An Improved Approximation Algorithm for Computing Disjoint QoS Paths

Chao Peng * Hong Shen
Graduate School of Information Science
Japan Advanced Institute of Science and Technology
1-1 Tatsunokuchi, Ishikawa, 923-1292, Japan
E-mail: p-chao@jaist.ac.jp

Abstract

The survivability of a network has assumed great importance in against of losing huge volumes of data due to a link cut or node failure. Recently some scholars have proposed some path restoration schemes which used two disjoint paths with multiple constraints to satisfy both the survivability and the QoS requirements. In this paper we will study the issue of how to identify two paths that satisfy the multiple QoS constraints imposed by network applications. More specifically, we will focus on finding two link-disjoint paths that satisfy the delay constraints at a reasonable total cost. We present two efficient approximation algorithms with provable performance guarantees for this problem.

1 Introduction

As networks modernize and expand with the increasing deployment of optical technology, the large bandwidth offered by the optical fiber has brought tremendous potential for exploitation. The number of services offered to customers over a fiber network is proliferating, but the risk of losing huge volumes of data due to a span cut or node failure (due to equipment breakdown at a central office or other events such as fires, flooding, etc.) has also escalated. At the same time, the wireless ad hoc network is developing very fast and high quality video applications are expected to become available in wireless ad hoc networks in near future. But the unpredictable nature of the wireless environment is easily prone to link failures (e.g. due to channel fading or obstructions) and resulting path failures and data loss.

In both situations the survivability of a network has assumed great importance. Survivability refers to the ability of a network to provide continuity of service with no disruption, no matter how much the network may be damaged due to events such as link failures or node failures. Survivability and QoS can be achieved by maintaining two disjoint QoS-constrained paths to increase the probability that source can reach the destination via another path as the network undergoes topological changes, thus avoid a unreasonable loss of service quality.

Although many works have been done to find multiple node-disjoint or link-disjoint paths in a given network [2, 4], the problem of finding two disjoint QoS-constrained paths has got little attention. So far the best work done in this area is due to [12, 13], in which the authors proposed 4 algorithms to compute two delay-constrained link-disjoint paths with minimum total cost. If there exist two disjoint paths with delay less than D and total cost OPT, their algorithm 2DP-1 can find two paths with total delay less than $3D$ and total cost $(1.5+\varepsilon)OPT$. Other algorithms proposed in [12] can find two paths with total delay less than $2D(1+1/k)$ and total cost $k(1+\gamma)(1+\varepsilon)OPT$.

In this paper we propose two new approximation algorithms for this problem. Our first algorithm can find two paths with total delay less than $2D(1+1/k)$ and reduce the total cost to $(4\log k+3.5)OPT$, but it is a pseudo-polynomial algorithm. Our second algorithm is polynomial, it reduces the time complexity to $O(MN^4\log k/\varepsilon)$, and the cost is $(4\log k+3.5)(1+\varepsilon)OPT$.

The remainder of the paper is organized as follows. In Section 2, we describe the problem and the network model. In Section 3, we present our first approximation algorithm to reduce the total cost. In Section 4, we improve the time complexity of this algorithm and turn it into a polynomial algorithm. Finally, we conclude the paper in Section 5.

2 Model and Problem Formulation

The QoS constraints in a network can be divided into *bottleneck* constraints such as bandwidth, *additive* constraints such as delay or jitter and *multiplicative* constraints

*This research is conducted as a program for the "Fostering Talent in Emergent Research Fields" in Special Coordination Funds for promoting Science and Technology by Ministry of Education, Culture, Sports, Science and Technology.

Proceedings of the International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies (ICNICONSMCL'06)
0-7695-2552-0/06 $20.00 © 2006 IEEE

such as the packet loss rate or possibility. Bottleneck QoS constraints can be efficiently solved by removing links that violates the requirement. Multiplicative constraints can be reduced to additive constraints by a logarithm transformation. So here we only consider two additive constraints and we use *delay* and *cost* respectively to generically refer to two different *additive* constraints for simplicity of exposition.

We adopt the same model as used in [12], in which the network is represented by a directed graph $G(V, E)$, where V is the set of nodes and E is the set of links. The number of network nodes and links are respectively denoted by $N = |V|$ and $M = |E|$. An (s, t)-path is a finite sequence of distinct nodes $P = (s = v_0, v1, ..., t = v_n)$, such that, for $0 \leq i \leq n-1$, $(v_i, v_{i+1}) \in E$. Here, $n = |P|$ is the number of nodes in P. A cycle is a path whose source and destination nodes are identical.

Each link $l \in E$ has a delay guarantee d_l and a cost c_l which estimates the quality of the link in terms of resource utilization. The delay $D(P)$ of a path P is the sum of the delays of its links, i.e., $D(P) = \sum_{l \in P} d_l$. The cost $C(P)$ of a path P is defined to be the sum of the costs of its links, i.e., $C(P) = \sum_{l \in P} c_l$. We shall assume that all parameters (both delay guaranties and costs) are positive integers.

The following RSP Problem is a fundamental problem in QoS routing.

Problem RSP (Restricted Shortest Path): Given a source node s, a destination node t and a delay constraint D, find an (s, t)-path P such that

1) $D(P) \leq D$, and 2) $C(P) \leq C(\overline{P})$ for any other (s, t)-path $\overline{P}$ that satisfies $D(\overline{P}) \leq D$.

Problem RSP is NP-hard [3], but it admits an FPTAS with complexity of $O(MN/\varepsilon)$ [11]. But it requires that both the delay and the cost of each link must be positive. The algorithm in [10] has a time complexity of $O(MN(1/\varepsilon + \log\log N))$, it requires that both the delay and the cost of each link must be non-negative. Since in our algorithms we may set the cost of a link as zero in the residual graph, we shall use the latter algorithm as a basic stone and we shall refer to it as Algorithm RSP.

If we extend the Problem RSP to the case of two link-disjoint paths, we will get the problem we want to solve:

Problem 2DP (2-Restricted Link Disjoint Paths): Given a source node s, a destination node t and a QoS requirement D, find two link-disjoint (s, t)-paths P_1 and P_2 such that:

1) $D(P_1) \leq D$ and $D(P_2) \leq D$; 2) $C(P_1) + C(P_2) \leq C(\overline{P}_1) + C(\overline{P}_2)$ for every other pair of link-disjoint (s, t)-paths $(\overline{P}_1, \overline{P}_2)$ that satisfy $D(\overline{P}_1) < D$ and $D(\overline{P}_2) < D$.

We denote by OPT the cost of an optimal solution to Problem 2DP for (G, s, t, D). Problem 2DP includes Problem RSP as a special case; hence, it is NP-hard. In addition, it was proved in [12] that it is intractable to find a solution that does not violate the delay constraint of at least one of the paths. Furthermore, in most cases, we cannot provide an efficient solution without violating the delay constraint in both primary and restoration paths. So we can formulate a solution to Problem 2DP as a $(\alpha, \beta) - approximation$.

Definition 1 ($(\alpha, \beta) - approximations$) : Given an instance (G, s, t, D) of Problem 2DP, an (α, β)-approximate solution (P_1, P_2) to Problem 2DP is a solution for which:

1) $D(P_1) + D(P_2) \leq 2\alpha D$;

2) the total cost of two paths is at most β times more than that of the optimal solution, i.e., $C(P_1) + C(P_2) \leq \beta OPT$.

Let P_1 be the primary path with minimum delay, we have $D(P_1) \leq \alpha D$ and $D(P_2) \leq 2\alpha D$. Problem 2DP can be further extended to the following MCF problem:

Problem MCF (minimum constrained flow): Given a source node s, a destination node t and a delay requirement D, find an (s, t)-flow f such that:

1) $|f| = 2$; 2) $D(f) \leq 2D$; 3) $C(f) \leq C(\hat{f})$ for any other flow $\hat{f}$ that satisfies $|\hat{f}| = 2$ and $D(\hat{f}) \leq 2D$.

Since Problem MCF is a relaxation of Problem 2DP, the cost of the optimal solution to Problem MCF is no more than that of Problem 2DP. In next section we will use MCF in the process of computing and we will also use OPT to denote the cost of its optimal solution for convenience.

3 A $(1 + \frac{1}{k}, (4 \log k + 3.5))$-Approximation Algorithm for 2DP

In this section we present our approximation algorithm, which achieves an approximation ratio of $(1 + \frac{1}{k}, (4 \log k + 3.5))$. The basic idea of the algorithm is to identify a flow f from the source node s to the destination node t such that $f = 2$ and the total delay and cost of the flow satisfy some certain bounds, then we continuously find cycles with negative delay and bounded cost in the residual graph and augment f along these cycles. The algorithm stops when $D(f) \leq 2D(1 + \frac{1}{k})$.

The first step of the algorithm is to compute a flow f from s to t that satisfies the delay constraint $3D$ and the cost constraint $(1.5 + \varepsilon)OPT$. We use the algorithm 2DP-1 in [12] to achieve this.

The next step is to augment this flow in order to decrease its delay to $2D(1 + \frac{1}{k})$. To that end, we construct a residual network $G(f)$ imposed by the flow f. Intuitively, the residual network consists of links that can admit more flow.

Definition 3 (Residual Network) : Given a network G with unit capacities and flow f, the residual network $G(f)$ is constructed as follows. For each link $(u, v) \in G$ for which $f(u, v) = 0$, we add to $G(f)$ a link (u, v) of the same delay and cost as in G. For each link $(u, v) \in G$ for which $f(u, v) = 1$, we add to $G(f)$ a reverse link (v, u) to $G(f)$ with delay $-d_{(v,u)}$ and zero cost.

In the residual graph $G(f)$, we use algorithm MINDELAY to find a cycle W that minimizes the delay-to-cost ratio $\frac{D(W)}{C(W)}$. There are dozens of algorithm for finding such a minimum delay-to-cost ratio cycle, [9] is a good survey for those algorithms. Next, we augment flow f along W. This will decrease the total delay of f since the delay $D(W)$ is negative, but generally it will increase the total cost of f since the cost $C(W)$ must be positive. In case the negative cycle will bring huge cost penalty, we develop a new algorithm FINALIMPROVE which can find a negative delay cycle with bounded cost and delay-to-cost ratio. To find a feasible negative cycle, FINALIMPROVE will guess an estimation of OPT and call algorithm FCYCLE to check whether there exists such a cycle or not for such a guess. If FCYCLE fails, FINALIMPROVE will update the guess toward a correct direction and will finally find a feasible cycle. Then it augment the flow along this cycle and repeat the same process. FINALIMPROVE will stop when $D(f) \leq 2D(1+\frac{1}{k})$.

The final step is to decompose the flow f into two paths $\widehat{P}_1, \widehat{P}_2$ such that $D(\widehat{P}_1) \leq D(\widehat{P}_2)$. To do this we can adopt the same method in [8]. The following are the detailed description of the approximation algorithm 2DisjointPaths-1 and its fellow algorithms.

Algorithm 1. 2DisjointPaths-1(G, s, t, D, k)
Input:
G: the directed graph G=(V,E) with $\{d_l, c_l\}_{l \in E}$*;*
s : source node; *t : destination node;*
D: the delay constraint;
k : the approximation index;
Output: *(*$\widehat{P}_1, \widehat{P}_2$*): An* $(1+\frac{1}{k}, 4\log k + 3.5)$*-Approximation solution to Problem 2DP;*

```
1   (P1, P2) ← 2DP-1(G, s, t, D, 1/N);
2   f^0 ← {P1, P2};
3   if D(f^0) ≤ 2D(1 + 1/k) then
4       return (P1, P2) ;
5    f ← MINDELAY(G, f^0, D, k)
6   Decompose f into P̂1, P̂2 with D(P̂1) ≤ D(P̂2);
7   return (P̂1, P̂2) ;
```

Algorithm 2. MINDELAY(G, f^0, D, k)
Input:
G,k: the same as that in 2DisjointPaths-1;
f^0 *: the original flow;*
D: the delay constraint;
Output: f*: An improved flow with* $D(f) \leq 2D(1+\frac{1}{k})$ *for 2DP;*

```
1   f ← f^0;
2   while D(f) > 2D(1 + 1/k) do
3       Construct the residual network G(f)
        of G imposed by f:
4       Add to G(f) each link l in G with f_l = 0;
5       for each link (u, v) ∈ G with f_(u,v) = 1 do
6           Add a link (v, u) to G(f)
            with d_(v,u) = −d_(u,v) and c_(v,u) = 0;
7       Find a cycle W in G(f) which
        minimizes D(W)/C(W);
8       f^1 ← f, μ ← D(W)/C(W);
9       Augment flow f along W:
10  if C(f) > 2 * C(f^1) and C(f) > C(f^0) then ;
11      f ← FinalImprove(G, f^1, D, k, C(f^0), C(f), μ);
12  return f ;
```

Algorithm 3. FINALIMPROVE($G, f^1, D, k, C(f^0), C(f), \mu$)
Input:
G,D,k: the same as that in MinDelay;
f^1 *: the flow to be improved;*
$C(f^0)$ *: the cost of the original flow by 2DP-1;*
$C(f)$ *: the cost of the next flow with* $D(f) \leq 2D(1+\frac{1}{k})$*;*
μ *: the current minimum delay-to-cost ratio;*
Output: f^1*: An improved flow with* $D(f^1) \leq 2D(1+\frac{1}{k})$ *for Problem 2DP;*

```
1   LB ← (2D − D(f^1))/μ;
2   if C(f) ≤ LB then return f;
3   OPT2 ← max{LB, C(f^0)/2, C(f^1)/(2 log k + 2)};
4   while D(f^1) > 2D(1 + 1/k) do
5       Construct the residual network G(f^1):
6       Add to G(f^1) each link l in G with f^1_l = 0;
7       for each link (u, v) ∈ G with f^1_(u,v) = 1 do
8           Add (v, u) with d_(v,u) = −d_(u,v) & c_(v,u) = 0;
9       W ← FCYCLE(G(f^1), f^1, OPT2, (2D − D(f^1))/OPT2);
10      while W = ∅ do
11          OPT2 ← 2 * OPT2;
12          W ← FCYCLE(G(f^1), f^1, OPT2, (2D − D(f^1))/OPT2);
13      Augment flow f^1 along W;
14      OPT2 ← OPT2/2 ;
15  return f^1 ;
```

Algorithm 4. FCYCLE(G, f^2, OPT_2, μ)
Input:
G: the residual graph; f^2 *: the flow under check;*
OPT_2 *: the lower bound of* OPT*;*
μ *: the delay-to-cost ratio;*
Output: W*, a negative cycle* W *with* $\frac{D(W)}{C(W)} < \mu$ *or* $\emptyset$*.*

```
1   for each link l ∈ G do
2       d_l = d_l − c_l * μ;
3   for each node g ∈ f^2 do
4       W ← SCYCLE(G, g, OPT2, OPT2);
5       if W ≠ ∅ then return W;
6   return ∅ ;
```

Algorithm 5. SCYCLE(G, g, L, U)
Input:
G: the residual graph G=(V,E) with $\{d_l, c_l\}_{l\in E}$*;*
g *: the node under check;*
L,U: the lower bound and upper bound of the cycle;
Output: W*, a negative cycle* W *in* $G(f)$ *or* $\emptyset$*.*

```
1   S ← L/(n+1);
2   for each l ∈ E do
3       define c̃_l ≡ ⌊c_l/S⌋ + 1;
4   Ũ ← ⌊U/S⌋ + n + 1;
5   for all v ≠ g do
6       D(v,0) ← ∞;
7   D(g,0) ← 0;
8   for i = 1,2,...,Ũ do
9       for v ∈ V do
10          D(v,i) ← D(v,i−1);
11          for l ∈ {(u,v) | c̃_(u,v) ≤ i} do
12            D(v,i) ← min{D(v,i), d_l + D(u,i−c̃_l)};
13      if D(g,i) ≤ −1 then
14          return the cycle W in the path;
15  if there exist a negative cycle W in other paths
16      then    return W;
17  return ∅ .
```

Theorem 1 : If there exists a negative delay cycle W with $C(W) \leq U$ and passes node $g \in V$, algorithm SCYCLE will find a feasible cycle W' with $C(W') \leq C(W) + L$. The time complexity of SCYCLE is $O(MNU/L)$.

Proof: In SCYCLE, the upper bound of cost for any cycle W is $C(W) \leq \widetilde{U}S \leq U + (n+1)S = U + L$. If there is a negative cycle W and suppose it is the minimum delay cycle at the cost $C(W)$. For all such negative delay cycles, let's consider about the one with the minimum cost $C(W_{min})$. Since W_{min} is such a minimum negative delay cycle, it should also be the minimum delay path from g to itself at a cost of no more than $C(W_{min})$. Then within time $C(W_{min}) + nS \leq C(W_{min}) + L$, $D(g, C(W_{min}))$ will be no more than $D(W_{min})$. Since the delay d_l of a link l in G is a positive integer, $D(W) \leq -1$ and lines 13, 14 in SCYCLE will find a negative cycle W' with $C(W') \leq C(W_{min}) + L \leq C(W) + L$.

For the computational complexity, since $\widetilde{U} = \lfloor U/S \rfloor + n + 1 = O(NU/L)$ and for each $1 \leq i \leq \widetilde{U}$ each link is examined at most once, so the time complexity of it is $O(MNU/L)$. □

Furthermore, SCYCLE may identify a negative cycle when that cycle is not so far away from g on cost.

Theorem 2 : If there exists a negative delay cycle W with $C(W) \leq OPT_2$ and $\frac{D(W)}{C(W)} < \mu$ in G, algorithm FCYCLE will find it and return it back. The time complexity of FCYCLE is $O(MN^2)$.

Proof: In FCYCLE, only edges in flow f^2 may have a negative delay value. So any negative cycle must pass at least one edge and two nodes in f^2. We use algorithm SCYCLE to check all nodes $g \in f^2$, thus we will find a negative cycle with $C(W) \leq OPT_2$ if there exists one.

Now let's check the delay-to-cost ratio of this cycle W. After the modification of the delay of each link, we have $d_l = d_l - c_l * \mu$. Since W is a negative cycle, $\sum_{l\in W}(d_l - c_l * \mu) < 0$. Alternatively, $\sum_{l\in W} d_l - \sum_{l\in W} c_l * \mu = D(W) - C(W) * \mu < 0$, so $\frac{D(W)}{C(W)} < \mu$.

Since $U = L = OPT_2$ when we call SCYCLE, its time complexity should be $O(MNU/L) = O(MN)$, and algorithm SCYCLE will be called at most N times. It is evident that the time complexity of FCYCLE is $O(MN^2)$. □

Theorem 3 : Let f be an (s,t)-flow in G such that $f = 2$ and $D(f) \geq 2D$, and let $G(f)$ be the residual network of G imposed by f. Then there exists a circulation $\overline{f}$ in $G(f)$ such that $D(\overline{f}) \leq 2D - D(f)$ and $C(\overline{f}) \leq OPT$. In this circulation there is a cycle W with $\frac{D(W)}{C(W)} \leq \frac{2D-D(f)}{OPT}$.

Proof: See *Lemma 2* and *Corollary 1* in [12].□

Theorem 4 : Suppose that an algorithm A iteratively minimizes some value z such that z^0 is the initial value of z, z^i is the value of z at the i-th iteration and z^* is the minimum objective function value. Furthermore, suppose that the algorithm A guarantees that, for every iteration i, $z^i - z^{i+1} > \zeta(z^i - z^*)$ for some constant ζ with $0 < \zeta < 1$. Then, within $2x/\zeta$ consecutive iterations $z \leq z^* + \frac{z^0 - z^*}{2^x}$ and within $\frac{2\log(z^0 - z^*)}{\zeta}$ iterations algorithm A terminates.

Proof: This can be easily followed from [8] (page 67).□

Theorem 5 : Algorithm MINDELAY will return a flow f with delay $D(f) \leq 2D(1 + \frac{1}{k})$ and cost $C(f) \leq (4\log k + 3.5)OPT$. The time complexity of MINDELAY is $O(MN^2 OPT \log k)$.

Proof: Algorithm MINDELAY includes two phases. The first phase is from line 2 to line 9, which adopts the same method in [12] to find a minimum delay-to-cost ratio negative cycle and augment the flow along it. The second phase is from line 11 to line 12, which uses algorithm FINALIMPROVE to find a negative cycle with guaranteed cost and delay-to-cost ratio. We suppose that the first phase finds n_1 cycles and the second phase finds n_2 cycles. We denote by W_i the i-th cycle which will be applied to f and we use f_i to denote the state of flow f before f was augmented along W_i.

For each cycle W_i in the first phase, we have that $C(W_i) \geq 1$ and $\frac{D(W_i)}{C(W_i)} \leq \frac{2D-D(f_i)}{OPT}$. Which implies that $D(f_i) - D(f_{i+1}) = W(i) \geq -D(W_i) \geq \frac{D(f_i)-2D}{OPT}$. According to *Theorem 4*, within $\frac{2\log k}{\zeta} = \frac{2\log k}{1/OPT} = 2\log kOPT$ augment steps, it holds that $D(f) \leq 2D + \frac{D(f^0)-2D}{k} \leq 2D(1 + \frac{1}{k})$.

If $C(W_i) = h > 1$, we can replace W_i by h virtual unit cost cycle $W_i^0, W_i^2, ..., W_i^{h-1}$ with $D(W_i^x) = \frac{D(W_i)}{C(W_i)}$ and

$C(W_i^x) = 1$ for $0 \le x \le h-1$. For flow f_i^x, it holds that $D(f_i^x) = D(f_i^0) + x\frac{D(W_i)}{C(W_i)} < D(f_i^0) = D(f_i)$, so $D(W_i^x) = \mu(W_i^x) = \frac{D(W_i)}{C(W_i)} \le \frac{2D-D(f_i)}{OPT} \le \frac{2D-D(f_i^x)}{OPT}$. Thus these virtual unit cycles satisfy the improvement requirement of *Theorem 4* and the original n_1 cycles can be replaced by $\sum_{i=1}^{n_1} C(W_i)$ virtual unit cycles. Since within $2\log kOPT$ augment steps $D(f) \le 2D(1+\frac{1}{k})$, it holds that $\sum_{i=1}^{n_1} C(W_i) < 2\log kOPT$.

Since there will be no more than $n_1 \le 2\log kOPT$ iterations in the first phase and the algorithm for finding the minimum delay-to-cost ratio negative cycle dominates the time complexity of each iteration. If we choose the *binary search algorithm* in [8] (page 152) whose complexity is $O(MN\log(CD))$, then the time complexity of the first phase will be $O(MN \cdot OPT \log k \log(CD))$; but if we choose the *Burns algorithm* in [6, 9] whose complexity is $O(N^2M)$, then the time complexity of the first phase will be $O(MN^2OPT\log k)$. Here we use the latter one.

For each cycle W_j in the second phase, the following *Lemma1* proves that $1 \le C(W_j) \le 2OPT$ and $\frac{D(W_j)}{C(W_j)} \le \frac{2D-D(f_j)}{2OPT}$ for $n_1+1 \le j \le n_1+n_2$. So the analysis will be the same as in the first phase except that $\zeta = \frac{1}{2OPT}$. Combine all cycles $W_0, ..., W_{n_1}, W_{n_1+1}, ..., W_{n_1+n_2}$ together, we can find that $\sum_{i=1}^{n_1+n_2-1} C(W_i) < 4\log kOPT$. Since the cost of the last cycle will be no more than $2OPT$, it holds that $\sum_{i=1}^{n_1+n_2} C(W_i) < (4\log k+2)OPT$. Thus the total cost of flow f returned by algorithm MINDELAY is at most $(4\log k+2)OPT + C(f^0) \le (4\log k+2+1.5)OPT$.

Because the number of cycles identified in the second phase will be no more $O(\log kOPT)$, the time complexity is $O(MN^2OPT\log^2 k)$. But if we check more carefully, we can find that the average number of calls to FCYCLE for each cycle in the second phase will be around 2. This is because the value of OPT_2 in line 9 and line 12 of algorithm FINALIMPROVE will change for no more than $2\log\frac{2kOPT}{OPT/(2+2logk)}$ times. Thus a more precise bound for the time complexity is $O(MN^2OPT\log k)$. Combine the two phases together, the total complexity of algorithm MINDELAY is $O(MN^2OPT\log k) + O(MN^2OPT\log k) = O(MN^2OPT\log k)$. □

Lemma 1: Let W be a cycle to be augmented in line 13 in algorithm FINALIMPROVE, then $1 \le C(W) \le 2OPT$ and $\frac{D(W)}{C(W)} \le \frac{2D-D(f_i)}{2OPT}$. The time complexity to identify a cycle is $O(MN^2\log k)$.

Proof: In the first three lines we set a lower bound for the cost of the optimum solution. At first, since μ is the current minimum delay-to-cost ratio for any negative cycle in $G(f)$, it holds that $\mu \le \frac{2D-D(f)}{OPT}$ and $OPT \ge \frac{2D-D(f)}{\mu}$. Second, since algorithm 2DP-1 will return a flow f with $C(f) \le 1.5(1+\varepsilon)OPT$, so $C(f^0)/2$ is another loose lower bound. The third bound $\frac{C(f^1)}{2\log k+2}$ comes from the analysis of the cost in the first phase.

By *Theorem 3*, there exists a cycle W which satisfies $\frac{D(W)}{C(W)} \le \frac{2D-D(f^1)}{OPT}$ and $C(W) \le OPT$. So if algorithm FCYCLE cannot find a negative cycle with $C(W) \le OPT_2$ and $\frac{D(W)}{C(W)} \le \frac{2D-D(f^1)}{OPT_2}$, then it must hold that $OPT_2 < OPT$. Because if $OPT_2 \ge OPT$, then $\frac{2D-D(f^1)}{OPT_2} \ge \frac{2D-D(f^1)}{OPT}$ and according to *Theorem 2*, FCYCLE will return a negative cycle back. On the other hand, if algorithm FCYCLE finds a negative cycle with $C(W) \le OPT_2$, then by *Theorem 1* we have $C(W) \le U + L = 2OPT_2$. Since in the previous iteration $OPT_2 < OPT$, it follows that $C(W) \le 2OPT$.

Notice that when FINALIMPROVE is called, the first phase has already found a cycle with cost no more than $(k+1)OPT$ and $C(f) \le 2kOPT$ [12]. So the *while* cycle at line 10 will run no more than $O(\log\frac{2kOPT}{OPT/(2+2logk)}) = O(\log k)$ iterations before it finds a cycle. And the time complexity for each cycle is $O(MN^2\log k)$. □

Theorem 6: Algorithm 2DisjointPaths-1 computes, in $O(MN^2OPT\log k)$ time, a $(1+\frac{1}{k}, 4\log k+3.5)$ approximate solution for Problem 2DP.

Proof: The delay ratio $1+\frac{1}{k}$ follows from the above analysis. Since the complexity of algorithm MINDELAY is $O(MN^2OPT\log k)$ while the complexity of algorithm 2DP-1 is $O(MN(\log\log N + 1/\varepsilon))$, it holds that the time complexity of algorithm 2DisjointPaths-1 is $O(MN^2OPT\log k)$.□

4 Improve the Time Complexity

The above algorithm 2DisjointPaths-1 is a pseudo-polynomial algorithm since its time complexity $O(MN^2OPT\log k)$ is proportional to the cost OPT of the optimal solution. So it will not be very efficient when OPT is very large. But a good news for a pseudo-polynomial algorithm is that usually we may adopt some *cost scaling* approach to reduce the time complexity and turn it into a polynomial time approximate solution. We can see such kind of technique in many approximation algorithms [7, 10, 11], but here we can employ the method used in [12] directly since the basic framework is the same.

Since the difficulty lies in the possible huge value of OPT, which comes from the number of cycles to be found and augmented. If we can scale it down to a reasonable polynomial bound, then the time complexity of the new algorithm will be polynomial. This is the basic idea of algorithm 2DP-3 in [12] and our following algorithm 2DisjointPaths-2.

Algorithm 6. 2DisjointPaths-2(G, s, t, D, k)

1 $(P_1, P_2) \leftarrow$ *2DP-1*($G, s, t, D, \frac{1}{N}$);

2 $f^0 \leftarrow \{P_1, P_2\}$;

3 **if** $D(f^0) \leq 2D(1+\frac{1}{k})$ **then**
4 **return** (P_1, P_2) ;
5 $L, U \leftarrow$ *Bound*(G, s, t, f, D);
6 $\Delta \leftarrow \frac{L\varepsilon}{2N}$;
7 **for** *each link* $l \in E$ **do**
8 $c_l \leftarrow \lfloor \frac{c_l}{\Delta} \rfloor + 1$;
9 $f \leftarrow$ *MINDELAY*(G, f^0, D, k)
10 *Decompose* f *into* $\widehat{P}_1, \widehat{P}_2$ *with* $D(\widehat{P}_1) \leq D(\widehat{P}_2)$;
11 **return** $(\widehat{P}_1, \widehat{P}_2)$;

Algorithm 2DisjointPaths-2 calls the procedure BOUND to calculate the lower bound L and the upper bound U on OPT with $U/L \leq 2N$, its time complexity is $O((M + N\log N)\log N)$. For the details of the procedure BOUND, please refer to [10, 12].

Theorem 7: Algorithm 2DisjointPaths-2 computes, in $O(MN^4 \log k/\varepsilon)$ time, a $(1+\frac{1}{k}, (4\log k + 3.5)(1+\varepsilon))$-approximation solution for Problem 2DP.

Proof: Let OPT be the cost of the original optimal solution f and let OPT' be the optimal solution f' in the scaled network. Then the scaled cost of f will be $C'(f) = \sum_{e_i \in f} \lfloor (\frac{c_{e_i}}{\Delta} \rfloor + 1) \leq \sum_{e_i \in f} \lfloor \frac{c_{e_i}}{\Delta} \rfloor + 2N \leq \sum_{e_i \in f} \frac{c_{e_i}}{\Delta} + 2N = \frac{\sum_{e_i \in f} c_{e_i}}{\Delta} + 2N = \frac{OPT}{\Delta} + 2N = \frac{OPT \cdot 2N}{L\varepsilon} + 2N$. Since $U/L \leq 2N$ and $L \leq OPT \leq U$, we have that $1 \leq \frac{OPT}{L} \leq 2N$. Thus $C'(f) = \frac{OPT \cdot 2N}{L\varepsilon} + 2N \leq 2N + 4N^2/\varepsilon$. So for the optimal solution f' in the scaled network, we have that $OPT' \leq 2N + 4N^2/\varepsilon$. Thus the time complexity of algorithm 2DisjointPaths-2 is $O((M + N\log N)\log N) + O(MN^2 \log k \cdot N^2/\varepsilon) = O(MN^4 \log k/\varepsilon)$.

Now let's consider the original cost of the returned flow OPT'. Since $OPT' \leq \frac{OPT}{\Delta} + 2N$, its original cost should be no more than $OPT' * \Delta \leq (\frac{OPT}{\Delta} + 2N)\Delta = OPT + L\varepsilon \leq (1+\varepsilon)OPT$. So the delay of the final flow will be no more than $(4\log k + 3.5)(1+\varepsilon)OPT$. □

5 Conclusion

The major contribution of this paper are two approximation algorithms for finding two Delay-Restricted Link Disjoint Paths with minimum total cost. For any fixed $\varepsilon > 0$ and $k > 0$, the previous best result can find a solution that violates the delay constraint by factors of at most $1 + 1/k$ and $2(1+1/k)$ for the primary and restoration paths respectively with cost $k(1+\gamma)(1+\varepsilon)$ times more than the optimum. Our first algorithm reduces the cost to $4\log k + 3.5$ times of the optimum. Our second algorithm reduces the time complexity to $O(MN^4 \log k/\varepsilon)$ at the penalty of a larger cost bound as $(4\log k + 3.5)(1+\varepsilon)$ times of the optimum. Furthermore, we introduce a new technique to find a negative cycle with bounded cost and delay-to-cost ratio, which can be applied to other similar problems.

References

[1] E.W. Dijkstra, "A Note on Two Problems in Connexion with Graphs". *Numer. Math.*, 1:269-271,1959.

[2] J.W. Suurballe, "Disjoint Paths in a Network," *Networks*, vol. 4, pp. 125-145, 1974.

[3] Michael R. Garey , David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, NY, 1979.

[4] J.W. Suurballe and R. Tarjan, "A Quick Method for Finding Shortest Pairs of Disjoint Paths," *Networks*, vol. 14, pp. 325-336, 1984.

[5] C.L. Li, T. McCormick, and D. Simchi-Levi, "The Complexity of Finding Two Disjoint Paths with Min-Max Objective Function," *Discrete Applied Mathematics*, vol. 26, pp. 105-115, 1990.

[6] S.M. Burns, "Performance Analysis and Optimization of Asynchronous Circuits," *PhD Thesis*, California Institute of Technology, 1991.

[7] R. Hassin, "Approximation Schemes for the Restricted Shortest Path Problem," *Mathematics of Operations Research*, vol. 17, no. 1, pp. 36-42, February 1992.

[8] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Networks Flows*, Prentice-Hall, NJ, USA, 1993.

[9] Ali Dasdan, Sandy Irani, Rajesh K. Gupta. "Efficient Algorithms for Optimum Cycle Mean and Optimum Cost to Time Ratio Problems," DAC 1999: 37-42.

[10] D.H. Lorenz and D. Raz, "A Simple Efficient Approximation Scheme for the Restricted Shortest Path Problem," *Operations Research Letters*, vol. 28, no. 5, pp. 213-219, June 2001.

[11] F. Ergun, R. Sinha, and L. Zhang, "An Improved FPTAS for Restricted Shortest Path," *Information Processing Letters*, vol. 83, no. 5, 237-293, 2002.

[12] A. Orda and A. Sprintson. "Efficient Algorithms for Computing Disjoint QoS Paths," in *Proceedings of IEEE Infocom'2004*, Hongkong, March 2004.

[13] A. Orda and A. Sprintson. "Efficient Algorithms for Computing Disjoint QoS Paths," to appear in *IEEE/ACM Transactions on Networking*.