Available online at www.sciencedirect.com
SCIENCE DIRECT®

Computer Communications 29 (2006) 392–401

computer communications

www.elsevier.com/locate/comcom

Finding a least hop(s) path subject to multiple additive constraints

Gang Cheng, Nirwan Ansari*

Authors are with the Advanced Networking Laboratory, ECE Department, NJIT, Newark, NJ 07012, USA

Received 1 November 2004; revised 9 May 2005; accepted 10 May 2005
Available online 13 June 2005

Abstract

In this paper, for the purpose of saving network resources, we first introduce and investigate a new problem referred to as the least hop(s) multiple additively constrained path (LHMACP) selection, which is NP-complete. Then, we propose the k-shortest paths Extended Bellman-Ford (k-EB) algorithm, which is capable of computing All Hops k-shortest Paths (AHKP) between a source and a destination. Through extensive analysis and simulations, we show that the heuristic algorithm, based on k-EB, is highly effective in finding a least hop path subject to multiple additive constraints with very low computational complexity; it achieves near 100% success ratio in finding a feasible path while minimizing its average hop count.
© 2005 Elsevier B.V. All rights reserved.

Keywords: Multiple additively constrained QoS routing; Cost function; NP-complete; Least hop(s)

1. Introduction

The tremendous growth of the global Internet has given rise to a variety of applications that require quality-of-service (QoS) beyond what is provided by the current best-effort IP packet delivery service. One of the challenging issues is to select feasible paths that satisfy different quality-of-service (QoS) requirements. This problem is known as QoS routing. QoS requirements are diverse, subject to demands of different applications. Bandwidth, delay, delay jitter, and loss ratio are the commonly required QoS metrics. These requirements can be classified into three types [1]: concave, additive, and multiplicative. Since the concave type can be easily pruned by selecting the bottleneck, and the multiplicative type can be converted into the additive constraints by the logarithmic operation, the constraints considered in this paper are additive, unless otherwise mentioned. In general, state distribution and routing strategy are the two issues related to QoS routing. State distribution addresses the issue of exchanging the state information throughout the network [2]. Routing strategy is used to find a feasible path meeting the QoS requirements. According to how the state information is aintained and how the search is carried out, routing strategy can be further ivided into three categories [1]: source routing, distributed routing, and hierarchical routing. In this paper, we focus on source routing, and assume that accurate network state information is available to each node. A number of research works have also addressed inaccurate information [3–6], which is, however, beyond the scope of this paper.

Since multiple additively constrained QoS routing has been proved to be NP-complete [7], tackling this problem requires heuristics. The limited path heuristic proposed by Yuan [8] maintains a limited number of candidate paths, say x, at each hop. The computational complexity is $O(x2\ \mathrm{nm})$ for the Extended Bellman-Ford algorithm for two constraints, where m and n are the number of links and nodes, respectively. For the purpose of improving the response time and reducing the computation load on a network, precomputation-based methods [9] have been proposed. Korkmaz and Krunz [10] provided a heuristic with the computational complexity compatible to that of the Dijkstra algorithm in finding the least cost path subject to multiple constraints. An algorithm [11], called A*Prune, is capable of locating multiple shortest feasible paths from the maintained heap in which all candidate paths are stored. The computational complexity of A*Prune is $O(Qn(M+h+\log \mathrm{Q}))$, where Q, M, and h are the number of paths in the heap, the number of constraints, and the maximum number of hops of computed paths, respectively. For the case

* Corresponding author. Tel./fax: +1 973 596 3670.
E-mail address: ansari@njit.edu (N. Ansari).

0140-3664/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2005.05.009

that only inaccurate link state information is available to nodes, approximate solutions [12] have been provided for the Most Probable Bandwidth Delay Constrained Path (MP-BDCP) selection problem by decomposing it into two sub-problems: the Most Probable Delay Constrained Path (MP-DCP) and the Most Probable Bandwidth Constrained Path (MP-BCP). In [13], a heuristic algorithm was proposed based on a linear cost function for two additive constraints; this is an *MCP* (*Multiple Constrained Path Selection*) problem with two additive constraints. A binary search strategy for finding the appropriate value of β in the linear cost function $w_1(p)+\beta w_2(p)$ or $\beta w_1(p)+w_2(p)$, where $w_i(p)$ $(i=1,2)$ are the two respective weights of the path p, was proposed, and a hierarchical Dijkstra algorithm was introduced to find the path. It was shown that the worst-case complexity of the algorithm is $O(\log B(m+n \log n))$, where B is the upper bound of the parameter β. The authors in [14] simplified the multiple constrained QoS routing problem into the shortest path selection problem, in which the Weighted Fair Queuing (*WFQ*) service discipline is assumed. Hence, this routing algorithm cannot be applied to networks where other service disciplines are employed. Similar to [13], LAgrange Relaxation based Aggregated Cost (*LARAC*) was proposed in [15] for the Delay Constrained Least Cost path problem (*DCLC*). This algorithm is based on a linear cost function $c_\lambda=c+\lambda d$, where c denotes the cost, d the delay, and λ an adjustable parameter. It differs from [13] on how λ is defined: λ is computed by Lagrange Relaxation instead of the binary search. It was shown that the computational complexity of this algorithm is $O(m^2\log^4 m)$. However, in [16], for the same problem (*DCLC*), a non-linear cost function was proposed. Many researchers have posed the QoS routing problem as the *k*-shortest path problem [17,18]. The authors in [19] proposed an algorithm, called TAMCRA, for MCP by using a non-linear cost function and a *k*-shortest path algorithm. The computational complexity of TAMCRA is $O(kn \log (kn)+k^3mM)$, where k is the number of shortest paths. To solve the delay-cost-constrained routing problem, Chen and Nahrstedt [20] proposed an algorithm, which maps each constraint from a positive real number to a positive integer. By doing so, the mapping offers a 'coarser resolution' of the original problem, and the positive integer is used as an index in the algorithm. The computational complexity is reduced to pseudo-polynomial time, and the performance of the algorithm can be improved by adjusting a parameter, but with a larger overhead. It was shown that only in specially constructed graphs with link weights carefully chosen, NP-complete behavior of QoS routing emerges [21]. Hence, the authors believed that the worst-case behavior (NP-complete) is very unlikely to occur in practical networks and QoS routing is feasible in practice.

Many (ε-approximation algorithms (the solution has a cost within a factor of $(1+\varepsilon)$ of the optimal one) subject to DCLC have been proposed in the literature. Lorenz and Orda [22] presented several ε-approximation solutions for both the DCLC and the multicast tree, in which the one subject to DCLC possesses the best-known computational complexity of $O(nm \log n(\log n)+mn/\varepsilon)$. Hassin [23] presented two ε-approximations algorithms for the Restricted Shortest Path problem (RSP) with complexities of $O((mn/\varepsilon)\log\log U$ and $O(|E|n^2\log(n/\varepsilon)/\varepsilon)$, respectively, where U is the upper bound of the cost of the path computed.

It can be observed from the above review that existing solutions suffer either high computational complexities or low success ratio in finding a feasible path. In this paper, based on a novel solution to All Hops k-shortest Paths selection (AHKP), we propose a high performance routing algorithm for finding the Least Hop(s) Multiple Additive Constrained Path (LHMACP). By extensive simulations, we show that our proposed algorithm not only achieves near 100% success ratio in finding a feasible path, but also essentially minimizes the average number of hops of the computed feasible paths. As a result, network resources can be saved with our proposed routing algorithm.

The rest of the paper is organized as follows. The problem is formulated in Section 2. The k-shortest paths Extended Bellman-Ford (k-EB) algorithm, which is capable of computing all hops k-shortest paths between a source and a destination, is proposed in Section 3. Based on k-EB, we present a high performance heuristic algorithm which achieves a high success ratio in finding the least hop(s) multiple additively constrained paths in Section 4. In Section 5, simulation results are presented. Finally, concluding remarks are given in Section 6.

2. Problem formulation

Since we only consider additive constraints in this paper, without loss of generality, the problem is formulated as follows:

Definition 1. *Least Hop(s) Multiple Additively Constrained Path Selection* (*LHMACP*): Assume a network is modeled as a directed graph $G(N,E)$, where N is the set of all nodes and E is the set of all links. Each link connected from node u to v, denoted by $e_{u,v}=(u,v)\in E$, is associated with M additive parameters: $w_i(u,v)\geq 0$, $i=1,2,\ldots,M$. Given a set of constraints $(c_1,c_2,\ldots,c_M)$ and a pair of nodes s and t, LHMACP is to find a least hop(s) path p from s to t subject to $W_i(p)=\sum_{e_{u,v}\in p} w_i(u,v)<c_i$, for all $i=1,2,\ldots,M$.

Definition 2. Any path p from s to t that meets the requirement, $W_i(p)=\sum_{e_{u,v}\in p} w_i(u,v)<c_i$, for all $i=1,2,\ldots,M$, is a feasible path.

We denote p_1+p_2 as the concatenation of two paths p_1 and p_2, and $c(p)$ as the cost of path p. Note that, given two paths, p_1 and p_2, and their costs, if the cost of a path p is defined as $c(p)=f(W_1(p), W_2(p),\ldots,W_m(p))$, where $f(\cdot)$ is a cost function, the computational complexity of computing the cost of p_1+p_2, $c(p_1+p_2)=f(W_1(p_1)+W_1(p_2),$

$W_2(p_1)+W_2(p_2),\ldots,W_M(p_1)+W_M(p_2))$, is $O(M)$, while it is 0(1) ($c(p_1+p_2)=c(p1)+c(p_2)$) if the cost of a path is defined as the sum of its link costs. Hence, we adopt the latter definition of the cost of a path for the sake of the computational complexity. The *least cost* path is also referred to as the *shortest* path in this paper.

3. *k*-Shortest paths extended bellman-ford algorithm

In this section, we will propose the *k*-shortest paths Extended Bellman-Ford algorithm (k-EB), which is capable of computing all hops *k*-shortest paths between a source and a destination. AHKP was first introduced in [24]. For completeness, its definition is provided below.

Definition 3. All Hops *k*-shortest Paths (AHKP) Problem: For a given source node $s\in N$ and maximal hop count H, $H(n$, find, for each hop count value h, $1\le h\le H$, and a destination node $u\in N$, the k least cost h-hop paths from s to u.

Given a network shown as Fig. 1, $k=1$ and $H=3$, by the above definition, we can compute the all hops shortest paths from the node 1 to node 4 as (1, 4), (1, 2, 4), and (1, 2, 3, 4), respectively, when h equals to 1, 2, and 3, where $(a_1,a_2,\ldots,a_h)$ represent a $(h-1)$-hop path from a_1 to a_h that sequentially traverses nodes $a_1,a_2,\ldots,a_h$. Intuitively, given any path, the predecessor node of the destination must be one of its neighboring node(s). Hence, for any hop count value h, $2\le h\le H$, the k least cost h-hop paths from s to a node must be in the set of the paths constructed by concatenating the k least cost $(h-1)$-hop paths from s to its neighboring nodes and the corresponding links. For instance, as shown in Fig. 2, assume the neighboring nodes of node d are nodes a, b, and c; the 3 least cost h-hop paths from s to d, $p_1^h(s,d)$, $p_2^h(s,d)$, and $p_3^h(s,d)$, must be included in the set of paths $\{\ \{p_g^{h-1}(s,u)+e(d,u),\ \ u=a,b,c$ and $g=1,2,3\}, p_g^{h-1}(s,u)+e(d,u)$, and where $p_g^h(s,i)$ represents the gth least cost h-hop path from s to i.

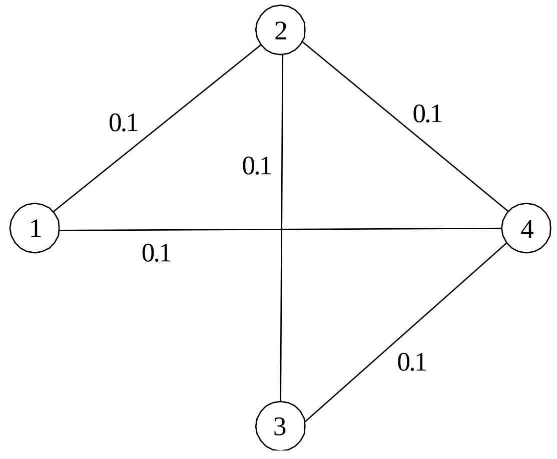


Fig. 1. A 4-nodes network.

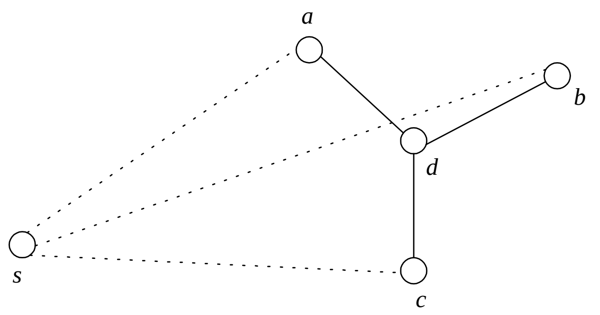


Fig. 2. Node d has three neighboring nodes, a, b, and c.

As shown in Fig. 3, denote d_i as the degree of node i, and n_1^i, $n_2^i,\ldots,n_{d_i}^i$, as its neighboring nodes, and assume there exists a virtual link $\hat{e}(s,i)$ between the source node s and node i, whose cost is infinity. We can compute the k least cost h-hop paths from $s$to i, $p_1^h(s,i)$, $p_2^h(s,i),\ldots$, $p_k^h(s,i)$, as follows:

1. $\forall i\in Np_1^1(s,i)=e(s,i)$ and $p_g^1(s,i)=\hat{e}(s,i)$, $g=2,3,\ldots,k$, (if, in reality, no link between the source s and node i exists, $p_1^1(s,i)=\hat{e}(s,i)$).
2. $p_i^h(s,i)$, $p_2^h,\ldots,p_k^h(s,i)$ represent the k shortest h-hop paths among the paths $p_g^{h-1}(s,n_d^i)+e(n_d^i,i)$, $1\le d\le d_i$, $1\le g\le k$. If, in reality, the total number of h-hop paths from s to i is less than k, we assume that there exist virtual h-hop paths whose costs are infinity.

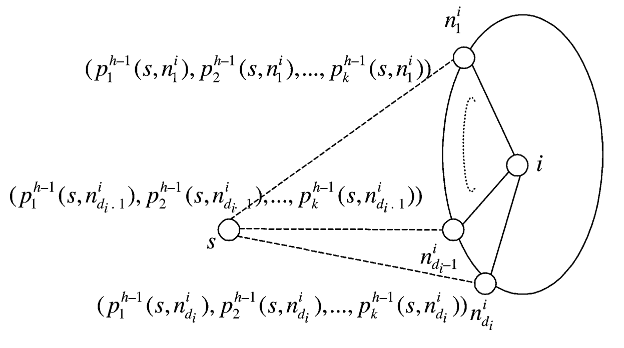
Next, we theoretically prove that $p_1^h(s,i)$, $p_2^h(s,i),\ldots$, $p_k^h(s,i)$, are the h-hop k shortest paths among the all h-hop paths from source s to node i. Denote $D_{i,g}^h$ as the cost of $p_g^h(s,i)$.

[**Proposition**.] $p_1^h(s,i)$, $p_2^h(s,i),\ldots,p_k^h(s,i)$, are the h-hop k shortest paths among all the h-hop paths from source s to node i.

Proof. When $h=1$, from the definition of the initial values of $D_{i,g}^1(i\ne s)p_g^1(s,i)$, $g=1,2,\ldots,k$, are the one hop k-shortest paths from s to i.

We assume that the proposition is correct for $h=m$. We shall prove by deduction that it is true for $h=m+1$.

Assume when $h=m+1$, if $\exists\, i\ne s$ such that $p_g^{m+1}(s,i)$, $1\le g\le k$, is not one of the k-shortest paths in


n_1^i
$(p_1^{h-1}(s,n_1^i), p_2^{h-1}(s,n_1^i),\ldots, p_k^{h-1}(s,n_1^i))$
i
$(p_1^{h-1}(s,n_{d_i-1}^i), p_2^{h-1}(s,n_{d_i-1}^i),\ldots, p_k^{h-1}(s,n_{d_i-1}^i))$
s
$n_{d_i-1}^i$
$(p_1^{h-1}(s,n_{d_i}^i), p_2^{h-1}(s,n_{d_i}^i),\ldots, p_k^{h-1}(s,n_{d_i}^i))$
$n_{d_i}^i$
Fig. 3. Illustration of the neighboring nodes of node i.

all $(m+1)$-hop paths from s to i ($D_{i,g}^{m+1}$ is larger than the cost of any $(m+1)$-hop k-shortest path from s to i). Further assume that path $\hat{p}^{m+1}(s,i)$ is not one of $p_g^{m+1}(s,i)$, $g=1,2,\ldots,k$, and has a length smaller than that of $p_g^{m+1}(s,i)$. The predecessor node of node i in $\hat{p}^{m+1}(s,i)$ is j, the path from s to j in $\hat{p}^{m+1}(s,i)$ is $\hat{p}^{m}(s,j)$ (note that $\hat{p}^{m}(s,j)$ may not be one of the k-shortest m-hop paths from s to j, and by the earlier assumption, $p_g^m(s,j)$, $g=1,2,\ldots,k$, are the k-shortest m-hop paths from s to j), the cost of $\hat{p}^{m+1}(s,i)$ is c, and the cost of $\hat{p}^{m}(s,j)$ is c'. Thus,

$$c < D_{i,g}^{m+1} \tag{1}$$

If $\hat{p}^{m}(s,j)$ is one of $p_g^m(s,j)$, $g=1$, $g=1,2,\ldots,k$, or $\exists g \in \{1,2,\ldots,k\}$ such that $c' = D_{j,g}^m$, $\hat{p}^{m+1}(s,i)$ is resulted by concatenating $\hat{p}^{m}(s,i)$ with link $e(i,j)$, i.e. $\hat{p}^{m+1}(s,i)$ is one of $p_g^m(s,n_d^i)+e(n_d^i,i)$, $d=1,2,\ldots,d_i$, $g=1,2,\ldots,k$. Since $\hat{p}^{m+1}(s,i)$ is not one of $p_g^{m+1}(s,i)$, $g=1,2,\ldots,k$, and $p_g^{m+1}(s,i)$, $g=1,2,\ldots,k$, are the k-shortest paths of $p_g^m(s,n_d^i)+e(n_d^i,i)$, $d=1,2,\ldots,d_\mathrm{i}$, $g=1,2,\ldots,k$, $\forall g \in \{1,2,\ldots,k\}$,

$$c = D_j^m + c(i,j) \geq D_{i,g}^{m+1} \tag{2}$$

which contradicts (1), where D_j^m and $c(i,j)$are the costs of $\hat{p}^{m}(s,j)$ and $e(i,j)$, respectively. Hence, $\hat{p}^{m}(s,j)$ is not one of $p_g^m(s,j)$, i.e. $\forall g \in \{1,2,\ldots,k\}$,

$$c' \geq D_{j,g}^m \tag{3}$$

So, $\forall g,u \in \{1,2,\ldots,k\}$, the cost of $\hat{p}^{m+1}(s,i)$ is

$$c = c' + c(i,j) \geq D_{j,g}^m + c(i,j) \geq D_{i,g}^{m+1} \tag{4}$$

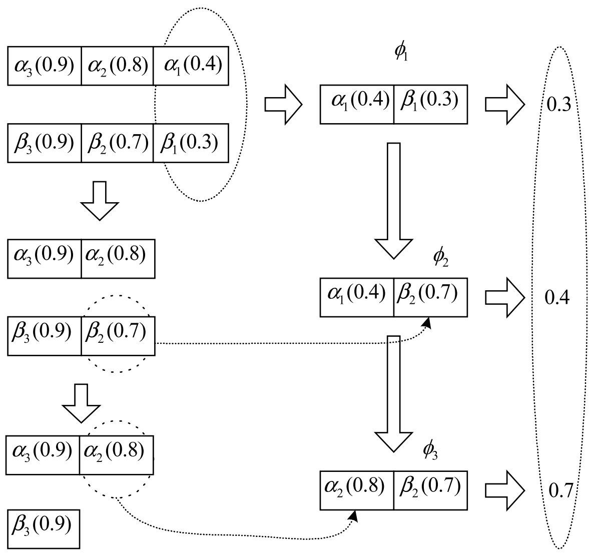
which contradicts (1). Thus, when $h=m+1$, $\forall i \in \{1,2,\ldots,N\}$, $p_g^{m+1}(s,i)$, $g=1,2,\ldots,k$, are the k-shortest paths in all $(m+1)$-hop paths from s to i.

Therefore, for any node $i \in \{1,2,\ldots,N\}$, $p_g^h(s,i)$, $g=1,2,\ldots,k$, must be the k-shortest paths in all h-hop paths from s to i. ■

For the purpose of avoiding loops, we adopt a simple method: associating paths with indicators. For example, we associate a path traversing nodes s, 1, 5, and 7 with an integer array of size n, in which the 1st, 5th, and 7th array elements are set to 1, and the rest 0. Hence, we can easily find out if node i is in the path by only checking the corresponding element′s value in the array. By this method, we can prevent loops without increasing the worst-case computational complexity. It should be noted that the adoption of this method also does not increase the worst-case memory complexity, which will be proved later.

In order to reduce the computational complexity, we introduce an algorithm, Fast Algorithm (FA), to select $p_1^h(s,i)$, $p_2^h(s,i),\ldots,p_k^h(s,i)$ from the paths $p_g^{h-1}(s,n_d^i)+e(n_d^i,i)$, $1 \leq d \leq d_i, 1 \leq g \leq k$. We first illustrate how FA works by a simple example. As shown in Fig. 4, there are two sorted path sets ($k=3$), $\{a_1,a_2a_3\}$ and $\{\beta_1,\beta_2,\beta_3\}$, whose costs are $\{0.4,0.8,0.9\}$and $\{0.3,0.7,0.9\}$, respectively. Note that

$$\min\{\min_{j=1,2,3}\{c(\alpha_j)\},\min_{j=1,2,3}\{c(\beta_j)\}\} = \min\{c(\alpha_1),c(\beta_1)\} \tag{5}$$



$\alpha_3(0.9)$ $\alpha_2(0.8)$ $\alpha_1(0.4)$
$\beta_3(0.9)$ $\beta_2(0.7)$ $\beta_1(0.3)$
ϕ_1
$\alpha_1(0.4)$ $\beta_1(0.3)$
0.3
$\alpha_3(0.9)$ $\alpha_2(0.8)$
$\beta_3(0.9)$ $\beta_2(0.7)$
ϕ_2
$\alpha_1(0.4)$ $\beta_2(0.7)$
0.4
$\alpha_3(0.9)$ $\alpha_2(0.8)$
$\beta_3(0.9)$
ϕ_3
$\alpha_2(0.8)$ $\beta_2(0.7)$
0.7

Fig. 4. The demonstration of the FA algorithm.

Let $\phi_1=\{\alpha_1,\beta_1\}$; the least cost path in the two sets is the least cost path in ϕ_1. In this example, it is β_1. Furthermore, since the two path sets are sorted by their costs and β_1 is the least cost path, the second least cost path in the two sets must be the least cost path between α_1 and β_2, i.e. let $\varphi_2=(\varphi_1 \cap \{\bar{\beta}_1\}) \cup \{\beta_2\}$, and the second least cost path in the two sets is the least cost path in ϕ_2. Similarly, the jth least cost path in the two sets is the least cost path in ϕ_j, which can be proved by deduction, where $\varphi_j=(\varphi_{j-1} \cap \{\bar{\pi}_{j-1}\}) \cup \{\nu_j\}$, π_{j-1} is the least cost path in ϕ_{j-1}, and ν_j is the next path to π_{j-1} in the corresponding set. Moreover, $1 \leq j(\nu \leq k$, $c(\pi_\mathrm{j}) \leq c(\pi_\nu)$, i.e. the paths (π_j) are sequentially computed in increasing order of their costs. As shown in Fig. 4, the output paths sequentially computed by FA are β_1, α_1, and β_2 with the corresponding costs of 0.3, 0.4, and 0.7, respectively, which are in increasing order.

Denote P_i^h as the set $\{p_g^h(s,i)1 \leq g \leq k\}$. The following FA procedure is used to compute the k least cost $(h+1)$-hop paths from the source node s to node i:

Set $P_i^{h+1}=\Phi$ and $P=\{p_1^h(s,n_d^i)+e(n_d^i,i),\quad d=1,2,\ldots,d_i\}$, where Φ is the empty set. Sort the paths in P by their costs by the Heapsort Algorithm [25]. Since the number of paths in P is d_i, the computational complexity of this step is $O(d_i\log d_i+d_i)$(the computational complexity of using the Heapsort algorithm is $O(d_i\log d_i)$).

Denote u as the number of paths in P_i^{h+1}. Initialize $u=1$.

Assume the least cost path of P is $p_u=p_v^h(s,n_d^i)+e(n_d^i,i)$, for some $d \in \{1,2,\ldots,d_i\}$ and $v \in \{1,2,\ldots,k\}$. Remove p_u from P. Check if node $i$is included in $p_v^h(s,n_d^i)$ with its associated indicator array. If so, go to Step 5. Otherwise, place it at the tail of P_i^{h+1}, i.e. $p_u^{h+1}(s,i)=p_u$. Copy the indicator array associated with $p_v^h(s,n_d^i)$ to that of $p_u^{h+1}(s,i)$ and set the value of its ith element to 1.

```
Algorithm k-EB(G, s, t)
1   for all i from 1 to N
2       set $p_1^1(i) = c(s, i)$
3   end for
4   for h = 1 to H
5       for all i from 1 to N
6           run FA algorithm with input of i and h
7       end for
8   end for
```

Fig. 5. The pseudo-code of the k-EB algorithm.

If u is equal to k, the k least cost $(h+1)$-hop paths from the source node s to node i are obtained; Stop. Otherwise, $u=u+1$.

Insert $p^h_{v+1}(s,n^i_d)+e(n^i_d,i)$ into P by the binary search [25]. Since there are currently d_i-1d_i-1 elements in P, the computational complexity of this step is $O(\log(d_i-1)+1)$. Then, go to Step 3.

P is sorted in Step 1 in order to reduce the computational complexity of Step 5. Otherwise, the computational complexity of Step 5 is $O(d_i)$, which is resulted from finding the least cost path in P. Using the FA procedure, our proposed k-EB algorithm can be illustrated by the pseudo code shown in Fig. 5.

3.1. Computational complexity

Note that Step 5 is executed $k-1$ times in FA. Hence, the computational complexity of using FA to compute the k shortest $(h+1)$-hop $(1\le h\le H-1)$ paths from the source node s to node i is

$$O(d_i\log d_i+(k-1)\log(d_i-1)+k-1+d_i)$$
$$=O(d_i\log d_i+(k-1)\log(d_i-1)) \quad (6)$$

Accordingly, the computational complexity of using FA to compute the k shortest $(h+1)$-hop paths from the source node s to all other nodes is the sum of those of computing the k shortest $(h+1)$-hop paths from the source node s to every single node, which is

$$O\left(\sum_{i=1}^{n}(d_i\log d_i+(k-1)\log(d_i-1)\right) \quad (7)$$

Moreover, it can be observed from the pseudo code of k-EB that there are H loops to compute all hops k shortest paths from the source node s to all other nodes, i.e. the computational complexity of k-EB is

$$O\left(H\left(\sum_{i=1}^{n}(d_i\log d_i+(k-1)\log(d_i-1))\right)\right)$$
$$\le O\left(n\left(\sum_{i=1}^{n}(d_i\log d_i+(k-1)\log(d_i-1))\right)\right) \quad (8)$$

If there exists a bound D on the maximum node degree, i.e. $\forall i\in N, d_i\le D$, the worst-case computational complexity of k-EB is bounded by

$$O\left(n\left(\sum_{i=1}^{n}(d_i\log d_i+(k-1)\log(d_i-1))\right)\right)$$
$$\le O(nm\log D+nk\log D)\le O(nm\log n+nk\log n) \quad (9)$$

Note that this computational complexity bound of k-EB is very loose, but it is rather low already and almost does not increase with k when $k(m$; it could be much less in reality.

3.2. Memory complexity

The memory complexity of our proposed algorithm can be divided into two parts: the memory used to record the computed paths (for the purpose of reconstructing them after computing) and the one consumed during the computing procedure (FA). Denote pre(i,h,g) as the predecessor node of i on $p^h_g(s,i)$, and count(i,h,g) as the number satisfying that $p^h_g(s,i)=p^{h-1}_{\text{count}(i,h,g)}(s,\text{pre}(i,h,g))+e(i,\text{pre}(i,h,g))$, i.e. count($i$,$h$,$g$) is the number such that $p^h_g(s,i)$ is constructed by concatenating the count(i,h,g)th shortest $(h-1)$-hop path from s to pre(i,h,g) and the link e(i, pre(i,h,g)). Hence, for any given node i, hop count h, and $1\le g\le k$, define

- $n_0=i$ and $g_0=g$.
- $n_j=\text{pre}(n_{j-1},h-j+1g_{j-1})$ and $g_j=\text{count}(n_{j-1},\ h-j+1,g_{j-1}),\ j\le h$.

It can be observed that $p^h_g(s,i)=(s,n_{h-1},n_{h-2},\ldots,n_1,i)$, i.e. all paths can be backward reconstructed as long as for any node i, hop count h, and $1\le g\le k$, pre(i,h,g) and count(i,h,g) are available, where $(s,n_{h-1},n_{h-2},\ldots,n_1,i)$ represents a path sequentially consisting of nodes s,n_{h-1},n_{h-2},…,n_1,i. Hence, for a single node, the memory cost to record all k shortest h-hop paths is $O(k)$. Since there are H hops and n nodes, the first part of the memory cost is $O(kHn)\le O(kn^2)$. As mentioned before, we use indicator arrays (of size n) to avoid loops, which contribute to the memory cost of the second part. The total memory cost used for indicator arrays is $O(kn^2)$ for all n nodes and the k shortest h-hop paths. Observe that the indicators for the h-hop paths are only used when we compute the $(h+1)$-hop paths. We can erase the indicators associated with the h-hop paths when all the $(h+1)$-hop paths are computed. Hence, the part of the memory cost resulting from the indicators is $O(kn^2)$. Combining memory costs mentioned above together, the memory complexity of our proposed algorithm as $O(kn^2)$. It can be observed that the introduction of the indicators to avoid loops does not increase the worst-case memory complexity of our proposed algorithm.

4. Proposed routing algorithm

In this section, based on k-EB, a high performance QoS routing algorithm, which can not only achieve a high success ratio in finding a feasible path but also can minimize the hops of the solutions, is proposed. The basic idea behind is that k-EB is capable of iteratively finding all hops *k*-shortest paths, i.e. k-EB computes the 1-hops *k*-shortest paths in the first iteration, the 2-hops *k*-shortest paths in the second iteration, and so on. If there are multiple feasible paths available, the one with the least number of hops is computed first. For example, given a network shown in Fig. 6, assume both paths (1-2-5) and (1-3-4-5) are feasible paths. Since path (1-2-5) has less hops than path (1-3-4-5), path (1-2-5) will be generated before path (1-3-4-5), and thus the least hops feasible path is obtained first.

Now, we introduce Theorem 1 which is used to design our proposed routing algorithm.

[**Theorem 1**.] Given a cost function $\xi(\cdot)$ such that for any $1 \le i \le M$, $(\partial^2\xi(x_1, x_2, ..., x_M))/(\partial x_i) = 0$ and $(\partial\xi(x_1, x_2, ..., x_M))/(\partial x_i) \ge 0$, no feasible path exists if the least cost path has the cost larger than $\xi(c_1, c_2, \ldots, c_M)$.

Proof. By contradiction. Assume path $\hat{p}$ satisfies the constraint $(c_1, c_2, \ldots, c_M)$ and the least cost among all paths is larger than or equal to $\xi(c_1, c_2, \ldots, c_M)$; that is,

$$c(p) \ge \xi(c_1, c_2, \ldots, c_M), \forall p \Rightarrow c(\hat{p}) \ge \xi(c_1, c_2, \ldots, c_M) \quad (10)$$

Also, since $\xi(\cdot)$ is linear,

$$\xi(w_1(\hat{p}), w_2(\hat{p}), \ldots, w_M(\hat{p})) = c(\hat{p}) \quad (11)$$

Thus,

$$\xi(w_1(\hat{p}), w_2(\hat{p}), \ldots, w_M(\hat{p})) \ge \xi(c_1, c_2, \ldots, c_M) \quad (12)$$

However, since $(\partial\xi(x_1, x_2, ..., x_M))/(\partial x_i) \ge 0$ and path $\hat{p}$ satisfies the constraint $(c_1, c_2, \ldots, c_M)$,

$$w_i(\hat{p}) < c_i, \forall i \in \{1, 2, \ldots, M\},$$

$$\Rightarrow \xi(w_1(\hat{p}), w_2(\hat{p}), \ldots, w_M(\hat{p})) < \xi(c_1, c_2, \ldots, c_M) \quad (13)$$

which contradicts (12), and thus Theorem 1 is proved.■

Similarly, we can prove that a path can be a feasible path if and only if its cost is less than $\xi(c_1, c_2, \ldots, c_M)$. We divide our routing algorithm into two parts: forward k-EB and backward k-EB. We search for a feasible path from the source to the destination using the forward k-EB, and reverse the search by the backward k-EB. Therefore, the computational complexity of our proposed routing algorithm is just twice that of k-EB. The cost functions used in both searches should be different in order to avoid repeating the same path while speeding up the search. We adopt the following simple but effective cost function for the forward k-EB:

$$f(x_1, x_2, \ldots, x_M) = \sum_{i=1}^{M} \frac{x_i}{c_i} \quad (14)$$

Assume the shortest path *p* found by the forward k-EB is not a feasible path and $\exists i \in \{1, 2, \ldots, M\}$ such that $w_i(p) > c_i$. By Theorem 1, the second search (backward k-EB) is executed only when the cost of *p* is less than $f(c_1, c_2, \ldots, c_M)$. Since we already know that the least cost path *p* of the first search (forward EB) is not a feasible path, the cost function should be adjusted for the second search such that

- If a feasible path does exist, *p* should not be the least cost path computed by the second search;
- If *p* is the least cost path of the second search, $f'(w_1(\hat{p}), w_2(\hat{p}), ..., w_M(\hat{p})) \ge f'(c_1, c_2, \ldots, c_M)$ so that Theorem 1 can be invoked, where $f(\cdot)$ is the cost function for the backward k-EB.

Based on Eq. (14), the cost function for the backward k-EB is defined as:

$$f'(x_1, x_2, \ldots, x_M)$$

$$= \sum_{j=1, j \ne i}^{M} \frac{x_j}{c_j} + \left(\frac{f(c_1, c_2, \ldots, c_M) - c(p)}{w_i(p) - c_i} + \frac{1}{c_i}\right) x_i$$

$$= \sum_{j=1}^{M} \frac{x_j}{c_j} + \left(\frac{f(c_1, c_2, \ldots, c_M) - c(p)}{w_i(p) - c_i}\right) x_i \quad (15)$$

Fig. 7 shows the pseudo-code of our proposed QoS routing algorithm, referred to as Bk-EB (Bi-directional *k*-shortest path Extended Bellman-Ford). The key properties that distinguish Bk-EB from previously proposed algorithms are:

- Intuitively, the more links (hops) on a path, the more network resources are consumed. Hence, minimizing the length or hops of a feasible path is preferred. Based on k-EB, our algorithm can essentially minimize the hops of the feasible path.
- Assume the link weights are randomly distributed, and define $P_r\{W_1(p) \le c_1, W_2(p) \le c_2, \ldots, W_M(p) \le c_M | c(p) = \alpha, H(p) = n\}$ as the probability that a path *p* is a feasible path with $c(p) = \alpha$, and its hop count, $H(p) = n$. The probability of the shortest path to be a feasible path may

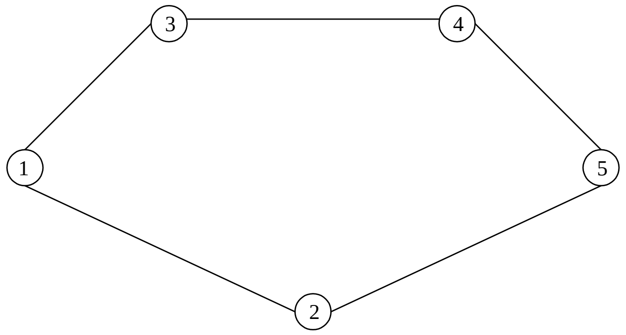


Fig. 6. A network consisting of 5 nodes.

```
Algorithm Bk-EB(G, s, t, c)
1   if k-EB(G, s, t, LeastCost) = SUCCESS
2       return SUCCESS
3   else
4       if LeastCost > $f(c_1, c_2, \ldots, c_M)$
5           return No Feasible Path Exists    // no feasible path exists
6       else
7           Compute New Cost Function $f^1(.)$
8           if k-EB(G, s, t, LeastCost ) = SUCCESS
9               return SUCCESS
10          else
11              if LeastCost > $f^1( c_1, c_2, \ldots, c_M)$
12                  return No Feasible Path Exists
13              end if
14          end if
15      end if
16  end if
17  return FAIL               // fail to find a feasible path
```

Fig. 7. The pseudo-code of the Bk-EB algorithm.

not be the largest in all possible paths. Note that, instead of computing only the shortest path, k-EB finds all hops k shortest paths from a source to a destination that increases the probability of finding a feasible path. In order to reduce the runtime, we stop the search whenever a feasible path is found.

5. Simulations

We evaluate the performance of our proposed routing algorithm (Bk-EB) by comparing it with the Binary Search Algorithm (BSA) [13], H_MCOP [10], and TAMCRA [19]. Note that H_MCOP was originally designed to solve the multiple constrained optimal path selection problem. It can also be used to solve the LHMACP problem by setting the cost of each link to 1. For comparison purposes, we adopt two performance indices, Success Ratio (SR) and Average Hop Ratio (AHR), where SR is defined below:

$$\mathrm{SR} = \frac{\text{Total number of success requests of the algorithm}}{\text{Total number of success requests of the optimal algorithm}} \tag{16}$$

The algorithm that can always locate a feasible path as long as it exists is refereed to as the optimal algorithm. Here, it is achieved simply by flooding which is rather exhaustive. We do not simply adopt the average hop of computed feasible paths as one performance index because it may introduce unfairness in comparison. For example, given a network as shown in Fig. 6 and a set of constraints, we conduct two searches (from node 1 to node 5) with two routing algorithms, α and β (the link QoS metrics are different in the two searches). In the first search, both algorithms locate a 2-hop feasible path (1-2-5), while in the second search, the algorithm α fails to find a feasible path, but algorithm β does (path 1-3-4-5). Obviously, algorithm β performs better than α in the simulation. However, if the average hop of the computed feasible path is adopted as the only performance index (the average hop of the feasible path computed by algorithm α in the two searches is 2, while it is 2.5 for algorithm β), it turns out that algorithm α outperforms β. Note that the optimal algorithm is achieved by hop-by-hop flooding, and it can always locate the least hop feasible path as long as a feasible path exists. Therefore, its average hop is the lower bound of all feasible paths. Furthermore, given any feasible path p, there must exist a corresponding optimal one (the least hop feasible path, which could be p) that has the same source and destination as p. Hence, instead of using the average hop of computed feasible paths as a performance index, we adopt the Average Hop Ratio (AHR), where AHR is defined as the ratio between the average number of hops of the computed feasible paths and that of the corresponding optimal ones, i.e.

$$\mathrm{AHR} = \frac{\text{Average hop of the computed paths of the algorithm}}{\text{Average hop of corresponding optimal paths}} \tag{17}$$

In addition to the 32-node network [20], two larger networks with 50 and 100 nodes, respectively, generated by using the Doar′s model [26] are used to conduct the simulations for comparison purposes. We only set $k=1,2$ for Bk-EB in our simulations because the success ratio of Bk-EB already approaches 100% for $k=2$. In all simulations, the link weights are independent and uniformly distributed from 0 to 1, and all data are obtained by running 100,000 requests. We adopt two constraints in the simulations, and set them

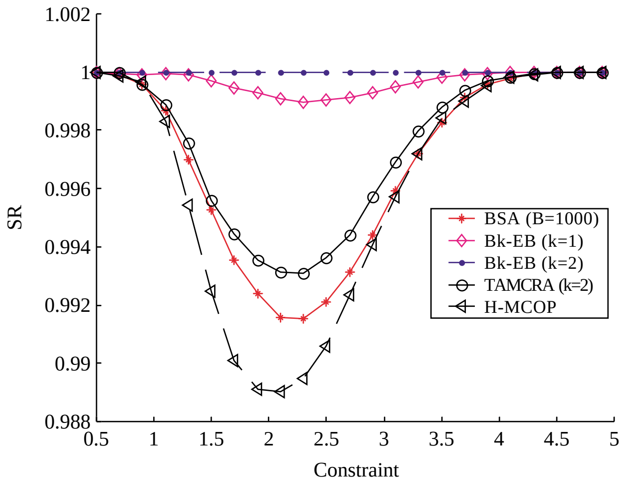


Fig. 8. SR in the 32-node network.

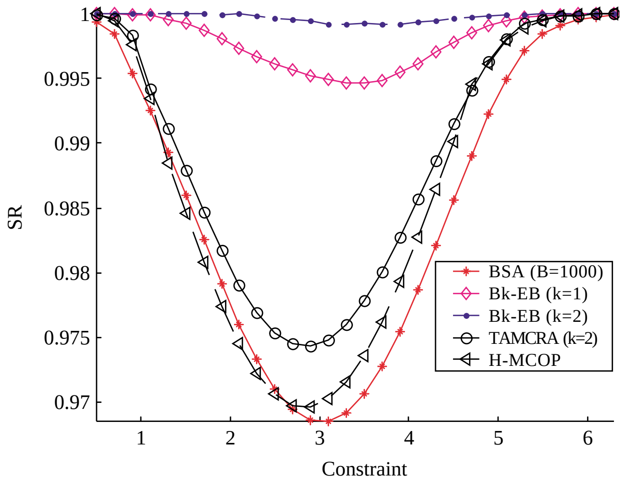


Fig. 10. SR in the 100-node network.

equal to each other. The constraints are increased from 0.5 to 6 with a step size of 0.2.

Figs. 8–10 illustrate the SR of different algorithms, in which the x-axis represents the value of constraints (two constraints are set equal to each other). Note that the success ratio of Bk-EB ($k=2$) in finding a feasible path is very close to 100% in all simulations. In fact, when a feasible path exists, Bk-EB ($k=2$) fails at most twice in every 100,000 requests in the 32-node network. Therefore, to the best of our knowledge, our proposed algorithm achieves so far the best success ratio in finding a feasible path with a computational complexity of only about 4 times that of the standard Bellman-Ford algorithm.

Intuitively, the larger the network, the harder it is to find a feasible path. Thus, the performance of an algorithm may degrade quickly with the network size. However, by deploying k-EB, our algorithm can essentially overcome this problem, i.e. our algorithm is scalable. As shown in Figs. 8–10, the success ratios of BSA, H_MCOP, and TAMCRA decrease much sharper than that of our algorithm as the network size increases; our is still close to 1.

Fig. 11 demonstrates the AHRs of different algorithms in the 100-node network. It can be observed that our proposed algorithm, Bk-EB, achieves near optimal average hop, i.e. our algorithm can minimize the hops of the feasible path found. Note that although H_MCOP achieves relatively low AHR (compared to TAMCRA and BSA), its success ratio in finding a feasible path is not satisfactory.

One might think the computational complexity of Bk-EB is higher than those of [13,10,19] because the Dijkstra-like algorithms are deployed in [13,10,19] (the algorithms used in [13,10,19] are based on the standard Dijkstra algorithm and Reference [27], a k-shortest paths algorithm based on the Dijkstra algorithm, respectively). First of all, when $B\rightarrow\infty$ and $k_{\mathrm{TAMCRA}}\rightarrow\infty$, the computational complexities of [13,10,19] can also approach infinity, where k_{TAMCRA} is the number of shortest paths of TAMCRA. Second, since the computational complexity of TAMCRA is proportional to

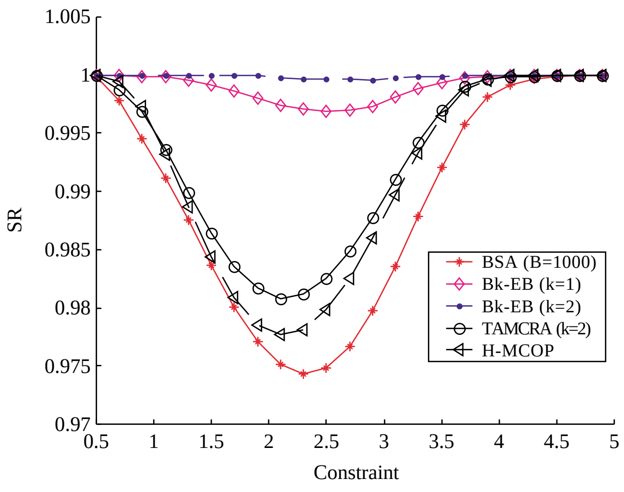


Fig. 9. SR in the 50-node network.

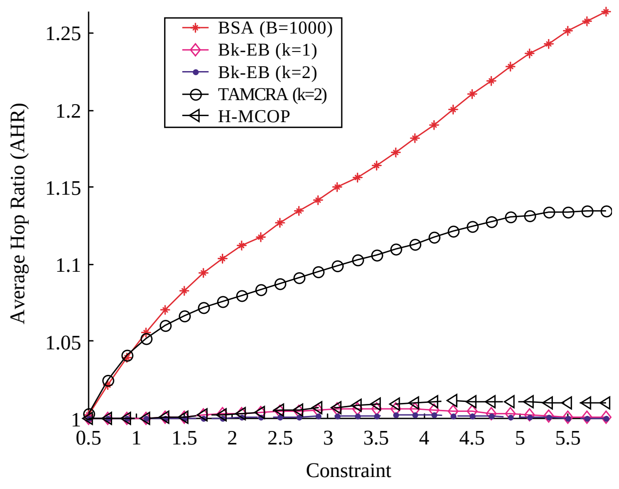


Fig. 11. AHRs of algorithms in the 100-node network.

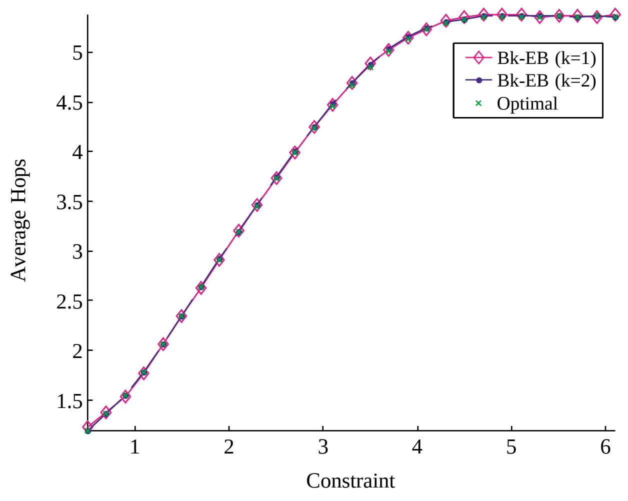


Fig. 12. Average hops of the computed feasible paths in the 100-node network.

k^3, it is higher than that of Bk-EB when k is large enough. Third, since we have already set $B=1000$ in our simulations, we believe that the upper bound ($B=\infty$) on the success ratio of the Binary Search algorithm in finding a feasible path is close to our simulation results, i.e. no matter how large B is, it is unlikely that the success ratio of the Binary Search algorithm can be higher than that of Bk-EB. Fourth, as shown in Fig. 12, the average hop of the feasible paths computed by Bk-EB are far less than the number of nodes (n) or the maximum hop count H. Note that Bk-EB stops searching as soon as it finds a feasible path. When a feasible path exists, the average computational complexity of Bk-EB is thus bounded by

$$O\left(\tilde{H}\left(\sum_{i=1}^{n}(d_i\log d_i+(k-1)\log(d_i-1))\right)\right)$$
$$\leq O(\tilde{H}(m\log n+k\log n)) \qquad (18)$$

where $\tilde{H}$ is the average hop of feasible paths. Hence, the actual runtime of Bk-EB could be very low. Finally, it is generally believed that the standard Bellman-Ford algorithm has better performance than the Disjktra algorithm in sparse networks, into which most communication networks can be classified [9]. Hence, even when k is small, our proposed algorithm may achieve better performance than [13,10,19] in terms of the success ratio in finding a feasible path, the average hop count of feasible paths, and the computational complexity.

6. Conclusions

We have proposed an efficient algorithm (Bk-EB), which can achieve a very high success ratio in finding a feasible path for the least hop(s) multiple additively constrained routing. Extensive simulations show that Bk-EB is a high performance routing algorithm in terms of both the success ratio in finding a feasible path and the average hop of solutions. With a slight modification, our algorithm can also be employed for solving many other problems, such as the DCLC problem. Moreover, the success ratio of our proposed algorithm may be further improved by, similar to [10], using a non-linear cost function, i.e. the cost of a path is the function of its weights, which, however, will increase the computational complexity.

Acknowledgements

This work has been supported in part by the National Science Foundation under Grant 0435250, and the New Jersey Commission on Science and Technology via the New Jersey Center for Wireless Networking and Internet Security.

References

[1] S. Chen, K. Nahsted, An overview of quality of service routing for next-generation high-speed network: problems and solutions, IEEE Network 12 (6) (1998) 64–79.
[2] A. Shaikh, J. Rexford, K.G. Shin, Evaluating the impact of stale link state on quality-of-service routing, IEEE/ACM Trans. Network. 9 (2) (2001) 162–176.
[3] R. Guerin, A. Orda, QoS based routing in networks with inaccurate information: theory and algorithms, Proc INFOCOM′97 (1997) 75–83.
[4] J. Wang, W. Wang, J. Chen, S. Chen, A randomized QoS routing algorithm on networks with inaccurate link-state information, Proc WCC-ICCT 2000 2 (2000) 1617–1622.
[5] D.H. Lorenz, A. Orda, QoS routing in networks with uncertain parameters, IEEE/ACM Trans. Network. 6 (6) (1998) 768–778.
[6] S. Chen, K. Nahrstedt, Distributed QoS routing with imprecise state information. Proceedings of Seventh International Conference on Computer Communications and Networks, 1998, pp. 614–621.
[7] Z. Wang, J. Crowcroft, Quality of Service routing for supporting multimedia applications, IEEE J Selected Areas Commun 14 (7) (1996) 1228–1234.
[8] X. Yuan, Heuristic algorithm for multiconstrained quality-of-service routing, IEEE/ACM Trans. Network. 10 (2) (2002) 244–256.
[9] A. Orda, A. Sprintson, Precomputation schemes for QoS routing, IEEE/ACM Trans. Network. 11 (4) (2003) 578–591.
[10] T. Korkmaz, M. Krunz, Routing multimedia traffic with QoS guarantees, IEEE Trans. Multimedia 5 (3) (2003) 429–443.
[11] G. Liu, K.G. Ramakrishnan, A*Prune: an algorithm for finding K shortest paths subject to multiple constraints, Proc. IEEE INFOCOM 2001 2 (2001) 743–749.
[12] T. Korkmaz, M. Krunz, Bandwidth-delay constrained path selection under inaccurate state information, IEEE/ACM Trans. Network. 11 (3) (2003) 384–398.
[13] T. Korkmaz, M. Krunz, S. Tragoudas, An efficient algorithm for finding a path subject to two additive constraints, Proc. ACM SIGMETRICS′ 2000 (2000) 318–327.
[14] C. Pomavalzi, G. Chakraborty, N. Shiratori, QoS based routing algorithm in integrated services packet networks, Proc. IEEE Conf. Network Protocols (1997) 167–174.

[15] A. Juttner, B. Szyiatovszki, I. Mecs, Z. Rajko, Lagrange releaxation based method for the QoS routing problem, Proc. IEEE INFOCOM 2001 2 (2001) 859–868.

[16] L. Guo, I. Matta, Search space reduction in QoS routing, Proceedings of 19th IEEE International Conference on Distributed Computing Systems, pp. 142–149, 1999.

[17] L. Gang, K.G. Ramakrishnan, A prune: an algorithm for finding k shortest paths subject to multiple constraints, Proc. IEEE INFOCOM 2001 2 (2001) 743–749.

[18] D. Eppstein, Finding the k shortest path, Proceedings of 35th Annual Symposium on Foundations of Computer Science, pp. 154–165, 1994.

[19] H. De Neve, P. Van Mieghem, A multiple quality of service routing algorithm for PNNI, Proc. IEEE ATM Workshop (1998) 324–328.

[20] S. Chen, K. Nahrsted, On finding multi-constrained path, Proc. IEEE ICC′98 2 (1998) 874–899.

[21] F.A. Kuipers, P. Van Mieghem, The impact of correlated link weights on QoS routing, Proc. IEEE INFOCOM 2 (2003) 1425–1434.

[22] D.H. Lorenz, A. Orda, Efficient QoS partition and routing of unicast and multicast, Proc. 8th Int. Workshop Quality Service (2001) 75–83.

[23] R. Hassin, Approximation schemes for the restricted shortest path problem, Math. Opert. Res. 2 (2) (1992) 36–42.

[24] G. Cheng, N. Ansari, Finding all hops k-shortest paths, Proc. IEEE PACRIM′03 1 (2003) 474–477.

[25] T.H. Cormen, C.E. Leiserson, R.L. Rivest, Introduction to Algorithms, MIT Press, Cambridge, MA, 1990.

[26] M.B. Doar, A better model for generating test networks, Proc. GLOBECOM′96 (1996) 86–93.

[27] E.I. Chong, S. Maddila, S. Morley, On finding single-source single destination k shortest paths, Proceedings of the Seventh International Conference on Computing and Information (ICCI′95), pp. 40–47, 1995.6, pp. 86–93, 1996.