A GRASP Heuristic for the Delay-Constrained Multicast Routing Problem

Nina Skorin-Kapov and Mladen Kos

Department of Telecommunications, Faculty of Electrical Engineering and Computing
University of Zagreb, 10000 Zagreb, Croatia.
nina.skorin-kapov@fer.hr; mladen.kos@fer.hr

Abstract

The increasing development of real-time multimedia network applications, many of which require multiple participants, has created the need for efficient multicast routing algorithms. Examples of such applications include video and tele-conferencing, video-on-demand, tele-medicine, distance education, etc. Several of them require multicasting with a certain Quality of Service (QoS) with respect to elements such as delay or bandwidth. This paper deals with Delay-Constrained Multicast Routing (DCMR) where the maximum end-to-end delay in a multicast session is bounded. The DCMR problem can be reduced to the Constrained Minimum Steiner Tree Problem in Graphs (CMStTG) which has been proven to be NP-complete. As a result, several heuristics have been developed to help solve it. In this paper, we developed a GRASP heuristic for the DCMR problem. Computational experiments on medium sized problems (50-100 nodes) from literature and comparison with existing algorithms have shown that the suggested GRASP heuristic is superior in quality for this set of problems.

Keywords: GRASP, multicast, constrained Steiner Tree, QoS

1. Introduction

Multicast is a mechanism which enables the simultaneous transmission of information to a group of destinations in a network. In other words, it is a technique that *logically* connects a subset of nodes in a network. The development of numerous real-time multimedia applications in the past several years has created an increasing need for this type of distribution of information. Many applications (e.g. video-conferencing, distance education, video-on-demand, and applications in finance) require packets of information to be sent with a certain Quality of Service (QoS). In this paper, we will discuss one of the most important QoS demands which is the demand for a bounded end-to-end delay from the source to any destination in a multicast session. Real-time applications do not allow the end-to-end delay to exceed a certain delay bound, which represents a measure of the quality of service of that application.

In order to support these real-time applications and their respective QoS demands, networks require efficient multicast routing protocols that provide the necessary Quality of Service while minimizing the use of network resources. The routing algorithms used in these protocols usually attempt to find a minimum cost tree that includes the source and all the destination nodes, while attempting to satisfy the delay constraint and other QoS demands. Other QoS demands could include the minimum required bandwidth, the maximum allowed packet loss ratio and the maximum delay jitter. The tree topology is most frequently used, since it enables parallel sending of packets to multiple destinations and duplicating the packets is only necessary where the tree branches.

Multicast routing is often reduced to the Minimum Steiner Tree Problem in Graphs (MStTG). Generally, for a given graph $G = (V, E)$, where V is a set of nodes and E is a set of edges, and a given subset of nodes, $D \subseteq V$, a Steiner tree is one which connects all the nodes in D using a subset of edges in E. This tree may or may not include nodes in $V \setminus D$. Nodes in $V \setminus D$ which are included in the Steiner tree are called Steiner nodes. The MStTG problem deals with searching for such a tree that is of minimal weight in a weighted graph. This basically reduces to searching for the set of Steiner nodes that gives the best solution. Since the MStTG problem has been proven to be NP-complete (Garey and Johnson (1979)), several heuristic algorithms have been developed to solve it suboptimally. Examples of such heuristics are found in Haberman and Rouskas (1997), Kompella, Pasquale, and Plyzos (1993), Zhang and Leung (1999), and Zhu, Parsa, and Garcia-Luna-Aceves (1995).

The MStTG problem can be augmented to include additional constraints giving rise to the constrained MStTG (CMStTG) problem. This paper is concerned with delay-constrained multicast routing (the DCMR

problem). This problem can be reduced to the Constrained Minimum Steiner Tree Problem in Graphs (CMStTG), where the constraint is the maximum end-to-end delay from the source to any destination. This problem refers to the search for the minimum Steiner tree that satisfies a delay constraint. We propose a heuristic algorithm for solving the CMStTG problem based on the GRASP search method. The algorithm was tested on small and medium sized problems (50 - 100 nodes) from SteinLib (Kock, Martin and Voß(2001)), and the results were compared with the results of the $TS-CST$ tabu search algorithm (Skorin-Kapov and Kos (2003)) and Kompella et al.'s centralized CST_C algorithm (Kompella, Pasquale, and Plyzos (1993)). SteinLib is a library of test data which includes optimal solutions for Steiner Tree problems and is available on the WWW. Results indicate that the proposed GRASP method is superior to both algorithms in solution quality. Further testing is required to determine more exact performance measures of this heuristic.

The rest of the paper is organized as follows. In Section 2 we formally define the DCMR problem, followed by a short introduction to GRASP in Section 3. In Section 4 we describe our GRASP heuristic algorithm for the DCMR problem. We introduce the test problem set and the experimental method in Section 5. In Section 6 we summarize the obtained computational results and finish with some concluding remarks in Section 7.

2. The Delay-Constrained Multicast Routing (DCMR) Problem Model

The communication network is modelled as a graph $G = (V, E)$, where V is the set of nodes and E is the set of edges. On the graph G we define the functions $c(i, j)$ and $d(i, j)$, where $c(i, j)$ is the cost of using edge $(i, j) \in E$ and $d(i, j)$ is the delay along edge $(i, j) \in E$. Given is a source node s and a set of destination nodes S, where $\{s\} \cup S \subseteq V$. The upper delay bound on the path from s to any node in S is denoted as Δ. The delay-constrained multicast routing problem (DCMR) searches for a tree $T = (V_T, E_T)$, where $V_T \subseteq V$ and $E_T \subseteq E$, while minimizing the cost of the tree, subject to the following constraints: $\{s\} \cup S \subseteq V_T$ and $D(s, v) < \Delta$ for every $v \in S$, where $D(s, v) = \sum_{i,j} d(i, j)$ for all edges $(i, j) \in E_T$ on the path from s to v in T.

It is important to note that we assume to have centralized information about the network topology. We also assume that the delay of an edge is a constant value which represents the sum of the propagation delay along the edge and the switching delay at the previous node. The cost of an edge is not necessarily proportional to its delay. The cost of an edge can represent various values such as the actual cost or the transfer capacity of the link.

3. The GRASP metaheuristic

GRASP (Greedy Randomized Adaptive Search Procedure) is an iterative metaheuristic used in a wide array of combinatorial optimization problems. Every GRASP iteration consists of two phases: a construction phase, followed by a local search phase. The construction phase builds a feasible solution by applying a randomized greedy algorithm. The randomized greedy algorithm builds a solution by iteratively creating a candidate list of elements that can be added to the partial solution and then randomly selecting an element from this list. Creating this candidate list, called the restricted candidate list (RCL), is done by evaluating the elements not yet included in the partial solution with a certain greedy function that depends on the specifications of the problem. Only the best elements according to this evaluation are included in the RCL. The size of this list can be limited either by the number of elements or by their quality with respect to the best candidate element. After every iteration of this greedy algorithm, the restricted candidate list is updated. The construction phase ends when all the elements needed to create a feasible solution are included. This solution is usually of good quality and offers fast local convergence as a result of the *greedy* aspect of the algorithm used. Since this greedy algorithm is *randomized*, exploration of the solution space is diversified.

The solution obtained in the construction phase is not necessarily locally optimal, so a local search phase is applied. This phase uses a local search algorithm which iteratively replaces the current solution with a better neighboring solution until no better solution can be found. This algorithm can use different strategies for neighborhood evaluation and for moving from one feasible solution to another. It can either search for the best neighboring solution or just choose the *first improving* one.

After applying the desired number of GRASP iterations, the best solution found overall is produced as the final result. Success of a particular GRASP metaheuristic depends on a number of different factors. Some of the most important include the efficiency of the randomized greedy algorithm used, the choice of the neighborhood structure, and the neighborhood search technique. A more detailed description of the GRASP procedure is described in Resende and Ribeiro (2003). GRASP algorithms have been used to help solve the Minimum Steiner Tree Problem in Graphs (MStTG) in Martins et al. (1999), Martins et al. (2000), and Ribeiro, Uchoa, and Werneck (2002), along with many other optimization problems. To the best of our knowledge, this method has not been applied to the Constrained MStTG problem or to multicast routing in general.

4. Description of the $GRASP - CST$ Algorithm

While solving the DCMR problem using our GRASP heuristic, the problem is first reduced to the Constrained Minimum Steiner Tree Problem in Graphs (CMStTG). In this problem, the constraint is the maximum end-to-end delay from the source node to each destination in the multicast group.

It has already been mentioned that for a given weighted graph $G = (V, E)$ and a set of nodes $D \subseteq V$, a minimum Steiner tree is such a tree which connects all the nodes in D using a subset of edges in E that give the minimum total weight. The constrained minimum Steiner tree is such a tree which is of minimum weight while satisfying the given constraint(s). In our problem, we distinguish between one source node s and a group of destination nodes S, so for ud $D = \{s\} \cup S$. Nodes in $V \setminus D$, which are included in the constrained Steiner tree, are called Steiner nodes.

4.A. Graph Reductions

Before implementing the GRASP method, reducing the size of the graph in accordance with the specifics of the problem is desirable. If we decrease the number of potential Steiner nodes in the graph, the solution space becomes smaller and there are less potential solutions among which to search. We will apply a few of the standard graph reductions described in Zhou, Chen and Zhu (2000), with a slight modification due to the added delay constraint.

First, we prune the graph of all *non-destination* nodes (nodes in $V \setminus D$) that are of degree 1 since they will surely not be included in the solution. Secondly, we observe that the adjacent edge of every *destination* node that is of degree 1 will always be in the Steiner tree. As a result of this, we can deem the adjacent node of every such destination node as a destination node itself (if it is not already deemed as such). This reduces the size of our problem, since it reduces the number of non-destination nodes among which we have to decide which are to be included in the Steiner tree.

For further reduction, we do the following: for every non-destination node k that is of degree 2 with adjacent nodes i and j, we can replace edges (i, k) and (k, j) from E with one edge (i, j), where $c(i, j) = c(i, k) + c(k, j)$ and $d(i, j) = d(i, k) + d(k, j)$. Node k is then deleted from the graph. If there already exists an edge (i, j) in E, we compare its cost and delay parameters to those of the newly constructed edge. If one of these edges has both a lesser cost and a lesser delay, we can eliminate the other from E. Otherwise, both edges remain in E. This is because for various delay bounds the cheaper edge with the greater delay may not satisfy the delay constraint while the more expensive one might. After performing these reductions, we execute our GRASP search algorithm on the reduced graph.

Begin GRASP
//Initialization:
Input nodes V and E from graph G
Input $s :=$ source node; $S :=$ destination nodes;
$s \cup S = D$;
Input α, Δ, $GraspIt$, $RandSeed$, $ItWithoutImprovement$;
Reduce graph G;
//current incumbent solution
$X := [x_1 \cdots x_{|V \setminus D|}]$, $x_i \in [0, 1]$, $i = 1, \ldots, |V \setminus D|$
C, $D := \infty$; ***//cost and delay of the current incumbent sol***

//the first iteration of GRASP finds the pure greedy solution ($\alpha = 1$)
$X_{pot} := ConstructGreedyRandSol(1, RandSeed, \Delta)$;
if a feasible solution exists (Δ can be met) **then**
  $X := TS - CST(ItWithoutImprovement, X_{pot}, \Delta)$;
  $C :=$ cost of $DCST(X)$;
  $D :=$ delay of $DCST(X)$;
end if

//the remaining iterations of GRASP use $\alpha > 1$
$i := 0$;
while $i < GraspIt - 1$ **do**
  $X_{pot} := ConstructGreedyRandSol(\alpha, RandSeed, \Delta)$;
  if a feasible solution exists (Δ can be met) **then**
    $X_{pot} := TS - CST(ItWithoutImprovement, Xpot, \Delta)$;
    $C_{pot} :=$ cost of $DCST(X_{pot})$;
    $D_{pot} :=$ delay of $DCST(Xpot)$;
    if $C_{pot} < C$ **then**
      $X := X_{pot}$; $C := C_{pot}$; $D := D_{pot}$;
    end if
  end if
end while
endGRASP

Fig. 1. Pseudocode of the $GRASP - CST$ algorithm

4.B. The $GRASP - CST$ Algorithm

We will refer to our GRASP heuristic as the Greedy Randomized Adaptive Search Procedure - Constrained Steiner Tree ($GRASP - CST$) algorithm. As already mentioned, the GRASP method is an iterative meta-heuristic algorithm where each iteration is composed of two phases: the construction phase and the local search phase. The construction phase builds a feasible solution with a randomized greedy algorithm which is further improved by executing a local search algorithm in the local search phase. After executing the desired number of iterations, the best found solution over all the iterations is kept. The efficiency and quality of various GRASP heuristic algorithms vary depending on the design of these two phases.

Potential solutions in our heuristic are potential constrained Steiner trees represented by binary sets consisting of $|V \setminus D|$ bits. Each bit corresponds to a different node in $V \setminus D$. Nodes whose corresponding bits are set to zero in a given configuration are Steiner nodes. Nodes whose corresponding bits are set to 1 are not included in the constrained Steiner tree. Each configuration corresponds to a *potential* constrained Steiner tree because there exists the possibility that for some configurations no constrained Steiner Tree

can be found. Such is the case if a configuration leaves the graph unconnected because then no Steiner tree exists. Another possibility is that for a given configuration, we cannot find a Steiner tree that satisfies the given delay bound. We denote the cost of such solutions as infinite.

The pseudocode of the $GRASP - CST$ algorithm is shown in Fig. 1. Details of the construction and local search phase follow.

4.B.1. The Construction Phase

In order to construct a good starting solution which is feasible with respect to the delay constraint, we do the following: we construct a constrained Steiner tree T which initially consists of only the source node s (i.e. $T = \{s\}$). Next, we create a candidate list by evaluating the cost of adding each destination node not yet included in the solution (nodes in $D \setminus T$) to the existing tree while making sure that the delay from the source to this candidate destination node is less than the given delay bound.

To perform this evaluation, we compute the shortest paths with respect to the cost function from each unconnected destination node to the existing tree (which in this first iteration consists only of the source node) using Dijkstra's single-source shortest paths algorithm (Cormen (1997)). For each node $i \in D \setminus T$, we denote its shortest path to the existing tree with respect to cost as $ShCPath(i)$. We also compute the shortest paths from each destination node to the source node with respect to the *delay* function. This path is denoted as $ShDPath(i)$. These paths can include any unconnected destination or non-destination node in the graph $(V \setminus T)$.

Note that the shortest delay paths are computed with respect to the source node s and not the existing tree T. In other words, they are only computed in the first iteration of the construction phase when $T = \{s\}$. This is because our delay constraint is defined as the maximum end-to-end delay from the source to any destination node in the multicast session. These shortest delay paths computed in this first iteration of the construction phase serve as our 'back up' paths when our shortest cost paths violate the delay constraint. The shortest delay path found for each destination node must satisfy the given delay constraint, otherwise no feasible solution exists.

We define the value of adding each unconnected destination node i to the existing tree, as the cost of its respective shortest *cost* path ($ShCPath(i)$) if this path satisfies the delay constraint, otherwise as the cost of its respective shortest delay path ($ShDPath(i)$). We denote this value as $ConnecCost(i)$ and its respective path as $ConnecPath(i)$. We then sort these candidate nodes with respect to the value of these determined connection costs.

To create a restricted candidate list (RCL), we include only those nodes $i \in D \setminus T$ for which

```
Begin ConstructGreedyRandSol(α, RandSeed, Δ)
T := s;
X_greedyRand := [x_1 ··· x_|V\D|], x_i ∈ [0,1], i = 1,...,|V\D|;
if the delay of ShDPath(i) > Δ, for any i ∈ S then
   exit the GRASP − CST algorithm; // no feasible solution exists
end if
for all i ∈ S do
   Find ShCPath(i) and ShDPath(i) from s;
   if delay of ShCPath(i) < Δ then
      ConnecPath(i) := ShCPath(i);
      ConnecCost(i) := cost of ShCPath(i);
   else
      ConnecPath(i) := ShDPath(i);
      ConnecCost(i) := cost of ShDPath(i);
   end if
end for
while D ⊈ T do
   BestConnecCost := min(ConnecCost(i)), i ∈ D \ T;
   Make RCL of all i ∈ D \ T where ConnecCost(i) ≤ α · BestConnecCost;
   Node k = random(RCL, RandSeed);
   T = T ∪ ConnecPath(k);
   Update ConnecCost and ConnecPath for all D \ T
end while
for all nodes in T \ D do
   Set their corresponding bits in X_greedyRand to 0;
end for
return X_greedyRand;
endConstructGreedyRandSol
```

Fig. 2. Pseudocode of the construction phase of $GRASP - CST$

$ConnecCost(i) \leq \alpha \cdot ConnecCost(j)$, where $\alpha \geq 1$ and $j \in D \setminus T$ for which $ConnecCost(j) \leq ConnecCost(k)$, for every $k \in D \setminus T$. If $\alpha = 1$, then the algorithm is pure greedy. This means that only the node(s) with the least connection cost can be in the RCL. If $\alpha > 1$, the RCL can also include other nodes whose connection costs are good, but not necessarily best.

We now choose a candidate node at random from the RCL. We add this chosen node i, $i \in D \setminus T$, along with all the other nodes found along its respective connection path $ConnecPath(i)$ to tree T. We update the connection costs and paths of the remaining unconnected destination nodes ($D \setminus T$) by computing their shortest paths to any of the newly connected nodes. If any of these computed paths improve their existing connection costs while satisfying the delay constraint, their respective connection costs and paths are updated. This procedure ends when all the destination nodes are included in the tree ($D \subseteq T$).

As already mentioned, the greedy aspect of the construction phase provides good solution quality and fast local convergence. The random aspect of the construction phase enables diversified exploration of the solution space. Diversification allows the search procedure to visit various areas of the solution space that may contain the optimal solution. Since the pure greedy algorithm gives high quality average solutions, our GRASP heuristic is designed in such a way that the first iteration of $GRASP-CST$ performs its construction

phase with $\alpha = 1$ (pure greedy). The remaining $GRASP-CST$ iterations perform their construction phases with $\alpha > 1$. This is done so that we have a pretty good solution even after the first $GRASP-CST$ iteration and then search for an even better one in the remaining number of iterations, depending on how much execution time we are willing to spend. In other words, since there is a trade off between solution quality and execution time, this method ensures that if in a certain situation it is more important to produce a solution in less time, we can run $GRASP-CST$ for only a few iterations, or even one, and we will still get a reasonably good result.

The pseudocode of the construction phase of $GRASP-CST$ is shown in Fig. 2.

4.B.2. The Local Search Phase

Since the feasible solution built in the construction phase is not necessarily locally optimal, applying a local search procedure to find the local optimum is desirable. A better solution might also be close by, but not necessarily local. For this purpose we designed the search phase of $GRASP-CST$ to enable us to explore further than just the local optimum if desired. Local search algorithms usually iteratively replace the current solution with a better neighboring solution until no better solution can be found. $GRASP-CST$ enables us to specify the desired 'number of iterations without improvement' so that the search procedure does not necessarily terminate at the first local optimum.

We use the tabu search heuristic algorithm $TS-CST$ suggested in Skorin-Kapov and Kos (2003) with a modification enabling us to specify the desired number of iterations without improvement. We also modify this algorithm so its initial solution is that obtained in the construction phase of $GRASP-CST$ instead of that suggested in Skorin-Kapov and Kos (2003). Here, we will briefly describe the $TS-CST$ algorithm. As already mentioned, potential solutions are represented by binary sets consisting of $|V \setminus D|$ bits. Each bit corresponds to a different node in $V \setminus D$. Nodes whose corresponding bits are set to zero in a given configuration of bits are Steiner nodes. Nodes whose corresponding bits are set to 1 are not included in the constrained Steiner tree. The neighborhood of a certain potential solution includes all those solutions whose binary sets differ from the chosen solution by exactly one bit. In other words, neighboring solutions are all those solutions obtained by either adding or removing exactly one Steiner node.

In each iteration of the $TS-CST$ algorithm, we start with some current solution, explore all its neighboring solutions and then choose the best neighboring solution which becomes the new current solution in the next iteration. This procedure is called an *move*. $TS-CST$ is a tabu search heuristic, which means that it has a memory structure of variable size called a tabu list which prevents the algorithm from visiting previously visited solutions. Therefore, when exploring the neighborhood of the current solution, those

```
Begin TS-CST(ItWithoutImprovement, X_pot, Δ)
TabuList := {}; i := 0; iter := 0;
X_TS−CST := X_pot; //the initial solution is that found in the construction phase
C_TS−CST := cost of DCST(X_TS−CST);
D_TS−CST := cost of DCST(X_TS−CST);
X_i := X_pot;
while iter < ItWithoutImprovement do
  C_it := ∞; X_it := {};
  for n = 1, ..., |V \ D| do
    if n is not on TabuList then
      X_neighbor = Flip bit n in X_i;
      Evaluate X_neighbor; //find DCST(X_i)
      C_neighbor := cost of DCST(X_neighbor);
      if unfeasible (Δ cannot be met or graph unconnected) then
        C_neighbor := ∞;
      end if
      if C_neighbor < C_it then
        C_it := C_neighbor; X_it := X_neighbor; n_it := n;
      end if
    end if
  end for
  if C_it = ∞ (no feasible neighbor was found) then
    n_it := i modulo |V \ D| + 1;
    add n_it to TabuList;
  else
    X_i := X_it;
  end if
  if C_it < C_TS−CST then
    X_TS−CST := X_it, C_TS−CST = C_it, D_TS−CST = D_it;
  else
    iter = iter + 1; //increment iterations without improvement
  end if
  Add n_it to TabuList; i := i + 1; //total iterations performed (with or without improvement)
end while
return X_TS−CST;
endTS-CST
```

Fig. 3. Pseudocode of the local search phase of $GRASP - CST$

neighboring solutions that are forbidden by the tabu list are ignored. Following every iteration or move, the tabu-list is updated circularly by adding the last performed move (or some attribute of this move) to the list and removing the oldest member. For our purposes, the size of tabu-list is set to one which is enough to prevent the algorithm from oscillating between neighboring solutions.

In order to evaluate each neighboring solution and select the best one to become the current solution in the next iteration, the following is done: First all the non-Steiner non-destination nodes (that is, those nodes whose corresponding bits are set to 1) are eliminated from graph G along with all their adjacent edges. Next, a spanning tree of the remaining graph that attempts to minimize the cost while satisfying the delay constraint is found. This is referred to as the Delay Constrained Spanning Tree (DCST). To find the DCST, a modified version of Prim's Minimum Spanning Tree algorithm (Cormen (1997)) is used so as to yield a solution in which the end-to-end delay from the source to every destination node is less than the given delay

bound Δ. The tree initially consists of only the source node. Then the algorithm subsequently searches for the closest node to the existing tree by examining all its adjacent edges. That edge which is cheapest but whose addition to the tree does not exceed the delay bound is chosen and added to the existing tree. The procedure is finished when all the nodes are included in the tree. The value of each neighboring solution is defined as the cost of the found Delay Constrained Spanning Tree.

In each iteration of the $TS-CST$ algorithm, the cost of the corresponding DCST of each neighbor of the current solution is found (except for those forbidden by the tabu list), and the best among them is chosen to pass into the next iteration. This solution does not necessarily have to improve the current solution. If it does not, we increment the number of iterations performed without improvement. If in some iteration i no feasible solution in the neighborhood of the current solution exists, we choose a non-feasible neighbor in a pseudo-random manner to become the new current solution in order to prevent the algorithm from getting stuck. This can be done in various ways. In our algorithm, we chose to flip the n^{th} bit of the current solution, where $n = (i) modulo (|V \setminus D| + 1)$, and this becomes the new current solution in the next iteration. For a more detailed description of the $TS - CST$ algorithm refer to Skorin-Kapov and Kos (2003). After running $TS - CST$ for the desired number of iterations without improvement the algorithm ends. If we set the number of iterations without improvement to 1, we find the local optimum. If this number is greater than one, we expand the search beyond the local optimum.

The pseudocode of the local search phase of $GRASP - CST$ is shown in Fig. 3.

5. The Set of Test Problems and the Experimental Method

We implemented $GRASP - CST$, along with the $TS - CST$ algorithm (Skorin-Kapov and Kos (2003)) and Kompella et al.'s centralized CST_C algorithm (Kompella, Pasquale, and Plyzos (1993)), in C++. We tested the above mentioned algorithms on problem set B from Steinlib (Kock, Martin and Voß(2001)) using the experimental method suggested in Skorin-Kapov and Kos (2003) which will be briefly described. All three algorithms were executed on a PC powered by a Pentium 2 450MHz processor.

Steinlib is a publicly available library of test data for the Minimum Steiner Tree Problem in Graphs (MStTG). Since the Delay-Constrained Multicast Routing (DCMR) problem that is the topic of this paper reduces to the Constrained MStTG problem the test data as such is not sufficient. Since the edges in the test data have only a cost function assigned, their respective delay values are generated randomly. Set D is the set of nodes given in the test data that must be spanned by the Steiner tree. The first node in set D is chosen to serve as our source s. The remaining nodes in $D \setminus \{s\}$ are destination nodes S.

Table 1. Characteristics of the problem set and the solution quality obtained while simulating the MStTG problem ($\Delta = \infty$)

Probl.	$\|V\|$	$\|D\|$	$\|E\|$	C_{opt}	*GRASP-CST*		*TS-CST*		*CSTc*	
					$\delta_{GRASP\text{-}CST}$ (%)	$D_{GRASP\text{-}CST}$	$\delta_{TS\text{-}CST}$ (%)	$D_{TS\text{-}CSTC}$	δ_{CSTc} (%)	D_{CSTc}
B01	50	9	63	82	0	30	0	30	0	30
B02	50	13	63	83	0	55	0	55	8.43	55
B03	50	25	63	138	0	78	0	78	1.45	78
B04	50	9	100	59	0	58	0	58	0	58
B05	50	13	100	61	0	43	1.64	39	4.92	26
B06	50	25	100	122	0	93	0	93	4.92	65
B07	75	13	94	111	0	51	0	51	0	51
B08	75	19	94	104	0	49	0	49	0	49
B09	75	38	94	220	0	66	0	66	2.27	51
B10	75	13	150	86	0	66	0	66	13.95	78
B11	75	19	150	88	0	65	11.36	91	4.55	75
B12	75	38	150	174	0	66	0	75	0	125
B13	100	17	125	165	0	38	0	38	6.06	53
B14	100	25	125	235	0	70	1.28	80	1.28	70
B15	100	50	125	318	0	81	0	81	2.52	77
B16	100	17	200	127	0	63	7.09	95	7.87	64
B17	100	25	200	131	0	59	1.53	71	2.29	66
B18	100	50	200	218	0	113	0	113	3.67	80

The algorithms were then run with a high enough value of the delay bound so as not to act as a constraint. (The delay bound Δ cannot actually be set to ∞ since the time complexity of the CST_C algorithm is $O(\Delta|V|^3)$). These obtained solutions are really the solutions to the (unconstrained) MStTG problem since the delay values of the edges play no role in constructing the Steiner tree. If the cost of the obtained solution is that supplied by the test data, we know that it is optimal. After running each algorithm, the cost of the obtained Steiner tree along with the maximum end-to-end delay from the source to any destination is calculated. The deviation (δ) of the calculated cost above the optimal cost supplied by the test data and the corresponding maximum end-to-end delay (D) are shown in Table 1.

The inhibiting factor in the Delay-Constrained Multicast Routing problem is, of course, the value of the delay bound. This means that the smaller the delay bound, the stronger the constraint. For this reason, the following is done: The smallest of the three corresponding maximum delay values found for each test

Table 2. Solution quality for $\Delta_1 = min(D_{GRASP-CST}, D_{TS-CST}, D_{CST_C}) + 1$

		GRASP-CST			*TS-CST*			CST_C		
Probl.	Δ_1	$C_{GRASP-CST}$	$D_{GRASP-CST}$	$T_{GRASP-CST}$ (s)	C_{TS-CST}	D_{TS-CST}	T_{TS-CST} (s)	C_{CSTc}	D_{CSTc}	T_{CSTc} (s)
B01*	31	**82***	30	0.170	82*	30	0.290	82*	30	0.591
B02*	56	**83***	55	0.199	83*	55	0.310	90	55	1.152
B03*	79	**138***	78	0.329	138*	78	0.410	140	78	1.692
B04*	59	**59***	58	1.041	59*	58	2.664	75	58	1.421
B05	27	**62**	26	1.292	76	26	3.143	63	26	0.561
B06	66	**126**	65	1.762	126	63	2.403	128	65	1.571
B07*	52	**111***	51	0.430	111*	51	0.890	118	51	3.153
B08*	50	**104***	49	0.480	104*	49	0.661	110	39	3.034
B09	52	231	48	0.830	231	48	0.670	225	51	3.134
B10*	67	**86***	66	2.733	86*	66	12.467	106	51	4.606
B11*	66	**88***	65	2.254	92	61	14.020	99	57	4.516
B12*	67	**174***	66	6.057	180	54	11.315	182	66	4.717
B13*	39	**165***	38	1.361	165*	38	2.913	187	38	5.197
B14*	71	**235***	70	1.252	239	64	3.634	238	70	9.874
B15	78	330	61	1.702	330	61	3.444	328	71	10.975
B16*	64	**127***	63	4.737	149	58	33.478	146	54	9.624
B17*	60	**131***	59	8.862	131*	59	33.569	165	50	9.033
B18	81	**219**	80	11.175	219	80	24.404	226	80	12.367

problem while simulating the MStTG problem (Table 1), is chosen. This value is then incremented by 1, and set as delay bound Δ_1. The cost (C) and maximum delay values (D) that correspond to the Steiner trees obtained by testing the algorithms with delay bound Δ_1 , along with their execution times (T), are shown in Table 2. The algorithms are then tested for two more delay bounds: Δ_2 (Table 3) and Δ_3 (Table 4). Δ_2 is 10% greater than Δ_1, rounded up to the nearest integer, while Δ_3 is 10% less than Δ_1, rounded down to the nearest integer.

Since the $GRASP - CST$ algorithm gave the optimal solution to all 18 test problems for the MStTG problem, we know the maximum delay that corresponds to all of the optimal solutions. As a result, if the delay bound of the CMStTG problem is set to a value *greater* than the maximum delay of the optimal solution to the MStTG problem, we know that this optimal solution to the MStTG problem is also optimal for the CMStTG problem. Such problems are marked with '*****' in Tables 2 and 3 to let us know that the

Table 3. Solution quality for $\Delta_2 = 1.1 \cdot \Delta_1$

		GRASP-CST			*TS-CST*			CST_C		
Probl.	Δ_2	$C_{GRASP\text{-}CST}$	$D_{GRASP\text{-}CST}$	$T_{GRASP\text{-}CST}$ *(s)*	$C_{TS\text{-}CST}$	$D_{TS\text{-}CST}$	$T_{TS\text{-}CST}$ *(s)*	C_{CSTc}	D_{CSTc}	T_{CSTc} *(s)*
B01*	35	**82***	30	0.160	82*	30	0.281	82*	30	0.680
B02*	62	**83***	55	0.199	83*	55	0.310	90	55	1.302
B03*	87	**138***	78	0.331	138*	78	0.410	140	78	1.872
B04*	65	**59***	58	1.031	59*	58	2.634	64	58	1.571
B05	30	**62**	26	1.252	76	29	3.114	66	27	0.631
B06	73	**124**	72	1.961	124	72	2.463	128	65	1.752
B07*	58	**111***	51	0.450	111*	51	0.901	118	51	3.554
B08*	55	**104***	49	0.509	104*	49	0.670	110	39	3.374
B09	58	**221**	57	0.850	221	57	0.740	225	51	3.534
B10*	87	**86***	66	3.044	86*	66	12.447	99	63	6.078
B11*	73	**88***	65	2.283	92	61	14.009	93	57	5.128
B12*	74	**174***	66	5.227	174*	66	11.456	175	71	5.127
B13*	43	**165***	38	1.391	165*	38	2.923	187	38	5.768
B14*	79	**235***	70	1.351	238	74	4.004	238	70	11.085
B15*	86	**318***	81	2.674	318*	81	3.854	322	50	12.677
B16*	71	**132**	50	4.987	149	58	33.709	137	64	10.754
B17*	66	**131***	59	8.573	134	59	32.858	148	49	9.914
B18	90	**219**	80	10.867	222	84	25.045	226	80	13.839

optimal solution for these cases is known. For problems where we do not know the optimal solution, we simply compare the performance of the three implemented algorithms. Unfortunately, to the best of our knowledge, there is no test data available for the CMStTG problem.

To determine appropriate values for the input parameters for the $GRASP-CST$ algorithms, a number of experiments were performed. The goal was to use a small number of GRASP iterations and a small number of iterations in the local search phase to reduce execution time, and yet obtain good solutions for this set of problems. Regarding the local search procedure, we first set the number of iterations without improvement to 1 to make it a strictly local neighborhood search. However, the neighborhood of the $TS-CST$ algorithm used in the local search procedure proved too restrictive. One of the reasons for this is the neighborhood structure of the $TS-CST$ algorithm. Namely, since potential solutions are represented by a set of Steiner nodes, and the neighborhood is defined as all those solutions where the status of only a single node is changed, the neighborhood of a current solution often consists of all infeasible solutions (i.e. unconnected

Table 4. Solution quality for $\Delta_3 = 0.9 \cdot \Delta_1$

		GRASP-CST			*TS-CST*			CST_C		
Probl.	Δ_3	$C_{GRASP\text{-}CST}$	$D_{GRASP\text{-}CST}$	$T_{GRASP\text{-}CST}$ **(s)**	$C_{TS\text{-}CST}$	$D_{TS\text{-}CST}$	$T_{TS\text{-}CST}$ **(s)**	C_{CSTc}	D_{CSTc}	T_{CSTc} **(s)**
B01	27	**-**	-	-	-	-	-	-	-	-
B02	50	**91**	43	0.281	91	43	0.260	91	43	1.062
B03	71	**144**	59	0.400	144	59	0.360	155	70	1.761
B04	53	**62**	35	0.860	64	42	2.554	80	48	1.232
B05	24	**66**	19	1.221	75	21	3.184	66	18	0.480
B06	59	138	58	1.302	127	53	2.604	135	52	1.091
B07	46	**-**	-	-	118	33	0.900	128	32	2.813
B08	45	**107**	36	0.429	107	34	0.720	111	34	2.703
B09	46	**-**	-	-	-	-	-	-	-	-
B10	60	**88**	51	2.464	91	57	12.798	100	51	4.106
B11	59	**89**	43	2.263	94	57	13.629	99	57	4.116
B12	60	**177**	56	6.329	190	58	12.046	200	57	4.217
B13	35	**172**	28	0.971	-	-	-	217	34	4.607
B14	63	**240**	62	1.292	246	55	3.524	243	50	8.953
B15	70	**330**	61	2.293	330	61	3.454	330	63	9.974
B16	57	**129**	51	5.877	153	55	33.919	146	54	8.502
B17	54	**136**	53	7.209	158	53	25.316	159	50	8.272
B18	72	**223**	67	11.816	227	69	25.716	228	70	10.914

trees). Allowing more flexibility drastically improved results. To provide this flexibility, we set the number of iterations without improvement to 2 and, thus, allowed 2 nodes to be changed with regards to their status as Steiner nodes before obtaining a solution better than the current one. This little nudge beyond the first local optimum significantly improved results. Of course, raising the value of this parameter even further could potentially lead to even better solutions but testing indicated that the gain on solution quality was not significant with respect to the increase in execution time. Also, for several of the cases tested, the obtained solutions were optimal. Thus, further raising this value seemed unnecessary.

Regarding the remaining parameters, it was necessary to determine a good balance between parameter α and the number of GRASP iterations. Parameter α should be large enough to enable a diversified search and yet small enough to intensify the search around good solutions. Recall that candidates in the restricted candidate list (RCL) are chosen according to the cost of adding them to the existing tree, i.e. if the cost of adding a node is less than or equal to the cost of the best candidate multiplied by factor α, the node is

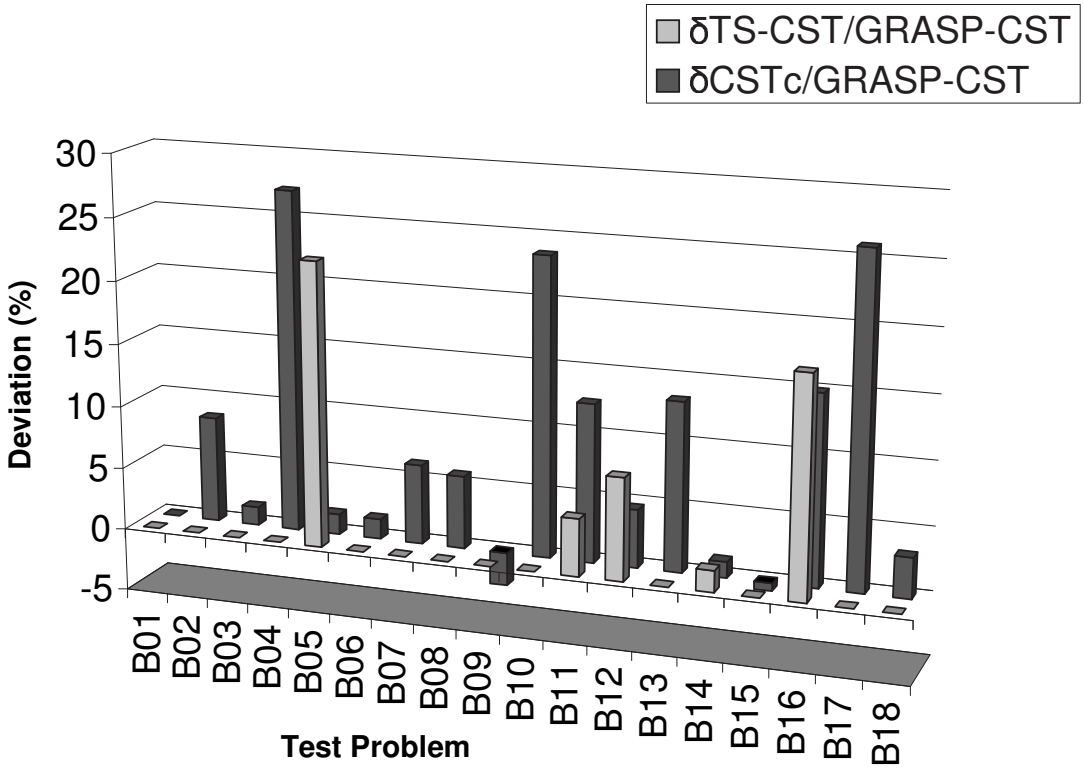

Fig. 4. Deviation of the cost of the solutions obtained by $TS-CST$ and CST_C over $GRASP-CST$ for Δ_1

included in the RCL. Since the costs on edges in the networks ranged from 1 to 10, the costs of the paths connecting various nodes to the existing tree often varied significantly. As a result, setting α to a value close to 1 proved fairly restrictive, resulting in a construction phase that ran almost like a pure greedy algorithm. This caused most of the GRASP iterations run to give the same solution. For most cases, this solution was good but it could still be improved. Tests showed that setting α to 5 provided enough flexibility in the construction phase to enable the $GRASP-CST$ algorithm to perform a diversified search. Values higher than 5 often obtained poor quality solutions in the construction phase and, thus, a large number of GRASP iterations had to be run to obtain good solutions. When parameter α was set to 5, only 5 iterations of $GRASP-CST$ were required to obtain high quality results for this problem set.

It follows that the results shown in Tables 1-4 are those obtained with the following input values: the number of GRASP iterations is set to 5, α is set to 5, and the number of iterations without improvement of the local search procedure is set to 2. Parameters for testing the $TS-CST$ algorithm are those chosen in Skorin-Kapov and Kos (2003) where the number of iterations for problems B01-B09 is 25, while the

remaining problems are run for 40 iterations.

For easier visualization of the obtained results, the deviation of the cost of the solutions found by the $TS-CST$ and CST_C algorithms above the cost of the corresponding solution obtained by the $GRASP-CST$ algorithm for the middle delay bound (Δ_1) are shown in Fig. 4. The average deviation of the cost of the constrained Steiner tree obtained by the $TS-CST$ algorithm over that obtained by the $GRASP-CST$ algorithm ($\delta_{TS-CST/GRASP-CST}$) is +3.01%. In the case of the CST_C algorithm ($\delta_{CST_C/GRASP-CST}$), the average deviation is +8.25%.

6. Computational Results

In Table 1, we can see that for the unconstrained multicast routing problem (reduced to the MStTG problem), $GRASP-CST$ gave the optimal solution in *all* cases, while the $TS-CST$ and CST_C algorithms found the optimal solution in 13 and 5 cases, respectively. These results indicate that the suggested GRASP heuristic is efficient for the general problem of multicast routing. Regarding QoS multicasting with a bounded end-to-end delay, Tables 2, 3 and 4 show the results of the algorithms for the CMStTG problem with various delay bounds. $GRASP-CST$ performed better than both the $TS-CST$ and CST_C algorithms for all three delay bounds. For Δ_1, $GRASP-CST$ gave better or equal solutions (marked in bold) for 16 out of 18 problems. For Δ_2, this was the case for all 18 problems, while for Δ_3, $GRASP-CST$ performed better or equal to the $TS-CST$ and CST_C algorithms for 16 out of 18 problems.

Regarding optimality, we can see from Table 2 that for Δ_1, $GRASP-CST$ obtained the optimal solution (denoted as '**') in all 13 cases where the optimal solution is known. The $TS-CST$ algorithm did so in 9 cases, while CST_C did so in only 1 case. For the problems for which the optimal solution is not known, we compare the obtained results with lower bounds. Namely, the optimal solutions for the unconstrained minimum Steiner tree problem (shown in Table 1) represent lower bounds on the solutions for the constrained problem. For Δ_1, the *maximum* deviation of a solution obtained by the $GRASP-CST$ algorithm over its corresponding lower bound was 5.00%. This occurred for problem B09. The largest deviations of solutions obtained by the $TS-CST$ and CST_C algorithms over the corresponding lower bounds were 24.59% (problem B05) and 27.12% (problem B04), respectively. Note that for problem B04, the optimal solution is known. Therefore, this deviation is the deviation over the optimal solution, and not just the lower bound.

We can see from Table 3 that for Δ_2, $GRASP-CST$ found the optimal solution in all but one of the 13 cases where the optimal solution is known. The $TS-CST$ algorithm found the optimal solution in 9 cases, while the CST_C algorithm in 1 case. For Δ_2, the solution obtained by the $GRASP-CST$ algorithm

deviated most over the lower bound (in this case, the optimal solution) for problem B16. $GRASP-CST$ gave a solution more expensive by 5 units of cost (3.94%). The $TS-CST$ algorithm deviated most for problem B05, giving a solution more expensive by 15 units of cost (24.59%). The maximum deviation of the CST_C algorithm was for problem B10. The obtained solution was 13 units (15.12%) more expensive than the optimal solution.

For the smallest delay bound, Δ_3, the optimal solutions are not known for any of the cases so we compare with lower bounds from Table 1. We can see from the results in Table 4, that for Δ_3 the $GRASP-CST$ algorithm deviated over the lower bound (for cases when a feasible solution was found[A]) most by 13.11% for problem B06. For cases where a feasible solution was found, the $TS-CST$ algorithm deviated most over the lower bound for problem B05, i.e. by 22.95%. The CST_C algorithm did so for problem B04 where it obtained a solution 35.59% more expensive than the lower bound. All these results indicate that the $GRASP-CST$ algorithm is more robust than $TS-CST$ and CST_C, and *consistently* gives high quality solutions.

Comparing the execution times of the algorithms is difficult since both $GRASP-CST$ and $TS-CST$ can be terminated at any time depending on the desired number of iterations. CST_C on the other hand ends deterministically. Even so, for the chosen number of iterations, $GRASP-CST$ performed better than, or equal, to $TS-CST$ for all but one case with respect to solution quality *and* all but two cases where both algorithms found a feasible solution with respect to execution time. Recall that the local search phase of $GRASP-CST$ uses the $TS-CST$ algorithm. Comparison of the execution times of the algorithms tested evidently shows that fewer iterations of $TS-CST$ are run in the local search phase of $GRASP-CST$ than in the $TS-CST$ algorithm itself as run in Skorin-Kapov and Kos (2003), and yet $GRASP-CST$ obtains better solutions. This shows that the construction phase of $GRASP-CST$ often gives good solutions and that the local search phase converges quickly. This is one of the main advantages of the GRASP metaheuristic.

The execution time of $GRASP-CST$ did not exceed 12 seconds for even the largest problems, while the execution time of $TS-CST$ ran up to 33.919 seconds and yet produced a solution of inferior quality. For the chosen number of iterations, $GRASP-CST$ also performed better than the CST_C algorithm in solution quality as well as in execution time. For each delay bound, $GRASP-CST$ was faster for all but two problems where both algorithms found a feasible solution. The average execution time of the $GRASP-CST$ algorithm run for the above specified number of iterations over all the tested problems for all three delay bounds was 2.722 seconds. The average execution time for the $TS-CST$ algorithm was 8.696 seconds, while

[A] Note that there are cases for all three algorithms where no feasible solution was found. Thus, the deviation over the lower bound for these cases is infinite.

the CST_C algorithm on average ran for 5.012 seconds. We can see that the $GRASP - CST$ algorithm gives superior solutions in less time than both $TS - CST$ and CST_C for this set of problems.

7. Conclusion

In this paper we proposed a GRASP heuristic algorithm for solving the Delay-Constrained Multicast Routing problem. In the past couple of years there has been an increased development of numerous multimedia network applications, many of which transfer information in real-time interactive environments to a group of users. Many of these applications can tolerate only a bounded end-to-end delay and therefore require delay-constrained multicast routing algorithms.

In the proposed algorithm, the Delay-Constrained Multicast Routing problem is first reduced to the Constrained Minimum Steiner Tree problem and then the GRASP method is applied. Testing on small and medium sized problems available in SteinLib has shown that the proposed algorithm gives near-optimal solutions in moderate time for this set of problems. The results were also compared to those obtained by a tabu-search algorithm (Skorin-Kapov and Kos (2003)) and Kompella et al.'s centralized algorithm (Kompella, Pasquale, and Plyzos (1993)) for the same problem. The proposed GRASP heuristic algorithm outperforms both of the above mentioned algorithms for this problem set.

GRASP (Greedy Randomized Adaptive Search Procedure) is a metaheuristic proven to be efficient for a wide array of optimization problems. This search procedure seems little used in research dealing with QoS-driven multicast routing. The encouraging results obtained in this paper indicate that further research in this field could be useful. Introducing multiple QoS demands to the multicast routing problem such as the minimum bandwidth or the maximum delay jitter could be interesting for further avenues of research. The adaptation of GRASP strategies to the problem of dynamic multicast routing, or rather *re-routing* when multicast members join or leave the group during the lifetime of the connection, could also prove interesting.

References and Links

[1] T.H. Cormen, C.E. Leiserson, and R.L. Revest. (1997). I*ntroduction to algorithms*, Cambridge: MIT Press.

[2] M.R. Garey and D.S. Johnson. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*, San Francisco: Freeman.

[3] B.K. Haberman and G. Rouskas. (1997). *Cost, Delay, and Delay Variation Concious Multicast Routing*, Technical Report TR/97/03, North Carolina State University.

[4] T. Koch, A. Martin, and S. Voß. (2001). *SteinLib: An Updated Library on Steiner Tree Problems in Graphs*. Available online at: http://elib.zib.de/steinlib.

[5] V. P. Kompella, J. C. Pasquale, and G.C. Plyzos. (1993). "Multicast routing problems", *IEEE/ACM Trans. on Networking*, 1(3), 286-292.

[6] S.L. Martins et al. (1999). "Greedy randomized adaptive search procedures for the Steiner problem in graphs", in P.M. Pardalos, S. Rajasejaran and J.Rolim (eds.), *Randomization Methods in Algorithmic Design*, Volume 43 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, American Mathematical Society.

[7] S.L. Martins et al. (2000). "A parallel GRASP for the Steiner tree problem in graphs using a hybrid local search strategy", *Journal of Global Optimization*, 17: 267-283.

[8] M.G.C. Resende and C.C. Ribeiro. (2003). "Greedy randomized search procedures". In F. Glover and G. Kochenberger (eds.), *Handbook of Metaheuristics*, Kluwer Academic Publishers.

[9] C.C. Ribeiro, E. Uchoa, and R.F. Werneck. (2002). "A hybrid GRASP with perturbations for the Steiner problem in graphs", *INFORMS Journal on Computing*, 14: 228-246.

[10] N. Skorin-Kapov and M. Kos. (2003). "The Application of Steiner Trees to Delay/Constrained Multicast Routing: a Tabu Search Approach", *Proc. Of Contel2003 - Conference on Telecommunications*, Zagreb.

[11] Q. Zhang and Y.W. Leung. (1999). "An orthogonal genetic algorithm for multimedia multicast routing", *IEEE Trans. on Evolutionary Computation*, 3(1), 53-61.

[12] X. Zhou, C. Chen and G. Zhu. (2000). "A Genetic Algorithm for Multicasting Routing Problem", *Proceedings of International Conference on Communication Technologies (ICCT2000)*, Beijing.

[13] Q. Zhu, M. Parsa, and J. J. Garcia-Luna-Aceves, (1995). "A source based algorithm for delay-constrained minimum-cost multicasting", *Proceedings of IEEE INFOCOM*, Boston, MA.