Noname manuscript No.
(will be inserted by the editor)

Discrete Particle Swarm Optimization for the minimum labelling Steiner tree problem

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

Received: date / Accepted: date

Abstract Particle Swarm Optimization is an evolutionary 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 evolution is obtained from the continuous movement of the particles that constitute the swarm submitted to the effect of the 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

Mathematics Subject Classification (2000) MSC 90C27 · MSC 90C35 · MSC 90C59

S. Consoli
CARISMA and NET-ACE, School of Information Systems, Computing and Mathematics, Brunel University, Uxbridge, Middlesex, UB8 3PH, United Kingdom
Tel.: +44 (0)1895 266820 Fax: +44 (0)1895 269732
E-mail: sergio.consoli@brunel.ac.uk

J.A. Moreno-Pérez
DEIOC, IUDR, Universidad de La Laguna, 38271, Santa Cruz de Tenerife, Spain
Tel.: +34-922318186 Fax: +34-922318170
E-mail: jamoreno@ull.es

K. Darby-Dowman
CARISMA and NET-ACE, School of Information Systems, Computing and Mathematics, Brunel University, Uxbridge, Middlesex, UB8 3PH, United Kingdom
Tel.: +44 (0)1895 265602 Fax: +44 (0)1895 269732
E-mail: kenneth.darby-dowman@brunel.ac.uk

N. Mladenović
CARISMA and NET-ACE, School of Information Systems, Computing and Mathematics, Brunel University, Uxbridge, Middlesex, UB8 3PH, United Kingdom
Tel.: +44 (0)1895 266151 Fax: +44 (0)1895 269732
E-mail: nenad.mladenovic@brunel.ac.uk

1 Introduction

Over the years, evolutionary algorithms have been widely used as robust techniques for solving hard combinatorial optimization (CO) problems. Their behaviour is directed by the evolution of a population in the search for an optimum. Particle Swarm Optimization (PSO) is an evolutionary algorithm 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). However, unlike Genetic Algorithms, it has no crossover and mutation operators and is easy to implement, requiring few parameter settings and computational memory. 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 focus on a very recent DPSO for combinatorial optimization, illustrating its effectiveness on 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 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 $x_i^k = x_i^{k-1} + v_i^k$.

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:

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

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.

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), several Discrete Particle Swarm Optimization (DPSO) methods have been proposed. In the DPSO proposed by Kennedy and Eberhart (1997) for problems with binary variables, 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 of the d-dimensional continuous space, $v_i \in \Re^d$. A different way to update the velocity was considered by Yang et al. (2004).

A DPSO whose particles at each iteration are affected alternatively by its best position and the best position among its neighbours was proposed by Al-kazemi and Mohan (2002). Pampara et al. (2005) solved binary problems by combining continuous PSO and Angle Modulation with only four parameters. Furthermore, several PSO variants applied to problems where the solutions are permutations were considered in (Onwubolu and Clerc 2004; Pang et al. 2004; Secrest 2001).

The multi-valued PSO (MVPSO) proposed by Pugh and Martinoli (2006) deals with variables with multiple discrete values. The position of each particle is a mono-dimensional array in the case of a continuous PSO, a 2-dimensional array in the case of a DPSO, and a 3-dimensional array for a MVPSO. Indeed, the position of particle i in the MVPSO is expressed by the term x_{ijk}, representing the probability that the i-th particle, in the j-th iteration, takes the k-th value.

Another DPSO was proposed in (Correa et al. 2006) for feature selection problems, which are problems whose solutions are sets of items. In this PSO version, the velocity vectors consist of positive numbers representing the relative likelihood of the corresponding binary component of the positions of the particles. The position of each particle is updated by randomly generating changes according to these likelihoods, and then continuing in similar way to the standard PSO.

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 loses sense; 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 the 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). In this paper we compare this method with other algorithms for the minimum labelling Steiner tree problem.

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.

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). 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 locations, without cycles, using the minimum number of companies, in order to minimize the costs.

Formally, the minimum labelling Steiner tree problem can be 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 on the set L of labels (or colours).
- 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 and such that $|L_T|$ is minimized (where L_T is the set of colours used by T).

- INSERT FIGURE 1 -

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 Figure 2.

- INSERT FIGURE 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 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; Consoli et al. 2008).

The MLSteiner problem was considered by Cerulli et al. (2006) where their Pilot Method (PM) is compared with some other metaheuristics: Tabu Search, Simulated Annealing, and Variable Neighbourhood Search. From their analysis, PM was shown to be the best performing heuristic for this problem.

The structure of the paper is as follows. Section 2 presents the details of the methods that we consider for the MLSteiner problem: an exact approach, the Pilot Method proposed by Cerulli et al. (2006), a basic Multi-Start (MS) metaheuristic (with and without an embedded local search), and the Jumping Particle Swarm Optimization. Section 3 shows the computational results of the comparison of these algorithms, followed by some conclusions in Section 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. Throughout the rest of the paper we will 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.

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 or the optimal objective function value is small, 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 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 (Tabú 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 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 (Martí 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; Martínez-García and Moreno-Pérez 2008), called Jumping Particle Swarm Optimization (JPSO), was chosen to deal with the minimum labelling Steiner tree problem, for its ease 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 (for the MLST problem, a swarm with $n = 100$ particles is considered). 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 feasible solutions emerge. 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. The number of jumps to be performed at each iteration is selected at random in the following way. Given

the probability c_i, we iteratively choose a random number ξ with uniform distribution [0,1]. If $\xi < c_i$, we perform a new jump; otherwise the jumps stop.

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^k whilst retaining 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.

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 the initially empty set of used colours for each iteration;
- Let $H = (V, E(C))$ be the subgraph of G restricted to V and edges with labels in C, where $E(C) = \{e \in E : L(e) \in C\}$;
- Set the size 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: ξ=*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* ;
- Update $H_{g^*} = (V, E(g^*))$;
$\Rightarrow$ Take any arbitrary spanning tree T of $H_{g^*} = (V, E(g^*))$.
end

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, $\ell \in \{0.25n, 0.5n, 1.25n\}$ 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 $\{0.8, 0.5, 0.2\}$.

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.

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

Our computational results are reported in Tables 1 - 4. In each table, the first two columns show the parameters characterising 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 times at which the best solutions are obtained. Where possible, the results of the metaheuristics are compared with the exact 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 *max-CPU-time* by the metaheuristics, in 3 hours by the exact method, a not found (NF) is reported in the tables. All the reported times have precision of ± 5 msec.

- INSERT TABLE 5 -

Table 5 shows the relative performance of the algorithms considered. The entry (i, j) in this table represents the number of instances where algorithm i had better performance than algorithm j. 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 (±5 ms). For example, PM generates 32 solutions that are better than those generated by MS, while JPSO generates 38 solutions better than those generated by MS+LS. In the right-most column, the row "TOTAL" gives the number of times each algorithm has outperformed all the others. The overall ranking (from best to worst) with respect to this evaluation is JPSO, MS+LS, PM, EXACT, and MS. Thus, from our analysis the most effective algorithm for the MLSteiner problem is JPSO. On average, it was the best performing with respect to solution quality and computational running time.

4 Conclusions

In this work we proposed a Discrete Particle Swarm Optimization (DPSO), called Jumping Particle Swarm Optimization (JPSO), for the minimum labelling Steiner tree (MLSteiner) problem. 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. This confirms the ability of nature-inspired methodologies to deal with NP-hard combinatorial problems.

Future research will consist of applying this very recent metaheuristic to other well-known CO problems, such as the Travelling Salesman Problem and Job Shop Scheduling, among others.

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

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. Journal of Telecommunications and Information Technology 4:39–45

Chang RS, Leu SJ (1997) The minimum labelling spanning trees. Information Processing Letters 63(5):277–282

Consoli S, Darby-Dowman K, Mladenović N, Moreno-Pérez JA (2008) Greedy randomized adaptive search and variable neighbourhood search for the minimum labelling spanning tree problem. European Journal of Operational Research, accepted for publication Doi:10.1016/j.physletb.2003.10.071

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

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

Garey MR, Graham RL, Johnson DS (1977) The complexity of computing Steiner minimal trees. SIAM Journal on Applied Mathematics 32:835–859

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, CA

Krumke SO, Wirth HC (1998) On the minimum label spanning tree problem. Information Processing Letters 66(2):81–85

Martí R (2003) Multi-Start Methods. In: Glover F, Kochenberger G (eds) Handbook in Metaheuristics, Kluwer Academic Publishers, pp 335–368

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

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

Onwubolu GC, Clerc M (2004) Optimal operating path for automated drilling operations by a new heuristic approach using particle swarm optimisation. International Journal of Production Research 42(3):473–491

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

Pang W, Wang K, Zhou C, Dong L (2004) Fuzzy discrete particle swarm optimization for solving traveling salesman problem. In: Proceedings of the 4th International Conference on Computer and Information Technology (CIT04), IEEE Computer Society, 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

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, New Jersey

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

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 Transactions on Evolutionary Computation 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

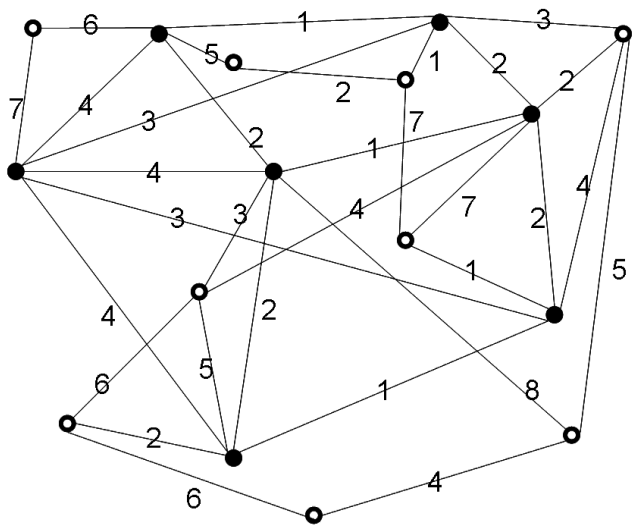

Fig. 1 Example of a labelled connected undirected graph, input of the MLSteiner problem. The solid vertices represent the basic nodes to cover.

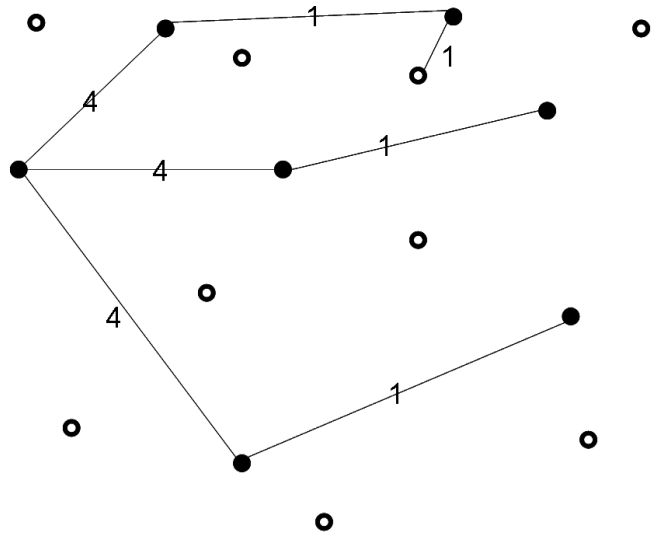

Fig. 2 Minimum labelling Steiner tree solution for the graph of Figure 1.

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

Parameters			**Average objective function values**				
n	ℓ	d	EXACT	PM	MS	MS+LS	JPSO
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

Parameters			**Computational times (msec)**				
n	ℓ	d	EXACT	PM	MS	MS+LS	JPSO
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*=6000 msec).

Parameters			**Average objective function values**				
n	ℓ	d	EXACT	PM	MS	MS+LS	JPSO
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

Parameters			**Computational times (msec)**				
n	ℓ	d	EXACT	PM	MS	MS+LS	JPSO
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

Table 3 Computational results for $n = 500$ and $q = 0.2n$ ($max\text{-}CPU\text{-}time$=$500*10^3$ msec).

Parameters			**Average objective function values**				
n	ℓ	d	EXACT	PM	MS	MS+LS	JPSO
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

Parameters			**Computational times (msec)**				
n	ℓ	d	EXACT	PM	MS	MS+LS	JPSO
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:			-	$2446.4*10^3$	$2480.7*10^3$	$2219.6*10^3$	$940.4*10^3$

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

Parameters			**Average objective function values**				
n	ℓ	d	EXACT	PM	MS	MS+LS	JPSO
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

Parameters			**Computational times (msec)**				
n	ℓ	d	EXACT	PM	MS	MS+LS	JPSO
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 Relative performance of the algorithms. Each element (i, j) represents the number of datasets for which algorithm i has better performance than algorithm j.

	EXACT	PM	MS	MS+LS	JPSO	TOTAL
EXACT	-	14	15	8	4	41
PM	29	-	32	21	3	85
MS	30	13	-	3	2	38
MS+LS	37	23	38	-	5	103
JPSO	44	43	43	38	-	168