Particle Swarm Optimization for the Steiner Tree in Graph and Delay-Constrained Multicast Routing Problems

Rong Qu[*] · Ying Xu · Juan P. Castro · Dario Landa-Silva

Abstract This paper presents the first investigation on applying a Particle Swarm Optimization (PSO) algorithm to both the Steiner tree problem and the delay constrained multicast routing problem. Steiner tree problems, being the underlining models of many applications, have received significant research attention within the meta-heuristics community. The literature on the application of meta-heuristics to multicast routing problems is less extensive but includes several promising approaches. Many interesting research issues still remain to be investigated, for example, the inclusion of different constraints, such as delay bounds, when finding multicast trees with minimum cost. In this paper, we develop a novel JPSO algorithm based on the jumping PSO (JPSO) algorithm recently developed by Moreno-Perez et al. (2007), and also propose two novel local search heuristics within our PSO framework. A path replacement operator has been used in particle moves to improve the positions of the particle with regard to the structure of the tree. We test the performance of our JPSO algorithm, and the effect of the integrated local search heuristics by an extensive set of experiments on multicast routing benchmark problems and Steiner tree problems from the OR library. The experimental results show the superior performance of the proposed JPSO algorithm over a number of other state-of-the-art approaches.

Keywords delay constrained multicast routing · Steiner tree problems · particle swarm optimization

1. Introduction

Multimedia applications such as video/audio conferencing and distance education demand multicast communications, where data streams are sent from the source node to a set of destinations within the same multicast group in computer networks. The objective is to maximize the multicast throughput within limited and constrained resources. The quality of service (QoS) requirements in the underlying computer network take into account several attributes such as cost, delay, delay variation, packet losses and hop count. Due to the rapid increase in the demand of multimedia services, as well as to the challenges related to the implementation of effective multicast communications, multicast routing problems have recently attracted an increasing attention from the meta-heuristics research community in both computer communications and operational research.

The underlying model for multicast routing problems and a number of other problems is the Steiner tree problem (Hwang and Richards, 1992), a well known NP-hard combinatorial optimization problem (Garey and Johnson, 1979). This problem has been widely studied for decades, and still presents a great research challenge. When solving real life multicast routing problems, more constraints need to be considered in addition to the problem of finding a Steiner tree. The two most common and important QoS requirements when constructing multicast trees are the delay and the cost. The end-to-end delay is the sum of the total delays long the paths from the source to each destination. In real time communications, this delay should be within a certain delay bound. The cost of the multicast tree is the sum of the cost of all links in the tree. A general form of the cost occurs from using and/or reserving network resources, such as the bandwidth, when sending data streams via the network links. Other specific costs can also be defined depending on the network being used in the problem.

In multicast routing problems, finding the multicast tree with the minimal cost while satisfying the delay bound constraint is equivalent to the problem of finding a delay-constrained Steiner tree, thus the former is also a NP-hard problem. The nature of the problem and the variety of constraints that exist in

[*] Rong Qu[1] (Corresponding author), Ying Xu[1,2], Juan P. Castro[1], Dario Landa-Silva[1]
[1] School of Computer Science, The University of Nottingham, Nottingham, NG8 1BB, UK; Tel: ++44 115 8466503, Fax: ++44 115 8467877, e-mail: rxq@cs.nott.ac.uk
[2] School of Computer and Communication, Hunan University, Changsha, 410082, CHINA

real life applications have motivated an increasing interest from the scientific community to developing various optimization and search algorithms, including meta-heuristics, for this problem (Oliveira and Pardalos, 2005).

In this paper, we investigate the Particle Swarm Optimization (PSO) algorithm (Kennedy and Eberhart, 1995) for solving the Delay-Constrained Least-Cost (DCLC) multicast routing problem and also the underlying Steiner tree problem. Based on the Jumping PSO (JPSO) recently developed by Moreno-Perez et al. (2007), operations which take into account the structure and features of the tree are carried out on selected paths within particle moves to reduce the cost of the tree. Two novel local search algorithms have been hybridized to intensify the search to neighboring solutions. To the best of our knowledge, no previous investigation has been carried out on using PSO (or other meta-heuristics) to tackle both problems. Most related papers in the literature have mainly focused on one of them. On the DCLC multicast routing problem, we compare our JPSO algorithm against the best known results reported in the literature. On the Steiner tree problem, we assess the efficiency and effectiveness of our JPSO algorithm by calculating the exact gap to the global optimal solutions known for the benchmark datasets used. A large amount of experiments and simulations demonstrate that our JPSO algorithm obtains the best quality solutions for the DCLC multicast routing problems and highly competitive results for the Steiner tree benchmark problems considered in this work.

The rest of the paper is organized as follows. We first present the network model and formulations for DCLC multicast routing problems and Steiner tree problems in Section 2. Related literature on both problems is also reviewed. The JPSO algorithm and our proposed JPSO algorithm are presented in Sections 3 and 4, and then evaluated through extensive experimentations in Section 5. Finally, Section 6 concludes the paper and proposes potential future work.

2. Problem Definitions and Related Work

2.1 The Delay-Constrained Least Cost (DCLC) Multicast Routing Problem

The DCLC multicast routing problem can be defined by using a directed graph $G = (V, E)$, where V is a set of nodes and E is a set of links, respectively. The nodes in V include a source node s, a set of destination nodes $R \subseteq V - \{s\}$ which receive data streams from the source, and a set of relay nodes which are intermediate hops on the paths from the source s to the destinations R. The set of paths linking the source to the destination nodes are also called multicast groups. The number of destination nodes $|R|$ is also called the group size.

Within the multicast network, each link $e = (i, j) \in E$ from node i to node j is associated with a link cost $C(e)$: $E \mapsto R^+$ and a link delay $D(e)$: $E \mapsto R^+$, where R^+ are nonnegative real numbers. In the general case, computer networks are asymmetric, i.e. the links in G are bidirectional, and it is possible that $C(e) \neq C(e')$ and $D(e) \neq D(e')$, with $e = (i, j) \in E$ and $e' = (j, i) \in E$, $i, j \in V$. A path $P(u, v)$ from node u to node v can be defined as an ordered set of links, $P(u, v) = \{(u, i), (i, j), \ldots, (k, v)\}$.

A multicast tree $T(s, R)$ is a tree rooted at the source s, spanning all destinations $r_i \in R$. We denote $P_T(r_i) \subseteq T$ as the path from the source s to a destination $r_i \in R$ in the multicast tree T. The delay of the path from s to a destination r_i, denoted by $Delay(r_i)$, can then be defined as the sum of the delays on all links along the paths $P_T(r_i)$:

$$Delay(r_i) = \sum_{e \in P_T(r_i)} D(e) \tag{1}$$

The delay of the overall multicast tree $T(s, R)$, denoted by $Delay(T)$, is the maximum delay among all the paths $P_T(r_i)$, $r_i \in R$:

$$Delay(T) = \max\{Delay(r_i) \mid \forall\, r_i \in R\} \tag{2}$$

The total cost of the multicast tree, denoted by $Cost(T)$, is the sum of the costs of all links on the paths in the multicast tree:

$$Cost(T) = \sum_{e \in T} C(e) \tag{3}$$

In real time computer network applications, different delay bounds δ_{ri} may exist for paths to different destinations $r_i \in R$. In DCLC multicast routing problems, the delay bound defines the upper bound to the sum of delays on all links along the path from the source s to each destination $r_i \in R$. In this paper and the other related work reviewed in Section 2.4, it is assumed that the paths to all destinations have the same upper bound, denoted by $\Delta = \delta_{ri}$, $r_i \in R$.

Given the above definitions, the Delay-Constrained Least Cost (DCLC) multicast routing problem can be formally defined as follows (Guo and Matta, 2000):

The Delay-Constrained Least Cost (DCLC) Multicast Routing Problem: Given a network G, a source node s, a set of destination nodes $r_i \in R$, a link delay function $D(\cdot)$, a link cost function $C(\cdot)$, and a delay bound Δ, the DCLC multicast routing problem is to construct a multicast tree $T(s, R)$ such that the delay bound of the path is satisfied and the tree cost $Cost(T)$ is minimized. We can define the objective function of the DCLC multicast routing problem as follows:

$$\text{Minimize } \{Cost(T) \mid T \in T(s, R)\} \text{ s.t. } Delay(r_i) \leq \Delta, \ \forall r_i \in R \tag{4}$$

2.2 The Steiner Tree Problem

The Steiner tree problem is the underlying model of the DCLC multicast routing problem and is defined by an undirected graph $G = (V, E)$, where V is a set of nodes and E is a set of links, respectively. Each link $e(i, j) \in E$ linking node i and node j is associated with a weight $W(e)$: $E \mapsto R^+$. The triangle inequality holds in Euclidean metric spaces, therefore $W(e_1) + W(e_2) \geq W(i, k)$, $e_1 = (i, j) \in E$, $e_2 = (j, k) \in E$, $\forall i, j, k \in V$. Nodes in V can be partitioned into a set of required destination nodes R and the remaining nodes S, $V = R \cup S$. Then, the metric Steiner tree problem can be defined as follows:

The Steiner Tree Problem is to find a minimum weighted tree T in G, $T \subseteq E$, that spans all nodes in R and if necessary some additional nodes (the so called Steiner nodes) in S. The total weight $W(T)$ of the tree is the sum of the weights of all links in the tree T, i.e. $W(T) = \sum_{e \in T} W(e)$

A collection of well known Steiner tree problems have been maintained in the OR library at http://people.brunel.ac.uk/~mastjjb/jeb/orlib/steininfo.html, originally published in Beasley (1990). During the years, these benchmark problems have been tested in the OR community by various meta-heuristics and exact methods. Branch and cut methods with pre-processing, reduction techniques and primal heuristics have been used to solve these instances to optimality (Koch and Martin, 1998). Such techniques have also been widely studied on other variants of the Steiner tree problems (Barahona and Ladanyi, 2006; Costa, Cordeau and Laporte, 2006).

The Steiner tree problem and its variants have also been widely tested by PSO algorithms in the literature. Consoli et al. (2010) present a JPSO for the minimum labeling Steiner tree problem, where the objective is to find a spanning tree with the smallest number of distinct labels on links, covering a given subset of nodes. The algorithm outperforms an exact method, a pilot method and a multi-start approach with and without local search, finding high quality solutions with short running times. Apart from the JPSO by Consoli et al. (2010), the only other discrete PSO algorithm (for which explores in discrete search space for solving combinatorial optimization problems) applied to the Steiner tree problems that we are aware of is developed by Zhong et al. (2008). A complete graph is firstly created by using the Floyd's algorithm (Floyd, 1962). A modified Prim's algorithm is used to re-create the minimal spanning tree, and a trimming operation is used to cut off redundant nodes. The algorithm compares well to a

genetic algorithm (GA) on both the *B* and *C* datasets in the OR library. We compare our JPSO algorithm with Zhong et al. (2008) on the same datasets in Section 5. Other search algorithms developed for the Steiner tree problem can be found in the review in Zachariasen (1999).

To our knowledge, there is no efficient exact method in the literature to solve the delay-bounded multicast routing problem. Existing related work is only limited to heuristic and meta-heuristic approaches. Our aim here is to demonstrate that the proposed JPSO can not only be applied to solve the delay-constrained least cost multicast routing problems, but also outperforms existing meta-heuristics on solving some of the widely tested benchmark Steiner tree problems. Experimental evaluations and comparisons to existing results on both of the benchmark problems in the literature provide scientific justification of the JPSO algorithms proposed in our work.

2.3 Particle Swarm Optimization for Different Multicast Routing Problems

A number of PSO algorithms have been developed in the literature for solving a range of multicast routing problems with different constraints and features. These interesting multicast routing problems are very different from the DCLC multicast routing problems considered in this work, and will be the subject of our future work.

Yuan et al. (2004) formulate the energy-aware multicast routing problem in wireless ad-hoc networks using an integer linear programming model, and apply a multi-phase discrete PSO to find optimal solutions. Total transmission power is minimized with connection and broadcast constraints. A symbiosis mechanism is used to handle the constraints and to allow both feasible and infeasible particles to evolve in the swarm. Sun et al. (2006) have considered a multi-objective multicast routing problem with a number of constraints including delay, bandwidth, cost, delay jitter, and packet losses. The particles in their PSO algorithm use an integer coding that associates a position with a list of nodes from the source to the destination. The PSO outperforms a GA on a small problem of 23 nodes.

In Wang et al. (2005) a hybrid approach between a GA and a PSO has been developed to minimize the tree cost in multicast routing problems with a number of constraints including the bandwidth, delay, and error rate in non-deterministic scenarios. The hybrid algorithm outperforms a GA on both the tree cost and the convergence rate. Another hybrid approach between GA and PSO developed in Li et al. (2007) has shown to outperform a standard GA on random multicast routing problems with two objectives, to minimize the average delay and the link utilization (traffic vs. link capacity). The best half of the GA population is firstly improved and then used as the starting positions for the particles in PSO.

2.4 Heuristics and Meta-heuristics for DCLC Multicast Routing Problems

Our proposed JPSO algorithm and a large number of algorithms in the literature belong to the class of source-based approaches, where each node in the multicast routing problem has all the necessary information to construct the multicast tree. As opposed to source-based approaches, destination-based approaches do not require that each node maintains the status information of the entire network, and multiple nodes participate in constructing the multicast tree. Later in experiment simulations we compare the performance of our proposed JPSO to all the algorithms reviewed in this section on a large number of DCLC multicast routing test instances.

A number of early source-based heuristics developed for constructing low-cost multicast trees with delay bounds are based on the well known Prim's shortest path heuristic (Prim, 1957; Cormen et al., 2001) and the *k*-shortest path algorithm (Eppstein, 1998). The first source-based heuristic for DCLC multicast routing problems is the Kompella-Pasquale-Polyzos (KPP) heuristic (Kompella et al., 1993), which uses the Prim's algorithm to obtain a minimum spanning tree with constrained paths, assuming that the link delays and delay bounds are integers. The Bounded Shortest Multicast Algorithm (BSMA) (Zhu et al., 1995), one of the best known delay-bounded multicast routing algorithms developed in the 1990s, iteratively refines the tree to lower costs based on the *k*-shortest path algorithm. Due to its good performance on tree cost, it is still being frequently used to compare the performance of many recent multicast routing algorithms. Although these early heuristics have very good performance with respect to

the tree cost, based on the Prim's algorithm or the *k*-shortest path algorithm, their computing times are usually extremely high for larger networks.

A large amount of recent research has been carried out to develop meta-heuristics for different large DCLC multicast routing problems. Several tabu search algorithms have been developed, where Dijkstra's algorithm has been widely used as the initialization method (Youssef et al., 2002; Skorin-Kapov and Kos, 2006; Ghaboosi and Haghighat, 2007a). Wang et al. (2004) find that the main disadvantage in the tabu mechanism is that the randomly selected paths often lead to a disjointed multicast tree. In Ghaboosi and Haghighat (2007a), initial solutions are iteratively refined by using a modified Prim's algorithm to switch links chosen from a backup path set. A candidate list strategy is used to intelligently select neighborhood moves and show to speed up the search and also improve the solution quality. In Skorin-Kapov and Kos (2006), a tabu search is applied to improve the solution quality within GRASP approach (Feo and Resende, 1995). The algorithm outperforms the KPP algorithm (Kompella et al., 1993) and a tabu search (Skorin-Kapov and Kos, 2003) on the tested problems.

A number of population based algorithms have also been developed, including genetic algorithms (Haghighat et al., 2004; Wang et al., 2001). Ghaboosi and Haghighat (2007b) develop a path relinking algorithm where the worst solutions in a reference set of random solutions are iteratively replaced using a path relinking process. Simulation results show that the path relinking algorithm outperforms other existing algorithms with respect to the tree cost. However, repairing infeasible solutions generated during the path relinking is time consuming as many infeasible solutions occur when the network size increases.

In our previous work (Qu et al., 2009), a variable neighborhood search algorithm has been developed for the DCLC multicast routing problem. Three neighborhoods are designed by using path replacement operators to iteratively replace high cost links in the tree. The algorithm outperforms a number of algorithms in the literature in terms of both the computing time and the tree cost. It is observed that the neighborhood design plays a crucial role in the performance of the algorithm, and effective initialization method leads to better final solutions within a shorter computing time. In Xu and Qu (2012), the analysis on the fitness landscape of the DCLC multicast routing problem further demonstrates that the DCLD problem is highly instance dependant, thus demands advanced robust algorithms concerning the complex constraints in the problem.

3. Variants of the Particle Swarm Optimization

3.1 The Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a bio-inspired population-based stochastic global optimization algorithm proposed by Kennedy and Eberhart (1995). It belongs to the class of swarm intelligence algorithms (Bonabeau, Theraulaz and Dorigo 1999; Eberhart, Shi and Kennedy, 2001). PSO simulates simplified natural social systems such as flocks of birds or schools of fish. A population (swarm) is made up of simple agents (particles) and evolves by following very simple rules in a decentralized way. Particles are typically modeled as entities moving in a multi-dimensional continuous space (search space) and interact by sharing both local and global information about their own positions (solutions). During the evolution (over iterations), the whole swarm evolves and complex behaviors emerge. For each particle i in the swarm at iteration j, its position (solution) $x_{i,j}$ and velocity (rate of change) $v_{i,j}$ are updated in the evolution by using the following two equations:

$$v_{i,j+1} = c_0 v_{i,j} + c_1 r_1 (b_i - x_{i,j}) + c_2 r_2 (g_j - x_{i,j}) + c_3 r_3 (g_{i,j} - x_{i,j}) \quad (5)$$

$$x_{i,j+1} = x_{i,j} + v_{i,j+1} \quad (6)$$

Equation (1) handles the velocity update by summing up four components:

- the first component, $c_0 v_{i,j}$, is called inertia and enables the particle to keep the flow of its previous movement, avoiding abrupt moves and premature convergence;

- the second component, $c_1r_1(b_i - x_{i,j})$, encourages the particle's self-learning (cognition) ability by using its best position achieved so far, b_i, as a reference;
- the third component, $c_2r_2(g_j - x_{i,j})$, is the social factor that leads the particle to the best position so far within the swarm, g_j, which remembers the best performance so far in the entire swarm;
- the fourth component, $c_3r_3(g_{i,j} - x_{i,j})$, uses the best location found so far by the particles belonging to a "social neighborhood" of the particle, i.e. in a neighborhood sub-swarm $g_{i,j}$. This is to enhance the exploration capacity of the particles and also to prevent premature convergence within the swarm.

Each of the four components in Equation (1) is associated with a weight c_x, $x = 0, \ldots, 3$, and $c_x \in [0, 1]$, to establish the importance of each component. Moreover, in order to confer a stochastic behavior, the latter three components are scaled by random values $R = (r_1, r_2, r_3)$ drawn from an uniform [0,1] distribution. Once the new velocity $v_{i,j+1}$ is calculated using Equation (1), Equation (2) adds the new velocity to the current position $x_{i,j}$ so that the particle moves to a new position.

PSO is relatively easy to understand and to implement. A number of variants exist in the literature and have been highly successful on a range of problems (see more details in Eberhart, Shi and Kennedy, 2001). PSO variants remain as our future work based on a good understanding of the present work in this paper. More information about developments of PSO can be found in (Kennedy and Eberhart, 1995) and at http://www.swarmintelligence.org/.

3.2 The Jumping Particle Swarm Optimization

While the original PSO algorithm has been designed for continuous optimization problems, a variant called Discrete Particle Swarm Optimization (DPSO) has been designed to deal with combinatorial optimization problems, where particles move in a multi-dimensional discrete exploration space. The first DPSO approach was introduced by Kennedy and Eberhart (1997), where the positions of the particles are encoded by using binary strings, while the velocity equation remained unchanged. By using a sigmoid function, the velocity is mapped to a value in [0, 1]. If a quasi-random number sampled from a uniform distribution between [0, 1] is greater than the mapped velocity, the position takes value 0, otherwise it takes value 1. Since then, many variations of DPSO have been proposed and tested on a range of problems including the travelling salesman problem (Onwubolu and Clerc, 2004), production scheduling problems (Allahverdi and Al-Anzi, 2006; Sha and Hsu, 2006; Tasgetiren, 2007; Anghinolfi and Paolucci, 2009), and resource constrained project scheduling (Zhang et al., 2006).

A particular DPSO strategy called Jumping Particle Swarm Optimization (JPSO) algorithm has recently been introduced by Moreno-Perez et al. (2007) to solve combinatorial optimization problems. Later, it has been used by Consoli et al. (2008) to tackle the minimum labeling Steiner tree problem and Castro et al. (2009) to deal with the vehicle routing problem with time windows. The JPSO algorithm does not use the concept of velocity to redefine how the swarm of particles moves in the search space. Instead, the metaphor behind JPSO is that of a number of elements (particles) moving (jumping) from position to position (solutions) in a discrete search space. If there is a particle with a good fitness in a certain region of the space, the other particles in the swarm will be attracted to its position in order to improve their own fitness.

In the original PSO, the four components in Equation (1) can be split to two parts. The first part $c_0v_{i,j}$ enables the particle to continue the exploration of its current position. The second part $c_1r_1(b_i - x_{i,j}) + c_2r_2(g_j - x_{i,j}) + c_3r_3(g_{i,j} - x_{i,j})$ encourages the particle to move towards a better location with respect to other particles' positions. Therefore, the particle that performs the move *follows* the other three different *attractors*, and is thus named the *follower* in the literature.

Each of the attractors has a likelihood c_x given by a weight vector, $\sum c_x = 1$, $x = 0, \ldots, 3$. In JPSO, only one type of moves is triggered at each iteration. When a stopping criterion has not been met, the algorithm generates a random number r uniformly distributed in [0, 1]. This number r along with the weights c_x determines how the particle moves as follows:

- If $r \in [0, c_0)$, the particle continues its current exploration (an inertial move);

- If $r \in [c_0, c_0+c_1)$, the follower particle will move towards (be attracted to) the attractor b_i, which is its own best position achieved so far (a cognitive move);
- If $r \in [c_0+c_1, c_0+c_1+c_2)$, the follower particle will move towards the attractor g_j, the best position in the swarm so far (a social move);
- If $r \in [c_0+c_1+c_2, c_0+c_1+c_2+c_3 = 1]$, the follower particle will move towards the attractor $g_{i,j}$, the best positioned particle in its neighborhood sub-swarm at the current iteration (a global move).

In the end, there are four types of moves, one from each component in the original formula (1). The swarm in JPSO *jumps* through the discrete space by following one of these four types of moves. One of the main advantages of JPSO is that it retains the simplicity of the original PSO but works on a discrete search space. Different components or techniques may be integrated in JPSO to improve the efficacy of the particles' movements. In the case of an inertial move, a mutation or neighborhood operator can be employed to explore the current position and to prevent premature convergence. For the other three follower-attractor moves, crossover operators may be used to partially imitate the structure of the attractors. This makes JPSO algorithm a good option for solving complex combinatorial optimization problems, and motivated our work on both Steiner tree and multicast routing problems in this paper.

4. The Proposed JPSO Algorithm

4.1 The Overall JPSO Procedure

The pseudo-code of our proposed JPSO algorithm (hereinafter named JPSOMR) for solving both the multicast routing problems and the Steiner tree problems is presented in Figure 1. A swarm of random particles (randomly generated trees) is firstly created. Starting from the source node, a random tree is constructed by randomly selecting the next link which connects to any on-tree node until all destination nodes have been added to the tree.

As explained above, a particle in the swarm of JPSOMR does not possess a velocity component. Instead, the swarm evolves based on different moves to the positions (solutions) of the particles. At every iteration of the evolution, each particle moves either based on its current position (an inertial move) or based on the position of the attractor which is chosen by using the weight vector (a cognitive, social or global move). Once the particle has jumped to a new position, a local search is applied. The particle's best position and the swarm's best position are then updated. The process is repeated until a stopping condition is met, and the best position obtained by the swarm is returned as the final solution after the evolution.

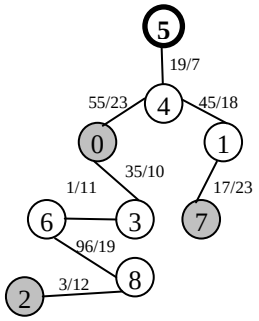
```
JPSOMR(G = (V, E), s, R, Δ, c0, c1, c2, c3)
{   // s: the source node; R: the destination set; Δ ≥ 0: the delay bound; cx: the weight vector;
    // p.c: the current position of the particle; p.n: the new position of the particle;
    // p.bn: the best neighboring position of the current particle;
    // p.b: the best position found by the particle; g: the global best position found by the swarm.

    Create |P| random initial feasible solutions for all particles in the swarm P
    while (stopping condition not met) do
        for each(particle p in swarm P) do
            Generate a random number r ∈ (0, 1]
            case (r) {                                          // moves, see section 4.4
                r is in c0: p.n = RandomMove(p.c);              // inertial move
                r is in c1: p.bn = GetBestNeighborhood(p);      // cognitive move
                            p.n = PathReplacement(p.bn, p.c);
                r is in c2: p.n = PathReplacement(p.b, p.c);    // social move
                r is in c3: p.n = PathReplacement(g, p.c);      // global move
            }
            Greedy or first improvement local search on the new position p.n   // see section 4.5
            Calculate the Cost and Delay of the new position p.n
            if (((Cost(p.n) < Cost(p.b)) and (Delay(p.n) < Δ)) or
                (Cost(p.n) == Cost(p.b)) and (Delay(p.n) < Delay(p.b)))
            then p.b = p.n;                                     // update the particle's best position
            if (((Cost(p.n) < Cost(g)) and (Delay(p.n) < Δ)) or
                (Cost(p.n) == Cost(g)) and (Delay(p.n) < Delay(g)))
            then g = p.n;                                       // update the swarm's best position
            p.c = p.n;
        end for
    end while
    return g;
}
```

RandomMove(*p*): a procedure that moves the particle *p* to a new position by choosing a *superpath* in the current position and replacing it by a random new path.
GetBestNeighborhood(*p*): a procedure that returns the best position among the neighborhood particles of *p*.
PathReplacement(*attractor*, *follower*): a procedure that replaces the path from the source *s* to a destination in the *follower* by choosing the best path in the *attractor*. See section 5.3.

Figure 1. The pseudo-code of the JPSOMR algorithm.

4.2 The Representation

In our JPSOMR algorithm, the tree is represented by using a predecessor array with $|V| = n$ elements corresponding to the n nodes in the tree. The value of each element in the array is set to the index of the node's predecessor. Figure 2 presents an example of the representation of a tree, where the array (4-4-8-0-5-*x*-3-1-6) represents the predecessor node of each corresponding node (node 0 to node 8) in the tree.



the predecessor array:	4	4	8	0	5	*x*	3	1	6
corresponding nodes:	0	1	2	3	4	5	6	7	8

Figure 2. An illustrative example of the representation of the tree, with three *superpaths*: (5-4), (4-0-3-6-8-2) and (4-1-7). Values "*a*/*b*" denotes "cost/delay" of the links in the tree. Shaded nodes are the destination nodes, and node 5 is the source node (with no predecessor node, indicated by *x*).

4.3 The Path Replacement Operator

The path replacement operator is used to update a particle's position based on that of a chosen attractor. Figure 3 shows three trees corresponding to a follower particle T_0, a chosen attractor particle T_a and the newly generated particle T'. The path replacement operator firstly finds the cheapest path (5-4-1-7-8-2) in the attractor tree T_a, and then uses it to replace the corresponding path (5-4-0-3-6-8-2) with the same destination node 0 in the current particle T_0. This newly generated tree T' becomes the new position of the follower particle.

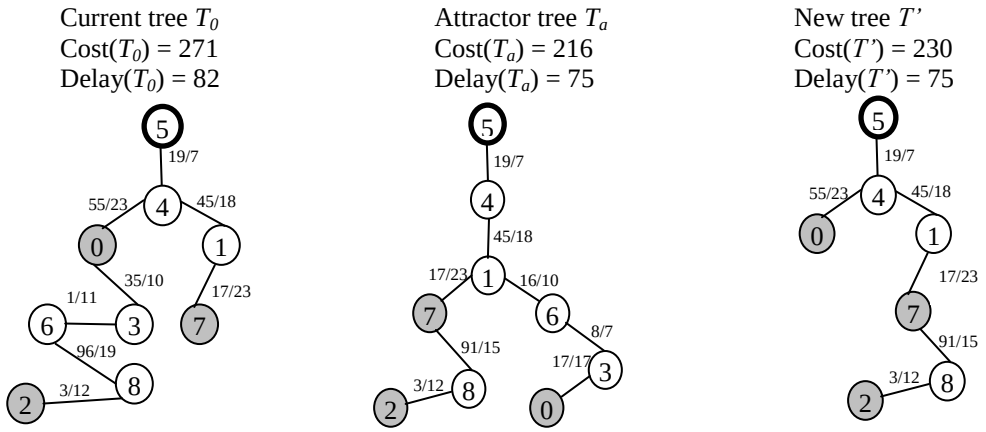


Figure 3. The path replacement operator replaces the path (5-4-0-3-6-8-2) in T_0 by using the cheapest path (5-4-1-7-8-2) in T_a to create a new tree T'. Node 5 is the source node, and shaded nodes are destination nodes. The delay bound $\Delta = 82ms$. Values "a/b" denotes "cost/delay" of the links in the tree.

In the path replacement operation, the selected path (to the same destination node of the cheapest path in the attractor) is firstly removed from the follower particle. If a path $P_T(r_i)$ to another destination r_i is included in the selected path, this path $P_T(r_i)$ will not be removed from the follower particle tree. For example, in the follower particle tree T_0 in Figure 3, the path (5-4-0) to another destination node 0 within the selected path (5-4-0-3-6-8-2) will remain in the tree. To avoid cycles in the generated tree, the path replacement operator always replaces the old path by starting from the destination node of the new path until the new path connects to an on-tree node. In Figure 3, the new path (5-4-1-7-8-2) in the attractor particle tree T_a is added to follower T_0 by starting from node 2, and adding node 8 until it connects to the on-tree node 7.

For the DCLC multicast routing problem, the key constraint, the delay bound Δ, restricts the generation of the multicast trees. The smaller the delay bound, the tighter the problem is constrained. The delay of path is always checked while implementing the path replacement to guarantee that the delay bound is satisfied in the newly generated tree.

4.4 Moves of Particles

In JPSOMR, there are two types of moves: moves towards an attractor and moves around the current position (no attractor involved). Depending on where the sampled random value r falls within the intervals defined by the weight vector c_x, a specific attractor (or none) is selected to influence how the particle moves from its current position to a new position. The particle moves as follows:

- If $r \in [0, c_0)$, no attractor is selected, thus the particle moves around its current position. This is done by randomly removing a *superpath* in the current tree, and reconnecting the resulting two sub-trees by using a random link.

  As in our previous work in (Qu et al., 2009), *superpath* has been used in operations within the inertial particle moves to reduce the tree cost (see Sections 4.4 below). The *superpath* is the longest simple path between two end nodes in the tree, where all internal nodes, except the two end nodes of

the path, have a node degree of 2. In the tree presented in Figure 2, there are three *superpaths* which may be involved in the inertial particle moves.

- If $r \in [c_0, c_0+c_1)$, the best position achieved by the particle so far (b_i) is chosen as the attractor. If $r \in [c_0+c_1, c_0+c_1+c_2)$, the attractor is the best position achieved by the whole swarm so far (g_j). If $r \in [c_0+c_1+c_2, 1]$, the best located particle in the neighborhood of the current particle ($g_{i,j}$) acts as the attractor.

  Once the attractor is selected, the path replacement operator adds the cheapest path from the source to the destination in the selected attractor and removes the corresponding path in the follower (see Section 4.3 above). The added path should not be already in the follower particle; otherwise a random move is applied to the follower particle.

Based on the predecessor array representation, the movements of the particles are implemented by replacing the predecessors of the nodes in the original path by the nodes in the new selected link/path.

4.5 Local Search Heuristics

After each move, a local search is applied to improve the new particle's position. In the local search implemented here, a simple neighborhood operator operates upon the nodes in the tree. A neighbor of the current tree is obtained by removing a non-destination node and creating a new spanning tree of the remaining nodes using the Prim's spanning tree algorithm (Betsekas and Gallager, 1992). Two variants of local search have been tested in our JPSOMR algorithm. The first local search uses a *greedy* heuristic to select the best neighbor from all neighboring solutions of the current tree. The second uses a *first improvement* heuristic, where the first improving neighbor solution is selected.

If the newly generated particle after the local search corresponds to a better tree, this new particle replaces its best position and/or the best global position so far. A tree is seen as better if it has a lower cost and satisfies the delay bound constraint, or it has the same cost and with a smaller delay. Therefore, the particle in JPSOMR finishes its jump by updating its best position and the best global position so far.

5. Performance Evaluation

5.1 Simulation Environment for Steiner Tree and DCLC Multicast Problems

We implement and evaluate our JPSOMR algorithm by using the multicast routing problem simulator (MRSIM), which is adopted based on the generator developed by Salama et al. (1997). The simulator generates random network topologies by using a graph generation algorithm (Waxman, 1988).

For the DCLC multicast routing problems, the link delay function $D(e)$ is defined within the simulator as the propagation delay of the link. We assume that queuing and transmission delays are negligible. The link cost function $C(e)$ is defined as the current total bandwidth on the links in the computer network. The network nodes are randomly positioned over a simulated rectangular area of size 4000×4000 km^2. The Euclidean metric is used to determine the distance $l(u, v)$ between pairs of connected nodes (u, v). Within the simulator, links $e = (u, v)$ connecting nodes u and v are placed with a probability $P_{u,v}$ given by:

$$P_{u,v} = \beta\, e^{-l(u, v) / \alpha L} \qquad \alpha, \beta \in (0,1] \qquad (7)$$

L in (7) is the upper bound of the distance between two nodes in the network. The parameters α and β are used to generate different DCLC multicast routing problems within the desired networks of a range of characteristics. For example, setting a large β value gives nodes a higher average degree, and setting a large α value gives short distances between nodes. More details can be found in Salama et al. (1997). In our simulations, we set $\alpha = 0.25$ and $\beta = 0.40$. Different values of the delay bound Δ are set in our tests as reported below.

The simulator not only produces a wide range of network topologies with different characteristics, but also provides a unified framework for fair and sound comparisons. As mentioned in Section 2.4, in the literature different algorithms have been developed and tested using different datasets of DCLC multicast routing problems. Simulations have been run on different platforms thus making it difficult to run fair comparisons. To warrant a fair comparison, the JPSOMR algorithm and the selected competitor multicast algorithms from the literature have been re-implemented in this work within the same simulation environment and compared on exactly the same set of simulated DCLC multicast routing problems instances generated in the simulator.

For the Steiner tree problem, we selected two sets of problem instances from the OR library, focusing on instance sets *B* and *C*, and used them to test our algorithm and the existing algorithms in the literature. Although a large number of new instances have been added to the library later (Koch et al., 2002), during the years, these benchmark datasets still serve as the most widely tested problems in the OR community and motivated the development of meta-heuristic algorithms. Since in the problems the links are assigned only a cost function, we generated their delay values randomly in the simulator. The delay bound is set as infinity in the simulator so no delay restriction is enforced, i.e. the generated network is a true Steiner tree problem. All our simulations have been run 30 times for the small category *B* instances and 20 times on the larger category *C* instances.

All experiments have been run on a Windows XP PC with an Intel Core 2 Duo E8500 3.16GHZ processor and 8GB of RAM. In addition to reporting the comparison results on the solution quality and the computing time from of our proposed JPSOMR algorithm in this paper, to encourage scientific comparisons, we also provide the details of all the problem instances tested and the experimental results at http://www.cs.nott.ac.uk/~rxq/benchmarks.htm.

5.2 Experimental Results on Steiner Tree Problems

We first evaluate our JPSOMR algorithm on the benchmark categories *B* and *C* Steiner tree problems in the OR library. The category *B* instances are based on networks with 50, 75 and 100 nodes with 63 to 200 links, and the category *C* instances include a set of larger networks with 500 nodes with 625 to 12500 links. Details of their characteristics are given in Tables 1 and 2. The optimal solutions have been obtained in (Beasley, 1990) by incorporating the lower bound and problem reduction tests derived from the original problems within a tree search. Due to the different computing platform used to run this exact method, the computing time to find the optimal solution is not provided here as a reference. More details can be found in Beasley (1990).

Table 1. The characteristics of category *B* instances from the OR-library. |*V*|: the number of nodes; |*E*|: the number of links; |*R*|: the number of destinations in the instances; Opt.: the cost of the optimal solution.

	|*V*|	|*E*|	|*R*|	Opt.		|*V*|	|*E*|	|*R*|	Opt.		|*V*|	|*E*|	|*R*|	Opt.
B01	50	63	9	**82**	***B07***	75	94	13	**111**	***B13***	100	125	17	**165**
B02	50	63	13	**83**	***B08***	75	94	19	**104**	***B14***	100	125	25	**235**
B03	50	63	25	**138**	***B09***	75	94	38	**220**	***B15***	100	125	50	**318**
B04	50	100	9	**59**	***B10***	75	150	13	**86**	***B16***	100	200	17	**127**
B05	50	100	13	**61**	***B11***	75	150	19	**88**	***B17***	100	200	25	**131**
B06	50	100	25	**122**	***B12***	75	150	38	**174**	***B18***	100	200	50	**218**

Table 2. The characteristics of category *C* instances from the OR-library. |*V*|: the number of nodes; |*E*|: the number of links; |*R*|: the number of destinations in the instances; Opt.: the cost of the optimal solution.

No.	**|*V*|**	**|*E*|**	**|*R*|**	**Opt.**	**No.**	**|*V*|**	**|*E*|**	**|*R*|**	**Opt.**	**No.**	**|*V*|**	**|*E*|**	**|*R*|**	**Opt.**	**No.**	**|*V*|**	**|*E*|**	**|*R*|**	**Opt.**
C01	500	625	5	**85**	***C06***	500	1000	5	**55**	***C11***	500	2500	5	**32**	***C16***	500	12500	5	**11**
C02	500	625	10	**144**	***C07***	500	1000	10	**102**	***C12***	500	2500	10	**46**	***C17***	500	12500	10	**18**
C03	500	625	83	**754**	***C08***	500	1000	83	**509**	***C13***	500	2500	83	**258**	***C18***	500	12500	83	**113**
C04	500	625	125	**1079**	***C09***	500	1000	125	**707**	***C14***	500	2500	125	**323**	***C19***	500	12500	125	**146**
C05	500	625	250	**1579**	***C10***	500	1000	250	**1093**	***C15***	500	2500	250	**556**	***C20***	500	12500	250	**267**

5.2.1 The Size of the Swarm

To evaluate the impact of the swarm size in JPSOMR, in the first group of experiments, we compare JPSOMR with 6 different swarm sizes ($|P|$ = 1, 2, 5, 10, 20, 30) on the smaller category *B* Steiner tree instances. Note that the special case of when $|P|$ = 1 may be seen as a local search rather than a PSO, and thus this experiment also compares the performance of local search against PSO with different swarm sizes. Weight vector is set as $c_0 = c_1 = c_2 = c_3 = 0.25$, and the number of iterations in the evolution is set to 100. No local search is applied to obtain an unbiased view of the impact of the swarm size. The average tree costs are given in Table 3.

Table 3. JPSOMR with different swarm sizes and no local search. The best results are presented in bold. Opt.: the optimal values for each instance; $\Delta cost$ = (*cost* - Opt.)/Opt.; δ^* = the relative deviation δ to the optimum, i.e. δ/Opt. Best results are in bold (as in all other tables).

Prob.	**Opt.**	**$\|P\|$ = 1**		**$\|P\|$ = 2**		**$\|P\|$ = 5**		**$\|P\|$ = 10**		**$\|P\|$ = 20**		**$\|P\|$ = 30**	
		$\Delta cost$	δ^*	$\Delta cost$	δ^*	$\Delta cost$	δ^*	$\Delta cost$	δ^*	$\Delta cost$	δ^*	$\Delta cost$	δ^*
B01	**82**	0.172	0.194	0.052	0.075	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
B02	**83**	0.420	0.172	0.176	0.090	0.113	0.062	0.099	0.041	**0.084**	**0**	**0.084**	**0**
B03	**138**	0.265	0.069	0.192	0.056	0.171	0.040	0.149	0.004	**0.110**	**0.039**	0.118	0.026
B04	**59**	0.693	0.183	0.505	0.069	0.439	0.043	0.383	0.055	0.305	0.047	**0.266**	**0.058**
B05	**61**	0.415	0.200	0.249	0.132	0.100	0.046	0.090	0.045	0.062	0.019	**0.049**	**0.013**
B06	**122**	0.501	0.150	0.290	0.043	0.244	0.039	0.203	0.024	0.191	0.032	**0.175**	**0.031**
B07	**111**	0.303	0.172	0.233	0.109	0.039	0.035	0.076	0.031	0.025	0.013	**0.016**	**0.013**
B08	**104**	0.418	0.182	0.194	0.102	0.188	0.096	0.090	0.033	**0.085**	**0.019**	0.091	0.020
B09	**220**	0.153	0.072	0.105	0.048	0.055	0.009	0.059	0.008	0.050	0.007	**0.045**	**0**
B10	**86**	0.893	0.197	0.573	0.170	0.605	0.118	0.437	0.035	**0.424**	**0.007**	**0.424**	**0.010**
B11	**88**	1.013	0.203	0.750	0.112	0.681	0.071	0.591	0.023	0.563	0.013	**0.522**	**0.029**
B12	**174**	0.611	0.128	0.447	0.114	0.343	0.056	0.317	0.032	**0.289**	**0.014**	0.301	0.010
B13	**165**	0.356	0.112	0.253	0.098	0.112	0.046	0.088	0.031	0.069	0.043	**0.051**	**0.015**
B14	**235**	0.219	0.032	0.196	0.032	0.171	0.019	0.157	0.019	0.142	0.016	**0.129**	**0.028**
B15	**318**	0.173	0.045	0.147	0.038	0.112	0.009	0.108	0.006	0.100	0.008	**0.097**	**0.011**
B16	**127**	0.776	0.202	0.461	0.145	0.422	0.105	0.276	0	**0.266**	**0.018**	0.276	0
B17	**131**	0.858	0.277	0.466	0.343	0.163	0.097	0.092	0.016	**0.092**	**0.005**	**0.092**	**0**
B18	**218**	0.582	0.166	0.416	0.139	0.249	0.011	0.250	0.013	0.234	0	**0.231**	**0.004**

We carried out the paired *t*-test (Montgomery, 2005) to analyze the statistical difference between JPSOMR with different swarm sizes. If the *p*-value obtained from the *t*-test is smaller than 0.05, comparison results are usually referred as significantly different. This gives a better insight, compared to the average results, in justifying statistical difference between algorithm performances. The *p*-value on results of JPSOMR with $|P|$ = 10 and $|P|$ = 20 is 2.3E-6, indicating that JPSOMR with $|P|$ = 20 is significantly better than that of with $|P|$ = 10. However, the *p*-value of JPSOMR with $|P|$ = 20 and $|P|$ = 30 is 0.05, indicating that JPSOMR with $|P|$ = 20 and $|P|$ = 30 show no significant differences. Since the computing time of JPSOMR with $|P|$ = 20 is much less than that of $|P|$ = 30 (see Figure 4), we use $|P|$ = 20 as an appropriate swarm size for our problems.

The average computing time is presented Figure 4, clearly showing the increasing computing time for larger swarm size in JPSOMR. It can be seen that instance *B13* requires much less time compared to other instances, indicating that difficult Steiner tree problems are not necessarily of large size. Across all other instances, computing times are stable with small standard deviations in JPSOMR with larger swarm sizes $|P|$ = 20 and 30.

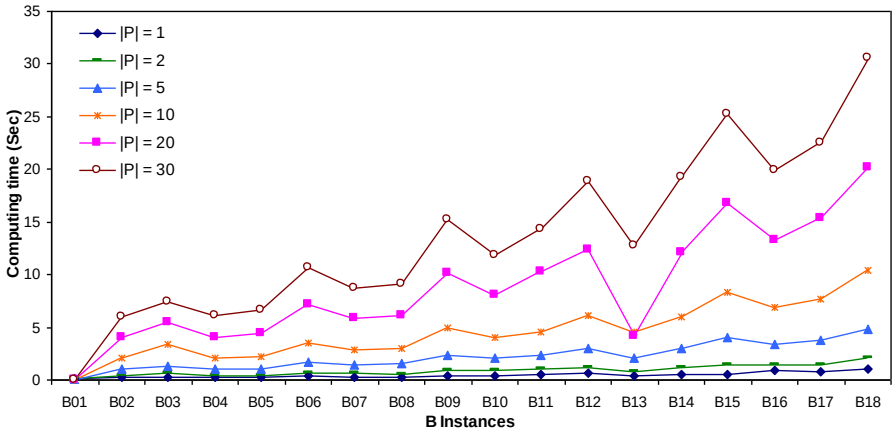


Figure 4. Computing time of JPSOMR of different swarm sizes without local search on small category *B* instances

Additional tests that we ran using JPSOMR and the greedy local search showed that, although no obvious difference has been found on small instances, JPSOMR with population size of $|P| = 20$ performed the best (compared against $|P| = 30$) on large instances. Although more particles are helpful for obtaining good results, a too large swarm actually hinders the algorithm's performance.

5.2.2 The Cooperation between Particles

A key feature of PSO is that during the search, particles in the swarm cooperate by sharing information of local or global better positions in order to better explore the search space. Our second set of experiments is conducted to find out if this is actually happening in our JPSOMR. Recall that the values in the weight vector $c_x = (c_0, c_1, c_2, c_3)$ determine the extent to which a particle updates its position based on its own behavior or that of the other attractor particles in the swarm. For example, setting $c_x = (1, 0, 0, 0)$ means that the particles do not cooperate but make their next move based on only their own previous positions. The JPSOMR is thus equivalent to a multi-start local search (since a local search is applied at the end of each particle move), where each particle carries out its very own search by inertial movements. When all c_0, c_1, c_2, c_3 take values which are different from zero, particles cooperate by using both inertial and attractor movements within a true JPSOMR process.

Based on the observation from the first set of experiments, we set the swarm size $|P| = 20$, and compared JPSOMR with two different weight vectors ($c_0 = 1$, $c_1 = c_2 = c_3 = 0$) and ($c_0 = c_1 = c_2 = c_3 = 0.25$), and with two different local search strategies. For a fair comparison, we set the same computing time for all the four variants of JPSOMR, namely 30 seconds for small instances *B01-B12* and 120 seconds for large instances *B13-B18*. The average tree costs by JPSOMR variants on the category *B* instances are given in Table 4.

Table 4. JPSOMR using different settings of cooperation between particles and different local search strategies. The best results are presented in bold. Opt.: the optimal values for each instance; LS1: first improvement local search; LS2: greedy local search; Δ*cost* = (*cost* - Opt.)/Opt. Computing time = 30 seconds for *B01-B12*, 120 seconds for *B13-B18*.

$\|P\| = 20$		Average Tree Δ*cost*				$\|P\| = 20$		Average Tree Δ*cost*			
		$c_0 = 1$		$c_x = 0.25$				$c_0 = 1$		$c_x = 0.25$	
Prob.	**Opt.**	**LS1**	**LS2**	**LS1**	**LS2**	**Prob.**	**Opt.**	**LS1**	**LS2**	**LS1**	**LS2**
B01	**82**	**0**	**0**	**0**	**0**	***B10***	**86**	**0.012**	0.042	**0.012**	**0**
B02	**83**	0.010	0.011	**0**	**0**	***B11***	**88**	**0**	0.012	**0**	**0**
B03	**138**	**0**	**0**	**0**	**0**	***B12***	**174**	0.041	0.040	0.005	**0**
B04	**59**	0.034	0.027	0.008	**0**	***B13***	**165**	0.024	0.024	0.024	**0**
B05	**61**	**0**	**0**	**0**	**0**	***B14***	**235**	0.029	0.029	0.019	**0.008**
B06	**122**	0.031	0.039	0.011	**0**	***B15***	**318**	0.035	0.028	0.011	**0.006**
B07	**111**	0.008	0.008	0.005	**0**	***B16***	**127**	0.013	**0**	0.009	**0**
B08	**104**	0.034	0.040	0.004	**0**	***B17***	**131**	0.028	0.027	0.014	**0.006**

B09	**220**	0.014	0.013	0.004	**0**	***B18***	**218**	0.051	0.049	0.005	**0**

It is obvious that JPSOMR with local search strategies achieves much better results than that of JPSOMR without local search, and we thus did not present the results of the latter in Table 4 (see the column "$|P|$ = 20" in Table 3 for detailed results). We can see that JPSOMR with cooperation (c_x = 0.25) and the greedy local search performs the best among the four JPSOMR variants with different settings. The *p*-value obtained from a paired *t*-test on JPSOMR with and without cooperation is 4.5E-4, showing that JPSOMR with cooperation is significantly better than JPSOMR without cooperation. These results clearly demonstrate that the cooperation among particles can greatly improve the performance of JPSOMR to achieve better results than its multi-start variant.

5.2.3 The Evolution of the Swarm

In this set of experiments we investigate the evolution of the swarm over the iterations on three instances of different sizes: instance *B12* with network size 75, instance *B16* with network size 100 and instance *C01* with network size 500. We show the evolution of the best particle in the swarm of JPSOMR for these three instances in Figure 5, Figure 6 and Figure 7.

Figure 5 shows that instance *B12* is easy to solve regardless of the different settings in JPSOMR, although the optimum is found earlier by the algorithm with greedy local search. For the medium size instance *B16*, Figure 6 shows that although JPSOMR reaches the optimum with different settings, the variant with the cooperation and greedy local search is the fastest, while the variant with no cooperation and greedy local search is the slowest. For the large instance *C01*, Figure 7 shows again that cooperation and greedy local search provide the fastest convergence to the optimum. In this case, the slowest algorithm is the JPSOMR with cooperation and without greedy local search. This demonstrates that particularly for the large instance, greedy local search contributes to a better performance in JPSOMR with or without the cooperation.

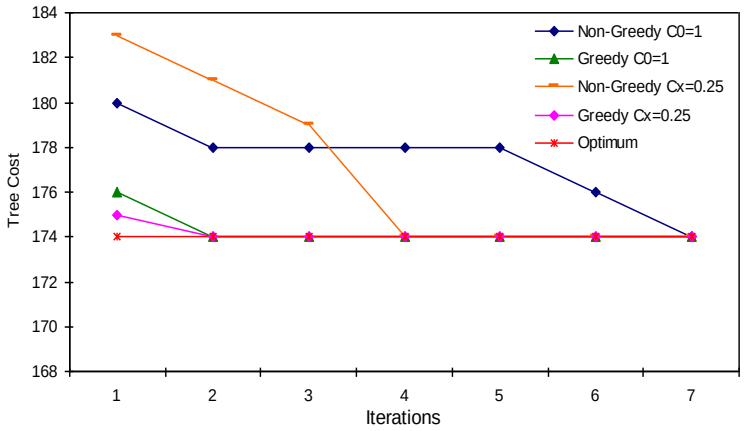


Figure 5. The evolution of JPSOMR with $|P|$ = 5, different settings of cooperation and local search on a Steiner tree problem of small size (network size = 75, instance *B12*)

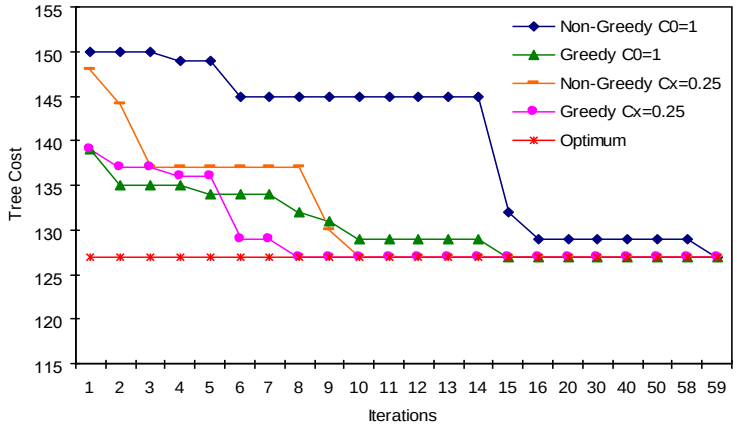


Figure 6. The evolution of JPSOMR with $|P|$ = 20, different settings of cooperation and local search on a Steiner tree problem of medium size (network size = 100, instance *B16*).

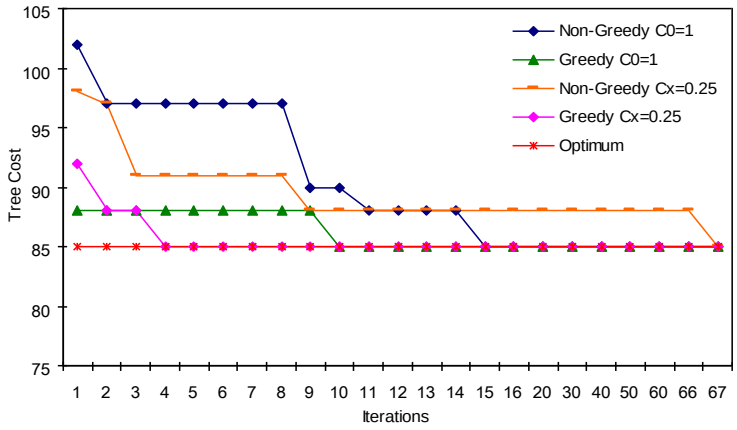


Figure 7. The evolution of JPSOMR with $|P|$ = 20, different settings of cooperation and local search on a Steiner tree problem of large size (network size = 500, instance *C01*).

Instance *B12* in Figure 5 is an easy problem thus the local search seems to do much of the work and the cooperative nature of JPSOMR is not obvious. However, it is interesting to see that, without the cooperation, JPSOMR with the first improvement local search is presents a slower convergence to the optimum. Instance *B16* in Figure 6 is harder to solve, and it is clear that the cooperation within the swarm helps the search to converge to the optimum. For the hardest instance *C01* in Figure 7, the greedy local search is clearly beneficial. For all instances, the local search has shown to highly affect the performance of JPSOMR. No matter what cooperation between particles in the swarm is applied, the JPSOMR algorithm with greedy local search converges faster than JPSOMR with the non-greedy local search. On the other hand, JPSOMR with c_x = 0.25 finds the optimal solution faster compared with JPSOMR with c_0 = 1 in Figure 6 and Figure 7. This indicates that particles cooperation is also important to the performance of JPSOMR, i.e. it performs better if particles cooperate (c_x = 0.25) rather than just randomly move (c_0 = 1) in the swarm.

5.3 Comparing JPSOMR with Other Approaches in the Literature

After studying the behavior of the proposed JPSOMR on selected instances, we now conduct extensive experiments to evaluate the overall performance of the proposed JPSOMR to other algorithms in the literature. We set ($c_0 = c_1 = c_2 = c_3 = 0.25$), with swarm size of $|P|$ = 20 and the greedy local search in JPSOMR.

5.3.1 JPSOMR and Other Approaches on Steiner Tree Problems

The JPSOMR algorithm is firstly compared with the GRASP algorithm developed by Skorin-Kapov and Kos (2006) on the category *B* instances. The stopping condition is set to a pre-defined number of iterations, namely 100 and 10 for category *B* and category *C* instances, respectively. To obtain comparable results we re-implemented the GRASP algorithm with the same parameters as in Skorin-Kapov and Kos (2006), i.e. the number of iterations is 5, the parameter α to manage the restricted candidate list is set as 5, and the number of iterations without improvement of the local search procedure is 2.

Table 5 presents the average tree cost, the best cost and the average computing time of the two algorithms on the category *B* instances. The average cost obtained by JPSOMR is better than that of GRASP in eight instances, however, by consuming more computing time. The proposed JPSOMR performs better than GRASP, and is able to find the optimal solution at each run, with only one exception (*B15*).

Table 5. Comparison between JPSOMR and GRASP (Skorin-Kapov and Kos, 2006) approaches on smaller category *B* instances. The best results are presented in bold. Opt.: the optimal values for each instance; ΔMean = (mean – Opt.) / Opt.; ΔBest = (best – Opt.) / Opt.

Δ = ∞		JPSOMR			GRASP			Δ = ∞		JPSOMR			GRASP		
Prob.	**Opt.**	**ΔMean**	**ΔBest**	**Time(s)**	**ΔMean**	**ΔBest**	**Time(s)**	**Prob.**	**Opt.**	**ΔMean**	**ΔBest**	**Time(s)**	**ΔMean**	**ΔBest**	**Time(s)**
B01	***82***	**0**	0	0.002	**0**	0	0.086	***B10***	***86***	**0**	0	1.469	0	0	2.394
B02	***83***	0.03	0	2.935	**0**	0	0.101	***B11***	***88***	**0**	0	1.141	0.002	0	0.646
B03	***138***	**0**	0	0.048	**0**	0	0.136	***B12***	***174***	**0**	0	0.8	**0**	0	1.255
B04	***59***	**0**	0	0.063	**0**	0	0.067	***B13***	***165***	**0**	0	93.421	0.039	0	2.099
B05	***61***	**0**	0	0.815	**0**	0	0.108	***B14***	***235***	**0.001**	0	239.922	0.002	0	1.816
B06	***122***	**0**	0	0.618	0.018	0	0.496	***B15***	***318***	**0.005**	0	320.513	0.012	0.006	5.178
B07	***111***	**0**	0	0.213	**0**	0	0.183	***B16***	***127***	**0**	0	10.503	0.018	0	1.611
B08	***104***	**0**	0	0.697	**0**	0	0.424	***B17***	***131***	0.002	0	159.33	**0**	0	1.799
B09	***220***	**0**	0	0.199	**0**	0	0.641	***B18***	***218***	**0**	0	1.136	0.001	0	4.502

We then test the computational expenses of JPSOMR and GRASP algorithms, the average time needed to find the optimal solution is presented in Table 6. If the algorithm failed to find the optimal solution within a limited time of 60 seconds, the best result obtained and the corresponding time is provided. For small instances (*B1-B12*) in category *B*, our JPSOMR spends less time to find the optimal solution for seven out of 12 instances. For larger instances, GRASP performs slightly better than our JPSOMR, although the differences are mostly very small.

Table 6. Comparison between JPSOMR and GRASP (Skorin-Kapov and Kos, 2006) on category *B* instances. The best results obtained are presented in bold. Opt.: the optimal values for each instance; Δcost = (cost – Opt.) / Opt.

Δ = ∞		JPSOMR		GRASP		Δ = ∞		JPSOMR		GRASP	
Prob.	**Opt.**	**Δcost**	**Time(s)**	**Δcost**	**Time(s)**	**Prob.**	**Opt.**	**Δcost**	**Time(s)**	**Δcost**	**Time(s)**
B01	**82**	**0**	**0.029**	**0**	0.086	***B10***	**86**	**0**	1.771	**0**	**0.442**
B02	**83**	0.030	2.935	0	**0.101**	***B11***	**88**	**0**	0.964	**0**	**0.811**
B03	**138**	**0**	**0.045**	**0**	0.136	***B12***	**174**	**0**	**1.073**	**0**	1.257
B04	**59**	**0**	**0.066**	**0**	0.067	***B13***	**165**	0.015	43.432	**0**	**15.163**
B05	**61**	**0**	**0.086**	**0**	0.108	***B14***	**235**	0.003	41.803	**0**	**2.557**
B06	**122**	**0**	**0.824**	**0**	1.201	***B15***	**318**	0.006	43.946	**0.001**	**27.484**
B07	**111**	**0**	0.31	**0**	**0.183**	***B16***	**127**	**0**	11.525	**0**	**2.557**
B08	**104**	**0**	0.538	**0**	**0.431**	***B17***	**131**	0.005	49.862	**0**	**1.81**
B09	**220**	**0**	**0.214**	**0**	0.644	***B18***	**218**	**0**	**1.056**	**0**	7.945

We further test JPSOMR compared against GRASP and another recent algorithm, DPSO by Zhong et al. (2008), on the larger and harder category *C* instances. The DPSO algorithm was set to stop after 1250 iterations, or after 250 continuous generations where no better results can be found. To enable the comparison among the three algorithms, we therefore set the same number of iterations (iterations = 10) in both JPSOMR and GRASP, which is much smaller than that of DPSO, and compare their results on category *C* instances in Table 7. For 12 of the 20 category *C* instances, JPSOMR successfully found the optimal solutions at least once. Although the DPSO algorithm found the best tree in 16 of 20 problems, the JPSOMR approach found smaller tree cost in seven of 20 problems using a much smaller number of iterations.

Table 7. Comparison between JPSOMR, GRASP (Skorin-Kapov and Kos, 2006) and DPSO (Zhong et al., 2008) approaches on larger category *C* instances.

$\Delta = \infty$		**JPSO**			**GRASP**			**DPSO**		
Prob.	**Opt.**	**ΔMean**	**ΔBest**	**ΔWorst**	**ΔMean**	**ΔBest**	**ΔWorst**	**ΔMean**	**ΔBest**	**ΔWorst**
C01	**85**	0.012	0	0.035	**0**	0	0	**2.0**	0	0
C02	**144**	0.035	0.014	0.056	**0**	0	0	**2.0**	0	0
C03	**754**	0.009	0.007	0.011	0.009	0.008	0.009	**2.001**	0	0.005
C04	**1079**	0.004	0.001	0.006	0.018	0.007	0.021	**2.0**	0	0.001
C05	**1579**	**0**	0	0	0.005	0.005	0.005	**2.0**	0	0
C06	**55**	**0**	0	0	**0**	0	0	**2.0**	0	0
C07	**102**	0.007	0	0.010	**0**	0	0	**2.0**	0	0
C08	**509**	**0.001**	0	0.002	0.017	0.016	0.018	2.002	0	0.006
C09	**707**	0.005	0.003	0.007	0.010	0.008	0.011	**2.003**	0.001	0.006
C10	**1093**	**0.001**	0	0.001	0.003	0	0.004	2.001	0	0.005
C11	**32**	0.009	0	0.031	0.019	0	0.031	**2.003**	0	0.031
C12	**46**	**0**	0	0	0.009	0	0.022	**2.0**	0	0
C13	**258**	**0.002**	0	0.004	0.017	0.008	0.023	2.010	0.004	0.019
C14	**323**	**0.003**	0	0.003	0.015	0.006	0.025	2.005	0.003	0.009
C15	**556**	**0**	0	0	0.001	0	0.005	2.001	0	0.004
C16	**11**	0.091	0.091	0.091	**0.036**	0	0.091	**2.036**	0	0.091
C17	**18**	**0.011**	0	0.056	0.044	0	0.056	2.022	0	0.056
C18	**113**	**0.024**	0.018	0.027	0.044	0.027	0.053	2.027	0.018	0.053
C19	**146**	0.014	0.014	0.014	0.037	0.034	0.041	**2.010**	0	0.021
C20	**267**	**0**	0	0.004	0.009	0.004	0.015	**2.0**	0	0

5.3.2 JPSOMR and Other Approaches on DCLC Multicast Problems

We now compare JPSOMR to other algorithms for the DCLC multicast problem in the literature. Details of these algorithms can be found in Section 2.4. Three random topologies have been generated for each network size of 10, 20, 30, 40, 50, 60, 70, 80, 90 and 100 nodes, where the link cost depends on the length of the link, and all the link delays are set to 1. The group size (number of destinations) is set as = 30% × network size. Different delay bounds Δ are set depend on the network size (Δ = 7*ms* for network sizes of 10-30, Δ = 8*ms* for network size 40-60, Δ = 10*ms* for network size 70-80, and Δ = 12*ms* for network size 90-100). This setting is the same as in the simulations designed by Ghaboosi and Haghighat (2007b) and Qu et al. (2009). The simulation has been run 10 times on each random network. The average tree cost for all the random networks are shown in Table 8.

Table 8. Average tree costs from existing approaches in the literature on DCLC multicast routing problems with random networks of 10-100 nodes. The lowest tree costs are in bold.

Algorithms		**Average Tree Cost**
Heuristics	KPP (Kompella et al., 1993)	905.581

	BSMA (Zhu et al., 1995)	872.681
GA-based Algorithms	Wang et al. (2001)	815.969
	Haghighat et al. (2004)	808.406
TS-based Algorithms	Skorin-Kapov and Kos (2003)	897.875
	Youssef et al. (2002)	854.839
	Wang et al. (2004)	869.291
	Ghaboosi and Haghighat (2007a)	739.095
Path Relinking	Ghaboosi and Haghighat (2007b)	691.434
VNSMR	Qu et al. (2009)	680.067
GRASP	Skorin-Kapov and Kos (2006)	669.880
JPSOMR	Our proposed JPSOMR algorithm	**662.100**

It is clear from Table 8 that JPSOMR outperformed all existing algorithms in the literature, producing the best average tree cost amongst all the algorithms tested. The GRASP and our previous VNSMR algorithm are the closest competitors. Note that in the literature all the other works had reported the average results on all problem instances of different sizes and thus we provided such comparison in Table 8.

To make a closer comparison between these three best approaches, Table 9 provides details of the average tree cost for each of the individual network of different size. Computing time is set as 60 seconds to all three competitor algorithms. The lowest tree costs highlighted in bold clearly show that JPSOMR has the best overall performance, obtaining the smallest tree costs on five of the largest network sizes, and the same best tree costs on the other two smaller networks. On only three of the networks, one of the other algorithms achieves better result than JPSOMR.

Table 9. Average tree cost of JPSOMR on 10 network sizes compared with VNSMR and GRASP in Table 8. Computing time = 60 seconds.

Network Size	VNSMR	GRASP	JPSOMR
10	**94.667**	**94.667**	**94.667**
20	282.333	**270.667**	**270.667**
30	415.667	**394.667**	395.667
40	**518**	526.467	526
50	726.667	697.067	**687.333**
60	812.333	761.133	**748.667**
70	805.333	797.533	**785.667**
80	922.333	902.667	**889.667**
90	**1182.67**	1201.933	1194
100	1040.67	1052	**1028.667**

6. Conclusions and Future Work

In this paper, we have presented a discrete particle swarm optimization algorithm to solve both the Steiner Tree problem in the OR Library and the Delay Constraint Least Cost multicast routing problems. The proposed JPSOMR algorithm has been developed based on the Jumping Particle Swarm Optimization (JPSO) developed in the literature. Particles in the swarm jump from one position to another in the discrete search space by making changes to the tree represented by their current position. This has been carried out by using path replacement operations which have been designed with regard to the specific structure and features of the multicast/Steiner tree. A local search is used to further improve solutions after the move of the particles.

We conducted an extensive set of experimentation on the benchmark Steiner tree problems to understand the behavior of our proposed JPSOMR algorithm with respect to the cooperation between particles, the size of the swarm and the effect of the different local search strategies. We found that the cooperative nature of JPSOMR is an important factor to its success in solving the problems in this study, and a good number of particles (20 in our case) are required for the algorithm to succeed. Further experiments have also been carried out to assess the performance of JPSOMR on solving a set of Delay

Constraint Least Cost multicast routing problems. Compared with several existing algorithms in the literature which have been re-implemented within the same simulation environment, the JPSOMR algorithm shows a very robust performance, and provides comparable or better results than those of the considered competitors on the exact same benchmark problem instances. We believe this is the first study that provides an extensive investigation of a JPSO algorithm on both the Steiner tree and the DCLC multicast routing problems tested in the literature. Comparative results and details of the problem instances have been made publicly available at http://www.cs.nott.ac.uk/~rxq/benchmarks.htm for scientific comparisons.

We conclude that swarm optimization is a good technique to tackle multicast routing problems and the underlying Steiner tree problems. The proposed JPSOMR has shown to be very successful for solving both problems. In our future work, we intend to consider a wider range of multicast routing problems with multiple objectives and more real life features and constraints.

Acknowledgement

This research is supported by Hunan University, China, and the School of Computer Science at The University of Nottingham, UK.

References

Allahverdi A., Al-Anzi F.S. (2006). A PSO and a tabu search heuristics for the assembly scheduling problem of the two-stage distributed database application. *Computers & Operations Research.* 33(4), 1056-1080.

Anghinolfi D., Paolucci M. (2009). A new discrete particle swarm optimization approach for the single-machine total weighted tardiness scheduling problem with sequence-dependent setup times. *European Journal of Operational Research*, 193(1), 73-85.

Barahona F., Ladanyi L. (2006). Branch and cut based on the volume algorithm: Steiner trees in graphs and max-Cut, *RAIRO Operations Research* 40(1), 53-73.

Betsekas, D., Gallager, R.: Data Networks (2nd edition). Englewood Cliffs, HJ: Prentice-Hall (1992)

Beasley J.E. (1990). OR-Library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11), 1069-1072.

Bonabeau E., Theraulaz G., Dorigo M. (1999). *Swarm Intelligence: From Natural To Artificial Systems*. Oxford University Press.

Castro J.P., Landa-Silva D., Moreno-Perez J.A. (2009). Exploring feasible and infeasible regions in the vehicle routing problem with time windows using a multi-objective particle swarm optimization approach, *Studies in Computational Intelligence volume 236*, pp. 103-114, Springer Berlin/Heidelberg.

Consoli S., Moreno-Perez J.A., Darby-Dowman K., Mladenovic N., (2010). Discrete particle swarm optimization for the minimum labelling Steiner tree problem, *Natural Computing*, 9(1): 29-46.

Cormen T.H., Leiserson C.E., Rivest R.L., Stein C. (2001). *Introduction to algorithms*, 2nd edition, MIT Press.

Costa A.M., Cordeau, J.F., Laporte G. (2006). Exact and approximate algorithms for a class of Steiner tree problems arising in network design and lot sizing, *4th US-European Workshop on Logistics and Supply Chain Management*, Hamburg.

Eppstein D. (1998). Finding the k shortest paths. *SIAM J. Computing*, 28(2), 652-673.

Feo, T.A., Resende, M.G. (1995). Greedy randomised adaptive search procedures. *Journal of Global Optimization*. 6, 109-133

Floyd, R.W. (1962). Algorithm 97: Shortest Path". *Communications of the ACM*, 5(6), 345.

Garey M.R., Johnson D.S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York.

Ghaboosi N., Haghighat A.T. (2007a). Tabu search based algorithms for bandwidth-delay-constrained least-cost multicast routing. *Telecommunication Systems* 34(3-4), 147-166.

Ghaboosi N., Haghighat A.T. (2007b). A path relinking approach for delay-constrained least-cost multicast routing problem. *19th IEEE International Conference on Tools with Artificial Intelligence*, pp. 383-390. IEEE Computer Society Washington, DC, USA.

Guo L., Matta I. (2000). QDMR: An efficient QoS dependent multicast routing algorithm. *Journal of communication and networks*. 2(2), 168-176.

Haghighat A.T., Faez K., Dehghan M., Mowlaei A., Ghahremani Y. (2004). GA-based heuristic algorithms for bandwidth-delay-constrained least-cost multicast routing. *Computer Communications*. 27, 111-127.

Hwang F.K., Richards D.S. (1992). Steiner tree problems. *Networks*. 22, 55-89.

Kennedy J., Eberhart R.C. (1995). Particle swarm optimization. *Proceedings of IEEE International Conference on Neural Networks*, Piscataway, NJ. pp. 1942-1948.

Eberhart R.C., Shi Y., Kennedy J. (2001). *Swarm Intelligence*. Morgan Kaufmann.

Kennedy J., Eberhart R.C. (1997). A discrete binary version of the particle swarm algorithm. *Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics 1997*, Piscataway, NJ. pp. 4104-4109.

Koch T., Martin A. (1998). Solving Steiner tree problems in graphs to optimality. *Networks*, 32, 207-232.

Koch T., Martin A., Voß S. (2002) SteinLib: An updated library on Steiner tree problems in graphs. In Du D.-Z., Cheng X. (eds.) *Steiner Trees in Industries*, pp. 285-325, Springer.

Kompella V.P., Pasquale J.C., Polyzos G.C. (1993). Multicast routing for multimedia communication. *IEEE/ACM Transactions on Networking*. 1, 286-292.

Li C., Cao C., Li Y., Yu Y. (2007). Hybrid of genetic algorithm and particle swarm optimization for multicast QoS routing. *The 6th IEEE International Conference on Control and Automation*, pp 2355-2359.

Montgomery D.C. (2005) *Design and Analysis of Experiments*, Wiley, 6th edition.

Moreno-Perez J.A., Castro-Gutierrez J.P., Martinez-Garcia F.J., Melian B., Moreno-Vega J.M., Ramos J. (2007). Discrete particle swarm optimization for the p-median problem. *Proceedings of the 7th Metaheuristics International Conference*, Montreal, Canada.

Oliveira C.A.S., Pardalos P.M. (2005). A survey of combinatorial optimization problems in multicast routing. *Computers & Operations Research*. 32(8), 1953-1981.

Onwubolu G.C., Clerc M. (2004). Optimal path for automated drilling operations by a new heuristic approach using particle swarm optimization. *International Journal of Production Research* 42(3), 473-491.

Prim R.C. (1957). Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36, 1389–1401.

Qu R., Xu Y., Kendall G. (2009). A variable neighborhood search algorithm for delay-constrained least-cost multicast routing. *Proceedings of Learning and Intelligent OptimizatioN* (LION 3), Trento, Italy, Jan 14-18.

Salama H.F., Reeves D.S., Viniotis Y. (1997). Evaluation of multicast routing algorithms for real-time communication on high-speed networks. *IEEE Journal on Selected Areas in Communications*. 15(3), 332-345.

Sha D.Y., Hsu C. (2006). A hybrid particle swarm optimization for job shop scheduling problem. *Computers & Industrial Engineering.* 51(4), 791-808.

Skorin-Kapov N., Kos M. (2003). The application of steiner trees to delay constrained multicast routing: a tabu search approach. *Proceedings of the 7th international Conference on Telecommunications*, pp. 443-448.

Skorin-Kapov N., Kos M. (2006). A GRASP heuristic for the delay-constrained multicast routing problem. *Telecommunication Systems*. 32(1), 55-69.

Sun J., Liu J., Xu W. (2006). QPSO-Based QoS Multicast Routing Algorithm Book Series, *Lecture Notes in Computer Science* Vol. 4247, pp. 261-268. Springer.

Tasgetiren M.F., Liang Y.C., Sevkli M., Gencyilmaz G. (2007). A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. *European Journal of Operational Research* 177(3), 1930-1947.

Wang H., Fang J.,Wang H., Sun Y.M. (2004). TSDLMRA: an efficient multicast routing algorithm based on tabu search. *Journal of Network and Computer Applications*. 27(2), 77-90.

Wang Z., Shi B., Zhao E. (2001). Bandwidth-delay-constrained least-cost multicast routing based on heuristic genetic algorithm. *Computer communications*. 24, 685-692.

Wang J., Wang X., Huang M. (2005). A Hybrid Intelligent QoS Multicast Routing Algorithm in NGI. *Proceedings of the Sixth International Conference on Parallel and Distributed Computing Applications and Technologies*, pp. 723-727.

Waxman B.M. (1988). Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*. 6(9), 1617-1622.

Xu, Y. and Qu, R.: An Iterative Local Search Approach based on Fitness Landscapes Analysis for the Delay-constrained Multicast Routing Problem. Computer Communications, 35: 352-365, (2012)

Youssef H., Al-Mulhem A., Sait S.M., Tahir M.A. (2002). QoS-driven multicast tree generation using tabu search, *Computer Communications*. 25(11-12), 1140-1149.

Yuan P., Ji C., Zhang Y., Wang Y. (2004) Optimal multicast routing in wireless ad hoc sensor networks, *IEEE International Conference on Networking, Sensing and Control*, pp. 367- 371.

Zachariasen M. (1999). Local search for the Steiner tree problem in the Euclidean plane. *European Journal of Operational Research*. 119, 282-300.

Zhang H., Li H., Tam C.M. (2006). Permutation-based particle swarm optimization for resource-constrained project scheduling, *Journal of Computing in Civil Engineering*, 20(2), 141-149.

Zhong W.L., Huang J., Zhang J. (2008). A novel particle swarm optimisation for the Steiner tree problem in graphs. *IEEE World Congress on Evolutionary Computation*, pp. 2460-2467.

Zhu Q., Parsa M., Garcia-Luna-Aceves J.J. (1995). A source-based algorithm for delay-constrained minimum-cost multicasting, *Proceedings of the 14th Annual Joint Conference of the IEEE Computer and Communication*, pp. 377-385. IEEE Computer Society Press, Boston, Massachusetts.