Ann Oper Res (2009) 172: 71–96
DOI 10.1007/s10479-008-0507-y

Variable neighbourhood search for the minimum labelling Steiner tree problem

Sergio Consoli · Kenneth Darby-Dowman · Nenad Mladenović · José Andrés Moreno-Pérez

Published online: 15 January 2009

Abstract We present a study on heuristic solution approaches to the minimum labelling Steiner tree problem, an NP-hard graph problem related to the minimum labelling spanning tree problem. Given an undirected labelled connected graph, the aim is to find a spanning tree covering a given subset of nodes of the graph, whose edges have the smallest number of distinct labels. Such a model may be used to represent many real world problems in telecommunications and multimodal transportation networks. Several metaheuristics are proposed and evaluated. The approaches are compared to the widely adopted Pilot Method and it is shown that the Variable Neighbourhood Search that we propose is the most effective metaheuristic for the problem, obtaining high quality solutions in short computational running times.

Keywords Metaheuristics · Combinatorial optimization · Minimum labelling Steiner tree problem · Variable neighbourhood search · Graphs

Introduction

Since the early years of Operational Research (OR), there has been much interest in combinatorial optimization (CO) problems formulated on graphs and their practical applica-

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.

S. Consoli (✉) · K. Darby-Dowman · N. Mladenović
CARISMA and NET-ACE, School of Information Systems, Computing and Mathematics, Brunel University, Uxbridge, Middlesex, UB8 3PH, UK
e-mail: sergio.consoli@brunel.ac.uk

J.A. Moreno-Pérez
Facultad de Matemáticas, DEIOC, IUDR, Universidad de La Laguna, 4a planta Astrofisico Francisco Sánchez s/n, 38271 Santa Cruz de Tenerife, Spain

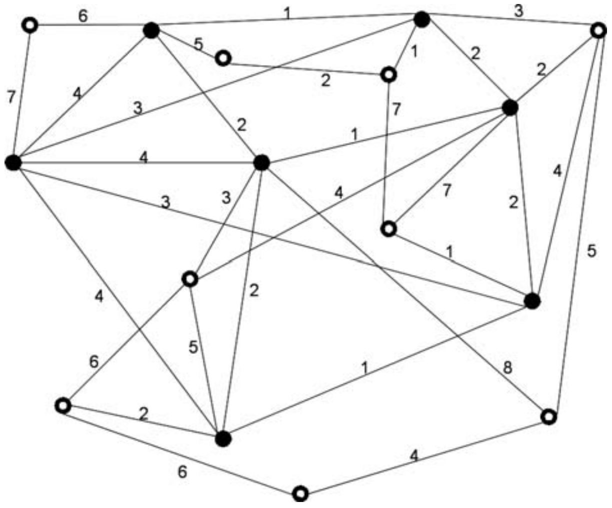

Fig. 1 Example of an input graph of the MLSteiner problem

tions (Avis et al. 2005). Most of these problems are NP-hard; thus, there is a need for heuristics and approximate solution approaches with performance guarantees.

In this paper, we focus on the *minimum labelling Steiner tree (MLSteiner) problem*. Given a graph with labelled (or colored) edges, one seeks a spanning tree covering a subset of nodes (basic nodes) of the graph, whose edges have the least number of distinct labels (or colors).

This problem has many applications in real-world problems. For example, in telecommunications networks, a node may communicate with other nodes by means of different types of communications media. Considering a set of basic nodes that must be connected, the construction cost may be reduced, in some situations, by connecting the basic nodes with the smallest number of possible communications types (Tanenbaum 1989).

Another example is given by multimodal transportation networks (Van-Nes 2002). A multimodal transportation network can be represented by a graph where a color is assigned to each edge, denoting a different company managing that edge, and each node represents a different location. It is often desirable to provide a complete service between a basic set of locations, without cycles, using the minimum number of companies, in order to minimize the cost.

The minimum labelling Steiner tree problem is formally defined as a network or graph problem as follows:

The minimum labelling Steiner tree problem (MLSteiner problem):
- Let $G = (V, E, L)$ be a labelled, connected, undirected graph, where V is the set of nodes, E is the set of edges, that are labelled on the set L of labels (or colors).
- Let $Q \subseteq V$ be a set of nodes that must be connected (basic nodes).
  $\Rightarrow$ Find an arbitrary spanning tree T of the subgraph connecting all the basic nodes Q such that $|L_T|$ is minimized (where L_T is the set of colors used by T).

Figure 1 shows an example of an input graph, where the solid vertices represent the basic nodes. The minimum labelling Steiner tree solution of this example is shown in Fig. 2.

In order to solve the MLSteiner problem, it is easier to work firstly with feasible solutions instead of spanning trees. A feasible solution is defined as a set of colors $C \subseteq L$, such that all the edges with labels in C represent a connected subgraph of G which spans all the basic nodes Q. If C is a feasible solution, then any spanning tree of C has at most $|C|$ labels.

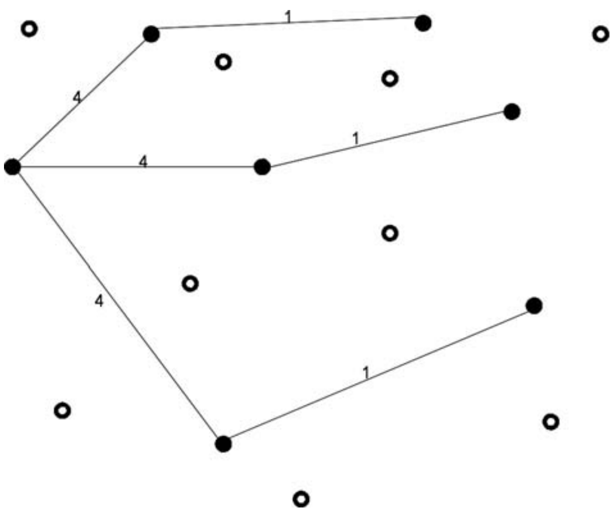

Fig. 2 Minimum labelling Steiner tree solution for the graph of Fig. 1

Thus, in order to solve the MLSteiner problem, we first seek a feasible solution with the smallest number of colors (Cerulli et al. 2006).

In this paper, we propose several new metaheuristics for the MLSteiner problem: Greedy Randomized Adaptive Search Procedure (GRASP), Discrete Particle Swarm Optimization (DPSO), Variable Neighbourhood Search (VNS), and Group-Swap Variable Neighbourhood Search (GS-VNS), which is a hybridization between Variable Neighbourhood Search and Simulated Annealing. Computational results for these approaches are compared with those from the Pilot Method (PM) by Cerulli et al. (2006), which is considered to be the best performing heuristic in the current literature, and with those from an exact method.

The structure of the paper is as follows. We first describe the problem, its origins, and we review the associated literature. As the minimum labelling Steiner tree problem is a direct extension of the well-known Steiner tree problem and of the minimum labelling spanning tree problem, we discuss these basic problems. Details of the methods considered are presented in Sect. 2. Section 3 contains a computational analysis and evaluation and, finally, conclusions are described in Sect. 4. For a survey on the basic concepts of metaheuristics and combinatorial optimization, the reader is referred to Voß et al. (1999) and Blum and Roli (2003).

1 Origin of the problem

The minimum labelling Steiner tree problem was introduced by Cerulli et al. (2006). It is a graph combinatorial optimization problem extending the well-known Steiner tree (Steiner) problem and the minimum labelling spanning tree (MLST) problem.

Given a graph with positive-weighted edges, and with a subset of basic nodes (or terminals), the *Steiner tree problem* consists of finding a minimum-weight tree spanning all the basic nodes. This problem dates back to Fermat, who formulated it as a geometric problem: find a point p in the Euclidean plane minimizing the sum of the distances to three given points. This was solved before 1640 by Torricelli (Krarup and Vajda 1997). Subsequently, Steiner worked on the general problem for n points. More details appear in Hwang et al. (1992). Expositions on the difficulty of the Steiner problem can be found in Karp (1975) and Garey et al. (1977), while several heuristics for the Steiner problem in graphs are reported in Voß (2000) and Grimwood (1994).

A large number of real-world applications of the Steiner problem exist, most of them relate to network design (Winter 1987) and telecommunications (Voß 2006). Steiner problems arising in the layout of connection structures in networks, such as topological network design, location, and in VLSI (Very Large Scale Integrated) circuit design, are discussed in Francis et al. (1992) and Korte et al. (1990). Furthermore, analogies can be drawn between minimum Steiner trees and minimum energy configurations in certain physical systems (Miehle 1958).

The *minimum labelling spanning tree (MLST) problem* is used where, given a graph with colored (labelled) edges, one seeks a spanning tree with the least number of colors (labels). The MLST problem may also be used to represent many real-world problems in telecommunications networks (Raghavan and Anandalingam 2003) and multimodal transportation networks (Van-Nes 2002). This problem was first introduced by Chang and Leu (1997), interested by its applications in communications network design. They established its NP-hard complexity, and proposed a polynomial time heuristic, the maximum vertex covering algorithm. Several further modifications of this heuristic are included in Krumke and Wirth (1998), Wan et al. (2002), and Xiong et al. (2005b), together with discussions on convergence.

Other heuristic approaches to the MLST problem have been proposed in the literature. Cerulli et al. (2005) presented a comparison of metaheuristics: Tabu Search, Simulated Annealing, Pilot Method, and an ad-hoc implementation of Variable Neighbourhood Search (VNS). Among them, the best results were obtained by the Pilot Method.

Genetic algorithms (GAs) were also applied to the MLST problem in Xiong et al. (2005a). In addition, Xiong et al. (2006) presented some improved approaches, obtained by modifying the Pilot Method and the GAs. They named their best performing implementation as Modified Genetic Algorithm (MGA).

In Consoli et al. (2008a, 2008b), a Greedy Randomized Adaptive Search Procedure and different versions of Variable Neighbourhood Search were proposed. A comparison with the results provided by the best performing methods from the literature, the Pilot Method by Cerulli et al. (2005) and the Modified Genetic Algorithm by Xiong et al. (2006), showed that these heuristics based on GRASP and VNS obtained the best performance, producing high-quality solutions in short computational running times.

The MLSteiner problem was first considered by Cerulli et al. (2006) as an extension of the Steiner problem and the MLST problem. They also compared their Pilot Method with some other metaheuristics for the MLSteiner problem: Tabu Search, Simulated Annealing, and some implementations of Variable Neighbourhood Search. From their analysis, the Pilot Method was shown to be the best performing heuristic for the problem (Cerulli et al. 2006).

The success of the heuristic solution approaches for the MLST problem proposed by Consoli et al. (2008a, 2008b) provided the motivation for considering the implementation of similar approaches for the MLSteiner problem, and this is the focus of the work reported in this paper.

2 Description of the algorithms

In this section, we introduce an exact method for the MLSteiner problem, and analyze the Pilot Method by Cerulli et al. (2006). We then describe the main features of other metaheuristics that we propose for the MLSteiner problem: a Greedy Randomized Adaptive Search

Procedure, a Discrete Particle Swarm Optimization, a Variable Neighbourhood Search, and a hybrid approach that we call Group-Swap Variable Neighbourhood Search.

Before going into the details of these algorithms, it is useful to define the concept of a Steiner component (Cerulli et al. 2006). Given an undirected, connected, labelled input graph, a Steiner component is a connected component of the input graph containing at least one basic node. We will make use of this concept throughout the paper.

2.1 Exact method

This exact approach to the MLSteiner problem is based on a backtracking procedure. Given a labelled connected undirected graph $G = (V, E, L)$ with n vertices, m edges, ℓ labels, and a subset $Q \subseteq V$ of basic nodes, the exact method performs a branch and prune procedure in the partial solution space based on a recursive procedure, *Test*. The details are specified in Algorithm 1.

The procedure *Test* starts from an empty set of colors and iteratively builds a solution by adding colors one by one until all the basic nodes, $Q \subseteq V$, are connected. In this method, all the possible combinations of colors are considered, and so its running time is computationally burdensome. The running time grows exponentially with the dimension of the graph (number of nodes and colors), and the reduction in the density of the graph.

```
Input: A labelled, undirected, connected graph $G = (V, E, L)$, with $n$ vertices, $m$ edges, $\ell$
        labels, and $Q \subseteq V$ basic nodes;
Output: A spanning tree $T$;
Initialization:
- Let $C \leftarrow 0$ be the initially empty set of used colors;
- 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 colors;
- 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 Steiner components of $C$, i.e., the number of connected
components of the subgraph $(Q, E(C))$;
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 color $c$ : $Test(C \cup \{c\})$;
        end
    end
end
```

Algorithm 1: Exact method for the MLSteiner problem

In order to speed up this method, the following procedure is adopted. Let $C^* \subseteq L$ be a current solution, and $C' \subseteq L$ be an incomplete solution to evaluate. If the dimension of C' is equal to $|C^*| - 2$, we should try to add all the colors one by one to check if it is possible to find a better solution for C^* with a smaller dimension, that is, $|C^*| - 1$. Instead of trying to add all the colors one by one to complete C', we only consider the colors with a frequency at least equal to the actual number of connected components minus 1 (in other words, we consider only the candidate colors which may yield a connected graph if added to the incomplete solution C'). If this requirement is not satisfied, the incomplete solution can be rejected, speeding up the search process.

If either the problem size is small or the optimal objective function value is small, the running time of this exact method is acceptable and it is possible to obtain the exact solution.

2.2 Pilot method

The Pilot Method (PM) metaheuristic was first introduced by Duin and Voß (1999) for the Steiner tree problem, and was applied with success to several combinatorial optimization problems (Voß et al. 2004). The core idea of this metaheuristic is to exhaust tentatively all the possible choices with respect to a reference solution, called the master solution, by means of a basic constructive heuristic. For each possible choice, the basic heuristic (or application process) works as a building block for the master solution, by adding components until a feasible solution is obtained. When all the possible choices have been evaluated, the master solution is updated with the best choice, and the procedure proceeds iteratively until the user termination conditions are reached. Further details are included in Voß et al. (2004).

Cerulli et al. (2005) applied the Pilot Method to the MLST problem, and following the same procedure to the MLSteiner problem (Cerulli et al. 2006). They also performed a comparison between PM and other ad hoc metaheuristics (Tabu Search, Simulated Annealing, and Variable Neighbourhood Search) for different instances of the MLSteiner problem (Cerulli et al. 2006). From their computational analysis, the Pilot Method obtained the best results.

The details of the Pilot Method proposed by Cerulli et al. (2006) for the MLSteiner problem are specified in Algorithm 2. PM starts from the null solution (an empty set of colors) as master solution, M. Then for each element $i \notin M$, it tries to extend tentatively a copy of M to a (fully grown) feasible solution including i, built by the application process. The application process is a greedy procedure, which at each step, inserts in the partial solution the color producing the minimum number of Steiner components at that specific step, and stopping when a feasible solution is obtained. At the end of the execution of the application process, a local search mechanism is included to try to greedily drop colors (i.e., the associated edges), from the least frequently occurring color to the most frequently occurring one, whilst retaining feasibility. The number of colors produced by the feasible solution obtained from $M \leftarrow M \cup \{i\}$ is used as objective function for each candidate $i \notin M$. When all the possible candidate colors with respect to the master solution have been evaluated, a candidate i^* with minimum objective function value is added to the master solution ($M \leftarrow M \cup \{i^*\}$). 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 mechanism is repeated for all the successive stages of the Pilot Method, until no further colors need to be added to the master solution (i.e., a feasible master solution is produced). Alternatively, some user termination conditions, such as the maximum allowed CPU time or the maximum number of iterations, may be imposed in order to allow the algorithm to proceed until these conditions are satisfied. The last master solution corresponds to the best solution to date and it is produced as the output of the method.

```
**Input**: A labelled, undirected, connected graph $G = (V, E, L)$, with $n$ vertices, $m$ edges, $\ell$
        labels, and $Q \subseteq V$ basic nodes;
**Output**: A spanning tree $T$;
*Initialization:*
- Let $M \leftarrow 0$ be the initially empty master solution;
- Let $H = (V, E(M))$ be 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 Steiner components of $H = (V, E(M))$;
- Let $M^* \leftarrow L$ and $C \leftarrow 0$ sets of colors;
- Let $H^* = (V, E(M^*))$ be 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 (not termination conditions) OR ($Comp(M) > 1$) do
        Set $C \leftarrow M$;
        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
                Select the unused color $u \in (L - M)$ that minimizes $Comp(M \cup \{u\})$;
                Add label $u$ to the solution: $M \leftarrow M \cup \{u\}$;
                Update $H = (V, E(M))$ 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
            Restore the current master solution: $M \leftarrow C$;
            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
        Select the unused color $u \in (L - M)$ that minimizes $Comp(M \cup \{u\})$;
        Add label $u$ to the solution: $M \leftarrow M \cup \{u\}$;
        Update $H = (V, E(M))$ and $Comp(M)$;
    end
    $\Rightarrow$ Take any arbitrary spanning tree $T$ of $H = (V, E(M))$.
end

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

Algorithm 2: The Pilot Method for the MLSteiner problem (Cerulli et al. 2006)

Note that when the application process is applied to complete a partial solution, in case of ties in the minimum number of Steiner components, a label is selected at random within the set of labels producing the minimum number of components. Furthermore, note that no external parameters need to be tuned by the user for the PM.

Considering an input graph $G = (V, E, L)$ with $|L| = \ell$ number of labels, the overall computational time of the application process is $O(\ell)$ since up to ℓ labels may be added. 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 PM is $O(\ell^2)$ times the computational time of the application process, leading to an overall complexity $O(\ell^3)$.

2.3 Greedy Randomized Adaptive Search Procedure

The GRASP (Greedy Randomized Adaptive Search Procedure) methodology was developed in the late 1980s, and the acronym was coined by Feo and Resende (1989). It was first used to solve set covering problems, but was then extended to a wide range of combinatorial optimization problems (Pitsoulis and Resende 2002).

GRASP is an iterative metaheuristic consisting of two phases: a construction phase, followed by a local search phase. The construction phase builds a feasible solution by applying a randomized greedy procedure. The randomized greedy procedure builds a solution by iteratively creating a candidate list of elements that can be added to the partial solution, and then randomly selecting an element from this list.

The candidate list (RCL_α: Restricted Candidate List of length α) is created by evaluating the elements not yet included in the partial solution. A greedy function, depending on the specifications of the problem, is used to perform this evaluation. Only the best elements, according to this greedy function, are included in RCL_α. The size α of the candidate list can be limited either by the number of elements, or by their quality with respect to the best candidate element.

At each iteration, one new element is randomly selected from RCL_α, added to the current solution, and the candidate list is updated. The construction phase stops when a feasible solution is obtained. The obtained solution is not necessarily locally optimal, so a local search phase is included to try to improve it. This phase uses a local search mechanism which, iteratively, tries to replace the current solution with a better neighbouring solution, until no better solution can be found. Different strategies may be used in order to evaluate the neighbourhood structure. 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. The final result of GRASP is the best solution obtained in all the computations.

The solutions obtained by GRASP are usually of good quality because it offers fast local convergence (high intensification capability) as a result of the greedy aspect of the procedure used in the construction phase, and of the local search mechanism; and also a wide exploration of the solution space (high diversification capability) for the randomization used in the selection of a new element from RCL_α. Success of a particular GRASP implementation depends on a number of different factors, such as the efficiency of the randomized greedy procedure used, the choice of the neighbourhood structure, and the neighbourhood search technique. Details can be found in Resende and Ribeiro (2003), including several new components extending the basic scheme of GRASP (e.g., parameter variations, bias functions, memory and learning, reactive GRASP, and hybrid schemas).

The GRASP that we propose for the MLSteiner problem is specified in Algorithm 3. For the construction phase of GRASP, we make use of a value-based restricted candidate list in

```
Input: A labelled, undirected, connected graph $G = (V, E, L)$, with $n$ vertices, $m$ edges, $\ell$
        labels, and $Q \subseteq V$ basic nodes;
Output: A spanning tree $T$;
Initialization:
- Let $C \leftarrow 0$ be the initially empty set of used colors for each iteration;
- Let $H = (V, E(C))$ be the subgraph of $G$ restricted to $V$ and edges with labels in $C$,
where $E(C) = \{e \in E : L(e) \in C\}$;
- Let $C' \leftarrow L$ be the global set of used colors;
- 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 Steiner components of $C$, i.e. the number of connected
components of the subgraph $(Q, E(C))$;
- Let $RCL_\alpha \leftarrow 0$ be the restricted candidate list of length $\alpha$;
begin
    repeat
        Set $C \leftarrow 0$ and update $H = (V, E(C))$;
        Construction-Phase(C);
        Local-Search(C);
        if $|C| < |C'|$ then
            Move $C' \leftarrow C$;
            Update $H' = (V, E(C'))$;
        end
    until termination conditions ;
    $\Rightarrow$ Take any arbitrary spanning tree $T$ of $H' = (V, E(C'))$.
end

Procedure Construction-Phase(C):
Set $RCL_\alpha \leftarrow L$ and $\alpha = \ell$;
Add a random color $c \in RCL_\alpha$ to the set of used colors: $C \leftarrow C \cup \{c\}$;
Update $H = (V, E(C))$ and $Comp(C)$;
while $Comp(C) > 1$ do
    Set $RCL_\alpha \leftarrow \{\forall c \in L /$ minimizes $Comp(C \cup \{c\})\}$;
    Add a random color $c \in RCL_\alpha$ to the set of used colors: $C \leftarrow C \cup \{c\}$;
    Update $H = (V, E(C))$ and $Comp(C)$;
end
```

Algorithm 3: Greedy Randomized Adaptive Search Procedure for the MLSteiner problem

order to select the colors to be placed in RCL_α. This is an extension of the classic greedy criterion used in GRASP, consisting of placing in the list only the candidate colors having a greedy value (the number of Steiner components in the case of the MLSteiner problem) not greater than a user-defined threshold (Resende and Ribeiro 2003). In our implementation, complete randomization is used to choose the initial color to add. This corresponds to setting the threshold to $+\infty$, meaning that the candidate list is filled with all the colors of the graph (length α = total number of colors). For the remaining colors to add, the list is formed by considering only the colors that result in the minimum number of Steiner components at the specific step, in order to further intensify the search process. This means fixing the threshold as the minimum number of Steiner components produced by the candidate colors at the specific step (i.e., only the colors producing the least number of Steiner components at that step constitute the candidate list).

At the end of the construction phase of GRASP, the successive local search phase consists of trying to greedily drop some labels (i.e., the associated edges) from the current solution, whilst retaining feasibility. It yields a further improvement to the intensification phase of the algorithm.

2.4 Discrete Particle Swarm Optimization

Over the years, evolutionary algorithms have been widely used as robust techniques for solving hard combinatorial optimization problems. Their behavior is directed by the evolution of a population searching for the optimum. Particle Swarm Optimization (PSO) is an evolutionary algorithm proposed by Kennedy and Eberhart (1995). As is the case with genetic algorithms, PSO is a population-based technique, inspired by the social behavior of individuals (or particles) inside swarms occurring in nature, such as flocks of birds or schools of fish. However, unlike genetic algorithms, it has no crossover and mutation operators, is easy to implement, and requires few parameter settings and low computational memory.

The standard PSO (Kennedy and Eberhart 2001) considers a swarm S containing n_s particles ($S = 1, 2, \dots, n_s$) in a d-dimensional continuous solution space. Each ith particle of the swarm has a position $x_i = (x_{i1}, x_{i2}, \dots, x_{ij}, \dots, x_{id})$ associated with it, and a velocity $v_i = (v_{i1}, v_{i2}, \dots, v_{ij}, \dots, v_{id})$. The position x_i represents a solution for the problem, while the velocity v_i gives the change rate for the position of particle i in the next iteration. Indeed, considering an iteration k, the position of particle i is adjusted according to

$$x_i^k = x_i^{k-1} + v_i^k. \tag{1}$$

Each particle i of the swarm communicates with a social environment or neighbourhood $N(i) \subseteq S$, which may change dynamically and represents the group of particles with which particle i communicates. In nature, a bird adjusts its position in order to find a better position, according to its own experience and the experience of its companions. In the same manner, consider an iteration k of the PSO algorithm. Each particle i updates its velocity reflecting the attraction of its best position so far (b_i) and the best position (g_i) of its social neighbourhood $N(i)$, following the equation:

$$v_i^k = c_1 \xi v_i^{k-1} + c_2 \xi \left(b_i - x_i^{k-1}\right) + c_3 \xi \left(g_i - x_i^{k-1}\right). \tag{2}$$

The parameters c_i are positive constant weights applied to the three factors that influence the velocity of the particle i, while the term ξ refers to a random number with uniform distribution in [0, 1] that is independently generated at each iteration.

Since the original PSO is applicable to optimization problems with continuous variables, several adaptations of the method to discrete problems, known as Discrete Particle Swarm Optimization (DPSO), have been proposed (Kennedy and Eberhart 1997). In this paper, we make use of the DPSO procedure introduced by Moreno-Pérez et al. (2007).

This DPSO considers a swarm S containing n_s particles ($S = 1, 2, \dots, n_s$) whose positions x_i evolve in the discrete solution space, jumping from a solution to another. In such a case, the notion of velocity used in the standard PSO loses its meaning, and it is not considered. Furthermore, the weights of the updating equation used in the standard PSO are interpreted as probabilities that at each iteration, each particle has a random behavior, or acts in a manner guided by the effect of attractors. The effect of the attraction of a position causes the given particle to jump toward this attractor. An inspiration from nature for this process is found in frogs, which jump from lily pad to lily pad in a pool.

```
**Input**: A labelled, undirected, connected graph $G = (V, E, L)$, with $n$ vertices, $m$
    edges, $\ell$ labels, and $Q \subseteq V$ basic nodes;
**Output**: A spanning tree $T$;
*Initialization:*
- Let $C \leftarrow 0$ be a set of colors, initially empty;
- 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 $n_s$ of the swarm $S$;
**begin**
    Generate the initial swarm $S$ with positions at random:
    $X = [x_0, x_1, \dots, x_{n_s}] \leftarrow$*Generate-Swarm-At-Random(G)*;
    Update the vector of the best positions $B = [b_0, b_1, \dots, b_{n_s}] \leftarrow X$;
    Extract the best position among all the particles: $g^* \leftarrow$ *Extract-The-Best(S, X)*;
    **repeat**
        **for** $i = 1$ *to* $n_s$ **do**
            **if** $i = 1$ **then** Initialize the best position of the social neighbourhood of $i$:
            $g_i \leftarrow \ell$;
            **else** Update the best position of the social neighbourhood of $i$: $g_i \leftarrow g_{i-1}$;
            Select at random a number between 0 and 1: $\xi$=*Random(0, 1)*;
            **if** $\xi \in [0, 0.25[$ **then** *selected* $\leftarrow x_i$;
            **else if** $\xi \in [0.25, 0.5[$ **then** *selected* $\leftarrow b_i$;
            **else if** $\xi \in [0.5, 0.75[$ **then** *selected* $\leftarrow g_i$;
            **else if** $\xi \in [0.75, 1[$ **then** *selected* $\leftarrow g^*$;
            Combine particle $i$ and the selected particle: $x_i \leftarrow$ *Combine($x_i$, selected)*;
            *Local-Search(i, $x_i$)*;
            **if** $|x_i| < |b_i|$ **then** Update the best position of the given particle $i$: $b_i \leftarrow x_i$;
            **if** $|x_i| < |g_i|$ **then** Update the best position of the social neighbour. of $i$:
            $g_i \leftarrow x_i$;
            **if** $|x_i| < |g^*|$ **then** Update the global best position to date: $g^* \leftarrow x_i$;
        **end**
    **until** *termination conditions* ;
    Set $C \leftarrow g^*$;
    Update $H = (V, E(C))$;
    $\Rightarrow$ Take any arbitrary spanning tree $T$ of $H = (V, E(C))$.
**end**

***Procedure Combine($x_i$, selected):***
Let $Comp(x_i)$ be the number of Steiner components of $x_i$;
Select a random integer between 0 and $|x_i|$: $\psi$=*Random(0, $|x_i|$)*;
**for** $j = 1$ *to* $\psi$ **do**
    Select at random a number between 0 and 1: $\xi$=*Random(0, 1)*;
    **if** $\xi \leq 0.5$ **then**
        Select at random a color $c' \in x_i$;
        Delete label $c'$ from the position of the given particle: $x_i \leftarrow x_i - \{c'\}$;
    **else**
        Select at random a color $c' \in$ *selected*;
        Add label $c'$ to the position of the given particle $i$: $x_i \leftarrow x_i \cup \{c'\}$;
    **end**
**end**
Update $Comp(x_i)$;
**while** $Comp(x_i) > 1$ **do**
    Select at random an unused color $u \in (L - x_i)$;
    Add label $u$ to the position of the given particle $i$: $x_i \leftarrow x_i \cup \{u\}$;
    Update $Comp(x_i)$;
**end**
```

Algorithm 4: Discrete Particle Swarm Optimization for the MLSteiner problem

Given a particle i, three attractors are considered: its own best position (b_i), the best position of its social neighbourhood (g_i), and the global best position (g^*). Indeed, considering a generic iteration k, the update equation for the position x_i of a particle i is

$$x_i^k = c_1 x_i^{k-1} \oplus c_2 b_i \oplus c_3 g_i \oplus c_4 g^*. \tag{3}$$

The meaning of this equation is that the ith particle with position x_i performs random jumps with respect to its current position with probability c_1, improving jumps approaching b_i with probability c_2, improving jumps approaching g_i with probability c_3, and improving jumps approaching g^* with probability c_4. In order to implement this operation, a random number ξ is generated in order to select the type of jump to be chosen. A jump approaching an attractor consists of modifying a feature of the current solution with the corresponding feature of the selected attractor (or giving an arbitrary value in the case of the random jump). For the MLSteiner problem, the features of a solution are the colors that are included in the solution, while the parameters c_1, c_2, c_3, and c_4, are set to 0.25. Further details of the DPSO that we propose for the MLSteiner problem are specified in Algorithm 4.

The position of a particle in the swarm is encoded as a feasible solution to the MLSteiner problem. The initial positions $X = [x_0, x_1, \ldots, x_{n_s}]$ of the swarm S, containing n_s particles, are generated by starting from empty sets of colors and adding at random colors until feasible solutions emerge. Then for each particle of the swarm, a random number ξ between 0 and 1 is selected. Considering the ith particle of the swarm, if ξ belongs to $[0, 0.25[$ the current position of the given particle is selected ($selected \leftarrow x_i$) in order to perform a random jump. Otherwise, if ξ is in $[0.25, 0.5[$ the best position to date (b_i) of the given particle is selected ($selected \leftarrow best\_s(p)$) as attractor for the movement of x_i. Instead, if $\xi \in [0.5, 0.75[$, the selected attractor is the best position g_i of the social neighbourhood, interpreted as the best position obtained within the swarm in the current iteration. For the remaining case, if $\xi \in [0.75, 1[$ the selected attractor is the best position to date obtained by all the particles, which is called the global best position to date (g^*).

Afterward, the ith particle with current position x_i performs a jump approaching the selected attractor by means of the procedure *Combine*. This procedure first selects a random integer ψ between 0 and $|x_i|$. Successively, it either drops some colors from x_i, or randomly picks up some colors from the selected attractor and adds to x_i, until ψ colors have been added or deleted with respect to x_i. Note that if an infeasible x_i is obtained at this stage, further colors are added at random to x_i in order to restore feasibility. At the end of the procedure *Combine*, a local search procedure is applied to the resulting particle (*Local-Search(i, x_i)*), in order to try to delete some colors from x_i whilst retaining the feasibility. Then all the attractors (b_i, g_i, g^*) are updated, and the same procedure is repeated for all the particles in the swarm. The entire algorithm continues until the user termination conditions are satisfied.

2.5 Variable Neighbourhood Search

Variable Neighbourhood Search (VNS) is an effective metaheuristic introduced by Hansen and Mladenović (1997). The basic idea behind this method is to define a neighbourhood structure for the solution space, and to explore different increasingly distant neighbourhoods whenever a local optimum is reached by a prescribed local search.

At the starting point, a set of $k_{\max}$ (a parameter) neighbourhoods (N_k, with $k = 1, 2, \ldots, k_{\max}$), is selected. A stopping condition is determined (either the maximum allowed CPU time, or the maximum number of iterations, or the maximum number of iterations between two successive improvements), and an initial feasible solution found (at random, in our case). Denoting by $N_k(C)$, the set of solutions in the kth neighbourhood of the solution C, the simplest and most common choice is a structure in which the neighbourhoods have increasing cardinality: $|N_1(C)| < |N_2(C)| < \cdots < |N_{k_{\max}}(C)|$. The process of changing neighbourhoods when no improvement occurs diversifies the search. In particular, the choice of neighbourhoods of increasing cardinality yields a progressive diversification.

Although a VNS for the MLSteiner was implemented by Cerulli et al. (2006), our implementation is motivated by the successful VNS proposed for the MLST problem in Consoli et al. (2008a). The two approaches mainly differ in the implementation of the neighbourhood structures, in the way the initial solution is obtained, and in the maximum size of the shaking phase $k_{\max}$, among others. The VNS by Cerulli et al. (2006) uses three different neighbourhood structures (*k—Switch Neighbourhood, k—Covering Neighbourhood, k—Mixed Neighbourhood* (see Cerulli et al. 2006 for more details), in order to check whether one neighbourhood is better than another. For each neighbourhood, the procedure starts from an initial feasible solution provided by a greedy algorithm, and then tries to find an improved solution by selecting one of the considered neighbourhoods. After a specified number of iterations, another neighbourhood is chosen to be explored in subsequent iterations. For each neighbourhood, the parameter $k_{\max}$ varies during the execution, determined by $k_{\max} = \min(|C|, \frac{|L|}{4})$, where C is the current feasible solution and L is the set of labels. In contrast, our VNS implementation for the MLSteiner is specified as follows.

Before going into detail, consider the following notation. Given a labelled graph $G = (V, E, L)$, with n vertices, m edges, ℓ labels, and $Q \subseteq V$ basic nodes, each solution is encoded by a binary string, i.e. $C = (c_1, c_2, \ldots, c_\ell)$ where

$$c_i = \begin{cases} 1 & \text{if color } i \text{ is included in the solution } C \\ 0 & \text{otherwise} \end{cases} \qquad (\forall i = 1, \ldots, \ell).$$

Now, let us define the solution space, S, as the set of all the possible solutions, and let

$$\rho(C_1, C_2) = |C_1 - C_2| = \sum_{i=1}^{\ell} \lambda_i \tag{4}$$

define the Hamming distance between any two solutions C_1 and C_2, where $\lambda_i = 1$ if color i is included in one of the solutions but not in the other, and 0 otherwise, $\forall i = 1, \ldots, \ell$. The kth neighbourhood induced by (S, ρ), of a given solution C, may be defined as

$$N_k(C) = \{S \subset L : (\rho(C, S)) = k\} \quad (\forall k = 1, \ldots, k_{\max}). \tag{5}$$

The value of $k_{\max}$ represents the maximum size of the neighbourhood structure. It is an important parameter to tune in order to obtain an optimal balance between intensification and diversification capabilities. Choosing a small value for $k_{\max}$ produces a high intensification capability and a small diversification capability, resulting in a fast algorithm, but with a high probability of being trapped at a local minimum. Conversely, a large value for $k_{\max}$ decreases the intensification capability and increases the diversification capability, resulting in a slower algorithm, but able to escape from local minima. According to our experience, the value $k_{\max} = (|C| + |C|/3)$ gives a good trade-off between these two factors.

In order to construct the neighbourhood of a solution C, the algorithm starts by deleting colors from C. After all the colors are removed, additional colors are included at random in C, from the set of unused colors $(L - C)$, if a further expansion of the neighbourhood structure is required (case $k > |C|$). In case of infeasibility, that may be produced by the

```
Input: A labelled, undirected, connected graph $G = (V, E, L)$, with $n$ vertices, $m$ edges, $\ell$
        labels, and $Q \subseteq V$ basic nodes;
Output: A spanning tree $T$;
Initialization:
- Let $C \leftarrow 0$ be the global set of used colors;
- 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 $C'$ be a set of colors;
- 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 Steiner components of $C'$, i.e. the number of connected
components of the subgraph $(Q, E(C'))$;
begin
    $C$ = Generate-Initial-Solution-At-Random();
    repeat
        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
        end
    until termination conditions;
    Update $H = (V, E(C))$;
    $\Rightarrow$ Take any arbitrary spanning tree $T$ of $H = (V, E(C))$.
end

Function Shaking-Phase($N_k(C)$):
Set $C' \leftarrow C$;
for $i = 1$ to $k$ do
    if $i \le |C|$ then
        Select at random a color $c' \in C'$;
        Delete label $c'$ from the set of used colors: $C' \leftarrow C' - \{c'\}$;
    else
        Select at random an unused color $c'$, i.e., $c' \in (L - C)$;
        Add label $c'$ to the set of used colors: $C' \leftarrow C' \cup \{c'\}$;
    end
    Update $H' = (V, E(C'))$ and $Comp(C')$;
end
while $Comp(C') > 1$ do
    Select at random an unused color $u \in (L - C')$ that minimizes $Comp(C' \cup \{u\})$;
    Add label $u$ to the set of used colors: $C' \leftarrow C' \cup \{u\}$;
    Update $H' = (V, E(C'))$ and $Comp(C')$;
end
```

Algorithm 5: Variable Neighbourhood Search for the MLSteiner problem

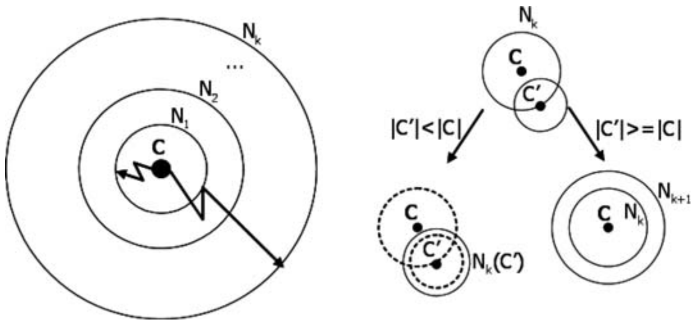

Fig. 3 Basic schema of Variable Neighbourhood Search

deletion of colors in the shaking phase, further colors may be added following the greedy criterion of adding the color with the minimum number of Steiner components at the specific step, in order to restore feasibility.

The main loop of this basic VNS is shown in Algorithm 5, and illustrated in Fig. 3. After defining the neighbourhood structure and obtaining the initial random solution C, the algorithm applies a shaking phase, letting parameter k vary throughout the execution. The shaking phase consists of the random selection of a solution C' in the neighbourhood $N_k(C)$ of the current solution C, with the intention of providing a better starting point for the successive local search phase. It represents the core idea of VNS, that of changing the neighbourhood structure when the local search is trapped at a local minimum.

The successive local search is not restricted to $N_k(C)$, but considers the entire solution space S. As in PM and GRASP, it tries to delete colors one by one from the current solution, whilst maintaining feasibility. At this stage, if no improvements are obtained ($|C'| \geq |C|$), the neighbourhood is increased ($k = k + 1$), resulting in a higher diversification of the search process. Otherwise, if $|C'| < |C|$, the algorithm moves to the improved solution ($C \leftarrow C'$), restarting the search with the smallest neighbourhood ($k = 1$). The algorithm proceeds until the established stopping conditions are reached.

2.6 Group-Swap Variable Neighbourhood Search

Several variants of VNS have been proposed in order to improve its performance in some circumstances (Hansen and Mladenović 2003). For example, Pérez-Pérez et al. (2007) proposed a hybridization between VNS and a path-relinking metaheuristic to solve the p-hub median problem, while Pacheco et al. (2007) mixed VNS and Tabu search for 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 for the MLSteiner problem, with a view to obtaining improved results. For this purpose, we use a Group-Swap VNS (GS-VNS), which is a variant of the one proposed for the MLST problem (Consoli et al. 2008b), in order to improve the diversification of the search process. The motivation for introducing a high diversification capability is to obtain a better performance in large problem instances.

The details of GS-VNS are specified in Algorithm 6. The algorithm starts from an initial feasible solution (C^{BEST}) generated at random. As in the previous algorithms, a local search procedure is applied which tries to delete colors one by one from the specific solution, whilst

```
Input: A labelled, undirected, connected graph $G = (V, E, L)$, with $n$ vertices, $m$ edges, $\ell$
       labels, and $Q \subseteq V$ basic nodes;
Output: A spanning tree $T$;
Initialization:
- Let $C^{BEST} \leftarrow 0$, $C \leftarrow 0$, and $C' \leftarrow 0$ be sets of colors, initially empties;
- Let $H^{BEST} = (V, E(C^{BEST}))$ be the subgraph of $G$ restricted to $V$ and edges with labels
in $C^{BEST}$, where $E(C^{BEST}) = \{e \in E : L(e) \in C^{BEST}\}$;
- 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 Steiner components of $C$, i.e., the number of connected
components of the subgraph $(Q, E(C))$;
- Let $COMPL \leftarrow (L - C^{BEST})$ the complementary space of the best solution $C^{BEST}$;
begin
    $C^{BEST}$ = Generate-Initial-Solution-At-Random();
    Local-Search($C^{BEST}$);
    repeat
        Perform the swapping of the best solution: $C$ = Group-Swap($C^{BEST}$);
        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 $|C| < |C^{BEST}|$ then Move $C^{BEST} \leftarrow C$;
    until termination conditions;
    Update $H^{BEST} = (V, E(C^{BEST}))$;
    $\Rightarrow$ Take any arbitrary spanning tree $T$ of $H^{BEST} = (V, E(C^{BEST}))$.
end

Function Group-Swap($C^{BEST}$):
Set $C \leftarrow 0$;
while $(Comp(C) > 1)$ AND $((COMPL - C) \neq 0)$ do
    Geometric Group-Swap cooling schedule for the temperature:
    $T_{GroupSwap}(|C| + 1) = \frac{T_{GroupSwap}(0)}{\alpha^{|C|}}$,   where $T_{GroupSwap}(0) = \alpha = |C^{BEST}|$;
    foreach $c \in (COMPL - C)$ do
        Calculate the probabilities $P(c)$, normalizing the values given by the Boltzmann
        function: $\exp\left(-\frac{Comp(C \cup \{c\}) - Comp_{\min}}{T_{GroupSwap}(|C|+1)}\right)$, where $Comp_{\min}$ is the minimum number
        of Steiner components at the specific step;
    end
    Select at random a color $u \in (COMPL - C)$ following the probabilities values $P(\cdot)$;
    Add label $u$ to the set of used colors: $C \leftarrow C \cup \{u\}$;
    Update $H = (V, E(C))$ and $Comp(C)$;
end
```

Algorithm 6: Group-Swap Variable Neighbourhood Search for the MLSteiner problem

maintaining feasibility. Then the Group-Swap (GS) operation is applied. It consists of extracting a solution from the complementary space of the current solution. Given the solution C^{BEST}, its complementary space ($COMPL$) is defined as the set of all the colors that are not contained in C^{BEST}, that is $(L - C^{BEST})$. To yield the solution, the Group-Swap applies a constructive heuristic to the subgraph of G with labels in ($COMPL$). In our implementation, we use the Probabilistic MVCA heuristic, already applied in (Consoli et al. 2008b). The Probabilistic MVCA uses an idea similar to the basic one of the Simulated Annealing metaheuristic (Aarts et al. 2005): the introduction of probabilities for the choice of the next colors to add to incomplete solutions. The introduction of this probabilistic element makes GS-VNS a hybridization between VNS and Simulated Annealing.

The Probabilistic MVCA begins from an initial solution, and successively selects a candidate move at random. 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. Consider a color x. The deterioration Δ of the objective function value is $(Comp(x) - Comp_{\min})$, where $Comp(x)$ represents the number of Steiner components obtained by inserting x in the partial solution, and $Comp_{\min}$ is the minimum number of Steiner components at the specific step. Thus, following the criteria of Simulated Annealing, the acceptance probability is computed according to the Boltzmann function as $\exp(-\Delta/T)$, using a temperature T as control parameter (Kirkpatrick et al. 1983). Probability values assigned to each color are inversely proportional to the number of Steiner components they give. The colors with a lower number of Steiner components will have a higher probability of being chosen. Conversely, colors with a higher number of Steiner components will have a lower probability of being chosen. Thus, the possibility of choosing less promising labels to be added to incomplete solutions is allowed, producing an improvement to the diversification of the search process.

The value of the parameter T is initially high, which allows many worse moves to be accepted, and is gradually reduced following a geometric cooling schedule:

$$T_{GroupSwap}(|C|+1) = \frac{T_{GroupSwap}(|C|)}{\alpha} = \frac{T_{GroupSwap}(0)}{\alpha^{|C|}}, \tag{6}$$

where experimentally we have found that $T_{GroupSwap}(0) = \alpha = |C^{BEST}|$ produces good results. This cooling schedule is very fast for the MLSteiner problem, yielding a good balance between intensification and diversification. At each step the probabilities of selecting colors giving a smaller number of Steiner components will be higher than the probabilities of selecting colors with a higher number of Steiner components. Furthermore, these differences in probabilities increase step by step as a result of the reduction of the temperature given by the cooling schedule. It means that the difference between the probabilities of two colors giving different numbers of Steiner components is higher as the algorithm proceeds.

The Group-Swap operation stops if either a feasible solution C is obtained, or the set of unused colors contained in the complementary space is empty, (i.e., $(COMPL - C) = 0$), producing a final infeasible solution. After the Group-Swap operation, a shaking phase is applied to the resulting solution, denoted by C. It 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 kth neighbourhood $N_k(C)$ as all the different sets of colors that are possible to obtain from C by removing k colors, where $k = 1, 2, \ldots, k_{\max}$. In a more formal way, the kth neighbourhood of a solution C 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 colors of the current feasible solution whose neighbourhood is being explored ($k_{\max} = |C|$). Since deletion of colors often gives an infeasible solution, additional colors may be added in order to restore feasibility. Addition of

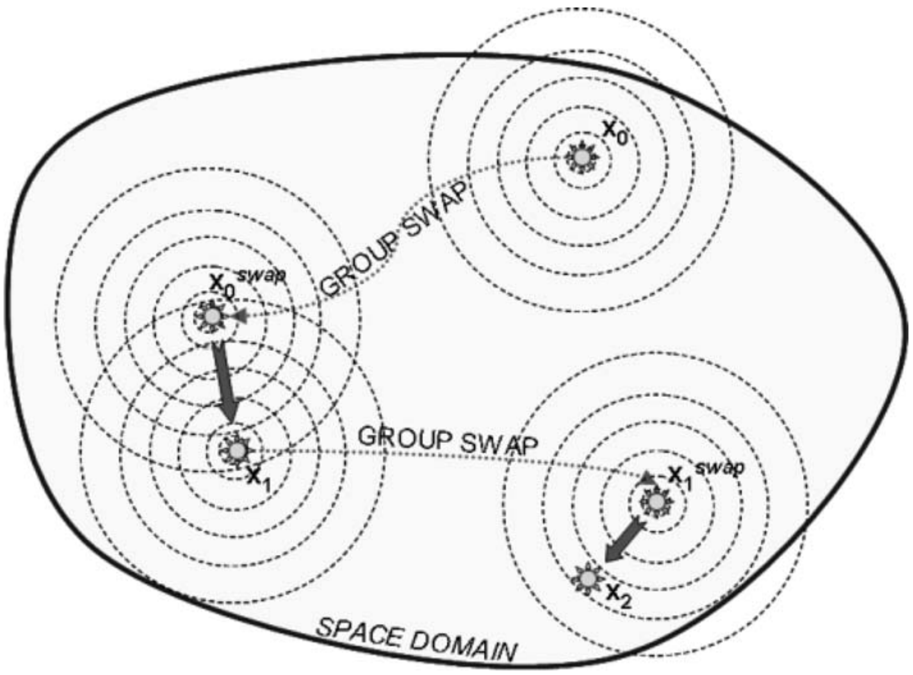

Fig. 4 Example illustrating the steps of Group-Swap VNS

colors at this step is according to the Probabilistic MVCA heuristic, i.e., the same non-deterministic heuristic used in the Group-Swap operation to yield a solution. Afterward, a local search is applied to the solution obtained from the shaking (C') and, if no improvements are obtained, i.e. if $|C'| \geq |C|$, the neighbourhood structure is increased ($k = k + 1$), yielding a progressive diversification ($|N_1(C)| < |N_2(C)| < \cdots < |N_{k_{\max}}(C)|$). Otherwise, the algorithm moves to the solution C' restarting the search with the smallest neighbourhood ($k = 1$). After the entire shaking phase, the Group-Swap operation is applied again to the actual best solution (C^{BEST}) and the algorithm continues iteratively with the same procedure until the user termination conditions are satisfied.

Figure 4 shows an example of GS-VNS. Given an initial random solution X_0, the algorithm searches for new solutions in its increasingly distant neighbourhoods. In this example, no better solutions are detected, and the current solution remains X_0. The Group-Swap procedure is applied to X_0, extracting a feasible solution, X_0^{swap}, from its complementary space ($L - X_0$). 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 reached. In the example, the final solution is denoted by X_2.

3 Computational results

To test the performance and the efficiency of the algorithms presented in this paper, we randomly generate instances of the MLSteiner problem with respect to the number of nodes (n), the density of the graph (d), the number of labels (ℓ), and the number of basic nodes (q).

In our experiments, we consider 48 different datasets, each one containing 10 instances of the problem (yielding a total of 480 instances), with $n = 100; 500$, $\ell = 0.25n; 0.5n; 1.25n$, and $q = 0.2n; 0.4n$. 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, q, and/or ℓ), and the reduction in the density of the graph. All the considered data are available from the authors in Consoli (2007).

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 experimentally with respect to the dimension of the problem instance. For the Discrete Particle Swarm Optimization, we consider a swarm of 100 particles and we use a variable number of iterations for each instance, determined such that the computations take approximately *max-CPU-time* for the specific dataset. Selection of the maximum allowed CPU time as the stopping criterion is made in order to have a direct comparison of the metaheuristics with respect to the quality of their solutions.

Our results are reported in Tables 1–4. The algorithms have been implemented using the C++ programming language. 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 values of q determine the different tables. The remaining columns give the computational results of the considered algorithms, identified with the abbreviations: EXACT (Exact method), PM (Pilot Method), GRASP (Greedy Randomized Adaptive Search Procedure), DPSO (Discrete Particle Swarm Optimization), VNS (Variable Neighbourhood Search), GS-VNS (Group-Swap Variable Neighbourhood Search).

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 average times at which the best solutions are obtained. For the exact method, a time limit of 3 hours is used. If an exact solution is not found within this time limit for any instance within a dataset, a not found status (NF) is reported. All the reported times have precision of ± 5 ms. For each dataset in the tables, the performance of an algorithm is 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. It is interesting to note that in all the problem instances for which the exact method obtains the solution, also VNS, GS-VNS, and DPSO yielded the exact solution.

Table 1 and Table 2 examine relatively small instances of the MLSteiner problem ($n = 100$, with $q = 0.2n$ and $q = 0.4n$, respectively). Looking at these tables, all the heuristics performed well for the considered problem instances. However, PM produces the worst solutions with respect to solution quality and computational running times. DPSO is slower than the remaining metaheuristics, as a result of a poor intensification capability and an excessive diversification capability for these instances, while GRASP is faster than DPSO, but it may produce slightly worse solutions with respect to solution quality, as in the instance $[n = 100, \ell = 125, d = 0.2]$ in Table 1. GRASP exhibits an opposite behavior to that of DPSO, being characterized by a limited diversification capability which sometimes does not allow the search process to escape from local optima. The performance of VNS and GS-VNS are both comparable for these small instances of the problem, obtaining the best solutions in terms of solution quality and running times with respect to the other algorithms. In particular, VNS is slightly faster than GS-VNS, and it can be considered the best performing heuristic for these problem instances.

Table 1 Computational results for $n = 100$ and $q = 0.2n$ (*max-CPU-time* for heuristics = 5000 ms)

Parameters			Average objective function values					
n	ℓ	d	EXACT	PM	GRASP	DPSO	VNS	GS-VNS
100	25	0.8	*1*	1	1	1	1	1
		0.5	*1.5*	1.5	1.5	1.5	1.5	1.5
		0.2	*2.1*	2.1	2.1	2.1	2.1	2.1
	50	0.8	*1.9*	1.9	1.9	1.9	1.9	1.9
		0.5	*2*	2	2	2	2	2
		0.2	*3.2*	3.2	3.2	3.2	3.2	3.2
	100	0.8	*2*	2	2	2	2	2
		0.5	*3*	3	3	3	3	3
		0.2	*4.6*	4.6	4.6	4.6	4.6	4.6
	125	0.8	*2.8*	2.8	2.8	2.8	2.8	2.8
		0.5	*3.3*	3.3	3.3	3.3	3.3	3.3
		0.2	*5.2*	5.4	5.3	5.2	5.2	5.2
TOTAL:			*32.6*	32.8	32.7	32.6	32.6	32.6

Parameters			Computational times (milliseconds)					
n	ℓ	d	EXACT	PM	GRASP	DPSO	VNS	GS-VNS
100	25	0.8	*14.7*	14.1	6.7	1.6	1.5	1.5
		0.5	*26.3*	20.3	6.3	3.2	4.7	4.8
		0.2	*16.2*	15.6	4.7	6.1	4.6	6.2
	50	0.8	*59.4*	56.1	9.4	6.4	1.6	7.9
		0.5	*66.3*	67.2	6.1	10.9	4.7	7.8
		0.2	*40.6*	75.1	15.6	15.7	1.5	9.5
	100	0.8	*306.3*	270.3	40.6	75.1	28.2	43.8
		0.5	*251.6*	275.1	7.6	31.2	7.3	12.6
		0.2	*0.9* $\cdot 10^3$	314.1	32.8	45.3	32.9	40.4
	125	0.8	*78.2*	381.2	14.1	48.4	15.3	32.8
		0.5	*451.5*	443.9	93.8	157.7	96.9	218.8
		0.2	*4.7* $\cdot 10^3$	518.8	68.8	322	136	162.4
TOTAL:			*6.9* $\cdot 10^3$	$2.5 \cdot 10^3$	306.5	723.6	335.2	548.5

Table 3 and Table 4 show larger instances of the problem ($n = 500$, with $q = 0.2n$ and $q = 0.4n$, respectively). The motivation to 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. VNS always obtains the solutions with the best quality, but it loses a lot, sometimes, in terms of computational running time with respect to GS-VNS (see, for example, the instances [$n = 500$, $\ell = 125$, $d = 0.2$] in Table 3, [$n = 500$, $\ell = 500$, $d = 0.2$] in Table 4, and [$n = 625$, $\ell = 500$, $d = 0.2$] in Table 4, among others). GS-VNS exhibits an optimal tuning between solution quality and computational running time for these larger problem instances, although sometimes slightly lacks in terms of intensification of the search space with respect to the VNS approach. As in the previous analysis, GRASP and DPSO show the same relative behavior for the considered instances: excessive diversification and poor intensification capabilities for DPSO and, conversely, excessive intensification

Table 2 Computational results for $n = 100$ and $q = 0.4n$ (*max-CPU-time* for heuristics = 6000 ms)

Parameters			Average objective function values					
n	ℓ	d	EXACT	PM	GRASP	DPSO	VNS	GS-VNS
100	25	0.8	*1*	1	1	1	1	1
		0.5	*1.9*	1.9	1.9	1.9	1.9	1.9
		0.2	*3*	3	3	3	3	3
	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	*4.3*	4.4	4.3	4.3	4.3	4.3
	100	0.8	*3*	3	3	3	3	3
		0.5	*3.6*	3.6	3.6	3.6	3.6	3.6
		0.2	*NF*	6.5	6.4	6.4	6.4	6.4
	125	0.8	*3*	3	3	3	3	3
		0.5	*4*	4	4	4	4	4
		0.2	*NF*	7	6.9	6.9	6.9	6.9
TOTAL:			–	41.6	41.3	41.3	41.3	41.3

Parameters			Computational times (milliseconds)					
n	ℓ	d	EXACT	PM	GRASP	DPSO	VNS	GS-VNS
100	25	0.8	*24.7*	15.6	6.3	9.3	1.6	4.6
		0.5	*29.7*	21.7	6.4	6.4	1.6	1.5
		0.2	*36.9*	29.8	3.2	23.6	3	9.3
	50	0.8	*60.9*	53	7.2	20.4	3.1	7.9
		0.5	*117.2*	76.6	15.1	34.3	17.2	23.4
		0.2	*314.1*	111	34.4	45.1	28.1	29.7
	100	0.8	*175*	260.9	10.9	39.2	9.4	17.4
		0.5	*389.1*	312.5	38.4	96.8	32.3	39.7
		0.2	*NF*	472	79.8	350	79.7	99.9
	125	0.8	*354.6*	440.7	18.7	57.6	23.4	20.3
		0.5	*479.6*	507.8	73.4	67.1	60.9	70.4
		0.2	*NF*	811	177.8	411	191.7	197
TOTAL:			–	$3.1 \cdot 10^3$	471.6	$1.2 \cdot 10^3$	459.8	521.1

and poor diversification capabilities for GRASP. PM confirms the worst performance for all the problem instances in terms of solution quality and computational running time.

In addition, to confirm this evaluation, the algorithms are ranked for each dataset, assigning a rank of 1 to the best performing algorithm, rank 2 to the second best one, and so on. Obviously, if the exact method records a NF for a dataset, the worst rank is assigned to it 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.

According to the ranking, VNS is the best performing algorithm, followed respectively by GS-VNS, GRASP, DPSO, PM, and EXACT. To analyze the statistical significance of differences between these ranks, we follow the same procedure proposed in Consoli et al. (2008a) which make use of the *Friedman Test* (Friedman 1940) and its corresponding *Ne-*

Table 3 Computational results for $n = 500$ and $q = 0.2n$ (*max-CPU-time* for heuristics $= 500 \cdot 10^3$ ms)

Parameters			Average objective function values					
n	ℓ	d	EXACT	PM	GRASP	DPSO	VNS	GS-VNS
500	125	0.8	*1.1*	1.1	1.1	1.1	1.1	1.1
		0.5	*2*	2	2	2	2	2
		0.2	*3*	3	3	3	3	3
	250	0.8	*2*	2	2	2	2	2
		0.5	*2.9*	2.9	2.9	2.9	2.9	2.9
		0.2	*NF*	4.4	4.3	4.3	4.3	4.3
	500	0.8	*3*	3	3	3	3	3
		0.5	*NF*	3.9	3.9	4	3.9	3.9
		0.2	*NF*	6.8	6.8	6.9	6.7	6.7
	625	0.8	*NF*	3.8	3.8	3.8	3.8	3.8
		0.5	*NF*	4.8	4.8	4.8	4.7	4.7
		0.2	*NF*	8	8	7.9	7.9	8
TOTAL:			–	45.7	45.6	45.7	45.3	45.4
Parameters			**Computational times (milliseconds)**					
n	ℓ	d	EXACT	PM	GRASP	DPSO	VNS	GS-VNS
500	125	0.8	$\mathit{1.5} \cdot 10^3$	$1.2 \cdot 10^3$	173.4	$3.4 \cdot 10^3$	172.2	404.7
		0.5	$\mathit{2.1} \cdot 10^3$	$2.5 \cdot 10^3$	149.8	575	26.5	104.8
		0.2	$\mathit{4.1} \cdot 10^3$	$7.1 \cdot 10^3$	318.8	$5.9 \cdot 10^3$	265.7	634.4
	250	0.8	$\mathit{13.6} \cdot 10^3$	$17.4 \cdot 10^3$	270	$9.7 \cdot 10^3$	115.6	859.4
		0.5	$\mathit{37.3} \cdot 10^3$	$46.8 \cdot 10^3$	334.6	$8.8 \cdot 10^3$	148.4	301.6
		0.2	*NF*	$48.1 \cdot 10^3$	$14.5 \cdot 10^3$	$36.7 \cdot 10^3$	$11.9 \cdot 10^3$	$17 \cdot 10^3$
	500	0.8	$\mathit{300.8} \cdot 10^3$	$304.4 \cdot 10^3$	$2.3 \cdot 10^3$	$22.1 \cdot 10^3$	$1.8 \cdot 10^3$	$1.9 \cdot 10^3$
		0.5	*NF*	$325.8 \cdot 10^3$	$109.7 \cdot 10^3$	$106.5 \cdot 10^3$	$85.7 \cdot 10^3$	$388.6 \cdot 10^3$
		0.2	*NF*	$425.2 \cdot 10^3$	$17.9 \cdot 10^3$	$170.4 \cdot 10^3$	$27.7 \cdot 10^3$	$29 \cdot 10^3$
	625	0.8	*NF*	$465.6 \cdot 10^3$	$36.9 \cdot 10^3$	$180.2 \cdot 10^3$	$32.8 \cdot 10^3$	$51.9 \cdot 10^3$
		0.5	*NF*	$403 \cdot 10^3$	$2.5 \cdot 10^3$	$110.4 \cdot 10^3$	$6.7 \cdot 10^3$	$9.4 \cdot 10^3$
		0.2	*NF*	$399.3 \cdot 10^3$	$36.7 \cdot 10^3$	$285.7 \cdot 10^3$	$79.5 \cdot 10^3$	$36.2 \cdot 10^3$
TOTAL:			–	$2446.4 \cdot 10^3$	$221.8 \cdot 10^3$	$940.4 \cdot 10^3$	$246.8 \cdot 10^3$	$536.3 \cdot 10^3$

menyi Post-hoc Test (Nemenyi 1963). For more details on the issue of statistical tests for comparison of algorithms over multiple datasets, see Demśar (2006) and Hollander and Wolfe (1999).

According to the Friedman test, a significant difference between the performance of the metaheuristics, with respect to the evaluated ranks, exists (at the 5% 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. In our case, considering a significance level of the Nemenyi test of 5%, this critical difference is 1.09. The differences between the average ranks of the algorithms are reported in Table 6.

Table 4 Computational results for $n = 500$ and $q = 0.4n$ (*max-CPU-time* for heuristics $= 600 \cdot 10^3$ ms)

Parameters			Average objective function values					
n	ℓ	d	EXACT	PM	GRASP	DPSO	VNS	GS-VNS
500	125	0.8	*1.9*	1.9	1.9	1.9	1.9	1.9
		0.5	*2*	2	2	2	2	2
		0.2	*NF*	4.1	4.1	4.1	4.1	4.1
	250	0.8	*2*	2	2	2	2	2
		0.5	*3*	3	3	3	3	3
		0.2	*NF*	6.2	6.1	6.3	6.1	6.1
	500	0.8	*NF*	3.7	3.7	3.7	3.7	3.7
		0.5	*NF*	5	5	5	5	5
		0.2	*NF*	9.9	9.9	9.9	9.8	9.8
	625	0.8	*NF*	4	4	4	4	4
		0.5	*NF*	5.8	5.8	5.7	5.7	5.7
		0.2	*NF*	11.5	11.5	11.4	11.2	11.3
TOTAL:			–	59.1	59	59	58.5	58.6
Parameters			Computational times (milliseconds)					
n	ℓ	d	EXACT	PM	GRASP	DPSO	VNS	GS-VNS
500	125	0.8	*218.8*	$1.1 \cdot 10^3$	231	778.2	187.5	93.9
		0.5	*2.8* $\cdot 10^3$	$2.6 \cdot 10^3$	230	$4.3 \cdot 10^3$	184.2	218.7
		0.2	*NF*	$8.3 \cdot 10^3$	$1.1 \cdot 10^3$	$8.8 \cdot 10^3$	853	$3.3 \cdot 10^3$
	250	0.8	*44.6* $\cdot 10^3$	$20.2 \cdot 10^3$	615.7	$12.5 \cdot 10^3$	393.7	$1.2 \cdot 10^3$
		0.5	*48.8* $\cdot 10^3$	$49.8 \cdot 10^3$	864.2	$13.4 \cdot 10^3$	650	$3.1 \cdot 10^3$
		0.2	*NF*	$48.7 \cdot 10^3$	$20.4 \cdot 10^3$	$122.2 \cdot 10^3$	$38.1 \cdot 10^3$	$24.8 \cdot 10^3$
	500	0.8	*NF*	$201.1 \cdot 10^3$	$13.1 \cdot 10^3$	$19.4 \cdot 10^3$	$12.1 \cdot 10^3$	$13.7 \cdot 10^3$
		0.5	*NF*	$193.1 \cdot 10^3$	$5.5 \cdot 10^3$	$19.6 \cdot 10^3$	$4.9 \cdot 10^3$	$5 \cdot 10^3$
		0.2	*NF*	$579.7 \cdot 10^3$	$75.9 \cdot 10^3$	$195.3 \cdot 10^3$	$258.4 \cdot 10^3$	$133.3 \cdot 10^3$
	625	0.8	*NF*	$384 \cdot 10^3$	$6.9 \cdot 10^3$	$18.5 \cdot 10^3$	$6.2 \cdot 10^3$	$6.5 \cdot 10^3$
		0.5	*NF*	$421.2 \cdot 10^3$	$50.5 \cdot 10^3$	$32.6 \cdot 10^3$	$321.5 \cdot 10^3$	$12.7 \cdot 10^3$
		0.2	*NF*	$397.9 \cdot 10^3$	$95.4 \cdot 10^3$	$232.1 \cdot 10^3$	$115.9 \cdot 10^3$	$68.6 \cdot 10^3$
TOTAL:			–	$2307.7 \cdot 10^3$	$270.7 \cdot 10^3$	$679.5 \cdot 10^3$	$739.3 \cdot 10^3$	$272.5 \cdot 10^3$

Table 5 Average ranks of the algorithms (ordering from the best one to the worst one)

VNS	GS-VNS	GRASP	DPSO	PM	EXACT
1.38	2.48	2.56	3.88	5.21	5.49

From this table, it is possible to identify four groups of algorithms with different performance. The best performing group consists of just VNS, because it obtains the smallest rank which is significantly different from all the other ranks. The remaining groups are, in order, GS-VNS and GRASP, then DPSO, and finally PM and EXACT. From this further analysis, the results reinforce the conclusion that VNS is an effective metaheuristic for the MLSteiner problem. Although a VNS for the MLSteiner, along with other heuristic approaches, was

Table 6 Pairwise differences of the average ranks of the algorithms (Critical difference = 1.09 for a significance level of 5% for the Nemenyi test)

	VNS	GS-VNS	GRASP	DPSO	PM	EXACT
VNS	–	**1.1**	**1.18**	**2.5**	**3.83**	**4.11**
GS-VNS	–	–	0.08	**1.4**	**2.73**	**3.01**
GRASP	–	–	–	**1.32**	**2.65**	**2.93**
DPSO	–	–	–	–	**1.33**	**1.61**
PM	–	–	–	–	–	0.28
EXACT	–	–	–	–	–	–

implemented by Cerulli et al. (2006), it has been shown that our VNS implementation is fast, simple, and particularly effective for the MLSteiner problem, obtaining high-quality solutions in short computational running times. The superiority of Variable Neighbourhood Search with respect to the other algorithms is further evidenced by its ease implementation and simplicity.

4 Conclusions

In this paper, we considered the minimum labelling Steiner tree (MLSteiner) problem, an extension of the minimum labelling spanning tree problem to the case where only a subset of specified nodes, the basic nodes, need to be connected. The MLSteiner problem is NP-hard and, therefore, heuristics and approximate solution approaches with performance guarantees are of interest.

We presented some metaheuristics for the problem: a Greedy Randomized Adaptive Search Procedure (GRASP), a Discrete Particle Swarm Optimization (DPSO), a Variable Neighbourhood Search (VNS), and a VNS-based version that we have called Group-Swap Variable Neighbourhood Search (GS-VNS). Considering a wide range of problem instances, we compared these metaheuristics with the Pilot Method (PM) by Cerulli et al. (2006), the most popular MLSteiner heuristic in the literature. Based on this experimental analysis, all the proposed procedures clearly outperformed PM and, in particular, the best performance was obtained by VNS.

Although a VNS for the MLSteiner, along with other heuristic approaches, was implemented by Cerulli et al. (2006), this paper shows that the VNS proposed here is fast, simple, and particularly effective for the MLSteiner problem, obtaining high-quality solutions in short computational running times. This analysis provides further evidence of the ability of Variable Neighbourhood Search to deal with NP-hard combinatorial problems.

References

Aarts, E., Korst, J., & Michiels, W. (2005). Simulated annealing. In E. K. Burke & G. Kendall (Eds.), *Search methodologies: introductory tutorials in optimization and decision support techniques* (pp. 187–210). Berlin: Springer.

Avis, D., Hertz, A., & Marcotte, O. (2005). *Graph theory and combinatorial optimization*. New York: Springer.

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

Cerulli, R., Fink, A., Gentili, M., & Voß, S. (2005). 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* (pp. 93–106). New York: Springer.

Cerulli, R., Fink, A., Gentili, M., & Voß, S. (2006). Extensions of the minimum labelling spanning tree problem. *Journal of Telecommunications and Information Technology*, *4*, 39–45.

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

Consoli, S. (2007). Test datasets for the minimum labelling Steiner tree problem. [Online], available at http://people.brunel.ac.uk/~mapgssc/MLSteiner.htm.

Consoli, S., Darby-Dowman, K., Mladenović, N., & Moreno-Pérez, J. A. (2008a). Greedy randomized adaptive search and variable neighbourhood search for the minimum labelling spanning tree problem. *European Journal of Operational Research*. doi:10.1016/j.ejor.2008.03.014.

Consoli, S., Darby-Dowman, K., Mladenović, N., & Moreno-Pérez, J. A. (2008b). *Heuristics based on greedy randomized adaptive search and variable neighbourhood search for the minimum labelling spanning tree problem*. Technical Report TR/01/07, Brunel University, West London, UK. Available: http://hdl.handle.net/2438/737.

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

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

Feo, T. A., & Resende, M. G. C. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, *8*, 67–71.

Francis, R. L., McGinnis, L. F., & White, J. A. (1992). *Facility layout and location: an analytical approach*. Englewood Cliffs: Prentice-Hall.

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

Garey, M. R., Graham, R. L., & Johnson, D. S. (1977). The complexity of computing Steiner minimal trees. *SIAM Journal on Applied Mathematics*, *32*, 835–859.

Grimwood, G. R. (1994). *The Euclidean Steiner tree problem: simulated annealing and other heuristics*. Master's thesis, Victoria University, Wellington, New Zealand, URL http://www.isor.vuw.ac.nz/~geoff/thesis.html.

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

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

Hollander, M., & Wolfe, D. A. (1999). *Nonparametric statistical methods* (2nd edn.). New York: Wiley.

Hwang, F. K., Richards, D. S., & Winter, P. (1992). *The Steiner tree problem*. Amsterdam: North-Holland.

Karp, R. M. (1975). On the computational complexity of combinatorial problems. *Networks*, *5*, 45–68.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of the 4th IEEE international conference on neural networks*, Perth, Australia (pp. 1942–1948).

Kennedy, J., & Eberhart, R. (1997). A discrete binary version of the particle swarm algorithm. In *IEEE conference on systems, man, and cybernetics* (Vol. 5, pp. 4104–4108).

Kennedy, J., & Eberhart, R. (2001). *Swarm intelligence*. San Francisco: Morgan Kaufmann.

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

Korte, B., Prömel, H. J., & Steger, A. (1990). Steiner trees in VLSI-layout. In B. Korte, L. Lovász, H. J. Prömel, & A. Schrijver (Eds.), *Paths, flows, and VLSI-layout* (pp. 185–214). Berlin: Springer.

Krarup, J., & Vajda, S. (1997). On Torricelli's geometrical solution to a problem of Fermat. IMA. *Journal of Mathematics Applied in Business and Industry*, *8*(3), 215–224.

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

Miehle, W. (1958). Link-minimization in networks. *Operations Research*, *6*, 232–243.

Moreno-Pérez, J. A., Castro-Gutiérrez, J. P., Martínez-García, F. J., Melián, B., Moreno-Vega, J. M., & Ramos, J. (2007). Discrete particle swarm optimization for the p-median problem. In *Proceedings of the 7th metaheuristics international conference*, Montréal, Canada.

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

Pacheco, J., Casado, S., & Nuñez, L. (2007). Use of VNS and TS in classification: variable selection and determination of the linear discrimination function coefficients. *IMA Journal of Management Mathematics*, *18*(2), 191–206.

Pitsoulis, L. S., & Resende, M. G. C. (2002). Greedy randomized adaptive search procedure. In P. Pardalos & M. G. C. Resende (Eds.), *Handbook of applied optimization* (pp. 168–183). Oxford: Oxford University Press.

Pérez-Pérez, M., Almeida-Rodríguez, F., & Moreno-Vega, J. M. (2007). A hybrid VNS-path relinking for the p-hub median problem. *IMA Journal of Management Mathematics*, *18*(2), 157–171.

Raghavan, S., & Anandalingam, G. (2003). *Telecommunications network design and management*. New York: Springer.

Resende, M. G. C., & Ribeiro, C. C. (2003). Greedy randomized adaptive search procedure. In F. Glover & G. Kochenberger (Eds.), *Handbook in metaheuristics* (pp. 219–249). Dordrecht: Kluwer.

Tanenbaum, A. S. (1989). *Computer networks*. Englewood Cliffs: Prentice-Hall.

Van-Nes, R. (2002). *Design of multimodal transport networks: a hierarchical approach*. Delft: Delft University Press.

Voß, S. (2000). Modern heuristic search methods for the Steiner tree problem in graphs. In D. Z. Du, J. M. Smith, & J. H. Rubinstein (Eds.), *Advances in Steiner tree* (pp. 283–323). Boston: Kluwer.

Voß, S. (2006). Steiner tree problems in telecommunications. In M. Resende & P. Pardalos (Eds.), *Handbook of optimization in telecommunications* (pp. 459–492). New York: Springer. Chap 18.

Voß, S., Martello, S., Osman, I. H., & Roucairol, C. (1999). *Meta-heuristics. Advanced and trends local search paradigms for optimization*. Norwell: Kluwer.

Voß, S., Fink, A., & Duin, C. (2004). Looking ahead with the Pilot method. *Annals of Operations Research*, *136*, 285–302.

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

Winter, P. (1987). Steiner problem in networks: a survey. *Networks*, *17*, 129–167.

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

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

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