October 2002

Computing Shortest Paths for Any Number of Hops

Roch A. Guérin
University of Pennsylvania, guerin@acm.org

Ariel Orda
Technion-Israel Institute of Technology

Copyright 2002 IEEE. Reprinted from *IEEE/ACM Transactions on Networking*, Volume 10, Issue 5, October 2002, pages 613-620.
Publisher URL: http://ieeexplore.ieee.org/xpl/tocresult.jsp?isNumber=22314&puNumber=90

Computing Shortest Paths for Any Number of Hops

Abstract
In this paper, we introduce and investigate a "new" path optimization problem that we denote the *all hops optimal path* (AHOP) problem. The problem involves identifying, for all hop counts, the optimal, i.e., minimum weight, path(s) between a given source and destination(s). The AHOP problem arises naturally in the context of quality-of-service (QoS) routing in networks, where routes (paths) need to be computed that provide services guarantees, e.g., delay or bandwidth, at the minimum possible "cost" (amount of resources required) to the network. Because service guarantees are typically provided through some form of resource allocation on the path (links) computed for a new request, the hop count, which captures the *number* of links over which resources are allocated, is a commonly used cost measure. As a result, a standard approach for determining the cheapest path available that meets a desired level of service guarantees is to compute a minimum hop shortest (optimal) path. Furthermore, for efficiency purposes, it is desirable to *precompute* such optimal minimum hop paths for all possible service requests. Providing this information gives rise to solving the AHOP problem. The paper's contributions are to investigate the computational complexity of solving the AHOP problem for two of the most prevalent cost functions (path weights) in networks, namely, additive and bottleneck weights. In particular, we establish that a solution based on the Bellman–Ford algorithm is optimal for additive weights, but show that this does not hold for bottleneck weights for which a lower complexity solution exists.

Keywords
Hop-restricted shortest paths, maximum bandwidth, minimum delay, networks, quality-of-service routing

Computing Shortest Paths for Any Number of Hops

Roch Guérin, *Fellow, IEEE,* and Ariel Orda, *Senior Member, IEEE*

***Abstract*—In this paper, we introduce and investigate a "new" path optimization problem that we denote the *all hops optimal path* (AHOP) problem. The problem involves identifying, for all hop counts, the optimal, i.e., minimum weight, path(s) between a given source and destination(s). The AHOP problem arises naturally in the context of quality-of-service (QoS) routing in networks, where routes (paths) need to be computed that provide services guarantees, e.g., delay or bandwidth, at the minimum possible "cost" (amount of resources required) to the network. Because service guarantees are typically provided through some form of resource allocation on the path (links) computed for a new request, the hop count, which captures the *number* of links over which resources are allocated, is a commonly used cost measure. As a result, a standard approach for determining the cheapest path available that meets a desired level of service guarantees is to compute a minimum hop shortest (optimal) path. Furthermore, for efficiency purposes, it is desirable to *precompute* such optimal minimum hop paths for all possible service requests. Providing this information gives rise to solving the AHOP problem. The paper's contributions are to investigate the computational complexity of solving the AHOP problem for two of the most prevalent cost functions (path weights) in networks, namely, additive and bottleneck weights. In particular, we establish that a solution based on the Bellman–Ford algorithm is optimal for additive weights, but show that this does not hold for bottleneck weights for which a lower complexity solution exists.**

***Index Terms*—Hop-restricted shortest paths, maximum bandwidth, minimum delay, networks, quality-of-service routing.**

I. Introduction

THE problem that this paper investigates, namely, the all hops optimal path (AHOP) problem, is one that arises naturally from several routing problems that have recently received significant attention in the data networking community.[1] Data networks are increasingly used by streaming applications that require both bandwidth and delay guarantees. Similarly, the introduction of virtual private network (VPN) offerings and the use of service level agreements (SLAs) are also associated with service guarantees such as minimum bandwidth or maximum delay. Hence, it has become important to ensure that traffic with specific service guarantees is routed over paths that are capable of meeting them (see [1] for a comprehensive survey). Furthermore, for efficiency purposes, it is also important to do so at the minimum possible cost to the network.

From an algorithmic standpoint, this calls for algorithms that compute paths satisfying specific (service) constraints, while minimizing the network cost function. This corresponds to a standard optimal path selection under constraints that has been widely studied in many different settings (see [2] and [3] for a general discussion and presentation of numerous algorithms). The problem in its most general form, i.e., multiple additive constraints, is known to be NP-complete [4], and, as a result, much effort has been devoted to developing efficient heuristics (again, see [1] for a recent survey). However, there are a number of "special" cost functions that are of interest in data networks, not only because they provide practical cost measures, but also because they are amenable to tractable solutions (see, e.g., [5], [6]). In particular, the number (count) of hops in a path is both a realistic and popular measure of cost.

Hop count accurately captures network cost as it measures the number of links over which network resources are expended. For example, a path that has been guaranteed a minimum available bandwidth of, say, 10 Mb/s (10^7 bits/s), will consume that amount of bandwidth on each one of the links that it traverses. Hence, the more links on a path, the more expensive in terms of the total amount of network resources that are consumed. As a result, minimizing path length, or hop count, is a natural criterion for computing efficient paths for traffic with specific service requirements. In addition to being a reasonably realistic cost measure, minimum hop paths that meet service constraints can be computed using simple algorithms, e.g., the Bellman–Ford (BF) algorithm, as we shall discuss later, and this makes them attractive from a computational standpoint. As a result, computing minimum hop paths with service constraints has become a recurring problem in data networks, e.g., see [7] and [8] for recent examples and proposals.

The extension from this single minimum hop path computation to the AHOP problem, originates from the need to amortize the associated computational cost over multiple path requests. This is because even if the algorithms involved are tractable, performing them repeatedly for every new request can translate into a significant computational load (see [9] for an experimental investigation). One option for minimizing this overhead is to *precompute* minimum cost (hop count) paths for all possible requests. Assuming that this is feasible, such a precomputation can then be performed only occasionally, e.g., as the state of the network changes significantly (see [10] for a discussion of this issue), and the precomputed paths are reused each time a new request is made.

Manuscript received May 30, 2000; revised April 3, 2001; approved by IEEE TRANSACTIONS ON NETWORKING Editor G. Sasaki. The work of R. Guérin was supported in part by the National Science Foundation under Grant ANI99-06855, Grant ANI99-02943, and Grant ANI00-85930.

R. Guérin is with the Department of Electrical Engineering, University of Pennsylvania, Philadelphia, PA 19104-6314 USA (e-mail: guerin@ee.upenn.edu).

A. Orda is with the Department of Electrical Engineering, Technion–Israel Institute of Technology, Haifa 32000 Israel (e-mail: ariel@ee.technion.ac.il).

Digital Object Identifier 10.1109/TNET.2002.803917.

[1]Archives of the mailing lists of several IETF (http://www.ietf.org) Working Groups such as MPLS and Traffic Engineering, contain ample evidences of associated activities and numerous pointers to relevant publications, RFCs, and Internet Drafts.

The main challenge associated with such a precomputation is that the full range of possible future requests is typically not known ahead of time. Solving the AHOP problem eliminates this difficulty by computing, for each hop count, the *best* service guarantees that are feasible between a source (the computing node) and all other destinations in the network. For example, when service guarantees correspond to bottleneck metrics such as bandwidth guarantees, solving the AHOP problem amounts to computing, for each hop count, the maximum bandwidth path between the source node and all destinations. Higher hop count paths are considered only if they offer higher bandwidth. Once this information is available, the optimal (cheapest) path meeting the bandwidth requirements of a new request is then found simply by choosing from the computed paths the one with the smallest hop count for which the available bandwidth exceeds the required amount. A similar approach can be used when service guarantees are associated with additive metrics such as delay. As a result, solving the AHOP problem has the potential for offering a practical and computationally efficient solution for computing paths that meet certain service guarantees. It is, therefore, important to develop efficient solutions to the AHOP problem, especially for service metrics that are commonly used by users of data networks. A first step in that direction is to develop a better understanding of the intrinsic complexity of the AHOP problem, and this is the focus of this paper.

Specifically, in this paper we investigate the computational complexity of the AHOP problem for the two most important service metrics in practice, namely, bottleneck and additive metrics. In the case of bottleneck metrics, the weight of a path is given by the maximum (or minimum) value of its link weights. The most common example of a bottleneck metric in data networks is bandwidth, where many users require paths that can guarantee the availability of a given amount of bandwidth. As illustrated in our earlier example, setting link weights to the *inverse* of their available bandwidth yields a minimum (bottleneck) weight path that is one with maximum bandwidth. Additive metrics arise in many settings. For example, end-to-end delay, jitter (delay variation), and reliability all correspond to additive metrics for which the weight of a path is given by the sum of its link weights, i.e., the sum of link delays, or delay variations, or, in the case of reliability, the sum of the logarithms of the link failure probabilities. Our goal in this paper is to provide a thorough understanding of the computational cost of solving the AHOP problem for these two major types of service metrics, and, in particular, either identify minimum complexity solutions or obtain bounds on the complexity of any solution.

Our starting point is the solution proposed in [8], which relies on the BF algorithm. This algorithm is a natural match for the AHOP problem, as it proceeds by increasing hop count, and can therefore be readily modified to generate shortest paths for *each* hop count. Consequently, we know from basic results on the complexity of the BF algorithm that for a network with M edges and a maximum possible hop count of H for simple paths,[2] there exists a solution to the AHOP problem of complexity $O(H \cdot M)$. The basic question we are interested in answering is whether this is indeed the best we can do for the two types of metrics we consider. For example, in the case of the standard shortest path problem, there exists a solution of lower complexity than the BF algorithm, namely, of complexity $O(N \log N + M)$, which is offered by the Fibonacci-heap implementation of Dijkstra's algorithm.

The answer to this question is somewhat surprising. Unlike the standard shortest path problem, for which the same (efficient) solution can be used for *both* bottleneck and additive metrics, we show in the paper that this does not hold for the AHOP problem. Specifically, we establish that for additive metrics, $\Omega(N^3)$ is a (tight) lower bound on the complexity of any solution to the AHOP problem among a certain class of *path-comparison-based* algorithms, which includes standard solutions such as Dijkstra's and Bellman–Ford's. The proof is based on an extension of a similar result, obtained in [11] for the all-pairs shortest path problem, and establishes the optimality of a BF-based solution for additive metrics. However, for bottleneck metrics, we show that a lower complexity solution is available. Specifically, we present and validate an algorithm, based on several modifications to the BF scheme, which solves the AHOP problem within $O(N^3/\log N)$ steps. These findings have several interesting theoretical as well as practical implications, which are discussed in Section V of this paper.

The rest of the paper is organized as follows. The AHOP problem is formulated more precisely in Section II. The lower bound for the case of additive metrics is established in Section III, while the existence of a more efficient algorithm for bottleneck metrics is derived in Section IV. Section V summarizes the results of the paper and their significance, and points to some potentially interesting extensions.

II. Model and Problem Formulation

Given a directed graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ is the set of edges, let $N = |\mathcal{V}|$ and $M = |\mathcal{E}|$. A *(simple) path* is a finite sequence of nodes $\mathbf{p} = (v_0, v_1, \ldots, v_h)$, such that for $0 \leq n \leq h-1$, $(v_n, v_{n+1}) \in \mathcal{E}$, and for all $0 \leq m < n \leq h$, $v_m \neq v_n$; h is then said to be the *number of hops* (or *hop count*) of $\mathbf{p}$. We denote by H the maximal possible number of hops in a path.

Each edge $(u, v) \in \mathcal{E}$ has a *weight* $w_{uv} \geq 0$. Corresponding to a path $\mathbf{p}$, there is a *path weight* $W(\mathbf{p})$, which is a nondecreasing function of the weights of the links along the path. Formally, for additive weights, we have

$$W(\mathbf{p}) = \sum_{(u,v)\in\mathbf{p}} w_{u,v}$$

while for bottleneck weights we have

$$W(\mathbf{p}) = \max_{(u,v)\in\mathbf{p}} w_{u,v}.$$

Given edge weights and source and destination nodes $s, t \in \mathcal{V}$, an *optimal path* is a path between s and t of minimum weight. An h-hop constrained optimal path is a path of

[2]Obviously, $H \leq N-1$, but the actual value of H is usually much smaller, not only because in typical network topologies the diameter is considerably smaller than N, but also because the value of H is often restricted *a priori* by the routing protocol.

minimum weight, among paths between s and t with hop count of at most h. We are now ready to define the AHOP problem.

All Hops Optimal Path (AHOP) Problem: For a given source node $s \in \mathcal{V}$ and maximal hop count H, $H < N$, find, for each hop count value h, $1 \leq h \leq H$, and destination node $u \in \mathcal{V}$, an h-hop constrained optimal path between s and u.

The distinction between additive and bottleneck weights gives rise to two classes of the AHOP problem, namely, *Additive AHOP (Add-AHOP)* and *Bottleneck AHOP (Bot-AHOP)*. A straightforward solution to both Add-AHOP and Bot-AHOP, of complexity $O(M \cdot H)$, is offered by the BF shortest path algorithm [3]. It is a property of the BF algorithm that, at its hth iteration, it identifies the optimal path between a given source and each destination, *among paths of at most h hops*. In the worst case, the complexity of this solution is $\Theta(N^3)$.

III. Lower Bound for Additive Weights

In this section, we show that $\Omega(N^3)$ is a (tight) lower bound on the complexity of any solution to the Add-AHOP problem that belongs to a class of *comparison-based* algorithms.

We begin by quoting, from [11], the presentation of the class of comparison-based algorithms to which our result applies.

Definition: A *path-comparison-based* shortest-path algorithm $\mathcal{A}$ accepts as input a graph $\mathcal{G}$ and a weight function. The algorithm $\mathcal{A}$ can perform all standard operations. However, the only way it can access the edge weights is to compare the weights of two different paths.

We can think of a path-comparison-based algorithm as being given only the graph and a black-box path-weight comparator. The path weights can be accessed only through the black box. The algorithm must output a reasonable encoding of the shortest paths in the graph. We require that the output has no information about path weights. For example, the weighted graph itself is not a reasonable encoding of the solution. Most commonly used shortest-path algorithms, in particular Dijkstra's and Bellman–Ford's, fit into this class; this is the case also with many other algorithms, e.g., [12], [13]. However, some algorithms are not path-comparison-based, e.g., [14], as it adds weights of edges that do not form a single path. The latter belongs to a class of "algebraic" algorithms that consider shortest paths from the perspective of matrix multiplication. We note that, to the best of our knowledge, such algorithms have not been deployed in communication networks.

In [11], it is established that $\Omega(N^3)$ is a lower bound on the complexity of a path-comparison-based solution to the *all-pairs shortest path problem*. Specifically, it is shown there that there exists a directed graph of $3n$ nodes on which any path-comparison-based algorithm for that problem must perform at least $n^3/2$ path weight comparisons. We shall establish our proof by employing a similar idea. To that end, we begin by presenting the following construction, depicted in Fig. 1. The constructed graph $\mathcal{G}$ is a directed graph with $4n$ nodes and $\Omega(n^3)$ paths. The idea is to show that, if a path-comparison-based algorithm $\mathcal{A}$ (for the Add-AHOP problem) fails to examine one of these paths, then the link weights can be modified to make that path "optimal" (i.e., part of the required solution) without $\mathcal{A}$ being able to detect the change. We proceed to describe the details of the construction.

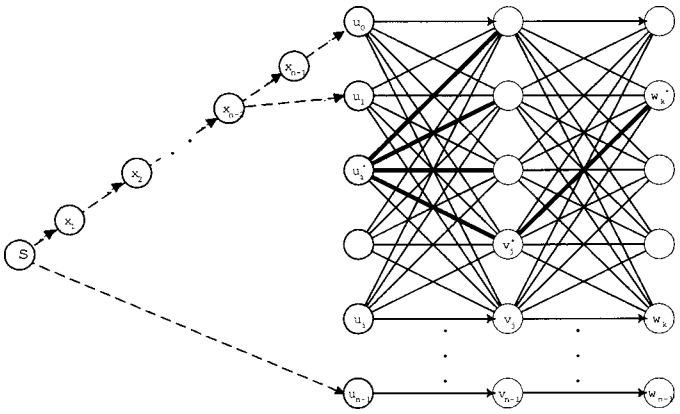

Fig. 1. Graph used for lower bound.

The graph $\mathcal{G}$ is composed of a "core" (depicted by full-line edges) and a "prefix" (depicted by dashed-line edges). The core of $\mathcal{G}$ is a directed tripartite graph on vertices u_i, v_j, and w_k, where i, j, k range from 0 to $n-1$. The edge set is $\{(u_i, v_j)\} \cup \{v_j, w_k)\}$. The prefix is composed of a source s and $n-1$ nodes x_j, $1 \leq j \leq n-1$. For $0 \leq i \leq n-2$, s is connected to each core node u_i through a path $(s, x_1, x_2, \ldots, x_{n-i-1}, u_i)$; in addition, there is a (direct) edge (s, u_{n-1}). Note that s is connected to each u_i through a path of $n-i$ hops. We proceed to define the edge weights. For a base b, define (for any positive integer $r > 0$)

$$[a_r, \ldots, a_0]_b = \sum_{i=0}^{r} a_i b^i$$

where b^i denotes b to the power of i. Then, the weights of core edges are defined to be

$$\begin{aligned} w(u_i, v_j) &= [1, 0, i, 0, j, 0, 0]_{n+1} \\ w(v_j, w_k) &= [0, 1, 0, k, 0, -j, 0]_{n+1}. \end{aligned} \tag{1}$$

Note that negative digits are allowed to appear in the numbers [i.e., note the "$-j$" in (1)]. The standard positive-digit representation of these numbers would require that a carry be taken from the next number to the left; as in [11], this does not affect the correctness of the upcoming proof. The weights of prefix edges, i.e., on a path between s and a node u_i, are simply 0. We note that all the above weights comply with our model assumptions. A path weight is defined as the sum of weights of its edges, i.e., we assume additive weights. We use the above construction in order to establish the following lower bound for the Add-AHOP problem.

Theorem 1: There exists a directed graph of $\Theta(N)$ nodes and some $H < N$, for which any path-comparison-based solution for the Add-AHOP problem, with maximum hop-count value H, must perform $\Omega(N^3)$ comparisons.

Proof: By contradiction, assume that there is such a solution, of lower complexity.

Consider an instance of the Add-AHOP problem on the graph depicted in Fig. 1, where the source node is s and $H = n+2$. A solution to Add-AHOP would need to find, in particular, paths to each of the nodes $w_0, w_1, \ldots, w_{n-1}$.

In [11], it is shown that the optimal path between a node u_i and a node w_k goes through v_0 and has weight $[1, 1, i, k, 0, 0, 0]_{n+1}$. It follows that, for $3 \le h \le H$, the optimal h-hop constrained path between s and w_k goes through u_i and v_0, where $i = n + 2 - h$, and has the above weight.

Suppose that we have a solution which avoids the comparison of paths that have the suffix $(u_{i^*}, v_{j^*}, w_{k^*})$, $j^* > 0$.[3] Define new weights $w'(u, v)$ for some of the "core" edges[4] as follows:

- For all $j \le j^*$, $w'(u_{i^*}, v_j) = [1, 0, i^*, 0, 0, j, j]_{n+1}$;
- $w'(v_{j^*}, w_{k^*}) = [0, 1, 0, k^*, 0, -j^*, -n]_{n+1}$;
- otherwise, $w'(u, v) = w(u, v)$.

Following [11], it can be shown that, under the new weights, $(u_{i^*}, v_{j^*}, w_{k^*})$ is the optimal path between u_{i^*} and w_{k^*}, its weight is $[1, 1, i^*, k^*, 0, 0, j^* - n]_{n+1}$, and the ordering by weight of all other paths remains the same. It can be verified that, with the new weights, for $h^* = n + 2 - i^*$, the optimal h^*-hop constrained path between s and w_{k^*} goes through u_{i^*} and v_{j^*}. Therefore, if we run the algorithm on the modified weights, all comparisons of paths which do no involve $(u_{i^*}, v_{j^*}, w_{k^*})$ as their suffix give the same result as with the original weights, and since the algorithm never performed a comparison involving such paths while running on the original metric, we deduce that it still outputs the same solution, which, for h^*, is now incorrect.

We conclude that any solution must perform at least $\Omega(n^3)$ path comparisons. As $N = 4n$, this establishes the claim. □

IV. Improved Algorithm for Bottleneck Weights

In this section, we show that for bottleneck metrics, the above lower bound does not hold. Specifically, we present and validate an algorithm that solves the Bot-AHOP problem within $O(M \log N + H \cdot (N^2/\log N)) \le O(N^3/\log N)$ steps.

Before proceeding, we point out that our focus is on the key ideas behind the new solution, hence, we do not attempt to optimize storage space nor number of computations, as long as the asymptotic worst case computational complexity expression remains unaffected. We also note that a proper choice of the link weights, which would correspond to the formulation of Section II, would be the reciprocal of the link's bandwidth. Nonetheless, for the sake of simplicity, we shall consider the link metric to be its (plain) bandwidth.

We begin by specifying, in pseudocode, the BF algorithm for a bottleneck (bandwidth) metric.

Algorithm BF

```
Input:
  V = set of vertices, labeled by integers
      1 to N.
  E = set of edges, labeled by ordered
      pairs (u, v) of vertex labels.
  s = source vertex.
  For all edges (u, v) in E:
    b(u, v) = bandwidth of edge (u, v).
  H = maximum hop count.

Variables:
  B[1,...,N][1,...,H]: the maximal bandwidth
       computed for each node at each
       iteration (initialized to
       B[s][h] = infinity and B[u][h] = 0
       for u <> s, for h = 0,...,H).

The Algorithm:

begin
  for h := 1 to H
    for all u in V do B[u,h] := B[u,h-1];
    for all (u, v) in E do
      if (min(B[u,h-1], b(u,v))) > B[v,h]
        then B[v,h] := min(B[u,h-1], b(u,v));
end.
```

The improved algorithm, to be presented in the following, is based on the exploitation of the following two simple observations.

1) If a node v has already at least as much bandwidth as an incoming link (u, v), then that link cannot affect any longer the outcome for node v, hence, it can be deleted.

2) At any iteration, a link (u, v) should be considered only if the current bandwidth of node u is higher than that of node v.

Before proceeding, we note without proof[5] the following properties of Algorithm BF.

Lemma 1: Upon completion of Algorithm BF, if for some h it holds that $B[v][h] \ge b(u, v)$, then $B[v][h'] \ge b(u, v)$ for all $h' \ge h$.

Lemma 2: In Algorithm BF, the following relation holds, for all $(u, v) \in \mathcal{E}$ and $1 \le h \le H$:

$$\max(B[v][h], \min(B[u][h-1], b(u, v))) = \begin{cases} B[v][h]: & B[v][h] \ge \min(B[u][h-1], b(u, v)) \quad \text{(A)} \\ B[u][h-1]: & b(u, v) > B[u][h-1] > B[v][h] \quad \text{(B)} \\ b(u, v): & B[u][h-1] \ge b(u, v) > B[v][h]. \quad \text{(C)} \end{cases} \tag{2}$$

Corollary 1: In Algorithm BF, for an edge $(u, v) \in \mathcal{E}$, the result of (2) is: 1) according to either line (A) or line (B) at most once per iteration h, and 2) according to line (C) at most once over all iterations.

Proof: The first part is trivial; the second part is due to Lemma 1. □

Another simple but useful observation is the following.

Lemma 3: During the execution of Algorithm BF, $B[u][h]$ takes at most M different values, over all $u \in \mathcal{V}$, $u \ne s$, and all iterations $1 \le h \le H$.

Proof: It is easy to verify (e.g., by induction on the iteration number h) that for all h and $u \ne s$, there is some $(u', v') \in \mathcal{E}$ such that $B[u][h] = b(u', v')$. □

Consider now the following assumption, which will be dropped later.

[3]Note that, otherwise, the solution involves $\Omega(n^3)$ path comparisons, as there are $\Theta(n^3)$ triplets (u_i, v_j, w_k), for $0 \le i \le n-1$, $0 < j \le n-1$, $0 \le k \le n-1$.

[4]In Fig. 1, edges with modified weights are depicted by thicker lines.

[5]Proofs are either immediate or directly obtainable from basic properties of the BF algorithm.

Assumption A: The test $B[u][h-1] > B[v][h]$ can be performed with no penalty, in terms of computation complexity.

With this assumption, we can use the following modified algorithm.

Algorithm BF′:

```
begin
  for h := 1 to H
    for all u in V do B[u, h] := B[u, h − 1];
  for all (u, v) in E:
  i)  if B[v][h] < B[u][h − 1]
      then begin
  i.a)  if B[u][h − 1] < b(u, v)
          then B[v][h] := B[u][h − 1]
        else if B[v][h] < b(u, v)
          then B[v][h] := b(u, v);
  i.b)  if B[v][h] >= b(u, v)
        then E := E\(u, v);
      end;
end.
```

By Assumption *A*, the test i) would incur no computational penalty. Next, consider step i.a): each execution of that step falls within the category of one of the three cases (A), (B), or (C), as defined in (2); hence, the computational penalty incurred by that step can be obtained by counting the times it is executed under the conditions of each of the three cases. By Corollary 1, step i.a) would be performed $O(M)$ times as case (C) and $O(H \cdot M)$ as case (B); moreover, it is easy to verify that, due to the test in step i), step i.a) is performed as case (A) only when $B[v][h] \geq b(u, v)$ holds, which, in view of step i.b), implies that the step would be performed as case (A) $O(M)$ times. Finally, turning to step i.b), it is clear that it would be performed at most $O(H \cdot M)$ times.

Next, suppose that we maintain the list of nodes ordered by decreasing value of $B[u][h]$, such that at each iteration h they are scanned (in the "*for all* u *in* V" loop) according to that order. Then, we would perform step i.a) as case (B) at most once, per node v and iteration h. Indeed, suppose, by way of contradiction, that we perform $B[v][h] := B[u][h-1]$ and later $B[v][h] := B[w][h-1]$, for some $u \neq w$; due to the test in step i), we have that $B[w][h-1] > B[u][h-1]$, contradicting the assumed order in which nodes are scanned. The above also implies that the test at step i.b) can fail at most once per node v and iteration h, which then implies that this step is performed $O(H \cdot N + M)$ times over all iterations.

Thus, we conclude that, under the above assumptions, step i.a) would be performed, overall, $O(M)$ times as either case (A) or (C), and $O(H \cdot N)$ times as case (B), while step i.b) would be performed, overall, $O(H \cdot N + M)$ times.

Next, we establish the penalty incurred for maintaining the ordered list of nodes and for dropping Assumption *A*. Specifically, we show the following:

Claim 1: The node ordering can be done with an overall complexity of $O((H \cdot N + M) \cdot \log N)$.

Claim 2: The test $B[u][h-1] > B[v][h]$ can be done with an overall complexity of $O(H \cdot (N^2/\log N))$.

We begin by establishing Claim 1. Consider an array $trees[\,]$ of size $H+1$, where each entry $trees[h]$ is a tree data structure. On each of the trees, each (tree) vertex t_v consists of a (bandwidth) value $b[t\_v]$ and a linked list of network nodes $list[t\_v]$. Assume, inductively, that, at the beginning of the hth iteration:

1) the tree $trees[h-1]$ is balanced according to the (bandwidth) values of its vertices;
2) the linked list $list[t\_v]$ of each of its vertices vertex t_v consists of all network nodes u with bandwidth $b[t\_v]$ at the end of iteration $h-1$, i.e., all nodes u for which $B[u][h-1] = b[t\_v]$;
3) for each network node u there is a pointer to its location in $trees[h-1]$.

Note that the tree $trees[h-1]$ consists of $O(N)$ elements, which can be scanned in linear time by decreasing bandwidth values. At the beginning of iteration h, the tree $trees[h-1]$ is copied into a new tree $trees[h]$, and, for each network node, a pointer to its location in $trees[h]$ is created; this operation incurs $O(N)$ steps. During iteration h, we scan the nodes according to $trees[h-1]$ and by decreasing bandwidth value. Upon every change performed in step i.a), we need to update $trees[h]$ and keep it balanced. Each such operation incurs $O(\log N)$ steps. The number of such changes is as follows.

- A change according to case (B) happens at most once per network node per iteration. Thus, we have an overall complexity of $O(H \cdot N \cdot \log N)$.
- A change according to case (C) happens at most once per edge, over all iterations. Thus, we have an overall complexity of $O(M \cdot \log N)$.

Hence, the overall complexity for maintaining the node ordering is indeed $O((H \cdot N + M) \cdot \log N)$.

Next, we proceed to establish Claim 2. Suppose that, for each node $u \in \mathcal{V}$, there is a (dynamic) N-bits array N_u which indicates which nodes are ("active") neighbors of u. We shall also keep an array of $O(N/\log N)$ words of size $\log N$ bits each, which together constitute N bits; each of these N bits serves as a flag for one of the nodes, which signals whether that node should still be considered within a certain iteration (such a node shall be said to be "in the forward direction"). At the beginning of an iteration, all flags are initialized to T, incurring an overall complexity of $O(H \cdot (N/\log N))$.[6] When, at an iteration h, we begin handling a new tree vertex t_v in $trees[h-1]$, we scan all tree vertices t_w in $trees[h]$ for which $b[t\_w] \geq b[t\_v]$ and reset (to F) the flags of all nodes in their lists $list[t\_w]$, incurring an overall complexity of $O(H \cdot N)$. Then, for each of the nodes u in $list[t\_v]$, we perform a bitwise AND between the $O(N)$ bits in the neighbors array N_u and the $O(N)$ flag bits. Denote the resulting N flags as $forward\_neighbor[v]$, for all $v \in \mathcal{V}$. The production of the array $forward\_neighbor[\cdot]$ constitutes the "test" in Claim 2, and it incurs an overall complexity of $O(H \cdot N \cdot (N/\log N)) = O(H \cdot (N^2/\log N))$, thus, establishing Claim 2.

While we have shown how we can efficiently identify the "forward" neighbors through the flag arrays

[6]Note that operations on N bits incur $O(N/\log N)$ complexity.

$forward\_neighbor[\cdot]$, it still remains to be shown how they can be efficiently accessed, i.e., their identities reconstructed, through these flags. This will be done by establishing the following claim.

Claim 3: When scanning a node u at some iteration h, the identities of its forward neighbors can be reconstructed in $O(K(u, h)+(N/\log N))$ steps, where $K(u, h)$ is the number of forward neighbors of node u at the hth iteration.

The above claim indicates that accessing the forward neighbors requires $O(1)$ steps per such neighbor, plus $O(H \cdot (N^2/\log N))$ over all scanned nodes and iterations. While the first term forms part of the $O(1)$ iterations required for dealing with a forward neighbor, the second term is equal to that of Claim 2. Hence, Claim 3 implies that accessing the forward neighbors incurs no additional computational complexity. We proceed to establish this. To that end, we maintain a (precomputed) table $T[\cdot]$ of size $O(N)$, in which each row consists of a linked list of the values of the positions in which the address of that row has "1" bits. For example, for $N = 4$, $T[\cdot]$ has a null list in the first row, the list "1" in the second row, the list "2" in the third row, and the list "1,2" in the fourth row. We split $forward\_neighbor[\cdot]$ into $O(N/\log N)$ words of size $\log N$ bits each, and use each word as a pointer to the table $T[\cdot]$, in the following way: when the kth ($\log N$-bits long) word of $forward\_neighbor[\cdot]$ is considered, the identity of each forward neighbor is obtained by adding the offset $(k-1)\cdot\log N$ to each value obtained from the corresponding linked list in $T[\cdot]$. Thus, we access the table $O(N/\log N)$ times, and, in addition, perform $O(1)$ operations for reconstructing the identity of each forward neighbor, hence, establishing Claim 3.

Thus, we conclude:

Theorem 2: For bottleneck metrics, there is a solution[7] which solves the AHOP problem with maximal hop count H within $O(M \log N + H \cdot (N^2/\log N))$ [hence, at most $O(N^3/\log N)$] operations.

Proof: We have $O(H \cdot (N^2/\log N))$ for the test at step i) and $O(H \cdot N + M)$ for steps i.a) and i.b) except for the maintenance of $trees[]$, which incurs $O((H \cdot N + M) \cdot \log N)$. □

In the Appendix, we make the ideas that led to Theorem 2 more concrete by providing a pseudocode specification of the algorithm.

V. Conclusion

In this paper, we introduced the class of AHOP problems, which, given a path-weight function, seek an optimal hop-constrained path for all possible hop values. This problem was motivated by computational cost considerations (precomputation) in QoS routing problems. The contributions of this paper are threefold. First, we formulated the problem in a general framework of edge and path weights, and showed that, for additive weights, $\Omega(N^3)$ is a lower bound for the (worst case) computational complexity of any path-comparison-based solution to the problem. This result implies that the standard BF shortest path algorithm is optimal in terms of worst case computational complexity. Second, we relied on a proof technique presented in [11] to establish the result, and we feel that illustrating how this technique can be used in the context of networking problems is useful in itself. Third, we showed that in the important case of bottleneck weights, such as bandwidth, a more efficient solution existed. In particular, we presented an algorithm (Algorithm IBF, given in the Appendix) that solved this instance of the AHOP problem with $O(M \log N + H \cdot (N^2/\log N)) \leq O(N^3/\log N)$ complexity.

The main significance of Algorithm IBF is in indicating that the lower bound does not necessarily apply to all metrics, and, in particular, not to bottleneck metrics. However, in practice, the basic BF algorithm may still be appropriate for communication networks, as their topology is typically sparse. Indeed, for sparse topologies, the complexity of the basic BF algorithm, $O(H \cdot M)$, is below $O(H \cdot (N^2/\log N))$, while that of Algorithm IBF remains unaffected, i.e., it is still $\Theta(H \cdot (N^2/\log N))$. Therefore, a potential topic for future research is to devise a solution for the AHOP problem under bottleneck metrics which would improve upon the basic BF solution in the case of sparse topologies; see [15] for related results. More generally, it would be interesting to obtain a lower bound for the AHOP problem under bottleneck metrics, in both cases of general and sparse topologies, and, in addition, to obtain a tight, i.e., optimal, solution with respect to that bound. Another potentially interesting extension of our study is to generalize the AHOP problem to include not only all possible hop counts and destinations, but also all possible sources. This is motivated by the fact that a *route server* would be faced with such a task. Route servers are potentially important in practice, e.g., see [16], as they allow the deployment of new QoS routing algorithms with minimal impact, i.e., upgrade requirements, on the existing infrastructure.

On a more general level, the results obtained for IBF indicate that the AHOP problem is fundamentally simpler for bottleneck weights than for additive weights. It is well known that additive metrics are often more difficult to handle than bottleneck metrics; for example, the restricted shortest path problem, which considers two link weights, is NP-hard when both weights are additive [4], but solvable in polynomial time when at least one is bottleneck. On the other hand, standard shortest path problems (e.g., finding a path of minimum delay or finding a path of maximum bandwidth) can usually be solved using the same algorithmic scheme, incurring the same complexity. However, our results indicate that there are path optimization problems, such as AHOP, which are solvable for the two types of metrics, yet still fundamentally easier for bottleneck metrics.

Finally, our study has several practical implications. One is to provide designers of new routing (in particular, QoS routing) protocols with a greater confidence in the fact that using the BF scheme is an adequate solution, at least when additive metrics are involved. Another important implication is the additional evidence on the advantage of substituting additive metrics with bottleneck metrics. For example, certain (rate-based) schedulers often enable mapping delay constraints onto rate constraints (see, e.g., [17], [5], [6], [18]–[20]), hence, our results may provide additional support for their deployment.

[7]Algorithm Improved Bellman–Ford (IBF), specified in the Appendix.

APPENDIX
PSEUDOCODE OF ALGORITHM IBF

This Appendix provides a pseudocode specification of the algorithm that led to Theorem 2.

Algorithm IBF:

```
Input
  V = set of vertices, labeled by integers
    1 to N.
  E = set of M edges, labeled by ordered
      pairs (n, m) of vertex labels.
  s = source vertex.
  For all edges (u, v) in E:
    b(u, v) = the bandwidth of edge (u, v).
  H = maximum hop count.

Variables

  neighbors[1,...,N][1,...,N]: adjacency matrix
    (assumed to be initialized according
    to topology).

  B[1,...,N][0,...,H]: the maximal bandwidth
    computed for each node at each
    iteration.

  trees[0,...,H]: an array of (balanced)
   trees.
    Each tree vertex t_v consists of a
    (bandwidth) value b[t_v] and a linked
    list of nodes list[t_v].

  tree-pointer[0,...,H][1,...,N]: two-dimensional
    array of pointers, where tree-pointer[h][u]
    points to the location of node u in
    trees[h].

  forward[1,...,N]: array of N flags; shows
    for each node whether it is still in
    the forward direction.

  forward_neighbor[1,...,N]: array of N flags;
    shows for each node whether it is a
    neighbor of the currently scanned node
    that is in the forward direction.

begin

  B[s][] := infinity;
    /* by "X[]" we denote all entries in an
    array X */
  For all nodes u <> s do B[u][] := 0;
  Create a balanced tree trees[0] according
  to the values B[][0];
  Set the pointers tree-pointer[0][] accord-
  ingly;

for h := 1 to H do
begin
  B[][h] := B[][h - 1];
  trees[h] := trees[h - 1];
  tree-pointer[h][] := tree-pointer[h - 1][];
  forward[] := T;
  consider all vertices in trees[h - 1] and
  trees[h] as "unscanned";
  while there are unscanned vertices in
  trees[h - 1] do
  begin;
    let t_v be the next (by decreasing
    bandwidth value) unscanned vertex in
  trees[h - 1];
  mark t_v as "scanned";
  while there are unscanned vertices
  t_w in trees[h] with b[t_w] >= b[t_v] do
  begin;
    let t_w be the next (by decreasing
      bandwidth value) unscanned vertex
      in trees[h];
      mark t_w as "scanned";
      for all nodes u in list[t_w] do
      forward[u] := F;
    end;
    for all network node u in list[t_v] do
      for all v in V do
        forward_neighbor[v] :=
          Neighbors[u][v] and forward[v];
    Reproduce_List_of_Neighbors;
      /* As explained, this can be done
          in O(N/(log N) + F), where F is
          the number of forward neighbors,
          i.e., "1" bits in
          forward_neighbor[]. */
    for all forward neighbors v do
    begin
      if b(u, v) > B[v][h] then
      begin
        B[v][h] := min(B[u][h - 1], b(u, v));
        move node v to a vertex t_w in
        trees[h] for which b[t_w] = B[v][h];
        update tree-pointer[h][v];
        if B[v][h] = B[u][h - 1] then
          Forward[v] := F;
      end;
      if B[v][h] >= b(u, v) then
        Neighbors[u][v] := F;
      end;
    end;
  end;
end.
```

REFERENCES

[1] S. Chen and K. Nahrstedt, "An overview of quality-of-service routing for next-generation high-speed networks: Problems and solutions," *IEEE Network Mag.*, vol. 12, pp. 64–79, Nov./Dec. 1998.

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Networks Flows*. Englewood Cliffs, NJ: Prentice-Hall, 1993.
[3] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Cambridge, MA: MIT Press, 1990.
[4] M. R. Garey and D. S. Johnson, *Computers and Intractability*. San Francisco, CA: Freeman, 1979.
[5] R. Guérin and A. Orda, "QoS-based routing in networks with inaccurate information: Theory and algorithms," *IEEE/ACM Trans. Networking*, vol. 7, pp. 350–364, June 1999.
[6] A. Orda, "Routing with end to end QoS guarantees in broadband networks," *IEEE/ACM Trans. Networking*, vol. 7, pp. 365–374, June 1999.
[7] E. Crawley, R. Nair, B. Rajagopalan, and H. Sandick, "A framework for QoS-based routing in the internet," Internet Engineering Task Force (IETF), RFC (Informational) 2386, Aug. 1998.
[8] G. Apostolopoulos, R. Guérin, S. Kamat, A. Orda, T. Przygienda, and D. Williams, "QoS routing mechanisms and OSPF extensions," Internet Engineering Task Force (IETF), RFC (Experimental) 2676, Aug. 1999.
[9] G. Apostolopoulos, R. Guérin, and S. Kamat, "Implementation and performance measurements of QoS routing extensions to OSPF," in *Proc. IEEE INFOCOM*, New York, Mar. 1999, pp. 680–688.
[10] G. Apostolopoulos, R. Guérin, S. Kamat, and S. Tripathi, "Quality-of-service-based routing: A performance perspective," in *Proc. SIGCOMM*, Vancouver, BC, Canada, Sept. 1998, pp. 17–28.
[11] D. R. Karger, D. Koller, and S. J. Phillips, "Finding the hidden path: Time bounds for all-pairs shortest paths," *SIAM J. Computing*, vol. 22, no. 6, pp. 1199–1217, Dec. 1993.
[12] A. M. Frieze and G. R. Grimmet, "The shortest-path problem for graphs with random arc lengths," *Discrete Appl. Math.*, vol. 10, pp. 57–77, 1985.
[13] P. M. Spira, "A new algorithm for finding all shortest paths in a graph of positive arcs in average time $O(n^2 \log^2 n)$," *SIAM J. Computing*, vol. 2, pp. 28–32, 1973.
[14] M. L. Fredman, "New bounds on the complexity of the shortest-path problem," *SIAM J. Computing*, vol. 5, pp. 83–89, 1976.
[15] A. Orda and A. Sprintson, "QoS routing: The precomputation perspective," in *Proc. IEEE INFOCOM*, Tel Aviv, Israel, Mar. 2000, pp. 128–136.
[16] G. Apostolopoulos, R. Guérin, S. Kamat, and S. Tripathi, "Server-based QoS routing," in *Proc. GLOBECOM*, Rio de Janeiro, Brazil, Nov. 1999, pp. 762–768.
[17] L. Georgiadis, R. Guérin, V. Peris, and K. N. Sivarajan, "Efficient network QoS provisioning based on per node traffic shaping," *IEEE/ACM Trans. Networking*, vol. 4, pp. 482–501, Aug. 1996.
[18] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The multiple node case," *IEEE/ACM Trans. Networking*, vol. 2, pp. 137–150, Apr. 1994.
[19] S. Shenker, C. Partridge, and R. Guérin, "Specification of guaranteed quality of service," Internet Engineering Task Force (IETF), RFC (Proposed Standard) 2212, Sept. 1997.
[20] H. Zhang and D. Ferrari, "Rate-controlled service disciplines," *J. High Speed Netw.*, vol. 3, no. 4, pp. 389–412, 1994.

Roch Guérin (S'85–M'86–SM'91–F'01) received the Engineer degree from ENST, Paris, France, in 1983, and the M.S. and Ph.D. degrees in electrical engineering from the California Institute of Technology, Pasadena, CA, in 1984 and 1986, respectively.

He joined the Department of Electrical and System Engineering, University of Pennsylvania, Philadelphia, in 1998, where he is the Alfred Fitler Moore Professor of Telecommunications Networks. Before joining the University of Pennsylvania, he was with the IBM T. J. Watson Research Center, Yorktown Heights, NY, for over twelve years in a variety of technical and management positions. He is on the Technical Advisory Board of France Telecom and has been consulting for numerous companies in the networking area. His research has been in the general area of networking, with a recent focus on routing and traffic engineering. He is also interested in problems related to enforcement and verification of service guarantees, and in mapping network performance to user and application-level quality measures.

Dr. Guérin served as the Editor of the ACM SIGCOMM technical newsletter (CCR) from 1998 to 2001. He has been an Editor for the *Journal of Computer Networks*, the *IEEE Communications Surveys*, the IEEE/ACM TRANSACTIONS ON NETWORKING, the IEEE TRANSACTIONS ON COMMUNICATIONS, and the *IEEE Communications Magazine*. He was Guest Editor of an IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS issue on Internet QoS published in December 2000, and chaired the IEEE Technical Committee on Computer Communications from 1997 to 1999. He was elected a Member-at-Large of the Board of Governors of the IEEE Communications Society in 1999. He served as General Chair of the IEEE INFOCOM'98 Conference, and as Technical Program Co-Chair of the ACM SIGCOMM'2001 Conference. In 1994, he received an IBM Outstanding Innovation Award for his work on traffic management in the BroadBand Services Network Architecture.

Ariel Orda (S'94–M'92–SM'97) received the B.Sc. (*summa cum laude*), M.Sc., and D.Sc. degrees in electrical engineering from the Technion–Israel Institute of Technology, Haifa, Israel, in 1983, 1985, and 1991, respectively.

Since 1994, he has been with the Department of Electrical Engineering, Technion, where he is currently an Associate Professor and the Academic Head of the Computer Networking Laboratory. During 1993–1994, he was with the Center for Telecommunication Research, Columbia University, New York, NY, as a Visiting Scientist, and with AT&T Bell Laboratories, Murray Hill, NJ, as a Scientific Consultant. During the summers of 1992 and 1995–1999, he held visiting positions at AT&T Bell Laboratories, IBM T. J. Watson Research Center, Hawthorne, NY, and Bell Laboratories–Lucent Technologies, Holmdel, NJ. Since 1991, he has held several consulting positions with Israeli industry. His current research interests include QoS provisioning and routing, the application of game theory to computer networking, network pricing, and distributed network algorithms.

Dr. Orda received the Award of the Chief Scientist in the Ministry of Communication in Israel, a Gutwirth Award for outstanding distinction, the Research Award of the Association of Computer and Electronic Industries in Israel, and the Jacknow Award for Excellence in Teaching. He is an Editor of the IEEE/ACM TRANSACTIONS ON NETWORKING and served as Technical Program Co-Chair of the IEEE INFOCOM 2002 Conference.