Efficient Computation of Delay-sensitive Routes from One Source to All Destinations

Ashish Goel
Univ. of Southern California

K.G. Ramakrishnan
Winphoria Networks

Deepak Kataria
Lucent Bell Labs

Dimitris Logothetis
Lucent Bell Labs

Abstract— **In this paper we describe an efficient algorithm for the constrained shortest path problem which is defined as follows. Given a directed graph with two weights on each link e, a cost l_e and a delay t_e, find the cheapest path from a source to all destinations such that the delay of each path is no more than a given threshold. The constrained shortest path problem arises in Quality-of-Service-sensitive routing in data networks and is of particular importance in real-time services. The problem formulation and the algorithmic framework presented are quite general; they apply to IP, ATM, and optical networks.**

Unlike previous algorithms, our algorithm generates paths from one source to all destinations. Our algorithm is strongly polynomial, and is asymptotically *faster* than earlier algorithms. We corroborate our analysis by a preliminary simulation study.

I. Introduction

The basic problem in QoS-sensitive routing for emerging services such as VoIP (Voice over IP), video, interactive multimedia etc. is to find the cheapest route from a source to a destination (or destinations) that satisfies one or more QoS criteria. The most important QoS criteria are end-to-end bandwidth requirement and end-to-end delay threshold. This basic problem is important for IP networks as well as optical networking.

Most QoS criteria fall into one of two categories depending on the routing induced by the criterion. Monotone criteria result in the routing to a specific destination taking place along a tree. A key property of monotone criteria is that sub-paths of optimal paths are themselves optimal. End-to-end bandwidth requirement is an example of a monotone QoS criterion. These can be implemented using simple link filters. In general, since monotone criteria result in a tree, OSPF extensions for QoS are particularly well suited for monotone criteria [2].

The other kind of QoS criteria are what we call additive metrics. The prime example of additive QoS metrics is delay: the delay of a path is the sum of the delays of the links along the path. The goal is to find the cheapest path with respect to the cost metric, such that the delay of the path is less than a user-specified end-to-end delay threshold. Unlike monotone criteria, additive metrics need not result in routing to a specific destination taking place along a tree. In figure 1, for example, the cheapest paths from the source S which satisfies a delay threshold of 15 do not form a tree. Since the routes do not form a tree, it is infeasible to store routing tables inside the network as simple next hop tables per destination like OSPF and RIP [5]. Storing the entire path or even the next hop for each separate source-destination pair and delay threshold is impractical since the routing tables would grow too large. Instead, we need an efficient algorithm to compute these delay sensitive routes at the source.

This problem is known to be NP-hard [3]. Previous theoretical work on this problem has focussed on finding approximately optimal paths from one source to a single destination. In this paper we make some simple engineering observations which lead to an algorithm for computing approximately optimal paths from one source to all destinations in *less time* asymptotically than previous algorithms for the single-source single-destination problem. The single source multiple destination nature of our algorithm makes it particularly well suited for computing route caches at the source. Our algorithm also has various other desirable properties which we describe later. Our algorithm is currently being implemented for use in Lucent Technologies' optical network management system.

Background

The problem of computing optimal delay constrained routes at the source is an instance of the constrained shortest paths problem. Formally, we are given a network $G(V, E)$ where V is the set of nodes and E the set of links. The network has n nodes and m links. There are two metrics defined on each link: a cost metric l and a delay metric t. We are also given a user-specified delay threshold T and a source node s. The goal is to find the cheapest paths (as measured using the cost metric l) from s to all destinations that satisfy the delay threshold T. This problem is known to be NP-hard [3]. Several Lagrangian relaxation algorithms have been proposed (see [6] for an example) but no theoretical bounds are known for these algorithms. Further, Lagrangian relaxation methods are restricted to single-source single-destination computations. Hassin proposed an algorithm which computes an approximately optimal path from a source s to a *specific* destination d in time $O(\frac{mn}{\epsilon} \log\log \frac{\text{UB}}{\text{LB}})$, where UB and LB are the costs of the fastest and the cheapest path from source s to destination d. This algorithm is guaranteed to find a path, if one exists, which satisfies the delay threshold exactly. Also, the cost of the path is guaranteed to be within $(1 + \epsilon)$ of the cost of the cheapest path which satisfies the delay threshold[1]. This algorithm is not strongly polynomial i.e. the running time depends not only on the size of the graph but also on the costs of the links. Strongly polynomial variants of this algorithm have also been proposed [8], [7].

Ashish Goel is at the Department of Computer Science, University of Southern California. He was at Lucent Bell Labs when this work was done. Email: agoel@cs.usc.edu

K.G. Ramakrishnan is the Director of the Network Planning and Optimization group at Winphoria Networks. He was at Lucent Bell Labs when this work was done. Email:ram@winphoria.com

Deepak Kataria is at Lucent Bell Labs, Murray Hill, NJ. Email: kataria@lucent.com

Dimitris Logothetis is at Lucent Bell Labs. Email: dlogothetis@lucent.com

[1] This algorithm is a fully polynomial time approximation scheme, or FPTAS.

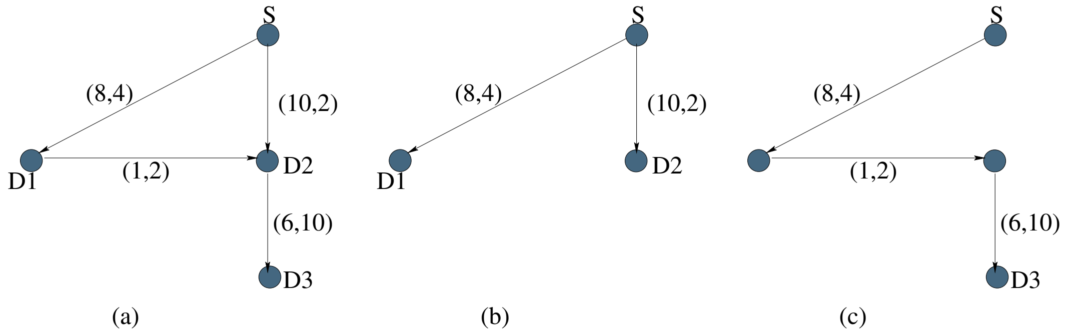

Fig. 1. An example to show that the cheapest paths satisfying a delay threshold do not form a tree. The pair of numbers at each link denote (delay,cost). The delay threshold is 15. S is the source. Part (b) shows the optimal paths from S to D1 and D2, and (c) shows the optimal path from S to D3.

Relaxing the delay constraint

Our algorithm is based on the observation that it might be acceptable to violate the delay threshold[2] upto a small constant fraction ϵ. Thus we will attempt to find the cheapest paths which satisfy the delay threshold T but we will also accept paths which have a delay of up to $T(1+\epsilon)$. In other words, instead of relaxing the costs, we will relax the delay constraint. As we will see later this relaxation allows us to compute single source multiple destination paths.

Our results

Given a source s and a delay threshold T, a destination d is said to be feasible if there is a path from s to d with delay at most T. In this paper we present an algorithm that computes paths from the source s to *all* feasible destinations d such that

1. The cost of the path from s to d is at most the cost of the cheapest path from s to d with delay at most T.
2. The delay of the the path from s to d is at most $(1+\epsilon)T$.

The running time of our algorithm is $O((m + n\log n)D/\epsilon)$ where D is the length, in hops, of the longest path that we find. D can be at most $n-1$ in the worst case. This compares favorably with the best known single-source single-destination running time of $O(mn(1/\epsilon + \log n))$ due to Raz and Lorenz [7]. While there may be instances where $mn < (m + n\log n)D$, in realistic networks, D is much smaller than $n/\log n$. Our algorithm is strongly polynomial, and is *progressive* i.e. the quality of the solution improves progressively so that the algorithm can be interrupted at any time. Further, the algorithm does not need to know D in advance. We use standard dynamic programming techniques to obtain our result. Our main contribution is the observation that relaxing the delay constraint leads to a much simpler, faster, and single-source multiple destination solution. Our analysis is corroborated by a preliminary simulation study on random networks.

Section II presents our algorithm and running time analysis. Section III presents our simulation results.

[2]The approach in [4], [8], [7] is to discretize the cost and then "search" for the right discretization. This leads to an approximation for the cost. Since the right discretization depends on the destination, this approach seems doomed to find single-source single-destination paths.

```
Inputs: Graph $G$ with link costs $l_{ij}$ and delays $t_{ij}$ and a delay threshold $T$.
Outputs: Tables $L(v,t)$ and $P(v,t), 1 \le v \le n, 0 \le t \le t$. The entry $L(v,t)$ is the cost of the cheapest path from 1 to $v$ whose delay is no more than $t$. The entry $P(v,t)$ encodes the cheapest path from 1 to $v$, whose delay is no more than $t$.
1. Initialize $L(1,t) = 0, \quad t = 0, \cdots, T$
2. Initialize $L(j,0) = \infty, \quad j = 2, \cdots, n$
3. Compute $L(j,t) = \min\{L(j,t-1),$
        $\min_{k|t_{kj} \le t \text{ and } (k,j) \in E} \{L(k, t - t_{kj}) + l_{kj}\}\}$
where $j = 2, \cdots, N, t = 1, \cdots, T$
```

Fig. 2. Subroutine DAD which iterates on integer delays.

II. Delay Scaling Algorithm

In this section we present our algorithm, DSA (Delay Scaling Algorithm); the name comes from the fact that the algorithm works by scaling delays. Recall that there are n nodes, m links, and the delay threshold is T. We will assume that the nodes are numbered from 1 to n, with the source being numbered 1. Recall that a node d is feasible if there is a path from s to d with delay at most T. We will preprocess the network by pruning out all nodes which are not feasible using one single run of Dijkstra's shortest path algorithm using the delay metric.

DSA uses the simple dynamic programming algorithm DAD(G,T), described in figure 2 as a subroutine. In this subroutine we assume that all link-delays as well as the delay threshold T are positive integers. Let $L(i,t)$ be the cost of a cheapest path from source node 1 to node i with delay at most t. DAD(G,T) builds up the table $L(i,t)$. The time complexity of the subroutine DAD is $O(mT)$, where m is the number of links in the network.

As mentioned earlier, the delay-scaling algorithm, DSA(G,T,ϵ) finds an approximate solution to the constrained shortest path problem. More specifically, it computes paths from the source s to *all* destinations d such that
1. The cost of the path found from s to d is at most the cost of the cheapest path from s to d with delay at most T.
2. The delay of the the path from s to d is at most $(1+\epsilon)T$.

The delay scaling algorithm attempts to balance the tradeoff between scaling down the delay requirement (so that DAD can work faster) and the inaccuracy introduced by truncating scaled delays to integers. The algorithm works as follows:

A τ-scaling G_τ of graph G is obtained by multiplying the delay on each link in G by τ/T and then truncating the new delay to an integer. Some of the delays may now become zero – we will ignore this fact in the initial description and analysis of the algorithm and will get back to it later. The corresponding delay constraint for the new scaled graph is taken to be τ. The algorithm DSA(G, T, ϵ) works as follows:

1. Set $\tau = \tau_0$ (a small number $\ll T$)
2. Call $DAD(G_\tau, \tau)$ to compute the $L(v,t)$ and $P(v,t)$ tables
3. Compute the delays in the original graph G for each of paths $P(v,\tau)$ and store the delays in $D(v)$.
4. If $\exists v \in G$ such that $D(v) > T(1+\epsilon)$ set $\tau = 2\tau$ and go to Step 2.

The table calculated during the last invocation of DAD contains the constrained shortest paths.

Theorem II.1: The worst case complexity of the algorithm is $O(\frac{mD}{\epsilon})$, where D is the maximum length in hops of any optimal constrained shortest path in the graph G.

Proof: To obtain this result we need the following lemma. The proof of the lemma is straightforward and is omitted.

Lemma II.1: Let $P_\tau(v)$ denote the cheapest path from the source to any vertex v in G_τ satisfying the delay constraint τ. The cost of $P_\tau(v)$ is no more than the cost of the cheapest path between s and v satisfying the delay constraint T in G. Further, the delay of $P_\tau(v)$ in G is at most $T(1 + D/\tau)$ where D is the maximum length, in hops, of any cheapest path satisfying the delay constraint τ.

We now continue with the proof of the theorem. If $\tau \geq D/\epsilon$ then the delay of $P_\tau(v)$ in G is at most $T(1+\epsilon)$, and the algorithm $DSA(G, T, s, \epsilon)$ terminates during step 3. Thus, the running time of DSA is at most of order

$$m\tau_0 + 2 \cdot m\tau_0 + \ldots + 2^k \cdot m\tau_0,$$

where $k = \left\lceil \log \frac{D/\epsilon}{\tau_0} \right\rceil$. Summing up, the running time is at most of order

$$4 \cdot 2^{k-1}\tau_0 < 4mD/\epsilon.$$

Notice that the algorithm guarantees a running time of $O(mD/\epsilon)$ without needing to know D. The constant hidden inside the O-notation is quite small. ■

Clearly $D < n$. The worst case running time of this algorithm is independent of the delay/cost values and hence, this algorithm scales well as these values change. Further, a single invocation of DSA finds the constrained shortest path from the source to all destinations.

To handle zero-delay links, we must run an invocation of Dijkstra's algorithm during each iteration of the dynamic program DAD to update costs due to zero delay. The only links considered during this invocation will be zero delay links. Since Dijkstra's algorithm runs in time $O(m+n\log n)$, the overall running time of our algorithm would be $O((m+n\log n)D/\epsilon)$. As stated earlier, this is asymptotically better than $O(mn\log n + mn/\epsilon)$ of Lorenz and Raz [7] in most realistic scenarios; further, DSA can compute paths from one source to all destinations.

Several properties of DSA are worth reiterating:

1. The time complexity result has a value D, but the algorithm does not need to know this value.
2. A slightly weaker time complexity result for our algorithm is $O(\frac{(m+n\log n)n}{\epsilon})$, but in most real-life cases the tighter complexity result we present above is orders of magnitude better than the weaker result.
3. DSA has a progressive property that is useful in practice: the accuracy of the solution progressively improves with successive invocations of DAD. DSA can be terminated in the middle and still produce reasonable answers. For instance, when there is a call processing time budget during call setup, we can terminate the execution when the time runs out, and still have a reasonably accurate solution in the form of the last table computed using DAD.
4. DSA is strongly polynomial, i.e. the running time bound does not depend on the actual delay and cost values on the links. Strongly polynomial algorithms are traditionally considered more robust. Hassin's original algorithm was not strongly polynomial [4].

Delay can be replaced by any additive metric in the above discussion. One interesting metric is packet loss probability, which is not additive itself, but becomes additive when we take its logarithm. Thus the above result could also be used to find the cheapest path satisfying a given packet loss threshold. The basic ideas in this paper can also be extended to multiple constraining metrics.

III. Simulation Results

In this section we compare our delay scaling algorithm to Hassin's algorithm in terms of executions times with the same ϵ $(= 5\%)$. The results here should be taken as a preliminary indication. A more complete simulation study would require a suite of realistic network topologies with both cost and delay data as well as careful implementations of several candidate algorithms (not just Hassin's algorithm); such a study is beyond the scope of this paper.

These results were obtained on a Sun Enterprise 3000 with 1 GB of memory and running SunOS 5.6. All running times reported are in seconds. For our experiments we used random graphs of a given node size n and connectivity p which we defined as the probability of a link existence between nodes i and j. The link cost and delay are also random numbers uniformly distributed in the open interval [1, 1000]. The quantity D (the maximum hop-count) is $O(\log n)$ for this class of networks. We also scale the mean of the delay distribution by a factor $0 < r < \infty$ to achieve the effect of light/heavy network load. Hassin's algorithm is referred to as the cost-scaling algorithm. The column "Dijkstra" refers to the classical shortest path algorithm [1] that finds shortest paths to all destinations; this algorithm ignores the delay metrics on each link and finds the cheapest path to all nodes using only the cost metric. The result from "Dijkstra" is not feasible, and these numbers are only provided to establish a benchmark. We report times for single-source multiple-destinations for all algorithms.

As can be seen from tables I, II, III, IV, and V,the delay scaling algorithm runs much faster than the cost-scaling algorithm. Each running time reported is a mean of twenty independent replications.

No. of nodes	Connec-tivity	cost-scaling	delay-scaling	Dijks-tra
100	0.1	2.192	0.160	0.006
100	0.2	2.730	0.375	0.006
100	0.3	2.582	0.634	0.006
150	0.1	6.252	0.701	0.014
150	0.2	7.394	1.116	0.014
150	0.3	7.802	1.514	0.014
200	0.1	15.432	1.088	0.024
200	0.2	16.110	1.782	0.074
200	0.3	16.721	1.670	0.074
250	0.1	27.447	1.479	0.038
250	0.2	30.011	1.855	0.038
250	0.3	31.372	2.244	0.138
300	0.1	47.220	1.502	0.104
300	0.2	51.379	2.491	0.055
300	0.3	54.273	3.414	0.155

TABLE I
LOAD = 0.1 (VERY LIGHTLY LOADED NETWORK)

No. of nodes	Connec-tivity	cost-scaling	delay-scaling	Dijks-tra
100	0.1	2.236	0.160	0.006
100	0.2	2.525	0.425	0.006
100	0.3	2.717	0.632	0.006
150	0.1	6.295	0.601	0.013
150	0.2	7.469	1.116	0.014
150	0.3	7.705	1.415	0.064
200	0.1	15.443	0.887	0.124
200	0.2	16.142	1.788	0.024
200	0.3	16.533	1.460	0.025
250	0.1	27.516	1.527	0.140
250	0.2	29.947	1.799	0.038
250	0.3	31.254	2.096	0.088
300	0.1	47.340	1.504	0.054
300	0.2	51.582	2.487	0.155
300	0.3	54.216	3.218	0.055

TABLE II
LOAD = 0.5 (LIGHTLY LOADED NETWORK)

No. of nodes	Connec-tivity	cost-scaling	delay-scaling	Dijks-tra
100	0.1	2.222	0.358	0.006
100	0.2	2.517	0.617	0.006
100	0.3	2.652	0.622	0.006
150	0.1	6.334	0.541	0.064
150	0.2	7.551	1.006	0.064
150	0.3	7.774	1.547	0.014
200	0.1	15.647	0.878	0.074
200	0.2	16.379	1.765	0.024
200	0.3	17.018	1.420	0.025
250	0.1	28.078	1.572	0.038
250	0.2	30.738	1.779	0.038
250	0.3	32.012	2.491	0.139
300	0.1	48.099	1.574	0.054
300	0.2	52.409	2.732	0.056
300	0.3	55.078	3.733	0.162

TABLE III
LOAD = 1.0 (MODERATELY LOADED NETWORK)

No. of nodes	Connec-tivity	cost-scaling	delay-scaling	Dijks-tra
100	0.1	2.125	0.211	0.006
100	0.2	2.575	0.419	0.006
100	0.3	2.438	0.532	0.056
150	0.1	6.405	0.494	0.014
150	0.2	7.595	0.959	0.064
150	0.3	7.622	1.301	0.014
200	0.1	15.697	1.172	0.024
200	0.2	16.237	1.919	0.024
200	0.3	16.784	1.537	0.075
250	0.1	28.155	1.604	0.040
250	0.2	30.938	1.788	0.038
250	0.3	32.144	2.459	0.188
300	0.1	47.929	1.478	0.155
300	0.2	52.039	2.686	0.155
300	0.3	55.315	3.831	0.207

TABLE IV
LOAD = 1.5 (HEAVILY LOADED NETWORK)

IV. CONCLUSIONS

We studied the problem of delay sensitive routing. Relaxing the delay constraint instead of approximating the costs results in an algorithm that is asymptotically faster than earlier algorithms, and more importantly, computes routes from a single source to all destinations; earlier algorithms were single-source single-destination.

REFERENCES

[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1976.

[2] G. Apostolopoulos and S. Guerin, R. Kamat. Implementation and performance measurements of qos routing extensions to ospf. *Proceedings of INFOCOM'99*, 1999.

[3] M.R. Garey and D.S. Johnson. *Computers and Intractability: A guide to the theory of NP-completeness*. Freeman, San Francisco, 1978.

[4] R. Hassin. Approximation schemes for the restricted shortest path problem. *Mathematics of Operations Research*, 17:36–42, 1992.

[5] C. Huitema. *Routing in the Internet*. Prentice Hall, 1995.

[6] J.M. Jaffe. Algorithm for finding paths with multiple constraints. *Networks*, 14:95–116, 1984.

[7] D. Lorenz and D. Raz. A simple efficient approximation scheme for the restricted shortest paths problem. *Bell Labs Technical Memorandun*, 1999.

[8] C. Phillips. The network inhibition problem. *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, May 1993.

No. of nodes	**Connec-tivity**	**cost-scaling**	**delay-scaling**	**Dijks-tra**
100	0.1	2.202	0.156	0.006
100	0.2	2.454	0.315	0.006
100	0.3	2.603	0.572	0.006
150	0.1	6.527	0.544	0.014
150	0.2	7.295	0.654	0.064
150	0.3	7.537	1.246	0.064
200	0.1	15.655	1.064	0.074
200	0.2	16.111	1.752	0.024
200	0.3	16.918	1.616	0.125
250	0.1	27.692	1.461	0.038
250	0.2	30.355	1.920	0.088
250	0.3	31.641	2.557	0.089
300	0.1	47.605	1.566	0.054
300	0.2	52.264	2.696	0.105
300	0.3	54.561	3.805	0.155

TABLE V

LOAD = 2.0 (VERY HEAVILY LOADED NETWORK)