ALGORITHMS FOR ROUTING WITH MULTIPLE CONSTRAINTS

by

Anuj Puri and Stavros Tripakis

Memorandum No. UCB/ERL M01/7

30 January 2001

ALGORITHMS FOR ROUTING WITH MULTIPLE CONSTRAINTS

by

Anuj Puri and Stavros Tripakis

Memorandum No. UCB/ERL M01/7

30 January 2001

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Algorithms for Routing with Multiple Constraints

Anuj Puri and Stavros Tripakis
Department of Electrical Engineering and Computer Science,
University of California, Berkeley, CA 94720

Abstract— **In this paper, we study the problem of routing under multiple constraints. We consider a graph where each edge is labeled with a cost and a delay. We then consider the problem of finding a path from a source vertex to a destination vertex such that the sum of the costs on the path satisfy the cost constraint and the sum of the delays satisfy the delay constraint. We study the complexity of this problem and then present three different algorithms for solving the problem. These algorithms have varying levels of complexity and solve the problem with varying degrees of accuracy. We present an implementation of these algorithms and discuss their performance on different graphs.**

I. Introduction

Routing data from one node to another is among the most basic problems in computer networking. A model for such a problem is a graph where an edge in the graph represents a physical link in the network. Associated with each edge is its length. This length could represent the delay of the physical link, or its cost, or a summary of various properties of the link. The routing problem is to find a path from the source to the destination of minimum length.

This formulation requires us to summarize all properties of a link with a single number, or to focus only on a single property at the expense of others. For example, we may have a choice between a low cost link which has a high delay (such as a satellite link), or a high cost link with a low delay (such as a fiber optic link). It is not clear how to summarize these links with a single number in the routing problem.

In this paper, we present a more general formulation of the problem. We model a physical link with an edge in the graph that is labeled with two numbers: a delay and a cost. We are given a delay constraint D and a cost constraint C. Our objective is to find a path from the source to the destination such that the sum of all delays on the path is less than D, and the sum of all costs is less than C. We first show that this problem is NP-Complete. We then present three different algorithms for solving the problem. The first algorithm is a pseudo-polynomial time algorithm which solves the problem exactly in time $O(|V||E|min\{C, D\})$ where $|V|$ is the number of vertices and $|E|$ is the number of edges in the graph. The algorithm either reports back with a path satisfying the constraints or states that no such path exists. The second algorithm solves the problem approximately but with an error of at most ϵ. That is, either it states that no path satisfying the constraints exists, or it finds a path such that the sum of costs on the path is at most $C \cdot (1+\epsilon)$, and the sum of delays is at most $D \cdot (1+\epsilon)$. The complexity of this algorithm is $O(|V|^2|E|(1+\frac{1}{\epsilon}))$. The third algorithm finds a path with an error of at most $\epsilon = 1$. This algorithm requires a solution of the shortest path problem on the given graph. Although most of the paper is focused on dealing with two constraints, the first two algorithms generalize in a straightforward manner to more than two constraints.

In Section II, we define our problem more formally and introduce our notation. In Section III, we show the problem is NP-Complete. In Section IV, we present a pseudo-polynomial time algorithm for the problem. In Section V, we present a polynomial time approximation scheme. In Section VI, we present an algorithm based on solving the shortest path problem on the graph. In Section VII, we present a linear programming solution to a relaxed problem. In Section VIII we discuss other possible extensions. Section IX discusses the performance results for the different algorithms and Section X is the conclusion.

Relationship to other work

The routing problem with more than one constraint seems to have been studied by several re-

searchers. It seems to be well known that the problem is NP-Complete [3] [2] [1]. An explicit proof of this is provided in [5]. In [3], a pseudo-polynomial time algorithm is presented for exactly solving the problem. This algorithm is similar to the algorithm presented in Section IV, however, [3] states that the complexity of the algorithm is $O(|V|^5 max\{C,D\}log(|V|max\{C,D\}))$. By a more careful analysis and using the data structures in a more clever manner, we can show the complexity of our algorithm is $O(|V||E|min\{C,D\})$. In [3], an approximation algorithm that solves the problem with approximation error $\epsilon = 1$ using the shortest path algorithm is also presented. Although this is similar to our algorithm in Section VI, our algorithm in general will perform better because we solve a series of shortest path problems, each obtaining a better solution than the last one. In [2], several algorithms are presented for approximating the solution to the problem. Although the author restricts to acyclic graphs, extensions to general graphs should be straightforward. The complexity of the two approximation algorithms are $O(loglogB(\frac{|V||E|}{\epsilon} + loglogB))$ and $O(|E|\frac{|V|^2}{\epsilon}log(\frac{n}{\epsilon}))$ where ϵ is the error of the approximation and $B = max\{C,D\}$. Our approximation algorithm in Section V has complexity $O(|V|^2|E|(1+\frac{1}{\epsilon}))$. The algorithms also use somewhat different techniques. Our algorithm is essentially a generalization of the Bellman-Ford algorithm where we keep track of errors during the iteration.

Although several algorithms have been proposed for solving the routing problem with multiple constraints, there seems to be no results available about the actual implementation or the performance of these algorithms. In our work, we present several new ideas for solving the problem, a complete implementation of these ideas and a comparison of the performance of these algorithms.

II. Problem Formulation

We consider a directed *2-weight* graph $G = (V,E)$, where V is the set of vertices and E is the set of edges. An edge $e \in E$ is $e = (v,w,c,d)$ where the edge goes from v to w, and has delay $\text{delay}(e) = d$ and $\text{cost}(e) = c$. We write this as $v \xrightarrow{(c,d)} w$. When there is no confusion, we may also write the edge as (v,w) and say the edge is labeled with (c,d).

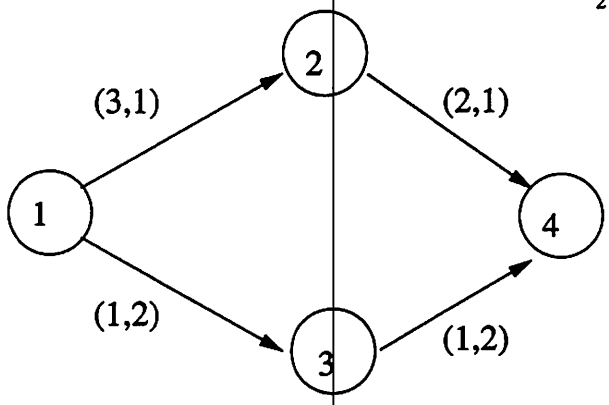

Fig. 1. A Simple Network

A path is $p = v_1 \xrightarrow{(c_1,d_1)} v_2 \xrightarrow{(c_2,d_2)} v_3 \xrightarrow{(c_3,d_3)} \ldots \xrightarrow{(c_n,d_n)} v_{n+1}$. The cost of a path is $\text{cost}(p) = \sum_{i=1}^{n} c_i$ and its delay is $\text{delay}(p) = \sum_{i=1}^{n} d_i$.

Given a path p and cost constraint $C \geq 1$ and delay constraint $D \geq 1$, we say p is *feasible* provided $\text{cost}(p) \leq C$ and $\text{delay}(p) \leq D$. The problem of *routing under two constraints* is, given $G = (V,E)$, cost constraint C and delay constraint D, a *source* node $s \in V$ and a *destination* node $t \in V$, find a feasible path p from s to t, or decide that no such path exists.

Example II.1: Consider the 2-weight graph of Figure 1. Each edge is labeled with (c,d) where c is the cost of the edge and d is the delay of the edge. For example, the edge from vertex 1 to vertex 2 has cost 3 and delay 1. Suppose the source vertex is 1, the destination vertex is 4, the cost constraint is $C = 5$ and the delay constraint is $D = 2$. Then, the path $1 \xrightarrow{(3,1)} 2 \xrightarrow{(2,1)} 4$ is feasible, whereas $1 \xrightarrow{(1,2)} 3 \xrightarrow{(1,2)} 4$ is not (since it violates the delay constraint).

The reader can check that if $C = 4$ and $D = 3$, then there is no feasible path.

Rather than checking to see if a graph has a feasible path, it is sometimes useful to try to minimize the following objective function

$$M(p) = max\{\frac{max\{\text{cost}(p),C\}}{C}, \frac{max\{\text{delay}(p),D\}}{D}\}.$$

Observe that for any path p, $M(p) \geq 1$ and $M(p) = 1$ iff p is feasible. But even if a feasible path does not exist or is hard to find, by trying to minimizing $M(p)$ we can get a path that comes "close" to satisfying the constraints.

Formally, we define the error of a path p as

$$\text{error}(p) = \frac{M(p) - M(p^*)}{M(p^*)}$$

where p^* is the path which minimizes M (in case more than one paths minimize M, we pick p^* arbitrarily among them, since the minimal value $M(p^*)$ is the same for all of them).

Notice that $\text{error}(p) \geq 0$ and $\text{error}(p) = 0$ iff p is feasible. Also note that if $\text{cost}(p) \leq C \cdot (1+\epsilon)$ and $\text{delay}(p) \leq D \cdot (1+\epsilon)$ then $\text{error}(p) \leq \epsilon$. Indeed, the two above conditions imply that $M(p) \leq 1+\epsilon$ and, since $M(p^*) \geq 1$, we get $\text{error}(p) \leq \epsilon$.

In case it is too difficult to find p^*, we look for a path p for which $\text{error}(p)$ is small. We will next consider the complexity of finding a feasible path, and algorithms for finding a feasible path and for finding paths for which $\text{error}(p)$ is small.

III. Complexity

We show that the routing problem with two constraints is NP-Complete.

Theorem III.1: The routing problem with two constraints is NP-Complete.

Proof: We will provide a reduction from the knapsack problem. Recall that in the knapsack problem, we are given positive integers $c_1, c_2, \ldots, c_n$, and N, and the objective is to find a subset $S \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in S} c_i = N$.

From the knapsack problem, we construct a graph with vertices $\{1, \ldots, n\}$. There are two edges from vertex i to vertex $i+1$: edge $(i, i+1, c_i, 0)$ and edge $(i, i+1, 0, c_i)$. Figure 2 shows the scenario. Our objective is to find a path from vertex 1 to vertex n with cost constraint N and delay constraint $\sum_{i=1}^{n} c_i - N$. It is easy to check that there is a path that satisfies the constraints iff there is a solution to the knapsack problem. ∎

IV. An $O(|V| \cdot |E| \cdot min\{C, D\})$ pseudo-polynomial algorithm

In this section, we propose an algorithm for the problem of routing under two constraints with worst-case complexity $O(|V| \cdot |E| \cdot min\{C, D\})$. That is, the algorithm is polynomial on the size of the graph (quadratic on the number of vertices and linear on the number of edges), but also linearly depends on the smaller of the bounds C and D. Therefore, it is a *pseudo*-polynomial algorithm.

Let us begin by making a safe hypothesis. Given a 2-weight graph $G = (V, E)$, where $|V| = n$, let $\text{cost}_{max} = max\{c \mid (\_, \_, c, \_) \in E\}$ and $\text{delay}_{max} = max\{d \mid (\_, \_, d, \_) \in E\}$ be the maximum cost and delay associated with any edge of G. Now, assume that $n \cdot \text{cost}_{max} \leq C$. Then, given $u, v \in V$, there exists a feasible path from u to v iff there exists a path p from u to v such that $\text{delay}(p) \leq D$. To see this, observe that if there is a path p from u to v such that $\text{delay}(p) \leq D$, then there is a *simple* path (i.e., with no cycles) from u to v. Assuming p to be simple, p has length at most n, thus, $\text{cost}(p) \leq n \cdot \text{cost}_{max} \leq C$, which implies that p is feasible. The inverse direction is trivial.

Given this observation, finding a feasible path in G from u to v comes down to finding the smallest-delay path from u to v, that is, the path p that minimizes $\text{delay}(p)$. This can be easily done using a shortest-path algorithm, with cost $O(|V| \cdot |E|)$. Since this is less than $O(|V| \cdot |E| \cdot min\{C, D\})$, this case is not interesting. The case where $n \cdot \text{delay}_{max} \leq D$ is symmetric.

So, from now on we assume that $n \cdot \text{cost}_{max} > C$ and $n \cdot \text{delay}_{max} > D$. We also assume that the greatest common divisor of $\{C, \text{cost}(e) \mid e \in E\}$ is 1, and similarly for the delays (otherwise we could just divide all costs/delays by their greatest common divisor, without affecting the problem).

Informally, the algorithm works as follows. For each vertex w, we compute a set of *cost-delay pairs* F_w. Each $(c, d) \in F_w$ will represent the cost and delay of a possible path from w to the destination vertex v. To keep the size of F_w manageable, we eliminate from F_w all elements corresponding to infeasible paths (i.e., all (c, d) such that $c > C$ or $d > D$). Moreover, we eliminate from F_w all redundant elements, that is, all elements with both cost and delay greater from some other element. Let us make these more precise below.

A. Cost-delay sets

A *cost-delay set* for a vertex w is a set $F_w \subseteq \mathbb{N} \times \mathbb{N}$. An element (c, d) of F_w is called *infeasible* if either $c > C$ or $d > D$. An element (c, d) of F_w is called *redundant* if there exists a different $(c', d') \in F_w$ such that $c' \leq c$ and $d' \leq d$.

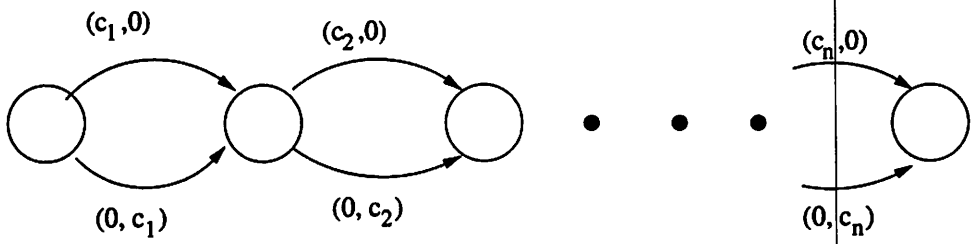

Fig. 2. Graph obtained from the knapsack problem

A cost-delay set F is said to be *minimal* if it contains no infeasible or redundant elements. The following properties hold (assuming C and D fixed):

Proposition IV.1: If F is minimal, then $|F| \leq min\{C, D\}$. To every cost-delay set F corresponds a unique greatest minimal subset $F' \subseteq F$.
We write minimal(F) to denote the greatest minimal subset of F.

Figure 3 displays the typical structure of a cost-delay set and its minimal. Black and grey bullets are infeasible and redundant elements, respectively.

Minimal cost-delay sets admit an efficient canonical representation as sorted lists. Consider a minimal set $F = \{(c_1, d_1), (c_2, d_2), ..., (c_n, d_n)\}$ and assume, without loss of generality, that $c_1 \leq c_2 \leq \cdots \leq c_n$. Then, $d_1 \geq d_2 \geq \cdots \geq d_n$ must hold, otherwise there would be at least one redundant element in F. Consequently, F can be represented as the list $(c_1, d_1)\ (c_2, d_2)\ \cdots\ (c_n, d_n)$, sorted using cost as the "key". This representation is canonical in the sense that two minimal sets F_1, F_2 are equal iff their list representations are identical.

The algorithm works with minimal cost delay sets and uses two operations, namely, union and translation with respect to a vector $(c, d) \in \mathbb{N}^2$. We present these operations and discuss how they can be implemented using sorted-lists and preserving the canonical representation.

Given minimal (i.e., feasible and non-redundant) F_1, F_2, the union $F_1 \cup F_2$ is always feasible, but not necessarily non-redundant. In order to compute $F = \text{minimal}(F_1 \cup F_2)$ directly from the list representations L_1, L_2 of F_1, F_2, we can use a simple modification of a usual merge-sort algorithm on lists. The latter takes as input L_1, L_2 and produces L, the list representation of F. In order to guarantee the absence of redundant points in L, it compares at each step the heads (c_1, d_1) and (c_2, d_2) of (the remaining parts of) L_1, L_2. If $c_1 \leq c_2$ and $d_1 \leq d_2$ then (c_2, d_2) is redundant and is skipped. If $c_2 \leq c_1$ and $d_2 \leq d_1$ then (c_1, d_1) is skipped. Otherwise, the pair with the smallest c_i is inserted in L and the head pointer move one element ahead in the corresponding list L_i. It is easy to see that this algorithm is correct. The cost of the algorithm is $n_1 + n_2$, where n_i is the length of L_i. Therefore, from proposition IV.1, the worst-case complexity of computing the union of cost-delay sets is $O(min\{C, D\})$.

Translation is defined on a cost-delay set F and a pair $(c, d) \in \mathbb{N}^2$:

$$F + (c, d) \stackrel{\text{def}}{=} \{(c' + c, d' + d) | (c', d') \in F\}$$

If F is minimal, then $F + (c, d)$ is non-redundant, however, it may contain infeasible points. These can be easily eliminated, however, while building the list L' for $min(F + (c, d))$: the list of F is traversed, adding (c, d) to each of its elements, (c_i, d_i); if $c_i + c \leq D$ and $d_i + d \leq D$ then $(c_i + c, d_i + d)$ is inserted at the end of L', otherwise it is infeasible and it is skipped. At the end, L' will be sorted by cost. The complexity of translation is $O(min\{C, D\})$.

B. The algorithm

The algorithm iteratively computes the (minimal) cost-delay sets of all vertices in the graph. Let F_w^j denote the cost-delay set for vertex w at iteration j. Initially, all vertices have empty cost-delay sets, $F_w^0 = \emptyset$, except v, for which $F_v^0 = \{(0, 0)\}$. At each iteration, each vertex updates its cost-delay set with respect to all its successor vertices. Computation stops when no cost-delay set is updated any more. We now present the operations performed at each iteration at each vertex w.

Let $w_1, ..., w_k$ be the successor vertices of w, that is, $w \stackrel{(c_i, d_i)}{\longrightarrow} w_i$, for $i = 1, ..., k$ (note that $w_1, ..., w_k$ might not be distinct). Then, the cost-delay set of w

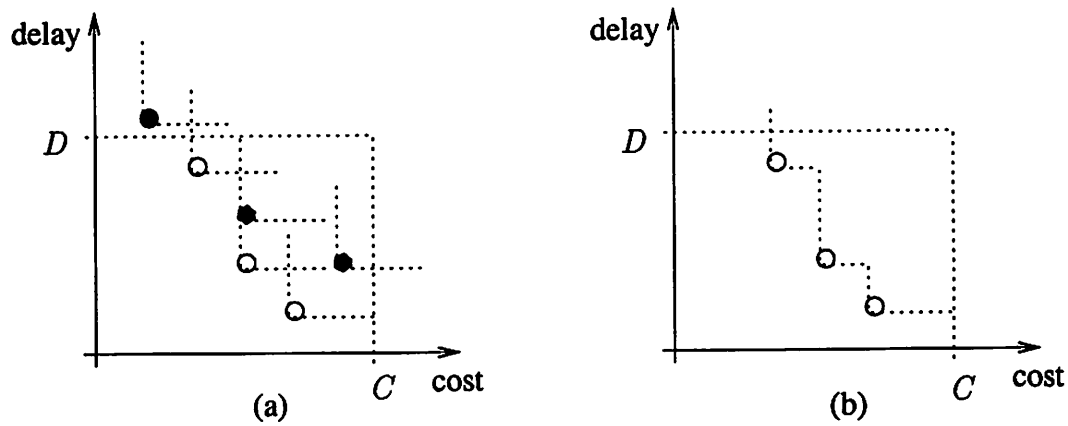

Fig. 3. A cost-delay set (a) and its minimal (b)

at iteration $j+1$ will be:

$$F_w^{j+1} = \text{minimal}\Big(F_w^j \cup \bigcup_{i=1}^{k} (F_{w_i}^j + (c_i, d_i))\Big) \quad (1)$$

That is, we add to the possible cost-delay values for w all values obtained by taking an edge to some successor vertex w_i, and then continuing with a possible cost-delay value for w_i.

The following proposition proves termination and correctness of the algorithm.

Proposition IV.2: (Termination) The updating of the cost-delay sets will stabilize after at most $|V|$ iterations, that is, for any vertex w, $F_w^{|V|+1} = F_w^{|V|}$. (Correctness) A feasible path from w to v exists iff $F_w^{|V|} \neq \emptyset$. For any $(c,d) \in F_w^{|V|}$, there exists a path p from w to v such that $\text{cost}(p) = c$ and $\text{delay}(p) = d$.

C. Worst-case complexity of the algorithm

Proposition IV.2 implies that the algorithm stops after at most $|V|$ iterations. At each iteration, the cost-delay set of each vertex is updated with respect to all its successor vertices. Thus, there are at most $|E|$ updates at each iteration. Each update involves a translation and a union, both of which have complexity $O(min\{C, D\})$. Therefore, the overall worst-case complexity of the algorithm is $O(|V| \cdot |E| \cdot min\{C, D\})$.

D. Incorporating routing information

As defined, cost-delay sets do not contain any routing information, that is, at the end of the algorithm, we know that a point in F_w represents the cost-delay value of a possible feasible path from w to v, but we do not know which path. This information is easy to incorporate, at the expense of associating to each $(c,d) \in F_w$, the edge $e = (w, w_1, c', d')$, and a pointer to the element $(c_1, d_1) \in F_{w_1}$, from which (c,d) was generated. The edge and (c_1, d_1) element are unique, and come from the operation $F_w \cup (F_{w_1} + (c', d'))$. In order to reconstruct the path from w with cost-delay (c,d) we follow the edge e to w_1, then look for the path from w_1 with cost-delay (c_1, d_1), and so on.

V. A BOUNDED-ERROR APPROXIMATIVE ALGORITHM

In this section we give an approximative algorithm for the problem of routing under two constraints. The algorithm is approximative in the sense that, it might not yield a feasible path, even if such a path exists. However, the error in the path p returned by the algorithm can be bounded: $\text{error}(p) \leq \epsilon$, where ϵ is an input parameter. The algorithm has worst-case complexity $O(|V|^2 \cdot |E| \cdot (1 + \frac{1}{\epsilon}))$, which implies that it is worth using only when $|V|$ is (much) smaller than $\frac{\epsilon}{1+\epsilon} \cdot min\{C, D\}$. Otherwise, the algorithm of section IV, being exact and less expensive, would be preferable. In the rest of this section we assume that $|V| < \frac{\epsilon}{1+\epsilon} \cdot min\{C, D\}$.

A. Minimal-distance cost-delay sets

The approximative algorithm is similar to the one of section IV, with the additional fact that it eliminates elements of cost-delay sets which are "too close" to some other element. More formally, for

$(c_1, d_1), (c_2, d_2) \in \mathsf{N}^2$, define:

$$||(c_1, d_1), (c_2, d_2)|| \stackrel{\text{def}}{=} max\{|c_1 - c_2|, |d_1 - d_2|\}$$

Then, a cost-delay set F is said to have *minimal distance* δ iff for all distinct $(c_1, d_1), (c_2, d_2) \in F$, $||(c_1, d_1), (c_2, d_2)|| \geq \delta$.

Given a cost-delay set F and some $\delta \geq 2$, we would like to find a subset $F' \subseteq F$, such that:

1. F' has minimal distance δ, and
2. for all $x \in F - F'$, there exists $y \in F'$ such that $||x, y|| < \delta$.

Condition 2 ensures that no elements of F are dropped unnecessarily (were condition 2 to be missed, the trivial subset $F' = \emptyset$ would satisfy condition 1). A subset $F' \subseteq F$ satisfying the above conditions is called a *maximal δ-distance subset of F*. In general, there may be more than one maximal δ-distance subsets of a given F (any one of them is good for our purposes). We now give a procedure to compute, given F, a maximal δ-distance subset $F' \subseteq F$.

The procedure takes as input the list representation L of F and generates as output a list L'. Assume $L = (x_1, ..., x_n)$. Initially, $L' = (x_1)$. Let y denote the last element of L', at each point during the execution of the procedure. For each $i \geq 2$, if $||x_i, y|| \geq \delta$ then x_i is appended at the end of L' and y is updated to x_i, otherwise, x_i is skipped. It can be shown that the list built that way represents a legal δ-distance subset of F. From now on, we denote this set by $\mathsf{min\_dist}(\delta, F)$.

Definition V.1: We define the *step error*, δ_ϵ, to be

$$\frac{min\{C, D\} \cdot \epsilon}{|V|}.$$

B. The algorithm

The approximative algorithm is obtained by the algorithm of section IV by the following modification. Given $\epsilon \in [0, 1]$, instead of keeping a minimal set F_w for each node w, we keep a set B_w such that:

1. B_w has no redundant elements,
2. for each $(c, d) \in B_w$, $c \leq (1 + \epsilon) \cdot C$, $d \leq (1 + \epsilon) \cdot D$ (that is, the feasibility region is extended by $(\epsilon \cdot C, \epsilon \cdot D)$,
3. B_w has minimal distance δ_ϵ.

That is, in the approximative algorithm, the fix-point equations are as follows:

$$B_w^{j+1} = \mathsf{min\_dist}\Big(\delta_\epsilon, \mathsf{minimal}\Big(B_w^j \cup \bigcup_{i=1}^{n} (B_{w_i}^j + (c_i, d_i))\Big)\Big)$$

As in the case of the exact algorithm, termination of the approximative algorithm is ensured in $|V|$ steps.

Proposition V.1: Consider a graph G, nodes u, v of G, and cost-delay constraints C, D. Then, for given ϵ:

(1) If $B_u = \emptyset$ at the end of the approximative algorithm, then no feasible path from u to v exists.

(2) If $B_u \neq \emptyset$, then for each $(c, d) \in B_u$, there exists a path p from u to v such that $\mathsf{cost}(p) = c$, $\mathsf{delay}(p) = d$ and $\mathsf{error}(p) \leq \epsilon$.

Proof (sketch):

Let w be a node and F_w, B_w be the final cost-delay sets computed for w by the exact and approximative algorithms, respectively. The result is based on the fact that, for any $(c, d) \in F_w$, there exists $(c', d') \in B_w$, such that $||(c, d), (c', d')|| \leq |V| \cdot \delta_\epsilon$. This is because at most δ_ϵ "error" accumulates at each step of the algorithm, when eliminating pairs during the min_dist operation.

By definition of δ_ϵ, we have that $||(c, d), (c', d')|| \leq min\{C, D\} \cdot \epsilon$. Then, assuming (c, d) to be the cost and delay of an optimal path p^* and (c', d') the cost and delay of a path p computed by the approximative algorithm, it is easy to prove that $\mathsf{error}(p) \leq \epsilon$. For (1), notice that if p^* is feasible then $c' \leq (1 + \epsilon) \cdot C$ and $d' \leq (1 + \epsilon) \cdot D$, This means that (c', d') is indeed "inside" the extended feasibility region, thus, is not eliminated from B_w during the approximative algorithm. ∎

C. Worst-case complexity

The only difference from the algorithm of section IV is in the worst-case size of the cost-delay sets B_w. Since the latter have minimal distance δ_ϵ and are bounded by the feasibility region $((1+\epsilon)\cdot C, (1+\epsilon)\cdot D)$, we have $|B_w| \leq \frac{(1+\epsilon)\cdot min\{C,D\}}{\delta_\epsilon}$. By definition of δ_ϵ, we get $|B_w| \leq \frac{(1+\epsilon)}{\epsilon}|V|$. The union, translation and min_dist operations can be implemented using sorted lists to represent the sets B_w (the canonical representation is not affected by minimal distance). The cost of the operations is, as previously, linear on

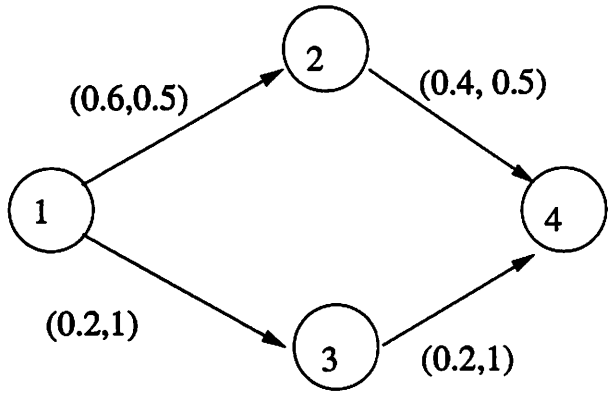

Fig. 4. A network with normalized costs and delays

the size of the lists, which yields an overall worst-case complexity of $O(|V|^2 \cdot |E| \cdot (1 + \frac{1}{\epsilon}))$.

VI. Satisfying constraints by using the shortest path algorithm

In this section, we consider an algorithm for finding a path which satisfies the two constraints by using the shortest path algorithm. Our objective will be to use the shortest path algorithm to find a path p which minimizes $M(p)$.

For the rest of this section we assume that we have normalized the costs and delays by dividing the costs by C and the delays by D. Figure 4 shows the figure from Example II.1 where the cost constraint is 5 and the delay constraint is 2.

A path is feasible in the new graph if $\text{cost}(p) \leq 1$ and $\text{delay}(p) \leq 1$. Note that a path is feasible in the new graph iff it was feasible in the original graph. Furthermore, $M(p)$ is the same in both graphs.

To find a path satisfying two constraints by using the shortest path algorithm, we choose an $0 \leq \alpha \leq 1$ and replace the cost c and the delay d associated with an edge with the weight $\alpha c + (1 - \alpha)d$. We then use the shortest path algorithm to find a path with the smallest weight. We refer to this path as $SP(G, \alpha)$. As the next lemma shows, $p = SP(G, \alpha)$ has an error $\text{error}(p)$ of at most 1 for $\alpha = \frac{1}{2}$.

Lemma VI.1: For a graph $G = (V, E)$, $M(p^*) \leq M(p) \leq 2M(p^*)$, where $p = SP(G, \alpha)$ and p^* is the path which minimizes M.

Proof: Recall that for all paths p', $M(p') \geq 1$. If $M(p) = 1$, then clearly $M(p) \leq 2M(p^*)$. So assume $M(p) > 1$. Then

$$\begin{aligned} M(p) &\leq \text{cost}(p) + \text{delay}(p) \\ &\leq \text{cost}(p^*) + \text{delay}(p^*) \\ &\leq 1 + 1 \leq M(p^*) + M(p^*) = 2M(p^*). \end{aligned}$$

■

The previous lemma shows that by choosing $\alpha = \frac{1}{2}$, we can obtain a path p with $\text{error}(p) \leq 1$. We now present an algorithm that minimizes $M(SP(G, \alpha))$ by choosing the appropriate α.

The algorithm uses binary search: assume we know that the optimal value of α lies in the interval $[l, u]$; we find $p = SP(G, \alpha)$ for $\alpha = \frac{l+u}{2}$; if $\text{cost}(p) \leq \text{delay}(p)$, we eliminate the interval $(\frac{l+u}{2}, u]$ from consideration, otherwise, we eliminate $[l, \frac{l+u}{2})$. The algorithm terminates when $SP(G, l) = SP(G, u)$.

The reason that half of the interval can be eliminated follows from the following lemma.

Lemma VI.2: Suppose $p = SP(G, \alpha)$ and $\text{cost}(p) \leq \text{delay}(p)$. Then for $\alpha' \geq \alpha$ and $p' = SP(G, \alpha')$, $\text{cost}(p') \leq \text{cost}(p)$ and $\text{delay}(p') \geq \text{delay}(p)$.

Proof: There are four cases:

1. $\text{cost}(p') > \text{cost}(p)$ and $\text{delay}(p') > \text{delay}(p)$.
2. $\text{cost}(p') < \text{cost}(p)$ and $\text{delay}(p') < \text{delay}(p)$.
3. $\text{cost}(p') > \text{cost}(p)$ and $\text{delay}(p') < \text{delay}(p)$.
4. $\text{cost}(p') \leq \text{cost}(p)$ and $\text{delay}(p') \geq \text{delay}(p)$.

Case 1 is not feasible because then path p improves on $p' = SP(G, \alpha')$. Case 2 is not feasible because then path p' improves on $p = SP(G, \alpha)$. Case 3 is not feasible because $\alpha\text{cost}(p') + (1 - \alpha)\text{delay}(p') \geq \alpha\text{cost}(p) + (1 - \alpha)\text{delay}(p)$ and $(\alpha' - \alpha)\text{cost}(p') + (\alpha - \alpha')\text{delay}(p') > (\alpha' - \alpha)\text{cost}(p) + (\alpha - \alpha')\text{delay}(p)$, and hence $\alpha'\text{cost}(p') + (1 - \alpha')\text{delay}(p') > \alpha'\text{cost}(p) + (1 - \alpha')\text{delay}(p)$ — a contradiction since $p' = SP(G, \alpha')$. Therefore, 4 is the only feasible case. ■

Now assume we found $p = SP(G, \alpha)$ and $\text{delay}(p) > 1$ and $\text{cost}(p) \leq \text{delay}(p)$. Then from Lemma VI.2, for $\alpha' \geq \alpha$, for $p' = SP(G, \alpha')$, $\text{cost}(p') \leq \text{cost}(p)$ and $\text{delay}(p') \geq \text{delay}(p)$. Therefore $M(p') \geq M(p)$, and hence the interval $(\alpha, u]$ can be eliminated from consideration. By similar reasoning, if $\text{delay}(p) \leq \text{cost}(p)$, then the interval $[l, \alpha)$ can be eliminated.

Here is a more formal statement of the algorithm:

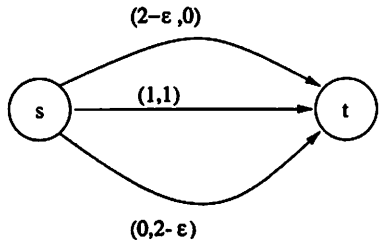

Fig. 5. A graph for which the error is error$(SP(G,\alpha)) = 1-\epsilon$ for all $0 \le \alpha \le 1$

Algorithm to find α to minimize $M(SP(G,\alpha))$:

$l = 0$
$u = 1$
$p_l = SP(G,l)$
$p_u = SP(G,u)$
Repeat
  $\alpha = \frac{l+u}{2}$
  $p_\alpha = SP(G,\alpha)$
  if (cost$(p) \le$ delay(p))
    $l = \alpha$
    $p_l = p_\alpha$
  else
    $u = \alpha$
    $p_u = p_\alpha$
Until $(p_l = p_u)$

Theorem VI.1: The above algorithm terminates in polynomial number of steps, and the α^* computed by the algorithm satisfies $M(SP(G,\alpha^*)) \le M(SP(G,\alpha))$ for $\alpha \in [0,1]$.

Notice that, although error$(SP(G,\alpha^*)) \le 1$ (lemma VI.1), there are "bad" examples where the error can be arbitrarily close to 1.

Example VI.1: Consider the example in Figure 5 where the cost constraint is $C = 1$ and the delay constraint is $D = 1$. It is easy to check that for any $0 \le \alpha \le 1$, error$(SP(G,\alpha)) = 1-\epsilon$.

VII. A Linear Programming solution to a relaxed problem

In this section, we relax the requirements of our problem. Rather than asking for a single path that satisfies the cost and delay requirements, we allow for the data to be routed over multiple paths. But we require the average delay and average cost to satisfy the constraints.

Let us define f_e to be fraction of the data from the source to the destination that flows over the edge e. We then have the following constraints ($out(v)$ are the outgoing edges and $in(v)$ are the incoming edges of a node v):

$$\text{For each } e \in E, f_e \ge 0$$

$$\sum_{e \in out(s)} f_e = 1 \quad (2)$$

$$\sum_{e \in in(t)} f_e = 1 \quad (3)$$

$$\text{For } v \ne s \text{ and } v \ne t, \sum_{e \in in(v)} f_e = \sum_{e \in out(v)} f_e \quad (4)$$

$$\sum_{e \in E} f_e \text{ cost}(e) \le C \quad (5)$$

$$\sum_{e \in E} f_e \text{ delay}(e) \le D \quad (6)$$

Equation 2- 4 are the balance equations for the nodes. Equation 5 states that the average cost must be less than the cost constraint C, and equation 6 states that the average delay must be less than the delay constraint D.

A feasible solution of the above linear program tells us how the data should be routed from the source to the destination so that average cost and delay constraints are satisfied.

Example VII.1: Consider again Example II.1 and Figure 1 with cost constraint 4 and delay constraint 3. If we formulate the above set of linear constraints for this problem, we note that $f_e = \frac{1}{2}$ for each edge e is a solution. This means that half of the data from the source is routed to node 2, and the other half to node 3. The average delay corresponding to this solution is $\frac{1}{2}(2+2+1+1) = 3$, and the average cost is $\frac{1}{2}(3+2+1+1) = 3.5$. Notice that even though the average cost and delay satisfy the constraints, individual paths may not (e.g., the path $(1,3)(3,4)$ does not satisfy the delay constraint).

If we restrict ourselves to integer solutions of the above linear program (i.e, a integer programming problem), then each solution represents a single path that satisfies the delay and cost constraints. Of course, checking feasibility of integer linear programs is NP-complete.

VIII. Other Extensions

In this section we discuss the extension of the problem to the case with more than two constraints. We also discuss a somewhat different, but sometimes more useful problem in practice, where we minimize the cost subject to a delay constraint.

A. More than two constraints

In a problem with k constraints, we are given a k-weight graph, where each edge is labeled with a k-tuple $(c_1, c_2, \ldots, c_k)$. We are required to find a path such that the sum of the ith weight along the path is less than a bound C_i.

By a straightforward extension of the algorithms in Section IV and Section V, it is easy to show that we can get an exact algorithm with complexity $O(|V||E|\prod_{i=1}^{k} C_i)$, and a bounded-error approximative algorithm with complexity $O(|V|^k|E|(1+\frac{1}{\epsilon})^k)$, where ϵ is the maximum error allowed. The basic idea of the extension is that cost-delay sets now become general *Pareto* sets containing k-tuples of the form $(a_1, \ldots, a_k)$. Such a tuple in the Pareto set associated with some vertex w means that it is possible to get from w to the destination vertex along a path in which the sum of the ith weight is a_i.

It is also possible to extend some parts of the algorithm in Section VI. It is possible to obtain a path with error at most $\epsilon = k-1$ for a problem with k constraints by solving the shortest path algorithm. Also, it seems possible to extend the algorithm which iterates over shortest path problems to the case of three constraints [4].

B. An alternative formulation

An alternative and sometimes more useful formulation is when a bound is given on the delay, and subject to this, we are required to minimize the cost. The algorithms of Section IV and Section V can be straightforwardly extended to solve this problem. In the final cost-delay set of the source node, we find the pair (c_i, d_i) with the smallest c_i. This corresponds to an optimal path to the destination with minimal cost c_i.

To be able to solve this alternate formulation, we also augmented the algorithm of Section VI. To find a path which meets the delay constraint D and has minimum cost C, we solve a problem with delay constraint D and cost constraint C where C is initially chosen to be large. We then find the minimum cost by performing a binary search on C.

In the experimental results presented in Section IX, this alternative formulation of the problem is solved.

IX. Experimental results

We have implemented in C the bounded-error approximative algorithm (section V) and the shortest-path based algorithm (section VI). In this section, we report results obtained by applying the algorithms on a number of multi-weight graphs. Our objective was to see how well the algorithms perform on graphs of medium to large size. Also, to check how sensitive the algorithms were to different parameters (e.g., number of weights, source/destination pairs, step-error).

The graphs were obtained by translating elevation maps of physical landscapes [1]. A landscape of dimension $n_1 \times n_2$ resulted in a graph with $n_1 \cdot n_2$ vertices and approximately $4 \cdot n_1 \cdot n_2$ edges (central vertices having four successors, "north, south, east, west"). The cost c of an edge was taken to be the difference in elevation between the destination and source vertices. The "delays" d_1 and d_2 (the second delay was used only in 3-weight graphs) were generated randomly according to a Gaussian distribution. Tables II, III, and IV present the results. The notation used in these tables is explained in table I.

From tables II and III, the following observations can be made:

- The shortest-path algorithm is two or more orders of magnitude faster than the bounded-error approximative algorithm, while at the same time producing paths which are both feasible (w.r.t. d_1) and as good as the paths produced by the bounded-error algorithm (w.r.t. c).
- The bounded-error approximative algorithm is sensitive to the step-error parameter, δ_ϵ. Reducing δ_ϵ by one or two orders of magnitude resulted in dramatic increases in running time.
- The algorithms are not very sensitive on the particular source/destination pair.

[1] I.e., 2-dimensional arrays, the i,j-th element giving the altitude of the longitude-latitude point corresponding to coordinates i,j.

n	Number of vertices
m	Number of edges
t	Execution time (CPU) in seconds
c	"Cost" of path
d_1	"Delay 1" of path
d_2	"Delay 2" of path
δ_ϵ	Step-error for bounded-error approximate algorithm

TABLE I

NOTATION FOR TABLES II AND III.

	1st source/dest. pair			2nd source/dest. pair		
	$\delta_\epsilon = 10^{-4}$	$\delta_\epsilon = 10^{-5}$	$\delta_\epsilon = 10^{-6}$	$\delta_\epsilon = 10^{-4}$	$\delta_\epsilon = 10^{-5}$	$\delta_\epsilon = 10^{-6}$
graph 1	$t = 70$	$t = 413$	$t = 1274$	13	51	118
$n = 4641$	$c = 1532$			$c = 722$		
$m \approx 4 \cdot n$	$d_1 = 2.15\%$			$d_1 = 0.2\%$	$d_1 = 0.14\%$	
graph 2	$t = 56$	$t = 1278$	$t = 16740$	$t = 153$	$t = 4668$	$t = 32477$
$n = 36417$	$c = 4152$			$c = 1172$		
$m \approx 4 \cdot n$	$d_1 = 0.2\%$	$d_1 = 0.15\%$		$d_1 = 0.3\%$		
graph 3	$t = 725$	$t = 3503$	$t = 9971$	$t = 193$	$t = 419$	$t = 1387$
$n = 225680$	$c = 9409$			$c = 4298$		
$m \approx 4 \cdot n$	$d_1 = 0.01\%$	$d_1 = 0\%$		$d_1 = 0.02\%$	$d_1 = 0\%$	

TABLE II

RESULTS OF THE BOUNDED-ERROR APPROXIMATIVE ALGORITHM ON 2-WEIGHT GRAPHS.

	1st source/dest. pair	2nd source/dest. pair
graph 1	$t = 0.94$	$t = 0.48$
$n = 4641$	$c = 1564$	$c = 723$
$m \approx 4 \cdot n$	$d_1 = 1.27\%$	$d_1 = 0.14\%$
graph 2	$t = 7.16$	$t = 3.48$
$n = 36417$	$c = 4220$	$c = 1184$
$m \approx 4 \cdot n$	$d_1 = 0.04\%$	$d_1 = 0\%$
graph 3	$t = 47.96$	$t = 31.56$
$n = 225680$	$c = 9411$	$c = 4487$
$m \approx 4 \cdot n$	$d_1 = 0\%$	$d_1 = 0\%$

TABLE III

RESULTS OF THE SHORTEST-PATH ALGORITHM ON 2-WEIGHT GRAPHS.

From table IV, we see that adding one more weight/constraint to the problem dramatically increases the execution time of the bounded-error approximative algorithm, with respect to 2-weight graphs. Whereas we have been able to execute the algorithm in 2-weight graphs of size up to 225680 vertices, why we could only treat 3-weight graphs of relatively small sizes (up to 1700 vertices).

X. CONCLUSION

In this paper, we have presented several different algorithms for solving the routing problem with multiple constraints. These algorithms vary in their complexity and the accuracy of their solutions. We have implemented these algorithms and compared their performance on different graphs. In our future work we will look to turning these algorithms into a dis-

	$\delta_\epsilon = 10^{-4}$	$\delta_\epsilon = 10^{-5}$	$\delta_\epsilon = 10^{-6}$
graph 4	$t = 1.49$	$t = 1.80$	$t = 1.85$
$n = 777$	$c = 720$		
$m \approx 4 \cdot n$	$d_1 = 5.6\%$		
	$d_2 = 6.43\%$		
graph 5	$t = 4.25$	$t = 153.97$	$t = 6095.42$
$n = 1178$	$c = 994$		
$m \approx 4 \cdot n$	$d_1 = 1.79\%$		
	$d_2 = 5.39\%$		
graph 6	$t = 1352.54$	$t = 17631.51$	$t = 29274.12$
$n = 1722$	$c = 1024$		
$m \approx 4 \cdot n$	$d_1 = 3.68\%$		
	$d_2 = 4.66\%$		

TABLE IV

RESULTS OF THE BOUNDED-ERROR ALGORITHM ON 3-WEIGHT GRAPHS.

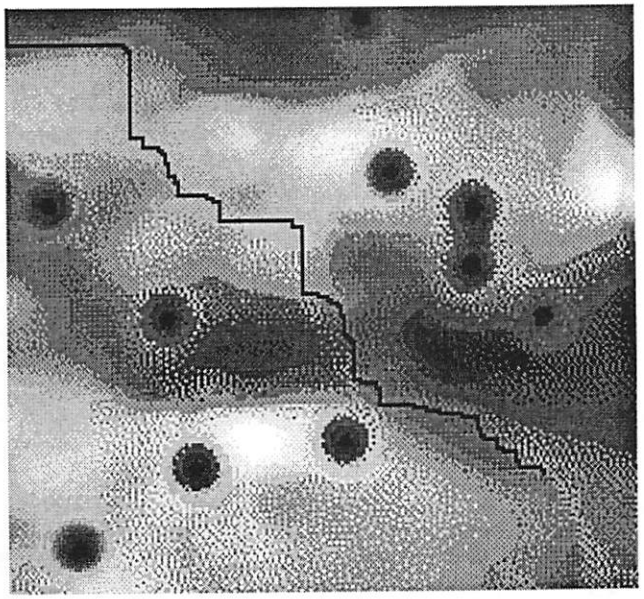

Fig. 6. An output of the bounded-error approximative algorithm on a map translated into a 2-weight graph: the solid black line depicts the path; red dots are "high-delay" zones; the grey scale background represents the elevation variations of the landscape (white: high, black: low).

tributed protocol for QOS routing.

REFERENCES

[1] M.R. Garey and D.S. Johnson, *Computers and Intractability — A Guide to the theory of NP-Completeness*, Freeman, San Francisco, 1979.

[2] R. Hassin, Approximation Schemes for the Restricted Shortest Path Problem, *Mathematics of Operations Research*, Vol. 17, No. 1, Feb. 1992.

[3] J. M. Jaffe, Algorithms for Finding Paths with Multiple Constraints, *Networks*, Vol. 14, 1984, pp. 95-116.

[4] R. Ogier. Personnal communication.

[5] Z. Wang and J. Crowcroft, Quality-of-Service Routing for Supporting Multimedia Applications, *IEEE Journal on Selected Areas in Communications*, Vol. 14, No. 7, Sept. 1996.