# Optimization Algorithms for Fault-tolerant and QoS Routing in Networks

by

Chao PENG

submitted to
Japan Advanced Institute of Science and Technology
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

*Supervisor:* Professor Yasuo Tan

*School of Information Science*
*Japan Advanced Institute of Science and Technology*

September, 2006

# Abstract

In this dissertation, we investigate the optimization issues on four different topics related to fault-tolerant and QoS routing in networks. Research on fault-tolerant and QoS routing covers lots of topics from the real-world engineering problem to the computability and computation complexity analysis, here we will focus on the algorithm design and analysis aspect.

The first problem, which is known as the Delay-Constrained Shortest Disjoint $s$-$t$-Path Pair Problem, is to find two edge-disjoint paths in a graph $G$ with minimum total cost from a source node $s$ to a sink node $t$ which satisfies a given delay bound. This problem is $\mathcal{NP}$-hard, so we focus on designing approximation algorithms for it. We tackle this problem by two different methods. The first method is based on existing work, which finds two edge-disjoint paths by using a Fully Polynomial-Time Approximation Scheme (FPTAS) algorithm for the well-known restricted shortest path problem as a building block to find a suitable flow and then it reduces the delay at the tradeoff of cost increase. By this method we propose an algorithm which achieves the current best bicriteria approximation factor. The second method uses Lagrangian Relaxation. Applying this method, we simply modify the weight of each edge and then do binary search in the solution space. Our second algorithm can compute a reasonable solution at a much quicker speed than all previous algorithms.

The second problem is on the minimum-cost single-source unsplittable flow. We first focus our efforts on two cases: either with arbitrary capacities or with very large capacities. For a graph $G$ with arbitrary edge capacity, we present a new algorithm to compute a minimum-cost single-source unsplittable flow in polynomial-time. This algorithm has a performance guarantee same as the current best bound for this problem, while the time complexity is lower and the implementation is easier. After that we modify this algorithm to cope with the situation when the largest demand in $G$ is far less than the minimum capacity in it. Our last algorithm in this area is for the minimum-cost single-source $k-$splittable flow problem, it achieves the best performance guarantee so far.

The third problem is on computing the Inner-node Weighted Minimum Spanning Trees. Here we present a general framework which can find a multi-logarithm approximation algorithm for this problem. Based on this framework, we further propose two polynomial-time approximation algorithms which can achieve a better performance guarantee than all existing results.

Finally we study the energy-efficient broadcasting issue in mobile ad hoc networks. We present a distributed and localized algorithm for saving energy during a broadcast session and show by simulation data that it is energy efficient and scalable.

# Acknowledgments

I would like to express my deep sense of gratitude to my advisor Professor Hong Shen and Professor Yasuo Tan for their invaluable guidance, constructive advice and constant encouragement during this work. Professor Shen's deep intuition for theory and his eye for the detail, while never losing track of the big picture were really inspiring. Professor Tan's passion and dedication to his research and the whole laboratory have brought unmatched influence to me.

My sincere thanks to Professor Xavier Defago, the advisor of my sub-theme, for his fruitful suggestions and helpful comments on my sub-theme research. I would like to thank Professor Teruo Matsuzawa, Professor Mineo Kaneko and Professor Nakano Koji for taking time from their busy schedules to serve on my dissertation committee. I also would like to express my gratitude to Professor Yoichi Shinoda for acting as my supervisor in the final term.

Thanks to my master advisor Hong Zhu for introduing me to research and helping me develop a liking for it.

Acknowledgment is appreciatively extended to all members of the Network Laboratory during my PhD research. I thank Dr. Haibin Kan and Dr. Xiaohong Jiang for their kind help on both my research and my life in Jaist. It was also a great pleasure working with Dr. Gui Xie, Dr. Keqiu Li, Dr. Wenyu Qu, Dr. Hui Tian, Dr. Zonghua Zhang, Mr. Yingpeng Sang, Ms. Qing Gong, Mr. Haibo Zhang and Ms. Yawen Chen.

My special thanks to the Graduate Research Program of JAIST, which supports my research and life in JAIST for three years.

Finally, and most importantly, thanks to my family for the countless sacrifices they have made on my account, and for being very supportive with every decision I have made regarding my life. This thesis is dedicated to them.

# Contents

# Chapter 1

# Background and Motivations

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. Emerging multimedia-based applications require both traditional *Quality of Service* (QoS) guarantees (such as bandwidth and end-to-end delay) and routing reliability during transmission. 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. In today's highly competitive environment, customers have come to expect the highest QoS, including sustained continuity of service during the time they pay for the service. Such services include multimedia streaming, video conference and some other real-time broadcasting programs.

At the same time, the wireless ad hoc network technology 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. Additionally, node failures (e.g. due to power loss or mobility) are also common in ad-hoc networks.

In both situations the reliability and survivability of a network have assumed great importance. Reliability 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. Network design for reliability includes implementation of robust hardware and software components to lengthen the expected mean time to failure or mean time between failures. Similarly, the availability of a network refers to the probability that network services are accessible to users, upon request, given that the network may experience failures and subsequent repairs. Network design for availability includes implementation of repair models to shorten the mean time to repair. Often, reliability and availability are collectively studied as part of network dependability where a typical assumption is that network services are not available during failure periods. An overriding goal, therefore, is to prevent failures, and to facilitate a quick repair if they occur.

The survivability of a network refers to the degree to which a network maintains its functionality in the wake of failures. A typical assumption in a survivable network design is that network components will fail, and may remain in a failed state for an extended period of time. Hence, focus of a survivable network design is on placement of redundant or spare resources and implementation of strategies to dynamically make use of these backup resources, when needed, in order to maintain network operations in the presence of faults. The ideal goal of survivability is to make a network failure imperceptible to the user by providing service continuity and minimizing network congestion. Since cost is always an issue, the challenge is to provide an acceptable level of service for a set of failure scenarios in a cost effective manner.

As a result, network designers are beginning to incorporate provision of services over link/node-disjoint paths, so that if one path fails due to a link or node failure, the second path can carry the traffic to its destination and thus avoid any unexpected 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 MANETs.

Improving routing reliability by using precalculated alternate paths is one of the main consideration of our research. Calculating appropriate alternate paths is essential if alternate path routing is applied to achieve high reliability. We attempt to select multiple disjoint paths which meet the predefined constraints on reliability and other QoS demands such as bandwidth and end-to-end delay. Any overlapping of the alternate paths has a major impact on reliability. Hence, common links and nodes among the paths have to be limited.

Sometimes we need to find multiple disjoint paths between a pair of nodes with different bandwidth requirements. In this case, we cannot simply use the edge-disjoint algorithm anymore. Actually, this is a variant of the more generalized unsplittable flow problem. The unsplittable flow problem has lots of application, not only in fault-tolerant computing, but also in other QoS routing models.

When we are building an optical network, we need to calculate the cost of all fibres and intermediate optical switches so that the terminal users can connect their computers to the ports in a most economical way. This is an instance of the Inner-node Weighted Minimum Spanning Tree Problem, which is also an important topic in our research.

For QoS issues in mobile ad hoc networks, we will inevitably consider the energy-efficiency problem since the limited energy reserve of nodes in a MANET is definitely a bottleneck for the normal operation of the whole network. For this reason, we have also studied the energy-concerned broadcasting problem in MANETs.

Generally speaking, the work we have performed in this thesis can be classified into the four topics mentioned above. Since all these optimization problems are intractable, we focus on designing approximation algorithms or heuristic algorithms for them. In the following subsection we will describe the difficulties of these problems and our contributions.

## 1.1 Research Challenges and Our Contributions

- We first consider the problem of finding two Delay-Restricted Link Disjoint $s$-$t$-Paths with minimum total cost. This is a classical $\mathcal{NP}$-hard problem and has attracted considerable attentions recently. The difficulty of this problem lies in that we have to find *two multi*-constrained paths. As we know, there will usually be a tradeoff among the constraints when a problem is multi-constrained. For example, if we decrease the total delay of a path, we may have to face an increase of its cost. For the single path situation, we can design a Fully Polynomial-Time Approximation

Scheme (FPTAS) algorithm even if there are more than two additive constraints. This is because there exists a pseudo-polynomial-time algorithm for computing the exact-length single path. Yet the two edge-disjoint paths problem does not have such a property.

To deal with this problem, in the existing work researchers used the delay-restricted shortest path as a basic stone. Their method first computes two edge-disjoint delay-restricted shortest paths, then tries to decrease the delay by relaxing the cost. By this way they could find two paths with a total delay less than $2D(1+1/k)$ and total cost $k(1+\gamma)(1+\varepsilon)OPT$. We look inside this model and propose a new method for finding a cost-bounded negative-delay cycle. This method is based on the scaling technique and dynamic programming approach. By this method we can bound the cost tradeoff when we decrease the delay. Our algorithms can find two paths with total delay less than $2D(1+1/k)$ and reduce the total cost to $(2\log k + 3.5 + 1/\log k)(1+\varepsilon)OPT$ within $O(mn^4 \log k \log\log k/\varepsilon)$ time. Thus we can improve the approximation factor from $(1+1/k, O(k))$ to $(1+1/k, O(\log k))$, which largely decreases the cost while the delay bound remains near optimal.

A common problem in the above model is that the designed algorithms are not easy to implement for the complicated computing steps involved. Thus we carefully study the underlying mathematical structure of this problem and develop a totally new approach - Lagrangian Relaxation. By the Lagrangian Relaxation method, we need only modify the weight of each edge and then do binary search. This method looks easy and even naive, but we show its power by rigorous mathematical proof based on the solution space. Our algorithm is much more time-efficient and implementable than all previous algorithms for this problem. And we also propose some extensions of our method for solving other problems with similar underlying mathematical structures.

- The second part of our work is on the unsplittable flow problem, which is a natural generalization of the disjoint paths problem. This problem has a large family of variants and has attracted huge attentions in both theoretical computer science and operations research. The feasibility question for the unsplittable flow problem is strongly $\mathcal{NP}$-complete even without a budget constraint. Moreover, most researchers have put their efforts on the version with the assumption that the maximum demand is no more than the minimum capacity. This assumption makes the computation and analysis more convenient, but it does not hold on the current Internet.

  So we focus our efforts on the two extreme cases, either with small capacities or with very large capacities. Our first work is a new algorithm for compute a minimum-cost single-source unsplittable flow in a graph with arbitrary edge capacity. We introduce a novel rounding technique: instead of rounding the flow, we round the flow portion. By this method we need not to cancel flow and deal with the troublesome numeric issues. The performance guarantee of our algorithm is also the current best bound for this problem. For the situation when the largest demand in $G$ is far less than the minimum capacity in it, we can also use the similar technique and compute a flow with congestion depending on the relationship between the maximum demand and the minimum capacity. We also study the minimum-cost single-source $k-$splittable flow problem and present an algorithm with the best performance guarantee obtained so far.

- The third part of our work is on the problem of computing the Inner-node Weighted Minimum Spanning Trees. This is a rather new topic and is very useful in building high-speed fibre networks. Although this problem was proposed as a variant of the Minimum Spanning Tree problem, the real difficulty lies in that it is also a generalization of the connected dominating set problem (CDS) which is strongly $\mathcal{NP}$-hard.

  We present a general framework which can find a $\frac{k}{k-1}\ln n$-approximation Algorithm for this problem. Our framework uses a method by repeatedly finding the minimum $k$-structure in the remain graph and then contracting it. By this way we can decrease the size of the problem step-by-step until we obtain a feasible solution. Based on this framework, we further propose two polynomial-time approximation algorithms which can achieve a better performance guarantee than other results. We also consider the fault-tolerant issues in this problem.

- The last topic of this thesis is about the energy issue in mobile ad hoc networks and wireless sensor networks. The problem of computing a minimum-energy broadcasting tree in a MANET can be proved to be $\mathcal{NP}$-hard by reduction from the SET COVER problem. What's more, we cannot afford a very complicated algorithm in MANET since the energy capacity of a mobile node is very small. A routing protocol must also refrain itself from frequent communication since data sending and receiving are the most energy-consuming actions. All in all, a good algorithm for energy saving in MANET should be simple, distributed, localized and effective.

  Based on the above considerations, we present a distributed and localized algorithm for saving energy during a broadcast session. Different from the centralized approximation algorithms in previous chapters, this algorithm is a heuristic algorithm. Here we use a new localized method for computing the neighbor forwarding tree, which decreases the communication complexity and thus the energy consumption effectively. We compare the performance of our algorithm with some other popular protocols by using MATLAB and we find that our solution is energy efficient and scalable for large MANETs.

## 1.2 Organization of the Thesis

The remainder of this dissertation is organized as follows.

Chapter 2 provides a succinct introduction of the common terminology and some existing results that we are going to use in following chapters. We hope that by this chapter we can make the readers more comfortable for reading the following chapters. Readers who are familiar with theoretical computer science or algorithm designing can skip this chapter.

Chapter 3 addresses the problem of finding two Delay-Restricted Link Disjoint $s$-$t$-Paths with minimum total cost. We first show the difficulty of this problem and its relationship with the minimum cost flow problem. Then we present an approximation algorithm for this problem. Our algorithm is based on the previous work, but we improve it by decreasing the cost of the returned solution to a large extent. This improvement is obtained by our new technique for finding a cost-bounded negative-delay cycle. Based on our first algorithm, we present two improved versions to further decrease the time complexity and the solution cost. We also extend our results to the problem of computing more than two disjoint paths.

Chapter 4 presents a totally different method for finding two Delay-Restricted Link Disjoint $s$-$t$-Paths with minimum total cost. Instead of using the restricted shortest path as a basic stone, this algorithm uses the Lagrangian Relaxation method. We show how we can bypass the extremely complicated cycle cancelling process and find a good solution by slightly modifying the weight of each edge. Detailed proofs for the correctness and time-efficiency of our algorithm have been presented. We also propose some extensions of our method at the end of this chapter .

Chapter 5 extends our study from disjoint paths to the unsplittable flow problem. We briefly survey the research topic of Disjoint paths at the beginning, then we extend it to the single-source unsplittable flow problem. We develop an efficient algorithm for computing a minimum-cost single-source unsplittable flow in a graph with arbitrary edge capacity. Then we extend our algorithm to deal with some special cases such as the situation when the largest demand in $G$ is far less than the minimum capacity in it. After that we study the minimum-cost single-source $k-$splittable flow problem and present an algorithm with a better performance guarantee than the previous algorithms. Finally we present some applications of the single-source unsplittable flow problem on the huge-volume video transfer in large networks.

Chapter 6 aims to design a general framework which can find a $\frac{k}{k-1}\ln n$-approximation Algorithm for computing the Inner-node Weighted Minimum Spanning Trees. We describe this framework at the beginning of this chapter and prove its correctness based on some assumptions. Based on this framework, we further propose two polynomial-time approximation algorithms by validating the aforementioned assumptions. Later we further consider some fault-tolerant issues in this problem. We also show how we can use the methods for other related problems to solve the IWMST problem.

Chapter 7 is about energy efficient broadcasting in mobile ad hoc networks. Different from all previous chapters that deal with centralized approximation algorithms, this chapter study the distributed and localized algorithm for saving energy during a broadcast session. We show by simulation that our protocol is energy efficient and very flexible and scalable for large MANETS.

Chapter 8 gives a summary of this dissertation and discusses some future research directions.

# Chapter 2

# Preliminaries

A communication network is usually abstracted as a graph, either directed or undirected. By representing networks by graphs, we can investigate the properties of networks using graph theory and other theoretical computational tools. The main focus of this dissertation is the design and analysis of algorithms to solve various problems that are related to survivable and QoS concerned network communication. To avoid any confusion, in this chapter we will give a succinct introduction of the common terminology and some existing results that we are going to use in following chapters.

## 2.1 Common Definitions for Network Algorithms

A graph $G = (V, E)$ consists of a set $V$ of nodes (vertices) and a set $E$ of edges (links) connecting nodes in $V$. Nodes represent communicating equipments such as routers, switches, or computers in a network, while edges represent communication links connecting these equipments.

In a *directed graph* $G$, edges are ordered pairs of distinct nodes. We denote an edge $e$ from node $u$ to node $v$ by $u \to v$, where $u$ and $v$ are called the *tail* and *head* of the edges, respectively. If edges are bidirectional or non-directed, then we call $G$ an undirected graph. We use an unordered pair $\{u, v\}$ to denote an undirected edge $e$ connecting nodes $u$ and $v$, both $u$ and $v$ are called *endpoints* of the edge. We will also call the *tail* and the *head* of an edge its *endpoints*. Sometimes we will use $(u, v)$ to denote either an undirected edge $\{u, v\}$ or a directed edge $u \to v$. If the edges in a graph is capacitated, we will call it a *network*.

We use $n = |V|$ to denote the number of nodes and $m = |E|$ to denote the number of edges in graph $G = (V, E)$. Two edges are called *adjacent* if they share a common endpoint. The *degree* of a node is the number of edges which has this nodes as its endpoint.

A *path* $P = (s, (s, v_1), v_1, ..., v_{k-1}, (v_{k-1}, t), t)$ from a node $s$ to a node $t$ in a graph $G$ is an alternating tuple of nodes in $V$ and edges in $E$ starting from $s$ and ending in $t$ so that every node is adjacent to its neighboring edges. If all nodes in $P$ are pairwise distinct we will say $P$ is a *simple* or *loopless* path. The nodes $s$ and $t$ are *connected* if the graph contains at least one path $P$ from $s$ to $t$, we will call $s$ and $t$ the *end* of $P$ and call $P$ an $s$-$t$ path. The nodes $v_1, ..., v_{k-1}$ are the *inner* nodes of $P$ and the number of edges in $P$ is its *length*. An undirected graph is *connected* if every pair of nodes is connected.

Two paths are said to be *edge-disjoint* if they have no inner edge in common, while they are said to be *node-disjoint* if they have no intermediate node in common. Clearly, two node-disjoint paths are also edge-disjoint, but two edge-disjoint paths are not necessary node-disjoint.

A *cycle* is a simple path $(v_0, (v_0, v_1), v_1, ..., v_{k-1}, (v_{k-1}, v_k), v_k)$ together with the edge $(v_k, v_0)$. A simple path contains no cycle and thus can also be called an *acyclic* path. A directed graph which contains no cycle will be called a *DAG* (Directed Acyclic Graph).

A graph $G' = (V', E')$ is a *subgraph* of $G = (V, E)$ is $V' \subseteq V$ and $E' \subseteq E$. It is a *spanning subgraph* if $V' = V$ and $E' \subseteq E$. A *tree* is a connected graph that contains no cycle. A tree $T$ is a *spanning tree* of $G$ if $T$ is also a spanning subgraph of $G$.

A *cut* is a partition of the node set $V$ into two parts $S$ and $\bar{S} = V - S$. Each cut defines a set of edges consisting of those edges that have one endpoint in $S$ and the other in $S$. An *s-t-cut* is defined with respect to two distinguished nodes $s$ and $t$ and is a cut $[S, \bar{S}]$ satisfying the property that $s \in S$ and $t \in \bar{S}$.

## 2.2 Some Basic Network Problems

### 2.2.1 Shortest Paths

Given a weighted graph $G = (V, E)$ and cost $c(e) \in R^+$ for each $e \in E$, the *shortest path problem* (We abbreviate it by SP, throughout the thesis we will abbreviate other problems in the same style.) is to compute the minimum cost (or shortest) path from a node $s$ to another node $t$. Here we denote by $c_{uv}$ the cost of edge $(u, v)$. The cost of a path is the summation of the costs of each edge in this path.

According to the book by Ahuja, Magnanti and Orlin, there are two optimality properties for a shortest path in $G$.

**Property 1**: If the path $P = (s, (s, v_1), v_1, ..., v_{k-1}, (v_{k-1}, t), t)$ is a shortest path from node $s$ to node $t$, then for every $i = 1, 2, ..., k-1$, the subpath $(s, ..., v_i)$ in $P$ is a shortest path from node $s$ to node $v_i$.

**Property 2**: Let $d(j)$ denote the shortest path distance from node $s$ to node $j$. Then a directed path $P$ from $s$ to $t$ is a shortest path if and only if $d(j) = d(i) + c_{ij}$ for every edge $(i, j) \in P$.

There are two general approaches for solving the shortest path problem efficiently: label-setting and label-correcting methods.

The label-setting algorithms assign tentative distance labels to the nodes and then iteratively identify a true shortest path distance to one or more nodes at each step. A true distance figure will remain unchanged after it is settled, thus it is called a permanent label.

In 1959, Dijkstra [Dij59] gave his famous shortest path algorithm with complexity $O(n^2)$. It uses the label-setting method and its running time can be further reduced for sparse graphs if we use efficient data structures. By using $d$-heap, Johnson [Joh77] came up with an algorithm of $O(m \log_{1+m/n} n)$ time. By using Fibonacci heap, Fredman and Tarjan [FT84, FT87] presented an $O(m + n \log n)$-time implementation. While different implementations of Dijkstra's algorithms require all edge lengths to be nonnegative, nonnegative edge is not a necessary condition for the existence of a shortest path.

The label-correcting algorithms maintain a distance label with each node and iteratively update these labels until they satisfies the optimality condition. This optimality condition is a natural deduction of the above Property 2, it requires that $d(j) \leq d(i) + c_{ij}$ for every edge $(i, j) \in P$.

Bellman [Bel58] and Ford [FF62] proved that a shortest path exists if and only if there does not exist a cycle with negative total cost in $G$. They gave a shortest path algorithms (called the Bellman-Ford algorithm) with complexity $O(mn)$, which uses the label-correcting technique and allows negative length edges in $G$, it can also reports error when there exist negative cycles in $G$.

Although the shortest path problem is to find a shortest length path between two nodes, both Dijkstra's algorithm and the Bellman-Ford algorithm find shortest paths from $s$ to every other node in $G$. There is a related problem called all-pair shortest paths problem, which asks for the shortest paths between every two nodes in $G$. In graphs allowing negative edge lengths, Floyd [Floyd62] and Warshall [War62] gave an algorithm with complexity $O(n^3)$, which is more efficient than applying the Bellman-Ford algorithm on every node.

For more details, please refer to the excellent book [AMO93] by Ahuja, Magnanti and Orlin.

### 2.2.2 Minimum Spanning Tree

The *minimum spanning tree problem* (MST) in a graph $G = (V, E)$ with cost $c(e) \in R^+$ for each $e \in E$ is to find a spanning tree of minimal total cost in $G$. As for the shortest path case, optimality conditions play a central role in developing algorithms for the problem.

**Cut Optimality Conditions** : A spanning tree $T$ is a minimum spanning tree if and only if it satisfies the following cut optimality conditions: For every tree edge $(i, j) \in T$, we have $c_{ij} \leq c_{kl}$ for every edge $(k, l)$ that is contained in the cut formed by deleting edge $(i, j)$ from $T$.

**Path Optimality Conditions** : A spanning tree $T$ is a minimum spanning tree if and only if it satisfies the following path optimality conditions: For every non-tree edge $(k, l)$ of $G$, we have $c_{ij} \leq c_{kl}$ for every edge $(i, j)$ that is contained in the path in $T$ connecting nodes $k$ and $l$.

The following Prim's Algorithm [Prim57] is based on the cut optimality conditions: We build a spanning tree from scratch by fanning out from a single node and adding edges one at a time. It maintains a spanning tree on a subset $S$ of nodes and adds a nearest neighbor to $S$. It does so by identifying an edge $(i, j)$ of minimum cost in the cut $[S, \bar{S}]$. This algorithm can be implemented in $O(m + n \log n)$ time by using Fibonacci heaps.

Based on the path optimality conditions, Kruskal [Kru56] presented the following algorithm: We first sort all edges in non-decreasing cost order and define a set, LIST, that is the set of edges we have chosen as part of a minimum spanning tree. Initially, LIST is empty. We examine the edges in sorted order one by one and check whether adding the edge we are currently examing to LIST will create a cycle with the edges already in LIST. If it does not, we add the edge to LIST; otherwise we discard it. We terminate when the number of edges in LIST is $n - 1$. At termination, the edges in LIST constitute a minimum spanning tree. Kruskal's algorithm can be implemented in $O(m + n \log n)$ time if we use a $O(n \log n)$ time sorting algorithm.

### 2.2.3 Maximum Flows

Given a directed graph $G = (V, E)$, a source node $s \in V$, a sink node $t \in V$, and a non-negative capacity $u_{ij}$ associated with each edge $(i, j) \in E$. The Maximum Flow Problem is to find the maximum flow from the source node $s$ to the sink node $t$ that satisfies the edge capacities and mass balance constraints at all nodes. We can state the problem formally as follows:

Maximize $\quad v$
Subject to

$$\sum_{\{j:(i,j)\in E\}} x_{ij} - \sum_{\{j:(j,i)\in E\}} x_{ji} = \begin{cases} v & for\ i = s, \\ 0 & for\ all\ i \in V - \{s,\ t\}, \\ -v & for\ i = t. \end{cases}$$

$$0 \le x_{ij} \le u_{ij} \qquad for\ each\ (i,j) \in E.$$

The first family of constraints states that we have flow conservation at all nodes except the source node $s$ which has excess outflow $v$ and the sink node $t$ which has excess inflow $v$. The second family of constraints state that flows are non-negative and bounded by the edge capacities. The goal is to maximize the flow leaving source $s$. This problem has a dual problem: Minimum *s-t-cut*. This dual relationship have very important positions in network flow problems. Lots of researchers have designed algorithms for solving the Maximum Flow Problem. Among them the Highest-label preflow-push algorithms can be implemented in $O(n^2\sqrt{m})$ time; the FIFO preflow-push algorithm achieves a complexity of $O(n^3)$, this complexity can be improved to $O(mn\log(n^2/m))$ by using a dynamic tree data structure [AMO93] .

### 2.2.4 Minimum Cost Flows

Given a directed graph $G = (V, E)$, a source node $s \in V$, a sink node $t \in V$, a non-negative cost $c_{ij}$ and capacity $u_{ij}$ associated with each edge $(i, j) \in E$. The Minimum Cost Flow Problem is to find a minimum cost $s$-$t$-flow $f$ that satisfies the edge capacities, the flow demand $|f| = k$ and the mass balance constraints at all nodes. We can state the problem formally as follows:

Minimize $\quad z(x) = \sum_{(i,j)\in E} c_{ij}x_{ij}$
Subject to

$$\sum_{\{j:(i,j)\in E\}} x_{ij} - \sum_{\{j:(j,i)\in E\}} x_{ji} = \begin{cases} k & for\ i = s, \\ 0 & for\ all\ i \in V - \{s,\ t\}, \\ -k & for\ i = t. \end{cases}$$

$$0 \le x_{ij} \le u_{ij} \qquad for\ each\ (i,j) \in E.$$

In 1971, Ford and Fulkerson published their famous book *Flows in Networks* [FF62], in which they proved the Max-Flow Min-Cut theorem and presented the Ford-Fulkerson algorithm for finding a maximum flow from $s$ to $t$. In addition, they also solved the minimum cost flow problem, using the primal-dual algorithm. According the book by Ahuja et al, the fastest algorithm for the minimum cost flow problem is

$$O(min\{nm\log(n^2/m)\log(nL), nm(\log\log C)\log(nL), m\log n(m + n\log n)\}),$$

where $L$ is the maximum edge length, $C$ is the larger value of $k$ and the maximum finite edge capacity.

### 2.2.5 Shortest Disjoint Paths

Given a weighted graph $G = (V, E)$ with cost $c(e) \in R^+$ for each $e \in E$, the problem of *shortest disjoint paths* (SDP) is to compute the $k$ disjoint paths from a node $s$ to another node $t$ with minimum total cost of all paths. This problem is a special case of minimum cost flow problem with flow value equals to $k$ and every edge capacity equals to 1. In 1974, Suurballe [Su74] gave a successive shortest path algorithm to find $k$ shortest disjoint paths by recursively finding shortest paths on residual graphs. When $k$ is a constant, after applying the Dijkstra's algorithm with $d$-heap, this algorithm achieves the complexity of $O(m \log_{1+m/n} n)$.

## 2.3 A Brief Introduction to the Complexity Theory

Is there a way to develop a structural understanding of algorithms or for the problems to which we wish to apply these algorithms? This simple question opened a new research field known as computational complexity theory in early 1970s. This theory allows us to classify a problem into two broad classes: easy problems that can be solved by polynomial-time algorithms and hard problems that are not likely to be solved in polynomial-time and for which all known algorithms require exponential running time.

When studying the complexity theory, it is easier to restate a problem so that its solution has only two answers: *yes* or *no*. In this case, the problem is called a decision problem. In contrast, an optimization problem is a problem that is concerned with the minimization or maximization of a certain quantity. For example, the aforementioned shortest path problem is an optimization problem. Its decision version will be like this: does there exist an $s$-$t$-path with total cost of no more than $B$?

### 2.3.1 Major Complexity Classes

Throughout this thesis, we shall refer to three classes of problems: $\mathcal{P}$, $\mathcal{NP}$ and $\mathcal{NPC}$, the latter class being the $\mathcal{NP}$-complete problem. We say that a decision problem $Q$ belongs to the complexity class $\mathcal{P}$ if its *yes/no* solution can be obtained using a *deterministic* algorithm that runs in *polynomial* number of steps, i.e., in $O(n^k)$ steps, for some nonnegative number $k$, where $n$ is the input size.

The class $\mathcal{NP}$ consists of those problems for which there exist a *deterministic* algorithm which, when presented with a claimed solution to an instance of this problem, will be able to verify its correctness in *polynomial* time. That is, if the claimed solution leads to a *yes* answer, there is a way to *verify* this solution in *polynomial* time. Note that it is much easier to verify that, for example, a given assignment satisfies a boolean formula, than deciding that there is no satisfying assignment. So it is evident that $\mathcal{P} \subseteq \mathcal{NP}$.

The class $\mathcal{NPC}$ denotes the subclass of decision problems in $\mathcal{NP}$ that are hardest in the sense that if one of them is proven to be solvable by a polynomial-time deterministic algorithm, then $\mathcal{NP} = \mathcal{P}$.

Let $\Pi$ and $\Pi'$ be two decision problems. We say that $\Pi$ reduces to $\Pi'$ in polynomial-time, symbolized as $\Pi \propto_{poly} \Pi'$, if there exists a *deterministic* algorithm $A$ that behaves as follows. When $A$ is presented with an instance $I$ of problem $\Pi$, it transforms it into an instance $I'$ of problem $\Pi'$ such that the answer to $I$ is *yes* if and only if the answer to $I'$ is *yes*. Moreover, this transformation must be achieved in *polynomial* time.

A decision problem $\Pi$ is said to be $\mathcal{NP}$-hard if for every problem $\Pi' \in \mathcal{NP}$ we have $\Pi' \propto_{poly} \Pi$. $\Pi$ is said to be $\mathcal{NP}$-complete if $\Pi \in \mathcal{NP}$ and $\Pi$ is $\mathcal{NP}$-hard.

The problem SATISFIABILITY is the first problem shown to be $\mathcal{NP}$-complete by Cook in 1971, now there are lots of $\mathcal{NP}$-complete problems include the *Hamiltonian Cycle* problem and the *Longest Simple Path* problem. The book by Garey and Johnson provides comprehensive coverage of the computational complexity theory, it also has a rich compendium containing information on the $\mathcal{NP}-$completeness results.

When we analyze the performance of an algorithm in this thesis we assume the common *RAM-model* and use asymptotic analysis, the Big-Oh ($O$) notation. Sometimes the running time of a graph algorithm will be expressed by a polynomial function of $n, m$ and some other number $C$ given in the instance (e.g., the maximum edge cost). Such a running time is not polynomial since the input size of $C$ is $\log C$ instead of $C$. We call complexity in such a form *pseudo-polynomial*.

### 2.3.2 Approximation Algorithms

There are many hard combinatorial optimization problems that cannot be solved efficiently since we do not know polynomial-time algorithms for them. We are often interested in algorithms that provide a feasible but suboptimal solution in polynomial-time, together with a provable guarantee regarding its degree of sub-optimality.

We call an algorithm $A$ an *$\varepsilon$-approximation* algorithm for a minimization problem with optimal cost $OPT$, if for each instance of the problem, algorithm $A$ runs in polynomial time and returns a solution with cost $C_A$, so that $C_A \leq (1+\varepsilon)OPT$. Symmetrically, for a maximization problem we require $C_A \geq OPT/(1+\varepsilon)$.

An algorithm $A$ is said to be a *Fully Polynomial-Time Approximation Scheme* (FPTAS) for problem $\Phi$ if, for any instance $\phi$ of $\Phi$ and for any rational number $\varepsilon$, $A$ returns an *$\varepsilon$-approximation* solution in time polynomial both in $\phi$ and in $1/\varepsilon$.

An FPTAS is rather powerful from a theoretical view-point, given that we do not know an exact polynomial-time algorithm for the problem, as it allows an arbitrarily close approximation to the optimal solution.

## 2.4 Modelling of Fault-tolerance and QoS Metrics

The increasin demand for using multimedia applications over the Internet has triggered a spur of research on how to satisfy the fault-tolerance and quality of service (QoS) requirements of these applications, e.g., requirements regarding bandwidth, delay, jitter and packet loss rate. These efforts resulted in the proposals of several QoS-based frameworks, such as Integrated Services (Intserv), Differentiated Services (Diffserv), and Multi-Protocol Label Switching (MPLS).

Since we are interested in core algorithmic issues rather than practical implementations, we aim at exhibiting the essence of each question in a mathematically meaningful way. Although the ultimate motivation is practical networking, in the theoretical models we will peel off various layers of practical issues that are piled on the core algorithmic problem. This approach may also allow us to generalize the question, follow its own intrinsic logics and thus get some interesting unsolved problems.

### 2.4.1 Motivations of QoS Routing

Traditional Internet routing protocols calculate the shortest path based on a single metric (e.g. hop count) to determine the path between a node pair. This routing scheme has been successfully deployed in the last two decades for routing of Best Effort traffic in the Internet. However, the emergence of realtime multimedia services raises the question of whether path selection can be improved when QoS requirements are taken into account.

The problem of QoS-based routing is to find a routing mechanism that determines the path of a flow based on knowledge of both the available network resource and the QoS requirements of the flow.

As the current broadband networks is developing faster and faster, one may argue that bandwidth will be abundant and proper network provisioning can eliminate potential bottlenecks, so there is no need for this extra complexity. But one may also argue that bandwidth is not abundant yet and never will be, and although the long term network capacity may be sufficient, short term periods of congestion will always arise. Furthermore, link failures may destroy the perfect match between supply and demand. Therefore there is a need for QoS-based routing.

### 2.4.2 Framework of QoS Routing

The issues and requirements related to QoS-based routing are discussed and a framework is presented in [RFC2386], in which the following QoS-based routing objectives are listed:

1. Path selection should be based on both the network conditions (e.g. resource availability) and QoS requirements of the flow.
2. Optimization of network resource usage.
3. Graceful performance degradation under overload conditions.

**QoS-metrics:**

- The QoS-metrics of some links can be difficult to determine, they should be selected and modeled carefully to meet the QoS requirements of the flow.
- QoS-metrics which are considered useful are: bandwidth, delay, energy cost, packet loss rate and jitter. QoS-metrics apply to both the node and the outgoing link of that node. A uniform representation across different routing domains is required.

  Metrics can be divided into three classes. Let $d(e)$ be a metric for link $e$. For any path $P = (e_1, e_2, ..., e_n)$, metric $d$ is:

  *Additive* if $d(P) = d(e_1) + d(e_2) + ... + d(e_n)$;

  *Multiplicative* if $d(P) = d(e_1) * d(e_2) * ... * d(e_n)$;

  *Bottleneck* if $d(P) = min\{d(e_1), d(e_2), ..., d(e_n)\}$;

  According to this definition, the metrics delay, jitter, cost and hop count are additive. The metric reliability is multiplicative and the metric bandwidth is concave (bottleneck).

There are some other issues to be considered with QoS-based routing such as the scalability, the granularity of routing, the overall network performance, and even the administrative controls.

It is noted that always selecting the path with the best QoS may potentially deteriorate the overall network performance since flows might converge to some paths occupying relatively large amount of network resources.

A routing protocol can aid to distribute information about the expected cost of QoS paths. This information might be used to charge users, that require QoS services. QoS route selection should be based on multiple path calculation. Alternate path calculation is important, as the shortest path may deteriorate and cannot provide the desired QoS.

### 2.4.3 Fault-tolerance Design and Implementation

Many researchers count Fault-tolerance as a QoS metrics, but in this thesis we will consider it independently, because a network design problem will be much harder if we add a fault-tolerance requirement than the version with one more additive or multiplicative QoS requirement.

In the most general model, both the nodes and the links can fail, but here we will deal with a simplified model in which only the links can fail and the nodes are considered perfect. Links can be disabled because of two factors: (a) link congestion (a situation in which flow demand exceeds flow capacity and a link is blocked or an excessive queue builds up at a node), and (b) failures from broken links (fibre cut or energy deficiency).

Our focus in this thesis can actually be called the topological design problem and is an abstraction of the complete design problem. The complete design problem includes many other considerations, such as the following:

- The capacity in bits per second of each line is an important consideration, and connections between nodes can fail if the number of messages per minute is too large for the capacity of the line, if congestion ensues, and if a queue of waiting messages forms that causes unacceptable delays in transmission.

- If messages do not go through because of interrupted transmission paths or excessive delays, information is fed back to various nodes. An algorithm, called the routing algorithm, is stored at one or more nodes, and alternate routes are generally invoked to send such messages via alternate paths.

- When edge or node failures occur, messages are rerouted, which may cause additional network congestion.

- Edges between nodes are based on communication lines (twisted-copper and .ber-optic lines as well as coaxial cables, satellite links, etc.) and represent a discrete (rather than continuous) choice of capacities.

- Sometimes, the entire network design problem is divided into a backbone network design (discussed previously) and a terminal concentrator design (for the connections within a building).

- Some other considerations such as political sensitivity or military security issues sometimes govern node placement.

# Chapter 3

# The Constrained Shortest Disjoint $s$-$t$-Path Pair Problem

The growing demand for transmitting large volume data over the Internet has triggered a spur of research on quality of service (QoS) routing which aims to satisfy requirements regarding bandwidth, delay, jitter etc. At the same time, survivability of a network has assumed great importance in against of losing huge volumes of data due to a link cut or node failure.

To tackle these problems, 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. Multiple QoS-constrained paths can also be employed to achieve better load balance and improve quality of service in bandwidth-constrained MANETs.

Although many work have been done to find multiple node-disjoint or link-disjoint paths in a given network [Su74, ST84], 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 [OS04, OS05], 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 [OS04] can find two paths with total delay less than $2D(1+1/k)$ and total cost $k(1+\gamma)(1+\varepsilon)OPT$.

In this chapter we propose three 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 reduces the time complexity to $O(mn^4\log k/\varepsilon)$, and the cost is $(4\log k+3.5)(1+\varepsilon)OPT$. The third one further reduces the cost to $(2\log k + 3.5 + 1/\log k)(1+\varepsilon)OPT$ within $O(mn^4\log k\log\log k/\varepsilon)$ time. This is an improvement over [OS04]. At the end of this chapter, we extend this problem to the case of finding $k$ link-disjoint paths and present an algorithm for it.

## 3.1 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 violate 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 [OS04], 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 $m = |E|$.

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 $D(\overline{P}) \leq D$.

Although Problem RSP is intractable, there exist pseudo-polynomial solutions. This give rise to fully polynomial-time approximation schemes (FPTAS), whose computational complexity is reasonable [LoR01, ESZ02]. The most efficient scheme, presented in [ESZ02], 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 [LoR01] 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 [OS04, OS05, PS06] may set the cost of a link as zero in the residual graph, the latter algorithm was adopted and referred 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 $\mathcal{NP}$-hard. In addition, it was proved in [OS04] 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 an $(\alpha, \beta)$-*approximation.*

**Definition 1** **($(\alpha, \beta)$-*approximations*)** : Given an instance $(G, s, t, D)$ of Problem 2DP, an $(\alpha, \beta)$-approximation 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 $\beta$ 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 following sections 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.

## 3.2 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 [OS04] 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 2 (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, [DIG99] 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 [AMO93]. The following are the detailed description of the approximation algorithm 2DisjointPaths-1 and its fellow algorithms.

**Algorithm 3.1. 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   $(P_1, P_2) \leftarrow$ 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    $f \leftarrow$ MINDELAY($G, f^0, D, k$)
  6   Decompose $f$ into $\widehat{P}_1, \widehat{P}_2$ with $D(\widehat{P}_1) \leq D(\widehat{P}_2)$;
  7   return $(\widehat{P}_1, \widehat{P}_2)$ ;
```

**Algorithm 3.2. MINDELAY($G, f^0, D, k$)**

```
Input:
  G: the directed graph G=(V,E) with $\{d_l, c_l\}_{l \in E}$;
  $f^0$ : the original flow;
  D: the delay constraint;
  k : the approximation index;
Output:
  $f$: An improved flow with $D(f) \leq 2D(1+\frac{1}{k})$ for 2DP;
  1   $f \leftarrow f^0$;
  2   while $D(f) > 2D(1+\frac{1}{k})$ 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 $(u,v) \in G$ 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      $f^1 \leftarrow f$, $\mu \leftarrow D(W)/C(W)$;
  9      Augment flow $f$ along $W$:
  10  if  $C(f) > 2 * C(f^1)$ and $C(f) > C(f^0)$ then ;
  11     $f \leftarrow$ FinalImprove($G, f^1, D, k, C(f^0), C(f), \mu$);
```

```
12  return $f$ ;
```

**Algorithm 3.3. 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 $D(f) \leq 2D(1+\frac{1}{k})$;
  $\mu$ : 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   $LB \leftarrow \frac{2D-D(f^1)}{\mu}$;
2   if $C(f) \leq LB$ then return $f$;
3   $OPT_2 \leftarrow max\{LB, \frac{C(f^0)}{2}, \frac{C(f^1)}{2\log k+2}\}$;
4   while $D(f^1) > 2D(1+\frac{1}{k})$ 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 $(u,v) \in G$ with $f^1_{(u,v)} = 1$ do
8         Add $(v,u)$ with $d_{(v,u)} = -d_{(u,v)} \& c_{(v,u)} = 0$;
9       $W \leftarrow$ FCYCLE($G(f^1)$,$f^1$,$OPT_2$, $\frac{2D-D(f^1)}{OPT_2}$);
10      while $W = \emptyset$ do
11          $OPT_2 \leftarrow 2 * OPT_2$;
12          $W \leftarrow$ FCYCLE($G(f^1)$, $f^1$,$OPT_2$,$\frac{2D-D(f^1)}{OPT_2}$);
13      Augment flow $f^1$ along $W$;
14      $OPT_2 \leftarrow OPT_2/2$ ;
15  return $f^1$ ;
```

**Algorithm 3.4. FCYCLE($G, f^2, OPT_2, \mu$)**
**Input:**

```
  $G$: the residual graph;
  $f^2$ : the flow under check;
  $OPT_2$ : the lower bound of $OPT$;
  $\mu$ : the delay-to-cost ratio;
```

**Output:**

```
  $W$, a negative cycle $W$ with $\frac{D(W)}{C(W)} < \mu$ or $\emptyset$.
1   for each link $l \in G$ do
2       $d_l = d_l - c_l * \mu$;
3   for each node $g \in f^2$ do
4       $W \leftarrow$ SCYCLE($G, g, OPT_2, OPT_2$);
5       if $W \neq \emptyset$ then     return $W$;
6   return $\emptyset$ ;
```

**Algorithm 3.5. 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   $S \leftarrow \frac{L}{n+1}$;
2   for each $l \in E$ do
3       define $\widetilde{c}_l \equiv \lfloor c_l/S \rfloor + 1$;
4   $\widetilde{U} \leftarrow \lfloor U/S \rfloor + n + 1$;
5   for all $v \neq g$ do
6       $D(v, 0) \leftarrow \infty$;
7   $D(g, 0) \leftarrow 0$;
8   for $i = 1, 2, ..., \widetilde{U}$ do
9       for $v \in V$ do
10          $D(v, i) \leftarrow D(v, i-1)$;
11          for $l \in \{(u, v) \mid \widetilde{c}_{(u,v)} \leq i\}$ do
12            $D(v, i) \leftarrow min\{D(v, i), d_l + D(u, i - \widetilde{c}_l)\}$;
13      if $D(g, i) \leq -1$ 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 $\emptyset$ .
```

**Theorem 3.1.** *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 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 3.2.** *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 3.3.** *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 [OS04]. □

**Theorem 3.4.** *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 $\zeta$ 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 [AMO93] (page 67). □

**Theorem 3.5.** *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 [OS04] 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 3.4*, 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 $0 \leq x \leq h-1$. 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 $D(W_i^x) = \mu(W_i^x) = \frac{D(W_i)}{C(W_i)} \leq \frac{2D - D(f_i)}{OPT} \leq \frac{2D - D(f_i^x)}{OPT}$. Thus these virtual unit cycles satisfy the improvement requirement of *Theorem 3.4* 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 $D(f) \leq 2D(1 + \frac{1}{k})$, it holds that $\sum_{i=1}^{n_1} C(W_i) < 2\log kOPT$.

Since there will be no more than $n_1 \leq 2\log kOPT$ 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 [AMO93] (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 [Burns91, DIG99] whose complexity is $O(n^2m)$, then the time complexity of the first phase will be $O(mn^2 OPT \log k)$. Here we use the latter one.

For each cycle $W_j$ in the second phase, the following *Lemma 3.1* proves that $1 \leq C(W_j) \leq 2OPT$ and $\frac{D(W_j)}{C(W_j)} \leq \frac{2D-D(f_j)}{2OPT}$ for $n_1+1 \leq j \leq n_1+n_2$. 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 $(4\log k+2)OPT + C(f^0) \leq (4\log k + 2 + 1.5)OPT$.

Because the number of cycles identified in the second phase will be no more $O(\log kOPT)$, the time complexity is $O(mn^2 OPT \log^2 k)$. 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 $O(mn^2 OPT \log k)$. Combine the two phases together, the total complexity of algorithm MINDELAY is $O(mn^2 OPT \log k) + O(mn^2 OPT \log k) = O(mn^2 OPT \log k)$. □

**Lemma 3.1.** *Let $W$ be a cycle to be augmented in line 13 in algorithm FINALIMPROVE, then $1 \leq C(W) \leq 2OPT$ and $\frac{D(W)}{C(W)} \leq \frac{2D-D(f_i)}{2OPT}$. 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 $\mu$ is the current minimum delay-to-cost ratio for any negative cycle in $G(f)$, it holds that $\mu \leq \frac{2D-D(f)}{OPT}$ and $OPT \geq \frac{2D-D(f)}{\mu}$. Second, since algorithm 2DP-1 will return a flow $f$ with $C(f) \leq 1.5(1+\varepsilon)OPT$, 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 3.3*, there exists a cycle $W$ which satisfies $\frac{D(W)}{C(W)} \leq \frac{2D-D(f^1)}{OPT}$ and $C(W) \leq OPT$. So if algorithm FCYCLE cannot find a negative cycle with $C(W) \leq OPT_2$ and $\frac{D(W)}{C(W)} \leq \frac{2D-D(f^1)}{OPT_2}$, then it must hold that $OPT_2 < OPT$. Because if $OPT_2 \geq OPT$, then $\frac{2D-D(f^1)}{OPT_2} \geq \frac{2D-D(f^1)}{OPT}$ and according to *Theorem 3.2*, FCYCLE will return a negative cycle back. On the other hand, if algorithm FCYCLE finds a negative cycle with $C(W) \leq OPT_2$, then by *Theorem 3.1* we have $C(W) \leq U + L = 2OPT_2$. Since in the previous iteration $OPT_2 < OPT$, it follows that $C(W) \leq 2OPT$.

Notice that when FINALIMPROVE is called, the first phase has already found a cycle with cost no more than $(k+1)OPT$ and $C(f) \leq 2kOPT$ [OS04]. 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 3.6.** *Algorithm 2DisjointPaths-1 computes, in $O(mn^2 OPT \log k)$ 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 $O(mn^2 OPT \log k)$ 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 $O(mn^2 OPT \log k)$. □

## 3.3 Improve the Time Complexity

The above algorithm 2DisjointPaths-1 is a pseudo-polynomial algorithm since its time complexity $O(mn^2 OPT \log k)$ 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 [Has92, LoR01, ESZ02], but here we can employ the method used in [OS04] 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 [OS04] and our following algorithm 2DisjointPaths-2.

**Algorithm 3.6. 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 [LoR01, OS04].

**Theorem 3.7.** *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 $(4 \log k + 3.5)(1 + \varepsilon)OPT$. □

## 3.4 Further Decrease the Cost

*C. The Final Improve Algorithm to decrease the cost of the final flow*

**Algorithm 3.7. 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 computed by 2DP-1;*
  $C(f)$ : *the cost of the next flow with $D(f) \leq 2D(1+\frac{1}{k})$;*
  $\mu$ : *the current minimum delay-to-cost ratio;*
**Output:**
  $f^1$: *An improved flow with $D(f^1) \leq 2D(1+\frac{1}{k})$ for 2DP;*

```
1   $LB \leftarrow \frac{2D-D(f^1)}{\mu}$;
2   if $C(f) \leq LB$ then return $f$;
3   $OPT_2 \leftarrow max\{LB, \frac{C(f^0)}{2}, \frac{C(f^1)}{2\log k+2}\}$;
4   while $D(f^1) > 2D(1+\frac{1}{k})$ do
5      Construct the residual network $G(f^1)$
       of $G$ imposed by $f^1$:
6      Add to $G(f^1)$ each link $l$ in $G$ for which $f^1_l = 0$;
7      for each link $(u,v) \in G$ for which $f^1_{(u,v)} = 1$ do
8         Add a link $(v,u)$ to $G(f^1)$
          with $d_{(v,u)} = -d_{(u,v)}$ and $c_{(v,u)} = 0$;
9      $W \leftarrow$ FCYCLE($G(f^1), f^1, OPT_2, \frac{2D-D(f^1)}{OPT_2}$);
10     while $W = \emptyset$ do
11        $OPT_2 \leftarrow 2 * OPT_2$;
12        $W \leftarrow$ FCYCLE($G(f^1), f^1, OPT_2, \frac{2D-D(f^1)}{OPT_2}$);
13     $LB \leftarrow \frac{OPT}{2}$; $UB \leftarrow OPT$;
14     while $\frac{UB}{LB} \geq 1 + \frac{1}{\log k}$ do
15        $OPT_2 \leftarrow \frac{UB+LB}{2}$;
16        $W \leftarrow$ FCYCLE($G(f^1), f^1, OPT_2, \frac{2D-D(f^1)}{OPT_2}$;
17        if $W = \emptyset$ then $LB \leftarrow OPT_2$;
18           else $UB \leftarrow OPT_2$;
19     Augment flow $f^1$ along $W$;
20     $OPT_2 \leftarrow OPT_2/2$ ;
21  return $f^1$ ;
```

**Theorem 3.8.** *Algorithm MINDELAY will return a flow $f$ with delay $D(f) \leq 2D(1+\frac{1}{k})$ and cost $C(f) \leq (2\log k + 3.5 + 1/\log k)OPT$. The time complexity of MINDELAY is $O(mn^2 OPT \log k \log\log k)$.*

**Proof**: Algorithm MINDELAY includes two phases. The first phase is from line 2 to line 9, which adopts the same method in [OS04] 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 3.4*, 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 $0 \leq x \leq h-1$. 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 $D(W_i^x) = \mu(W_i^x) = \frac{D(W_i)}{C(W_i)} \leq \frac{2D-D(f_i)}{OPT} \leq \frac{2D-D(f_i^x)}{OPT}$. Thus these virtual unit cycles satisfy the improvement requirement of *Theorem 3.4* 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 $D(f) \leq 2D(1+\frac{1}{k})$, it holds that $\sum_{i=1}^{n_1} C(W_i) < 2\log kOPT$.

Since there will be no more than $n_1 \leq 2\log kOPT$ 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 [AMO93] (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 [Burns91, DIG99] 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 *Lemma 3.2* proves that $1 \leq C(W_j) \leq (1+1/\log k)OPT$ and $\frac{D(W_j)}{C(W_j)} \leq \frac{2D-D(f_j)}{(1+1/\log k)OPT}$ for $n_1+1 \leq j \leq n_1+n_2$. So the analysis will be the same as in the first phase except that $\zeta = \frac{1}{(1+1/\log k)OPT}$. Combine all cycles $W_1, ..., W_{n_1}, W_{n_1+1}, ..., W_{n_1+n_2-1}$ together, we can find that $\sum_{i=1}^{n_1+n_2-1} C(W_i) < 2(1+1/\log k)\log kOPT$. Since the cost of the last cycle will be no more than $(1+1/\log k)OPT$, it holds that $\sum_{i=1}^{n_1+n_2} C(W_i) < (2\log k+1+1+1/\log k)OPT$. Thus the total cost of flow $f$ returned by algorithm MINDELAY is at most $(2\log k+2+1/\log k)OPT + C(f^0) \leq (2\log k+2+1.5+1/\log k)OPT$.

Because the number of cycles identified in the second phase will be no more than $O(\log kOPT)$, and by *Lemma 3.2* the time complexity for computing each cycle is $O(mn^2(\log k+\log\log k))$. So the total time cost for the second phase is at most $O(mn^2OPT\log k(\log k+\log\log k))$. If we check more carefully, we can find that the guess $OPT_2$ will be halved (at line 20) after we find a cycle and augment the flow along it. So the value of $OPT_2$ will be halved for no more than $2\log\frac{2kOPT}{OPT/(2+2logk)}$ times, it will keep increasing at other situations. Thus a more precise bound for the time complexity is $O(mn^2OPT\log k\log\log k)$. Combine the two phases together, the total complexity of algorithm MINDELAY is $O(mn^2OPT\log k) + O(mn^2OPT\log k\log\log k) = O(mn^2OPT\log k\log\log k)$. □

**Lemma 3.2.** *Let $W$ be a cycle to be augmented in line* 13 *in algorithm FINALIMPROVE, then $1 \leq C(W) \leq (1+1/\log k)OPT$ and $\frac{D(W)}{C(W)} \leq \frac{2D-D(f_i)}{(1+1/\log k)OPT}$. The time complexity to identify a cycle is $O(mn^2\log k\log\log k)$.*

**Proof**: In the first three lines we set a lower bound for the cost of the optimum solution. At first, since $\mu$ is the current minimum delay-to-cost ratio for any negative cycle in $G(f)$, it holds that $\mu \leq \frac{2D-D(f)}{OPT}$ and $OPT \geq \frac{2D-D(f)}{\mu}$. Second, since algorithm

2DP-1 will return a flow $f$ with $C(f) \leq 1.5(1+\varepsilon)OPT$, 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 3.3*, there exists a cycle $W$ which satisfies $\frac{D(W)}{C(W)} \leq \frac{2D-D(f^1)}{OPT}$ and $C(W) \leq OPT$. So if algorithm FCYCLE cannot find a negative cycle with $C(W) \leq OPT_2$ and $\frac{D(W)}{C(W)} \leq \frac{2D-D(f^1)}{OPT_2}$, then it must hold that $OPT_2 < OPT$. Because if $OPT_2 \geq OPT$, then $\frac{2D-D(f^1)}{OPT_2} \geq \frac{2D-D(f^1)}{OPT}$ and according to *Theorem 3.2*, FCYCLE will return a negative cycle back. On the other hand, if algorithm FCYCLE finds a negative cycle with $C(W) \leq OPT_2$, then by *Theorem 3.1* we have $C(W) \leq U + L = 2OPT_2$. Since in the previous iteration $OPT_2 < OPT$, and from line 14 to line 18 we further bound that $UB/LB \leq 1 + 1/\log k$, it follows that $C(W) \leq (1 + 1/\log k)OPT$.

Notice that when FINALIMPROVE is called, the first phase has already found a cycle with cost no more than $(k+1)OPT$ and $C(f) \leq 2kOPT$ [OS04]. 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 *while* cycle at line 14 will run no more than $\log\log k$ iterations to achieve the bound. So the time complexity for each cycle is $O(mn^2(\log k + \log\log k))$. □

The above algorithm MINDELAY is a pseudo-polynomial algorithm since its time complexity $O(mn^2 OPT \log k \log\log k)$ 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 [Has92, LoR01, ESZ02], but here we can employ the method used in [OS04] 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 [OS04] and our algorithm 2QoSDP.

Algorithm 2QoSDP 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 [LoR01, OS04].

**Theorem 3.9.** *Algorithm 2QoSDP computes, in $O(mn^4 \log k \log\log k/\varepsilon)$ time, a $(1 + \frac{1}{k}, (2\log k + 3.5 + 1/\log k)(1+\varepsilon))$-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 2QoSDP is $O((m + n\log n)\log n) + O(mn^2 \log k \log\log k \cdot n^2/\varepsilon) = O(mn^4 \log k \log\log k/\varepsilon)$.

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 $(2\log k + 3.5 + 1/\log k)(1 + \varepsilon)OPT$. □

## 3.5 A $((k+1)/2, (k+1)(1+\varepsilon)/2)$ -approximation Algorithm for KRDP

**Problem KRDP (k Restricted Link Disjoint Paths)**: Given a source node $s$, a destination node $t$ and a QoS requirement $D$, find $k$ link-disjoint $(s,t)$-paths $P_1, ..., P_k$ that:
1) $D(P_i) \leq D, 1 \leq i \leq k$;
2) $\sum_i C(P_i) \leq \sum_i C(\overline{P}_i)$ for every other set of $k$ link-disjoint $(s,t)$-paths $(\overline{P}_1, ..., \overline{P}_k)$ that satisfy $D(\overline{P}_i) < D, 1 \leq i \leq k$.

Problem KRDP is a natural extension of Problem 2DP. In the following we extend the algorithm for Problem 2DP in [OS04] to solve Problem KRDP.

**Algorithm 3.8. k-RDP**$(G, s, t, D, \varepsilon, k)$
**Input:**
*$G$,$s$,$t$,$D$,$\varepsilon$: the same as that in 2QoSDP;*
*$k$ : the number of disjoint paths required;*
**Output:**
*$(\widehat{P}_1, \widehat{P}_2, \widehat{P}_3)$: three edge disjoint paths from s to t in G;*

```
1   Identify path $P_1$ in $G$ such that $D(P_1) \leq D$ by using Algorithm RSP;
2   $i \leftarrow 1, f \leftarrow \{P_1\}$;
3   while $i < k$ do
4      Construct the residual network $G(f)$ of $G$ imposed by $f$:
5         Add to $G(f)$ each link in $G$ that does not belong to $f$;
6         for each link $(u,v) \in f$ do
7            Add a link $(v,u)$ to $G(f)$ with
8               $d_{(v,u)} = 0$ and $c_{(v,u)} = 0$;
9      Identify path $P_{i+1}$ in $G$ such that $D(P_{i+1}) \leq (i+1) \cdot D$ by using Algorithm RSP;
10        Augment flow $f$ along path $P_{i+1}$:
11        for each link $(u,v) \in P_{i+1}$ do
12           if $f_{(v,u)} = 0$ then
13              $f_{(v,u)} \leftarrow 1$;
14           else
15              $f_{(v,u)} \leftarrow 0$;
16     Decompose flow $f$ into $i+1$ edge disjoint paths $\widetilde{P}_1 ... \widetilde{P}_{i+1}$
             with $D(\widetilde{P}_1) \leq ... \leq D(\widetilde{P}_{i+1})$;
17      $i \leftarrow i+1, f \leftarrow \{\widetilde{P}_1, ..., \widetilde{P}_{i+1}\}$;
18  return $\{\widetilde{P}_1, ..., \widetilde{P}_k\}$ ;
```

**Lemma 3.3.** *Let $G(f)$ be the residual network of $G$ imposed by $f_i = \{\widetilde{P}_1, ..., \widetilde{P}_i\}$. Then there exists a path $P_{i+1} \in G(f)$ such that $D(P_{i+1}) \leq (i+1)D$ and $C(P_{i+1}) \leq OPT_{i+1}$. Where $OPT_{i+1}$ denotes the optimal cost of $i+1$ delay-restricted link-disjoint paths.*

**Proof**: Suppose $f^*_{i+1}$ is the flow of the $i+1$ delay-restricted link-disjoint paths with the optimal cost. Let $G' \subseteq G$ be the union of $f_i$ and $f^*_{i+1}$ and $G'(f)$ be the residual graph of $G'$ imposed by $f_i$. Since $G'$ admits a $i+1$ flow and by the Path Augmentation Theorem in [AMO93], we can always identify a $s$-$t$ path $P_{i+1}$ in $G'$. Furthermore, since all links in $f_i$ has zero delay and cost, it is evident that $D(P_{i+1}) \leq (i+1)D$ and $C(P_{i+1}) \leq OPT_{i+1}$. □

**Theorem 3.10.** *Algorithm k-RDP computes, in $\mathcal{O}(kmn(\frac{1}{\varepsilon} + \log\log n))$ time, a $((k+1)/2, (k+1)(1+\varepsilon)/2)$-approximation solution for Problem KRDP.*

**Proof**: By Lemma 3.3, we can iteratively find the $k$ link-disjoint paths with delay $D(P_1) \leq D$, $D(P_2) \leq 2D$,... and $D(P_k) \leq kD$. Their costs are $C(P_1) \leq OPT_1 \leq OPT/k$, $C(P_2) \leq OPT_2 \leq 2OPT/k$,... and $C(P_k) \leq OPT_k \leq OPT$. Thus their total delay is $(k+1)kD/2$ while the optimal total delay is $kD$. And their total cost is $(k+1)OPT/2$, so the final flow is a $((k+1)/2, (k+1)(1+\varepsilon)/2)$-approximation solution.

Since Algorithm RSP costs $\mathcal{O}(mn(\frac{1}{\varepsilon} + \log\log n))$ time and it will run $k$ rounds. Thus the time complexity of Algorithm k-RDP is $\mathcal{O}(kmn(\frac{1}{\varepsilon} + \log\log n))$. □

## 3.6 A Small Example

The algorithms in this chapter is very complex to implement, and hard to understand. To make it a little bit apprehensible, we will give an example in this section.

In the following Figure 3.1, there are 10 nodes and 13 edges in graph $G$, each edge has a cost value and a delay value. The source node is $s$ and the sink node is $t$, we want to find two edge-disjoint paths from $s$ to $t$ with total delay no more than 40 and minimum possible total cost.

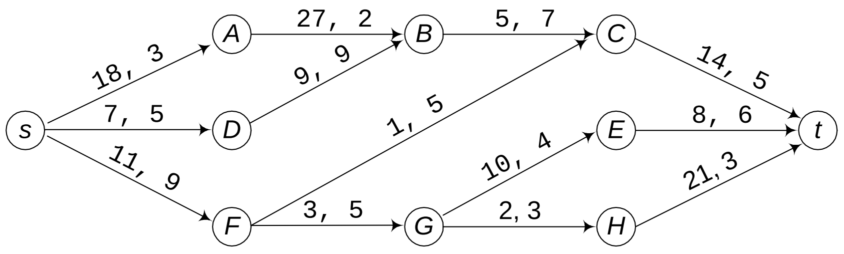


Figure 3.1: Each link has a cost (the first value) and a delay (the second value). The total delay-bound is $20 * 2 = 40$.

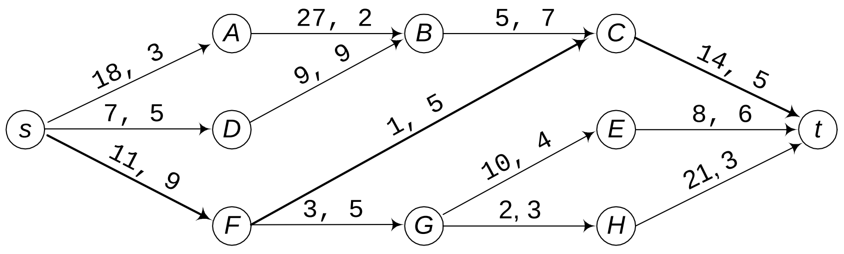


Figure 3.2: The shortest path is marked by bold lines, it has cost 26 and delay 19.

In the first step, we use algorithm RSP to compute a minimum-cost path from $s$ to $t$ under the delay bound 20. The returned path is $s \rightarrow F \rightarrow C \rightarrow t$ in Figure 3.2, its cost is 26 and its delay is 19.

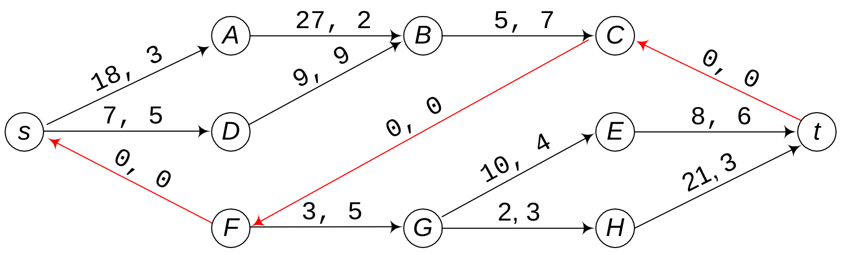


Figure 3.3: The reversed path is marked by red color.

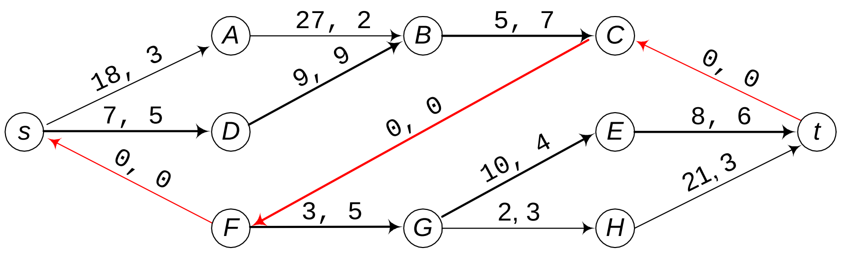


Figure 3.4: The shortest path in the residual graph under delay bound 40, the solution path has cost 42 and delay 36.

In the second step, we construct the residual graph of the above computed shortest path under delay bound 20. We reverse the direction of each edge in this path and set both its cost and delay value to zero .

In the 3rd step, we call algorithm RSP another time to find in this residual graph the shortest path under delay bound 40. The solution path is $s \to D \to B \to C \to F \to G \to E \to t$, its cost is 42 and its delay is 36.

In the 4th step, we decompose the flow computed in Step 1-3 into two paths. Since edge $C \to F$ appears in the first path and edge $F \to C$ appears in the second path, we will remove them in the decomposed solution. This solution includes two paths $s \to D \to B \to C \to t$ and $s \to F \to G \to E \to t$, it has total cost $35 + 32 = 67$ and delay $26 + 24 = 50 > 40$.

In the 5th step, we construct the residual graph of the above two paths. We reverse the direction of each edge in this path and set its cost to zero, but we will make its delay to be the negative value of the original one.

In the 6th step, we use our algorithm 2DisjointPaths to find in this residual graph a set of negative delay cycles so that we will decrease the delay to the desired bound

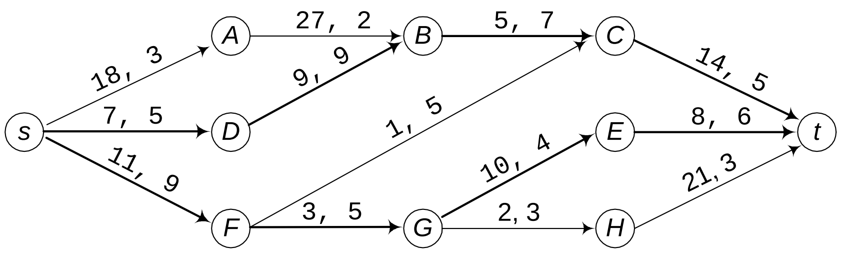


Figure 3.5: Decompose the flow into two paths, this solution has cost 67 and delay 50.

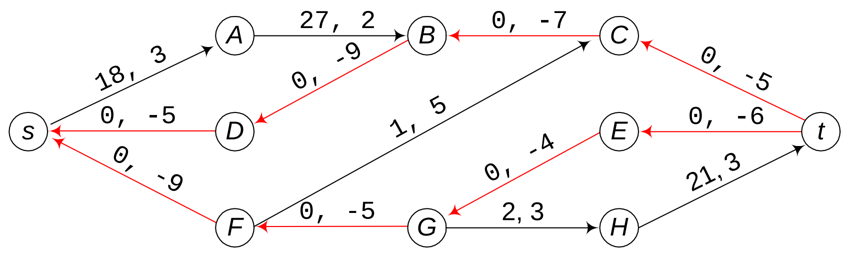


Figure 3.6: The residual graph of the two paths in Figure 3.5 .

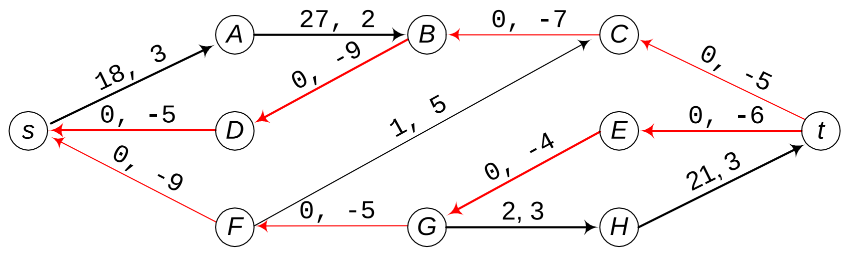


Figure 3.7: We can identify two negative delay cycles in this graph.

by cancelling these cycels. We can see in Figure 3.7 that there are two such cycles: $s \to A \to B \to D \to s$ and $t \to E \to G \to H \to t$.

In the final step, we cancel the above-computed cycles on the previous flow and get the final solution. Check in Figure 3.8 the two paths: $P_1\{s \to A \to B \to C \to t\}$ and $P_2\{s \to F \to G \to H \to t\}$. $P_1$ has delay 20 and cost 64, while $P_2$ has delay 20 and cost 37, so this final solution has a total delay of 40 and its total cost is 101. In this example, it happens to be the optimal solution.

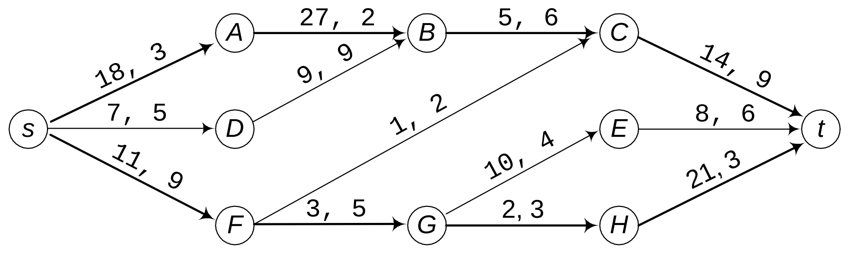


Figure 3.8: The final solution: $P_1$: cost 64, delay 20; $P_2$: cost 37, delay 20.

## 3.7 Concluding Remarks

The major contributions of this chapter are several 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. The third one further reduces the cost to $(2\log k + 3.5 + 1/\log k)(1+\varepsilon)OPT$ within $O(mn^4 \log k \log\log k/\varepsilon)$ time. 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.

# Chapter 4

# Solve the 2DP Problem by Lagrangian Relaxation

In this chapter we propose a new simple and efficient approximation algorithm for the 2DP Problem. This algorithm can find two disjoint paths with total delay less than $(1+1/k)D$ 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 $O(m \log_{1+m/n} n \log \frac{(k+1)C(f_d)}{C(f_c)})$, which is much lower than the previous algorithms.

## 4.1 A $(1+\frac{1}{k}, 1+k)$ -Approximation 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 [OS04] 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.

Here 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 $\alpha$, we can find a corresponding optimal flow $f^\alpha$ 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 $\alpha$ 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[Su74, ST84, Bhan99]. 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)** [AMO93]: 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 [Su74], 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 [Dij59], 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 $\alpha$ is fixed, then we can find an optimal flow. So we will naturally come to the approach of testing different values of $\alpha$ and try to find a feasible solution.

**Theorem 4.1.** *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 $0 < \alpha_1 \le \alpha_2$, then $D(f_1) \ge D(f_2)$ and $C(f_1) \le C(f_2)$.*

**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 $W(f_1,\alpha_1) = C(f_1) + \alpha_1 * D(f_1) \le C(f_2) + \alpha_1 * D(f_2)$ . 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 $W(f_2,\alpha_2) = C(f_2) + \alpha_2 * D(f_2) \le C(f_1) + \alpha_2 * D(f_1)$. Add the two inequations together and use the fact that $0 < \alpha_1 \le \alpha_2$:

$$\alpha_1 * D(f_1) + \alpha_2 * D(f_2) \le \alpha_1 * D(f_2) + \alpha_2 * D(f_1) \quad \Rightarrow \quad D(f_2) \le D(f_1);$$
$$C(f_1)/\alpha_1 + C(f_2)/\alpha_2 \le C(f_2)/\alpha_1 + C(f_1)/\alpha_2 \quad \Rightarrow \quad C(f_1) \le C(f_2).$$

□

For the MCF problem, we assume that there is at least one flow $f^*$ with $D(f^*) \le D$ 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 4.1, $x$-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 $\alpha$, we can deem them as a single pair. To make it clear, we divide the first quadrant into 6 faces:

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

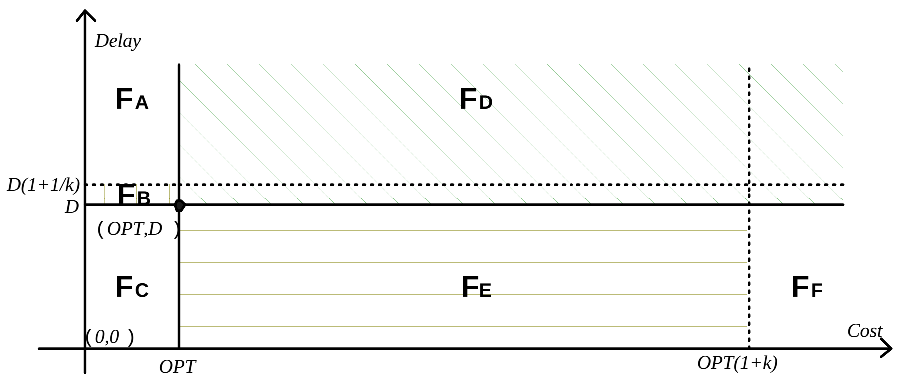


Figure 4.1: The solution space for the MCF Problem, each point in the plane corresponds to a set of flows with the same cost and delay.

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 $\alpha$, 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 $\alpha$. 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 4.2.** *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 $f^\alpha$ 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 $\alpha$(Figure 4.2), 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 4.3). 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 $\alpha$ 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

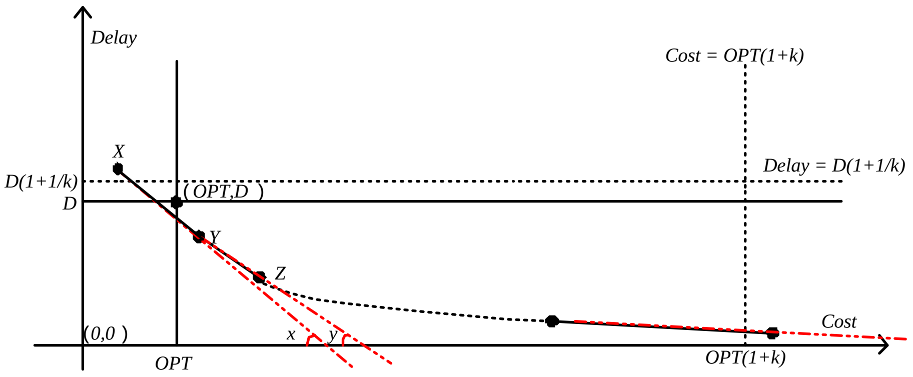


Figure 4.2: If we connect all solutions that can be found by Algorithm DDP, then they will form a convex and piecewise-linear line.

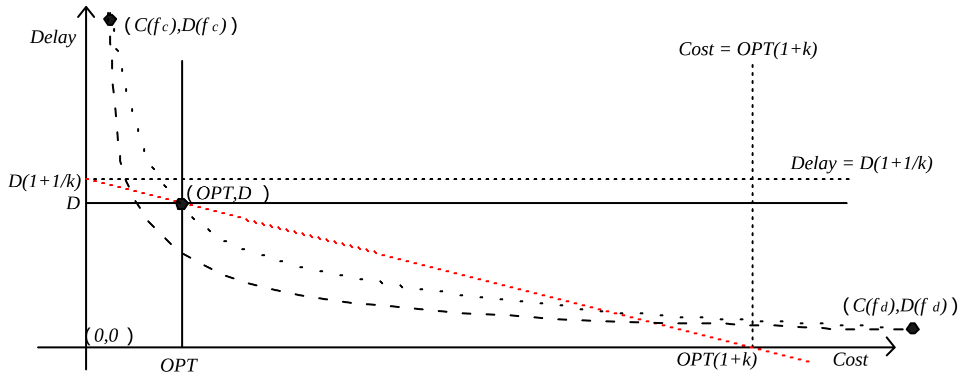


Figure 4.3: The solution line will start from $(C(f_c), D(f_c))$ and end at $(C(f_d), D(f_d))$.

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 measure 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 $\alpha$ if $D(f^\alpha) > (1+\frac{1}{k})D$ and decreasing it when $D(f^\alpha) < D$ by adjusting the value of $\alpha$. If we increase(decrease) the value of $\alpha$ 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 $D < D(f^\alpha) \le (1+\frac{1}{k})D$, 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 $W(f,\alpha) = C(f) + D(f) * \frac{k*OPT}{D} \le W(f^*,\alpha) = C(f^*) + D(f^*) * \frac{k*OPT}{D} = OPT * (1+k) = D * \alpha * (1+\frac{1}{k})$. 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 4.3.** *: Let $f$ be the flow returned by $DDP(\alpha)$ which satisfies $W(f,\alpha) = D * \alpha * (1+\frac{1}{k})$, then $D(f) \le (1+\frac{1}{k})D$ and $C(f) \le (1+k)OPT$.*

**Proof:** According to the assumption we have

$$C(f) + D(f) * \alpha = W(f, \alpha) = D * \alpha * (1 + \frac{1}{k}).$$

Thus

$$D(f) * \alpha \leq D * \alpha * (1 + \frac{1}{k}) \quad \Rightarrow \quad D(f) \leq D * (1 + \frac{1}{k}).$$

Let $f^*$ be the optimal flow with $D(f^*) \leq D$ and $C(f^*) = OPT$, then

$$D * \alpha * (1 + \frac{1}{k}) = W(f, \alpha) \leq W(f^*, \alpha) = C(f^*) + \alpha * D(f^*) \leq OPT + \alpha * D$$

$$\Rightarrow D * \alpha * \frac{1}{k} \leq OPT \quad \Rightarrow \quad D * \alpha \leq OPT * k$$
$$\Rightarrow C(f) \leq W(f, \alpha) \leq OPT + \alpha * D \leq (k+1) * OPT.$$

□

We will further prove that we can find such an $\alpha$ by the following theorem.

**Theorem 4.4.** *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 $\alpha_0$ 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 $\alpha$. So if we define $W'(f, \alpha) = W(f, \alpha) - D * \alpha * (1 + \frac{1}{k})$, then it is also a linear function of $\alpha$.

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

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

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

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

Since $\alpha_1 < \alpha_2$, we have that $D(f_1) > D(f_2)$ by Theorem 4.1. 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 4.4.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:

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

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 $\alpha_0$ which satisfies $U'(\alpha_0) = 0$ and $W(f_0, \alpha_0) = \alpha_0 * (1 + \frac{1}{k}) * D$. □

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

```
Algorithm 4.1. 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:
  $\{\widetilde{P_1}, \widetilde{P_2}\}$: 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 $W(f,\alpha) - D * \alpha(1 + \frac{1}{k}) > 0$ then $\alpha_{LB} \leftarrow \alpha$, $f_{LB} \leftarrow f$;
10           else $\alpha_{UB} \leftarrow \alpha$, $f_{UB} \leftarrow f$;
11        if $D(f_{LB}) \leq D * (1 + \frac{1}{k})$ then
12           $f_{UB} \leftarrow f_{LB}$;
```

```
13          Break the "while" loop;
14      if $D(f_{LB}) = D(f_{UB})$ then Break the "while" loop;
15  Decompose flow $f_{UB}$ into 2 link disjoint paths $\widetilde{P}_1$ and $\widetilde{P}_2$ with $D(\widetilde{P}_1) \leq D(\widetilde{P}_2)$;
16  return $\{\widetilde{P}_1, \widetilde{P}_2\}$ ;
```

**Theorem 4.5.** *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 4.4 we know that there exists an $\alpha_0$ 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 4.3. 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 4.1.

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 4.4.a).

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

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. □

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).

**Theorem 4.6.** *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.*

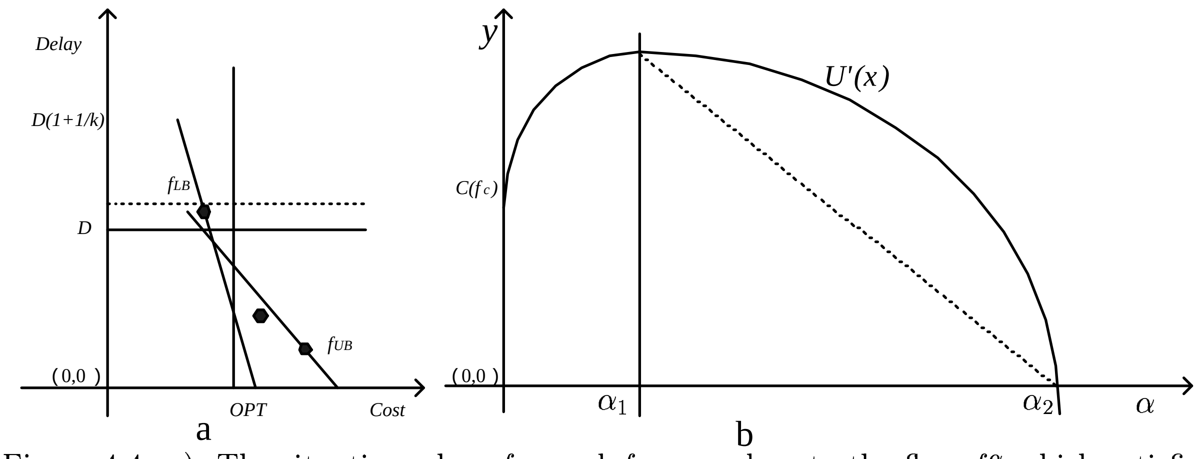


Figure 4.4: a), The situation when $f_{_{LB}}$ and $f_{_{UB}}$ are close to the flow $f^{\alpha}$ which satisfies $U'(\alpha) = 0$. b), Function $U'(\alpha)$ will be monotone increasing when $D(f^{\alpha}) \geq \frac{1+k}{k}D$ and monotone decreasing when $D(f^{\alpha}) \leq \frac{1+k}{k}D$.

**Proof:** According to the proof in [Su74], 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 4.4, 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 4.4.b).

Let $\alpha_1$ be the value when $U'(\alpha_1)$ is maximum and $\alpha_2$ 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 $\alpha_1$ and $\alpha_2$, fortunately we can estimate their difference via its relationship between $U'(\alpha_1)$. Let us first consider the difference between $U'(\alpha)$ and $U'(\alpha - 1)$:

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

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 4.1). 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 $\alpha$ from $\alpha_1$ to $\alpha_2$, 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 $\alpha$ by 1, the value of $U'(\alpha)$ will decrease by less than $\frac{1+k}{k}D - D(f^{\alpha_2})$. So we have $(\alpha_2 - \alpha_1) * (\frac{1+k}{k}D - D(f^{\alpha_2})) \geq U'(\alpha_1) - 0 \quad \Rightarrow \quad \alpha_2 - \alpha_1 \geq \frac{U'(\alpha_1)}{\frac{1+k}{k}D - D(f^{\alpha_2})}$. 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. □

## 4.2 Some Extensions

The method developed in the previous chapter is very simple and efficient. Whet'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 4.7.** *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 4.2. 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   $\alpha_{LB} \leftarrow 0, \quad \alpha_{UB} \leftarrow k\frac{C(T_d)}{D}$;
4   $T_{LB} \leftarrow T_c, \quad T_{UB} \leftarrow T_d$;
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       $T \leftarrow$ 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 $D(T_{LB}) \le D*(1+\frac{1}{k})$ then
12            $T_{UB} \leftarrow T_{LB}$;
13            Break the "while" loop;
14         if $D(T_{LB}) = D(T_{UB})$ then Break the "while" loop;
15   return $T_{UB}$ ;
```

**Theorem 4.8.** *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 $D(T) \le (1 + \frac{1}{k})D$ and $C(T) \le (1 + k)OPT$. 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 [GGST86] 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 4.6. □

## 4.3 Concluding Remarks

The major contribution of this chapter is a polynomial-time approximation algorithm for finding two Delay-Restricted Link Disjoint Paths with minimum total cost. For any fixed number $k$, the previous best result can find a solution that violates the delay constraint by factor of at most $1 + \frac{1}{k}$ with cost $k(1 + \gamma)(1 + \varepsilon), (\gamma > \frac{\log k}{k}, \varepsilon > 0)$ times more than the optimum. 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.

| | Delay | Cost | Time Complexity |
|---|---|---|---|
| [OS04] | $1+\frac{1}{k}$ | $(k+2\log k+1)(1+\varepsilon)$ | $O(\frac{mn^2k^2\log k}{\varepsilon}\log(CD))$ |
| [PS06] | $1+\frac{1}{k}$ | $(4\log k+3.5)(1+\varepsilon)$ | $O(mn^4\log k/\varepsilon)$ |
| This chapter | $1+\frac{1}{k}$ | $1+k$ | $O(m\log_{1+\frac{m}{n}} n\log\frac{(k+1)C(f_d)}{C(f_c)})$ |

From the above table we can see that the complexity 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.

# Chapter 5

# Edge-disjoint Paths and Unsplittable Flows

In the previous two chapters, we have studied the problem of finding multiple edge-disjoint paths with multiple restrictions from a source node $s$ to a sink node $t$ in graph $G$. A natural extension of this problem is to find edge-disjoint paths between multiple source-sink pairs. This problem is known as the Edge-disjoint Paths Problem (EDP), it has attracted considerable attention in recent years from research areas such a graph theory, VLSI design and network routing/flow. Notice that here we have multiple source nodes and each source node has one or more corresponding sink nodes, which makes the problem much more difficult than the single source node situation.

The Unsplittable Flow Problem (UFP) is the generalization of EDP where every edge is capacitated and each source-sink pair has a commodity with a given demand. UFP was introduced in the PhD theis of Kleinberg, which solidified the various stands of work on disjoint-path problems until then and gave impetus to new research.

In this chapter we first develop an efficient algorithm for computing a minimum-cost single-source unsplittable flow in a graph with arbitrary edge capacity. Then we consider the special case when the largest demand in $G$ is far less than the minimum capacity in it. After that we study the minimum-cost single-source $k-$splittable flow problem and present an algorithm with provable performance guarantee for it.

## 5.1 The Edge-disjoint Paths Problem

**Edge-disjoint Paths** (EDP)

- INSTANCE: Graph $G = (V, E)$, a collection of $l$ source-sink pairs $F = \{(s_i, t_i) : s_i \in V, t_i \in V,\ i = 1, \ldots, l,\ l \geq 2\}$.
- SOLUTION: A set of edge-disjoint paths $P_1, P_2, ..., P_l$ where $P_i$ is an $s_i - t_i$ path, $i = 1, ..., l$.
- MEASURE: The number of edge-disjoint paths.

Based on whether $G$ is directed or undirected and the edge-disjointness condition one obtains four basic problem versions. Among them, an undirected problem can be reduced to its directed counterpart by replacing an undirected edge with an appropriate gadget; both reductions maintain planarity. And edge-disjoint problem can be reduced to its node-disjoint counterpart by replacing $G$ with its line graph. Directed node-disjoint paths reduce to directed edge-disjoint paths by replacing every vertex with a pair of new vertices connected by an edge. There is no know reduction from a directed problem to an undirected problem.

For general $k$ all four basic problems are $\mathcal{NP}$-complete. The undirected vertex-disjoint paths problem was shown to be $\mathcal{NP}$-complete by Knuth in 1974 via a reduction from SAT. This implies the $\mathcal{NP}$-completeness of directed vertex-disjoint paths and directed edge-disjoint paths. Even, Itai and Shamir [EIS76] showed that both problems remain $\mathcal{NP}$-complete on directed acyclic graphs (DAGs). In the same paper the undirected edge-disjoint paths problem was shown $\mathcal{NP}$-complete even when $T$ contains only of two distinct pairs of terminals. In the case when all source nodes are the same, all four versions are in $P$ as special cases of maximum flwo. For planar graphs Lynch's [Lyn75] reduction shows $\mathcal{NP}$-completeness for undirected node-disjoint paths; Kramer and van Leeuwen show that undirected EDP is $\mathcal{NP}$-complete. The $\mathcal{NP}$-completeness of the directed planar version follows.

For fixed $k$, the directed versions are $\mathcal{NP}$-complete even for the case of two pairs with opposing source-sinks, i.e.,$(s, t)$ and $(t, s)$[FHW80]. Undirected node-disjoint paths, and by implication edge-disjoint paths as well, can be solved in polynomial-time. This is an outcome of the celebrated project of Robertson and Seymour [RS95] on graph minors. It is notable that for fixed $k$, node-disjoint paths, and by consequence EDP, can be solved on DAGs by a fairly simple polynomial-time algorithm. Earlier polynomial-time algorithms for $k - 2$ include the one by Perl and Shiloach [PS78] on DAGs and the ones derived independently by Seymour, Shiloach and Thomassen [Shi80] for node-disjoint paths on general undirected graphs.

For planar graphs and fixed $k$ the directed node-disjoint path problem is in $P$ while the complexity of the edge-disjoint case is OPEN. When the input graph is a tree, Garg, Vazirani and Yannakakis [GVY97] gave a polynomial-time algorithm to maximize the number of pairs that can be connected by edge-disjoint paths. The algorithm extends for node-disjoint paths. By total unimodularity, the EDP maximization problem is polynomial-time solvable on $di-trees$ as well, i.e., directed graphs in which there is a unique directed path from $s_i$ to $t_i$, for all $i$; a reduction to a minimum-cost circulation problem is also possible in this case [CLR03]. Reducing directed node-disjoint paths to EDP maintains the di-tree property, hence the former problem is in $P$ as well. Oberserve that directed out-trees and in-trees are special cases of di-trees.

## 5.2 The Minimum-Cost Single-Source Unsplittable Flow Problem

The unsplittable flow problem is the generalization of EDP where every edge has a positvie capacity, and every commodity has a demand which should be routed in an unsplittable manner. If we relax the requirement that each commodity should use exactly one path, we will obtain the multicommodity flow problem which is well known to be solvable in polynomial-time. When all sources of a multicommodity flow instance coincide at a vertex $s$ and all the sinks at a vertex $t$, we obtain the classical maximum flow problem to which we also refer to as $s$-$t$-flow. In the chapter we will focus on the single-source case.

Let $G = (V, E)$ be a directed graph with edge capacities $u : E \to \mathbf{R}^+$, a designated source vertex $s \in V$ and $k$ commodities each associates with a terminal $t_i$ and a demand $d_i \in \mathbf{R}^+$, $1 \leq i \leq k$. The Single-Source Unsplittable Flow Problem (UFP) asks for a feasible unsplittable flow which can route $d_i$ units of commodity $i$ along a single $s - t_i$ path for each $i$ without violate the capacity constraint of any edge $e \in E$. Here we use $T \subseteq V$ to denote the set of $k$ terminals, $|T| = k$, $|V| = n$, $|E| = m$.

In this paper we will adopt the standard model used by previous researchers. A *splittable* flow on the graph $G$ is a function $f : E \to \mathcal{R}^+$ satisfying the flow conservation constraints in each vertex and the source vertex $s$ is the only vertex where the outflow may exceed the inflow. We will say a flow *satisfies* all demands if the inflow minus the outflow at each vertex except $s$ is equal to the sum of all demands located at it. A flow $f$ will be called *feasible* if the total flow on each edge does not exceed the capacity of this edge.

The Minimum-Cost Single-Source Unsplittable Flow Problem (MSUFP) is the edge-weighted extension of UFP. In MSUFP, there is one more edge cost function $c : E \to \mathbf{R}^+$. An unsplittable flow $f$ contains a set of paths $\{P_1, ..., P_k\}$, where $P_i$ starts at the source $s$ and ends at $t_i$. The cost of a path $P_i$ is defined as $c(P_i) = \sum_{e \in P_i} c_e$ and the cost $c(f)$ of a flow $f$ is give by $c(f) = \sum_{e \in E} c_e \cdot f_e$. MSUFP asks for a feasible unsplittable flow with minimum cost. Since UFP and MSUFP are $\mathcal{NP}$-complete[KleinTh], most efforts are devoted to designing approximation algorithms. We will use the bicriteria $(\alpha, \beta)$-approximation to denote a solution for MSUFP which simultaneously has congestion index at most $\alpha$ and cost at most $\beta$ times the optimal.

After been introduced by Kleinberg [KleinTh], the UFP problem attracted considerable attention [Klein96, KleinTh, DGG99, KS02, Sku02]. There are three optimization versions of it: a), the Minimum Congestion version seeks the minimum $\alpha > 1$ such that the returned solution will be a feasible unsplittable flow if all capacities are multiplied by $\alpha$. b), the Minimum Number of Rounds version aims to partition the set of commodities into a minimum number of subsets for which we know an unsplittable flow solution. c), the Maximum Routable Demand version wants to find a subset which has maximum total demand and for which we know an unsplittable flow solution.

In this section we will focus on the congestion optimization version. A commonly used *congestion assumption* is that the maximum demand does not exceed the minimum capacity (i.e., $d_{max} \leq u_{min}$). Under this assumption Kleinberg gave the first constant $16-$approximation algorithm for UFP. Later Dinitz, Garg and Goemans [DGG99] obtained a congestion bound of 2 and proved that this ratio was the best possible, their algorithm has a congestion rate $3 + 2\sqrt{2}$ if we remove the congestion assumption. For the MSUFP problem, Kolliopoulos and Stein [KS02] gave a bicriteria $(3, 2)-$approximation algorithm with ratio 3 for congestion and 2 for cost under the congestion assumption. Built on this result, Skutella [Sku02] cleverly improve the approximation guarantee to $(3, 1)$, which is currently the best bicriteria bound.

UFP contains several well know $\mathcal{NP}$-complete problems as special cases: Partition, Bin Packing, Scheduling on parallel machines to minimize makespan[Klein96]. Among them the scheduling problem has a bicriteria $(2, 1)-$approximation algorithm, and so does its generalization - the Generalized Assignment Problem [ST93]. Naturally we will wonder whether a bicriteria $(2, 1)-$approximation algorithm is possible for MSUFP. This is the best possible bicriteria bound since Erlebach and Hall proved that for MSUFP and arbitrary $\varepsilon > 0$ there is no bicriteria $(2 - \varepsilon, 1)-$approximation algorithm for unless

$\mathcal{P} = \mathcal{NP}$[EH02].

## 5.3 A New Approximation Algorithm for MSUFP

In this section we focus on the case without the congestion assumption, thus each edge has arbitrary capacity. We give a $(3+2\sqrt{2},1)-$approximation algorithm to MSUFP. We adopt a different rounding method which decreases the time complexity and makes the computation and implementation more convenient.

Basically, most previous research work follows the method which first find a minimum-cost splittable flow that satisfies all demands, then turn this flow into an unsplittable flow while increasing the total flow through any edge by a certain amount. The following linear program is for the minimum-cost splittable flow problem.

$$\begin{array}{lllr}
\textbf{Minimize} & \sum\limits_{(i,j)\in E} c_{ij}x_{ij} & & \\
\textbf{Subject to} & \sum\limits_{\{j:(i,j)\in E\}} x_{ij} - \sum\limits_{\{j:(j,i)\in E\}} x_{ji} = \sum\limits_{\{j:1\le j\le k\}} d_j, & for\ i = s & \\
& \sum\limits_{\{j:(i,j)\in E\}} x_{ij} - \sum\limits_{\{j:(j,i)\in E\}} x_{ji} = -d_i, & for\ i \in T & (5.1)\\
& \sum\limits_{\{j:(i,j)\in E\}} x_{ij} - \sum\limits_{\{j:(j,i)\in E\}} x_{ji} = 0, & for\ i \in V-\{s\}-T & \\
& 0 \le x_{ij} \le u_{ij}, & for\ each\ (i,j)\in E. &
\end{array}$$

Since the minimum-cost splittable flow has a cost which is always no more than that of an unsplittable flow, we can make sure the cost of a solution will be within the optimal if it is built on the minimum-cost splittable flow and the cost never increase during the rounding process. The tradeoff is that we may have to expand the capacity of each edge. If we bound the cost of the solution to be no more than that of the minimum-cost splittable flow, all existing algorithms can only guarantee a congestion rate of no less than 3 times the capacity for each edge, under the assumption that the maximum demand does not exceed the minimum capacity.

But if we want to remove this congestion assumption, the common minimum-cost splittable flow will not be an ideal basic step because in such a flow a very large flow may route a small portion along an edge that has the minimum capacity in the graph. Since for a solution of the unsplittable flow, a commodity $i$ with demand $d_i$ cannot pass by a *bottleneck* edge with capacity less than $d_i$, we may consider add some restrictions to the above Linear Program. In the above Linear Program, we use $x_{ij}$ do denote the flow size on edge $(i,j)\in E$; while in the following Linear Program we will replace it by $x_{ij}^l$ which denotes the flow of commodity $l$ on edge $(i,j)\in E$.

$$\begin{array}{lll}
\textbf{Minimize} & \sum\limits_{1\leq l\leq k}\sum\limits_{(i,j)\in E} c_{ij} * d_k * x_{ij}^k & \\
\textbf{Subject to} & \sum\limits_{\{j:(i,j)\in E\}} x_{ij}^l - \sum\limits_{\{j:(j,i)\in E\}} x_{ji}^l = d_l, & for\ i = s\ and\ each\ demand\ l \in T \\
 & \sum\limits_{\{j:(i,j)\in E\}} x_{ij}^l - \sum\limits_{\{j:(j,i)\in E\}} x_{ji}^l = -d_l, & for\ l \in T\ and\ i = l \\
 & \sum\limits_{\{j:(i,j)\in E\}} x_{ij}^l - \sum\limits_{\{j:(j,i)\in E\}} x_{ji}^l = 0, & for\ i \in V - \{s\} - T,\ l \in T\ and\ i \neq l \\
 & 0 \leq x_{ij}^l * d_l \leq u_{ij}, & for\ each\ (i,j) \in E\ and\ l \in T, \\
 & x_{ij}^l = 0, & for\ each\ (i,j) \in E\ with\ d_l > u_{ij}.
\end{array} \tag{5.2}$$

Next we will also first compute a minimum-cost splittable flow, then we adopt a different rounding method which is modified from the method used in [ST93]. We will directly use the following well-known results on splittable flows[AMO93, Sku02].

**Theorem 5.1.** *Let $G = (V, E)$ be a directed graph with capacities and costs on the edges. Moreover, there is a source vertex $s \in V$ and $k$ sinks $t_1, ..., t_k \in V$ with demands $d_1, ..., d_k$ respectively.*

*a) There exists a feasible (splittable) flow satisfying all demands if and only if, for any subset $\mathcal{C} \subseteq V - \{s\}$, the sum of capacities of edges in the directed cut $(V - \mathcal{C}, \mathcal{C})$ is at least $\sum_{i:\ t_i \in \mathcal{C}} d_i$. We refer to this condition as* **cut** *condition.*

*b) If the cut condition is satisfied and all demands and capacities are integral, then there exists a feasible (splittable) flow satisfying all demands with minimum cost such that the flow value on any edge is integral. Moreover, such a flow can be computed in polynomial-time.*

Let $f$ be a minimum-cost splittable flow in $G$ with no commodity pass by a bottleneck edge, we will first decompose it into a set of $s - t_i-$paths. For each commodity $i$ in $T$, there is a corresponding set of $s - t_i-$paths with total demand $d_i$. We can immediately route those demands whose corresponding set has only one path, thus we assume that all demands has multiple corresponding paths. Without loss of generality we assume that $d_1 \geq d_2 \geq ... \geq d_k$, this can be easily achieved by sorting the demands in the preprocessing step. Now we will divide all commodities into different groups according to the size of each demand.

$$[d_{max} = d_1, d_{max} * z]|(d_{max} * z, d_{max} * z^2]|...|(d_{max} * z^{y-1}, d_{max} * z^y] \tag{5.3}$$

In the above equation, $0 < z < 1$ and a commodity $i$ will be classified into the $j$-th group if $d_{max} * z^j \leq d_i < d_{max} * z^{j-1}$. Let $y$ be the integer satisfies $d_{max} * z^y \leq d_{min} < d_{max} * z^{y-1}$, then we will have no more than $y$ groups. Since we can simply remove those empty groups, we have that the total number of groups will be no more than $min\{y, k\}$.

After we put all commodities to different groups, we will route them group-by-group. Consider the $g$-th group, let $d_{max}^g$ be the commodity with maximum demand in it and let $d_{min}^g$ be the commodity with minimum demand in it. Then we have $d_{max}^g \leq d_{min}^g/z$. We pick out the flow $f_g$ of commodities in the $g$-th group from $f$. Then for each edge $e$ in

$f_g$, there is at least one commodity from the $g$-th group. Let them be $e_1, ..., e_h$, let their demands be $d_{e_1}, ..., d_{e_h}$ and their flow be $x_e^{e_1}, ..., x_e^{e_h}$. Let $X_e^g = \sum_{1 \le i \le h} x_e^{e_i}$, then we can deem $X_e^g$ as the number of *Unit* paths from the $g$-th group on edge $e$. Now we change the capacity of edge $e$ from $u_e$ to $\lceil X_e^g \rceil$ and we let the demand of each commodity in the $g$-th group be 1.

After we process all edges in $f_g$, we will find that the current graph for $f_g$ is a graph with a unit capacity on each edge. So we can run a minimum-cost unit flow algorithm on it and we will get a flow $f'_g$. Finally, we decompose $f'_g$ into edge-disjoint paths (they are edge-disjoint since all edges has a unit capacity). Each of these paths in $f'_g$ corresponding to a single path in $G$ for some commodity in the $g$-th group.

**Algorithm 5.1. Min-Cost-UFP($G, s, T$)**
**Input:**

```
  G: the directed graph G=(V,E) with $\{u_l, c_l\}_{l \in E}$;
  $s$ : the source node;
  T: the set of commodities, each commodity $i$ has a terminal $t_i$ and a demand $d_i$;
```

**Output:**

*A Minimum-cost Unsplittable Flow satisfying all demands with a congestion bound $3 + 2\sqrt{2}$;*

```
  1   Compute a minimum-cost splittable flow $f$ without bottleneck edges in $G$;
  2   Decompose $f$ into a set of sub-flows for different groups of commodities;
  3   Modify the edge capacities and commodity demands in each sub-flow;
  4   Compute a minimum-cost unit flow $f'_g$ for each group $g$;
  5   Decompose $f'_g$ into a set of paths in each group;
  6   Find the corresponding $s - t_i -$paths in $G$ for each commodity $i$;
  7   Route each commodity along its path.
```

**Lemma 5.1.** *Algorithm Min-Cost-UFP runs in polynomial-time.*

**Proof**: As we know, the minimum-cost flow problem can be solved in polynomial-time and a flow can be decomposed into a set of paths in polynomial-time. Furthermore, a Linear Program is also polynomial-time solvable. Thus we are sure that Algorithm Min-Cost-UFP can finish in polynomial-time. □

**Lemma 5.2.** *Algorithm Min-Cost-UFP computes a solution with cost of no more than that of the Minimum-cost Splittable Flow.*

**Proof**: Notice that a Minimum-cost Single-source Unsplittable Flow in $G$ is also a feasible solution of the second Linear Program, so we have that the flow $f$ computed in the $1-$st step will have a cost less than that of the optimal solution. For each group of commodities, we will route them respectively. Let we consider any group $g$: we find that $f_g$ is still a feasible solution for commodities in group $g$ after we modify the demands and edge capacities. Since the modified graph has unit capacity on each edge and unit demand for each commodity, by Theorem 5.1 we will find a unit flow with minimum cost in polynomial-time. Thus the cost will not increase for any group of commodities during the computing process. So the cost of the final solution will be bounded by that of the optimal solution. □

**Lemma 5.3.** *Algorithm Min-Cost-UFP computes a solution with congestion on each edge $(i, j)$ of no more than $(3 + 2\sqrt{2})u_{ij}$.*

**Proof**: Let us consider any given edge $e$ in $G$. Suppose it belongs to the flow $f_g$ for the commodity group $g$ and the capacity of $e$ used by $f_g$ is $u_e^g = X_e^g$. After we modify the capacity when we are computing $f'_g$, we have that the capacity of $e$ will be $\lceil X_e^g \rceil$. Thus there will be no more than $\lceil X_e^g \rceil$ paths for commodities in group $g$ and the real capacities used on $e$ will be no more than $\lceil X_e^g \rceil * d_{max}^g$, while the capacity of $e$ allotted $f_g$ will be more than $\lceil X_e^g \rceil * d_{min}^g - d_{min}^g$. So we have that the congestion cause by commodity group $g$ is no more than

$$cong(e, g) = \lceil X_e^g \rceil * d_{max}^g / (\lceil X_e^g \rceil * d_{min}^g - d_{min}^g) \leq d_{max}^g + u_e^g \frac{d_{max}^g}{d_{min}^g} = d_{max}^g + u_e^g / z. \quad (5.4)$$

Edge $e$ might have been used by more than one group of commodities. The extreme case will be used by all $y$ groups. In this case, we have that the congestion on $e$ will be:

$$\begin{aligned} cong(e) &= \sum_{1 \leq i \leq y} cong(e, i) \\ &= d_{max}^1 + u_e^1/z + \ldots + d_{max}^y + u_e^y/z \\ &= \sum_{1 \leq i \leq y} d_{max}^i + \sum_{1 \leq i \leq y} u_e^i/z \\ &\leq \frac{u_e}{z} \sum_{1 \leq i \leq y} z^i + u_e/z \\ &= u_e \frac{1}{z(1-z)} + u_e/z. \end{aligned} \quad (5.5)$$

The value of $u_e \frac{1}{z(1-z)} + u_e/z$ will be lower-bounded by $(3+2\sqrt{2})u_e$ when we let $z = \frac{1}{\sqrt{2}}$. Thus we can make sure that the solution computed by algorithm Min-Cost-UFP will have a congestion on each edge $(i, j)$ of no more than $(3 + 2\sqrt{2})u_{ij}$. □

**Theorem 5.2.** *Algorithm Min-Cost-UFP computes in polynomial-time an unsplittable flow whose cost is no more than that of the optimal solution and whose congestion on each edge $(i, j)$ is no more than $(3 + 2\sqrt{2})u_{ij}$.*

**Proof**: This theorem follows from the above three lemmas. □

Notice here that Algorithm Min-Cost-UFP differs from all previous algorithms by rounding the number of paths on an edge instead of the number of flows, which makes the computation much more convenient since we need only to round them to an integer and thus avoid the complex numeric processing steps. What's more, we do not need to compute the longest paths and remove the leftover flows at the very beginning, which also decreases the whole complexity.

## 5.4 Algorithms for Some Special Cases of MSUFP

### 5.4.1 On the case when $d_{max} \ll u_{min}$

As we mentioned above, it is a commonly accepted assumption that $d_{max} < u_{min}$. In the previous section we are dealing with the case when we do not have this restriction and subsequently the congestion in the computed solution is larger than that of the case with the assumption. In this section we will consider, on the contrary, the case when

$d_{max} \ll u_{min}$. For example, if $d_{max} \leq u_{min}/10$, can we compute a flow with a congestion better than $2 + 1/10$?

Skutella present a $(3, 1)-$approximation algorithm for the case with the assumption that $d_{max} < u_{min}$. More precisely, the Skutella Algorithm can compute a $(2u_e + d_{max}, 1)-$approximation solution in polynomial-time. This algorithm first compute a minimum-cost single-source splittable flow $f$, then it rounds the demand $d_i$ and any commodity $i$ to $d_{min} * 2^{\lfloor \log \frac{d_i}{d_{min}} \rfloor}$ by cancelling $d_i - d_{min} * 2^{\lfloor \log \frac{d_i}{d_{min}} \rfloor}$ units of $s - t_i$ flow in $f$ along the longest $s - t_i$ paths in $f$. After this it compute an unsplittable flow $f'$ with congestion $u_e + d_{max}$ on the rounded $f$. Then it decompose $f'$ into a set of paths in $G$ and route each commodity according to its corresponding path. Thus the final congestion will be $2u_e + d_{max}$ and the final cost will be within the optimal.

Now we see that the congestion $2u_e$ in $2u_e + d_{max}$ comes from the rounding procedure. Since we have to round a demand to almost half its original size when $d_i - d_{min} * 2^{\lfloor \log \frac{d_i}{d_{min}} \rfloor} \approx d_i/2$. So if $d_{max} \ll u_{min}$ and we want to further decrease the congestion, we can focus on improving the rounding process.

The Skutella Algorithm rounds demands by a scale of 2, here we will decrease this scale to $1 + z(0 \leq z < 2)$. Surprisingly, we find that we can apply the same rounding technique in the previous section.

**Algorithm 5.2. Min-Cap-UFP$(G, z, s, T)$**
**Input:**
*G: the directed graph G=(V,E) with $\{u_l, c_l\}_{l \in E}$;*
*$z$ : the scale number;*
*$s$ : the source node;*
*T: the set of commodities, each commodity $i$ has a terminal $t_i$ and a demand $d_i$;*
**Output:**
*A Minimum-cost Unsplittable Flow satisfying all demands with a congestion bound $\frac{1}{z(1-z)} d_{max} + \frac{1}{z} u_e$;*

```
1   Compute a minimum-cost splittable flow $f$ in $G$;
2   Divide all commodities into different groups by the scale $z$;
3   Decompose $f$ into a set of sub-flows for different groups of commodities;
4   Modify the edge capacities and commodity demands in each sub-flow;
5   Compute a minimum-cost unit flow $f'_g$ for each group $g$;
6   Decompose $f'_g$ into a set of paths in each group;
7   Find the corresponding $s - t_i-$paths in $G$ for each commodity $i$;
8   Route each commodity along its path.
```

We can see that the above algorithm is almost the same as Algorithm Min-Cost-UFP except that we compute a minimum-cost single-source splittable flow in the first step since we have the assumption that $d_{max} \ll u_{min}$ now.

**Theorem 5.3.** *Algorithm Min-Cap-UFP computes in polynomial-time an unsplittable flow whose cost is no more than that of the optimal solution and whose congestion on each edge $(i, j)$ is no more than $\frac{1}{z(1-z)} d_m ax + \frac{1}{z} u_e$.*

**Proof**: The polynomial-time computability and the optimal cost feature proof is the same as that for Algorithm Min-Cost-UFP. For the congestion on $e$, it will be:

$$\begin{aligned}
cong(e) &= \sum_{1\le i\le y} cong(e,i) \\
&= d^1_{max} + u^1_e/z + \ldots + d^y_{max} + u^y_e/z \\
&= \sum_{1\le i\le y} d^i_{max} + \sum_{1\le i\le y} u^i_e/z \\
&\le \frac{d_{max}}{z}\sum_{1\le i\le y} z^i + u_e/z \\
&= d_{max}\frac{1}{z(1-z)} + u_e/z.
\end{aligned} \tag{5.6}$$

Thus we prove the theorem. □

So given an instance of MSUFP, we can check the relationship between $d_{max}$ and $u_{min}$. If $d_{max} > u_{min}$, we can use Algorithm Min-Cost-UFP to compute a minimum-cost solution with congestion $3+2\sqrt{2}$. Else if $2u_e+d_{max} \le min_{z:0<z<1}\{d_{max}\frac{1}{z(1-z)}+u_e/z\}$, then we can use the Skutella Algorithm to compute a minimum-cost solution with congestion $2u_e + d_{max}$. If $d_{max} \ll u_{min}$, then we can use Algorithm Min-Cap-UFP to compute a minimum-cost solution with congestion $min_{z:0<z<1}\{d_{max}\frac{1}{z(1-z)} + u_e/z\}$.

### 5.4.2 On the case with demands from the set $\{p, 2p\}$

In [KS02] Kolliopoulos and Stein propose an algorithm for the case when all demands are from the set $\{p, 2p\}$ for some $p > 0$ while the minimum capacity in $G$ is $2p$. Their algorithm achieves a congestion rate of 3/2, but it does not take into account the cost of a flow. In this subsection we will present an algorithm which compute a $(3/2, 1)-$approximation algorithm for this special case.

**Algorithm 5.3. Min-2Demands-UFP**$(G, s, T)$
**Input:**

```
  G: the directed graph G=(V,E) with $\{u_l, c_l\}_{l\in E}$;
  s : the source node;
  T: the set of commodities, each commodity $i$ has a terminal $t_i$ and a demand $d_i$;
```

**Output:**

```
  A Minimum-cost Unsplittable Flow satisfying all demands with a congestion bound $3/2$;
  1  Compute a minimum-cost splittable $p-$integral flow $f$ in G;
  2  Decompose $f$ into $f_1$ and $f_2$ for commodities with demands $p$ and $2p$ respectively;
  3  Round the capacity on each edge $e$ in flow $f_2$ to $2p * \lceil f_2^e/2p \rceil$;
  4  Compute a minimum-cost unit $2p$ flow $f_2'$ in the modified graph of $f_2$;
  5  Decompose $f_1$ and $f_2'$ into a set of $s - t_i-$paths for all commodities;
  6  Route a $2p$ flow along its corresponding path in $f_2'$ for each commodity $i$ with $d_i = 2p$;
  7  Route a $p$ flow along its corresponding path in $f_1$ for remain commodities;
```

**Theorem 5.4.** *Given an UFP instance $(G, s, T)$ with demands from the set $\{p, 2p\}(p > 0)$, Algorithm Min-2Demands-UFP computes in polynomial-time an unsplittable flow whose cost is no more than that of the optimal solution and whose congestion on each edge $(i, j)$ is no more than $\frac{3}{2}u_e$.*

**Proof**: During the computing process, we only increase the capacity of any edge by at most $p$ in Step 3. While the minimum capacity in $G$ is no less than $2p$ by assumption. So the congestion rate is at most 3/2. For the cost, since $f_2$ is also a fractional solution in the modified graph for commodities with demand $2p$ while $f_2'$ is the minimum-cost $2p-$integral flow. Thus the cost of $f_2'$ will be no more than that of $f_2$. Finally, it is evident that Algorithm Min-2Demands-UFP will finish in polynomial-time. Thus the theorem can be proved. □

## 5.5 Approximation Algorithms for $k$-splittable Flow

The Minimum-Cost Single-Source $k$-Splittable Flow Problem (MSkFP) is a relaxation of MSUFP in which we allow the demand of each commodity to be split along at most $k$ paths. Baier, Köhler, and Skutella have designed a (2,1)-approximation algorithm for the Minimum-Cost $k$-Splittable $s$-$t$-Flow Problem in [BKS02]. Kolliopoulos considered the Minimum-Cost Single-Source 2-Splittable Flow Problem and proposed a (2,1)-approximation algorithm in [Koll04]. In this section we will follow the methods used by them and present two approximation algorithms for MSkFP. For a given $k$, our first algorithm will achieve a simultaneous $(\frac{1}{1-2^{-k}} \cdot \frac{3}{2}, 1)$-approximation for the congestion and cost while the second algorithm will find a bicriteria $(\frac{1}{1-2^{-(k-2^{\lfloor \log k \rfloor}+1+\lfloor \log k \rfloor)}} \cdot (1+\frac{1}{2^{\lfloor \log k \rfloor}}), 1)$-approximation. Notice that we use $k$ to denote the number of splittable paths instead of the number of commodities only in this section since this problem was orginally proposed as a $k$-splittable flow problem.

We adopt the same model as in [Koll04] and set $d_{max} = max_{1 \leq i \leq g} d_i$, $d_{min} = min_{1 \leq i \leq g} d_i$ and $u_{min} = min_{e \in E} u_e$. We assume that $d_{max} \leq u_{min} = 1$. The algorithm POWER-ALG proposed by Kolliopoulos and Stein [Koll04] will be used as a basic stone.

**Theorem 5.5.** *[Koll04] Given a SUFP instance where all demands are powers of* $1/2$ *and an inital fractional flow solution, the algorithm POWER-ALG will find in polynomial-time an unsplittable flow $f$ that violates the capacity of any edge by at most $d_{max} - d_{min}$ and whose cost is bounded by the cost of the initial fractional flow.* □

We first extend the method in [Koll04] to the case of single-source $k$-splittable ($k \geq 2$) flow. Let $floor_2(x)$ denote the largest number which is a power of $1/2$ and does not exceed $x$. The operator $\lfloor \cdot \rfloor_2$ is defined as follows:

$$\lfloor x \rfloor_2 = \begin{cases} 1/2 & if\ x = 1, \\ floor_2(x) & if\ 0 < x < 1, \\ 0 & if\ x = 0. \end{cases}$$

Now for any given instance $I$ we create a new UFP instance $I^k$ with no more than $k \cdot g$ commodities. For a commodity $i$ with demand $d_i$ in the original instance, we create a series of $k$ commodities in $I^k$ with demands of $\{d_i^1, ...d_i^h..., d_i^k\}$ $(1 \leq h \leq k)$ where $d_i^1 = \lfloor d_i \rfloor_2$ and $d_i^h = \lfloor d_i - \sum_{1 \leq j < h} d_i^j \rfloor_2$. Then we remove those commodities with a zero demand and run the POWER-ALG on $I^k$ to obtain a flow $f$.

**Lemma 5.4.** *Given the UFP instance $I^k$ with initial fractional solution $f_0^k$ one can find an unsplittable flow $f$ which (i) violates the capacity of any edge by at most $\lfloor d_{max}^1 \rfloor_2$ and (ii) whose cost is bounded by the cost of the initial fractional flow $f_0^k$. Flow $f$ corresponds to a $k$-splittable flow for instance $I$ which routes $\sum_{1 \leq j \leq k} d_i^j$ units of flow for commodity $i (i = 1, ..., g)$.*

**Proof**: Since $d_i \leq 1$ $(i = 1, ..., g)$ and function $floor_2(\cdot)$ is a non-decreasing function, we have that $\lfloor d^1_{max} \rfloor_2$ is the largest demand in instance $I^k$, by Theorem 5.5 it follows. $\Box$

Since we have routed $\sum_{1 \leq j \leq k} d^j_i$ units of flow for commodity $i$, the remain demand of commodity $i$ will be $d_i - \sum_{1 \leq j \leq k} d^j_i < 2^{-k} d_i$. Thus we have $\sum_{1 \leq j \leq k} d^j_i > d_i - 2^{-k} d_i \Rightarrow \sum_{1 \leq j \leq k} d^j_i / d_i > 1 - 2^{-k}$. If we scale the flow on each of the at most $k$ $s - t_i$ paths used in $f$ by the same amount $\lambda_i = \frac{d_i}{\sum_{1 \leq j \leq k} d^j_i} \in (1, \frac{1}{1-2^{-k}})$, then we obtain a $k$-splittable flow $f'$ which satisfies all demands $d_i$ and satisfies $f'_e \leq \lambda_i u_e + \lambda_i \lfloor d^1_{max} \rfloor_2$ for all edges $e \in E$. Since $1 \leq \lambda_i \leq \frac{1}{1-2^{-k}}$ and $\lfloor d^1_{max} \rfloor_2 \leq 1/2 < u_e/2$, we have that $f'_e \leq \frac{1}{1-2^{-k}}(u_e + 1/2) \leq \frac{1}{1-2^{-k}} \frac{3}{2} u_e$.

Next we use the same method proposed by Skutella [Sku02] and upgraded in [Koll04] to bound the cost to be no more than the cost of an optimal fractional solution. Then we can prove the following theorem.

**Theorem 5.6.** *Given a SUFP instance $I$ with initial fractional solution $f_0$ one can find in polynomial-time a $k$-splittable flow $f'$ such that (i) $f'$ satisfies all demands $d_i$ (i=1,...,g) (ii) $f'_e \leq \frac{1}{1-2^{-k}}(u_e + 1/2)$ for all $e \in E$ and (iii) the cost of $f'$ is bounded by the cost of the initial fractional flow $f_0$.* $\Box$

**Corollary 5.1.** *Given a SUFP instance $I$ one can find in polynomial-time a $k$-splittable flow solution that achieves a simultaneous $(\frac{1}{1-2^{-k}} \cdot \frac{3}{2}, 1)$-approximation for the congestion and cost.*$\Box$

We can find that the approximate ratio $\frac{1}{1-2^{-k}} \cdot \frac{3}{2}$ will approach $3/2$ when we $k$ increases. But it will be no less than $\frac{3}{2}$ for any $k \geq 2$. However, we can improve this approximate ratio by using a new operator $\lfloor \cdot \rfloor^k_2$ which is defined as follows:

$$\lfloor x \rfloor^k_2 = \begin{cases} \frac{1}{2^{\lfloor \log k \rfloor}} & if\ x \geq \frac{1}{2^{\lfloor \log k \rfloor}}, \\ floor_2(x) & if\ 0 < x < \frac{1}{2^{\lfloor \log k \rfloor}}, \\ 0 & if\ x = 0. \end{cases}$$

All other parts of the algorithm remain the same, thus the cost will be also bounded by that of the optimal fractional solution. Now the task remains is to analyze the approximate ratio for this new algorithm. For a commodity $i$ with demand $d_i$ in the original instance, we create a series of $k$ commodities in $I^k$ with demands of $\{d^1_i, ...d^h_i ..., d^k_i\}$ $(1 \leq h \leq k)$ where $d^1_i = \lfloor d_i \rfloor^k_2$ and $d^h_i = \lfloor d_i - \sum_{1 \leq j < h} d^j_i \rfloor^k_2$. Let $d^\Delta_i = d_i - \sum_{1 \leq j < h} d^j_i$, then $d^\Delta_i < d^k_i$. Notice that $d^k_i \leq d_i \cdot 2^{-(k - 2^{\lfloor \log k \rfloor} + 1 + \lfloor \log k \rfloor)}$ and this maximum value happens only if $d_i$ is no less than $\frac{1}{2^{\lfloor \log k \rfloor}} \cdot (2^{\lfloor \log k \rfloor} - 1) + \sum_{h = \lfloor \log k \rfloor + 1}^{k - 2^{\lfloor \log k \rfloor} + 1 + \lfloor \log k \rfloor} \frac{1}{2^h}$.

Thus if we scale the flow on each of the $s - t_i$ paths used in $f$ by the same amount $\lambda_i = \frac{d_i}{d_i - d^\Delta_i} < \frac{2^{(k - 2^{\lfloor \log k \rfloor} + 1 + \lfloor \log k \rfloor)}}{2^{(k - 2^{\lfloor \log k \rfloor} + 1 + \lfloor \log k \rfloor)} - 1}$, then we obtain a $k$-splittable flow $f'$ which satisfies all demands $d_i$.

Let $d_{max}$ be largest demand in $I$, then $d^1_{max}$ will be the largest demand in $I^k$ and $d^1_{max} \leq \frac{1}{2^{\lfloor \log k \rfloor}}$. Thus the final solution will satisfy $f'_e \leq \lambda_i u_e + \lambda_i \lfloor d^1_{max} \rfloor_2 \leq \frac{1}{1 - 2^{-(k - 2^{\lfloor \log k \rfloor} + 1 + \lfloor \log k \rfloor)}}(u_e + \frac{1}{2^{\lfloor \log k \rfloor}})$ for all edges $e \in E$.

**Theorem 5.7.** *Given a SUFP instance $I$ with initial fractional solution $f_0$ one can find in polynomial-time a $k$-splittable flow $f'$ such that (i) $f'$ satisfies all demands $d_i$ (i=1,...,g) (ii) $f'_e \leq \frac{1}{1 - 2^{-(k - 2^{\lfloor \log k \rfloor} + 1 + \lfloor \log k \rfloor)}}(u_e + \frac{1}{2^{\lfloor \log k \rfloor}})$ for all $e \in E$ and (iii) the cost of $f'$ is bounded by the cost of the initial fractional flow $f_0$.* $\Box$

**Corollary 5.2.** *Given a SUFP instance $I$ one can find in polynomial-time a $k$-splittable flow solution that achieves a simultaneous $(\frac{1}{1-2^{-(k-2^{\lfloor \log k \rfloor}+1+\lfloor \log k \rfloor)}} \cdot (1+\frac{1}{2^{\lfloor \log k \rfloor}}), 1)$-approximation for the congestion and cost.* □

Notice that for $k = 3$ we have $\frac{1}{1-2^{-(k-2^{\lfloor \log k \rfloor}+1+\lfloor \log k \rfloor)}} \cdot (1+\frac{1}{2^{\lfloor \log k \rfloor}}) = \frac{1}{1-2^{-(3-2+1+1)}} \cdot (1+\frac{1}{2}) = \frac{12}{7}$. At this special case, the improved algorithm will return a same solution as the original one. Here we can also bound the maximal demand in $I^3$ to be no more than $1/4$, then the scale ratio will be no more than $4/3$ for each flow and the final solution will be a $(5/3, 1)$-approximation for the congestion and cost.

For $k \geq 4$, since both $\frac{1}{1-2^{-(k-2^{\lfloor \log k \rfloor}+1+\lfloor \log k \rfloor)}}$ and $(1+\frac{1}{2^{\lfloor \log k \rfloor}})$ are non-increasing functions, we have $\frac{1}{1-2^{-(k-2^{\lfloor \log k \rfloor}+1+\lfloor \log k \rfloor)}} \cdot (1+\frac{1}{2^{\lfloor \log k \rfloor}}) \leq \frac{1}{1-2^{-(1+\lfloor \log k \rfloor)}} \cdot (1+\frac{1}{2^{\lfloor \log k \rfloor}}) = \frac{2^{\lfloor \log k \rfloor}+1}{2^{\lfloor \log k \rfloor}-1/2} \leq \frac{4+1}{4-1/2} = \frac{10}{7} < \frac{3}{2}$. Actually $\frac{2^{\lfloor \log k \rfloor}+1}{2^{\lfloor \log k \rfloor}-1/2}$ will approach 1 when $k$ increases.

## 5.6 Applications in QoS Video-on-Demand Routing

The edge/node-disjoint problem and the unsplittable flow problem have lots of application in huge-volume data transfer applications such as video conference and video-on-demand. For such an application, usually we need to consider not only the network congestion but also the reliability.

**Problem**. *Minimum Distortion 2-Path Routing Problem* (**MD2PR**).

**Instance**: Directed graph $G = (V, E)$, weight $c(e) \in R^+$, $d(e) \in R^+$ and $l(e) \in R^+$ for each $e \in E$, where $c(e)$ is its bandwidth capacity, $d(e)$ is its delay and $l(e)$ is its packet loss probability. Without loss of generality, we assume that $c(e), d(e)$ are integers and $0 \leq l(e) < 1$. Since the delay of an edge in real networks is not a fixed value, we assume that the delay value of $e$ is exponentially distributed with mean $d(e)$.

Suppose we want to transfer a video from $s$ to $t$. The video signal consists of a sequence of frames. Let $F_1$ denote the coded bitstream of encoding the even-indexed frames and $F_2$ the coded bitstream of encoding the odd-indexed frames. Denote their bit rates by $r_1$ and $r_2$ respectively.

Assume that there are two paths $P_1$ and $P_2$ connecting the sender to receiver. We calculate the end-to-end delay of $P_i (i = 1, 2)$ as $D_i = \sum_{e, e \in P_i} d(e)$.

For any edge $e$ with capacity $c_e$ in $P_1$ and $P_2$, we calculate the flow $R_i^e$ of $P_i$ on $e$ as follows:

$$R_i^e = \begin{cases} min\{r_i, c_e\}, & e \text{ appears only in } P_i, \\ \frac{r_i}{r_1+r_2} min\{r_1 + r_2, c_e\}, & e \text{ appears in both } P_1 \text{and} P_2. \end{cases}$$

Then we can get the flow on $P_i$ as $R_i = min_{e, e \in P_i}\{R_i^e\}$. For the end-to-end packet loss rate $L_i$ of $P_i$, we use the following formula:

$$L_i = 1 - Pr(D_i \leq DeadLine) * \prod_{e, e \in P} (1 - l(e)).$$

Here $Pr(D_i \leq DeadLine)$ is the probability that the data sent along path $P_i$ can reach the sink $t$ before the *Deadline* bound, because the terminal application might skip packet if the delay is too large, which is another kind of packet loss. This probability can be calculated under the assumption that any edge delay value is exponentially distributed with mean $d(e)$.

The final distortions corresponding to transmitting two descriptions are as follows:

$$Dis(P_i) = \alpha_i * (L_i + 1 - \frac{R_i}{r_i}) + \beta_i.$$

Where $\alpha_i$ and $\beta_i$ can be determined by a curve fitting tool, thus they are known a priori.

**Question 1**: Given a positive bound $B$, are there two loop-free paths $P_1$ and $P_2$ in $G$ such that their total distortion $Dis(P_1) + Dis(P_2)$ does not exceed $B$?

**Question 2**: Given a positive bound $B$, are there two loop-free paths $P_1$ and $P_2$ in $G$ such that $max\{Dis(P_1), Dis(P_2)\}$ does not exceed $B$?

**Analysis**: Since usually Question 2 is more difficult than Question 1, we consider Question 1 first.

This problem can be classified into the Unsplittable Flow Problem family. The special point is that we have only one sink and two demands, but we need to consider more constraints. Beside a capacity $c(e)$, each edge $e$ has a delay $d(e)$ and a loss rate $l(e)$. All three parameters will inflect the performance of the final solution. Since $Dis(P_i) = \alpha_i * (L_i + 1 - \frac{R_i}{r_i}) + \beta_i$, we can see that $c(e)$ may contribute to the part $(1 - \frac{R_i}{r_i})$ when we need to deliberately drop packets due to insufficient bandwidth, while $d(e)$ and $l(e)$ will decide the unconsciously end-to-end packet loss rate $L_i$.

Let us simplify the model to make it easier for the first step. We can assume that $\alpha_1 = \alpha_2$ and $\beta_1 = \beta_2$, by this we reduce Question 1 to the problem of minimize $(L_1 + L_2 - \frac{R_1}{r_1} - \frac{R_2}{r_2})$. Now we see that both $d(e)$ and $l(e)$ are multiplicative factors while $c(e)$ is a bottleneck factor. As we know, how much water a barrel can contain doesn't depend on the longest board but on the shortest one. Thus we should give the flow capacity a priority.

If the capacity requirement can be satisfied ( i.e., $\frac{R_1}{r_1} + \frac{R_2}{r_2} = 2$), then we can further minimize $L_1 + L_2$. Since usually the packet loss rate for a given edge is very small and comparatively stable, we need to pay more attention to the end-to-end delay of a path, which might vibrate by a large extent. So we can put the capacity requirement as the first constraint and add a end-to-end delay bound to the final solution, then we find the two paths which minimize the total packet loss rate under these two constraints. Based on this heuristics, we can design several algorithms with different complexity.

**Algorithms:** We first assume $r_1 = r_2$ so that we can deem them as a unit flow (this is reasonable since here the encoded bit rates of the odd/even descriptions are equal), we further assume that both the capacity requirement and the delay constraint can be easily satisfied. Now our target is to minimize the total packet loss rate under these two constraints. Since $l(e)$ is a multiplicative factor, we transfer it into an additive factor for convenient of computation. To minimize $1 - \prod_{e,e\in P}(1 - l(e))$ is to maximize $\prod_{e,e\in P}(1 - l(e))$, which is equal to maximize $\sum_{e,e\in P}\log(1 - l(e))$ since $\prod_{e,e\in P}(1-l(e)) = 2^{\log\prod_{e,e\in P}(1-l(e))} = 2^{\sum_{e,e\in P}\log(1-l(e))}$. Since $\log(1-l(e)) < 0$, we can minimize $\sum_{e,e\in P}(-\log(1-l(e))) = \sum_{e,e\in P}\log(\frac{1}{1-l(e)})$ instead of maximize $\sum_{e,e\in P}\log(1-l(e))$.

Thus we will assign a new weight $l'(e) = \log(\frac{1}{1-l(e)})$ to each edge $e$.

To restrict the probability that the data sent along path $P_i$ can reach the sink $t$ before the *Deadline*, we can calculate a delay bound $D$ for the mean end-to-end delay based on the value of *Deadline* and the distribution of edge delays. If both paths have a total delay of no more than $D$, then we are sure that packet skip rate at the terminal will be small enough.

**Algorithm 5.4. NaiveRouting**($G, s, t, D, r$)
**Input:**

```
  G: the directed graph G=(V,E) with $\{d(e), c(e), l(e)\}_{e\in E}$;
  s : source node;
  t : destination node;
  D: the delay constraint;
  r : the flow bandwidth;
```

**Output:**

```
  Two Paths from s to t;

 1   Calculate $l'(e) = \log(\frac{1}{1-l(e)})$ for each edge e and the delay bound D ;
 2   $c(e) \leftarrow c(e)/r$ ;
 3   Use Algorithm RSP to compute path $P_1$ with total delay within D
     and total loss rate minimized;
 4   Construct the residual network $G(P_1)$ of G imposed by $P_1$:
 5      for each link $e \in P_1$ do
 6          $c(e) \leftarrow c(e) - 1$;
 7   Use Algorithm RSP to compute path $P_2$ in $G(P_1)$ with total delay within D;
 8   if we successfully find $P_1$ and $P_2$ then
 9      return $(P_1, P_2)$ ;
10   else return FAIL;
```

Usually we will reverse the direction of the edges in an established flow in the residual graph when we are computing the maximum flow in a given graph. But here we simply delete the occupied bandwidth from the capacity of each edge in the established path. The reason to do this is because we focus on finding two paths that satisfy the delay bound $D$. If we reverse the direction of all edges in $P_1$, the following $P_2$ may use some of those reversed edges. Thus although the final solution will also be a $2r$ flow which contains two $s$-$t$ paths, the end-to-end delay $D_1$ and $D_2$ might be very different and $max\{D_1, D_2\}$ might be around $2D$. Then the path with large delay will be useless since all packets delivered along it might be discarded.

The shortcoming of the Naive Routing algorithm is that it cannot guarantee that we can always find a solution if it does exist. Thus we propose the following algorithm which use negative cycle cancelling method.

**Algorithm 5.5. NCCRouting**($G, s, t, D, r$)
**Input:**

```
  G: the directed graph G=(V,E) with $\{d(e), c(e), l(e)\}_{e\in E}$;
  s : source node;
  t : destination node;
  D: the delay constraint;
  r : the flow bandwidth;
Output:
  Two Paths from $s$ to $t$;

1   Calculate $l'(e) = \log(\frac{1}{1-l(e)})$ for each edge $e$ and the delay bound $D$ ;
2   $c(e) \leftarrow c(e)/r$ ;
3   Use Algorithm RSP to compute path $P_1$ with total delay within $D$
    and total loss rate minimized;
5   Construct the residual network $G(P_1)$ of $G$ imposed by $P_1$:
6       Add to $G(P_1)$ each link $l$ in $G$ which does not appear in $P_1$;
7       for each link $(v,u) \in P_1$ do
8           if $(u,v)$ is not in $G(P_1)$ then
9               Add a link $(u,v))$ to $G(P_1)$ with $d((u,v)) = 0$ and $c((u,v)) = 0$;
10          $c((u,v))) \leftarrow c((u,v))) + 1$;
11  Use Algorithm RSP to compute path $P_2$ in $G(P_1)$ with total delay within $D$;
12  Decompose the flow $P_1 \cup P_2$ into two paths $P_1$ and $P_2$ with $D_1 < D_2$;
13  Run the cycle cancelling algorithm on $P_2$ until $D_2 \leq D$;
14  if we successfully find $P_1$ and $P_2$ then
15      return $(P_1, P_2)$ ;
16  else return FAIL;
```

## 5.7 Concluding Remarks

In this chapter we consider the unsplittable flow problem. We first briefly survey the research topic of Disjoint paths, then we extend it the single-source unsplittable flow problem. We first develop an efficient algorithm for compute a minimum-cost single-source unsplittable flow in a graph with arbitrary edge capacity. Our algorithm can compute a $(3+2\sqrt{2}, 1)-$approximation solution in polynomial-time, and the implementation is more convenient than previous algorithms. We extend our algorithm to deal with some special cases such as the situation when the largest demand in $G$ is far less than the minimum capacity in it. After that we study the minimum-cost single-source $k-$splittable flow problem and we present an algorithm with best performance guarantee than previous algorithms. Finally we present some applications of the single-source unsplittable flow problem on the huge-volume video transfer in large networks.

# Chapter 6

# Algorithms for Computing Inner-node Weighted Minimum Spanning Trees

## 6.1 Introduction

Given a graph $G = (V, E)$ with nonnegative costs assigned to its edges, the Minimum Spanning Tree (MST) problem is to find the spanning tree of $G$ with minimum cost. This problem is in $P$ and many polynomial-time algorithms have been designed. A natural generalization of this problem is to assign nonnegative costs to both nodes and edges, and we only count the cost of the edges and inner nodes (non-leaf nodes) in the final solution. This problem was first proposed in [FGLTW05], by the name of *minimum spanning tree with inner nodes cost problem (MSTI)*. Since this paper follows the line of [GK96, GK99, KR95], we hereby rename the problem as the *Inner-node Weighted Minimum Spanning Tree Problem (IWMST)* according to the definitions in [GK96].

The IWMST problem has lots of applications in real world. For example, we need to calculate the cost of all fibres and intermediate optical switches when we are building a fibre networks while the terminal users can connect their computers to the ports directly. The construction of other electronic networks and pipe networks can be similarly modelled.

In [FGLTW05], the authors applied the method used in [KR95] to IWMST and developed a $2 \ln n$-approximation algorithm. However, some of the details and proofs in it are rather elusive. In this chapter, we study the model of IWMST and some other related problems and we first present a general framework for developing poly-logarithmic approximation algorithms in Section 3, this framework aims to find a $\frac{k}{k-1} \ln n$-approximation Algorithm for the IWMST problem. Based on this framework, we further design two polynomial-time approximation algorithms. The first one is presented in Section 4, it can find a $2 \ln n$-approximation solution in $O(mn \log n)$ time. The second one proposed in Section 5 can compute a $1.5 \ln n$-approximation solution in $O(n^2\Delta^6)$ time. After this we consider some fault-tolerant variants of IWMST in Section 6. In Section 7we also show how we can use the methods for other related problems to solve the IWMST problem. Finally we conclude the paper in Section 8.

## 6.2 Preliminaries

**Problem** . *Inner-node Weighted Minimum Spanning Tree Problem* (**IWMST**).
**Instance**: Undirected graph $G = (V, E)$, weight $c(e) \in R^+$ for each $e \in E$ and weight $w(v) \in R^+$ for each $v \in V$, a positive bound $B$.
**Question**: Is there a spanning tree $T$ for $G$ such that the sum of the weights of the edges in $T$ and the weights of the inner (non-leaf) nodes in $T$ does not exceed $B$?

Although this problem was proposed as a variant of the Minimum Spanning Tree problem, the real difficulty lies in that it is also a generalization of the connected dominating set problem (CDS). If we set $c(e) = 0$ for each $e \in E$ and $w(v) = 1$ for each $v \in V$, then it is exactly the classical CDS problem. If we set $c(e) = 0$ for each $e \in E$ but preserve $w(v)$ for each $v \in V$, then it becomes the node weighted connected dominating set problem (NWCDS) [GK96]. However, IWMST is different from the edge weighted connected dominating set problem by including the leaf-edge costs (EWCDS). So we can expect a poly-logarithmic approximation algorithm for IWMST although EWCDS was proved to be as difficult as the set TSP problem in [GK96].

## 6.3 A $\frac{k}{k-1} \ln n$-approximation Algorithm Framework for IWMST

In this section we generalize the method used in [GK96, GK99, KR95] and propose a framework of developing poly-logarithmic approximation algorithms for the IWMST problem. Starting from the node set, we use a greedy policy to connect a set of isolated nodes which is "optimal" in each iteration according to some heuristics, thus by less than $n$ iterations we can connect all nodes in the graph. To find an "optimal" set, we first define the following structure:

**Definition 4**: A $k$-structure in $G$ is a tree structure subgraph of $G$ with at least $k$ nodes and satisfies some certain constraints.

Since we want find a solution for the IWMST problem here, we will adopt the minimum average cost as the heuristic to be applied on a $k$-structure. The average cost of a given $k$-structure is its total cost divided by the number of nodes (not necessary the nodes in the original graph) inside it. To calculate the total cost of a given $k$-structure, we will add up the cost of all edges and all inner-tree nodes (which are not paid before). If we want to "contract" a $k$-structure, we remove the cost of all these unpaid edges and nodes in the original graph, by which we say that these edges and nodes are "paid" by this $k$-structure.

We can use three different colors $\{white, gray, black\}$ to differentiate whether a node is paid or not. Originally all nodes and edges of the graph are in *white* color. When we want to contract a $k$-structure, we alter the color of all inner nodes and edges paid by this $k$-structure to *black* while those leaf nodes in *white* will be changed into *gray*. Thus a contracted $k$-structure has a *black* backbone whose nodes and edges are paid (by this $k$-structure and all previous contracted $k$-structures contained in it).

To present our algorithm, we need to propose the following two assumptions in advance:

**Assumption 1**: Any solution to the IWMST problem can be decomposed into a set of $k$-structures without increasing the cost.

**Assumption 2**: The minimum average cost $k$-structure located at any node in a given graph can be computed in polynomial time.

Based on the above two assumptions, we present the main framework in the following figure.

**Algorithm 6.1. Framework-IWMST($G$)**
**Input:**
*The undirected graph $G = (V, E)$ with $\{w_v\}_{v\in V}$ and $\{c_e\}_{e\in E}$;*
**Output:**
*A spanning tree $T(V, E')$ with $\sum_{e\in E'} c_e + \sum_{v,|\sum_{(u,v)\in E'} 1|>1} w_v \leq \frac{k}{k-1} \ln nOPT$ ;*

```
1  while there are more than $k$ node in $G$
2      Find a $k$-structure in $G$ with minimum ratio;
3      Contract the above $k$-structure to a single super-node;
4      Update $G$;
5  Connect the remain nodes optimally;
6  Unwrap all the super-node in the solution;
7  Connect each leaf-node to its nearest neighbor which is an inner tree node;
8  Return back the final spanning tree.
```

**Theorem 6.1.** *If the $k$-structure in line 2 satisfies Assumption1 and Assumption2, then the greedy algorithm Framework-IWMST will find a solution of cost at most $\frac{k}{k-1} \ln n$ times the optimal.*

**Proof**: Suppose that tree $T^*$ is an optimal solution. Let $t_1, t_2, ..., t_i$ be the set of $k$-structures selected in the first $i$ iterations. Denote $G_i$ as the updated graph and $T_i^*$ as the updated $T^*$ after $t_1, t_2, ..., t_i$ are contracted, then it follows that $|T_i^*| = |T_{i-1}^*| - (|t_i| - 1)$ where $|T_i^*|$ is the number of nodes in $T_i^*$. Start from $G = G_0$, if we contract $t_1$ into a single super-node and remove the weight of all nodes and edges which have paid to $t_1$, then some nodes in $T^*$ may be connected together without increasing the weight of $T^*$. We can run a minimum spanning tree algorithm in this updated $T^*$ and get another tree $T_1^*$ which spans $G_1$. Thus the weight of $T_1^*$ is no more than that of $T^*$. By this way we can get that $OPT = w(T^*) \geq w(T_1^*) \geq w(T_2^*) .. \geq w(T_i^*)$.

Now let's consider $G_{i-1}$ in the $i$-th iteration. According to the assumption, any solution to the IWMST problem can be decomposed into a set of $k$-structures without increasing the cost. Thus $T_{i-1}^*$ in $G_{i-1}$ could also be decomposed into a set of $k$-structures $K_1, K_2, ..., K_j$, let the number of nodes in each of them be $|K_1|, |K_2|, ..., |K_j|$. Since $t_i$ is the $k$-structure with the minimum ratio in $G_{i-1}$, we have $\frac{w(t_i)}{|t_i|} \leq \frac{w(K_r)}{|K_r|}, 1 \leq r \leq j$. Thus

$$\frac{w(T_i)}{|t_i|} \sum_{1\leq r\leq j} |K_r| \leq \sum_{1\leq r\leq j} w(K_r) \;\Rightarrow\; \frac{w(t_i)}{|t_i|}|T_{i-1}^*| = \frac{w(T_i)}{|t_i|} \sum_{1\leq r\leq j} |K_r| \leq \sum_{1\leq r\leq j} w(K_r) \leq w(T^*)$$

By the definition of a $k$-structure, we have $|t_i| \geq k$ which implies that $|t_i| - 1 \geq \frac{k-1}{k}|t_i|$. Since $|T_i^*| = |T_{i-1}^*| - (|t_i| - 1)$, it follows that

$$|T_i^*| \;=\; |T_{i-1}^*| - (|t_i| - 1) \;\leq\; |T_{i-1}^*| - \frac{k-1}{k}|t_i| \;\leq\; |T_{i-1}^*| - \frac{k-1}{k}\frac{w(t_i)}{w(T^*)}|T_{i-1}^*|$$

$$\Rightarrow\; \frac{|T_i^*|}{|T_{i-1}^*|} \;\leq\; 1 - \frac{k-1}{k}\frac{w(t_i)}{w(T^*)} \;\Rightarrow\; \ln(\frac{|T_i^*|}{|T_{i-1}^*|}) \;\leq\; \ln(1 - \frac{k-1}{k}\frac{w(t_i)}{w(T^*)}) \;\leq\; -\frac{k-1}{k}\frac{w(t_i)}{w(T^*)}$$

The last step uses the approximation $\ln(1+x) \le x$. Now suppose that the algorithm finish in $z$ iterations, summing over all $t_1, t_2, ..., t_z$ we will have:

$$\ln\left(\frac{|T^*_{i-1}|}{|T^*_i|}\right) \ge \frac{k-1}{k}\frac{w(t_i)}{w(T^*)} \Rightarrow \sum_{1\le i\le z} \ln\left(\frac{|T^*_{i-1}|}{|T^*_i|}\right) \ge \sum_{1\le i\le z} \frac{k-1}{k}\frac{w(t_i)}{w(T^*)}$$

$$\Rightarrow \ln\left(\frac{|T^*_0|}{|T^*_z|}\right) = \ln n \ge \frac{k-1}{w(T^*)k}\sum_{1\le i\le z} w(t_i) \Rightarrow \sum_{1\le i\le z} w(t_i) \le \frac{k}{k-1}\ln n \cdot w(T^*).$$

□

With the above $\frac{k}{k-1}\ln n$-approximation algorithm framework for IWMST, we reduce the job of designing a $\frac{k}{k-1}\ln n$-approximation Algorithm to the job of finding a $k$-structure which satisfies Assumption1 and Assumption2. In the following two sections, we will follow this path and present two approximate algorithms which can achieve a factor of $2\ln n$, and $1.5\ln n$ respectively.

## 6.4 A $2\ln n$-approximation Algorithm for IWMST

The simplest structure which includes at least 2 nodes is an edge. We can use this when we are computing a minimum spanning tree, which is exactly the idea of the famous Kruskal's Algorithm for MST [Kru56]. But here we have to take into account the cost of all inner-tree nodes, which looks similar to a node weighted and edge weighted connected dominating set with the leaf edge cost. Thus we need to count in at least the cost of one node in the edge, unfortunately sometimes a tree cannot be decomposed into a set of disjoint edges without increasing the total cost. For example, a tree with a center node and $n-1$ leaf nodes cannot be decomposed since for each edge we will count the center node once.

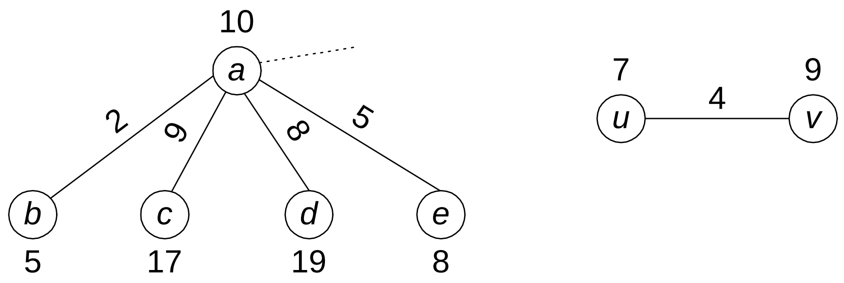


Figure 6.1: In the optimal solution, the subtree roots at $a$ has a cost of 34. But during the calculation, if we decompose it into 4 edges, the total cost counted will be 64. Else if we only assign the cost of node $a$ to edge $ab$, then the cost of of $ae$ will be 5, which is smaller than the current optimal edge $uv$.

The reason why we cannot use an edge as a 2-structure here lies in that we do not know how to distribute the cost of an inner node to its children if it has no less than 2 children. To bypass this, we choose the $2^+$-star as a 2-structure.

**Definition 5**: A $k^+$-star is a bipartite graph of the form $K_{1,n}$ with $n \ge k-1$.

We can see that an edge is the minimum $2^+$-star.(Figure 6.1)

**Lemma 6.1.** *Any solution to the IWMST problem with no less than* 2 *nodes can be decomposed into a set of* $2^+$*-stars without increasing the cost.*

**Proof**: We can prove this lemma by induction on the number of nodes $n$ in the tree. Start from $n = 2$, it is evident to see that this is a $2^+$-star whose cost can be optimally computed (Figure 6.2.(a)). Suppose the lemma is correct for $2 \leq n \leq i$ and consider a tree with $n = i + 1$ nodes (Figure 6.2.(b)). Set an arbitrary leaf $u$ as the root, then randomly select an inner node $v$ and check whether it has an inner node as a child. If all its children are leaves, we choose $v$ as the center of a $2^+$-star and contract it. Else we go down the tree from one of its inner node child until we reach an inner node who has only leaf children and contract the $2^+$-star located there. Thus we can reduce the number of nodes in the tree by at least one and then we can apply the induction hypothesis to further decompose the remain parts.

Now let's check the cost of the original tree and the decomposed $2^+$-star sets. Since the cost of a solution to the IWMST problem will only count the weight of all edges and inner nodes in the tree, the weight of a leaf node can be neglected. Consider the tree in Figure 6.2.(c). Suppose we will contract the $2^+$-star (sub-tree $T_a$) located at inner node $a$. Before the decomposition, the cost of the tree is $c(T_1) + 6 + 8 + 2 + 8 + 5 = c(T_1) + 29$. While the cost after the decomposition is $c(T_1) + 6 + c(T_a) = c(T_1) + 29$, which is the same as that before the decomposition.

By the above analysis, we can conclude that the decomposed $2^+$-star set will cover all nodes in the solution tree without increasing the cost. $\square$

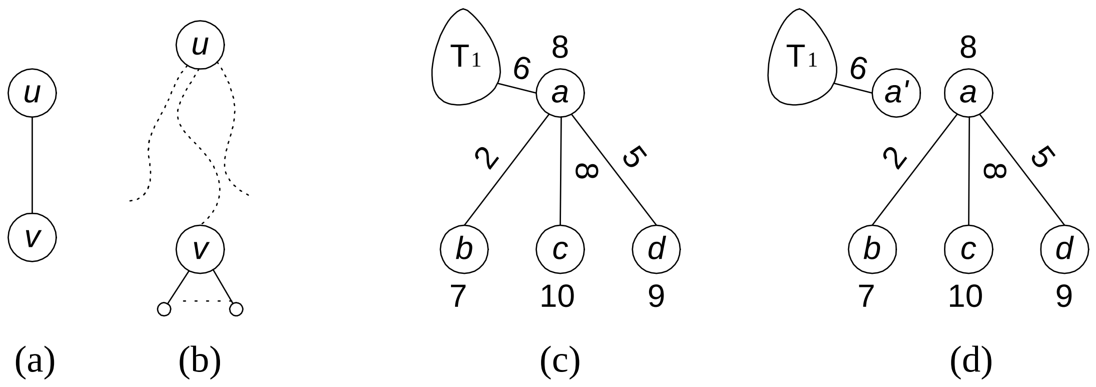


Figure 6.2: (a) A tree with 2 nodes is a trivial $2^+$-star. (b) An inner node $v$ which has only leaf node children can be contracted as a $2^+$-star. (c) The image of the tree before the decomposition of the $2^+$-star located at node $a$. (d) The remain part and the $2^+$-star sub-tree after the decomposition.

**Algorithm 6.2. Min$2^+$Star($G'$)**
**Input:**
*The undirected graph $G' = (V', E')$, each node in $V'$ is a sub-tree in $G$;*
**Output:**
*A $2^+$-Star with the minimum average cost in $G'$;*

```
1   Set the current minimum $2^+$-Star as an empty set with an upper-bound average cost;
2   for each node $v \in V'$ do
3     if $v$ is a white node:
4       Compute the cost $cost_{(v,u)}$ of each unpaid edge $(v,u) \in E'$
5         case 1, $u$ is a white node: $cost_{(v,u)} = c_{(v,u)}$;
6         case 2, $u$ is a super node:
7           if the incoming node $h \in V$ of $(v,u)$ in $u$ is black, then $cost_{(v,u)} = c_{(v,h)}$;
```

```
8           else $cost_{(v,u)} = c_{(v,h)} + w(h)$;
9       For all edges connecting to the same super node,
            choose the one with minimum cost;
10      Sort all neighbor nodes according to the connecting
            costs by non-decreasing order $N_v^1, .., N_v^j$;
11      Let $i, 1 \le i \le j$ be the minimum index that $\frac{w(v)+\sum_{1\le k\le i} cost_{(v,N_v^k)}}{i+1} < \frac{w(v)+\sum_{1\le k\le i+1} cost_{(v,N_v^k)}}{i+2}$;
12      Set $\{v, N_v^1, .., N_v^i\}$ and their edges as the minimum $2^+$-Star located at $v$;
13    else if  $v$ is a super node:
14      Compute the cost $cost_{(v,u)}$ of each unpaid edge $(v,u) \in E'$ whose outgoing
          node $g$ is black, using the same method as that for the $white$ node above,
          divide this cost by $2$ and set it as its average connecting cost;
15      For all edges going out from a $gray$ node $g$ in $v$, compute the minimum $2^+$-Star
          located at $g$, using the same method as that for the $white$ node above;
16      Choose one edge going out from a $black$ node or the set of edges going out from
          going out from a $gary$ node with the minimum average cost
          as the minimum $2^+$-Star located at $v$;
17    if  the average cost of this $2^+$-Star is less than the current minimum $2^+$-Star
          or with more nodes at the same average cost;
18      Set this $2^+$-Star as the minimum $2^+$-Star;
19  return the final minimum $2^+$-Star;
```

**Lemma 6.2.** *The $k^+$-star with the minimum average cost in graph $G'$ can be computed in $O(m \log n)$ time.*

**Proof**: Given a center $v$ and $j$ neighbor nodes $N_v^1, N_v^2, ..., N_v^j (j \ge k-1)$, with cost $cost_{(v,N_v^1)} \le cost_{(v,N_v^2)} \le ... \le cost_{(v,N_v^j)}$. We want to find the set which contains $v$ and no less than $k-1$ neighbor nodes with minimum average cost. Given three positive number $\{A, B, C\}$, we have that $\frac{A}{B} \ge C$ implies $\frac{A}{B} \ge \frac{A+C}{B+1} \ge C$ and $\frac{A}{B} \le C$ implies $\frac{A}{B} \le \frac{A+C}{B+1} \le C$. Thus the series $\frac{w(v)+\sum_{1\le k\le i} cost_{(v,N_v^k)}}{i+1}$ will be monotonically decreasing at first (except that $w(v) \le cost_{(v,N_v^1)}$) and then monotonically increasing. Let $i, k-1 \le i \le j$ be the minimum index that $\frac{w(v)+\sum_{1\le k\le i} cost_{(v,N_v^k)}}{i+1} < \frac{w(v)+\sum_{1\le k\le i+1} cost_{(v,N_v^k)}}{i+2}$, it can be easily followed that $\{v\} \cup \{N_v^1, ..., N_v^i\}$ is the minimum $k^+$-star of $v$. So we can conclude that a minimum average cost $k^+$-star located at any node in $G'$ can be optimally computed (if the minimum degree in the graph is no less than $k-1$). Since we will check all nodes in $G'$, the algorithm will return a $k^+$-Star with minimum average cost in $G'$. As for the time complexity, at each *white* node and *gray* node we need to sort its neighbors, this will cost no more than $O(n \log n)$. Since each edge in $G'$ will attend such kind of sorting in 2 iterations, the total cost will be within $O(m \log n)$ time. □

**Corollary 6.1.** *Algorithm 6.2 computes, in $O(mn \log n)$ time, a $2 \ln n$-approximation solution for the IWMST problem.*

**Proof**: Since the number of nodes will decrease by at least one in each iteration, algorithm 6.2 will finish in at most $n$ iterations. So the time complexity is within $O(mn \log n)$. Combine *Lemma 6.1* and *Lemma 6.2*, this conclusion can be naturally followed from **Theorem 6.1**. □

## 6.5 A $1.5 \ln n$-approximation Algorithm for IWMST

To compute a $1.5 \ln n$-approximation solution for IWMST, we need to find a feasible 3-structure. Can we follow the method used in algorithm Min2$^+$Star($G'$) and use the 3$^+$-Star structure? Unfortunately, we cannot use the 3$^+$-Star since it cannot express the following 3-structure in Figure 6.3.(b).

**Definition 6**: A 3$^+$-crab rooted at $u$ is a tree with $n \geq 3$ nodes and all other node in this tree has less than 3 nodes in its subtree.

**Lemma 6.3.** *Any solution to the IWMST problem with no less than* 3 *nodes can be decomposed into a set of* 3$^+$*-crabs without increasing the cost.*

**Proof**: To prove this lemma, we need only to modify the decompose method, others can be similarly followed from *Lemma 6.1.* A tree rooted at $u$ with 2 children is a trivial 3$^+$-crab whose cost can be optimally computed (Figure 6.3.(a)). By induction on the number of nodes $n$ in the tree, suppose the lemma is correct for $3 \leq n \leq i$ and consider a tree with $n = i + 1$ nodes (Figure 6.3.(c)(d)(e)). Set an arbitrary leaf $u$ as the root, then randomly select an inner node $v$ and check whether the sub-tree rooted there has no less than 3 nodes or not. If the number of nodes is less than 3, go upside along the tree until we find an inner node has no less than 3 offspring nodes. Now check whether there is a child of this inner node who has a 3$^+$-crab rooted there. If there is no such child, we choose $v$ as the center of a 3$^+$-star and contract it. Else we go down the tree from one of such inner node child until we reach an inner node who has no 3$^+$-crab children and contract the 3$^+$-crab located there. Thus we can reduce the number of nodes in the tree by at least two and then we can apply the induction hypothesis to further decompose the remain parts. □

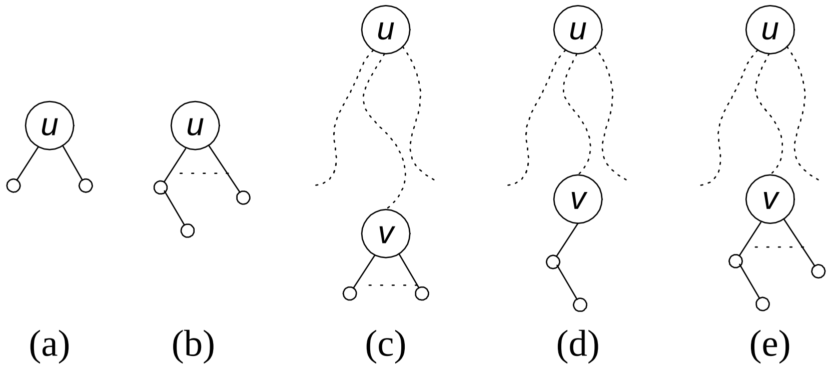


Figure 6.3: (a) A tree located at $u$ with 2 children is a trivial 3$^+$-crab and also a 3$^+$-star. (b) A 3$^+$-crab which is not a star. (c)(d)(e) An inner node $v$ which has more than 3 subtree nodes and has no 3$^+$-crab node as a child can be contracted as a 3$^+$-crab.

Now we need to design an algorithm to compute the 3$^+$-crab with the minimum average cost rooted at any given node in graph $G'$.

Let $N_v^{1*}$ be the set of neighbor nodes of node $v$ and $N_v^{2*}$ be the set of nodes which are within 2-edge distance from node $v$. Given a node $u \in N_v^{1*}$, we use $C_u^{1*}$ to denote the minimum cost for connecting $u$ to $v$. Given a node $u \in N_v^{1*}$ and another node $z \in N_v^{2*}$, we use $C_{\{u,z\}}^{2*}$ to denote the minimum cost for connecting both $u$ and $z$ to $v$. We can use the method in Algorithm Min2$^+$Star($G'$) to compute $C_u^{1*}$ for each $u \in N_v^{1*}$. As for $C_{\{u,z\}}^{2*}$, if $u$

and $z$ are not connected but both of them are included in $N_v^{1*}$, them $C_{\{u,z\}}^{2*} = C_u^{1*} + C_z^{1*}$. If $u$ and $z$ are connected and exactly one them (say, $u$) is included in $N_v^{1*}$, them $C_{\{u,z\}}^{2*}$ equals the sum of $C_u^{1*} + w(u)$ and the cost of connecting $u$ to $v$. If $u$ and $z$ are connected and both of them are included in $N_v^{1*}$, then we have three methods to add $u$ and $z$ into the tree (i.e., both $u$ and $z$ are children of $v$ in or just one of them), $C_{\{u,z\}}^{2*}$ will be the minimum value among these three cases. If $u$ and $z$ are not connected and at least one of them is not included in $N_v^{1*}$, or $u$ and $z$ are connected but both of them are not included in $N_v^{1*}$, then we assign the maximum value to it.

**Algorithm 6.3. Min$3^+$-KCrab($G', v, k$)**
**Input:**
*$G' = (V', E')$, $v \in V$ is a gray node or white node, $k \geq 2$ is an integer;*
**Output:**
*A $3^+$-Crab of $v$ with $k$ nodes in $G'$ and the average cost minimized;*

```
1   Construct a new graph $G_v$ on $N_v^{2*}$, in which the cost of an edge between any
     two nodes $u$ and $z$ is $C_{\{u,z\}}^{2*}$ if $C_{\{u,z\}}^{2*} < \infty$ ;
2   if $k$ is an odd number:
3       Add $|N_v^{2*}| - (k-1)$ dummy nodes into $G_v$;
4       Connect each dummy node to all nodes in $N_v^{2*}$ with zero cost dummy edges;
5       Find the minimum weight perfect matching in $G_v$;
6       return the $\frac{k-1}{2}$ pairs which are not connected by dummy edges in the matching.
7   else if $k$ is an even number:
8       For each node $x \in N_v^{1*}$ do
9           Delete $x$ and all its edges from $G_v$;
10          Add $|N_v^{2*}| - (k-2)$ dummy nodes into $G_v$;
11          Connect each dummy node to all nodes in $N_v^{2*} - \{x\}$
                with zero cost dummy edges;
12          Find the minimum weight perfect matching in this $G_v$;
13          Set the final cost of this matching as its own cost plus $C_u^{1*}$;
14      Select the one with minimum final cost among the above matchings;
15      Let $y$ be the single node outside of this matching;
16      return
            $y$ and the $\frac{k-2}{2}$ pairs which are not connected by dummy edges in the matching.
```

**Lemma 6.4.** *Algorithm Min$3^+$-KCrab($G', v, k$) can compute the $3^+$-Crab of $v$ with $k$ nodes in $G'$ and minimized average cost in $O(|N_v^{1*}|^2 |N_v^{2*}|^2)$ time.*

**Proof**: Suppose $T_v^k (k \geq 3)$ is the $3^+$-crab rooted at $v$ with exactly $k$ nodes whose average cost is minimized. If $k$ is an odd number, then there are $k-1$ nodes from $N_v^{2*}$. These nodes can either connect $v$ directly or via another neighbor node of $v$. Thus $v$ will have an even number of single-node branches and the other nodes will form a set of twin-nodes branches. So we can pair all single-node branches according to the non-decreasing order of their connecting costs. These node pairs and the twin-nodes pairs will form a $\frac{k-1}{2}$-pair matching in $G_v$. Since we will find the minimum weight perfect matching in this $G_v$ whose cost comes from such a $\frac{k-1}{2}$-pair matching, it follows that the cost of the final solution is minimized.

If $k$ is an even number, then there is at least one single-node branch and the remain nodes can be paired by the same way as above. We will try all nodes in $N_v^{1*}$ and subsequently find the minimum average cost $3^+$-Crab of $v$ with $k$ nodes in $G'$.

To compute the minimum weight perfect matching in $G_v$, we can use the implementation of the famous Edmonds' algorithm [Edm65a, Edm65b] designed by Gabow [Gab90] with time complexity of $O(n(m + n \log n))$ (which is currently the best known result in terms of $n$ and $m$). Since the number of nodes in $G_v$ is $|N_v^{2*}|$ and the number of edges is within $|N_v^{1*}| * |N_v^{2*}|$ while we will try kicking out a node for at most $|N_v^{1*}|$ times, the total time complexity will be within $O(|N_v^{1*}| * (|N_v^{2*}| * (|N_v^{1*}| * |N_v^{2*}| + |N_v^{1*}| * \log |N_v^{1*}|))) = O(|N_v^{1*}|^2 |N_v^{2*}|^2)$. □

**Algorithm 6.4. Min$3^+$-Crab($G'$)**
**Input:**
*The undirected graph $G' = (V', E')$, each node in $V'$ is a sub-tree in $G$;*
**Output:**
*A $3^+$-Crab with the minimum average cost in $G'$;*

```
1   Set the current minimum $3^+$-Crab as an empty set with an upper-bound average cost;
2   for each node $v \in V'$ do
3     if $v$ is a white node:
4       Use the method in Algorithm Min$2^+$Star($G'$) to compute $C_u^{1*}$ for each $u \in N_v^{1*}$;
5       Compute $C_{\{u,z\}}^{2*}$ for each pair $\{u, z\} \subseteq N_v^{2*}$ according to the above analysis;
6       for $3 \le i \le |N_v^{2*}| + 1$ do
7         Call Algorithm Min$3^+$-KCrab($G', v, i$) to compute
                the minimum $3^+$-crab with $i$ nodes;
8       Select the one with minimum average cost in the above computed $3^+$-crabs;
9     else if $v$ is a super node:
10      Compute $C_u^{1*}$ and $C_{\{u,z\}}^{2*}$ by the same way as for the white node;
11      For all gray node $g$ in $v$, compute its minimum $3^+$-Crab use Min$3^+$-KCrab($G', g, i$);
12      Sort the connecting cost for each neighbor node $u$ and the average cost of each
          minimum $3^+$-Crab located at a gary node according to the non-decreasing order;
13      Choose the one with minimum average cost in the following three combinations:
          1)the minimum two edges, 2) the minimum $3^+$-Crab,
          3) the minimum edge and the minimum $3^+$-Crab ;
14    if the average cost of this $3^+$-Crab is less than the current minimum$3^+$-Crab
          or with more nodes at the same average cost;
15      Set this $3^+$-crab as the minimum $3^+$-crab;
16  return the final minimum $3^+$-crab;
```

**Corollary 6.2.** *Algorithm 6.4 computes, in $O(n^2\Delta^6)$ time ($\Delta$ is the maximum degree), a $1.5 \ln n$-approximation solution for the IWMST problem.*

**Proof**: By Lemma 6.4, we can compute the minimum average cost $3^+$-Crab of $v$ with $k$ nodes in $G'$ for any white node or gray node $v$. The gray node case will only be checked when we are computing the minimum average cost $3^+$-Crab of the dominating super node. For a super node, we can deem its core as a zero cost node since the inner nodes have been paid before. Thus we need only to consider the minimum two single-node branch and the minimum $3^+$-Crab located at one of its gray node. Thus Algorithm 6.4

will succeed in finding the minimum average cost $3^+$-Crab of any node $v$ in $G'$. According to **Theorem 6.1**, by the assumptions proved in *Lemma 6.3* and *Lemma 6.4*, we are sure that Algorithm 6.4 will return a $1.5 \ln n$-approximation solution for the IWMST problem.

Now let's check the time complexity. Since $|N_v^{1*}| \leq \Delta$ and $|N_v^{2*}| \leq \Delta^2$ while the edges in $|N_v^{2*}|$ will be no more than $|N_v^{1*}|^2$, the cost of computing the minimum average cost $3^+$-Crab of any node $v$ will be within $\Delta^6$. Since we will run no more than $n/2$ iterations and in each iteration we will check at most $n$ nodes, the total time complexity will be bounded by $O(n^2\Delta^6)$. □

## 6.6 Fault Tolerant IWMST in Metric Space

In metric space, the cost of each edge in $G$ satisfies the triangle inequality. For the metric IWMST, we can use algorithms for the metric facility location problem as basic stones. The following procedure is based on the work of [JMS02].

### 6.6.1 A Simple Algorithm for Metric IWMST

1. We turn an instance of IWMST into an instance of Facility Location by deem each node in $G$ as both a facility with open cost as the weight of this node and a client with demand 1. Let $t$ be the number of current time step which starts from 0. At time 0, all nodes are unconnected, all facilities are unopened, and the budget of every node $j$, denoted by $B_j$, is initialized to 0. At every moment, each node $j$ offers some money from its budget to each unopened facility $i$. The amount of this offer is computed as follows: If $j$ is unconnected, the offer is equal to $max(B_j - c_{ij}, 0)$ (i.e., if the budget of $j$ is more than the cost that it has to pay to get connected to $i$, it offers to pay this extra budget to $i$ ); If $j$ is already connected to some other facility $i'$, then its offer to facility $i$ is equal to $max(c_{i'j} - c_{ij}, 0)$ (i.e., the amount that j offers to pay to $i$ is equal to the amount $j$ would save by switching its facility from $i'$ to $i$).

2. While there is an unconnected node, increase the time, and simultaneously, increase the budget of each unconnected node at the same rate (i.e., every unconnected node $j$ has $B_j = t$ at time $t$), until one of the following events occur. If multiple events occur at the same time, process them in an arbitrary order.

    a) For some unopened facility $i$, the total offer that it receives from cities is equal to the cost of opening $i$. In this case, we open facility $i$, and for every node $j$ (connected or unconnected) which has a non-zero offer to $i$, we connect $j$ to $i$. The amount that $j$ had offered to $i$ is now called the contribution of $j$ toward $i$, and $j$ is no longer allowed to decrease this contribution.

    b) For some unconnected node $j$, and some facility $i$ that is already open, the budget of $j$ is equal to the connection cost between $j$ and $i$. In this case, we connect node $j$ to facility $i$. The contribution of $j$ toward $i$ is zero.

3. For every node $j$, set $\alpha_j$ (the share of $j$ of the total expenses) equal to the budget of $j$ at the end of algorithms. Notice that this value is also equal to the time that $j$ first gets connected.

4. Connect all the opened facility nodes by a minimum spanning tree in $G$.

We denote the above algorithm as the JMS algorithm. It computes a $3.105-$approximation solution for metric IWMST. We will extend this algorithm for computing the fault-tolerant IWMST problem.

### 6.6.2 The Fault-tolerant Metric IWMST Problem

Here we will consider two variants of the IWMST problem for reliability consideration. The first version focuses on the leaf nodes, while the second one focuses on the backbone nodes.

**Fault-tolerant Metric IWMST Problem (Version 1)** (FMIwmst1)

- INSTANCE: Undirected graph $G = (V, E)$, weight $c(e) \in R^+$ for each $e \in E$ and weight $w(v) \in R^+$ for each $v \in V$, a positive bound $B$.
- SOLUTION: A backbone tree $T$ in $G$.
- MEASURE: The total weights of nodes and edges in $T$ plus summation of the cost to connect at least 2 nodes in $T$ for each node in $G - T$.

We can modify the JMS algorithm to deal with the Problem FMIwmst1. Now that each leaf node need to be connected to at least two inner nodes, we have to set the demand as 2 for each node in $G$. In the first two steps, we will say a node unconnected if it is not connected to any opened facility and we will say it is connected if it has already connected to two opened facilities. For the case when a node is connected to exactly one opened facility, we will check it in both situation. Other steps remain the same. This algorithm has an approximation ratio 1.62 for computing the facilities to be opened. Since the minimum spanning tree in the final step will be no more than two times the cost of the optimal solution, we have that we can compute a $3.62-$approximation solution for this problem.

**Fault-tolerant Metric IWMST Problem (Version 2)** (FMIwmst2)

- INSTANCE: Undirected graph $G = (V, E)$, weight $c(e) \in R^+$ for each $e \in E$ and weight $w(v) \in R^+$ for each $v \in V$, a positive bound $B$.
- SOLUTION: A $2-$connected subgraph $T$ in $G$.
- MEASURE: The total weights of nodes and edges in $T$ plus summation of the cost to connect at least 1 node in $T$ for each node in $G - T$.

To solve the Problem FMIwmst2, we need only to add one more step at the end of the JMS Algorithm. After we compute the minimum spanning tree, we can run the approximation algorithm for metric travelling salesman which turn a minimum spanning tree into a circle with cost no more than 3/2 times the cost of the minimum spanning tree. Thus we can compute in polynomial-time a $4.62-$approximation solution for this problem.

## 6.7 Some Other Related Problems

**Problem 2. Node Weighted Steiner Connected Dominating Set Problem (NWS-CDS).**
**Instance**: Undirected graph $G = (V, E)$, weight $w(v) \in R^+$ for each $v \in V$, a set $R \subseteq V$, a positive bound $B$.
**Question**: Is there a set $S \subseteq V$ such that each node in $R$ has at least one neighbor node included in $S$ and the sum of the weights of the nodes in $S$ does not exceed $B$?

**Lemma 6.5.** *A polynomial approximation algorithm for the node weighted steiner connected dominating set problem with factor $k$ would imply a polynomial approximation algorithm for the IWMST problem with factor $k$.*

**Proof**:We can reduce the IWMST problem to the NWS-CDS problem. Given an instance of IWMST problem $G = (V, E)$, we construct a new graph $G' = (V', E')$. For each node $u \in V$, we add it to $V'$ with the same weight. For each edge $(u, v) \in E$, we add a new node $x$ to $V'$ and set $f(x) = c((u, v))$. Then we add two edges $(u, w)$ and $(w, v)$ to $E'$, all edges have no weight. Set $R = V$, we can see that an optimal solution for the NWS-CDS problem on $G'$ is exactly the optimal solution for the IWMST problem on $G$. And for any polynomial approximation algorithm of the NWS-CDS problem, this reduction will preserve the factor on the IWMST problem. □

**Problem 3. Node Weighted Steiner Tree Problem (NWS-T).**
**Instance**: Undirected graph $G = (V, E)$, weight $w(v) \in R^+$ for each $v \in V$, a set $R \subseteq V$, a positive bound $B$.
**Question**: Is there a tree $T$ for $G$ such that all nodes in $R$ are connected by $T$ and the sum of the weights of the nodes in $T$ does not exceed $B$?

**Lemma 6.6.** *A polynomial approximation algorithm for the node weighted steiner connected dominating set problem with factor $f(n)$ would imply a polynomial approximation algorithm for the IWMST problem with factor $f(n) + \ln \Delta$.*

**Proof**:We can solve the IWMST problem by using a approximation algorithm for the NWS-T problem. Given an instance of IWMST problem $G = (V, E)$, we can construct an instance for the weighted set cover problem in which $V$ is the universe set. For each node $v$ in $V$ we build a set $s_v$ which include this node and all its neighbor nodes, we set its weight as $w(s_v) = w(v) + \sum_{u,(v,u)\in E} c((v, u))$. Then we can use a weighted set cover approximation algorithm to find a dominating set in $G$.

Now we clear the weight on all nodes in the dominating set and its neighbor edges. For each uncovered edge, we put a new node on it and move its weight to this node. Then we run the polynomial approximation algorithm of the NWS-T problem with the dominating set as the required terminal set.

Since the weighted set cover problem can be approximated to $\ln \Delta$, if there is a polynomial approximation algorithm for the NWS-T problem with factor $f(n)$, then our final solution can be bounded by $f(n) + \ln \Delta$. □

## 6.8 Concluding Remarks

In this chapter we have designed a general framework which aims to find a $\frac{k}{k-1}\ln n$-approximation Algorithm for the IWMST problem. Based on this framework, we further propose two polynomial-time approximation algorithms. The first one can find a $2\ln n$-approximation solution in $O(mn\log n)$ time, while the second one can compute a $1.5\ln n$-approximation solution in $O(n^2\Delta^6)$ time. We also show how we can use the methods for other related problems to solve the IWMST problem. The techniques in our framework can be extended to some other related problems. For future work we can follow this method to develop algorithms with more tight bound such as $\frac{4}{3}\ln n$. We will also further study the metric version of this problem.

# Chapter 7

# Energy Efficient Broadcasting Algorithms in MANETs

## 7.1 Introduction

A Mobile Ad Hoc Network (MANET) is a mobile network with dynamic, sometimes rapidly-changing, random, multihop topologies which usually contains relatively bandwidth-constrained wireless links. MANET can be rapidly deployed without relying on pre-existing fixed network infrastructure and thus has potential applications in military mission, emergency disaster relief and etc.

Each node in a MANET has a limited energy resource and an omnidirectional antenna to transmit and receive signals. A communication session can be established either through a single-hop transmission if the communication parties are close enough or through relaying by multi-hop path otherwise.

In this chapter we focus on the design of algorithms for energy-efficient broadcast communications. A broadcast message originated from a source node needs to be forwarded to all the other nodes in the network, which is an important mechanism to communicate information such as states updating packets and other control data in a MANET. There are already some existing solutions, most of which are globalized and each node needs knowledge of the whole network topology to make decision. But a small local alternation may cause global changes in any MST based structure and must be propagated throughout the network for any globalized solution, which may bring unacceptable communication overhead for ad-hoc networks. Thus a scalable protocol should let each node decide on its own behavior based only on the information from all nodes within several-hop distance. Usually the less the knowledge required, the more scalable the protocol would be. So it would be ideal if nodes make decisions based solely on the knowledge of its 1-hop or 2-hops neighbors, and the distances to them. Such distributed algorithms and protocols are called localized [CAR03, CHU02, QVL02, WL99].

According to the source node distribution, the current existing protocols can be roughly classified into two families: topology control oriented protocols and broadcast oriented protocols. The first family assigns the transmission power for each node and the network is connected independently of broadcast utilization, while the second family considers the broadcast process from a given source node. According to the antenna transmission angle, we can also distinguish three communication models: one-to-all model, one-to-one model and variable angular range model[CAR03]. Here we are mainly interested in broadcast oriented protocols in one-to-all communication model in MANET by using omnidirectional antennas, in which a node can send data to many neighboring nodes via a single transmission. This feature is called the wireless multicast advantage and is very useful for broadcasting communications.

Our main contribution in this paper is that we propose an algorithm that only requires local information and extend it into a distributed protocol. In our localized protocols, each node requires only the 2-hop distance neighborhood info, i.e., the knowledge of its distance to its neighboring nodes and the neighbor-distance info of them. Here we do not need GPS service, distances can be measured mutually by using signal strength, time delay or more sophisticated techniques like microwave distance[Ben00]. The information needed here are far less than that needed by existing protocols like BIP[WNE00] for large scale MENETs.

The rest of this chapter is organized as follows. Section 2 gives the MANET model, which is essentially the same as that used in other existing protocols. Section 3 shortly analyzes the target and presents a centralized algorithm CNFT, then it is extended to the localized algorithm DNFT and based on it a protocol is proposed. In Section 4, we give the results of simulations showing the energy savings obtained by using our protocol and compare it with some other existing protocols. Section 5 concludes this chapter.

## 7.2 System Model

Before we present our algorithm, it is necessary to give a short description of the basic system model for Energy Efficient Broadcasting in a MANET.

At first, we assume throughout this paper that there is ample bandwidth, and that each node has enough transceivers to accommodate all service requests. In real application we can neglect those nodes without enough bandwidth to attend the communication. Second, we assume in our algorithm that the nodes in a MANET are static, actually current there is few energy efficient protocols which take into account the mobility and we will further our research to this issue in our next step. Third, we assume that each node in a MANET has equipped an omnidirectional antenna. A node also has the capacity to modify the area of coverage with its transmission. It is this kind of transmission power control that allows to reduce significantly the energy consumption and so to increase lifetime of the network. However, the adjustment of transmission signal strength usually brings connectivity change and topology alterations. Hence, nodes have to manage their transmission area under the constraint that the network should remain connected.

In broadcasting communication, we can represent a MANET by a directed graph $G = (V, E)$ where $V$ is the set of nodes and $E$ is the set of edges. Without loss of generality, we assume that $G$ is double connected. Since we only require that a broadcast message from the source can be transmitted to all nodes in the network, then what we need is the connectivity from source node to all other nodes and thus a directed spanning tree $T$ rooted at the source node is enough[NTCS99].

Each node has a power resource and the consumption rate of power is related to the distance to the furthest node it can and want to communicate directly. The maximum power level that a node $i$ can use is $p_i^{max}$ and the related distance it can reach is $r_i^{max}$, while the corresponding neighbor nodes of $i$ is $N_i^{max}$. We assume that each node $i$ has a set of discrete power levels $P_i$ (or else at any power level in the range $[0, p_i^{max}]$) and a set of vicinity radius $R_i$, it can dynamically adjust its transmitting power according to the distance of the receiving nodes and environment for the purpose of energy conservation.

For example, we show the neighborhood of a node $s$ by using different energy levels in the following Figure 7.1. According to this graph, node $s$ can use four different communicating ranges with energy consumption rates. By steadily increasing its signal strength, node $s$ can find out that $N_s^1 = \{r\}$, $N_s^2 = \{r, t, u\}$, $N_s^3 = \{r, t, u, v, w, x\}$, $N_s^4 = N_s^{max} = \{r, t, u, v, w, x, y, z\}$. Consequently, a node can know the rough relative distances to its neighbor nodes even when there is no GPS equipment. We can also draw the neighborhood tree $NT_s$ of $s$, which is showed in the right side of Figure 7.1. Finally, we can get the whole connectivity map by combining all neighborhood trees of nodes in $V$.

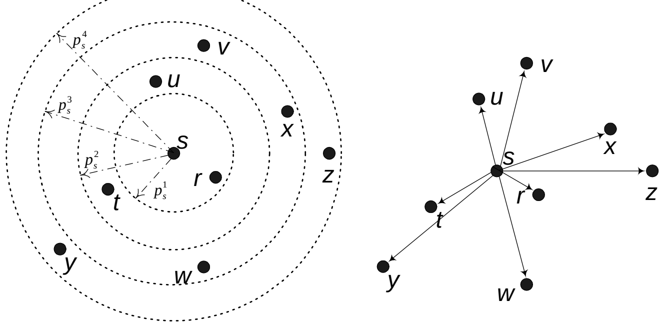


Figure 7.1: Left: The neighborhood of $s$ by different energy levels. Right: $NT_s$.

Thus we have $E = \{(u, v)|u, v \in V \wedge \; dist(u, v) \leq r_u^{max}\}$, where $dist(u, v)$ is the distance between node $u$ and node $v$. We adopt a commonly used wireless propagation model whereby the received signal power attenuates as $r^{-\alpha}$, where $r$ is the transmission range and $\alpha$ is a propagation loss constant that takes value between 2 and 5 [Rap96, KKK97]. If we take into account the environment and the overhead for signal processing and energy needed for successful reception and MAC control messages, then the general energy consumption formula for a node $u$ in the final broadcasting tree $T$ will be

$$\mathcal{E}(u) = k \cdot max\{dist(u, v)|(u, v) \in T\}^\alpha + c.$$

The problem of constructing the minimum-energy, source-based broadcast tree for each newly arriving broadcast session request involves the choice of transmitter-power levels and relay nodes. As noted earlier, we mainly consider the transmission energy. Thus, the total energy of the broadcast tree is simply the sum of the energy expended at all transmitting nodes in the tree; leaf nodes will not transmit and will not contribute to this quantity. Since we are considering session traffic, all transmitting nodes transmit for the entire duration of each session. Therefore, the total transmission energy is proportional to the total power needed to maintain the whole tree. Hence, we evaluate performance in terms of the total power $\mathcal{E}(T)$ required to maintain the tree $T$. Which is:

$$\mathcal{E}(T) = \sum_{u \in V} \mathcal{E}(u).$$

However, the problem of computing such a tree $T$ that can minimize the value of $\mathcal{E}(T)$ was already proved to be $\mathcal{NP}$-Complete and so there won't exist a polynomial-time centralized algorithm for it until $P = NP$[CHE02, GJ79].

## 7.3 A Power Efficient Protocol for Broadcasting

We start from the simple small MANETs with destinations all reachable from the source directly, and then extend our approach to larger multi-hop MANETs by means of a recursive technique. There is a crucial difference between wired and wireless networks. In wired networks, the broadcasting problem can be formulated as the well-known minimum-cost spanning tree (MST) problem, which can be solved in polynomial-time [GHS83]. This formulation is based on the existence of a cost associated with each link in the network and the total cost of the broadcast tree is simply the sum of the link costs. However, the situation in wireless networks is different because of the wireless multicast advantage property which permits all nodes within communication range to receive a transmission without additional expenditure of transmitter power. Therefore, the standard MST problem, which reflects the link-based nature of wired networks, does not capture the node-based nature of wireless ad hoc networks with omnidirectional antennas.

### 7.3.1 The centralized algorithm

Let us first consider the simple case where all nodes are within one hop and thus they know the distance of each other, which will form a whole graph. We use the link-based minimum energy path tree (MEPT) as the initial step. The main reason we take MEPT is that it performs quite well even as a final solution to our problem. As discussed above, although MST structure closely resembles energy requirements of a unicast routing task, it does not necessarily capture the structural properties in case of broadcasting. Unicast is a one dimension routing task, while broadcast in a plane is a two dimension routing task. If we draw the MST and MEPT of the same MANET under two different situation: use distance as edge weight in one and use power consumption as weight in another. We will find that the MST in both case are the same, while the MEPTs will be totally different. And what's more, we randomly generate a small graph and use linear programming to find the minimum energy path tree (MEBT), then we find that usually the MEPT topology is more close to the MEBT, which can be seen from Figure 7.2.

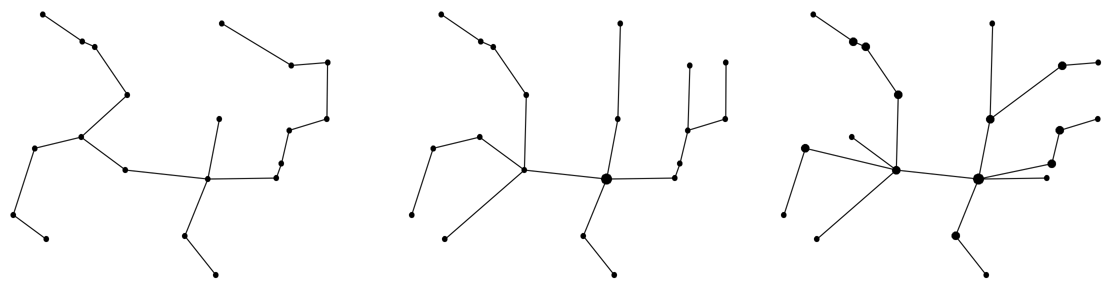


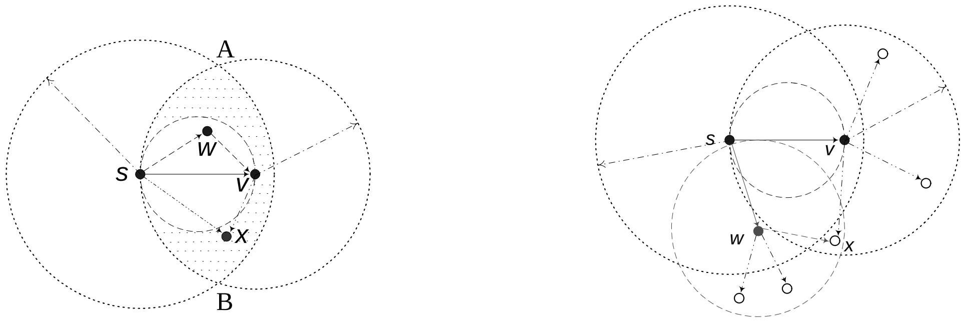
Figure 7.2. The MST, MEPT and MEBT of a small MANET.

Notice that although we use link-based MEPT, which doesn't exploit WMA, the evaluation of its cost takes into consideration the WMA. We will now describe in detail our algorithm, which we call Centralized Neighbor Forwarding Tree and refer to as CNFT. An example is provided in Figure 7.3. Let us first introduce some notations. Let $G$ be the graph, $C$ denote the set of covered nodes in a network, $T$ the set of core transmitting nodes of the broadcast tree, and $U$ the set of uncovered nodes. Notice that the contents of the above sets change throughout the execution of the CNFT, and that the sets do not hold any information about the MEPT. Initially, $C = r$, where $s$ is the source node.

**Algorithm 7.1. CNFT**

```
1   $C = \{s\}$, $T = \{s\}$, $U = G - C$;
2       Compute the MEPT of $s$;
3   For each unvisited node $t$ in $T$ do
4       Let $f$ be the neighbor of $t$ in MEPT with maximum energy distance;
5       $T = T \cup \{f\}$, set $p(t)$ as the necessary power level to cover $f$;
6       For all nodes $d$ in $N_t^{p(t)} \cap U$ do
7           $C = C \cup \{d\}$, $U = U - \{d\}$;
8       For all nodes $u$ in $U$ do
9           If there is no other node in $T$ on the path $u \to t$ then
10              Let $v$ be the nearest node covered by $t$, $T = T \cup \{v\}$;
11      Mark $t$ as visited;
```

For any transmitting node, the CNFT algorithm will add two kind of forwarding transmitting nodes for it. One is the uncovered neighbor node in its MEPT with maximum distance, this node must be set as a transmitting node not only because it will save energy but also because it is necessary for maintain the connectivity of the result broadcasting tree. And we also use its distance to set the power level of the current source node. For this node, there will be no other shortcut node between itself and the current source node. Consider the node $v$ in the following Figure 7.3(a). It will not be selected since $w$ is a shortcut node. Actually any node in the circle between node $u$ and $v$ will be a shortcut node for $v$ as long as we let the power consumption rate be proportional to the square of the radius.



a). $v$ is not in $T$ because of $w$. b).$w$ will be selected into $T$ by $x$.

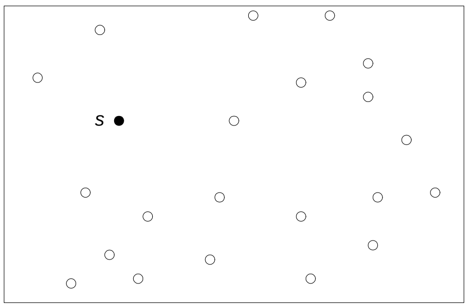
Figure 7.3. The selection of transmitting nodes in CNFT.

Another kind of transmitting node will be selected by the request of those uncovered neighbor nodes. When a uncovered node find that there is another transmitting node on its shortest path to the current source node, it will deem itself as in a safe status, which means that it will be covered later. Else it will request that one of the covered node in its shortest path to the source node should be set as a transmitting node. And this node should make all other uncovered nodes in this branch to be safe. For example, in the above Figure 7.3(b): suppose $v$ is the first kind of transmitting node selected by node $s$, and so $w$ will be marked as covered but not as a transmitting node at first; then node $x$ will find that it is not safe since its shortest path is $x \to w \to s$, so $w$ will also be set as a transmitting node and this will also make the other two uncovered nodes safe.

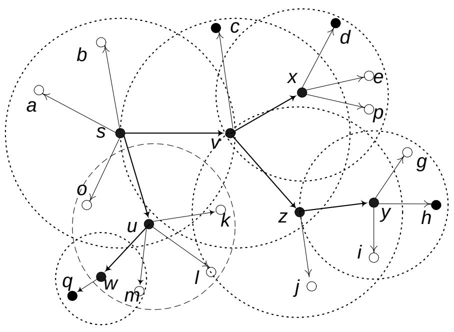
Proposition 1: When Algorithm CNFT is finished, $U = \emptyset$.

Proof: We will prove this proposition by proving the following loop invariant: $U \neq \emptyset \Rightarrow$ there exists unvisited node in $T$. This is evidently true at the initial step. For the following runs, we can check the "for" loop in line 8. Since for any uncovered node, there will be a transmitting node to make it safe. And when we come to the run when this transmitting node is deemed as a source, the algorithm should either cover that uncovered node or find another transmitting node to make it safe. So this loop invariant is always true. And when Algorithm CNFT is finished, all nodes in $T$ will be visited and thus $U = \emptyset$.

We can run CNFT on the following Figure 7.4. Node $s$ will initiate the algorithm. It will add $v$ to $T$ at first, then set its radius and cover $\{a, b, o, u, v\}$. All nodes in the branch of $v$ will be safe, but the branch of $u$ is not and they will make $s$ to add $u$ into $T$ too. $u$ will add $w$ into $T$ and cover $\{w, m, l, k\}$. $w$ will add $q$ into $T$ and cover $\{q\}$. $v$ will add $c, x, z$ into $T$ and cover $\{c, x, z\}$. $x$ will add $d$ into $T$ and cover $\{d, e, p\}$. $z$ will add $y$ into $T$ and cover $\{y, i, j\}$. $y$ will add $h$ into $T$ and cover $\{h, g, i\}$. Now all the nodes are all covered. Notice that nodes although $c, d, h, q$ are added into $T$, they cannot find any more uncovered neighbor node and will not join the transmitting. The complexity of this algorithm is $O(n^3)$ at the worst case and can be improved to $O(n^2)$ by using some smoothing technique. On the other hand, usually the number of mobile hosts within a hop distance is within two digits. So Algorithm CNFT can be extend to a scalable distributed algorithm.



a). A MANET with node S as the source.



b). The result broadcasting tree.

Figure 7.4. A small running example of CNFT.

### 7.3.2 The distributed algorithm

Since the above centralized algorithm only consider a small graph within one hop distance, it cannot be used in a real MANET, in which the transmitting radius of each node is limited and usually a multi-hop path is necessary for two nodes to communicate with each other. So we need to extend it to the distributed case, the following Distributed Neighbor Forwarding Tree Algorithm will be run at each transmitting node $t$ during the broadcasting tree building session.

**Algorithm 7.2. Algorithm DNFT**

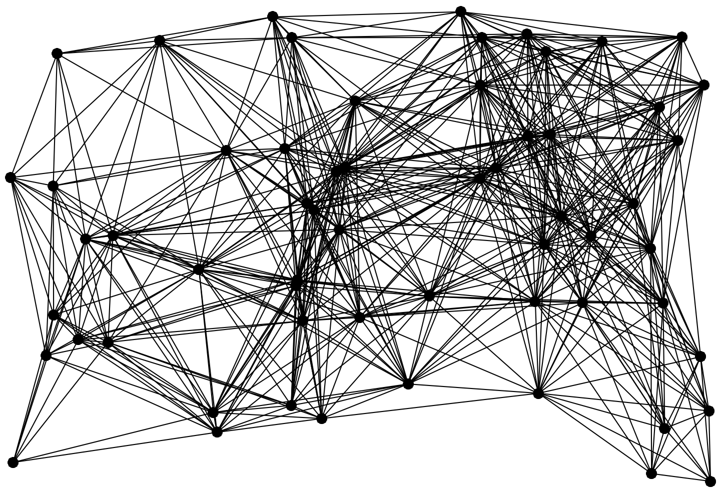
```
1    Combine the whole graph of $N_t^{max}$ and the 2-hop nodes to form $G'$ ;
2    Cover $t$ and get the uncovered node sets $U$ in $G'$;
3    Compute the MEPT of $t$ in $G'$;
4    Let $f$ be the neighbor of $t$ in MEPT with maximum energy distance;
5    Mark $f$ as a $T$ node, set $p(t)$ as the power level to cover $f$;
6    For all nodes $d$ in $N_t^{p(t)} \cap U$ do
```

```
7         $C = C \cup \{d\}$, $U = U - \{d\}$;
8      For all nodes $u$ in $U$ do
9         If there is no other unvisited $T$ node on the path $u \to t$ then
10            Let $v$ be the nearest node covered by $t$, $T = T \cup \{v\}$;
11     Mark $t$ as visited;
```

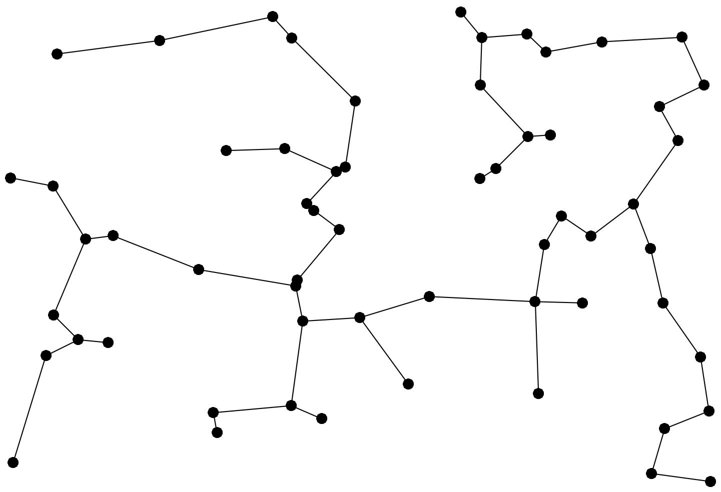
In the distributed case, for the first time, a source node $s$ will initiate a broadcast session by send out a broadcast tree construction request message $< btc_{s,id} >$ to all his neighbors reachable by power level $p^{s}_{max}$. Based on Algorithm DNFT, the nodes in the MANET will cooperatively discover the set of transmitting nodes :

- The source node $s$:
  1. Send a broadcasting tree construction request message $< btc_{s,id} >$ to all his neighbors reachable by power level $p^{max}_{s}$;
  2. Based on the $ACK$ message from all neighbors, compute $N^{max}_{s}$;
  3. Based on the $N^{max}_{v}$ message from each neighbor $v$, compute the 2-hop graph $G'$;
  4. Run Algorithm DNFT to compute the transmitting neighbor set;
  5. Send $< btt_{s,id} >$ to all transmitting neighbors.
- On receipt of a $btc$ message at node $v$:
  1. Send back an $ACK$ message;
  2. Broadcast its own $btn$ message;
  3. Based on the $ACK$ message from all neighbors, compute its $N^{max}_{v}$;
  4. Send back an $N^{max}_{v}$ message.
- On receipt of a $btn$ message: Send back an $ACK$ message.
- On receipt of a $btt$ message: Run the same procedures as a source node.

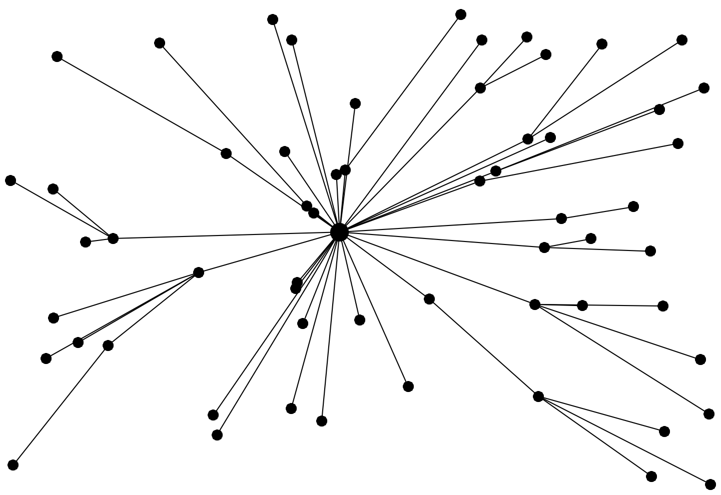
A node will discard an already received message in this session. And after it every transmitting node will have a list of neighbor transmitting nodes. Every node will know which node cover himself. Now a source node will broadcast the data according to the range computed in the tree construction session. And only those transmitting nodes will forward the data according their computed power rang respectively. All the duplicated message will be discarded, and all other common nodes need only to receive data from a transmitting node. The following Figure 7.5 shows the cover map of a 60 nodes $1km \times 1km$ MANET computed by our protocol.



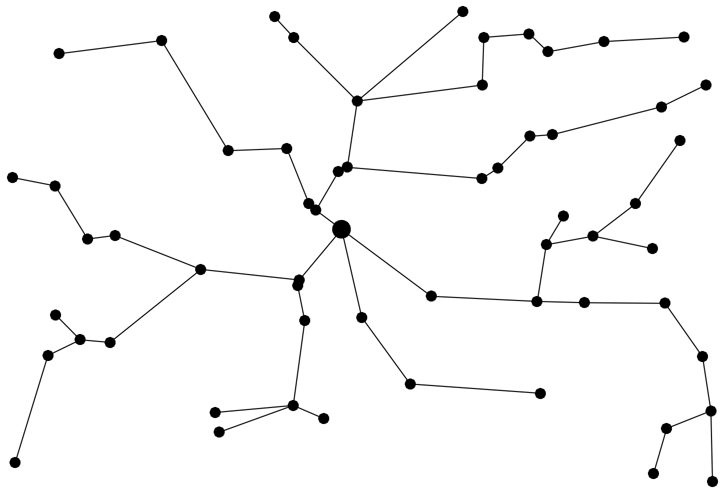
a). The topology graph of a MANET

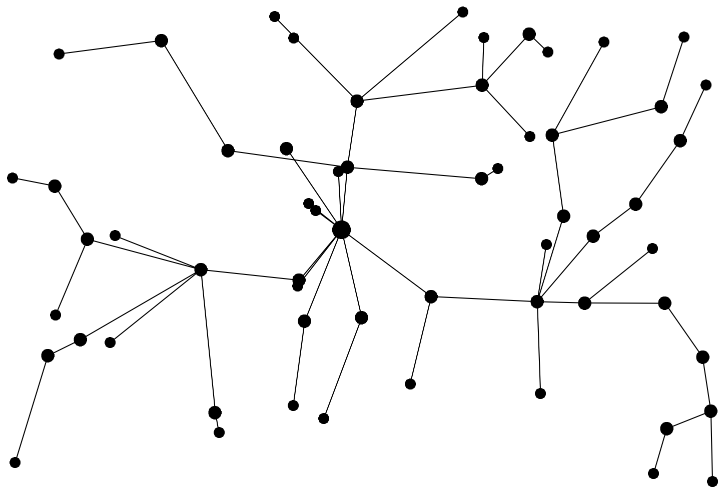


b). The MST using energy as edge weight



c). The MEPT using distance as edge weight


d). The MEPT using energy as edge weight


e). The energy efficient broadcasting tree.

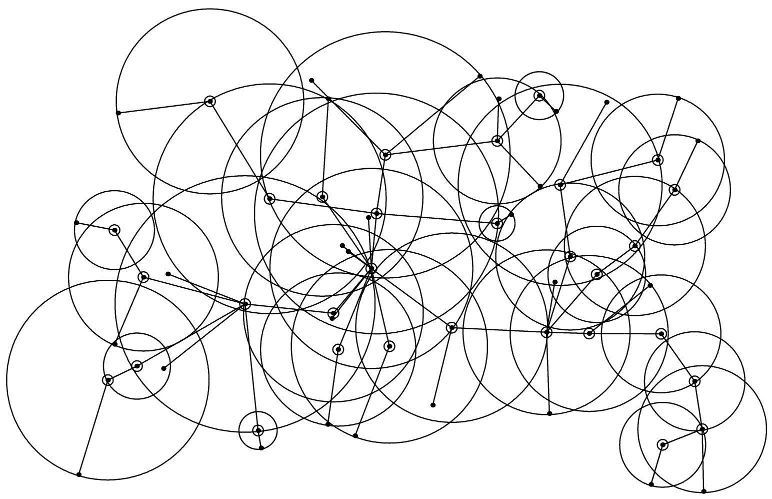

f). The broadcasting relay map.

Figure 7.5. A running result example for DNFT.

## 7.4 Performance Evaluation

We conducted a simulation study to evaluate our protocol and compared its performance with the RBOP protocol proposed by J. Cartigny et. al in [CAR03]. The authors showed that RBOP was one of the first presented distributed algorithms for computing the energy efficient broadcasting tree in a MANET and that its performance could compete with the centralized BIP protocol in [WNE00].

The parameters used in our simulations are almost the same as those in [CAR03] for consistency. The number of nodes $n$ is always 100 and nodes are static. The maximum communication radius $r^{max}$ is fixed to 5*50 meters. Nodes are uniformly distributed in a square area whose size is adjusted to obtain a given density (from 6 nodes per communication zone to 30). For each measure, 5000 broadcasts have been run. Because of ideal MAC layer and the proved integrality nature of our protocol, we are sure that all nodes receive broadcasted messages. Hence, the reachability is always 100. The observed parameter is the energy consumption according to two commonly used energy models: $k = 1, \alpha = 2, c = 0$ and $k = 1, \alpha = 4, c = 10^8$[RM99, LiR01]. For each broadcast session, we calculate the total energy consumption:

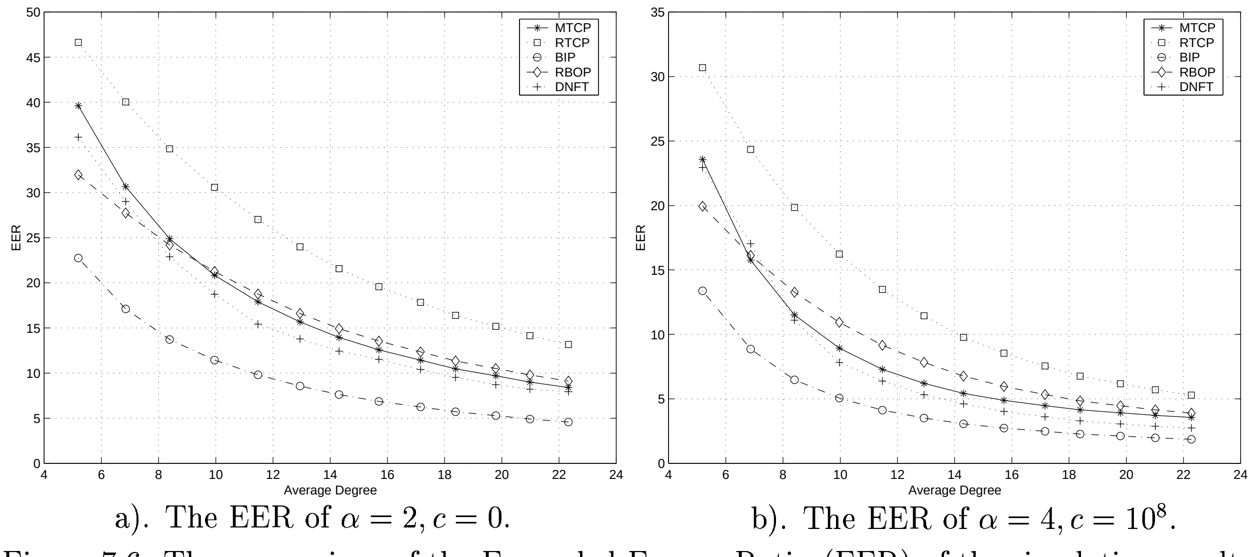
$$\mathcal{E}(T) = \sum_{u \in V} \mathcal{E}(u).$$

Usually $\mathcal{E}(T)$ is very large, so we divided it by the total energy consumption needed for blind flooding prtocol with maximal range:

$$\mathcal{E}_{flooding} = n \times (R^{\alpha} + c).$$

The percentage number of this quotient is named to be the average Expended Energy Ration (EER):

$$\mathcal{EER} = 100 \times \mathcal{E}(T)/\mathcal{E}_{flooding}.$$



a). The EER of $\alpha = 2, c = 0$. b). The EER of $\alpha = 4, c = 10^8$.

Figure 7.6. The comparison of the Expended Energy Ratios(EER) of the simulation results.

We show the comparison of DNFT with RBOP, BIP and RTCP in Figure 7.6, from which we can observe that our DNFT is better than RBOP. This is because RBOP will choose all its RNG-neighbors as transmitting nodes. But sometimes the distance from the source node to its RNG-neighbors will be far different to each other, and some short range nodes will be unnecessarily put into transmitting. In DNFT, only the uncovered neighbor with the maximum distance will be selected at the first step. And the other transmitting nodes are also selected when it is necessary to maintain the connectivity of the whole graph. So the final number of transmitting nodes in the graph in DNFT will be less than that in RBOP.

It is not surprising to see that the best algorithm is the globalized $BIP$, since with the whole knowledge one can always make better choice. But when the density rises, the difference between all these protocols converge together. And the direction is certainly downward since with high density the blind flooding will choose more unnecessary nodes as transmitting nodes.

## 7.5 Conclusion

In this chapter we first present the *CNFT* Algorithm for the computation of an energy efficient broadcasting tree in a small MANET, we prove its validity and its efficiency. Then we extend it to the distributed case algorithm *DNFT*, based on it we propose our protocol for the construction of an energy efficient broadcasting tree in large MANETs. Finally we demostrate by simulation that our protocol is more energy efficient than *RBOP*, and it is also very flexible and scalable for large MANETS.

# Chapter 8

# Summary and Future Work

In this final chapter, we summarize our research on the design of fault-tolerant and QoS routing algorithms. Then, we conclude with some topics for future research.

## 8.1 Summary

In this thesis we concentrate on computing fault-tolerant edge-disjoint paths, finding QoS concerned flow algorithms, building low cost routing structures and designing energy-efficient broadcasting schemes in wireless networks. Generally, the work presented in this thesis can be summarized as follows:

- The problem of finding two Delay-Restricted Link Disjoint $s$-$t$-Paths with minimum total cost is the first focus of our work. This is a classical $\mathcal{NP}$-hard problem and has attracted considerable attentions recently. In this thesis we first showed the difficulty of this problem and its relationship with the minimum cost flow problem. Based on the existing work, we presented an approximation algorithm for this problem. Previous result could reduce the delay of paths to be near-optimal, but the cost sacrifice was too large. We looked inside the existing model and proposed a new technique for finding a cost-bounded negative-delay cycle. By this technique we can bound the cost tradeoff while reducing the delay. Our result has improved the approximation factor from $(1+1/k, O(k))$ to $(1+1/k, O(\log k))$, thus it reduced the cost by factor $\frac{k}{\log k}$ while maintaining the same delay bound of the previous result. Following this line, we proposed two improved versions to further reduce the time complexity and the solution cost. However, these versions have a common problem of containing too complicated computing steps to be implemented. To make it simple and efficient, we analyzed the mathematical structure of this problem and developed a totally new approach of applying Lagrangian Relaxation. With this method, we only need to modify the weight of each edge and then do binary search. This method looks easy and even naive, but we validated its power by rigorous mathematical proof. Our algorithm is much more time-efficient and implementable than all previous algorithms for this problem. Furthermore, we also proposed some extensions of our method for solving other problems with similar underlying mathematical structures.
- The unsplittable flow problem is a natural generalization of the disjoint paths problem. This problem has a large family of variants and has attracted huge attentions

in both theoretical computer science and operations research. In this thesis we studied several versions of this problem. Our first result was a new algorithm for computing a minimum-cost single-source unsplittable flow in a graph with arbitrary edge capacity. We introduced a novel rounding technique: instead of rounding the flow, we round the flow portion. By this method we need not to cancel flow and to deal with the troublesome numeric issues. We proved that the approximation factor of our algorithm was also the current best bound for this problem. What's more, we found that our algorithm was also very useful for the situation when the largest demand in $G$ is far less than the minimum capacity in it. Later we studied the minimum-cost single-source $k-$splittable flow problem and we presented an algorithm with the best performance guarantee so far.

- The Inner-node Weighted Minimum Spanning Tree structure is very useful in building high-speed fibre networks. In this thesis we presented a general framework which can find a $\frac{k}{k-1}\ln n$-approximation algorithm for solving this problem. Our framework employed a method of repeatedly finding the minimum $k$-structure in the remain graph and contracting it. Both the correctness and the polynomial-time computability of our approach have been proved. Based on this framework, we further developed two polynomial-time approximation algorithms which can achieve a better performance guarantee than all existing results. Two different fault-tolerant versions of this problem have been studied, one version for avoiding leaf-node failure and another for avoiding backbone failure. We have proposed simple but efficient algorithms for both of them.

- Finally, we have investigated the energy-efficient broadcasting issue in mobile ad hoc networks and wireless sensor networks. After we examined the routing model and the energy-consumption mechanism in mobile computing, we proposed a centralized heuristic algorithm for computing a broadcasting tree in a wireless network. Grounded on this centralized algorithm, we designed a distributed and localized algorithm for saving energy during a broadcast session. We introduced a new localized method for computing the neighbor forwarding tree, which reduced the communication complexity and thus the energy consumption. We conducted experiments by using MATLAB, the simulation data validated the high energy-efficiency and scalability of our protocol.

## 8.2 Future Work

Fault-tolerant and QoS routing is an extremely flourishing area. In the future, the research presented in this thesis can be extended in the following directions:

- For the two Delay-Restricted Link Disjoint $s$-$t$-Paths Problem, it will be a very challenging topic to design a bicriteria $(1+\alpha, 1+\beta)-$approximation algorithm $(0 \leq \alpha < 1/2, 0 \leq \beta < 1/2)$. This means that we need to find a solution with both delay and cost near optimal. We know such kind of Fully Polynomial-Time Approximation Schemes (FPTAS) exist for the constrained shortest path problem, and [PY00] shows that some problems have an FPTAS even under more than two constraints. But usually this kind of problems have a pseudo-polynomial-time algorithm.

- For the Minimum-Cost Single-Source Unsplittable Flow Problem. Currently the best bicriteria approximation factor for this problem with the congestion assumption is $(3, 1)$ and it has been proved that a $(2-\varepsilon, 1)$-approximation algorithm is impossible if $P \neq NP$. So we are really wondering about the existence of a $(2, 1)$-approximation algorithm for it.

- For the Minimum-Cost Single-Source Unsplittable Flow Problem without arbitrary edge capacity, our algorithm achieves a $(3 + 2\sqrt{2}, 1)$-approximation solution. Can we improve on the congestion rate?

- We will consider the problem of finding the capacitated steiner tree problem. Such a steiner tree can act as a sub-structure for routing multi-commodity flows if they can be transferred at different times. Furthermore, we will consider the capacitated 2-connected sub-graph problem. This problem aims to find a fault-tolerant sub-structure for delivering a given set of commodities.

- The metric version of Inner-node Weighted Minimum Spanning Tree Problem will also be further studied in our future work. We will also consider its applications in optical networks.

- For the QoS routing issues in MANETs, we will continue our efforts on designing energy-efficient algorithms. We will consider the problem of maximum-lifetime routing for this topic. We will also extend our protocol to multicast communications. Then we will study the case where the MANET model changes, such as to add mobility into the MANET and to alter the transmission pattern of the antennas.

# Bibliography

[AMO93] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Networks Flows*, Prentice-Hall, NJ, USA, 1993.

[AK04] J. N. Al-Karaki and A. E. Kamal, “Routing Techniques in Wireless Sensor Networks : A Survey 04,” *IEEE Wireless Communications*, 11(6):6-28, Dec. 2004.

[ASSC02] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam and E. Cayirci, “A Survey on Sensor Networks,” *IEEE Communications Magazine*, pages 102-114, August, 2002.

[AY05] K. Akkaya and M. Younis, “A Survey on Routing Protocols for Wireless Sensor Networks,” *Elsevier Ad Hoc Network Journal*, 3(3):325-349, 2005.

[BKS02] G. Baier, E. Köhler, and M. Skutella, “On the k-splittable flow problem,” In *Proceedings of ESA 2002*, pages 101-112, 2002

[Bel58] Richard Bellman, “On a routing problem,” *Quarterly of Applied Mathematics*, 16(1):87-90, 1958.

[Bhan99] Ramesh Bhandari, *Survivable Networks - Algorithms for Diverse Routing*, Kluwer Academic Publishers, 1999.

[BE02] D. Braginsky and D. Estrin, “Rumor Routing Algorithm for Sensor Networks,” In *Proceedings of WSNA 2002*, Georgia, USA, 2002.

[Ben00] A. Benlarbi, J. -C. Cousin, R. Ringot, A. Mamouni, and Y. Leroy , “Interferometric positioning systems by microwaves,” In *Proceedings of Microwaves Symp. (MS’2000)*, Tetuan, Morocco, 2000.

[Burns91] S. M. Burns , “Performance Analysis and Optimization of Asynchronous Circuits,” *PhD Thesis*, California Institute of Technology, 1991.

[CAR03] J. Cartigny, D. Simplotand I. Stojmenovic, “Localized Minimum-Energy Broadcasting in Ad-hoc Networks,” *Proceeding of IEEE INFOCOM’2003*, San Francisco, USA, 2003.

[CGKNR02] C. Chekuri, A. Gupta, A. Kumar, S. Naor, and D. Raz , “Building edge-failure resilient networks,” *Integer Programming and Combinatorial Optimization (IPCO)*, pages 439-456, 2002.

[CHE02] Mario Cagalj, Jean-Pierre Hubaux, and Christian Enz , “Minimum-energy broadcast in all-wireless networks: NP-completeness and distribution issues,” In *Proceedings of ACM MobiCom’2002*, Atlanta, USA, 2002.

[CHU02] T. Chuand I. Nikolaidis , "Energy efficient broadcast in mobile ad hoc networks," In *Proceedings of Ad-Hoc Networks and Wireless (ADHOC-NOW)*, Toronto, Canada, pages 177-190, 2002.

[CLR03] M. Costa, L. Létocart, F. Roupin, "A greedy algorithm for multicut and integral multiflow in rooted trees," *Operations Research Letters*, 31(1): 21-27, 2003.

[CT04] J. -H. Chang, and L. Tassiulas, "Maximum Lifetime Routing In Wireless Sensor Networks," *IEEE/ACM Transactions on Networking*, 12(4):601-619, 2004.

[CV04] D. Chen and P. K. Varshney, "QoS Support in Wireless Sensor Networks : A Survey," In *Proceedings of International Conference on Wireless Networks*, pages 227-233, Nevada, USA, June 21-24, 2004.

[DIG99] Ali Dasdan, Sandy Irani, Rajesh K. Gupta, "Efficient Algorithms for Optimum Cycle Mean and Optimum Cost to Time Ratio Problems," In *Proceedings of DAC'1999*, pages 37-42, 1999.

[Dij59] E. W. Dijkstra, "A Note on Two Problems in Connexion with Graphs," *Numer. Math.*, vol. 1, pages 269-271, 1959.

[DGG99] Y. Dinitz, N. Garg, and M. X. Goemans, "On the single-source unsplittable flow problem," *Combinatorica*, 19:1-25, 1999.

[Edm65a] J. Edmonds, "Maximum matching and a polyhedron with 0,1 - vertices," *Journal of Research of the National Bureau of Standards*, 69B:125-130, 1965.

[Edm65b] J. Edmonds, "Paths, trees and flowers," *Canadian Journal of Mathematics*, 17:449-467, 1965.

[EIS76] S. Even, A. Itai, A. Shamir, "On the Complexity of Timetable and Multicommodity Flow Problems," *SIAM Jounal on Computing*, 5(4):691-703, 1976.

[ESZ02] F. Ergun, R. Sinha, and L. Zhang, "An Improved FPTAS for Restricted Shortest Path," *Information Processing Letters*, vol. 83, no. 5, pages 237-293, 2002.

[EH02] T. Erlebach and A. Hall, "NP-hardness of broadcast sheduling and inapproximability of single-source unsplittable min-cost flow," In *Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms*, 2002

[Fei96] U. Feige, "A threshold of lnn for approximating set-cover," In *Proceedings of of 28th ACM Symposium on Theory of Computing*, pages 314-318, 1996.

[FF62] L. R. Ford and D. R. Fulkerson, *Flows in Networks*, Princeton, 1962.

[FT84] M. L. Fredman, R. E. Tarjan, "Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms," *Proceedings of FOCS'1984*, pages 338-346, 1984.

[FT87] M. L. Fredman, R. E. Tarjan, "Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms," *Journal of the ACM*, 34:596-615, 1987.

[FGLTW05] R. Fleischer, Q. Ge, J. Li, S. Tianand H. Wang, "Approximating Spanning Trees With Inner Nodes Cost," In *Proceedings of of 6th International Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 148-152, 2005.

[Floyd62] Robert W. Floyd, "Algorithm 97: Shortest path," *Commun. ACM*, 5(6): 345, 1962.

[FHW80] S. Fortune, J. E. Hopcroft, J. Wyllie, "The Directed Subgraph Homeomorphism Problem," *Theoretical Computer Science*, 10: 111-121, 1980.

[Gab90] H. N. Gabow, "Data Structures for Weighted Matching and Nearest Common Ancestors with Linking," In *Proceedings of of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 434-443, 1990.

[GGST86] 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), pages 109-122, 1986.

[GHS83] R. G. Gallager, P. A. Humblet, P. M. Spira, "A Distributed Algorithm for Minimum-Weight Spanning Trees," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, v.5 n.1, pages 66-77, 1983.

[GJ79] Michael R. Garey and David S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, New York, NY, 1979.

[GK96] S. Guha and S. Khuller , "Approximation algorithms for connected dominating sets," In *Proceedings of of 4th Annual European Symposium on Algorithms*, 1996.

[GK99] S. Guha and S. Khuller , "Improved Methods for Approximating Node Weighted Steiner Trees and Connected Dominating Sets," *Inf. Comput.*, Vol. 150, pages 57-74, 1999.

[GVY97] N. Garg, V. V. Vazirani, M. Yannakakis, "Primal-Dual Approximation Algorithms for Integral Flow and Multicut in Trees," *Algorithmica*, 18(1):3-20, 1997.

[Has92] R. Hassin, "Approximation Schemes for the Restricted Shortest Path Problem," *Mathematics of Operations Research*, vol. 17, no. 1, pages 36-42, 1992.

[HIM03] M. T. Hajiaghayi, N. Immorlica and V. S. Mirrokni, "Power Optimization in Fault-Tolerant Topology Control Algorithms for Wireless Multi-hop Networks," In *Proceedings of ACM Mobicom 2003*, California, USA, 2003.

[HM02a] P. -H. Ho and H. T. Mouftah, "A Framework of Scalable Optical Metropolitan Networks for Improving Survivability and Class of Service," *IEEE Network, Special issue on Scalability in Communication Networks*, pages 29-35, July/Aug 2002.

[HM02b] P. -H. Ho and H. T. Mouftah, "A Framework for Service-Guaranteed Shared Protection in WDM Mesh Networks," *IEEE Communications Magazine*, pages 97-103, Feb. 2002.

[Ho04] Pin-Han Ho, "State-of-the-Art Progresses in Developing Survivable Routing Strategies in the Optical Internet," *IEEE Communications Surveys and Tutorials*, Vol. 6, No. 4, the fourth quarter, 2004.

[JMS02] K. Jain, M. Mahdian, and A. Saberi, "A new greedy approach for facility location problems," In *Proceedings of STOC'2002*, pages 731-740, May 19-21, 2002.

[JMMSV03] K. Jain, M. Mahdian,E. Markakis, A. Saberi, and V. V. Vazirani, "Greedy Facility Location Algorithms Analyzed Using Dual Fitting with Factor-Revealing LP," *Journal of the ACM*, Vol. 50, No. 6, pages 795-824, November 2003.

[Joh77] D. B. Johnson, "Efficient Shortest Path Algorithm," *Journal of the ACM*, 24:1-13, 1977.

[KKK97] L. Kirousis, E. Kranakis, D. Krizanc, and A. Pelc, "Power consumption in packet radio networks," In *Proceedings of 14th Symp. on Theoretical Computer Science (STACS'97)*, LNCS 1200. Springer-Verlag, Berlin, pages 363-374, 1997.

[KL001] M. Kodialam and T. V. Lakshman, "Minimum interference routing with applications to MPLS traffic engineering," In *Proceedings of IEEE Infocom'2004*, pages 884-893, 2000.

[KL002] M. Kodialam and T. V. Lakshman, "Dynamic routing of bandwidth guaranteed tunnels with restoration," In *Proceedings of IEEE Infocom'2004*, pages 902-911, 2000.

[KR95] P. N. Klein and R. Ravi, "A nearly best-possible approximation algorithm for node- weighted Steiner trees," *J. Algorithms*, 19(1):104-114, 1995.

[Klein96] J. M. Kleinberg, "Single-source unsplittable flow," In *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science*, pp. 68-77, October 1996.

[KleinTh] J. M. Kleinberg, "Approximation algorithms for disjoint paths problems," *Ph.D. thesis*, M.I.T., 1996.

[KS02] S. G. Kolliopoulos and C. Stein, "Approximation algorithms for single-source unsplittable flow," *SIAM Journal on Computing*, 31:919-946, 2002.

[Koll04] S. G. Kolliopoulos, "Minimum-Cost Single-Source 2-Splittable Flow," *Information Processing Letters*, 94:15-18, 2005.

[Koll05] S. G. Kolliopoulos, "Edge-disjoint Paths and Unsplittable flow," *Handbook of Approximation Algorithms and Metaheuristics*, forthcoming, edited by T.R. Gonzalez.

[Kru56] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," In *Proceedings of of the American Mathematical Society* , 7:48-50, 1956.

[LMS90] 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, pages 105-115, 1990.

[LoR01] D. H. Lorenz and D. Raz, "A Simple Efficient Approximation Scheme for the Restricted Shortest Path Problem," *Operations Research Letters*, vol. 28, no. 5, pages 213-219, 2001.

[LiR01] S. Lindsey and C. Raghavendra, "Energy efficient broadcasting for situation awareness in ad hoc networks," In *Proceedings of Int. Conf. Parallel Processing (ICPP'01)*, Valencia, Spain, 2001.

[Lyn75] J. F. Lynch, "The equivalence of theorem proving and the interconnection problem," *ACM SIGDA Newsletter*, vol. 5, no. 3, 1975.

[MANET] MANET. IETF mobile Ad-hoc Network Working Group, MANET. *http://www.ietf.org/html.charters/manet-charter.html.*

[NTCS99] Sze-Yao Ni, Yu-Chee Tseng, Yuh-Shyan Chen, and Jang-Ping Sheu, "The Broadcast Problem in a Mobile Ad Hoc Network," In *Proceedings of ACM Mobicom* , Seattle, Washington, 1999.

[OS04] A. Orda and A. Sprintson, "Efficient Algorithms for Computing Disjoint QoS Paths," In *Proceedings of IEEE Infocom'2004*, Hongkong, March 2004.

[OS05] A. Orda and A. Sprintson, "Efficient Algorithms for Computing Disjoint QoS Paths," *IEEE/ACM Transactions on Networking*, To Be Published.

[PCV04] M. Patel, R. Chandrasekaran and S.Venkatesan, "Energy-efficient Capacity-constrained Routing in Wireless Sensor Networks," In *Proceedings of International Conference on Wireless Networks*, pages 447-453, Nevada, USA, June 21-24, 2004.

[PR04] P. Paul and S. V. Raghavan, "Survey of QoS Routing," In *Proceedings of the 15th International Conference on Computer Communications*, August 12-14, 2002.

[PS05] J. Park and S. Sahni, "Maximum Lifetime Routing In Wireless Sensor Networks," *IEEE Transactions on Computers*, 54(9):1081-1090, 2005.

[PS06] 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'2006)*, Mauritius, April 21-26, 2006.

[PS78] Y. Perl, Y. Shiloach, "Finding Two Disjoint Paths Between Two Pairs of Vertices in a Graph," *Journal of the ACM*, 25(1): 1-9, 1978.

[Prim57] R. C. Prim, "Shortest connection networks and some generalizations," *Bell System Technical Journal*, 36:1389-1401, 1957.

[PY00] C. H. Papadimitriou and M. Yannakakis, "On the Approximability of Trade-offs and Optimal Access of Web Sources," In *Proceedings of FOCS'2000*, pages 86-92, 2000.

[QVL02] A. Qayyum, L. Viennot, and A. Laouiti, "Multipoint relaying for flooding broadcast messages in mobile wireless networks," In *Proceedings of 35th Annual Hawaii International Conference on System Sciences (HICSS'02)*, Hawaii, USA, 2002.

[Rap96] Theodore S. Rappaport, *Wireless Communications: Principles and Practice.* IEEE Press, Piscataway, NJ, 1996.

[RM99] V. Rodoplu and T. H. Meng, “Minimum energy mobile wireless networks,” *IEEE Journal on Selected Areas in Communications* , 17(8), August 1999.

[RS95] N. Robertson, P. D. Seymour, “Graph Minors .XIII. The Disjoint Paths Problem,” *J. Comb. Theory, Ser. B*, 63(1): 65-110, 1995.

[SR02] Rahul C. Shah and Jan M. Rabaey, “Energy Aware Routing for Low Energy Ad Hoc Sensor Networks,” In *Proceedings of the IEEE WCNC’2002*, pages 350-355, 2002.

[SS01] C. Schurgers and M. B. Srivastava, “Energy efficient routing in wireless sensor networks,” In *Proceedings of the MILCOM on Communications for Network-Centric Operations: Creating the Information Force.*, pages 357-361, Virginia, 2001.

[Shi80] Yossi Shiloach, “A Polynomial Solution to the Undirected Two Paths Problem,” *Journal of the ACM*, 27(3):445-456, 1980.

[ST93] D. B. Shmoys and É. Tardos, “An approximation algorithm for the generalized assignment problem,” *Mathematical Programming A*, 62:461-474, 1993.

[Sku02] M. Skutella, “Approximating the single-source unsplittable min-cost flow problem,” *Mathematical Programming B*, 91:493-514, 2002.

[ST84] J. W. Suurballe and R. Tarjan , “A Quick Method for Finding Shortest Pairs of Disjoint Paths,” *Networks*, vol. 14, pages 325-336, 1984.

[Su74] J. W. Suurballe, “Disjoint Paths in a Network,” *Networks*, vol. 4, pages 125-145, 1974.

[SWC98] S. Singh, M. Woo, and C. S. Raghavendra, “Power-aware Routing in Mobile Ad hoc Networks,” In *Proceedings of ACM/IEEE Mobicom*, Dallas, Texas, October 1998.

[SX05] S. Sahni and X. Xu, “Algorithms For Wireless Sensor Networks,” *International Journal of Distributed Sensor Networks*, 1(1):35-56, 2005.

[WEX05] D. Wang, F. Ergun, and Z. Xu, “Unicast and Multicast QoS Routing with Multiple Constraints,” In *Proceedings of QoS-IP’2005*, pages 481-494, 2005.

[War62] Stephen Warshall, “A Theorem on Boolean Matrices,” *Journal of the ACM*, 9(1):11-12, 1962.

[WL99] J. Wu and H. Li, “A dominating-set-based routing scheme in ad hoc wireless networks,” In *Proceedings of 3rd Int’l Workshop Discrete Algorithms and Methods for Mobile Computing and Comm (DIALM’99)*, Seattle, USA, pages 7-14, 1999.

[WNE00] J. Wieselthier, G. Nguyen, and A. Ephremides, “On the construction of energy-efficient broadcast and multicast trees in wireless networks,” *Proceeding of IEEE Infocom’2000*, Tel Aviv, Israel, pages 585-594, 2000.

# Publications

[1] Chao Peng, Hong Shen, “A Storage-Aware Scheduling Scheme for VOD”, Springer-Verlag Lecture Notes in Computer Science 3251, October, 2004.

[2] Chao Peng, Hong Shen, “Storage-Aware Harmonic Broadcasting Protocol for Video-on-Demand”, Springer-Verlag Lecture Notes in Computer Science 3320, December, 2004.

[3] Chao Peng, Hong Shen, “A Localized Algorithm for Minimum-Energy Broadcasting Problem in MANET”, Springer-Verlag, Lecture Notes in Computer Science 3795, December, 2005.

[4] Chao Peng, Hong Shen, “An Improved Approximation Algorithm for Computing Disjoint QoS Paths”, in the 5th IEEE International Conference on Networking (ICN’06), April 21-26, 2006, Mauritius.

[5] Chao Peng, Hong Shen, “Approximation Algorithms for Inner-node Weighted Minimum Spanning Trees”, under review by Information Processing Letters.

[6] Chao Peng, Hong Shen, “Discrete Broadcasting Protocol for Video-on-Demand”, to appear in the 2006 International Conference on High Performance Computing and Communications, September 12-15, 2006, Munich, Germany.

[7] Chao Peng, Hong Shen, “Improved Approximation Algorithms for 2-Disjoint Restricted Shortest Path Problem”, accepted by IEICE TRANSACTIONS on Information and Systems.

[8] Chao Peng, Hong Shen, “A New Approximation Algorithm for Computing 2- Restricted Disjoint Paths”, under review by IEICE TRANSACTIONS on Information and Systems.

[9] Chao Peng, Hong Shen, “A New Approximation Algorithm for the Minimum-Cost Single-Source Unsplittable Flow Problem ”, submitted to IEICE TRANSACTIONS on Information and Systems.

[10] Chao Peng, Hong Shen, “An Improved Approximation Algorithm for Computing Minimum-Cost Single-Source k-Splittable Flow”, submitted to the Information Processing Letters.