Nat Comput (2010) 9:29–46
DOI 10.1007/s11047-009-9137-9

Discrete Particle Swarm Optimization for the minimum labelling Steiner tree problem

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

Published online: 17 June 2009

Abstract Particle Swarm Optimization is a population-based method inspired by the social behaviour of individuals inside swarms in nature. Solutions of the problem are modelled as members of the swarm which fly in the solution space. The improvement of the swarm is obtained from the continuous movement of the particles that constitute the swarm submitted to the effect of inertia and the attraction of the members who lead the swarm. This work focuses on a recent Discrete Particle Swarm Optimization for combinatorial optimization, called Jumping Particle Swarm Optimization. Its effectiveness is illustrated on the minimum labelling Steiner tree problem: given an undirected labelled connected graph, the aim is to find a spanning tree covering a given subset of nodes, whose edges have the smallest number of distinct labels.

Keywords Combinatorial optimization · Discrete Particle Swarm Optimization · Heuristics · Minimum labelling Steiner tree problem · Graphs and networks

1 Introduction

Biological and natural processes have always been a source of inspiration for computer science and information technology in many real-world applications (Holland 1992). It is well known that biological entities, from single cell organisms—like bacteria—to humans,

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

K. Darby-Dowman
e-mail: kenneth.darby-dowman@brunel.ac.uk

N. Mladenović
e-mail: nenad.mladenovic@brunel.ac.uk

J. A. Moreno-Pérez
DEIOC, IUDR, Universidad de La Laguna, 38271 Santa Cruz de Tenerife, Spain
e-mail: jamoreno@ull.es

often engage in a rich repertoire of social interaction that could range from altruistic cooperation to open conflict. One specific kind of social interaction is cooperative problem solving, where a group of autonomous entities work together to achieve a certain goal (Holland 1992). Over the years, mathematical strategies influenced by nature and natural systems for the solution of complex problems have been widely used as robust techniques for solving hard combinatorial optimization problems. Their behaviour is directed by the natural evolution of a population in the search for an optimum. These strategies are referred to as *nature-inspired algorithms*.

Particle Swarm Optimization (PSO) is a nature-inspired algorithm first proposed by Kennedy and Eberhart (1995). It has been applied with success in many areas and appears to be a suitable approach for several optimization problems (Kennedy and Eberhart 2001). Similarly to Genetic Algorithms, PSO is a population-based technique, inspired by the social behaviour of individuals (or particles) inside swarms in nature (for example, flocks of birds or schools of fish). Since the original PSO was 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 examine the effectiveness of DPSO for the minimum labelling Steiner tree (MLSteiner) problem.

1.1 Discrete Particle Swarm Optimization

The standard PSO considers a swarm S containing n particles ($S = 1,2,\ldots,n$) in a d-dimensional continuous solution space (Kennedy and Eberhart 1995, 2001). Each i-th particle of the swarm has a position $x_i = (x_{i1}, x_{i2},\ldots,x_{ij},\ldots,x_{id})$, and a velocity $v_i = (v_{i1}, v_{i2},\ldots,v_{ij},\ldots,v_{id})$. The position x_i represents a solution to the problem, while the velocity v_i gives the rate of change for the position of particle i at the next iteration. Indeed, considering iteration k, the position of particle i is adjusted according to $x_i^k = x_i^{k\text{-}1} + v_i^k$.

Each particle i of the swarm communicates with a social environment or neighbourhood, $N(i) \subseteq S$, representing the group of particles with which it communicates, and which could change dynamically. 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, considering iteration k of the PSO algorithm, each particle i updates its velocity reflecting the attractiveness of its best position so far (b_i) and the best position (g_i) of its social neighbourhood $N(i)$, according to the equation (Kennedy and Eberhart 1995, 2001):

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

The parameters c_i are positive constant weights representing the degrees of confidence of particle i in the different positions that influence its dynamics, while the term ξ refers to a random number with uniform distribution [0,1] that is independently generated at each iteration.

The original PSO algorithm can only optimize problems in which the elements of the solution are continuous real numbers since, in words of the inventors of PSO, it is not possible to "throw to fly" particles in a discrete space (Kennedy and Eberhart 1995). In the last years, several modifications of the PSO algorithm for solving problems with discrete variables have been proposed in the literature. They are referred to as Discrete Particle Swarm Optimization (DPSO) methods.

Kennedy and Eberhart (1997) developed a DPSO algorithm for problems with binary-valued solution elements where the position of each particle is a vector

$x_i = (x_{i1}, x_{i2}, \ldots, x_{ij}, \ldots, x_{id})$ of the d-dimensional binary solution space, $x_i \in \{0, 1\}^d$, but the velocity is still a vector $v_i = (v_{i1}, v_{i2}, \ldots, v_{ij}, \ldots, v_{id})$ of the d-dimensional continuous space, $v_i \in \Re^d$. As with the standard PSO strategy by Kennedy and Eberhart (1995), the velocity is still updated by means of Eq. 1. However the significance of the velocity term has been changed to indicate the probability of the corresponding solution element assuming a value of 0 or 1. In other words, the continuous value v_{ij} refers to the probability that the j-th binary variable within the position of the i-th particle, x_{ij}, assumes a value of 0 or 1 at the next iteration. For assigning a new position value to a particle i, each position variable x_{ij} is randomly set with probability of selecting a value of 1 given by the sigmoid function:

$$\frac{1}{1 + \exp(-v_{ij})}. \tag{2}$$

In this variation of the PSO algorithm, the velocity term is limited to $|v_{ij}| < V_{\max}$, where $V_{\max}$ is some value typically close to 6.0. This setting prevents the probability of the particle element assuming either a value of 0 or 1 from being too high. Though this DPSO has been shown to be capable of optimizing several combinatorial problems (Kennedy and Eberhart 1997), it is limited only to discrete problems with binary-valued solution elements.

Other PSO techniques for discrete optimization include the work of Al-kazemi and Mohan (2002) who developed a method, based on the DPSO proposed by Kennedy and Eberhart (1997), whose particles are influenced alternatively by their own best position and the best position among their neighbours. In this DPSO strategy, the velocity is updated as in the standard PSO by means of Eq. 1, with the difference that the coefficients c_i assume only the values 1 and -1, i.e. $c_i \in \{-1, 1\}$, and only a given number of combinations of the coefficients are possible. The different combinations of the coefficients are referred to as phases of the particles, and determine the directions of movement of the particles. At any given time, each particle is in one of the possible phases, and the next phase to select is determined by means of the previous phase of the particle and the number of iterations executed so far. The smallest possible non-trivial number of phases, used in the DPSO method by Al-kazemi and Mohan (2002), consists of two phases. In the first phase, each i-th particle uses coefficients (1, -1, 1), by directing the particle movement toward g_i, that is the best position of its social neighbourhood $N(i)$. Instead, in the second phase, each i-th particle uses coefficients (1, 1, -1), by directing the particle movement toward its own best position b_i. Phase change occurs if no improvement of the best solution to date is obtained within a given number of iterations (typically 10 iterations) in the current phase.

A similar strategy was developed by Yang et al. (2004) by considering a larger number of combinations of the coefficients, referred to as quantum states of the particles, and a slightly different update equation for the velocity inspired by the principles of quantum computing. Both the methods developed in Al-kazemi and Mohan (2002) and in Yang et al. (2004) use the same principles as the DPSO by Kennedy and Eberhart (1997) and both are limited to discrete problems with binary-valued variables. A non-binary version of the DPSO by Kennedy and Eberhart (1997) was presented in Secrest (2001) and applied to the Travelling Salesman Problem. In this DPSO algorithm, the particles in the swarm were represented as linked lists of cities and genetic operators, such as mutation and recombination, were adopted to induce the move of the swarm.

Pampara et al. (2005) developed an indirect DPSO method by reducing a binary problem into a continuous trigonometric function having only four parameters to optimize. This reduction is obtained by means of Angle Modulation, a popular technique used in the

field of signal processing from telecommunications (Pampara et al. 2005). The standard PSO algorithm by Kennedy and Eberhart (1995, 2001) is then applied to optimise the four parameters of the continuous trigonometric function. Successively the function is sampled at even intervals by producing a continuous value for each interval. If a value is positive, the corresponding bit value assigned to that interval assumes value 1, otherwise the corresponding bit value assumes value 0. The set of all generated bit values associated with the intervals represents the binary solution vector to the original binary problem. The benefit of this technique is that a larger dimensional binary space can be represented by a smaller 4-dimensional continuous space, by allowing a faster convergence of the optimization phase with respect to the other binary PSO methods in the literature. However, in some circumstances, the reduction process may be too costly for some complex binary problems in terms of computational running time and, therefore, the overall benefits of the method are nullified. Furthermore, the method is limited to considering only discrete problems with binary-valued solution elements.

The multi-valued PSO (MVPSO) proposed by Pugh and Martinoli (2006) deals with variables with multiple discrete values. While in the case of a continuous PSO the position of each particle is a mono-dimensional array, and in the case of a DPSO is a 2-dimensional array, in a MVPSO algorithm it is expressed by means of a 3-dimensional array x_{ijk}, representing the probability that the i-th particle, in the j-th iteration, assumes the k-th value. To evaluate the fitness of a particle, the solution elements x_{ijk} are generated probabilistically following a sigmoid distribution, by making the evaluation process inherently stochastic. Because particle terms are real-valued, this representation allows the velocity to be used in the same way as in the standard PSO by Kennedy and Eberhart (1995, 2001), where v_{ijk} represents the velocity of x_{ijk}. Therefore, it is still possible to update the velocity v_{ijk} by means of Eq. 1.

Another DPSO algorithm was developed by Correa et al. (2006) to tackle the data mining task of attribute selection, in order to classify data sets into classes or categories of the same type. The objective of attribute selection is to simplify a data set by reducing its dimensionality and redundancy in the information provided by the selected attributes, and by identifying relevant underlying attributes without sacrificing predictive accuracy. The attribute selection problem is a combinatorial problem, where each attribute is identified by a unique positive integer number. The DPSO by Correa et al. (2006) differs from other traditional PSO algorithms because its swarm contains particles representing combinations of selected attributes of different size, from 1 to the total number of attributes, λ. Every particle is associated with a velocity vector of cardinality 1-by-λ containing positive numbers, each one representing the proportional likelihood of the corresponding attribute be selected. The updating process for the velocity of a particle is based on a procedure analogous to the standard PSO by Kennedy and Eberhart (1995, 2001), by using the concepts of best position of the particle and best position among its neighbours and by applying Eq. 1. In order to obtain the new position of a particle from the corresponding updated velocity vector, each component of the velocity is multiplied by a random number ξ uniformly distributed in [0,1]. The new length of the particle is determined by selecting another random number k smaller than the total number of attributes, i.e. $k \in [0, \lambda]$, and finally the k attributes with the largest likelihood in the velocity vector are selected to compose the new position of the particle. Although the efficiency of this DPSO algorithm has been proved for attribute selection problems, its complex implementation makes it difficult to apply this methodology to other discrete problems.

Moraglio et al. (2008) showed a close connection between Particle Swarm Optimization and Evolutionary Algorithms by using a geometric framework for the interpretation of the

crossover operator. The advantage of their PSO algorithm, referred to as Geometric Particle Swarm Optimization (GPSO), is that it can be derived rigorously for any combinatorial space. Firstly, Moraglio et al. (2008) derived their GPSO for Euclidean, Manhattan, and Hamming spaces and discussed how to derive a GPSO for virtually any representation in a similar way. They tested the GPSO theory experimentally by reporting extensive experimental results of each GPSO algorithm implemented. In particular, they showed how to apply their GPSO to solve efficiently the Sudoku puzzle, which is a nontrivial constrained combinatorial problem.

A new DPSO proposed in Moreno-Pérez et al. (2007) and Martínez-García and Moreno-Pérez (2008) does not consider any velocity since, from the lack of continuity of the movement in a discrete space, the notion of velocity has less meaning; however they kept the attraction of the best positions. They interpret the weights of the updating equation as probabilities that, at each iteration, each particle has a random behaviour, or acts in a way guided by the effect of an attraction. The moves in a discrete or combinatorial space are jumps from one solution to another. The attraction causes the given particle to move towards this attractor if it results in an improved solution. An inspiration from nature for this process is found in frogs, which jump from a lily pad to a pad in a pool. Thus, this new discrete PSO is called *Jumping Particle Swarm Optimization* (JPSO). This methodology has been recently applied with success to the vehicle routing problem with time windows (Castro-Gutiérrez et al. 2008) and to the Steiner tree in graph and delay-constrained routing problems (Qu et al. 2009). This paper extends and more rigorously tests the Jumping Particle Swarm Optimization algorithm for the minimum labelling Steiner tree problem first introduced in Consoli et al. (2008b). We compare this strategy with other algorithms used to solve the problem considered, and we show that JPSO is able to obtain good approximate solutions for large instances of the problem. The effectiveness of this approach and its superiority with respect to the other methods is further confirmed by means of a rigorous statistical analysis of the results.

1.2 The minimum labelling Steiner tree problem

The minimum labelling Steiner tree (MLSteiner) problem is an NP-hard graph problem introduced by Cerulli et al. (2006) and defined as follows. 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 (not necessarily properly) on the set L of labels (or colours). Let $Q \subseteq V$ be a set of nodes that must be connected (basic vertices or nodes). The MLSteiner problem searches for an acyclic connected subgraph $T \subseteq G$, spanning all basic nodes Q and using the minimum number of different colours.

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

This problem has many real-world applications. 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). The multimodal transportation network is represented by a graph where each edge is assigned a colour, 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

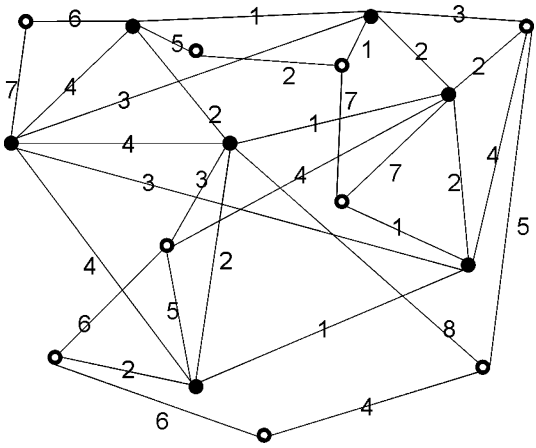

Fig. 1 Example of a labelled connected undirected graph, input of the MLSteiner problem. The solid vertices represent the basic nodes to cover

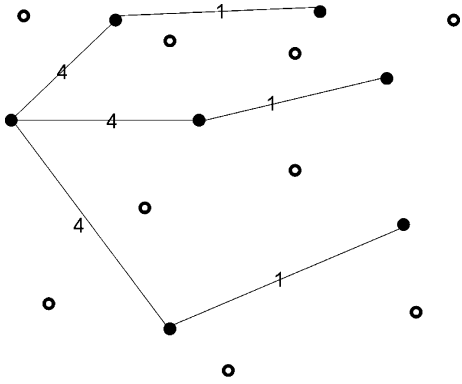

Fig. 2 Minimum labelling Steiner tree solution for the graph of Fig. 1

locations, without cycles, using the minimum number of companies, in order to minimise cost.

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 colours $C \subseteq L$, such that all the edges with labels in C represent a connected subgraph of G and span all the basic nodes Q. If C is a feasible solution, then any spanning tree of C has at most $|C|$ labels. Thus, in order to solve the MLSteiner problem we seek a feasible solution with the smallest number of colours (Cerulli et al. 2006).

The MLSteiner problem is an extension of the well-known Steiner tree problem, and of the minimum labelling spanning tree problem. Given a graph with positive-weighted edges, the Steiner tree (Steiner) problem searches a minimum-weight tree spanning a subset of nodes, called basic nodes (or terminals) (Garey et al. 1977). Several heuristics for the Steiner problem in graphs are reported in Voß (2000). The minimum labelling spanning tree (MLST) problem is used where, given a graph with coloured edges, one seeks a spanning tree with the least number of colours (Chang and Leu 1997; Krumke and Wirth 1998). Several heuristics have been proposed in Cerulli et al. (2005), Xiong et al. (2006) and Consoli et al. (2008a). The MLSteiner problem was considered by Cerulli et al. (2006) where their Pilot Method (PM) was compared with some other metaheuristics: Tabu

Search, Simulated Annealing, and Variable Neighbourhood Search. From their analysis, PM was shown to be an effective heuristic for this problem.

The structure of the paper is as follows. In Sect. 2 the details of Pilot Method are presented, along with an exact approach, a basic Multi-Start (MS) metaheuristic (with and without an embedded local search), and the Jumping Particle Swarm Optimization (JPSO). Section 3 shows the experimental comparison of these algorithms and a rigorous statistical analysis of their results by means of the *Friedman Test* (Friedman 1940) and the *Nemenyi Post-hoc Test* (Nemenyi 1963). It is shown that JPSO is able to obtain solutions of good quality in short computational running times for large instances of the problem (say $|V| = 500$ nodes with $0.2 \cdot |V|$ and $0.4 \cdot |V|$ basic vertices). Finally, the paper ends with some conclusions and suggestions for further research in Sect. 4.

2 Algorithms considered

In this section we describe the algorithms that we consider for the MLSteiner problem: an exact method, the Pilot Method by Cerulli et al. (2006), a basic Multi-Start method (with and without an embedded local search), and finally the Jumping Particle Swarm Optimization.

2.1 Exact method

This exact approach to the MLSteiner 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.

The recursive procedure starts from an empty set of colours and iteratively builds a solution by adding colours one by one until all the basic nodes, $Q \subseteq V$, are connected. This method is based on an enumeration of all the possible combinations of colours, so its computational running time grows exponentially with the number of colours. Some heuristic rules are applied to the branch-and-prune strategy in order to reduce the running time. If either the problem size is moderate (say $n \leq 100$) or the optimal objective function value is small (say up to 4–5 colours), the running time of this exact method is reasonable 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. Its core idea consists of exhausting all the possible choices with respect to a so called master solution, by means of a basic heuristic. This basic heuristic tentatively performs iterations with respect to the master solution until all the possible local choices are evaluated. The best solution to date becomes the new master solution, and the procedure proceeds until the user termination conditions are reached.

Cerulli et al. (2006) 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. From their computational analysis, PM obtained the best results. Their Pilot Method for the MLSteiner focuses on the initial label to add, using the null solution (an empty set of colours) as master solution. The basic heuristic consists of inserting in the partial solution the colour which decreases the number

of Steiner components of the partial subgraph. PM tries to add each label at the initial step, and then it performs iterations of the basic heuristic until a feasible solution is obtained. At the final stage, a local search mechanism tries to greedily drop colours (i.e., the associated edges) whilst retaining feasibility. After exhausting all the iterations, the best solution to date represents the output of the method.

2.3 Multi-Start method

The Multi-Start (MS) method that we consider for the MLSteiner problem starts from an empty set of colours, and at each iteration adds one colour at random, until a connected subgraph spanning all the basic nodes is obtained. This process is repeated, continuing until the user termination condition (maximum allowed CPU time, maximum number of iterations, or maximum number of iterations between two successive improvements) is reached. The best solution to date is produced as the output of this method.

A local search phase may be embedded in this process to try to improve the intensification capability of the algorithm. This local search consists of trying to greedily drop some labels (i.e., the associated edges) at the end of each iteration of the MS method, whilst retaining feasibility. Further details on Multi-Start techniques to combinatorial optimization can be found in Martíy (2003).

2.4 Jumping Particle Swarm Optimization

The spirit of nature to deal with some real-life problems is often based on simple processes. Trying to emulate this aspect of life, the discrete PSO proposed in Moreno-Pérez et al. (2007) and Martínez-García and Moreno-Pérez (2008), called Jumping Particle Swarm Optimization (JPSO), was chosen to deal with the minimum labelling Steiner tree problem (Consoli et al. 2008b) for its easy implementation and simplicity.

JPSO considers a swarm S containing n particles ($S = 1, 2, \ldots, n$) whose positions x_i evolve in the solution space, jumping from one solution to another. The number of particles in the swarm was chosen after preliminary experimentation which indicated that using a swarm size of $n = 100$ particles is a reasonable choice. The position of a particle is encoded as a feasible solution to the MLSteiner problem. At each iteration, each particle has a random behaviour, or jumps to another solution in a manner guided by the effect of some attractors.

JPSO considers three attractors for the movement of each particle i: its own best position to date (b_i), the best position of its social neighbourhood (g_i), interpreted as the best position obtained within the swarm in the current iteration, and the best position to date obtained by all the particles, which is called the global best position (g^*). A jump approaching an attractor consists of changing a feature of the current solution by a feature of the attractor. Each particle is further allowed to have a random behaviour by performing random jumps. A random jump consists of selecting at random a feature of the solution and changing its value. For the MLSteiner problem the features of a solution are the colours that are included in the solution. Thus, a particle performs a jump with respect to the selected attractor by randomly adding some colours to its current position from the selected attractor, or dropping from its current position further colours that are not included in the attractor.

Further details of the DPSO that we propose for the MLSteiner problem are specified in Algorithm 1. The algorithm proceeds as follows. The initial positions of the swarm S are generated by starting from empty sets of colours and adding at random colours until

Algorithm 1: Discrete Particle Swarm Optimization for the MLSteiner problem

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

Output: A spanning tree T;

Initialization:

- Let $C \leftarrow 0$ be a set of colours, initially empty set;

- 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:

$X = [x_0, x_1, \ldots, x_{n_s}] \leftarrow$ *Generate-Swarm-At-Random(G)*;

- Update the vector of the best positions $B = [b_0, b_1, \ldots, b_{n_s}] \leftarrow X$;

- 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: $g_i \leftarrow \ell$;

else

- Update the best position of the social neighbourhood i: $g_i \leftarrow g_{i-1}$;

end

- 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: $b_i \leftarrow x_i$;

end

if $|x_i| < |g_i|$**then**

- Update the best position of the social neighbourhood of i : $g_i \leftarrow x_i$;

end

if $|x_i| < |g^*|$**then**

- Update the global best position to date: $g^* \leftarrow x_i$;

end

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

Algorithm 2: 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 $|x_i|$: $\psi \leftarrow$ *Random*(0,$|x_i|$);
for $j \leftarrow 1$ to ψ **do**
  - Select at random a number between 0 and 1: $\xi \leftarrow$ *Random*(0,1);
  if $\xi \leq 0.5$ **then**
    - Select at random a label $c' \in x_i$;
    - Delete label c' from the position of the given particle: $x_i \leftarrow x_i - \{c'\}$;
  else
    - Select at random a label $c' \in$ *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 label $u \in (L - x_i)$;
  - Add label u to the position of the given particle i: $x_i \leftarrow x_i \cup \{u\}$;
end

Algorithm 3: Procedure *Local-Search*(i, x_i)

- Let *Comp*(x_i) be the number of Steiner components of x_i;
for $j = 1$ *to* $|x_i|$ **do**
  if *Comp*($x_i - \{j\}$) $= 1$ **then**
    - Delete label j from the position x_i of the given particle, i.e. $x_i \leftarrow x_i - \{j\}$;
    Update *Comp*(x_i);
  end
end

feasible solutions emerge. According to our experience this is the best choice for the initial positions of the swarm. The position x_i of a particle i is a 0–1 vector denoting which labels are present in particle i. At each iteration, the positions of the particles are updated. Considering the i-th particle of the swarm ($i \in S$) and a generic iteration k, the update procedure to obtain the new position x_i^k of i from its previous position x_i^{k-1} is as follows. Position x_i^k is obtained by performing random jumps with respect to its current position x_i^{k-1} 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 $c_4 = (1 - c_1 - c_2 - c_3)$. For the MLSteiner problem the value of the parameters c_1, c_2, c_3, c_4, are set to 0.25, giving the same probability value to each of the possible jumps to be selected. That is, a random number ξ between 0 and 1 is selected. 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* $\leftarrow b_i$) 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 (*selected* $\leftarrow g^*$).

The jump of the i-th particle towards the selected attractor (*selected*) is performed by means of the procedure *Combine*(x_i, *selected*) (see Algorithm 2). In this procedure we make use of the concept of a Steiner component (Cerulli et al. 2006), which is a connected subgraph of the input graph containing at least one basic node. A solution is feasible if and only if it contains only one Steiner component, by assuring that all the basic nodes are connected. Let *Comp*(x_i) be the number of Steiner components of x_i. The procedure *Combine* first selects a random integer ψ between 0 and $|x_i|$. Successively, it either drops some colours from x_i, or randomly picks up some colours from the selected attractor and adds to x_i, until ψ colours have been added or deleted with respect to x_i. At this point the number of Steiner components of the solution x_i is updated (Update *Comp*(x_i)). If an infeasible x_i is obtained at this stage (i.e. *Comp*(x_i) > 1), further colours are added at random to x_i from the set of unused colours ($L - x_i$) until the feasibility is restored (*Comp*(x_i) $= 1$).

At the end of this stage, a local search procedure is applied to the resulting particle, in order to try to delete some colours from x_i (i.e., the associated edges) whilst retaining feasibility (see Algorithm 3). After the local search phase, all the attractors (b_i, g_i, g^*) are updated. Note that the best position (g_i) of the social neighbourhood of a particle i consists of the best position obtained within the swarm in the current iteration. Thus, when the first particle in the swarm is evaluated ($i = 1$), the best position of its social neighbourhood is initialized with the worst possible solution, that is the total number of colours ($g_i \leftarrow \ell$). In contrast, all the successive particles in the swarm ($i \geq 1$) initialize the best position of the social neighbourhood with the best position of the social neighbourhood of the previous particle, that is: $g_i \leftarrow g_{i-1}$.

The same procedure is repeated for all the particles in the swarm, and the entire algorithm continues until the user termination conditions are satisfied.

3 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: EXACT (exact method), PM (Pilot Method), MS (Multi-Start method), MS + LS (Multi-Start method with the local search mechanism), and JPSO (Jumping Particle Swarm Optimization).

Different sets of instances of the problem have been generated considering combinations of the following parameters:

- the total number of edges of the graph (m);
- the total number of nodes of the graph (n);
- the number of basic nodes of the graph (q);
- the total number of colours assigned to the edges of the graph (ℓ).

In our experiments, we consider 48 different datasets, each one containing 10 instances of the problem, with $n \in \{100, 500\}$ nodes, $\ell \in \{0.25n, 0.5n, n, 1.25n\}$ colours, and $q \in \{0.2n, 0.4n\}$ basic nodes. The number of edges, m, is obtained indirectly from the density d, whose values are chosen to be in {0.8, 0.5, 0.2}. We thank the authors of Cerulli et al. (2006), who kindly provided data for use in our experiments.

For each dataset, solution quality is evaluated as the average objective function 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. Since the Pilot Method considers, for each instance, every label as the initial colour to add, *max-CPU-time* is chosen 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.

Our computational results are reported in Tables 1, 2, 3, and 4. In each table, the first two columns show the parameters characterizing the different datasets (ℓ, d), while the values of n and q determine the different tables. All the heuristic methods run for *max-CPU-time* and, in each case, the best solution is recorded. All the computations have been made on a Pentium Centrino microprocessor at 2.0 GHz with 512 MB RAM. The computational times reported in the tables are the average times at which the best solutions are obtained. Where possible, the results of the metaheuristics are compared with the exact

Table 1 Computational results for $n = 100$ and $q = 0.2n$ (*max-CPU-time* = 5,000 ms)

Parameters			EXACT	PM	MS	MS + LS	JPSO
n	ℓ	d					
Average objective function values							
100	25	0.8	1	1	1	1	1
		0.5	1.5	1.5	1.5	1.5	1.5
		0.2	2.1	2.1	2.1	2.1	2.1
	50	0.8	1.9	1.9	1.9	1.9	1.9
		0.5	2	2	2	2	2
		0.2	3.2	3.2	3.2	3.2	3.2
	100	0.8	2	2	2	2	2
		0.5	3	3	3	3	3
		0.2	4.6	4.6	5.7	4.6	4.6
	125	0.8	2.8	2.8	2.8	2.8	2.8
		0.5	3.3	3.3	3.6	3.3	3.3
		0.2	5.2	5.4	6.5	5.4	5.2
Total:			32.6	32.8	35.4	32.8	32.6
Computational times (ms)							
100	25	0.8	14.7	14.1	10.6	10.6	1.6
		0.5	26.3	20.3	10.5	10.5	3.2
		0.2	16.2	15.6	20.9	13.2	6.1
	50	0.8	59.4	56.1	22.6	11.6	6.4
		0.5	66.3	67.2	26.4	24.6	10.9
		0.2	40.6	75.1	199.9	51.4	15.7
	100	0.8	306.3	270.3	167.6	51.8	75.1
		0.5	251.6	275.1	309	57.7	31.2
		0.2	914	314.1	635.8	792.1	45.3
	125	0.8	78.2	381.2	233.8	121.8	48.4
		0.5	451.5	443.9	482.8	469	157.7
		0.2	4703.2	518.8	1659.4	1007.9	322
Total:			6828.3	2451.8	3779.3	2622.2	723.6

Table 2 Computational results for $n = 100$ and $q = 0.4n$ (*max-CPU-time* = 6,000 ms)

Parameters			EXACT	PM	MS	MS + LS	JPSO
n	ℓ	d					
Average objective function values							
100	25	0.8	1	1	1	1	1
		0.5	1.9	1.9	1.9	1.9	1.9
		0.2	3	3	3	3	3
	50	0.8	2	2	2	2	2
		0.5	2.2	2.2	2.2	2.2	2.2
		0.2	4.3	4.4	4.5	4.3	4.3
	100	0.8	3	3	3	3	3
		0.5	3.6	3.6	3.6	3.6	3.6
		0.2	NF	6.5	8.7	6.8	6.4
	125	0.8	3	3	3	3	3
		0.5	4	4	4.4	4.1	4
		0.2	NF	7	10.7	8	6.9
Total:			–	41.6	48	42.9	41.3
Computational times (ms)							
100	25	0.8	24.7	15.6	11.2	11.6	9.3
		0.5	29.7	21.7	14.8	11.6	6.4
		0.2	36.9	29.8	25.6	25	23.6
	50	0.8	60.9	53	15.6	13.1	20.4
		0.5	117.2	76.6	47.5	39.7	34.3
		0.2	314.1	111	1093.8	129	45.1
	100	0.8	175	260.9	169.6	48	39.2
		0.5	389.1	312.5	1148.4	157.9	96.8
		0.2	NF	472	1604.7	870.7	350
	125	0.8	354.6	440.7	237.5	81.1	57.6
		0.5	479.6	507.8	643.7	887.6	67.1
		0.2	NF	811	2012.7	1072	411
Total:			–	3112.6	7025.1	3347.3	1160.8

solution (EXACT). The solution given by the exact method is reported unless a single instance computes for more than 3 hours of CPU time. In the case that no solution is obtained in 3 hours by the exact method, a not found (NF) is reported in the tables. All the reported times have precision of ± 5 ms.

Tables 1 and 2 examine relatively small instances of the MLSteiner problem. Looking at these tables, all the metaheuristics performed well for the considered problem instances. The Multi-Start method obtained the worst performance with respect to the solution quality and computational running time. The addition of the local search mechanism improves the results of the Multi-Start method, although MS + LS is generally slower than PM and JPSO as a result of a poor intensification capability and an excessive diversification capability for these instances. PM is characterized sometimes by a limited diversification capability which does not allow the search process to escape from local optima. For

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

Parameters			EXACT	PM	MS	MS + LS	JPSO
n	ℓ	d					
Average objective function values							
500	25	0.8	1.1	1.1	1.1	1.1	1.1
		0.5	2	2	2	2	2
		0.2	3	3	3	3	3
	50	0.8	2	2	2	2	2
		0.5	2.9	2.9	2.9	2.9	2.9
		0.2	NF	4.4	5	4.8	4.3
	100	0.8	3	3	3.1	3	3
		0.5	NF	3.9	4.5	4.5	4
		0.2	NF	6.8	9.4	8.3	6.9
	125	0.8	NF	3.8	4	4	3.8
		0.5	NF	4.8	5.7	5.3	4.8
		0.2	NF	8	11	9.8	7.9
Total:			–	45.7	53.7	50.7	45.7
Computational times (ms)							
500	25	0.8	1.5×10^3	1.2×10^3	2.5×10^3	876.1	3.4×10^3
		0.5	2.1×10^3	2.5×10^3	1.6×10^3	640.1	575
		0.2	4.1×10^3	7.1×10^3	7.2×10^3	1.6×10^3	5.9×10^3
	50	0.8	13.6×10^3	17.4×10^3	22×10^3	3.6×10^3	9.7×10^3
		0.5	37.3×10^3	46.8×10^3	28.1×10^3	10.5×10^3	8.8×10^3
		0.2	NF	48.1×10^3	82.5×10^3	47.1×10^3	36.7×10^3
	100	0.8	300.8×10^3	304.4×10^3	360×10^3	235.3×10^3	22.1×10^3
		0.5	NF	325.8×10^3	361.7×10^3	332.3×10^3	106.5×10^3
		0.2	NF	452.2×10^3	326.5×10^3	399.1×10^3	170.4×10^3
	125	0.8	NF	465.6×10^3	305.7×10^3	383.5×10^3	180.2×10^3
		0.5	NF	403×10^3	494.3×10^3	361.9×10^3	110.4×10^3
		0.2	NF	399.3×10^3	488.6×10^3	443.2×10^3	285.7×10^3
Total:			–	2446.4×10^3	2480.7×10^3	2219.6×10^3	940.4×10^3

example, PM was faster than MS + LS in the instance [$n = 100$, $\ell = 50$, $d = 0.2$] in Table 2 but it produced a slightly worse result with respect to solution quality. JPSO obtained the best performance in terms of solution quality and computational running time with respect to the other algorithms for all these small problem instances.

Tables 3 and 4 show larger instances of the problem ($n = 500$ with $q = 0.2n$ and $q = 0.4n$ respectively). The inspection of these tables demonstrates that JPSO is able to obtain good performance also in large problem instances. Indeed, JPSO obtained the solutions with the best quality and computational running times in most of the cases, even though the running times increase considerably with the size of the instances. As in the previous analysis, PM and MS + LS showed the same relative behaviour for the considered instances. However, MS + LS showed an excessive diversification and poor intensification capabilities, obtaining poor performance in terms of solution quality. Again, the Multi-Start method without the local search mechanism produced the worst results with

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

Parameters			EXACT	PM	MS	MS + LS	JPSO
n	ℓ	d					
Average objective function values							
500	25	0.8	1.9	1.9	1.9	1.9	1.9
		0.5	2	2	2	2	2
		0.2	NF	4.1	4.4	4.1	4.1
	50	0.8	2	2	2	2	2
		0.5	3	3	3	3	3
		0.2	NF	6.2	8.5	7.4	6.3
	100	0.8	NF	3.7	4	4	3.7
		0.5	NF	5	5.7	5.5	5
		0.2	NF	9.9	16.6	12.6	9.9
	125	0.8	NF	4	5	4.8	4
		0.5	NF	5.8	6.9	6.6	5.7
		0.2	NF	11.5	18.6	14.6	11.4
Total:			–	59.1	78.6	68.5	59
Computational times (ms)							
500	25	0.8	218	1.1×10^3	1.2×10^3	900	778.2
		0.5	2.8×10^3	2.6×10^3	2.5×10^3	6.5×10^3	4.3×10^3
		0.2	NF	8.3×10^3	70×10^3	101×10^3	8.8×10^3
	50	0.8	44.6×10^3	20.2×10^3	21.1×10^3	12.7×10^3	12.5×10^3
		0.5	48.8×10^3	49.8×10^3	59.8×10^3	46.9×10^3	13.4×10^3
		0.2	NF	48.7×10^3	180×10^3	160.2×10^3	122.2×10^3
	100	0.8	NF	201.1×10^3	282.6×10^3	282.5×10^3	19.4×10^3
		0.5	NF	193.1×10^3	269.9×10^3	229.8×10^3	19.6×10^3
		0.2	NF	579.7×10^3	470.8×10^3	497.5×10^3	195.3×10^3
	125	0.8	NF	384×10^3	329.9×10^3	353.7×10^3	18.5×10^3
		0.5	NF	421.2×10^3	428.1×10^3	375×10^3	32.6×10^3
		0.2	NF	397.9×10^3	479.4×10^3	397.5×10^3	232.1×10^3
Total:			–	2307.7×10^3	2595.3×10^3	2422.2×10^3	679.5×10^3

Table 5 Pairwise differences of the average ranks of the algorithms*

Algorithm (rank)	JPSO (1.42)	MS + LS (2.57)	PM (2.85)	MS (3.92)	EXACT (4.24)
JPSO (1.42)	–	*1.15*	*1.43*	*2.5*	*2.82*
MS + LS (2.57)	–	–	0.28	*1.35*	*1.67*
PM (2.85)	–	–	–	*1.07*	*1.39*
MS (3.92)	–	–	–	–	0.32
EXACT (4.24)	–	–	–	–	–

* Critical difference = 1.05 for a significance level of 1% for the Nemenyi test

respect to solution quality and computational running time. It is also interesting to note that in all the problem instances in Tables 1–4 for which the exact method obtains the solution, JPSO also yielded the exact solution.

From this analysis, JPSO obtained the best performance for all the considered datasets. In addition, to confirm this evaluation, the ranks of the algorithms for each dataset are evaluated, with a rank of 1 assigned to the best performing algorithm, a rank of 2 to the second best one, and so on. The performance of an algorithm is considered better than another one if either it obtains a smaller average objective function value, or an equal average objective function value but in a shorter computational time. 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: EXACT = 4.24, PM = 2.85, MS = 3.92, MS + LS = 2.57, JPSO = 1.42. According to the ranking, JPSO is the best performing algorithm, followed by MS + LS, then PM, MS and finally EXACT achieving the worst results. Thus, this evaluation further indicates the superiority of JPSO with respect to the other approaches.

To analyse the statistical significance of differences between the evaluated ranks, we make use of the *Friedman Test* (Friedman 1940) and its corresponding *Nemenyi 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 1% of significance level). Since the equivalence of the algorithms is rejected, the Nemenyi post-hoc test is applied in order to perform pairwise comparisons. It considers the performance of two algorithms significantly different if their corresponding average ranks differ by at least a specific threshold critical difference. In our case, considering a significance level of the Nemenyi test of 1%, this critical difference is 1.05. The differences between the average ranks of the algorithms are reported in Table 5. The italicized values illustrate the pairwise differences larger than the critical difference.

From this table, it is possible to identify three groups of algorithms with different performance. The best performing group consists of just JPSO, because it obtains the smallest rank which is significantly different from all the other ranks. The remaining groups are, in order, MS + LS and PM, and then MS and EXACT. Considering a significance level 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, JPSO is the best performing algorithm. On the other hand, MS + LS and PM have comparable performance, better than that of MS and EXACT. From this further analysis, the results reinforce the conclusion that JPSO is an effective metaheuristic for the MLSteiner problem. On average, it was the best performing algorithm with respect to solution quality and computational running time.

4 Conclusions

In this work we considered a Discrete Particle Swarm Optimization (DPSO), called Jumping Particle Swarm Optimization (JPSO), for the minimum labelling Steiner tree (MLSteiner) problem (Consoli et al. 2008b). This is a NP-hard graph problem extending the well-known Steiner tree problem, and the minimum labelling spanning tree problem to the case where only a subset of specified nodes, the basic nodes, need to be connected. Considering a wide range of problem instances, JPSO was compared with other algorithms: an exact approach, the Pilot Method (PM) by Cerulli et al. (2006) (the most popular

MLSteiner heuristic in the literature), and a basic Multi-Start (MS) technique (with and without an embedded local search). Based on our computational analysis, JPSO clearly outperformed all the other procedures, obtaining high-quality solutions in short computational running times.

An interesting area of future research would be to apply JPSO to other important combinatorial optimization problems such as the Travelling Salesman Problem and Job Shop Scheduling and evaluating its performance with respect to sensitivity to initialisations and parameter settings and also comparing performance with that of other DPSO algorithms.

Acknowledgements 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). The research of José Andrés Moreno-Pérez was partially 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.

References

Al-kazemi B, Mohan CK (2002) Multi-phase discrete particle swarm optimization. In: Fourth international workshop on frontiers in evolutionary algorithms, Kinsale, Ireland

Castro-Gutiérrez JP, Landa-Silva D, Moreno-Pérez JA (2008) Exploring feasible and infeasible regions in the vehicle routing problem with time windows using a multi-objective particle swarm optimization approach. In: Proceedings of international workshop on nature inspired cooperatives strategies for optimization (NICSO 2008)

Cerulli R, Fink A, Gentili M, Voß S (2005) Metaheuristics comparison for the minimum labelling spanning tree problem. In: Golden BL, Raghavan S, Wasil EA (eds) The next wave on computing, optimization, and decision technologies. Springer-Verlag, New York, pp 93–106

Cerulli R, Fink A, Gentili M, Voß S (2006) Extensions of the minimum labelling spanning tree problem. J Telecommun Inf Technol 4:39–45

Chang RS, Leu SJ (1997) The minimum labelling spanning trees. Inf Process Lett 63(5):277–282

Consoli S, Darby-Dowman K, Mladenović N, Moreno-Pérez JA (2008a) Greedy randomized adaptive search and variable neighbourhood search for the minimum labelling spanning tree problem. Eur J Oper Res 196(2):440–449

Consoli S, Moreno-Pérez JA, Darby-Dowman K, Mladenović N (2008b) Discrete particle swarm optimization for the minimum labelling Steiner tree problem. In: Krasnogor N, Nicosia G, Pavone M, Pelta D (eds) Nature inspired cooperative strategies for optimization, studies in computational intelligence, vol 129. Springer-Verlag, New York, pp 313–322

Correa ES, Freitas AA, Johnson CG (2006) A new discrete particle swarm algorithm applied to attribute selection in a bioinformatic data set. In: Proceedings of GECCO 2006, pp 35–42

Demśar J (2006) Statistical comparison of classifiers over multiple data sets. J Mach Learn Res 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

Friedman M (1940) A comparison of alternative tests of significance for the problem of m rankings. Ann Math Stat 11:86–92

Garey MR, Graham RL, Johnson DS (1977) The complexity of computing Steiner minimal trees. SIAM J Appl Math 32:835–859

Holland JH (1992) Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. The MIT Press, Cambridge

Hollander M, Wolfe DA (1999) Nonparametric statistical methods, 2nd edn. Wiley, New York

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. Morgan Kaufmann Publishers, San Francisco

Krumke SO, Wirth HC (1998) On the minimum label spanning tree problem. Inf Process Lett 66(2):81–85

Martí R (2003) Multi-start methods. In: Glover F, Kochenberger G (eds) Handbook in metaheuristics. Kluwer, Dordrecht, pp 335–368

Martínez-García FJ, Moreno-Pérez JA (2008) Jumping frogs optimization: a new swarm method for discrete optimization. Technical Report DEIOC 3/2008, Department of Statistics, O.R. and Computing, University of La Laguna, Tenerife, Spain

Moraglio A, Di Chio C, Togelius J, Poli R (2008) Geometric particle swarm optimization. J Artif Evol Appl. doi:10.1155/2008/143624

Moreno-Pérez JA, Castro-Gutiérrez JP, Martínez-García FJ, Melián B, Moreno-Vega JM, Ramos J (2007) Discrete Particle Swarm Optimization for the p-median problem. In: Proceedings of the 7th metaheuristics international conference, Montréal, Canada

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

Pampara G, Franken N, Engelbrecht AP (2005) Combining Particle Swarm Optimisation with angle modulation to solve binary problems. In: Proceedings of the IEEE congress on evolutionary computing, vol 1, pp 89–96

Pugh J, Martinoli A (2006) Discrete multi-valued particle swarm optimization. In: Proceedings of IEEE swarm intelligence symposium, vol 1, pp 103–110

Qu R, Xu Y, Castro-Gutiérrez JP, Landa-Silva D (2009) Particle swarm optimization for the Steiner tree in graph and delay-constrained multicast routing problems. Swarm Intelligence (submitted)

Secrest BR (2001) Traveling salesman problem for surveillance mission using Particle Swarm Optimization. Master's thesis, School of Engineering and Management of the Air Force Institute of Technology

Tanenbaum AS (1989) Computer Networks. Prentice-Hall, Englewood Cliffs

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

Voß S (2000) Modern heuristic search methods for the Steiner tree problem in graphs. In: Du DZ, Smith JM, Rubinstein JH (eds) Advances in Steiner tree. Kluwer, Boston, pp 283–323

Xiong Y, Golden B, Wasil E (2006) Improved heuristics for the minimum labelling spanning tree problem. IEEE Trans Evol Comput 10(6):700–703

Yang S, Wang M, Jiao L (2004) A Quantum Particle Swarm Optimization. In: Proceedings of CEC2004, the congress on evolutionary computing, vol 1, pp 320–324