12-6-2006

QOS Multimedia Multicast Routing: A Component Based Primal Dual Approach

Faheem Akhtar Hussain

QOS MULTIMEDIA MULTICAST ROUTING
A COMPONENT BASED PRIMAL DUAL APPROACH

by

FAHEEM A HUSSAIN

Under the Direction of Professor Alexander Zelikovsky

ABSTRACT

The QoS Steiner Tree Problem asks for the most cost efficient way to multicast multimedia to a heterogeneous collection of users with different data consumption rates. We assume that the cost of using a link is not constant but rather depends on the maximum bandwidth routed through the link. Formally, given a graph with costs on the edges, a source node and a set of terminal nodes, each one with a bandwidth requirement, the goal is to find a Steiner tree containing the source, and the cheapest assignment of bandwidth to each of its edges so that each source-to-terminal path in the tree has bandwidth at least as large as the bandwidth required by the terminal. Our main contributions are: (1) New flow-based integer linear program formulation for the problem; (2) First implementation of 4.311 primal-dual constant factor approximation algorithm; (3) an extensive experimental study of the new heuristics and of several previously proposed algorithms.

INDEX WORDS: QOS, Multimedia multicast, Steiner tree, QOSST, 4.311 Primal dual, Maxemchuck, Naive PD, Restarting PD, Flow ILP, MST, Rate, Bandwidth

QOS MULTIMEDIA MULTICAST ROUTING
A COMPONENT BASED PRIMAL DUAL APPROACH

by

FAHEEM A HUSSAIN

Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Science

in the College of Arts and Sciences

Georgia State University

2006

QOS MULTIMEDIA MULTICAST ROUTING
A COMPONENT BASED PRIMAL DUAL APPROACH

by

FAHEEM A HUSSAIN

Major Professor:	Alexander Zelikovsky
Committee:	Anu Bourgeois
	Saeid Belkasim

Electronic Version Approved:

Office of Graduate Studies
College of Arts and Sciences
Georgia State University
December 2006

To my mother.

ACKNOWLEDGMENTS

Thanks to my advisor Dr. Alex Zelikovsky for his tireless efforts and help to make this happen. I also would like to thank the other committee members, Dr. Anu Bourgeois and Dr. Saeid Belkasim for their invaluable time spent in reviewing my thesis.

In addition, I would like to thank my colleagues Kelly Westbrooks, Dumitru Brinza and Gulsah Altun for their help in accomplishing my thesis.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Motivation

Recent progress in audio, video, and data storage technologies has given rise to a host of high-bandwidth real-time applications such as video conferencing. These applications require Quality of Service (QoS) guarantees from the underlying networks. Multicast routing algorithms that manage network resources efficiently and satisfy the QoS requirements have come under increased scrutiny in recent years [20]. The focus on multimedia data transfer capability in networks is expected to further increase as applications such as video conferencing gain popularity.

It is becoming apparent that new network mechanisms will be required to provide differentiated quality guarantees for customers with diverse demands. Of particular importance is the problem of optimum multimedia distribution from a source to a disparate collection of users.

Multimedia distribution is usually done via multicast trees. There are two reasons for basing efficient multicast routes on trees: (a) the data can be transmitted concurrently to destinations along the branches of the tree, and (b) only a minimum number of copies of the data must be transmitted since information replication is limited to the forks of the tree [23]. The bandwidth savings obtained from the use of multicast trees can be maximized by using optimal or nearly optimal multicast tree algorithms. Future networks will undoubtedly integrate such algorithms into basic operational performance [4].

1.2 Problem Formulation

Several versions of the QoS multicast problem have been studied in the literature. These versions seek routing tree cost minimization subject to (1) end-to-end delay, (2) delay variation, and/or (3) minimum bandwidth constraints (see, e.g., [4, 18, 14]). This thesis deals with the case of minimum bandwidth constraints, that is, the problem of finding an optimal multicast tree when each terminal possesses a different rate of receiving information. This problem is a generalization of the classical Steiner tree problem and therefore NP-hard [8]. Formally, given a graph $G = (V, E)$, a source s, a set of terminals S, and two functions: $length : E \rightarrow R^+$ representing the length of each edge and $rate : S \rightarrow R^+$ representing the rate of each terminal, a *multicast tree* T is a tree in G spanning s and S. The *rate* of an edge e in a multicast tree T, denoted by $rate(e, T)$, is the maximum rate of a downstream terminal, i.e., of a terminal in the connected component of $T - e$ which does not contain s. The *cost* of a multicast tree T is defined as

$$cost(T) = \sum_{e \in T} length(e) \cdot rate(e)$$

Quality of Service Multicast Tree (QoSMT) Problem: Given a network $G = (V, E, length, rate)$ with source $s \in V$ and set of terminals $S \subseteq V$, find a minimum cost multicast tree in G.

Further it is assumed that the rates belong to a given discrete set of possible rates: $0 = r_0 < r_1 < \ldots < r_N$. The QoSMT problem is equivalent to the Grade of Service Steiner Tree problem [22], which has a slightly different formulation. The network has no source node; edge rates r_e need to be assigned so that the minimum edge rate on the tree path from a terminal with rate r_i to a terminal with rate r_j is at least $min(r_i, r_j)$. Charikar et al. [8] also consider the QoSMT with Priorities problem, where the cost of an edge e is given arbitrarily instead of being equal to the

length times the rate. In other words, edge costs in QoSMT with Priorities are not required to be proportional to edge rates. This generalization seems more difficult – the best known approximation ratio is logarithmic which holds also for multiple multicast groups [8].

1.3 Previous Work

The QoSMT problem was introduced in the context of multimedia distribution over communication networks by Maxemchuk [14]. Maxemchuk suggested a low-complexity heuristic which can be used to build reliable multicast tree in many practical applications. Following Maxemchuk, Charikar et al [8] gave a useful approximation algorithm that finds a solution within $e\alpha$ of the optimal, where $\alpha < 1.550$ is the best approximation ratio of an algorithm for the Steiner tree problem. This is the first known algorithm with a constant approximation ratio for this problem. Finally, an approximation ratio of 3.802 based on accurate estimation of Steiner tree length has been achieved in [13].

Surprisingly, the problem was also previously considered (under different names) in the optimization literature. A number of results for particular instances of the problem were obtained: Current et al. [10] gave an impractical integer programming formulation for the problem and proposed a heuristic algorithm for its solution. Some results for the case of few rates were obtained by Balakrishnan et al. in [1] and [2]. Specifically, [2] (see also [22]) suggested an algorithm for the case of two non-zero rates with approximation ratio of $\frac{4}{3}\alpha < 2.066$. A different approximation algorithm with the factor 1.960 has been proposed in [13]. For the case of three non-zero rates, Mirchandani [15] gave an 1.522-approximation algorithm.

1.4 Contributions and Organization

In Section 3.2 we introduce a mixed integer linear program to find the optimal tree for the QOSST problem. This MILP is feasible upto a a network of 30 vertices. For the networks above this limit we also introduce a linear program to get the lower bound for the problem which work for networks with as many as 100 vertices in a reasonable time. We used the resultant tree of MILP and LP as a benchmark and compared the results of our newly implemented, naturally distributed approximation primal-dual algorithm described in Section 2.2. We also described the previously proposed and implemented algorithms for QOSST problem and compare the performance of all the algorithms. We chose to focus on the primal-dual algorithms due to their simplicity and distributed nature. Contrary to the centeralized algorithms the primal-dual algorithms can work even when the multimedia distibutor does not have the exact knowledge of the network topology. In Section 4.1 we describe a technique for generating random networks. The resulting networks from our generator are uniformly distributed and can be presented as planar graphs. In order to closely observer the behavior of the implemented algorithms we also implemented and network visualization software which is described in Section 4.2. This software package presents the generated networks as planar graphs and gives the user the ability to manipulate the network and view the impact on the behavior of algirthms visually. Finally we conclude with the extensive experimental comparison of several heuristics showing advantage of the primal-dual approach in chapter 6 and 7.

CHAPTER 2

HEURISTICS

2.1 Maxemchuk's Approach

Maxemchuk [14] proposed a heuristic algorithm for the QoS Steiner Tree Problem. His algorithm is a modification of the MST heuristic for Steiner Trees [21] (see Figure 2.1).

The extensive experiments given in [14] demonstrate that this method works well in practice. Nevertheless, the following example shows that the method may produce arbitrarily large error (linear in the number of rates) compared with the optimal tree. Consider the natural generalization of the example in Figure 2.2 with an arbitrary number k of distinct rates. Its optimal solution has a cost of about 1, whereas Maxemchuk's method returns a solution of cost about $(k+1)/2$. As there are $2^{k-1}+1$ nodes, this cost can also be written as $1+\frac{1}{2}\log_2(n-1)$, where n is the number of nodes in the graph. We conclude that the approximation ratio of Maxemchuk's algorithm

---

Input: A graph $G = (V, E, length, rate)$ with a source s in V and a collection of terminals $S \subseteq V$.
Output: A QoS Steiner tree spanning the source and the terminals.

---

(1) Initialize the current tree to $\{s\}$.

(2) Find a non-reached terminal t of highest rate with the shortest distance to the current tree.

(3) Add t to the current tree along with a shortest path connecting it to the current tree.

(4) Repeat until all terminals are spanned.

Figure 2.1. Maxemchuk's Algorithm for the QoS Steiner Tree Problem.

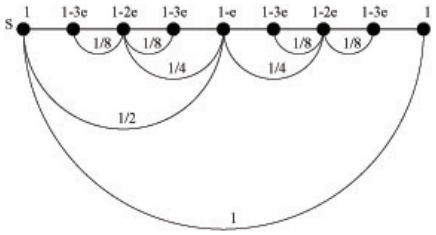

Figure 2.2. A bad example for Maxemchuk's algorithm, with $k = 4$ rates. In the figure, $\varepsilon = 1/2^{2k-1}$. The rate of each node is given above the node. The edge lengths are given on the thin curved arcs, while on the solid horizontal line each segment has length $1/2^{k-1}+\varepsilon$. The optimum, of total cost $1+2^{k-1}\varepsilon = 1+2^{k-1}(1/2^{2k-1}) = 1+1/2^k$, uses the solid horizontal line at rate 1. Maxemchuk's algorithm picks the thin curved arcs at a cost of $1+(1/2)(1-\varepsilon)+2(1/4)(1-2\varepsilon)+4(1/8)(1-3\varepsilon) \geq ((k+1)/2)(1-1/2^k)$.

is no better than linear in the number of rates and no better than logarithmic in the number of nodes in the graph.

2.2 Naive Primal-Dual Method

The primal-dual framework applied to network design problems usually grows uniformly the dual variables associated to the "active" components of the current forest [12]. This approach fails to take into account the different rates of different nodes in the QoS problem. In Figure 2.3 we give a modification, referred to as the "Naive Primal-Dual" algorithm. Our modification takes into account the different rates by varying the speed at which each component grows. While the simulations in the ensuing sections show that this is a good method in practice, the solution it produces on some graphs may be very large compared to the optimal solution, as shown by the following example.

The Frame Example. Consider two nodes of rate 1 connected by an edge of length 1 (see Figure 4.1). There is an arc between these two nodes, and on this arc there is a chain of nodes of rate ϵ. Each two consecutive nodes in the chain are at a distance δ from each other, where $\delta < 1$. Each extreme node in the chain is at a distance $\delta/2$ of its neighboring rate-1 node.

Input: A graph $G = (V, E, length, rate)$ with a source s in V and a collection of terminals $S \subseteq V$.
Output: A QoS Steiner tree spanning the source and the terminals.

(1) Start from the spanning forest of G with no edges.

(2) Grow y_C with speed r_C for each "active" component C of the current forest. (A component C is *inactive* if it contains s and all vertices of rate r_C.)

(3) Stop growing once the dual inequality for a pair (e, r) becomes tight, with e connecting two distinct components of the forest.

(4) Add e to the forest, collapsing the two components.

(5) Terminate when there is no active component left.

(6) Keep an edge of the resulting tree at the minimum needed rate.

Figure 2.3. The Naive Primal-Dual algorithm for the QoS Steiner Tree Problem.

Figure 2.4. The Restarting Primal-Dual avoids the mistake of the Naive Primal-Dual. Part (a) shows duplication of the edges. Part (b) shows the components growing along the respective edges.

The Naive Primal-Dual applied to this graph connects the rate-ϵ nodes first, since $\frac{\delta}{2} < \frac{1}{2}$. So, the algorithm connects the rate-1 nodes via the rate-ϵ nodes, and not via the direct edge connecting them. Thus, the Naive Primal-Dual can make arbitrarily large errors (just take an arbitrarily long chain).

2.3 Restarting Primal-Dual Algorithm

An improved algorithm is given in Figure 2.5. One can easily see that this is a primal-dual algorithm. Indeed, each addition of an edge to the current solution is the result of growing dual variables. Moreover, since the feasibility requirement for edge a is $\Sigma_{a \in \delta(C)} y_C \leq r \cdot length(a)$, this addition preserves the feasibility of the dual solution. The algorithm maintains forests F^{r_i} given by the edges picked at rate r_i,

Input: A Graph $G' = (V, E, cost, rate)$ with source s, and a collection of terminals S.
Output: A QoS Steiner Tree spanning the source and the terminal.

(1) Grow each active C_{r_i} with speed r_i along incident edges (e, r_j), $j \leq i$, picking edges which become tight.

(2) Continue this process until there is no active component of rate r_k.

(3) Remove all edges which are not necessary for maintaining connectivity of nodes of rate r_k.

(4) Accept (keep in the solution) and contract all edges of C_{r_k} (i.e., set their length/cost to 0)

(5) Restart the algorithm with the new graph

Figure 2.5. The Restarting Primal-Dual algorithm for the QoS Steiner Tree Proble.

and the connected components of F^{r_i}, seen as sets of vertices, are denoted in the algorithm by C_{r_i}. Such a component is *active* if $r_{C_{r_i}} = r_i$ and C_{r_i} is disjoint from components of higher rate.

The Restarting Primal-Dual avoids the mistake made by the Naive Primal-Dual on the frame example in Figure 2.4. Then, at time $\frac{\delta}{2}$ the rate-ϵ nodes become connected. This means that $\delta(1-\epsilon)$ of each rate-1 edge between the ϵ-rate nodes is not covered. Meanwhile, the rate-1 nodes are growing on the respective edges as shown in Figure 4.1(b).

Let us assume that the Restarting Primal-Dual uses the chain of rate-ϵ nodes to connect the two rate-1 nodes instead of the direct edge. This would imply that it takes less time to cover the chain, i.e., $\frac{1}{2}\delta(1-\epsilon)n \leq \frac{1}{2} - \frac{\delta}{2}$, where n is the number of rate-ϵ nodes. With ϵ small, we obtain $n\delta \leq 1$, so if the Restarting Primal-Dual uses the chain then it is correct to do so.

2.4 4.311 approximation Primal-Dual algorithm

A primal-dual constant-factor approximation algorithm is obtained based on the enhanced integer linear programming formulation below. It takes into account the

fact that if a set $C \subset V \setminus \{s\}$ is connected to the source with edges of rate $r' > r_C$, then there should be at least **two** edges of rate r' with exactly one endpoint in C. The integer program is

$$\begin{aligned} \min \quad & \sum_{(e,r)\in E'} x_{(e,r)} \cdot r \cdot \mathit{length}(e) \\ \text{s.t.} \quad & \sum_{\substack{e\in\delta(C)\\ r=r_C}} x_{(e,r)} + \frac{1}{2} \sum_{\substack{e\in\delta(C)\\ r>r_C}} x_{(e,r)} \;\geq\; 1, \;\; \forall C \subseteq V \setminus \{s\} \\ & x_{(e,r)} \in \{0,1\} \end{aligned}$$

The corresponding dual of the LP relaxation is

$$\begin{aligned} \max \quad & \sum_{C\subseteq V\setminus\{s\}} y_C \\ \text{s.t.} \quad & \sum_{\substack{C\,:e\in\delta(C)\\ r_C=r}} y_C + \frac{1}{2} \sum_{\substack{C\,:e\in\delta(C)\\ r_C<r}} y_C \;\leq\; r \cdot \mathit{length}(e) \\ & y_C \;\geq\; 0 \end{aligned} \tag{2.1}$$

The core of the algorithm is presented in Figure 2.6. Before that, we do a random bucketing of rates following [8]. Let a be a real (to be picked later) and γ be a real picked uniformly at random from the interval $[0\,..\,1]$. Every node of rate r is replaced by a node of rate $a^{\gamma+j}$, where j is the integer satisfying $a^{\gamma+j-1} < r \leq a^{\gamma+j}$.

The primal-dual part follows the classical framework [12], and works in stages starting from the lower rate to the highest. During the execution of the algorithm, edges are picked at a certain rate (in other words, $x_{(e,r)}$ is set to 1) one by one. Before executing step 3 at rate r for the ith time, the set of edges picked at rate r by the algorithm forms a forest F_i^r. (An edge can be picked at several rates, but it is kept in at most one such rate in the final solution because of the reverse delete step.) A component C of F_i^r is called an *r-component* if $r_C = r$.

Input: A graph $G = (V, E, \mathit{length}, \mathit{rate})$ with source s in V and a collection of terminals $S \subseteq V$.
Output: A QoS Steiner tree spanning the source and the terminal.

(1) For each $r = r_1, r_2, \ldots, r_k$, execute steps 2-6.

(2) Start from the spanning forest F^r of G with no edges.

(3) Grow y_C uniformly for each r-component C of the current forest F^r.

(4) Stop growing once the dual inequality for a pair (e, r) becomes tight, with e connecting two distinct components of F^r.

(5) Add (e, r) to F^r, collapsing two of its components.

(6) Terminate when there is no r-component of F^r left.

(7) Traversing the list of picked edges in reverse order, remove an edge (e, r) from F^r if after (e, r)'s removal the set of edges picked form a feasible tree.

Figure 2.6. The 4.311-approximation algorithm for QoS Steiner Tree.

Using Constraint (2.1), it follows by induction on j that, for an edge e and a rate $a^{\gamma+j}$, we have

$$\begin{aligned} \sum_{\substack{C\,:e\in\delta(C)\\ r_C\le a^{\gamma+j}}} y_C &\le \mathit{length}(e)a^{\gamma+j}\sum_{i=0}^{j}\left(\frac{1}{2a}\right)^i \\ &\le \mathit{length}(e)a^{\gamma+j}\frac{2a}{2a-1}. \end{aligned}$$

For an edge picked by the algorithm at rate r, Constraint (2.1) is tight and therefore

$$\sum_{\substack{C\,:e\in\delta(C)\\ r_C\le a^{\gamma+j}}} y_C \ge \mathit{length}(e)\frac{2a-2}{2a-1}a^{\gamma+j}. \tag{2.2}$$

Exactly as in [12], we have that the number of edges of rate r in the final solution which cross the active r-components at some moment (an edge being counted twice if it crosses two r-components) is at most twice the number of active r-components. Using Equation (2.2) and exactly the same argument as in Theorem 4.2 of [12], we obtain that the cost of the solution of the algorithm is bounded by $(2(2a-1)/(2a-2))\sum y_C \le$

$((2a-1)/(a-1))\,opt$, as any feasible solution for the dual linear program has value at most the value of any feasible solution of the primal.

The same argument as in [8] shows that the approximation ratio of the algorithm above is $(2a-1)/\ln a$. Numerically picking the best value for a, we obtain:

Theorem 2.4.1 *The output cost of the algorithm on Figure 2.6 is at most* 4.311 *times the optimum cost.*

2.4.1 Algorithm Explanation

In order to overcome the NP-completeness of the problems, there must be a compromise on the quality of the solution for the sake of computing a suboptimal solution quickly. Approximation algorithms are polynomial-time heuristics which reach a solution for all instances of the problem with values close to optimum. This closeness can be determined by the approximation ratio, defined for a minimization problem as the maximum value over all instances of the input over the optimal solution value for the instance.

This 4.311-approximation algorithm consists of ten steps, as shown in Figure 1. The basic structure of the algorithm maintains a forest F of edges, which is initially empty (Step 1). The edges of F will be candidates for the set of edges to be output.

A component C_p is said to be active if it contains at least one vertex which is a sink and the source vertex is in a different component, C_q. On the other hand, a component C_p is said to be inactive if all sinks of the same rate have been connected to the component of the higher rate vertices. Function $f(C_p)$ returns either one or zero determining if component C_p is active or inactive, respectively.

The algorithm starts with the lowest rate sinks in one component C_p and the all the higher rate vertices including the source in the other component C_q. The algorithm loops while active components exist in C (Step 5) and, on every iteration, selects an edge e between two distinct connected components (Step 6). Since $f(V) = 0$, the loop will finish after at most $n - 1$ iterations. This edge selection is a key part of the algorithm.

ε is calculated by taking the cost of an edge c_e and subtracting the d values of both its vertices and dividing it by either two or one, depending whether each vertex belongs to an active component or only one of them, respectively. The value of ε could be thought as the growth factor of active components, which is used to increment the value d of all vertices belonging to active components (Step 8). Value d of a vertex

can be thought as a radius of growth for each active component, so they are growing trying to find and satisfy their requirements, whereas inactive components stay static since they are already connected or their requirements have been met.

In Step 9 merged components are added to the set of components C. After no more active components exist, the loop terminates and a set $F^{'}$ of edges is created from only those edges of F needed to connect all the sinks to the source or the higher rate vertices(Step 10). This process of cleaning the forest F is done in the reverse order.

Figure 2.7 Figure and 2.8 show a step by step walkthrough of the 4.311-approximation algorithm being run in parallel over two different instances. Vertices in red are active and vertices in blue are inactive. Every iteration of the algorithm shows the pair of graphs as edges are selected, as well as the d values and epsilons.

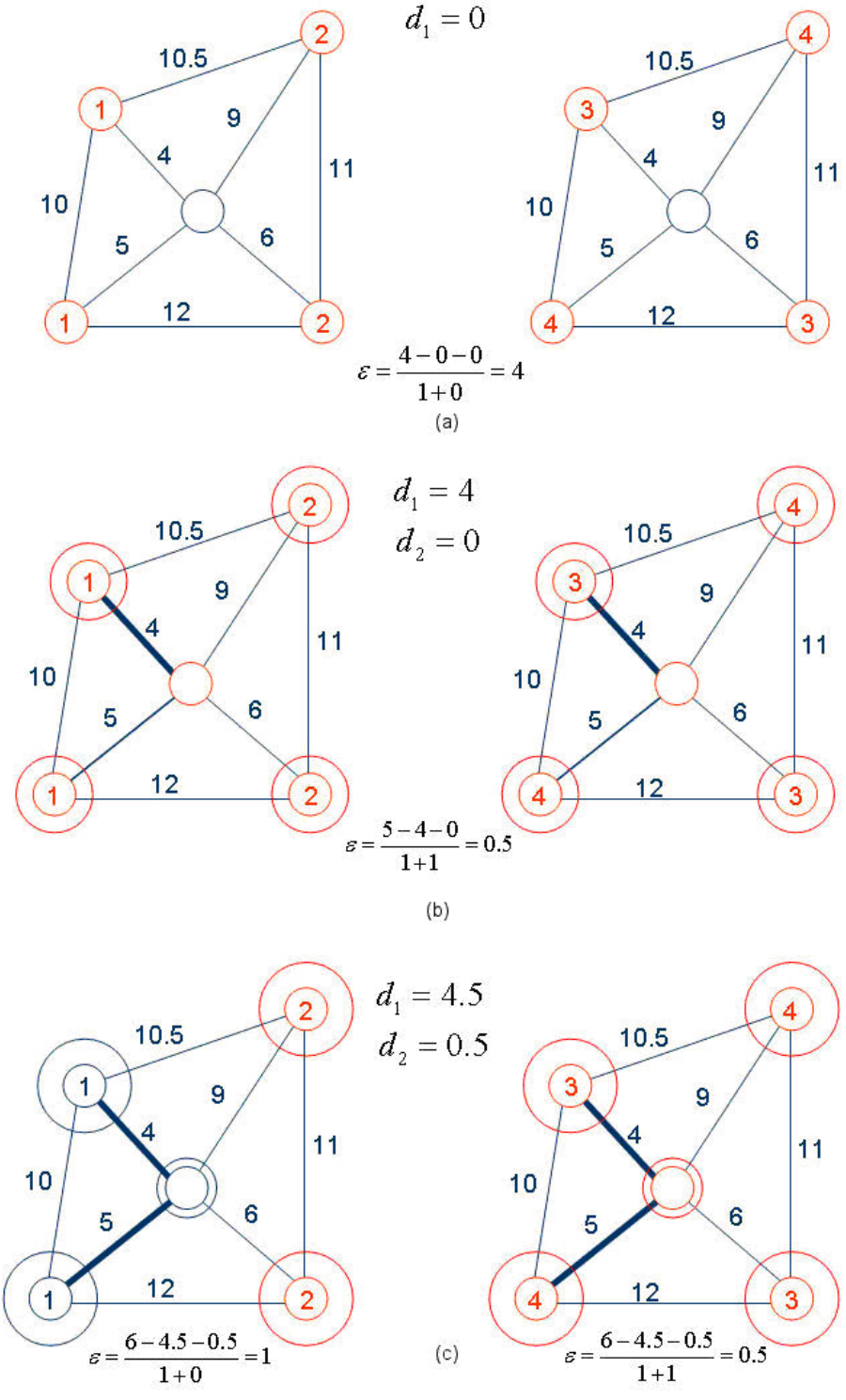

Figure 2.7. The 4.311 approximation algorithm walkthrough.

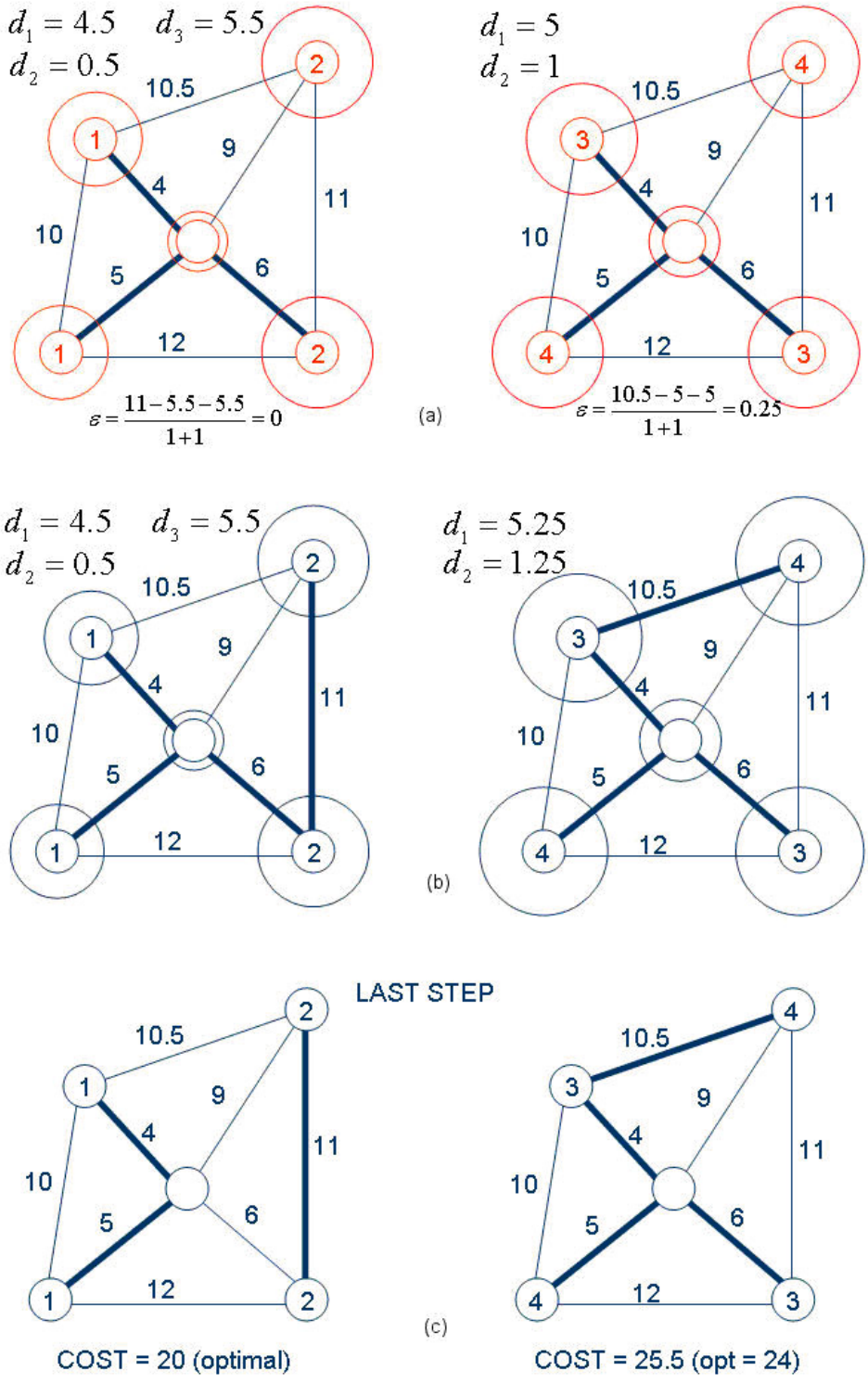

Figure 2.8. Another 4.311 approximation algorithm walkthrough.

CHAPTER 3

AN EXACT INTEGER LINEAR PROGRAM SOLUTION

Linear programming, sometimes known as linear optimization, is the problem of maximizing or minimizing a linear function over a convex polyhedron specified by linear and non-negativity constraints. Linear programming theory falls within convex optimization theory and is also considered to be an important part of operations research. Linear programming is extensively used in business and economics, but may also be used to solve certain engineering problems.

3.1 What is Linear Programming?

For any linear program, there are primal and dual linear program formulations. The primal usually refers to the most natural way to describe the original problem. The dual represents an alternative way to specify the original problem such that it is a minimization problem if the primal is a maximization problem and vice versa. Solving the dual is equivalent to solving the original problem.

Examples from economics include Leontief's input-output model, the determination of shadow prices, etc., an example of a business application would be maximizing profit in a factory that manufactures a number of different products from the same raw material using the same resources, and example engineering applications include Chebyshev approximation and the design of structures (e.g., limit analysis of a planar truss).

Linear programming can be solved using the simplex method which runs along polytope edges of the visualization solid to find the best answer. Khachian (1979)

found a $O(x^5)$ polynomial time algorithm. A much more efficient polynomial time algorithm was found by Karmarkar (1984). This method goes through the middle of the solid (making it a so-called interior point method), and then transforms and warps. Arguably, interior point methods were known as early as the 1960s in the form of the barrier function methods, but the media hype accompanying Karmarkar's announcement led to these methods receiving a great deal of attention.

3.1.1 Application areas

Linear programming is an important field of optimization for several reasons. Many practical problems in operations research can be expressed as linear programming problems. Certain special cases of linear programming, such as *network flow* problems and *multicommodity flow* problems are considered important enough to have generated much research on specialized algorithms for their solution. A number of algorithms for other types of optimization problems work by solving LP problems as sub-problems. Historically, ideas from linear programming have inspired many of the central concepts of optimization theory, such as *duality*, *decomposition*, and the importance of *convexity* and its generalizations. Likewise, linear programming is heavily used in microeconomics and business management, either to maximize the income or minimize the costs of a production scheme.

3.1.2 Standard form

Standard form is the usual and most intuitive form of describing a linear programming problem. It consists of the following three parts:

- A **linear function to be maxmized**

  e.g. maximize $c_1x_1 + c_2x_2$

- **Problem constraints** of the following form

  e.g. $a_{11}x_1 + a_{12}x_2 \leq b_1$

  $a_{21}x_1 + a_{22}x_2 \leq b_2$

  $a_{31}x_1 + a_{32}x_2 \leq b_3$

- Non-negative variables

  e.g. $x_1 \leq 0$

  $x_2 \geq 0$

  The problem is usually expressed in matrix form, and then becomes:

  maximize $c^T x$

  subject to $Ax \leq b, x \geq 0$

  Other forms, such as minimization problems, problems with constraints on alternative forms, as well as problems involving negative variables can always be rewritten into an equivalent problem in standard form.

3.1.3 Integer Linear Programs and Their Relaxations

If the variable is boolean, i.e., $x \in \{0, 1\}$ then the linear program is called boolean or integer linear program (ILP). Unlike linear program, the integer linear program is NP-complete and therefore can not be solved exactly in polynomial time. CPLEX and GLPK have tools (divide and conquer and others) for solving smaller instances of ILP.

The relaxation of ILP is a linear program where the integer constraint $x \in \{0, 1\}$ is replaced with the interval constraint $0 \leq x \leq 1$. For minimization problems (such as QOSST) the relaxation gives a lower bound on the optimal solution given by ILP. In Chapter 6 we will give both exact solutions and lower bounds for smaller instances of QOSST. For larger instances of QOSST (for which ILP can not be solved in reasonable time) we give only the lower bound obtained by solving the relaxation.

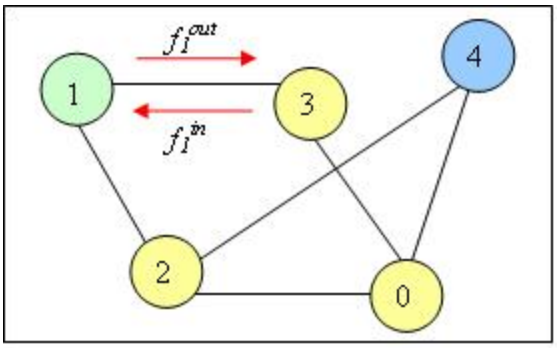

Figure 3.1. Flow diagram

3.2 ILP Formulation for QOSST problem

In this study we have developed and implemented an exact algorithm for QOSST problem based on a flow-base integer linear program. The flow formulation consists on creating a flow per each requirement. The requirement is to connect each sink to the source with the required bandwidth using any intermediate nodes. So having a flow for each pair of a sink and the source will guarantee the connectivity. The starting vertex of the flow would be called the source and the vertex in which the flow ends will be called the sink. Since edges are undirected, for each edge e, two flow variables are used to detect if a flow is an outgoing flow, denoted as $f_{e'}$, or an incoming flow, denoted as f_e, as shown in Figure 3.1.

3.2.1 Variables

In our problem we will have variables for each edge as described in the objective function and will be denoted by x_i. Since the connections will be established using the flows, for each edge $e = (u, v)$ we will have two flow variables i.e. f_e(outgoing from u) and $f_{e'}$(incoming to u). Once again as we know that we have multiple rates with which an edge can be used and so we will need as many flow variables as many rates we will have in our source graph. Therefore for each edge e we will have 2 times the rates, flow variables.

3.2.2 Objective Function

The objective of our ILP program is to minimize the cost of connecting the source to the sinks. If we denote the edges $e_0, e_1, ..., e_n$ with x_i variables and the length of each edge with l_i then the objective will be to minimize the sum of the product of all the edges being used with their length. Now the trick here is that each edge can be used with multiple bandwidths. In this case we will have a separate edge variable for all the rates for each edge. Therefore the objective function will look like this:

$$min \sum_{e=0}^{n} x_e l_e$$

3.2.3 Flow Constraints

To each target we send its own flow from the source. Thus we will have as many flows as many targets. For each flow we have three types of constraints.
For the source v the outgoing flow should exceed incoming flow by 1:

$$\sum_{e \ incident \ to \ v} f_e - \sum_{e \ incident \ to \ v} f_{e'} = 1$$

For each target v the incoming flow should exceed outgoing flow by 1:

$$\sum_{e \ incident \ to \ v} f_e - \sum_{e \ incident \ to \ v} f_{e'} = -1$$

For every intermediate node v the incoming flow should be equal to the outgoing:

$$\sum_{e \ incident \ to \ v} f_e - \sum_{e \ incident \ to \ v} f_{e'} = 0$$

The next step is to link the flow variables with the edge variable so that while mini-

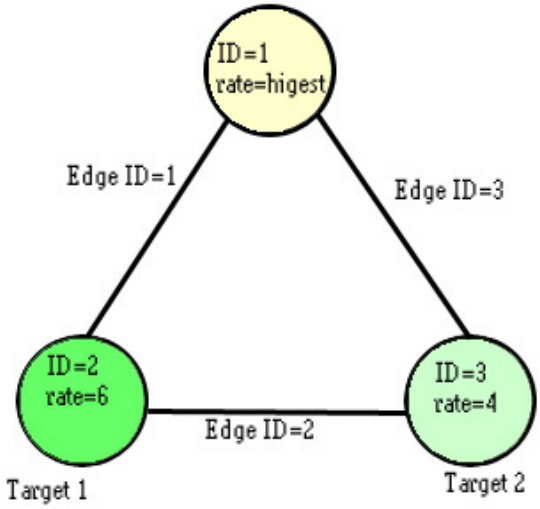

Figure 3.2. An instance of QOSST with source v_1 and two targets v_2 and v_3 with rates 6 and 4, respectively.

mizing the objective function the edge variables can used. Following constraints will link the flow variables with the edge variables.

$$x_e \geq r f_e \qquad \forall \quad r \in R, e \in E$$

$$x_e \geq r f_{e'} \qquad \forall \quad r \in R, e' \in E$$

3.2.4 Example

We implemented this ILP using two different languages. The first choice was CPLEX. Following is an example of a graph with it's model and solution. The graph contains three nodes. One is a source node and two are the sinks with different rate requirements. The graph is given below in the figure.

3.2.4.1 ILP representation of QOSST in CPLEX

```
\Objective function

Minimize
    obj:  x_1_2 +  x_2_3 + 2 x_1_3

\Constraints \ flow constraints = flows x 2 x edges

Subject To \ Flow 2 edge constraints

outflow_e_1_2_2: x_1_2 - 6 f_1_2_2 >= 0 inflow_e_1_2_2: x_1_2 - 6
f_2_1_2 >= 0 outflow_e_2_3_2: x_2_3 - 6 f_2_3_2 >= 0 inflow_e_2_3_2:
x_2_3 - 6 f_3_2_2 >= 0 outflow_e_1_3_2: x_1_3 - 6 f_1_3_2 >= 0
inflow_e_1_3_2: x_1_3 - 6 f_3_1_2 >= 0

\Flow 2 Source constraints outflow_source_2: f_1_2_2 + f_1_3_2 -
f_2_1_2 - f_3_1_2 = 1

\Flow 2 Sink constraints inflow_sink_2: f_2_1_2 + f_2_3_2 - f_1_2_2
- f_3_2_2 = -1

\ Flow 2 intermediate nodes constraints flow_conservation_3: f_3_1_2
+ f_3_2_2 - f_1_3_2 - f_2_3_2 = 0

\ Flow 3 edge constraints outflow_e_1_2_3: x_1_2 - 4 f_1_2_3 >= 0
inflow_e_1_2_3: x_1_2 - 4 f_2_1_3 >= 0 outflow_e_2_3_3: x_2_3 - 4
```

```
f_2_3_3 >= 0 inflow_e_2_3_3: x_2_3 - 4 f_3_2_3 >= 0 outflow_e_1_3_3:
x_1_3 - 4 f_1_3_3 >= 0 inflow_e_1_3_3: x_1_3 - 4 f_3_1_3 >= 0

\Flow 3 Source constraints outflow_source_3: f_1_2_3 + f_1_3_3 -
f_2_1_3 - f_3_1_3 = 1

\Flow 3 Sink constraints inflow_sink_3: f_3_1_3 + f_3_2_3 - f_1_3_3
- f_2_3_3 = -1

\Flow 3 Intermediate nodes constraints flow_conservation_2: f_2_3_3
+ f_2_1_3 - f_3_2_3 - f_1_2_3 = 0

\Binary variables Binary f_1_2_2 f_2_1_2 f_2_3_2 f_3_2_2 f_1_3_2
f_3_1_2

f_1_2_3 f_2_1_3 f_2_3_3 f_3_2_3 f_1_3_3 f_3_1_3

End \EOF }
```

3.2.4.2 ILP representation of QOSST in MathProg

```
# QOS for multimedia distribution # Faheem A Hussain, Dr Alex
Zelikovsky # # This model finds the optimal solution # to connect
the source node to the destination # nodes in the hetrogenous
network
```

```
/* The sets for the model */ set NODES; set EDGEIDS; set EDGES dimen
3; set FLOWEDGES dimen 3; set SINKS;


/* parameters */ param source; param rate {i in NODES}; param length
{i in EDGEIDS};

/* decision variables: xi, i in {1,...,n}
 xi = 1 -> edge is selected
 xi = 0 -> edge is not selected */
var x {i in EDGEIDS} >=0; var flow{i in SINKS, j in NODES, k in
NODES} binary;

/* Objective function */ minimize Cost: sum{i in EDGEIDS}   x[i] *
length[i];

/* Constraints */ s.t. edge_con {i in SINKS, (j,k,l) in EDGES}: x[j]
- ( rate[i] * flow[i,k,l] ) >= 0; s.t. edge_con1 {i in SINKS,
(j,k,l) in EDGES}: x[j] - ( rate[i] * flow[i,l,k] ) >= 0; s.t.
source_constraint {i in SINKS}: sum{(j,k,l) in FLOWEDGES: k ==
source } flow[i,k,l] - sum{(m,n,o) in FLOWEDGES: o == source }
flow[i,n,o] == 1; s.t. sink_constraint {i in SINKS}: sum{(j,k,l) in
FLOWEDGES: k == i } flow[i,k,l] - sum{(m,n,o) in FLOWEDGES: o==i }
flow[i,n,o] == -1;

s.t. intermediate_node {i in SINKS, j in NODES : j != source && j !=
i }: sum{(k,l,m) in FLOWEDGES: l == j } flow[i,l,m] - sum{(n,o,p) in
```

```
FLOWEDGES: p == j } flow[i,o,p] = 0;

data; set NODES := 1 2 3;

set EDGEIDS := 1 2 3;

set EDGES :=  (1,1,2) (2,2,3) (3,1,3);

set FLOWEDGES := (1,1,2) (2,2,3) (3,1,3) (4,2,1) (5,3,2) (6,3,1);

set SINKS := 2 3;

param source := 1;


param rate := 1   6 2   6 3   4;

param length := 1   1 2   1 3   2;
```

3.2.4.3 Output of GLPK

```
Problem: Rows:       18 Columns:    15 (12 integer, 12 binary)
Non-zeros:  48 Status:     INTEGER OPTIMAL Objective:  obj = 10
(MINimum) 10 (LP)

   No.   Row name        Activity     Lower bound   Upper bound
```

```
------ ------------    ------------- ------------- -------------
     1 outflow_e_1_2_2
                                   0             0
     2 inflow_e_1_2_2
                                   6             0
     3 outflow_e_2_3_2
                                   4             0
     4 inflow_e_2_3_2
                                   4             0
     5 outflow_e_1_3_2
                                   0             0
     6 inflow_e_1_3_2
                                   0             0
     7 outflow_source_2
                                   1             1             =
     8 inflow_sink_2
                                  -1            -1             =
     9 flow_conservation_3
                                   0             0             =
    10 outflow_e_1_2_3
                                   2             0
    11 inflow_e_1_2_3
                                   6             0
    12 outflow_e_2_3_3
                                   0             0
    13 inflow_e_2_3_3
                                   4             0
```

```
    14 outflow_e_1_3_3
                                    0             0
    15 inflow_e_1_3_3
                                    0             0
    16 outflow_source_3
                                    1             1             =
    17 inflow_sink_3
                                   -1            -1             =
    18 flow_conservation_2
                                    0             0             =

   No. Column name       Activity     Lower bound   Upper bound
------ ------------    ------------- ------------- -------------
     1 x_1_2                         6             0
     2 x_2_3                         4             0
     3 x_1_3                         0             0
     4 f_1_2_2      *                1             0             1
     5 f_2_1_2      *                0             0             1
     6 f_2_3_2      *                0             0             1
     7 f_3_2_2      *                0             0             1
     8 f_1_3_2      *                0             0             1
     9 f_3_1_2      *                0             0             1
    10 f_1_2_3      *                1             0             1
    11 f_2_1_3      *                0             0             1
    12 f_2_3_3      *                1             0             1
    13 f_3_2_3      *                0             0             1
    14 f_1_3_3      *                0             0             1
```

```
    15 f_3_1_3      *              0             0             1

Integer feasibility conditions:

INT.PE: max.abs.err. = 0.00e+000 on row 0
        max.rel.err. = 0.00e+000 on row 0
        High quality

INT.PB: max.abs.err. = 0.00e+000 on row 0
        max.rel.err. = 0.00e+000 on row 0
        High quality

End of output
```

CHAPTER 4

SOFTWARE PACKAGE

4.1 Planar Graph Generation

The objective is to generate planar graphs as instances of the problem due to the practical planar applications, like VLSI routing or networking connectivity, plus the ease of viewing them in a 2D environment. In order to accomplish this, a Delaunay triangulation is implemented. First we create all vertices of the instance by using a pair of single precision float numbers created randomly, per vertex, as its x and y coordinates. Then a complete graph is created where the cost c_{ij} for every edge is the Euclidian distance from vertex i to vertex j. We visit every edge checking all its intersecting edges, removing those which have a cost greater than or equal to the visited edge. If the visited edge costs more than an intersecting one, then it is removed. When all edges are visited, the resulting graph is planar.

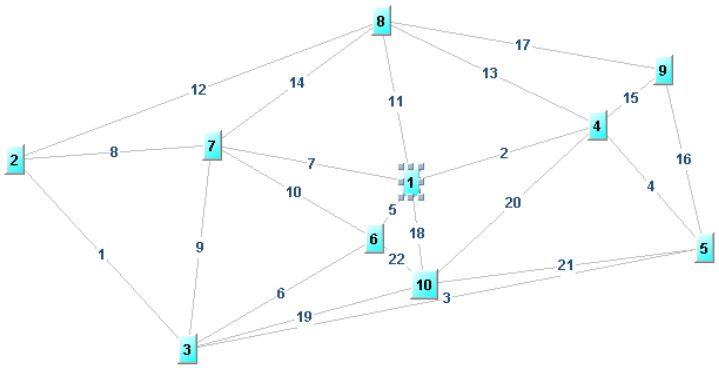

Figure 4.1. The simple planar graph with ten vertices in it. The vertices are not assigned any rates yet, neither have the source and targets been set up in this figure.

4.2 Graphical User Interface

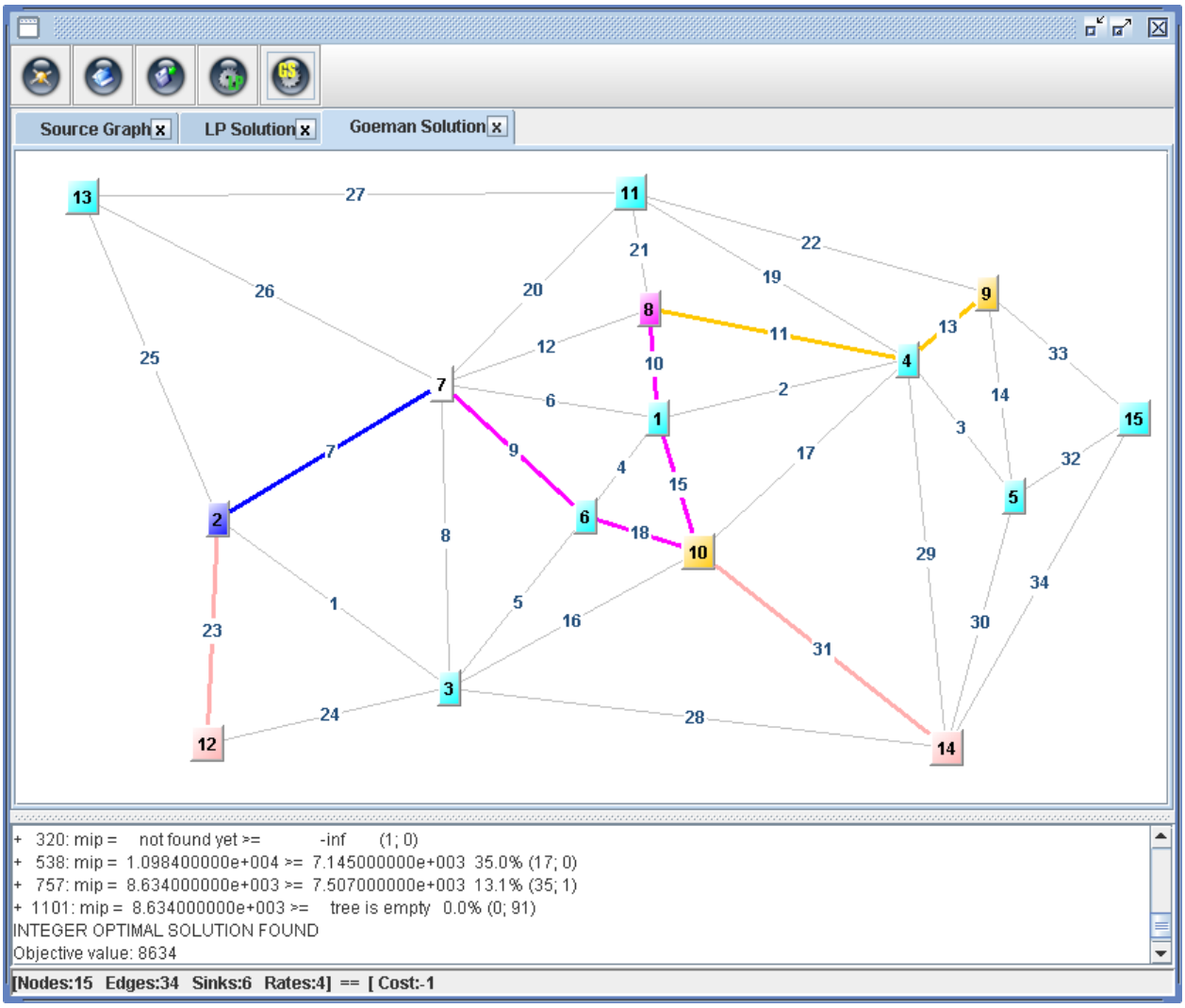

Figure 4.2. Graphical user interface for SimQ.

CHAPTER 5

IMPLEMENTATION

This section will describe the implementation details of the approximation algorithm and the simulation environment that we developed for creating and solving the QOS network problems.

Our system is named as "SimQ". The object oriented modeling tools are used to design and develop the SimQ system. While designing this system it was kept in mind that this system can be used as a basic framework to implement further problems and have visual solutions. So far most of the algorithms for the network problems have been implemented in the command line mode. This type of system's give the solutions but sometimes it gets really hard to visualize the solution. SimQ generates the graphs and shows the solution graph visually. Further it can color code the vertices and edges. The graph can also be modified when the source graph is generated.

5.1 Specifications

- Create a graph generator for creating undirected weighted planar graphs.
- Write adapter to convert the existing graph samples into the required format of the SimQ system.
- Create a system that can create and show the graphs visually. This system will have the functionality to create a random planar graph with given number

of vertices. This system will also have the functionality of letting the user manipulate the graphs once they are generated. The vertices can be moved around to visualize the graph clearly.

- Since the source graph in our particular problem have different requirements. Which include being able to define the source and sinks. Sinks being the vertices which need to be connected to the source. Each sink will have different bandwidth requirements. There will also be intermediate nodes which can be used to connect the source with the sinks but need not be connected to the source in the final solutions graph. Therefore the graph creating and visualizing system should be able to assign vertices the required bandwidths with which these vertices would be connected to the source. This system should also be able to define the source and sinks in the graph to be solved. It is just like the user laying out the network and then finding the final solution.

- Create the persistence system which will persist the generated graphs in multiple formats (old format and XML), to be used in later experimentation.

- Implement the LP (linear programming) solution to solve the source graphs. Some open source LP solver package can be used for this purpose. In this case a source model generator and some other adapters will be needed to be able to use any existing LP solver package.

- Implement the GS algorithm, which will take the source graph and generate the solution graph and display that in the Graph visualizer.

- Take existing source graphs in old format and generate multiple other source graphs with heterogeneous sizes and requirements.

- Run the experiments on the collected source graphs and save the solutions generated using NAPD(existing algorithm), LP and GS algorithm.

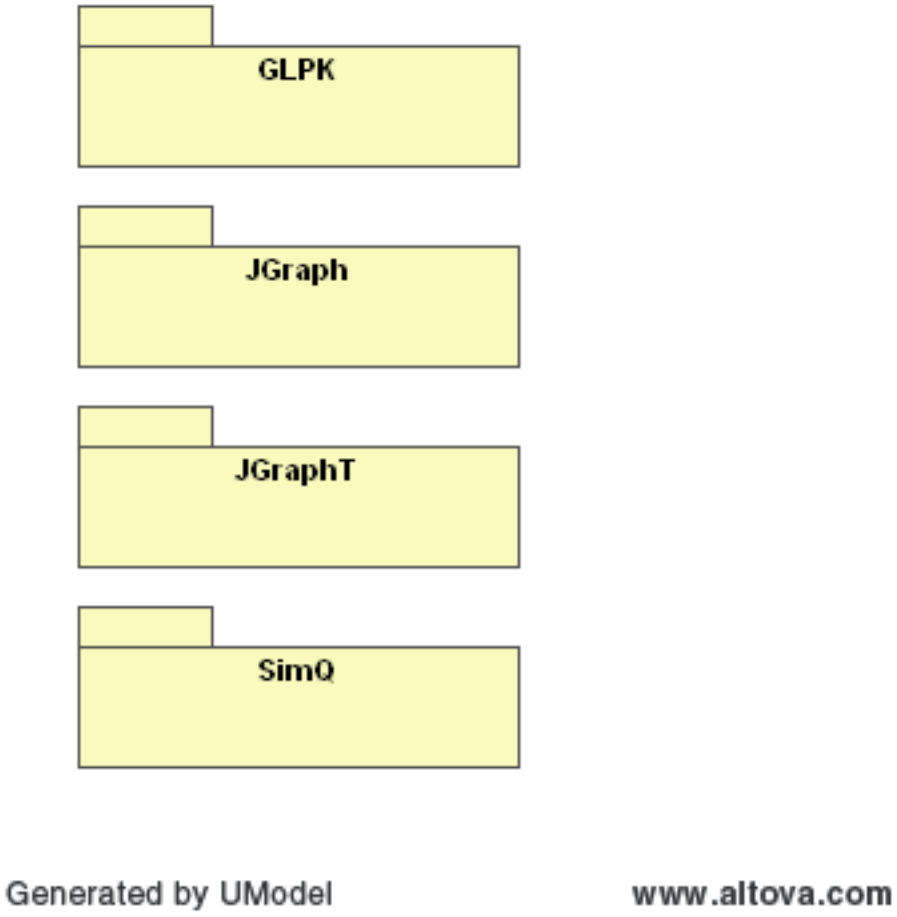

Figure 5.1. Package diagram

- Analyze and compare the results obtained by running multiple algorithms on the same source graphs.

5.2 Design

5.2.1 Package Diagram

These are the four packages that will be used to create our system. Following is a brief description of what each package is responsible for.

5.2.1.1 GLPK

This is an open source package to solve linear programming problems. We will use this package to solve our QOS problems and find the perfect solution. The challenge to use this package was that this system is written in C++ whereas our solution was written in JAVA. So to use this package we needed a JNI package which will be able to feed the source models and obtain the resultant solutions from this package.

5.2.1.2 JGraphT

This open source package is basically very useful to manipulate the backend data structures of all sorts of graphs. Although to use this package we had to subclass and modify a lot of classes but still it simplified our heavy load of work. This package also has some of the basic graph algorithms implemented such as DFS and BFS. Sometimes it was really hard to implement some of the features that we needed in our system due to the nature of design of this package, however finally we were able to modify this package to our needs.

5.2.1.3 JGraph

This package is available both in basic which is open source and free to use, and commercial versions. We however only needed the basic version which helped us plot the graphs and visualize the source and solution graphs. This package is used solely to take the graph objects and draw them based on provided attributes. Using this package the visual graphs can be modified such that the style, appearance and color of a graph's vertices and edges can be customized. This feature was used to distinguish the source graph from the solution graph. In the solution graph the edges are color coded to show which edges have been used and with what rate.

5.2.1.4 SimQ

This is the package where all our implementations exist. There are 38 classes and around 10,000 lines of code in this package. The design of this package is simple and flexible enough to be used to implement the functionality of same sort of algorithms. There will be a complete list of classes described later in this section for this package. Basically this package has the implementation of LP and GS algorithm.

5.2.2 Classes

Following is a list of classes in the SimQ package in alphabetical order and a brief description of what they do. ClosableTabbedPaneUI

A modified tabbed pane of JAVA swing interface which has a close button.

5.2.2.1 ColoredComboBoxRenderer

A modified combo box of Java Swing interface which shows color coded rate options.

5.2.2.2 ColoredRate

The underlying class for setting and getting the color option for the colored combo box.

5.2.2.3 CompoundVertexView

Vertex view that supports visual vertex nesting to show inclusion edges in a compound graph.

5.2.2.4 ConsoleListener

The listener for listening the messages pumped to the system console.

5.2.2.5 EdgeComponent

A critical part of the GS algorithm where each edge will be used with different rate and that information is stored in the objects of this class.

5.2.2.6 ForestConnectivityIterator

This is a depth first iterator to test the connectivity of the sinks in the Goeman's solution forest during the pruning phase.

5.2.2.7 Goeman

This class implements the basic solution of GS algorithm. It takes a source graph and creates a solution graph which sets the attributes of the edges used to connect the sinks to the source with the rates they are used.

5.2.2.8 GoemanConnectivityIterator

This depth first iterator is customized to suit the specialized needs of GS algorithm where we look at only the current rate vertices, whereas the current and higher rate edges are used to iterate over the current rate sinks.

5.2.2.9 GoemanEdgesForest

This class holds the collection of EdgeComponents that are used in the Goeman solution graph.

5.2.2.10 GraphFactory

The factory class, that generates a random planar graph.

5.2.2.11 GraphFileFilter

The filter to filter out the graph files when browsing to open a source graph file.

5.2.2.12 GraphLoader

This class reads the source graph from a file and loads it into the QOSGraph object.

5.2.2.13 LocalConsole

This is class is used to redirect the console messages to the text area in our GUI instead of the black system screen.

5.2.2.14 LPCreator

This class creates the source Linear Program model from a given graph to be fed to GLPK.

5.2.2.15 LPSolver

This class takes a source graph and used JNI to feed the model to GLPK to find the solution. Using this class we can call the GLPK API to get the solution instead of having it to create a solution file.

5.2.2.16 LPSolverFrame

The frame to show the LPSolver progress.

5.2.2.17 LPSolverThread

In order for our system to not wait for the solution we run the LPSolver in a separate thread.

5.2.2.18 NodePropertyDialog

This dialog is used to set the properties of a node including source, sink and rate.

5.2.2.19 QOSConnectivityIterator

Another Depth First iterator.

5.2.2.20 QOSEdge

A subclass of the Edge class to accommodate our needs of QOS problems.

5.2.2.21 QOSFileWriter

This class writes out any QOSGraph to a file in XML.

CHAPTER 6

SIMULATION RESULTS

We simulated two different type of sample instances. The results given in the tables are an average of the ten experiment results. We decided to generate two different type of sample instances. All the experiments were run on ten instances of the graph with the same number of nodes and rates.

- **The smaller instances**. The limit of number of nodes in these instances was 30. The reason for this strategy was to compare the results with the exact optimal results generated by running the ILP on these instances. Since the time taken by ILP for instances larger than 50 nodes was not feasible therefore the limit for these instances was set to 30.
- **The larger instances**. These instances have 50 to 150 number of nodes.

The graph instances were generated using four different types of parameters. Following is the list of these parameters:

- Number of nodes.
- Number of rates.
- Percentage of nodes that are intermediate.
- Sequence of rates

We ran the following seven algorithms on the same instances:

(1) Maxemchuk (max)

(2) Charikar (char)

(3) Naieve Primal-Dual (NA-PD)

(4) Restarting Primal-Dual (Re-PD)

(5) 4.311 Approximation Primal-Dual (4.311)

(6) Mixed Integer Linear Program (LP)

(7) Integer Linear Program (ILP)

In the result tables the Rates are denoted by R and Nodes by N, respectively.

6.0.3 Small instances with 50% Intermediate nodes, Arithmetic Progression

The table 6.1 has the results by running simulations on the small network instances with rates increasing by arithmetic progression. The reason for generating the small instances was to get the ILP results as well. Since the LP solver runs for unreasonable time for large instances so that was not a feasible solution.

Table 6.1. 50% Intermediate nodes, Arithmatic Progression.

R	N	Max	Char	NA-PD	Re-PD	4.311	MILP	ILP
1	10	7583	7566	7433	7408	1784	1768	1381
1	15	6494	6481	6374	6354	2140	2128	1589
1	20	5865	5854	5765	5748	2718	2412	1713
2	10	5445	5435	5352	5343	2926	2843	2318
2	15	5208	5199	5121	5111	3584	3469	2715
2	20	5149	5142	5065	5057	4700	4570	3296
5	10	5011	5004	4932	4924	3771	3700	3039
5	15	5123	5110	5032	5025	6322	5905	4733
5	20	5384	5372	5295	5283	8471	7052	5499
10	10	6478	6505	6469	6465	6523	6375	5251
10	15	8769	8732	8591	8570	11263	10503	8517
10	20	9516	9494	9316	9283	11110	-	-

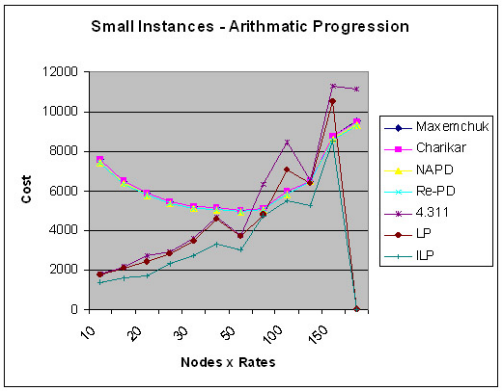

Figure 6.1. Small instances, 50% intermediate nodes, Arithmetic Progression.

6.0.4 Small instances with 50% Intermediate nodes, Geometric Progression

Table 6.2 has the same number of nodes and rates as the table 6.1 except that the rates grow with a geometric progression.

Table 6.2. 50% Intermediate nodes, Geometric Progression.

R	N	Max	Char	NA-PD	Re-PD	4.311	ILP	ILP
1	10	1784	1784	1784	1784	1784	1768	1381
1	15	2140	2140	2140	2140	2140	2128	1589
1	20	2718	2718	2718	2718	2718	2709	1894
2	10	2923	2923	2876	2912	2926	2843	2318
2	15	3550	3550	3505	3485	3584	3469	2715
2	20	4681	4681	4616	4626	4700	4489	3225
5	10	4319	4319	4310	4300	4342	4263	3589
5	15	9277	9277	9258	9429	9374	8976	7575
5	20	12509	12509	12930	12597	12771	-	-
10	10	14795	14795	15357	14779	14880	14772	13071
10	15	59986	59986	60050	61517	60642	-	-
10	20	26316	26316	27857	26591	26445	25967	21104

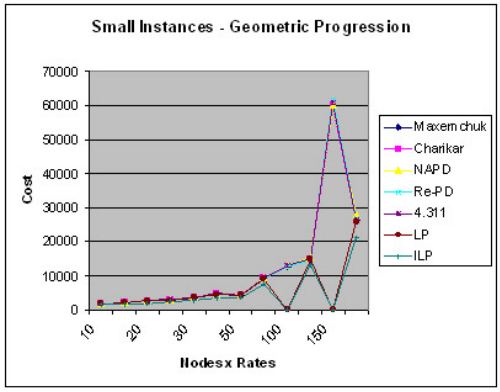

Figure 6.2. Small instances, 50% intermediate nodes, Geometric Progression.

6.0.5 Large instances with 50% Intermediate nodes, Arithmetic Progression

The large instances generated have three different number of nodes ranging from 50 to 150. The ILP results for this large number of nodes could not be obtained except the instances with 50 nodes and 1 rate. The table 6.3 shows the results of simulations run on these instances with rates generated using the arithmetic progression.

Table 6.3. 50% Intermediate nodes, Arithmatic Progression.

R	N	Max	Char	NA-PD	Re-PD	4.311	ILP
1	50	4217	4217	4217	4217	4217	1836
1	100	5904	5904	5904	5904	5904	-
1	150	7073	7073	7074	7074	7073	-
2	50	7595	7595	7485	7491	7628	3640
2	100	10832	10832	10643	10600	10872	-
2	150	12807	12807	12763	12713	12880	-
5	50	15521	15198	15290	15159	16110	9073
5	100	23584	23527	23078	22988	24498	-
5	150	29394	29225	28284	28131	30232	-
10	50	29116	28247	28140	28098	30600	16781
10	100	40956	40510	39826	39309	42373	-
10	150	56135	55636	53283	53035	59082	-

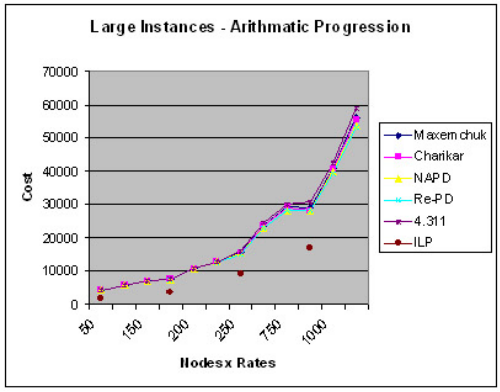

Figure 6.3. Large instances, 50% intermediate nodes, Arithmetic Progression.

6.0.6 Large instances with 50% Intermediate nodes, Geometric Progression

Table 6.4 shows the results of instances with nodes 50 to 150 and rates generated using geometric progression.

Table 6.4. 50% Intermediate nodes, Geometric Progression.

R	N	Max	Char	NA-PD	Re-PD	4.311	ILP
1	50	4217	4217	4217	4217	4217	1836
1	100	5904	5904	5904	5904	5904	3465
1	150	7073	7073	7074	7074	7073	-
2	50	7595	7595	7485	7491	7628	3640
2	100	10832	10832	10643	10600	10870	-
2	150	12807	12807	12763	12713	12880	-
5	50	37471	31471	33611	32346	32522	20884
5	100	54596	54596	55492	54800	56096	-
5	150	70961	70961	72813	71854	72405	-
10	50	403755	403755	423769	410586	424123	299236
10	100	561079	561079	571414	563776	573906	-
10	150	1451722	1451722	1524655	1510602	1490925	-

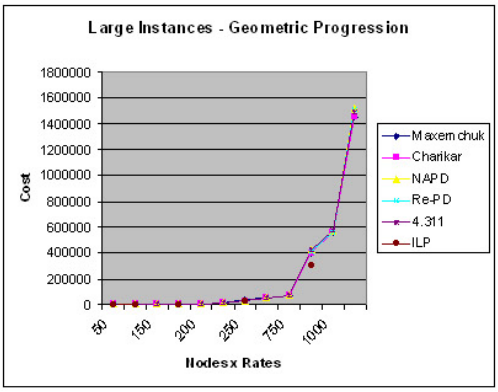

Figure 6.4. Large instances, 50% intermediate nodes, Geometric Progression.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

The new ILP solution and 4.311 primal dual algorithm was implemented and simulations were run on all previously implemented algorithms and new algorithms on same instances. The results show that restarting-PD is still the best of all.

The next step of our research will be to compare the efficiency of new approach by creating clustered networks to make it more realistic. Different clusterings will be used for random networks that are intended to look like national and international networks. In a national network there are more nodes in major population centers and in the international network there are more nodes on land masses. Another improvement could be the use of edge contraction technique.

BIBLIOGRAPHY

[1] A. Balakrishnan, T.L. Magnanti, P. Mirchandani, *Modeling and Heuristic Worst-Case Performance Analysis of the Two-Level Network Design Problem*, Management Science, **40**: 846-867, (1994)

[2] A. Balakrishnan, T.L. Magnanti, P. Mirchandani, *Heuristics, LPs, and Trees on Trees: Network Design Analyses*, Operations Research, **44**: 478-496, (1996)

[3] P. Berman and V. Ramaiyer, *Improved Approximations for the Steiner Tree Problem*, Journal of Algorithms 17, 381-408 (1994)

[4] R. Bajaj, C.P. Ravikumar, and S. Chandra, *Distributed Delay Constrained Multicast Path Setup High Speed Networks*, Proceedings of the Fourth International Conference on High Performance Computing, 438–442, 1997.

[5] A. Borchers and D.Z. Du, *The k-Steiner Ratio in Graphs*, SIAM Journal on Computing, **26**:857-869, (1997)

[6] G. Calinescu, C. Fernandes, I. Mandoiu, A. Olshevsky, K. Yang and A. Zelikovsky, *Primal-Dual Algorithms for QoS Multimedia Multicast*, Proceedings of IEEE GLOBECOM 2003, pp. 3631–3635 (2003)

[7] K. Calvert, M. Doar and E. W. Zegura, "Modeling Internet Topology," IEEE Communications Magazine, June 1997.

[8] M. Charikar, J. Naor, and B. Schieber, *Resource Optimization in QoS Multicast Routing of Real-Time Multimedia*, IEEE/ACM Transactions on Networking, **12**: 340-348 (2004)

[9] C.J. Colbourn and G.L. Xue, *Grade of service Steiner trees in series-parallel networks,* Advances in Steiner Trees (Ding-Zhu Du, J.M. Smith, and J.H. Rubinstein, eds.), Kluwer Academic Publishers, 2000, pp. 163–174.

[10] J.R. Current, C.S. Revelle, and J.L.Cohon, *The Hierarchical Network Design Problem*, European Journal of Operations Research, **27**: 57-66, (1986)

[11] http://www.cc.gatech.edu/fac/Ellen.Zegura/gt-itm/gt-itm.tar.gz

[12] M. Goemans and D. Williamson, *The Primal-Dual Method for Approximation Algorithms and its Application to Network Design Problems*, in Approximation Algorithms, D. Hochbaum, Ed., 1997.

[13] M. Karpinski, I. Măndoiu, A. Olshevsky, and A. Zelikovsky, *Improved Approximation Algorithms for the Quality of Service Steiner Tree Problem*, Algorithmica, **42**:109-120, (2005)

[14] N. Maxemchuk, *Video Distribution on Multicast Networks*, IEEE Journal on Selected Areas in Communications **15**:357-372 (1997)

[15] P. Mirchandani, *The Multi-Tier Tree Problem*, INFORMS Journal on Computing, **8**: 202-218, (1996)

[16] K. Mehlhorn, *A faster approximation algorithm for the Steiner problem in graphs*, Information Processing Letters **27**: 125-128, (1988)

[17] H. Promel and A. Steger, *A New Approximation Algorithm for the Steiner Tree Problem with Performance Ratio* $\frac{5}{3}$, Journal of Algorithms, **36**: 89-101,(2000)

[18] G.N. Rouskas and I. Baldine, *Multicast routing with end-to-end delay and delay variation constraints*, IEEE J. on Selected Areas in Communications **15**:346–356, (1997).

[19] G. Robins and A. Zelikovsky, *Tighter Bounds for Graph Steiner Tree Approximation*, SIAM Journal on Discrete Mathematics, **19**:122-134, (2005)

[20] H.F. Salama, D.S. Reeves, and Y. Viniotis, *Evaluation of Multicast Routing Algorithms for Real-Time Communication on High-Speed Networks*, IEEE Journal on Selected Areas in Communications, **3**:332–345 (1997).

[21] H. Takahashi and A. Matsuyama, *An Approximate Solution for the Steiner Problem in Graphs*, Math. Japonica, **6:** 573-577, (1980)

[22] G. Xue, G.-H. Lin, D.-Z. Du, *Grade of Service Steiner Minimum Trees in the Euclidean Plane*, Algorithmica, **31**: 479-500, (2001)

[23] H. Ural and K. Zhu, *An Efficient Distributed QoS Based Multicast Routing Algorithm*, Proceedings of the 21st IEEE International Performance, Computing, and Communication Conference, 27–36 (2002).

[24] A. Zelikovsky, *An 11/6-approximation algorithm for the network Steiner problem*, Algorithmica **9**: 463-470, (1993)

[25] A. Zelikovsky, *A faster approximation algorithm for the Steiner tree problem in graphs*, Information Processing Letters **46**: 79-83, (1993)