Two Innovative Algorithms for Fast Computation of Constrained Shortest Paths

[1]R.Pravallika, [2] V.Udaya Bhanu
Gudlavalleru Engineering College, India

Abstract
Computing constrained shortest paths is fundamental to some important network functions such as QoS routing, MPLS path selection, ATM circuit routing, and traffic engineering. The problem is to find the cheapest path that satisfies certain constraints. In particular, finding the cheapest delay-constrained path is critical for real-time data flows such as voice/video calls. Because it is NP-complete, much research has been designing heuristic algorithms that solve the - approximation of the problem with an adjustable accuracy. A common approach is to discretize (i.e., scale and round) the link delay or link cost, which transforms the original problem to a simpler one solvable in polynomial time. The efficiency of the algorithms directly relates to the magnitude of the errors introduced during discretization. In this paper, we propose two techniques that reduce the discretization errors, which allows faster algorithms to be designed. Reducing the overhead of computing constrained shortest paths is practically important for the successful design of a high-throughput QoS router, which is limited at both processing power and memory space. Our simulations show that the new algorithms reduce the execution time by an order of magnitude on power-law topologies with 1000 nodes. The reduction in memory space is similar.

Keywords
Approximation algorithms, constrained shortest paths, QoS routing.

I. Introduction

MAJOR obstacle against implementing distributed multimedia applications (e.g., web broadcasting, video teleconferencing, and remote diagnosis) is the difficulty of en- suring quality of service (QoS) over the Internet. A fundamental problem that underlies many important network functions such as QoS routing, MPLS path selection, and traffic engineering, is to find the constrained shortest path - the cheapest path that satisfies a set of constraints [1-10]. For interactive real-time traffic, the delay-constrained least-cost path has particular importance [11]. It is the cheapest path whose end-to-end delay is bounded by the delay requirement of a time-sensitive data flow. The additional bandwidth requirement, if there is one, can be easily handled by a pre-processing step that prunes the links without the required bandwidth from the graph. The algorithms for computing the constrained shortest paths can be used in many different circumstances, for instance, laying out virtual circuits in ATM networks, establishing wave- length-switching paths in fiber-optics networks, constructing label-switching paths in MPLS based on the QoS requirements in the service contracts, or applying together with RSVP. There are two schemes of implementing the QoS routing algorithms on routers. The first scheme is to implement them as on-line algorithms that process the routing requests as they arrive. In practice, on-line algorithms are not always desired. When the request arrival rate is high (major gateways may receive thousands or tens of thousands of requests every second), even the time complexity of Dijkstra's algorithm will overwhelm the router if it is executed on a per-request basis.1 To solve this problem, the second scheme is to extend a link-state pro- tocol (e.g., OSPF) and periodically pre-compute the cheapest delay-constrained paths for all destinations, for instance, for voice traffic with an end-to-end delay requirement of 100 ms. The computed paths are cached for the duration before the next computation. This approach provides support for both constrained unicast and constrained multicast. The computation load on a router is independent of the request arrival rate. Moreover, many algorithms, including those we will propose shortly, have the same time complexity for computing constrained shortest paths to all destinations or to a single destination. This paper studies the second scheme.

A path that satisfies the delay requirement is called a feasible path. Finding the cheapest (least-cost) feasible path is NP-complete. There has been considerable work in designing heuristic solutions for this problem. Xue [12] and Juttner et al. [13] used the Lagrange relaxation method to approximate the delay-constrained least-cost routing problem. However, there is no theoretical bound on how large the cost of the found path can be. Korkmaz and Krunz used a nonlinear target function to approximate the multi-constrained least-cost path problem [14]. It was proved that the path that minimizes the target function satis- fies one constraint and the other constraints multiplied by, where is a predefined constant and is the number of con- straints. However, no known algorithm can find such a path in polynomial time. Ref. [14] proposed a heuristic algorithm, which has the same time complexity as Dijkstra's algorithm. It does not provide a theoretical bound on the property of the re- turned path, nor provide conditional guarantee in finding a fea- sible path when one exists. In addition, because the construction of the algorithm ties to a particular destination, it is not suitable for computing constrained paths from one source to all destina- tions. For this task, it is slower than the algorithms proposed in this paper by two orders of magnitude based on our simulations. Another thread of research in this area is to design polyno- mial time algorithms that solves the NP-complete problem with an accuracy that is theoretically bounded. Let m and n be the number of links and the number of nodes in the network, respectively. Given a small constant ε, Hassin's algorithm [15] has a time complexity of $O((mn/\varepsilon)\log\log(\mathrm{UB}/\mathrm{LB}))$, where UB and LB are the costs of the fastest path and the cheapest path from the source node to the destination node, respectively. The algorithm finds a feasible path if there exists one. The cost of the path is within the cost of the cheapest feasible path multiplied by $(1+\varepsilon)$. Lorenz and Raz improved the time complexity to $O(mn(1/\varepsilon+\log n))$ [16]. Chen and Nahrstedt solved a similar problem in time $O((m+n\log n)x)$, where $x=O(n/\varepsilon)$ in order to achieve the ε-approximation [17]. Goel *et al.*'s algorithm [18] has the best-known complexity of $O((m+n\log n)(L/\varepsilon))$, where L is the length (hops) of the longest path in the network. However, its approximation model is different. It computes a path whose cost is no more than the cost of the cheapest feasible path, while the delay of the path is within $(1+\varepsilon)$ of the delay requirement. The algorithms proposed in this paper follow Goel's model.

One common technique of the above algorithms [15, 17, 18] is to discretize the link delay (or link cost). Due to the dis- cretization, the possible number of different delay values (or cost values) for a path

is reduced, which makes the problem solvable in polynomial time. The effectiveness of this technique depends on how much error is introduced during the discretization. The existing discretization approaches have either positive discretization error for every link or negative error for every link. Therefore, the discretization error on a path is statistically proportional to the path length as the errors on the links along the path add up. In order to bound the maximum error, the dis- cretization has to be done at a fine level, which leads to high execution time of the algorithms.

Given the limited resources and ever-increasing tasks of the routers, it is practically important to improve the efficiency of the network functions. While QoS routing is expensive due to its nonlinear nature, it has particular significance to reduce the router's overhead in computing the constrained shortest paths. In this paper, we propose two techniques, randomized discretization and path delay discretization, which reduce the discretization errors and allow faster algorithms to be designed. The randomized discretization cancels out the link errors along a path. The larger the topology, the greater the error reduction. The path delay discretization works on the path delays instead of the individual link delays, which eliminates the problem of error accumulation. Based on these techniques, we design fast algorithms to solve the approximation of the constrained shortestpath problem. We prove the correctness and complexities of the algorithms. Although the new algorithms have the same worstcase complexity as Goel et al.'s algorithm [18], we believe (and our simulations suggest) that they run much faster on the average case. The simulations show that the new algorithms are faster than Goel et al.'s algorithm by an order of magnitude on powerlaw topologies with 1000 nodes. The rest of the paper is organized as follows. Section II reviews the existing approaches. Section III describes the randomized discretization, and Section IV describes the path delay discretization. Sections V and VI provide analytical and simulation results, respectively. Section VII draws the conclusion.

II. Problem Definition and Existing Discretization Approaches

Consider a network $G(V, E)$, where V is a set of n nodes and E is a set of m directed links connecting the nodes. The delay and the cost of a link $(u, v) \in E$ are denoted as $d(u, v)$ and $c(u, v)$, respectively. The delay and the cost of a path P are denoted as $d(P)$ and $c(P)$, respectively. $d(P) = \sum_{(u,v)\in P} d(u, v)$, and $c(P) = \sum_{(u,v)\in P} c(u, v)$. Let $h(P)$ be the length (number of hops) of P, and L be the length of the longest path in the network.

Given a delay requirement r, P is called a *feasible path* if $d(P) \leq r$. Given a source node s, let V_s be the set of nodes to which there exist feasible paths from s. For any $t \in V_s$, the *cheapest feasible path* $P_{s,t}$ from s to t is defined as

$$d(P_{s,t}) < r$$
$$c(P_{s,t}) = \min\{c(P) | d(P) < r, \forall \text{ path } P \text{ from } s \text{ to } t\}$$

The *delay-constrained least-cost routing* problem (DCLC) is to find the cheapest feasible paths from s to all nodes in V_s, which is NP-complete [19]. However, if the link delays are all integers and the delay requirement is bounded by an integer λ, the problem can be solved in time $O((m + n \log n)\lambda)$ by Joksch's dynamic programming algorithm [20] or the extended Dijkstra's algorithm [17].

Joksch's algorithm is described as follows. $\forall v \in V, \quad i \in [0..\lambda]$, let $w[v, i]$ be a variable storing the cost of the cheapest path P from s to v with $d(P) \leq i$, and $\pi[v, i]$ storing the last link of the path. Initially, $w[v, i] = \infty, \forall v \neq s$, and $w[s, i] = 0$, $\pi[v, i] = \text{NIL}$. Assuming that all link delays are positive, the dynamic programming is given below.

$$w[v, i] = \min\{w[v, i-1], w[u, i-d(u, v)] + c(u, v) | (u, v) \in E, d(u, v) \leq i\}$$

Now suppose the link delays are allowed to be zero. We need to add one more step. Let G_z be the subgraph consisting of all zero-delay links. For each $i \in [0..\lambda]$, immediately after Joksch's algorithm calculates $w[v, i], \forall v \in V$, Dijkstra's algorithm is executed on G_z to improve $w[v, i]$ on zero-delay paths [18].

The above polynomially solvable special case with integer delays points out a heuristic solution for the general NP-complete problem with arbitrary delays. The idea is to discretize (scale and then round) arbitrary link delays to integers [15], [17], [18], [21]. There are two existing discretization approaches, *round to ceiling* [17] and *round to floor* [18]. Both approaches map the delay requirement r to a selected integer λ, and then discretize the link delays as follows.

Round to ceiling (RTC): For every link (u, v), the delay value is divided by r/λ. If the result is not an integer, it is rounded to the nearest larger integer.

$$d^c(u, v) = \left\lceil \frac{d(u, v)}{r} \lambda \right\rceil \quad (1)$$

Round to floor (RTF): For every link (u, v), the delay value is divided by r/λ. If the result is not an integer, it is rounded to the nearest smaller integer.

$$d^f(u, v) = \left\lfloor \frac{d(u, v)}{r} \lambda \right\rfloor \quad (2)$$

The value of λ controls the rounding error (up to r/λ) introduced by discretization. With a larger λ, the rounding error accounts for a smaller portion of the link delay. When λ is large enough and thus the discretization error is small enough, we can approximate the DCLC problem by a new problem with integer delays after discretization. The solution to the new problem will serve as the solution of the original problem. However the computation overhead is directly related to λ.

After discretizing the link delays by RTC or RTF, either Joksch's algorithm or the extended Dijkstra's algorithm can solve the ε-approximation of DCLC, which is to find a path P for every node $t \in V_s$, such that

$$d(P) \leq (1 + \varepsilon)r$$
$$c(P) \leq c(P_{s,t})$$

where ε is a small percentage. The delay of the path is allowed to exceed the requirement by a percentage of no more than ε, while the cost should be no more than that of the cheapest feasible path $P_{c,d}$. Using RTF, the delay scaling algorithm (DSA) proposed by Goel *et al.* achieves the best time complexity $O((m + n\log n)L/\varepsilon)$ among all existing algorithms [18].

The discretization error of a link (u, v) is defined as

$$\Delta^c(u,v) = d(u,v) - d^c(u,v)\frac{r}{\lambda} \quad (3)$$

$$\Delta^d(u,v) = d(u,v) - d^d(u,v)\frac{r}{\lambda} \quad (4)$$

The discretization error of a path P is defined as

$$\Delta^c(P) = \sum_{(u,v)\text{ on }P} \Delta^c(u,v) \quad (5)$$

$$\Delta^d(P) = \sum_{(u,v)\text{ on }P} \Delta^d(u,v) \quad (6)$$

By (1), we know that $\Delta^c(u,v) \le 0$ is true for all links (u, v). Therefore, $\Delta^c(P) \le 0$ is true for all paths P. Similarly, by (4), $\Delta^d(u,v) \ge 0$ and $\Delta^d(P) \ge 0$ are always true.

III. Randomized Discretization

RTC creates negative rounding errors on links. The error accumulates along a path. The accumulated error is proportional to the path length. The larger the topology, the longer a path, the larger the accumulated error. The same thing is true for RTF, which has positive rounding errors on links. In order to achieve ε-approximation, the accumulated error on a path cannot be too large. To reduce the error on a path, the existing algorithms based on RTC or RTF must reduce the discretization errors on the links by using a large λ value. Given the time complexity $O((m + n\log n)\lambda)$, the computation time is increased in proportion to λ.

The insight is that if we can reduce the error introduced by discretization without using a larger λ, we can improve the performance of the algorithm. We develop two new techniques. The first one is called randomized discretization. It rounds to ceiling or to floor according to certain probabilities. The idea is for some links to have positive errors and some links to have negative errors. Positive errors and negative errors cancel out one another along a path in such a way that the accumulated error is minimized statistically. We will prove that, when the following discretization approach is used, the mean of the accumulated error on a path P is zero and the standard deviation is bounded by $r\sqrt{h(P)}/2\lambda$. In comparison, the mean of the accumulated error is $-(r/2\lambda)h(P)$ for RTC and $(r/2\lambda)h(P)$ for RTF.

Round randomly (RR): For every link (u, v), the delay value is divided by r/λ. If the result is not an integer, it is rounded to the nearest smaller integer or to the nearest larger integer randomly such that the mean error is zero.

$$d^d(u,v) = \begin{cases} \left\lceil \frac{d(u,v)}{r}\lambda \right\rceil & \text{with prob. } p_1 = \frac{d(u,v)}{r}\lambda - \left\lfloor \frac{d(u,v)}{r}\lambda \right\rfloor \\ \left\lfloor \frac{d(u,v)}{r}\lambda \right\rfloor & \text{with prob. } p_2 = 1 - p_1 \end{cases} \quad (7)$$

The discretized delay of a path P is

$$d^d(P) = \sum_{(u,v)\in P} d^d(u,v) \quad (8)$$

The discretization error of a link (u, v) is

$$\Delta^c(u,v) = d(u,v) - d^c(u,v)\frac{r}{\lambda} \quad (9)$$

and the discretization error of a path P is

$$\Delta^c(P) = \sum_{(u,v)\text{ on }P} \Delta^c(u,v) = d(P) - d^c(P)\frac{r}{\lambda} \quad (10)$$

We design the randomized discretization algorithm (RDA), which is based on Dijkstra's algorithm but considers two additive metrics, *delay* and *cost*. It uses RR to discretize the link delays. We will prove that it solves the ε-approximation of DCLC.

The pseudo code of RDA is given below. A two-dimensional array, $w[v, i], \forall v \in V, \ i \in [0..\lambda]$, stores the cost of the cheapest path P from s to v with $d^d(P) = i$. Another two dimensional array, $\pi[v, i]$, stores the last link of the path. An auxiliary two-dimensional array, $\delta[v, i]$, keeps track of the minimum discretization error on paths whose discretized delays are i from node s to node v.

Given any value of λ, RDA_Dijkstra(G, s, λ) computes $w[v, i]$ and $\pi[v, i]$. For any destination v, the function finds the cheapest paths at different path delays, $d^d(P) \in [0..\lambda]$. Let P^v be the cheapest among these paths. RDA(G, s) iteratively calls RDA_Dijkstra with an increasing λ until the delay of P^v is smaller than $(1 + \varepsilon)r$ for all v.

RDA assumes a preprocessing step that removes all nodes to which there are no feasible paths from s. This step can be done by calling Dijkstra's algorithm because only one metric (*delay*) is considered.

```
Initialize(V, s, λ)
1.  for each vertex v ∈ V, each i ∈ [0..λ] do
2.      w[v, i] := ∞, π[v, i] := NIL, δ[v, i] := ∞
3.  w[s, 0] := 0, δ[s, i] := 0

Relax_RDA(u, v, i, λ)
4.  i' := i + d^d(u, v)
5.  error := δ[u, i] + Δ^c(u, v)
6.  if error < 0 then
7.      error := error + r/λ
8.      i' := i − 1
9.  if i' ≤ λ and w[v, i'] > w[u, i] + c(u, v) then
10.     w[v, i'] := w[u, i] − c(u, v)
11.     π[v, i'] := u
12.     δ[v, i'] := min{δ[v, i'], error}

RDA_Dijkstra(G, s, λ)
13. Initialize(V, s, λ)
14. for i = 0 to λ do
15.     Q := V
16.     while Q ≠ ∅ do
17.         u := Extract_Min(Q)
18.         if w[u, i] = ∞ then
19.             break out of the while loop
20.         for every adjacent node v of u do
```

21. Relax_RDA(u, v, i, λ) RDA(G, s)
22. $\lambda := \lambda_0$
23. **do**
24. $\lambda := 2\lambda$
25. RDA_Dijkstra(G, s, λ)
26. **while** $\exists v \in V, d(P^v) > (1+\varepsilon)r$,
where P^v is the path with $\min\{w[v,i] | i \in [0..\lambda]\}$

The correctness of the algorithm is given in the theorem below. The proof can be found in Appendix I.

Theorem 1: RDA solves the ε-approximation of DCLC in time $O((m + n \log n)L/\varepsilon)$.

RDA has the same worst-case time complexity as DSA [18], which uses RTF. The reason is that, *in the worst case*, it could happen that $d^r(u,v) = d^f(u,v)$ for all links (u,v), which makes RR identical to RTF. But such occurrence is *extremely unlikely*. More important than the worst-case complexity is the average-case running time of the algorithm. By its nature, RR is a statistical approach. It does not improve the performance of the algorithm for the rare worst case when round-to-floor happens at all links, but it improves for an average case where round-to-floor and round-to-ceiling happen probabilistically as specified in (7). Because positive errors and negative errors cancel out each other along a path, RDA requires a much smaller λ to complete than DSA, which accumulates positive errors on a path. Consequently, RDA runs much faster than DSA on average, which will be evident from our analytical and simulation results.

IV. Path Delay Discretization

Each unit of discretized delay represents the amount r/λ of real delay. Due to rounding, each time discretization is performed, a discretization error up to r/λ is introduced between the discretized delay and the real delay. The maximum discretization error of a path is determined by the number of times that discretization is performed on the path. RTF, RTC, and RR perform discretization at the link level. Because discretization is carried out on each link, the maximum error on the path is linear to the path length. In order to achieve ε-approximation, the accumulated error on a path cannot be too large. There are two ways to reduce the error. One is to use a larger λ, which increases the execution time of an algorithm whose complexity is linear to λ. The other way is to reduce the number of discretizations performed on the path.

Our second technique to control error is to perform discretization on the path level, using the interval partitioning method for combinatorial approximation [22]. For a path P, ideally, discretization is performed once as follows.

$$d'(P) = \left\lfloor \frac{d(P)}{r}\lambda \right\rfloor \quad (11)$$

Because only one discretization is performed, the maximum discretization error on any path is bounded by r/λ, independent of the path length.

Below we design the path discretization algorithm (PDA) based on the above intuition. The algorithm solves the ε-approximation with the same worst-case complexity as RDA. However, its average execution time is better than RDA according to our simulations. An auxiliary two-dimensional array, $x[v,i]$, keeps track of the minimum delay of paths whose discretized delays are i from node s to node v.

PDA_Dijkstra is omitted because it is identical to RDA_Dijkstra except that it calls Relax_PDA.

Initialize(V, s, λ)
1. **for** each vertex $v \in V$, each $i \in [0..\lambda]$ **do**
2. $w[v,i] := \infty$, $\pi[v,i] :=$ NIL, $x[v,i] := \infty$
3. $w[s,0] := 0$, $x[s,0] := 0$

Relax_PDA(u, v, i, λ)
4. $i' := \lfloor (x[u,i] + d(u,v)/r)\lambda \rfloor$
5. **if** $i' \le \lambda$ and $w[v,i'] > w[u,i] + c(u,v)$ **then**
6. $w[v,i'] := w[u,i] + c(u,v)$
7. $\pi[v,i'] := u$
8. $x[v,i'] := \min\{x[v,i'], x[u,i] + d(u,v)\}$

PDA(G, s)
9. $\lambda := \lambda_0$
10. **do**
11. $\lambda := 2\lambda$
12. PDA_Dijkstra(G, s, λ)
13. **while** $\exists v \in V, d(P^v) > (1+\varepsilon)r$,
where P^v is the path with $\min\{w[v,i] | i \in [0..\lambda]\}$

The correctness of the algorithm is given in the theorem below. The proof can be found in Appendix II.

Theorem 2: PDA solves the ε-approximation of DCLC in time $O((m + n \log n)L/\varepsilon)$.

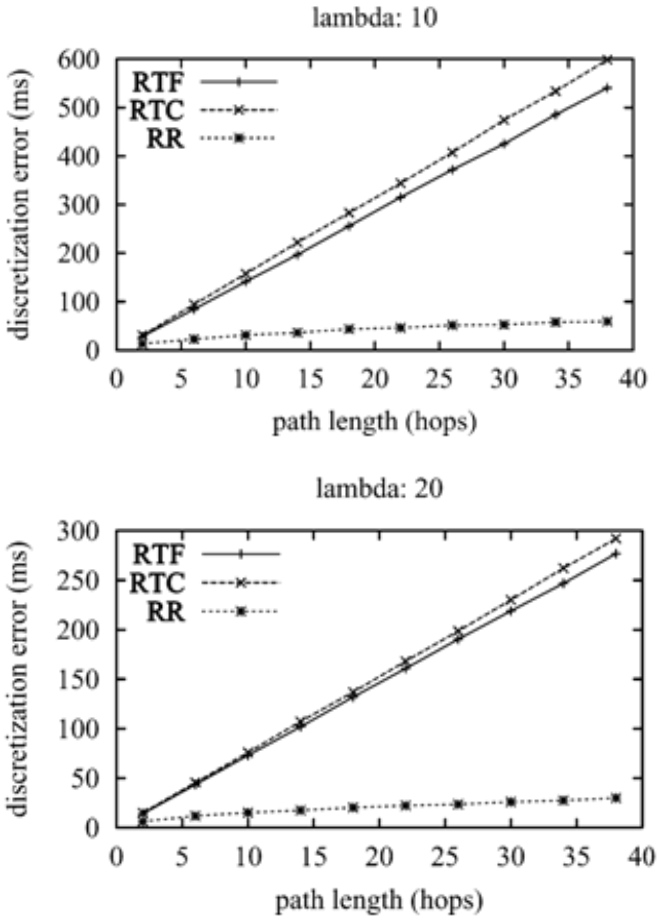


Fig. 1 : Comparison of the average discretization errors of RTF, RTC, and RR with respect to different path lengths. The vertical axis is the average of j (P)j, j (P)j, or j (P)j over 10 000 sample paths.

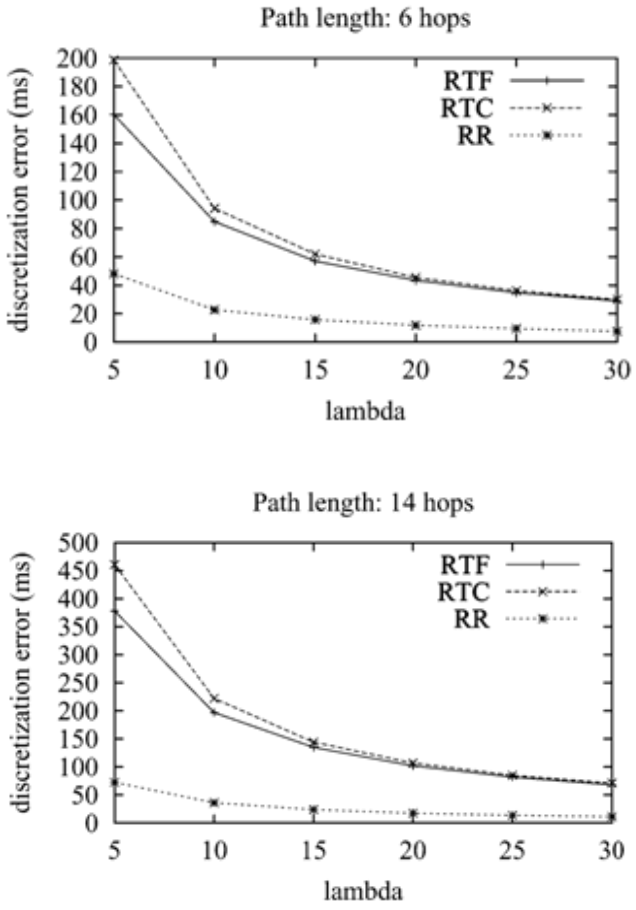


Fig. 2 : Comparison of the average discretization errors of RTF, RTC, and RR with respect to different values. The vertical axis is the average of j (P)j,
j (P)j, or j (P)j over 10 000 sample paths.

V. Analysis

When RTF is used, all links have non-negative discretization errors with a tight upper bound of ϵ/λ. Hence, the discretization errors on links of a path P will add up to a non-negative value with a tight upper bound of $(\epsilon/\lambda)|(P)|$, which is linear to the path length. Statistically, the longer the path, the larger the error. For instance, if $\Delta^r(u,v)$, $\forall(u,v) \in P$, is uniformly distributed in $[0, (\epsilon/\lambda)]$, the mean of $\Delta^r(P)$ is $(\epsilon/2\lambda)|(P)|$.

When RTC is used, all links have non-positive discretization errors with a tight lower bound of $-(\epsilon/\lambda)$. If $\Delta^c(u,v)$, $\forall(u,v) \subset P$, is uniformly distributed in $[-(\epsilon/\lambda), 0]$, the mean of $\Delta^c(P)$ is $-(\epsilon/2\lambda)|(P)|$.

The error of the proposed path-delay discretization is always non-negative with a tight upper bound of ϵ/λ, independent of the path length.

To study RR, we model $d(u,v), \forall(u,v) \subset E$, as a random variable. For any path P, $\Delta^r(P)$ is also a random variable. Assuming the delays of different links are independent, we prove the following theorem in Appendix III.

Theorem 3: Given a path P, the mean of $\Delta^r(P)$ is zero and the standard deviation of $\Delta^r(P)$ is at most $\epsilon\sqrt{|(P)|}/2\lambda$, regardless of the probability distributions of the link delays.

We also perform simulations to compare the discretization errors of different approaches. Fig. 1 shows how the discretization errors of RTF, RTC and RR grow with the path length. The link delay is randomly generated, following an exponential distribution with a mean at 100 ms. The discretization errors of RTF and RTC grow linearly with the path length,[2] while the error of RR grows sublinearly. Fig. 2 shows that, in order to achieve certain discretization error goal, RR requires much smaller λ than RTF and RTC, which means that algorithms based on RR are likely to have less execution time.

VI. Simulation

A. Simulation Setup

The simulation uses two types of network topologies that are generated based on the Power-Law model [23] and the Waxman model [24]. In a Power-Law topology, the degrees of 10% nodes are one, and the degrees of other nodes follow a power law distribution, i.e., the frequency f_d of a degree is proportional to the degree $d(\geq 2)$ raised to the power of a constant $O = \quad 2.2$.

$$f_d \propto d^O$$

After each node is assigned a degree according to the power law distribution, a spanning tree is formed among the nodes to ensure a connected graph. Additional links are inserted to fulfill the remaining degrees of every node with the neighbors selected according to probabilities proportional to their respective unfulfilled degrees. A Waxman topology is formed as follows: the nodes are randomly placed in a one-by-one square, and the probability of creating a link between node u and node v is

$$p(u,v) = \beta e^{-d(u,v)/\alpha L},$$

where $d(u,v)$ is the distance between u and v, $\alpha = 0.6$, and L is the maximum distance between any two nodes. The average node degree is 3.

The default simulation parameters are: The link delays (costs) are randomly generated, following the exponential distribution with a mean of 100. $\epsilon = 0.1$. $\lambda_0 = 3$. Each data point is the average over 1000 randomly generated routing requests. More specifically, we randomly generate ten topologies. On each topology, 100 routing requests are generated with the source node randomly selected from the topology. We run DSA, RDA, and PDA to find a cheapest feasible path to every destination for which a feasible path exists. All simulations were done on a PC with PIV 2 GHz CPU and 512 Megabytes memory.

The performance metrics used to evaluate the routing algorithms are defined as follows.

$$\text{avg execution time} = \frac{\text{total execution time for all requests}}{\text{total number of routing requests}}$$

$$\text{avg cost} = \frac{\text{total cost of returned paths}}{\text{number of returned paths}}$$

$$\text{success rate} = \frac{\text{number of returned paths that are feasible}}{\text{number of returned paths}}$$

All algorithms under simulation guarantee that the delay of any returned path is bounded by $(1 + \epsilon)r$.

B. Comparing RDA and PDA With DSA

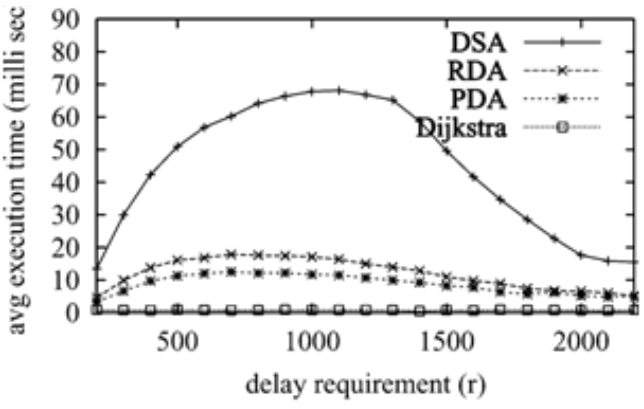
We compare RDA and PDA with DSA [18], which is the best known ε-approximation algorithm for DCLC. Fig. 3 shows the simulation results on Power-Law topologies with 500 nodes. Both RDA and PDA are much faster than DSA, with PDA achieving the best execution time. The average costs of the three algorithms are comparable. The success ratio of RDA is slightly better than the other two. Because the three algorithms are close in terms of average cost and success rate in all simulations, we shall focus on execution time in the sequel.

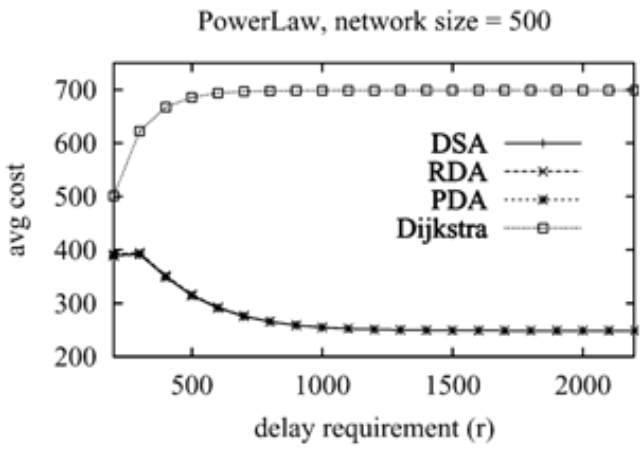
Fig. 4 compares DSA, RDA, and PDA on Waxman topologies with 1000 nodes. Both RDA and PDA again outperform DSA significantly.

Fig. 5 compares the scalability of the three algorithms with respect to the network size. The performance gap between RDA/PDA and DSA increases for larger topologies. The improvement exceeds an order of magnitude for 1000-node networks.

Fig. 6 compares the algorithms with different ε values. The performance gap between RDA/PDA and DSA increases when ε is smaller, i.e., the ε-approximation is performed at the finer level.

In summary, the simulation results confirmed our basic idea that the execution time could be greatly improved by reducing the discretization error, which was achieved very effectively by RDA and PDA. With 1000 nodes and one constraint, RDA and PDA computes the constrained shortest paths within 38 milliseconds and 25 milliseconds, respectively, which makes them practical solutions for routers to compute the QoS routing paths periodically.





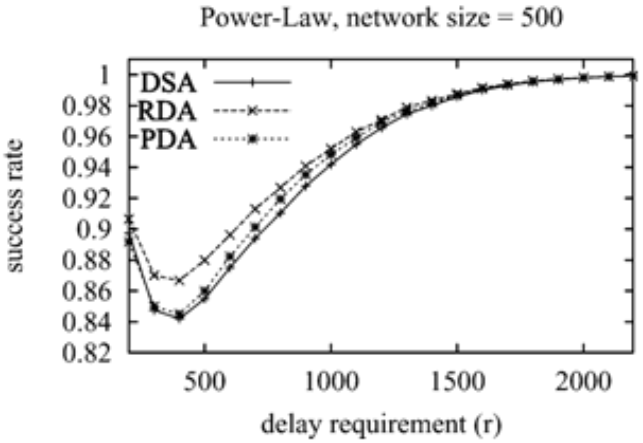


Fig. 3 : Compare DSA, RDA, and PDA on Power-Law topologies. Both RDA and PDA run much faster than DSA. They run slower than Dijkstra's algorithm, but achieve much smaller average path cost. The success rates of DSA, RDA, and PDA are comparable, with RDA slightly better.

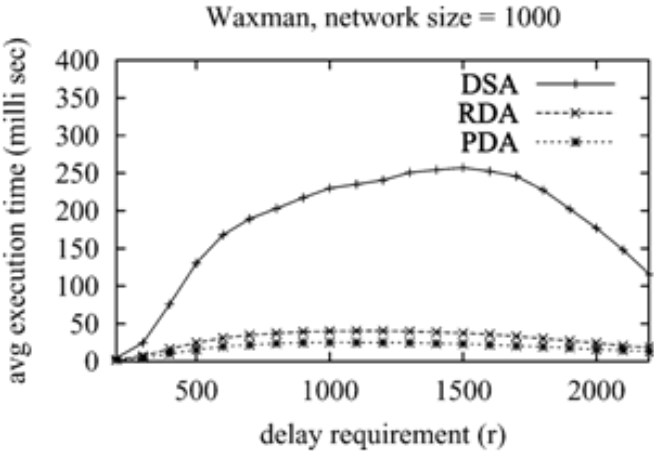
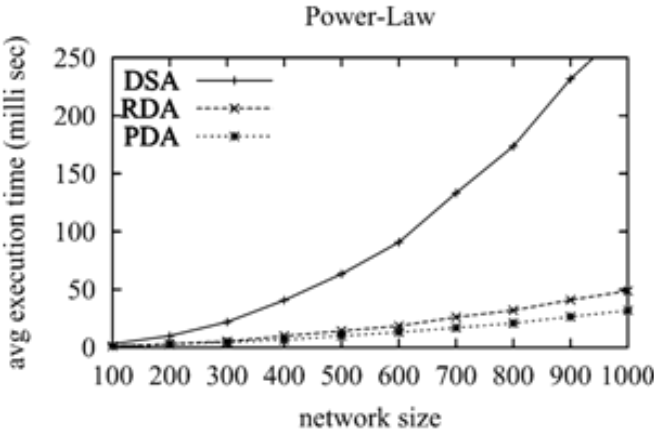


Fig. 4 : Compare DSA, RDA, and PDA on Waxman topologies. Both RDA and PDA run much faster than DSA. PDA is slightly better than RDA.



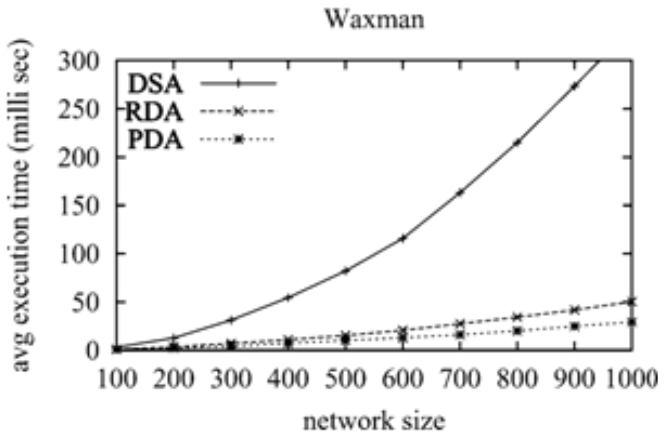


Fig. 5. Scalability comparison. The delay requirement is 1500. Both RDA and PDA scale much better than DSA, with PDA the best.

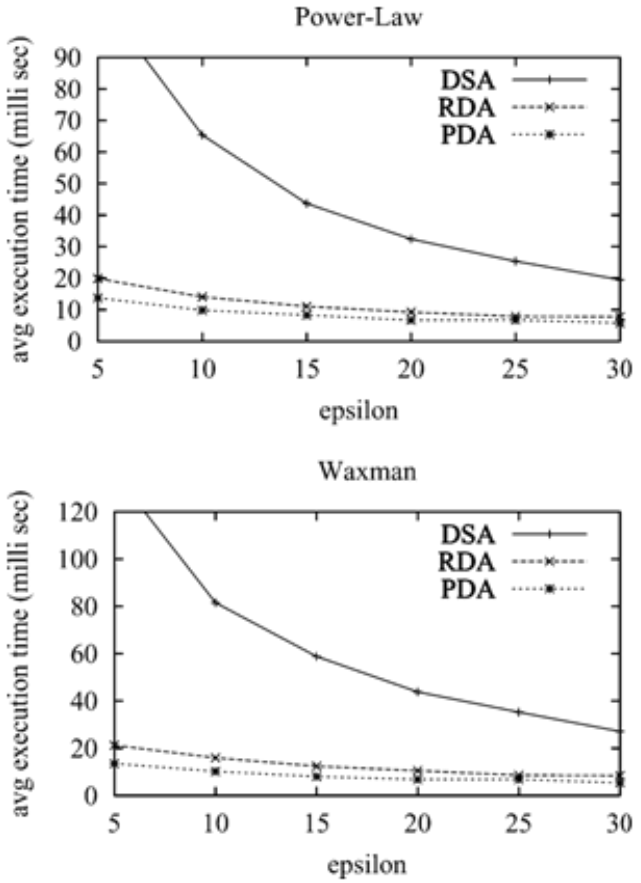


Fig. 6 : Compare DSA, RDA, and PDA with respect to different " values. The delay requirement is 1500, and the network size is 500. Both RDA and PDA run much faster than DSA. PDA is slightly better than RDA.

C. Comparing RDA and PDA With H_MCOP

We compare RDA and PDA with a fast heuristic algorithm H_MCOP [14], whose time complexity is the same as that of Dijkstra's algorithm. H_MCOP does not solve the - approximation of DCLC. Its goal is to use heuristics to greatly reduce the computation time. To construct a feasible path with low cost from a particular source node to a particular destination node, H_MCOP requires building a shortest-path tree from all nodes

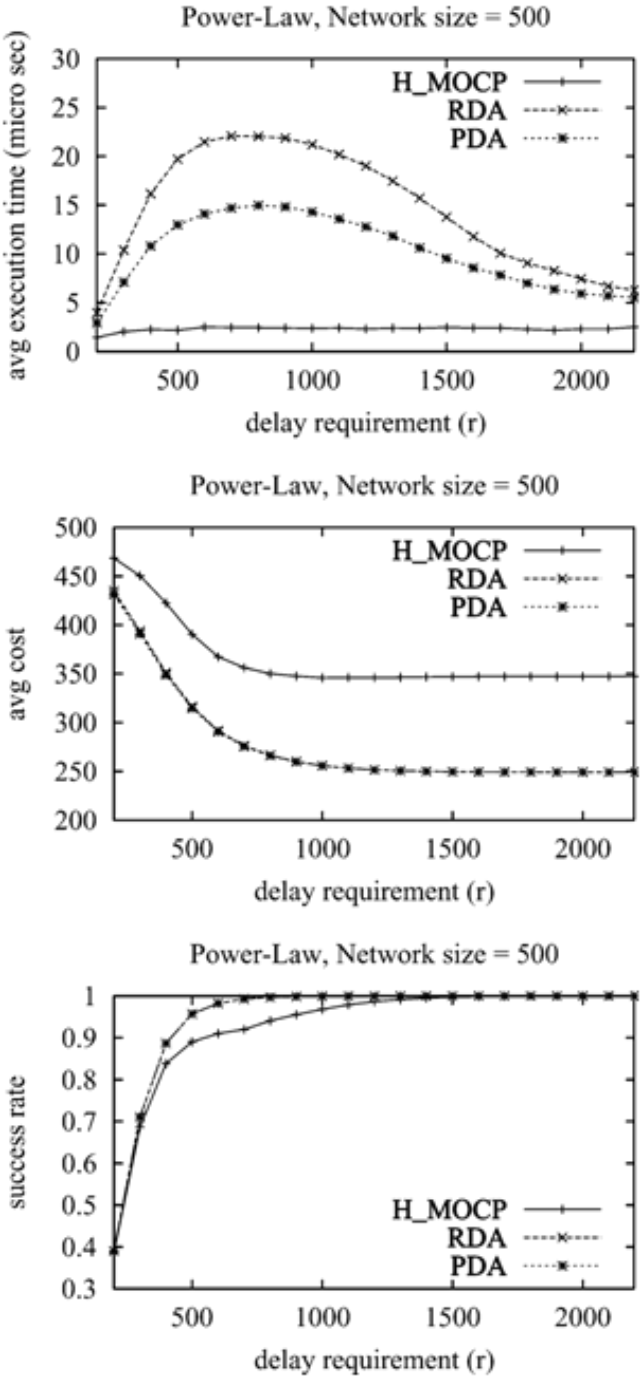


Fig. 7 : Compare RDA and PDA with H_MCOP.

to the destination node and a tree from the source node to all nodes. By the algorithm's two-tree design, it is efficient in computing a low-cost feasible path from one source to one destination, but it is not suitable to find low-cost paths from one source to all destinations. To solve this problem, H_MCOP would have to repeat n times, one for each destination and with a total time complexity of $O(nm + n^2 \log n)$. In comparison, RDA and PDA solve the .-approximation of DCLC, and they find constrained shortest paths for all destinations with the same complexity $O(|n-n\log n|(L/\epsilon))$ as finding a constrained shortest path for a single destination.

The comparison of RDA/PDA and H_MCOP is made under two scenarios. The first scenario is to use them as on-line algorithms that process delay-constrained least-cost unicast routing requests as they arrive. The results are shown in fig. 7. H_MCOP has

also a parameter called lambda for a different purpose, which is however insignificant when there is only one constraint (for delay). H_MCOP significantly outperforms RDA/PDA in average execution time, RDA/PDA are better in terms of average cost and success rate because they relax the delay requirement by a factor of . H_MCOP is a more efficient on-line algorithm than RDA/PDA.

Table 1 : Execution Time (Milliseconds) of Finding Delay-Constrained Least-Cost Paths from A Source to All Destinations On Power-Law Topologies

no. of nodes	RDA	PDA	H_MCOP
100	1.3	0.8	35.2
200	3.1	2.2	159.4
300	5.8	3.8	369.5
400	8.3	6.9	673.4
500	13.3	9.4	1092.2
600	19.6	12.6	1615.6
700	25.3	17.0	2285.9
800	32.1	20.8	3024.2
900	40.4	26.5	3946.1
1000	48.2	32.0	4964.8

In practice, on-line algorithms are not always desired. When the request arrival rate is high (major gateways may receive thousands or tens of thousands of requests every second), even the time complexity of Dijkstra's algorithm (executed on a per- request basis) will overwhelm the router. One typical approach to solve this problem is to extend a link-state protocol (e.g., OSPF) and periodically pre-compute delay-constrained least- cost paths for all destinations. In this way, the computation load on a router is independent of the request arrival rate. Under such scenario, RDA/ PDA significantly outperforms H_MCOP by or- ders of magnitude when the number of nodes is large, as shown in Table I.
Therefore, H_MCOP is more suitable as an online algorithm, while RDA/PDA are more suitable to calculate DCLC paths from one source to all destinations so that a routing table for certain QoS service class can be established. In addition, RDA/PDA are the choice when a constrained multicast tree is calculated centrally.

VII. Conclusion

In this paper, we proposed two techniques, randomized discretization and path delay discretization, to design fast algo- rithms for computing constrained shortest paths. While the pre- vious approaches (RTF and RTC) build up the discretization error along a path, the new techniques either make the link errors to cancel out each other along the path or treat the path delay as a whole for discretization, which results in much smaller errors. The algorithms based on these techniques run much faster than the best existing algorithm that solves the approximation of DCLC.

Appendix Proof of Theorem 1

Refer to Section III for the lines of the pseudo code of RDA (randomized discretization algorithm).

Lemma 1: It always holds that $\delta[u,i] \geq 0, \forall u \in V, i \in [0..\lambda]$.

Proof: It holds initially. The value of δ changes only at Line 12. Suppose $\delta[u,i] > 0$ and $\delta[v,i'] \geq 0$ before Relax_RDA$(\ldots)$ is called. Because $-(r/\lambda) < \Delta^r(u,v) < r/\lambda$, Lines 6–7 make sure that $error \geq 0$. Hence, $\delta[u,i'] \geq 0$ after Line 12. The lemma remains true after the call. □

Lemma 2: Let P_v^u be the path stored by $\pi[u,i]$. It always holds that $d(P_v^u) \geq i(r/\lambda) + \delta[u,i], \forall u \in V, i \in [0..\lambda]$.

Proof: Suppose it holds before Relax_RDA$(\ldots)$ is called. $P_v^u \geq i(r/\lambda) - \delta[u,i]$. The new path under consideration is $P_v^u + (u,v)$.

$$\begin{aligned} d(P_v^u - (u,v)) &= d(P_v^u) + d(u,v) \\ &\geq i\frac{r}{\lambda} + \delta[u,i] + d^r(u,v)\frac{r}{\lambda} \quad \Delta^r(u,v) \\ &= (i + d^r(u,v))\frac{r}{\lambda} + \delta_{u,i} \quad \Delta^r(u,v) \end{aligned}$$

After Lines 4–8, $d(P_v^u + (u,v)) \geq i'(r/\lambda) + error$. After Line 12, $\delta[v,i'] \leq error$. Hence, $d(P_v^u - (u,v)) \quad d(P_v^u) \geq i'(r/\lambda) \quad \delta[v,i']$. The lemma holds after the call. □

Lemma 3: Let P_v^u be the path stored by $\pi[u,i]$. It always holds that $d(P_v^u) \leq (i + l(P_v^u))(r/\lambda), \forall u \in V, i \in [0..\lambda]$, where $l(P_v^u)$ is the length (hops) of P_v^u.

Proof: Suppose it holds before Relax_RDA$(\ldots)$ is called. $P_v^u < (i + l(P_v^u))(r/\lambda)$. The new path under consideration is $P_v^u + (u,v)$.

$$\begin{aligned} d(P_v^u - (u,v)) &= d(P_v^u) + d(u,v) \\ &\leq (i + l(P_v^u))\frac{r}{\lambda} - d^r(u,v)\frac{r}{\lambda} + \Delta^r(u,v) \\ &= (i + d^r(u,v))\frac{r}{\lambda} + \Delta^r(u,v) + l(P_v^u)\frac{r}{\lambda} \\ &\leq i'\frac{r}{\lambda} \quad (l(P_v^u) - 1)\frac{r}{\lambda} \end{aligned}$$

After Line 12, $d(P_v^u + (u,v)) = d(P_v^u) \leq (i' + l(P_v^u))(r/\lambda)$. The lemma holds after the call. □

Theorem 1: RDA solves the ε-approximation of DCLC in time $O((m + n\log n)L/\varepsilon)$.

Proof: We first prove that if RDA terminates, it solves the ε-approximation of DCLC. Consider an arbitrary node t. Let $P_{s,t}$ be the cheapest feasible path. Assume this is the only path from s to t. Consider Relax_RDA$(\ldots)$ is called on a link (u,v) of $P_{s,t}$. After Lines 4–5, $i' \quad i - d^r(u,v) \quad i + (d(u,v) \quad \Delta^r(u,v))(\lambda/r) = i \quad (d(u,v) \quad error + \delta[u,i])(\lambda/r)$. After Lines 6–8, because $error \geq 0$, $i' \leq i + \delta[u,i](\lambda/r) + d(u,v)(\lambda/r)$. By Lemma 2, $i \quad \delta[u,i](\lambda/r) \leq d(P_v^u)(\lambda/r)$. We have $i' < (d(P_v^u) - d(u,v))(\lambda/r) < d(P_{s,t})(\lambda/r) < \lambda$. Lines 9–12 will be executed. Eventually, $P_{s,t}$ will be stored by $\pi[t,i]$ for some $i \leq \lambda$.

Now if there exist other paths from s to t and one of them replaces $P_{s,t}$ during the relaxation, the path must have a smaller cost than $P_{s,t}$. Hence, when RDA terminates, let p^t be the path returned by RDA for t with $\min\{w[t,i] | i \in [0..\lambda]\}$. We must have $c(p^t) \leq c(P_{s,t})$, and $d(P^t) \leq (1 \quad \varepsilon)r$ because otherwise RDA will not terminate.

We now prove that RDA terminates in time $O((m + n\log n)L/\varepsilon)$. When $\lambda > L/\varepsilon$, by Lemma 3, $\forall t \in V$, $d(P^t) \leq (\lambda + l(P^t))(r/\lambda) \leq r + l(P^t)(\varepsilon r/L) \leq (1-\varepsilon)r$. By Line 26, RDA terminates.

The time complexity of each execution of RDA_Dijkstra$(\ldots)$ is $O((m - n\log n)\lambda)$. Since λ doubles each time, the time of the last execution is larger than the combined time of all previous executions. Therefore, the complexity of RDA is $O((m - n\log n)L/\varepsilon)$. □

Appendix Proof of Theorem 2

Refer to Section IV for the lines of the pseudo code of RDA (randomized discretization algorithm).

Lemma 4: Let P_i^u be the path stored by $\pi[u,i]$. It always holds that $z[u,i] \le d(P_i^u)$, $\forall u \in V, i \in [0..\lambda]$.

The proof is trivial based on Line 8.

Lemma 5: Let P_i^u be the path stored by $\pi[u,i]$. It always holds that $z[u,i] \ge i(r/\lambda)$, $\forall u \in V, i \in [0..\lambda]$.

Proof: Suppose it holds before Relax_PDA(...) is called, namely, $z[u,i] \ge i(r/\lambda)$ and $z[v,i'] \ge i'(r/\lambda)$. The new path under consideration is $P_i^u + (u,v)$.

$$\begin{aligned} i' &\le \left\lfloor \frac{z[u,i] + d(u,v)}{r}\lambda \right\rfloor \\ i' &< \frac{z[u,i] - d(u,v)}{r}\lambda \\ z[u,i] + d(u,v) &\ge i'\frac{r}{\lambda} \end{aligned}$$

After Line 8 is executed, $z[v,i'] \ge i'(r/\lambda)$ remains true. □

Lemma 6: Let P_i^u be the path stored by $\pi[u,i]$. It always holds that $d(P_i^u) < (i + l(P_i^u))(r/\lambda), \forall u \in V, i \in [0..\lambda]$, where $l(P_i^u)$ is the length (hops) of P_i^u.

Proof: Suppose it holds before Relax_PDA(...) is called. $d(P_i^u) \le (i + l(P_i^u))(r/\lambda)$. The new path under consideration is $P_{i'}^v + (u,v)$.

$$\begin{aligned} d(P_{i'}^v + (u,v)) &= d(P_{i'}^v) \\ &\le (i + l(P_i^u))\frac{r}{\lambda} - d(u,v) \\ &< z[u,i] + d(u,v) + l(P_{i'}^v)\frac{r}{\lambda} \\ &\le i'\frac{r}{\lambda} + (l(P_i^u) + 1)\frac{r}{\lambda} \end{aligned}$$

After Lines 5–8, $d(P_{i'}^v + (u,v)) = d(P_{i'}^v) < (i' + l(P_{i'}^v))(r/\lambda)$. The lemma holds after the call. □

Theorem 2: PDA solves the ε-approximation of DCLC in time $O((\lambda m + n\log n)L/\varepsilon)$.

The proof is similar to that for Theorem 1 in Appendix I.

Appendix Proof of Theorem 3

Theorem 3: Given a path P, the mean of $\Delta^r(P)$ is zero and the standard deviation of $\Delta^r(P)$ is at most $r\sqrt{l(P)/2\lambda}$, regardless of the probability distributions of the link delays.

Proof: Consider an arbitrary link (u,v) on P.

$$\Delta^r(u,v) = d(u,v) - d^r(u,v)\frac{r}{\lambda} = \begin{cases} d(u,v) - \left\lfloor \frac{d(u,v)}{r}\lambda \right\rfloor \frac{r}{\lambda} & \text{with prob. } p_1 = \frac{d(u,v)}{r}\lambda - \left\lfloor \frac{d(u,v)}{r}\lambda \right\rfloor \\ d(u,v) - \left\lfloor \frac{d(u,v)}{r}\lambda \right\rfloor \frac{r}{\lambda} & \text{with prob. } p_2 = 1 - p_1 \end{cases}$$

The mean (or expected value) of $\Delta^r(u,v)$ is

$$\begin{aligned} E(\Delta^r(u,v)) &= \left(d(u,v) - \left\lfloor \frac{d(u,v)}{r}\lambda \right\rfloor \frac{r}{\lambda}\right) \\ &\quad + \left(d(u,v) - \left\lfloor \frac{d(u,v)}{r}\lambda \right\rfloor \frac{r}{\lambda}\right) \end{aligned}$$

There are two cases.

- Case 1: If $(d(u,v)/r)\lambda$ is an integer, i.e., $(d(u,v)/r)\lambda = \lceil (d(u,v)/r)\lambda \rceil = \lfloor (d(u,v)/r)\lambda \rfloor$, then it is clear that $E(\Delta^r(u,v)) = 0$.
- Case 2: If $(d(u,v)/r)\lambda$ is not an integer, then

$$\begin{aligned} p_2 &= 1 - p_1 = \frac{d(u,v)}{r}\lambda - \left\lfloor \frac{d(u,v)}{r}\lambda \right\rfloor \\ E(\Delta^r(u,v)) &= \left(d(u,v) - \left\lfloor \frac{d(u,v)}{r}\lambda \right\rfloor \frac{r}{\lambda}\right) \cdot \left(\frac{d(u,v)}{r}\lambda - \left\lfloor \frac{d(u,v)}{r}\lambda \right\rfloor\right) \\ &\quad + \left(d(u,v) - \left\lfloor \frac{d(u,v)}{r}\lambda \right\rfloor \frac{r}{\lambda}\right) \cdot \left(\left\lfloor \frac{d(u,v)}{r}\lambda \right\rfloor + 1 - \frac{d(u,v)}{r}\lambda\right) \\ &= 0 \end{aligned} \tag{12}$$

Denote $E(\Delta^r(u,v))$ as μ for clarity. Since the probability density function of $d(u,v)$ is $f_{u,v}(x)$, $x \in [0,+\infty)$, the variance of $\Delta^r(u,v)$ is

$$\begin{aligned} V(\Delta^r(u,v)) &= \int_0^\infty \left[\left(x - \left\lfloor \frac{x}{r}\lambda \right\rfloor \frac{r}{\lambda} - \mu\right)^2 \cdot p_1 + \left(x - \left\lfloor \frac{x}{r}\lambda \right\rfloor \frac{r}{\lambda} - \mu\right)^2 \cdot p_2\right] \cdot f_{u,v}(x)dx \\ &= \frac{r^2}{\lambda^2}\int_0^\infty \left(\frac{x}{r}\lambda - \left\lfloor \frac{x}{r}\lambda \right\rfloor\right)^2 \cdot p_1 + \left(\frac{x}{r}\lambda - \left\lfloor \frac{x}{r}\lambda \right\rfloor\right)^2 \cdot p_2\right] \cdot f_{u,v}(x)dx \end{aligned}$$

When $d(u,v) = x$, $p_1 = (x/r)\lambda - \lfloor (x/r)\lambda \rfloor$ by (7), and $p_2 = \lceil (x/r)\lambda \rceil - (x/r)\lambda$ by (12). Hence,

$$\begin{aligned} V(\Delta^r(u,v)) &= \frac{r^2}{\lambda^2}\int_0^\infty \left[\left(\frac{x}{r}\lambda - \left\lfloor \frac{x}{r}\lambda \right\rfloor\right)^2 \cdot \left(\frac{x}{r}\lambda - \left\lfloor \frac{x}{r}\lambda \right\rfloor\right) \right. \\ &\quad \left. + \left(\frac{x}{r}\lambda - \left\lfloor \frac{x}{r}\lambda \right\rfloor\right)^2 \cdot \left(\left\lceil \frac{x}{r}\lambda \right\rceil - \frac{x}{r}\lambda\right)\right] \cdot f_{u,v}(x)dx \\ &= \frac{r^2}{\lambda^2}\int_0^\infty \left(\left\lceil \frac{x}{r}\lambda \right\rceil - \frac{x}{r}\lambda\right) \cdot \left(\frac{x}{r}\lambda - \left\lfloor \frac{x}{r}\lambda \right\rfloor\right) \cdot f_{u,v}(x)dx \\ &= \frac{r^2}{\lambda^2}\int_0^\infty \left(\left\lceil \frac{x}{r}\lambda \right\rceil - \frac{x}{r}\lambda\right) \cdot \left(1 - \left(\left\lceil \frac{x}{r}\lambda \right\rceil - \frac{x}{r}\lambda\right)\right) \cdot f_{u,v}(x)dx \end{aligned}$$

References

[1] S. Chen, K. Nahrstedt, "An overview of quality of service routing for the next generation high speed networks: Problems and solutions", IEEE Network, Special Issue on Transmission and Distribution of Dig- ital Video, vol. 12, no. 6, pp. 64–79, Nov./Dec. 1998.

[2] R. Guerin, A. Orda, "QoS based routing in networks with inaccurate information: theory and algorithms", in Proc.

IEEE INFOCOM, 1997, pp. 75–83.
[3] T. Korkmaz, M. Krunz, "Source-oriented topology aggregation with multiple QoS parameters in hierarchical ATM networks", in Proc. IEEE IWQoS'99, Jun. 1999, pp. 137–146.
[4] D. Raz, Y. Shavitt, "Optimal partition of QoS requirements with discrete cost functions," in Proc. IEEE INFOCOM, 2000, pp. 613–622. [5] J. L. Sobrinho, "Algebra and algorithms for QoS path computation and hop-by-hop routing in the Internet", in Proc. IEEE INFOCOM, 2001, pp. 727–735.
[6] Z. Wang, J. Crowcroft, "QoS routing for supporting resource reser- vation", IEEE J. Sel. Areas Commun., vol. 14, no. 7, pp. 1228–1234, Sep. 1996.
[7] X. Yuan, X. Liu, "Heuristic algorithms for multi-constrained quality of service routing", in Proc. IEEE INFOCOM, 2001, pp. 844–853.
[8] A. Orda, A. Sprintson, "Efficient algorithms for computing disjoint QoS paths," in Proc. IEEE INFOCOM, 2004, pp. 727–738.
[9] Y. Xiao, K. Thulasiraman, G. Xue, "Approximation and heuristic algorithms for delay constrained path selection under inaccurate state information", in Proc. 1st Int. Conf. Quality of Service in Heteroge- neous Wired/Wireless Networks (QShine), Oct. 2004, pp. 130–137.
[10] Y. Xiao, K. Thulasiraman, G. Xue, "Constrained shortest link-dis- joint paths selection: a network programming based approach", IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 53, no. 5, pp. 1174–1187, May 2006.
[11] H. F. Salama, D. S. Reeves, Y. Viniotis, "A distributed algorithm for delay-constrained unicast routing", in Proc. IEEE INFOCOM, 1997, pp. 84–91.
[12] G. Xue, "Primal-dual algorithms for computing weight-constrained shortest paths and weight-constrained minimum spanning trees", in Proc. IEEE IPCCC'00, Feb. 2000, pp. 271–277.
[13] A. Juttner, B. Szviatovszki, I. Mecs, Z. Rajko, "Lagrange relaxation based method for the QoS routing problem", in Proc. IEEE INFOCOM, 2001, pp. 859–868.
[14] T. Korkmaz, M. Krunz, "Multi-constrained optimal path selection", in Proc. IEEE INFOCOM, 2001, pp. 834–843.
[15] R. Hassin, "Approximation schemes for the restricted shortest path problem", Math. Oper. Res., vol. 17, pp. 36–42, 1992.
[16] D. Lorenz, D. Raz, "A simple efficient approximation scheme for the restricted shortest path problem", Bell Labs Tech. Memo., 1999.
[17] S. Chen, K. Nahrstedt, "On finding multi-constrained paths", in IEEE ICC'98, Jun. 1998, pp. 874–879.
[18] A. Goel, K. G. Ramakrishnan, D. Kataria, D. Logothetis, "Effi- cient computation of delay-sensitive routes for one source to all desti- nations", in Proc. IEEE INFOCOM, 2001, pp. 854–858.
[19] M. Garey, D. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness. New York: W. H. Freeman and Co., 1979.
[20] H. C. Joksch, "The shortest route problem with constraints", J. Math. Anal. Applicat., vol. 14, pp. 191–197, 1966.
[21] D. Lorenz, D. Raz, "A simple efficient approximation scheme for the restricted shortest paths problem", Bell Labs Tech. Memo., 1999.
[22] S. Sahni, "General techniques for combinatorial approximation", Oper. Res., vol. 26, no. 5, 1977.
[23] M. Faloutsos, P. Faloutsos, C. Faloutsos, "On power-law relationships of the Internet topology", presented at the ACM SIGCOMM, Boston, MA, 1999.
[24] B. M. Waxman, "Routing of multipoint connections", IEEE J. Sel. Areas Commun., vol. 6, no. 9, pp. 1617–1622, Dec. 1988.
[25] H. de Neve, P. van Mieghem, "A multiple quality of service routing algorithm for PNNI", in Proc. IEEE ATM Workshop, 1998, pp. 324–328.

R.Pravallika pursuing M.Tech in Gudlavalleru Engineering College,Gudlavalleru, JNTUK University and received B.Tech degree in Computer Science & Engineering from V.R.S & Y.R.N COLLEGE,Chirala in 2009.