Lagrange Relaxation Based Method for the QoS Routing Problem

Alpár Jüttner, Balázs Szviatovszki, Ildikó Mécs, Zsolt Rajkó
Ericsson Research,
Traffic Analysis and Network Performance Laboratory
Budapest, Hungary
e-mail: {Alpar.Juttner, Balazs.Szviatovszki, Ildiko.Mecs, Zsolt.Rajko}@eth.ericsson.se

Abstract—**In this paper a practically efficient QoS routing method is presented, which provides a solution to the delay constrained least cost routing problem. The algorithm uses the concept of aggregated costs and provides an efficient method to find the optimal multiplier based on Lagrange relaxation. This method is proven to be polynomial and it is also efficient in practice. The benefit of this method is that it also gives a lower bound on the theoretical optimal solution along with the result. The difference between the lower bound and the cost of the found path is very small proving the good quality of the result. Moreover, by further relaxing the optimality of paths, an easy way is provided to control the trade-off between the running time of the algorithm and the quality of the found paths. We present a comprehensive numerical evaluation of the algorithm, by comparing it to a wide range of QoS routing algorithms proposed in the literature. It is shown that the performance of the proposed polynomial time algorithm is close to the optimal solution computed by an exponential algorithm.**

Keywords—**QoS routing, delay, optimization, Lagrange relaxation**

I. Introduction

Providing network-wide delay guarantee for real-time flows in the Internet is a challenging task. Thinking of some voice application there are numerous sources of delay: queuing delay, propagation delay, codec delay, serialization delay, and other e.g., shaping/reshaping delays.

Queuing delay depends on the used scheduling algorithms. One can determine queuing delay by calculating worst-case delay, or by calculating average delay on some statistical means. In the literature several articles concentrate on guaranteeing worst-case delay and bandwidth for a simple flow [1], [2], [3], [4] utilizing the concept of the so-called 'network calculus' and 'service curves' [5]. All of these papers concentrate on providing delay bounds when flows are treated separately. Grouping flows that belong to guaranteed delay service, may result in lower resource requirement than handling all the flows separately. In [6] it was shown that grouping two flows with the same source-destination pair may result in some gain of allocated resource.

It was shown in [9] that by using an enhanced routing information distribution component [8], [10] and assuming a rate-based service model [11], queuing delay can be determined for each hop when the burst size, maximum packet size and minimum guarantied service rate reserved for that hop is known.

Propagation delay can be considered as a constant attribute of the link depending on the distance of the source and destination nodes. A paper [7] investigating the end-to-end effect of Internet path selection found that there is a substantial degree of inefficiency in path selection. The study showed, that *congestion and propagation delay* both play significant roles in the observed inefficiencies of Internet paths and neither one is the single dominant factor.

Knowing the computed per-hop worst-case queuing delay together with the propagation delay we can assign a single additive delay metric for each link. In our approach, unlike in [9] we do not take into account the inaccuracy of (e.g. OSPF) state information, but we consider a network optimization criterion (that was neglected in the mentioned study). For example such an optimization criterion can be to have good resource utilization which can be achieved by minimizing hop-count. Another example could be to minimize an additive monetary cost [12], or some price depending on the delay guaranty requested [13].

It has been proven in [14] that a routing problem is $\mathcal{NP}$-complete, if the number of additive QoS parameters that should be minimized are more then or equal to two. In the literature most of the algorithms does not try to solve this complex problem, instead they define simpler problems. In this paper we concentrate on a simplified problem which is to find a path that is minimal for a cost, and the delay of it remains under a given bound. We formulate this problem —referred to as the *Delay Constrained Least Cost path problem* (DCUR)— in Section I-A.

In Section II we survey previous work in the field. We first present a method which provides the optimal paths for the DCLC problem, but with exponential running time. Then we group the rest of the QoS routing algorithms in two groups: (1) heuristics combining paths calculated on different metrics, and (2) heuristics calculating a single metric from multiple requirements.

As the main result of the paper, in Section III a new algorithm called *Lagrange Relaxation based Aggregated Cost* (LARAC) is proposed, that belongs to the second category. The LARAC algorithm provides a polynomial heuristic solution for the DCLC problem. Instead of using heuristics to manipulate the weights of the different metrics in the aggregated cost function, our method uses the Lagrange relaxation method to find the optimal multiplier. Besides describing the details of the LARAC algorithm we show that it has the following properties:

- It always gives back a path that satisfies the delay constraint if such a path exists.
- Its running time is proven to be polynomial. Moreover, in practical cases the running time of the algorithm competes with the most elementary heuristics.
- Though there is no general theoretical guarantee on the optimality of the found path's cost, the algorithm gives a lower

bound of the optimal cost, that is it gives individual guarantees for each solution.

- The lower bound and the cost of the found path are very close in practical cases, proving the high quality of the result.
- It can also be used for multiple QoS requirements.

An extension to the basic algorithm is also proposed in Section III-E.2, which can be used to control the trade-off between the optimality of the result and the running time of the algorithm.

In Section IV we show that the algorithm gives good simulation result compared to previously developed algorithms. With the proposed extension, the running time of our algorithm is very good, and the path costs are close to the optimal one.

A. Problem Formulation

Assume that a communication network can be modeled as a directed, connected graph $G = (V, E)$, where V represents the set of nodes, and E represents the set of directed links. Let $n = |V|$ and $m = |E|$, the number of nodes and the number of edges, respectively. (Usually we have duplex links, so in the graph the existence of a link $e = (u, v)$ from node u to node v implies the existence of a link $e' = (v, u)$ for any $u, v \in V$.)

Each link $e \in E$ is characterized by the following values: *Delay* $d(e)$, which is a constant value related to the propagation and queuing delay of the link/hop, *Cost* $c(e)$, which is e.g., a simple hop count, a monetary cost or some measure of link's capacity.

The definition of the *Delay Constrained Least Cost path problem* (referred hereafter simply as DCLC) is the following [15]:

Given a directed, connected graph $G(V, E)$, a non-negative cost $c(e)$ and a non-negative delay $d(e)$ for each link $e \in E$, a source node s, a destination node t, and a positive delay constraint Δ_{delay}. The constrained minimization problem is:

$$\min_{p \in P'(s,t)} \sum_{e \in p} c(e), \tag{1}$$

where $P'(s, t)$ is the set of path from s to t for which the end-to-end delay is bounded by Δ_{delay}. Therefore $P'(s, t) \subseteq P(s, t)$. Namely a $p \in P(s, t)$ is in $P'(s, t)$ if and only if

$$\sum_{e \in p} d(e) \leq \Delta_{delay}. \tag{2}$$

The DCLC problem is $\mathcal{NP}$-hard [14], however it is worth mentioning that an important special case is solvable in polynomial time if all link costs or all link delays are equal, or have a few constant different values [21]. Although a number of heuristic algorithms have been proposed to solve the DCLC problem, the appropriate choice of routing algorithms is still an open issue.

II. Previous work

In this section we give a survey on the different routing algorithms proposed in the literature. Up to our best knowledge there is no widely accepted algorithm that can give the optimal solution of the DCLC QoS routing problem in polynomial time.

Widyono [16] proposed a routing method that gives the optimal path —having the lowest possible cost without violating the delay constraint—, unfortunately with an exponential running time. The *Constrained Bellman-Ford (CBF)* routing algorithm performs a breadth-first search, discovering paths of monotonically increasing delay while recording and updating lowest cost path to each node it visits. It stops, when the highest constraint is exceeded, or there is no more possibility of improving the paths. Since this extension uses delay instead of hopcount, which is a continuous metric, the routing table which contains entries for all possible delays can be very large, even of unrealizable size. As we mentioned the CBF algorithm has an exponential running time. Widyono have not analyzed it conclusively, but stated that despite of its exponential nature, the performance of the algorithm was reasonable in practice.

As a further development of the CBF algorithm, using a kind of scaling technique, the authors of [31], [32], [33] and [34] gave *fully polynomial time approximation schemes* for the DCLC problem, namely they proved that for any $\epsilon > 0$ there exists a polynomial time algorithm that is able to find a path satisfying the delay constraint with cost no greater than a factor of $1 + \epsilon$ from the optimum. The running time of the best known approximation scheme is $O(nm \log n \log\log n + \frac{nm}{\epsilon})$ [34] . Unfortunately, *in practical cases* the running time of these methods for sufficiently small ϵ will be worse than even of the CBF algorithm, which makes these results rather theoretical.

All other algorithms try to use some kind of *heuristics or approximation*. A very simple method proposed by W. C. Lee [17] does not give optimal paths, but provides a simple heuristic solution to the DCLC problem. The *Fallback* routing algorithm assumes that there are ranked metrics in the network. First, the routing algorithm calculates the shortest path for the first metric (i.e. cost), and then checks whether it can guarantee all the other QoS requirements. If the path failed, the algorithm tries to find another one for the next metric, until an appropriate path is found, or the routing fails for all the metrics. This algorithm is very simple, fast and always gives an appropriate solution if it exists, but there is no guarantee of finding the optimal route and we do not know anything about the quality of the found path.

The algorithm proposed by Pornavalai [18] improves the previous idea by *combining paths* calculated on the different metrics, by first calculating the paths based on the metrics from the source node to the others and from all the nodes to the destination, and then trying all the possible combinations of them. There are two more algorithms improving the idea of Fallback. Ishida [19] proposed a distributed algorithm in which the nodes always choose the least cost path until it fulfills the delay requirements and from that point they choose the path with the least delay. The *Delay Constrained Unicast Routing* (DCUR) algorithm proposed by Salama [15] is similar to it, but it can choose between the least cost and least delay paths independently from the choice of the previous node. These three algorithms have higher but still reasonable running time, and it is more likely that they find a solution near the optimal than it was in case of the Fallback algorithm, but neither these algorithms have guarantees for the optimality of the path.

Cheng and Nahrstedt [20] give an algorithm to find a path that meets two requirements in polynomial time. The algorithm *reduces* the original problem to a simpler one by modifying the cost function, based on which the problem can be solved using an *extended* shortest path algorithm. The shortcoming of this method is that the algorithm has to use high granularity in approximating the metrics, so it can be very costly in both time and space, and it can not guarantee that the simpler problem has a solution if the original problem has.

The last group of algorithms [21], [22], [24], [25] is based on calculating a *simple metric* from the multiple requirements. By doing so, we can use a simple shortest-path algorithm based on a single cost aggregated as a combination of weighted QoS parameters. The main drawback of this solution is that the result is quite sensitive to the selected aggregating weights, and there is *no clear guideline on how the weights should be chosen*. To handle the problem of determining the weights for the aggregated cost solution, several routing algorithms has been proposed. For example the algorithm proposed in [26] tries to find an appropriate path based on a single cost of weighted link metrics. If it fails, it tries to find another path by iterative changing the values of weights so as to get more possible paths, until an appropriate path that guarantees QoSs can be found. Another problem with this solution is that if the path is correct for the aggregated cost, it does not mean that it is correct for all of the user's QoS requirements. In other words, *information is lost in the aggregation process*.

Wang and Crowcroft [12] say that single (mixed) metric approach can not be sufficient for QoS routing. It can at best be an indicator in path selection but does not provide enough information for making reliable quantitative estimation of whether the path meets the requirements or not. For example, he proposes a mixed metric from delay, bandwidth and loss probability. A path with a large value of $\frac{B(p)}{D(p)\cdot L(p)}$ is likely to be a better choice in terms of bandwidth, delay and loss probability. However, it is not sure that a path, which minimizes the above expression, is suitable for any of the constraints (bandwidth, delay or loss).

On the contrary we state that this problem can be solved, and there are algorithms based on aggregated costs, which can find paths that meet the QoS constraints. In the next section we present the LARAC algorithm which uses the Lagrange relaxation method to find the optimal weighting of the aggregated cost function to achieve this.

III. Proposed algorithm

A. Lagrange Relaxation based Aggregated Cost

The original DCLC problem is to minimize the cost c of the path, while keep the delay d under a given constraint Δ_{delay}. In a formal description, we are looking for

$$\min\{c(p) : p \in P(s,t) \text{ and } d(p) \le \Delta_{delay}\}, \tag{3}$$

where $P(s,t)$ is the set of paths from the source node s to the destination node t.

Unfortunately, as we mentioned this problem has been proven to be $\mathcal{NP}$-hard, so we cannot hope an algorithm that can find the theoretical optimum and runs in polynomial time.

Our heuristic is based on the Lagrange relaxation.

Lagrange Relaxation is a common technique for calculating lower bounds, and finding good solutions for this problem. First Held and Karp raised with this technique for the Traveling Salesman Problem in [27], [28].

We propose to use this technique to solve the DCLC problem. The presented **LARAC - Lagrange Relaxation based Aggregated Cost** algorithm is based on the heuristic of minimizing $c_\lambda := c + \lambda \cdot d$ modified cost function. For a given (fixed) λ we can easily calculate the minimal path (p_λ). If $\lambda = 0$ and $d(p_\lambda) \le \Delta_{delay}$ we found an optimal solution for the original problem as well. If $d(p_\lambda) > \Delta_{delay}$ we must increase λ, to increase the dominance of delay in the modified cost function. So we increase λ while the optimal solution of c_λ suits the delay requirements.

In this section we show how to find the value of λ that gives the best result. Moreover, the algorithm will give an upper bound on the badness of the solution as a byproduct, based on the following Claim.

Claim 1: Let

$$L(\lambda) := \min\{c_\lambda(p) : p \in P(s,t)\} - \lambda\Delta_{delay}. \tag{4}$$

Then $L(\lambda)$ is a *lower bound* to problem (3) for any $\lambda \ge 0$.
Proof. Let p^* denote an optimal solution of (3). Then

$$\begin{aligned} L(\lambda) &= \min\{c_\lambda(p) : p \in P(s,t)\} - \lambda\Delta_{delay} \\ &\le c_\lambda(p^*) - \lambda\Delta_{delay} \\ &= c(p^*) + \lambda(d(p^*) - \Delta_{delay}) \le c(p^*) \end{aligned}$$

proves the claim. □

To obtain the best lower bound we need to maximize the the function $L(\lambda)$, that is we are looking for the value

$$L^* := \max_{\lambda \ge 0} L(\lambda), \tag{5}$$

and the maximizing λ^*. Now, some properties of the function $L(\lambda)$ are given. The simple proofs are left to the reader.

Claim 2: L is a concave piecewise linear function, namely the minimum of the linear functions $c(p) + \lambda(d(p) - \Delta_{delay})$ for all $p \in P(s,t)$. □

Claim 3: For any $\lambda \ge 0$ and c_λ-minimal path p_λ, $d(p_\lambda)$ is a supgradient of L in the point λ. □

Claim 4: Whenever $\lambda < \lambda^*$, then $d(p_\lambda) \ge \Delta_{delay}$ and if $\lambda > \lambda^*$, then $d(p_\lambda) \le \Delta_{delay}$ for each c_λ-minimal path p_λ. □

Claim 5: A value λ maximizes the function $L(\lambda)$ *if and only if* there are paths p_c and p_d which are both c_{λ^*}-minimal and for which $d(p_c) \ge \Delta_{delay}$ and $d(p_d) \le \Delta_{delay}$. ($p_c$ and p_d can be the same, in this case $d(p_d) = d(p_c) = \Delta_{delay}$.) □
Our algorithm will give these paths along with λ^*.

Claim 6: Let $0 \le \lambda_1 < \lambda_2$, and $p_{\lambda_1}, p_{\lambda_2} \in P(s,t)$ λ_1-minimal and λ_2-minimal paths. Then $c(p_{\lambda_1}) \le c(p_{\lambda_2})$ and $d(p_{\lambda_1}) \ge d(p_{\lambda_2})$. □

These two latter Claims together gives that the λ^* maximizing the function $L(\lambda)$ gives the best modified cost function, that is λ^* is the smallest value for which there exists a c_{λ^*}-minimal path p_d which satisfies the delay constraint.

Summing up, we neglect the constraining conditions (this is the relaxation), and build them into the object function. The solutions feasible to the original problem can certainly suit the relaxation conditions as well, so we can get a lower bound of the original problem. If the path found is not feasible for the constraining conditions, we increase the dominance of it in the modified cost function, enforcing the solution to approach to the optimal solution. Moreover, decrease the difference between the obtained lower bound and the optimum of the original problem as well. This is the base of the Lagrange-relaxation. For a more detailed description, the reader is referred to [23].

With the help of Lagrange relaxation we have an algorithm which can find the optimal λ for a given source destination pair, thus among the aggregated cost routing methods this algorithm gives the best solution that can be obtained. Moreover it gives an estimation for the optimal solution, although we have no guarantee of finding the optimal solution, we always get a bound for the solution, which tells that at most how far is it from the optimal solution.

In the following section we survey the LARAC algorithm, and in Section III-E.1 and III-E.2 we mention two possibilities of improving the algorithm.

B. The description of the algorithm

In this section LARAC algorithm is described:

1. In the first step the algorithm sets $\lambda = 0$. It calculates shortest path on c_λ modified cost function with Dijkstra algorithm. It means that the first shortest path found by the algorithm is the shortest on the original cost c. If the path found meets the delay requirement Δ_{delay}, this is the optimal path, and the algorithm stops.
2. Otherwise the algorithm stores the path as the best path that does not satisfy Δ_{delay} (this path is denoted by p_c in the following), and checks whether an appropriate solution exists or not: Calculates shortest path on delay d. If the obtained path suits the delay requirement a proper solution exists, so the algorithm stores this path as the best appropriate path found till now (denoted by p_d). Otherwise there is no suitable path from s to t that can fulfill the delay requirement, so the algorithm stops.

In the further steps we obtain the optimal λ, through updating p_c and p_d repeatedly with another paths.

3. Let's see the current paths p_c and p_d. If for a certain λ, both p_c and p_d are c_λ-minimal, then using the Claim 5, this λ maximizes $L(\lambda)$. But in this case $c_\lambda(p_c) = c_\lambda(p_d)$, from which we get that the only possible λ is

$$\lambda := \frac{c(p_c) - c(p_d)}{d(p_d) - d(p_c)}. \tag{6}$$

So, we set it as the new candidate for the optimal solution. Then we find a c_λ-minimal path r. If $c_\lambda(r) = c_\lambda(p_c)$ then p_c and p_d are also c_λ-minimal, so we are done by setting $\lambda^* = \lambda$ and resulting p_d. If $c_\lambda(r) < c_\lambda(p_c)$ then we replace either p_c or p_d with r according whether it fails or fulfills the delay constraint, and repeats this step.

e

```
procedure LARAC($s, t, c, d, \Delta_{delay}$)
  $p_c$ := Dijkstra($s, t, c$)
  if $d(p_c) \leq \Delta_{delay}$ then return $p_c$
  $p_d$ := Dijkstra($s, t, d$)
  if $d(p_d) > \Delta_{delay}$
    then return "There is no solution"
  repeat
    $\lambda := \frac{c(p_c)-c(p_d)}{d(p_d)-d(p_c)}$
    $r$ := Dijkstra($s, t, c_\lambda$)
    if $c_\lambda(r) = c_\lambda(p_c)$ then return $p_d$
      else if $d(r) \leq \Delta_{delay}$ then $p_d := r$
                                   else $p_c := r$
  end repeat
end procedure,

where Dijkstra($s, t, c$) returns a $c$-minimal
path between the nodes $s$ and $t$.
```

Fig. 1. The LARAC algorithm

C. Running time of the algorithm

Let $p_c^1, p_c^2, p_c^3, \cdots$ and $p_d^1, p_d^2, p_d^3, \cdots$ denote the sequences of paths generated by the algorithm. It can be seen that

$$d(p_c^1) > d(p_c^2) > d(p_c^3) > \cdots > \Delta_{delay} \tag{7}$$

and

$$d(p_d^1) < d(p_c^2) < d(p_c^3) < \cdots \leq \Delta_{delay}, \tag{8}$$

so, because there are only finite number of different path, the algorithm finds the optimal λ in finite number of steps. Moreover the following stronger result can be proved using similar technique to the proof of strongly polynomiality of the so-called Newton-method [30].

Theorem 1: The LARAC algorithm terminates after $O(m \log^3 m)$ iteration, so the running time of the algorithm is $O(m^2 \log^4 m)$. □

We omit the long and technical proof.

D. The optimality of the path

As we stated it above (in Section III-A) the optimality of λ does not mean the optimality of the path found as well. In fact there is no guarantee of finding the optimal path, or some bound which is higher than the ratio of the obtained, and the optimal path. There are two cases, when the algorithm can not find the optimal solution.

In the first case, the algorithm could find the optimal solution, but it is not sure, that it finds it. In Figure 2 we can find an example of such a network, and the belonging cost functions of the paths. In this network there are several path with the same c_λ at the optimal λ, therefore the algorithm can not decide which path is the optimal on the base of c_λ.

Here the optimality of the path found depends on which path is chosen by the Dijkstra algorithm if more then one path have a cost of equal length. The first idea of constraining Dijkstra to choose the path with the lowest cost among the path with equal c_λ, does not results in a proper path. As it can be seen in the figure, it will always find the path with the lowest cost, which violates the delay constraint. In real networks this case is not likely to occur, so it is not worthy to study the solution of it profoundly. The other case is more likely, but as we will see it later it neither cause a great anxiety. Considering the network, and the Lagrange function in Figure 3 we can see that it can easily occur, that the modified cost (c_λ) of the optimal path is higher than a suboptimal path at the optimal λ.

Moreover, c_λ of the optimal path is higher than $L(\lambda)$ for any λ, so this path can not be found by this algorithm by no means. In fact this problem can not be handled with any of the algorithms based on aggregated cost.

A possibility of improving the performance of the algorithm in this case is to increase the influence of cost in the modified cost function. If the cost is raised to the second power in c_λ, the effect of cost becomes more strong, therefore the chance of the optimal path to have the smallest c_λ at the optimal λ increases. Since the delay remains linear $c_\lambda - \Delta_{delay}$ remains linear function in λ for all the paths, it does not violate the convergence of the algorithm. In Figure 4 we can see the effect of raising cost to the second power.

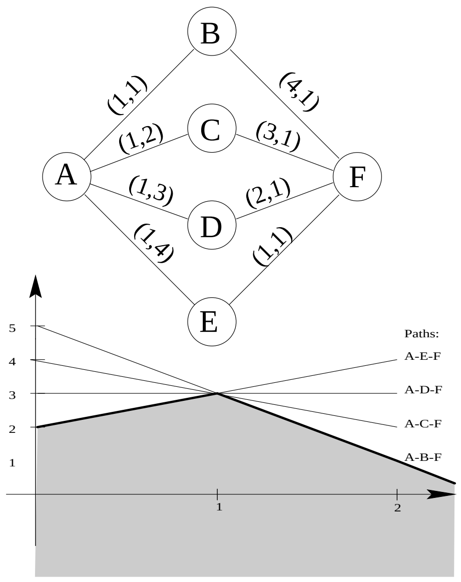

Fig. 2. A network (a), where there are several paths with the same c_λ at λ_{opt}, and the Lagrange function of it (b)

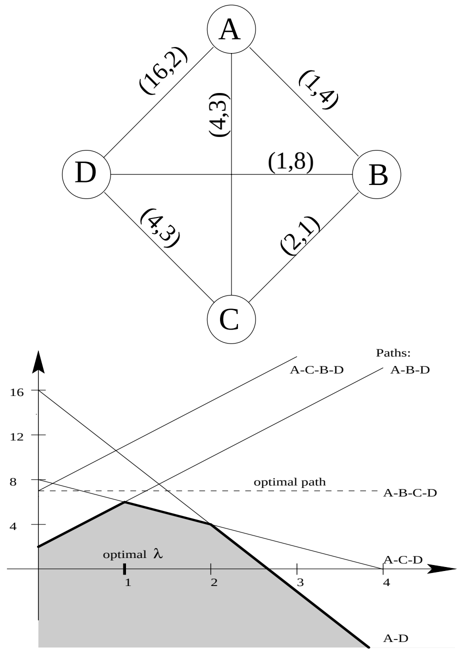

Fig. 3. A network (a), where the c_λ of the optimal path is higher than a suboptimal path at λ_{opt}, and the Lagrange function of it (b)

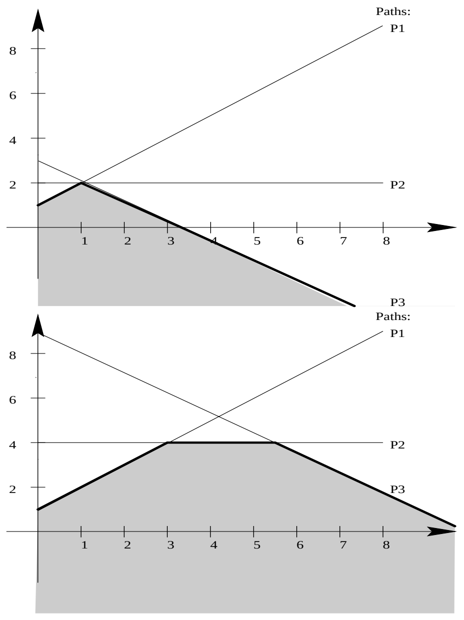

Fig. 4. The Lagrange function of a network with linear cost (a), and with square cost (b)

Notice, that this modifying of c_λ does not influence the slope of the lines, i.e., the dependence of $c_\lambda - \Delta_{delay}$ on λ. It just shifts the lines farther off each other. Hence, the path that has a lower cost will be placed to a lower position in comparison with another path with a higher cost.

Certainly this trial does not solve the problem, since it is easy to find other networks which behave in the same way with this square cost as the network in the previous example. The only benefit of it is that for some of the networks it gives better solution than the algorithm with a linear cost function. (Other exponents can be applied as well, but the increasing of the exponent may results in very high values of costs. It usually can not be decided that it is worth working with higher costs, or it does not give much better solution in a given network.)

E. Improvements to LARAC algorithm

In this section we mention two important possibilities of improving LARAC.

E.1 Storing former results

In reality router computes paths for a lot of destinations, instead of doing it for a single one. The original LARAC algorithm can be improved by utilizing this fact, by reusing results of previous calculations.

So, whenever we call Dijkstra algorithm with a certain λ we store its whole result, that is, we store the c_λ-minimal paths to all destinations. These paths form a tree, so we need to store only m edges. Moreover we store the c-cost and the d-cost of these path.

The new method is very simple. The only difference is that when we are looking for a path to a new destination, we are looking for the largest *previously calculated* λ_1 for which $d(p_{\lambda_1}) \geq$

Δ_{delay} and the smallest λ_2 for which $d(p_{\lambda_2}) \leq \Delta_{delay}$, and we initialize p_c and p_d with these paths instead of c- and d-minimal ones.

E.2 Improved stop condition

We optimize the algorithm to gain faster running time, by giving up the requirements on the minimality of the found path's cost. When we compute a path for a given source-destination pair, we calculate with different λ values. As the iteration number increases, so decreases the difference between the subsequent λ values, and the difference between the cost of the paths also decreases.

So, let $p_c^0, p_c^1, p_c^2, \cdots, p_c^j$ be the series of paths (say from source s to destination t) that do not satisfy the delay constraint, and let be $p_d^0, p_d^1, p_d^2, \cdots, p_d^k$ the series of paths that satisfy the delay constraint.

Our aim is to stop the algorithm before it finds the optimal solution, but only when can assure that the result is not far from the optimum.

The algorithm gives p_d^k as the final solution, so we are interested in the difference between cost of the tags of the p_d-series and the last tag p_d^j. Let us examine the running algorithm in a certain intermediate iteration and let suppose that the current paths at the end of this iteration are p_c^a and p_d^b. Because $c(p_c^a) < c(p_d^k)$, we get that

$$c(p_d^b) - c(p_d^k) < c(p_d^b) - c(p_c^a). \qquad (9)$$

So, if the difference between $c(p_d^b)$ and $c(p_c^a)$ is small enough, then the difference between $c(p_d^b)$ and $c(p_d^k)$ will be possibly negligible, in which case there is no need to continue the running of the iteration.

We define an upper bound named *Maximal Difference* (MD), which shows in percentage that how much difference is tolerated between the cost of the original solution (that would be given by the original algorithm) and the cost of the new solution's path (that is given by the improved algorithm). Thus, if

$$c(p_d^b) \leq (1 + MD) * c(p_c^a) \qquad (10)$$

stands at the end of an iteration, then the algorithm can stop, since no further significant improvement on the resulting path's cost can be earned. Our tolerance toward the minimality of cost is represented in the value of MD.

IV. Numerical results

Since the complexity of the routing model precludes closed form analytical expressions, we used simulations to evaluate the performance of the LARAC algorithm and other QoS routing algorithms proposed in the literature. This section is structured as follows: we survey the implemented algorithms and measurements in Section IV-A, then in Section IV-B we give a description of the simulation environment. Finally in Section IV-C we introduce our simulation experiments.

A. Implemented Algorithms and Measurements

We investigated the performance of the *LARAC* algorithm and compared it to the most promising three algorithms selected from the ones surveyed in Section II. The *CBF* algorithm has been selected, because it gives the optimal solution, which can be a useful measure for the other algorithms. The *Fallback* algorithm has been selected for its simplicity; *DCUR*, because it was the most promising among the algorithms that compose the constrained path from other paths obtained by simple shortest path algorithms. Since LARAC can give the best solution that can be obtained with aggregated costs, we did not implement any other algorithms of this category.

We compared the behavior of the implemented algorithms with the help of the following measures:

- *Average Number of Unreachable Nodes*: It can happen that with a given delay constrain Δ_{delay} there is no path in the network to a certain destination. We need this measure to know whether all the algorithms can find paths to the same number of destination nodes.
- *Average Cost*: this attribute measures the average cost of the paths from a source node to all the reachable destination nodes.
- *Average Delay*: It is the average delay of the paths from a source node to all the reachable destination nodes.
- *Average Number of Steps*: This attribute measures the running time of the algorithms. Since both Dijkstra and CBF works with a heap, the number of step represents the events when the algorithm changes the contents of the heap. It will not give exact results, since the calculation of the number of steps is different for CBF and the other algorithms that use Dijkstra, however complexity measurements based on these calculations still provide us valuable feedback. For those algorithms that execute Dijkstra several times during its calculations, the complexity measure gives comparable results.
- *Cost Inefficiency*: this measure can be calculated from the optimal cost measure of the CBF algorithm [25], [15]. It can be defined as:

$$\delta_A := \frac{C(p_A) - C(p_{CBF})}{C(p_{CBF})} \qquad (11)$$

where A stands for the algorithm for which the cost inefficiency is calculated. We will use $\overline{\delta}_A$ and $max\delta_A$ to characterize comprehensively the results of an algorithms provided for different delay constraints.

B. Simulation Environment

For the simulation we built 100 random networks with 40 nodes and an average node degree of 4 (reflecting real network values [15]). The results of the measurements have been averaged. The topology of the networks was generated by a Random Graph Generator program [29]. The cost value on links varies from 1 to 15 based on uniform distribution. The propagation delay on links is selected from three ranges to resemble delay characteristics of a nationwide network e.g. in the US. The first range (1-5 ms) represents short local links, the second range (5-8) represents longer local links, while the third range (20-30 ms) represents continental links. The ratio of long local links was configured to be 20 percent, while the ratio of continental connections in networks was configured to be 5 percent.

In the simulations we changed the delay constraint Δ_{delay} and investigated its effect on the found paths of different algorithms. Since the average number of unreachable nodes is the same for all algorithms, the other measures of the routing algorithms are comparable.

C. Experiments

In Section IV-C.1 we first investigate the basic LARAC algorithms without the proposed improvements described in details in Section III-E, then Experiments of the improved LARAC algorithm are presented in Section IV-C.2.

C.1 Comparison of the four algorithms

As it was mentioned before an important result of the simulation is that the average number of unreachable nodes is the same for all algorithms. This means that all algorithms managed to find a path that satisfies the delay constraint. However, in minimizing the other additive metric (i.e., cost) the algorithms provide different results. In Figure 5 we can see the average cost of the paths found by the algorithms. The LARAC algorithm found almost the same paths as CBF, while DCUR found paths with higher cost, and the worst cost was achieved by Fallback. The result of Fallback meets our expectations, however we could not tell before that DCUR is worse than LARAC, although it could be seen that DCUR can easily find paths with higher cost than the optimal. Finally, LARAC's lower bound for the optimal path cost is very close to the results of CBF.

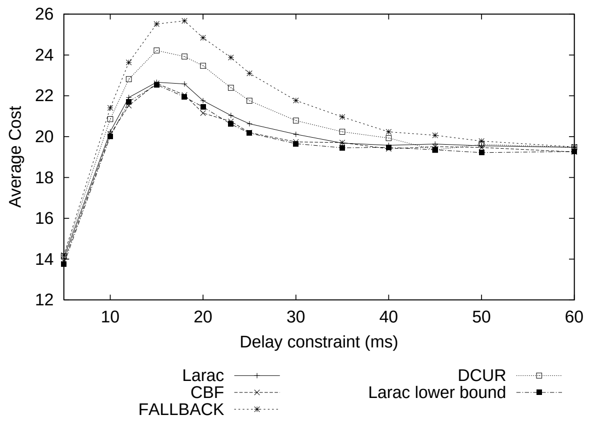

Fig. 5. Average cost

The maximum worst case cost $\max \delta_A$, and the average worst case cost $\overline{\delta}_A$ of the algorithms compared to the cost of the optimal CBF are the following (Table I):

TABLE I

MAXIMUM AND AVERAGE COST INEFFICIENCY

Algorithm	$\max \delta_A$	$\overline{\delta}_A$
LARAC	102.9%	101.3%
DCUR	110.9%	104.7%
Fallback	117.4%	108.7%
LARAC lower bound	99.27%	99.7%

It can be seen in Figure 5 that at the first section of the curves ($\Delta_{delay} < 15$) the found paths have very low cost, and after the cost of the path increases with the delay bound. The explanation of this phenomenon is that it is impossible to find path for all source destination pairs with a small Δ_{delay}, and the found paths are short, thus the costs of them are also small. As the delay bound increases the algorithms find more and more (longer) paths, therefore the average cost of the paths increases. After the point where a path can be found for all source destination pairs, the average cost of the paths will decrease with the delay bound, because the algorithms are able to find the paths with lower costs and higher delay bounds.

This effect can be noticed on the delay curves (Figure 6) as well. We expected CBF to have the highest delay, then LARAC, DCUR and Fallback. The result we got is almost meets our expectations, but for surprise DCUR has the highest delay.

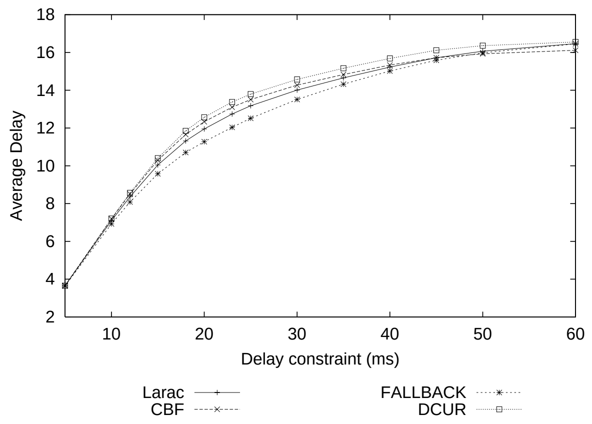

Fig. 6. Average delay

In Figure 7 the average number of steps of the algorithms are compared. As we can see all algorithms have a very characteristic curve for the number of steps.

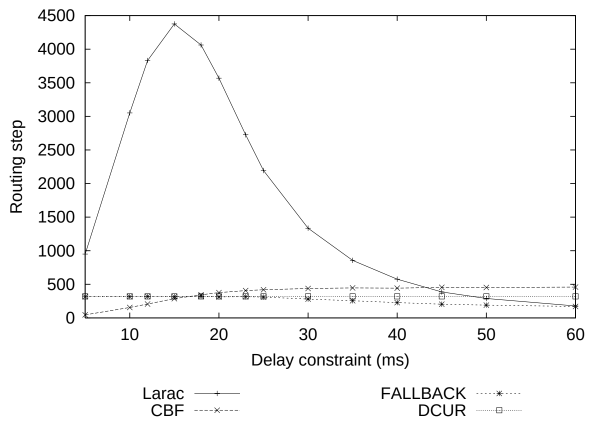

Fig. 7. Average number of step

For the small delay constraints LARAC has an increasing running time, since as it can find paths to more and more destinations, the number of Dijkstra executions increases. As it reaches the delay constraint that allows paths to all the nodes in the network, the number of steps starts to decrease since minimum cost paths computed first can more likely satisfy the looser delay bounds, thus eliminating the execution of the second Dijkstra algorithm to get the minimum delay path. This is the reason why the average number of steps for the LARAC algorithm approximates the number of step of a single Dijkstra algorithm for high delay values.

Fallback has a decreasing characteristic as well, the difference is that it is linear in the first section, and equals to the running time of 2 Dijkstra algorithms. After this first section, the number of steps decreases to the number of steps of 1 Dijkstra, for similar reasons as it was explained for LARAC.

We implemented the DCUR algorithm in a centralized manner for the simulations. For this reason the running time of it would be huge ($2n$ times the running time of a Dijkstra algorithm), so in the figure an estimation is plotted for a distributed implementation. We used the execution time of 2 Dijkstra algorithms for the running time estimation of the distributed DCUR.

CBF has a totally different characteristic. Since the algorithm exit condition depends only on the delay constraint, it increases with Δ_{delay}. The last horizontal section shows, that the algorithm stopped because all the paths from the heap has been already used.

C.2 Improvements to LARAC algorithm

We investigated both optimization proposal's effect on the performance of the LARAC algorithm. The first optimization (storing former results) aims at improving the running time of the basic LARAC algorithm. Based on our simulations we can say that the gain on the routing step is quite significant. Under the critical value of delay bound (which is 15 ms) up to 39% gain is earned.

The second optimization (improved stop condition) affects not only the running time but also the cost of the found paths, since it modifies the exit condition of the algorithm.

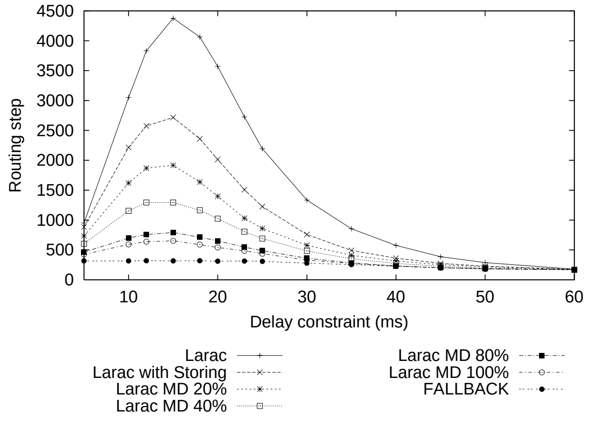

Fig. 8. Average number of step

Figure 8 and 9 show the average routing step of the algorithms and average cost of the computed paths. As we can see on figure 8, the running time greatly decreases as we increase the value of Maximal Difference (or shortly MD). Four different results are presented, one for each of the different MD values. Table II summarizes the improvement of the algorithms' running time with the help of *average iteration number*. This provides us the average number of times the Dijkstra algorithm was executed with a given algorithm. As we can notice, the original LARAC algorithm had an average iteration number of 7.94, while the improved one with e.g., MD 40% this value decreased to 4.34.

These values are very impressive, but the gain on routing steps is only one side of the coin. We have to examine the average

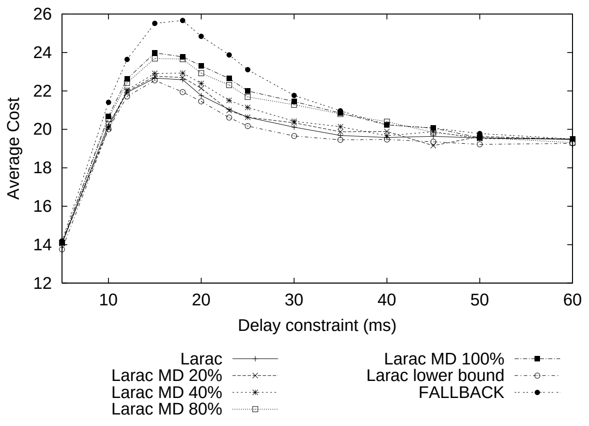

Fig. 9. Average cost

cost of the found paths as well. In Figure 10 and in Table IV-C.2 the increase in cost is presented. When the MD value 40% was used, the average cost inefficiency $\overline{\delta}_A$ increased to 102.4%. Similarly at the MD value 80%, $\overline{\delta}_A$ equals to 104.6%. The gain on running time compared to the loss on the optimality of path costs seems to be worthy. Moreover, our algorithm provides a good way of controlling the trade-off between optimality of the path, and running time of the algorithm.

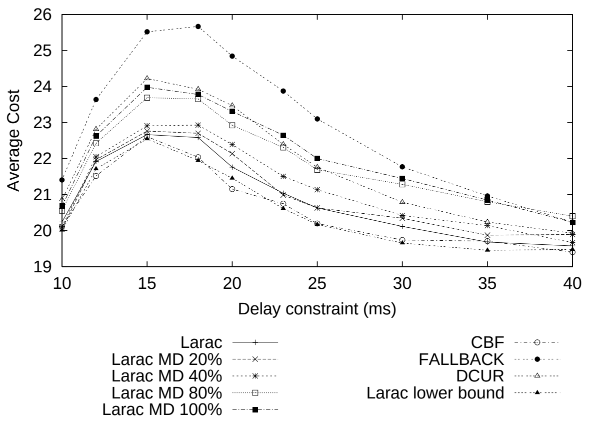

Fig. 10. Average cost

C.3 A special network for CBF

In the 40 node random networks CBF had an acceptable running time, although for high delay constraints it has higher running time than the other algorithms. We found that the execution

TABLE II
SUMMARY OF AVERAGE ITERATION NUMBER AND OPTIMALITY OF COSTS

Algorithm	Average Iteration Number	$\max \delta_A$	$\overline{\delta}_A$
LARAC	7.94	102.9%	101.3%
MD 20%	5.80	104.7%	101.5%
MD 40%	4.34	105.8%	102.4%
MD 80%	2.92	108.4%	104.6%
MD 100%	2.57	110.2%	105.4%

time of CBF increases faster than the other algorithms as the number of nodes in the network increases, therefore for large networks it may have an undesirable high execution time. In Figure 11 a network type is shown, where CBF has a worse execution time in some of the nodes than the other algorithms. In

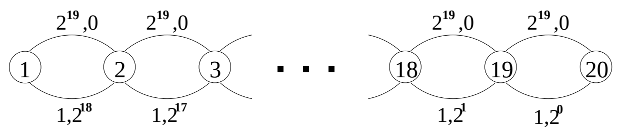

Fig. 11. A special network for CBF

this network each node i is connected to node $i + 1$ with two edges. The cost and delay of the links are marked on the arrows. The "upper" links have high cost, and low delay, in such a way that the cost of one upper link is always higher than the sum of the costs of the lower links. This arrangement increase the running time of CBF, since the delay will be different for all paths that can be obtained.

It can be easily seen that on these kind of chain networks the running time of CBF is exponential in number m of the links. By adding a node and two more links, the running time of the algorithm doubles.

V. Conclusions and Future Work

We proposed a new algorithm called LARAC for the well known Delay Constrained Least Cost problem, which is the most common QoS routing problem when delay sensitive traffic should be considered. We evaluated the performance of different QoS routing algorithms by simulation. We compared the running time of the algorithms and the optimality of the paths they found in terms of cost and delay.

The original LARAC algorithm was the best among the polynomial heuristic algorithms proposed so far in the literature. Its cost are the closest to the optimal cost computed by the CBF algorithm, moreover it gives an estimate of the optimality of the found paths. We also proposed two improvements for the LARAC algorithm. The first one decreased the running time significantly without effecting the cost of the paths, while the second improvement further decreased the running time but with a controllable loss of optimality. The improved LARAC algorithm running time is only slightly greater than that of CBF, and due to its stability in all kind of networks, it can be a reasonable algorithm to solve the DCLC problem.

Also we concluded from the simulations, that the DCUR algorithm finds routes with both higher cost and delay than LARAC. The Fallback algorithm has a very good execution time, and it is easy to implement, but the cost of it is the worst among the implemented algorithms. CBF gives the optimal cost solution and it has a surprisingly good execution time in most of the cases, but in certain situations the running time of it could be undesirably high.

A possible further improvement to the LARAC algorithm can be its extension to handle two given constraints while minimizing a third one. An example for the two constraints can be delay and hop-number. The aggregated cost function then should contain three tag (cost, delay, hop-number). Our expectations are that the complexity of the algorithm does not increase in a great manner during the extension, while this is not true in the case of CBF algorithm.

Acknowledgments

The authors would like to acknowledge the suggestions of Áron Szentesi.

References

[1] L. Georgiadis, R. Guerin, V. Peris and R. Rajan, *'Efficient Support of Delay and Rate Guarantees in an Internet'*, IBM T.J. Watson Research Center

[2] A.K. Parekh and R.G. Gallager, *'A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks - The Multiple Node Case'*, IEEE/ACM Transactions on Networking, 1(3):344-357, June 1993.

[3] A.K. Parekh and R.G. Gallager, *'A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks - The Multiple Node Case'*, IEEE/ACM Transactions on Networking, 2(2):137-150, April 1994.

[4] R. Szabó, P. Barta, J. Bíró, F. Németh, C-G. Perntz, *'Non Rate-Proportional Weighting of Generalized Processor Sharing Schedulers'* GLOBECOM'99, Dec. 1999.

[5] R. L. Cruz, *'A Calculus for Network Delay'*, IEEE Transactions on Information Theory, vol.37, no. 1, pp.114-141, January 1991.

[6] S. Rampal, R. Guerin, *'Flow Grouping For Reducing reservation Requirements for Guaranteed Delay Service'*, July 1997

[7] Stefan Savage, Andy Collins, Eric Hoffman, Thomas Anderson, *'The End-to-End Effects of Internet Path Selection'* ACM SIGCOMM '99 pp.289-299 September 3, 1999.

[8] Roch A. Guerin, et al., *'QoS Routing mechanisms and OSPF Extensions'*, Internet Engineering Task Force, Internet Draft, December 1998, (Work in Progress).

[9] Guerin Roch A.; Orda Ariel, *'QoS routing in networks with inaccurate information: theory and algorithms'* IEEE/ACM-Transactions-on-Networking. vol.7, no.3; p.350-64; June 1999.

[10] G. Apostolopoulos, R. Guérin and S. Kamat, *'Implementation and Performance measurements of QoS routing extensions to OSPF'* in Proceedings of INFOCOM'1999, New York, March 1999.

[11] S. Shenker, C. Partridge, R. Guérin, *'Specification of guaranteed quality of service'* Request For Comments (Proposed Standard) RFC2212, Internet Engineering Task Force, September 1997.

[12] Zheng Wang and Jon Crowcroft, *'Bandwidth-Delay based routing algorithms'*, IEEE GlobeCom'95, Singapore, November 1995.

[13] Funda Ergun, Rakesh Sinha, Lisa Zhang, *'QoS Routing with Performance-Dependent Costs'* IEEE INFOCOM'2000 pp., March, 2000.

[14] M. S. Garey, D.S. Johnson, *'Computers and Intractability: Guide to the Theory of NP-Completeness'*, W. H. Freeman, New York, 1979.

[15] Hussein F. Salama, Douglas S. Reeves and Yannis Viniotis, *'A Distributed Algorithm for Delay-Constrained Unicast Routing'*, IEEE Infocom'97, Kobe, Japan, April 1997.

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

[17] W.C.Lee, et al. *'Multi-Criteria Routing subject to Resource and Performance Constraints'*, ATM Forum 94-0280, March 1994.

[18] C. Pornavalai, G. Chakraborty and N. Shiratori, *'Routing with multiple QoS requirements for supporting multimedia applications'*, Journal of High Speed Networks, 1998.

[19] Kenji Ishida and Kitsutaro Amano, *'A Delay-Constrained Least-Cost Path Routing Protocol and the Synthesis Method'*, IEEE 1998.

[20] Shigang Cheng and Klara Nahrstedt, *'On finding multi-constrained paths'* ICC'98, Atlanta, Georgia, 1998.

[21] Jeffrey Jaffe, *'Algorithms for finding path with multiple constraints'*, Networks, vol. 14, pp.95-116, 1984.

[22] David Blokh and George Gutin, *'An approximation algorithm for combinatorial optimization problems with two parameters'*, 1995.

[23] Ravindra K. Ahuja, Tomas L. Magnanti and James B. Orlin, *'Network Flows'* PRENTICE HALL, Upper Saddle River, New Jersey 07458, 1993.

[24] H. de Neve, P. van Mieghem, *'A multiple quality of service routing algorithm for PNNI'*, IEEE ATM'98 Workshop, pp.306-314, Fairfax, Virginia, May 1998.

[25] Liang Guo and Ibrahim Matta, *'Search Space Reduction in QoS Routing'* Technical Report NU-CCS-98-09, October 1998.

[26] Atsushi Iwata, et al. *'ATM Routing Algorithms with Multiple QoS Requirements for Multimedia Internetworking'*, IEICE Trans. Commun., vol.E79-B, pp.999-1007, August 1996
[27] M. Held and R. Karp, *'The traveling salesman problem and minimum spanning trees'*, Operations Research 18, pp.1138-1162, 1970.
[28] M. Held and R. Karp, *'The traveling salesman problem and minimum spanning trees, Part II.'*, Mathematical Programming 6, pp.62-88, 1971.
[29] Balázs Gábor Józsa and Dániel Orincsay, *'Random Graph Generator (for telecommunication networks)'*, Technical Report, April 1999.
[30] T. Radzik, *'Fractional combinatorial optimization'* , In *Handbook of Combinatorial Optimization*, editors DingZhu Du and Panos Pardalos, vol. 1, Kluwer Academic Publishers, December 1998.
[31] R. Hassin, *'Approximation schemes for the restricted shortest path problem'*, Mathematics of Operations Research, 17(1):36–42, Feb 1992.
[32] D.H. Lorenz and A. Orda, *'QoS routing in networks with uncertain parameters'*, IEEE/ACM Transactions on Networking, 6(6):768–778, Dec 1998.
[33] Danny Raz and Yuval Shavitt, *'Optimal Partition of QoS requirements with Discrete Cost Functions'*, INFOCOM 2000
[34] Dean H. Lorenz, Ariel Orda, Danny Raz, and Yuval Shavitt, *'Efficient QoS Partition and Routing of Unicast and Multicast'*, IWQoS 2000, June 2000.