

PAPER

A New Approximation Algorithm For Computing 2-Restricted Disjoint Paths*

Chao PENG[†], *Student Member and* Hong SHEN^{††}, *Nonmember*

SUMMARY In this paper we study the problem of how to identify multiple disjoint paths that have the minimum total cost OPT and satisfy a delay bound D in a graph G . This problem has lots of applications in networking such as fault-tolerant quality of service (QoS) routing and network-flow load balancing. Recently, several approximation algorithms have been developed for this problem. Here, we propose a new approximation algorithm for it by using the Lagrangian Relaxation method. We then present a simple approximation algorithm for finding multiple link-disjoint paths that satisfy the delay constraints at a reasonable total cost. If the optimal solution under delay-bound D has a cost OPT , then our algorithm can find a solution whose delay is bounded by $(1 + \frac{1}{k})D$ and the cost is no more than $(1+k)OPT$. The time complexity of our algorithm is much better than the previous algorithms.

key words: *Approximation Algorithms, Fault-Tolerant Routing, Disjoint Paths, Lagrangian Relaxation*

1. Introduction

As networks modernize and expand with the increasing deployment of optical technology, the large bandwidth offered by optical fiber has brought tremendous potential. 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. Thus the survivability of a network has assumed great importance.

On the other hand, customers have come to expect the highest Quality of Service (QoS), including sustained continuity of service during the time they pay for the service. Such service includes multimedia streaming, video conference and other real-time broadcasting programs.

Survivability and QoS can be achieved by maintaining multiple disjoint QoS-constrained paths to in-

crease the probability that source can reach the destination via another path as the network undergoes topological changes, thus avoiding an unreasonable loss of service quality. Multiple QoS-constrained paths can also be employed to achieve better load balance and improve quality of service in bandwidth-constrained wireless networks.

Although many works have been done to find multiple node-disjoint or link-disjoint paths in a given network [?], [?,], the problem of finding two disjoint QoS-constrained paths has got little attention. Recently Orda and Sprintson [?], [?,] proposed four algorithms to compute two delay-constrained link-disjoint paths with minimum total cost. If there exist two disjoint paths in a network with total delay less than D and total cost OPT , their algorithm 2DP-1 can find two paths with total delay less than $\frac{3}{2}D$ and total cost $(1.5 + \varepsilon)OPT$. The other three algorithms proposed in [?,] can find two paths with total delay less than $(1 + \frac{1}{k})D$ and total cost $(k + 2 \log k + 1)(1 + \varepsilon)OPT$. Algorithm 2DP-1 has a complexity of $O(MN(\frac{1}{\varepsilon} + \log \log N))$, while the complexity of Algorithm 2DP-4 is $O(\frac{MN^2 k^2 \log k}{\varepsilon} \log(CD))$ (Here C is the maximum cost of a single edge in G). Based on their framework, authors of [?,] further improved the approximate ratio of the cost to $(4 \log k + 3.5)(1 + \varepsilon)$ while the delay factor is still $(1 + 1/k)$. But the complexity increases to $O(MN^4 \log k/\varepsilon)$.

In this paper we propose a new approximation algorithm for this problem which is simple and efficient. Our algorithm can find two disjoint paths with total delay less than $(1 + \frac{1}{k})D$ while the total cost is no more than the $(1+k)OPT$. Furthermore, either the delay or the cost of our solution will be better than that of the optimal solution. The complexity of our new algorithm is $O(M \log_{1+M/N} N \log \frac{(k+1)C(f_d)}{C(f_c)})$ (f_c and f_d are the link-disjoint path pairs with minimum total cost and minimum total delay respectively in graph G), which is much lower than the previous algorithms.

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 approximation algorithm to reduce the total cost and give a detail analysis of its performance and complexity. In Section 4, we present some other extensions of our method. Finally, we conclude the paper in Section 5.

Manuscript received March 16, 2006.

Manuscript revised August 3, 2006.

Final manuscript received October 28, 2006.

[†]The author is with the School of Information Science, Japan Advanced Institute of Science and Technology, 1-1 Asahidai, Nomi, Ishikawa, 923-1292 Japan.

^{††}The author is with the School of Computer Science, University of Adelaide, Adelaide, SA 5005, Australia.

*Supported by "Fostering Talent in Emergent Research Fields" program in Special Coordination Funds for promoting Science and Technology by Ministry of Education, Culture, Sports, Science and Technology. E-mail: p-chao@jaist.ac.jp.

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 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 [?,], 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 $P = (s = v_0, v_1, \dots, t = v_n)$, such that, for $0 \leq i \leq n - 1$, $(v_i, v_{i+1}) \in E$. Here, $n = |P|$ is the number of edges 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.

A fundamental problem in QoS routing is to identify a minimum cost path between a source s and a destination t that satisfies more than two additive constraints such as delay and cost. This can be formulated as a Restricted Shortest Path problem.

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

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

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

Problem RSP is NP-hard [?,], Problem 2DP is also NP-hard for it is an extension of Problem RSP. In addition, it was proved in [?,] 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 (α, β) -approximation.

Definition 1 ((α, β) -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) $D(P_1) + D(P_2) \leq \alpha D$;
- 2) the total cost of two paths is at most β times more than that of the optimal solution, i.e., $C(P_1) + C(P_2) \leq \beta OPT$.

In general, the path with minimum delay among P_1 and P_2 serves as a primary path. Thus, the primary and restoration paths violate the delay constraint by factors of at most $\alpha/2$ and α , respectively, i.e., $D(P_1) \leq \alpha \frac{D}{2}$ and $D(P_2) \leq \alpha D$.

Furthermore, Problem 2DP can be extend to the *minimum constrained flow* (MCF) problem which seeks a minimum cost (s, t) -flow f such that $|f| = 2$ and $D(f) \leq D$ where D is a given delay constraint.

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

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 the next section we will use MCF in the process of computing and we will use *OPT* to denote the cost of its optimal solution for convenience.

Although Problem RSP is intractable, there exist pseudo-polynomial solutions. This give rise to fully polynomial time approximation schemes (FP-TAS), whose computational complexity is reasonable [?,], [?,], [?,]. The most efficient scheme, presented in [?,], has a computational complexity of $O(MN/\varepsilon)$, and computes a path with delay of at most D and cost of at most $(1 + \varepsilon)$ times the optimum, but it requires that both the delay and the cost of each link must be positive. The algorithm in [?,] 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 all algorithms in [?], [?], [?,] may set the cost of a link as zero in the residual graph, the latter algorithm was adopted and referred as Algorithm RSP.

3. A $(1 + \frac{1}{k}, 1 + k)$ -Approximate Algorithm for 2DP by Lagrangian Relaxation

In this section we present our approximation algorithm, which achieves an approximation ratio of $(1 + \frac{1}{k}, 1 + k)$. Previous algorithms are all based on the RSP Algorithm. The 2DP-1 Algorithm in [?,] uses RSP to find

the first path and then to identify an augmenting path in the residual graph. Based on 2DP-1, other improved algorithms use the negative delay cancelling method to minimize the delay constraint.

In this paper we adopt a totally different approach. The basic idea of the algorithm is to use the Lagrangian Relaxation method to find a solution which is “nearby” the optimal solution.

For the MCF Problem, we aim to minimize the cost $C(f)$ of a flow f with $|f| = 2$ and $D(f) \leq D$. Instead of considering the delay and the cost respectively, we combine them together. For each link l , we attach a new value $w_l = c_l + \alpha * d_l$. Then for a flow f we have $W(f, \alpha) = C(f) + \alpha * D(f)$. Now we need only to run a polynomial algorithm for finding two link-disjoint paths with minimum total cost on w -weight. Given a value of α , we can find a corresponding optimal flow f^α in G with $W(f^\alpha) = C(f^\alpha) + \alpha * D(f^\alpha)$. Then we can check the delay and cost of this flow, and based on them we can adjust the value of α so that we can improve the delay and cost toward the right direction. The algorithm stops with a flow f which satisfies $D(f) \leq (1 + \frac{1}{k})D$ and $C(f) \leq (1 + k)OPT$.

There are several algorithms for finding the shortest pair of link-disjoint paths[?], [?], [?,]. The basic idea is to find the shortest path P_1 in the graph at first, then construct the residual graph (see the following definition) for this path and find the shortest path P_2 in the residual graph. Now combine the two paths and delete the overlapping edges, then we can decompose them into a shortest pair of link-disjoint paths.

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 $(u, v) \in G$ with $f(u, v) = 1$ and weight $w_{(u,v)}$, we add to $G(f)$ a reverse link (v, u) to $G(f)$ with weight $-w_{(u,v)}$.

The same method can be used for finding k disjoint paths. In [?,], Suurballe gave a successive shortest path algorithm to find k disjoint paths with minimum total cost by recursively finding shortest paths on residual graphs. By applying the Dijkstra’s algorithm with d -heap [?,], this algorithm achieves a complexity of $O(M \log_{1+M/N} N)$ if k is a constant. We will use $DDP(\alpha)$ to denote this algorithm running on G with a modified cost of $w_l = c_l + \alpha * d_l$ in the remain part of this paper.

Since our target flow should have a delay of no more than $(1 + \frac{1}{k})D$ and a cost which is less than or equal to the cost of $(1+k)$ times the optimal cost to the MCF Problem, we need to find a way to approach the “neighborhood” of a feasible solution. We know that if α is fixed, then we can find an optimal flow. So we will naturally come to the approach of testing different values of α and try to find a feasible solution.

Theorem 1: In graph $G = (V, E)$, let f_1 be the flow with the minimum total weight when we set $w_l = c_l + \alpha_1 * d_l$ and f_2 be the flow with the minimum total weight when we set $w_l = c_l + \alpha_2 * d_l$. If $0 < \alpha_1 \leq \alpha_2$, then $D(f_1) \geq D(f_2)$ and $C(f_1) \leq C(f_2)$.

Proof: Since f_1 is the flow with the minimum total weight when we set $w_l = c_l + \alpha_1 * d_l$, we have that $W(f_1, \alpha_1) = C(f_1) + \alpha_1 * D(f_1) \leq C(f_2) + \alpha_1 * D(f_2)$. Since f_2 is the flow with the minimum total weight when we set $w_l = c_l + \alpha_2 * d_l$, we have that $W(f_2, \alpha_2) = C(f_2) + \alpha_2 * D(f_2) \leq C(f_1) + \alpha_2 * D(f_1)$. Add the two inequations together and use the fact that $0 < \alpha_1 \leq \alpha_2$:

$$\begin{aligned} \alpha_1 * D(f_1) + \alpha_2 * D(f_2) &\leq \alpha_1 * D(f_2) + \alpha_2 * D(f_1) \Rightarrow \\ &D(f_2) \leq D(f_1); \\ C(f_1)/\alpha_1 + C(f_2)/\alpha_2 &\leq C(f_2)/\alpha_1 + C(f_1)/\alpha_2 \Rightarrow \\ &C(f_1) \leq C(f_2). \end{aligned}$$

□

For the MCF problem, we assume that there is at least one flow f^* with $D(f^*) \leq D$ and $C(f^*) = OPT$. To check whether there is such a flow or not, we can run Algorithm DDP to find the flow with minimum total delay and compare it with D . If all disjoint path pairs have a total delay of more than D , we simply return a message to show that there is no such solution. Otherwise we can make sure that we will find a solution whose performance is within a $(1 + \frac{1}{k}, 1 + k)$ factor of the optimal solution.

In the following Figure ??, x -axis is the cost of a flow and y -axis is the delay of a flow. For all disjoint path pairs with the same cost $C(f)$ and delay $D(f)$, we can use a point $(C(f), D(f))$ in the plane to denote them. Since the weight of these disjoint path pairs will remain the same for all different values of α , we can deem them as a single pair. To make it clear, we divide the first quadrant into 6 faces:

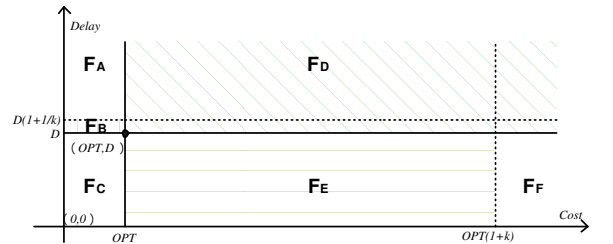


Fig. 1 The solution space for the MCF Problem, each point in the plane corresponds to a set of flows with the same cost and delay.

Point (OPT, D) is the unique joint point of face F_C and face F_D , there will be no other solution in face F_C by the optimality of (OPT, D) while all other solutions in face F_D are dominated by (OPT, D) because their cost is larger than OPT and their delay is larger than D . Thus for any value of α , Algorithm $DDP(\alpha)$ will never return a solution in face F_C and face F_D . On the

other hand, any solution in face F_B and face F_E is a feasible solution since it satisfies the $(1 + \frac{1}{k}, 1+k)$ bound, but the problem is that the corresponding disjoint path pair may be blocked by other disjoint path pairs in face F_A and face F_F when we are running $DDP(\alpha)$ with a given value of α . Fortunately, we are sure that we can always find a point in either face F_B or face F_E (or, the optimal solution) by the following theorem.

Theorem 2: In graph $G = (V, E)$, if there is an optimal flow f^* with $D(f^*) \leq D$ and $C(f^*) = OPT$, then there exists a flow f^α with minimum total weight when we set $w_l = c_l + \alpha * d_l$ and satisfies $D(f^\alpha) \leq (1 + \frac{1}{k})D$ and $C(f^\alpha) \leq (1 + k)OPT$.

Proof: Let f_c be the disjoint path pair with its total cost minimized and f_d be the disjoint path pair with minimum total delay, their corresponding points in the plane are $(C(f_c), D(f_c))$ and $(C(f_d), D(f_d))$ respectively. If we connect the points of all possible solutions returned by $DDP(\alpha)$ with different α (Figure ??), then it will form a line which connects $(C(f_c), D(f_c))$ and $(C(f_d), D(f_d))$ in the plane (Figure ??). There will be no other solution which is left to or below this line, since else it will be better than all solutions on the line for some α and cause a contradiction.

If there is no solution in face F_B and face F_E , then for any other solutions in face F_A and face F_F (we need not to consider face F_C and face F_D), either its delay is larger than $(1 + \frac{1}{k})D$ or its cost is larger than $(1 + k) * OPT$. Set $\alpha = \frac{k * OPT}{D}$, then $W(f^*, \alpha) = OPT + D * \frac{k * OPT}{D} = (1 + k)OPT$. While for a flow f in face F_A and face F_F , either $W(f, \alpha) = C(f) + D(f) * \alpha > C(f) + (1 + \frac{1}{k})D * \frac{k * OPT}{D} > (1 + k)OPT$ or $W(f, \alpha) = C(f) + D(f) * \alpha > (1 + k) * OPT + D * \frac{k * OPT}{D} > (1 + k)OPT$. So $W(f^*, \alpha)$ is the optimal solution for $\alpha = \frac{k * OPT}{D}$ and will be found by $DDP(\frac{k * OPT}{D})$.

If there are some other solutions in face F_B and face F_E , we can still set $\alpha = \frac{k * OPT}{D}$ and identify a near optimal solution. This is because we know that all solutions in face F_A and face F_F will be dominated by f^* when $\alpha = \frac{k * OPT}{D}$. So if $DDP(\frac{k * OPT}{D})$ returns back a solution, it is either f^* itself or another solution in face F_B or face F_E . \square

Notice that we set $\alpha = \frac{k * OPT}{D}$ in the above proof, this will be enough as an evidence of the capability of the Lagrangian Relaxation method for finding a feasible solution. But since OPT is not a given parameter, we cannot use it directly in our algorithm. Actually to calculate the exact value of OPT itself is intractable. So we have to find another practical criterion to mea-

F_A	$0 < Cost < OPT$	$\frac{k+1}{k}D < Delay$
F_B	$0 < Cost < OPT$	$D < Delay \leq \frac{k+1}{k}D$
F_C	$0 < Cost \leq OPT$	$0 < Delay \leq D$
F_D	$OPT \leq Cost$	$D \leq Delay$
F_E	$OPT < Cost \leq (1+k)OPT$	$0 < Delay < D$
F_F	$(1+k) * OPT < Cost$	$0 < Delay < D$

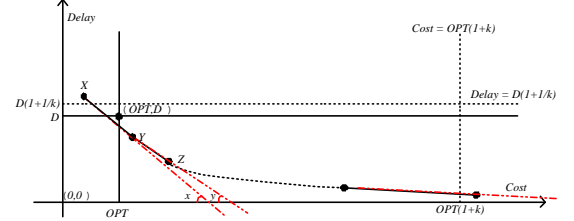


Fig. 2 If we connect all solutions that can be found by Algorithm DDP , then they will form a convex and piecewise-linear line.

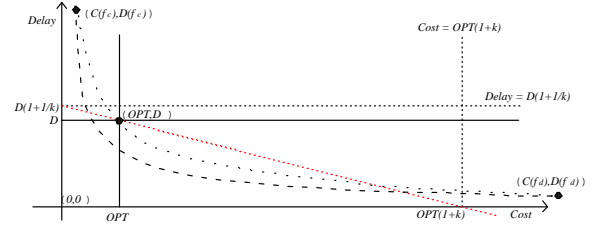


Fig. 3 The solution line will start from $(C(f_c), D(f_c))$ and end at $(C(f_d), D(f_d))$.

sure the performance of a solution returned back by $DDP(\alpha)$.

Since a feasible solution needs to satisfy the $(1 + \frac{1}{k})D$ bound for its delay, we may think of increasing the value of α if $D(f^\alpha) > (1 + \frac{1}{k})D$ and decreasing it when $D(f^\alpha) < D$ by adjusting the value of α . If we increase(decrease) the value of α by a certain amount, there will be an expected decrease(increase) of the total delay. And after enough rounds of iterations, we may fix the total delay to be no more than $(1 + \frac{1}{k})D$. But the problem here is that we still cannot figure out whether its cost is within $(1 + k) * OPT$ or not since we do not know the value of OPT . And although we can make sure $C(f^\alpha) < (1 + k)OPT$ if we have $D < D(f^\alpha) \leq (1 + \frac{1}{k})D$, there is a possibility that all such solutions will be blocked by the flows in face F_A and face F_E and thus can never be found by Algorithm DDP .

Let us reconsider the solution returned back when we set $\alpha = \frac{k * OPT}{D}$, its modified total cost will be $W(f, \alpha) = C(f) + D(f) * \frac{k * OPT}{D} \leq W(f^*, \alpha) = C(f^*) + D(f^*) * \frac{k * OPT}{D} = OPT * (1 + k) = D * \alpha * (1 + \frac{1}{k})$. This means we can compare the total cost of the solution returned back by $DDP(\alpha)$ with the value of $D * \alpha * (1 + \frac{1}{k})$ to see whether it is close enough to the optimal solution or not. The following theorem establishes a sufficient condition for finding a feasible solution which will satisfy both requirements.

Theorem 3: Let f be the flow returned by $DDP(\alpha)$ which satisfies $W(f, \alpha) = D * \alpha * (1 + \frac{1}{k})$, then $D(f) \leq (1 + \frac{1}{k})D$ and $C(f) \leq (1 + k)OPT$.

Proof: According to the assumption we have $C(f) + D(f) * \alpha = W(f, \alpha) = D * \alpha * (1 + \frac{1}{k})$. Thus $D(f) * \alpha \leq D * \alpha * (1 + \frac{1}{k}) \Rightarrow D(f) \leq D * (1 + \frac{1}{k})$. Let f^* be the optimal flow with $D(f^*) \leq D$ and $C(f^*) =$

OPT, then

$$\begin{aligned}
 D * \alpha * (1 + \frac{1}{k}) &= W(f, \alpha) \leq W(f^*, \alpha) \\
 &= C(f^*) + \alpha * D(f^*) \leq OPT + \alpha * D \\
 \Rightarrow D * \alpha * \frac{1}{k} &\leq OPT \Rightarrow D * \alpha \leq OPT * k \\
 \Rightarrow C(f) &\leq W(f, \alpha) \leq OPT + \alpha * D \leq (k+1) * OPT.
 \end{aligned}$$

We will further prove that we can find such an α by the following theorem.

Theorem 4: In graph $G = (V, E)$, if there is an optimal flow f^* with $D(f^*) \leq D$ and $C(f^*) = OPT$, then there exists an α_0 such that the flow f_0 returned back by $DDP(\alpha_0)$ satisfies $W(f_0, \alpha_0) = D * \alpha_0 * (1 + \frac{1}{k})$.

Proof: For any given link-disjoint path pair f in G , $W(f, \alpha) = C(f) + D(f) * \alpha$ is a linear function of α . So if we define $W'(f, \alpha) = W(f, \alpha) - D * \alpha * (1 + \frac{1}{k})$, then it is also a linear function of α .

Denote $U(\alpha) = W(DDP(\alpha), \alpha)$ as the modified weight of the flow returned back by $DDP(\alpha)$ and further define $U'(\alpha) = U(\alpha) - D * \alpha * (1 + \frac{1}{k})$. We can see that $U'(\alpha)$ is the minimum of $W'(f, \alpha)$ over all possible link-disjoint path pairs, thus it is piecewise-linear and continuous.

Given $\alpha_1 < \alpha_2$, let f_1 and f_2 be the flow returned back by $DDP(\alpha_1)$ and $DDP(\alpha_2)$ respectively, then we have $C(f_1) + D(f_1) * \alpha_1 \leq C(f_2) + D(f_2) * \alpha_1 < C(f_2) + D(f_2) * \alpha_2 \leq C(f_1) + D(f_1) * \alpha_2$. Thus

$$\begin{aligned}
 U'(\alpha_1) &= C(f_1) + D(f_1) * \alpha_1 - D * \alpha_1 * \frac{1+k}{k}; \\
 U'(\alpha_2) &= C(f_2) + D(f_2) * \alpha_2 - D * \alpha_2 * \frac{1+k}{k}; \\
 C(f_1) + D(f_1) * \alpha_1 &\leq C(f_2) + D(f_2) * \alpha_1 \\
 \Rightarrow C(f_1) - C(f_2) &\leq D(f_2) * \alpha_1 - D(f_1) * \alpha_1; \\
 C(f_2) + D(f_2) * \alpha_2 &\leq C(f_1) + D(f_1) * \alpha_2 \\
 \Rightarrow C(f_1) - C(f_2) &\geq D(f_2) * \alpha_2 - D(f_1) * \alpha_2.
 \end{aligned}$$

$$\begin{aligned}
 U'(\alpha_1) - U'(\alpha_2) &= C(f_1) + D(f_1) * \alpha_1 - D * \alpha_1 * \frac{1+k}{k} \\
 &\quad - C(f_2) - D(f_2) * \alpha_2 + D * \alpha_2 * \frac{1+k}{k} \\
 &\leq D(f_2) * \alpha_1 - D(f_1) * \alpha_1 + D(f_1) * \alpha_1 \\
 &\quad - D\alpha_1 \frac{1+k}{k} - D(f_2) * \alpha_2 + D\alpha_2 \frac{1+k}{k} \\
 &\leq D(f_2)\alpha_1 - D\alpha_1 \frac{1+k}{k} - D(f_2)\alpha_2 + D\alpha_2 \frac{1+k}{k} \\
 &\leq (\alpha_1 - \alpha_2)(D(f_2) - D \frac{1+k}{k}).
 \end{aligned}$$

$$\begin{aligned}
 U'(\alpha_1) - U'(\alpha_2) &= C(f_1) + D(f_1) * \alpha_1 - D * \alpha_1 * \frac{1+k}{k} \\
 &\quad - C(f_2) - D(f_2) * \alpha_2 + D * \alpha_2 * \frac{1+k}{k}
 \end{aligned}$$

$$\begin{aligned}
 &\geq D(f_2) * \alpha_2 - D(f_1) * \alpha_2 + D(f_1) * \alpha_1 \\
 &\quad - D\alpha_1 \frac{1+k}{k} - D(f_2) * \alpha_2 + D\alpha_2 \frac{1+k}{k} \\
 &\geq D(f_1)\alpha_1 - D\alpha_1 \frac{1+k}{k} - D(f_1)\alpha_2 + D\alpha_2 \frac{1+k}{k} \\
 &\geq (\alpha_2 - \alpha_1)(D \frac{1+k}{k} - D(f_1)).
 \end{aligned}$$

Since $\alpha_1 < \alpha_2$, we have that $D(f_1) > D(f_2)$ by Theorem 1. If $D(f_2) \geq \frac{1+k}{k}D$, then $(\alpha_1 - \alpha_2)(D(f_2) - D * \frac{1+k}{k}) < 0$. Thus $U'(\alpha)$ is monotone increasing when $D(f^\alpha) \geq \frac{1+k}{k}D$ and similarly it will be monotone decreasing when $D(f^\alpha) \leq D * \frac{1+k}{k}$ (see Figure ??b). As before, we use f_c and f_d to denote the disjoint path pair with minimum total cost and minimum total delay respectively. If we set $\alpha_{UB} = k \frac{C(f_d)}{D}$ and let f_{UB} be the corresponding flow, then:

$$\begin{aligned}
 U'(\alpha_{UB}) &= C(f_{UB}) + D(f_{UB})\alpha_{UB} - D\alpha_{UB} \frac{1+k}{k} \\
 &\leq C(f_d) + D(f_d)\alpha_{UB} - k \frac{1+k}{k} C(f_d) \\
 &\leq C(f_d) + Dk \frac{C(f_d)}{D} - k \frac{1+k}{k} C(f_d) = 0.
 \end{aligned}$$

Since $U'(0) = C(f_d) > 0$ and $U'(\alpha)$ is continuous over $[0, k \frac{C(f_d)}{D}]$, we are sure that there exists an α_0 which satisfies $U'(\alpha_0) = 0$ and $W(f_0, \alpha_0) = \alpha_0 \frac{k+1}{k} D$. \square

Based on the above analysis, we present the following algorithm.

Algorithm 1: Lag-2DP(G, s, t, D, k)

Input:

- G: the directed graph $G=(V,E)$;
- s : source node;
- t : destination node;
- D: the delay constraint;
- k : the approximation index;

Output:

- $\{P_1, P_2\}$: two link disjoint paths from s to t in G;

- 1 Compute the minimum total cost flow f_c and the minimum total delay flow f_d in G ;
- 2 **if** $D(f_d) > D$ **then return** "No Such Solution!";
- 3 $\alpha_{LB} \leftarrow 0$, $\alpha_{UB} \leftarrow k \frac{C(f_d)}{D}$;
- 4 $f_{LB} \leftarrow f_c$, $f_{UB} \leftarrow f_d$;
- 5 **while** $\alpha_{UB} - \alpha_{LB} \geq \frac{1}{D}$ **do**
- 6 $\alpha \leftarrow \frac{\alpha_{UB} + \alpha_{LB}}{2}$;
- 7 Set $w_l = c_l + \alpha * d_l$ for each link l ;
- 8 $f \leftarrow DDP(\alpha)$;
- 9 **if** $W(f, \alpha) - D * \alpha(1 + \frac{1}{k}) > 0$
- 10 **then** $\alpha_{LB} \leftarrow \alpha$, $f_{LB} \leftarrow f$;
- 11 **else** $\alpha_{UB} \leftarrow \alpha$, $f_{UB} \leftarrow f$;
- 12 **if** $D(f_{LB}) \leq D * (1 + \frac{1}{k})$ **then**
- 13 $f_{UB} \leftarrow f_{LB}$;
- 14 **Break** the "while" loop;
- 15 **if** $D(f_{LB}) = D(f_{UB})$ **then Break**;

- 15 Decompose flow f_{UB} into 2 link disjoint paths \tilde{P}_1 and \tilde{P}_2 with $D(\tilde{P}_1) \leq D(\tilde{P}_2)$;
- 16 return $\{\tilde{P}_1, \tilde{P}_2\}$;

Theorem 5: In graph $G = (V, E)$, if there is an optimal flow f^* with $D(f^*) = D$ and $C(f^*) = OPT$, then Algorithm Lag-2DP will return a flow f of two link-disjoint paths which satisfies $D(f) \leq (1 + \frac{1}{k})D$ and $C(f) \leq (1 + k)OPT$.

Proof: By Theorem 4 we know that there exists an α_0 such that the flow f_0 returned by $DDP(\alpha_0)$ satisfies $U'(\alpha_0) = 0$, which means that $D(f_0) \leq (1 + \frac{1}{k})D$ and $C(f_0) \leq (1 + k)OPT$ by Theorem 3. Now let us consider the following three different situations that bring Algorithm Lag-2DP to an end:

- 1): $D(f_{LB}) \leq D * (1 + \frac{1}{k})$. In this situation, the final solution $f_{UB} = f_{LB}$ (line 12). Since $U'(\alpha_{LB}) \geq 0$ while $U'(\alpha_0) = 0$, we have that $\alpha_{LB} \leq \alpha_0$. Thus $C(f_{LB}) \leq C(f_0) \leq (1 + k)OPT$ by Theorem 1.
- 2): $D(f_{LB}) = D(f_{UB})$. In this situation we have $f_{LB} = f_{UB} = f_0$, since $DDP(\alpha_{LB})$ and $DDP(\alpha_{UB})$ return the same flow and $\alpha_{LB} \leq \alpha_0 \leq \alpha_{UB}$. So f_{UB} satisfies $D(f_{UB}) \leq (1 + \frac{1}{k})D$ and $C(f_{UB}) \leq (1 + k)OPT$.
- 3): $\alpha_{UB} - \alpha_{LB} \leq \frac{1}{D}$. In this situation we can also prove that f_{UB} satisfies the requirement (Figure ??a).

$$\begin{aligned}
C(f_{UB}) + D(f_{UB}) * \alpha_{UB} &\leq D \frac{1+k}{k} * \alpha_{UB} \\
\Rightarrow D(f_{UB}) &\leq \frac{1+k}{k} D; \\
C(f_{LB}) + D(f_{LB}) * \alpha_{LB} &> D \frac{1+k}{k} * \alpha_{LB} \\
\Rightarrow W(f^*, \alpha_{LB}) &> D \frac{1+k}{k} * \alpha_{LB} \\
\Rightarrow W(f^*, \alpha_{LB}) &= OPT + D\alpha_{LB} > D \frac{1+k}{k} \alpha_{LB} \\
\Rightarrow k * OPT &> D * \alpha_{LB} \\
\Rightarrow C(f_{UB}) &< D \frac{1+k}{k} \alpha_{UB} < D \frac{1+k}{k} (\alpha_{LB} + \frac{1}{D}) \\
&= \frac{1+k}{k} (D\alpha_{LB} + 1) \leq k \frac{1+k}{k} OPT = (1+k)OPT.
\end{aligned}$$

Thus we have $C(f_{UB}) \leq (1+k)OPT$ and $D(f_{UB}) \leq (1 + \frac{1}{k})D$.

Since in all three situations Algorithm Lag-2DP will return a feasible solution, we conclude that this theorem is correct. \square

Notice that a solution f returned back by Algorithm Lag-2DP not only satisfies the $(1 + \frac{1}{k}, 1 + k)$ bound, but also has the following very nice property: $C(f) \geq OPT$ implies $D(f) < D$ and if $D(f) \geq D$, then $C(f) < OPT$. This means that either the delay or the cost of f will be better than that of the optimal solution (but of course not both).

α_1 α_2 α

Fig. 4 (a) The situation when f_{LB} and f_{UB} are close to the flow f^α which satisfies $U'(\alpha) = 0$. (b) Function $U'(\alpha)$ will be monotone increasing when $D(f^\alpha) \geq \frac{1+k}{k}D$ and monotone decreasing when $D(f^\alpha) \leq \frac{1+k}{k}D$.

Theorem 6: Let f_c and f_d be the link-disjoint path pairs with minimum total cost and minimum total delay respectively in graph $G = (V, E)$, Algorithm Lag-2DP will terminate in $O(M \log_{1+M/N} N \log \frac{C(f_d)(k+1)}{C(f_c)})$ time.

Proof: According to the proof in [?,], the complexity of Algorithm DDP for finding two link disjoint paths is $O(M \log_{1+M/N} N)$.

Since we set $\alpha_{LB} = 0$ and $\alpha_{UB} = k \frac{C(f_d)}{D}$, while the loop condition is $\alpha_{UB} - \alpha_{LB} \geq \frac{1}{2D}$, we instantly have that Algorithm Lag-2DP will end in $\log(kC(f_d))$ rounds and the time complexity is within $O(M \log_{1+M/N} N \log(kC(f_d)))$.

Now let's check line 11 in Algorithm Lag-2DP, we find that this step will decrease the complexity of the whole algorithm. By Theorem 4, we know that $U'(\alpha)$ is monotone increasing when $D(f^\alpha) \geq \frac{1+k}{k}D$ and monotone decreasing when $D(f^\alpha) \leq \frac{1+k}{k}D$ (see Figure ??b).

Let α_1 be the value when $U'(\alpha_1)$ is maximum and α_2 be the value when $U'(\alpha_2) = 0$. Then $D(f^{\alpha_1}) = \frac{1+k}{k}D$, and $D(f^{\alpha_{LB}}) \leq \frac{1+k}{k}D$ if $\alpha_{LB} \geq \alpha_1$. Since $\alpha_{LB} \leq \alpha_2 \leq \alpha_{UB}$, we will have $\alpha_{LB} \geq \alpha_1$ if $\alpha_{UB} - \alpha_{LB} < \alpha_2 - \alpha_1$. Thus $D(f^{\alpha_{LB}}) \leq \frac{1+k}{k}D$ and Algorithm Lag-2DP will be able to return a feasible solution by line 11. So we need to analyze the time complexity to make $\alpha_{UB} - \alpha_{LB} < \alpha_2 - \alpha_1$. But we do not know the exact value of α_1 and α_2 , fortunately we can estimate their difference via its relationship between $U'(\alpha_1)$. Let us first consider the difference between $U'(\alpha)$ and $U'(\alpha - 1)$:

$$\begin{aligned}
&U'(\alpha) - U'(\alpha - 1) \\
&= C(f^\alpha) + D(f^\alpha)\alpha - D * \alpha * \frac{1+k}{k} - C(f^{(\alpha-1)}) \\
&\quad - D(f^{(\alpha-1)})(\alpha - 1) + D * (\alpha - 1) * \frac{1+k}{k} \\
&= C(f^\alpha) + D(f^\alpha)\alpha - C(f^{(\alpha-1)}) - D(f^{(\alpha-1)})(\alpha - 1) - \frac{1+k}{k}D.
\end{aligned}$$

If $f^\alpha = f^{(\alpha-1)}$, then $U'(\alpha) - U'(\alpha - 1) = D(f^\alpha) - \frac{1+k}{k}D$. Else then $D(f^\alpha) < D(f^{(\alpha-1)})$ (by Theorem 1). Since $C(f^\alpha) + D(f^\alpha)\alpha < C(f^{(\alpha-1)}) - D(f^{(\alpha-1)})\alpha$, we

have $U'(\alpha) - U'(\alpha - 1) < D(f^{(\alpha-1)}) - \frac{1+k}{k}D$; while $C(f^{(\alpha-1)}) + D(f^{(\alpha-1)})(\alpha - 1) < C(f^\alpha) + D(f^\alpha)(\alpha - 1)$ implies that $D(f^\alpha) - \frac{1+k}{k}D < U'(\alpha) - U'(\alpha - 1)$. Now we have $U'(\alpha) - U'(\alpha - 1) < D(f^{(\alpha-1)}) - \frac{1+k}{k}D < U'(\alpha - 1) - U'(\alpha - 2)$, so the value of $U'(\alpha) - U'(\alpha - 1)$ will be monotone non-increasing when $\alpha < \alpha_1$ and the value of $U'(\alpha - 1) - U'(\alpha)$ will be monotone non-decreasing when $\alpha > \alpha_1$.

Now if we adjust α from α_1 to α_2 , then the value of $U'(\alpha)$ will fall down from its maximum to zero at a “speed” of no more than the speed around $U'(\alpha_2 - 1) - U'(\alpha_2) = -(U'(\alpha_2) - U'(\alpha_2 - 1)) \leq \frac{1+k}{k}D - D(f^{\alpha_2})$. This means that each time we increase α by 1, the value of $U'(\alpha)$ will decrease by less than $\frac{1+k}{k}D - D(f^{\alpha_2})$. So we have $(\alpha_2 - \alpha_1) * (\frac{1+k}{k}D - D(f^{\alpha_2})) \geq U'(\alpha_1) - 0 \Rightarrow \alpha_2 - \alpha_1 \geq \frac{U'(\alpha_1)}{\frac{1+k}{k}D - D(f^{\alpha_2})}$. Since originally $\alpha_{UB} - \alpha_{LB} = k\frac{C(f_d)}{D}$ and we can find a solution when $\alpha_{UB} - \alpha_{LB} < \frac{U'(\alpha_1)}{\frac{1+k}{k}D - D(f^{\alpha_2})} \leq \alpha_2 - \alpha_1$, Algorithm Lag-2DP will terminate in $\log\{k\frac{C(f_d)}{D}\frac{\frac{1+k}{k}D - D(f^{\alpha_2})}{U'(\alpha_1)}\}$ rounds (by binary search). Since $U'(\alpha_1) > C(f_c)$ while $\frac{k(\frac{1+k}{k}D - D(f^{\alpha_2}))}{D} = \frac{D*(1+k) - k*D(f^{\alpha_2})}{D} < \frac{D*(1+k)}{D} = (k + 1)$, we have $\log\{k\frac{C(f_d)}{D}\frac{\frac{1+k}{k}D - D(f^{\alpha_2})}{U'(\alpha_1)}\} < \log\frac{(k+1)C(f_d)}{C(f_c)} = \log\frac{C(f_d)(k+1)}{C(f_c)}$.

Thus we can conclude that Algorithm Lag-2DP will end in $O(M \log_{1+M/N} N \log \frac{C(f_d)(k+1)}{C(f_c)})$ time. \square

4. Some Extensions

The method developed in the previous chapter is very simple and efficient. What's more, we can apply the same method for solving other problems. A very natural extension is to solve the problem of finding t link-disjoint delay-restricted paths with shortest total cost. We need only to let Algorithm DDP find t paths instead of 2 in Algorithm Lag-2DP. Let this modified algorithm be Algorithm Lag-TDP, we instantly get the following theorem.

Theorem 7: In graph $G = (V, E)$, if there is an optimal flow f^* of t link-disjoint paths with $D(f^*) = D$ and $C(f^*) = OPT$, then Algorithm Lag-TDP will return a flow f of t link-disjoint paths which satisfies $D(f) \leq (1 + \frac{1}{k})D$ and $C(f) \leq (1 + k)OPT$. Let f_c and f_d be the t link-disjoint paths with minimum total cost and minimum total delay respectively in graph $G = (V, E)$, Algorithm Lag-TDP will terminate in $O(tM \log_{1+M/N} N \log \frac{(k+1)C(f_d)}{C(f_c)})$ time.

We can also use this method to compute the total-delay-restricted spanning tree with minimum total cost. The modification is to replace Algorithm DDP with Algorithm MST.

Algorithm 2: Lag-2MST(G, D, k)

Input:

G: the directed graph $G=(V,E)$;

D: the total delay constraint;

k : the approximation index;

Output:

T: a spanning tree of G which satisfies $D(T) \leq (1 + \frac{1}{k})D$ and $C(T) \leq (1 + k)OPT$;

- 1 Compute the MST T_c with minimum total cost and T_d with minimum total delay in G ;
- 2 **if** $D(T_d) > D$
 then return "No Such Solution!";
- 3 $\alpha_{LB} \leftarrow 0, \alpha_{UB} \leftarrow k\frac{C(T_d)}{D}$;
- 4 $T_{LB} \leftarrow T_c, T_{UB} \leftarrow T_d$;
- 5 **while** $\alpha_{UB} - \alpha_{LB} \geq \frac{1}{D}$ **do**
- 6 $\alpha \leftarrow \frac{\alpha_{UB} + \alpha_{LB}}{2}$;
- 7 Set $w_l = c_l + \alpha * d_l$ for each link l ;
- 8 $T \leftarrow \text{MST}(\alpha)$;
- 9 **if** $W(T, \alpha) - D * \alpha(1 + \frac{1}{k}) > 0$
 then $\alpha_{LB} \leftarrow \alpha, T_{LB} \leftarrow T$;
- 10 **else** $\alpha_{UB} \leftarrow \alpha, T_{UB} \leftarrow T$;
- 11 **if** $D(T_{LB}) \leq D * (1 + \frac{1}{k})$ **then**
- 12 $T_{UB} \leftarrow T_{LB}$;
- 13 Break the "while" loop;
- 14 **if** $D(T_{LB}) = D(T_{UB})$ **then** Break;
- 15 **return** T_{UB} ;

Theorem 8: In graph $G = (V, E)$, if there is a spanning tree T^* with $D(T^*) = D$ and $C(T^*) = OPT$, then Algorithm Lag-2MST will return a spanning tree T which satisfies $D(T) \leq (1 + \frac{1}{k})D$ and $C(T) \leq (1 + k)OPT$. Let T_c and T_d be the spanning trees with minimum total cost and minimum total delay respectively in graph $G = (V, E)$, Algorithm Lag-2MST will terminate in $O(tM \log \log^* N \log \frac{(k+1)C(T_d)}{C(T_c)})$ time.

Proof: We adopt the algorithm presented in [?,] as MST() to compute the minimum spanning tree, the complexity of this algorithm is $O(M \log \log^* N)$. Other proof is the same as that in Theorem 6. \square

5. Conclusion

The major contribution of this paper is a polynomial time approximation algorithm for finding two Delay-Restricted Link Disjoint Paths with minimum total cost. Our algorithm maintains the delay performance of no more than $(1 + \frac{1}{k})D$ and reduces the cost factor to $(k + 1)$ times the optimum, and either the delay or the cost of our solution will be better than that of the optimal solution.

	Cost	Time Complexity
[?,]	$(k + 2 \log k + 1)(1 + \varepsilon)$	$O(\frac{MN^2k^2 \log k}{\varepsilon} \log(CD))$
[?,]	$(4 \log k + 3.5)(1 + \varepsilon)$	$O(MN^4 \log k/\varepsilon)$
We	$1 + k$	$O(M \log_{1+\frac{M}{N}} N \log \frac{(k+1)C(f_d)}{C(f_c)})$

From the above table we can see that the complex-


ity of our new algorithm is much lower than previous algorithms. Our algorithm can be easily extended to find more than two edge-disjoint 2-restricted paths. What's more, we adopt the Lagrangian Relaxation method as a new technique to find 2-restricted link disjoint paths and trees, which can be applied to other problems with similar characterizations.

Acknowledgment


We thank the anonymous referees for their careful reading and many useful comments.

References

- [1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Networks Flows*, Prentice-Hall, 1993.
- [2] Ramesh Bhandari, *Survivable Networks - Algorithms for Diverse Routing*, Kluwer Academic Publishers, 1999.
- [3] E.W. Dijkstra, "A Note on Two Problems in Connexion with Graphs", *Numer. Math.*, vol. 1, pp. 269-271, 1959.
- [4] F. Ergun, R. Sinha, and L. Zhang, "An Improved FPTAS for Restricted Shortest Path", *Information Processing Letters*, vol. 83, no. 5, pp. 237-293, September 2002.
- [5] H. N. Gabow, Z. Galil, T. H. Spencer, and R. E. Tarjan, "Efficient algorithms for finding minimum spanning trees in undirected and directed graphs", *Combinatorica*, vol. 6(2), pp. 109-122, 1986.
- [6] 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.
- [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] 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.
- [9] A. Orda and A. Sprintson, "Efficient Algorithms for Computing Disjoint QoS Paths", in *Proceedings of IEEE Infocom'2004*, Hongkong, March 2004.
- [10] A. Orda and A. Sprintson, "Efficient Algorithms for Computing Disjoint QoS Paths", *IEEE/ACM Transactions on Networking*, vol. 13, no.3, pp. 648-661, 2005.
- [11] C. Peng and H. Shen, "An Improved Approximation Algorithm for Computing Disjoint QoS Paths", in *Proceedings of the 5th IEEE International Conference on Networking (ICN'06)*, Mauritius, April 2006.
- [12] J.W. Suurballe. "Disjoint Paths in a Network", *Networks*, vol. 4, pp. 125-145, 1974.
- [13] J.W. Suurballe and R. Tarjan. "A Quick Method for Finding Shortest Pairs of Disjoint Paths", *Networks*, vol. 14, pp. 325-336, 1984.



Chao Peng received his B.S. degree from East China Normal University in 1999 and M.S. diploma from Fudan University in 2002. He is currently working toward his Ph.D. degree in School of Information Science, Japan Advanced Institute of Science and Technology. His research interests include optimization algorithm design, game theory, mobile ad hoc networking and wireless sensor networks.



Hong Shen is Professor (Chair) of Computer Science in the School of Computer Science, University of Adelaide. He received his B.Eng. degree from Beijing University of Science and Technology, M.Eng. degree from University of Science and Technology of China, Ph.Lic. and Ph.D. degrees from Abo Akademi University, Finland, all in Computer Science. He has extensive academic experiences internationally including 9 years service at

Griffith University (Australia) from Lecturer to Professor and 5 years service at the Japan Advanced Institute of Science and Technology (JAIST) as Professor and Chair of the Computer Networks Laboratory. With main research interests in parallel and distributed computing, algorithms, high performance networks, data mining and multimedia systems, he has published more than 200 papers, including over 100 papers in major journals such as a variety of IEEE and ACM Transactions. He has been an editor/associate editor /editorial-board member for 7 international journals; chaired numerous international conferences; served on Program Committee for more than 100 international conferences. He was the co-recipient of the 1991 National Education Commission Science and Technology Progress Award and 1992 Chinese Academy of Sciences Natural Sciences Award.