An Efficient Algorithm for Finding a Path Subject to Two Additive Constraints*

Turgay Korkmaz
Dept. of Elec. & Comp. Eng.
University of Arizona
Tucson, AZ 85721
turgay@ece.arizona.edu

Marwan Krunz
Dept. of Elec. & Comp. Eng.
University of Arizona
Tucson, AZ 85721
krunz@ece.arizona.edu

Spyros Tragoudas
Dept. of Elec. & Comp. Eng.
University of Illinois
Carbondale, IL
spyros@engr.siu.edu

Abstract

One of the key issues in providing end-to-end quality-of-service (QoS) guarantees in packet networks is how to determine a *feasible* route that satisfies a set of constraints while simultaneously maintaining high utilization of network resources. In general, finding a path subject to multiple *additive* constraints (e.g., delay, delay-jitter) is an NP-complete problem that cannot be exactly solved in polynomial time. Accordingly, heuristic and approximation algorithms are proposed to this problem. However, these algorithms suffer from either excessive computational cost or low performance. In this paper, we provide an efficient approximation algorithm for finding a path subject to two additive constraints. The worst-case computational complexity of this algorithm is within a logarithmic number of calls to Dijkstra's shortest path algorithm. In practice, the average complexity of the algorithm is even much lower, since the worst-case complexity is rarely needed. The performance of the proposed algorithm is justified via theoretical bounds that are provided for the optimal version of the path selection problem. To achieve further performance improvement, several extensions to the basic algorithm are also provided at very low computational cost. Extensive simulations are used to demonstrate the high performance of the proposed algorithm and to contrast it with other path selection algorithms.

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

1 Introduction

Integrated network services (e.g., ATM, Intserv, Diffserv) are being designed to provide quality-of-service (QoS) guarantees to various applications such as audio, video, and data. Many of these applications have multiple QoS requirements in terms of bandwidth, delay, delay-jitter, loss, etc. One of the important problems in QoS-based service offerings is how to determine a route that satisfies multiple constraints (or QoS requirements) while simultaneously achieving efficient utilization of network resources. This problem is known as QoS-based (or constraint-based) routing, and is being extensively investigated in the research community [8, 25, 13, 4, 37, 33, 22, 24].

In general, routing consists of two basic tasks: distributing the state information of the network and searching this information for a feasible path with respect to (w.r.t.) given constraints. In this paper, we focus on the second task, and assume that the true state of the network is available to every node (e.g., via link-state routing) and that nodes use this state information to determine an end-to-end feasible path (see [17] for QoS routing under inaccurate information). Each link in the network is associated with multiple QoS parameters. These parameters can be classified into additive and non-additive [2, 34]. For additive constraints (e.g., delay), the cost of an end-to-end path is given, exactly or approximately, by the *sum* of the individual link parameters (or weights) along that path. In contrast, the cost of a path w.r.t. a non-additive constraint (such as bandwidth) is determined by the value of that constraint at the bottleneck link. Non-additive constraints can be easily dealt with as a preprocessing step by pruning all links that do not satisfy the requested non-additive QoS values [35]. Hence, in this paper we will mainly focus on additive constraints. The underlying problem of path selection subject to two additive constraints can be stated as follows.

Definition 1 *Multiple Constrained Path Selection (MCP)*: Consider a network that is represented by a directed graph $G = (V, E)$, where V is the set of nodes and E is the set of links. Each link $(u, v) \in E$ is associated with two non-negative additive QoS values: $w_1(u, v)$ and $w_2(u, v)$. Given two constraints c_1 and c_2, the problem is to find a path p from a source node s to a destination node t such that $w_1(p) \leq c_1$ and $w_2(p) \leq c_2$, where $w_1(p) \stackrel{\text{def}}{=} \sum_{(u,v)\in p} w_1(u, v)$ and $w_2(p) \stackrel{\text{def}}{=} \sum_{(u,v)\in p} w_2(u, v)$.

The MCP decision problem is known to be NP-complete [16, 23]. In other words, there is no efficient (polynomial-time) algorithm that can *surely* find a feasible path which simultaneously satisfies both constraints. A related yet slightly different problem is known as the restricted shortest path (RSP) problem, in which the returned path is required to

*This work was supported by the National Science Foundation under Grant ANI 9733143 and Grant CCR 9815229. Authors names are listed in alphabetical order.

satisfy one constraint while being optimal w.r.t. another parameter. Any solution to the RSP problem can be also applied to the MCP problem. However, the RSP problem is also known to be NP-complete [16, 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 [23, 20]. However, these algorithms are computationally expensive if the values of the link weights and the size of network are large. To cope with the NP-completeness of these problems, researchers have resorted to several heuristics and approximation algorithms.

One common approach to the RSP problem is to find the k-shortest paths w.r.t. a cost function defined based on the link weights and the given constraint, hoping that one of these paths is feasible and near-optimal [19, 31, 15, 18]. The value of k determines the performance and overhead of this approach; if k is large, the algorithm has good performance but its computational cost is prohibitive. A similar approach to the k-shortest paths is to *implicitly* enumerate all feasible paths [3], but this approach is also computationally expensive. In [36] the author proposed the Constrained Bellman-Ford (CBF) algorithm. 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 was improved in [38, 21]. In [20] 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 expensive in large networks [28]. Accordingly, the author in [28] investigated the hierarchical structure of such networks and provided a new approximation algorithm with better scalability.

Although both the RSP and MCP problems are NP-complete, the latter problem seems to be easier than the former in the context of devising approximate solutions. Accordingly, in [23] 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. $l(e) \stackrel{\text{def}}{=} \alpha w_1(p) + \beta w_2(p)$ 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 [23] Jaffe determined two sets of values for α and β based on minimizing an objective function of the form $f(p) = \max\{w_1(p), c_1\} + \max\{w_2(p), c_2\}$. For the RSP problem, the authors in [6] proposed a similar approximation algorithm that dynamically adjusts the values of α and β. However, the computational complexity of this algorithm grows exponentially with the size of the network. Chen and Nahrstedt proposed another heuristic algorithm that modifies the problem by scaling down the values of one link weights to bounded integers [7]. 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. To achieve a high probability of finding a feasible path, x needs to be as large as $10n$, resulting in computational complexity of $\mathcal{O}(n^4)$. In [14] Neve and Mieghem used the k-shortest paths algorithm in [9] with a nonlinear cost function to solve the MCP problem. The resulting algorithm, called TAMCRA, has the complexity of $\mathcal{O}(kn\log(kn) + k^3Km)$, where K is the number of constraints. As mentioned above, the performance and overhead of this algorithm depend on k.

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) [35]. Several path selection algorithms based on different combinations of bandwidth, delay, and hop-count were discussed in [27, 26, 5] (e.g., widest-shortest path, shortest-widest path). In addition, new algorithms were proposed to find more than one feasible path w.r.t. bandwidth and delay (e.g., Maximally Disjoint Shortest and Widest Paths) [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 [26, 29]. Specifically, if Weighted Fair Queueing (WFQ) scheduling 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 representing all the constraints in terms of 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 [10].

Contributions and Organization of the Paper

Previously proposed algorithms suffer from either excessive computational complexities or low performance in finding feasible paths. We first provide an efficient approximation algorithm for the MCP problem under two additive constraints in Section 3. Indeed, our algorithm is based on the minimization of the same linear cost function $\alpha w_1(p) + \beta w_2(p)$ presented in [23]. This formulation is similar to that used in the Lagrange relaxation technique. However, this technique provides a general platform, rather than a solution, by formulating constrained optimization problems as a linear composition of constraints. The solution to the Lagrange problem requires searching for the appropriate linear composition (Lagrange multipliers); the appropriate values of α and β in our case. Any combinatorial algorithm (heuristic) that has been or will be proposed for linear optimization problems is a careful refinement of the search for the appropriate multipliers in the Lagrangian problem. When formulated as a Lagrangian multipliers problem, the search would typically be based on computationally expensive methods, such as enumeration, linear programming, and subgradient optimization [1]. Instead, we *provide a binary search strategy* to find the appropriate value of k in the composite function $w_1(p) + kw_2(p)$ or $kw_1(p) + w_2(p)$ within a logarithmic number of calls to Dijkstra's algorithm. This fast search is one of the main contributions in the paper. The algorithm always returns a path p. If p is not feasible, then it has the following properties: (a) $w_j(p) \leq c_j$, and (b) $w_i(p)$ is within a given factor from a feasible path f for which $w_i(f)$ is minimum, where (i, j) are either $(1, 2)$ or $(2, 1)$. Our basic algorithm performs a binary search in the range $[1, B]$ by calling a hierarchical version of Dijkstra's algorithm, which is described in Section 2. Using an efficient implementation of Dijkstra's al-

gorithm with complexity of $\mathcal{O}(m+n\log n)$ [1], the worst-case complexity of our basic algorithm is $\mathcal{O}(\log B(m+n\log n))$. Its average complexity is observed to be much less than that. The space complexity is $\mathcal{O}(n)$. By proper interpretation of the bounds in (a) and (b), we also present two extensions to our basic algorithm in Section 4 to achieve further improvement in routing performance at small extra computational cost. Simulation results, which are provided in Section 5, demonstrate the high performance of our algorithm and contrast it with other path selection algorithms. Conclusions and future work are presented in Section 6.

2 Hierarchical Shortest Path Algorithm

In this section, we describe a hierarchical version of Dijkstra's shortest path algorithm that is used iteratively in our algorithm with a composite link weight $l(e) \stackrel{\text{def}}{=} \alpha w_1(e) + \beta w_2(e)$. In addition to finding one of the shortest paths w.r.t. $l(e)$, the hierarchical version of Dijkstra's algorithm determines the minimum $w_1()$ and $w_2()$ among all shortest paths. To carry out these tasks, some modifications are needed in the relaxation process of the standard Dijkstra's algorithm (lines 4–14 in Figure 1). The standard Dijkstra's

```
Relax(u,v)
1  if d[v] > d[u] + l(u,v) then
2      d[v] := d[u] + l(u,v)
3      π[v] := u
4      w1[v] := w1[u] + w1(u,v)
5      w2[v] := w2[u] + w2(u,v)
6      min_w1[v] := w1[v]
7      min_w2[v] := w2[v]
8  else if d[v] = d[u] + l(u,v) then
9      if min_w1[v] > min_w1[u] + w1(u,v) then
10         min_w1[v] := min_w1[u] + w1(u,v)
11     end if
12     if min_w2[v] > min_w2[u] + w2(u,v) then
13         min_w2[v] := min_w2[u] + w2(u,v)
14     end if
15 end if
```

Figure 1: New relaxation procedure for the hierarchical version of Dijkstra's algorithm.

algorithm maintains two labels for each node [12]: $d[u]$ to represent the estimated total cost of the shortest path from the source node s to node u w.r.t. the composed weight $l(e)$, and $\pi[u]$ to represent the predecessor of node u along the shortest path. The hierarchical version of Dijkstra's algorithm maintains additional labels: $w_1[u]$ and $w_2[u]$ to represent the cost of the shortest path w.r.t. the individual weights, and labels $min\_w_1[u]$ and $min\_w_2[u]$ to represent the minimum w_1 and w_2 weights among all shortest paths.[1] The standard relaxation process (lines 1–3 in Figure 1) tests whether the shortest path found so far from source node s to node v can be improved by passing through node u. If so, $d[v]$ and $\pi[v]$ are updated [12]. Under this condition, we add the update of $w_1[v]$, $w_2[v]$, $min\_w_1[v]$, and $min\_w_2[v]$. In addition, if the cost of the shortest path found so far from node s to node v is the same as that of the path passing through node u, then $min\_w_1[v]$ and $min\_w_2[v]$ are also updated if passing through node u would improve their values.

[1]Notice that $w_i[.]$ is a node label, whereas $w_i(.)$ indicates the weight of a link or the cost of a path.

3 Basic Approximation Algorithm For MCP

Our algorithm, shown in Figure 2, first executes the hierarchical version of Dijkstra's algorithm with link weight $l(e) = w_1(e) + w_2(e)$, i.e., $\alpha = 1$ and $\beta = 1$. If p is feasible, then the algorithm terminates. Otherwise, p is not feasible and several other cases need to be considered. If both $w_1(p) > c_1$ and $w_2(p) > c_2$, then it is guaranteed that there is no feasible path [23], so the algorithm terminates. If $min\_w_1[t] \le c_1$ or $min\_w_2[t] \le c_2$, then there might be a feasible path that can be found using different values of α and β. The challenging problem is how to determine appropriate values for α and β as fast as possible so that a feasible path can be identified quickly. Finding the appropriate values for α and β can also be formulated as a Lagrangian multipliers problem. But in this case, finding the Lagrange multipliers would typically be done using computationally expensive methods (e.g., enumeration, linear programming, subgradient optimization technique) [1]. Instead, we carefully refine the search required by the Lagrangian problem and provide a binary search strategy for α and β that is guaranteed to terminate within a logarithmic number of calls to Dijkstra's algorithm.

```
BasicApproximation(G(V,E), s, t, c1, c2)
// Find a path p from s to t in the network G = (V,E)
// such that w1(p) = w1[t] ≤ c1 and w2(p) = w2[t] ≤ c2.
1  Set l(e) := w1(e) + w2(e) ∀e ∈ E
2  Execute hierarchical Dijkstra's algorithm
       with link weights {l(e) : e ∈ E}
3  if w2[t] ≤ c2 and w1[t] ≤ c1 then
4      return SUCCESS
5  end if
6  if w1[t] > c1 and w2[t] > c2 then
7      return FAILURE
8  end if
9  if min_w2[t] ≤ c2 then
10     Execute Binary_Search(i = 1, j = 2) /* Phase 1 */
11 else if min_w1[t] ≤ c1 then
12     Execute Binary_Search(i = 2, j = 1) /* Phase 2 */
13 end if
end BasicApproximation
```

Figure 2: Approximation algorithm for finding a feasible path subject to two additive constraints.

If $min\_w_1[t] \le c_1$ or $min\_w_2[t] \le c_2$, then the algorithm executes the binary search presented in Figure 3 with $(i = 1, j = 2)$ or $(i = 2, j = 1)$. These two cases are called Phase 1 and Phase 2. In Phase 1, the algorithm executes the binary search using link weight $l(e) = kw_1(e) + w_2(e)$, i.e., $\alpha = k$ and $\beta = 1$. In Phase 2, the algorithm executes the binary search using link weight $l(e) = w_1(e) + kw_2(e)$, i.e., $\alpha = 1$ and $\beta = k$. If the returned shortest path w.r.t. $l(e)$ is not feasible, the algorithm repeats the hierarchical Dijkstra's algorithm up to a logarithmic number of different values of k in the range $[1, B]$, where $B = n \cdot \max\{w_j(e) \mid e \in E\}$,

```
Binary_Search(i, j)
1  k_min := 1
2  k_max := n · max{w_j(e) | e ∈ E}
3  while( k_min <= k_max ) do
4      k := ⌈√(k_min * k_max)⌉
5      Set l(e) := kw_i(e) + w_j(e) ∀e ∈ E
6      Execute hierarchical Dijkstra's algorithm
       with link weights {l(e) : e ∈ E}
7      if w_1[t] ≤ c_1 and w_2[t] ≤ c_2 then
8          return SUCCESS
9      end if
10     if min_w_j[t] ≤ c_j then
11         k_min := k + 1        /* k will be increased */
12     else
13         k_max := k − 1        /* k will be decreased */
14     end if
15 end while
end Binary_Search
```

Figure 3: Binary search for our approximation algorithm.

which is an upper bound on the total cost of the longest path w.r.t. link weight w_j. Lemma 1 in Section 3.2 shows that a binary search argument in the above range can be used to determine an appropriate value for k. Furthermore, we show (in Lemma 2) that if the binary search fails to return a feasible path, then it returns a path p such that $w_j(p) \leq c_j$ and $w_i(p) \leq w_i(f) + (w_j(f) - w_j(p))/k$, where f is some feasible path and (i, j) are either $(1, 2)$ or $(2, 1)$. This is a reasonable scenario for searching fast for a feasible path that satisfies one of the constraints and that is very likely to satisfy the other constraint. According to this bound, k needs to be maximized; the above binary search tries to achieve this goal. In addition to maximizing k, the algorithm may attempt to minimize the difference $(w_j(f) - w_j(p))$ to make the approximation bound tighter. This is an extension to the basic algorithm that is presented in Section 4.

3.1 How the Basic Algorithm Works

The systematic adjustment of k is illustrated in the examples in Figures 4 and 5 for two different phases. The shaded area indicates the feasibility region. Black dots represent the costs of different paths from source node s to destination node t. Each line in the figure shows the equivalence class of equal-cost paths w.r.t. the composed weight. The approximation algorithm determines a line for the given value of k, and then moves this line outward from the origin in the direction of the arrow. Whenever this line hits a path (i.e., black dot in the figure), the algorithm returns this path which is the shortest w.r.t. the composed weight at the given k. The approximation algorithm in [23] makes a good guess for k (e.g., $k = 1$) and returns a path based on this k. However, if this path is infeasible the algorithm in [23] cannot proceed. As shown in Figures 4 and 5, the likelihood of finding a feasible path is much higher if one tries different values of k (e.g., $k = 4$ in these examples results in a feasible path). The advantage of our algorithm over the one in [23] is that ours searches systematically for a good value for k instead of fixing it in advance. If the returned path p is not feasible, then the algorithm decides to increase or decrease the value of k based on whether $min\_w_j(p) \leq c_j$ or not. Figure 4 illustrates Phase 1 where the returned path with

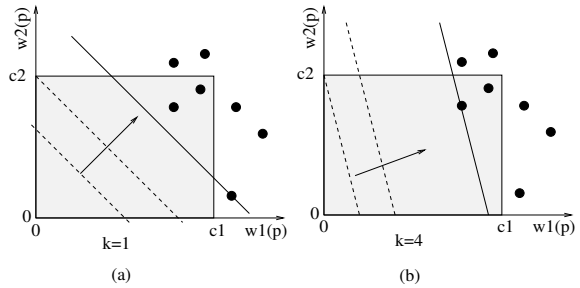

Figure 4: Searching for a feasible path in Phase 2.

$k = 1$ satisfies c_2 but not c_1. The algorithm executes the binary search with $i = 1$ and $j = 2$ and returns a feasible path when $k = 4$, as shown in Figure 4(b). Figure 5 illustrates Phase 2 where the returned path with $k = 1$ satisfies c_1 but not c_2. In this case, the algorithm executes the binary

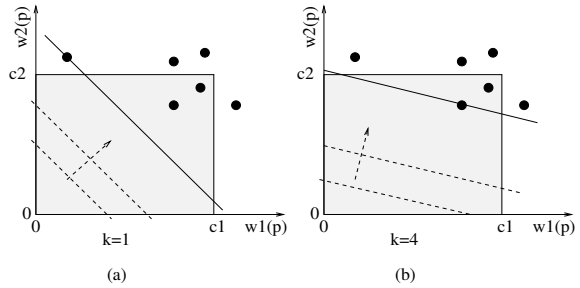

Figure 5: Searching for a feasible path in Phase 2.

search with $i = 2$ and $j = 1$, and finally returns a feasible path when $k = 4$. If the binary search fails, then the basic algorithm stops even though there might be a feasible path in the network. In Section 4, we illustrate such a case and provide possible remedies to it based on a scaling extension.

3.2 Binary Search

Lemma 1 *Suppose that each link $e \in E$ is assigned a weight $l(e) = kw_i(e) + w_j(e)$, where k is an integer, and the pair (i, j) is either $(1, 2)$ or $(2, 1)$, depending on the phase. During the execution of the binary search, if the algorithm cannot find a path p for which $l(p)$ is minimum and $w_j(p) \leq c_j$, then such a path p cannot be found with larger values of k.*

Proof of Lemma 1: The binary search is applied to find the largest k such that there exists a shortest path p w.r.t. $l(e) = kw_i(e) + w_j(e)$ with $w_j(p) \leq c_j$. Assume that $k = 2r$ for some integer r. Let $\mathcal{P}$ be the set of all paths from s to t w.r.t. $l(e)$ and let p be a path that the algorithm selects during the binary search. When $k = 2r$, since all edges are assigned weights $l(e) = 2rw_i(e) + w_j(e)$, we have

$$l(p) = \min_{q \in \mathcal{P}} \{ \sum_{e \in q} 2rw_i(e) + w_j(e) \}.$$

In order to prove the lemma, it suffices to show that if

$$\sum_{e \in p} w_j(e) > c_j$$

then the algorithm should never search for a path p' that satisfies the c_j constraint by assigning $l(e) = rw_i(e) + w_j(e)$.

By explicitly checking $min\_w_j[t]$ in line 10 of Figure 3, the algorithm guarantees that $\sum_{e \in q} w_j(e) > c_j$ for all shortest paths $q \in \mathcal{P}$, where $l(q) = l(p)$. Thus, it suffices to show that if the algorithm assigns weights $l(e) = 2rw_i(e) + w_j(e)$ and fails to find a feasible path w.r.t. constraint c_j, then no path p' for which

$$\sum_{e \in p'} 2rw_i(e) + w_j(e) > \sum_{e \in p} 2rw_i(e) + w_j(e) \quad (1)$$

will satisfy both

$$\sum_{e \in p'} rw_i(e) + w_j(e) < \sum_{e \in p} rw_i(e) + w_j(e)$$

and

$$\sum_{e \in p'} w_j(e) < c_j$$

when the value of k is reduced to r. In other words, it is useless to weight with the rule $l(e) = rw_i(e) + w_j(e)$ in order to search for a path p' whose $\sum_{e \in p'} 2rw_i(e) + w_j(e)$ is not minimum but satisfies the c_j constraint.

Since path p violates the c_j constraint, in order for path p' to satisfy this constraint, we must have:

$$\sum_{e \in p} w_j(e) - \sum_{e \in p'} w_j(e) > 0. \quad (2)$$

Observe that Equation (1) can be rewritten as

$$2r \left(\sum_{e \in p'} w_i(e) - \sum_{e \in p} w_i(e) \right) > \sum_{e \in p} w_j(e) - \sum_{e \in p'} w_j(e)) \quad (3)$$

From (3) and (2), we have

$$\sum_{e \in p'} w_i(e) - \sum_{e \in p} w_i(e) > 0. \quad (4)$$

Based on (2) and (4), we know that the right-hand side and the left-hand side of the inequality (3) are positive. Thus, it can be implied that

$$r \left(\sum_{e \in p'} w_i(e) - \sum_{e \in p} w_i(e) \right) > \sum_{e \in p} w_j(e) - \sum_{e \in p'} w_j(e)) \quad (5)$$

from which we conclude that

$$\sum_{e \in p'} rw_i(e) + w_j(e) > \sum_{e \in p} rw_i(e) + w_j(e)$$

This, in turn, implies that p' will not be selected by the algorithm, and thus the claim about the optimality of the proposed binary search scheme is true. ■

3.3 Performance Bounds

Lemma 2 *If the binary search fails to return a feasible path w.r.t. both constraints, then it returns a path p that satisfies the c_j constraint and whose $w_i()$ cost is upper bounded as follows:*

$$w_i(p) \leq w_i(f) + (w_j(f) - w_j(p))/k$$

where f is a feasible path, k is the maximum value that the binary search determines at the termination, and the pair (i, j) is either $(1, 2)$ or $(2, 1)$, depending on the phase.

Note that the worst-case approximation bound for Lemma 2 is obtained when $k = 1$ and $w_j(p) = 0$. In this case, the bound will be

$$w_i(p) \leq w_i(f) + w_j(f) \leq c_1 + c_2 \quad (6)$$

Clearly, since this bound is obtained in the first step where the algorithm starts with $k = 1$ and decides which composite cost function needs to be used in the binary search, this bound holds for the overall algorithm. Note that in this case the only feasible path happens to be on the far corner of the feasibility region and the other paths have $w_i(p) = c_1 + c_2$ and $w_j(p) = 0$. In most cases, feasible paths are scattered all around in the feasibility region, allowing the algorithm to return a larger k, which in turn results in a tighter bound than (6). In addition, $w_j(p)$ is often greater than zero, further tightening the bound.

Proof of Lemma 2: Let f be a feasible path for which $w_i(f)$ is the smallest possible among all feasible paths, where i is either 1 or 2, depending on the phase. Furthermore, assume that f is not dominated by any other feasible path. (We say that a feasible path dominates another path if it has shorter costs w.r.t. both $w_1()$ and $w_2()$ weights.) This implies that $w_j(f) > w_j(f')$ for any other feasible path f'. We know that $w_j(f')$ cannot be more than c_j and it is less than $w_j(p)$; otherwise, the algorithm would have returned f'. Assume that the path p is infeasible. Since it is the shortest path, we have

$$kw_i(p) + w_j(p) \leq kw_i(f) + w_j(f). \quad (7)$$

In addition, the path p satisfies the constraint c_j. From (7), we can write a bound on $w_i(p)$ as follows

$$w_i(p) \leq w_i(f) + (w_j(f) - w_j(p))/k$$

■

4 Extensions of the Basic Algorithm

4.1 Finding a Path with the Closest Cost to a Constraint

From Lemma 2, it is clear that one way to improve the performance of the basic algorithm is to minimize the difference $(w_j(f) - w_j(p))$. This minimization can be achieved by obtaining a path p for which $w_j(p)$ is as close as possible to c_j. This can be done with the following modification to the basic algorithm of Section 3. Without loss generality, we assume that $i = 1$ and $j = 2$. Note that this extension must be used if the returned path is not feasible but both $min\_w_1[t] \leq c_1$ and $min\_w_2[t] \leq c_2$.

For the given k, a DAG (directed acyclic graph) that contains all possible shortest paths w.r.t $l(e)$ is constructed. In fact, this can be done during the execution of the hierarchical Dijkstra's algorithm at no extra cost. A path p from this DAG is selected in such a way that $w_2(p)$ is maximized, but it is still less than or equal to c_2. Although, a path p with the maximum or minimum $w_2()$ weight can be found

in the DAG, it is not easy to find a path p for which $w_2(p)$ is as close as possible to c_2 in polynomial-time. However, very efficient heuristics can be developed based on the fact that we can compute the maximum and the minimum $w_2()$ from the source to every node and from every node to the destination. Let the following labels be maintained for each node u: $M[u]$, $m[u]$, $\widetilde{M}[u]$, and $\widetilde{m}[u]$. Labels $M[u]$ and $m[u]$ indicate, respectively, the maximum and minimum $w_2()$ from the source to every node u. Labels $\widetilde{M}[u]$ and $\widetilde{m}[u]$ indicate, respectively, the maximum and minimum $w_2()$ from every node u to the destination. Labels $M[u]$, $m[u]$, $\widetilde{M}[u]$ and $\widetilde{m}[u]$ are determined by using a simple forward and backward topological traversal algorithm [1]. Considering the pairwise sum of these labels as follows, we can estimate a non-additive weight $\sigma(u,v)$ for every link (u,v) in the DAG to indicate how close $w_2()$ of the paths passing through the link (u,v):

$$\sigma(u,v) = \text{min\_non\_neg} \left\{ \begin{array}{l} c_2 - (M[u] + w_2(u,v) + \widetilde{M}[u[v]), \\ c_2 - (M[u] + w_2(u,v) + \widetilde{m}[u[v]), \\ c_2 - (m[u] + w_2(u,v) + \widetilde{M}[u[v]), \\ c_2 - (m[u] + w_2(u,v) + \widetilde{m}[u[v]), \\ +\infty \end{array} \right\}$$

where min_non_neg returns the minimum non-negative value. Then, the closest path to c_2 can be found via a simple graph traversal algorithm as follows. Starting from the source node s, the algorithm selects the link (s,u) with the minimum σ. It then goes to node u and again selects the link (u,v) with the minimum σ. The algorithm keeps selecting links with minimum σ until it hits t.

Figure 6 depicts an example of how a DAG of shortest paths is constructed. The original network is shown in Figure 6(a). Suppose a path p is to be found from s to t such that $w_1(p) \leq c_1 = 10$ and $w_2(p) \leq c_2 = 10$. Consider the

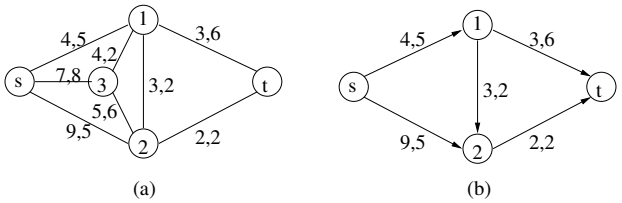

Figure 6: An example of a network and the DAG containing the three shortest paths from s to t.

case when $k = 1$, i.e., the algorithm minimizes $w_1(p)+w_2(p)$. There are three shortest paths from s to t: $p_1 =< s,1,t >$ with $w_1(p_1) = 7$ and $w_2(p_1) = 11$, $p_2 =< s,2,t >$ with $w_1(p_2) = 11$ and $w_2(p_2) = 7$, and $p_3 =< s,1,2,t >$ with $w_1(p_3) = 9$ and $w_2(p_3) = 9$. For these shortest path, both $min\_w_1[t] = 7$ and $min\_w_2[t] = 7$ are less than the respective constraints, so we can apply this extension. The corresponding DAG which contains all these shortest paths is shown in Figure 6(b). By traversing forward and backward on this DAG, we compute the labels $M[u]$, $m[u]$, $\widetilde{M}[u]$, and $\widetilde{m}[u]$ (see Figure 7(a)). After calculating σ for each link as shown in Figure 7(b), the algorithm first selects link $(s,1)$, followed by link $(1,2)$, and finally link $(2,t)$. Thus, the closest path p_3 is found. Since this heuristic step tends to minimize the additive difference in the approximation bound presented in Lemma 2, the returned path p is very likely to satisfy both c_1 and c_2.

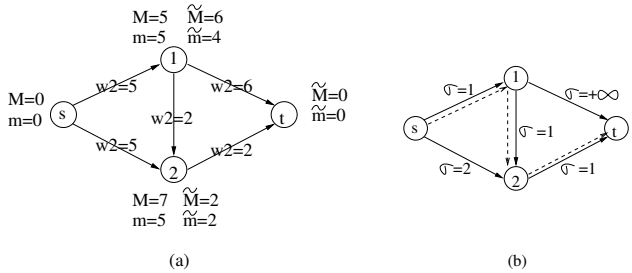

Figure 7: Finding the closest path to c_2.

4.2 Scaling

In some pathological cases, any algorithm using the linear composite cost fails to return a feasible path that does exist. We illustrate this situation by an example. Again we assume that $i = 1$ and $j = 2$ without loss generality. Consider the network in Figure 8(a). Suppose a path p is to be found from s to t such that $w_1(p) \leq c_1 = 10$ and $w_2(p) \leq c_2 = 10$. As shown in Figure 8(b), there are three paths from s to t:

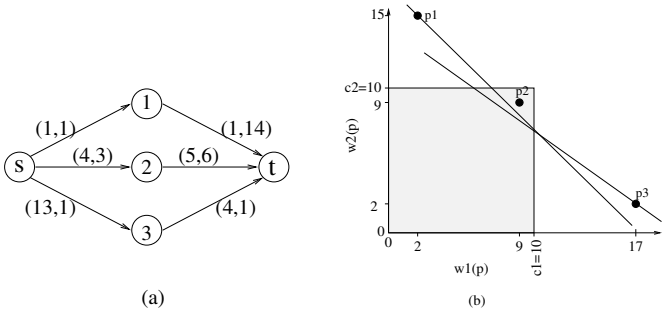

Figure 8: A scenario in which the basic algorithm fails to find a feasible path from s to t.

$p_1 =< s,1,t >$ with $w_1(p_1) = 2$ and $w_2(p_1) = 15$, $p_2 =< s,2,t >$ with $w_1(p_2) = 9$ and $w_2(p_2) = 9$, and $p_3 =< s,3,t >$ with $w_1(p_3) = 17$ and $w_2(p_3) = 2$. Only p_2 is feasible. The approximation algorithm returns a path based on the minimization of the composed weight $l(p) = w_1(p) + kw_2(p)$. To return the feasible path p_2, the algorithm needs to find an appropriate value for k such that $l(p_2)$ is less than both $l(p_1)$ and $l(p_3)$. Hence, the value of k needs to be greater than $7/6$ to satisfy $(l(p_2) = 9 + 9k) < (l(p_1) = 2 + 15k)$ and also less than $8/7$ to satisfy $(l(p_2) = 9 + 9k) < (l(p_3) = 17 + 2k)$. However, it is impossible to find a value for k such that $7/6 < k < 8/7$.

To circumvent such pathological cases, we provide an extension to our basic algorithm based on the scaling in [7]. A new weight $w_2'(e)$ is assigned to every link in the original graph as follows:

$$w_2'(e) = \left\lceil \frac{w_2(e) \cdot x}{c_2} \right\rceil \qquad (8)$$

where x is an adjustable positive integer in the range $[1, c_2]$. The problem reduces to finding a path in the scaled graph such that $w_1(p) \leq c_1$ and $w_2'(p) \leq x$. It has been shown that a solution in the scaled graph is also a solution in the original one [7]. If we scale the network in Figure 8(a) by $x = 3$, the scaled graph is shown in Figure 9(a). If the approximation algorithm uses the composed weights $l_2(p) = w_1(p) + kw_2'(p)$

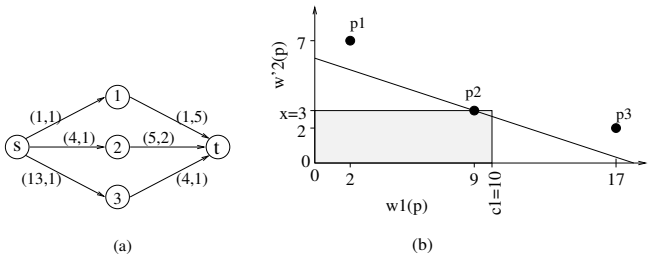

Figure 9: Scaling the network in Figure 8 by $x = 3$ allows the algorithm to find a feasible path.

in the scaled graph with $k = 3$, it will return the feasible path p_2 (see Figure 9(b)), since $l(p_2) = 18$ is less than both $l(p_1) = 20$ and $l(p_3) = 23$.

Using the above scaling function, one may increase the number of shortest paths in the scaled graph. If we apply our basic approximation algorithm to the scaled graph, the algorithm will consider more shortest paths (in the scaled graph) in each iteration of the binary search. It is intuitively true that the algorithm will terminate with a better (i.e., larger) value of k. It is important to note that in contrast to the algorithm in [7], the value of x does not effect the complexity of our algorithm. Choosing x as small as possible may increase the number of shortest paths as desired. However, this also decreases the number of paths for which $w_2'(p) \leq x$, i.e., the algorithm may not return a feasible path. The tradeoff between the value of x and the associated performance improvement after scaling by x is shown in Figure 10. Here, we measure the performance of the path selection algorithm by the success ratio (more on that is discussed in Section 5).

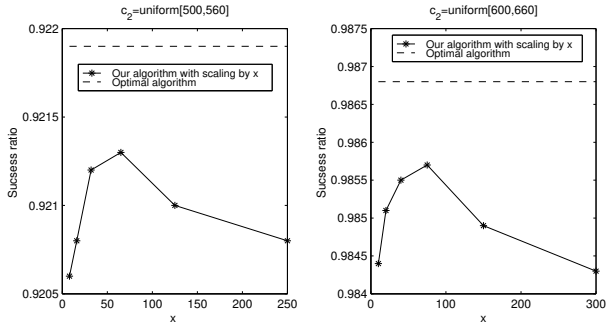

Figure 10: Performance of the path selection algorithm for different values of the scaling factor x.

When the basic algorithm fails to return a feasible path, we scale the graph using different values of x and run the algorithm again. Clearly, one needs to select an appropriate value for x to improve the performance. The following lemma shows that a binary search argument can be used to determine an appropriate x in the range $[1, c_2]$.

Lemma 3 *If the algorithm cannot find a path p for which $w_2'(p) \leq x$ in the scaled graph by x, then such a path cannot be found in a graph that is scaled by $x' < x$.*

Proof of Lemma 3: Let the graph G be scaled by $x = 2r$ for some integer r, and let $\mathcal{P}$ be the set of all possible paths in the scaled graph. If the algorithm fails to return a path p for which $\sum_{e \in p} \left\lceil \frac{w_2(e) \cdot 2r}{c_2} \right\rceil \leq 2r$, then

$$\sum_{e \in p} \left\lceil \frac{w_2(e) \cdot 2r}{c_2} \right\rceil > 2r \quad \forall p \in \mathcal{P}. \tag{9}$$

In order to prove the lemma, it suffices to show that if (9) is true then the algorithm should never search for a path p' for which $w_2'(p') \stackrel{\text{def}}{=} \sum_{e \in p'} \left\lceil \frac{w_2(e) \cdot r}{c_2} \right\rceil \leq r$ when the links of the graph are scaled down by $x = r$. Since we know that

$$2 \left\lceil \frac{w_2(e) \cdot r}{c_2} \right\rceil \geq \left\lceil \frac{w_2(e) \cdot 2r}{c_2} \right\rceil \tag{10}$$

we can rewrite (9) as

$$2 \sum_{e \in p'} \left\lceil \frac{w_2(e) \cdot r}{c_2} \right\rceil > 2r \quad \forall p' \in \mathcal{P} \tag{11}$$

from which we conclude

$$\sum_{e \in p'} \left\lceil \frac{w_2(e) \cdot r}{c_2} \right\rceil > r \quad \forall p' \in \mathcal{P}. \tag{12}$$

This, in turn, implies that no path $p' \in \mathcal{P}$ will be selected by the algorithm, and the claim is true. ■

5 Simulation Results and Discussion

In this section, we contrast the performance of our basic algorithm with Jaffe's second approximation algorithm [23], Chen's heuristic algorithm in [7], and the first ϵ-optimal algorithm in [20]. In [23] Jaffe presents two approximation algorithms for the MCP problem based on the minimization of $w_1(p) + d w_2(p)$, where $d = 1$ in the first algorithm and $d = \sqrt{c_1/c_2}$ in the second. Of the two approximations, the latter one provides better performance, and hence it will be used in our comparisons. As a point of reference, we also report the results of the exact (exponential-time) algorithm, which considers all possible paths in the graph to determine whether there is a feasible path or not. The performance has been measured for various network topologies. For brevity, we report the results for one of these topologies under both homogeneous and heterogeneous links.

5.1 Simulation Model and Performance Measures

In our simulation model, a network is given as a directed graph. Link weights, the source and destination of a connection request, and the constraints c_1 and c_2 are all randomly generated. If the path selection algorithm returns a feasible path for a connection request, we count this request as a *routed connection request*. In order to contrast the performance of various path selection algorithms, we use a measure called the *success ratio* (**SR**), which shows how often an algorithm finds a feasible path [7]:

$$\text{SR} = \frac{\textit{Total number of routed connection requests}}{\textit{Total number of connection requests}}$$

Another important performance aspect is the computational complexity. In here, we measure the complexity of path selection algorithms by the number of performed Dijkstra's iterations. While the algorithm in [23] requires only one iteration, the algorithm in [7] always requires x^2 iterations,

where x is an adjustable positive integer. The number of iterations in our algorithm varies in the range $[1, \log B]$, where B is the upper bound on the longest path according to one of the link weights. For our algorithm, the average number of Dijkstra's iterations (ANDI) per connection request is measured and compared with the deterministic number of Dijkstra's iterations in the other algorithms.

5.2 Results under Homogeneous Link Weights

We consider the network topology in Figure 11, which has been modified from ANSNET [11] by inserting additional links. Link weights are randomly selected with $w_1(u,v) \sim uniform[0,50]$ and $w_2(u,v) \sim uniform[0,200]$. The same network topology was used in [7]. For different ranges of c_1

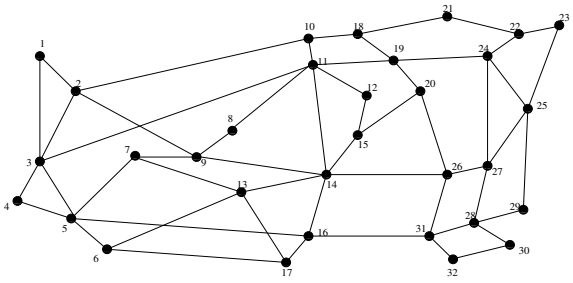

Figure 11: An irregular network topology.

and c_2, Table 1 shows the SR of various algorithm based on twenty runs; each run is based on 2000 randomly generated connection requests. For our algorithm, the ANDI for each range in Table 1 is given by 2.49/2.63/2.23/1.61/1.21, respectively. The number of feasible paths, and thus the SR, increases as the constraints gets looser in the table. As this happens, the ANDI in our algorithm goes down. The overall average complexity per connection request is about two iterations of Dijkstra's algorithm.

In terms of SR, our algorithm performs as good as the exact one. The results show that our algorithm provides significantly superior performance to Jaffe's approximation algorithm. To compare our algorithm with Chen's heuristic algorithm [7] and the ϵ-optimal algorithm [20], we need to properly set the values of x and ϵ, respectively, In theory, as x goes to infinity and as ϵ goes to 0, the performance of the corresponding algorithms approaches that of the exact one. However, since the complexities of these algorithms depend on x and ϵ, large values for x and small values for ϵ clearly make the corresponding algorithms impractical. To get as close as possible to achieving about the same average computational complexity of our algorithm, we set $x = 2$ and $\epsilon = 10$. With $x = 2$, the performance of Chen's algorithm lags significantly behind ours. Even if we increase x to ten, making the computational requirement of Chen's algorithm several times that of our algorithm, its performance still lags behind ours. For $\epsilon = 10$, both our algorithm and the ϵ-optimal algorithm have roughly the same average complexity; with our algorithm providing better performance. More specifically, it improves the success rate of the ϵ-optimal algorithm by about 50%. The ϵ-optimal algorithm uses a dynamic-programming approach that maintains a scaled cost array with size of (n/ϵ) at each node and it can determine paths whose scaled cost is less than (n/ϵ). When the values of constraints are increased, more longer paths becomes feasible, but the ϵ-optimal algorithm cannot determine them unless ϵ gets very small. For example, the performance of the ϵ-optimal algorithm becomes close to that of our algorithm if ϵ is set to 1. However, in that case, the average complexity of our algorithm is about 10% of that of the ϵ-optimal algorithm.

5.3 Performance Under Heterogeneous Links

The uniformity of link weights in a network may severely impact the performance of a path selection algorithm. Hence, before drawing any general conclusions, we need to examine the performance in a network with heterogeneous links. For this purpose, we consider the same network topology in Figure 11. We divide the network into three parts, as shown in Figure 12. Each link is associated with two weights w_1

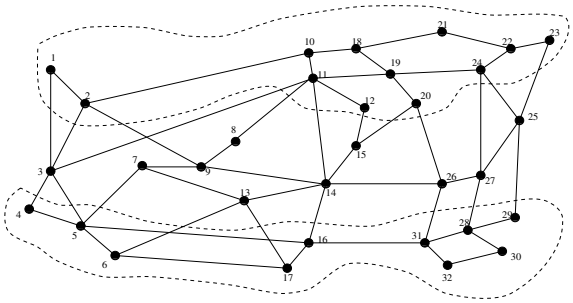

Figure 12: Network topology with heterogeneous link weights.

and w_2, which are determined as follows: if u is a node that belongs to the upper part of the network, then $w_1(u,v) \sim uniform[70,85]$ and $w_2(u,v) \sim uniform[1,5]$; if u is in the middle part, then $w_1(u,v) \sim uniform[45,55]$ and $w_2(u,v) \sim uniform[45,55]$; and if u is in the lower part, then $w_1(u,v) \sim uniform[1,5]$ and $w_2(u,v) \sim uniform[70,85]$. The source node is randomly chosen from nodes 1 to 5. The destination node is randomly chosen from nodes 22 to 30.

For different ranges of c_1 and c_2, Table 2 shows the SR of various algorithms based on twenty runs; each run is based on 2000 randomly generated connection requests. For our algorithm, the ANDI is 4.03/4.59/4.55/4.52/2.75 for each range in Table 2, respectively. Under heterogeneous link weights, the performance of approximate or heuristic path selection algorithms, in general, is not as good as in the case of homogeneous links. One can attribute this performance degradation to the "symmetric" nature of the constraint functions, which favor links with homogeneous characteristics.

Our algorithm still provides superior performance to Jaffe's approximation algorithm. To compare our algorithm with the others, as discussed in the homogeneous case, we need to properly set the values for x and ϵ. To achieve about the same computational complexity of our algorithm, x and ϵ are set to three and ten, respectively. With these values, these algorithms lag behind our algorithm. Their performance gets better as x increases and ϵ decreases. However, in that case, their computational complexities become at least ten times that of our algorithm.

6 Conclusions and Future Work

QoS-based routing subject to multiple additive constraints is an NP-complete problem that cannot be exactly solved in polynomial time. To this problem, we presented an efficient approximation algorithm using a binary search strat-

Range of c_1 and c_2	Exact	Our Alg	Jaffe's	Chen's		ϵ-optimal	
				$(x = 2)$	$(x = 10)$	$(\epsilon = 1)$	$(\epsilon = 10)$
$c_1 \sim uniform[50, 65]$ $c_2 \sim uniform[200, 260]$	0.2594	0.2590	0.2505	0.1935	0.2554	0.2524	0.2099
$c_1 \sim uniform[75, 90]$ $c_2 \sim uniform[300, 360]$	0.5220	0.5190	0.4906	0.3004	0.5003	0.5057	0.3479
$c_1 \sim uniform[100, 115]$ $c_2 \sim uniform[400, 460]$	0.7595	0.7535	0.7088	0.3308	0.7216	0.7430	0.4703
$c_1 \sim uniform[125, 140]$ $c_2 \sim uniform[500, 560]$	0.9219	0.9145	0.8674	0.3308	0.8787	0.9079	0.5639
$c_1 \sim uniform[150, 165]$ $c_2 \sim uniform[600, 660]$	0.9868	0.9819	0.9524	0.3308	0.9609	0.9805	0.6281

Table 1: SR performance of several path selection algorithms.

Range of c_1 and c_2	Exact	Our Alg	Jaffe's	Chen's		ϵ-optimal	
				$(x = 3)$	$(x = 10)$	$(\epsilon = 1)$	$(\epsilon = 10)$
$c_1 \sim uniform[200, 215]$ $c_2 \sim uniform[200, 215]$	0.1278	0.1146	0.0829	0.0740	0.0964	0.1276	0.0742
$c_1 \sim uniform[215, 230]$ $c_2 \sim uniform[215, 230]$	0.1739	0.1568	0.1210	0.0927	0.1494	0.1739	0.1067
$c_1 \sim uniform[230, 250]$ $c_2 \sim uniform[230, 250]$	0.2841	0.1906	0.1678	0.1081	0.2160	0.2841	0.1508
$c_1 \sim uniform[250, 300]$ $c_2 \sim uniform[250, 300]$	0.4572	0.3236	0.2771	0.1131	0.4113	0.4559	0.2891
$c_1 \sim uniform[300, 360]$ $c_2 \sim uniform[300, 360]$	0.8709	0.6704	0.5805	0.1131	0.7590	0.8535	0.6260

Table 2: SR performance of several path selection algorithms.

egy. Our algorithm is supported by performance bounds that reflect the effectiveness of the algorithm in finding a feasible path. We studied the performance of the algorithm via simulations under both homogeneous and heterogeneous link weights. Our results show that the proposed algorithm outperforms existing ones in complexity, performance, or both. We also presented two extensions to our basic algorithm that can be used to further improve its performance at little extra computational cost. The first extension, which is motivated by the presented theoretical bounds, attempts to find the closest feasible path to a constraint. The other extension, namely scaling, improves the likelihood of finding a feasible path by perturbing the linearity of the search process (or equivalently, changing the relative locations of the paths in the parameter space). Our basic approximation algorithm runs hierarchical version of Dijkstra's algorithm up to $\log B$ times, where B is an upper bound on the longest path w.r.t. one of link weights. When scaling is used, the algorithm runs Dijkstra's algorithm up to $\log c_2 \log B$ times. These are worst-case complexities that are rarely used in practice. In fact, simulation results indicate much better average complexities for the basic algorithm and its extensions. The space complexity of our algorithm is $\mathcal{O}(n)$.

The proposed algorithm assumes a flat network topology and complete knowledge of the network state. In practice, the true state of the network is not available to every source node at all times due to network dynamics, aggregation of state information (in hierarchical networks), and latencies in the dissemination of state information. Our future work will focus on investigating the MCP problem in the presence of inaccurate state information and the tradeoffs between the accuracy of the path selection process and that of topology aggregation (for spatial scalability) and/or the frequency of advertisements (for temporal scalability). Another aspect that we plan to investigate is that of renegotiation. When our algorithm fails to return a feasible path, it always returns a path which is close to satisfying the given constraints. Hence, we plan to investigate how such a path can be advantageously used in the renegotiation process to achieve further performance improvements.

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 et al. QoS routing mechanisms and OSPF extensions. Technical Report draft-guerin-qos-routing-ospf-05.txt, Internet Engineering Task Force, April 1998.

[5] 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.

[6] D. Blokh and G. Gutin. An approximation algorithm for combinatorial optimization problems with two parameters. IMADA preprint PP-1995-14, May 1995.

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

[8] 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.

[9] E. I. Chong, S. R. Sanjeev Rao Maddila, and S. T. Morley. On finding single-source single-destination k shortest paths. In *the Seventh International Conference on Computing and Information (ICCI '95)*, pages 40–47, July 5-8, 1995.

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

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

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

[13] 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).

[14] 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.

[15] 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.

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

[17] 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.

[18] 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.

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

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

[21] 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.

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

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

[24] K. Lee et al. QoS based routing for integrated multimedia services. In *Proceedings of the GLOBECOM '97 Conference*, volume II, pages 1047–1051. IEEE, 1997.

[25] 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.

[26] 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.

[27] Q. Ma and P. Steenkiste. Routing traffic with quality-of-service guarantees in integrated services networks. In *Proceedings of NOSSDAV '98*, July 1998.

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

[36] 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.

[37] X. Xiao and L. M. Ni. Internet QoS: a big picture. *IEEE Network*, 13(2):8–18, March-April 1999.

[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.