Two Techniques for Fast Computation of Constrained Shortest Paths

Shigang Chen Meongchul Song Sartaj Sahni
Department of Computer & Information Science & Engineering
University of Florida
{sahni, sgchen, msong}@cise.ufl.edu

Abstract— **Computing constrained shortest paths is fundamental to some important network functions such as QoS routing, which 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 calls. Because it is NP-complete, much research has been designing heuristic algorithms that solve the ε-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 the costly computation for 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. When there are multiple constraints, the improvement is more dramatic.**

I. Introduction

A major obstacle against implementing distributed multimedia applications (e.g., web broadcasting, video teleconferencing, and remote diagnosis) is the difficulty of ensuring QoS (Quality of Service) over the Internet. Besides the issues of packet scheduling, admission control, resource reservation, and traffic engineering, the QoS routing is a critical element for QoS provision. It is to find a constrained shortest path — a network path that satisfies a given set of constraints (e.g., minimum bandwidth requirement and bounded end-to-end delay) [1], [2], [3], [4], [5], [6], [7]. For interactive real-time traffic, the delay-constrained least-cost path has particular importance [8]. It is the cheapest path whose end-to-end delay is bounded by the delay requirement of a time-sensitive data flow such as a voice call. 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 (long-term or short-term) virtual circuits in ATM networks, establishing wavelength-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 protocol (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 computaion. This approach provides support for both constrained unicast and constrained multicast. The computation load on a router is independent of the request arrival rate. 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. Let m and n be the number of links and the number of nodes in the network, respectively. Juttner et al. used the Lagrange relaxation method to approximate the delay-constrained least-cost routing problem in time $O(m^2 \log^4 m)$ [9]. However, there is no theoretical bound on how large the cost of the found path can be. Korkmaz and Krunz used a non-linear target function to approximate the multi-constrained least-cost path problem [10]. It was proved that the path that minimizes the target function satisfies one constraint and the other constraints multiplied by $\sqrt[\lambda]{k}$, where λ is a predefined constant and k is the number of constraints. However, no known algorithm

[1] Note that path caching does not eliminiate the problem of finding constrained shortest paths because those paths must be calcuated before being cached.

can find such a path in polynomial time. [10] 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 returned path, nor provide conditional guarantee in finding a feasible 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 destinations. 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 polynomial time algorithms that solves the NP-complete problem with an accuracy that is theoretically bounded. Given a small constant ε, Hassin's algorithm [11] has a time complexity of $O(\frac{mn}{\varepsilon}\log\log\frac{\text{UB}}{\text{LB}})$, 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 $(1+\varepsilon)$. Chen and Nahstedt solved a similar problem in time $O((m+n\log n)x)$, where $x=O(n/\varepsilon)$ in order to achieve the ε-approximation [12]. Goel et al.'s algorithm [13] has the best-known complexity of $O((m+n\log n)\frac{L}{\varepsilon})$, where L is the length (hops) of the longest path in the network. 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.

One common technique of the above algorithms [11], [12], [13] is to discretize the link delay (or link cost). Due to the discretization, 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 discretization 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 non-linear 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 statistic mean of the discretization error is zero on a path P and the standard deviation is proportional to $\sqrt{l(P)}$, where $l(P)$ is the length of P. 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 ε-approximation of the constrained shortest-path problem. We prove the correctness of the algorithms, and demonstrate their efficiency by simulations.

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. Section V and VI provide analytical and simulation results, respectively. Section VII draws the conclusion.

II. Problem Definition and Existing Discretization Approaches

Consider a network $G\langle V,E\rangle$, 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 $l(P)$ 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 $d(P)\le r$. 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

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

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 [14]. 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 [15] or the extended Dijkstra's algorithm [12].

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

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

Now suppose the link delays are allowed to be zero. 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 [13].

The above integer-delay special case points out a heuristic solution for the general NP-complete problem, which is to discretize (scale and then round) arbitrary link delays to integers [11], [12], [16], [13]. There are two existing discretization approaches, *round to ceiling* [12] and *round to floor* [13]. Both approaches map the delay requirement r to a selected integer λ, while the link delays are discretized as follows.

Round to ceiling (RTC): For every link (u,v), the delay value is divided by $\frac{r}{\lambda}$. If the result is not an integer, it is rounded to the nearest larger integer.

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

Round to floor (RTF): For every link (u,v), the delay value is divided by $\frac{r}{\lambda}$. If the result is not an integer, it is rounded to the nearest smaller integer.

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

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

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

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 $P_{s,t}$. 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 [13].

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

$$\Delta^f(u,v) = d(u,v) - d^f(u,v)\frac{r}{\lambda} \tag{3}$$
$$\Delta^c(u,v) = d(u,v) - d^c(u,v)\frac{r}{\lambda} \tag{4}$$

The discretization error of a path P is defined as

$$\Delta^f(P) = \sum_{(u,v) \text{ on } P} \Delta^f(u,v) \tag{5}$$
$$\Delta^c(P) = \sum_{(u,v) \text{ on } P} \Delta^c(u,v) \tag{6}$$

III. Randomized Discretization

RTC creates positive rounding error on every link. The error accumulates along a path. The larger the topology, the longer a path, the larger the accumulated error. The same thing is true for RTF, which has negative rounding error on every link. The insight is that if we can reduce the error introduced by discretization, we can improve the performance of the algorithm. With a smaller error, the new problem after discretization is closer to the original problem. The solution to the new problem will also be closer to the solution of the original problem.

Our first approach is 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.

Round randomly (RR): For every link (u,v), the delay value is divided by $\frac{r}{\lambda}$. 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.

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

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

$$\Delta^r(u,v) = d(u,v) - d^r(u,v)\frac{r}{\lambda} \tag{8}$$

and the discretization error of a path P is

$$\Delta^r(P) = \sum_{(u,v) \text{ on } P} \Delta^r(u,v) = d(P) - d^r(P)\frac{r}{\lambda} \tag{9}$$

Following the iterative approach of [13], the randomized discretization algorithm (RDA) is described below. Let λ_0 be a small constant. We use the extended Dijkstra's shortest path algorithm (EDSP), which is equivalent to Joksch's algorithm, except that $w[v,i]$ stores the cost of the cheapest path P from s to v with $d^r(P) = i$.

The algorithm assumes a preprocessing step that removes all nodes to which there are no feasible paths from s.

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

Relax_RDA(u, v, i, λ)
4. $i' := i + d^r(u,v)$
5. $error := \delta[u,i] + \Delta^r(u,v)$

6. **if** $error < 0$ **then**
7. $error := error + r/\lambda$
8. $i' := i' - 1$
9. **if** $i' \leq \lambda$ and $w[v, i'] > w[u, i] + c(u, v)$ **then**
10. $w[v, i'] := w[u, i] + c(u, v)$
11. $\pi[v, i'] := u$
12. $\delta[v, i'] := \min\{\delta[v, i'],\ error\}$

EDSP_RDA(G, s, λ)
13. Initialize(V, s, λ)
14. **for** $i = 0$ to λ **do**
15. $Q := V$
16. **while** $Q \neq \emptyset$ **do**
17. $u :=$ Extract_Min(Q)
18. **if** $w[u, i] = \infty$ **then**
19. break out of the while loop
20. $Q := Q - \{u\}$
21. **for** every adjacent node v of u **do**
22. Relax_RDA(u, v, i, λ)

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

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 $\delta[u, i] \geq 0$ and $\delta[v, i'] \geq 0$ before Relax_RDA(...) is called. Because $-\frac{r}{\lambda} \leq \Delta^r(u, v) \leq \frac{r}{\lambda}$, Lines 6-7 make sure that $error \geq 0$. Hence, $\delta[v, i'] \geq 0$ after Line 12. The lemma remains true after the call. □

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

Proof: Suppose it holds before Relax_RDA(...) is called. $P_i^u \geq i\frac{r}{\lambda} + \delta[u, i]$. The new path under consideration is $P_i^u + (u, v)$.

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

After Lines 4-8, $d(P_i^u + (u, v)) \geq i'\frac{r}{\lambda} + error$. After Line 12, $\delta[v, i'] \leq error$. Hence, $d(P_i^u + (u, v)) = d(P_{i'}^v) \geq i'\frac{r}{\lambda} + \delta[v, i']$. The lemma holds after the call. □

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

Proof: Suppose it holds before Relax_RDA(...) is called. $P_i^u \leq (i + l(P_i^u))\frac{r}{\lambda}$. The new path under consideration is $P_i^u + (u, v)$.

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

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

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

Proof: We first prove that if RDA terminates, it solves the ε-approaximation 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(...) is called on a link (u, v) of $P_{s,t}$. After Lines 4-5, $i' = i + d^r(u, v) = i + (d(u, v) - \Delta^r(u, v))\frac{\lambda}{r} = i + (d(u, v) - error + \delta[u, i])\frac{\lambda}{r}$. After Lines 6-8, because $error \geq 0$, $i' \leq i + \delta[u, i]\frac{\lambda}{r} + d(u, v)\frac{\lambda}{r}$. By Lemma 2, $i + \delta[u, i]\frac{\lambda}{r} \leq d(P_i^u)\frac{\lambda}{r}$. We have $i' \leq (d(P_i^u) + d(u, v))\frac{\lambda}{r} \leq d(P_{s,t})\frac{\lambda}{r} \leq \lambda$. 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 $\min\{w[t, i] \mid i \in [0..\lambda]\}$. We must have $c(p^t) \leq c(P_{s,t})$, and $d(P^t) \leq (1+\varepsilon)r$ because otherwise RDA won't terminate.

We now prove that RDA terminates in time $O((m + n \log n)L/\varepsilon)$. When $\lambda \geq \frac{L}{\varepsilon}$, by Lemma 3, $\forall t \in V$, $d(P^t) \leq (\lambda + l(P^t))\frac{r}{\lambda} \leq r + l(P^t)\frac{\varepsilon r}{L} \leq (1 + \varepsilon)r$. By Line 27, RDA terminates.

The time complexity of each execution of EDSP_RDA(...) 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 $O((m + n \log n)L/\varepsilon)$. □

It is easy to see why RDA has the same worst-case time complexity as DSA. It could happen that $d^r(u, v) = d^f(u, v), \forall (u, v) \in E$, which makes RDA identical to DSA. However, with much larger probabilities, $d^r(u, v)$ follows a distribution with positive errors and negative errors cancelling out each other along a long path, which allows RDA to run much faster than DSA on an average case. For an arbitrary routing instance, it can be proved that if DSA can terminate with a λ value, then RDA must be able to terminate with the same λ value; the opposite statement is not true. RDA usually terminates with a smaller λ value than

DSA.

IV. Path Delay Discretization

RTF, RTC, and RR all perform discretization at the link level. Each link carries certain amount of error, which may accumulate along a path. Another way to control the total error is to perform discretization on the path level, using the interval partitioning method for combinatorial approximation [17]. Given a path P,

$$d'(P) = \lfloor \frac{d(P)}{r} \lambda \rfloor \tag{10}$$

The error is independent of the path length. The path discretization algorithm (PDA) is shown below. EDSP_PDA is omitted because it is identical to EDSP_RDA except that it calls Relax_PDA.

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

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

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

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 $z[u,i] \ge i\frac{r}{\lambda},\ \forall u \in V, i \in [0..\lambda]$.

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

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

When Line 8 is executed, $z[v,i'] \ge i'\frac{r}{\lambda}$ remains true. □

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

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

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

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

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

The proof is similar to that for Theorem 1.

Both RDA and PDA can be easily extended to handle more than one constraints (Appendix A).

V. Analysis

For RTF, the discretization error of every link is non-negative with a tight upper bound of $\frac{r}{\lambda}$. Hence, the discretization errors of links on a path P will add up to a non-negative value with a tight upper bound of $\frac{r}{\lambda}l(P)$, which is linear to the path length. Statistically, the longer the path, the larger the error. For instance, if $\Delta^f(u,v),\ \forall (u,v) \in P$, is uniformly distributed in $[0, \frac{r}{\lambda})$, the mean of $\Delta^f(P)$ is $\frac{r}{2\lambda}l(P)$.

For RTC, the discretization error of every link is always non-positive with a tight lower bound of $-\frac{r}{\lambda}$. If $\Delta^c(u,v),\ \forall (u,v) \in P$, is uniformly distributed in $(-\frac{r}{\lambda}, 0]$, the mean of $\Delta^c(P)$ is $-\frac{r}{2\lambda}l(P)$.

The error of the path delay discretization is always non-negative with a tight upper bound of $\frac{r}{\lambda}$, independent of the length of the path.

To study RR, we model $d(u,v), \forall (u,v) \in E$, as a random variable, whose probability density function is $f_{u,v}(x), x \in [0, +\infty)$. For any path P, $\Delta^r(P)$ is also a random variable. Assume the delays of different links are independent.

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 $\frac{r\sqrt{l(P)}}{2\lambda}$, regardless of the probability distributions of the link delays.

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

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

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

$$\begin{aligned}
E(\Delta^r(u,v)) = (d(u,v) - \lceil \frac{d(u,v)}{r}\lambda \rceil \frac{r}{\lambda}) \cdot p_1 \\
+ (d(u,v) - \lfloor \frac{d(u,v)}{r}\lambda \rfloor \frac{r}{\lambda}) \cdot p_2
\end{aligned}$$

There are two cases.

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

$$\begin{aligned}
p_2 &= 1 - \frac{d(u,v)}{r}\lambda + \lfloor \frac{d(u,v)}{r}\lambda \rfloor \\
&= \lceil \frac{d(u,v)}{r}\lambda \rceil - \frac{d(u,v)}{r}\lambda
\end{aligned} \tag{11}$$

$$\begin{aligned}
&E(\Delta^r(u,v))\\
&= (d(u,v) - \lceil \frac{d(u,v)}{r}\lambda \rceil \frac{r}{\lambda}) \cdot (\frac{d(u,v)}{r}\lambda - \lfloor \frac{d(u,v)}{r}\lambda \rfloor)\\
&\quad + (d(u,v) - \lfloor \frac{d(u,v)}{r}\lambda \rfloor \frac{r}{\lambda}) \cdot (\lceil \frac{d(u,v)}{r}\lambda \rceil - \frac{d(u,v)}{r}\lambda)\\
&= 0
\end{aligned}$$

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

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

When $d(u,v) = x$, $p_1 = \frac{x}{r}\lambda - \lfloor \frac{x}{r}\lambda \rfloor$ by (7), and $p_2 = \lceil \frac{x}{r}\lambda \rceil - \frac{x}{r}\lambda$ by (11). Hence,

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

Because $y(1-y), y \in [0,1)$, reaches its maximum value of 1/4 when $y = 1/2$, we have

$$\begin{aligned}
V(\Delta^r(u,v)) &\le \frac{r^2}{\lambda^2} \int_0^\infty \frac{1}{4} f_{u,v}(x)dx\\
&= \frac{r^2}{4\lambda^2}
\end{aligned}$$

Therefore, given an *arbitrary* probability density function of $d(u,v)$, we showed that $E(\Delta^r(u,v)) = 0$ and $V(\Delta^r(u,v)) \le \frac{r^2}{4\lambda^2}$, for every (u,v) on P. The mean and the variance of $\Delta^r(P)$ are

$$\begin{aligned}
E(\Delta^r(P)) &= E(\sum_{(u,v) \text{ on } P} \Delta^r(u,v))\\
&= \sum_{(u,v) \text{ on } P} E(\Delta^r(u,v)) = 0
\end{aligned}$$

$$\begin{aligned}
V(\Delta^r(P)) &= V(\sum_{(u,v) \text{ on } P} \Delta^r(u,v))\\
&= \sum_{(u,v) \text{ on } P} V(\Delta^r(u,v)) \le \frac{r^2 l(P)}{4\lambda^2}
\end{aligned}$$

The standard deviation of $\Delta^r(P)$ is

$$\sigma(\Delta^r(P)) = \sqrt{V(\Delta^r(P))} \le \frac{r\sqrt{l(P)}}{2\lambda}$$

□

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. As shown in the figure, the discretization errors of RTF and RTC grow linearly with the path length,[2]

[2]When the link delay follows an exponential distribution, the average error caused by RTF is smaller than that caused by RTC. However, when the link delay follows a uniform distribution, the average error by RTF is the same as that by RTC.

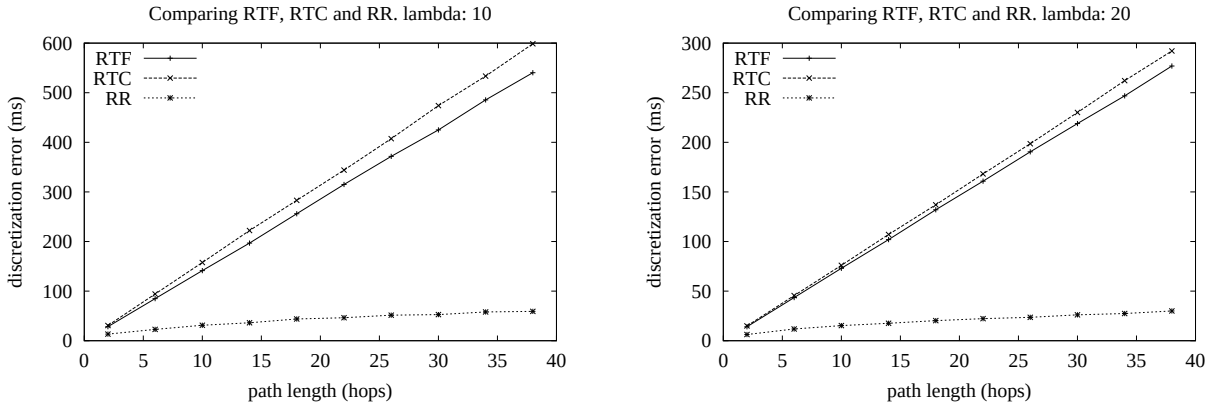

Fig. 1. Compare the average discretization errors of RTF, RTC and RR with respect to different path lengths. The vertical axis is the average of $|\Delta^f(P)|$, $|\Delta^c(P)|$, or $|\Delta^r(P)|$ over 10000 sample paths.

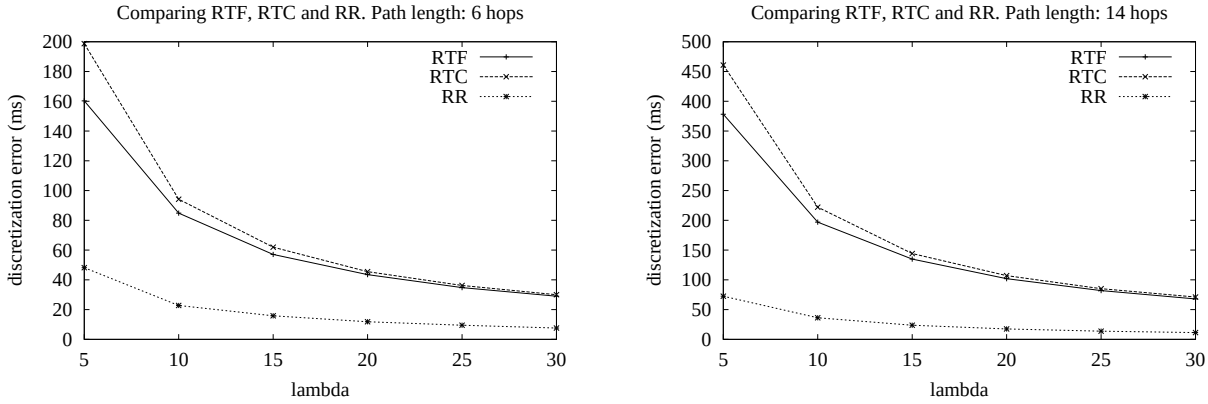

Fig. 2. Compare the average discretization errors of RTF, RTC and RR with respect to different λ values. The vertical axis is the average of $|\Delta^f(P)|$, $|\Delta^c(P)|$, or $|\Delta^r(P)|$ over 10000 sample paths.

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 [18] and the Waxman model [19]. For a Power-Law topology, 10% of the nodes has a degree of one, and the degrees of the other nodes follow a power law, i.e., the frequency f_d of a degree is proportional to the degree d (≥ 2) raised to the power of a constant $O = -2.2$.

$$f_d \propto d^O$$

After each node is assigned a degree according to the power law, 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

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

where $d(u, v)$ is the distance between u and v, $\beta = 0.6$, 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. $\varepsilon = 0.1$. $\lambda_0 = 3$. 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 2GHz CPU and 512 Megabytes memory.

The performance metrics used to evaluate the routing al-

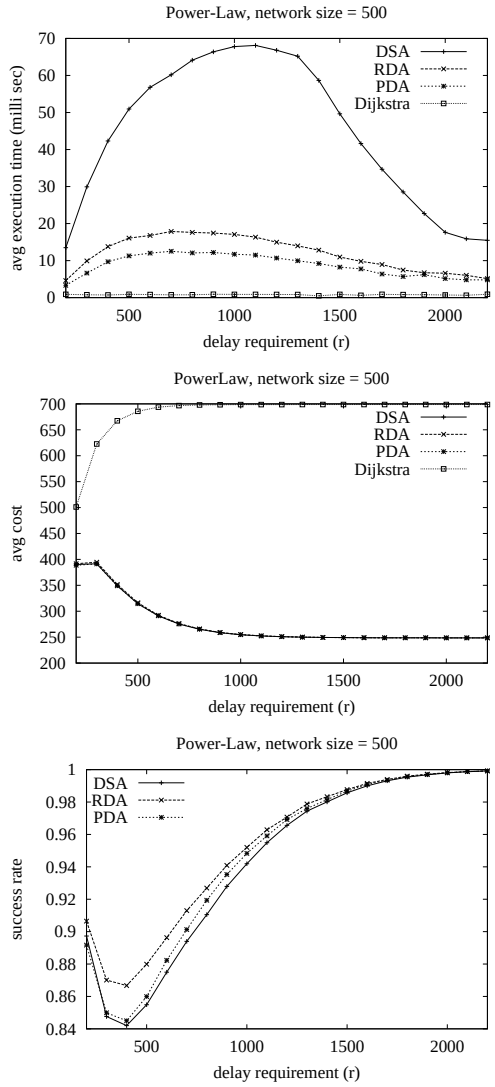

Fig. 3. Compare DSA, RDA, and PDA on Power-Law topologies

gorithms 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 $(1+\varepsilon)r$.

B. Comparing RDA and PDA with DSA

Fig. 3 compares DSA, RDA, and PDA 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

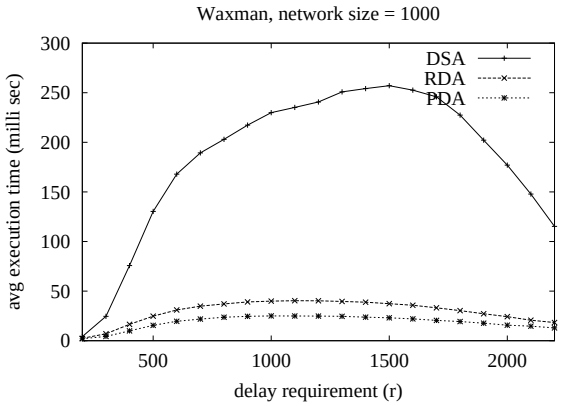

Fig. 4. Compare DSA, RDA, and PDA on Waxman topologies

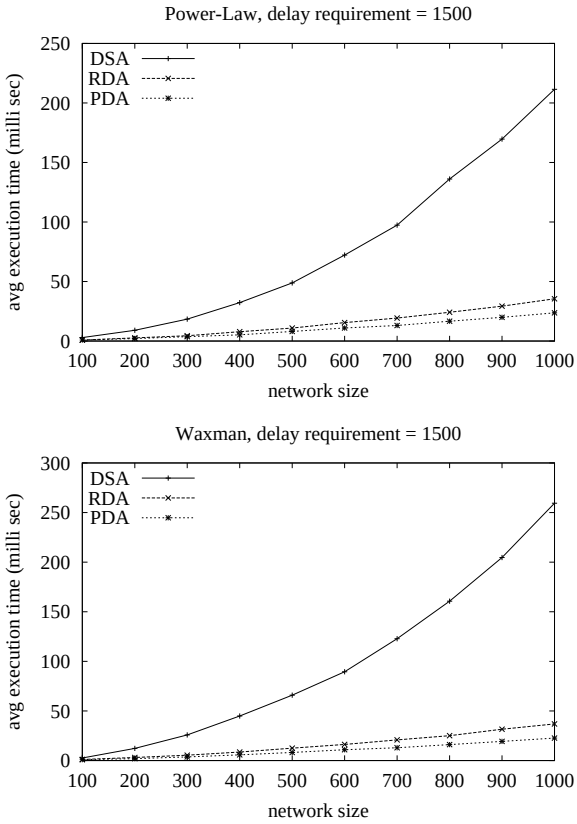

Fig. 5. Scalability comparison

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 gains by RDA and PDA increase for larger topologies. The improvement exceeds an order of magnitude for 1000-node networks.

Fig. 6 compares the algorithms with different ε values. The differences are bigger when ε is smaller.

In summary, the simulations confirmed our prediction that

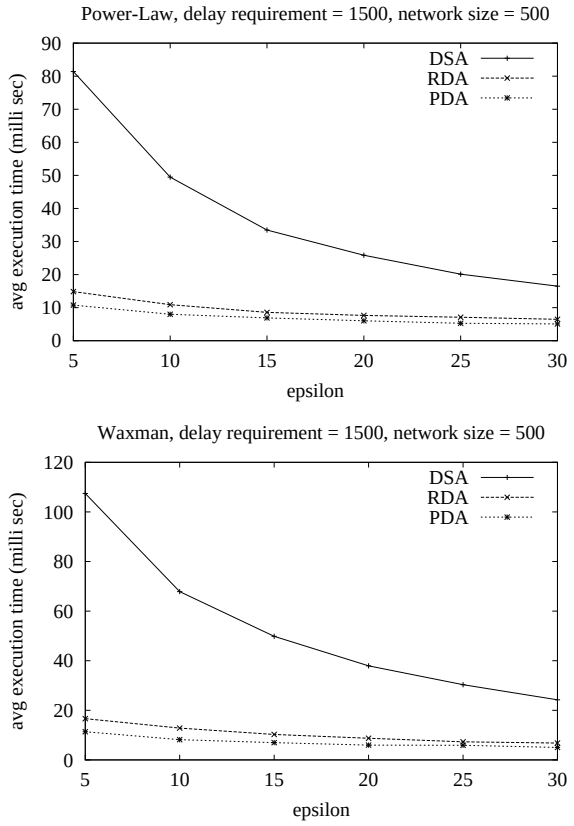

Fig. 6. Compare DSA, RDA, and PDA with respect to different ε values

the execution time could be greatly improved by reducing the discretization error, which was achieved very effectively by RDA and PDA. Even 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.

C. Multicast-Constrained Shortest Paths

The comparsion when there are two constraints (e.g., bounded delay and bounded lay jitter) are shown in Tables I and II. For networks with 300 nodes, PDA outperforms DSA by 21 times on Powerlaw topologies and 29 times on Waxman topologies.

The last iteration of DSA, RDA, or PDA dominates in terms of both execution time and memory usage, which are $O((m + n \log n)\lambda^k)$ and $O(n\lambda^k)$, respectively, where k is the number of constraints. Therefore, for any run of the algorithms, the memory usage is proportional to the execution time. The previous comparison on execution time thus provides a relative comparison on memory usage as well.

D. Comparing RDA and PDA with H_MCOP

We compare RDA and PDA with a fast heuristic algorithm H_MCOP [10], whose time complexity is the same as

TABLE I

EXECUTION TIME (MILLISECONDS) ON POWER-LAW TOPOLOGIES WITH TWO CONSTRAINTS

no. of nodes	DSA	RDA	PDA
100	100.0	12.0	4.4
200	471.1	44.1	22.2
300	1437.2	116.3	67.5
400	3127.3	264.1	129.2
500	6252.7	638.8	328.9

TABLE II

EXECUTION TIME (MILLISECONDS) ON WAXMAN TOPOLOGIES WITH TWO CONSTRAINTS

no. of nodes	DSA	RDA	PDA
100	115.5	8.0	5.6
200	551.3	38.1	23.9
300	2056.3	150.5	70.3
400	3951.2	236.1	111.6
500	8550.9	473.0	211.3

TABLE III

EXECUTION TIME (MILLISECONDS) OF FINDING DELAY-CONSTRAINED LEAST-COST PATHS FROM A SOURCE TO ALL DESTINATIONSON ON POWER-LAW TOPOLOGIES

no. of nodes	RDA	PDA	H_MCOP, $\lambda = 2$
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

that of Dijkstra's algorithm. H_MCOP relies on building a shortest-path tree from all nodes to a specific destination and a source tree from a source to all other nodes. This design is suitable to construct a delay-constrained least-cost (DCLC) path from one source to one destination. It is not suitable to construct DCLC paths from one source to all destinations. To solve this problem, H_MCOP would have to repeat n times, with a total time complexity of $O(nm + n^2 \log n)$. On the other hand, RDA or PDA constructs a single constrained shortest path or a constrained shortest-path tree with unchanged time complexity of $O((m + n \log n)\frac{L}{\epsilon})$.

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

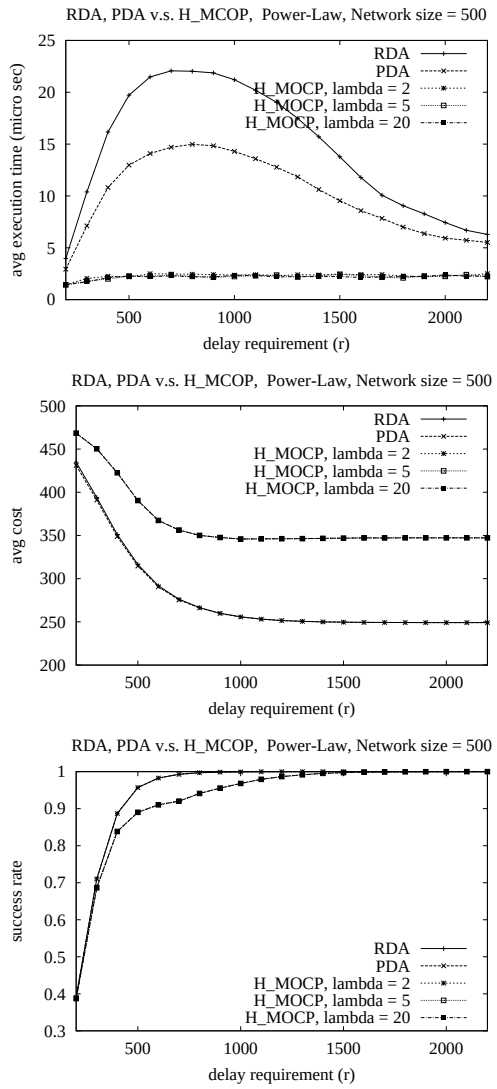

Fig. 7. Compare RDA and PDA with H_MCOP

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 significant only when there are two or more constraints. In this paper, we only have one constraint, the delay constraint. 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 $(1+\varepsilon)$. H_MCOP is a more efficient *on-line algorithm* than RDA/PDA.

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, for instance, for voice traffic with an end-to-end delay requirement of 100 ms. 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 orders of magnitude when the number of nodes is large, as shown in Table III.

Therefore, H_MCOP is more suitable as an on-line 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 algorithms for the delay-constrained least-cost routing problem. While the previous 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. Although the techniques were described in the context of delay-constrained least-cost routing, they can be easily used for multi-constrained least-cost routing such as delay-delayjitter-constrained least-cost routing.

References

[1] S. Chen and 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 Digital Video*, December 1998.

[2] R. Guerin and A. Orda, "QoS-based Routing in Networks with Inaccurate Information: Theory and Algorithms," *IEEE INFOCOM'97, Japan*, April 1997.

[3] T. Korkmaz and M. Krunz, "Source-oriented topology aggregation with multiple QoS parameters in hierarchical ATM networks," *IEEE IWQoS'99*, June 1999.

[4] Danny Raz and Yuval Shavitt, "Optimal Partition of QoS Requirements with Discrete Cost Functions," *IEEE INFOCOM'2000*, March 2000.

[5] J. L. Sobrinho, "Algebra and Algorithms for QoS Path Computation and Hop-by-Hop Routing in the Internet," *IEEE INFOCOM'2001*, April 2001.

[6] Z. Wang and Jon Crowcroft, "QoS Routing for Supporting Resource Reservation," *IEEE JSAC*, September 1996.

[7] X. Yuan, "Heuristic Algorithms for Multi–Constrained Quality of Service Routing ," *IEEE INFOCOM'2001*, April 2001.

[8] H. F. Salama, D. S. Reeves, and Y. Viniotis, "A Distributed Algorithm for Delay-Constrained Unicast Routing," *IEEE INFOCOM'1997*, pp. 84–91, April 1997.

[9] A. Juttner, B. Szviatovszki, I. Mecs, and Z. Rajko, "Lagrange Relaxation Based Method for the QoS Routing Problem," *IEEE INFOCOM'2001*, April 2001.

[10] T. Korkmaz and M. Krunz, "Multi-Constrained Optimal Path Selection," *IEEE INFOCOM'2001*, April 2001.

[11] R. Hassin, "Approximation Schemes for the Restricted Shortest

Path Problem," *Mathematics of Operations Research*, vol. 17, pp. 36–42, 1992.

[12] S. Chen and K. Nahrstedt, "On Finding Multi-Constrained Paths," *IEEE International Conference on Communications (ICC'98)*, June 1998.

[13] A. Goel, K. G. Ramakrishnan, D. Kataria, and D. Logothetis, "Efficient Computation of Delay-Sensitive Routes for One Source to All Destinations," *IEEE INFOCOM'2001*, April 2001.

[14] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, New York: W.H. Freeman and Co., 1979.

[15] H. C. Joksch, "The Shortest Route Problem with Constraints," *Journal of Mathematical Analysis and Applications*, vol. 14, pp. 191–197, 1966.

[16] D. Lorenz and D. Raz, "A Simple Efficient Approximation Scheme for the Restricted Shortest Paths Problem," *Bell Labs Technical Memorandun*, 1999.

[17] S. Sahni, "General Techniques for Combinatorial Approximation," *Operations Research*, vol. 26, no. 5, 1977.

[18] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos, "On Power-Law Relationships of the Internet Topology," *ACM Proceedings of SIGCOMM '99*, 1999.

[19] B. M. Waxman, "Routing of Multipoint Connections," *IEEE Journal of Selected Area in Communications*, pp. 1617–1622, Dec. 1988.

[20] H. De Neve and P. Van Mieghem, "A Multiple Quality of Service Routing Algorithm for PNNI," *IEEE ATM Workshop*, May 1998.

APPENDIX A. RDA AND PDA FOR MULTIPLE CONSTRAINTS

Suppose each link (u, v) has $(k + 1)$ additive metrics, $d_j(u, v),\ j \in [1..k]$, and $c(u, v)$. The *cheapest feasible path* $P_{s,t}$ from s to t is defined as

$$d_j(P_{s,t}) \le r_j,\ j \in [1..k]$$
$$c(P_{s,t}) = \min\{c(P) \mid d_1(P) \le r_1 \wedge ... \wedge d_k(P) \le r_k, \forall \text{path } P \text{ from } s \text{ to } t\}$$

The randomized discretization becomes, $\forall j \in [1..k]$,

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

$$\Delta_j^r(u,v) = d_j(u,v) - d_j^r(u,v)\frac{r_j}{\lambda}$$

Initialize(V, s, λ)
  for each vertex $v \in V$ **do**
    for each $i_1 \in [0..\lambda]$, ..., each $i_k \in [0..\lambda]$ **do**
      $w[v, i_1, ..., i_k] := \infty,\ \pi[v, i_1, ..., i_k] :=$ NIL
    for $j = 1$ to k, $i = 0$ to λ **do**
      $\delta_j[v, i] := \infty$
  $w[s, 0, ..., 0] := 0,\ \delta_1[s, 0] := 0,\ \delta_k[s, 0] := 0$

Relax_RDA($u, v, i_1, ..., i_k, \lambda$)
  for $j = 1$ to k **do**
    $i'_j := i_j + d_j^r(u, v)$
    $error_j := \delta_j[u, i_j] + \Delta_j^r(u, v)$
    if $error_j < 0$ **then**
      $error_j := error_j + r_j/\lambda$
      $i'_j := i'_j - 1$
  if $i'_1 \le \lambda$ ... and $i'_k \le \lambda$ **then**
    if $w[v, i'_1, ..., i'_k] > w[u, i_1, ..., i_k] + c(u, v)$ **then**
      $w[v, i'_1, ..., i'_k] := w[u, i_1, ..., i_k] + c(u, v)$
      $\pi[v, i'_1, ..., i'_k] := u$
      for $j = 1$ to k **do**
        $\delta_j[v, i'_j] := \min\{\delta_j[v, i'_j],\ error_j\}$

EDSP_RDA(G, s, λ)
  Initialize(V, s, λ)
  for $i_1 = 0$ to λ, ..., $i_k = 0$ to λ **do**
    $Q := V$
    while $Q \neq \emptyset$ **do**
      $u :=$ Extract_Min(Q)
      if $w[u, i_1, ..., i_k] = \infty$ **then**
        break out of the while loop
      $Q := Q - \{u\}$
      for every adjacent node v of u **do**
        Relax_RDA($u, v, i_1, ..., i_k, \lambda$)

RDA(G, s)
  $\lambda := \lambda_0$
  do

$\lambda := 2\lambda$
EDSP_RDA(G, s, λ)
while $\exists v \in V, \exists j \in [1..k],\ d_j(P^v) > (1+\varepsilon)r_j \wedge d_j(P^v) \neq \infty$,
where P^v is the path with min$\{w[v, i_1, ..., i_k] \mid$
$i_1, ..., i_k \in [0..\lambda]\}$

EDSP_PDA is omitted because it is identical to EDSP_RDA except that it calls Relax_PDA.

Initialize(V, s, λ)
for each vertex $v \in V$ **do**
for each $i_1 \in [0..\lambda]$, ..., each $i_k \in [0..\lambda]$ **do**
$w[v, i_1, ..., i_k] := \infty,\ \pi[v, i_1, ..., i_k] :=$ NIL
for $j = 1$ to k, $i = 0$ to λ **do**
$z_j[v, i] := \infty$
$w[s, 0, ..., 0] := 0,\ z_1[s, 0] := 0,\ z_k[s, 0] := 0$

Relax_PDA($u, v, i_1, ..., i_k, \lambda$)
for $j = 1$ to k **do**
$i'_j := \lfloor \frac{z_j[u,i_j]+d_j(u,v)}{r_j} \lambda \rfloor$
if $i'_1 \leq \lambda$ ... and $i'_k \leq \lambda$ **then**
if $w[v, i'_1, ..., i'_k] > w[u, i_1, ..., i_k] + c(u, v)$ **then**
$w[v, i'_1, ..., i'_k] := w[u, i_1, ..., i_k] + c(u, v)$
$\pi[v, i'_1, ..., i'_k] := u$
for $j = 1$ to k **do**
$z_j[v, i'_j] :=$ min$\{z_j[v, i'_j],\ z_j[u, i_j] + d_j(u, v)\}$

PDA(G, s)
$\lambda := \lambda_0$
do
$\lambda := 2\lambda$
EDSP_PDA(G, s, λ)
while $\exists v \in V, \exists j \in [1..k],\ d_j(P^v) > (1+\varepsilon)r_j \wedge d_j(P^v) \neq \infty$,
where P^v is the path with min$\{w[v, i_1, ..., i_k] \mid$
$i_1, ..., i_k \in [0..\lambda]\}$
