MISTA 2009

A GRASP approach for the delay-constrained multicast routing problem

Ying Xu • Rong Qu

Abstract The rapid development of real-time multimedia applications requires Quality of Service (QoS) based multicast routing in underlying computer networks. The constrained minimum Steiner tree problem in graphs as the underpinning mathematical model is a well-known NP-complete problem. In this paper we investigate a GRASP (Greedy Randomized Adaptive Search Procedure) approach with VNS (Variable Neighborhood Search) as the local search strategy for the Delay-Constrained Least-Cost (DCLC) multicast routing problems. A large number of simulations carried out on the benchmark problems in the OR-library and a group of randomly generated graphs demonstrate that the proposed GRASP algorithm with VNS is highly efficient in solving the DCLC multicast routing problem. It outperforms other existing algorithms and heuristics in the literature.

1 Introduction

Multicast routing is a mechanism which transfers information from a source to a group of destinations simultaneously. With the increasing development of numerous multicast network applications (e.g. E-learning, E-commerce, video-conferencing), the underlying computer network requires multicast routing with certain QoS (Quality of Service) constraints. One example is that many of these real-time applications can tolerate only a bounded end-to-end delay. Other QoS constraints in reality include cost, bandwidth, delay variation, lost ratio and hop count, etc. Multicast QoS routing has received significant research attention in the area of computer networks and algorithmic network theory [1-3]. This paper is concerned with two of the most important QoS demands, the total cost of the multicast tree from the source to all the destinations and the end-to-end delay bound for the total delay from the source to any destination in the multicast group.

Multicast routing problems can be reduced to Minimum Steiner Tree Problems in Graph (MStTG) [4]. Generally, given an undirected graph $G = (V, E)$, where V is a set of nodes, E is a set of edges, and a subset of nodes $D \subseteq V$, a Steiner tree is a tree which connects all the nodes in D using a subset of edges in E. Extra nodes in $V \backslash D$ may be added to the Steiner tree, called the Steiner nodes. The MStTG problem is to search for a minimal Steiner tree with respect to the total edge costs $c(e)$, $e \in E$, which has been proven to be NP-complete [5]. The Delay-Constrained Least-Cost (DCLC) multicast routing problem searches for a Delay-Constrained Steiner Tree (DCST), which is also NP-complete [6]. An early survey on protocol functions and mechanisms for data transmission within a group and related solutions was given in [7]. A recent overview has been presented in [8] on applications of combinatorial optimization problems and associated algorithms for multicast routing.

In this paper, we investigate a GRASP approach with VNS as the local search method for DCLC multicast routing problems. To our knowledge, very little attention has been given to

Ying Xu
The Automated Scheduling, Optimisation and Planning (ASAP) Group, School of Computer Science, University of Nottingham, Nottingham, NG8 1BB, UK
School of Computer and Communication, Hunan University, Changsha, 410082, CHINA
E-mail: yxx@cs.nott.ac.uk
Rong Qu
The Automated Scheduling, Optimisation and Planning (ASAP) Group, School of Computer Science, University of Nottingham, Nottingham, NG8 1BB, UK
E-mail: rxq@cs.nott.ac.uk

the GRASP approach on multicast routing and we know only one exception in [9]. We tested our proposed *GRASP-VNS* algorithm on a set of benchmark problems (*steinb* [10]) for Steiner tree problems in the OR library. Computational results indicate that the *GRASP-VNS* algorithm obtains the same or better quality solutions compared with other three algorithms: *Multi-VNS* (a multi-start algorithm of the extension of our previous variable neighborhood search algorithm [11]), the GRASP algorithm in [9] and *VNSMR2* [11]. Furthermore, we tested our *GRASP-VNS* algorithm on a set of random graphs. Our proposed *GRASP-VNS* algorithm performs the best in terms of the total tree cost in comparison with the existing algorithms and heuristics.

The rest of the paper is organized as follows. In section 2, we present the problem definition and related work. Section 3 presents the proposed *GRASP-VNS* algorithm. We evaluate our algorithm by computer simulations on a range of problem instances and summarize the obtained simulation results in section 4. Finally, section 5 concludes this paper and presents the possible future work.

2 The problem definition and related work

2.1 The network model and problem definition

A computer network can be modeled as a connected, directed graph $G = (V, E)$ with $|V| = n$ nodes and $|E| = l$ links, where V is a set of nodes and E is a set of links. Each link $e = (i, j) \in E$ is associated with two real value functions, namely link cost $c(e)$ and link delay $d(e)$. The link cost $c(e)$ is a measure of the utilization of the corresponding link's resources. The link delay $d(e)$ is the delay caused by transferring messages along the link. Due to the asymmetric nature of computer networks, for link $e = (i, j)$ and link $e' = (j, i)$, it is possible that $c(e) \neq c(e')$ and $d(e) \neq d(e')$. The nodes in V include a source node s and a set of destination nodes D called multicast groups, which receive data stream from the source, denoted by $D \subseteq V \backslash \{s\}$.

We define a path from node u to v as an ordered set of links, denoted by $P(u, v) = \{(u, i), (i, j), \ldots, (k, v)\}$. A multicast tree $T(s, D) \subseteq E$ is a tree rooted at source s and spanning all destination nodes in D. We denote by $P(s, r_i) \subseteq T$ the set of links in T that constitute the path from s to $r_i \in D$. The end-to-end delay from s to each destination r_i, denoted by $Delay(r_i)$, is the sum of the delays of all links along $P(s, r_i)$, i.e.

$$Delay(r_i) = \sum_{e \in P(s, r_i)} d(e), \ \forall r_i \in D \quad (1)$$

The delay of the tree, denoted by $Delay(T)$, is the maximum delay among all $Delay(r_i)$ from source to each destination, i.e.

$$Delay(T) = \max\{ Delay(r_i) \mid \forall r_i \in D \} \quad (2)$$

The total cost of the tree, denoted by $Cost(T)$, is defined as the sum of the cost of all links in the tree, i.e.

$$Cost(T) = \sum_{e \in T} c(e) \quad (3)$$

The delay bound is the upper bound for each $Delay(r_i)$ along the path from s to r_i. Applications may assign different upper bound δ_i to each destination $r_i \in D$. In this paper, we assume that the upper bounds for all destinations are the same, and is denoted by $\Delta = \delta_i$.

Given these definitions, we formally define the Delay-Constrained Steiner Tree (DCST) problem as:

The Delay-Constrained Steiner Tree (DCST) Problem: Given a network G, a source node s, a destination node set D, a link cost function $c(\cdot)$, a link delay function $d(\cdot)$, and a delay bound Δ, the objective of the DCST Problem is to construct a multicast tree $T(s, D)$ such that the delay bound is satisfied, and the tree cost $Cost(T)$ is minimized. We define the objective function as:

$$\min\{\, Cost(T) \mid P(s, r_i) \subseteq T(s, D),\ Delay(r_i) \leq \Delta,\ \forall r_i \in D \,\} \qquad (4)$$

2.2 Related work

The DCST problem has received extensive studies, and consequently many exact and heuristic algorithms have been developed since the first DCLC multicast routing algorithm KPP [12] was presented in 1993. Most of these algorithms can be classified as source-based or destination-based multicast routing algorithms. In source-based algorithms, each node has all the necessary information to construct the multicast tree [12-16]. While destination-based algorithms do not require that each node maintains the entire network status information, and multiple nodes participate in constructing the multicast tree [6,17-19].

In recent years, metaheuristic algorithms such as simulated annealing [20,21], genetic algorithm [22,23], tabu search [24-27], GRASP [9], path relinking [28] and VNS [11] have been investigated for various multicast routing problems. Although GRASP algorithms [29,30] have been applied to solve the Minimum Steiner Tree Problem in Graphs (MStTG), along with many other optimization problems, little attention has been given to GRASP for solving the multicast QoS routing problem. Martins et al. [29] describe a hybrid GRASP heuristic with two local search strategies for the Steiner problem. The proposed algorithms were tested on a group of parallel processors. The computing results show that their GRASP heuristic has high possibilities of finding the optimal solutions. Ribeiro et al. [30] present a hybrid GRASP with weight perturbations and two adaptive path-relinking heuristics on a set of elite solutions for the Steiner problem in graphs. One is the path relinking with complementary move; the other is the path relinking with weight penalization. Experiment results on a broad set of benchmark problems illustrate the effectiveness and the robustness of their GRASP algorithm. Both [29] and [30] are restricted to deal with Steiner tree problems with no constraints. In [9], a GRASP heuristic is developed for delay-constrained multicast routing problem. In the GRASP construction phase, a randomized feasible solution is constructed by Dijkstra's shortest path algorithm. The tabu search heuristic [25] is applied in the local search phase. The best found solution after a given number of iterations is accepted as the final solution. Experiment results show that their algorithm outperforms the KPP [12] and the tabu search algorithm [25] on the same problems. A variable neighborhood descent search algorithm *VNSMR2* [11] is proposed in our previous work for the DCLC multicast routing problem. It employs three neighborhood structures, one is node-based and the other two are based on a path replacement strategy. The three neighborhoods are designed to reduce the tree cost and at the same time satisfy the delay constraint.

3 The GRASP-VNS algorithm

GRASP (Greedy Randomized Adaptive Search Procedure) is an efficient multi-start metaheuristic for a wide range of optimization problems, where each iteration consists of two phases: a construction phase and a local search phase [32]. After creating a feasible solution in the construction phase, a local search is applied to explore the neighborhood of the feasible solution until a local minimum is found. The construction phase builds the feasible solution in a greedy randomized manner by iteratively creating a candidate list of elements, called the restricted candidate list (RCL), by evaluating the elements not yet included in the unfinished solution with a certain greedy function. Elements in RCL can be randomly selected and added in the unfinished solution until a feasible solution is obtained. The size of RCL is limited either by the number of the elements or by the quality of the elements with respect to the best candidate element. To further improve the feasible solution generated in the construction phase, a local search is applied to search for better neighboring solutions of the feasible solutions. Different local search strategies can be used by employing the designed neighborhood structures. After a given number of iterations, the best overall solution is kept as the final solution. More detailed descriptions of the GRASP heuristic can be found in [31,32].

The construction phase of GRASP makes the search diversified, while the local search phase in the GRASP intensifies the search by exploring the neighborhood of the current solution. GRASP has been successfully applied to a wide range of combinatorial optimization problems [33-35]. An especially appealing characteristic of GRASP is that it is easy to implement and few parameters need to be set and tuned. Our motivation is to apply GRASP in conjunction with our previous VNSMR algorithm for the multicast routing problem. Fig.1 illustrates the pseudo code of our proposed *GRASP-VNS* algorithm.

```
GRASP-VNS(G =(V, E), s, D, Δ, Iter, α )
{ // s: source node; D: destination set; Δ ≥ 0: the delay bound
  // Iter: the number of iterations;  α : parameter for creating greedy randomized solution
   if Delay(r_i) of the Dijkstra's least delay path P(s, r_i) > Δ, ∀ r_i ∈ D;
   then return FAILED; // no feasible solution exists
   else
   {
        i = 0;
        while i < Iter do {
             if i ==0 // pure greedy solution ( α =1) in the first iteration
               T_best = GreedyRandomSolution(G , s, D, Δ, 1); //the construction phase, see Fig. 2
             else // the remaining iterations of GRASP-VNS ( α >1)
              T_0= GreedyRandomSolution(G, s, D, Δ, α );  //the construction phase, see Fig. 2
              T = VNSMR2(G, s, D, Δ, T_0);  // the local search phase
              if ((Cost(T)<Cost(T_best))||((Cost(T)==Cost(T_best))&(Delay(T)<Delay(T_best))))
              then T_best = T;
              i++;
        } //end of while loop
   }
   return T_best;
}
```

Fig. 1. The pseudo code of the *GRASP-VNS* algorithm

3.1 The construction phase

In the construction phase of our *GRASP-VNS* algorithm, we use the similar greedy strategy in [9] to create the randomized initial solution. A delay-constrained Steiner tree T is constructed in the following three steps. The pseudo code of the construction phase is given in Fig.2.

(1) Starting from the source node, i.e. $T = \{s\}$, we calculate the shortest path which connects s and each destination by using the Dijkstra's shortest path algorithm. This path is denoted as *ConnectPath*[r_i], for each destination node $r_i \in D\backslash T$. We set *ConnectPath*[r_i] as the least cost path from s to r_i if the delay of the path satisfies the delay bound; otherwise, *ConnectPath*[r_i] is set as the least delay path from s to r_i. We denote *ConnectCost*[r_i] as the cost of the path.

(2) We create the restricted candidate list (RCL) by choosing those nodes $r_i \in D\backslash T$, for which *ConnectCost*[r_i] $\leq \alpha \cdot$ *BestConnectCost*, where $\alpha \geq 1$ and *BestConnectCost* = min{*ConnectCost*[r_j], $\forall r_j \in D\backslash T$ }. If α =1, then the algorithm is purely greedy. This means the RCL contains only the destination node with the least connect cost. If α >1, the RCL includes the nodes with the path cost within the range of $\alpha \cdot$ *BestConnectCost*.

(3) We randomly choose a destination node r from the RCL and add *ConnectPath*[r] to the tree T. We update the *ConnectPath*[r_i] and *ConnectCost*[r_i] of the remaining unconnected destination nodes in $D\backslash T$ by searching the new shortest path from these destination nodes to the current tree T. After that, the algorithm will return to step (2). The procedure ends until all the destination nodes are included in the tree.

In order to construct a high quality starting solution, we use the pure greedy algorithm in the first iteration i.e. α = 1. In the remaining iterations, we set α > 1 which gives more diversity to the construction phase to explore the solution space.

3.2 The local search phase

After a feasible solution is generated in the construction phase, we apply our *VNSMR2* developed in [11] as the local search method to improve the initial solution. It systematically changes the employment of different neighborhoods within a local search, thus the search is more flexible to traverse among different search spaces and potentially leads to better solutions. Three neighborhood structures are designed for multicast routing problems. The first neighborhood based on the nodes operation generates a neighboring solution by deleting a node from the current solution and then using Prim's algorithm to create the minimum spanning tree of the remaining nodes. The other two neighborhoods operate on the paths by using a path replacement strategy based on the Dijkstra's shortest path algorithm to reduce the tree cost of the current solution and at the same time satisfy the delay bound. *VNSMR2* was implemented based on the variable neighborhood decent search algorithm, a variant of variable neighborhood search (VNS) [36], where the current solution is always updated by the best neighboring solution in each neighborhood structure. In this paper we hybridize *VNSMR2* within the GRASP approach for solving not only random multicast routing problems but also more challenging benchmark problems in the OR Library.

```
GreedyRandomSolution(G =(V, E), s, D, Δ, α )
{  // s: source node; D: destination set; Δ ≥ 0 is the delay bound
   // α  is parameter to create the RCL (restricted candidate list)
     for all r_i ∈ D
          Calculate ConnectPath[r_i] and ConnectCost[r_i];
     T = s;
     while D ⊄ T do {
          BestConnectCost = min{ConnectCost[r_i], ∀ r_i ∈ D\T};
          Create RCL of all r_i ∈ D\T where ConnectCost[r_i] ≤ α ·BestConnectCost;
           r = Random(RCL, Randseed); //Randomly choose r from RCL
                                          //Randseed ∈ (0,1]
           T = T ∪ ConnectPath[r];
          Update ConnectPath[r_j] and ConnectCost[r_j] for all r_j ∈ D\T;
     }  //end of while loop
     return T;
}
```

Fig. 2. The pseudo code of the construction phase of *GRASP-VNS*

4 Performance Evaluations

We use a multicast routing simulator (MRSIM) implemented in C++ based on Salama's generator [1] to generate random network topologies. The simulator defines the link delay function $d(e)$ as the propagation delay of the link (queuing and transmission delays are negligible). The link cost function $c(e)$ is defined as the current total bandwidth reserved on the link in the network, and is related to the distance of the link. The Euclidean metric is used to determine the distance $l(u, v)$ between pairs of nodes (u, v). Links connecting nodes (u, v) are placed with a probability

$$p(u,v) = \beta \exp(-l(u,v)/\alpha L) \qquad \alpha, \beta \in (0,1] \qquad (5)$$

where parameters α and β can be set to obtain desired characteristics in the graph. For example, a large β gives nodes a high average degree, and a small α gives long connections in the networks. L is the maximum distance between two nodes. In our simulations, we set α = 0.25, β = 0.40, average degree = 4. All simulations were run on a Windows XP computer with PVI 3.4GHZ, 1G RAM. More detailed information of the test datasets and some example solutions obtained by the algorithms on the tested instances are publicly available at http://www.cs.nott.ac.uk/~yxx/resource.html.

4.1 Experiments on benchmark problems (*steinb*) in OR-library

Firstly, experiments were carried out on 18 small and medium sized (50-100 nodes) steinb benchmark problems in the OR library, details given in Table 1. Since *steinb* problems are for the Steiner tree problem, only a cost function is assigned to the links in each benchmark problem. In our experiments, we randomly set the delays of the links when generating the network topology in the simulator for each *steinb* problem. The simulation was run 10 times on each instance. We implemented our proposed *GRASP-VNS* and re-implemented the *GRASP-CST* algorithm in [9]. We also extended *VNSMR2* to a multi-start algorithm by running it for a fixed number of iterations, named *Multi-VNS*. Each iteration of *Multi-VNS* starts from a greedy random initial solution, which is generated by randomly choosing a starting destination node and connecting it with the source by using the Dijkstra's shortest path algorithm, then repeatedly connecting the unvisited destination nodes with the sub-tree until all the destination nodes have been mounted to the tree.

Table 1. The problem characteristics of dataset *steinb* from the OR-library (|V|, |E|, |D| denote the number of nodes, the number of edges and the number of destinations in the instances respectively, 'OPT' denotes the optimal solution)

No.	**|V|**	**|E|**	**|D|**	**OPT**	**No.**	**|V|**	**|E|**	**|D|**	**OPT**	**No.**	**|V|**	**|E|**	**|D|**	**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	50	125	318
B04	50	100	9	59	**B10**	75	150	13	86	**B16**	100	17	200	127
B05	50	100	13	61	**B11**	75	150	19	88	**B17**	100	25	200	131
B06	50	100	25	122	**B12**	75	150	38	174	**B18**	100	50	200	218

To determine appropriate settings for the parameters in our *GRASP-VNS* algorithm, a number of initial tests were carried out. Our aim was to obtain good quality solutions by using as less number of iterations as possible to reduce the execution time. Parameter α (see Fig. 2.) should be properly set to find the balance between diversification and intensification of the search over the search space. In other words, we should find a trade off between the solution quality and the execution time. After the initial tests, the number of iterations is set as 4, α is set as 5 in the *GRASP-VNS* algorithm. Parameters for *GRASP-CST* are set as the same as in [9], where the number of iterations is 5, α is 5, and the number of iterations without improvement of the local search procedure is 2. The number of iterations is set to 4 in *Multi-VNS*

In the first group of experiments, we set the delay bound to a large enough number so that the problems are reduced to the unconstrained Steiner tree problem since the delays of the links play no role in constructing the Steiner tree. The average, best, worst tree cost and the computing time of *GRASP-VNS*, compared with that of *Multi-VNS*, *GRASP-CST* and *VNSMR2*, are illustrated in Table 2. We can see that the *GRASP-VNS* algorithm gave the best solutions (marked in bold) on all the instances except one (B13) with respect to the average tree cost, while *Multi-VNS* algorithm got 14 best solutions, both *GRASP-CST* and *VNSMR2* found 10 best results. The *GRASP-VNS* algorithm always found the optimal solutions in 14 out of 18 instances, which is better than *Multi-VNS*, *GRASP-CST* and *VNSMR2* which always got the optimal solutions on 13, 10 and 10 instances, respectively. Both *GRASP-VNS* and *GRASP-CST* found the optimal solution at least once out of 10 runs for each instance. It should be noticed that the *GRASP-VNS* algorithm achieved better results by consuming longer computing time than *GRASP-CST*.

For the DCST multicast routing problem, the delay bound is the key factor which affects the solutions obtained. The smaller the delay bound, the stronger the constraint. In the second group of experiments, we set the delay bound Δ_1 =1.1**Delay*(T_{OPT}), where T_{OPT} is the multicast tree of the optimal solution with the minimal cost and delay. With the bounded end-to-end delay, the *GRASP-VNS* algorithm still performs the best among these three algorithms in terms of average tree costs in Table 3. *GRASP-VNS* found the best solutions in 15 out of 18 instances, compared with that of *Multi-VNS* (13 best solutions), *GRASP-CST* and *VNSMR2* (9 best solutions). *GRASP-VNS* always found the optimal solutions in 12 out of 18 cases, while *Multi-VNS, GRASP-CST* and *VNSMR2* did so in 11, 8 and 10 cases, respectively.

Table 2. Performance of *GRASP-VNS*, *Multi-VNS*, *GRASP-CST* and *VNSMR2* for unconstrained Steiner tree problems ($\Delta = \infty$). The values marked with '*' denote the optimal solution.

No.	*GRASP-VNS*				*Multi-VNS*				*GRASP-CST*				*VNSMR2*			
	Avg.	**Best**	**Worst**	**Time(s)**	**Avg.**	**Best**	**Worst**	**Time(s)**	**Avg.**	**Best**	**Worst**	**Time(s)**	**Avg.**	**Best**	**Worst**	**Time(s)**
B01	**82***	82*	82*	0.79	**82***	82*	82*	0.49	**82***	82*	82*	0.63	**82***	82*	82*	0.13
B02	**83***	83*	83*	1.46	**83***	83*	83*	1.95	**83***	83*	83*	0.80	**83***	83*	83*	0.18
B03	**138***	138*	138*	2.12	**138***	138*	138*	2.09	**138***	138*	138*	1.10	**138***	138*	138*	0.26
B04	**59***	59*	59*	1.41	**59***	59*	59*	1.74	**59***	59*	59*	0.68	**59***	59*	59*	0.15
B05	**61***	61*	61*	2.11	**61***	61*	61*	2.63	**61***	61*	61*	0.86	**61***	61*	61*	0.22
B06	**122.8**	122*	124	3.49	124	124	124	4.11	123.1	122*	125	1.80	125	125	125	0.41
B07	**111***	111*	111*	3.39	**111***	111*	192	3.81	**111***	111*	111*	1.86	**111***	111*	111*	0.26
B08	**104***	104*	104*	6.11	105.2	104*	107	5.02	**104***	104*	104*	2.80	107	107	107	0.55
B09	**220***	220*	220*	9.38	**220***	220*	220*	8.68	**220***	220*	220*	4.92	**220***	220*	220*	1.11
B10	**86***	86*	86*	4.9	**86***	86*	86*	7.39	86.5	86*	91	2.39	88.5	86*	91	0.73
B11	**88***	88*	88*	6.31	**88***	88*	88*	10.12	88.1	88*	89	2.93	89.6	88*	90	0.79
B12	**174***	174*	174*	14.86	**174***	174*	174*	14.56	**174***	174*	174*	6.76	**174***	174*	174*	1.75
B13	168.6	165*	172	14.06	**166.6**	165*	169	18.96	169.7	165*	173	6.98	172	172	172	2.23
B14	**235.2**	235*	236	21.01	236	236	236	47.4	235.4	235*	236	8.57	236	236	236	1.78
B15	**319.8**	318*	320	37.39	320	320	320	41.21	321.2	318*	322	16.56	321	321	321	2.94
B16	**127***	127*	127*	16.9	**127***	127*	127*	26.38	128	127*	132	6.41	**127***	127*	127*	3.06
B17	**131***	131*	131*	28.93	**131***	131*	131*	24.79	**131***	131*	131*	10.3	**131***	131*	131*	4.44
B18	**218***	218*	218*	41.12	**218***	218*	218*	34.41	218.4	218*	219	19.28	218.1	218*	219	3.76

Table 3. Performance of *GRASP-VNS*, *Multi-VNS*, *GRASP-CST* and *VNSMR2* for Steiner tree problems with Δ_1 =1.1**Delay*(T_{OPT})

No.	Δ	*GRASP-VNS*				*Multi-VNS*				*GRASP-CST*				*VNSMR2*			
		Avg.	**Best**	**Worst**	**Time(s)**	**Avg.**	**Best**	**Worst**	**Time(s)**	**Avg.**	**Best**	**Worst**	**Time(s)**	**Avg.**	**Best**	**Worst**	**Time(s)**
B01	145	**82***	82*	82*	0.48	**82***	82*	82*	0.46	**82***	82*	82*	0.54	**82***	82*	82*	0.07
B02	228	**83***	83*	83*	1.45	**83***	83*	83*	1.95	**83***	83*	83*	0.8	**83***	83*	83*	0.18
B03	248	**138***	138*	138*	2.14	**138***	138*	138*	2.05	**138***	138*	138*	1.09	**138***	138*	138*	0.26
B04	173	**59***	59*	59*	1.23	**59***	59*	59*	1.71	**59***	59*	59*	0.64	**59***	59*	59*	0.14
B05	125	**61***	61*	61*	2.01	**61***	61*	61*	2.64	**61***	61*	61*	0.81	**61***	61*	61*	0.21
B06	281	**122.2**	122*	124	3.58	124	124	124	3.9	123	122*	124	1.73	125	125	125	0.41
B07	212	**111***	111*	111*	3.36	**111***	111*	111*	3.77	**111***	111*	111*	1.83	**111***	111*	111*	0.26
B08	209	**104***	104*	104*	6.35	104.6	104*	107	5.44	**104***	104*	104*	2. 8	107	107	107	0.55
B09	280	**220***	220*	220*	9.33	**220***	220*	220*	8.6	**220***	220*	220*	4.87	**220***	220*	220*	1.11
B10	262	86.5	86*	91	4.91	**86***	86*	86*	7.51	87	86*	91	2.20	88.5	86*	91	0.73
B11	235	**88***	88*	88*	6.7	88.2	88*	90	9.74	88.3	88*	90	2.92	89.7	88*	91	0.9
B12	225	**174***	174*	174*	15.87	**174***	174*	174*	12.52	175	174*	178	6.37	**174***	174*	174*	2.07
B13	190	169.4	165*	172	15.48	**167**	165*	169	19.05	170.6	165*	173	6.67	172	172	172	2.23
B14	221	**235***	235*	235*	15.5	244.1	236	245	14.28	235.2	235*	236	8.19	236	236	236	1.81
B15	308	**319.5**	318*	320	36.22	320	320	320	38.76	319.8	318*	320	17.94	321	321	321	5.27
B16	291	**127***	127*	127*	17.06	**127***	127*	127*	24.93	130	127*	132	6.12	**127***	127*	127*	3.07
B17	219	131.5	131*	132	25.13	**131***	131*	131*	24.21	131.9	131*	132	9.64	132	132	132	4.46
B18	425	**218.1**	218*	219	40.32	**218.1**	218*	219	34.37	**218.1**	218*	219	19.54	218.4	218*	219	3.77

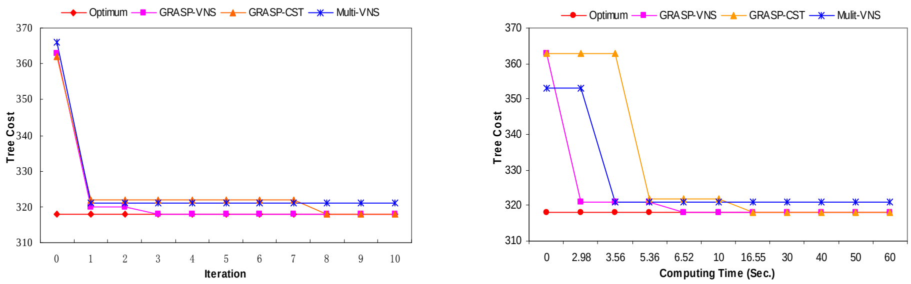
In the third group of experiments, we set the delay bound Δ_2 to a smaller value *Delay*(T_{OPT}), thus the optimal solutions are not known to any of the cases. Table 4 shows that again *GRASP-VNS* outperforms *Multi-VNS* and *GRASP-CST* upon the average tree costs on 13 instances. *GRASP-VNS*, *GRASP-CST* and *VNSMR2* could not find the feasible solutions on 2 instances (B03, B07) due to the tighter delay bound. The *Multi-VNS* also failed to find the feasible solutions on instance B07 and B14. The *GRASP-CST*, *Multi-VNS*, *GRASP-CST* and *VNSMR2* obtained 13, 8, 5 and 5 best solutions out of 16 instances, respectively. With regards to the average computing time for the instances, *Multi-VNS* requires longer time (13.872 seconds) than *GRASP-VNS* (11.803 seconds), *GRASP-CST* (5.333 seconds) and *VNSMR2* (1.464 seconds).

Table 4. Performance of *GRASP-VNS*, *Multi-VNS*, *GRASP-CST* and *VNSMR2* for Steiner tree problems with $\Delta_2 = 0.9 * Delay(T_{OPT})$. "\" denotes that no feasible solution was obtained.

No.	Δ	*GRASP-VNS*				*Multi-VNS*				*GRASP-CST*				*VNSMR2*			
		Avg.	Best	Worst	Time(s)	Avg.	Best	Worst	Time(s)	Avg.	Best	Worst	Time(s)	Avg.	Best	Worst	Time(s)
B01	118	**83**	83	83	0.42	**83**	83	83	0.31	**83**	83	83	0.46	**83**	83	83	0.06
B02	187	**84**	84	84	0.77	**84**	84	84	1.43	**84**	84	84	0.63	**84**	84	84	0.09
B03	203	\	\	\	\	**142**	142	142	1.81	\	\	\	\	\	\	\	\
B04	142	**62.4**	62	64	1.26	63.8	62	64	2.75	62.6	62	65	0.65	66.2	62	76	0.2
B05	102	**62**	62	62	1.52	62.1	62	63	2.26	64.7	62	65	0.54	**62**	62	62	0.26
B06	199	**123.9**	123	124	3.81	124.3	124	125	4.8	124	124	124	1.58	129	129	129	0.4
B07	173	\	\	\	\	\	\	\	\	\	\	\	\	\	\	\	\
B08	171	**107**	107	107	5.75	**107**	107	107	5.19	**107**	107	107	2.59	**107**	107	107	0.55
B09	229	**221**	221	221	8.68	**221**	221	221	9.15	**221**	221	221	5.05	**221**	221	221	1.05
B10	215	88.3	88	91	5.08	**88**	88	88	7.72	88.3	88	91	2.12	89.2	88	91	0.8
B11	180	**89.2**	89	91	6.48	91.5	89	93	9.75	**89.2**	89	90	3.09	91	90	93	1.05
B12	184	**176.8**	176	177	12.95	177.2	177	179	16.41	178.5	177	189	6.37	197.6	177	202	1.73
B13	139	171.2	169	172	15.72	**169**	169	169	19.99	172	168	173	6.62	173	173	173	1.38
B14	180	**237.2**	237	239	16.66	\	\	\	\	242.1	237	243	6.83	239	239	239	3.49
B15	194	**327.1**	326	329	38.84	328.7	326	329	40.27	330.1	328	331	15.62	341.6	339	342	7.27
B16	238	**131.1**	129	132	12.84	132	132	132	26.99	131.7	129	132	5.46	132	132	132	1.8
B17	180	134.2	134	135	20.98	**134**	134	134	33.53	134.8	134	135	8.34	135	135	135	4.36
B18	348	**219.2**	219	220	38.93	219.8	219	220	39.59	219.6	219	220	18.55	219.5	219	220	3.73

Further experiments are designed to test how the algorithms evolve within a given period of time. Examples of the evolution process of these algorithms on one instance (B15) in the above *steinb* problems are shown in Fig.3. Here, we set the delay bound to a very large number so as not to act as a constraint. In Fig.3.(a), we can see that our *GRASP-VNS* algorithm converged faster than *GRASP-CST* and *Multi-VNSMR2* in the given 10 iterations. The *GRASP-VNS* algorithm finds the optimal solution at iteration 3, *GRASP-CST* finds the optimal solution at iteration 8, while *Multi-VNS* does not find the optimum within the 10 iterations. Furthermore, the three algorithms were tested by giving the exact same amount of time (60 seconds). Fig.3.(b) shows that our *GRASP-VNS* still converged faster than *GRASP-CST* and *Multi-VNS* on the instance B15.

All experiment results demonstrate that the *GRASP-VNS* algorithm has the overall best performance compared with *Multi-VNS*, *GRASP-CST* and *VNSMR2* in terms of the average tree cost for this set of benchmark problems. The proposed *GRASP-VNS* algorithm, which applies GRASP metaheuristic by building the RCL in the construction phase, outperforms the simple multi-start *Multi-VNS* algorithm with only randomized initial solutions in terms of both average tree cost and computing time in most cases. Both *GRASP-VNS* and *Multi-VNS*, the extension of our previous *VNSMR2* algorithm, outperform *VNSMR2*, indicating the two multi-start algorithms are more robust and efficient than the single *VNSMR2* algorithm with only one variable neighborhood search phase when exploring the solution space of hard problems.



(a) Tree cost vs. iterations (b) Tree cost vs. computing time

Fig. 3. Evolution process of the three algorithms on instance B15

4.2 Experiments on random graphs

In order to compare our *GRASP-VNS* algorithm with other existing algorithms, we tested *GRASP-VNS* on a group of randomly generated graphs which are the same simulation data tested in our previous work [11]. These random graphs include three random topologies for each network size (10, 20, 30, 40, 50, 60, 70, 80, 90, 100 nodes). The link costs are set depending on the length of the link; all the link delays are set to 1. The group size was set to 30% of the network size in each graph, the delay bounds were set to different values depending on the network sizes (Δ = 7 for network size 10-30, Δ = 8 for network size 40-60, Δ = 10 for network size 70-80 and Δ = 12 for network size 90-100). The simulation was run 10 times on each random graph.

Table 5. Average tree costs of our *GRASP-VNS* and some existing heuristics and algorithms on the random graphs

Algorithms		Average Tree Cost
Heuristics	KPP1 [12]	905.581
	KPP2 [12]	911.684
	BSMA [16]	872.681
GA-based Algorithms	Wang et al. [22]	815.969
	Haghighat et al. [23]	808.406
TS-based Algorithms	Skorin-Kapov and Kos [25]	897.875
	Youssef et al. [24]	854.839
	Wang et al. [26]	869.291
	Ghaboosi and Haghighat [27]	739.095
Path relinking	Ghaboosi and Haghighat [28]	691.434
VNS Algorithms	VNSMR1 [11]	680.067
	VNSMR2 [11]	658.967
	Multi-VNS	653.257
GRASP Algorithms	GRASP-CST [9]	669.927
	Our proposed GRASP-VNS algorithm	**649.203**

Table 5 shows that our *GRASP-VNS* algorithm performs the best in terms of the average tree cost. Both *Multi-VNS* and *VNSMR2* get better average tree costs than *GRASP-CST*. Details of the average tree cost, standard deviation and execution time of these three algorithms on each network size are given in Table 6. It shows that *GRASP-VNS* found the best solutions in 8 out 10 network sizes, while *GRASP-CST* and *Multi-VNS* obtained the best results twice on the 10 types of random graphs. With respect to the average standard deviation σ , *GRASP-VNS* has lower σ (5.316) than *GRASP-CST* (7.238), but higher σ than *Multi-VNS* (4.641). To summarize, our *GRASP-VNS* algorithm gives high quality solutions and stable on most of the tested random graphs. *GRASP-VNS* has longer average computing time (11.006 seconds) in comparison with *GRASP-CST* (3.289 seconds), however it is better than *Multi-VNS* which has the longest computing time (12.737 seconds).

Table 6. Average tree cost, standard deviation of the tree cost and execution time of *GRASP-VNS*, *Multi-VNS*, *GRASP-CST* and *VNSMR2* on the random graphs

Network Size	*GRASP-VNS*			*Multi-VNS*			*GRASP-CST*			*VNSMR2*		
	Cost	σ	**Time(s)**	**Cost**	σ	**Time(s)**	**Cost**	σ	**Time(s)**	**Cost**	σ	**Time(s)**
10	**94.67**	0	0.008	**94.67**	0	0.008	**94.67**	0	0.009	**94.67**	0	0.003
20	272.07	2.25	0.085	275.33	0	0.089	**271.13**	1.48	0.048	275.33	0	0.032
30	**392.33**	0	0.353	395.27	3.95	0.461	394.67	0	0.156	399.67	0	0.17
40	**512.8**	1.55	0.857	513.33	0	1.238	526.47	1.79	0.388	514	0	0.362
50	**662.33**	1.94	2.109	665.07	3.92	3.027	697.07	3.43	0.815	674.67	0	0.859
60	**757.33**	13.48	3.894	757.37	14.01	4.894	761.13	17.13	1.625	777.67	0	1.392
70	**780.83**	2.96	9.029	796.2	4.48	9.268	797.53	1.64	2.648	805	0	2.571
80	**868.87**	7.73	19.421	896.2	4.48	16.365	902.67	5.49	5.941	905.33	0	5.127
90	1155.57	19.02	32.621	**1136.23**	11.17	38.76	1201.93	18.02	10.27	1137.67	0	11.705
100	**995.23**	4.23	41.681	1002.9	4.4	53.256	1052	23.4	10.983	1005.67	0	15.332

5. Conclusions

In this paper, we have investigated a *GRASP-VNS* algorithm (Greedy Randomized Adaptive Search Procedure approach with Variable Neighborhood Search) for solving Delay-Constrained Least-Cost multicast routing problems. The problem is a special case of Delay-Constrained Steiner tree (DCST) problem and has been proved to be NP-complete. Although GRASP is an efficient metaheuristic for optimization problems, little attention has been given on applying it for solving the QoS constrained multicast routing problem. A large number of experiments have been carried out on a set of benchmark problems for Steiner tree problems in the OR-library and a group of random graphs. Experiment results show that the proposed algorithm performs the best in comparison with some existing algorithms for all the tested instances in terms of average tree cost. Experiments demonstrate that our *GRASP-VNS* algorithm is able to find high quality solutions for DCST multicast routing problems and efficiently solve benchmark Steiner tree problems.

Many interesting future research directions could be explored. The introduction of multiple QoS constraints, such as the bandwidth, delay-variation or node degrees to the multicast routing problem deserves further investigation. In reality, network scenarios are mostly dynamic with multicast members leaving and joining the multicast group at various times. The adaptation and extension of GRASP approaches to the problem of dynamic multicast routing is worthy of further investigation.

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

References

1. Salama H.F., Reeves D.S., Viniotis Y., Evaluation of multicast routing algorithms for real-time communication on high-speed networks, IEEE Journal on Selected Areas in Communications, 15(3), 332--345 (1997)
2. Yeo C.K., Lee B.S., Er M.H., A survey of application level multicast techniques, Computer Communications, 27(15), 1547--1568 (2004)
3. Masip-Bruin X., Yannuzzi M., Domingo-Pascual J., Fonte A., Curado M., Monteiro E., Kuipers F., Van Mieghem P., Avallone S., Ventre G., Aranda-Gutierrez P., Hollick M., Steinmetz R., Iannone L., Salamatian K., Research challenges in QoS routing, Computer Communications, 29(5), 563--581 (2006)

4. Cheng X., Du D.Z., (eds.) Steiner Trees in Industry, Kluwer Academic Publishers, Dordrecht, Netherlands (2001)
5. Garey M.R., Johnson D.S., Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman and Company, New York (1979)
6. Guo L., Matta I., QDMR: An efficient QoS dependent multicast routing algorithm. In: Proceedings of the 5th IEEE RealTime Technology and Applications Symposium, pp. 213--222 (1999)
7. Diot C., Dabbous W., Crowcroft J., Multipoint communication: a survey of protocols, functions, and mechanisms, IEEE Journal on Selected Areas in Communications, 15(3), 277--290 (1997)
8. Oliveira C.A.S., Pardalos P.M., A survey of combinatorial optimization problems in multicast routing. Computers & Operations Research, 32(8), 1953--1981 (2005)
9. Skorin-Kapov N., Kos M., A GRASP heuristic for the delay-constrained multicast routing problem, Telecommunication Systems, 32(1), 55--69 (2006)
10. http://people.brunel.ac.uk/~mastjjb/jeb/orlib/steininfo.html
11. Qu R., Xu Y., Kendall G., A Variable Neighborhood Descent Search Algorithm for Delay-Constrained Least-Cost Multicast Routing, In: Proceedings of Learning and Intelligent OptimizatioN (LION3), Trento, Italy (2009)
12. Kompella V.P., Pasquale J.C., Polyzos G.C., Multicast routing for multimedia communication, IEEE/ACM Transactions on Networking, 1, 286--292 (1993)
13. Widyono R., The design and evaluation of routing algorithms for realtime channels. Technical Report, ICSI TR-94-024, International Computer Science Institute, U.C. Berkeley (1994)
14. Sun Q., Langendoerfer H., An efficient delay-constrained multicast routing algorithm. Technical Report, Internal Report, Institute of Operating Systems and Computer Networks, TU Braunschweig, Germany (1997)
15. Sun Q., Langendoerfer H., Efficient multicast routing for delay-sensitive applications, In: Proceedings of the 2nd workshop on protocols for multimedia systems, pp. 452--458 (1995)
16. Zhu Q., Parsa M., Garcia-Luna-Aceves J. J., A source-based algorithm for delay-constrained minimum-cost multicasting, In: Proceedings of the 14th Annual Joint Conference of the IEEE Computer and Communication (INFOCOM’95), pp. 377--385. IEEE Computer Society Press, Washington, DC, USA (1995)
17. Kompella V.P., Pasquale J.C., Polyzos G.C., Two distributed algorithms for the constrained Steiner tree problem, In: Proceedings of the 2nd International Conference on Computer Communications and Networking, pp.343--349 (1993)
18. Shaikh A., Shin K., Destination-driven routing for low-cost multicast, IEEE Journal on Selected Areas in Communications, 15, 373--381 (1997)
19. Jia X., A distributed algorithm of delay-bounded multicast routing for multimedia applications in wide area networks, IEEE/ACM Transactions on Networking, 6, 828--837 (1998)
20. Wang X.L., Jiang Z., QoS multicast routing based on simulated annealing algorithm, In: Proceedings international and applications, pp. 511--516 (2004)
21. Zhang K., Wang H., Liu F.Y. Distributed multicast routing for delay and delay variation-bounded Steiner tree using simulated annealing. Computer Communications 28(11): 1356-1370 (2005)
22. Wang Z., Shi B., Zhao E., Bandwidth-delay-constrained least-cost multicast routing based on heuristic genetic algorithm, Computer communications, 24, 685--692 (2001)
23. Haghighat A.T., Faez K., Dehghan M. Mowlaei A., Ghahremani Y., GA-based heuristic algorithms for bandwidth-delay-constrained least-cost multicast routing, Computer Communications, 27, 111--127 (2004)
24. Youssef H., Al-Mulhem, A., Sait S.M., Tahir M.A., QoS-driven multicast tree generation using tabu search, Computer Communications, 25(11-12), 1140--1149 (2002)
25. Skorin-Kapov N., Kos M., The application of Steiner trees to delay constrained multicast routing: a tabu search approach, In: Proceedings of the seventh international Conference on Telecommunications, Zagreb, Croatia (2003)
26. Wang H., Fang J.,Wang H., Sun Y.M., TSDLMRA: an efficient multicast routing algorithm based on tabu search, Journal of Network and Computer Applications, 27, 77--90 (2004)
27. Ghaboosi N., Haghighat A.T., A tabu search based algorithm for multicast routing with QoS constraints, In: 9th International Conference on Information Technology, pp. 18--21 (2006)
28. Ghaboosi N., Haghighat A.T., A path relinking approach for Delay-Constrained Least-Cost Multicast routing problem, In: 19th International Conference on Tools with Artificial Intelligence, pp. 383--390 (2007)
29. Martins S.L., Resende M.G.C., Ribeiro C.C., Pardalos P.M., A parallel GRASP for the Steiner tree problem in graphs using a hybrid local search strategy, Journal of Global Optimization, 17, 267--283 (2000)

30. Ribeiro C.C., Uchoa E., Werneck R.F., A Hybrid GRASP with Perturbations for the Steiner Problems in Graphs, INFORMS Journal on Computing, 14(3), 228--246 (2002)
31. Feo T.A., Resende M.G.C., Greedy randomized adaptive search procedures, Journal of Global Optimization, 6, 109--133 (1995)
32. Resende M.G.C., Ribeiro C.C., Greedy randomized adaptive search procedures, In: Glover F., Kochenberger G. (Eds.), Handbook of Metaheuristics, Kluwer Academic Publishers, Dordrecht. 219--249 (2003)
33. Kontoravdis G., Bard J.F. A GRASP for the vehicle routing problem with time windows. ORSA Journal on Computing, 7, 10--23 (1995)
34. Pardalos P.M., Qian T., Resende M.G.C. A greedy randomized adaptive search procedure for the feedback vertex set problem. Journal of Combinatorial Optimization, 2, 399--412 (1999)
35. Resende M.G.C., Pitsoulis L.S., Pardalos P.M. Fortran subroutines for computing approximate solutions of weighted MAX-SAT problems using GRASP. Discrete Applied Mathematics, 100, 95--113 (2000)
36. Mladenovic N., Hansen P., Variable neighborhood search, Computers & Operations Research, 24, 1097--1100 (1997)