QoS Routing with Performance-Dependent Costs

Funda Ergun, Rakesh Sinha, Lisa Zhang
Bell Laboratories
600-700 Mountain Avenue
Murray Hill, NJ 07974
{ergun, rks1, ylz}@research.bell-labs.com

Abstract— **We study a network model in which each network link is associated with a set of delays and costs. These costs are a function of the delays and reflect the prices paid in return for delay guarantees. Such a cost structure can model a setting in which the service provider provides multiple service classes with a different price and delay guarantee for each class.**

We are given a source node s, a sink node t, and an end-to-end delay constraint D. Our aim is to choose an s-t path and determine the per link delay constraints along this path so as to satisfy the constraint D while minimizing the total cost incurred. In the case where the s-t path is known, we aim to optimally partition the end-to-end delay constraint into link constraints along the path.

We present approximation algorithms for both problems, since they are known to be NP-hard. Our algorithms guarantee to produce solutions that are within a factor $1+\varepsilon$ of the optimal, where ε is a parameter of our choice. The running times are polynomial in the input size and $1/\varepsilon$. We also provide a number of heuristics for the second problem and present simulation results. All our results apply to general cost functions.

Previous work on related problems either focuses on optimal solutions for special cost functions or on heuristics that have no performance guarantees.

Keywords— **QoS routing, performance-dependent costs, approximation algorithms, heuristics.**

I. INTRODUCTION

Today's Internet deploys *best effort* routing. That is, its primary focus is on providing connectivity without any assurance of service quality. Given the dynamic nature of the traffic on the Internet, it is possible and allowable, for instance, for data packets to encounter unexpected delay. The best effort routing paradigm has been adequate for traditional applications like File Transfer Protocol (FTP). However, many modern applications such as real time audio or video transmission need to satisfy strict performance criteria to be acceptable.

There is a growing consensus that the future Internet will have to support various *quality of service* (QoS) classes in addition to the best effort service class. Each class will have its own set of service guarantees, and data packets from certain classes of applications will be favored over others with the aim of satisfying the performance requirements of each application. Naturally, the owners of the applications with more stringent requirements should be charged a higher fee than the ones with less stringent requirements. (Internet will be moving from a socialistic model to a capitalistic model.) Simple forms of QoS routing have been used in the past for type of service routing [1], and several proposals have been advanced for supporting various forms of guaranteed service classes [2], [3], [4].

A core component of many of these proposals is the identification of a routing path of an application based on its QoS requirements and the resource availability of the network. The QoS requirements of an application are specified either as a set of *path-constraints* (requirements on the entire path) or a set of *link-constraints* (requirements on individual links) [5], [6]. A path with sufficient resources to satisfy the QoS requirements of an application is called a *feasible* path. In addition, one or more optimization criteria may further narrow the selection among feasible paths. The goal of QoS routing [7] then is to find the optimal feasible path.

In addition to improving the performances of individual applications, QoS routing may also improve the overall network efficiency [8], [5], [9]. However, these gains have to be carefully weighed against the increased complexity of maintaining extra state in the network and the computational cost of running QoS routing algorithms. Thus, there is a critical need for efficient QoS routing algorithms. Unfortunately, most QoS problems belong to a problem class called "multi-objective optimization", which largely turns out to be NP-hard. Past efforts have concentrated on finding various heuristics, i.e. sub-optimal algorithms (often, without any performance guarantees) or algorithms that are guaranteed to be optimal only for certain special cases.

In this paper, we consider a model in which each application pays performance-dependent fees. In particular, to traverse each network link an application pays a fee that reflects the desired delay guarantee on the link.

Such a cost structure can model a setting in which the service provider provides multiple service classes with a different price and delay guarantee for each class, e.g. [10]. It can also model *network with uncertain parameters* where the delay on each link is given as a probability distribution [11], [12].

The Problems. We consider a network of n nodes and m links in which each link is characterized by a set of delays and their corresponding costs. We use function $c_e(d)$ to represent the cost incurred by delay d on link e. For a given source node s and a destination node t, an end-to-end delay constraint of D needs to be satisfied. We study the following two problems:

1. Constrained minimum cost path (PATH): We are to choose an s-t path and determine the delay bound to be required from every link along this path. Our goal is to minimize the sum of the link costs along the path subject to the end-to-end delay constraint D.
2. Constrained minimum cost partition (PARTITION): s-t path P of p links is already chosen. We are to determine the delay bound to be imposed on every link along path P such that the sum of the link costs is minimized subject to the end-to-end delay constraint D.

PARTITION maps a path constraint of PATH into link constraints. PATH is a generalization of the restricted shortest path problem (RSP), in which each link has a fixed cost c_e and delay d_e.

Our Results. Since both PATH and PARTITION are NP-hard, we present approximation algorithms. An ε-*approximation algorithm* produces a path cost that is within a factor $1+\varepsilon$ of the optimal (minimum) path cost. The parameter ε here allows a trade-off between running time and goodness of approximation. The general flavor is that a running time polynomial in $\frac{1}{\varepsilon}$ guarantees a solution within $1+\varepsilon$ of the optimal. Depending on the application, we can tune the performance of our algorithms by choosing an appropriate ε. Our results are as follows.

APPROXIMATION ALGORITHMS. We present ε-approximation algorithms for PATH and PARTITION . The running time is polynomial in the input size and the approximation parameter, i.e. polynomial in $m, n, \log D$ and $\frac{1}{\varepsilon}$. See Theorems 2 and 3.

We note that our algorithms are the first to have polynomial running time and performance guarantees for *general* cost functions. Our algorithms also work for any additive constraints, i.e. the value of the constraint for the path is summation of the values of the constraints for individual links on the path. If the constraints are multiplicative, e.g. loss rate, one can work with the logarithms. For the special case in which all cost functions are convex, optimal algorithms with polynomial running times are given in [11], [10].

HEURISTICS. We present a number of heuristics for PARTITION . We show through simulation results that our heuristics work well both in terms of performance and running time. For instance, one of our heuristics returns near-optimal solutions and runs in $O(\log D(p + \log p \log D + \log^2 D))$ time on the average where p is the number of links along the path. (We analyze both average and worst-case running times in Section IV-A.)

As pointed out in [11], [13], [12], [14], full information on network parameters is typically unavailable. In other words, even an optimal algorithm will have to base its decision on somewhat imprecise information, resulting in a sub-optimal solution. Thus it makes practical sense to have efficient algorithms that solve the problem approximately.

Related Work. It is easy to see that both PATH and PARTITION can be solved optimally with dynamic programming [12]. However, the running time $O(D^2 \cdot m)$ is pseudo-polynomial.

The restricted shortest path problem (RSP), a special case of PATH , is also NP-hard [15]. Hassin [16] gave approximation algorithms for solving RSP. He also introduced a nice technique called *rounding-and-scaling*, i.e. mapping the set of possible delay values into a smaller range, which formed the basis of many subsequent heuristics and approximation algorithms for RSP [17], [18]. Our approximation algorithms are also inspired by Hassin's ideas.

Several studies on frameworks related to PARTITION have been conducted, e.g. [19], [6], [10] and the references therein.

Lorenz and Orda [11] and Guerin and Orda [12] investigated both PATH and PARTITION in the context of *networks with uncertain parameters*. In their studies, the delay on each link is given as a probability distribution $p_e(d)$ and their goal is to maximize the *product* of the probabilities of the path links. If our cost function $c_e(d)$ is set to $-\log p_e(d)$, minimizing the sum of the costs is equivalent to maximizing the product of the probabilities.

In [11] the authors presented a greedy algorithm which is optimal when $-\log p_e(d)$ is convex for all links e. (This is the interesting case of practical importance under the probabilistic setting.) Variants of the greedy algorithm for PARTITION with improved polynomial running time are given in [10] for convex cost functions.

Chen and Nahrstedt [14] consider a slightly different model of imprecision in the network information. Instead of maintaining a probability distribution of delay for each link, they maintain a range of possible delay for each link.

They give a distributed heuristic for PATH .

Organization of the Paper. In Section II we give the basic notations. In Section III we present polynomial-time approximation algorithms for PATH and PARTITION . Section IV contains heuristics for PARTITION , and in Section V we present our conclusions.

II. Preliminaries

Network Parameters. We represent our network N as a graph of n nodes and m links. Given an s-t path P, let p be the number of links on the path.

Each link e has an associated cost function c_e such that $c_e(d)$ denotes the cost of buying a guarantee from the provider that link e will have delay no more than d. Note that this definition implies that $c_e(d)$ is a nonincreasing function, i.e. $c_e(d_1) \geq c_e(d_2)$ for $d_1 \leq d_2$. We follow the convention of previous work and work with integral delays and costs. Our cost functions are completely unconstrained otherwise.

Given link e and delay d, we can retrieve the cost $c_e(d)$ in constant time. We use $d_e(c)$ to denote the "inverse" of $c_e(d)$ and it returns the smallest delay that incurs cost at most c on link e. If no delay incurs cost c then $d_e(c)$ is infinite. For given link e and cost c, we can compute the delay in $\log D$ time using binary search.

As described in Section I we are given an end-to-end delay constraint D from a source node s to a destination node t. We use OPT to denote the cost of the optimal s-t path subject to the delay constraint. We use C to denote the maximum possible cost on any link, i.e. $C = \max_e c_e(1)$.

Graph representation. There are two standard data-structures for representing graphs: adjacency list and adajacency matrix. Since the cost of transforming from one representation to the other (at most $O(n^2)$) is dominated by the running time of our algorithms, we do not explicitly specify which data structure we are using. Instead, we assume whichever data structure simplifies the presentation.

III. Polynomial-time Approximation

Our approximation algorithms for PATH and PARTITION are based on the approximation for RSP. Recall that RSP is a restricted version of PATH , where each link has a fixed cost and delay. The following result is given by Hassin in [16].

Theorem 1: RSP has an ε-approximation algorithm with running time $O(\frac{1}{\varepsilon}mn\log\log U)$, where U is any upper bound on the optimal solution OPT. (For example, U can be nC.)

In the following sections we present three ε-approximation algorithms for PATH and PARTITION . Combining their running times of the three algorithms, we have the following theorems for general cost functions.

Theorem 2: PATH has an ε-approximation algorithm with running time $O(X\frac{1}{\varepsilon}mn\log\log U)$, where $X = \min\{D, \frac{\log C}{\varepsilon} + \log D, \frac{n}{\varepsilon} + \log D\}$ and U is any upper bound on the optimal solution OPT.

The network for PARTITION effectively consists of the nodes and links along the given s-t path. Hence,

Theorem 3: PARTITION has an ε-approximation algorithm with running time $O(X\frac{1}{\varepsilon}p^2\log\log U)$, where $X = \min\{D, \frac{\log C}{\varepsilon} + \log D, \frac{p}{\varepsilon} + \log D\}$ and U is any upper bound on the optimal solution OPT.

If the "inverse" function $d_e(c)$ can be computed in constant time for given cost c as in [19], then the running times in Theorems 2 and 3 can be further improved. In the following presentation, we focus on the problem PATH ; our techniques apply to PARTITION by setting the network to be the s-t path.

A. Algorithm 1

In order to derive an approximation algorithm for PATH from the approximation algorithm for RSP, we transform the network N (for PATH) into a network N_1 (for RSP) such that any optimal RSP solution of N_1 is equivalent to an optimal PATH solution of N. In particular, we replace each link e of N by D parallel links $e_1, \ldots, e_D$, where link e_i has cost $c_e(i)$ and delay i.

We apply Hassin's approximation algorithm for RSP to N_1. Then we map the resulting s-t path to a path in network N by replacing link e_i with link e with delay i and cost $c_e(i)$. It is straightforward to verify that we have obtained an ε-approximation of PATH on N.

Now we analyze the running time, which has two components: time of creating N_1 and time of applying approximation algorithm for RSP. Since each link in N is replaced by D links in N_1, the time for creating N_1 is mD and the time for running Hassin's algorithm is $O\left(D\frac{1}{\varepsilon}mn\log\log U\right)$.

Lemma 4: Algorithm 1 is an ε-approximation of PATH with running time $O\left(D\frac{1}{\varepsilon}mn\log\log U\right)$.

This running time is reasonable for small D. We also remark that each link e of N can be replaced by $\min\{c_e(1), D\}$ links in N_1. This is because if $c_e(i) = c_e(j)$ for $i < j$ then we discard link e_j. Hence, the number of links in N_1 is $m\min\{C, D\}$ and the running time in Lemma 4 can be improved to $O\left(mD + \min\{C, D\}\frac{1}{\varepsilon}mn\log\log U\right)$.

B. Algorithm 2

In Algorithm 1, we created a new network N_1 such that any optimal RSP solution of N_1 is equivalent to an optimal PATH solution of N. There is a price to be paid for such a strong guarantee, namely we replace link e with D links. The key idea of Algorithm 2 is to achieve a weaker guarantee on the transformation with far fewer links. We will first describe this informally. In Algorithm 1, the goal was to capture all possible choices of cost and delay assignments for link e. In Algorithm 2, instead of getting all possible cost and delay pairs, we subdivide the range of cost, $[1, c_e(1)]$, into $\log_{1+\varepsilon} c_e(1)$ many sub-ranges and pick at most one representative from each sub-range. We illustrate this transformation in Figure 1.

The advantage is that we have reduced the linear "blow-up" to a logarithmic "blow-up." The disadvantage is that we are approximating an entire sub-range of cost by its representative. The goal is to keep each sub-range small enough so that any cost is within $1+\varepsilon$ of the representative cost in its sub-range. Then we apply the approximation algorithm for RSP to this new network. As will be shown later, the transformation may introduce an error factor of $1+\varepsilon$ and the approximation algorithm may introduce another error factor of $1+\varepsilon$, resulting in an $(1+\varepsilon)^2$-approximation algorithm. By starting with an ε' such that $(1+\varepsilon')^2 \approx \varepsilon$, we get an ε-approximation algorithm.

A precise description follows. Consider each link e in N and each semi-open subrange,

$$\left(\left\lceil \frac{c_e(1)}{(1+\varepsilon)^{i+1}} \right\rceil, \left\lceil \frac{c_e(1)}{(1+\varepsilon)^{i}} \right\rceil \right],$$

where $0 \leq i \leq \log_{1+\varepsilon} c_e(1)$. We find the minimum delay d_i that incurs a cost within the above range. If such a delay exists, we create a link in N_2 with delay d_i and cost $c_e(d_i)$.

We apply the approximation algorithm for RSP to N_2 and map the resulting s-t path to a path in network N (as in Algorithm 1). To see that the approximate cost is at most $(1+\varepsilon)^2 OPT$, we observe that the optimal RSP solution to N_2 is at most $(1+\varepsilon)OPT$. (Recall OPT is the optimal cost of PATH on N and also the optimal cost of RSP on N_1 of Algorithm 1.) Suppose a link along the optimal path of N_1 corresponds to link e in N and has delay d and cost $c_e(d)$. Due to the construction of N_2, we can find a link in N_2 that corresponds to link e and has delay at most d and cost at most $(1+\varepsilon)c_e(d)$.

In terms of running time, each link e of n is replaced by $\log_{1+\varepsilon} c_e(1) \leq \log_{1+\varepsilon} C$ links. Creating N_2 requires at most $m \log_{1+\varepsilon} C$ many computations of $d_e(\cdot)$ function, resulting in running time $O\left(m \log_{1+\varepsilon} C \log D\right)$. The number of links in N_2 is at most $m \log_{1+\varepsilon} C$. Note that $(1+\varepsilon)^2 \approx 1+2\varepsilon$ and $\ln(1+\varepsilon) \approx \varepsilon$ for small ε. Thus, by setting ϵ' to a value slightly under $\epsilon/2$ ($\epsilon' = \epsilon/2.5$ is good enough, larger values of ϵ' can be used if ϵ is smaller) and running the algorithm for ϵ', without changing the asymptotic running time, we obtain an ε-approximation in time $O\left(\frac{m}{\varepsilon} \log C \log D + \frac{\log C}{\varepsilon} \frac{mn}{\varepsilon} \log\log U\right)$.

Lemma 5: Algorithm 2 is an ε-approximation of PATH with running time $O\left(\frac{m}{\varepsilon} \log C \log D + \frac{\log C}{\varepsilon} \frac{mn}{\varepsilon} \log\log U\right)$.

C. Algorithm 3

We now present an algorithm that is based on the RSP approximation of Hassin [16]. The high-level idea is as follows. Suppose we had an exact test procedure that determines whether or not OPT is greater than some given value V. Then we could start with some upper bound U on OPT and use this procedure to do a binary search for the exact value of OPT.

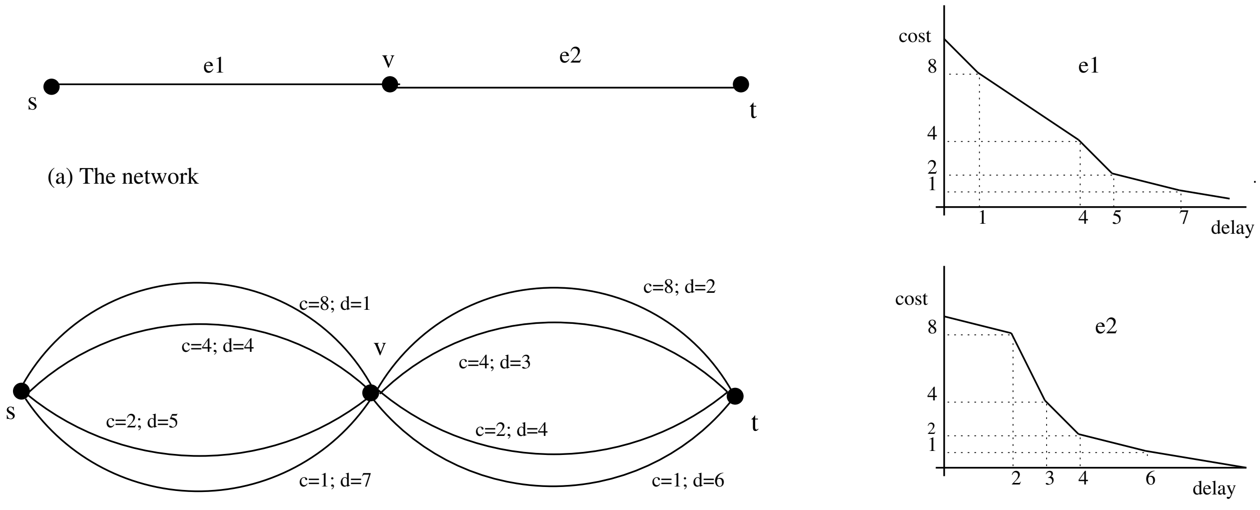
We do not know how to design such a procedure efficiently. Instead we have an approximate version. Specifically, we create a test procedure $TEST$ that determines *approximately* whether or not OPT is more than some given value V. More precisely, $TEST$ determines whether $OPT \geq V$ or $OPT \leq (1+\varepsilon)V$.

We maintain a range, $[L, (1+\varepsilon)U]$, for OPT. To approximate OPT, we repeatedly narrow the gap between its upper and lower bound. Initially, U can be set to nC and L can be set to 1. If $TEST$ is applied to value $V = \sqrt{U \cdot L}$, then we can either reduce the upper bound to $V(1+\varepsilon)$ or increase the lower bound to V. Once U is within a constant factor of L, we determine our solution path using a variation of the $TEST$ procedure.

An Exact Solution. To understand the $TEST$ procedure let us first study a pseudo-polynomial time algorithm that produces an exact solution via dynamic programming. Let $g_v(c)$ be the minimum delay path from s to v whose total cost is c. We begin with $c = 0$. We compute $g_v(c)$ for all nodes v based on $g_v(0), g_v(1), \ldots, g_v(c-1)$. (Line 5 of Figure 2.) Since the minimum delay s-v path goes through some intermediate node u (where uv is a link) and accumulates some cost b up to node u, we obtain $g_v(c)$ by minimizing over all possible intermediate nodes u and all possible costs $b < c$.

By definition, OPT is the smallest delay of an s-t path with delay at most D. So $g_t(OPT) \leq D$ and for any $c < OPT$, $g_t(c) > D$. Therefore the smallest c such that $g_t(c)$ is no more than D is equal to OPT.

Each iteration of the for-loop on line 4 involves computing m instances of the function $d_e(\cdot)$ on line 5 and a minimization over all links and all values of c on line 6. Thus the running time of one iteration is $O(m \log D + mc)$. The



(a) The network

(b) Cost functions for the links.

(c) Modified network.

Fig. 1. Transformation of network for approximation.

$EXACT$

1 initialization
2 for all $c \geq 0$, set $g_s(c) = 0$
3 for all $v \neq s$, set $g_v(0) = \infty$
4 for $c = 1, 2, 3, \ldots$
5 for all e, compute $d_e(c)$
6 for all $v \neq s$, set $g_v(c) = \min_{b:b<c} \min_{u:uv \in E} \{ g_u(b) + d_{uv}(c-b) \}$
7 if $g_t(c) \leq D$
8 output $OPT = c$ and its corresponding path, exit.

Fig. 2. An exact solution.

for-loop has OPT iterations, so that the overall running time is $O(mOPT \log D + mOPT^2)$.

The TEST Procedure. We now define our polynomial-time ε-approximate test procedure. The procedure $TEST$ approximately determines if a given value V is greater than OPT. It relies on a "rounding-and-scaling" technique. $TEST$ resembles $EXACT$, except that the costs $c_e(d)$ are scaled down by a factor of $V\varepsilon/n$ and for-loop of line 4 is executed for $c = 1, 2, \ldots, \lceil n/\varepsilon \rceil$ only. If a path of delay at most D is found for some $c \leq \lceil n/\varepsilon \rceil$, then $TEST$ outputs $OPT \leq (1+\varepsilon)V$; otherwise $TEST$ outputs $OPT \geq V$.

The procedure $TEST$ is described in Figure 3. The scaled down cost and its inverse are denoted by $\hat{c}_e(d)$ and $\hat{d}_e(c)$ respectively. In particular, $\hat{c}_e(d) = \left\lfloor \frac{c_e(d)}{V\varepsilon/n} \right\rfloor$, and $\hat{d}_e(c)$ returns the smallest delay that incurs cost at most c on link e under the cost function $\hat{c}$. It takes constant(resp. $O(\log D)$) time to compute $\hat{c}_e(\cdot)$ (resp. $\hat{d}_e(\cdot)$).

Lemma 6: $TEST$ is an ε-approximation test. Its running time is bounded by $O\left(\frac{n}{\varepsilon} m \log D + \frac{n^2}{\varepsilon^2} m\right)$.

Proof: If $TEST$ outputs $OPT \leq (1+\varepsilon)V$, then $TEST$ has found a path with delay at most D and cost at most n/ε under the cost function $\hat{c}_e(d)$. Let P be this path. We want to determine the cost of P under the cost function $c_e(\cdot)$. Observe that $\hat{c}_e(\cdot) + 1 = \left\lfloor \frac{c_e(\cdot)}{V\varepsilon/n} \right\rfloor + 1 > \frac{c_e(\cdot)}{V\varepsilon/n}$. Thus the cost of P under the cost function $c_e(\cdot)$ is

$$
\begin{aligned}
& \sum_{e \in P} c_e(\cdot) \\
< & \sum_{e \in P} \left(\hat{c}_e(\cdot) + 1 \right) V\varepsilon/n \\
= & V\varepsilon/n \left(\sum_{e \in P} \hat{c}_e(\cdot) + \sum_{e \in P} 1 \right) \\
= & V\varepsilon/n \left(\text{cost of } P \text{ under } \hat{c}_e(\cdot) + \text{number of links in } P \right) \\
\leq & V\varepsilon/n \cdot (n/\varepsilon + n) \\
= & (1+\varepsilon)V.
\end{aligned}
$$

Since OPT is upper bounded by the cost of path P, OPT indeed is at most $(1+\varepsilon)V$.

```
TEST(V)
initialization
      for all c ≥ 0,       set   g_s(c) = 0
      for all v ≠ s,       set   g_v(0) = ∞
for c = 1, 2, ..., ⌊n/ε⌋
      for all e,           compute  d̂_e(c)
      for all v ≠ s,       set   g_v(c) = min_{b:b<c} min_{u:uv∈E} { g_u(b) + d̂_uv(c − b) }
      if g_t(c) ≤ D
            output OPT ≤ (1 + ε)V, exit
output OPT ≥ V
```

Fig. 3. A polynomial-time ε-approximation test.

If $TEST$ outputs $OPT \geq V$, then every path with delay at most D must have cost at least n/ε under the cost function $\hat{c}_e(\cdot)$. Hence, all these paths have costs at least $(n/\varepsilon)(V\varepsilon/n) = V$ under the cost function $c_e(\cdot)$. Hence, OPT indeed is at least V.

The bound on running time follows by an argument similar to that of procedure $EXACT$. ■

The Approximation Algorithm. In Figure 4 we present an ϵ-approximation algorithm.

Lemma 7: $APPROX$ is an ε-approximation algorithm with running time $O\left(\left(\frac{n}{\varepsilon} + \log D\right)\frac{1}{\varepsilon}mn \log\log U\right)$.

Proof: Let P be the optimal path with cost OPT (under cost function $c_e(\cdot)$). Let $\hat{P}$ be the path returned by $EXACT$ under the scaled cost $\hat{c}_e(\cdot)$. (See lines 8-9 of Figure 4.) We want to show that the cost of path $\hat{P} = \sum_{e\in\hat{P}} c_e(\cdot) \leq (1+\varepsilon)OPT$. We first observe, that since $\hat{P}$ is optimal under the cost $\hat{c}_e(\cdot)$.

$$\sum_{e\in\hat{P}} \hat{c}_e(\cdot) \quad \leq \quad \sum_{e\in P} \hat{c}_e(\cdot), \tag{1}$$

Due to assignment to $\hat{c}_e(\cdot)$ on line 8,

$$\begin{aligned} \sum_{e\in P} \hat{c}_e(\cdot) &= \sum_{e\in P} \left\lfloor \frac{c_e(\cdot)}{L\varepsilon/n} \right\rfloor \\ &\leq \sum_{e\in P} \frac{c_e(\cdot)}{L\varepsilon/n} = \frac{OPT}{L\varepsilon/n} \end{aligned} \tag{2}$$

and

$$\begin{aligned} \sum_{e\in\hat{P}} \hat{c}_e(\cdot) &= \sum_{e\in\hat{P}} \left\lfloor \frac{c_e(\cdot)}{L\varepsilon/n} \right\rfloor \\ &\geq \sum_{e\in\hat{P}} \left(\frac{c_e(\cdot)}{L\varepsilon/n} - 1 \right) \\ &\geq \sum_{e\in\hat{P}} \frac{c_e(\cdot)}{L\varepsilon/n} - n. \end{aligned} \tag{3}$$

Combining (1) and (2), we obtain

$$\sum_{e\in\hat{P}} \hat{c}_e(\cdot) \quad \leq \quad \frac{OPT}{L\varepsilon/n}, \tag{4}$$

and combining (3) and (4) we obtain,

$$\sum_{e\in\hat{P}} \frac{c_e(\cdot)}{L\varepsilon/n} - n \leq \frac{OPT}{L\varepsilon/n},$$

which implies that

$$\text{Cost of path } \hat{P} \text{ under } c_e(\cdot) \; = \sum_{e\in\hat{P}} c_e(\cdot) \leq OPT + L\varepsilon.$$

Since L is a lower bound for OPT, we conclude that the cost of path $\hat{P}$ under the cost function $c_e(\cdot)$ is at most $OPT + OPT\varepsilon = (1+\varepsilon)OPT$.

Running time is the summation of running time of $EXACT$ on line 9 and running time of $TEST$ inside the while loop. In terms of running time of $EXACT$, the for-loop of Figure 2 is executed $\sum_{e\in\hat{P}} \hat{c}_e(\cdot)$ times. From (4), $\sum_{e\in\hat{P}} \hat{c}_e(\cdot) \leq \frac{OPT}{L\varepsilon/n}$. Since $EXACT$ is executed when $OPT \leq U \leq 2L$, our analysis above implies,

$$\sum_{e\in\hat{P}} \hat{c}_e(\cdot) \leq \frac{OPT}{L\varepsilon/n} \leq 2n/\varepsilon.$$

Therefore, the running time of $EXACT$ is $O\left(\frac{n}{\varepsilon}m\log D + \frac{n^2}{\varepsilon^2}m\right)$. The procedure $TEST$ is executed $\log\log U$ times and each execution takes time $O\left(\frac{n}{\varepsilon}m\log D + \frac{n^2}{\varepsilon^2}m\right)$. The overall running time of $APPROX$ is $O\left(\left(\frac{n}{\varepsilon}m\log D + \frac{n^2}{\varepsilon^2}m\right)\log\log U\right)$. ■

Proof: (of Theorem 2) The proof follows by a case analysis of $X = \min\{D, \frac{\log C}{\varepsilon} + \log D, \frac{n}{\varepsilon} + \log D\}$. If $X = D$, we use Algorithm 1; if $X = \frac{n}{\varepsilon} + \log D$, we use Algorithm 3; otherwise we use Algorithm 2. In the first

$APPROX$

initialization

  set $L = 1$

  set $U = nC$

while $U > 2L$

  set $V = \sqrt{U \cdot V}$

  if $TEST(V)$ outputs $OPT \geq V$,   set $L = V$

  if $TEST(V)$ outputs $OPT \leq (1+\varepsilon)V$,   set $U = (1+\varepsilon)V$

set $\hat{c}_e(d) = \left\lfloor \frac{c_e(d)}{L\varepsilon/n} \right\rfloor$

call $EXACT$ with cost $\hat{c}_e(d)$

Fig. 4. An ε-approximation algorithm.

two cases, upper-bounding the running time is straightforward (Lemma 4 and 7). In case 3, $\frac{\log C}{\varepsilon} + \log D < \frac{n}{\varepsilon} + \log D$, implying $\log C < n$. Thus the running time of Algorithm 2 is $O(\frac{m}{\varepsilon} n \log D + \frac{\log C}{\varepsilon} \frac{mn}{\varepsilon} \log\log U)$, which is $O\left(\left(\log D + \frac{\log C}{\varepsilon}\right) \frac{mn}{\varepsilon} \log\log U\right)$. ■

Theorem 3 follows in the same manner by setting the s-t path to be the whole graph and thus letting $m = n = p$, where p is the number of links along the path.

IV. Heuristics for PARTITION

In the previous section, we gave three different ε-approximation algorithms for the PATH problem and PARTITION problem. In this section we describe several heuristics for PARTITION . Although these heuristics have no performance guarantees, they work well both in terms of running time and performance in our simulations.

Let P be the given s-t path with the end-to-end delay constraint delay D, and let p be the number of links on P.

Greedy Algorithm We begin with a greedy approach proposed by Lorenz and Orda [11], [10]. Figure 5 contains a description of the algorithm in [11] which carries the essence of this greedy approach. Variants of the algorithm with improved running times, $O(p \log p \log(D/p))$, are given in [10].

In Figure 5, one unit of delay is initially assigned to each link along path P and the total cost is high. In each iteration of the for-loop (line 3), one unit of delay is assigned to the link that causes the largest reduction in total cost.

The greedy algorithm returns an optimal solution if the cost function $c_e(d)$ on every link is convex. Recall that a curve is called *convex* (resp. *concave*) if its slope is nondecreasing (resp. nonincreasing). We remark that the greedy algorithm is highly sensitive to even the slightest concavity. For example, consider a path with two links e_1 and e_2 and a delay constraint $D = 12$. The cost functions for e_1 and e_2 shown in Figure 6(a) are convex. The function for e_1 shown in Figure 6(b) has a slight concavity, and the function for e_2 is unchanged. For (a), the greedy algorithm allocates eleven units of delay to link 1 and one unit of delay to link 2 for a total cost of 6; whereas for (b), the greedy algorithm allocates two unit to link 1 and ten units to link 2 for a total cost of 17. The optimal solutions to both cases is in fact the solution given in (a). It is easy to modify this example to make the greedy solution arbitrarily worse than the optimal solution.

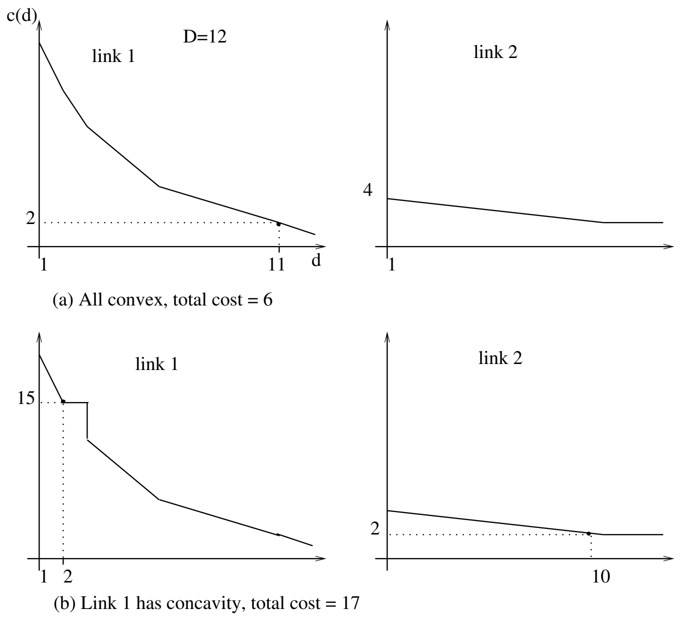


Fig. 6. Output of greedy algorithm on links with (a) convex, (b) non-convex costs.

In the context of networks with uncertain parameters [11], it is realistic to expect that the cost functions (logarithm of probability) are convex. However, for general cost functions one may expect nonconvexity. In our heuristics we make no convexity assumption.

```
GREEDY (P, D)
set delay = D
while delay > 0
    for all e ∈ P
        set g_e = reduction in cost for adding 1 unit delay to e
    pick e* such that g_e* = max_e(g_e)
    add 1 unit delay to e*
    set delay = delay − 1
```

Fig. 5. Greedy algorithm.

A. Heuristics

The greedy algorithm always makes the locally optimal choice. However, for non-convex cost functions one might have to make some locally non-optimal choices to reach the globally optimal solution. In this section, we propose two heuristics which try to rectify the "one step look-ahead" property of the greedy algorithm. As in the greedy algorithm, they both start out with one unit of delay assigned to each link (at a high cost) and proceed by adding delay to chosen links (thus reducing price) until all D units of delay are allocated. The first heuristic is a backtracking algorithm. It continues making greedy choices until it realizes that it has made a mistake, at which point, it tries to undo some of the recent choices. The second heuristic considers adding delay of various units and picks the one with the largest cost reduction. That is, instead of considering what is optimal for one step, it considers what is optimal for one step, two steps, four steps etc.

Greedy Heuristic with Rollback (Figure 7). Our first heuristic uses backtracking: it proceeds greedily by always adding one unit of delay to the link that offers the largest cost reduction. However, if the cost reduction (say g) due to the current delay allocation is greater than the previous reduction (meaning that a concave region is reached), our algorithm checks every link and performs a rollback where possible. The rollback consists of removing delay units from each link until a unit whose reduction was at least g is reached. In this way recent bad choices are removed to make room for future good choices. Note that if all cost functions are convex, no rollbacks are possible and our algorithm has same performance as the greedy algorithm.

Although the number of rollbacks can be large in the worst case, our simulations suggest that rollbacks happen infrequently in many scenarios.

Variable Step Size Heuristic (Figure 8). In this heuristic we allow different delay allocation sizes per iteration and pick the best one among them. The motivation is that if a portion of a curve offers large cost reduction, we make sure that we reach that portion.

For every iteration, we calculate the cost reduction for different delay allocation sizes for each of the p links. We consider two variants of this algorithm. In the first variant, the delay allocations are all possible powers of 2, and therefore, at most $\log D$ of them are computed. In the second variant, we consider all possible delay allocation between 1 and D. Figure 8 describes the first variant of the algorithm.

Let us concentrate on the running time of the first variant. To keep the presentation simple, we temporarily ignore ensuring that d^* is at most $delay$ (lines 13-15). *COSTDROP*(e) calculates $\log D$ cost reductions for each link e and stores the maximum of these in $S_e[logdelay]$. Its running time is $O(\log D)$. PH is a heap consisting of maximum possible reduction for each link e. Building this heap initially (lines 3-5) involves p invocations of *COSTDROP* and a build-heap procedure on p elements, which takes $O(p\log D + p) = O(p\log D)$ time. In each iteration of the while-loop (line 6), we perform delay allocation on the edge with largest possible cost reduction. This involves two heap operations and one *COSTDROP* . Therefore the running time per iteration is $O(\log p + \log D)$. If D' is the number of iterations of the while loop then the running time (ignoring lines 13-15) is $O(p\log D + D'(\log D + \log p))$.

In order to ensure that d^* is at most $delay$ (lines 7-10), we need to do some extra work every time $delay$ is dropped across a power of 2 (lines 13-15). In particular, some of the elements in PH may correspond to delays that are no longer valid (in other words, greater than the current value of $delay$). So we find the maximum valid cost reduction for each link and rebuild the heap PH with these new reduction values. Each execution of lines 13-15 takes time $O(P)$ and this is repeated at most $\log D$ times over the course of the algorithm. Thus the running time of the algorithm is $O(p\log D + D'(\log D + \log p))$.

Since $D' \leq D$, the worst case running time is

GREEDY WITH ROLLBACK(P,D)
set $delay = D$
while $delay > 0$
    for all $e \in P$
        set g_e= drop in cost for adding one unit of delay to e
    pick e^* such that $g_{e^*} = max_e(g_e)$
    add one unit of delay to e^*
    set $delay = delay - 1$
    for every $e_i \in P$, $e_i \neq e^*$
        Rollback($e^*, e_i, delay$)

ROLLBACK($e^*, e_i, delay$)
while drop in cost for last delay on $e_i <$ on e^*
    deallocate one unit of delay from e_i
    set $delay = delay + 1$

Fig. 7. Greedy algorithm with rollback.

VARYING STEPS(P, D)
set $delay = D$
set $logdelay = \lfloor \log(delay) \rfloor$
for all links $e \in P$
    COSTDROP (e)
set PH = heap consisting of $S_e[logdelay]$
while $delay > 0$
    set $g = deletemax(PH)$
    set e^*, d^* = link, step size for yield g
    add d^* delay units to e^*
    set $delay = delay - d^*$
    COSTDROP ($e*$)
    insert $S_{e^*}[logdelay]$ to PH
    if ($\lfloor \log(delay) \rfloor < logdelay$)
        set $logdelay = \lfloor \log(delay) \rfloor$
        set PH = heap consisting of $S_e[logdelay]$

COSTDROP (e)
/* Generates data structure for link e; $S_e[logdelay]$ is the largest yield for link e */
for $i = 0, 1, \ldots logdelay$
    set $g_{e,2^i}$ = drop in cost from adding d delay to e
for $i = 0, 1, \ldots logdelay$
    set $S_e[i] = \max_{0 \leq l \leq i}\{g_{e,2^l}\}$

Fig. 8. Greedy heuristic with variable step size.

$O(p \log D + D(\log D + \log p))$. However, this bound is pessimistic. If the curve contains concave regions then D' can be much smaller than D. For an extreme case in which the curves are purely concave, D' can even be constant. If the expected delay allocation per step is about $\frac{2D}{\log D}$, the average of all possible step sizes, then D' is expected to be $O(\log^2 D)$ and the running time is $O(\log D(p + \log p \log D + \log^2 D))$.

In the second variant of the heuristic, the running time becomes $O(D(p + D'))$. The worst-case running time is

$O(pD + D^2)$; the expected time is $O(pD + D \log D)$.

We remark that it is reasonable to consider other choices of step sizes. For example, step sizes may be delays corresponding to cost equal to $1, 2, 4, \ldots$. In this case, at most $\log C$ values are computed per edge, where C is the highest cost on any edge.

B. Simulation Results

The following table summarizes our simulation results. All experiments are run on a path of 30 links with $D = 250$. Each group 1-7 of experiments is the average of 100 runs of different cost functions. Each group of experiments aims to test cost functions with different shapes; the cost functions for the earlier groups tend to have more alternating convex/concave regions than the later groups. The actual cost functions are generated randomly with given expected sizes for the convex/concave regions. Each number in our table shows the percentage error for the corresponding heuristic [1]. Out of the greedy algorithms in [11], [10] we implement the simplest one for our simulation.

	Simple Greedy	Greedy w/ Rollback	Varying Step Variant I	Varying Step Variant II
1	24.3	16.4	5.0	0.1
2	27.1	14.5	3.9	0.4
3	25.7	14.3	5.0	0.0
4	20.8	12.2	3.5	0.0
5	22.3	9.6	4.0	0.0
6	17.6	9.1	3.1	0.0
7	20.4	10.2	2.2	0.0

As we can see, the relative performance of the heuristics remain largely unaffected by the shape of the curves. The heuristics with varying step sizes consistently perform the best. We observe the trade-off between running time and performance of the two variants of the varying step heuristics. Variant II has near-optimal performance, and its running time is better than the straight-forward dynamic programming approach as shown before. The greedy heuristic without rollback is unable to reach global optimality due to its local choices.

For our simulations on paths of different lengths, we observe similar results. We omit the data here.

As remarked earlier, all heuristics give optimal solutions for convex functions, since they degenerate to the greedy heuristic.

[1] The percentage error is calculated with respect to the difference between the initial cost and the optimal cost. This error measurement does not affect the relative performance of any two heuristics.

V. Conclusion

In this paper, we presented the first polynomial-time ε-approximations for the PATH and PARTITION problems with general cost functions. Whether or not their running times can be improved remains an interesting open question. We also gave heuristics for PARTITION . Applying our results to more complicated structures, such as multicast trees, is left for future research.

References

[1] Q. Ma, *Quality of Service Routing in Integrated Services Network*, Ph.D. thesis, Computer science Department, Carnegie Mellon University, 1998.

[2] R. Braden, D. Clark, and S. Shenker, "Integrated services in the internet architecture: an overview," RFC 1633.

[3] "Differentiated service for the internet," URL = `diff-serv.lcs.mit.edu`.

[4] G. Apostolopoulos, R. Guerin ad S. Kamat, A. Orda, T. Przygienda, and D. Williams, "QoS routing mechanisms and OSPF extensions," Experimental RFC.

[5] W. Lee, M. Hluchyi, and P. Humblet, "Routing subject to quality of service constraints in integrated communication networks," *IEEE Networks*, July 1995.

[6] R. Nagarajan, J. F. Kurose, and D. Towsley, "Allocation of local quality of service constraints to meet end-to-end requirements," in *IFIP Workshop on the Performnace Analysts of ATM Systems*, 1993.

[7] E. Crawley, R. Nair, B. Rajagopalan, and H. Sandick, "A framework for QoS-based routing in the internet," RFC 2386, Aug. 1998.

[8] G. Apostolopoulos, R. Guerin ad S. Kamat, and S. Tripathi, "Quality of service routing: a performance perspective," in *ACM Sigcomm*, 1998.

[9] Z. Wang and J. Crowcroft, "Quality of service routing for supporting multimedia applications," *IEEE Journal selected areas in communications*, vol. 14, no. 7, pp. 1228–1234, 1996.

[10] D. Lorenz and A. Orda, "Optimal partition of QoS requirements on unicast paths and multicast trees," in *IEEE Infocom*, 1999.

[11] D. Lorenz and A. Orda, "QoS routing in networks with uncertain parameters," *IEEE/ACM Transactions on Networking*, , no. 6, 1998, To appear.

[12] R. Guerin and A. Orda, "QoS based routing in networks with inaccurate information: Theory and algorithms," in *IEEE Infocom*, 1997.

[13] A. Orda, "Routing with end to end QoS guarantees in broadband networks," in *IEEE Infocom*, 1998.

[14] S. Chen and K. Nahrstedt, "Distributed QoS routing with imprecise state information," Tech. Rep., Computer Science Department, University of Illinois, 1998.

[15] M. R. Garey and D. S. Johnson, *Computers and Intractability: a Guide to the Theory of NP-Completeness*, W. H. Freeman, 1979.

[16] R. Hassin, "Approximation schemes for the restricted shortest path problem," *Mathematics of Operations Research*, vol. 17, no. 1, pp. 36 – 42, February 1992.

[17] C. Phillips, "The network inhibition problem," in *ACM Symposium on the theory of computing (STOC)*, May 1993.

[18] S. Chen and K. Nahrstedt, "On finding multi-constrained paths," in *IEEE International Conference on Communication*, June 1998.

[19] V. Firoiu and D. Towsley, "Call admission and resource reservation for multicast sessions," in *IEEE Infocom*, 1996.