarrow C$.
    1.3. *elsif* $|C| < |C^*|-1$ *then*
        1.3.1. *for each* $c \in (L-C)$ *do*
            1.3.1.1. Try to add colour *c: Test*($C \cup \{c\}$).
        1.3.2. *end for.*
    1.4. *end if.*
2. *end if.*

Algorithm 7. The Exact Method for the MLST.

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 colours. 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 colours $|C'|=|C^*|-2$, we should try to add the colours 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 colour 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 reasonable and the method obtains the exact solution. The complexity of the instances increases with the dimension of the graph (number of nodes and colours), 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).

4.2 Experimental analysis:

In our experiments, we have considered two different groups of datasets, including instances with a number of vertices, n, and a number of colours, ℓ, from 20 up to 500. 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.

Group 1 examines small instances with the number of vertices equal to the number of colours. 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 Group 1 datasets, and the corresponding average computational times, with a *max-CPU-time* of 1 second.

- INSERT TABLE 1 -

Group 2 considers larger instances with a fixed number of vertices, and a number of colours $\ell=0.25\cdot n$, $0.5\cdot n$, n, and $1.25\cdot n$. Thus, the datasets of Group 2 are n=100, 200 and 500 vertices, $\ell=0.25\cdot n$, $0.5\cdot n$, n, and $1.25\cdot n$ 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 2-4 respectively. All the instances of the problem are available from the authors [7].

- INSERT TABLE 2, TABLE 3, TABLE 4 -

Looking at the tables, for a single dataset the performance of a metaheuristic is 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. The average number of colours of each metaheuristic among all 480 instances is

PILOT	MGA	GRASP	B-VNS	GS-VNS
5.67	5.68	5.61	5.59	5.59

The best performing algorithm with respect to the average values of the objective function are GS-VNS and B-VNS, giving the same value (5.59), followed respectively by GRASP, PILOT, and MGA.

The motivation of introduce a high diversification capability in GS-VNS is to obtain a better performance in large problem instances. Inspection of Table 4 shows that this aim is achieved.

4.3 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) and (Demšar, 2006) for more details). Since ANOVA is based on assumptions that are violated in this context, we make use of the *Friedman Test*, the non-parametric equivalent of ANOVA (Friedman, 1937, 1940), 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. For more details, see Appendix A.

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 then computed. The metaheuristics are ordered with respect to the average ranks from the best one to the worst one, as follows:

B-VNS	GS-VNS	GRASP	PILOT	MGA
1.9	2	2.26	4.4	4.45

According to the ranking, B-VNS is the best performing algorithm, immediately followed by GS-VNS, and GRASP, with PILOT and MGA achieving the worst results.

Now, we analyse the statistical significance of differences between these ranks. Consider the Iman and Davenport (1980)'s version of the Friedman test for k=5 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)) = (4, 188) degrees of freedom, is computed (Appendix A). This value is 100.4, which is greater than the critical value (3.42 for α=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 (Appendix A). Considering a significance level α=1%, the critical value is $q_{0.01} \cong 3.26$. The critical difference (CD) for the Nemenyi test is

$$CD = 3.26 \cdot \sqrt{\frac{5 \cdot 6}{6 \cdot 48}} \cong 1.05.$$

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 B-VNS, GS-VNS, and GRASP, while the second group includes PILOT and MGA. Considering a significance level α=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, B-VNS, GS-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 B-VNS, GS-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 the following (ranking from the best to the worst algorithm):

B-VNS	GS-VNS	GRASP	MGA	PILOT
100	100	99.7	99.7	96.7

B-VNS and GS-VNS obtain 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 11 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 B-VNS, GS-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 9 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 B-VNS, GS-VNS, and GRASP are effective metaheuristics for the MLST problem. Furthermore, the algorithm which appears to be the most suitable for the proposed problem is B-VNS, thanks to the following features: ease of implementation, user-friendly code, high quality of the solutions, and shorter computational running times.

5. Conclusions

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), a Basic Variable Neighbourhood Search (B-VNS), and a VNS-based version that we have called 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, the structure of the network and the distribution of the labels on the edges. Applying the nonparametric statistical tests of Friedman (1937, 1940), and Nemenyi (1963), we concluded that B-VNS, GS-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. B-VNS, GS-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 B-VNS, GS-VNS, and GRASP are fast and extremely effective metaheuristics for the MLST problem. In addition, B-VNS is particularly recommended for the proposed problem because its simplicity and its ability to obtain high-quality solutions in short computational running times.

Appendix A. Statistical tests

Friedman Test (Friedman, 1937, 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 (1937, 1940) is

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

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:

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

which is distributed according to the F-distribution with (k−1) and (k−1)(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)

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

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).

Acknowledges

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 A. 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] D. Avis, A. Hertz and O. Marcotte, *Graph Theory and Combinatorial Optimization*, New York: Springer-Verlag, 2005.

[2] T. Brüggemann, J. Monnot and G. J. Woeginger, "Local search for the minimum label spanning tree problem with bounded colour classes," *Operations Research Letters*, vol. 31, pp. 195–201, 2003.

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

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

[5] R. S. Chang and S. J. Leu, "The minimum labelling spanning trees," *Information Processing Letters*, vol. 63, no. 5, pp. 277–282, 1997.

[6] S. Consoli and K. Darby-Dowman, "Combinatorial optimization and metaheuristics," Brunel University, West London, UK, Tech. Rep. TR/01/06, Nov. 2006. Available: http://hdl.handle.net/2438/503.

[7] S. Consoli, "Test datasets for the Minimum Labelling Spanning Tree problem [online]". Available at http://people.brunel.ac.uk/~mapgssc/MLSTP.htm [accessed 12 March 2007].

[8] J. Demšar, "Statistical Comparison of Classifiers over multiple Data Sets," *Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006.

[9] C. Duin and S. Voss, "The Pilot Method: A strategy for heuristic repetition with applications to the Steiner problem in graphs," *Wiley InterScience*, vol. 34(3), pp. 181–191, 1999.

[10] M. Friedman, "A comparison of alternative tests of significance for the problem of m rankings," *Annals of Mathematical Statistics*, vol. 11, pp. 86–92, 1940.

[11] M. Friedman, "The use of ranks to avoid the assumption of normality implicit in the analysis of variance," *Journal of the American Statistical Association*, vol. 32, pp. 675–701, 1937.

[12] F. Glover and G. A. Kochenberger, *Handbook of metaheuristics*, Norwell, MA: Kluwer Academic Publishers, 2003.

[13] D. E. Goldberg, K. Deb, and B. Korb, "Don't worry, be messy," in *Proc. 4th Intern. Conf. on Genetic*

Algorithms, Morgan-Kaufmann, La Jolla, CA, 1991.

[14] P. Hansen and N. Mladenović, "Variable neighbourhood search," in *Handbook of metaheuristics,* F. Glover and G. A. Kochenberger (Eds), Norwell, MA: Kluwer Academic Publishers, 2003, ch. 6, pp. 145–184.

[15] P. Hansen and N. Mladenović, "'Variable neighbourhood search: Principles and applications," *European Journal of Operational Research,* vol. 130, pp. 449–467, 2001.

[16] P. Hansen and N. Mladenović, "Variable neighbourhood search", *Computers and Operations Research,* vol. 24, pp. 1097–1100, 1997.

[17] M. Hollander and D. A. Wolfe, *Nonparametric statistical methods*, 2nd ed., New York: John Wiley & Sons, 1999.

[18] R. L. Iman and J. M. Davenport, "Approximations of the critical region of the Friedman statistic," *Communications in Statistics*, vol. 9, pp. 571–595, 1980.

[19] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science,* vol. 220, no. 4598, pp. 671–680, 1983.

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

[21] P. B. Nemenyi, "Distribution-free multiple comparisons," Ph.D. thesis, Princeton Univ., New Jersey, 1963.

[22] L. S. Pitsoulis and M. G. C. Resende, "Greedy randomized adaptive search procedure," in *Handbook of Applied Optimization*, P. Pardalos and M. G. C. Resende (Eds), Oxford University Press, 2002, pp. 168–183.

[23] M. G. C. Resende and C. C. Ribeiro, "Greedy randomized adaptive search procedures", in *Handbook in Metaheuristics*, F. Glover and G. Kochenberger (Eds), Kluwer Academic Publishers, 2002, pp. 219–249.

[24] R. Sedgewick, *Algorithms in C++: Fundamentals, Data Structures, Sorting, Searching, and Graph Algorithms*, Boston: Addison-Wesley, 2001.

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

[26] R. Van Nes, *Design of multimodal transport networks: A hierarchical approach*, Delft, The Netherlands: Delft University Press, 2002.

[27] S. Voss, A. Fink, and C. Duin, "Looking ahead with the Pilot Method," *Annals of Operations Research*, vol. 136, pp. 285–302, 2004.

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

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

[30] Y. Xiong, B. Golden, and E. Wasil, “Improved Heuristics for the Minimum Labelling Spanning Tree Problem,” *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 6, pp. 700–703, Dec. 2006.

[31] Y. Xiong, B. Golden, and E. Wasil, “A One-Parameter Genetic Algorithm for the Minimum Labelling Spanning Tree Problem,” *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 1, pp. 55–60, Feb. 2005.

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

[33] Y. Xiong, “The Minimum Labelling Spanning Tree Problem and some variants,” Ph.D. thesis, Grad. School Maryland Univ., USA, 2005.

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	B-VNS	GS-VNS
20	20	0.8	*2.4*	2.4	2.4	2.4	2.4	2.4
		0.5	*3.1*	3.2	3.1	3.1	3.1	3.1
		0.2	*6.7*	6.7	6.7	6.7	6.7	6.7
30	30	0.8	*2.8*	2.8	2.8	2.8	2.8	2.8
		0.5	*3.7*	3.7	3.7	3.7	3.7	3.7
		0.2	*7.4*	7.5	7.4	7.4	7.4	7.4
40	40	0.8	*2.9*	2.9	2.9	2.9	2.9	2.9
		0.5	*3.7*	3.7	3.7	3.7	3.7	3.7
		0.2	*7.4*	7.7	7.4	7.4	7.4	7.4
50	50	0.8	*3*	3	3	3	3	3
		0.5	*4*	4.1	4.1	4	4	4
		0.2	*8.6*	8.6	8.6	8.6	8.6	8.6
TOTAL:			*55.7*	56.3	55.8	55.7	55.7	55.7
Parameters			Computational times (milliseconds)					
n	ℓ	d	EXACT	PILOT	MGA	GRASP	B-VNS	GS-VNS
20	20	0.8	*0*	3.2	15.6	1.6	0	0
		0.5	*0*	3	22	0	1.6	0
		0.2	*11*	3.2	23.4	0	1.6	0
30	30	0.8	*0*	6.3	9.4	1.6	0	1.5
		0.5	*0*	7.8	26.5	0	0	0
		0.2	*138*	9.3	45.4	1.5	1.4	3.1
40	40	0.8	*2*	17.2	12.5	1.5	0	1.5
		0.5	*3.2*	18.9	28.2	1.5	1.6	3.1
		0.2	$100.2 \cdot 10^3$	21.8	120.3	15.6	6.2	6.2
50	50	0.8	*3.1*	26.5	21.8	3	1.4	3.1
		0.5	*21.9*	34.3	531.3	9.4	3.2	6.2
		0.2	$66.3 \cdot 10^3$	34.4	93.6	3.2	7.7	8
TOTAL:			$166.7 \cdot 10^3$	185.9	950	38.9	24.7	32.7

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	B-VNS	GS-VNS
100	25	0.8	*1.8*	1.8	1.8	1.8	1.8	1.8
		0.5	*2*	2	2	2	2	2
		0.2	*4.5*	4.5	4.5	4.5	4.5	4.5
	50	0.8	*2*	2	2	2	2	2
		0.5	*3*	3.1	3	3	3	3
		0.2	*6.7*	6.9	6.7	6.7	6.7	6.7
	100	0.8	*3*	3	3	3	3	3
		0.5	*4.7*	4.7	4.7	4.7	4.7	4.7
		0.2	*NF*	10.2	9.9	9.8	9.7	9.7
	125	0.8	*4*	4	4	4	4	4
		0.5	*5.2*	5.4	5.2	5.2	5.2	5.2
		0.2	*NF*	11.3	11.1	11	11	11
TOTAL:			-	58.9	57.9	57.7	57.6	57.6
Parameters			Computational times (milliseconds)					
n	ℓ	d	EXACT	PILOT	MGA	GRASP	B-VNS	GS-VNS
100	25	0.8	*9.4*	9.4	26.5	0	0	0
		0.5	*14*	14.1	29.7	4.6	1.6	4.5
		0.2	*34.3*	21.9	45.3	9.3	1.5	4.8
	50	0.8	*17.8*	48.3	23.5	6.4	3.2	12.6
		0.5	*23.5*	70.3	106.2	51.6	20.3	21.7
		0.2	*$10.2 \cdot 10^3$*	67.2	148.3	57.8	22	26.5
	100	0.8	*142.8*	229.6	254.7	61	132.9	146.9
		0.5	*$2.4 \cdot 10^3$*	242.2	300	28.2	59.1	75.9
		0.2	*NF*	242.1	$9.4 \cdot 10^3$	$1.2 \cdot 10^3$	460.8	514
	125	0.8	*496.9*	340.7	68.7	9.4	9.3	20.2
		0.5	*$179.6 \cdot 10^3$*	359.3	759.4	595.4	490.7	345.4
		0.2	*NF*	353.1	$2 \cdot 10^3$	562.9	448.2	$1.2 \cdot 10^3$
TOTAL:			-	$2 \cdot 10^3$	$13.2 \cdot 10^3$	$2.6 \cdot 10^3$	$1.7 \cdot 10^3$	$2.4 \cdot 10^3$

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	B-VNS	GS-VNS
200	50	0.8	*2*	2	2	2	2	2
		0.5	*2.2*	2.2	2.2	2.2	2.2	2.2
		0.2	*5.2*	5.2	5.2	5.2	5.2	5.2
	100	0.8	*2.6*	2.6	2.6	2.6	2.6	2.6
		0.5	*3.4*	3.4	3.4	3.4	3.4	3.4
		0.2	*NF*	8.3	8.3	8.1	7.9	7.9
	200	0.8	*4*	4	4	4	4	4
		0.5	*NF*	5.5	5.4	5.4	5.4	5.4
		0.2	*NF*	12.4	12.4	12.2	12	12
	250	0.8	*4*	4.1	4	4.1	4	4
		0.5	*NF*	6.3	6.3	6.3	6.3	6.3
		0.2	*NF*	14	14	13.9	13.9	13.9
TOTAL:			-	70	69.8	69.4	68.9	68.9
Parameters			Computational times (milliseconds)					
n	ℓ	d	EXACT	PILOT	MGA	GRASP	B-VNS	GS-VNS
200	50	0.8	*29.7*	42.2	26.5	20.5	0	0
		0.5	*32.7*	92.1	68.8	14.2	10.9	34.4
		0.2	*$5.4 \cdot 10^3$*	207.9	326.6	37.5	170.4	232.8
	100	0.8	*138.6*	492.0	139.3	45.3	106.2	140.8
		0.5	*807.8*	746.9	$1.6 \cdot 10^3$	176.6	140.5	159.4
		0.2	*NF*	851.6	$2.2 \cdot 10^3$	667.2	$2.5 \cdot 10^3$	$2.9 \cdot 10^3$
	200	0.8	*$22.5 \cdot 10^3$*	$3.2 \cdot 10^3$	204.6	43.6	29.4	79.7
		0.5	*NF*	$3.3 \cdot 10^3$	$16.1 \cdot 10^3$	885.6	$2.2 \cdot 10^3$	876.1
		0.2	*NF*	$3.2 \cdot 10^3$	$12.7 \cdot 10^3$	$9.4 \cdot 10^3$	$29.8 \cdot 10^3$	$33.7 \cdot 10^3$
	250	0.8	*$20.6 \cdot 10^3$*	$5 \cdot 10^3$	$2.2 \cdot 10^3$	$4.9 \cdot 10^3$	$1.4 \cdot 10^3$	$1.5 \cdot 10^3$
		0.5	*NF*	$5.2 \cdot 10^3$	$17.6 \cdot 10^3$	506.0	$2.6 \cdot 10^3$	$2.3 \cdot 10^3$
		0.2	*NF*	$4.6 \cdot 10^3$	$26.4 \cdot 10^3$	$1.4 \cdot 10^3$	$1.6 \cdot 10^3$	$1.5 \cdot 10^3$
TOTAL:			-	$26.9 \cdot 10^3$	$79.6 \cdot 10^3$	$18.1 \cdot 10^3$	$40.6 \cdot 10^3$	$43.4 \cdot 10^3$

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	B-VNS	GS-VNS
500	125	0.8	*2*	2	2	2	2	2
		0.5	*2.6*	2.6	2.6	2.6	2.6	2.6
		0.2	*NF*	6.3	6.2	6.2	6.2	6.2
	250	0.8	*3*	3	3	3	3	3
		0.5	*NF*	4.2	4.3	4.2	4.1	4.1
		0.2	*NF*	10	10.1	9.9	9.9	9.9
	500	0.8	*NF*	4.7	4.7	4.7	4.7	4.7
		0.5	*NF*	6.7	7.1	6.5	6.5	6.5
		0.2	*NF*	15.9	16.6	15.9	15.8	15.8
	625	0.8	*NF*	5.1	5.4	5.1	5.1	5.1
		0.5	*NF*	7.9	8.3	7.9	7.9	7.9
		0.2	*NF*	18.5	19.1	18.4	18.3	18.3
TOTAL:			-	86.9	89.4	86.4	86.1	86.1
Parameters			Computational times (milliseconds)					
n	ℓ	d	EXACT	PILOT	MGA	GRASP	B-VNS	GS-VNS
500	125	0.8	370	$1.2 \cdot 10^3$	18	152	21.9	45
		0.5	597	$2.9 \cdot 10^3$	$2.6 \cdot 10^3$	455	860.8	560
		0.2	*NF*	$7.3 \cdot 10^3$	$57.1 \cdot 10^3$	$4 \cdot 10^3$	$3.9 \cdot 10^3$	$3.7 \cdot 10^3$
	250	0.8	*$5.3 \cdot 10^3$*	$20.9 \cdot 10^3$	516	248	67.1	490
		0.5	*NF*	$29.6 \cdot 10^3$	$28 \cdot 10^3$	583	$96.2 \cdot 10^3$	$26.9 \cdot 10^3$
		0.2	*NF*	$30.4 \cdot 10^3$	$181.2 \cdot 10^3$	$3.3 \cdot 10^3$	$16.1 \cdot 10^3$	$10.2 \cdot 10^3$
	500	0.8	*NF*	$128.3 \cdot 10^3$	$117.5 \cdot 10^3$	$28.1 \cdot 10^3$	$15.1 \cdot 10^3$	$8.6 \cdot 10^3$
		0.5	*NF*	$131.9 \cdot 10^3$	$170.9 \cdot 10^3$	$90.9 \cdot 10^3$	$26 \cdot 10^3$	$110.2 \cdot 10^3$
		0.2	*NF*	$115.9 \cdot 10^3$	$241.8 \cdot 10^3$	$20.2 \cdot 10^3$	$235.7 \cdot 10^3$	$50.3 \cdot 10^3$
	625	0.8	*NF*	$204 \cdot 10^3$	$51.9 \cdot 10^3$	$4.9 \cdot 10^3$	$67.9 \cdot 10^3$	970
		0.5	*NF*	$200.9 \cdot 10^3$	$222.2 \cdot 10^3$	$35.7 \cdot 10^3$	$132.7 \cdot 10^3$	$33.9 \cdot 10^3$
		0.2	*NF*	$181.7 \cdot 10^3$	$297.8 \cdot 10^3$	$53.1 \cdot 10^3$	$175.9 \cdot 10^3$	$60 \cdot 10^3$
TOTAL:			-	$1055 \cdot 10^3$	$1371.5 \cdot 10^3$	$213.8 \cdot 10^3$	$770.4 \cdot 10^3$	$395.9 \cdot 10^3$

Table 5
Pairwise differences of the average ranks of the algorithms

Algorithm (rank)	B-VNS (1.9)	GS-VNS (2)	GRASP (2.26)	PILOT (4.4)	MGA (4.45)
B-VNS (1.9)	-	0.1	0.36	**2.5**	**2.55**
GS-VNS (2)	-	-	0.26	**2.4**	**2.45**
GRASP (2.26)	-	-	-	**2.14**	**2.19**
PILOT (4.4)	-	-	-	-	0.05
MGA (4.45)	-	-	-	-	-

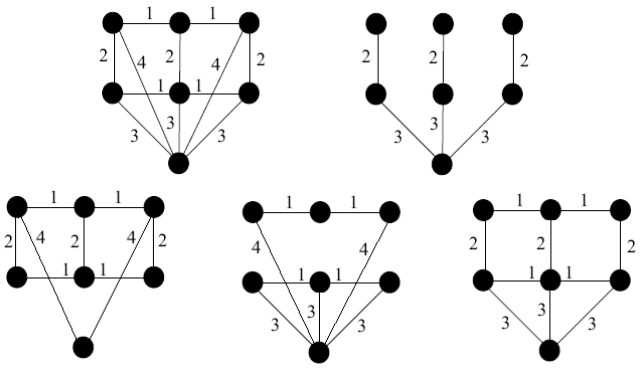

Figure 1: The top two graphs show a sample graph and its optimal solution. The bottom three graphs show some feasible solutions.

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

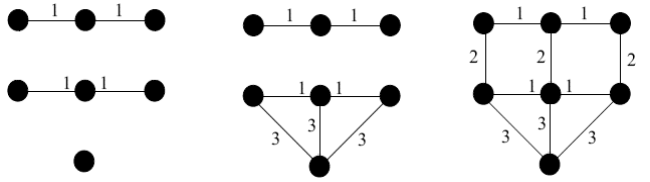

Figure 2: Example illustrating the steps of the revised MVCA.

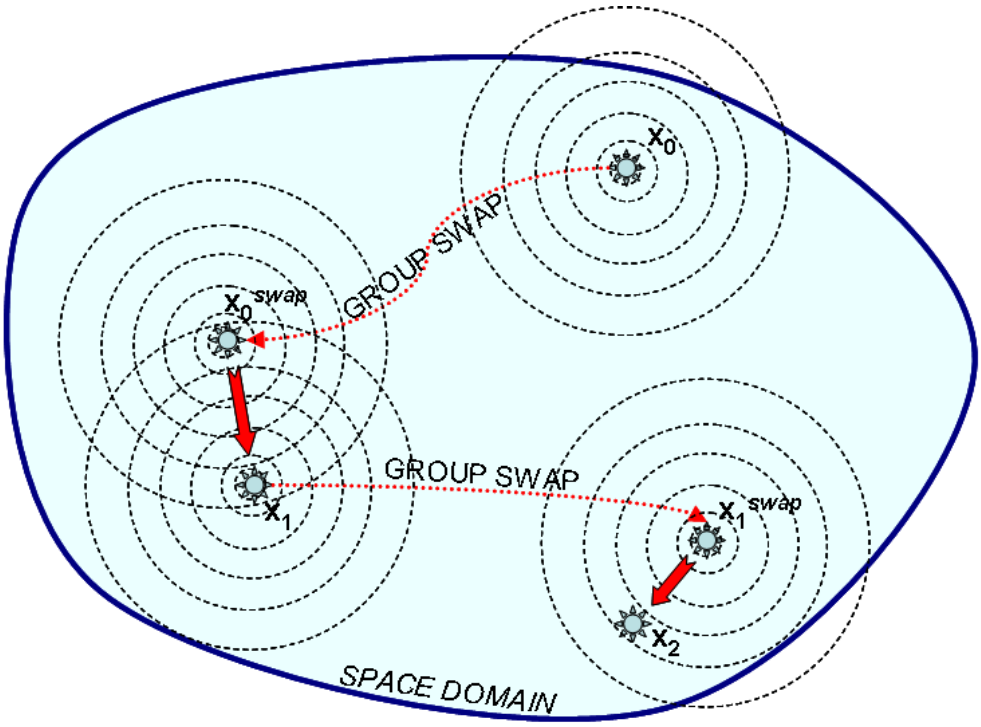

Figure 3: Example illustrating the steps of Group-Swap VNS.