UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

CONSTRAINED SHORTEST PATHS FOR QOS ROUTING AND PATH PROTECTION IN COMMUNICATION NETWORKS

A Dissertation

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirement for the

degree of

Doctor of Philosophy

By

YING XIAO
Norman, Oklahoma
2005

CONSTRAINED SHORTEST PATHS FOR QOS ROUTING AND PATH PROTECTION IN COMMUNICATION NETWORKS

A Dissertation APPROVED FOR THE
SCHOOL OF COMPUTER SCIENCE

BY

Krishnaiyan Thulasiraman, Committee Chair

Sudarshan Dhall

Sivaramakrishnan Lakshmivarahan

Sridhar Radhakrishnan

Theodore B. Trafalis

Acknowledgement

First, I thank my advisor Dr. Krishnaiyan Thulasiraman for his support and inspiration during my doctoral research. Krishnaiyan was always there for discussion and to give advice. He helped me become a real researcher from a student with only a little research experience. His research philosophy and approach to make complex problems simple and concise will accompany me and influence my future career. I also thank Dr. Guoliang Xue from Arizona State University and Dr. Christoforos Hadjicostis from University of Illinois. Working with them extended my knowledge to different areas and brought new ideas to our research.

Besides my advisor, I would like to thank all other members of the dissertation committee: Dr. Sudarshan Dhall, Dr. Sivaramakrishnan Lakshmivarahan, Dr. Sridhar Radhakrishnan, and Dr. Theodore Trafalis for their questions and comments on my research.

During the course of my doctoral research, I was supported by the school of computer science, NSF ITR grants ANI-0312435, NSF ITR grants ECS-0426831, and Hitachi Chair in Computer Science. A special thank goes to Dr. Ming Xue who provided me the initial financial support so that I could start my study at the University of Oklahoma.

Last, but not the least, I thank my family: my wife Mingjing Tong, my son Albert Xiao, my parents Mr. Shirong Xiao and Mrs. Keqiong Pan, and parents in-law Mr. Guangsheng Tong and Mrs. Shufen Zhou. Without their support and encouragement, the research would not have been possible.

Table of Contents

List of Tables

List of Figures

Abstract

We have studied certain combinatorial optimization problems that arise in the context of two important problems in computer communication networks: end-to-end Quality of Service (QoS) and fault tolerance. These problems can be modeled as constrained shortest path(s) selection problems on networks with each of their links associated with additive weights representing the cost, delay etc.

First we studied the QoS single route selection problem, i.e., the constrained shortest path (CSP) problem. The goal of the CSP problem is to identify a minimum cost route which incurs a delay less than a specified bound. It can be formulated as an integer linear programming (ILP) problem which is computationally intractable. The LARAC algorithm reported in the literature is based on the dual of the linear programming relaxation of the ILP formulation and gives an approximate solution. We proposed two new approximation algorithms solving the dual problem. Next, we studied the CSP problem using the primal simplex method and exploiting certain structural properties of networks. This led to a novel approximation algorithm.

The CSDP (k) problem requires the selection of a set of $k > 1$ link-disjoint paths with minimum total cost and with total delay bounded by a given upper bound. This problem arises in the context of provisioning paths in a network that could be used to provide resilience to link failures. Again we studied the LP relaxation of the ILP formulation of the problem from the primal perspective and proposed an approximation algorithm.

The problems considered above assume that the network status is known and accurate. However, in real networks, this assumption is not realistic. So we considered the QoS route selection problem under inaccurate state information. Here the goal is to find a

path with the highest probability that satisfies a given delay upper bound. We proposed a pseudo-polynomial time approximation algorithm, a fully polynomial time approximation scheme, and a strongly polynomial time heuristic for this problem.

Finally we studied the constrained shortest path problem with multiple additive constraints. Using the LARAC algorithm as a building block and combining ideas from mathematical programming, we proposed a new approximation algorithm.

Chapter 1. Introduction

This dissertation is concerned with the design of algorithms for different classes of constrained shortest path problems. In these problems path are required to satisfy certain pre-specified constraints on the path weights. These problems arise in i) the selection of routes that satisfy certain quality of service (QoS) guarantees and ii) the provision of alternate paths that provide protection against failures of links on the path.

Routing is a fundamental problem in communication networks. In traditional data networks, routing is achieved by best effort routing. Best effort routing is primarily concerned with providing connectivity. FIFO provides best-effort service. Here, flows are not differentiated and are serviced on a first-come, first-served basis. In best effort routing the routing protocol usually characterizes the network with a single metric such as hop-count or delay and uses a shortest path algorithm for path computation. Whereas the best-effort routing paradigm is adequate to serve the needs for traditional applications such as FTP (File Transfer Protocol) it is quite inadequate in providing the stringent quality of service (QoS) guarantees demanded by popular multimedia applications such as real time digital video or audio transmission. To support a broad range of QoS requirements, routing protocols need to consider more complex models that incorporate multiple metrics such as cost, delay, delay variation, loss probability, and bandwidth. This has triggered efforts towards proposals for QoS based frameworks such as DiffServe and IntServ, QoS routing protocols that accommodate multiple QoS requirements such as Q-OSPF and PNNI, and QoS routing algorithms (See [12, 15, 46, 52]). Despite these efforts, there is no standardized QoS routing protocol for the Internet.

To the best of our knowledge the only standardized QoS routing protocol is ATMF PNNI [46].

Two activities are involved in routing: *i*) capturing the network state information and disseminating the information throughout the network. This requires detection of significant changes, topology updates, distributed broadcasting (flooding) of the information to each node in the network etc. (*ii*) routing algorithms that compute the paths that satisfy certain performance guarantees.

In this dissertation we are concerned with the latter, namely, QoS routing algorithms. QoS measures can be classified into two types of metrics, non-additive (also called bottleneck, e.g., bandwidth) and additive constraints. Each measure is modeled by associating a weight with each link. For a non-additive measure QoS weight of a path is the minimum weight along the path. In the case of additive measures such as cost, delay, reliability, and delay-jitter, the QoS weight of a path is the sum of the QoS weights of the links on the path. Non-additive measures can be handled easily by simply removing from the network the links that do not satisfy the required QoS measure.

Fault tolerance is a topic of great interest in the study of communication networks. In the context of routing, a problem of importance is to select a set of disjoint paths between a source node and a destination node that provide path protection against one or more link failures. Here arises the problem of finding a set of disjoint path that satisfy certain QoS requirements.

The central theme of the dissertation is to study different classes of constrained shortest path problems that arise in the applications discussed above: QoS routing and path protection.

1.1. QoS Routing and Path Protection Problems

In this dissertation we are concerned with finding paths that satisfy additive QoS metrics. In particular, we are interested in the following classes of constrained shortest path problem that arise in QoS routing and path protection.

Consider a directed network $G(V, E)$ where V is the set of nodes and E is the set of links of the network. Each link $(u, v) \in E$ is associated with two integer weights $c_{uv} > 0$ (cost) and $d_{uv} > 0$ (delay). Also given are two nodes s and t. The cost $c(p)$ and delay $d(p)$ of a directed s-t path are defined as follows:

$$c(p) = \sum_{(u, v) \in p} c_{uv} \text{ and } d(p) = \sum_{(u, v) \in p} d_{uv}.$$

Given an integer $\Delta > 0$, a directed s-t path p is said to be feasible if $d(p) \leq \Delta$. In the following a directed s-t path will simply be called an s-t path.

Constrained Shortest Path (CSP) problem: Find an s-t path $p_{opt} = \arg\min\{c(p) | \ p$ is a feasible s-t path$\}$. This is illustrated with the example in Figure 1.1.

Constrained *k* Shortest Disjoint-Paths (CSDP(*k*)) problem: Here the objective is to find a set of k link-disjoint paths between a source node and a destination node with minimum total cost and with the total delay satisfying certain pre-specified bound. This problem arises in the context of providing alternate QoS paths to achieve protection against link failures.

Most Probable Delay Constrained Path selection under inaccurate information (MP-DCP): Objective here is to identify a path that has the highest probability of satisfying a delay bound. The delay of each link is a random variable. This problem is

of great importance since accurate state of a network (parameter information) is not often available.

Constrained Shortest Path Problem under Multiple Additive Constraints (CSP(k)): Suppose that each link (u, v) is associated with a set of k + 1 additive non-negative integer weights $\boldsymbol{C_{uv}} = (c_{uv}, w^1_{uv}, w^2_{uv}\ldots, w^k_{uv})$. Here c_{uv} is called the cost of link (u, v) and w^i_{uv} is called the ith delay of (u, v). For an s-t path p define

$$c(p) \equiv \sum_{(u,v)\in p} c_{uv} \text{ and } d_i(p) \equiv \sum_{(u,v)\in p} w^i_{uv},\ i = 1,\ldots k.$$

The value $c(p)$ is called the cost of path p, and $d_i(p)$ is called the ith delay of path p. Given k positive integers r_1, $r_2\ldots$, r_k, an s-t path is called feasible (*resp*. strictly feasible) if $d_i(p) \le r_i$ (*resp.* $d_i(p) < r_i$), for all i = 1, 2… k (r_i is called the bound on the ith delay of a path). The CSP(k) problem is to find a minimum cost feasible s-t path.

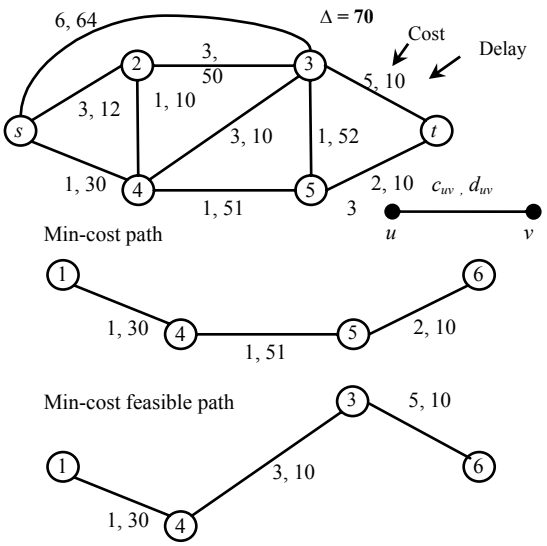


Figure 1.1: An example of the CSP problem

1.2. Overview of Literature

In this section we give an overview of literature on the four problems defined in the previous section.

1.2.1. The Constrained Shortest Path (CSP) Problem

It has been shown in [13, 58] that the CSP problem is NP-hard even for acyclic networks. So, in the literature, heuristic approaches and approximate algorithms have been proposed. Heuristics, in general, do not provide performance guarantees on the quality of the solution produced, though they are usually fast in practice. On the other hand, ε-approximation algorithms deliver solutions with cost within $(1 + \varepsilon)$ time the optimal cost, but are usually very slow in practice because they guarantee the quality of the solutions produced.

1.2.1.1. Heuristics with Performance Guarantees

As regards heuristics, several of them have appeared in the literature providing different levels of performance with regard to the quality of the solution as well as the computation time required. For instance, the LHWHM algorithm [37] is a simple heuristic which is very fast (requiring only two invocations of Dijkstra's shortest path algorithm for a feasible problem). Reference [48] also discusses further enhancements of the LHWHM algorithm as well as a heuristic based on the Bellman-Ford-Moore (BFM) algorithm for the shortest path problem. It should be emphasized that in all these cases, only simulations are used to evaluate the performance of the algorithms. Usually,

theoretical analysis is not given as regards the quality of the solution. A comprehensive overview of a number of QoS routing algorithms may be found in [9, 31].

There are heuristics that are based on sound theoretical foundations. These algorithms are based on solutions to the integer relaxation or the dual of the integer relaxation of the CSP problem. To the best of our knowledge, the first such algorithm was reported in [17] by Handler and Zang. This is based on the geometric approach (also called the hull approach [39, 69]). More recently, in an independent work, Jüttner etc. [23] developed the LARAC algorithm which solves the Lagrangian relaxation of the CSP problem (Here, the Lagrangian relaxation method is equivalent to the dual method). In contrast to the geometric method, they used an algebraic approach. They also presented several interesting results relating to the structure of the optimal solutions of the Lagrangian relaxation. In another independent work, Blokh and Gutin [6] defined a general class of combinatorial optimization problems (that are called the MCRT problems, namely, Minimum Cost Restricted Time Combinatorial Optimization problems) of which the CSP problem is a special case, and proposed an approximation algorithm to this problem. In recent work, Xiao etc. [60, 61] drew attention to the fact that the algorithms in [6] and [23] are equivalent. Mehlhorn and Ziegelmann [39] and Ziegelmann [69] have also observed this equivalence and have developed several insightful results. They arrived at these results using the hull (geometric) approach. In view of this equivalence, we shall refer to these algorithms as the LARAC algorithm. The work in [61] also establishes certain results using the algebraic approach. These results also hold true in the case of the general optimization problem considered in [6]. In another independent work, Xue [66] also arrived at the LARAC algorithm using the primal-dual method of

linear programming. A more recent variant of these approaches may be found in [30]. As regards computational complexity, in an unpublished work [25], Jüttner proves the strong polynomiality of the LARAC algorithm, both for the general case and for the CSP problem. He has used certain results from the general area of fractional combinatorial optimization. An application of the parametric search method to the general class of combinatorial optimization problems involving two additive parameters may be found in [24]. Radzik [47] gives an excellent exposition of approaches to fractional combinatorial optimization problems. Binary search based algorithms for the integer relaxation of the CSP problem are discussed in [30], [61] and [69]. They also establish the polynomial complexity of this approach using geometric and algebraic methodologies, respectively. Several interesting algorithms related to the CSP problem and motivated by applications have appeared in the literature. For examples, see [36] and [49].

1.2.1.2. ε-Approximation Algorithms:

A Fully Polynomial Time Approximation Scheme (FPTAS) is a type of approximation algorithms for optimization problems (most often, NP-hard optimization problems). A FPTAS for a minimization problem is an algorithm which takes an instance of an optimization problem and a parameter $\varepsilon > 0$ and produces a solution of an optimization problem that is within ε factor of being optimal and the running time of the algorithm is a polynomial of the problem size and $1 / \varepsilon$. We shall simply use ε-approximation algorithm to denote a FPTAS.

For a graph / network optimization problem, a FPTAS is called a Strongly Polynomial Time Approximation Scheme (SPTAS) if its running time is a polynomial of the number of nodes / edges and 1 / ε.

Approximation algorithms for CSP problem are usually based on scaling and rounding of data. Certain fundamental techniques presented by Sahni [50] and Ibarra and Kim [20] have been used by later researchers for designing ε-approximation algorithms for the CSP problem. To the best of our knowledge, Warburton [57] was the first to develop a fully polynomial time approximation algorithm for the CSP problem on acyclic networks. Hassin [18] later improved upon this to derive two fully polynomial time approximation schemes. His methods are applicable for general networks. The first one is based on a combination of dynamic programming and scaling/rounding and has a complexity of $O(\log\log(U/L)[mn\,\varepsilon^{-1} + \log\log(U/L)])$, where m and n are, respectively, the number of nodes and links in the network, and U and L are, respectively, an upper bound and a lower bound on the optimal cost. In a more recent work Lorenz and Raz [35] improved upon this result by giving a strongly polynomial time approximation scheme of complexity $O(mn\ (\log\log n + \varepsilon^{-1}))$. This is also applicable to general networks. The second algorithm of Hassin is based on the interval partitioning technique developed by Sahni [50]. This is applicable only to acyclic networks. In [45], Philips proposed another strongly polynomial time approximation scheme applicable for general networks. In a subsequent work, Hong, Chung and Park [19] drew attention to certain flaws in the second algorithm of Hassin and the algorithm of Philip's. Other related approximation schemes providing certain improvements to Hassin's algorithm may be found in [36]. In another interesting paper

[14], the authors considered the problem of determining a delay sensitive path whose delay is at most $(1 + \varepsilon)$ times the specified delay bound and whose cost is no greater than that of the minimum cost path of the CSP problem.

1.2.2. The Constrained *k* Shortest Disjoint-Paths (CSDP(*k*)) Problem:

Recall that the CSDP(k) problem is to select a set of k link-disjoint paths from s to t such that the total cost of these paths is minimum and that the total delay of these paths is not greater than a specified bound. This problem, being a generalized version of the CSP problem, is NP-hard. The CSDP(k) problem arises in the context of provisioning paths in a network that could be used to provide resilience to failures in one or more of these paths. Orda *et al.* [43] have studied the CSDP(2) problem extensively and have provided several approximation algorithms. A special case of the CSDP(k) problem which does not have the delay requirement has been studied in [54]. The algorithms in [6] and [54] can be integrated to provide an approximate solution to the CSDP(k) problem.

1.2.3. Most Probable Delay Constrained Path Selection under Inaccurate Information (MP-DCP Problem)

In the definition of the CSP problem it is assumed that the exact state of the network is known. However, in practice this is not the case. For several reasons [16, 28], full knowledge of the network state is not available. This has led researchers to study the

routing problem with uncertain parameters [16, 28, 34, 64]. The objective in these papers is to identify a path that is most likely to satisfy the delay requirement. This problem is referred to as the MP-DCP problem. In their pioneering works [16] and [34], the authors studied several aspects of this problem and related computational issues. Unlike the CSP problem that involves two deterministic metrics, namely, link cost and link delay, only one link metric, say delay, is considered in the MP-DCP problem. In [28], an approximate algorithm similar to the LARAC algorithm was proposed based on the assumption that the path delay is normally distributed. This assumption is fully justified by the central limit theorem and extensive numerical simulations. In [64], Xiao *et al*. proposed an exact algorithm, a FPTAS, and a strongly polynomial time approximation algorithm for the MP-DCP problem.

1.2.4. Constrained Shortest Paths Problem under Multiple Additive Constraints (CSP (k) Problem)

The CSP(k) problem is more general than the CSP problem in that it asks for a minimum cost path from a source node to a target node satisfying multiple constrains on the path weights. A variation of CSP(k) problem, called the Multi-Constrained Path (MCP) problem has also been a topic of extensive study. The difference between CSP(k) and MCP problems is that the MCP problem only asks for a path satisfying all the constraints simultaneously without the requirement of minimizing the cost. Several heuristics and approximation algorithms for the MCP problem can be found in [9, 22, 29, 40, 41, 42, 65, 67, 68].

Two methods for the CSP(k) problem based on mathematical programming have been proposed by Beasley and Christofides [2], and Mehlhorn and Ziegelmann [39]. Reference [2] uses a subgradient procedure to compute the Lagrangian relaxation function of the ILP formulation. With the geometrical interpretation of the algorithm of [17], the authors of [39] proposed an algorithm which is a special case of cutting planes method [51].

1.3. Scope of the Dissertation

The main contributions of the dissertation are organized into five chapters as follows.

Chapter 2: In this chapter we first present the CSP problem and the general class of optimization problems, namely the MCRT problem [6]. We then present the LARAC algorithm of [23] for the CSP problem and the MCRT algorithm of [6] for the MCRT problem. We point out the equivalence of the LARAC algorithm and the MCRT algorithm. We present an algebraic study of the integer relaxation of the CSP problem. In view of the equivalence of the LARAC and the MCRT algorithms, one would expect the results in [23] (stated without proof), though originally intended for the CSP problem, to hold true for the MCRT problem too. We establish these results and certain new results for the general case without involving the properties of shortest paths. We present a binary search approach for the CSP problem and also show that both the LARAC algorithm and this algorithm can be embedded with a tuning parameter whose value can be specified in advance depending on the allowable deviation of the cost of the path produced from the optimal cost. Finally we develop a strongly polynomial time

algorithm for the integer relaxation of the CSP problem. This is based on the parametric approach developed by Megiddo [38] for fractional combinatorial optimization problems. We conclude the chapter showing how one can integrate the LARAC algorithm with ε-approximation techniques to achieve considerable speedup of approximation algorithms. Simulation results demonstrating the value of the integrated approach are also presented.

Chapter 3: In this chapter we present a novel approach to the QoS routing problem, making a departure from currently available approaches. We study the problem using the primal simplex method of Linear Programming (LP) and exploiting certain structural properties of networks. We start with the Integer Linear Programming (ILP) formulation of the CSP problem and its integer relaxation formulation. The relaxed problem is the same as the LP formulation of the minimum cost flow problem [1, 4, 5] except for an additional constraint due to the delay requirement. This additional constraint gives rise to several questions that need to be investigated to achieve an efficient implementation of the primal simplex method. This leads us to the definition of an equivalent problem on a transformed network, called the TCSP problem. We discuss several issues that arise in the application of the revised simplex method of linear programming on the TCSP problem and strategies to achieve an implementation of the revised simplex method. This results in an algorithm allowing degenerate pivots and using an anti-cycling strategy specifically designed for the TCSP problem. Another algorithm called NBS algorithm avoids degenerate pivots completely. Both these algorithms are of pseudo polynomial complexity. We also show how to extract an approximate solution to the original CSP problem from the optimum solution to the

RELAX-TCSP problem and derive bounds on the quality of this solution with respect to the optimum solution. Finally experimental results comparing the NBS algorithm with the LARAC algorithm [23], the LHWHM algorithm [37], and general purpose LP solvers are presented.

Chapter 4: In this chapter we study the CSDP(k) problem which is also NP-hard. So our goal is to design an efficient algorithm for constructing an approximate solution to this problem. Towards this end, we study the LP relaxation of CSDP(k) problem using the revised simplex method of linear programming. This relaxed problem is an upper bounded LP problem. We have discussed several issues relating to an efficient implementation of our approach. We have shown that an approximate solution to the CSDP(k) problem can be extracted from an optimal solution to the relaxed problem. We have derived bounds on the quality of this solution with respect to the optimal solution. Our work can be considered as the study of the CSDP(k) problem from a primal perspective in contrast to the dual perspective employed in the G-LARAC(k) algorithm which is based on the algorithms in [23] and [54]. Simulation results comparing our algorithm with the general LP solvers are also presented.

We denote a general version of CSDP(k) problem as GCSDP(k) problem which requires that the delay of each individual path satisfies a specified bound, in contrast to the CSDP(k) problem where the constraint is on the total delay of all the k link-disjoint paths. We have shown that the LP relaxations of the two problems have the same optimal objective value. Thus, if one is interested in obtaining the optimal objective values of RELAX-GCSDP(k) and RELAX-CSDP(k) problems, then starting with the RELAX-CSDP(k) does not result in any loss of generality. However, the paths

produced by the approximate solution derived from the optimal solution to RELAX-CSDP(k) may not satisfy the individual path delay requirements of the GCSDP(k) problem. Our simulation results indicate that in most cases the individual delays of the paths produced starting from RELAX-CSDP(k) do not deviate in a significant way from the individual delay requirements of the GCSDP(k) problem.

Chapter 5: In this chapter we studied the stochastic shortest path problem aimed at identifying the most probable delay constrained path (MP-DCP problem). Our work is based on the formulation given in [28]. The work in [28] focused on developing approximate approaches using the Lagrangian relaxation or line search techniques. In contrast, our focus has been on developing polynomial time ε-approximation and heuristic algorithms. For the case (Case 1) when there is a path whose mean delay is less than or equal to the specified delay bound T, we presented an exact algorithm of pseudo polynomial time complexity, a FPTAS, and a strongly polynomial time heuristic algorithm. In the unlikely case (Case 2) when every path violates this assumption we have shown that the problem is NP-hard. We have also shown that for this case no pseudo polynomial time exact algorithm or fully polynomial time constant factor approximation algorithm is possible unless P = NP. The difficulty in this case arises because we need to find a path minimizing one path metric and maximizing another path metric simultaneously.

Chapter 6: In this chapter we present a new approach to the CSP(k) problem using Lagrangian relaxation. We first show that for $k = 1$, an approximation solution can be computed in $O((m + n \log n)^2)$ time which is better than the complexity of the Hull approach and the LARAC algorithm. Because this algorithm and the LARAC algorithm

are both based on the same methodology and obtain the same solution, we also denote our algorithm as LARAC. For arbitrary k, we use the LARAC algorithm as a building block and combine it with ideas from mathematical programming to achieve progressively higher values of the Lagrangian function. We present the resulting GEN-LARAC algorithm and prove its correctness and convergence properties. Simulation results comparing our algorithm with two other algorithms are presented. We conclude the chapter by pointing out that our approach is quite general and is applicable for the general class of combinatorial optimization problems studied in [6].

Chapter 2. The CSP Problem: Lagrangian Relaxation Based Algorithmic Approaches and an Algebraic Study

2.1. Introduction

Shortest path, minimum cost flow, and maximum flow computations are fundamental problems in operations research. Though interesting in their own right, algorithms for these problems also serve as building blocks in the design of algorithms for complex problems encountered in large scale industry applications. So, over the years there has been an extensive literature on various aspects of these two problems. Both these problems are solvable in polynomial time. But adding one or more additional additive constraints makes these problems intractable.

In this chapter, we focus on the constrained shortest path (CSP) problem. This problem requires determination of a minimum cost path from a source node to a destination node of a network subject to the condition that the total delay of the path be less than or equal to a specified value. The CSP problem has attracted considerable attention from different research communities: operations research, computer science, and telecommunications. The interest from the telecommunications community arises from the great deal of emphasis on the need to design communication protocols that deliver certain performance guarantees. This need, in turn, is the result of an explosive growth in high bandwidth real time applications that require stringent QoS guarantees. It is for this reason that the CSP problem has assumed great importance in telecommunication network applications.

The chapter is organized as follows. In Section 2.2, we present the CSP problem and the general class of optimization problems, namely the MCRT problem [6]. We also present the LARAC algorithm of [23] for the CSP problem and MCRT algorithm of [6]. We point out the equivalence of the LARAC algorithm and the MCRT algorithm. In Section 2.3 we present an algebraic study of the integer relaxation of the CSP problem. In view of the equivalence of the LARAC and the MCRT algorithms, one would expect the results in [23] (stated without proof), though originally intended for the CSP problem, to hold true for the MCRT problem too. We establish these results and certain new results for the general case without involving the properties of shortest paths. These results provide the basis for other algorithms considered in later sections. In Section 2.4, we present a binary search based approach for the CSP problem and also show that both the LARAC algorithm and this algorithm can be embedded with a tuning parameter whose value can be specified in advance depending on the allowable deviation of the cost of the path produced from the optimal cost. In Section 2.5, we develop a strongly polynomial time algorithm for the integer relaxation of the CSP problem. This is based on the parametric search approach developed by Megiddo [38] for fractional combinatorial optimization problems. Finally in Section 2.6, we show how the LARAC algorithm can be integrated with ε-approximation techniques to achieve considerable speedup of ε-approximation algorithms. Simulation results demonstrating the value of the integrated approach are also presented.

The results in this chapter have been repeated in [61].

2.2. The Constrained Shortest Path (CSP) Problem and Generality of the LARAC Algorithm

We first recall the definition of the CSP stated in Chapter 1.

Constrained Shortest Path Problem (CSP): Consider a network $G(V, E)$. Each link $(u, v) \in E$ is associated with two weights $c_{uv} > 0$ (say, cost) and $d_{uv} > 0$ (say, delay). Also are given two distinguished nodes s and t and a real number $\Delta > 0$. Let P_{st} denote the set of all s-t paths and for any s-t path p, define

$$c(p) = \sum_{(u,v)\in p} c_{uv} \quad and \quad d(p) = \sum_{(u,v)\in p} d_{uv} .$$

Let $P_{st}(\Delta)$ be the set of all the s-t paths p such that $d(p) \le \Delta$. A path in the set $P_{st}(\Delta)$ is called a feasible path. The CSP problem is to find a path $p^* = \arg\min\{c(p) | \, p \in P_{st}(\Delta)\}$. In other words, the CSP problem is to find a minimum cost feasible s-t path. It can be formulated as the following integer linear program.

CSP:

Minimize $\sum_{(u,v)\in E} c_{uv} x_{uv}$

subject to $\forall \, u \in V$,

$$\sum_{\{v|(u,v)\in E\}} x_{uv} - \sum_{\{v|(v,u)\in E\}} x_{vu} = \begin{cases} 1 & for \quad u = s \\ -1 & for \quad u = t \\ 0 & otherwise \end{cases}$$

$$\sum_{(u,v)\in E} - d_{uv} \cdot x_{uv} - w = -\Delta, w \ge 0$$

$$x_{uv} = 0 \; or \; 1, \; \forall (u,v) \in E$$

The CSP problem is known to be NP-hard [13, 58]. The main difficulty lies with the integrality condition that requires that the variables x_{uv} be 0 or 1. Removing or relaxing this requirement from the above integer linear program and letting $x_{uv} \geq 0$ leads to RELAX-CSP, the relaxed CSP problem. It is often convenient to solve the dual of the relaxed form of the CSP problem which we present below.

The dual involves *s*-*t* paths and a variable $\lambda \geq 0$. For each link (u, v), let the aggregated cost c_λ be defined as $c_{uv} + \lambda\, d_{uv}$. For a given λ, let $c_\lambda(p)$ denote the aggregated cost of the path p. Finally define $L(\lambda)$ as:

$$L(\lambda) = \min\{c_\lambda(p) |\, p \in P_{st}\} - \lambda\, \Delta. \tag{2.1}$$

Note that in the above, $\min\{c_\lambda(p) |\, p \in P_{st}\}$ is the same as the minimum aggregated cost of an *s*-*t* path with respect to a given value of λ. This can be easily obtained by applying Dijkstra's algorithm using aggregated link costs. Let the *s*-*t* path which has minimum aggregated cost with respect to a given λ be denoted as p_λ. Then $L(\lambda) = c_\lambda(p_\lambda) - \lambda\, \Delta$ and the dual of the RELAX-CSP can be presented in the following form.

DUAL-RELAX-CSP: Find $L^* = \max\{L(\lambda) \mid \lambda \geq 0\}$.

We note that the problem of maximizing $L(\lambda)$ as above is also called the Lagrangian dual problem. The value of λ that achieves the maximum $L(\lambda)$ in DUAL-RELAX-CSP will be denoted by λ^*. Note that L^*, the optimum value of DUAL-RELAX-CSP is a lower bound on the optimum cost of the path solving the corresponding CSP problem. The key issue in solving DUAL-RELAX-CSP is how to search for the optimal λ and determining the termination condition for the search. The LARAC algorithm of [23] presented in Figure 2.1 is one such efficient search procedure.

Procedure LARAC(s, t, d, Δ)
$p_c := Dijkstra(s,t,c)$
if $d(p_c) \le \Delta$ then return p_c
$p_d := Dijkstra(s,t,d)$
if $d(p_d) > \Delta$ then return "there is no solution"
repeat
$\lambda := \dfrac{c(p_c) - c(p_d)}{d(p_d) - d(p_c)}$
$r := Dijkstra(s,t,c_\lambda)$
if $c_\lambda(r) = c_\lambda(p_c)$ then return p_d
else if $d(r) \le \Delta$ then $p_d := r$ else $p_c := r$
end repeat
end procedure

Figure 2.1: LARAC algorithm

Description of the algorithm: In the LARAC algorithm of Figure 2.1, *Dijkstra*(s, t, c), *Dijkstra*(s, t, d), and *Dijkstra*(s, t, c_λ) denote, respectively, Dijkstra's shortest path algorithm using link costs, link delays, and aggregated link costs with respect to the multiplier λ.

1. In the first step, the algorithm calculates the shortest path on link costs. If the path found meets the delay constraint, this is surely the optimal path. Otherwise, the algorithm stores the path as the latest infeasible path, simply called the p_c path. Then it determines the shortest path on link delays denoted as p_d. If p_d is infeasible, there is no solution to this instance.
2. Set $\lambda = (c(p_c) - c(p_d))/(d(p_d) - d(p_c))$. With this value of λ, we can find a new c_λ-minimal path r. If $c_\lambda(r) = c_\lambda(p_c)$ $(= c_\lambda(p_d))$, we have obtained the optimal λ according to Claim 2.5 to be proved in Section 2.3. Otherwise, set r as the new p_c or p_d according to whether r is infeasible or feasible.

We next define the Minimum Cost Restricted Time Combinatorial Optimization (MCRT) Problem studied in [6].

Procedure MCRT (h)
$F := A(0, 1)$
if $x(F) \le h$ **then return** F.
$H := A(1, 0)$
if $x(H) > h$ **then return** "no solution"
repeat
$a := y(H) - y(F)$
$b := x(F) - x(H)$
$c := x(F)y(H) - x(H)y(F)$ (a)
$G := A(a, b)$
if $c = ax(G) + by(G)$ **then** (b)
if $x(G) \le h$ **then return** G **else return** H
if $c > ax(G) + by(G)$ **then** (c)
if $x(G) \le h$ **then** $H := G$ **else** $F := G$.
end repeat
end procedure

Figure 2.2: MCRT algorithm

Minimum Cost Restricted Time Combinatorial Optimization (MCRT) Problem: Given a finite set P, finite family set S of subsets of P, non-negative threshold h, and two non-negative real-valued functions y: $P \rightarrow R^+$ (say, cost) and x: $P \rightarrow R^+$ (say, delay). The MCRT problem is to seek a solution $F^* = \arg\min\{y(F) | F \in S, x(F) \le h\}$, where $z(G) = \sum_{g \in G} z(g)$ for z $\in \{x, y\}$ and $G \in S$.

Evidently, the CSP problem is a special case of the MCRT problem and so the MCRT problem is also NP hard. Therefore, we consider solving the integer relaxation of the MCRT problem. This is achieved by the MCRT algorithm given in [6] and

presented in Figure 2.2. In this algorithm, it is assumed that there is an effective algorithm $A(a, b)$ for the corresponding minimum cost problem with respect to $a\,x(p) + b\,y(p)$, $p \in S$, where a, b are the multipliers. For instance, in the case of the CSP problem, Dijkstra's algorithm for the minimum cost path problem can play the role of algorithm A. In Figure 2.2, algorithm $A(a, b)$ returns $p = \arg\min\{ax(r) + by(r) | r \in S\}$.

Equivalence of LARAC and MCRT Algorithms: Following the definition of the variables in Figure 2.1 and Figure 2.2, it can be seen that H corresponds to p_d while F corresponds to p_c and λ corresponds to a/b because

$$\frac{a}{b} = \frac{y(H) - y(F)}{x(F) - x(H)}.$$

Furthermore,

$$\frac{c}{b} = \frac{x(F)y(H) - x(H)y(F)}{x(F) - X(H)} = \frac{y(H) - y(F)}{x(F) - x(H)}x(F) + y(F) = y(F) + \frac{a}{b}x(F).$$

If the expressions (a), (b) and (c) in procedure MCRT are scaled by b, the MCRT algorithm reduces to the LARAC algorithm. In view of the equivalence of the LARAC algorithm and the MCRT algorithm, in the rest of the chapter we shall refer to both these algorithms as simply LARAC.

To conclude this section, to the best of our knowledge, the LARAC algorithm was first presented in [17]. More recently, Xue [66] presented another variant of this algorithm. Mehlhorn and Ziegelmann [39] and Ziegelmann [69] point out that the algorithm as presented in [6] can be derived from what they call the hull approach. Blokh and Gutin [6] also use geometric ideas in developing the MCRT algorithm. On

the other hand, Jüttner et al. [23] developed this algorithm using a purely algebraic approach.

2.3. An Algebraic Study of the Relax-CSP Problem and its Generalization

The LARAC algorithm as developed in [23] was originally intended for the CSP problem. In view of its generality as discussed in the previous section, one would expect that the claims in [23] (stated without proof) on which the LARAC algorithm is based do not depend on the properties of shortest paths. In other words, we would like to establish these claims without invoking properties of shortest paths. This is indeed true. In this section, we will present proofs of some of these claims for the sake of completeness. Furthermore, in the following section we also establish certain other new results that throw much insight into the structure of the solutions of the DUAL-RELAX-CSP problem. Though our proofs below do not involve shortest paths or their properties, we have decided to retain the terms such as "minimal path" whose interpretation in the general context should be obvious.

Claim 2.1[23]: Let $L(\lambda) = \min\{c_\lambda(p) | \ p \in P_{st}\} - \lambda \Delta$. Then $L(\lambda)$ is a lower bound to the optimum objective of the CSP problem for any $\lambda \geq 0$.

Claim 2.2[23]: L is a concave piecewise linear function, namely, the minimum of the linear functions $c(p) + \lambda(d(p) - \Delta)$ for all $p \in P_{st}$.

Claim 2.3[23]**:** For any $\lambda \geq 0$ and c_λ-minimal path p_λ, $d(p_\lambda)$ is a supgradient of L in the point λ.

Claim 2.4[23]: If $\lambda < \lambda^*$, then $d(p_\lambda) \geq \Delta$ and if $\lambda > \lambda^*$, then $d(p_\lambda) \leq \Delta$ for each c_λ-minimal path p_λ.

Proof: Let p and p^* denote a c_λ-minimal path and c_{λ^*}-minimal path respectively

$L(\lambda^*) = c(p^*) + \lambda^* d(p^*) - \lambda^* \Delta \leq c(p) + \lambda^* d(p) - \lambda^* \Delta = L(\lambda) + (\lambda^* - \lambda)(d(p) - \Delta)$.

Since $L(\lambda^*) \geq L(\lambda)$, $(\lambda^* - \lambda)(d(p) - \Delta) \geq 0$.

Therefore, if $\lambda < \lambda^*$ then $d(p_\lambda) \geq \Delta$ and if $\lambda > \lambda^*$ then $d(p_\lambda) \leq \Delta$ for each c_λ-minimal path p_λ. ■

Claim 2.5[23]: A value $\lambda > 0$ maximizes the function $L(\lambda)$ if and only if there are paths p_c and p_d which are both c_λ-minimal and for which $d(p_c) \geq \Delta$ and $d(p_d) \leq \Delta$ (p_c and p_d can be the same, in this case $d(p_d) = d(p_c) = \Delta$).

Proof: **a)** Proof of ***only if*** part: Suppose λ is the optimal value that maximizes $L(\lambda)$. Let p be the corresponding c_λ-minimal path and thus $L(\lambda) = c(p) + \lambda(d(p) - \Delta)$. Without loss of generality, we only consider the case $d(p) > \Delta$. If the λ is slightly increased to λ' ($> \lambda$), $c(p) + \lambda\ (d(p) - \Delta)$ is also increased. Since $L(\lambda)$ is optimal, p cannot be the $c_{\lambda'}$-minimal path any more; otherwise $L(\lambda') > L(\lambda)$. Let p' be the new $c_{\lambda'}$-minimal path. If $|\lambda - \lambda'|$ is small enough, p' is also the c_λ-minimal path because there are only a finite number of paths. It follows that $c(p') + \lambda'(d(p') - \Delta) = L(\lambda') \leq L(\lambda) = c(p') + \lambda\ (d(p') - \Delta)$.

Hence $\lambda'\ (d(p') - \Delta) \leq \lambda\ (d(p') - \Delta) \Rightarrow d(p') \leq T$ since $\lambda' > \lambda$.

Let $p_c = p$ and $p_d = p'$ completing the proof of the ***only if*** part.

b) Proof of ***if*** part**:** Let p_c and p_d be two c_λ-minimal paths and $d(p_c) \geq \Delta$ and $d(p_d) \leq \Delta$. Without loss of generality, assume λ^* maximizes the function $L(\lambda^*)$ and $\lambda^* > \lambda$.

Since $\lambda < \lambda^*$, $d(p_c) \geq \Delta$ and $d(p_d) \leq \Delta$, it follows that $d(p_d) = \Delta$.

Let p^* denote the c_{λ^*}-minimal path. Then,

$$L(\lambda^*) = c(p^*) + \lambda^* d(p^*) - \lambda^* \Delta \le c(p_d) + \lambda^* d(p_d) - \lambda^* \Delta$$
$$= L(\lambda) + (\lambda^* - \lambda)(d(p_d) - \Delta) \le L(\lambda)$$

Therefore, $L(\lambda) = L(\lambda^*)$, which proves that λ maximizes $L(\lambda)$. ■

Claim 2.6[23]: Let $0 \le \lambda_1 < \lambda_2$, and $p_{\lambda_1}, p_{\lambda_2} \in P_{st}$ be c_{λ_1}-minimal and c_{λ_2}-minimal paths. Then $c(p_{\lambda_1}) \le c(p_{\lambda_2})$ *and* $d(p_{\lambda_1}) \ge d(p_{\lambda_2})$.

Proof: Note that $c_\lambda(p) = c(p) + \lambda\, d(p)$.

Because $p_{\lambda_1}, p_{\lambda_2} \in P_{st}$ are c_{λ_1}-minimal and c_{λ_2}-minimal paths

$c_{\lambda_1}(p_{\lambda_1}) \le c_{\lambda_1}(p_{\lambda_2}) \Leftrightarrow c(p_{\lambda_1}) + \lambda_1 d(p_{\lambda_1}) \le c(p_{\lambda_2}) + \lambda_1 d(p_{\lambda_2})$, and

$c_{\lambda_2}(p_{\lambda_1}) \ge c_{\lambda_2}(p_{\lambda_2}) \Leftrightarrow c(p_{\lambda_1}) + \lambda_2 d(p_{\lambda_1}) \ge c(p_{\lambda_2}) + \lambda_2 d(p_{\lambda_2})$. Then

$$(\lambda_1 - \lambda_2) d(p_{\lambda_1}) \le (\lambda_1 - \lambda_2) d(p_{\lambda_2}) \Rightarrow d(p_{\lambda_1}) \ge d(p_{\lambda_2})$$
$$c(p_{\lambda_1}) \le c(p_{\lambda_2}) + \lambda_1 [d(p_{\lambda_2}) - d(p_{\lambda_1})] \le c(p_{\lambda_2}).$$

Hence the claim holds. ■

The convergence of the LARAC algorithm is guaranteed by the following result.

Claim 2.7[23]**:** Let $p_c^1, p_c^2, p_c^3, \ldots$ and $p_d^1, p_d^2, p_d^3, \ldots$ denote the sequences of paths generated by the LARAC algorithm. Then

$d(p_c^1) > d(p_c^2) > d(p_c^3) > \ldots > \Delta$ and $d(p_d^1) < d(p_d^2) < d(p_d^3) < \ldots \le \Delta$.

Proof: Suppose p_c and p_d are the current paths in the LARAC algorithm with λ_c and λ_d as the corresponding λ values. Suppose that neither of these two λ values is the maximizing value.

Let $\lambda = \dfrac{c(p_c) - c(p_d)}{d(p_d) - d(p_c)}$ and p_λ be the corresponding c_λ-minimal path.

Evidently, $c_\lambda(p_c) = c_\lambda(p_d)$ (recalling that $c_\lambda(p) = c(p) + \lambda\, d(p)$).

Suppose λ is not the maximizing value either; otherwise, the algorithm stops immediately. We also have

$$c(p_c) + \lambda_c\, d(p_c) \le c(p_d) + \lambda_c\, d(p_d),$$

$$c(p_c) + \lambda_d\, d(p_c) \ge c(p_d) + \lambda_d\, d(p_d).$$

In fact, the equality cannot hold because neither λ_c nor λ_d is the maximizing multiplier.

So $$\lambda_c < \frac{c(p_c) - c(p_d)}{d(p_d) - d(p_c)} = \lambda < \lambda_d\,.$$

Consider 2 cases:

1) $d(p_\lambda) \le \Delta$: In this case, because $d(p_\lambda) \ge d(p_d)$ by Claim 2.6, it suffices to show that $d(p_\lambda) \ne d(p_d)$.

Assume $d(p_\lambda) = d(p_d)$. Consider the following inequalities

$$c(p_\lambda) + \lambda\, d(p_\lambda) \le c(p_d) + \lambda\, d(p_d) \text{ and } c(p_\lambda) + \lambda_d\, d(p_\lambda) \ge c(p_d) + \lambda_d\, d(p_d).$$

Because $d(p_\lambda) = d(p_d)$, it follows that $c(p_\lambda) = c(p_d)$. Hence $c_\lambda(p_c) = c_\lambda(p_d) = c_\lambda(p)$, which implies that λ is the maximizing value. This contradiction establishes the theorem.

2) $d(p_\lambda) > \Delta$: Proof in this case follows along the same lines as above. ■

Theorem 2.1: Consider the problem:

$$\text{Minimize } y\, c(p_d) + (1 - y)\, c(p_c) \tag{2.2}$$

$$\text{subjects to } y\, d(p_d) + (1 - y)\, d(p_c) = \Delta \text{ and } 0 \le y \le 1, \tag{2.3}$$

where p_c and p_d are two *s*-*t* paths such that $d(p_d) > \Delta$ and $d(p_c) < \Delta$.

Let $\lambda = \dfrac{c(p_d) - c(p_c)}{d(p_c) - d(p_d)}$ and suppose that for all *s*-*t* path *p*, $d(p) \ne \Delta$.

Then p_d and p_c minimize (2.2) if and only if they both are c_λ-minimal.

Proof: First, we prove that

$$y\, c(p_d) + (1-y)\, c(p_c) \geq L(\xi),\ \xi \in R^+. \tag{2.4}$$

In fact,

$$\begin{aligned} &L(\xi) = \min\{c_\xi(p) \mid p \in P_{st}\} - \xi\Delta \\ &\leq y\, c_\xi(p_d) + (1-y)\, c_\xi(p_c) - \xi\,(y\, d(p_d) + (1-y)\, d(p_c)) \\ &= y\,(c_\xi(p_d) - \xi\, d(p_d)) + (1-y)(c_\xi(p_c) - \xi\, d(p_c)) \\ &= y\, c(p_d) + (1-y)\, c(p_c). \end{aligned}$$

Using (2.3), (2.2) can be rewritten as:

$$y\, c(p_d) + (1-y)\, c(p_c) = c(p_c) + \lambda\,(d(p_c) - \Delta) = c(p_d) + \lambda\,(d(p_d) - \Delta). \tag{2.5}$$

Evidently, $d(p_c) \neq \Delta$ and $d(p_d) \neq \Delta$.

a) Proof of the *if* part: Suppose p_d and p_c are c_λ-minimal paths. Then

$$L(\lambda) = c(p_c) + \lambda\,(d(p_c) - \Delta) = y\, c(p_d) + (1-y)\, c(p_c),$$

where $y\, d(p_d) + (1-y)\, d(p_c) = \Delta$, $0 \leq y \leq 1$. So (2.2) is minimized.

b) Proof of the *only if* part: Suppose p_d and p_c minimize (2.2) or rather (2.5). Assume p is a c_λ-minimal path and p_d and p_c are not c_λ-minimal. Consider the case when p is infeasible (if p is feasible, the theorem can be proven similarly). We have

$$c(p) + \lambda d(p) < c(p_d) + \lambda d(p_d). \tag{2.6}$$

Then

$$\lambda' = \frac{c(p_d) - c(p)}{d(p) - d(p_d)} > \lambda.$$

Thus

$$y'c(p_d) + (1-y')c(p) = c(p_d) + \lambda'(d(p_d) - \Delta)$$
$$< c(p_d) + \lambda(d(p_d) - \Delta) = y\,c(p_d) + (1-y)c(p_c),$$

where $y'd(p_d) + (1-y')d(p) = y\,d(p_d) + (1-y)d(p_c) = \Delta$.

The contradiction above proves that p_c and p_d are c_λ-minimal paths. ■

From the above proof, it can be shown that the value of λ defined by the optimal solution p_c and p_d of (2.2) is equal to the maximizing λ searched by LARAC algorithm. Also the optimum value of RELAX-CSP is equal to the optimum value $L(\lambda^*)$ of DUAL-RELAX-CSP.

There may be more than one maximizing λ. Assume that there is some multiplier λ such that the delay of the corresponding path p_λ is equal to the delay bound. In this case, an interval will serve as the maximizing multiplier and we can find the actual optimal path for the original CSP problem with that λ, recalling that $c(p_\lambda) = L(\lambda)$ which is the lower bound on the cost of the actual optimal path.

Theorem 2.2: If $\exists\ \lambda$ and the corresponding path p_λ such that $d(p_\lambda) = \Delta$, the maximizing λ is one unique interval (maybe just one point); Otherwise, the maximizing λ^* is unique.

***Proof*:** This is a direct consequence of the concavity of the function $L(\lambda)$ as stated in Claim 2.2. ■

Theorem 2.3: Given λ1 and λ2, such that $d(p_{\lambda 1}) > \Delta \geq d(p_{\lambda 2})$. If we start the LARAC algorithm by initializing p_c and p_d as $p_{\lambda 1}$ and $p_{\lambda 2}$, respectively, then the LARAC algorithm finds a maximizing multiplier λ* satisfying $\lambda 1 < \lambda^* \leq \lambda 2$.

2.4. LARAC-BIN: A Binary Search Based Approach to the DUAL-RELAX-CSP Problem

In this section we present a new algorithm called LARAC-BIN that uses the binary search technique to find the maximizing multiplier. LARAC-BIN as presented in Figure 2.3 stops when $L(\lambda^*) - L(\lambda) < \tau$. The parameter τ serves as a tuning parameter and can be specified in advance depending on the allowable deviation of the cost of the produced solution from the optimum value. We also establish an optimality condition. This criterion can be used to terminate the algorithm and at termination the optimum value of $L(\lambda)$ will be obtained.

Procedure LARAC-BIN (s,t,Δ,τ)
  $p_c := Dijkstra(s,t,c)$
  if $d(p_c) \le \Delta$ **then return** p_c
  $p_d := Dijkstra(s,t,d)$
  if $d(p_d) > \Delta$ **then return "**there is no solution**"**
  if $d(p_d) = \Delta$ **or** $c(p_d) = c(p_c)$ **then return** p_d
  $\lambda_{begin} := 0, \lambda_{end} := (c(p_d) - c(p_c))/(\Delta - d(p_d))$
  while $(\lambda_{end} - \lambda_{begin})(\Delta - d(p_d)) > \tau$
    $\lambda := (\lambda_{begin} + \lambda_{end})/2$
    $r := Dijkstra(s,t,c_\lambda)$
    if $d(r) = \Delta$ **then return** r
    else if $d(r) < \Delta$ **then** $\lambda_{end} := \lambda$ **else** $\lambda_{begin} := \lambda$
  end while
  return $r := Dijkstra(s,t,c_{\lambda_{end}})$
end procedure

Figure 2.3: LARAC-BIN algorithm

In effect, the goal of the LARAC-BIN is to find the minimum λ with which we can obtain a feasible path because the smaller the λ, the smaller the cost of the path obtained. This goal is compatible with that of the LARAC algorithm searching for the maximizing λ* and $L(\lambda^*)$. To put it formally, we have the following theorem.

Theorem 2.4: Let λ* denote the smallest maximizing value for $L(\lambda)$ and p_λ denote a path corresponding to λ. Then $c(p_{\lambda^*}) \le c(p_\lambda)$ for all λ such that $d(p_\lambda) \le \Delta$.

Proof: According to Claim 2.6, if $\lambda^* \le \lambda$, $c(p_{\lambda^*}) \le c(p_\lambda)$. So assume $\lambda^* > \lambda$.

In this case, $d(p_\lambda) \le \Delta$ implies $d(p_\lambda) = T$ by Claim 2.4. Hence $L(\lambda) = L(\lambda^*)$ according to Claim 2.5, which is impossible because λ* is the smallest maximizing value for $L(\lambda)$.

The above contradiction proves the theorem. ■

The initial values of λ_{begin} and λ_{end} in Figure 2.3 are to be selected such that p_{begin} is infeasible and p_{end} is feasible. We can initialize λ_{end} as in the following theorem.

Theorem 2.5: If $\lambda = \dfrac{c(p_d) - c(p_c)}{\Delta - d(p_d)}$, $d(p_d) < \Delta$ and $c(p_d) > c(p_c)$, then the c_λ-minimal path is feasible, where p_c and p_d are the minimal cost and minimal delay path, respectively.

***Proof*:** Assume that p is a c_λ-minimal path and $d(p) > \Delta$. It follows that

$$c(p_d) + \lambda\, d(p_d) \ge c(p) + \lambda\, d(p).$$

Then

$$0 \le c(p_d) - c(p) - \frac{c(p_d) - c(p_c)}{\Delta - d(p_d)}(d(p) - d(p_d))$$

$$< c(p_d) - c(p) - (c(p_d) - c(p_c)) = c(p_c) - c(p) \le 0.$$

The above contradiction proves the theorem. ■

Theorem 2.6: Let λ* denote the smallest maximizing Lagrangian multiplier of $L(\lambda)$ and p^* be the resulting path. Let p_{begin} and p_{end} be the minimal aggregated cost paths with respect to λ_{begin} and λ_{end}, where λ_{begin} and λ_{end} are as defined in the LARAC-BIN algorithm in Figure 2.3. Here p_{begin} is infeasible and p_{end} is feasible. Then

$$0 \le L(\lambda^*) - L(\lambda_{end}) \le (\lambda_{end} - \lambda_{begin})(\Delta - d(p_{end})) .$$

Proof: The left inequality holds because $L(\lambda^*)$ is the maximum value.

Evidently, $d(p_{end}) \le \Delta$, $\lambda_{begin} \le \lambda^* \le \lambda_{end}$, and

$$c(p^*) + \lambda^* d(p^*) \le c(p_{end}) + \lambda^* d(p_{end}).$$

It follows that

$$\begin{aligned} L(\lambda^*) - L(\lambda_{end}) &= c(p^*) + \lambda^* d(p^*) - \lambda^* \Delta - [c(p_{end}) + \lambda_{end} d(p_{end}) - \lambda_{end} \Delta] \\ &= \{c(p^*) + \lambda^* d(p^*) - [c(p_{end}) + \lambda^* d(p_{end})]\} - (\lambda_{end} - \lambda^*) d(p_{end}) + (\lambda_{end} - \lambda^*)\Delta \\ &\le (\lambda_{end} - \lambda^*)(\Delta - d(p_{end})) \le (\lambda_{end} - \lambda_{begin})(\Delta - d(p_{end})). \end{aligned}$$ ■

Note that we have used the result of the above theorem in the termination of the LARAC-BIN algorithm (Figure 2.3).

Since a number of optimization problems only involve integer values (integer problems) or can be converted to integer problems, we now derive a termination condition for the LARAC-BIN algorithm when all the link costs and delays are integers. If terminated according to this condition, the algorithm computes the maximizing λ* with polynomial time complexity.

Consider the set of rational numbers $Q(D) = \{p / q \mid \text{GCD}(p, q) = 1, q \le D, \text{and } p, q, D \in N^+\}$. Define the density of $Q(D)$ as $\text{DENS}(Q(D)) = \min\{|x_1 - x_2|: x_1, x_2 \in Q(D)$ and

$x_1 \neq x_2\}$. It is easy to show that DENS($Q(D)$) =$1/D^2$ and that for $x, y \in Q(D)$, $x = y$ if $|x - y| <$ DENS($Q(D)$).

Suppose that we modify LARAC-BIN so that it terminates when $| \lambda_{begin} - \lambda_{end}| < 1 / D^2$ and that the paths at termination are p_{end} and p_{begin}, where $D = |d(p_{begin}) - d(p_{end})|$. Let

$$\lambda = \lambda' = \frac{c(p_{end}) - c(p_{begin})}{d(p_{begin}) - d(p_{end})}.$$

Theorem 2.7: λ' defined as above is a maximizing multiplier.

Proof: Consider $Q(D)$, where $D = |d(p_{begin}) - d(p_{end})|$. Because

$$c(p_{begin}) + \lambda_{begin} d(p_{bebin}) \leq c(p_{end}) + \lambda_{begin} d(p_{end}) \text{ and}$$
$$c(p_{begin}) + \lambda_{end} d(p_{bebin}) \geq c(p_{end}) + \lambda_{end} d(p_{end}),$$

$$\lambda_{begin} \leq \lambda' = \frac{c(p_{end}) - c(p_{begin})}{d(p_{begin}) - d(p_{end})} \leq \lambda_{end}.$$

Suppose that $\lambda_{\text{begin}} \leq \lambda^* \leq \lambda_{end}$, where λ* is the maximizing Lagrangian multiplier obtained by LARAC algorithm initialized with $p_c = p_{begin}$ and $p_d = p_{end}$.

Clearly $\lambda^* = (c(p_{\lambda 1}) - c(p_{\lambda 2})) / (d(p_{\lambda 2}) - d(p_{\lambda 1}))$ for some paths $p_{\lambda 1}$ and $p_{\lambda 2}$ w.r.t. the Lagrangian multipliers λ1 and λ2. It can be seen that λ1 and λ2 $\in [\lambda_{begin}, \lambda_{end}]$ following the similar argument above. Hence $| d(p_{\lambda 2}) - d(p_{\lambda 1})| \leq D$ according to Claim 2.6, i.e., λ* $\in Q(D)$.

Evidently $|d(p_{begin}) - d(p_{end})| = D \leq D$ and thus $\lambda \in Q(D)$.

Because $| \lambda' - \lambda^*| < |\lambda_{begin} - \lambda_{end}| < 1 / D^2 =$ DENS ($Q(D)$), the only possibility is that λ' = λ*. ■

For the CSP problem, the size of D is bounded as $D \leq n \max \{d_{ij} \mid (i,j) \in E\}$, where n is the number of nodes in the network. If the LARAC-BIN algorithm is terminated using the condition given above, then we have the following complexity result.

Theorem 2.8: LARAC-BIN terminates in $O((m + n \log n)(\log (COST \times DELAY^2)))$ time where *COST* is the cost of the minimum delay path and *DELAY* is the delay of the minimum cost path in the network.

2.5. Strong Polynomiality of DUAL-RELAX-CSP: A Parametric Search Based Algorithm

In an unpublished work, Jüttner [25] has shown that the LARAC algorithm for DUAL-RELAX-CSP is strongly polynomial. We wish to recall that the time complexity of an algorithm for a graph/network problem is strongly polynomial if the computational time is a polynomial of only m and n, where m and n are respectively the number of links and the number of nodes in the graph/network. In this section, we present another strongly polynomial time algorithm, namely the PSCSP (Parametric Search based Constrained Shortest Path) algorithm (Figure 2.4), for solving DUAL-RELAX-CSP. This method is based on a methodology first proposed by Megiddo [38] to solve fractional combinatorial optimization problems. In this section, we only handle the shortest path problem without generalization due to the nature of the parametric search. The algorithm PSCSP in Figure 2.4 is based on the BFM algorithm.

Let $\lambda^* \geq 0$ denote the maximizing Lagrangian multiplier for the $L(\lambda)$ function. Assume node 1 is the source node and node n is the sink node. Each node v of the

network is associated with a pair $M_v = (x_v, y_v)$, where x_v and y_v keep track of the cost and delay of some 1-v path during the execution of the PSCSP algorithm. M is initialized as $M_1 = (0, 0)$ and $M_v = (\infty, \infty)$ for $v \neq 1$. The algorithm computes the c_{λ^*}-minimal $1 - n$ path. This algorithm does not guarantee the feasibility of the obtained path. In order to get a feasible c_{λ^*}-minimal $1 - n$ path, we can revise the BFM algorithm using lexicographic ordering on the combined link costs and link delays [30, 53]. We shall give the details of the algorithm to compute a feasible c_{λ^*}-minimal path in Chapter 6.

Procedure PSCSP (s, t, Δ)
  $M_v = (x_v, y_v) = (\infty, \infty)$ for $v = 2, \ldots n$
  $M_1 = (0,0)$
  for i $\leftarrow$ 1 to n - 1 do
   for $u \leftarrow 1$ to n do
     for each v such that $(u, v) \in E$ do
      *[if $(x_v + \lambda * y_v$
            $\geq x_u + \lambda * y_u + c_{uv} + \lambda * d_{uv})$] (2.7)
        $M_v \leftarrow (x_u + c_{uv}, y_u + d_{uv})$

Figure 2.4: PSCSP Algorithm (λ* is unknown)

In the algorithm in Figure 2.4, we need extra steps to decide whether the Boolean expression (2.7) (it is called oracle test) is true or false since λ* is unknown.

If $x_v = \infty$, $y_v = \infty$, then the inequality holds. Assume x_v and y_v are finite non-negative values. Then it suffices to evaluate the following Boolean expression.

$$(x_u + c_{uv} - x_v) + \lambda^*(y_u + d_{uv} - y_v) = p + q\,\lambda^* \leq 0, \qquad (2.8)$$

where $p = x_u + c_{uv} - x_v$ and $q = (y_u + d_{uv} - y_v)$.

If $p \cdot q \geq 0$, then it is trivial to tell whether (2.8) holds or not. Suppose $p \cdot q < 0$, i.e., $-p/q > 0$.

Let $\lambda = -p/q$ and let $r = Dijkstra(s, t, c_\lambda)$, where *Dijkstra* computes a c_λ-minimal path.

Now consider three cases:

a) $d(r) > \Delta$: By Claim 2.4 of Section 2.3, $\lambda \leq \lambda^*$ and thus (2.8) can be decided according to whether q is positive or negative.

b) $d(r) < \Delta$: By Claim 2.4, $\lambda \geq \lambda^*$ and (2.8) can be evaluated similarly.

c) $d(r) = \Delta$: Return the path r as the optimal path (by Claim 2.5).

If PSCSP is based on Dijkstra's algorithm, instead of the BFM algorithm, the complexity of the resultant algorithm is reduced to $O((m + n \log n)^2)$. Thus we have the following result.

Theorem 2.9: The parametric search algorithm PSCSP for DUAL-RELAX-CSP is strongly polynomial with time complexity $O((m + n \log n)^2)$. ■

In the implementation of the PSCSP algorithm, the number of invocations of Dijkstra's algorithm is reduced by maintaining an interval $[a, b]$ containing λ^*, where a is the maximum known value of $-p/q < \lambda^*$ and b is the minimum known value of $-p/q > \lambda^*$ during the execution of the algorithm. We only need to call *Dijkstra* algorithm for λ within the interval $[a, b]$ and update the interval accordingly. A discussion of the application of the parametric approach to the general class of optimization problems involving two additive parameters may be found in [24].

2.6. Closing the Gap: An Integrated Approach to ε-Approximation Algorithm Design for the CSP Problem

In this section, we show how the LARAC algorithm can be used to considerably speed up an ε-approximation scheme. A few definitions are now in order.

An approximation algorithm for a minimization problem obtains a solution whose cost is within a specified multiple of the optimum cost. This idea is formally stated as follows [50].

An approximation scheme for a problem P is an algorithm that, given an instance I and a desired degree of accuracy $\varepsilon > 0$, constructs a problem solution with value $\hat{F}(I)$, such that, if $F^*(I) > 0$ is the value of an optimal solution to I, then

$$\frac{|F^*(I) - \hat{F}(I)|}{F^*(I)} \leq \varepsilon .$$

A fully polynomial time approximation scheme for a graph/network optimization problem is an approximation scheme whose computing time is a polynomial function of the input size and $1/\varepsilon$. A strongly polynomial time approximation scheme for a graph/network optimization problem is an approximation scheme whose computing time is a polynomial function of the number of nodes and $1/\varepsilon$.

In the literature, there has been an extensive discussion of approximation algorithms for the CSP problem. Of particular interest to us are Hassin's algorithm [18] and the more recent algorithm due to Lorenz and Raz [35]. Hassin presents a fully polynomial time ε-approximation and Lorenz and Raz present a strongly polynomial time

approximation scheme (denoted as SEA algorithm). There are two phases in the design of approximation algorithms:

Phase1:

Start with an interval [*LB*, *UB*] where *LB* and *UB* are lower and upper bounds to the objective value of the optimum solution to the CSP problem, and iteratively shrink the interval until the ratio of the upper bound and the lower bound is below some constant (say, 2). This is achieved using a combination of a dynamic programming algorithm and a test procedure to determine whether the optimum is greater than or equal to a specified value.

Phase 2:

Determine an ε-approximate solution using the dynamic programming algorithm with the lower and upper bounds obtained in the phase 1.

Since LARAC/LARAC-BIN is very fast, we can use them to construct Phase 1. This considerably improves the computational time over the original ε-approximation algorithm which does not use LARAC for the first phase. The details of this integration are given below.

LARAC algorithm terminates with two paths p_c and p_d one of which is feasible, denoted by p_d, and the other is infeasible, denoted by p_c. It is easy to see that the cost of the infeasible path is the lower bound and the cost of the feasible path is the upper

bound on the optimal cost. The cost of p_c at termination of LARAC is also a lower bound on the cost of the optimal path to the CSP problem. Given a parameter ε, if the cost of p_d at termination is less than $(1 + \varepsilon)\ c(p_c)$, then p_d is an ε-approximation to the CSP problem. If this is not the case, then the paths p_c and p_d can be used to get the initial lower and upper bounds required by ε-approximation algorithms. The integrated algorithm incorporating the above ideas is presented in Figure 2.5. Here we have used the SEA algorithm presented in [35] for Phase 2.

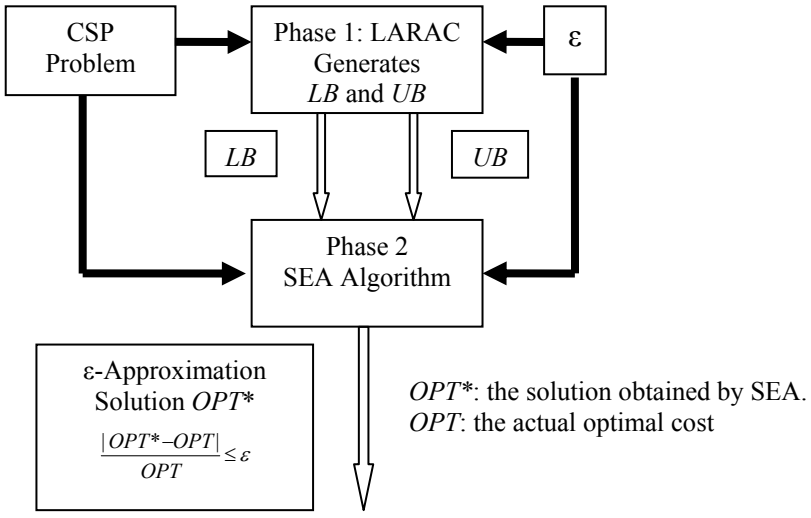
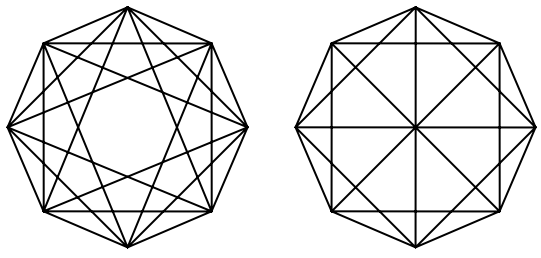


Figure 2.5: An integrated approximation algorithm: LARAC + SEA



(a) $H_{6,8}$ (b) $H_{7,9}$

Figure 2.6: $H_{k,n}$ graphs

We next discuss results of our simulation of the integrated approach. In our experiments we have used regular graphs $H_{k,n}$ (See Figure 2.6) proposed by Harary (See [55]), where k is the degree and n is the number of nodes, respectively. The link costs are randomly generated integers in the range 2 to 198 and delays are assigned values as follows: $d_{ij} = 200 - c_{ij}$, where c_{ij} and d_{ij} are the cost and delay of link (i, j), respectively. For each pair of vertex and degree, 10 experiments are carried out and the average value is given in Table 2.1.

As we can see from column six in the table, the ratio of the cost of p_d and the cost of p_c returned by LARAC is very close to 1. This is much better than the ratio of 2 which Phase 1 tries to achieve. Column seven shows that the total time for Phase 1 (when LARAC is used) is only about 5% of the total running time. We also note that Phase 1 when LARAC is used takes only 0.1% of the time for Phase 1 when the dynamic programming approach is used. Furthermore, we can also see from the last column in the table that the integrated approach achieves a speedup of 6.

Table 2.1. Simulation results

R = the ratio of the cost of p_d and the cost of p_c returned by LARAC
LT = the ratio of the time used by LARAC and the total running time (LARAC + SEA)
T = the ratio of the time used by LARAC+SEA and the time used by pure SEA algorithm

NODE	DEG	ε	SEA	LARAC+SEA				
			Cost	LARAC			Cost	T
				Cost	R	LT		
1000	6	.05	13290	13290	1.1	.005	13388	.16
1000	16	.05	9696	9748	1.1	.002	9696	.27
1000	32	.05	5946	6196	1.2	.004	5966	.14
2000	6	.05	28002	29888	1.1	.002	28000	.14
2000	16	.05	19704	20222	1.1	.003	19704	.10
2000	32	.05	11634	11778	1.1	.002	11636	.12

2.7. Summary

In this chapter, we have studied several aspects of the constrained shortest path (CSP) problem. This is an NP-complete problem and so in the literature, the focus has been on solving the integer relaxation of the problem called RELAX-CSP. We first pointed out the equivalence of the algorithms presented in [6], [17] and [23]. In view of this equivalence, we call these algorithms simply as the LARAC algorithm. Whereas the algorithms in [17] and [23] were intended for the CSP problem, the one in [6] was intended for a general class of combinatorial optimization problems (MCRT problem) involving two additive parameters. Using an algebraic approach, we have shown in Section 2.3 that all the claims in [23] also hold for the MCRT problem. We have also established certain new results on the properties of the solutions obtained by the LARAC algorithm. In particular, we have shown that the paths p_c and p_d that result at the termination of LARAC have an interesting property and, in fact, solve another optimization problem (Theorem 2.1). In Section 2.4, we presented a heuristic called LARAC-BIN based on binary search. The new heuristic involves a tuning parameter whose value can be specified in advance depending on the allowable deviation of the cost of the path produced by the heuristic from the optimum value. Whereas binary search is a commonly employed technique for algorithm design, incorporation of the tuning parameter as in LARAC-BIN enhances the value of the binary search based approaches.

In Section 2.5, we presented a strongly polynomial time algorithm for DUAL-RELAX-CSP. This algorithm is based on Megiddo's parametric search method [38] and

certain techniques from fractional combinatorial optimization [47]. To the best of our knowledge, this algorithm has the best time complexity to date for DUAL-RELAX-CSP.

In Section 2.6, we pointed out how LARAC and LARAC-BIN can be used in conjunction with ε-approximation techniques to generate paths whose costs are guaranteed to be within certain factor of the optimum. The value of $L(\lambda)$ at termination of these algorithms is a lower bound on the cost of the optimum solution to the CSP problem. Given a parameter ε, if the cost of the path p_d at termination is less than $(1 + \varepsilon)$ $c(p_c)$, then p_d is an ε-approximation to the CSP problem. If this is not the case, then the paths p_c and p_d can be used to generate lower and upper bounds needed for an ε-approximation algorithm. An integrated approach to the design of ε-approximation algorithms based on these ideas has been presented in Section 2.6. Effectiveness of this integrated approach has been illustrated through simulation.

Chapter 3. The CSP Problem: Approximation Algorithms Based on the Primal Simplex Method of Linear Programming

The approaches discussed in Chapter 2 including the LARAC algorithm for the CSP problem are based on the dual of the integer relaxation of the ILP formulation of the CSP problem. In this chapter, we present a novel approach to the CSP problem, making a departure from these currently available approaches. We study the problem using the primal simplex method of linear programming and exploiting certain structural properties of networks.

The chapter is organized as follows. In Section 3.1, we define the CSP problem and present its integer linear programming (ILP) formulation as well as its linear programming (LP) relaxation. This formulation is the same as the LP formulation of the minimum cost flow problem [1] except for an additional constraint due to the delay requirement. This additional constraint gives rise to several questions that need to be investigated to achieve an efficient implementation of the primal simplex method. This leads us to the definition in Section 3.2 of an equivalent problem on a transformed network, called the TCSP problem. Section 3.3 deals with the structure of the basic solutions of the RELAX-TCSP problem, the relaxed form of the TCSP problem. Section 3.4 discusses the revised simplex method of linear programming, its application on RELAX-TCSP, and several strategies to achieve an efficient implementation. This results in an algorithm allowing degenerate pivots and using an anti-cycling strategy developed in Section 3.4.6. Another algorithm called NBS algorithm presented in Section 3.5 avoids degenerate pivots completely. Both these algorithms are of pseudo

polynomial complexity. In Section 3.5.3.2, we show how to extract an approximate solution to the original CSP problem from the optimum solution to the RELAX-TCSP problem and derive bounds on the quality of this solution with respect to the optimum solution. In Section 3.6, experimental results comparing the NBS algorithm with the LARAC algorithm [23], the LHWHM algorithm [37], and the general purpose LP solvers are presented. Section 3.7 concludes with a summary of the main contributions.

The results in this chapter have been repeated in [63].

3.1. The CSP Problem: LP Formulation and Integer Relaxation

We first recall the definition of the CSP problem and its ILP formulation.

Definition 3.1: Consider a directed network $G(V, E)$ where V is the set of nodes and E is the set of links of the network. Each link $(u, v) \in E$ is associated with two integer weights $c_{uv} > 0$ (representing cost, the expense imposed by using or installing the link) and $d_{uv} > 0$ (transmission delay along the link). For any path p (or cycle with a given orientation) define the cost $c(p)$ and delay $d(p)$ of p as

$$c(p) = \sum_{(u,v) \in p^+} c_{uv} - \sum_{(u,v) \in p^-} c_{uv} \text{ and } d(p) = \sum_{(u,v) \in P^+} d_{uv} - \sum_{(u,v) \in P^-} d_{uv} ,$$

where p^+ (p^-) is the set of forward (backward) links on p as we traverse p from the start node to the end node of p.

Notice that our assumption that link weights are integers does not involve any loss of generality because, in digital systems, all numbers are represented discretely and can be scaled and rounded to integers. In order to simplify our presentation, we assume all the values to be integers. We also assume that only links impose costs and delays. If the

nodes impose costs and delays, we can use the node splitting technique to transform node costs and delays into link costs and delays (See Chapter 2.4 of [1]).

We use the terms "link" and "arc" interchangeably. Without loss of generality, we assume that for every node i, there is a directed path from i to the destination node t. In the rest of the chapter, $m = |E|$ and $n = |V|$.

A path is called a directed path (cycle) if there are no backward links in the path (cycle). Given two nodes s, t and an integer $\Delta > 0$, a directed s-t path p is said to be feasible if $d(p) \leq \Delta$.

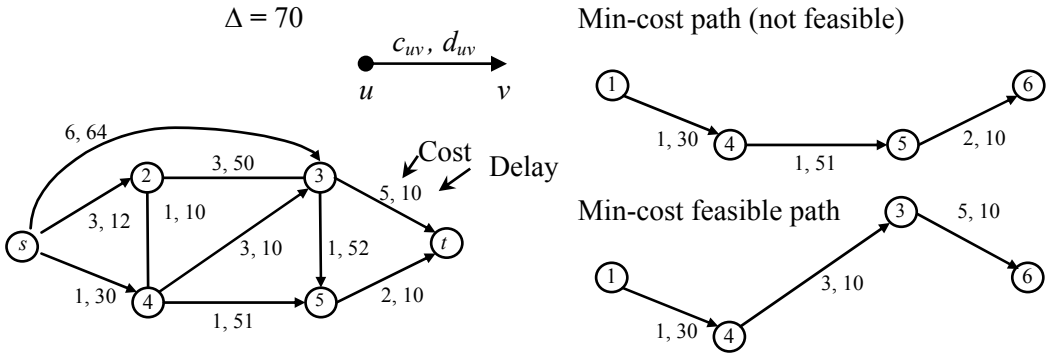


Figure 3.1: An example of CSP problem

Constrained Shortest Path (CSP) problem: Find an s-t path $p_{opt} = \arg\min\{c(p) | p$ is a feasible s-t path$\}$. This is illustrated with the example in Figure 3.1.

The CSP problem can be formulated as an ILP problem as follows.

CSP: Minimize $\sum_{(u,v)\in E} c_{uv} \cdot x_{uv}$ (3.1)

subject to

$$\forall\, u \in V,\ \sum_{\{v|(u,v)\in E\}} x_{uv} - \sum_{\{v|(v,u)\in E\}} x_{vu} = \begin{cases} 1, & if \quad u = s; \\ -1, & if \quad u = t; \\ 0, & otherwise. \end{cases} \tag{3.2}$$

$$\sum\nolimits_{(u,v)\in E} - d_{uv} \cdot x_{uv} - w = -\Delta\,, \tag{3.3}$$

$$\forall\, (u, v) \in E,\ x_{uv} = 0 \text{ or } 1. \tag{3.4}$$

In (3.3), w is the slack variable for the delay constraint.

The main difficulty with the CSP problem lies with the integrality condition that requires that the variables x_{uv} be 0 or 1. Removing or relaxing this requirement from the above integer linear program leads to RELAX-CSP, the relaxed CSP problem.

RELAX-CSP:

Minimize $\sum_{(u,v)\in E} c_{uv} \cdot x_{uv}$ (3.5)

subject to

$$\forall\, u \in V,\ \sum_{\{v|(u,v)\in E\}} x_{uv} - \sum_{\{v|(v,u)\in E\}} x_{vu} = \begin{cases} 1, & if \quad u = s; \\ -1, & if \quad u = t; \\ 0, & otherwise. \end{cases} \tag{3.6}$$

$$\sum\nolimits_{(u,v)\in E} - d_{uv} \cdot x_{uv} - w = -\Delta\,, \tag{3.7}$$

$$\forall\, (u, v) \in E,\ x_{uv} \geq 0. \tag{3.8}$$

We will show later that by using a transformation and applying certain pivot rules we can enforce $x_{uv} \leq 1$ (the discussion after Theorem 3.3, Section 3.5.5).

LARAC algorithm considered in Chapter 2 solves the dual of RELAX-CSP. In the rest of the chapter we study this problem using the primal simplex method.

3.2. A Transformed Problem and Basic Concepts

In order to achieve an efficient implementation of our approach we transform the problem to an equivalent one on a transformed network defined below.

- The graph of the transformed network is the same as that of the original problem, i.e., $G(V, E)$,
- For $(u, v) \in E$, d'_{uv} and c'_{uv} in the transformed problem are given by $d'_{uv} = 2\, d_{uv}$ and $c'_{uv} = c_{uv}$, and
- The new upper bound in the transformed problem is $\Delta' = 2\,\Delta + 1$. (3.9)

The transformed problem will be referred to as the TCSP problem.

Theorem 3.1: An *s-t* path p^* is a feasible solution (*resp.* an optimal solution) to the CSP problem iff it is a feasible solution (*resp.* an optimal solution) to the TCSP problem.

Proof: Please see the proof to Theorem 4.3 which is a general version of this theorem. ■

In view of the above result, we consider in the rest of the chapter only the relaxed form of the TCSP problem, namely, RELAX-TCSP (same as RELAX-CSP except that the network is the transformed one as defined above). Also we use Δ (being odd) and d_{uv} (being even) to denote the delay bound and link delay in the transformed problem, respectively. Notice that the transformation does not change the cost of any path in the network.

In the rest of the section we shall define certain terminology leading to a matrix representation of RELAX-TCSP. Let the links be labeled as $e_1, e_2 \ldots, e_m$ and the nodes

be labeled as 1, 2 …., n. We shall denote the delay of edge e_i as d_i and the cost of e_i as c_i. The incidence matrix of G has m columns, one for each link and n rows, one for each node [55]. The rank of this matrix is $(n - 1)$, and removing any row of this matrix will result in a matrix of rank $(n - 1)$. We denote this resulting matrix as $\boldsymbol{H}$. We also assume that the row removed from the incidence matrix corresponds to node n. Also we assume that the column of $\boldsymbol{H}$ corresponding to link e_k will be denoted by the vector $\boldsymbol{h}_k$. For $e_k = (i, j)$, we have $\boldsymbol{h}_k = (h_{1,k}..., h_{i,k} ..., h_{j,k} ..., h_{n-1,k})^t$ with all its components being 0 except for $h_{i,k} = 1$ and $h_{j,k} = -1$. Let

$$A = \begin{pmatrix} H & 0 \\ D & -1 \end{pmatrix} = (a_1, a_2 ... a_m, a_{m+1}), \tag{3.10}$$

where $\boldsymbol{D} = (-d_1, -d_2 \ldots, -d_m)$, (3.11)

$$a_i = \begin{pmatrix} h_i \\ -d_i \end{pmatrix}, i \le m, \text{ and} \tag{3.12}$$

$$a_{m+1} = \begin{pmatrix} 0 \\ -1 \end{pmatrix}. \tag{3.13}$$

Also, let $\boldsymbol{x}$ be the column vector of the m flow variables x_{uv} and the slack variable w, and $\boldsymbol{c}$ be the row vector of the costs $(c_1 \ldots, c_m, 0)$. Note that the cost of the slack variable is 0. Let $\boldsymbol{b} = (b_1 \ldots, b_{n-1}, -\Delta)^t$ with $b_s = 1$, $b_t = -1$, and $b_i = 0$ for $i \neq s, t$. The LP formulation of the RELAX-TCSP problem can now be written in matrix form as follows.

RELAX-TCSP

Minimize $\boldsymbol{c}\,\boldsymbol{x}$

$$\text{subject to} \quad \boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{b} \qquad (3.14)$$

$$\boldsymbol{x} \geq 0 \text{ for } \forall(u, v) \in E$$

The rest of the chapter deals with the primal simplex based solution of RELAX-TCSP.

3.3. Simplex Method: Basic Solutions of RELAX-TCSP

Simplex method of linear programming starts with a basic solution and proceeds by constructing one basic solution from another. A basic solution consists of two sets of variables, basic and non-basic. For the RELAX-TCSP problem under consideration, all the non-basic variables in a basic solution will have zero values. Given a basic solution, we shall denote by G_b the subgraph of G corresponding to the basic variables (except the slack variable if it is in the basic solution) in this solution. Note that there is no link associated with the slack variable. The subgraph G_b will be called the subgraph of the basic solution or simply the basis graph. The non-singular submatrix of $\boldsymbol{A}$ defined by the basic variables is called a basis matrix or simply, a basis. In this section we present certain important properties of the basic solutions of the RELAX-TCSP problem.

Lemma 3.1 [55]: Let $G(V, E)$ be a directed network with at least one cycle W (not necessarily directed). Assigning an arbitrary orientation to W, let $\boldsymbol{U(W)} = (u_1 \ldots, u_m)^t$, where

$$u_j = \begin{cases} 1, & \text{if } e_j \in W \text{ and the direction of } e_j \text{ agrees with the orientation of } W; \\ -1, & \text{if } e_j \in W \text{ and the direction of } e_j \text{ disagrees with the orientation of } W; \\ 0, & \text{otherwise.} \end{cases}$$

Then, $\boldsymbol{H} \bullet \boldsymbol{U}(W) = 0$.

We shall denote by $d(W)$ the signed algebraic sum of the delays of the links in a cycle W as we traverse around the cycle along the given orientation.

Lemma 3.2: The subgraph G_b of a basic solution contains at most one cycle.

Proof: Assume that there is more than one cycle, say W_1 and W_2, in G_b. Suppose W_1 has k links and W_2 has l links. According to Lemma 3.1, there exist $\lambda_1 \ldots, \lambda_k$ and $u_1 \ldots, u_l$ such that

$$\sum_{e_i \in W_1} \lambda_i \cdot h_i = 0 \text{ and } \sum_{e_i \in W_2} u_i \cdot h_i = 0 .$$

So $$\sum_{e_i \in W_1} \lambda_i \cdot a_i = \begin{pmatrix} 0 \\ -d(W_1) \end{pmatrix} \text{ and } \sum_{e_i \in W_2} u_i \cdot a_i = \begin{pmatrix} 0 \\ -d(W_2) \end{pmatrix}.$$

Without loss of generality, assume that $d(W_1) \neq 0$ and $d(W_2) \neq 0$ (Otherwise, $rank(G_b) < n$). Then

$$d(W_2) \cdot \sum_{e_i \in W_1} \lambda_i \cdot a_i - d(W_1) \cdot \sum_{e_i \in W_2} u_i \cdot a_i = 0 .$$

Since $W_1 \neq W_2$, the above implies rank$(G_b) < n$ which is the desired contradiction. ■

Lemma 3.3: If there is a cycle W in G_b, then $d(W) \neq 0$.

***Proof*:** Let $B = \begin{pmatrix} E_{n-1,n} \\ D_{1,n} \end{pmatrix}$ be a basis matrix (submatrix of $\boldsymbol{A}$), where $\boldsymbol{H}_{n-1,n}$ is a $(n-1) \times n$ submatrix of H and $D_{1,n}$ is the vector of n components (corresponding to the basic variables) of the last row of $\boldsymbol{A}$. Then $E_{n-1,n}\, U(W) = 0$ by Lemma 3.1.

On the other hand, $D_{1,n}\ U(W) = -d(W)$.

Since *rank* (*B*) = *n*, we have

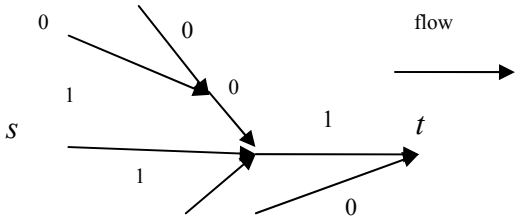
$$B\ U(W) = \begin{pmatrix} E_{n-1,n} \bullet U(W) \\ -d(W) \end{pmatrix} = \begin{pmatrix} 0 \\ -d(W) \end{pmatrix} \neq 0\,.$$

Thus the lemma follows. ■

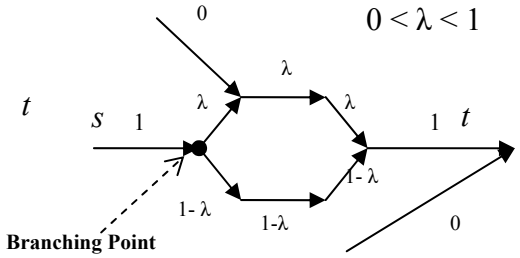
Lemma 3.4: If the basis subgraph G_b contains no cycle that is not a directed cycle, there are exactly two *s*-*t* paths in G_b.

***Proof*:** The proof follows from the flow balance constraints and the transformation.

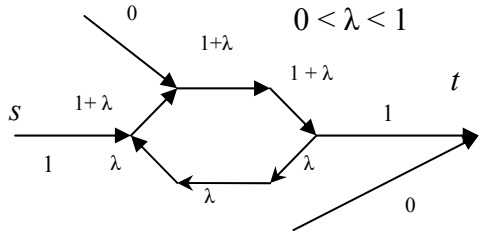
■



a) Tree basic solution



b) Basic solution with a cycle (not directed)



c) Basic solution with a directed cycle

Figure 3.2: Structure of basis graph

Thus it follows from the above lemma that the transformation we introduced guarantees that the structure of the basis subgraph will be one of the three forms shown in Figure 3.2 (a spanning tree or a spanning tree plus an extra link). In a later section we shall introduce a pivot rule which will ensure that the basis subgraph will not contain any directed cycle, thereby eliminating the structure in Figure 3.2(*c*).

3.4. Revised Simplex Method on the RELAX-TCSP Problem

In this section, we first briefly present the different steps in the revised simplex method of linear programming that is described in detail in [11]. We then derive formulas required to identify the entering and the leaving variables.

3.4.1. Revised Simplex Method

Consider an arbitrary linear programming (LP) problem in the standard form.

Minimize $\boldsymbol{c}\,\boldsymbol{x}$

subject to $\boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{b}$, $\boldsymbol{x} \geq 0$.

Here $\boldsymbol{A}$ is an $n \times (m + 1)$ matrix with $rank\ (\boldsymbol{A}) = n$, $\boldsymbol{x} = (x_1 \ldots, x_{m+1})^t$, $\boldsymbol{c} = (c_1 \ldots, c_{m+1})$, and $\boldsymbol{b} = (b_1 \ldots, b_n)^t$. Each feasible basic solution $\boldsymbol{x}^*$ is partitioned into two sets, one set consisting of the n basic variables and the other set consisting of the remaining $m + 1 - n$ non-basic variables. This partition induces a partition of $\boldsymbol{A}$ into $\boldsymbol{B}$ and $\boldsymbol{A_N}$, a partition of $\boldsymbol{x}$ into $\boldsymbol{x_B}$ and $\boldsymbol{x_N}$, and a partition of $\boldsymbol{c}$ into $\boldsymbol{c_B}$ and $\boldsymbol{c_N}$, corresponding to the set of basic variables and the set of non-basic variables, respectively. The basis matrix $\boldsymbol{B}$ is nonsingular.

Revised Simplex Method [11]:

Step 1: Solve the system $\boldsymbol{Y B} = \boldsymbol{c_B}$, where $\boldsymbol{Y} = (y_1, y_2 \ldots, y_n)$.

Step 2: Choose an entering column. It may be any column $\boldsymbol{a_i}$ of $\boldsymbol{A_N}$ such that $\boldsymbol{Y a_i}$ is greater than the corresponding component of $\boldsymbol{c_N}$. The current solution is optimal if there is no such column.

Step 3: Solve the system $\boldsymbol{B V} = \boldsymbol{a_i}$, where $\boldsymbol{V} = (v_1, v_2 \ldots, v_n)^t$.

Step 4: Find the largest t such that $\boldsymbol{x^*_B} - t\ \boldsymbol{V} \geq 0$. If there is no such t, then the problem is unbounded; otherwise, at least one component of $\boldsymbol{x^*_B} - t\ \boldsymbol{V}$ is equal to 0 and the corresponding variable leaves the basis.

Step 5: Set the value of the entering variable as t and replace the values $\boldsymbol{x^*_B}$ of the basic variables by $\boldsymbol{x^*_B} - t \bullet \boldsymbol{V}$. Replace the leaving column of $\boldsymbol{B}$ by the entering column and in the basis heading, replace the leaving variable by the entering variable. Then go to Step 1.

In the following we solve the systems of equations in Steps 1 and 3 and derive explicit formulas for $\boldsymbol{Y}$ and $\boldsymbol{V}$. If a link flow variable is chosen as the entering variable then the corresponding link is called the in-arc. Out-arcs are similarly defined.

3.4.2. Initialization

To construct an initial basic feasible solution we first determine a spanning tree containing a feasible *s*-*t* path. This can be done by applying Dijkstra's algorithm to compute the shortest path tree with respect to the delay from all nodes to the destination node *t*. If the resulting *s*-*t* path in the tree is infeasible, then no feasible path exists and

the algorithm terminates. Without loss of generality we assume that the *s*-*t* path is feasible.

Clearly in the basic solution corresponding to the spanning tree selected as above, the flows in all the links in the *s*-*t* path in the spanning tree will be equal to one, and flows in all other links will be zero. Since the delay of every link in the TCSP problem is even and the upper bound Δ on path delay is odd, the slack variable $w > 0$ and so it is in the initial basic feasible solution.

3.4.3. Solving the System $\boldsymbol{Y B = c_B}$

Let $\boldsymbol{Y} = (y_1 \ldots, y_{n-1}, \gamma)$. Here $y_1 \ldots, y_{n-1}, \gamma$ are called potentials (or dual variables) and $\boldsymbol{Y}$ is called the potential vector. Each y_i, $i = 1, 2 \ldots, n-1$ is the potential associated with node i (or the row i) and γ is the potential associated with the last row (delay constraint row) of $\boldsymbol{A}$.

Now consider

$$\boldsymbol{Y B = c_B} \tag{3.15}$$

This system of equations has n equations in n variables. We get the following from (3.15).

For each link $e_k = (i, j)$ in G_b, $(y_1 \ldots, y_{n-1}, \gamma)\ h_k = c_{ij}$. That is,

$$\begin{aligned} y_i - y_j - \gamma\, d_{ij} &= c_{ij}, \text{ if } i \neq n \text{ and } j \neq n, \\ y_i - \gamma\, d_{in} &= c_{in}, \text{ if } j = n, \text{ and} \\ -y_j - \gamma\, d_{nj} &= c_{nj}, \text{ if } i = n. \end{aligned} \tag{3.16}$$

From the above, we can see that we can set the potential of node n at any constant. In all computations that follow, we shall set the potential of node n equal to zero.

Definition 3.2:

1) For link $e_k = (i, j)$, $c(e_k, \gamma) = \gamma d_{ij} + c_{ij}$ is called the active cost of link (i, j),

2) $r(i, j) = y_j - y_i + \gamma d_{ij} + c_{ij}$ is called the reduced cost of link (i, j),

3) The reduced cost of w is given by $r(w) = \gamma$, and

4) The reduced cost of a path p is defined as

$$r(p) = \sum_{(i,j) \in p^+} r(i, j) - \sum_{(i,j) \in p^-} r(i, j) .$$

It can be seen from (3.16) that for any link (i, j) in G_b

$$r(i, j) = y_j - y_i + \gamma d_{ij} + c_{ij} = 0. \tag{3.17}$$

From (3.17) we also have that for any path p from i to j and any cycle W in G_b

$$r(p) = y_j - y_i + \gamma d(p) + c(p) = 0 \text{ and } r(W) = \gamma d(W) + c(W) = 0. \tag{3.18}$$

Lemma 3.5: If G_b contains a cycle W, then $\gamma = -c(W) / d(W)$; Otherwise, $\gamma = 0$.

***Proof*:** If there is no cycle in G_b then the slack variable w is a basic variable and the corresponding column $[0, 0\ldots, 0, -1]^t$ will be a column of B. Since the cost of the slack variable is zero, we get from (3.15) that $\gamma = 0$. Suppose that G_b contains a cycle W. By (3.18), we get

$\gamma d(W) + c(W) = 0$. By Lemma 3.3, $d(W) \neq 0$. So $\gamma = -c(W) / d(W)$. ■

Lemma 3.6: A link is eligible to enter the basis if its reduced cost is negative and the slack variable is eligible to enter the basis if $\gamma < 0$.

Proof: The proof follows from Step 2 of the revised simplex method. ■

Once we have computed the value of γ as in Lemma 3.5, the other potentials y_i's can be calculated using equation (3.18) and selecting the path in G_b from node n to node i.

Summarizing the above, we have the following procedure for solving $\boldsymbol{Y\ B} = \boldsymbol{c_B}$ and calculating the potentials.

(1) Set the potential of node n to zero.

(2) Compute γ as in Lemma 3.5.

(3) For each node i, let p_i be a simple path in G_b from node n to node i. If there are two paths in G_b due to the cycle, we will get the same results no matter which path is selected.

(4) Set $c(p) = \sum_{(u,v)\in p^+} c_{uv} - \sum_{(u,v)\in p^-} c_{uv}$ and $d(p) = \sum_{(u,v)\in P^+} d_{uv} - \sum_{(u,v)\in P^-} d_{uv}$,

where p_i^+ and p_i^- are the sets of forward and backward links on p_i, respectively, as we traverse the path from node n to node i.

Once the potentials are determined, an entering variable, if it exists, can be selected as in Step 2 of the revised simplex method.

3.4.4. Solving the System $\boldsymbol{B\ V = a_k}$

We next show how to solve the system of equations $\boldsymbol{B\ V} = \boldsymbol{a}_k$. We consider three cases:

Case *a*): G_b contains no cycle, that is, G contains only $n - 1$ links and the slack variable w is a basic variable. The link $e_k = (i, j)$ is the entering variable.

Case *b*): G_b contains a cycle (that is, G_b has n links) and the entering variable is $e_k = (i, j)$.

Case *c*): G_b contains a cycle (G_b has n links) and the entering variable is the slack variable.

Solutions in all the three cases are summarized in the following theorem.

Theorem 3.2: *a*) If G_b contains no cycle and the entering variable is an in-arc $e_k = (i,$

j), then the vector $\boldsymbol{V}$ defined below is the desired solution to $\boldsymbol{B} \bullet \boldsymbol{V} = \boldsymbol{a}_k$, where W' is the new cycle formed by adding the in-arc e_k and the orientation of W' is chosen to agree with the direction of e_k. The vector $\boldsymbol{V} = (v_1 \ldots, v_n)^t$ is defined as:

$$v_i = \begin{cases} -1, \textit{ if } i < n \textit{ and the link corresponding to the i th column of B is in } W' \\ \qquad \textit{and its orientation agrees with the cycle orientation;} \\ 1, \textit{ if } i < n \textit{ and the link corresponding to the i th column of B is in } W' \\ \qquad \textit{and its orientation disagrees with the cycle orientation;} \\ d(W'), \textit{ if } i = n \\ 0, \textit{ otherwise} \end{cases} \tag{3.19}$$

b) If G_b contains a cycle W and link $e_k = (i, j)$ enters the basis, then $\boldsymbol{V} = -\boldsymbol{V}'_p + (d(W')/d(W)) \bullet \boldsymbol{V}_0$ is the solution of $\boldsymbol{B} \bullet \boldsymbol{V} = \boldsymbol{a}_k$, where $d(W')$ and $d(W)$ are the delays of cycle W' and W, respectively and $\boldsymbol{V}'_p$ and $\boldsymbol{V}_0$ are vectors defined by the cycles W' and W as in Lemma 3.1.

c) If G_b contains a cycle W and the entering variable is the slack variable w, then $\boldsymbol{V} = (1/d(W))\ \boldsymbol{V}_0$ is the solution to $\boldsymbol{B} \bullet \boldsymbol{V} = \boldsymbol{a_k}$, where $\boldsymbol{V}_0$ is defined by cycle W.

***Proof*:** Case *a*): G_b contains no cycle, that is, G contains only $n - 1$ links and the slack variable w is a basic variable. The link $e_k = (i, j)$ is the entering variable. In this case,

$$\boldsymbol{B} = \begin{pmatrix} H_{n-1,n-1}, & 0 \\ D_{1,n-1}, & -1 \end{pmatrix},$$

where $\boldsymbol{H}_{n-1,\,n-1}$ is associated with the $n - 1$ links and the $n - 1$ nodes in G_b, and $D_{1,n-1}$ is the vector of $n - 1$ components (corresponding to the delays of the $n - 1$ links in the basis subgraph) of the last row of $\boldsymbol{A}$.

Let W' denote the new cycle formed by adding the in-arc $e_k = (i, j)$ and let the orientation of W' be chosen to be the same as the direction of the in-arc. Using Lemma

3.2 and the cycle W', it is easy to verify that the vector $\boldsymbol{V} = (v_1 \ldots, v_n)^t$ defined in (3.19) solves the system $\boldsymbol{B} \bullet \boldsymbol{V} = \boldsymbol{a_k}$.

Case *b*): G_b contains a cycle (that is, G_b has n links) and the entering variable is $e_k = (i, j)$. Note that the slack variable w is not a basic variable. Hence,

$$B = \begin{pmatrix} H_{n-1,n} \\ D_{1,n} \end{pmatrix},$$

where $\boldsymbol{H}_{n-1,\,n}$ corresponds to n links and $n - 1$ nodes, $\boldsymbol{D}_{1,\,n}$ is the vector of n components (corresponding to the basic variables) of the last row of $\boldsymbol{A}$ and

$$\boldsymbol{a_k} = \begin{pmatrix} h_k \\ -d_{ij} \end{pmatrix}.$$

We need to solve the system of equations

$$\begin{pmatrix} H_{n-1,n} \\ D_{1,n} \end{pmatrix} V = \begin{pmatrix} h_k \\ -d_{ij} \end{pmatrix}. \tag{3.20}$$

First, let us consider

$$\boldsymbol{H}_{n-1,\,n}\ \boldsymbol{V} = \boldsymbol{h_k}. \tag{3.21}$$

Because there are n links in G_b, there is exactly one cycle, denoted by W.

Therefore according to Lemma 3.1,

$$\exists \boldsymbol{V}_0, \boldsymbol{H}_{n-1,\,n}\ \boldsymbol{V}_0 = 0. \tag{3.22}$$

After adding link $e_k = (i, j)$, we get a new cycle W' and let us choose the orientation of this cycle to be the same as that of e_k. Then by Lemma 3.1,

$$\exists\ V' = \begin{pmatrix} V'_p \\ 1 \end{pmatrix},\ (H_{n-1,n}, h_k) \begin{pmatrix} V'_p \\ 1 \end{pmatrix} = 0. \tag{3.23}$$

(Note: V'_p can be derived using Lemma 3.1 and the cycle W'). So

$$\boldsymbol{H}_{n-1,n}(-\boldsymbol{V}'_p) = \boldsymbol{h_k}. \tag{3.24}$$

Because $rank(\boldsymbol{H}_{n-1,n}) = n-1$, $-\boldsymbol{V}'_p + u \bullet \boldsymbol{V}_0$, $u \in R$ is the solution space of (3.21).

Using the equation $\boldsymbol{D}_{1,n} \bullet V = -d_{ij}$, we can compute u as follows.

$$\boldsymbol{D}_{1,n} \bullet (-\boldsymbol{V}'_p + u\ \boldsymbol{V}_0) = -d_{ij}. \tag{3.25}$$

Since $\boldsymbol{D}_{1,n}\boldsymbol{V}_0 = -d(W)$ and $\boldsymbol{D}_{1,n}(-\boldsymbol{V}'_p) + d_{ij} = d(W')$, we get from (3.25)

$d(W') - u\,d(W) = 0$ and thus $u = d(W') / d(W)$. Note: $d(W) \neq 0$, by Lemma 3.3.

Hence $\boldsymbol{V} = -\boldsymbol{V}'_p + (d(W') / d(W))\boldsymbol{V}_0$ is the desired solution to $\boldsymbol{B}\,\boldsymbol{V} = \boldsymbol{a}_k$.

Case *c*): G_b contains a cycle W and the entering variable is the slack variable w.

Following the arguments in Case 2, we can show that

$$V = (1 / d(W))\,V_0. \tag{3.26}$$

is the desired solution of the equation system $\boldsymbol{B}\,\boldsymbol{V} = \boldsymbol{a}_k$. Here V_0 is defined by W. ■

3.4.5. A Pivot Rule and Structure of Basic Feasible Solutions

In this subsection we present a pivot rule and study the structure of subgraphs of basic solutions generated by the simplex method. The subgraph G_b of the initial basic feasible solution has $n-1$ links and the nth variable in this basic solution is the slack variable $w > 0$. At this initial step, $\gamma = 0$ (Lemma 3.5). Define $d(G_b) = \sum_{(u,v) \in G_b} x_{uv} d_{uv}$. By (3.7), $d(G_b) = \Delta - w$. Now one of the following two possibilities occurs in the next pivot.

1. The simplex method constructs a new spanning tree solution with the slack variable w remaining nonzero in the new solution.

2. The simplex method constructs a G_b that contains one cycle W (formed by adding the in-arc) and w becomes non-basic with respect to this solution. The cycle W cannot be a directed cycle. If it were a directed cycle, then the reduced cost of the entering link

will be equal to the sum of the costs of the links in W. This sum is a positive number contradicting the requirement that the reduced cost of the entering link must be negative (Step 2 of the revised simplex method). By Lemma 3.4, there will be exactly two *s*-*t* paths in G_b. Also, the flow values on all the links in W must be nonzero, for otherwise all the link flows will be either 0 or 1 making w nonzero and hence basic.

Summarizing, when the first time a G_b with a cycle is encountered, it will be necessarily of the form shown in Figure 3.2(*b*). Flows on the links in the cycle will be λ or $1 - \lambda$. The simplex method will select the value of $\lambda > 0$ in such a way that $d(G_b) = \Delta$.

Though the cycle in the G_b encountered the first time after initialization will not be a directed cycle, in a subsequent step, a G_b with a directed cycle may be created. To achieve an efficient implementation of the simplex method, we would like to avoid generating any G_b containing a directed cycle. This can be achieved by the pivot rule P1 given next.

Pivot Rule P1**:** Select the slack variable w as the entering variable if it is eligible to enter.

Theorem 3.3: If the pivot rule P1 is followed and the simplex method on the RELAX-TCSP problem is initialized as in Section 3.4.2, then no basic solution subgraph G_b will contain a directed cycle.

Proof: Assume that a G_b with a directed cycle W' is created and let $e_{ij} = (i, j)$ be the in-arc with which this cycle is created.

Suppose $W' = e_{ij} e_{jj_1} e_{j_1 j_2} ..., e_{j_k i}$ and p_{ji} is the directed path from j to i in W'.

Since e_{ij} is an in-arc and $\boldsymbol{Y} = (y_1, y_2 \ldots, y_{n-1}, \gamma)$ is the potential vector, we have

$$r(i, j) = y_j - y_i + \gamma d_{ij} + c_{ij} < 0 \text{ and } r(p_{ji}) = y_i - y_j + \gamma d(p_{ji}) + c(p_{ji}) = 0.$$

Summing the above, we obtain $\gamma d(W') + c(W') < 0$.

Since $d(W') > 0$ and $c(W') > 0$, $\gamma < 0$. This implies that the slack variable is eligible to enter the basis but was not selected. This is a contradiction. ■

Theorem 3.3 implies that pivot rule P1 along with the transformation introduced in Section 3.2 guarantees that G_b will take only the structures shown in Figure 3.2(*a*) and Figure 3.2(*b*). Under these conditions we are also guaranteed that the values of the variables x_{uv} will be restricted to the range $0 \leq x_{uv} \leq 1$.

3.4.6. An Anti-Cycling Strategy

A basic solution in which one or more basic variables assume zero values is called degenerate. A degenerate basic solution may result in a pivot that does not alter the basic solution. Such pivots are called degenerate. Furthermore, a basic solution generated at one pivot and reappearing at another will lead to cycling. Since degenerate pivots do not result in any improvement of the solutions, they are also a cause of inefficiency. We present two strategies to handle degeneracy. The first one to be presented in this subsection is the anti-cycling strategy which is a variation and extension of Cunningham's anti-cycling strategy in [1], [4], and [11]. The second strategy to be presented in Section 3.5 is designed to avoid degenerate pivots completely.

Definition 3.3: Given a feasible basic solution subgraph G_b and a node called the root, we say that the link $(u, v) \in G_b$ is oriented toward (*resp*. away from) the root if any path in G_b from the root to u (*resp. v*) passes through v (*resp*. u). A feasible basic

solution G_b with corresponding flow vector $\boldsymbol{x}$ is strongly feasible if every link (u, v) of G_b with $x_{uv} = 0$ is oriented toward the root**.**

If the out-arc (u, v) is not a link of the cycle in the basic solution, then $G_b - (u, v)$ contains exactly two components $G_b(u)$ and $G_b(v)$ such that $u \in G_b(u)$ and $v \in G_b(v)$. If the root is in $G_b(v)$, link (u, v) is oriented toward the root; otherwise it is oriented away from the root. See Figure 3.3(*a*), (*b*) for examples of a strongly feasible G_b. We shall select node *t* as the root node.

Lemma 3.7: For any degenerate pivot, the out-arc is not on the cycle of the current G_b.

Proof: A degenerate pivot does not alter the basic solution. This means that each variable has the same value in the current basic solution as well as in the basic solution resulting from the degenerate pivot. The flow on each link in a cycle is non-zero. If a link on a cycle were to leave the basis, then after the degenerate pivot it would become non-basic with flow 0. But that would contradict that the current pivot is degenerate. ■

If the out-arc is not on the cycle in the current G_b, then the potentials can be updated easily as described next (See Chapter 5.1.2 of [4]). Let $\boldsymbol{T}$ be the current G_b and $\boldsymbol{e} = (u, v)$ and $\boldsymbol{e'} = (u', v')$ be the out-arc and the in-arc, respectively. Let $\boldsymbol{T'} = \boldsymbol{T} - \boldsymbol{e} + \boldsymbol{e'}$ be the subgraph of the new basic variables. If $\boldsymbol{e}$ is not on the cycle in the current G_b, the new potential vector $\boldsymbol{Y'}$ associated with $\boldsymbol{T'}$ can be obtained as follows (notice that γ does not change in this case):

$$y'_u = \begin{cases} y_u + r_{u'v'} & \text{if } u \in T_{u'} \\ y_u & \text{if } u \in T_{v'} \end{cases}, \tag{3.27}$$

where $r_{u'v'} = c(e_{u'v'}, \gamma) + y_{v'} - y_{u'}$ and $\boldsymbol{T_{u'}}(\boldsymbol{T_{v'}})$ is the component of $\boldsymbol{T} - \boldsymbol{e}$ containing u' (v').

Theorem 3.4: If the subgraphs G_b's of feasible basic solutions generated by the simplex method are strongly feasible, then the simplex method does not cycle.

Proof: First observe that in any sequence of degenerate pivots, the value of the slack variable will remain the same. So the leaving and entering variables can only be the links in the network. Let G_b be a feasible basic solution subgraph and t be the root. We define two unique values for G_b: $C(G_b) = \sum_{(u,v)\in E} c_{uv} x_{uv}$ and $W(G_b) = \sum_{u\in V} (y_t - y_u)$.

Notice that for a given G_b, the value of $W(G_b)$ is unique even though the values of the potentials $\boldsymbol{Y}$ may not be unique.

Consider two consecutive basic solutions $G_b^i = G_b$ and $G_b^{i+1} = G_b^i + e - f$, where e and f are the in-arc and out-arc, respectively.

We first show that either $C(G_b^{i+1}) < C(G_b^i)$ or $W(G_b^{i+1}) > W(G_b^i)$.

Indeed if the pivot that generates G_b^{i+1} from G_b^i is nondegenerate, then $C(G_b^{i+1}) < C(G_b^i)$. If it is degenerate, we have $C(G_b^{i+1}) = C(G_b^i)$. In this case we shall prove $W(G_b^{i+1}) > W(G_b^i)$.

Here the in-arc $e = (u, v)$ still has zero flow in G_b^{i+1}. By Lemma 3.7, f is not a link on the cycle in G_b^i, so the value of γ does not change. Because G_b^{i+1} is strongly feasible, in G_b^{i+1}, link e must be oriented toward the root node t, which implies that node t belongs to $G_b(v)$ (the component of $G_b^i - f$ containing v). Now we can obtain the potentials using equation (3.27).

Since $r_{uv} = c(e_{uv}, \gamma) + y_v - y_u < 0$, $W(G_b^{i+1}) = W(G_b^i) - |G_b(u)|\, r_{uv} > W(G_b^i)$.

If the simplex method cycles, then for some $i < j$, $G_b^i = G_b^j$. This means $G_b^i = G_b^{i+1}$... $= G_b^j$. But then $W(G_b^i) > W(G_b^{i+1}) > \ldots > W(G_b^j) = W(G_b^i)$ contradicting that $W(G_b^i) = W(G_b^j)$. ■

3.5. A Strategy for Avoiding Degenerate Pivots and the Network Simplex Based (NBS) Algorithm

In this section we first present in Section 3.5.1 a strategy for avoiding degenerate pivots. We then show in Section 3.5.2 how to select a leaving variable. In Section 3.5.3 we present a complete description of the new Network Based Simplex (NBS) algorithm and its complexity analysis. We also show how to extract an approximate solution to the TCSP (hence the original CSP) problem and performance bounds on the approximate solution.

3.5.1. Avoiding Degenerate Pivots

In this section we shall develop a strategy which avoids performing degenerate pivots which is based on the following pivot rule.

Enhanced Pivot Rule P2: If there is a choice for selecting the entering variables, then select an entering variable in the following order of preference:

a) The slack variable if it is eligible to enter.

b) Eligible links whose tail nodes are on the directed *s*-*t* path(s) in the current G_b.

As we discussed in Section 3.4, rule *a*) above guarantees that every G_b is of one of the two forms in Figure 3.2 (*a*), (*b*). Both these subgraphs of basic solutions are strongly

feasible. Consider next rule *b*). Suppose we can find an in-arc $e = (u, v)$ according to rule *b*). Let W' denote the new cycle in $G_b + e$ with its orientation defined as the direction of e. It can be seen that the flows on all links in W' whose directions disagree with that of W' are nonzero and thus we can push positive amount of flow along the cycle until the flows on some links of the *s*-*t* path (whose directions disagree with the orientation of W') reach zero. By removing one such link with zero flow, we obtain a new G_b. In fact, we can select the out-arc in such a way that the resulting G_b is also strongly feasible (see next subsection). This pivot will not lead to degeneracy. On the other hand, if no such link is eligible to enter the basis (note: in this case γ is nonnegative), then we have no option but to perform a degenerate pivot. To avoid performing degenerate pivots we proceed as follows.

Let P be the set of nodes on the *s*-*t* path(s) in the current basis subgraph G_b. Assign costs to links in the network as follows: Link cost c_{uv} with $u \notin P$ and $v \in P$ is set as $c(e_{uv}, \gamma) + y_v > 0$; Otherwise c_{uv} is set as $c(e_{uv}, \gamma)$.

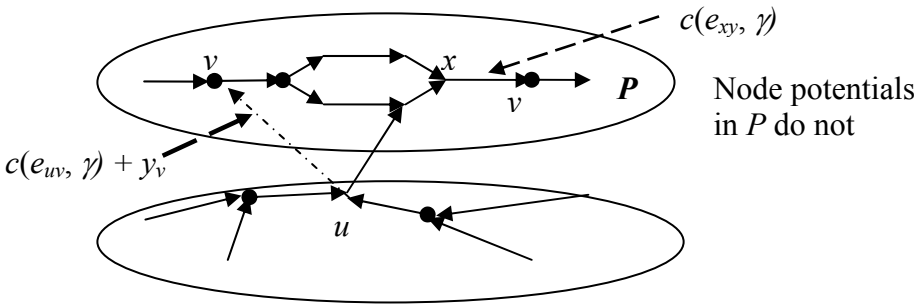


Figure 3.3: Link costs for the transformed graph

Now condense all the nodes in P into a single node, say, R, and reverse the directions of all the links. Let the resulting network be called N'. Note that none of the links with both its ends in P will be in N'. Now use Dijkstra's algorithm on N' and obtain the

shortest path tree with node R as the start node. The links of G corresponding to the links of the shortest path tree of N' and the links with their both end nodes in P will be a new basis subgraph G'_b (Notice that this operation preserves the strongly feasibility of G_b and will not change the value of γ). Let the shortest distance value of the node u computed by the algorithm be $d(u)$. Then we set the potentials of the nodes with respect to G'_b: For each node $u \notin P$, $y_u = d(u)$, and for all other nodes (all the nodes in P) the potentials are the same as in the previous G_b.

Now, $\forall$ (u, v), $u \notin P$, $y_u = d(u) \leq d(v) + c(e_{uv}, \gamma) = y_v + c(e_{uv}, \gamma)$, which implies that for all such links, $r(u, v) = y_v - y_u + \gamma d_{uv} + c_{uv} \geq 0$ and those links whose tails are not in P are not eligible for choice as in-arc. Since the above operation does not affect the value of γ, w is not eligible either. Thus we can only consider arcs whose tails are in P (part (b) of enhanced rule P2). If we still cannot find an in-arc according to enhanced rule P2 after the above operation, it implies that we have got the optimal basic solution since no entering variable is available.

We will show in the following section how to choose a leaving variable using Theorem 3.2.

3.5.2. Finding a Leaving Arc (Out-Arc)

Suppose the current feasible basic solution G_b is strongly feasible and link $e = (u, v)$ is the in-arc. If G_b contains a cycle W, then the flow can be decomposed into exactly two s-t paths. We define the branching point as the first node on W as we traverse the paths from node s to t (see Figure 3.2(b)). In this subsection, e and e' always denote the in-arc and out-arc, respectively.

Claim 3.1: If the current basic solution G_b is strongly feasible and is not optimal, then one of the arcs e' incident to the branching node or the tail node of the in-arc e is eligible for choice as out-arc and $G_b + e - e'$ is still strongly feasible**.**

We prove the claim by enumerating all possible cases and determining the leaving variable in each case using Theorem 3.2 and Step 4 of the revised simplex method. Let the cycle created by adding the in-arc be denoted by W' with its orientation defined as that of the in-arc.

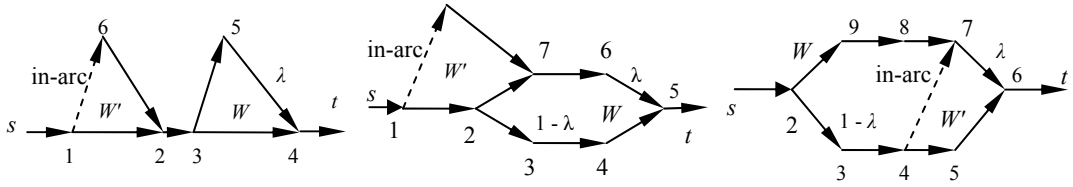
Case 1: Slack variable w is in the basic solution (the current G_b is a tree, $\gamma = 0$ and $w > 0$). This corresponds to Theorem 3.2 (*a*). According to Step 4 of the revised simplex method, we need to consider only the entries of $\boldsymbol{V}$ that are 1 or $d(W')$ if $d(W') > 0$. Without loss of generality, assume $d(W') > 0$. These entries correspond to the links of W' that lie on the *s*-*t* path of the current G_b or the slack variable w. The corresponding entries in the current basic solution $\boldsymbol{x^*_B}$ are 1 for the links and its current value for w. The minimum value of t satisfying the constraint $\boldsymbol{x^*_B} - t \bullet \boldsymbol{V} \geq 0$ will be determined by the inequalities $1 - t \geq 0$ and $w - t\, d(W') \geq 0$. Thus the maximum value of t will be $\min\{1, w / d(W')\}$. Since $w = \Delta - d(G_b)$ is odd and $d(W')$ is even, $w / d(W') \neq 1$. So, if $w < d(W')$, w will leave the basis. Otherwise, the links in W' that lie on some *s*-*t* path in the current G_b are eligible to leave the basis. We shall select the unique link e' on the *s*-*t* path in G_b that is incident to the tail node of the in-arc. This guarantees that the new G_b, denoted as G'_b, is strongly feasible.

Notice that if w leaves the basis, $w = 0$ in G'_b. This means that $d(G'_b) = \Delta$. In this case, G'_b contains two *s*-*t* paths p_1 and p_2 with flow λ and $1 - \lambda$, respectively (see Figure 3.2).

The value of λ can be calculated from the equation: $\lambda\, d(p_1) + (1 - \lambda)\, d(p_2) = \Delta$.

Case 2: The basic solution consists of n links, i.e., there is a cycle W with branching point a in the basic solution. The slack variable w is eligible to enter the basis if $\gamma < 0$. Then according to part a) of pivot rule P2, we let w enter the basis and shall select one of the two links in the current G_b that are incident on the branching point a to leave the basis. The choice can be made using Theorem 3.2 (c) of Section 3.4.4.

Suppose $\gamma > 0$. An in-arc will create a new cycle W'. This corresponds to Theorem 3.2(b) in Section 3.4.4. We need to consider three sub-cases that capture all possibilities. Without loss of generality, we assume that the orientation of W is clockwise and the orientation of W' agrees with the direction of the in-arc.



a)Sub-case 2.1 *b*) Sub-case 2.2 *c*) Sub-case 2.3

Figure 3.4: Find leaving variable: sub-cases of Case 2

Case 2.1 (Figure 3.4(a)): Possible out-arcs: (1, 2), (3, 5) and (3, 4). Here, (x_{12}, x_{35}, x_{34}) = (1, λ, 1 – λ) and thus the out-arc corresponds to the first zero component in the following formula as t increases from 0.

$$(1, \lambda, 1-\lambda) - t\,(1, d(W')/d(W), -d(W')/d(W))$$

$$= (1-t, \lambda - t\,d(W')/d(W), 1-\lambda + t\,d(W')/d(W)).$$

Case 2.2 (Figure 3.4(b)): Possible out-arcs: (1, 2), (2, 7) and (2, 3). Link (7, 6) is not eligible for out-arc because otherwise $w \neq 0$ in the next basic solution due to the property of the transformed network. The out-arc is decided by the following formula as in Case 2.1.

$$(x_{12}, x_{27}, x_{23}) - t\,(1,\ 1 + d(W')/d(W),\ -d(W')/d(W))$$

$$= (1, \lambda, 1 - \lambda) - t\,(1,\ 1 + d(W')/d(W),\ -d(W')/d(W)).$$

Case 2.3 (Figure 3.4(*c*)): Possible out-arcs: (2, 3), (2, 9) and (4, 5). The out-arc corresponds to the first zero component in the following formula when t increases.

$$(x_{23}, x_{29}, x_{45}) - t\,(-d(W')/d(W),\ d(W')/d(W),\ 1 - d(W')/d(W)).$$

3.5.3. NBS Algorithm, Complexity Analysis, and an Approximate Solution

We now present a complete description of the Network Based Simplex (NBS) algorithm that uses the strategies developed in Section 3.5.1 and 3.5.2 for the RELAX-TCSP problem. We show in Section 3.5.3.1 that the algorithm is of pseudo-polynomial time complexity. In Section 3.5.3.2 we show how to extract from an optimum solution to the RELAX-TCSP problem a feasible solution to the TCSP problem and hence to the original CSP problem and derive bounds on the deviation of this solution from the cost of the optimum solution.

Procedure NBS

```
Transform the original network as in Section 3.2
Find an initial feasible basic solution as in Section 3.4.2
loop {
 if (γ < 0) then
    Let slack variable w be the entering variable (rule (a) of Pivot rule P2)
 else if an in-arc satisfying rule (b) of Pivot Rule P2 is available then
          Choose one of them as the entering variable
       else {
            Invoke Dijkstra's algorithm on the active costs to update the potentials.
            if an in-arc satisfying rule (b) of Pivot Rule P2 is available then
                 Choose one of them as the entering variable
            else stop /*has reached the optimal condition*/
     }
 }
 Determine a leaving variable as in Section 3.5.2
 Update the flows and the potentials as in steps of Section 3.4.3
```

Figure 3.5: Network based simplex algorithm

3.5.3.1. *Complexity Analysis*

Fact 1: If there is no cycle in the basic solution subgraph, then for each link e_{uv}, the link flow x_{uv} is either 1 or 0. If there is a cycle W in G_b, x_{ij} is 0 or at least $1 / |d(W)|$.

***Proof*:** If there is no cycle, the proof is trivial. Assume there is a cycle W. It can be seen that the flow on links not on the two paths are 0 and the flows on the paths but not on the cycle is 1. Since there is a cycle, the flow can be decomposed into two paths p_1 and p_2. Consider flows on the cycle W. Suppose the flow on p_1 and p_2 are λ and $1 - \lambda$ with $0 < \lambda < 1$.

Assume $d(p_1) \geq d(p_2)$. Since $d(p_1)$ and $d(p_2)$ are both even and Δ is odd, $d(p_1) \neq \Delta$ and $d(p_2) \neq \Delta$. Also by Lemma 3.3, $d(W) \neq 0$.

So $d(p_1) \neq d(p_2)$ because $d(W) = d(p_1) - d(p_2)$.

We also have $\lambda\, d(p_1) + (1 - \lambda)\, d(p_2) = \lambda(d(p_1) - d(p_2)) + d(p_2) = \Delta$. So,

$$\min\{d(p_1), d(p_2)\} \leq \Delta \leq \max\{d(p_1), d(p_2)\} \text{ and } \lambda = (\Delta - d(p_2)) / (d(p_1) - d(p_2)).$$

Hence $\lambda \geq 1 / d(W)$ because $\Delta - d(p_2) \geq 1$ and $d(W) = d(p_1) - d(p_2) > 0$.

Similarly, we can prove that $1 - \lambda \geq 1 / d(W)$. ■

Fact 2: If e_{uv} is the in-arc and W' and W are the newly created cycle and the old cycle (if it exists), respectively, we have

$$0 < |y_u - y_v - \gamma\, d_{uv} - c_{uv}| = |\gamma\, d(W') + c(W')| =$$

$$\begin{cases} |c(W')|, & \gamma = 0; \\ |c(W')d(W) - d(W')c(W)| / |d(W)|, & \gamma \neq 0. \end{cases}$$

***Proof*:** Suppose the cycle W' is $e_1 e_2 \ldots e_k$ where $e_1 = e_{uv}$. Since all the links but e_{uv} on W' are in the basic solution, the reduced costs on all these links but e_{uv} are 0. So $|y_u -$

$y_v - \gamma\, d_{uv} - c_{uv}| = |\gamma\, d(W') + c(W')|$. Recalling that $\gamma = -\, c(W) / d(W)$, if there exists a cycle W in the basic solution or $\gamma = 0$ if no such cycle exists, we get the rightmost equality.

Since e_{uv} is an in-arc, $|y_u - y_v - \gamma\ d_{uv} - c_{uv}| > 0$. ■

Fact 3: Let t be the maximal flow that can be pushed on the new cycle W'. Suppose that e_{uv} and x_{uv} are a link and its flow in the basic solution, respectively. Then the strictest constraint on t is given by $x_{uv} - t\,(1 + |d(W') / d(W)|) \geq 0$, $t \geq 0$ and $t \leq 1$. Hence $\max t \geq \min\{1,\ 1 / (|d(W)| + |d(W')|)\} = 1 / (|d(W)| + |d(W')|)$.

***Proof*:** First assume there is a cycle W in the current basic solution. If we push flow t on the new cycle W', according to Theorem 3.2 and Step 4 of the revised simplex method, in the worse case, the flow on all links will be decreased by at most $t(1+|d(W')/d(W)|)$. Proof follows if we recall that $x_{uv} \geq 1 / d(W)$. The proof is similar if there is no cycle in the basic solution. ■

Fact 4: Let T and T' be two consecutive feasible basic solutions in the simplex method and $c(T)$ denote the cost of the flow associated with the basic solution T. If $c(T') < c(T)$ and D is the maximal link delay, then $|c(T') - c(T)| = t\,|y_u - y_v - \gamma\, d_{uv} - c_{uv}| \geq 1 / (2n^2D^2)$.

***Proof*:** Follows from $|c(T') - c(T)| = t\,|y_u - y_v - \gamma\, d_{uv} - c_{uv}|$ and Facts 2 and 3. ■

Theorem 3.5: NBS algorithm terminates within $2n^3D^2C$ pivots, where $n = |V|$ and D (*resp*. C) is the maximum link delay (*resp*. cost) and hence its complexity is pseudo-polynomial.

***Proof*:** Let T_0, $T_1 \ldots T_l$ be the sequence of consecutive feasible basic solutions. It suffices to show that $l \le 2(n\ D)^3$. According to Fact 4, $c(T_0) - c(T_l) \ge l\ /(2\ (n\ D)^2)$ and $c(T_0) \le n\ C$.

This implies that $l \le 2\ n^3\ D^2\ C$. Since each pivot requires O(m) operations, the NBS algorithm is of pseudo-polynomial complexity. ■

Using similar arguments, the revised simplex method that allows degenerate pivots but only uses the anti-cycling strategy of Section 3.4.6 can also be shown to be of pseudo-polynomial time complexity.

3.5.3.2. *An Approximate Solution to the TCSP / CSP Problem and Performance Bounds*

If the optimal basic solution subgraph for the RELAX-TCSP problem contains no cycle, then clearly the *s*-*t* path in this subgraph is also the optimum solution to the original CSP problem. On the other hand, if the optimal basic solution graph contains a cycle, then the optimum flow can be decomposed into flows along two directed *s*-*t* paths p_1 and p_2 with positive flow along each path.

Lemma 3.8: If $c(p_2) \le c(p_1)$, then either $c(p_2) \le c(p^*) \le c(p_1)$ and $d(p_2) \ge \Delta \ge d(p_1)$, where p^* is the optimal path of the original CSP problem or one of the two paths p_1 and p_2 is optimal.

***Proof*:** Let $0 < \lambda < 1$ and $1 - \lambda$ be the flows on p_1 and p_2, respectively. We have

$$\lambda\ d(p_1) + (1 - \lambda)\ d(p_2) = \Delta, \text{ and} \tag{3.28}$$

$$\lambda\ c(p_1) + (1 - \lambda)\ c(p_2) \le c(p^*). \tag{3.29}$$

It follows from (3.28) that

$$\min\{d(p_1), d(p_2)\} \le \Delta \le \max\{d(p_1), d(p_2)\}. \tag{3.30}$$

By (3.29), $c(p_1)$ and $c(p_2)$ cannot both be greater than $c(p^*)$. So $c(p_2) \le c(p^*)$.

If $c(p_2) = c(p^*)$ then by (3.29), $c(p_1) \le c(p^*)$ which implies p_1 or p_2 is optimal.

Assume $c(p_2) < c(p^*)$. Now $\min(d(p_1), d(p_2)) = d(p_1) \le \Delta$, for otherwise p_2 will be a feasible solution to the CSP problem with cost smaller than $c(p^*)$.

So we have the required inequality $d(p_2) \ge \Delta \ge d(p_1)$.

Also path p_1 is feasible for the original CSP problem by Theorem 3.1. So $c(p_1) \ge c(p^*)$. Thus we have the required inequality $c(p_2) \le c(p^*) \le c(p_1)$. ■

It follows from the above lemma that the path p_1 is a feasible solution to the TCSP problem. We may use this as an approximate solution to the original CSP problem. We next evaluate the quality of this approximate solution.

Theorem 3.6: Let p_1 and p_2 be the two paths derived from the optimal solution to the RELAX-TCSP problem with $c(p_1) \ge c(p_2)$, then

$$\frac{c(p_1)}{c(p^*)} \le 1 + \frac{1-\lambda}{\lambda}\left(1 - \frac{c(p_2)}{c(p_1)}\right) \text{ and } \frac{d(p_2)}{\Delta} \le 1 + \frac{\lambda}{1-\lambda}\left(1 - \frac{d(p_1)}{\Delta}\right),$$

where λ is the flow on path p_1 at termination and Δ is the delay bound.

***Proof*:** From $\lambda \cdot c(p_1) + (1-\lambda)c(p_2) \le c(p^*)$, we obtain

$$\frac{c(p_1)}{c(p^*)} \le \frac{c(p^*) - (1-\lambda)\cdot c(p_2)}{\lambda \cdot c(p^*)} = \frac{1}{\lambda} - \frac{1-\lambda}{\lambda}\frac{c(p_2)}{c(p^*)}.$$

Because $c(p^*) \le c(p_1)$, $\frac{1}{\lambda} - \frac{1-\lambda}{\lambda}\frac{c(p_2)}{c(p^*)} \le \frac{1}{\lambda} - \frac{1-\lambda}{\lambda}\frac{c(p_2)}{c(p_1)} = 1 + \frac{1-\lambda}{\lambda}\left(1 - \frac{c(p_2)}{c(p_1)}\right)$.

Similarly, we can prove that $\frac{d(p_2)}{\Delta} \le 1 + \frac{\lambda}{1-\lambda}\left(1 - \frac{d(p_1)}{\Delta}\right)$. ■

Using a special example below, we can show that no constant factor approximation solution based on relaxation approach (including NBS and LARAC algorithm) is possible (However, simulations show that the approximate solution is very close to the optimum). For closing the gap between the optimum value and the approximate value see Section 2.6.

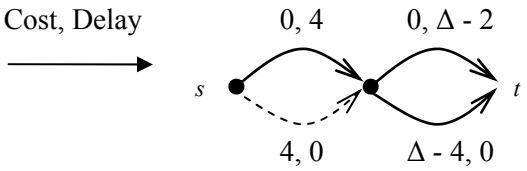


Figure 3.6: An example showing that the gap can be arbitrarily large

Let OPT, OPT_S, and Δ denote the optimal cost, the cost of the path returned by relaxation method, and the delay upper bound. In Figure 3.6, the solid links correspond to the basic variables in the optimal basis. Thus $OPT_S = \Delta - 4$. Since $OPT = 4$, $|OPT_S - OPT| / OPT = (\Delta - 8) / 4$, where Δ can be specified arbitrarily.

3.6. Simulation and Comparative Performance Evaluation

We compared our NBS algorithm with the general purpose LP solvers, LARAC algorithm [23], parametric search based LARAC algorithm [61] (denoted as PARA), and the LHWHM algorithm [37]. The LARAC algorithm has time complexity of $O(m^2 \log^4 m)$ [25] while the parametric search based LARAC algorithm has better complexity, namely, $O((m + n \log n)^2)$ [65]. However, the complexity results are derived using the worst scenario and thus they may not be an accurate indicator of the performance of algorithms on average basis. So we compared the four methods using simulations**.**

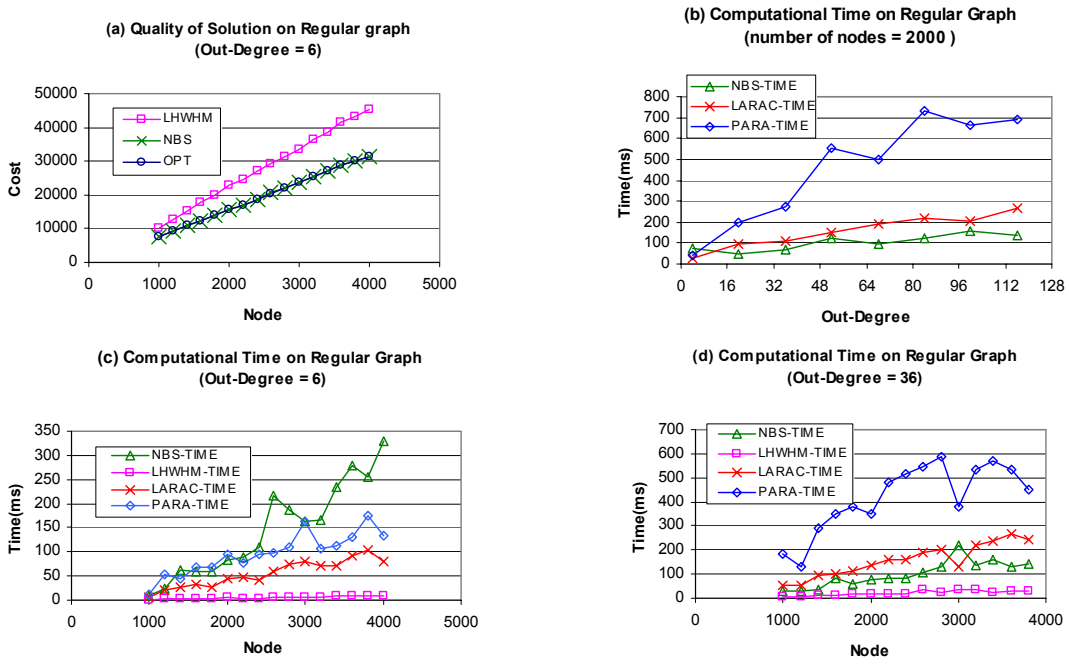


Figure 3.7: Simulation on regular graphs

We use three classes of network topologies: regular graphs $H_{k,n}$ (see [55]), Power-Law Out-Degree graph [44], and Waxman's random graph [59]. For a network $G(V, E)$, the nodes are labeled as 1, 2…, $n = |V|$. Nodes $n / 2$ and n are chosen as the source and target nodes. For the Power-Law Out-Degree graph and Waxman's random graph, the hop number of feasible s-t paths is usually very small even when the network is very large. This will bias the results in favor of the LHWHM algorithm. So, for Waxman's random graphs, a link joining node u and v is added if $|u - v| < |V| / 50$ besides other rules for generating random graphs. We keep the original version of Power-Law Out-Degree graph as in [44]. Even though this kind of graphs favors the LHWHM algorithm, the comparison of the performance of the LARAC and NBS algorithms is still an indicator of the merits of NBS. The link costs and delays are randomly independently

generated even integers in the range from 1 to 200. The delay bound is 1.2 times the delay of the minimum delay *s*-*t* paths in *G*.

The results are shown in Figure 3.7-3.10. Experiments show that NBS algorithm can usually find better solutions than the LARAC algorithm by selecting the best feasible path encountered during the execution instead of the optimum path to the RELAX-TCSP problem. We also find that for sparse graphs (Figure 3.7(c)), NBS takes more time than the LARAC algorithm. However, when the network is dense (large out-degree, See Figure 3.7(d)), NBS beats LARAC. Basically, NBS algorithm is a neighbor search algorithm in which a better solution is derived from the current solution. At each pivot, the NBS algorithm tries all the nodes in the *s*-*t* path in the current basic graph in order to find an in-arc emanating from a node in the path. When the graph is dense, it is more likely that an eligible in-arc can be found in fewer tries. On the other hand, the LARAC algorithm invokes a series of Dijkstra's shortest path algorithm. When the graph is denser, each step in Dijkstra's algorithm takes more time since Dijkstra's algorithm checks all the neighbors of the currently processed node.

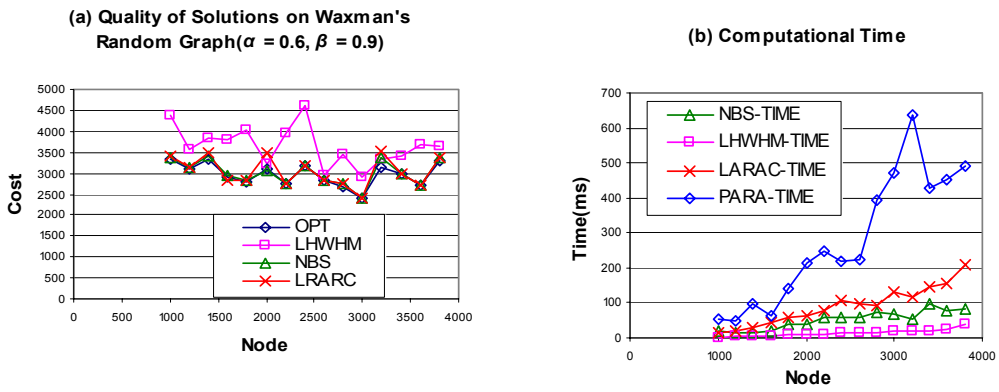


Figure 3.8: Waxman's random graphs

We also compared the NBS algorithm with general purpose LP solvers: CPLEX 8.0 (www.ilog.com/products/cplex), QSopt (www2.isye.gatech.edu/~wcook/qsopt), and CLP (www.coin-or.org). Among all the three solvers, CPLEX is always the fastest (this is not surprising because CPLEX is recognized as one of the best LP solvers). So we only report the experiments with CPLEX. In our experiments with CPLEX, we have used the same graphs as above. Using CPLEX package, we may choose different optimizers such as the primal dual method, network simplex *etc*. Our experiments show that the CPLEX using the primal dual uses the least time and so our comparison is with respect to this optimizer. Notice that CPLEX can also retrieve the network structure underlying the CSP problem. But we found that this does not help decrease the running time. Actually, it takes longer time to find the optimal solution if CPLEX is directed to use the special structure of the networks. The numerical simulation results in Figure 3.10 show that the NBS algorithm is faster.

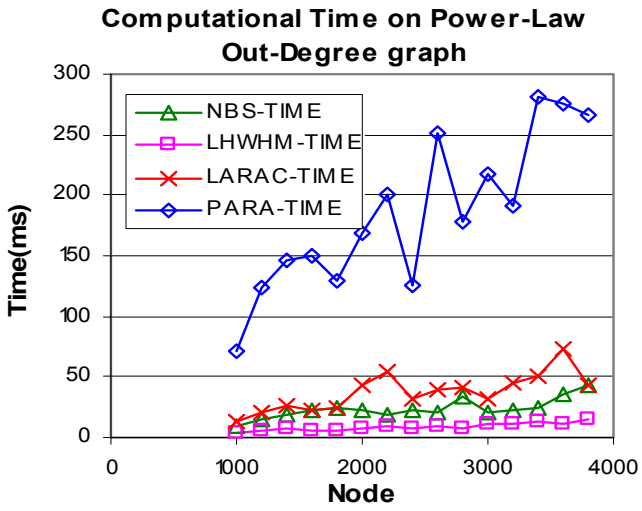


Figure 3.9: Power-Law Out-Degree graphs

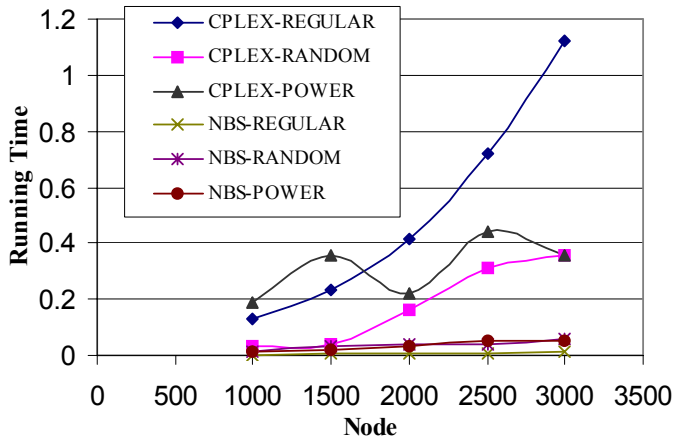


Figure 3.10: NBS and CPLEX comparison

3.7. Summary

In this chapter, we studied the QoS routing problem (or equivalently the CSP problem) from the primal perspective in contrast to most of the currently available approaches that studied the problem from a dual perspective. Specifically we applied the revised simplex method on the primal form of the RELAX-TCSP problem. Several strategies are employed to achieve efficient implementation of the revised simplex method. These strategies include: explicit formulas to solve the systems of equations needed to find entering and leaving variables, an anti-cycling strategy, and a strategy to avoid degenerate pivots. These result in two algorithms. One of these allows degenerate pivots and uses an anti-cycling strategy developed in this chapter. The other algorithm called NBS algorithm avoids degenerate pivots. We show that both algorithms are of pseudo-polynomial-time complexity. We have also shown how to extract an approximate solution to the original CSP problem from the optimum solution to the RELAX-TCSP

problem and derive bounds on the quality of this solution with respect to the optimum solution. Extensive simulation results are presented to demonstrate that our approach compares favorably with the LARAC algorithm and is faster on dense graphs. Also, our algorithm is faster than the general purpose LP solvers.

Chapter 4. Constrained Shortest Link-Disjoint Paths Selection (CSDP(k)): A Network Programming Based Approach

4.1. Introduction

In this chapter we are interested in selecting a set of paths satisfying certain constraints. This problem is a fundamental one and arises in several applications. Specifically the problem, denoted as the CSDP(k) problem, is to select a set of k link-disjoint paths from s to t such that the total cost of these paths is minimum and that the total delay of these paths is not greater than a specified bound. The CSDP(k) problem arises in the context of provisioning paths in a network that could be used to provide resilience to failures in one or more of these paths. Note that this is a generalized version of the CSP problem considered in Chapter 3 and so it is NP-hard. This has led researchers to propose heuristics and approximation algorithms for these problems.

Orda *et al.* [43] have studied the CSDP(2) problem extensively and have provided several approximation algorithms. A special case of the CSDP(k) problem which does not have the delay requirement has been studied in [54]. The algorithms in [23] and [54] can be integrated to provide an approximate solution to the CSDP(k) problem. We call this the G-LARAC(k) algorithm.

The rest of the chapter is organized as follows. In Section 4.2 we define the CSDP(k) problem and a generalized version of this problem called the GCSDP(k) problem. The GCSDP(k) problem requires that the delay of each path in the set of link-disjoint paths

be bounded by a specified value. This is in contrast to the CSDP(k) problem wherein the delay constraint is with respect to the total delay of the paths. However, even finding two delay constrained link-disjoint paths is NP-hard and is not approximable within a factor of $2 - \varepsilon$ for any $\varepsilon > 0$ [32]. We first show that the optimal objective values of the LP relaxations of these two problems have equal value. Hence we focus our study on the relaxed version of the CSDP(k) problem, namely, the RELAX-CSDP(k) problem. In Section 4.3 we review the G-LARAC(k) algorithm which is a dual based approach to solving RELAX-CSDP(k). In Section 4.4 we introduce a transformation on the CSDP(k) problem. The transformed problem will be called the TCSDP(k) problem. We show that the CSDP(k) problem and the TCSDP(k) are equivalent. As we show later in the chapter the transformed problem has several properties that enable us to achieve an efficient implementation of our approach. In the remainder of the chapter we study the LP relaxation of the TCSDP(k) problem, namely, RELAX-TCSDP(k), using the revised simplex method of linear programming. In Section 4.5, several properties of basic solutions of RELAX-TCSDP(k) are established. We also show how to extract an approximate solution to the CSDP(k) problem starting from an optimal solution to RELAX-TCSDP(k). In Sections 4.6-4.7, the revised simplex method and several issues relating to an efficient implementation are discussed. We also develop an anti-cycling strategy and establish the pseudo-polynomial time complexity of the revised simplex method when applied on RELAX-TCSDP(k). Simulation results comparing our approach with the G-LARAC(k) algorithm and the commercially available CPLEX package are presented in Section 4.8. These results demonstrate that our algorithm is faster than currently available approaches. They also indicate that in most cases the

individual delays of the paths produced starting from RELAX-CSDP(k) do not deviate in a significant way from the individual delay requirements of the GCSDP(k) problem, thereby demonstrating that there is not much loss of generality in focusing on RELAX-CSDP(k) rather than on the relaxed version of the more complex GCSDP(k) problem. We conclude in Section 4.9 with a summary of our work and pointing to certain directions for future research.

The results in this chapter have been repeated in [62].

4.2. Constrained Shortest Link-Disjoint Paths Selection Problems: Formulations, Relaxations and Their Equivalence

We first define two classes of link-disjoint paths selection problems. One is a special case of the other. They both admit integer linear programming (ILP) formulations. They are computationally intractable because of the integrality constraints. For networks involving small numbers of nodes and links, these problems can be solved using any general purpose ILP package. For larger networks, faster algorithms that exploit the special network structure of the problems are desired. So, we are interested in solving these problems after relaxing the integrality requirement. The relaxed versions of these problems are upper bounded LP problems. The main result in this section is that the relaxed versions of both these problems are equivalent in the sense they have the same optimal objective value.

General Constrained Shortest *k*-Disjoint Paths Problem (GCSDP(*k*)): Given two nodes s and t and a positive integer T, the GCSDP(k) problem is to find a set of k ($k \geq 2$)

link-disjoint *s*-*t* paths $p_1, p_2 \ldots, p_k$ such that the delay of each path p_i is at most T and the total cost of the k paths is minimum.

Constrained Shortest *k*-Disjoint Paths Problem (CSDP(*k*)): Given two nodes s and t, and a positive integer T, the CSDP(k) problem is to find a set of k link disjoint *s*-*t* paths $p_1, p_2 \ldots, p_k$ such that the total delay of these paths is at most $k\,T$ and that the total cost of the k paths is minimum.

Both the above problems can be formulated as ILP problems. Relaxing the integrality constraints we get the following relaxed versions of these problems.

RELAX-GCSDP(*k*)

$$\text{Minimize: } \sum_{(u,v)\in E} c_{uv} \sum_{i=1}^{k} x_{uv}^{i} \tag{4.1}$$

subject to

For $i = 1, 2 \ldots, k, \forall u \in V$,

$$\sum_{\{v|(u,v)\in E\}} x_{uv}^{i} - \sum_{\{v|(v,u)\in E\}} x_{vu}^{i} = \begin{cases} 1 & \text{if} \quad u = s \\ -1 & \text{if} \quad u = t \\ 0 & \text{otherwise.} \end{cases} \tag{4.2}$$

$$\sum_{(u,\,v)\,\in\,E} d_{uv} \cdot x_{uv}^{i} \le T \tag{4.3}$$

$$\sum_{i=1}^{k} x_{uv}^{i} \le 1 \text{ and } x_{uv}^{i} \ge 0 \text{ for all } (u, v) \in E \tag{4.4}$$

The solutions to the above problem may not, in general, be integral. However, every integer solution defines a set of k link-disjoint *s*-*t* paths. In other words, an integer solution $\boldsymbol{X}^{i} = \{x_{uv}^{i}\}_{(u,\,v)\,\in\,E}$ for $i = 1, 2\ldots, k$ is the flow vector corresponding to the ith path p_i, i.e., link (u, v) is in path p_i *iff* $x_{uv}^{i} = 1$.

RELAX-CSDP(k)

$$\text{Minimize: } \sum_{(u,v)\in E} c_{uv} \cdot x_{uv} \tag{4.5}$$

subject to

$$\forall\ u \in V,\ \sum_{\{v|(u,v)\in E\}} x_{uv} - \sum_{\{v|(v,u)\in E\}} x_{vu} = \begin{cases} k & if \quad u = s \\ -k & if \quad u = t \\ 0 & otherwise. \end{cases} \tag{4.6}$$

$$\sum_{(u,v)\in E} d_{uv} \cdot x_{uv} \ \le\ k\ T \text{ and} \tag{4.7}$$

$$0 \le x_{uv} \le 1, \text{ for all } (u, v) \in E$$

We now proceed to show that the RELAX-GCSDP(k) and RELAX-CSDP(k) are equivalent in the sense that they both have optimal solutions with the same value for the objective. Let $\boldsymbol{\Lambda} = (\lambda_1, \lambda_2 \ldots, \lambda_k) \ge \mathbf{0}$ and define

$$L_G(k, \boldsymbol{\Lambda}) = \text{Minimize } \{ \sum_{(u,v)\in E} c_{uv} \sum_{i=1}^{k} x_{uv}^i \ + \sum_{i=1}^{k} \lambda_i \Big(\sum_{(u,v)\in E} d_{uv} \cdot x_{uv}^i - T \Big) \}.$$

Then the Lagrangian dual of RELAX-GCSDP(k) is as follows.

LAGRANGIAN-GCSDP(k)

Maximize $L_G(k, \boldsymbol{\Lambda})$ among all $\boldsymbol{\Lambda} \ge \mathbf{0}$

subject to

For $i = 1, 2 \ldots, k$, $\forall\ u \in V$,

$$\sum_{\{v|(u,v)\in E\}} x_{uv}^i - \sum_{\{v|(v,u)\in E\}} x_{vu}^i = \begin{cases} 1 & if \quad u = s \\ -1 & if \quad u = t \\ 0 & otherwise. \end{cases} \tag{4.8}$$

$$\sum_{i=1}^{k} x_{uv}^i \le 1 \text{ and } x_{uv}^i \ge 0 \text{, for all } (u, v) \in E \tag{4.9}$$

The vector $\boldsymbol{\Lambda}$ is called the Lagrangian multiplier. The above problem can be solved by finding the Lagrangian multiplier vector $\boldsymbol{\Lambda}$ that maximizes $L_G(k, \boldsymbol{\Lambda})$.

Property 4.1: Given any Lagrangian multiplier $\boldsymbol{\Lambda} = (\lambda_1 \ldots, \lambda_k)$, let $\boldsymbol{\Lambda}'$ be obtained by permuting the components of $\boldsymbol{\Lambda}$. Then $L_G(k, \boldsymbol{\Lambda}) = L_G(k, \boldsymbol{\Lambda}')$.

Property 4.2 ([7]): $L_G(k, \boldsymbol{\Lambda})$ is a concave function of $\boldsymbol{\Lambda}$.

Property 4.3: There exists $\boldsymbol{\Lambda}$ with all components equal that maximizes $L_G(k, \boldsymbol{\Lambda})$.

Proof: Let $\boldsymbol{\Lambda}^* = (\lambda_1 \ldots, \lambda_k)$ be a maximizing multiplier.

Let $S(\boldsymbol{\Lambda}^*)$ denote the set of vectors whose elements are permutations of the elements of $\boldsymbol{\Lambda}^*$.

Let $C = |S(\boldsymbol{\Lambda}^*)| = k!$ and $\theta = 1 / C$.

By Property 4.1, $\forall H \in S(\boldsymbol{\Lambda}^*)$, $L_G(k, \boldsymbol{\Lambda}^*) = L_G(k, \boldsymbol{H})$.

By the concavity of $L_G(k, \boldsymbol{\Lambda}^*)$,

$$L_G(k, \boldsymbol{\Lambda}^*) = \theta \; C\, L(k, \boldsymbol{\Lambda}^*) = \sum_{H \in S(\Lambda)} \theta L(k, H) \le L(k, \sum_{H \in S(\Lambda)} \theta H) \le L(k, \boldsymbol{\Lambda}^*).$$

So, $\boldsymbol{\Gamma} = \sum_{H \in S(\Lambda)} \theta H$ is also a maximizing multiplier and $\boldsymbol{\Gamma}$ has identical components. ■

By Property 4.3, $L_G(k, \boldsymbol{\Lambda})$ can be reformulated with respect to some $\boldsymbol{\Lambda} = (\lambda, \lambda \ldots, \lambda) \ge \mathbf{0}$ as follows:

$$L_G(k, \boldsymbol{\Lambda}) = \text{Minimum}\{ \sum_{(u,v) \in E} (c_{uv} \sum_{i=1}^{k} x_{uv}^i) + \lambda (\sum_{(u,v) \in E} (d_{uv} \sum_{i=1}^{k} x_{uv}^i) - k\, T)\}. \quad (4.10)$$

Let

$$\bar{x}_{uv} = \sum_{i=1}^{k} x_{uv}^i \text{ for all } (u, v) \in E. \quad (4.11)$$

We now define **UNIFORM-LAGRANGIAN-GCSDP**(k) as follows.

First let $L(k, \lambda) = \min \{ \sum_{(u,v) \in E} c_{uv} \bar{x}_{uv} + \lambda (\sum_{(u,v) \in E} d_{uv} \bar{x}_{uv} - k\, T)\}$.

UNIFORM-LAGRANGIAN-GCSDP(k):

Maximize $L(k, \lambda)$ among all scalars $\lambda \geq 0$ (4.12)

subject to

$$\forall\, u \in V,\ \sum_{\{v|(u,v)\in E\}} \bar{x}_{uv} - \sum_{\{v|(v,u)\in E\}} \bar{x}_{vu} = \begin{cases} k & if \quad u = s \\ -k & if \quad u = t \\ 0 & otherwise. \end{cases} \tag{4.13}$$

$$0 \leq \bar{x}_{uv} \leq 1, \text{ for all } (u, v) \in E \tag{4.14}$$

Note that (4.13) is obtained by summing up the k flow balance constraints in (4.8) and that λ is a scalar.

Theorem 4.1: UNIFORM-LAGRANGIAN-GCSDP(k) and **LAGRANGIAN-GCSDP(k)** have the same optimal value for the objective.

Proof: Let $\boldsymbol{\Lambda} = (\lambda, \lambda \ldots, \lambda) \geq 0$. We first show $L_G(k, \boldsymbol{\Lambda}) \geq L(k, \lambda)$.

Let $\{x^i_{uv}\}_{(u,\, v)\, \in\, E,\, \mathrm{i}\, =\, 1\, \ldots,\, k}$ minimize $L_G(k, \boldsymbol{\Lambda})$. Then we have

$$\begin{aligned} L_G(k, \boldsymbol{\Lambda}) &= \sum_{(u,v)\in E} c_{uv} \sum_{i=1}^{k} x^i_{uv} + \sum_{i=1}^{k} \lambda \Big(\sum_{(u,v)\in E} d_{uv} \cdot x^i_{uv} - T \Big) \\ &= \sum_{(u,v)\in E} c_{uv} \bar{x}_{uv} + \lambda \Big(\sum_{(u,v)\in E} d_{uv} \bar{x}_{uv} - k\, T \Big) \geq L(k, \lambda), \end{aligned}$$

where $\bar{x}_{uv}$ is defined as in (4.11).

It follows from the unimodularity [1] of the constraints (4.13)-(4.14) that for a given λ, there exists an optimal integer solution to UNIFORM-LAGRANGIAN-CSDP(k) problem. Also an integer solution $\boldsymbol{Y} = \{y_{uv}\}_{(u,v)\, \in\, E}$ of UNIFORM-LAGRANGIAN-CSDP(k) that achieves the minimum in $L(k, \lambda)$ defines a set of k link-disjoint s-t paths $P_k = (p_1, p_2, \ldots p_k)$. Let $\boldsymbol{X}^i = \{x^i_{uv}\}_{(u,v)\, \in\, E}$ be the flow vector for path p_i, i.e., $x^i_{uv} = 1$ *iff* $(u, v) \in p_i$; otherwise, $x^i_{uv} = 0$.

Observe that $y_{uv} = \sum_{i=1}^{k} x_{uv}^{i}$. Then

$$L(k, \lambda) = \sum_{(u,v)\in E} c_{uv} y_{uv} + \lambda \left(\sum_{(u,v)\in E} d_{uv} y_{uv} - k\, T \right)$$

$$= \sum_{(u,v)\in E} c_{uv} \sum_{i=1}^{k} x_{uv}^{i} + \sum_{i=1}^{k} \lambda \left(\sum_{(u,v)\in E} d_{uv} \cdot x_{uv}^{i} - T \right) \geq L_G(k, \boldsymbol{\Lambda}).$$

Hence, $L(k, \lambda) \geq L_G(k, \boldsymbol{\Lambda})$. So $L(k, \lambda) = L_G(k, \boldsymbol{\Lambda})$.

By Property 4.3, there exists a vector $\boldsymbol{\Lambda}^* = (\lambda^*, \lambda^* \ldots, \lambda^*)$ that maximizes $L_G(k, \boldsymbol{\Lambda})$. Let η^* be a maximizing multiplier for $L(k, \lambda)$ and denote $H^* = (\eta^* \ldots, \eta^*)$.

By definition of $\boldsymbol{\Lambda}^*$ and η^*, we have

$$L_G(k, \boldsymbol{\Lambda}^*) = L(k, \lambda^*) \leq L(k, \eta^*) = L_G(k, H^*) \leq L_G(k, \boldsymbol{\Lambda}^*).$$

Hence $L_G(k, \boldsymbol{\Lambda}^*) = L(k, \eta^*)$. ■

The above theorem has an important implication. It shows that the optimal objective to RELAX-GCSDP(k) can be obtained by solving UNIFORM-LAGRANGIAN-GCSDP(k). But UNIFORM-LAGRANGIAN-GCSDP(k) is the general linear programming dual of the RELAX-CSDP(k) problem (See page. 183 of [5]). Thus we have the following result by the strong duality theorem [5].

Theorem 4.2: RELAX-GCSDP(k) and RELAX-CSDP(k) have the same optimal objective value.

The intuition behind the above result is as follows. The indistinguishability of the k path constraints represented by (4.3) guarantees that if P is a set of feasible paths constituting a solution to RELAX-GCSDP(k) problem then any permutation of these paths is also a solution (Property 4.1). Also in the optimum solution there is no reason

for paths to be weighted differently (Property 4.3). As formally proved, these two properties lead to Theorem 4.2.

Theorem 4.2 implies that if we are interested only in obtaining the optimal objective value of the RELAX-GCSDP(k), then starting with the RELAX-CSDP(k) does not result in any loss of generality. In view of this, we shall focus on RELAX-CSDP(k) in the rest of the chapter.

4.3. G-LARAC(k) Algorithm: A Dual Based Approach to RELAX-CSDP(k)

The G-LARAC(k) algorithm [6] is a generalization of the LARAC algorithm [23] that was specifically designed for the CSP problem. The G-LARAC(k) algorithm may be viewed as an algorithm for solving RELAX-CSDP(k) problem using its Lagrangian dual which is the same as UNIFORM-LAGRANGIAN-GCSDP(k) repeated below.

In the following we use Δ in place of $k\,T$.

UNIFORM-LAGRANGIAN-GCSDP(k)

$$\text{Maximize} \quad L(k, \lambda) = \text{Minimize}\{ \sum_{(u,v)\in E} c_{uv}x_{uv} + \lambda \left(\sum_{(u,v)\in E} d_{uv}x_{uv} - \Delta \right)\} \text{ for all } \lambda \geq 0.$$

$$\text{subject to} \quad \sum_{\{v|(u,v)\in E\}} x_{uv} - \sum_{\{v|(v,u)\in E\}} x_{uv} = \begin{cases} k & if \quad u = s \\ -k & if \quad u = t \\ 0 & otherwise. \end{cases}$$

$$0 \leq x_{uv} \leq 1, \text{ for all } (u, v) \in E.$$

Given λ, $L(k, \lambda)$ = Minimum$\{ \sum_{(u,v)\in E} c_{uv}x_{uv} + \lambda (\sum_{(u,v)\in E} d_{uv}x_{uv} - \Delta)\}$ is achieved by a set of k link-disjoint paths with minimum total weight, where the weight associated with link (u, v) is given by $c_{uv} + \lambda\ d_{uv}$. The key issue is how to search for the optimal λ that maximizes $L(k, \lambda)$ and determining the termination condition for the search. The G-LARAC(k) algorithm presented in Figure 4.1 is one such efficient search procedure. In this procedure c_λ cost of a path (also called minimum aggregate cost) refers to the cost of the path computed using $c_{uv} + \lambda\ d_{uv}$ as the cost of link (u, v).

```
Procedure G-LARAC (s, t, Δ, k)
  P_c :=Disjoint (s, t, c, k) /*Compute minimum cost of a set of k link-disjoint s-t paths*/
  if d(P_c) ≤ Δ then return P_c
  P_d := Disjoint (s, t, d, k) /*Compute minimum delay of a set of k link-disjoint s-t paths*/
  if d(P_d) > Δ then return "no solution"
  repeat
       λ = (c(P_c) – c(P_d)) / (d(P_d) – d(P_c))
       R = Disjoint(s, t, c_λ, k) /*Compute minimum aggregate cost c_λ of s-t path*/
       if c_λ (R) = c_λ (P_c) then return P_d
         else if  d(R) ≤ Δ then  P_d := R else  P_c := R
   end repeat
end procedure
```

Figure 4.1: G-LARAC(k) algorithm

Basically G-LARAC(k) performs the following steps.

1. In the first step, the algorithm calculates the minimum cost of a set of k link-disjoint s-t paths using link costs. This can be done by the algorithm in [54]. If the total delay of these paths is at most Δ, this is surely the required set of paths. Otherwise, the algorithm stores this set as the latest infeasible set, simply called the P_c set. Then it determines the minimum delay of a set of k

link disjoint *s*-*t* paths, called the P_d set. If P_d is infeasible, there is no solution to this instance.

2. Set $\lambda = (c(P_c) - c(P_d)) / (d(P_d) - d(P_c))$. With this value of λ, we can find a set of k link-disjoint paths with minimum c_λ-cost. Let this set be denoted as R. If $c_\lambda(R) = c_\lambda(P_c)$ $(= c_\lambda(P_d))$, we have obtained the optimal λ. Otherwise, set R as the new P_c or P_d according to whether R is infeasible or feasible.

A detailed discussion of several issues relating to G-LARAC(k) and properties of solutions produced by G-LARAC(k) were presented in Chapter 2.

In contrast to the dual approach taken by the G-LARAC(k) algorithm our interest in the remainder of the chapter is to design an approach to obtain an approximate solution to the CSDP(k) problem using the primal simplex method of linear programming.

4.4. Transformation of the RELAX-CSDP(*k*) problem

To achieve an efficient implementation of our approach to the RELAX-CSDP(k) problem we consider problem TCSDP(k) on a transformed network defined as follows.

- The graph of the transformed problem is the same as that of the original problem, that is, $G(V, E)$.
- For all $(u, v) \in E$, d'_{uv} and c'_{uv} in the transformed problem are given by $d'_{uv} = 2\, d_{uv}$ and $c'_{uv} = c_{uv}$.
- The new upper bound Δ' in the transformed problem is given by $\Delta' = 2\,\Delta + 1$.

Note that the transformation above is the same as the transformation for the CSP problem presented in Chapter 3.

Let P^k denote a set of k link-disjoint s-t paths. Let $c(P^k)$ and $d(P^k)$ denote the total cost and the total delay of the k paths in P^k. The TCSDP(k) problem asks for a set of k link-disjoint s-t paths with minimum total cost and with total delay at most Δ'.

Theorem 4.3: P^k is a feasible solution (*resp.* an optimal solution) to the CSDP(k) problem *iff* it is a feasible solution (*resp.* an optimal solution) to the TCSDP(k) problem.

***Proof*:** Given the set of paths P^k, let $Tc(P^k)$ and $Td(P^k)$ denote the total cost and the total delay of the paths in P^k in the TCSDP(k) problem, respectively. Evidently, $Tc(P^k) = c(P^k)$ and $Td(P^k) = 2\ d(P^k)$. It suffices to show that P^k is feasible in the CSDP(k) problem *iff* it is feasible in the TCSDP(k) problem.

If P^k is feasible in the CSDP(k) problem, then $d(P^k) \le \Delta$ and $Td(P^k) = 2\ d(P^k) \le 2\ \Delta < 2\ \Delta + 1$. So P^k is also feasible in the TCSDP(k) problem.

If P^k is feasible in the TCSDP(k) problem, then $2\ d(P^k) = Td(P^k) \le \Delta' = 2\ \Delta + 1$. From the assumption that delays have integer values it follows that $d(P^k) \le \Delta$. So P^k is also feasible in the CSDP(k) problem. ■

In view of this equivalence we only consider, in the rest of the chapter, the RELAX-TCSDP(k) problem. Also we use Δ (being odd) and d_{uv} (being even) to denote the delay bound and link delay in the TCSDP(k) problem. Notice that the transformation does not change the costs of paths.

We conclude this section by recalling some terminology defined in Section 3.3 and presenting the RELAX-TCSDP(k) problem in matrix form.

Let the links be labeled as e_1, e_2 …, e_m and the nodes be labeled as 1, 2 …., n. We shall denote the delay of each edge e_i as d_i and the cost of e_i as c_i. The incidence matrix of G has m columns, one for each link and n rows, one for each node [55]. The rank of

this matrix is $n - 1$, and removing any row of this matrix will result in a matrix of rank $n - 1$. We denote this resulting matrix of rank $n - 1$ as $\boldsymbol{H}$. We also assume that the row removed from the incidence matrix corresponds to node n. We denote the column of $\boldsymbol{H}$ corresponding to e_k by the vector $\boldsymbol{h}_k$. For $e_k = (i, j)$, $i, j \neq n$ we have $\boldsymbol{h}_k = (h_{1,k}..., h_{i,k},\ldots h_{j,k},\ldots h_{n-1,k})^t$ with all its components being 0 except for $h_{i,k} = 1$ and $h_{j,k} = -1$. Also for $e_k = (i, n)$, $h_{i,k} = 1$, and for $e_k = (n, j)$, $h_{j,k} = -1$. Let

$$A = \begin{pmatrix} H & 0 \\ D & -1 \end{pmatrix} = (a_1, a_2 ... a_m, a_{m+1}), \text{ where } \boldsymbol{D} = (-d_1, -d_2 \ldots, -d_m), \tag{4.15}$$

$$a_i = \begin{pmatrix} h_i \\ -d_i \end{pmatrix}, i \leq m \text{ and } a_{m+1} = \begin{pmatrix} 0 \\ -1 \end{pmatrix}. \tag{4.16}$$

Let $\boldsymbol{x}$ be the column vector of the m flow variables x_{uv} and the slack variable w corresponding to the delay constraint (4.7), and $\boldsymbol{c}$ be the row vector $(c_1, c_2 \ldots, c_m, 0)$ of the costs. Note that the cost of the slack variable is 0. Then the RELAX-TCSDP(k) problem (see (4.5)-(4.7)) can be written in matrix form as follows. Note that to conform to the standard form for a minimization problem we have used "≥" form of (4.7) and added a slack variable w, i.e., $\sum_{(u,v)\in E} -d_{uv} \cdot x_{uv} - w = -\Delta$.

RELAX-TCSDP (k)

Minimize $\boldsymbol{c}\,\boldsymbol{x}$

subject to $\boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{b}$ (4.17)

$0 \leq \boldsymbol{x} \leq 1$, for $\forall (u, v) \in E$

$w \geq 0$,

where w is the slack variable added to (4.7) and $\boldsymbol{b} = (b_1, b_2\ldots, b_{n-1}, -\Delta)^t$ with $b_s = k$, $b_t = -k$ and $b_i = 0$ for $i \neq s, t$.

We note that the above problem is almost the same as the minimum cost flow problem except for the additional delay constraint.

The rest of the chapter is concerned with the simplex method based solution of RELAX-TCSDP(k) and several issues relating to its efficient implementation. We want to emphasize that most of these properties hold only with the transformation and we shall use "*" to denote those properties that also hold without the transformation. The cost of the optimal solution to RELAX-TCSDP(k) will be a lower bound to the optimal cost of the original CSDP(k) problem. We will show in the next section how to extract an approximate solution to TCSDP(k) (hence the CSDP(k)) problem from an optimal solution to the RELAX-TCSDP(k) problem.

4.5. Properties of Basic Solutions of RELAX-TCSDP(k) and Generation of an Approximate Solution to the CSDP(k) Problem

Simplex method of linear programming starts with a basic solution and proceeds by constructing one basic solution from another. A basic solution consists of two sets of variables, basic and nonbasic. For the RELAX-TCSDP(k) problem under consideration, all the nonbasic variables in a basic solution will be 0 or 1 [11]. Note that the value of the slack variable, when it is nonbasic, must be equal to 0 because it does not have an upper bound. Given a basic solution, we shall denote the subgraph of G corresponding to the basic variables (except the slack variable if it is in the basic solution) in this

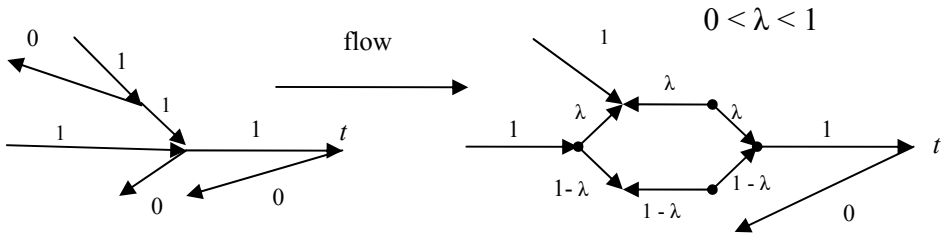
solution by G_b. The subgraph G_b will be called the subgraph of the basic solution or simply the basis graph. The nonsingular submatrix of $\boldsymbol{A}$ defined by the basic variables is called a basis matrix or simply, a basis and is denoted as $\boldsymbol{B}$. The rest of the matrix corresponding to the nonbasic variables is called the nonbasic matrix. In this section we present certain important properties of the basic solutions of the RELAX-TCSDP(k) problem. For the sake of continuity some of the lemmas proved in Chapter 3 will be stated here without proof.

Lemma 4.1* [55]: Let $G(V, E)$ be a directed network with at least one cycle W (not necessarily directed). Assigning an arbitrary orientation to W, let $\boldsymbol{U} = (u_1, u_2, u_3 \ldots, u_m)^t$, where

$$u_j = \begin{cases} 1, & \text{if } e_j \in W \text{ and the orientation of } e_j \text{ agrees with the orientation of } W \\ -1, & \text{if } e_j \in W \text{ and the orientation of } e_j \text{ disagrees with the orientation of } W \\ 0, & \text{otherwise.} \end{cases}$$

Then, $\boldsymbol{H}\,\boldsymbol{U} = 0$.

We shall use $\boldsymbol{U}(W)$ to denote the vector derived from cycle W as in the above lemma. We shall denote by $d(W)$ the signed algebraic sum of the delays of the links in a cycle W as we traverse around the cycle.



a) Tree basic solution b) Basic solution with a cycle

Figure 4.2: Structure of basic solutions

Lemma 4.2*: Every basis matrix contains the last row of $\boldsymbol{A}$.

Lemma 4.3*: The subgraph G_b of a basic solution contains at most one cycle (See Figure 4.2).

Lemma 4.4*: If there is a cycle W in a basic solution, then $d(W) \neq 0$.

Lemma 4.5: If there is no cycle in a basic solution, then for $\forall(u, v) \in E$, $x_{uv} = 0$ or 1. If there is a cycle W in a basic solution, then for $\forall(u, v) \in W$, $0 < x_{uv} < 1$ and for $\forall(u, v) \in E - W$, $x_{uv} = 0$ or 1.

***Proof*:** Let $\boldsymbol{B} = (\boldsymbol{b}_1, \boldsymbol{b}_2 \ldots, \boldsymbol{b}_n)$, $\boldsymbol{A}_N$, $\boldsymbol{x}_B$, and $\boldsymbol{x}_N$ denote the basis matrix, nonbasic matrix, column vector of basic variables, and column vector of nonbasic variables in the basic solution, respectively.

Let $\boldsymbol{b}' = \boldsymbol{b} - \boldsymbol{A}_N \boldsymbol{x}_N$, Then we have $\boldsymbol{B}\,\boldsymbol{x}_B = \boldsymbol{b}'$.

Since all the components in $\boldsymbol{A}_N\boldsymbol{x}_N$ and $\boldsymbol{b}$ are integers, so are all the components in $\boldsymbol{b}'$.

By Cramer's rule, we have

$$x_i = \det(\boldsymbol{B}_i) / \det(\boldsymbol{B}), \text{ where } \boldsymbol{B}_i = (\boldsymbol{b}_1 \ldots, \boldsymbol{b}_{i-1}, \boldsymbol{b}', \boldsymbol{b}_{i+1} \ldots, \boldsymbol{b}_n).$$

We first show x_i is an integer if the corresponding link is not in the cycle. We consider two cases:

Case 1: There is no cycle in the basic solution. Thus the slack variable is a basic variable. Also G_b is a spanning tree. Let the nth column in the basis $\boldsymbol{B}$ correspond to the slack variable.

Then $\boldsymbol{b}_n = (0 \ldots, 0, -1)^t$ and so $\boldsymbol{B}$ has the following form.

$$B = \begin{pmatrix} H' & 0 \\ D' & -1 \end{pmatrix}, \text{ where } \boldsymbol{H}' \text{ is the incidence matrix of } G_b \text{ and } \boldsymbol{D}' \text{ is the}$$

corresponding delay vector.

Since $\boldsymbol{H'}$ is the incidence matrix of a spanning tree it follows that $|\det(\boldsymbol{H'})| = 1$. So $|\det(\boldsymbol{B})| = 1$. Also, $\det(\boldsymbol{B_i})$ is an integer because all the components of $\boldsymbol{B_i}$ are integers. So x_i is also an integer for all i.

Case 2: There is a cycle W in the basic solution. That is, the slack variable is not in the basis.

Let $l = |W|$, i.e., l is the number of links in W.

In this case, we first show that the flow on any link i not on the cycle is an integer. Without loss of generality, let

$$B = \begin{pmatrix} H_W & H' \\ D_W & D' \end{pmatrix} \text{ and } B_i = \begin{pmatrix} H_W & H'_i \\ D_W & D'_i \end{pmatrix},$$

where the columns of $H_W = (h_1, h_2 \ldots, h_l)$ correspond to the links on the cycle W and the components of D_W are the delays of these links. Note that $\boldsymbol{H'_i}$ contains the column vector $\boldsymbol{b'}$.

Let $H'_W = (h_2 \ldots, h_l)$ and $D'_W = (d_2 \ldots, d_l)$. Defining the direction of the link h_1 as the orientation of the cycle W we get by Lemma 4.1 that $\boldsymbol{H_W\, U(W)} = 0$.

Using elementary column operations, $\det(\boldsymbol{B})$ and $\det(\boldsymbol{B_i})$ can be written as:

$$\det(\boldsymbol{B}) = \det\begin{pmatrix} 0 & H'_W & H' \\ D_W \times U(W) & D'_W & D' \end{pmatrix} = (-1)^{n+1} \det((H'_W\ H')) \bullet (D_W \times U(W)),$$

and

$$\det(\boldsymbol{B_i}) = \det\begin{pmatrix} 0 & H'_W & H'_i \\ D_W \times U(W) & D'_W & D'_i \end{pmatrix} = (-1)^{n+1} \det((H'_W\ H'_i)) \bullet (D_W \times U(W))$$

Hence $x_i = \det((H'_W\, H'_i)) / \det((H'_W\, H'))$.

Since all the components in matrix $(H'_W\, H'_i)$ are integers, $\det((H'_W\, H'_i))$ is also an integer.

The denominator is equal to ±1 because (H'_W H') is the incidence matrix of the spanning tree obtained by removing link i from G_b. So it is follows that x_i is an integer. Hence $x_{uv} = 0$ or 1 because $0 \leq x_{uv} \leq 1$.

We next show that if the basis graph contains a cycle, then the flow on each link on the cycle W is less than 1 and greater than 0. Assuming the contrary we establish a contradiction. First recall that the flow on each link that is not in G_b (that is, each nonbasic variable) is either 0 or 1. If the flow on any link on W is an integer (0 or 1) then it follows from the flow balance constraints that all the flows on the links on W will be integers. But this would mean that in the current basic solution the total delay of all the links is an even integer. This violates the requirement that the total delay must be equal to Δ which is odd. ■

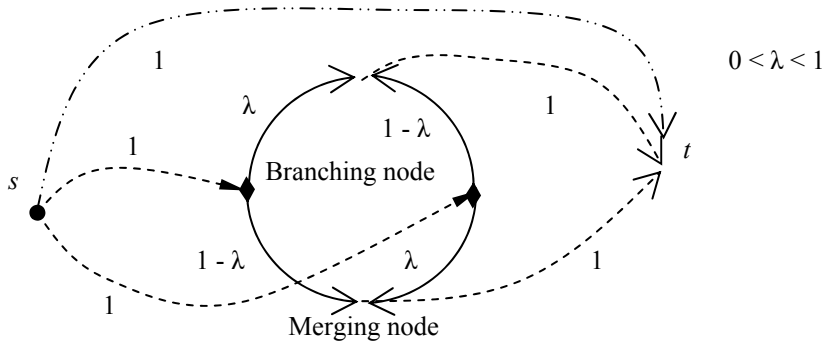


Figure 4.3: Branching and merging nodes

Definition 4.1:

(*a*) In a directed cycle, a node is called a branching (*resp*. merging) node if it is the tail (*resp*. head) of two links in the cycle (See Figure 4.3). A segment of the cycle is the

set of all the links on the cycle between two consecutive branching and merging nodes. A segment consists of consecutive links with the same direction and the direction of a segment is defined as the direction of its links.

(*b*) For a subgraph G_s of G, let

$d(G_s) = \sum_{(u,v) \in G_s} d_{uv}$ and $d_{\boldsymbol{x}}(G_s) = \sum_{(u,v) \in G_s} x_{uv} d_{uv}$ with respect to the flow vector $\boldsymbol{x}$.

Lemma 4.6: Suppose the basis graph G_b contains a cycle W. Let $e = (u, v) \in W$ and $x_{uv} = \lambda$ $(0 < \lambda < 1)$. Define the direction of e as the orientation of W. Then for any link $e' = (i, j)$, $x_{ij} = \lambda$ if the direction of e' agrees with the orientation of W and $x_{ij} = 1 - \lambda$ otherwise (See Figure 4.3).

Proof: This follows from the flow balance constraints and the fact the nonbasic variables have value 0 or 1. ■

Theorem 4.4: Given an optimal solution $\boldsymbol{x}$ to RELAX-TCSDP(k) with G_b as the corresponding basis graph.

(*a*) If G_b contains no cycle then G_b contains a set of k link disjoint s-t paths P^k with $d(P^k) < \Delta$, where Δ is the delay constraint in the TCSDP(k) problem. These paths constitute an optimal solution to the original CSDP(k) problem.

(*b*) Suppose that G_b contains a cycle W and let $G'(V', E')$ be obtained from $G(V, E)$ such that $V' = V$ and $E' = \{(u, v) \in E \mid x_{uv} > 0\}$. Then G' contains a set of k link disjoint s-t paths P^k with $d(P^k) < \Delta$, and a set of k link disjoint s-t paths Q^k with $d(Q^k) > \Delta$, such that $c(Q^k) \leq OPT \leq c(P^*) \leq c(P^k)$, where OPT is the optimal objective value of the RELAX-TCSDP(k) problem, P^* is the optimal integer solution to the TCSDP(k) problem (equivalently, the optimal solution to the CSDP(k)) problem). Also

$$\frac{c(P^k)}{c(P^*)} \le 1 + \frac{1-\lambda}{\lambda}\left(1 - \frac{c(Q^k)}{c(P^k)}\right) \text{ and } \frac{d(Q^k)}{\Delta} \le 1 + \frac{\lambda}{1-\lambda}\left(1 - \frac{d(P^k)}{\Delta}\right),$$

where λ is as defined in Lemma 4.6 with the orientation of the cycle W selected so that $d(W) < 0$.

Proof: (*a*) If there is no cycle in the optimal basis graph G_b, then all the link flows will be integers and the flow vector can be decomposed into unit flows along k link-disjoint *s*-*t* paths. The total delay of these paths will be even and hence less than Δ (because Δ is odd). These integral flows form an optimal integer solution to the RELAX-TCSDP(k) problem and hence an optimal solution to the original TCSDP(k) problem. By Theorem 4.3 this is also an optimal solution to the original CSDP(k) problem.

(*b*) Assume that the basic graph contains a cycle W. By Lemma 4.5, the flows on links in W are nonzero and thus G' contains W. Obviously, $OPT \le c(P^*)$. Also note that $d_x(G')$, the total delay contributed by the flow vector equals Δ (because the slack variable is nonbasic and thus its value in the current basic solution is 0).

Define the orientation of W such that $d(W) < 0$. Now push flow along the orientation of W until some link's flow reaches 0 or 1 (See Figure 4.4(*b*)). By Lemma 4.6, all the resulting link flows will be either 0 or 1. Remove all the links with zero flow from G' and let G_z denote the resulting network. Evidently, the flows on all links in G_z are 1 and $d(G_z) < d_x(G') = \Delta$ (because $d(W) < 0$, the network delay is reduced when we push the flow along the orientation of W). It can also be seen that $c(W) \ge 0$ for otherwise, the cost of the new flow will be less than the cost of the flow defined by G'.

Notice that the above operation does not change the amount of flow from s to t. Since the total flow from s to t in G_z is k and all the flows on all links in G_z are equal to 1, there must be k link disjoint s-t paths P^k and $d(P^k) = d(G_z) < \Delta$.

Similarly, we can obtain Q^k (See Figure 4.4(c)). Here, the flow is pushed along W in the reverse direction. Along this direction, $d(W) > 0$ and so $d(Q^k) > \Delta$.

It can be seen then that $c(W) < 0$ and thus $c(Q^k) \le OPT \le c(P^*) \le c(P^k)$.

In the rest of the proof, we use P^k, Q^k and W to denote the corresponding set of links. Let $W_P = P^k \cap W$ and $W_Q = Q^k \cap W$, i.e., W_P (*resp.* W_Q) is the set of links on both the cycle W and P^k (*resp.* Q^k). Evidently, $W_P \cap W_Q = \varnothing$ and $P^k - W_P = Q^k - W_Q$ (See Figure 4.4(b) and (c)). Then we have

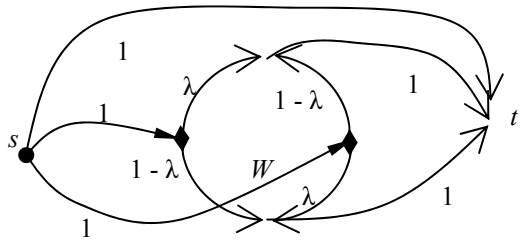
$$\begin{aligned} c(P^*) \ge OPT &= c(P^k - W_P) + \lambda\, c(W_P) + (1-\lambda)\, c(W_Q) \\ &= \lambda\, c(P^k - W_P) + (1-\lambda)\, c(Q^k - W_Q) + \lambda\, c(W_P) + (1-\lambda) c(W_Q) \\ &= \lambda\, (c(P^k - W_P) + c(W_P)) + (1-\lambda)\, (c(Q^k - W_Q) + c(W_Q)) \\ &= \lambda\, c(P^k) + (1-\lambda)\, c(Q^k) \end{aligned}$$

The second equality holds because $P^k - W_P = Q^k - W_Q$.

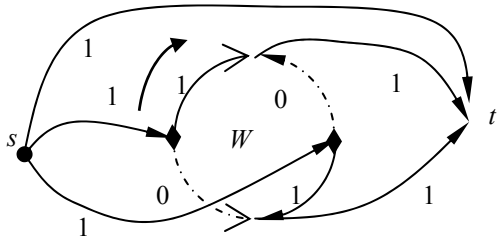
Because $c(P^*) \le c(P^k)$,

$$\begin{aligned} \frac{c(P^k)}{c(P^*)} &\le \frac{c(P^*) - (1-\lambda)\cdot c(Q^k)}{\lambda \cdot c(P^*)} = \frac{1}{\lambda} - \frac{1-\lambda}{\lambda}\frac{c(Q^k)}{c(P^*)} \\ &\le \frac{1}{\lambda} - \frac{1-\lambda}{\lambda}\frac{c(Q^k)}{c(P^k)} = 1 + \frac{1-\lambda}{\lambda}\left(1 - \frac{c(Q^k)}{c(P^k)}\right) \end{aligned}$$

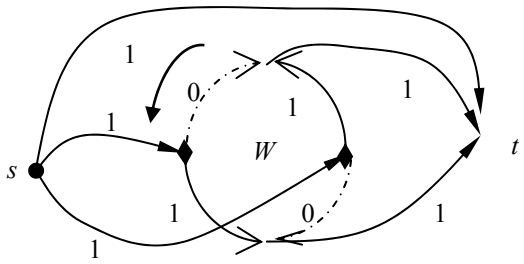
Similarly, we can prove that $\dfrac{d(Q^k)}{\Delta} \le 1 + \dfrac{\lambda}{1-\lambda}\left(1 - \dfrac{d(P^k)}{\Delta}\right)$. ■



a) The optimal basic solution.



b) Push flow along the direction of the arrow to obtain P^k. W_P consists of the two solid links on W. $P^k - W_P$ includes the remaining solid links.



c) Push flow along the direction of the arrow to obtain Q^k. W_Q consists of the solid two links on W. $Q^k - W_Q$ includes the remaining solid links.

Figure 4.4: Illustrations for the proof of Theorem 4.4

All the individual paths in the above theorem can be obtained using flow decomposition [1, 4].

4.6. Revised Simplex Method for the RELAX-CSDP(k) Problem

In this section, we first briefly present the different steps in the revised simplex method for upper bounded linear programming problem. A detailed description of this method may be found in Chapter 8 of [11]. Note that in [11] the revised simplex method is presented for a maximization problem. We then derive formulas required to identify the entering and the leaving variables that are needed to generate a new basic solution from a given basic solution.

Consider the following linear programming problem.

Minimize $\boldsymbol{c}\,\boldsymbol{x}$

subject to $\boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}$.

For the RELAX-CSDP(k) problem $\boldsymbol{A}$ is an $n \times (m + 1)$ matrix with rank($\boldsymbol{A}$) = n, $\boldsymbol{x} = (x_1 \ldots, x_{m+1})^t$, $\boldsymbol{c} = (c_1, c_2 \ldots, c_{m+1})$, $\boldsymbol{b} = (b_1, b_2 \ldots, b_n)^t$. Each feasible basic solution $\boldsymbol{x}^*$ of this linear program is partitioned into two sets, one set consisting of the n basic variables and the other set consisting of the remaining $m + 1 - n$ non-basic variables. This partition induces a partition of $\boldsymbol{A}$ into $\boldsymbol{B}$ and $\boldsymbol{A}_N$, a partition of $\boldsymbol{x}$ into $\boldsymbol{x}_B$ and $\boldsymbol{x}_N$ and a partition of $\boldsymbol{c}$ into $\boldsymbol{c}_B$ and $\boldsymbol{c}_N$, corresponding to the set of basic variables and the set of non-basic variables, respectively. The matrix $\boldsymbol{B}$ is the basis matrix and is nonsingular. See Sections 4.4 and 4.5 for the form of the basis matrix and properties of basic solutions for the RELAX-TCSDP(k) problem.

Revised Simplex Method [11]

1. Solve the system $\boldsymbol{Y} \bullet \boldsymbol{B} = \boldsymbol{c}_B$, where $\boldsymbol{Y} = (y_1, y_2 \ldots y_n)$.

2. Choose an entering variable x_j. This may be any nonbasic variable x_j such that, with $\boldsymbol{a}$ standing for the corresponding column of $\boldsymbol{A}$, we have either $\boldsymbol{Y}\,\boldsymbol{a} > c_j$, $x^*_j < u_j$ or $\boldsymbol{Y}\,\boldsymbol{a} < c_j$, $x^*_j > l_j$. If there is no such variable then stop; the current solution $\boldsymbol{x}^*$ is optimal.
3. Solve the system $\boldsymbol{B} \bullet \boldsymbol{V} = \boldsymbol{a}$, where $\boldsymbol{V} = (v_1, v_2 \ldots, v_n)^t$.
4. Define $x_j(t) = x^*_j + t$ and $\boldsymbol{x_B}(t) = \boldsymbol{x^*_B} - t\,\boldsymbol{V}$ in case $\boldsymbol{Y}\,\boldsymbol{a} < c_j$ and $x_j(t) = x^*_j - t$, $\boldsymbol{x_B}(t) = \boldsymbol{x^*_B} + t\,\boldsymbol{V}$ in case $\boldsymbol{Y}\,\boldsymbol{a} > c_j$. If the constraints $l_j \le x_j(t) \le u_j$, $\boldsymbol{l_B} \le \boldsymbol{x_B}(t) \le \boldsymbol{u_B}$ are satisfied for all positive t then the problem is unbounded. Otherwise set t as the largest value allowed by these constraints. If the upper bound imposed on t by the constraints $\boldsymbol{l_B} \le \boldsymbol{x_B}(t) \le \boldsymbol{u_B}$ is stricter than the upper bound imposed by $l_j \le x_j(t) \le u_j$, then determine the leaving variable. This may be any basic variable x_i such that the upper bound imposed on t by $l_i \le x_i(t) \le u_i$ alone is as strict as the upper bound imposed by all the constraints $\boldsymbol{l_B} \le \boldsymbol{x_B}(t) \le \boldsymbol{u_B}$.
5. Replace x^*_j by $x_j(t)$ and $\boldsymbol{x^*_B}$ by $\boldsymbol{x_B}(t)$. If the value of the entering variable x_j has just switched from one of its bounds to the other, then proceed directly to step 2 of the next iteration. Otherwise, replace the leaving variable x_j by the entering variable x_j in the basis heading, and replace the leaving column of $\boldsymbol{B}$ by the entering column $\boldsymbol{a}$.

Steps 2-5 in the revised simplex method that generate a new basic solution from a given basic solution are called a pivot.

Notice that the revised simplex method for the upper bounded linear programming problem is more complex than the one given in Chapter 3. However, the systems of equations in Steps 1 and 3 can be solved using the formulas developed in Section 3.4.

4.7. Initialization and Pivot Rules

4.7.1. Initialization

We first compute k minimum delay link-disjoint s-t paths using Suurballe's algorithm [54]. There is no feasible solution if the total delay of these paths is greater than Δ. Assume that this is not the case. A tree T' (not necessarily a spanning tree) rooted at t can be constructed from these paths by removing links incident with s to break cycles. Note that in T' every path from a node in T' to t is a directed path. Such a tree is called a directed tree rooted at node t [55]. We next obtain a directed spanning tree rooted at t and having T' as a subtree. Then we proceed as follows.

First condense or coalesce all the nodes in T' into a single node P. Then for the resulting network determine a directed spanning tree rooted at P with all links orientated away from P. Such a tree exists because of our assumption that there is a directed path from node s to each node in the network and similarly there is a directed path from each node to node t. The links of the directed tree selected as above and the links in T' together constitute a directed spanning tree T of the network N.

Assigning flow of 1 to all the links in the disjoint paths and flow of 0 to all other links, we obtain a basic solution represented by T.

Definition 4.2 ([1, 4, 11]): Given a feasible basic solution subgraph G_b, we say that the link $(u, v) \in G_b$ is oriented toward (*resp*. away from) the root if any of the paths in G_b from the root to u (*resp*. v) passes through v (*resp*. u). A feasible basic solution G_b with corresponding flow vector $\boldsymbol{x}$ is said to be strongly feasible if every link (u, v) of G_b with $x_{uv} = 0$ (*resp*. $x_{uv} = 1$) is oriented away from (*resp*. toward) the root**.**

Note that the definition of strong feasibility given above is different from the one defined in Chapter 3 due to the upper bounds on the flow variables.

It can be easily verified that the initial spanning tree T selected as above is strongly feasible.

4.7.2. Pivot Rules and an Anti Cycling Strategy

For an efficient implementation of the revised simplex method, we want to avoid directed cycles in basic solutions. This can be achieved by the following pivot rule:

P1: Slack variable w assumes the highest priority in choosing the entering variable (Step 2 of the Revised Simplex Method).

Lemma 4.7*: The slack variable w is eligible to enter the basis *iff* $\gamma < 0$.

Lemma 4.8*: Suppose the Pivot rule P1 is followed. If a directed cycle W is created in G_b during a pivot, then in the next pivot the slack variable w will enter the basis and a link on W will leave the basis.

Proof: Since W is a directed cycle, $c(W) \neq 0$ and $\gamma = -c(W)/d(W) < 0$. It follows that in the pivot after the directed cycle is created, w will enter the basis and the new basis graph will be a spanning tree. ■

A basic solution in which one or more basic variables assume zero values is called degenerate [11]. Simplex pivots that do not alter the basic solution are called degenerate. Furthermore, a basic solution generated at one pivot and reappearing at another will lead to cycling (or infinite looping and non-convergence). Thus we need a strategy to avoid cycling.

There are several anticylcing strategies for general linear programming problems. Cunningham developed a strategy specifically designed for the network simplex method used for solving minimum cost flow problems. Since RELAX-TCSDP(k) has almost the same structure as the minimum cost flow problem except for the presence of one additional constraint imposed by the delay requirement, we examine if Cunningham's strategy can be adopted for RELAX-TCSDP(k). We show next that the transformation introduced on the CSDP(k) problem in Section 4.4 indeed makes Cunningham's strategy suitable for avoiding cycling in the case of RELAX-TCSDP(k).

Lemma 4.9: For any degenerate pivot, the out-arc is not on the cycle of the current G_b.

If the out-arc is not on the cycle in the current G_b, then the potentials can be updated easily as described next (Chapter 5.1.2 of [4]). Let $\boldsymbol{T}$ be the current G_b and $\boldsymbol{e} = (u, v)$ and $\boldsymbol{e'} = (u', v')$ be the out-arc and the in-arc, respectively. Let $\boldsymbol{T'} = \boldsymbol{T} - \boldsymbol{e} + \boldsymbol{e'}$ be the subgraph of the new basic variables. If $\boldsymbol{e}$ is not on the cycle in the current G_b , then the new potential vector $\boldsymbol{Y'}$ associated with $\boldsymbol{T'}$ can be obtained as follows [4, 11].

$$y'_u = \begin{cases} y_u + r_{u'v'} & \text{if } u \in T_{u'} \\ y_u & \text{if } u \in T_{v'} \end{cases}. \tag{4.18}$$

In (418), $r_{u'v'} = c(e_{u'v'}, \gamma) + y_{v'} - y_{u'}$ and $\boldsymbol{T_{u'}}$ **(***resp.* $\boldsymbol{T_{v'}}$**)** is the component of $\boldsymbol{T} - \boldsymbol{e}$ containing u' (*resp.* v').

The convergence part of the following theorem closely follows the proof of Theorem 19.1 in [11].

Theorem 4.5: If the subgraphs G_b's of feasible basic solutions generated by the simplex method are strongly feasible then the simplex method does not cycle and its computational time complexity is pseudo-polynomial.

Proof: First observe that in any sequence of degenerate pivots, the value of every variable, in particular, the value of the slack variable will remain the same. Also if the slack variable is a basic variable then its value is nonzero; otherwise its value is zero. So during a given sequence of degenerate pivots, the slack variable will remain basic or nonbasic during the entire sequence of degenerate pivots. So the leaving and entering variables can only be the links in the network.

Let G_b be a feasible basic solution subgraph and t be the root. We define two values for G_b.

$$C(G_b) = \sum_{(u,v)\in E} c_{uv}x_{uv} \text{ and } W(G_b) = \sum_{u\in V}(y_t - y_u).$$

Consider two consecutive basic solutions $G_b^{\,i}$ with $G_b^{\,i+1} = G_b^{\,i} + e - f$, where e and f are the in-arc and out-arc, respectively.

We first show that either $C(G_b^{\,i+1}) < C(G_b^{\,i})$ or $W(G_b^{\,i+1}) < W(G_b^{\,i})$.

Indeed if the pivot that generates $G_b^{\,i+1}$ from $G_b^{\,i}$ is nondegenerate, then $C(G_b^{\,i+1}) < C(G_b^{\,i})$. If it is degenerate, we have $C(G_b^{\,i+1}) = C(G_b^{\,i})$. In this case we need to show that $W(G_b^{\,i+1}) < W(G_b^{\,i})$.

Note that the in-arc $e = (u, v)$ still has flow equal to 0 or 1 in $G_b^{\,i+1}$. By Lemma 4.9, f is not a link on the cycle in $G_b^{\,i}$. So the value of γ does not change. Because $G_b^{\,i+1}$ is strongly feasible, in $G_b^{\,i+1}$, link e must be oriented toward the root node t if $x_e = 1$ and oriented away from t if $x_e = 0$, which implies that node t belongs to $G_b^{\,i}(v)$ (the

component of $G_b^i - f$ containing v) if $x_e = 1$ and node t belongs to $G_b^i(u)$ if $x_e = 0$. The potentials with respect to G_b^{i+1} can be calculated using equation (4.18).

Then we have $W(G_b^{i+1}) = W(G_b^i) - |G_b(u)| r_{uv} < W(G_b^i)$, where $r_{uv} = c(e_{uv}, \gamma) + y_v - y_u > 0$ if $x_e = 1$ or $W(G_b^{i+1}) = W(G_b^i) + |G_b(u)| r_{uv} < W(G_b^i)$, where $r_{uv} = c(e_{uv}, \gamma) + y_v - y_u < 0$ if $x_e = 0$.

If the simplex method cycles, then for some $i < j$, $G_b^i = G_b^j$. This implies that $G_b^i = G_b^{i+1} ... = G_b^j$. But then $W(G_b^i) > W(G_b^{i+1}) > \ldots > W(G_b^j) = W(G_b^i)$ contradicting that $W(G_b^i) = W(G_b^j)$.

Thus we have proved that the simplex method when applied on RELAX-TCSDP(k) does not cycle if all the basic feasible solutions are strongly feasible.

We next establish the pseudo-polynomial time complexity of this method. We have

$$W(G_b^i) - W(G_b^{i+1}) = |G_b(u)| |r_{uv}| \geq |r_{uv}|$$

and

$$|r_{uv}| = |c(e_{uv}, \gamma) + y_v - y_u| = |c_{uv} + \gamma d_{uv} + y_v - y_u| .$$

We proceed to show that

$$0 < |y_u - y_v - \gamma d_{uv} - c_{uv}| = |\gamma d(W') + c(W')| = \begin{cases} |c(W')|, & \gamma = 0; \\ |c(W')d(W) - d(W')c(W)| / |d(W)|, & \gamma \neq 0. \end{cases} \quad (4.19)$$

Since e_{uv} is an in-arc, $|y_u - y_v - \gamma d_{uv} - c_{uv}| \neq 0$. To establish the equalities on the right hand side of (4.19) suppose that the new cycle W' in G_b is $e_1 e_2 \ldots e_k$ where $e_1 = e_{uv}$. Since all the links on W' except e_{uv} are in G_b, the reduced costs on all these links are 0. So we have $|r_{uv}| = |y_u - y_v - \gamma d_{uv} - c_{uv}| = |\gamma d(W') + c(W')|$. Recalling that $\gamma = -c(W)/d(W)$ if

there exists a cycle W in the basic solution or $\gamma = 0$ if no such cycle exists, we get the equalities on the right side of (4.19).

Since $|r_{uv}| \neq 0$, we get from (4.19) that $|r_{uv}| \geq |1 / d(W)| \geq 1 / (n\, d_{max})$, where d_{max} is the maximum link delay.

Also, the inequality below follows from the fact that the potential of a node is the sum of the active costs of the links on the path from that node to node t (See Section 3.2).

$W(G^i_b) = \sum_{u \in V} (y_t - y_u) \leq n^2(c_{max} + \gamma\, d_{max})$, where c_{max} is the maximum link cost.

If $\gamma \neq 0$, then by Lemma 4.7, $|\gamma| = |c(W) / d(W)| \leq n\, c_{max}$. Hence $W(G^{\,i}_b) \leq\ n^2(c_{max} + n\, c_{max}\, d_{max})$.

So the length of the sequence of degenerate pivots is bounded by a polynomial function of c_{max}, d_{max}, and n. Similarly, we can prove that the total number of non-degenerate pivots is also a polynomial function of m, n, c_{max}, and d_{max}. Pseudo polynomial complexity of the revised simplex method when applied on RELAX-TCSDP(k) follows since each pivot takes $O(m)$ steps [11]. ■

4.7.3. Leaving Variable

Now, we investigate how to find a leaving variable (out-arc) using Theorem 3.2. As before, let the cycle created by adding the in-arc be denoted by W' with its orientation defined as that of the in-arc.

We note that the reduced cost of the in-arc may be positive or negative. In the following we consider only the latter case. The former case can be treated in a similar way.

Case 1: Slack variable w is in the basic solution (the current G_b is a spanning tree and so $w > 0$). This corresponds to Theorem 3.2(a) of Section 3.5.4. According to Step 4 of the revised simplex method, we need to consider only the entries of V that are ± 1 or $d(W')$ if $d(W') \neq 0$. These entries correspond to the links of W' of the current G_b or the slack variable w. The maximum value of t is constrained by $\boldsymbol{x}^*_{\boldsymbol{B}} - t\,\boldsymbol{V} \geq 0$, and the corresponding constraining variables (links or w) are eligible to leave the basis. If certain links are eligible to leave the basis then we select the one which keeps the new basic solution strongly feasible (to be discussed next). In this case w will continue to be in the basis. If w is eligible to leave the basis, we select it to leave the basis. In that case the new basic subgraph G'_b will have a cycle. The flow values (λ or $1 - \lambda$) on the links in the cycle can be determined by the equation $d_{\boldsymbol{x}}(G'_b) = \Delta$ because the slack variable w is nonbasic and has zero value.

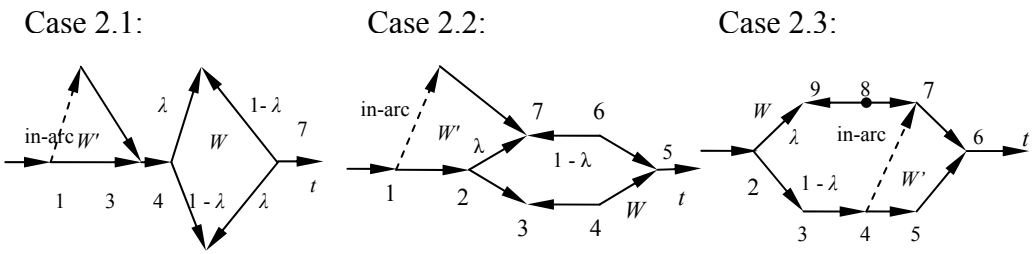


Figure 4.5: Basic solution structures in Case 2

Case 2**:** The basic solution consists of n links, i.e., there is a cycle W in G_b. The slack variable w is eligible to enter the basis if $\gamma < 0$. Then according to pivot rule P1, we let w enter the basis and shall select one of the links on W to leave the basis. The choice can be made according to the case (*c*) in Theorem 3.2.

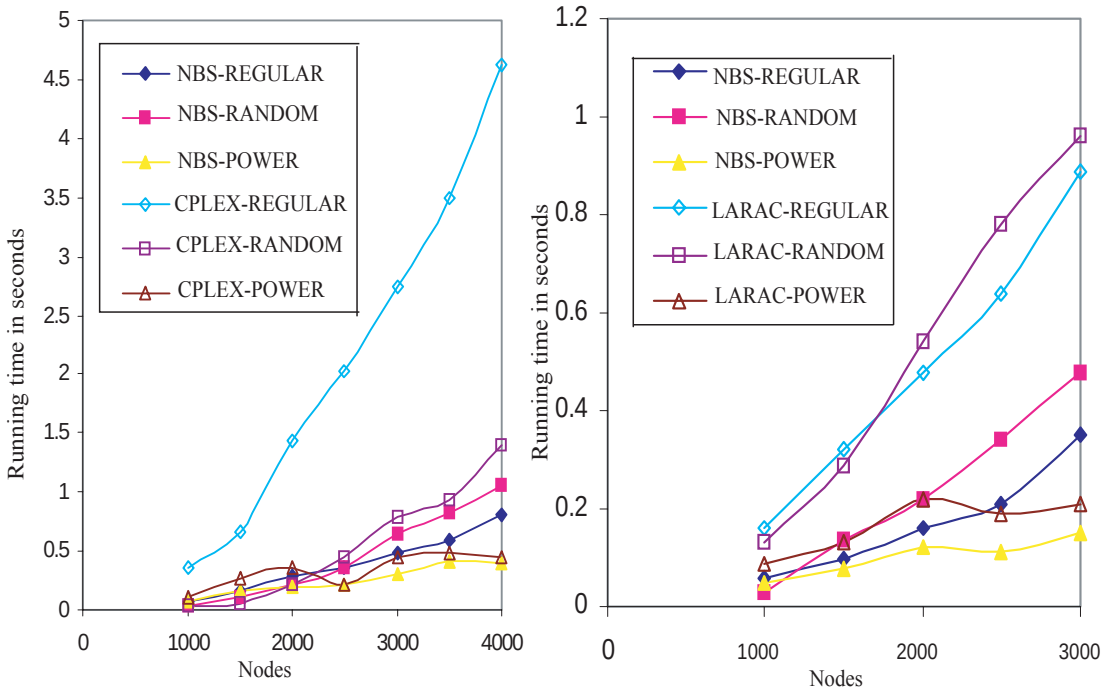
If $\gamma \geq 0$, an entering link will create a new cycle W' when added to the current G_b. We need to consider different subcases that capture all possibilities (See Figure 4.5). For each one of these subcases we can select the leaving variable using Theorem 3.2(*b*) and Step 4 in the Revised Simplex Method.

Now, we need to consider how to preserve the strong feasibility of the basic solutions. We define the join of a cycle in G_b as the node on the cycle that is closest to the node t in terms of hops. Without loss of generality, assume that the current basic solution is strongly feasible and consists of n links and that the leaving variable is a link f (other cases are trivial). Let $G_e = G_b + e$ be the network obtained by adding the in-arc e to G_b. Evidently, f is on some cycle $C \in \{W, W'\}$ in G_e. If $C = W'$, the orientation of C is chosen to agree (*resp.* disagree) with the orientation of e if $x_e = 0$ (*resp.* $x_e = 1$) in the current flow. If $C = W$, the orientation of C is defined such that $d(W') / d(W) < 0$ (*resp.* $d(W') / d(W) > 0$) if $x_e = 0$ (*resp.* $x_e = 1$), where the orientation of W' agrees with the direction of e. Starting from the join of C and traversing along the orientation of C, we choose the first link whose flow is 1 and whose direction agrees with the orientation of C or whose flow is 0 and whose direction disagrees with the orientation of C. This guarantees the strong feasibility of the resulting tree.

4.8. Simulation

We denote our algorithm as DISJOINT-NBS and compare its performance with CPLEX and G-LARAC(k). We use three classes of network topologies: regular graphs $H_{k,n}$ (see Chapter 8, [55]), power-law out-degree graphs [44] and Waxman's random graphs [59]. For a network $G(V, E)$, the nodes are labeled as 1, 2…, $n = |V|$. Nodes $\lfloor n / 2 \rfloor$ and n are chosen as the source and target nodes. The link costs and delays are randomly independently generated even integers in the range from 1 to 200. The delay bound is $1.2 \times k$ the delay of the minimum delay s-t paths in G. For regular graphs, $k = 4$. For random graphs and power-law graphs, $k = 2$. The results are shown in Figure 4.6, where the y-axis denotes the running time in seconds. In these figures, we use NBS to denote the DISJOINT-NBS algorithm and NBS-REGULAR to denote the running time of DISJOINT-NBS algorithm on regular graphs. Other labels can be interpreted in a similar manner. Experiments show that DISJOINT-NBS algorithm is faster than CPLEX and G-LARAC(k) on all the topologies. For the power-law out-degree graph and Waxman's random graph, the hop number of feasible s-t paths is usually very small even when the network is very large. So the running times of DISJOINT-NBS, G-LARAC(k), and CPLEX are close (but DISJOINT-NBS is still faster) for random graphs and power-law out-degree graphs.

Our simulation results in Tables 4.1-4.3 show that the delay of each path derived as in Theorem 4.4 deviates from the individual delay bound by a small fraction. Note that in these tables the second column specifies the delay bound on each path.



The experiments comparing CPLEX and NBS algorithm were carried on Intel Pentium 4 with Linux OS and the experiments comparing NBS and LARAC algorithm were carried out on IBM Regatta p690 with AIX 5.1 OS and Power4 1.1 GHz CPU.

Figure 4.6: Comparison of DISJOINT-NBS, CPLEX, and G-LARAC(k)

Table 4.1. Paths obtained from the optimal solution to RELAX-TCSDP(2) applied on random graphs

Size(#Nodes)	Delay Bound	Path-1(Cost, Delay)	Path-2(Cost, Delay)
1000	1087	(1240, 1056)	(1536, 1082)
2000	601	(1548, 560)	(1344, 604)
3000	409	(1496, 368)	(1328, 428)

Table 4.2. Paths obtained from the optimal solution to RELAX-TCSDP(2) applied on power-law graphs

Graph Size(#Nodes)	Delay Bound	Path-1(Cost, Delay)	Path-2(Cost, Delay)
1000	109	(426, 110)	(372, 72)
2000	134	(352, 82)	(190, 172)
3000	206	(380, 206)	(254, 138)

Table 4.3. Paths obtained from the optimal solution to RELAX-TCSDP(4) applied on regular graphs

Graph Size	Bound	Path-1	Path-2	Path-3	Path-4
1000	736	(2208, 686)	(2216, 686)	(2054, 764)	(1872, 782)
2000	1425	(3920,1412)	(4168, 1424)	(4014, 1454)	(4198, 1406)
3000	2127	(6092, 2044)	(6126, 2104)	(5862, 2242)	(5702, 2110)

4.9. Summary

In this chapter we studied the CSDP(k) problem which is NP-hard. So our goal has been to design an efficient algorithm for constructing an approximate solution to this problem. Towards this end, we studied the LP relaxation of CSDP(k) problem using the revised simplex method of linear programming. This relaxed problem is an upper bounded LP problem. We have discussed several issues relating to an efficient implementation of our approach. We have shown that an approximate solution to the CSDP(k) problem can be extracted from an optimal solution to the relaxed problem. We have derived bounds on the quality of this solution with respect to the optimal solution. Our work can be considered as the study of the CSDP(k) problem from a primal perspective in contrast to the dual perspective employed in the G-LARAC(k) algorithm which is based on the algorithms in [23] and [54]. Simulation results demonstrate that our algorithm is slightly faster than both the G-LARAC(k) algorithm and the commercial quality CPLEX package in the case of random graphs and power-law out-degree graphs. On the other hand, for regular graphs our algorithm is much faster.

The GCSDP(k) problem defined in Section 4.2 requires that the delay of each individual path satisfies a specified bound, in contrast to the CSDP(k) problem where

the constraint is on the total delay of all the k link-disjoint paths. We have shown in Theorem 4.2 that the LP relaxations of the two problems have the same optimal objective value. Thus, if one is interested in obtaining the optimal objective values of RELAX-GCSDP(k) and RELAX-CSDP(k) problems, then starting with the RELAX-CSDP(k) does not result in any loss of generality. However, the paths produced by the approximate solution derived from the optimal solution to RELAX-CSDP(k) may not satisfy the individual path delay requirements of the GCSDP(k) problem. Fortunately, our simulation results in Table 4.1-4.3 indicate that in most cases the individual delays of the paths produced starting from RELAX-CSDP(k) do not deviate in a significant way from the individual delay requirements of the GCSDP(k) problem.

If one were interested in studying the GCSDP(k) problem then the issue of finding feasible solutions to this problem will arise. This problem itself is *NP*-hard and no efficient algorithms are known. However techniques such as branch and bound and Tabu search can be used to find the feasible solutions. The algorithm in the present chapter can be used as a subroutine in a branch and bound scheme.

One direction of further research is to develop approximation schemes for the CSDP(k) problem along the lines of the approximation algorithms given in [43] for the CSDP(2) problem. Since the link-disjoint shortest paths problem is a special case of the minimum cost flow problem, it will be interesting to investigate if the ideas developed in this chapter could be used to design efficient algorithms for the constrained minimum cost flow problem.

Chapter 5. Delay Constrained Path Selection under Inaccurate State Information

In this chapter we study the MP-DCP problem that requires determination of the most probable delay constrained path (MP-DCP problem). Our work is based on the formulation given in [28]. The work in [28] focused on developing approximate approaches using the Lagrangian relaxation or line search techniques. In contrast, our focus has been on developing polynomial time ε-approximation and heuristic algorithms.

The rest of this chapter is organized as follows. In Section 5.1, we give the definition and formulation of the MP-DCP problem. In Section 5.2 we present an exact algorithm, a FPTAS, and a strongly polynomial approximation algorithm for Case 1. In Section 5.3 we study the MCP-DCP problem for Case 2.

The results in this chapter have been presented in [64].

5.1. The MP-DCP Problem and a Formulation

Consider a network represented by a graph $G(V, E)$, with $n = |V|$, and $m = |E|$. Given a maximum delay requirement D for a flow between a given source node s and a destination node $t \neq s$, and probability density function (*pdf*) $p_l(d)$ for all $l = (i, j) \in E$, where $p_l(d)$ is the probability that the link l will introduce a delay of at most d units, i.e., $d_l < d$. Let $d(i, j)$ be the random variable (*RV*) associated with the delay of the link (i, j). For a path p, define

$$\pi_D(p) = Pr[d(p) \leq D].$$

The **MP-DCP problem** is to find an *s-t* path p_{opt} such that $\forall p \in P_{st}$, $\pi_D(p_{opt}) \geq \pi_D(p)$.

To simplify this problem and following Korkmaz and Krunz [28], we assume that $d(i, j)$'s are nonnegative *RV's* with mean $\mu(i, j) > 0$ and variance $\delta^2(i, j) > 0$, and that for all links $(i, j) \in E$, $d(i, j)$'s are mutually independent. Without loss of generality, assume $\mu(i, j)$'s and $\delta^2(i, j)$'s are integers (this is because all numbers are represented by finite digits in computers and other digital devices). Furthermore, we assume that the *pdf* of $d(i, j)$ is continuous and differentiable on some interval (a, b). Under this assumption and according to the central limit theorem, the path delay is approximately normally distributed. Without loss of generality, we assume each *s-t* path is long enough that $d(p)$ is a normally distributed *RV* with mean $\mu(p) > 0$ and variance $\delta^2(p) > 0$. (Note: The sum of as small as three *RV*s tends to a normal distribution [28]). $\mu(p) > 0$ and variance $\delta^2(p) > 0$ are given by:

$$\mu(p) = \sum_{(i,j) \in p} \mu(i, j) \text{ and } \delta^2(p) = \sum_{(i,j) \in p} \delta^2(i, j).$$

With the above assumption,

$$\pi_D(p) \approx \Phi(\frac{D - \mu(p)}{\delta(p)}), \text{ where } \Phi(x) = (1/2\pi)^{1/2} \int_{-\infty}^{x} e^{-y^2/2} dy$$

Since $\Phi(x)$ is an increasing function, we can reduce the MP-DCP problem to one of identifying the path p that maximizes

$$\chi_D(p) = (D - \mu(p)) / \delta(p), \text{ where } \delta(p) \equiv (\sum_{(i,j) \in p} \delta^2(i, j))^{1/2}.$$

We call $\mu(p)$ and $\delta^2(p)$ as the mean delay and delay variance of path p. The difficulty with this problem arises from the nonseperable square root ($\sqrt{}$) function. As in [28], we distinguish two cases:

Case 1: There exists a path with mean delay less than or equal to the specified delay

bound D.

Case 2: The mean delay of each path is greater than D.

It can be easily verified that in Case 1(*resp.* Case 2), $\chi_D(p_{opt}) \geq 0$ and $\mu(p_{opt}) \leq D$ (*resp.* $\chi_D(p_{opt}) < 0$, $\mu(p_{opt}) > D$).

5.2. Algorithms for CASE 1

5.2.1. An Exact Algorithm

In the next three subsections, we first assume that there always exists some path p such that $\mu(p) \leq D$. In fact, if this assumption does not hold, the value of D may not be a realistic delay bound because in this case, $\forall p \in P_{st}$, $\pi_D(p) < \Phi(0) = 0.5$, i.e., any path p meets the delay bound with probability less than 0.5.

Since $\delta(p)$ is nonadditive, *Procedure exact-mp-dcp* in Figure 5.1 enumerates all the possible values of $\delta^2(p)$ that lie in [1, U] with $U = \min\{\delta^2(p^*) \mid p^* \in P\}$, where $P = \{p^* \mid \mu(p^*) = \min\{\mu(p) \mid p \text{ is an } s\text{-}t \text{ path}\}\}$.

For the sake of completeness we also present next the main algorithm for computing argmin $\{\mu(p) \mid \delta^2(p) \leq T_i\}$ in Procedure *exact-mp-dcp*. When we compute argmin $\{\mu(p) \mid \delta^2(p) \leq T_i\}$ for the first time, we call the Algorithm *CSP* adopted from the exact algorithm (See Figure 5.2) of [18] for the constrained shortest path problem with two metrics on deterministic networks. This is a dynamic programming algorithm with time complexity O(m T), where T is upper bound of the delay. We only need to call the Algorithm *CSP* once and then we can directly use the table $f_l(d)$ (defined in the Algorithm *CSP* below) created by its first invocation. For a given T_i and p^* = arg

$\min\{\mu(p) \mid \delta^2(p) \le T_i\}$, it can be seen that $\mu(p^*) = f_i(T_i)$ and $\delta^2(p^*) = T^*$, where T^* is the least value such that $f_i(T^*) = f_i(T_i)$. In fact, the value of $\delta^2(p^*)$ can be computed in constant time using extra data structures.

```
Procedure exact-mp-dcp
1    T_0 ← U
2    i ← 0
3    χ_opt ← - ∞
4    while (T_i > 0)
       /*The following is computed using
         CSP algorithm to be discussed*/
5      p* ← argmin{μ(p) | δ^2(p) ≤ T_i}
6      χ_opt ← max{χ_opt, (D – μ(p*)) / δ(p*)}
7      i ← i + 1
8      T_i ← δ^2(p*) – 1
9    end while
10   return χ_opt and the corresponding path p*
end procedure
```

Figure 5.1: The exact algorithm for Case 1

We keep the formulation of *exact-mp-dcp* as above to make the algorithm conceptually simple.

Let $f_j(d)$ be the minimum mean delay among all 1-j paths with delay variance $\le d$.

/* T is the delay variance upper bound*/

***Algorithm CSP* (*T*):**

/*1 is the source node, n is the target node*/

$f_1(d) = 0,\ d = 0, \ldots, T,$

$f_j(0) = \infty,\ j = 2, \ldots, n,$

$$f_j(d) = \min\left\{f_j(d-1),\ \min_{k \mid \delta^2(k,j) \le d}\{f_k(d - \delta^2(k,j)) + \mu(k,j)\}\right\}$$

$$, j = 2\ldots, n,\ d = 1\ldots, T$$

Figure 5.2: CSP algorithm

Theorem 5.1. Procedure *exact-mp-dcp* finds an optimal solution in $O(U\ m)$ steps if

for all links (i, j), $\delta^2(i, j) \neq 0$.

***Proof*:** The computation of $\arg\min\{\mu(p) \mid \delta^2(p) \leq T\}$ for the first time is done by the constrained shortest path algorithm which takes $O(T\ m)$ steps if there are no 0 delay variance links.

Obviously, the computation time of Algorithm *CSP* dominates all the other computations and so the complexity of the whole algorithm is $O(U\ m)$.

Let p_{opt} be one of the optimal solutions with the least delay variance and p^* be the path such that $U = \delta^2(p^*)$. We first show that $U = \delta^2(p^*) \geq \delta(p_{opt})$. By the definition of p_{opt} and p^*,

$$(D - \mu(p_{opt})) / \delta(p_{opt}) \geq (D - \mu(p^*)) / \delta(p^*) \text{ and } D \geq \mu(p_{opt}) \geq \mu(p^*).$$

If $D - \mu(p_{opt}) > 0$, we have

$$\delta(p^*) / \delta(p_{opt}) \geq (D - \mu(p^*)) / (D - \mu(p_{opt})) \geq 1 \text{ or } \delta(p^*) \geq \delta(p_{opt}).$$

If $D - \mu(p_{opt}) = 0$, then $\chi_D(p^*) = \chi_D(p_{opt}) = 0$ and $D = \mu(p_{opt}) = \mu(p^*)$.

Hence $\delta(p^*) \geq \delta(p_{opt})$ because p_{opt} is the optimal path with the least delay variance and p^* is one of the optimal paths.

We next show that Procedure *exact-mp-dcp* will find one of the optimal solutions at termination. Suppose in iteration i, $p_i = \arg\min\{\mu(p) | \delta^2(p) \leq T_i\}$. To prove the correctness of the algorithm, it suffices to show that if $\delta^2(p_{opt}) \leq T_i$ and p_i is not optimal, then $\delta^2(p_{opt}) < \delta^2(p_i) = T_{i+1} + 1$. Then we can see that the algorithm has enumerated all possible values of $\delta^2(p_{opt})$ at termination.

If this were not true, then $T_i \geq \delta^2(p_{opt}) \geq \delta^2(p_i)$. Obviously, $D \geq \mu(p_{opt}) \geq \mu(p_i)$ by the definition of p_i and $\delta^2(p_{opt}) \leq T_i$. We obtain

$$(D - \mu(p_{opt})) / \delta(p_{opt}) \leq (D - \mu(p_i)) / \delta(p_i).$$

Since p_i is not optimal by the assumption, this is the desired contradiction. ■

In the next section, based on Procedure *exact-mp-dcp*, we shall design a fully polynomial time approximation algorithm.

5.2.2. A Fully Polynomial Time Approximation Scheme: Case 1

To design a fully polynomial time approximation algorithm, we use scaling and rounding described in [35]. Without loss of generality, assume $U >> n$ and $\varepsilon < 1$.

Lemma 5.1: Let $G(N, E, \mu, \delta, D)$ denote a network with two metrics μ and δ on the link set E. Let $G_\tau(N, E, \mu, \delta', \tau)$ be the network transformed from G such that

$$\forall (i,j) \in E,\ \delta'^{\,2}(i,j) = \lfloor \tau\, \delta^2(i,j) / upper \rfloor + 1,$$

where τ (= O(n)) is some integer (to be discussed later) and $lower < upper \le U$.

Proof: We have

$\delta^2(i,j)(\tau / upper) \le \delta'^{\,2}(i,j) \le \delta^2(i,j)(\tau / upper) + 1$. So

$$\frac{D-\mu(p_\tau)}{\delta(p_\tau)} \ge \frac{D-\mu(p_\tau)}{(upper/\tau)^{1/2}\delta'(p_\tau)} \ge \frac{D-\mu(p_{opt})}{(upper/\tau)^{1/2}\delta'(p_{opt})}$$

$$\ge \left(1+\frac{upper}{\delta^2(p_{opt})}\frac{L(p_{opt})}{\tau}\right)^{-1/2}\frac{D-\mu(p_{opt})}{\delta(p_{opt})} \ge \left(1+\frac{upper}{lower}\frac{L(p_{opt})}{\tau}\right)^{-1/2}\frac{D-\mu(p_{opt})}{\delta(p_{opt})}.$$ ■

Let p_{opt} be the optimal solution to the MP-DCP problem on G, and p_τ be any path such that

$$(D-\mu(p_\tau)) / \delta'(p_\tau) \ge (D-\mu(p_{opt})) / \delta'(p_{opt}).$$

If $lower \le \delta^2(p_{opt}) \le upper$, then

$$\frac{D-\mu(p_\tau)}{\delta(p_\tau)} \geq (1+\frac{upper}{lower}\cdot\frac{L(p_{opt})}{\tau})^{-1/2}\frac{D-\mu(p_{opt})}{\delta(p_{opt})},$$

where $L(p)$ is the number of links of path p. ■

We next present an approximation algorithm Procedure *approx-mp-dcp* for the MP-DCP problem. In each iteration of Procedure *approx-mp-dcp*, the algorithm computes a path whose objective is no less than the optimum values among all the paths whose delay variance lies between given values of *lower* and *upper*. This is achieved by calling *approx-max-mp-dcp*(*lower*, *upper*, τ, χ_{opt}) which applies Procedure *exact-mp-dcp* on an appropriately scaled network.

Procedure approx-mp-dcp
1 $\tau \leftarrow 2n/\varepsilon$, $upper \leftarrow U$, $lower \leftarrow U/2$
2 $\chi_{opt} \leftarrow -\infty$
3 *while* $lower \geq 1$
4 $p^* \leftarrow$ *approx-max-mp-dcp*($lower$, $upper$, τ, χ_{opt})
5 $\chi_{opt} \leftarrow \max\{\chi_{opt}, (D-\mu(p^*))/\delta(p^*)\}$
6 $upper \leftarrow lower - 1$
7 $lower \leftarrow upper/2$
8 *end while*
9 *return* χ_{opt} and the corresponding path
end procedure

Procedure approx-max-mp-dcp(lower, upper, τ, χ_{opt})
1 $\delta'^2(i,j) = \lfloor \tau\,\delta^2(i,j)/upper \rfloor + 1$ for all link (i,j)
2 $L = (lower/upper)\,\tau$, $\Delta \leftarrow \tau + n$
3 *while* ($\Delta \geq L$)
/* **Using CSP Algorithm on δ'** */
4 $p_\Delta \leftarrow \text{argmin}\{\mu(p) \mid \delta'^2(p) \leq \Delta\}$
/* **Using $\delta(p_\Delta)$ not $\delta'(p_\Delta)$** */
5 $\chi_{opt} \leftarrow max\{\chi_{opt}, (D-\mu(p_\Delta))/\delta(p_\Delta)\}$
6 $\Delta \leftarrow \delta'^2(p_\Delta) - 1$
7 *end while*
8 return χ_{opt} and the corresponding path
end procedure

Figure 5.3: FPTAS for Case 1

Lemma 5.2: If p_{opt} is the optimal solution to the MP-DCP problem and $lower \leq \delta^2(p_{opt}) \leq upper$, where $lower = upper / 2$, $\tau = 2\ n / \varepsilon$, *approx-max-mp-dcp* finds a solution p_Δ such that $\chi_D(p_\Delta) \geq \chi_D(p_{opt})/(1 + \varepsilon)^{1/2}$ in time $O(m\ n / \varepsilon\ ))$.

***Proof*:** The complexity is easy to show (See Theorem 5.1). ■

We next prove the first part of this lemma.

Observe that for any path p in G with $lower \leq \delta^2(p) \leq upper$, $\tau / 2 \leq \delta'^2(p) \leq \tau + n$, where δ' is the metric defined in the line 1 of algorithm *approx-max-mp-dcp*.

Let $p_\tau = \arg\max\{(D - \mu(p)) / \delta'(p) \mid \tau / 2 \leq \delta'^2(p) \leq \tau + n\}$. We have

$$(D - \mu(p_\Delta)) / \delta(p_\Delta) \geq (D - \mu(p_\tau)) / \delta(p_\tau) \text{ and,}$$

$$(D - \mu(p_\tau)) / \delta'(p_\tau) \geq (D - \mu(p_{opt})) / \delta'(p_{opt})$$

This first inequality holds because p_τ is among all the paths delivered by the CSP algorithm invoked in line 4 of *approx-max-mp-dcp* and at termination, p_τ must have been compared with p_Δ (p_Δ is the winner at termination) in updating χ_{opt} (line 5 in *Procedure approx-max-mp-dcp*).

By the first inequality and Lemma 5.1, we obtain

$$\begin{aligned}\chi_D(p_\Delta) &= (D - \mu(p_\Delta))/\ \delta(p_\Delta) \geq (D - \mu(p_\tau))/\ \delta(p_\tau) \\ &\geq (1 + L(p_{opt})\ upper/(\tau\ lower))^{-1/2}\ (D - \mu(p_{opt}))/\delta(p_{opt}) \\ &\geq (1 + \varepsilon)^{-1/2}(D - \mu(p_{opt}))/\delta(p_{opt}) = \chi_D(p_{opt})/(1 + \varepsilon)^{1/2}.\end{aligned}$$ ■

Theorem 5.2: Procedure *approx-mp-dcp* finds in time $O((m\ n / \varepsilon) \log U)$ a path p^*

such that $\chi_D(p^*) \geq \chi_D(p_{opt})\ (1+\varepsilon)^{-1/2}$, where p_{opt} is the optimal path for the MP-DCP problem.

Proof: Obviously, the procedure terminates in O(log U) iterations of *Procedure approx-max-mp-dcp*. Next, we can see that *approx-mp-dcp* must have searched the interval containing the optimal path before termination as shown in Lemma 5.2 and thus the theorem is proven. ■

Our algorithm is an FPTAS. An interesting question is whether we can adopt the techniques in [18] and [35] to derive a strongly polynomial algorithm (the time complexity does not depend on U). Unfortunately, (due to the nonseparable nature of objective function), optimality conditions for the MP-DCP problem are not known. So, we are not able to design the *test* or *ε-test* procedures which are critical for the methods in [18, 35].

5.2.3. A Strongly Polynomial Approximation Algorithm: Case 1

In this section, using parametric search we design a strongly polynomial approximation algorithm for the MP-DCP problem.

We notice that the objective function of the MP-DCP problem is close to the form of fractional optimization problems that can be solved by Newton method [47] or parametric search [38]. For the MP-DCP problem, the only difficulty is the nonadditive nature of $\delta(p)$. In order to remove this barrier, we change the objective function and consider the following modified problem.

H-MP-DCP

Maximize $H\chi_D(p) = (D - \mu(p)) / \delta^2(p)$, where $\delta(p) \equiv (\sum_{(i,j) \in p} \delta^2(i,j))^{1/2}$.

Let p_H be the optimal path to the H-MP-DCP problem. Assume $H\chi_D(p_H) = OPT$. In parametric search, for any given λ, we need an oracle test to determine whether OPT is greater or less than λ. Even though the value of OPT is unknown, this can still be achieved by applying Dijkstra's shorts path algorithm on the weights $\mu(i, j) + \lambda\ \delta^2(i, j)$ for all links $(i, j) \in E$. Let p_λ denote the shortest path with respect to $W_\lambda(i, j) = \mu(i, j) + \lambda\ \delta^2(i, j)$. For the sake of brevity, we present our algorithm *h-mp-dcp*(G, s, t) using Bellman-Ford-Moore shortest path algorithm instead of Dijkstra's shortest path algorithm. For node u, define $N(u) = \{v \mid (u, v) \in E\}$. Each node v of the network is associated with a pair $M_v = (x_v, y_v)$, where x_v and y_v keep track of the mean delay and delay variance of some s-v path during the execution of the *h-mp-dcp* algorithm. M is initialized as $M_s = (0, 0)$ and $M_v = (\infty, \infty)$ for $v \neq s$. The algorithm computes the path p_H without knowing OPT. By the assumption that there always exists a path such that $\mu(p) \leq D$, it can be seen that $OPT \geq 0$.

Algorithm *h-mp-dcp*(G, s, t)

```
1   M_v = (x_v, y_v) = (∞, ∞) for all nodes
2   M_s = (0, 0)
3   for i ← 1 to n – 1 do
4     for each node u in the network
5       for each v such that v ∈ N(u)
          /*oracle test*/
6         [ if (x_v + OPT y_v
                ≥ x_u + OPT y_u + μ(u,v) + OPT δ²(u,v) ) ]
                  M_v ← (x_u + μ(u, v), y_u + δ²(u, v))
```

Figure 5.4: The approximation algorithm

In *h-mp-dcp*, extra steps are required to implement the following oracle test with

unknown *OPT*.

$$x_v + OPT\, y_v \geq x_u + OPT\, y_u + \mu(u, v) + OPT\, \delta^2(u, v).$$

If $x_v = \infty$, $y_v = \infty$, then the inequality holds. Assume x_v and y_v are finite (non-negative) values. Then it suffices to evaluate the following Boolean expression.

$$x_u + \mu(u, v) - x_v) + OPT\, (y_u + \delta^2\, (u, v) - y_v) = p + q\, OPT \leq 0,$$

where $p = x_u + \mu(u, v) - x_v$ and $q = (y_u + \delta^2(u, v) - y_v)$.

We then only need to determine the sign of $p + q\, OPT$ (> 0, < 0, and $= 0$). If $p \cdot q \geq 0$, the sign of $p + q\, OPT$ is the same as that of p or q recalling that $OPT \geq 0$. In this case implementing the oracle test is obvious.

Consider $p \cdot q < 0$, i.e., $-p / q > 0$. Let $\lambda = -p / q$ and let $p_\lambda = Dijkstra(s, t, W_\lambda)$, where $Dijkstra(s, t, W_\lambda)$ computes the minimal *s*-*t* path with respect to W_λ. Now three cases arise.

1. $\mu(p_\lambda) + \lambda\, \delta^2(p_\lambda) < D$: This implies that $\lambda < (D - \mu(p_\lambda)) / \delta^2(p_\lambda) \leq OPT$.

2. $\mu(p_\lambda) + \lambda\, \delta^2(p_\lambda) = D$: This implies that $\mu(p_\lambda) + \lambda\, \delta^2(p_\lambda) = D \leq \mu(p_H) + \lambda\, \delta^2(p_H)$. Thus $(D - \mu(p_\lambda)) / \delta^2(p_\lambda) = \lambda \geq (D - \mu(p_H)) / \delta^2(p_H) = OPT$ which implies $\lambda = OPT$.

3. $\mu(p_\lambda) + \lambda\, \delta^2(p_\lambda) > D$: Then $\mu(p_H) + \lambda\, \delta^2(p_H) > D$ and $\lambda > (D - \mu(p_H)) / \delta^2(p_H) = OPT$.

With the path p_λ, we can easily decide the sign of $p + q\, OPT$ by the above three cases.

Theorem 5.3: (*a*). The time complexity of algorithm *h-mp-dcp* is $O((m + n \log n)^2)$.

(*b*) Let p_{opt} be the optimal solution to the MP-DCP problem (the original problem). Then

(*i*) $\chi_D(p_H) \geq (\delta(p_{opt}) / \delta(p_H))^{-1/2}\, \chi_D(p_{opt})$, and

(*ii*) $\mu(p_{opt}) \leq \mu(p_H)$, $\delta(p_{opt}) \geq \delta(p_H)$.

If p_H does not meet the requirements of the applications, we may need to close or

reduce the gap between the approximate solution and the optimal solution by applying the approximation algorithm with proper approximation factor or the exact algorithm if necessary. On the other hand, the solution obtained by the heuristic algorithm can be used to reduce the computational time of the approximation and exact algorithms. According to (*b*) in Theorem 5.3 we know that $\delta^2(p_H) \leq \delta^2(p_{opt}) \leq U$. So the Procedure *approx-mp-dcp* (*resp.* Procedure *exact-mp-dcp*) can terminate safely once *upper* < $\delta^2(p_H)$ (*resp.* $T_i < \delta^2(p_H)$). Note that using *h-mp-dcp* as an initial pruning step does not affect the polynomial time complexity of these algorithms. The number of invocations of Dijkstra's shortest path algorithm in the parametric search can also be greatly reduced using techniques described in [47].

Table 5.1. Numeric simulation results on two classes of graph topologies

$\|V\|$	OPT	Φ(OPT)	H-OPT	Φ(H-OPT)	Error (%)
1000	0.835	**0.7981**	0.826	**0.7956**	**0.313**
1500	1.043	**0.8515**	1.036	**0.8499**	**0.188**
2000	1.209	**0.8867**	1.196	**0.8842**	**0.282**
2500	1.341	**0.9100**	1.327	**0.9077**	**0.253**
3000	1.456	**0.9273**	1.437	**0.9246**	**0.291**

(*a*) Regular graph (out degree = 6)

$\|V\|$	OPT	Φ(OPT)	H-OPT	Φ(H-OPT)	Error (%)
1000	0.643	**0.7399**	0.628	**0.7350**	**0.662**
1500	0.526	**0.7006**	0.515	**0.6967**	**0.557**
2000	0.505	**0.6932**	0.492	**0.6886**	**0.664**
2500	0.418	**0.6620**	0.413	**0.6602**	**0.274**
3000	0.459	**0.6769**	0.459	**0.6769**	**0.000**

(*b*) Waxman's random graph

$|V|$, OPT, Φ(OPT), H-OPT and Φ(H-OPT) denote the number of nodes of the network, the optimal $\chi_D(p)$, the optimal $\pi_D(p)$, the solution for $\chi_D(p)$ obtained by Algorithm *h-mp-dcp* and the corresponding $\pi_D(p)$. The Error column is computed as 100 (Φ (OPT) – Φ (H-OPT)) / Φ (OPT).

We present in Table 5.1 numerical simulation results for this heuristic. The experiments are carried out on two different classes of graphs: regular graph [55] and Waxman's random graph [59]. In these classes of graphs, for each link (i, j), $\mu(i, j)$ is randomly independently generated integers uniformly distributed in [1, 20] and $\delta^2(i, j)$ is randomly independently generated integers uniformly distributed in [1, 200]. The value of D is 115% of $\mu(p^*)$ where p^* is the s-t path with minimum mean delay. (Now, MP-DCP problem can be seen as defined on a deterministic network with two independent metrics: mean delay μ and delay variance δ^2). It can be seen that the optimal values and the approximate values of $\pi_D(p)$ are very close.

5.3. MP-DCP Problem: Case 2

In this section we consider the MP-DCP problem in the case when $\forall p, \mu(p) > D$.

Theorem 5.4: If $\forall p \in P(s, t), \mu(p) > D$, the MP-DCP problem is NP-hard.

Proof: Let us consider an instance of the longest path problem on graph $G(V, E)$. It is known that finding the longest simple path in terms of the number of links is NP-hard and it can also be seen that finding the longest simple path from a given node s to a node t is also NP-hard [26].

To prove the NP hardness of MP-DCP problem in Case 2, it suffices to show that the longest path problem is a subclass of the MP-DCP problem.

Define an MP-DCP problem instance on G with given bound $D = 1$ as follows:

Let $\delta^2(i, j) = 1$ for each link $(i, j) \in E$ (now $\delta^2(p)$ is equal to the number of hops of path p).

Let $M = n / ((1 + 1/n)^{1/2} - 1) = O(n^2)$.

Assign the $\mu(i, j)$ on each link $(i, j) \in E$ as follows:

$$\mu(i, j) = \begin{cases} 1 + M, & j = t \\ 1, & \text{otherwise} \end{cases}$$

We next show that the optimal path for the above MP-DCP problem is the longest *s*-*t* path in *G*.

Let p_{opt} and p_l denote the optimal MP-DCP *s*-*t* path and a longest *s*-*t* path, respectively.

We obtain $(1 - \mu(p_l)) / \delta(p_l) \leq (1 - \mu(p_{opt})) / \delta(p_{opt})$.

Assume that $\delta^2(p_{opt}) < \delta^2(p_l)$ $(< n)$.

Then we have the following contradiction.

$$1 + 1 / n < \delta^2(p_l)/\delta^2(p_{opt}) \leq ((\mu(p_l) - 1)/(\mu(p_{opt}) - 1))^2 < ((n + M)/M)^2 = 1 + 1 / n. \quad \blacksquare$$

Theorem 5.5: No pseudo polynomial exact algorithm or fully polynomial constant factor approximation algorithm can be obtained for Case 2 of the MP-DCP problem unless P = NP.

***Proof*:** According to Theorem 5.4, the longest path problem is a subclass of the MP-DCP problem with $D = 1$ (Case 2) and thus a pseudo polynomial exact algorithm for this problem, which involves only numbers bounded by polynomial function of *n*, is also applicable to the longest path problem. This would then contradict the fact that there is no pseudo polynomial algorithm for the longest path problem unless P = NP.

If there exists a fully polynomial constant factor approximation algorithm for the MP-DCP problem for Case 2, then let $\varepsilon < 1$ be the approximation factor, and let p_ε and p_l be

the approximate solution to MP-DCP problem and the longest *s*-*t* path, respectively. By the definition of approximation factor for maximum problem, we have

$$| (\chi_D(p_{opt}) - \chi_D(p_\varepsilon) | / \chi_D(p_{opt}) \leq \varepsilon.$$

Hence

$$\chi_D(p_\varepsilon) \geq (1 - \varepsilon)\, \chi_D(p_{opt})$$

So, $(1 - u(p_\varepsilon)) / \delta(p_\varepsilon) \geq (1 - \varepsilon)(1 - u(p_l)) / \delta(p_l)$.

Hence $\delta^2(p_l) / \delta^2(p_\varepsilon) \leq ((1 - \varepsilon)\, (u(p_l) - 1) / (u(p_\varepsilon) - 1))^2$

$$\leq (1 - \varepsilon)^2\, (1 + 1 / n) \leq 2\, (1 - \varepsilon)^2.$$

So, p_ε is a constant factor approximate solution to the longest path problem. This leads to the contradiction of the fact that no constant factor polynomial time approximation algorithm exists for the longest path problem [26]. ■

The barrier to extend the heuristic algorithm of Section 5.2.3 is that the optimum value *OPT* is negative under the assumption that $\forall p$, $\mu(p) > D$. Dijkstra's shortest path algorithm is not applicable due to the likely presence of negative link weights. So we need to use the BFM algorithm. Even this algorithm will fail if there is a negative weighted cycle in the network.

5.4. Summary

In this chapter, we studied the MP-DCP problem. For the case (Case 1) when there is a path whose mean delay is less than or equal to the specified delay bound *D*, we presented an exact algorithm of pseudo polynomial time complexity, an FPTAS, and a

strongly polynomial time heuristic algorithm. In the unlikely case (Case 2) when every path violates this assumption we have shown that the problem is NP-hard. We have also shown that for this case no pseudo polynomial time exact algorithm or fully polynomial time constant factor approximation algorithm is possible unless P = NP. The difficulty in this case arises because we need to find a path minimizing one path metric and maximizing another path metric simultaneously.

Chapter 6. GEN-LARAC: A Generalized Approach to the Constrained Shortest Path Problem under Multiple Additive Constraints

In this chapter we study the CSP(k) problem that requires determination of s-t paths that satisfy k >1 additive constraints. We develop a new approach using Lagrangian relaxation. We use the LARAC algorithm discussed in Chapter 2 as a building block in the design of our algorithm.

The results in this chapter have been repeated in [65].

6.1. Formulation of the CSP(k) Problem and Its Relaxation

Consider a directed graph $G(V, E)$ where V is the set of nodes and E is the set of links in G. Each link (u, v) is associated with a set of $k + 1$ additive non-negative integer weights $\boldsymbol{C_{uv}} = (c_{uv}, w^1_{uv}, w^2_{uv}\ldots, w^k_{uv})$. Here c_{uv} is called the cost of link (u, v) and w^i_{uv} is called the ith delay of (u, v). Given two nodes s and t, an s-t path in G is a directed simple path from s to t. Let P_{st} denote the set of all s-t paths in G. For an s-t path p define

$$c(p) \equiv \sum_{(u,v)\in p} c_{uv} \text{ and } d_i(p) \equiv \sum_{(u,v)\in p} w^i_{uv},\ i = 1,\ldots k.$$

The value $c(p)$ is called the cost of path p, and $d_i(p)$ is called the i^{th} delay of path p. Given k positive integers $r_1, r_2\ldots, r_k$, an s-t path is called feasible (*resp*. strictly feasible)

if $d_i(p) \leq r_i$ (*resp.* $d_i(p) < r_i$), for all $i = 1, 2\ldots k$ (r_i is called the bound on the ith delay of a path).

The CSP(k) problem is to find a minimum cost feasible s-t path. An instance of the CSP(k) problem is strictly feasible if all the feasible paths are strictly feasible. Without loss of generality, we assume that the problem under consideration is always feasible. In order to guarantee strict feasibility, we do the following transformation.

For $i = 1, 2\ldots, k$, transform the ith delay of each link (u, v) such that the new weight vector $\boldsymbol{C'_{uv}}$ is given by

$$\boldsymbol{C'_{uv}} = (c_{uv}, 2\, w^1_{uv,} 2\, w^2_{uv}\ldots, 2\, w^k_{uv}).$$

Also transform the bounds r_i's so that the new vector of bounds R' is given by

$$\boldsymbol{R'} = (2\, r_1 + 1, 2\, r_2 + 1\ldots, 2\, r_k + 1).$$

In the rest of the chapter, we only consider the transformed problem. Thus all link delays are even integers, and delay bounds are odd integers. We will use symbols with capital or bold letters to represent vectors. Also, for a matrix $\boldsymbol{A}$, $\boldsymbol{A}^T$ denotes its transpose. For simplicity of presentation, we will use $\boldsymbol{C_{uv}}$ and $\boldsymbol{R}$ instead of $\boldsymbol{C'_{uv}}$ and $\boldsymbol{R'}$ to denote the transformed weight vector and the vector of bounds.

Two immediate consequences of this transformation are stated below.

Lemma 6.1: $\forall\, p \in P_{st}$, $\forall i \in \{1, 2\ldots, k\}$, $d_i(p) \neq r_i$ in the transformed problem.

Lemma 6.2: An s-t path in the original problem is feasible (*resp.* optimal) iff it is strictly feasible (*resp.* optimal) in the transformed problem.

Starting with an ILP formulation of the CSP(k) problem and relaxing the integrality constraints we get the RELAX-CSP(k) problem below. In this formulation, for each s-t path p, we introduce a variable x_p.

RELAX-CSP(*k*)

$$\text{Minimize } \sum_p c(p)x_p \tag{6.1}$$

$$\text{subject to } \sum_p x_p = 1 \tag{6.2}$$

$$\sum_p d_i(p)x_p \le r_i \quad i = 1, \ldots, k \tag{6.3}$$

$$x_p \ge 0, \forall\ p \in P_{st} \tag{6.4}$$

The Lagrangian dual of RELAX-CSP(k) is given below.

DUAL-RELAX-CSP(*k*):

$$\text{Maximize} \quad w - \lambda_1 r_1 \ldots - \lambda_k r_k \tag{6.5}$$

$$\text{subject to} \quad w - d_1(p)\ \lambda_1 \ldots - d_k(p)\ \lambda_k \le c(p)\ ,\ \forall\ p \in P_{st} \tag{6.6}$$

$$\lambda_i \ge 0,\ i = 1, \ldots, k \tag{6.7}$$

In the above dual problem λ_1, $\lambda_2 \ldots$, λ_k and w are the dual variables, with w corresponding to (6.2) and each λ_i corresponding to the ith constraint in (6.3).

It follows from (6.6) that $w \le c(p) + d_1(p)\ \lambda_1 \ldots + d_k(p)\ \lambda_k\ \forall\ p \in P_{st}$. Since we want to maximize (6.5), the value of w should be as large as possible, i.e.

$$w = \min_{p \in Pst} \{c(p) + d_1(p)\ \lambda_1 + \ldots + d_k(p)\ \lambda_k\}.$$

With the vector $\boldsymbol{\Lambda}$ defined as $\boldsymbol{\Lambda} = (\lambda_1, \lambda_2 \ldots, \lambda_k)$, define

$$L(\boldsymbol{\Lambda}) = \min_{p \in Pst} \{c(p) + \lambda_1 (d_1(p) - r_1) \ldots + \lambda_k (d_k(p) - r_k)\}. \tag{6.8}$$

Notice that $L(\boldsymbol{\Lambda})$ is called the Lagrangian function in literature and is a concave continuous function of $\boldsymbol{\Lambda}$ [7].

Then DUAL-RELAX-CSP(k) can be written as follows.

DUAL-RELAX-CSP(k):

$$\text{Maximize } L(\boldsymbol{\Lambda}) \tag{6.9}$$

$$\text{subject to } \boldsymbol{\Lambda} \geq 0$$

The $\boldsymbol{\Lambda}^*$ that maximizes (6.9) is called the maximizing multiplier and is defined as

$$\boldsymbol{\Lambda}^* = \arg\max_{\Lambda \geq 0} L(\boldsymbol{\Lambda}) \tag{6.10}$$

Claim 6.1: If an instance of the CSP(k) problem is feasible and a path p_{opt} is an optimal path, then $\forall\, \boldsymbol{\Lambda} \geq 0$, $L(\boldsymbol{\Lambda}) \leq c(p_{opt})$.

We shall use $L(\boldsymbol{\Lambda})$ as an lower bound of $c(p_{opt})$ to evaluate the quality of the approximate solution obtained by our algorithm. Given $p \in P_{st}$ and $\boldsymbol{\Lambda}$, define

$$\boldsymbol{C}(p) \equiv (c(p), d_1(p), d_2(p) \ldots, d_k(p)) \text{ , } \boldsymbol{D}(p) \equiv (d_1(p), d_2(p) \ldots, d_k(p)),$$

$$\boldsymbol{R} \equiv (r_1, r_2 \ldots, r_k),\ c_{\boldsymbol{\Lambda}}(p) \equiv c(p) + d_1(p)\ \lambda_1 \ldots + d_k(p)\ \lambda_k, \text{ and}$$

$$d_{\boldsymbol{\Lambda}}(p) \equiv d_1(p)\ \lambda_1 \ldots + d_k(p)\ \lambda_k\ .$$

Here $c_{\boldsymbol{\Lambda}}(p)$ and $d_{\boldsymbol{\Lambda}}(p)$ are called the aggregated cost and the aggregated delay of path p, respectively. We shall use $P_{\boldsymbol{\Lambda}}$ to denote the set of s-t paths attaining the minimum aggregated cost w.r.t. to $\boldsymbol{\Lambda}$. A path $p_{\boldsymbol{\Lambda}} \in P_{\boldsymbol{\Lambda}}$ is called a $\boldsymbol{\Lambda}$-minimal path.

6.2. A Strongly Polynomial Time Approximation Algorithm for CSP(1) Problem

The key issue now is to search for the maximizing multiplier and termination conditions. If there is only one delay constraint, i.e., $k = 1$, we have the following claim from [23] also proved in Chapter 2.

Claim 6.2[23]: A value $\lambda > 0$ maximizes the function $L(\lambda)$ if and only if there are paths p_c and p_d which are both c_λ-minimal and for which $d(p_c) \geq r$ and $d(p_d) \leq r$. (p_c and p_d can be the same. In this case $d(p_d) = d(p_c) = r$).

Theorem 6.1: DUAL-RELAX-CSP(1) is solvable in $O((m + n \log n)^2)$ time.

Proof: We prove this theorem by presenting an algorithm with $O((m + n \log n)^2)$ time complexity.

Assume node 1 is the source and node n is the target. In Figure 6.1, we present an algorithm for computing a shortest path using lexicographic order on a pair of link weights (l_{uv}, c_{uv}) $\forall$ $(u, v) \in E$ based on parametric search, where $l_{uv} = c_{uv} + \lambda^* d_{uv}$ and λ^* is unknown. The algorithm is the same as BFM algorithm except for Step 4 which needs special care (We use BFM algorithm here because it is easy to explain. Actually we use Dijkstra's algorithm for better time complexity).

Step 1. $M_v = (x_v, y_v) = (+\infty, +\infty)$ for $v = 2, 3\ldots, n$ and $M_1 = (0, 0)$
Step 2. $i \leftarrow 1$
Step 3. $u \leftarrow 1$
Step 4*. $\forall$ v, $(u, v) \in E$, if $(x_v + \lambda^* y_v > x_u + \lambda^* y_u + c_{uv} + \lambda^* d_{uv})$ or
$(x_v + \lambda^* y_v = x_u + \lambda^* y_u + c_{uv} + \lambda^* d_{uv})$ and $(x_v > x_u + c_{uv})$)
$M_v \leftarrow (x_u + c_{uv}, y_u + d_{uv})$

Step 5. $u \leftarrow u + 1$ and if $u \leq n$, go to Step 4.
Step 6. $i \leftarrow i + 1$ and if $i < n$, go to Step 3.

Figure 6.1: Parametric search based algorithm for CSP(1) problem

In Figure 6.1, we need extra steps (Oracle test) to evaluate the Boolean expression in the if statement in Step 4 since $\lambda^* \geq 0$ is unknown. If $x_v = \infty$, $y_v = \infty$, then the inequality holds. Assume x_v and y_v are finite (non-negative) values. Then it suffices to evaluate the following Boolean expression.

$$p + q\,\lambda^* \le 0\ ?, \text{ where } p = x_u + c_{uv} - x_v \text{ and } q = (y_u + d_{uv} - y_v).$$

If $p \cdot q \ge 0$, then it is trivial to evaluate the Boolean expression. WLOG, assume $p \cdot q < 0$, i.e., $-p / q > 0$. The Oracle test algorithm is presented in Figure 6.2.

The time complexity of the Oracle test is $O(m + n \log n)$. On the other hand, we can revise the algorithm in Figure 6.1 using Dijkstra's algorithm and the resulting algorithm will have time complexity $O((m + n \log n)^2)$.

Next, we show how to compute the value of λ^* and $L(\lambda^*)$. The algorithm in Figure 6.1 computes a λ^*-minimal path p with minimal cost. Similarly, we can compute a λ^*-minimal path q with minimal delay. Then the value of λ^* is given by the following equation: $c(p) + \lambda^* d(p) = c(q) + \lambda^* d(q)$ and $L(\lambda^*) = c(p) + \lambda^*(d(p) - T)$, where T is the path delay constrain (here $k = 1$). Notice that $d(q) \neq d(p)$ is guaranteed by our transformation in Section 6.1. ■

T: The path delay constraint

Step 1. Let $\lambda = -p/q > 0$ for each link $(u, v) \in E$,
define its length $l_{uv} = c_{uv} + \lambda\, d_{uv}$.
Step 2. Compute two shortest paths p_c and p_d using the lexicographic order on (l_{uv}, c_{uv}) and (l_{uv}, d_{uv}), respectively [53].
Step 3. Obviously, $d(p_c) \ge d(p_d)$. Only four cases are possible:

a) $d(p_c) > T$ and $d(p_d) > T$: By Claim 6.2, $\lambda < \lambda^*$
and thus $p + q\,\lambda^* < 0$ if $q < 0$ and $p + q\,\lambda^* > 0$ otherwise.
b) $d(p_c) < T$ and $d(p_d) < T$: By Claim 6.2, $\lambda > \lambda^*$
and thus $p + q\,\lambda^* > 0$ if $q < 0$ and $p + q\,\lambda^* < 0$ otherwise.
c) $d(p_c) > T$ and $d(p_d) < T$: By Claim 6.2, $\lambda = \lambda^*$ and $p + q\,\lambda^* = 0$.
d) $d(p_c) = T$ or $d(p_d) = T$: By Lemma 6.1, this is impossible.

Figure 6.2: Oracle test algorithm

Because our algorithm and LARAC are based on the same methodology and obtain the same solution, we shall also call our algorithm LARAC. In the rest of the chapter,

we shall discuss how to extend it for $k > 1$. In particular we develop an approach that combines the LARAC algorithm as a building block with certain techniques in mathematical programming. We shall call this new approach as GEN-LARAC.

6.3. GEN-LARAC for the CSP(k) Problem

6.3.1. Optimality Conditions

Theorem 6.2: Given an instance of a feasible CSP(k) problem, a vector $\boldsymbol{\Lambda} \geq 0$ maximizes $L(\boldsymbol{\Lambda})$ iff the following problem in the variables u_j is feasible.

$$\sum_{p_j \in P_\Lambda} u_j \cdot d_i(p_j) = r_i, \forall i, \lambda_i > 0 \tag{6.11}$$

$$\sum_{p_j \in P_\Lambda} u_j \cdot d_i(p_j) \leq r_i, \forall i, \lambda_i = 0 \tag{6.12}$$

$$\sum_{p_j \in P_\Lambda} u_j = 1 \tag{6.13}$$

$$u_j \geq 0, \forall p_j \in P_\Lambda \tag{6.14}$$

Proof: **Sufficiency**: Let $\boldsymbol{x} = (u_1 \ldots, u_r, 0, 0 \ldots)$ be a vector of size $|P_{st}|$, where $r = |P_{\boldsymbol{\Lambda}}|$. Obviously, $\boldsymbol{x}$ is a feasible solution to RELAX-CSP(k). It suffices to show that $\boldsymbol{x}$ and $\boldsymbol{\Lambda}$ satisfy the complementary slackness conditions.

According to (6.6), $\forall p \in P_{st}$, $w \leq c(p) + d_1(p)\ \lambda_1 \ldots + d_k(p)\ \lambda_k$. Since we need to maximize (6.5), the optimal $w = c(p_{\boldsymbol{\Lambda}}) + d_1(p_{\boldsymbol{\Lambda}})\ \lambda_1 \ldots + d_k(p_{\boldsymbol{\Lambda}})\ \lambda_k\ \forall\ p_{\boldsymbol{\Lambda}} \in P_{\boldsymbol{\Lambda}}$. For all other paths p, $w - c(p) + d_1(p)\ \lambda_1 \ldots + d_k(p)\ \lambda_k < 0$. So $\boldsymbol{x}$ satisfies the complementary slackness conditions. By (6.11) and (6.12), $\boldsymbol{\Lambda}$ also satisfies complementary slackness conditions.

Necessary: Let $\boldsymbol{x}^*$ and $(w, \boldsymbol{\Lambda})$ be the optimal solution to RELAX-CSP(k) and DUAL-RELAX-CSP(k), respectively. It suffices to show that we can obtain a feasible solution to (6.11)-(6.14) from $\boldsymbol{x}^*$.

We know that all the constraints in (6.6) corresponding to paths in $P_{st} - P_{\boldsymbol{\Lambda}}$ are strict inequalities, and $w = c(p_{\boldsymbol{\Lambda}}) + d_1(p_{\boldsymbol{\Lambda}})\,\lambda_1 \ldots + d_k(p_{\boldsymbol{\Lambda}})\lambda_k \;\forall p_{\boldsymbol{\Lambda}} \in P_{\boldsymbol{\Lambda}}$. So, from complementary slackness conditions we get $x_p = 0, \forall\, p \in P_{st} - P_{\boldsymbol{\Lambda}}$.

Now let us set u_j corresponding to path p in $P_{\boldsymbol{\Lambda}}$ equal to x_p, and set all other u_j's corresponding to paths not in $P_{\boldsymbol{\Lambda}}$ equal to zero. The u_i'*s* so elected will satisfy (6.11) and (6.12) since these are complementary conditions satisfied by $(w, \boldsymbol{\Lambda})$. Since x_i's satisfy (6.2), u_j's satisfy (6.13). Thus we have identified a solution satisfying (6.11)-(6.14). ■

6.3.2. GEN-LARAC: A Coordinate Ascent Method

Our approach is based on the coordinate ascent method and proceeds as follows. Given a multiplier $\boldsymbol{\Lambda}$, in each iteration we try to improve the value of $L(\boldsymbol{\Lambda})$ by updating one component of the multiplier vector. If the objective function is not differentiable, the coordinate ascent method may get stuck at a corner $\boldsymbol{\Lambda}_s$ not being able to make progress by only changing one component. We call $\boldsymbol{\Lambda}_s$ pseudo optimal point which requires updates of at least two components to achieve improvement in the solution. We shall discuss in Section 6.3.3 how to jump to a better solution from a pseudo optimal point. Our simulations show that the objective value attained at pseudo optimal points is usually very close to the maximum value of $L(\boldsymbol{\Lambda})$.

Step 1: $\boldsymbol{\Lambda}^0 \leftarrow (0, 0\ldots, 0)$; $t \leftarrow 0$; *flag* $\leftarrow$ true; $B \leftarrow 0$
Step 2: (**Coordinate Ascent Steps**)
while (*flag*)
flag $\leftarrow$ false
for $i = 1$ to k
$\gamma \leftarrow \arg\ \max_{\xi \geq 0} L(\lambda^t_1 \ldots, \lambda^t_{i-1}, \xi, \lambda^t_{i+1} \ldots, \lambda^t_k)$.
if $(\gamma \neq \lambda^t_i)$ then
flag $\leftarrow$ true

$$\lambda_j^{t+1} = \begin{cases} \gamma & j = i, \\ \lambda_j^t & j \neq i. \end{cases}, j = 1, 2\ldots, k$$

$t \leftarrow t + 1$
end if
end for
end while
Step 3: If $\boldsymbol{\Lambda}^t$ is optimal then return $\boldsymbol{\Lambda}^t$.
Step 4: $B \leftarrow B + 1$ and go to Step 5 if $B < B_{max}$ (B_{max} is the maximum number of iteration allowed); Otherwise, stop.
Step 5: Compute a new vector $\boldsymbol{\Lambda}^+$ such that $L(\boldsymbol{\Lambda}^+) > L(\boldsymbol{\Lambda}^t)$.
Step 6: $t \leftarrow t + 1$, $\boldsymbol{\Lambda}^t \leftarrow \boldsymbol{\Lambda}^+$, and go to Step 2.

Figure 6.3: GEN-LARAC: A coordinate ascent algorithm

6.3.3. Verification of Optimality of Λ

In Step 3 we need to verify if a given $\boldsymbol{\Lambda}$ is optimal. We show that this can be accomplished by solving the following LP problem, where $\boldsymbol{P}_{\boldsymbol{\Lambda}} = \{p_1, p_2\ldots, p_r\}$ is the set of $\boldsymbol{\Lambda}$-minimal paths.

$$\text{Maximize} \quad 0 \tag{6.15}$$

$$\text{subject to} \quad \sum_{p_j \in P_\Lambda} u_j \cdot d_i(p_j) = r_i, \forall i, \lambda_i > 0 \tag{6.16}$$

$$\sum_{p_j \in P_\Lambda} u_j \cdot d_i(p_j) \leq r_i, \forall i, \lambda_i = 0 \tag{6.17}$$

$$\sum_{p_j \in P_\Lambda} u_j = 1 \tag{6.18}$$

$$u_j \geq 0, \forall p_j \in P_\Lambda \tag{6.19}$$

By Theorem 6.2, if the above linear program is feasible then the multiplier $\boldsymbol{\Lambda}$ is a maximizing multiplier.

Let ($y_1 \ldots, y_k, \delta$) be the dual variables corresponding to the above problem. Let $\boldsymbol{Y} = (y_1, y_2 \ldots, y_k)$. The dual of (6.15)-(6.19) is as follows

$$\text{Minimize } \boldsymbol{R}\,\boldsymbol{Y}^T + \delta \tag{6.20}$$

$$\text{subject to } \boldsymbol{D}(p_i)\,\boldsymbol{Y}^T + \delta \geq \boldsymbol{0},\ i = 1, 2..., r \tag{6.21}$$

$$y_i \geq 0,\ \forall\ i,\ \lambda_i > 0 \tag{6.22}$$

Evidently the LP problem (6.20)-(6.22) is feasible. From the relationship between primal and dual problems, it follows that if the linear program (6.15)-(6.19) is infeasible, then the objective of (6.20) is unbounded ($-\infty$). Thus, if the optimum objective of (6.20)-(6.22) is 0, then the linear program (6.15)-(6.19) is feasible and by Theorem 6.2 the corresponding multiplier $\boldsymbol{\Lambda}$ is optimal. In summary, we have the following lemma.

Lemma 6.3: If (6.15)-(6.19) is infeasible, then $\exists\ \boldsymbol{Y} = (y_1, y_2 \ldots, y_k)$ and δ satisfy (6.21)-(6.22) and $\boldsymbol{R}\,\boldsymbol{Y}^{\mathrm{T}} + \delta < 0$.

The $\boldsymbol{Y}$ in Lemma 6.3 can be identified by applying any LP solver on (6.20)-(6.22) and terminating it once the current objective value becomes negative.

Let $\boldsymbol{\Lambda}$ be a non-optimal Lagrangian multiplier and $\boldsymbol{\Lambda}(s, \boldsymbol{Y}) = \boldsymbol{\Lambda} + \boldsymbol{Y} / s$ for $s > 0$.

Theorem 6.3: If a multiplier $\boldsymbol{\Lambda} \geq 0$ is not optimal, then

$$\exists\ M > 0,\ \forall\ s > M,\ L(\boldsymbol{\Lambda}(s, \boldsymbol{Y})) > \boldsymbol{L}(\boldsymbol{\Lambda}).$$

Proof: If M is big enough, $P_{\boldsymbol{\Lambda}} \cap P_{\boldsymbol{\Lambda}(s, \boldsymbol{Y})} \neq \varnothing$. Let $p_J \in P_{\boldsymbol{\Lambda}} \cap P_{\boldsymbol{\Lambda}(s, \boldsymbol{Y})}$.

$$L(\boldsymbol{\Lambda}(s, \boldsymbol{Y})) = c(p_J) + (\boldsymbol{D}(p_J) - \boldsymbol{R})\,(\boldsymbol{\Lambda} + \boldsymbol{Y} / s)^T$$

$$= c(p_J) + (\boldsymbol{D}(p_J) - \boldsymbol{R})\,\boldsymbol{\Lambda}^T + (\boldsymbol{D}(p_J) - \boldsymbol{R})\,(\boldsymbol{Y} / s)^T$$

$$= L(\boldsymbol{\Lambda}) + (\boldsymbol{D}(p_J)\boldsymbol{Y}^T - \boldsymbol{R}\,\boldsymbol{Y}^T) / s.$$

Since $\boldsymbol{D}(p_J)\boldsymbol{Y}^T + \delta \geq 0$ and $\boldsymbol{R}\,\boldsymbol{Y}^T + \delta < 0$, $\boldsymbol{D}(p_J)\boldsymbol{Y}^T - \boldsymbol{R}\,\boldsymbol{Y}^T > 0$.

Hence $L(\boldsymbol{\Lambda}(s, \boldsymbol{Y})) > L(\boldsymbol{\Lambda})$. ■

We can find the proper value of M by binary search after computing $\boldsymbol{Y}$. The last issue is to compute $P_{\boldsymbol{\Lambda}}$. It can be expected that the size of $P_{\boldsymbol{\Lambda}}$ is usually very small. In our experiments, $|P_{\boldsymbol{\Lambda}}|$ never exceeded 4 even for large and dense networks. The k-shortest path algorithm can be adapted easily to computing $\boldsymbol{P}_{\boldsymbol{\Lambda}}$.

6.3.4. Analysis of the Algorithm

In this section, we shall discuss the convergence properties of GEN-LARAC.

Lemma 6.4: If there is a strictly feasible path, then for any given τ, the set $S_\tau = \{\boldsymbol{\Lambda} \mid L(\boldsymbol{\Lambda}) \geq \tau\} \subset R^k$ is bounded.

***Proof*:** Let p^* be a strictly feasible path. For any $\boldsymbol{\Lambda} = (\lambda_1 \ldots, \lambda_k) \in S_\tau$, we have

$$c(p^*) + \lambda_1(d_1(p^*) - r_1) \ldots + \lambda_k(d_k(p^*) - r_k) \geq L(\boldsymbol{\Lambda}) \geq \tau\,.$$

Since $d_i(p^*) - r_i < 0$ and $\lambda_i \geq 0$ for $i = 1, 2..., k$, $\boldsymbol{\Lambda}$ must be bounded. ■

By Claim 6.2, we have the following lemma.

Lemma 6.5: A multiplier $\boldsymbol{\Lambda} \geq 0$ is pseudo optimal iff $\forall i\ \exists\ p^i_c, p^i_d \in P_{\boldsymbol{\Lambda}}$, $d_i(p^i_c) \geq r_i$ and $d_i(p^i_d) \leq r_i$ ■

For an n-vector $\boldsymbol{V} = (v_1, v_2 \ldots, v_n)$, let $|\boldsymbol{V}|_1 = |v_1| + |v_2| \ldots + |v_n|$ denote the L^1-norm.

Lemma 6.6: Let $\boldsymbol{\Lambda}$ and $\boldsymbol{H}$ be two multipliers obtained in the same while-loop in Step 2 in Figure 6.3. Then $|L(\boldsymbol{H}) - L(\boldsymbol{\Lambda})| \geq |\boldsymbol{H} - \boldsymbol{\Lambda}|_1$.

Proof: Let $\boldsymbol{\Lambda} = \boldsymbol{\Lambda}^1, \boldsymbol{\Lambda}^2 \ldots, \boldsymbol{\Lambda}^j = \boldsymbol{H}$ be the consecutive sequence of multipliers obtained in Step 2. We first show that $| L(\boldsymbol{\Lambda}^{i+1}) - L(\boldsymbol{\Lambda}^i) | \geq |\boldsymbol{\Lambda}^{i+1} - \boldsymbol{\Lambda}^i|_1$.

Consider two cases.

Case 1: $\lambda_b^{i+1} > \lambda_b^i$. By Claim 6.2 and Lemma 6.1, $\exists\, p_{\boldsymbol{\Lambda}}^{i+1}$ and $d_b(p_{\boldsymbol{\Lambda}}^{i+1}) > r_b$.

By definition, we have:

$$c(p_{\boldsymbol{\Lambda}}^{i+1}) + \boldsymbol{\Lambda}^{i+1}\boldsymbol{D}^{\boldsymbol{T}}(p_{\boldsymbol{\Lambda}}^{i+1}) \leq c(p_{\boldsymbol{\Lambda}}^{i}) + \boldsymbol{\Lambda}^{i+1}\, \boldsymbol{D}^{\boldsymbol{T}}(p_{\boldsymbol{\Lambda}}^{i}), \text{ and}$$

$$c(p_{\boldsymbol{\Lambda}}^{i+1}) + \boldsymbol{\Lambda}^{i}\, \boldsymbol{D}^{\boldsymbol{T}}(p_{\boldsymbol{\Lambda}}^{i+1}) \geq c(p_{\boldsymbol{\Lambda}}^{i}) + \boldsymbol{\Lambda}^{i}\, \boldsymbol{D}^{\boldsymbol{T}}(p_{\boldsymbol{\Lambda}}^{i})$$

Then

$$\begin{aligned}
L(\boldsymbol{\Lambda}^{i+1}) - L(\boldsymbol{\Lambda}^i) &= c(p_{\boldsymbol{\Lambda}}^{i+1}) + \boldsymbol{\Lambda}^{i+1}\, (\boldsymbol{D}(p_{\boldsymbol{\Lambda}}^{i+1}) - \boldsymbol{R})^{\boldsymbol{T}} - [c(p_{\boldsymbol{\Lambda}}^{i}) + \boldsymbol{\Lambda}^{i}\, (\boldsymbol{D}(p_{\boldsymbol{\Lambda}}^{i}) - \boldsymbol{R})^{\boldsymbol{T}}] \\
&= c(p_{\boldsymbol{\Lambda}}^{i+1}) + \boldsymbol{\Lambda}^{i}\, (\boldsymbol{D}(p_{\boldsymbol{\Lambda}}^{i+1}) - \boldsymbol{R})^{T} + (\boldsymbol{\Lambda}^{i+1} - \boldsymbol{\Lambda}^{i})\, (\boldsymbol{D}(p_{\boldsymbol{\Lambda}}^{i+1}) - \boldsymbol{R})^{T} \\
&\qquad - [c(p_{\boldsymbol{\Lambda}}^{i}) + \boldsymbol{\Lambda}^{i}\, (\boldsymbol{D}(p_{\boldsymbol{\Lambda}}^{i}) - \boldsymbol{R})^{T}] \\
&\geq c(p_{\boldsymbol{\Lambda}}^{i}) + \boldsymbol{\Lambda}^{i}\, (\boldsymbol{D}(p_{\boldsymbol{\Lambda}}^{i}) - \boldsymbol{R})^{T} + (\boldsymbol{\Lambda}^{i+1} - \boldsymbol{\Lambda}^{i})\, (\boldsymbol{D}(p_{\boldsymbol{\Lambda}}^{i+1}) - \boldsymbol{R})^{T} \\
&\qquad - [c(p_{\boldsymbol{\Lambda}}^{i}) + \boldsymbol{\Lambda}^{i}\, (\boldsymbol{D}(p_{\boldsymbol{\Lambda}}^{i}) - \boldsymbol{R})^{T}] \\
&= (\boldsymbol{\Lambda}^{i+1} - \boldsymbol{\Lambda}^{i})\, (\boldsymbol{D}(p_{\boldsymbol{\Lambda}}^{i+1}) - \boldsymbol{R})^{T} = (\lambda_b^{i+1} - \lambda_b^{i})\, (d_b(p_{\boldsymbol{\Lambda}}^{i+1}) - r_b) \\
&\geq |\lambda_b^{i+1} - \lambda_b^{i}\,|.
\end{aligned}$$

Case 2: $\lambda_b^{i+1} < \lambda_b^i$. By Claim 6.2 and Lemma 6.1, $\exists\, p_{\boldsymbol{\Lambda}}^{i+1}$ and $d_b(p_{\boldsymbol{\Lambda}}^{i+1}) < r_b$. The rest of the proof is similar to Case 1.

Hence

$$\begin{aligned}
|L(\boldsymbol{\Lambda}^j) - L(\boldsymbol{\Lambda}^1)| &= | L(\boldsymbol{\Lambda}^2) - L(\boldsymbol{\Lambda}^1) + L(\boldsymbol{\Lambda}^3) - L(\boldsymbol{\Lambda}^2) \ldots + L(\boldsymbol{\Lambda}^j) - L(\boldsymbol{\Lambda}^{j-1})| \,. \\
&= | L(\boldsymbol{\Lambda}^2) - L(\boldsymbol{\Lambda}^1)| + |L(\boldsymbol{\Lambda}^3) - L(\boldsymbol{\Lambda}^2)| \ldots + |L(\boldsymbol{\Lambda}^j) - L(\boldsymbol{\Lambda}^{j-1})|
\end{aligned}$$

$$\geq |\boldsymbol{\Lambda}^2 - \boldsymbol{\Lambda}^1|_1 + |\boldsymbol{\Lambda}^3 - \boldsymbol{\Lambda}^2|_1 \ldots + |\boldsymbol{\Lambda}^j - \boldsymbol{\Lambda}^{j-1}|_1 \geq |\boldsymbol{\Lambda}^j - \boldsymbol{\Lambda}^1|_1.$$

The second equality holds because $L(\boldsymbol{\Lambda}^1) < L(\boldsymbol{\Lambda}^2) \ldots < L(\boldsymbol{\Lambda}^j)$. ■

Obviously, if the while-loop in Step 2 in Figure 6.3 terminates in a finite number of steps, the multiplier is pseudo optimal by definition. If the while loop does not terminate in a finite number of steps (this occurs only when infinite machine precision is assumed, in practice, GEN-LARAC terminates in finite steps), we have the following theorem.

Theorem 6.4: Let $\{\boldsymbol{\Lambda}^i\}$ be a consecutive sequence of multipliers generated in the same while-loop in Step 2 in Figure 6.3. Then the limit point of $\{L(\boldsymbol{\Lambda}^i)\}$ is pseudo optimal.

Proof: Since $L(\boldsymbol{\Lambda}^1) < L(\boldsymbol{\Lambda}^2) < \ldots$ and $\{\boldsymbol{\Lambda}^i\}$ is bounded from above, $\lim_{s\to\infty} L(\boldsymbol{\Lambda}^s)$ exists and is denoted as L^*. We next show the vector $\lim_{s\to\infty} \boldsymbol{\Lambda}^s$ also exists.

By Lemma 6.6, $\forall\ s, j > 0$,

$$|\boldsymbol{\Lambda}^{s+j} - \boldsymbol{\Lambda}^s|_1 \leq |L(\boldsymbol{\Lambda}^{s+j}) - L(\boldsymbol{\Lambda}^s)|$$

By Cauchy criterion, $\lim_{s\to\infty} \boldsymbol{\Lambda}^s$ exists. We denote it as $\boldsymbol{\Lambda}^*$.

We label all the paths in P_{st} as $p_1, p_2 \ldots, p_N$ such that $c_{\Lambda^*}(p_1) \leq c_{\Lambda^*}(p_2) \ldots \leq c_{\Lambda^*}(p_N)$. Obviously p_1 is a $\boldsymbol{\Lambda}^*$-minimal path.

Let

$$\theta = \min \{c_{\Lambda^*}(p_j) - c_{\Lambda^*}(p_i) \mid \forall\ p_i, p_j \in P_{st}, c_{\Lambda^*}(p_j) - c_{\Lambda^*}(p_i) > 0\},$$

and

$$\pi = \max \{d_w(p_j) - d_w(p_i) \mid \forall\ p_i, p_j \in P_{st}, w = 1, 2 \ldots, k\}.$$

Let M be a large number, such that $\forall\ t \geq M$, $|\Lambda^* - \Lambda^t|_1 < \theta / (2\pi)$.

Consider any component $j \in \{1, 2\ldots, k\}$, of the multiplier after computing

$\arg\max_{\xi \geq 0} L(\lambda_1\ldots, \lambda_{j-1}, \xi, \lambda_{j+1}\ldots, \lambda_k)$ in iteration $t \geq M$.

By Claim 6.2, $\exists\, p_c{}^t$ and $p_d{}^t \in P_\Lambda{}^t$ and $d_j(p_c{}^t) \geq l_j \geq d_j(p_d{}^t)$.

It suffices to show $P_\Lambda{}^t \subseteq P_\Lambda{}^*$. Given $p_\Lambda{}^t \in P_\Lambda{}^t$, we shall show $c_\Lambda{}^*(p_\Lambda{}^t) = c_\Lambda{}^*(p_1)$.

We have

$$0 \leq c_\Lambda{}^t(p_1) - c_\Lambda{}^t(p_\Lambda{}^t) = [c(p_1) + d_\Lambda{}^t(p_1)] - [c(p_\Lambda{}^t) + d_\Lambda{}^t(p_\Lambda{}^t)]$$

$$= c(p_1) + d_\Lambda{}^*(p_1) - [c(p_\Lambda{}^t) + d_\Lambda{}^*(p_\Lambda{}^t)] + (\Lambda^t - \Lambda^*)\,(D(p_1) - D(p_\Lambda{}^t))^T$$

$$= c_\Lambda{}^*(p_1) - c_\Lambda{}^*(p_\Lambda{}^t) + (\Lambda^t - \Lambda^*)\,(D(p_1) - D(p_\Lambda{}^t))^T$$

Since $|(\Lambda^t - \Lambda^*)\,(D(p_i) - D(p_j))^T| \leq \pi\,|(\Lambda^t - \Lambda^*)|_1 \leq \theta / 2$,

$$0 \leq c_\Lambda{}^t(p_1) - c_\Lambda{}^t(p_\Lambda{}^t) \leq c_\Lambda{}^*(p_1) - c_\Lambda{}^*(p_\Lambda{}^t) + \theta / 2.$$

Then

$c_\Lambda{}^*(p_\Lambda{}^t) - c_\Lambda{}^*(p_1) \leq \theta / 2$, which implies that $c_\Lambda{}^*(p_\Lambda{}^t) = c_\Lambda{}^*(p_1)$.

Hence, $\forall\, j \in \{1, 2..., k\}, \exists\, p_c{}^*, p_d{}^* \in P_\Lambda{}^*, d_j(p_c{}^*) \geq l_j \geq d_j(p_d{}^*)$ ■

6.4. Simulation

We use *COPT*, *OPT*, and *POPT* to denote the cost of the optimal path to the CSP(k) problem, the optimal value, and the pseudo optimal value of the Lagrangian function, respectively. In our simulation, we first verify that the objectives at pseudo optimal points are very close to the optimal objectives. We use 3 types of graphs: Power-law out-degree graph (PLO) [44], Waxman's random graph (RAN) [59], and regular graph (REG) [55]. The number of weights chosen are 4, 8 and 12, i.e., k = 4, 8, and 12. Link weights are random even integers uniformly distributed between 2 and 200. To decide

proper delay bounds, we randomly choose an *s*-*t* path p_{ran} and set $r_i = (1 + \varepsilon)\, d_i(p_{ran})$ where ε is a random variable evenly distributed in interval [– 0.25, 0.25]. For each type of topology, 10 different graphs with valid delay bounds and random source and target nodes are generated. The results reported are averaged over the 10 instances.

We use the following two metrics to measure the quality of path p in Table 6.1.

$$g(p) = c(p) / POPT \text{ and } f(p) = \max_{i = 1, 2..., k} d_i(p)/r_i$$

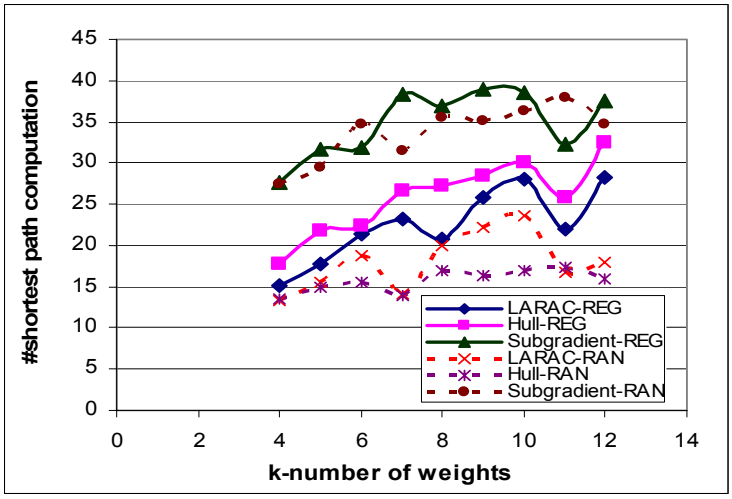
By Claim 6.1, $g(p)$ is the upper bound of the gap between the cost of p and *COPT*. The $f(p)$ indicates the degree of violation of p to the constraints on its delays.

Table 6.1: Quality of pseudo-optimal paths

Error: (OPT – POPT) / OPT #SP: Number of computation of shortest path

Type	*k*	OPT	POPT	Error	*g*(*p*)	*f*(*p*)	#SP	Time(*s*)
REG	4	1116.8	1109.9	0.006	1.01	1.07	15.2	0.14
REG	8	1078.3	1069.0	0.032	1.00	1.09	20.7	0.21
REG	12	1066.2	1057.8	0.008	1.00	1.08	28.2	0.32
PLO	4	401.75	382.71	0.047	1.00	1.25	9.6	0.07
PLO	8	328.95	320.77	0.025	1.01	1.15	7.2	0.04
PLO	12	368.43	342.43	0.071	1.02	1.24	18.3	0.21
RAN	4	1543.6	1531.5	0.008	1.01	1.08	13.4	0.13
RAN	8	1473.3	1456.5	0.011	1.00	1.09	20.0	0.25
RAN	12	1438.6	1423.7	0.010	1.00	1.01	17.9	0.45

In Figure 6.4, the label LARAC-REG means the results obtained by running GEN-LARAC algorithm on regular graphs. Other labels can be interpreted similarly. We only report the results on regular graphs and random graphs for better visibility.



The comparison of the number of shortest path computation among GEN-LARAC, Hull approach, and subgradient method. All algorithms terminate when have reached 99% of the OPT.

Figure 6.4: Comparison of GEN-LARAC, Hull approach, and subgradient method

We conducted extensive experiments to compare our algorithm with the Hull approach [39], the subgradient method [2], and the general-purpose LP solver CPLEX. Because the four approaches share the same objective, i.e., maximizing the Lagrangian function, they always obtain similar results. We only report the number of shortest path computations which dominate the running time of all the first three algorithms**.** Generally, GEN-LARAC algorithm and Hull approach are faster than the subgradient methods and CPLEX (See [69] for the comparison of Hull approach and CPLEX). But GEN-LARAC and Hull approach beat each other on different graphs. Figure 6.4 shows that on the regular graph, GEN-LARAC is the fastest. But for the random and Power-law out degree graphs, the Hull approach is the fastest. The probable reason is that the number of *s*-*t* paths is relatively small in these two types of graphs because the length (number of hops) of *s*-*t* paths is short even when the number of nodes is large. This will

bias the results in favor of Hull approach which adds one *s*-*t* path into the linear system in each iteration [39]. We choose the regular graph because we have a better control of the length of *s*-*t* paths.

6.5. Summary and Conclusion

In this chapter we developed a new approach to the constrained shortest path problem involving multiple additive constraints. Our approach uses the LARAC algorithm as a building block and combines it with certain ideas from mathematical programming to design a method that progressively improves the value of the Lagrangian function until optimum is reached. The algorithm is analyzed and its convergence property has been established. Simulation results comparing our approach with two other approaches show that the new approach is quite competitive.

Since the LARAC algorithm is applicable for the general class of optimization problems (involving one additive delay constraint) studied in [6] our approach can also be extended for this class of problems whenever an algorithm for the underlying optimization problem (such as Dijkstra's algorithm for the shortest path problem) is available.

Chapter 7. Summary

In this dissertation, we have studied the end-to-end Quality of Service (QoS) and fault tolerance issues in computer communication networks. Many problems that fall under this theme can be modeled as constrained shortest path(s) selection problems on networks with each of their links associated with two or more weights representing the cost, delay, reliability, delay-jitter, and others additive parameters. We considered four fundamental problems that are encountered in this area.

In Chapters 2 and 3, we studied the QoS single route selection problem, i.e., the constrained shortest path (CSP) problem. The goal of the CSP problem is to identify a minimum cost route which incurs a delay less than a specified bound. The CSP problem can be formulated as an integer linear programming (ILP) problem which is computationally intractable. A class of approximation algorithms has been proposed in the literature based on the linear programming relaxation of the ILP formulation.

In Chapter 2, we reviewed several algorithms solving the dual problem of the relaxed problem and showed their equivalence. We then proposed two new approximation algorithms solving the dual problem using binary search and parametric search, respectively. We also pointed out how to integrate these algorithms to speed up ε-approximation algorithms.

In Chapter 3, we proposed a novel approximation algorithm which is also based on the relaxed ILP formulation. However, we studied the problem using the primal simplex method of linear programming and exploiting certain structural properties of networks.

Our simulation shows that our algorithm is faster than the dual based algorithms on dense networks.

In Chapter 4, we generalized the algorithms in Chapters 2 and 3 for constrained shortest link-disjoint paths selection problem which requires a set of, say $k > 1$, link-disjoint paths with minimum total cost and with total delay bounded by a given upper bound. This problem, referred as the CSDP(k) problem, arises in the context of provisioning paths in a network that could be used to provide resilience to failures in one or more of these paths.

All the algorithms and methods in Chapters 2, 3, and 4 assume that the network status/parameters are known and accurate. However, in real networks, this assumption is not realistic. In Chapter 5, we considered the QoS route selection problem under inaccurate state information. In this problem, the link delays are random variables. This problem is also called stochastic shortest path selection problem. The goal of this problem is to find a path that has the highest probability to satisfy a given upper bound on delay. This problem is denoted as the MP-DCP problem. Base on central limit theorem, we proposed a pseudo polynomial time approximation algorithm, a fully polynomial time approximation scheme (FPTAS), and a strongly polynomial time heuristic.

In Chapter 6, we studied the constrained shortest path problem with multiple additive constrains. Using the dual algorithms discussed in Chapter 2 and combining ideas from mathematical programming, we proposed a new approximation algorithm. Simulation shows that our algorithm beats previously known algorithms such as the subgradient algorithm and the hull approach.

References

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

[2] J. Beasley and N. Christofides, “An algorithm for the Resource Constrained Shortest Path Problem,” Networks, 19:379-394, 1989.

[3] Y. Bejerano, Y. Breitbart, A. Orda, R. Rastogi, and A. Sprintson, "Algorithms for computing QoS paths with restoration," IEEE Trans. of Networking, vol. 13, no. 3, 2005, pp. 648-661.

[4] D. P. Bertsekas, *Network optimization: continuous and discrete models*, Athena Scientific, Belmont, Massachusetts, 1998.

[5] D. P. Bertsekas and J. N. Tsitsiklis, *Introduction to linear optimization*, Athena Scientific, Belmont, Massachusetts, USA, 1997.

[6] D. Blokh and G. Gutin, “An approximation algorithm for combinatorial optimization problems with two parameters,” *Australasian Journal of Combinatorics*, vol. 14, 1996, pp.157-164.

[7] Stephen Boyd and Lieven Vandenberghe, *Convex Optimization*, Cambridge University Press, Cambridge, UK, 2003.

[8] A. Chakrabarti and G. Manimaran, "Reliability constrained routing in QoS networks," IEEE Trans. of Networking, vol. 13, no. 3, 2005, pp. 662-675.

[9] S. Chen and K. Nahrstedt, “On finding multi-constrained path,” in *Proc*. *ICC*, 1998, pp. 874-879.

[10] S. Chen and K. Nahrstedt, "An overview of Quality-of-Service routing for the next generation high-speed networks: problems and solutions," *IEEE Network Magazine*, 12(6), pp. 64-79, Nov. / Dec, 1998.

[11] Chvátal, *Linear programming*, W. H. Freeman and Company, New York, USA 1983.

[12] E. Crawley, R. Nair, B. Rajagopalan, and H. Sandick, "A Framework for QoS Based Routing in the Internet", Internet Draft, fttp://fttp.ietf.org/internet-drafts/draft-ietf-qosr-framework-02.txt, November 1997.

[13] M.R. Garey and D.S. Johnson, *Computers and Intractability*, Freeman, San Francisco, CA, USA, 1979.

[14] A. Goel, K. G. Ramakrishnan, D. Kataria and D. Logothetis, "Efficient computation of delay-sensitive routes from one source to all destinations," *IEEE INFOCOM*, 2001, pp. 854-858.

[15] R. Guerin, S. Kamat, A. Orda and T. Przygienda, "QoS Routing Mechanisms and OSPF Extensions", Internet Draft, March, 1997.

[16] R. Guerin and A. Orda, "QoS routing in networks with inaccurate information: theory and algorithms," *IEEE/ACM Trans. on Networking*, vol. 7, pp. 350-364, June 1999.

[17] G. Handler and I. Zang, "A dual algorithm for the constrained shortest path problem," *Networks* 10, 1980, pp. 293-310.

[18] R. Hassin, "Approximation schemes for the restricted shortest path problem," *Math. Of Oper. Res.*, 17(1), 1992, pp.36-42.

[19] Sung-Pi Hong, Sung-Jin Chung and Bum Hwan Park, "On 'Strongly' and Fully Approximation Schemes for Restricted Shortest Path Problem," Technical Report, Seoul National University (csj@optima.sun.ac.kr).

[20] O. Ibarra and C. Kim, "Fast Approximation Algorithms for the Knapsack and Sum of Subsets Problems," *Journal of the Association for Computing Machinery*, vol. 22, 463-468, October, 1975.

[21] A. Iwata, R. Izmailov, D.-S. Lee, B. Sengupta, G. Ramamurthy, and H. Suzuki, "ATM routing algorithms with multiple QoS requirements for multimedia internetworking," in *IEICE Trans. Commun. E79-B*, vol. 8, 1996, pp. 999–1006.

[22] J. M. Jaffe, "Algorithms for finding paths with multiple constraints," *Networks*, vol. 14, 1984, pp. 95–116.

[23] A. Jüttner, B. Szviatovszki, I. Mécs and Z. Rajkó, "Lagrange relaxation based method for the QoS routing problem," in *Proc. IEEE INFOCOM*-2001, pp. 859-868.

[24] Alpár Jüttner, "On Budgeted Optimization Problems," under review for SIAM Journal on Discrete Mathematics.

[25] A. Jüttner, "On resource constrained optimization problems," *4th Japanese-Hungarian Symposium on Discrete Mathematics and Its Applications*, June 3-6, 2005, Budapest, Hungary.

[26] D. Karger, R. Motwani and G. D. S. Ramkumar, "On approximating the longest path in a graph", *Algorithmica* 18, pp. 82-98, 1997.

[27] T. Korkmaz and M. Krunz, "A randomized algorithm for finding a path subject to multiple QoS requirements," *Comput. Networks*, vol. 36, 2001, pp. 251–268.

[28] T. Korkmaz and M. Krunz, "Bandwidth-delay constrained path selection under inaccurate state information," *IEEE/ACM Trans*. on *Networking*, June 2003, pp. 384-398.

[29] T. Korkmaz and M. Krunz, "Multi-constrained optimal path selection," in *Proc*. *IEEE INFOCOM* 2001, vol. 2, pp. 834-843

[30] Turgay Korkmaz, Marwan Krunz and Spyros Tragoudas, "An Efficient Algorithm for Finding a Path Subject to Two Additive Constraints," *Computer Communications Journal* 25(3):225-238, 2002.

[31] F. A. Kuipers, T. Korkmaz, M. Krunz, and P. Van Mieghem, "An overview of constraint-based path selection algorithms for QoS routing," *IEEE Commun. Mag.*, vol. 40, pp. 50–55, Dec. 2002.

[32] C.-L. Li, S. T. McCormick, and D. Simchi-Levi, "The complexity of finding two disjoint paths with min-max objective function," *Discrete Applied Math*. 26, 1990, pp. 105-115.

[33] G. Liu and K. G. Ramakrishnan, "A*Prune: An algorithm for finding K shortest paths subject to multiple constraints," in *Proc. IEEE INFOCOM*, vol. 2, 2001, pp. 743–749.

[34] D. H. Lorenz and A. Orda, "QoS routing in networks with uncertain parameters," *IEEE/ACM Trans. Networking*, vol. 6, Dec. 1998, pp. 768-778.

[35] D. Lorenz and D. Raz, "A simple efficient approximation scheme for the restricted shortest paths problem," *Oper. Res. Letters*, vol. 28, 2001, pp. 213–219.

[36] Dean H. Lorenz, Ariel Orda, Danny Raz and Yuvait Shavitt, "Efficient QoS Partition and Routing of Unicast and Mulicast," in *IWQOS*, June 2000.

[37] G. Luo, K. Huang, J. Wang, C. Hobbs and E. Munter, "Multi-QoS constraints based routing for IP and ATM networks," in *Proc. IEEE Workshop on QoS Support for Real-Time Internet Applications,* Vancouver Canada, June 1, 1999.

[38] N. Megiddo, "Combinatorial optimization with rational objective functions," *Math. of Oper. Res.*, vol. 4, no. 4, Nov. 1979, pp. 1-12.

[39] K. Mehlhorn and M. Ziegelmann, "Resource constrained shortest path," in *Proc.* 8*th European Symposium on Algorithms* (*ESA*2000), pp 326–337.

[40] P. Van Mieghem, H. De Neve, and F. Kuipers, "Hop-by-hop quality of service routing," *Comput. Networks*, vol. 37, no. 3–4, 2001, pp. 407–423.

[41] P. Van Mieghem and F. Kuipers*,* "Concepts of Exact QoS Routing Algorithms," *IEEE/ACM Trans*. *On Networking*, vol. 12, Oct. 2004, pp. 851-864.

[42] H. De Neve and P. Van Mieghem, "TAMCRA: A tunable accuracy multiple constraints routing algorithm," *Comput. Commun*, vol. 23, 2000, pp.667–679.

[43] A. Orda and A. Sprintson, "Efficient algorithms for computing disjoint QoS paths," in *Proc. IEEE INFOCOM*, 2004, pp. 727–738.

[44] C. R. Palmer and J. G. Steffan, "Generating network topologies that obey power laws," in *Proc. IEEE GLOBECOM*, 2000, pp.434–438.

[45] C. Phillips, "The network inhibition problem," in *Proc*. *of the* 25^{th} *Annual ACM Symposium on the Theory of Computing*, May, 1993, pp. 776–785.

[46] Private Network-Network Interface Specification Version 1.0 (PNNI 1.0) ATM Forum Technical Committee, 1996.

[47] T. Radzik, Fractional combinatorial optimization, in handbook of combinatorial optimization, Editors DingZhu Du and Panos Pardalos, vol. 1, Kluwer Academic Publishes, Dec. 1998.

[48] R. Ravindran, K. Thulasiraman, A. Das, K. Huang, G. Luo and G. Xue, "Quality of services routing: heuristics and approximation schemes with a comparative evaluation," *ISCAS*, 2002, pp. 775–778.

[49] Douglas S. Reeves and Hussein F. Salama, "A Distributed Algorithm for Delay-Constrained Unicast Routing," *IEEE/ACM Trans. on Networking*, vol. 8, no 2, April 2000, pp. 239-250.

[50] Sartaj Sahni, General Techniques for Combinatorial Approximation, *Oper. Res.* 25, 1977, pp. 920-936.

[51] A. Schrijver, *Theory of linear and integer programming*, John Wiley, New York, 1986.

[52] S. Shenkar, C. Patridge and R. Guering, "Specification of Guaranteed Quality of Service", INTERNET-RFC 2212, IETF, September 1997.

[53] J. L. Sobrinho, "Algebra and Algorithms for QoS Path Computation and Hop-by-Hop Routing in the Internet," *IEEE/ACM Trans. on Networking*, vol. 10, no. 4, Aug. 2002, pp. 541-550.

[54] J. W. Suurballe and R. E. Tarjan, "A quick method for finding shortest pairs of disjoint paths," *Networks*, vol. 14, 1984, pp. 325-336.

[55] K. Thulasiraman and M. N. Swamy, *Graphs*: *Theory and algorithms*, Wiley Interscience, New York, 1992.

[56] K. Thulasiraman, Y. Xiao, G. Xue, "Advances in QoS path(s) Selection problem", in *Proc*. *ISCAS* 2005.

[57] A. Warburton, Approximation of pareto optima in multiple-objective shortest path problems, *Operations Research*, 35:70-79,1987

[58] Z. Wang and J. Crowcroft, "Quality-of-Service routing for supporting multimedia applications," *IEEE JSACs*, vol.14, no.7, Sept. 1996, pp.1228–1234.

[59] B. M. Waxman, "Routing of multipoint connections," *IEEE J. Selected Areas in Comm*. 6(9), Dec. 1988, 1617–1622

[60] Y. Xiao, K. Thulasiraman and G. Xue, "Equivalence, unification and generality of two approaches to the constrained shortest path problem with extension," in *Proc*. *Allerton Conference on Control, Communication and Computing*, University of Illinois, Urbana-Champaign, Oct., 2003

[61] Y. Xiao, K. Thulasiraman and G. Xue, "The constrained shortest path problem: algorithmic approaches and an algebra study with generalization," to appear in *AKCE International Journal of Graphs and Combinatorics*.

[62] Y. Xiao, K. Thulasiraman and G. Xue, "Constrained Shortest Link-Disjoint Paths Selection: A Network Programming Based Approach," accepted for publication in *IEEE Transactions on Circuits & Systems* I: Regular Papers. A preliminary version appeared in *Proc*. *Allerton Conference on Control, Communication and Computing*, University of Illinois, Urbana-Champaign, Oct. 2004.

[63] Y. Xiao, K. Thulasiraman and G. Xue, "QoS Routing in Communication Networks: Approximation Algorithms Based on the Primal Simplex Method of Linear

Programming," Submitted to IEEE transactions on computers, revised version under review. A preliminary version appeared in *Proc. QShine* 2004, pp. 120–129.

[64] Y. Xiao, K. Thulasiraman and G. Xue, "Approximation and Heuristic Algorithms for Delay Constrained Path Selection under Inaccurate State Information," *QShine* 2004, pp. 130-137.

[65] Y. Xiao, K. Thulasiraman and G. Xue, "GEN-LARAC: A Generalized Approach to the Constrained Shortest Path Problem under Multiple Additive Constraints," Accepted by 16th Annual International Symposium on Algorithms and Computation, Dec., 2005, Sanya, Hainan, China.

[66] G. Xue, "Minimum-cost QoS multicast and unicast routing in communication networks," *IEEE Trans*. *On Commun*. Vol. 51, May 2003, pp. 817-827

[67] G. Xue, A. Sen and R. Banka, "Routing with many additive QoS constraints," In *Proc. ICC*' 03, pp. 223–227.

[68] X. Yuan, "Heuristic algorithms for multiconstrained quality-of-service routing," *IEEE/ACM Trans. Networking*, vol. 10, pp. 244–256, Apr. 2003

[69] M. Ziegelmann, "Constrained shortest paths and related problems," PhD thesis, Max-Planck-Institut für Informatik, 2001.

Appendix A. Waxman's Random Graphs

Waxman's random graphs have some of the characteristics of actual networks. There are two types of random graphs defined in [59]. In our simulation, we have adopted the second type of graphs generated as follows:

1) Given a node set, for each pair of nodes, a distance is chosen in (0, $L > 0$) from a uniform random distribution.
2) An edge is introduced between any pair of nodes u, v with a probability given as $\Pr(\{u, v\}) = \beta \exp\{\frac{-d(u,v)}{L\alpha}\}$, where α and β are parameters in the range (0, 1).
3) The cost and delay of each link are chosen in [1, $C > 1$] and [1, $D > 1$] from a uniform random distribution.

To guarantee that the generated networks are connected, we first link all the nodes as a ring and then proceed with the above procedure to add more edges. The same rule is also applied to the generation of power law graphs to be defined next.

Appendix B. Power Law Graphs

Power law graphs used in our simulation are generated as follows [44]:

1) Randomly assign to each node a number of out-degree credits. Here credits are attained using a random number generator that generates numbers that follow the exponential distribution $\beta x^{-\alpha}$, where α and β are parameters.
2) Place edges in the adjacency matrix for the graph such that every node obtains the assigned out-degree. The edge placement loop picks a random pair of nodes and assigns an edge without violating the out degree constraints. When an edge is placed between a pair of nodes, decrease the out-degree credits of the head node in the pair.
3) The cost and delay of each link are chosen in [1, $C > 0$] and [1, $D > 0$] from a uniform random distribution.