A Randomized Algorithm for Finding a Path Subject to Multiple QoS Requirements[*]

Turgay Korkmaz and Marwan Krunz
Department of Electrical & Computer Engineering
University of Arizona
Tucson, AZ 85721
{turgay,krunz}@ece.arizona.edu

Last Revised: January 12, 2000

Abstract

An important aspect of quality-of-service (QoS) provisioning in integrated networks is the ability to find a *feasible* route that satisfies a set of end-to-end QoS requirements (or constraints) while efficiently using network resources. In general, finding a path subject to multiple *additive* constraints (e.g., delay, delay-jitter) is an NP-complete problem. We propose an efficient randomized heuristic algorithm to this problem. Although the algorithm is presented as a centralized one, the idea behind it can also be implemented in a distributed manner. The algorithm initially computes the cost of the best path from each node to a given destination with respect to individual link weights and their linear combination. It then randomly traverses the graph, discovering nodes from which there is a good chance to reach the final destination. The worst-case complexity of the initialization step is $\mathcal{O}(n^2)$ per path request and that of the randomized search is $\mathcal{O}(n+m)$; thus, the overall complexity of searching for a feasible path is $\mathcal{O}(n^2)$, where n is the number of nodes, and m is the number of links. Using extensive simulations, we show that the proposed algorithm has better performance than existing algorithms under the same order of computational complexity.

keywords: Multiple constrained path selection, QoS-based routing, scalable routing.

1 Introduction

One key problem in QoS-based networks (e.g., Intserv, Diffserv, ATM) is how to determine a route that satisfies multiple constraints while simultaneously achieving efficient utilization of network resources. This problem is known as QoS-based routing and is being extensively investigated in the research community [7, 23, 33, 21, 11, 4, 26]. In general, routing consists of two basic tasks: collecting the network state information and searching this information for a feasible path. In this paper, we focus on the second task, and assume that the true state of the network is available to every node

[*]This work was supported by the National Science Foundation under Career Award ANI 9733143. An abridged version of this paper was presented at the IEEE GLOBECOM '99 Conference – Symposium on Global Internet: Application and Technology, Rio de Janeiro, December 1999.

(e.g., via link-state routing) and that nodes use this information to determine end-to-end feasible paths (see [15] for QoS routing under inaccurate information). Each link in the network is associated with multiple QoS parameters, which can be roughly classified into additive and non-additive [2, 34]. For the additive parameters (e.g., delay, jitter, administrative weight), the cost of an end-to-end path is given, exactly or approximately, by the *sum* of the individual link values along that path. In contrast, the cost of a path with respect to (w.r.t.) a non-additive parameter, such as bandwidth, is determined by the value of that constraint at the bottleneck link. For the mere purpose of determining a feasible path, constraints associated with non-additive parameters can be easily dealt with as a preprocessing step by pruning all links that do not satisfy these constraints [36]. Hence, in this paper we will mainly focus on additive parameters. The underlying problem can be stated as follows.

Definition 1. *Multiple Constrained Path Selection* (MCP): Consider a communication network that is represented as a directed graph $G = (V, E)$, where V is the set of nodes and E is the set of links. Each link $(i, j) \in E$ is associated with K additive weights (QoS values) $w_k(i, j)$ for $k = 1, 2, \ldots, K$. Given K constraints c_k, $k = 1, 2, \ldots, K$, the problem is to find a path p from a source node s to a destination node t such that $w_k(p) \overset{\text{def}}{=} \sum_{(i,j)\in p} w_k(i, j) \leq c_k$ for $k = 1, 2, \ldots, K$.

For $K \geq 2$, MCP is an NP-complete problem [22, 36]. In other words, there is no efficient (polynomial-time) algorithm that can *surely* find a feasible path w.r.t. the given constraints. A related yet slightly different problem is known as the restricted (or constrained) shortest path (RSP) problem, in which the returned path is required to satisfy one constraint (e.g., delay) while being optimal w.r.t. another parameter (e.g., cost). The RSP problem is also NP-complete [14, 1]. Both the MCP and RSP problems can be solved via pseudo-polynomial-time algorithms in which the complexity depends on the actual values of the link weights (e.g., maximum link weight) in addition to the size of the network [22, 18]. However, these algorithms are computationally expensive if the values of the link weights are large.

To cope with the NP-completeness of these problems, researchers have resorted to several heuristic and approximation algorithms. One common approach is to find the k-shortest paths w.r.t. a cost function defined based on the link weights and the given constraints, hoping that one of these paths is feasible [17, 31, 13, 12, 16]. This approach is computationally inefficient if k is large. A similar approach to the k-shortest paths is to *implicitly* enumerate all feasible paths [3], but this approach is also computationally expensive. In [37] the author proposed the Constrained Bellman-Ford (CBF) algorithm, which performs a breadth-first-search by discovering paths of monotonically increasing delay while maintaining lowest-cost paths to each visited node. Although this algorithm exactly solves the RSP problem, its running time grows exponentially in the worst-case. The authors in [30] proposed a distributed heuristic solution for RSP with message complexity of $\mathcal{O}(n^3)$, where n is the number of nodes. This complexity has been improved in [38, 20]. In [18] the author presented two ϵ-optimal approximation algorithms for RSP with complexities of $\mathcal{O}(\log\log B(m(n/\epsilon) + \log\log B))$ and $\mathcal{O}(m(n^2/\epsilon)\log(n/\epsilon))$, where B is an upper bound on the solution (e.g., the longest path), m is the number of links, and ϵ is a quantity that reflects how far the solution is from the optimal one. Although the complexities of these ϵ-optimal algorithms are polynomial, they are still computationally prohibitive in large networks [28]. Accordingly, the author in [28] investigated the hierarchical structure of such networks and provided a new ϵ-optimal approximation algorithm with better scalability. The above studies are especially proposed for the RSP problem and their computational complexities are excessive in the worst-case.

In [22] Jaffe considered the MCP problem and proposed an intuitive approximation algorithm to it based on minimizing a linear combination of the link weights. More specifically, this algorithm returns the best path w.r.t. $l(e) = \alpha w_1(p) + \beta w_2(p)$ by using Dijkstra's shortest path algorithm, where α, $\beta \in Z^+$. The key issue here is to determine the appropriate α and β such that an optimal path w.r.t. $l(e)$ is likely to satisfy the individual constraints. In [22] Jaffe determined two sets of values for α and β based on minimizing an objective function of the form $f(p) = \max\{w_1(p), c_1\} + \max\{w_2(p), c_2\}$. Instead of fixing α and β a priori, the authors in [5] proposed a similar approximation algorithm that dynamically adjusts the values of α and β. However, the computational complexity of this algorithm is exponential. Chen and Nahrstedt proposed another heuristic algorithm (with two constraints) that modifies the problem by scaling down the values of one link weights to bounded integers [6]. They showed that the modified problem can be solved by using Dijkstra's (or Bellman-Ford) shortest path algorithm and that the solution to the modified problem is also a solution to the original one. When Dijkstra's algorithm is used, the computational complexity of their algorithm is $\mathcal{O}(x^2n^2)$; when Bellman-Ford algorithm is used, the complexity is $\mathcal{O}(xnm)$, where x is an adjustable positive integer whose value determines the performance and overhead of the algorithm. If x is small, the algorithm is fast but it does not have good performance. To achieve a high probability of finding a feasible path, the value of x need to be as large as $10n$, resulting in the computational complexity of $\mathcal{O}(n^4)$.

Other works in the literature were aimed at addressing special yet important cases of the QoS routing problem. For example, some researchers focused on an important subset of QoS requirements (e.g., bandwidth and delay). Showing that the feasibility problem under this combination is not NP-complete, the authors in [35] presented a *bandwidth-delay based routing algorithm* which simply prunes all links that do not satisfy the bandwidth requirement and then finds the shortest path w.r.t. delay in the pruned graph. Several path selection algorithms based on different combinations of bandwidth, delay, and hop-count were discussed in [26, 25] (e.g., widest-shortest path, shortest-widest path). In addition, new algorithms were proposed to find more than one feasible path w.r.t. the bandwidth and delay parameters (e.g., Maximally Disjoint Shortest and Widest Paths (MADSWIP)) [32]. Another approach to QoS routing is to exploit the dependencies between the QoS parameters and solve the path selection problem assuming specific scheduling schemes at network routers [25, 29]. Specifically, if Weighted Fair Queueing (WFQ) service discipline is being used and the constraints are bandwidth, queueing delay, jitter, and loss, then the problem can be reduced to standard shortest path problem by defining the other constraints based on bandwidth. Although queueing delay can be formulated as a function of bandwidth, this is not the case for the propagation delay, which is the dominant delay component in high-speed networks [8].

Contributions and Organization of the Paper

Previously proposed algorithms suffer from either excessive computational complexities and/or low performance. Moreover, most of them are only applicable to specific constraints. In this paper, we introduce an efficient randomized heuristic search algorithm to the MCP problem. We make no assumptions about the scheduling disciplines in the network or the number and interdependencies of the QoS parameters. The algorithm starts by computing the cost of the best path from each node u to a specific destination node t w.r.t. each link weight and also w.r.t a linear combination of all link weights. After this initialization, the algorithm starts from the source node and discovers neighboring nodes that fall in the *estimated* feasibility region. This region is calculated for each node based on the cost of the already traversed segment of an end-to-end path as well as the cost of the best remaining segment. The algorithm then randomly chooses one of the discovered nodes and uses it to reach other nodes. This randomization avoids unforeseen traps during the execution of the algorithm [24, 27] and results in searching fast for a feasible path. The worst-case complexity of the initialization step is $\mathcal{O}(n^2)$ per path computation and that of the randomized search is $\mathcal{O}(n+m)$; thus, the overall complexity is $\mathcal{O}(n^2)$ since n and $m \leq n^2$. Extensive simulation results show that the proposed randomized algorithm outperforms existing algorithms under the same order of computational complexity. Further performance enhancement in the randomized search is also presented at the expense of increasing the complexity from $\mathcal{O}(n+m)$ to $\mathcal{O}(n^2)$, but the overall complexity is still the same.

The rest of the paper is organized as follows. In Section 2, we present the randomized algorithm. Its enhanced version is presented in Section 3. We report simulation results in Section 4. Finally, Section 5 concludes this paper and points to some future directions.

2 Randomized Algorithm for MCP

The proposed heuristic search is a modification of the breadth-first-search (BFS) algorithm [10]. In contrast to BFS, which systematically discovers every node that is reachable from a source node s, our algorithm randomly discovers nodes from which there is a good chance to reach a destination node t. By using the information obtained in the initialization step (described next), the algorithm can decide whether this chance exists before discovering a node. If there is no chance, the algorithm can foresee the trap and randomly attempts to explore other nodes.

Initially, the algorithm computes the cost of the best path from each node u to t w.r.t. every link weight w_k and also w.r.t. the linear combination of these weights $w_1 + w_2 + \ldots + w_K$. This initialization step can be done by one of the following two approaches. In the first approach, the cost of the best path from each node u to t (w.r.t. each link weight and w.r.t. the linear combination of these weights) can be found by using Reverse-Dijkstra algorithm whenever a path is requested. Alternatively, the initialization can be done all at once for all possible destinations by using the Floyd-Warshal algorithm (or Bellman-Ford's algorithm in the distributed case as presented in Appendix C). The computational complexity of this algorithm is $\mathcal{O}(n^3)$ while the complexity of Reverse-Dijkstra algorithm is $\mathcal{O}(n^2)$ [1]. However, if the number of path requests is larger than n, the additional complexity of the Floyd-Warshal algorithm is amortized since the initialization step is performed once but the information that it provides is used several times by the randomized algorithm. Therefore, either of these approaches can be chosen based on the expected number of path requests within a state update interval. In the worst-case, the complexity of the initialization is $\mathcal{O}(n^2)$ per path request.

A pseudo-code of the randomized algorithm (R_MCP) is presented in Figure 1. The algorithm maintains the following labels for each node u: $B_k[u,v]$, $L[u,v]$, $D_k[u]$, and $\pi[u]$, $k = 1,2,\ldots,K$.

```
Initialization I (performed for each path request)
    /* Using the Reverse-Dijkstra algorithm */
    Compute $B_k[u,t]$, $k = 1,2,\ldots,K$ and $L[u,t]$ for all $u$
end Initialization I

Initialization II (performed once for all path requests)
    /* Using the Floyd-Warshal algorithm */
    Compute $B_k[u,v]$, $k = 1,2,\ldots,K$, and $L[u,v]$ for all $u$ and $v$
end Initialization II

R_MCP($G = (V,E), s, t, c_k, k = 1,2,\ldots,K$)
1  for $k = 1$ to $K$ do
2      if $B_k[s,t] > c_k$ then
3          return failure /* there is no feasible path */
4      end if
5  end for
6  if $L[s,t] > \sum_{k=1}^{K} c_k$ then
7      return failure /* there is no feasible path */
8  end if
9  /* there might be a feasible path, try to find it */
10 for $attemt = 1$ to $\gamma$ do
11     RH_BFS($G = (V,E), s, t$)
12     if $t$ is discovered (i.e., $\pi[t] \neq NIL$) then
13         return the path /* a feasible path is found */
14     end if
15 end for
end R_MCP
```

Figure 1: Randomized algorithm for path selection with multiple constraints.

Label $B_k[u,v]$ represents the cost of the shortest path from node u to every possible destination node v w.r.t. link weight w_k. Label $L[u,v]$ represents the same kind of information as $B_k[u,v]$ but with optimization being performed w.r.t. the linear combination of all weights $w_1 + w_2 + \ldots + w_K$. Label $D_k[u]$ represents the cost of a path from s to node u w.r.t. link weights w_k, $k = 1,2,\ldots,K$. The algorithm stores the predecessor of node u in $\pi[u]$. If node u has no predecessor, i.e., it has not been discovered yet, then $\pi[u] = NIL$.

In the first part (lines 1–8) of R_MCP, the algorithm checks whether the cost of the shortest path from s to t w.r.t. w_k is larger than the constraint c_k for any of the k's or if the cost of the shortest path w.r.t. the linear combination of all weights is larger than the linear combination of the constraints. If any of these conditions is true, then there is no feasible path that satisfies all the given constraints. If all these conditions are false, then there is a chance of finding a feasible path. To illustrate how the algorithm decides whether there might be a feasible path or not, consider the network in Figure 2(a) in which each link is associated with two weights (w_1, w_2). There are five paths from s to t, which

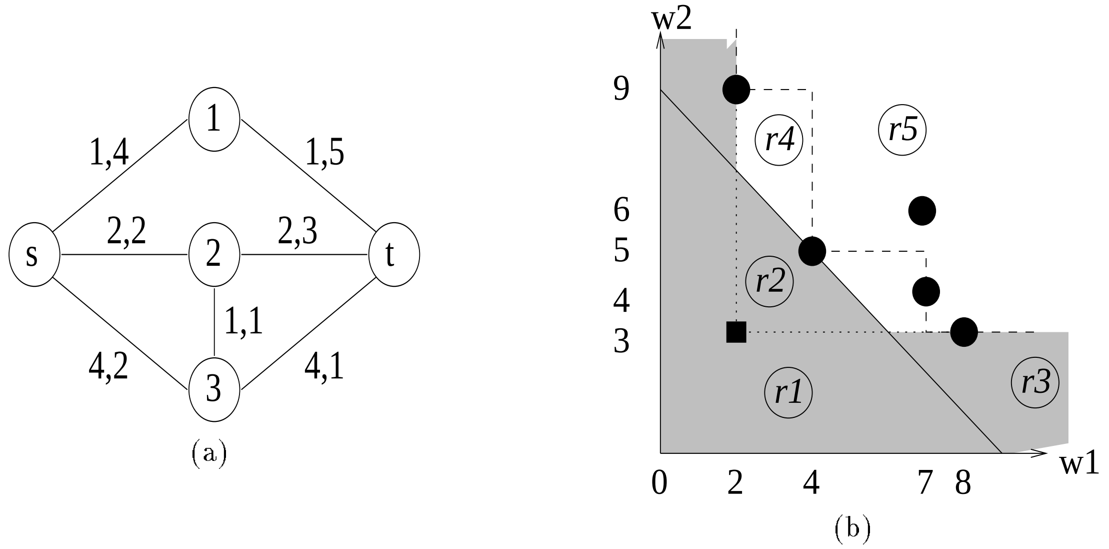


Figure 2: Example showing how R_MCP determines whether a feasible path might exist.

are indicated by the black circles in Figure 2(b). The best-cost values w.r.t. the individual weights are indicated by the black square. The path crossing the solid line is the best path w.r.t. the linear combination of link weights. The shaded area represents the "absolute rejection region" while the unshaded area represents the "estimated feasibility region" Suppose we have five connection requests $r1$, $r2$, $r3$, $r4$, and $r5$ with constraint values as indicated in Figure 2(b). By testing the conditions in lines 1–8 in R_MCP, the algorithm can determine that $r1$, $r2$, and $r3$ are definitely infeasible and that there might be feasible paths for $r4$ and $r5$ even though no path satisfies the requirements of $r4$.

If a request does not fall in the absolute rejection region, the algorithm will call RH_BFS (Randomized-Heuristic BFS) to randomly search for a feasible path for this request. Because of its randomized nature, this search can be applied more than once to increase the probability of finding a feasible path. Let γ be the maximum number of such attempts. In practice, we have observed that even with $\gamma = 1$, the algorithm gives a high success rate in finding feasible paths. Therefore, γ is always a small number.

A pseudo-code of RH_BFS is presented in Figure 3. This procedure is modified from BFS [10]. Instead of a first-in first-out queue which is used to manage the just discovered nodes in the original BFS, RH_BFS uses a random queue Q for this purpose. In the main loop, RH_BFS chooses a random node u from Q and tries to discover every node v in the adjacency list of u. While BFS systematically discovers every node that is reachable from s, RH_BFS discovers nodes from which there is a potential for having a feasible route to t. More specifically, it accumulates the cost of the already traversed

```
RH_BFS($G = (V, E), s, t$)
1  for each node $u \in V - \{s\}$ do
2      $D_k[u] = \infty$, $k = 1, 2, \ldots, K$
3      $\pi[u] = NIL$
4  end for
5  $D_k[s] = 0$, $k = 1, 2, \ldots, K$
6  $\pi[s] = -1$ /* $s$ has no predecessor */
7  $Q = \{s\}$
8  while $Q \neq \emptyset$ and $\pi[t] = NIL$ do
9      $u = random[Q]$
10     $Q = Q - \{u\}$
11     for each $v \in Adj[u]$ do
12         if $\pi[v] = NIL$ and
               $D_k[u] + w_k(u, v) + B_k[v, t] \leq c_k \ \forall \, k$ and
               $\sum_{k=1}^{K}(D_k[u] + w_k(u, v)) + L[v, t] \leq \sum_{k=1}^{K} c_k$ then
13             $D_k[v] = D_k[u] + w_k(u, v) \ \forall \, k$
14             $\pi[v] = u$
15             $Q = Q \cup \{v\}$
16         end if
17     end for
18 end while
end RH_BFS
```

Figure 3: Randomized heuristic search for path selection.

segment from s to u, the cost of link (u, v), and the cost of the best remaining segment from v to t w.r.t. each link weight and w.r.t. the linear combination of weights (line 12) so that it can determine an absolute rejection region as in Figure 2(b). If a connection request falls outside this region, then there is a chance to reach t via v, so it discovers node v from node u, updates $D_k[v]$, $k = 1, 2, \ldots, K$, and $\pi[v]$, and puts node v into Q. The search terminates as soon as t is discovered or if Q is empty.

Next, we establish the correctness of the randomized algorithm. Since the algorithm is based on a heuristic, it may not find a feasible path even if one exists. However, the following theorem shows that if the algorithm returns a path, then this path must satisfy the given constraints. Such an algorithm which either returns a correct solution or does not return any solution is known as *Las Vegas Algorithm* [27].

Theorem 1 *Suppose that R_MCP is run on $G = (V, E)$ to find a path p from s to t such that $w_k(p) \overset{\text{def}}{=} \sum_{(i,j) \in p} w_k(i, j) \leq c_k$ for $k = 1, 2, \ldots, K$. If R_MCP discovers t (i.e., $\pi[t] \neq NIL$) upon termination, then the constructed path p from s to t must satisfy the given constraints.*

Proof: see Appendix A. ■

We now evaluate the worst-case complexity of the algorithm. Let n be the number of nodes and m be the number of links in the graph. We assume that K and γ (the maximum number of RH_BFS iterations) are constants and much smaller than n. In the worst-case, the initialization step requires $(K+1)$ iterations of Reverse-Dijkstra algorithm whose complexity is $\mathcal{O}(n^2)$. Since K is a small constant, the complexity of the initialization step is $\mathcal{O}(n^2)$. After the initialization step, R_MCP performs $K+1$ comparisons (lines 1–8) to check the possible existence of a feasible path. It then calls RH_BFS at most γ times (lines 10–15). The computational complexity of RH_BFS is $\mathcal{O}(n+m)$ (same as BFS [10]). The computational complexity of R_MCP is $\mathcal{O}((K+1)+(\gamma*(n+m)))$. Since K and γ are small constants, the worst-case complexity of R_MCP is $\mathcal{O}(n+m)$. The overall complexity of searching for a feasible path is $\mathcal{O}(n^2)$ since n and $m \leq n^2$. The storage complexity of the algorithm is $\mathcal{O}(n^2)$ if the Floyd-Warshal algorithm is used and $\mathcal{O}(n)$ if the Reverse-Dijkstra algorithm is used.

Example

The following example illustrates the operation of the randomized algorithm. Consider the network in Figure 4. Each link is bidirectional and has two additive weights (w_1,w_2). Although links are symmetric in this example, the algorithm can run on asymmetric links with real-valued weights. Suppose we want to find a path from $s=0$ to $t=4$ with $c_1=13$ and $c_2=12$. Assume that

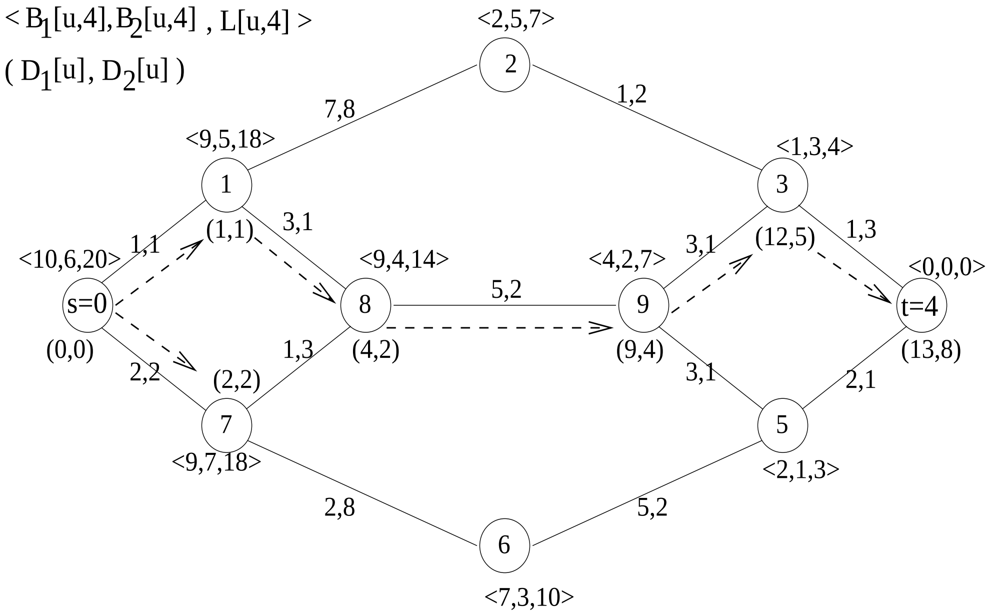


Figure 4: Example that illustrates how the randomized algorithm finds a feasible path under two additive constraints.

labels $B_k[u,4]$, $k=1,2$, and $L[u,4]$ for each node u have been computed as shown in Figure 4. Since $B_1[0,4]=10 \leq c_1$, $B_2[0,4]=6 \leq c_2$, and $L[0,4]=20 \leq c_1+c_2=25$, R_MCP algorithm proceeds to search for a feasible path. It calls RH_BFS whose execution is illustrated using dashed lines. RH_BFS randomly chooses a node u from Q and discovers the nodes in the adjacency list of u. By default, $s = 0$ is discovered first and placed in Q. Since $Q = \{0\}$, RH_BFS chooses node 0, discovers the adjacent nodes 1 and 7, and puts them in Q. Suppose that RH_BFS randomly chooses node 1 from Q. It then discovers node 8, resulting in $Q = \{7, 8\}$. At that time it cannot discover node 2 since $D_2[1] + w_2(1,2) + B_2[2,4] = 1 + 8 + 5 = 14 > c_2$. Next, it randomly chooses 8, from which it discovers node 9, resulting in $Q = \{7, 9\}$. Now, assume that it randomly chooses node 9. At this node, RH_BFS foresees that if it discovers node 5 from node 9, there is no chance to reach the destination since $D_1[9] + w_1(9,5) + B_1[5,4] = 9 + 3 + 2 = 14 > c_1$. However, it discovers node 3, resulting in $Q = \{7, 3\}$. Suppose RH_BFS randomly chooses node 3, from which it will discover the destination node 4. At this point, the algorithm stops and returns the feasible path $p = (0, 1, 8, 9, 3, 4)$.

One aspect of the randomized search is that even if the network sate does not change, different execution of the algorithm can result in different feasible paths between the same source and destination nodes. For some applications (e.g., IP telephony), it is desirable to route all or some of calls between the same source and destination together so that network resources are efficiently used via traffic aggregation. To address this issue, path caching can be used as follows. Whenever a new path is found, it is cached for a certain amount of time. For each connection request, one first looks for a feasible path in the cache; if there is none, then the randomized algorithm is executed to search for a new feasible path.

3 Enhancing the Randomized Search

The efficiency of the randomized search can be enhanced by modifying line 9 in RH_BFS. This is done by first ranking the nodes in Q according to a heuristic selection function described below and then selecting one of these nodes in a manner that favors the top nodes in the ranked Q. In the simulation section, we consider a special case (deterministic version) of the enhanced algorithm (ER_MCP_D) that always selects the top node in Q. This enhancement increases the complexity of the RH_BFS from $\mathcal{O}(n + m)$ to $\mathcal{O}(n^2)$, but the overall complexity of the algorithm stays the same.

The heuristic selection function is based on computing the area between the given constraints and the estimated feasibility region determined at each node. The larger the area, the better the chance of having a feasible path. Let's consider how the algorithm computes such area for each discovered node before selecting one of them. Consider again the example in Figure 4. After deciding that there might be a feasible path, the algorithm discovers nodes 1 and 7 form s. In contrast to randomly selecting one of these nodes from Q, the enhanced algorithm first determines the area between the given constraints and the estimated feasibility region at each node in Q. The shaded areas in Figure 5(a-b) show such areas for nodes 1 and 7, respectively. Using the geometric formulas, the areas in Figure 5(a-b) can be computed as 10 and 4, respectively. Thus, the nodes 1 and 7 will be ordered in Q as 1 and 7, and node 1 will have more chance to be selected. For three constraints, the geometric formulas for computing the volume between the constraints and the estimated feasibility region are used.

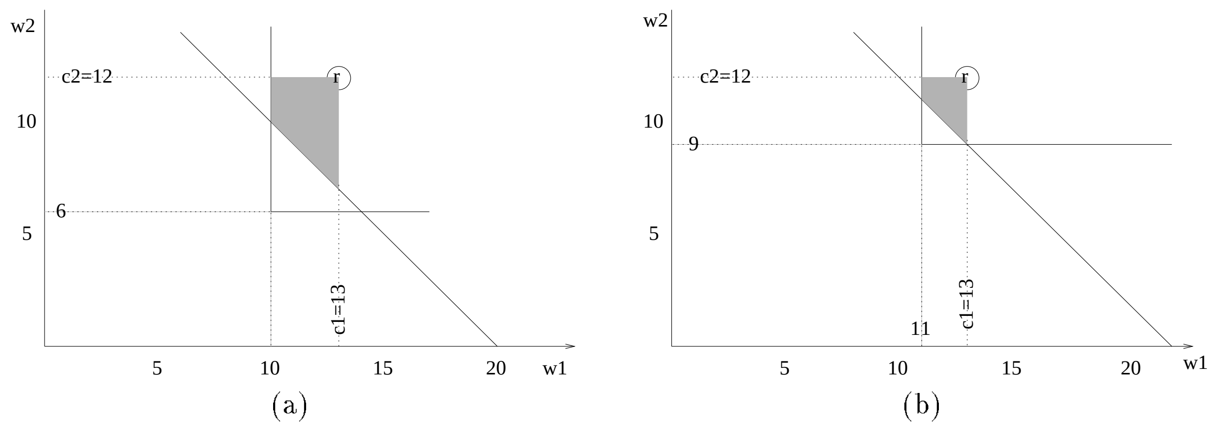


Figure 5: Example of how the enhanced randomized algorithm computes the area between the constraints and the tentative feasibility region under two constraints

4 Simulation Results

In this section, we study and contrast the performance of R_MCP and ER_MCP_D with Jaffe's two approximations (one with $\alpha = \beta = 1$ while the other with $\alpha = 1$ and $\beta = \sqrt{c_1/c_2}$) [22] and Chen's heuristic [6]. The worst-case computational complexities of R_MCP and ER_MCP_D are $\mathcal{O}(n^2)$, compared to $\mathcal{O}(n^2)$ for Jaffe's algorithm. The *exact* complexity of Chen's algorithm is $\mathcal{O}(x^2n^2)$ or $\mathcal{O}(xnm)$. Under certain distributions of link weights, Jaffe shows that his second approximation theoretically outperforms the first. However, only the first algorithm can be readily extended to more than two constraints. The performance has been studied for several network topologies. For brevity, we present the results for two topologies.

4.1 Simulation Model and Performance Measures

In the simulation model, a network is given as a directed graph. In each *experiment*, link weights (two or three) are generated randomly from a uniform distribution. For the same experiment, the simulation program generates random connection requests and tries to find a feasible path for each request using a given path selection algorithm. For comparison purposes, we have also implemented an optimal (exponential-time) algorithm that searches all paths to find a feasible one with minimum hop-count. If an algorithm finds a feasible path for a connection request, we count this request as a *routed connection request*. To contrast the performance of various algorithms, we use the following related measures:

$$\textit{success ratio}\,(\mathrm{SR}) = \frac{\textit{Total number of routed connection requests}}{\textit{Total number of connection requests}}$$

$$\textit{failure rate}\,(\mathrm{FR}) = 1 - \frac{\mathrm{SR}\ \textit{of a given algorithm}}{\mathrm{SR}\ \textit{of the optimal algorithm}}$$

SR shows how often an algorithm finds a feasible path [6], while FR represents the probability of missing a feasible path.

Although the main goal of a path selection algorithm is to achieve high SR or low FR, the algorithm should also try to do so while efficiently utilizing network resources. It has been shown that restricting routing to short paths achieves efficient resource utilization in QoS-based routing [25]. Therefore, a good algorithm should try to minimize hop-count while finding a feasible path. To measure the performance of the algorithms in this sense, we use the *average hop count* (AHC), which gives the average number of hops per *routed connection request*. Although we did not try to minimize the hop-count in our simulations, AHC can be further minimized by defining a lower bound (constraint) on hop-count and repeating the randomized algorithm while gradually increasing this bound (see Appendix B).

4.2 Results on Irregular Network Topology

We first consider the network topology in Figure 6 (also used in [6]), which has been modified from ANSNET [9] by inserting additional links. For each link (i,j), its weights are randomly selected as follows: $w_1(i,j) \sim uniform[0,50]$, $w_2(i,j) \sim uniform[0,200]$, and $w_3(i,j) \sim uniform[0,100]$. The

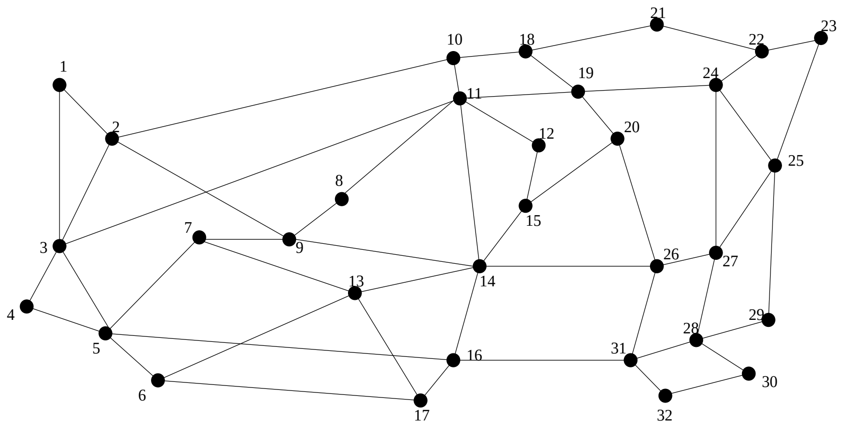


Figure 6: Network topology used in simulations.

source and destination nodes are selected randomly. The constraint values are randomly generated from uniform distributions with different ranges. Five sets of ranges are used in our experiments, which are given in Table 1. With these ranges, path selection algorithms can be contrasted over a

Range Number (**RN**)	c_1	c_2	c_3
1	$uniform[50,65]$	$uniform[200,260]$	$uniform[75,150]$
2	$uniform[75,90]$	$uniform[300,360]$	$uniform[100,200]$
3	$uniform[100,115]$	$uniform[400,460]$	$uniform[150,250]$
4	$uniform[125,140]$	$uniform[500,560]$	$uniform[200,300]$
5	$uniform[150,165]$	$uniform[600,660]$	$uniform[250,350]$

Table 1: Ranges of c_1, c_2, and c_3 values used in the simulations.

wide range of FR values. We obtain the SR, FR, and AHC averaged over twenty experiments; each of them considers randomly generated 2000 connection requests.

We first compare the algorithms under two constraints. The FR and SR values are shown in Figure 7(a-b). The results show that ER_MCP_D provides better performance than R_MCP when $\gamma \leq 2$. However, the performance of R_MCP gets better than that of ER_MCP_D as γ increases. Both R_MCP and ER_MCP_D provide significantly superior performance to both of Jaffe's approximations. The FR for R_MCP ranges from 0% (when RN = 1) to 0.54% (when RN = 3 and $\gamma = 1$) and for ER_MCP_D it ranges from 0% (when RN = 1) to 0.28% (when RN = 3). In contrast, the FR for Jaffe's algorithms ranges from 6.2% to 11.3% for the first approximation and from 3.4% to 6.7% for the second. In relative terms, the *average* FR for R_MCP is about 3.5% and 6.1% of its corresponding values in Jaffe's first and second approximations, respectively. Interestingly, the FR of Jaffe's second approximation is always about 50-60% that of his first approximation. To compare R_MCP and ER_MCP_D with Chen's heuristic, we need to properly set the value of x of the latter algorithm. In theory, as x goes to infinity the performance of Chen's algorithm approaches that of the optimal one. But given the $\mathcal{O}(x^2n^2)$ complexity of the algorithm, a large x clearly makes the algorithm impractical. To get as close as possible to having the same computational complexity of R_MCP (namely, three executions of Reverse-Dijkstra's algorithm and a few iterations of R_BFS), x must be set to two. Note that even with such a small value of x, the algorithm still requires four iterations of Dijkstra's algorithm. The resulting performance is shown in Figure 7(b) (column 8). With $x = 2$, the performance of Chen's algorithm lags significantly behind the others, with its FR ranging from 25% (when RN = 1) to 66% (when RN = 5). Even if we increase x to ten (column 9), making the computational requirement of Chen's algorithm several times that of R_MCP, its performance still lags behind R_MCP. The AHCs of the various algorithms are shown in Figure 7(c). R_MCP and Jaffe's algorithms perform close to the optimal one, especially when the given constraints are tight. Note that R_MCP can further minimize the AHC as described in Appendix B at an additional computational cost. Although the AHC of Chen's algorithm is also close to the optimal one when $x = 10$, the AHC is significantly less than the others when $x = 2$; the reason is that this algorithm cannot find feasible paths with large hop-counts unless x is increased.

Next, we compare the algorithms under three constraints. Figure 8(a-b) contrasts the SR and FR of R_MCP and ER_MCP_D with Jaffe's first approximation. Again, R_MCP and ER_MCP_D show a significant performance improvement over their contender. In terms of AHC, these three algorithms perform close to the optimal one, as shown in Figure 8(c).

4.3 Results on Regular Network Topology

Next, we consider a regular 10x10 mesh topology. For each link (i, j), its weights are randomly selected as follows: $w_1(i,j) \sim uniform[0, 30]$, $w_2(i,j) \sim uniform[0, 100]$, and $w_3(i,j) \sim uniform[0, 50]$. The source and destination nodes are selected randomly. For the constraint values, the previous five sets of ranges (Table 1) are used. Again we obtain the SR, FR, and AHC averaged over twenty experiments; each experiment considers randomly generated 2000 connection requests.

We present the comparisons under two constraints. The FR and SR values are shown in Figure 9(a-b). The results show that R_MCP and ER_MCP_D provide significantly superior performance to both of Jaffe's approximations. The FR for R_MCP ranges from 0.1% (when RN = 1 and $\gamma = 5$) to 1.9% (when RN = 3 and $\gamma = 1$). In contrast, the FR for Jaffe's algorithms ranges from 10% to 20% for the first approximation and from 5.1% to 12.1% for the second. For $x = 2$, the performance of Chen's algorithm lags significantly behind the others with its FR ranging from 64% (when RN = 1) to 89% (when RN = 5). Even for $x = 10$, the performance of Chen's algorithm still lags behind R_MCP and ER_MCP_D with its FR ranging from 9.5% (when RN = 1) to 22.9% (when RN = 5). The AHC under two constraints are shown in Figure 9(c). Again the AHCs of the randomized algorithm, Jaffe's algorithms, and Chen's algorithm with $x = 10$ are close to the optimal one, while Chen's algorithm with $x = 2$ results in significantly less SR and AHC than the others since it cannot find feasible paths with large hop-counts. Similar trends were observed under three constraints.

5 Conclusions and Future Work

Multiple constraint path selection (MCP) is an NP-complete problem, which can only be addressed through heuristics and approximation algorithms. Previously proposed algorithms suffer from excessive computational complexities and/or low performance. In this paper, we have introduced an efficient randomized algorithm (R_MCP), which is applicable to any number of constraints. After initialization step, the algorithm performs a heuristic search by randomly discovering nodes from which there is a chance to reach the final destination. We proved that any path found by this algorithm satisfies the given constraints. The computational complexity of the randomized search is $\mathcal{O}(n + m)$, where n is the number of nodes and m is the number of links. Since the complexity of initialization step is at most $\mathcal{O}(n^2)$ per path request, the overall complexity of searching for a feasible path is $\mathcal{O}(n^2)$. Simulations considering various network topologies indicated that R_MCP gives better performance (i.e., success rate) than other algorithms under the same complexity. In addition, we enhanced the randomized search and contrasted its deterministic version (ER_MCP_D) with other algorithms. It gives better performance than existing algorithms and improves the performance of R_MCP at the expense of negligible computational cost.

The randomized algorithm performs well when the true state of the network is given. However, the true state may not be available to every node at all times due to network dynamics, aggregation of state information, and latencies in state dissemination. As a future work, we will investigate how our algorithm performs in the presence of inaccurate state information. Because of randomization, different paths can be chosen for the same pair of source and destination at different times. We will also investigate how the random path selection strategy affects load balancing and compensates for the inaccuracy in the network state information.

Appendix

A Proof of Theorem 1

Assume that the algorithm returns a path $p = (v_0, v_1, v_2, \ldots, v_l)$, where $v_0 = s$ and $v_l = t$. We show that p satisfies the given constraints. Let v_{i-1} and v_i be two successive nodes that lie on p. RH_BFS discovers v_i from v_{i-1} if (line 12 in RH_BFS)

$$D_k[v_{i-1}] + w_k(v_{i-1}, v_i) + B_k[v_i, t] \le c_k \tag{1}$$

for all $k = 1, 2, \ldots, K$. After the discovery of v_i, we have (line 13 in RH_BFS)

$$D_k[v_i] = D_k[v_{i-1}] + w_k(v_{i-1}, v_i) \tag{2}$$

from which we conclude

$$w_k(v_{i-1}, v_i) = D_k[v_i] - D_k[v_{i-1}] \tag{3}$$

From (1) and (2), we have

$$D_k[v_i] + B_k[v_i, t] \le c_k \tag{4}$$

$$D_k[v_i] \le c_k - B_k[v_i, t] \tag{5}$$

Summing the weights (w_k for each $k = 1, 2, \ldots, K$) along the path p yields

$$\begin{aligned} w_k(p) &\overset{\text{def}}{=} \sum_{i=1}^{l} w_k(v_{i-1}, v_i) && (6)\\ &= \sum_{i=1}^{l} D_k[v_i] - D_k[v_{i-1}] && (7)\\ &= D_k[v_l] - D_k[v_0] && (8)\\ &= D_k[v_l] \le c_k - B_k[v_l, t] = c_k && (9) \end{aligned}$$

In this derivation, (8) comes from the telescoping sum on (7). Then, (9) first follows from $D_k[v_0] = D_k[s] = 0$; it then follows from $R_k[v_l, t] = B_k[t, t] = 0$. Thus, $w_k(p) \le c_k$ for each $k = 1, 2, \ldots, K$.

B MCP with Minimum Hop-count

By repeating R_BFS a maximum of n times, the *average hop count* can be minimized as follows. Let $\sum_{(i,j)\in p} 1 \le H$ be a new constraint which is similar to the other K constraints, where H is a bound on the hop-count, $0 \le H \le n-1$. If R_BFS is executed for each value of H, starting from $H = 0$ up to $H = n-1$ (terminating whenever a path is found), then we can find a feasible path with minimum hop-count. Although the computational complexity seems to be $\mathcal{O}(\gamma n(n+m))$ in the worst-case, source nodes may have a policy not to establish a route that has a number of hops greater than a predefined maximum-hop-length limit H_{max} [29]. In this case, R_BFS will be repeated H_{max} times, resulting in a worst-case complexity of $\mathcal{O}(\gamma H_{max}(n+m))$. In practice, $H_{max} \ll n$ [28].

C Distributed Randomized Algorithm

The randomized algorithm can be also implemented in a distributed manner as follows. The same labels $B_k[u,v]$, $k = 1, 2, \ldots, K$, and $L[u,v]$ are maintained at each node u for each destination node v. These labels are similar to distance vectors of the existing distributed algorithms [19] and can be computed using the same procedures (e.g., Bellman-Ford's algorithm) w.r.t. each link weight w_k and the linear combination of these weights, respectively. At the source node s, the algorithm first constructs a *path_request* message that contains t (the destination node), π (the already traversed segment of a path), D_k (the accumulated cost of this segment), and the given constraints c_k, $k = 1, 2, \ldots, K$. The algorithm then communicates with the neighboring nodes to determine the nodes that satisfy the tentative feasibility condition (lines 1–8 in R_MCP or line 12 in RH_BFS). After determining such nodes, the algorithm randomly selects one of them (or selects the best one as in ER_MCP_D), updates *path_request*, and sends it to the selected node. When a node receives a *path_request* message, the algorithm does the same things at this node until t is reached. Since the already traversed segment (π) is maintained in the *path_request* message, the constructed path can be easily made loop-free by not visiting nodes in π. In addition, by sending more than one *path_request* messages to the neighbors at the same time, the source can increase the chance of finding a feasible path and can determine multiple paths. The feasibility of this approach to the distributed QoS-based routing, its potential improvements, and performance evaluation will be reported in future work.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, Inc., 1993.

[2] A. Alles. ATM internetworking. White Paper, Cisco Systems, Inc., May 1995.

[3] Y. P. Aneja, V. Aggarwal, and K. P. K. Nair. Shortest chain subject to side constraints. *Networks*, 13:295–302, 1983.

[4] G. Apostolopoulos, R. Guerin, S. Kamat, and S. K. Tripathi. Quality of service based routing: A performance perspective. In *Proceedings of the ACM SIGCOMM '98 Conference*, pages 15–26, Vancouver, British Columbia, Canada, August-September 1998. http://www.acm.org/sigcomm/sigcomm98/tp/abs_02.html.

[5] D. Blokh and G. Gutin. An approximation algorithm for combinatorial optimization problems with two parameters. IMADA preprint PP-1995-14, May 1995, http://www.imada.ou.dk/Research/Preprints/Abstracts/1995/14.html.

[6] S. Chen and K. Nahrstedt. On finding multi-constrained paths. In *Proceedings of the ICC '98 Conference*, pages 874 –879. IEEE, 1998.

[7] S. Chen and K. Nahrstedt. An overview of quality-of-service routing for the next generation high-speed networks: Problems and solutions. *IEEE Network*, 12(6):64–79, Nov-Dec 1998.

[8] D. Clark et al. Strategic directions in networks and telecommunications. *ACM Computing Surveys*, 28(4):579–690, 1996.

[9] D. E. Comer. *Internetworking with TCP/IP*, volume I. Prentice Hall, Inc., third edition, 1995.

[10] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* The MIT press and McGraw-Hill book company, sixteenth edition, 1996.

[11] E. Crawley et al. A framework for QoS-based routing in the Internet. Internet draft, IETF, July 10, 1998. (draft-ietf-qosr-framework-06.txt).

[12] H. De Neve and P. Van Mieghem. A multiple quality of service routing algorithm for PNNI. In *Proceedings of the ATM Workshop*, pages 324 – 328. IEEE, May 1998.

[13] D. Eppstein. Finding the k shortest paths. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 154 – 165. IEEE, Nov. 1994.

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

[15] R. Guerin and A. Orda. QoS-based routing in networks with inaccurate information: Theory and algorithms. In *Proceedings of the INFOCOM '97 Conference*, pages 75–83. IEEE, 1997.

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

[17] G. Y. Handler and I. Zang. A dual algorithm for the constrained shortest path problem. *Networks*, 10:293–310, 1980.

[18] R. Hassin. Approximation schemes for the restricted shortest path problem. *Mathematics of Operations Research*, 17(1):36–42, 1992.

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

[20] K. Ishida, K. Amano, and N. Kannari. A delay-constrained least-cost path routing protocol and the synthesis method. In *Proceedings of the Fifth International Conference on Real-Time Computing Systems and Applications*, pages 58 – 65. IEEE, Oct. 1998.

[21] A. Iwata et al. ATM routing algorithms with multiple QOS requirements for multimedia internetworking. *IEICE Trans. Commun.*, E79-B(8):999–1006, August 1996.

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

[23] W. C. Lee, M. G. Hluchyi, and P. A. Humblet. Routing subject to quality of service constraints in integrated communication networks. *IEEE Network*, pages 46–55, July/August 1995.

[24] L. Lovasz. Randomized algorithms in combinatorial optimization. In W. Cook, L. Lovasz, and P. Seymour, editors, *Combinatorial Optimization*, volume 20 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 153–179. American Mathematical Society, 1995.

[25] Q. Ma and P. Steenkiste. On path selection for traffic with bandwidth guarantees. In *Proceedings of the IEEE International Conference on Network Protocols (ICNP '97)*, pages 191 –202, 1997.

[26] Q. Ma and P. Steenkiste. Routing traffic with quality-of-service guarantees in integrated services networks. In *Proceedings of NOSSDAV '98*, http://www.cs.cmu.edu/∼qma/Publications.html, July 1998.

[27] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[28] A. Orda. Routing with end-to-end QoS guarantees in broadband networks. *IEEE/ACM Transactions on Networking*, 7(3):365–374, 1999.

[29] C. Pornavalai, G. Chakraborty, and N. Shiratori. QoS based routing algorithm in integrated services packet networks. In *Proceedings of ICNP '97*, pages 167–174. IEEE, 1997.

[30] H. F. Salama, D. S. Reeves, and Y. Viniotis. A distributed algorithm for delay-constrained unicast routing. In *Proceedings of the INFOCOM '97 Conference*, volume 1, pages 84–91. IEEE, 7-11 April 1997.

[31] Christopher C. Skiscim and Bruce L. Golden. Solving k-shortest and constrained shortest path problems efficiently. *Ann. Oper. Res.*, 20(1-4):249–282, 1989.

[32] N. Taft-Plotkin, B. Bellur, and R. Ogier. Quality-of-service routing using maximally disjoint paths. In *the Seventh International Workshop on Quality of Service (IWQoS '99)*, pages 119 – 128, London, England, May/June 1999. IEEE.

[33] R. Vogel et al. QoS-based routing of multimedia streams in computer networks. *IEEE Journal on Selected Areas in Communications*, 14(7):1235–1244, September 1996.

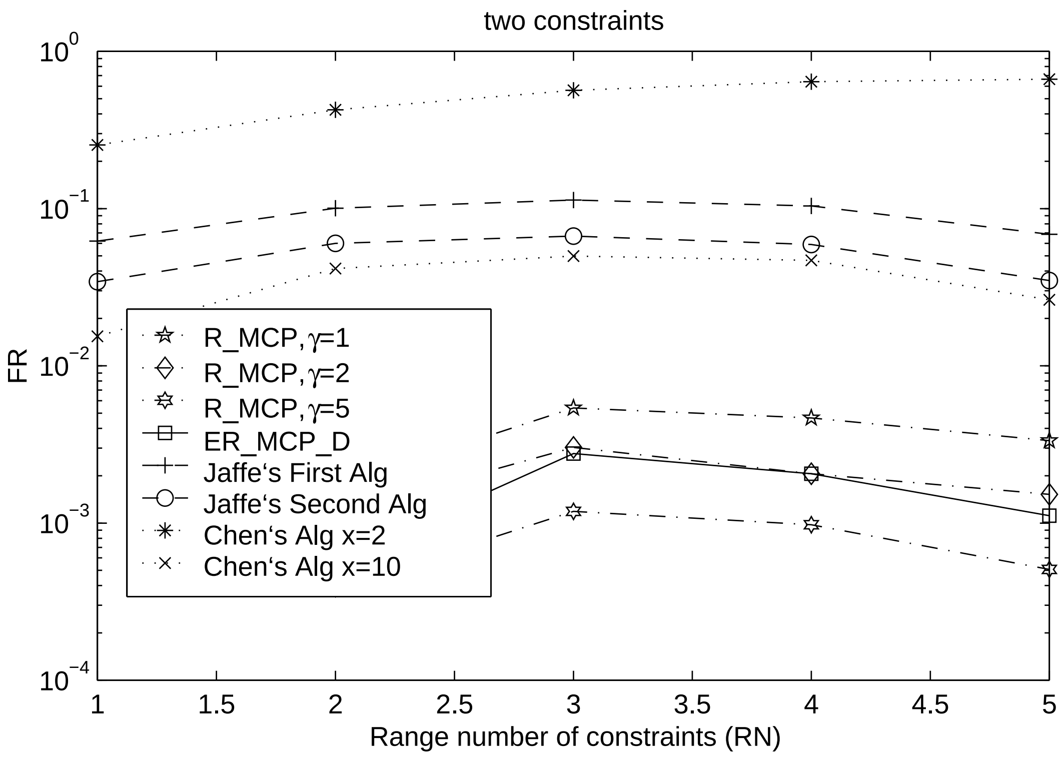
[34] Z. Wang. On the complexity of quality of service routing. *Information Processing Letters*, 69(3):111–114, 1999.

[35] Z. Wang and J. Crowcroft. Bandwidth-delay based routing algorithms. In *Proceedings of the GLOBECOM '95 Conference*, volume 3, pages 2129–2133. IEEE, Nov. 1995.

[36] Z. Wang and J. Crowcroft. Quality-of-service routing for supporting multimedia applications. *IEEE Journal on Selected Areas in Communications*, 14(7):1228–1234, September 1996.

[37] R. Widyono. The design and evaluation of routing algorithms for real-time channels. Technical Report TR-94-024, University of California at Berkeley & International Computer Science Institute, June 1994.

[38] J. Zhou. A new distributed routing algorithm for supporting delay-sensitive applications. In *Proceedings of ICCT '98*, pages S37–06(1–7). IEEE, 22-24 Oct. 1998.



(a)

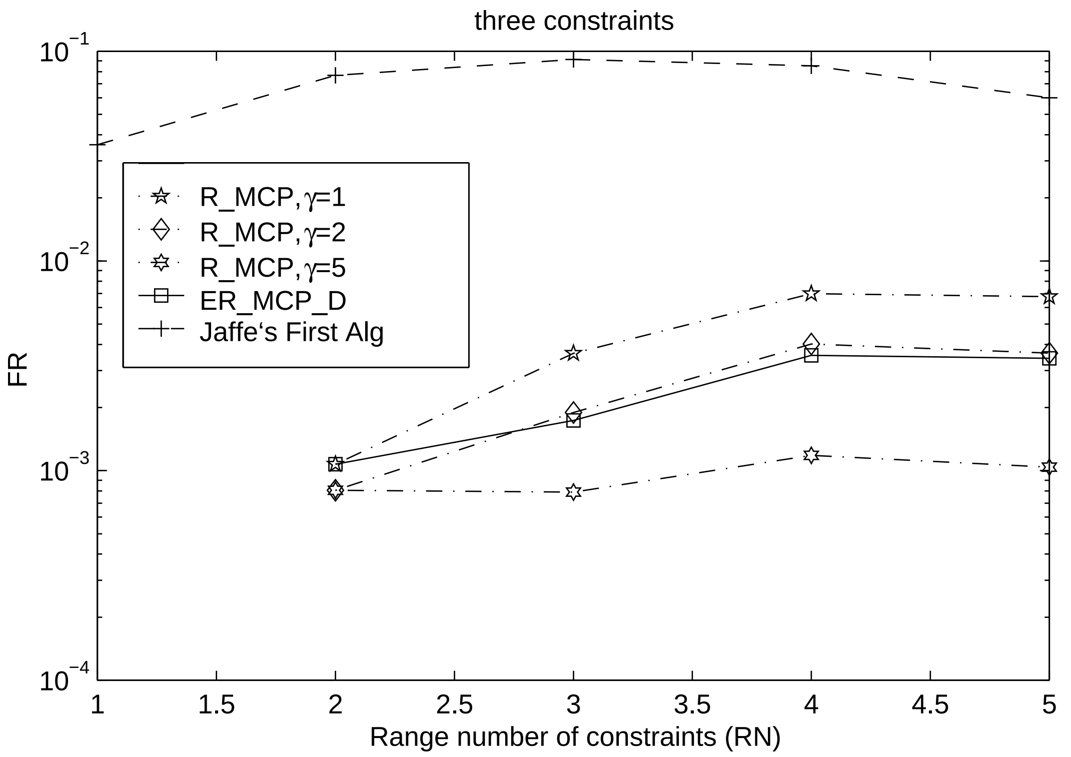
RN	Optimal	R_MCP			ER_MCP_D	Jaffe's [22]		Chen's [6]	
		$\gamma = 1$	$\gamma = 2$	$\gamma = 5$		First	Second	$x = 2$	$x = 10$
1	0.2594	0.2594	0.2594	0.2594	0.2594	0.2433	0.2505	0.1935	0.2554
2	0.5220	0.5211	0.5214	0.5218	0.5217	0.4696	0.4906	0.3004	0.5003
3	0.7595	0.7554	0.7572	0.7586	0.7574	0.6734	0.7088	0.3308	0.7216
4	0.9219	0.9176	0.9200	0.9210	0.9200	0.8260	0.8674	0.3308	0.8787
5	0.9868	0.9835	0.9853	0.9863	0.9857	0.9192	0.9524	0.3308	0.9609

(b)

RN	Optimal	R_MCP			R_MCP_D	Jaffe's [22]		Chen's [6]	
		$\gamma = 1$	$\gamma = 2$	$\gamma = 5$		First	Second	$x = 2$	$x = 10$
1	1.7784	1.7855	1.7851	1.7848	1.7892	1.7853	1.7806	1.4405	1.7821
2	2.3509	2.3904	2.3902	2.3924	2.3979	2.3912	2.3594	1.6384	2.3397
3	2.7708	2.8665	2.8759	2.8723	2.8564	2.8791	2.8095	1.6694	2.7723
4	3.0603	3.2503	3.2528	3.2531	3.1913	3.2539	3.1351	1.6694	3.0832
5	3.1742	3.4564	3.4614	3.4726	3.3435	3.4690	3.3257	1.6694	3.2627

(c)

Figure 7: Performance of different path selection algorithms under two constraints for the irregular topology in Figure 6: (a) FR, (b) SR, (c) AHC.



(a)

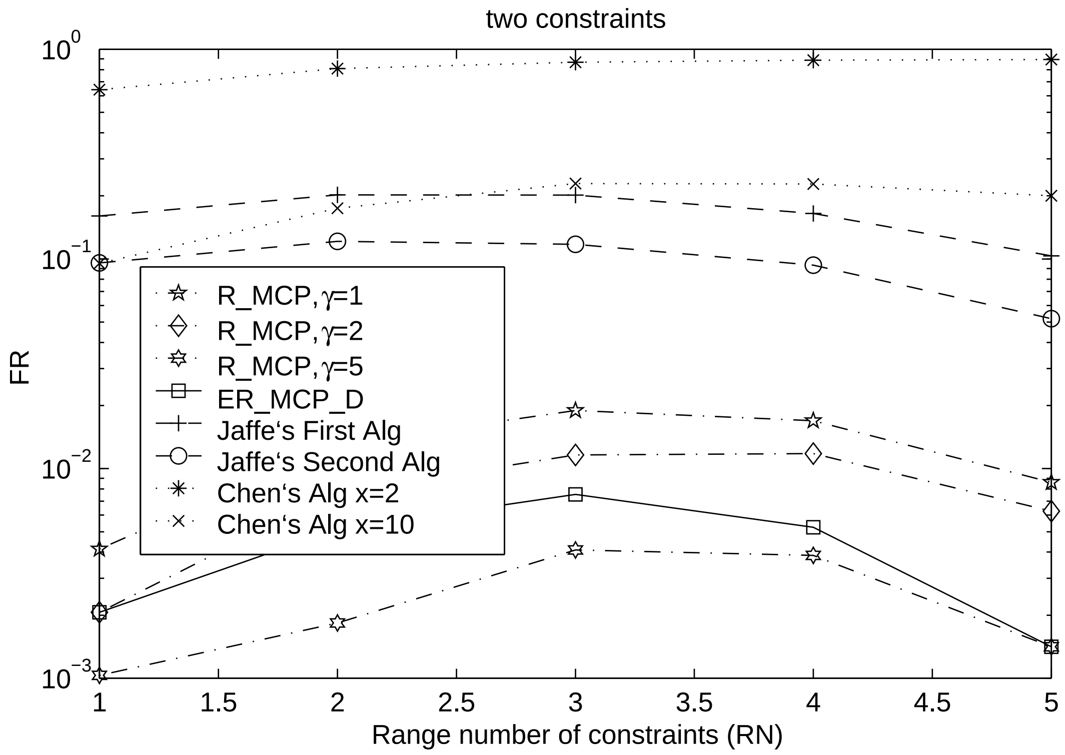
RN	Optimal	R_MCP			ER_MCP_D	Jaffe's [22]
		$\gamma = 1$	$\gamma = 2$	$\gamma = 5$		First
1	0.1758	0.1758	0.1758	0.1758	0.1758	0.1695
2	0.3727	0.3723	0.3724	0.3724	0.3723	0.3441
3	0.6336	0.6313	0.6324	0.6331	0.6325	0.5757
4	0.8462	0.8403	0.8428	0.8452	0.8432	0.7740
5	0.9618	0.9553	0.9583	0.9608	0.9585	0.9041

(b)

RN	Optimal	R_MCP			ER_MCP_D	Jaffe's [22]
		$\gamma = 1$	$\gamma = 2$	$\gamma = 5$		First
1	1.4546	1.4556	1.4551	1.4553	1.4561	1.4403
2	2.0018	2.0171	2.0172	2.0200	2.0135	1.9643
3	2.5053	2.5640	2.5633	2.5649	2.5387	2.4673
4	2.8925	3.0171	3.0201	3.0271	2.9459	2.8671
5	3.1142	3.3342	3.3360	3.3430	3.1928	3.1419

(c)

Figure 8: Performance of different path selection algorithms under three constraints for the irregular topology in Figure 6: (a) FR, (b) SR, (c) AHC.



(a)

RN	Optimal	R_MCP			ER_MCP_D	Jaffe's [22]		Chen's [6]	
		$\gamma = 1$	$\gamma = 2$	$\gamma = 5$		First	Second	$x = 2$	$x = 10$
1	0.2905	0.2893	0.2899	0.2902	0.2899	0.2439	0.2627	0.1038	0.2628
2	0.5450	0.5379	0.5408	0.5440	0.5422	0.4347	0.4788	0.1043	0.4498
3	0.7824	0.7676	0.7733	0.7792	0.7765	0.6245	0.6906	0.1043	0.6033
4	0.9333	0.9175	0.9223	0.9297	0.9284	0.7793	0.8461	0.1043	0.7205
5	0.9895	0.9810	0.9833	0.9881	0.9881	0.8870	0.9382	0.1043	0.7912

(b)

RN	Optimal	R_MCP			ER_MCP_D	Jaffe's [22]		Chen's [6]	
		$\gamma = 1$	$\gamma = 2$	$\gamma = 5$		First	Second	$x = 2$	$x = 10$
1	3.1130	3.1552	3.1437	3.1457	3.1372	2.9521	3.0268	1.6359	2.9101
2	4.3724	4.4796	4.4796	4.4921	4.4301	4.0181	4.1804	1.6376	3.8319
3	5.4707	5.6586	5.6717	5.7048	5.5630	4.9776	5.2239	1.6376	4.5584
4	6.2093	6.5194	6.5322	6.5794	6.3451	5.7281	5.9739	1.6376	5.0727
5	6.5365	6.9882	7.0173	7.0512	6.7119	6.2641	6.4520	1.6376	5.4084

(c)

Figure 9: Performance of different path selection algorithms under two constraints for the regular 10x10 mesh topology: (a) FR, (b) SR, (c) AHC.