The Application of Steiner Trees to Delay Constrained Multicast Routing: a Tabu Search Approach

Nina Skorin-Kapov, Mladen Kos
Dept. of Telecommunications, FER
University of Zagreb
Zagreb, Croatia
{nina.skorin-kapov|mladen.kos}@fer.hr

Abstract— **With today's development of information technology comes the increased development of numerous real-time multimedia network applications. Some examples include video and tele-conferencing, tele-medicine, video-on-demand, distance education, applications in finance, etc. Several of these applications require multicasting with a certain Quality of Service (QoS). One of the most important QoS parameters is the maximum end-to-end delay from the source to any destination in a multicast session. This paper deals with the problem of Delay-Constrained Multicast Routing (DCMR). The DCMR problem can be reduced to the Constrained Minimum Steiner Tree Problem in Graphs (CMStTG). Since the Minimum Steiner Tree Problem in Graphs (MStTG) has been proven to be NP-complete, several heuristics have been developed for solving MStTG and CMStTG. In this paper, we suggest a tabu search heuristic for the DCMR problem. This heuristic was developed on the basis of a tabu search heuristic designed for solving unconstrained minimum Steiner tree problems. Preliminary testing on data from a publicly available library, SteinLib, has shown that this heuristic gives near optimal solutions in moderate time and a moderate number of iterations for medium sized problems (50-100 nodes). Comparing with a well known algorithm for solving the CMStTG problem, tests have shown that our tabu search heuristic is superior in quality for medium sized problems.**

Keywords- multicast, constrained Steiner Tree, tabu search, QoS

I. INTRODUCTION

Multicast is a mechanism that enables information to be sent from one source to a group of destinations. In other words, it is a technique that *logically* connects a subset of nodes in a network. The development of multimedia applications in the past several years has created an increasing need for this type of distribution of information. Many applications, such as video-conferencing, distance education, video-on-demand, etc., require packets of information to be sent with a certain Quality of Service (QoS) . Among the most important QoS demands are those concerning bandwidth and delay. In this paper we will concentrate on the demand concerning end-to-end delay. 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.

To support these types of multimedia applications, specifically to support a large number of multicast sessions, networks require efficient routing algorithms. These algorithms must provide the neccessary Quality of Service while minimizing the use of network resouces. In a given network, multicasting usually consists of finding 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 frequentley used, since it enables parallel sending of packets to multiple destinations and duplicating the packets is only neccessary 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 $V \backslash D$. Nodes in $V \backslash D$ 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 wieght in a weighted graph. This basically reduces to searching for the set of Steiner nodes that gives the best solution. Since the MStTG problem belongs to the NP-complete class of problems [3], several heuristic algorithms have been developed to solve it suboptimally. Examples of such heuristics are found in [5], [6], [8], [9] and [11].

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 contraint 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 suggest a heuristic algorithm for solving the CMStTG problem implementing the tabu search method. Our heuristic was motivated by a tabu search heuristic suggested by Leguesdron, Levendovsky, Molnar and Vegso [7] for solving the MStTG problem. In our

approach, we incorporated a delay constraint which makes the problem more complex. The algorithm was tested on data from SteinLib ([4]), and compared to the results of Kompella et al.'s centralized algorithm CST_C [6]. SteinLib is a library of test data which includes optimal solutions for Steiner Tree problems and is available on the WWW. After testing on small and medium sized problems (50 – 100 nodes), results indicate that the proposed tabu search method is superior to Kompella et al.'s algorithm in solution quality. Further testing is required to determine more exact performance measures of this heuristic.

In Section II we formally define the DCMR problem. In Section III we introduce the tabu search method and describe our heuristic algorithm for the DCMR problem. In Section IV we present the performance analysis of our algorithm. We finish with some concluding remarks in Section V.

II. THE DCMR PROBLEM MODEL

The communication network is modeled 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 $\{s\} \bigcup S \subseteq V$. 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, $\sum_{(i,j) \in E_T} c(i,j)$ subject to the following constraints: $\{s\} \bigcup S \subseteq V_T$ 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.

We 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. We also assume that the cost of an edge is not neccessarily 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.

III. A TABU SEARCH HEURISTIC FOR THE DCMR PROBLEM

Tabu search is a meta-heuristic which guides other heuristics in such a way that they explore various areas of the solution space. As a result, the tabu search method prevents heuristics from remaining in local optimums.

A. The Idea Behind Tabu search

In every iterative step of the tabu search method, we begin with the current solution. We explore its 'neighborhood', and from this neighborhood we select the best possible solution that does not necessarily have to improve the current one. This new solution now becomes the current solution, and this is referred to as a 'move'. We define the 'neighborhood' of the current solution as that part of the solution space that is reachable by applying an elementary transformation to the current solution. These elementary transformations can be defined in numerous ways in various tabu search heuristics and depend on the specifics of the problem.

Certain problems arise while searching for the optimal solution in this manner. Let us suppose that the current solution is a local optimum. In this case, the new current solution obtained in the next move would have to be somewhat worse. The problem arises in the subsequent move because we would be forced to return to our local optimum. In every following iteration, we would alternate between these two solutions. Hence, this heuristic would not enable us to explore various areas of the solution space and would come to a standstill at the first local optimum.

To prevent this occurrence, the tabu search method introduces the idea of 'memory'. The tabu search method 'memorizes' the elementary transformations applied in a number of previous moves. The inverses of these transformations are placed in a so-called 'tabu-list'. While exploring the neighborhood of the current solution in each iteration, those potential solutions that can be reached by applying a transformation on the tabu-list are eliminated. In other words, moves that are the opposite of those already executed are forbidden. This prevents us from going back to an already used solution and getting stuck in a local optimum.

The tabu-list is updated following each move. This is done by adding the inverse of the last executed elementary transformation and removing the oldest one on the list of the list is full. The length of the tabu-list can vary depending on different problems. It is often determined experimentally in order to allow good exploration of the solution space.

B. Description of the Proposed Algorithm

While solving the DCMR problem using our tabu search heuristic, the problem is first reduced to the CMStTG problem. In this problem, the constraint is the end-to-end delay from the source to each destination. The heuristic is further developed through modifications of a heuristic suggested by Leguesdron, Levendovsky, Molnar and Vegso [7] for solving the MStTG problem.

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. In our problem, we distinguish between one source node s and a group of destination nodes S, so for us $D = s \cup S$. Nodes in $V \backslash D$, which are included in the Steiner tree, are called Steiner nodes.

1) Graph Reductions

In accordance with the problem, we can reduce the size of the graph before implementing the algorithm using a few of the standard graph reductions described in [10], with a slight modification due to the added delay constraint. First, we prune the graph of all *nondestinaton* nodes (nodes in $V \backslash D$) 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 nondestination nodes among wich we have to decide which are to be included in the Steiner tree.

For further reduction, we do the following: for every nondestination 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 tabu search algorithm on the reduced graph

2) The Proposed Tabu search Algorithm: TS-CST

We will refer to our tabu search heuristic as the Tabu Search – Constrained Steiner Tree (*TS-CST*) algorithm. Potential solutions in our heuristic are potential constrained Steiner trees represented by binary sets consisting of $|V\backslash D|$ bits. Each bit corresponds to a different node in $V\backslash D$. 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 evaluation of a potential solution starts by eliminating the non-Steiner non-destination nodes (that is, those nodes whose coresponding bits are set to 1) from graph G along with all their adjacent edges. The next step of the evaluation is to find a Delay Constrained Spanning Tree (DCST) of this modified graph while attempting to minimize its cost. There are exact algorithms for finding the Minimum Spanning Tree (MST) of a given graph in polynomial time. Such a solution, though, may not necessarily satisfy our delay constraint. In this paper, we modified Prim's algorithm for finding the MST 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. Prim's algorithm [1], does the following: the tree initially consists of one randomly selected node and in each following iteration, the closest node to the existing tree is added. This is done by examining all edges adjacent to the existing tree and choosing the cheapest one. The procedure ends when all the nodes are included in the tree. This tree represents the MST of the initial graph. To ensure that our delay constraint is met, we do the following: the tree initially consists of only the source node. When subsequently searching for the closest node to the existing tree, that is while examining all adjacent edges, we choose that edge which is cheapest but whose addition to the tree does not exceed the delay bound. The procedure is finished when all the nodes are in the tree. In every iteration of the *TS-CST* algorithm, to explore a neighborhood, we find the cost of the found DCST for every neighboring solution.

As already mentioned, neighboring solutions with respect to the current one are all those that can be reached by applying an elementary transformation to the current solution. In our heuristic, as in [7], we use a set of $|V\backslash D|$ elementary transformations, where the n-th transformation is defined as changing the value of the n-th bit in the configuration that represents the current solution. In accordance with this definition, the neighborhood of the current solution is a set of all the configurations which differ from the configuration of the current solution by only one bit. This basically means that in one 'move', our new current solution can only add or remove one Steiner node with respect to the previous current solution. Formally, if $X_{i-1}=x_1^{(i-1)}x_2^{(i-1)}...x_{|V\backslash D|}^{(i-1)}$ represents some configuration of bits, where every x_k , $k=1,2,...,|V\backslash D|$ is a bit which corresponds to the k-th node in $V\backslash D$, applying elementary transformation m_n, $n=1,2,...,|V\backslash D|$ to X_{i-1} gives us:

$$X_i = x_1^{(i)}x_2^{(i)}\ldots x_{|V\backslash D|}^{(i)} = m_n(X_{i-1}) =$$
$$= x_1^{(i-1)}\ldots x_{n-1}^{(i-1)}\ \overline{x_n^{(i-1)}}\ x_{n+1}^{(i-1)}\ldots x_{|V\backslash D|}^{(i-1)}. \quad (1)$$

In an i-th move, we select the neighboring solution X_i whose DCST has the minimum cost and which is obtained by applying an elementary transformation to X_{i-1} that is not on the tabu-list. This solution is compared with the best found solution, the current incumbent solution, and the better of the two is kept. The tabu-list is updated after every move by adding the inverse of the elementary transformation applied to X_{i-1} to get X_i and eliminating the oldest member on the list.

It is imporatant to note that there exists the possibility that no feasable solution can be found in the neighborhood of X_{i-1} (the graph is always left unconnected or Δ cannot be satisfied). In this case, it is neccessary to choose an infeasable neighboring solution to become the new current soultion X_i. This solution, although it cannot improve the current incumbent one, enables us to further explore the solution space in the following iteration. In our heuristic, we choose the infeasable neighboring solution $m_n(X_{i-1})$, where $n =(i) \text{ modulo } |V\backslash D|+1$ to become our new current solution X_i.

The initial configuration of the *TS-CST* algorithm is such that all the bits are set to zero. This means that the Steiner tree that corresponds to the initial configuration includes all the nodes in $V\backslash D$. In other words, all the nodes in $V\backslash D$ are Steiner nodes.

A simple description of the *TS-CST* algorithm follows:

```
Begin
//initialization
Input nodes V and edges E from graph G;
Reduce graph G;
//initial configuration
X0 = x1(0)x2(0)..x|V\D|(0), where xk(0)=0, k=1,2,…,|V\D|
Update the graph so as to correspond to
       configuration X0;
X := X0;      //global solution
C := ∞;       //cost of the global solution
```

```
Find the DCST for configuration X using our
       modification of Prim's MST alg;
If DCST(X) exists then
       C := cost of DCST(X); //cost of the global solution
EndIf
TabuList := {};
DelayBound := Δ;
i = 0;

//iterations
While    i < desired number of iterations  do
C_it = ∞ ;
X_it = {};
For n=1, ..., |V\D| do
       If m_n ∉ TabuList, then
              X_i := m_n (X_i-1);
              Update the graph so as to
                     correspond to config X_i;
              Find the DCST for config. X_i using
                     our modification of
                     Prim's MST alg;
              C_n :=  cost DCST(X_i);
              If graph is unconnected or Δ
                     can't be met (DCST doesn't
                     exist), then
                     C_n := ∞;
              EndIf
              If C_n < C_it then
                     C_it := C_n;
                     X_it := X_i;
              EndIf
       EndIf
EndFor
If C_it := ∞ then
       //no feasable neighbor was found
       n :=  i modulo |V\D|+ 1;
       X_i := m_n(X_i-1);
Else
        X_i := X_it;
EndIf
If C_it < C  then
       X := X_it , C = C_it;
EndIf
Add m_n^-1 to the Tabu list;
i := i+1;
EndWhile
End
```

IV. NUMERICAL RESULTS

In this section we describe our experimental method and briefly summarize the obtained results.

A. The Test Data and Experimental Method

The *TS-CST* algorithm, along with Kompella et al.'s centralized algorithm CST_C *[6]*, was implemented in C++ and tested on data from [4]. (A brief description of the CST_C algorithm is given in IV.B.) Both programs were executed on a PC powered by an intel Celeron 600MHz processor with 256MB RAM. The dimensions of the problems are shown in TABLE I. This test data was generated for the Minimum Steiner Tree problem in Graphs (MStTG) with no added constraints. As a result, the edges in the given test problems have assigned only a cost value. Since our problem has an added delay constraint we did the following: first we assigned a randomly generated delay value to each edge. Next we chose the first node in set *D* to serve as our source *s*. Set *D* is the set of nodes given in the test data that must be spanned by the Steiner tree. The remaining nodes in *D\{s}* are destination nodes *S*.

TABLE I. CHARACTERISTICS OF THE PROBLEM SET AND THE OBTAINED SOLUTION QUALITY WHILE SIMULATING THE MSTTG PROBLEM (Δ=∞)

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

Next we set the delay bound to a high enough value so as not to act as a constraint (i.e. we simulated the MStTG problem) and ran both CST_C and *TS-CST*. (The delay bound Δ cannot actually be set to ∞ since the time complexity of the CST_C algorithm is $O(\Delta\ |V|^3)$). If the cost of the obtained solution is that supplied by the test data, we know that it is optimal. We calculated the deviation of the obtained cost above the optimal one and the maximum delay from the source to any destination in the obtained solutions (TABLE I.). In our problem, the smaller the delay bound, the stronger the constraint. Therefore, to better test the performance of the algorithms, we chose the smaller of the found maximum delays and this value, incremented by 1, we set as our delay bound Δ_1 (TABLE II.). We also tested the algorithms with delay bounds 10% greater (Δ_2, TABLE III.) and 10% smaller (Δ_3, TABLE IV.) than Δ_1. Note that if the cost of the found solution to the MStTG problem that corresponds to the chosen maximum delay is optimal, then we know that this is also the optimal solution to the CMStTG problem when the delay bound is Δ_1 or Δ_2. Such problems are marked with * in Tables II and III and help give us a better idea of the quality of our results. It is also possible for the TS-CST algorithm to obtain the optimal solution provided by the test data for a smaller delay bound even though it did not find such a solution when simulating the MStTG problem. (Such is the case for problem B17 for Δ_3 (TABLE IV.).) This is because the algorithm can

TABLE II. COMPARISON OF THE SOLUTION QUALITY FOR $\Delta_1 = MIN\ (DTS\text{-}CST(\Delta=\infty)\ ,\ DCSTC(\Delta=\infty)\ ) + 1$

Probl.	Δ_1	TS-CST			CST_C			
		$C_{TS\text{-}CST}$	$D_{TS\text{-}CST}$	$T_{TS\text{-}CST2}$ (sec)	C_{CSTC}	D_{CSTC}	T_{CSTC} (sec)	$\delta_{CSTC\ /\ TS\text{-}CST}$ (%)
B01*	31	**82***	30	0.189	82*	30	0.331	0
B02*	56	**83***	55	0.249	90	55	0.930	+8.43
B03*	79	**138***	78	0.321	140	78	1.341	+1.45
B04*	59	**59***	58	2.063	75	58	1.112	+2.71
B05	27	76	26	2.484	63	26	0.450	-1.71
B06	66	**126**	63	1.892	128	65	1.232	+1.59
B07*	52	**111***	51	0.690	118	51	2.554	+6.30
B08*	50	**104***	49	0.531	110	39	2.424	+5.77
B09	52	231	48	0.541	225	51	2.523	-2.60
B10*	67	**86***	66	9.653	106	51	3.704	+23.26
B11	76	**92**	61	10.845	92	75	4.276	0
B12*	76	**174***	66	8.993	175	71	4.307	+0.57
B13*	39	**165***	38	2.234	187	38	4.456	+1.33
B14	71	239	64	2.813	238	70	8.021	-0.42
B15	78	330	61	2.624	328	71	8.812	-0.61
B16	65	149	58	25.656	143	64	8.001	-4.02
B17	67	**135**	49	25.095	148	49	8.683	+9.63
B18	81	**219**	80	19.418	226	80	10.064	+3.22
							AVG $\delta_{CSTC\ /\ TS\text{-}CST}$ (%): **+3.07**	

TABLE III. COMPARISON OF THE SOLUTION QUALITY FOR $\Delta_2 = 1.1 * \Delta_1$

Probl.	Δ_2	TS-CST			CST_C			
		$C_{TS\text{-}CST}$	$D_{TS\text{-}CST}$	$T_{TS\text{-}CST2}$ (sec)	C_{CSTC}	D_{CSTC}	T_{CSTC} (sec)	$\delta_{CSTC\ /\ TS\text{-}CST}$ (%)
B01*	35	**82***	30	0.199	82*	30	0.370	0
B02*	62	**83***	55	0.270	90	55	1.020	+8.43
B03*	87	**138***	78	0.321	140	78	1.511	+1.43
B04*	65	**59***	58	2.093	64	58	1.301	+8.47
B05	30	76	29	2.454	66	27	0.510	-13.16
B06	73	**124**	72	1.922	128	65	1.391	+3.26
B07*	58	**111***	51	0.700	118	51	2.873	+6.31
B08*	55	**104***	49	0.550	110	39	2.713	+5.77
B09*	58	**221**	57	0.571	225	51	2.853	+1.81
B10*	87	**86***	66	9.855	99	63	4.907	+15.12
B11	84	93	49	10.304	92	75	4.707	-1.08
B12*	84	182	80	9.834	176	82	4.737	-3.30
B13*	43	**165***	38	2.283	187	38	4.647	+13.33
B14	79	**238**	74	3.114	238	70	9.133	0
B15*	86	**318***	81	3.004	322	50	10.064	+1.26
B16	72	149	58	26.037	137	64	8.913	-8.05
B17	74	**133**	71	25.686	143	57	9.633	+7.52
B18	90	**222**	84	19.889	226	80	10.925	+1.80
							AVG $\delta_{CSTC\ /\ TS\text{-}CST}$ (%): **+2.71**	

be guided differently through the solution space for various delay bounds. For problems where we do not know the optimal solution, we simply compare the performance of the two implemented algorithms. Unfortunately, to the best of our knowledge, there is no test data available for the CMStTG problem. For comparison, we calculated the deviation of each solution obtained by the CST_C algorithm above the corresponding solution obtained by the *TS-CST* algorithm. For easier visualization of the obtained results we present this comparison graphically for the middle delay bound (Δ_1) in Figure 1.

For problems with 63, 94 and 100 edges (problems B01-B09) we ran our algorithm for 25 iterations, while for the remaining larger problems (B10-B18) we ran it for 40 iterations. We recorded the cost, delay, and time for these runs ($C_{TS\text{-}CST}$, $D_{TS\text{-}CST}$, $T_{TS\text{-}CST}$). It is important to keep in mind that $C_{TS\text{-}CST}$ was often reached in a lesser number of iterations in which case the execution time would be shorter.

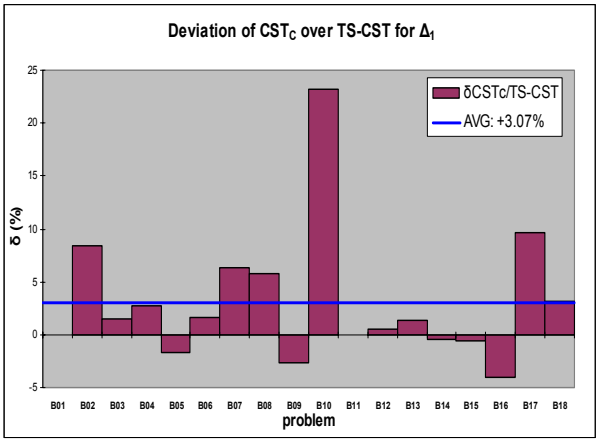

Figure 1. Deviation of the cost of the solution obtained by CSD_C over *TS-CST* for Δ_1

Testing showed that the length of the tabu list was of little importance. Similar results were obtained for lengths 1 to 10. The key is that the list contain at least one element to prevent the algorithm from oscilating between two neighboring solutions. The shown results are those in which the length of the tabu list is set to 1.

B. *Kompella et al.'s CST_C Algorithm*

Kompella et al.'s CST_C algorithm for the CMStTG problem does the following: First it finds the constrained cheapest path for every pair of nodes using a dynamic programming approach similar to Floyd's shortest path algorithm [2]. The constrained cheapest path between two nodes is the path that is of minimum cost while the delay along the path is less than the delay bound. Next, it constructs a *closure graph* which is a complete graph over all the nodes in D (the source and all the destination nodes) where the edge between two nodes represents the constrained cheapest path between those nodes. A constrained spanning tree of this closure graph is then found using a greedy approach. The tree starts with the source node and subsequently adds the cheapest edge that does not violate the delay contraint. Finally, the edges of the contrained spanning tree are expanded into the constrained paths in the original graph and any loops that may be created are removed.

C. *Summary of Results*

For all three delay bounds, the *TS-CST* algorithm performed better than the CST_C algorithm. For Δ_1, *TS-CST* gave better or equal solutions (marked in bold) for 13 out of 18 problems. For Δ_2, this was the case for 14 out of 18 problems, while for Δ_3, *TS-CST* performed better or equal to CST_C for 15 out of 18 problems. In the 21 cases where the optimal solution is known (denoted as *), *TS-CST* found the optimal solution in 19 cases, while CST_C found it only in 2 cases.

It is difficult to compare the speed of these algorithms since the *TS-CST* algorithm can be terminated at any time depending on the desired number of iterations. Of course, the more iterations, the more likely that the solution quality will

TABLE IV. COMPARISON OF THE SOLUTION QUALITY FOR $\Delta_3 = 0.9 * \Delta_1$

Probl.	Δ_3	TS-CST			CST_C			
		C_{TS-CST}	D_{TS-CST}	$T_{TS-CST2}$ (sec)	C_{CSTC}	D_{CSTC}	T_{CSTC} (sec)	$\delta_{CSTC/TS-CST}$ (%)
B01	27	-	-	-	-	-	-	-
B02	50	**91**	43	0.199	91	43	0.861	0
B03	71	**144**	59	0.280	155	70	0.510	+7.64
B04	53	**64**	42	1.992	80	48	0.981	+25.00
B05	24	75	21	2.514	66	18	0.369	-12.00
B06	59	**127**	53	2.042	135	52	1.091	+6.30
B07	46	**118**	33	0.750	128	32	2.244	+8.47
B08	45	**107**	34	0.561	111	34	2.163	+3.74
B09	46	-	-	-	-	-	-	-
B10	60	**91**	57	9.844	100	51	3.294	+9.89
B11	68	**92**	61	10.924	93	57	3.804	+1.09
B12	68	**177**	65	8.622	180	67	3.835	+1.69
B13	35	-	-	-	217	34	3.904	-
B14	63	246	55	2.723	243	50	7.280	-1.22
B15	70	**330**	61	2.713	330	63	7.981	0
B16	58	**146**	57	25.987	146	54	7.069	0
B17*	60	**131***	59	25.666	165	50	7.730	+25.95
B18	72	**227**	69	20.399	228	70	8.833	+0.44
					AVG of found $\delta_{CSTC/TS-CST}$ (%): **+6.41**			

improve. The CST_C algorithm, on the other hand, ends deterministically, and its execution time increases continually with the number of nodes and with the value of the delay bound. The execution time of *TS-CST* depends more on the density of the graph and, therefore, performs better than CST_C for the given runs of 25 and 40 iterations for problems B01-B03, B07-B09, and B13-B15 where the ratio of edges:nodes is smaller. For the remaining problems, the CST_C algorithm is faster. An advatage of *TS-CST* over CST_C is that *TS-CST* can always be terminated earlier. Provided that the delay bound could be met, the TS-CST algorithm produces a result (even if inferior) when terminated earlier, while the CST_C algorithm must run its entire course and only produces a result at the end.

V. CONCLUSION

In this paper we proposed a tabu search heuristic for solving the delay constrained multicast routing problem. This type of routing is needed by several multimedia network applications which require the transfer of information in real-time environments and can therefore tolerate only a bounded end-to-end delay. In the proposed heuristic, the problem is reduced to the Constrained Minimum Steiner Tree problem which belongs to the NP-complete class of problems. The proposed tabu search method explores various areas of the solution space of the given problem. Testing on a group of problems availabe in SteinLib has shown that this heuristic gives near-optimal solutions in moderate time for small and medium sized problems. As a result of preliminary testing and comparing with Kompella et al.'s centralized algorithm for multicast routing for multimedia communication, it is shown that the proposed heuristic gives comparable or better results for this set of problems.

Tabu search is a well known heuristic that has been applied to a wide array of optimization problems although it seems little used in research dealing with multicast routing. Based on the encouraging results of this study, research on further adaptation of tabu search strategies for this class of problems is desirable. Further avenues of research could include the following modifications of the problem: incorporating other QoS constraints such as bandwidth, dealing with variable delay values, and dealing with dynamic multicast routing (multicast members can join and leave the group during the lifetime of the connection).

REFERNCES

[1] T.H. Cormen, C.E. Leiserson, and R.L. Revest, *Introduction to algorithms*, MIT Press, 1997.

[2] R.W. Floyd. *Algorithm 97: Shortest Paths*. Comm. Of the AMC, 5:345, 1962.

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

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

[5] V. Kompella, *Multicast Routing Algorithms for Multimedia Traffic*, PhD thesis, University of California, Sandiego, 1993.

[6] V. P. Kompella, J. C. Pasquale, and G.C. Plyzos, *Multicast routing problems*, IEEE/ACM Trans.on Networking, Vol. 1, No. 3, pp.286-292, 1993.

[7] P. Leguesdron, J. Levendovsky, M. Molnar, and C. Vegso, *Multicast routing with bandwidth requirement in the case if incomplete information as a Steiner Tree problem*, RR INRIA no. 4343, December 2001.

[8] R. Widyono, *The design and evaluation of routing algorithms for real-time channels,* Technical Report TR-94-024, ICSI, University of California, Berkeley, 1994.

[9] Q. Zhang and Y.W. Leung, An orthogonal genetic algorithm for multimedia multicast routing, IEEE Trans. On Evolutionary Computation, Vol. 3, No. 1, pp. 53-61, 1999.

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

[11] Q. Zhu, M. Parsa, and J. J. Garcia-Luna-Aceves, A source based algorithm for delay-constrained minimum-cost multicasting, Proceedings of IEEE INFOCOM, Boston, M.A. pp.377-385, 1995.