Continuous-Time Dynamic Shortest Path Algorithms

by

BRIAN C. DEAN

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degrees of

Bachelor of Science in Electrical Engineering and Computer Science
and Master of Engineering in Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 21, 1999
[June 1999]

Author ....
Department of Electrical Engineering and Computer Science
May 21, 1999

Certified by .... [illegible]
Ismail Chabini
Assistant Professor
Department of Civil and Environmental Engineering
Massachusetts Institute of Technology
Thesis Supervisor

Accepted by .... [illegible]
Arthur C. Smith
Chairman, Departmental Committee on Graduate Theses

Continuous-Time Dynamic Shortest Path Algorithms

by
Brian C. Dean

Submitted to the Department of Electrical Engineering and Computer Science
on May 21, 1999, in partial fulfillment of the requirements for the degrees of
Bachelor of Science in Electrical Engineering and Computer Science
and Master of Engineering in Computer Science.

Abstract

We consider the problem of computing shortest paths through a dynamic network – a network with time-varying characteristics, such as arc travel times and costs, which are known for all values of time. Many types of networks, most notably transportation networks, exhibit such predictable dynamic behavior over the course of time. Dynamic shortest path problems are currently solved in practice by algorithms which operate within a discrete-time framework. In this thesis, we introduce a new set of algorithms for computing shortest paths in continuous-time dynamic networks, and demonstrate for the first time in the literature the feasibility and the advantages of solving dynamic shortest path problems in continuous time. We assume that all time-dependent network data functions are given as piece-wise linear functions of time, a representation capable of easily modeling most common dynamic problems. Additionally, this form of representation and the solution algorithms developed in this thesis are well suited for many augmented static problems such as time-constrained minimum-cost shortest path problems and shortest path problems with time windows.

We discuss the classification, formulation, and mathematical properties of all common variants of the continuous-time dynamic shortest path problem. Two classes of solution algorithms are introduced, both of which are shown to solve all variants of the problem. In problems where arc travel time functions exhibit First-In-First-Out (FIFO) behavior, we show that these algorithms have polynomial running time; although the general problem is NP-hard, we argue that the average-case running time for many common problems should be quite reasonable. Computational results are given which support the theoretical analysis of these algorithms, and which provide a comparison with existing discrete-time algorithms; in most cases, continuous-time approaches are shown to be much more efficient, both in running time and storage requirements, than their discrete-time counterparts. Finally, in order to further reduce computation time, we introduce parallel algorithms, and hybrid continuous-discrete approximation algorithms which exploit favorable characteristics of algorithms from both domains.

Thesis Supervisor: Ismail Chabini
Title: Assistant Professor, Department of Civil and Environmental Engineering,
Massachusetts Institute of Technology

Acknowledgments

There are many people whom I would like to thank for their support and encouragement during the process of writing this thesis.

I wish to thank my thesis supervisor, Professor Ismail Chabini, for his numerous insightful comments which helped me improve the quality of the text, and in general for his motivation and support.

I am grateful to my friends and colleagues in the MIT Algorithms and Computation for Transportation Systems (ACTS) research group for their encouragement, friendship, and advice.

I received helpful editorial advice from my friends Boris Zbarsky and Nora Szasz, and my discussions with both Ryan Rifkin and my sister Sarah were instrumental in the development of proofs of polynomial bounds on running time for several algorithms.

Finally, I would like to thank my girlfriend Delphine for her patience and encouragement during some very busy times, and for providing me with crucial insight into the most difficult proof in entire text. I dedicate this work to her.

Contents

Chapter 1

Introduction

Almost all networks have characteristics which vary with time. Many networks such as transportation networks and data communication networks experience predictable rising and falling trends of utilization over the course of a day, typically peaking at some sort of "rush-hour". Algorithms which account for the dynamic nature of such networks will be able to better optimize their performance; however, development of these algorithms is challenging due to the added complexity introduced by the network dynamics, and by the high demands they typically place on computational resources.

In the literature, one encounters several different definitions of a what constitutes a *dynamic network.* In general, a dynamic network is a network whose characteristics change over time. These networks typically fall into one of two main categories. In the first category, the future characteristics of the network are not known in advance. Algorithms which optimize the performance of this type of network are sometimes called re-optimization algorithms, since they must react to changes in network conditions as they occur by making small changes to an existing optimal solution so that it remains optimal under new conditions. These algorithms often closely resemble static network optimization algorithms, since they are intended to solve a succession of closely-related static problems. In the second category, if one knows in advance the predicted future conditions of the network, it is possible to optimize the performance of the network over all time in a manner which is consistent with the anticipated future characteristics of the network. Algorithms designed for this second class of dynamic networks typically represent a further departure from the familiar realm of static network optimization algorithms. In this thesis, we focus exclusively on this second class of dynamic networks, for which time-varying network characteristics are known for all time.

1.1 Overview of Research

This thesis considers the fundamental problem of computing shortest paths through a dynamic network. In some networks, most notably electronic networks, traditional static algorithms are often sufficient for such a task, since network characteristics change very little over the duration of transmission of any particular packet, so that the network is essentially static for the lifetime of a packet. However, in other network applications, primarily in the field of transportation, network characteristics may change significantly during the course of travel of a commodity over the network, calling for more sophisticated, truly dynamic algorithms. To date, all practical algorithms which solve dynamic shortest path problems work by globally discretizing time into small increments. These discrete-time algorithms have been well-studied in the past, and solution algorithms with optimal running time exist for all common variants of the problem. Unfortunately, there are several fundamental drawbacks inherent in all discrete-time approaches, such as very high memory and processor time requirements, which encourage us to search for more efficient methodologies. Although the literature contains some theoretical results for continuous-time problems, to date no practical algorithms have been shown to efficiently solve realistic dynamic shortest path problems in continuous-time. The goal of this thesis is to demonstrate not only the feasibility of continuous-time methods, but also to show that the performance of new continuous-time approaches is in many cases superior to that of existing discrete-time approaches. In doing so, we develop a new set of highly-efficient serial and parallel algorithms for solving all common variants of the dynamic shortest path problem in continuous-time, and we provide an extensive theoretical and computational study of these algorithms.

The representation of time-dependent data is an issue which deserves careful consideration in any continuous-time model. In this thesis, we impose the simplifying assumption that all time-dependent network data functions are specified as piece-wise linear functions of time. Our solution algorithms and their analyses will depend strongly on this assumption, and we further argue that it is this simplifying assumption which makes the continuous-time dynamic shortest path problem truly computationally feasible in practice. First, piece-wise linear functions are easy to store and manipulate, and they can easily approximate most "reasonable" time-dependent functions. Additionally, if arc travel time functions are piece-wise linear, then path travel time functions will also be piece-wise linear, since we shall see that path travel time functions are given essentially by a composition of the travel time functions of the arcs along a given path. Almost any other choice of representation for arc travel time functions results in extremely unwieldy path travel time functions. For example, the use of quadratic functions to describe arc travel times results in arbitrarily-high degree polynomial path travel time functions, which are cumbersome to store and evaluate. Finally, piece-wise linear functions make sense from a modeling perspective: in order to solve for optimal dynamic shortest paths, we must know the anticipated future characteristics of the network, and often these are not known with enough precision and certainty to warrant a more complicated form of representation. It is common to anticipate either increasing or decreasing trends in network data, or sharp discontinuities at future points in time when we know for example that travel along an arc may be blocked for a certain duration of time. Piece-wise linear functions are appropriate for modeling this type of simple network behavior. Certain variants of static shortest path problems, such as time-constrained minimum-cost problems and problems involving time windows are also easy to represent within this framework, as they involve arc travel times and costs which are piece-wise constant functions of time.

1.2 Thesis Outline

The material in this thesis is organized as follows:

In Chapter 2, we review previous results from the literature. We discuss key concepts underlying discrete-time dynamic shortest path algorithms, so as to compare these approaches with the new continuous-time algorithms developed in later chapters. Previous theoretical and algorithmic results related to the continuous-time dynamic shortest path problem are also examined.

Chapter 3 introduces the notation associated with continuous-time models, and classifies the different variants of the continuous-time dynamic shortest path problem. We will generally categorize problems by objective, based on whether we desire minimum-time or more general minimum-cost paths, and we will consider three fundamental problem variants based on configuration of source and destination nodes and times: the one-to-all problem for one departure time, the one-to-all problem for all departure times, and the all-to-one problem for all departure times.

In Chapter 4, we provide a mathematical formulation of these problem variants, and discuss important mathematical properties of these problems and their solutions. Optimality conditions are given, and it is proven that a solution of finite size always exists for all problem variants. We devote considerable attention to the discussion of properties of networks which exhibit First-In-First-Out, or FIFO behavior (defined later), and we will show that the minimum-time one-to-all problem for all departure times is equivalence through symmetry to the minimum-time all-to-one problem in a FIFO network.

Chapter 5 is the first of three core chapters which contain the algorithmic developments of the thesis. Within this chapter, we focus on fundamental results which apply to FIFO networks. We first describe methods of adapting static shortest path algorithms to compute minimum-time paths from a source node and a single departure time to all other nodes in a FIFO dynamic network – this is an important well-established result in the literature. Additionally, we present a new technique for developing parallel adaptations of any algorithm which computes all-to-one minimum-time paths or one-to-all minimum-time paths for all departure times in a FIFO network, in either continuous or discrete time.

The remaining algorithmic chapters describe two broad classes of methods which can be applied to solve any variant of the continuous-time dynamic shortest path problem, each with associated advantages and disadvantages. In Chapter 6 we introduce a new class of solution algorithms, which we call *chronological scan* algorithms. We first present a simplified algorithm which addresses a simple, yet common problem variant: the minimum-time all-to-one dynamic shortest path problem in networks with FIFO behavior. The simplified algorithm is explained first because it provides a gradual introduction to the methodology of chronological scan algorithms, and because it can be shown to solve this particular variant of the problem in polynomial time. We then present general extended chronological scan solution algorithms which solve all variants of the dynamic shortest path problem, and hybrid continuous-discrete adaptations of these algorithms which compute approximate solutions and require less computation time.

In Chapter 7, we discuss a second class of solution methods, which we call *label-correcting* algorithms, based on previous theoretical results obtained by Orda and Rom [16]. In FIFO networks, these algorithms are shown to compute minimum-time shortest paths in polynomial time, although they have a higher theoretical computational complexity than the corresponding chronological scan algorithms.

Chapter 8 contains the results of extensive computational testing. The two primary classes of algorithms developed within this thesis, chronological scan algorithms and label-correcting algorithms, are evaluated for a wide range of both randomly generated and real networks, and shown to be roughly equivalent in performance. We also compare the performance of these techniques with that of the best existing discrete-time dynamic shortest path algorithms, and show that in many cases the continuous-time approaches are much more efficient, both in terms of processing time and space.

Finally, we give concluding remarks in Chapter 9, including a summary of the results developed within this thesis, and suggestions for future directions of research.

Chapter 2

Literature Review

In this chapter, we briefly outline previous results from the literature so as to frame our new results within a historical perspective. Well-known previous results for computing shortest paths within FIFO networks are reviewed. We then discuss methods used to address discrete-time problems; these methods will be compared with new continuous-time approaches in later chapters. Finally, existing results which relate to the continuous-time case are reviewed.

2.1 Algorithms for FIFO Networks

It is common for many dynamic networks, for example transportation networks, to satisfy the well-known First-In-First-Out, or FIFO property. This property is mathematically described in Section 3.2; intuitively, it stipulates that commodities exit from an arc in the same order as they entered, so that delaying one's departure along any path never results in an earlier arrival at an intended destination. One of the most celebrated results in the literature on this subject is the fact that the minimum-time dynamic shortest paths departing from a single source node at a single departure time may be computed in a FIFO network by a slightly modified version of any static label-setting or label-correcting shortest path algorithm, where the running time of the modified algorithm is equal to that of its static counterpart. This result was initially proposed by Dreyfus [10] in 1969, and later shown to hold only in FIFO networks by Ahn and Shin [1] in 1991, and Kaufman and Smith [14] in 1993. As many static shortest path algorithms are valid for both integer-valued and real-valued arc travel times, modified static algorithms which solve the minimum-time dynamic shortest path problem from a single source node and departure time can be adopted for dynamic network models in either discrete time or continuous time. Such modified static algorithms are discussed in Section 5.1.

As the so-called "minimum-time one-to-all for one departure time" dynamic shortest path problem in a FIFO network is well-solved, this thesis focuses on the development of algorithms for all other common problem variants of the continuous-time dynamic shortest path problem. For the case of FIFO networks, we will discuss methods for computing shortest paths from all nodes and all departure times to a single destination node, and shortest paths from a source node to all destination nodes for all possible departure times, rather than a single departure time. Furthermore, we discuss solution algorithms for problem variants in networks which may not necessarily satisfy the FIFO property, and algorithms which determine paths which minimize a more general travel cost rather than just the travel time.

2.2 Discrete-Time Models and Algorithms

The dynamic shortest path problem was initially proposed by Cooke and Halsey [7] in 1966, who formulated the problem in discrete time and provided a solution algorithm. Since then, several authors have proposed alternative solution algorithms for some variants of the discrete-time dynamic shortest path problem. In 1997, Chabini [3] proposed a theoretically optimal algorithm for solving the basic all-to-one dynamic shortest path problem, and Dean [9] developed extensions of this approach to solve a more general class of problems with waiting constraints at nodes. Pallottino and Scutella (1998) provide additional insight into efficient solution methods for the all-to-one and one-to-all problems. Finally, Chabini and Dean (1999) present a detailed general framework for modeling and solving all common variants of the discrete-time problem, along with a complete set of solution algorithms with provably-optimal running time.

We proceed to discuss the methods used to model and solve discrete-time problems. Consider a directed network $G = (N, A)$ with $n = |N|$ nodes and $m = |A|$ arcs. Denote by $d_{ij}[t]$ and $c_{ij}[t]$ the respective travel time and travel cost functions of an arc $(i, j) \in A$, so that $d_{ij}[t]$ and $c_{ij}[t]$ give the respective travel time and cost incurred by travel along (i, j) if one departs from node i at time t. For clarity, we use square brackets to indicate discrete functions of a discrete parameter. If we want to minimize only travel time rather than a more general travel cost, then we let $c_{ij}[t] = d_{ij}[t]$. In order to ensure the representation of these time-dependent functions within a finite amount of memory, we assume they are defined over a finite window of time $t \in \{0, 1, 2, ..., T\}$, where T is determined by the total duration of time interval under consideration and the by granularity of the discretization we apply to time in this interval. Beyond time T, we can either disallow travel, or assume that network conditions remain static and equal to the value they assumed at time T.

It is easy to visualize and solve a discrete-time dynamic shortest path problem by constructing a *time-expanded network*, which is a static network that encapsulates the dynamic characteristics of G. The time-expanded network is formed by making a separate copy of G for every feasible value of time – its nodes are of the form (i, t), where $i \in N$ and $t \in \{0, 1, 2, ..., T\}$, and its arcs are of the form $((i, t), (j, min\{T, t + d_{ij}[t]\}))$, where $(i, j) \in A$ and $t \in \{0, 1, 2, ..., T\}$, and where the travel cost of every such arc is given by $c_{ij}[t]$. Since paths through the time-expanded network correspond exactly in cost and structure to paths through the dynamic network G, we can reduce the problem of computing minimum-cost dynamic paths through G to the equivalent problem of computing minimum-cost paths through the static time-expanded network, a problem which we may solve by applying any of a host of well-known static shortest path algorithms to the time-expanded network. However, since the time-expanded network has a special structure, specialized static shortest path algorithms may be employed to achieve a more efficient running time. This special structure is apparent from the construction of the network: there are multiple levels corresponding to the different copies of the network G for each value of time, and no arc points from a node of a later time-level to a node of an earlier time-level. The time-expanded network is therefore "multi-staged" – it is acyclic with the exception of arcs potentially connecting nodes within the same level of time.

The running time of solution algorithms for the discrete-time problem depends on the problem type, but worst case asymptotic running time is almost always pseudo-polynomial in T (the only significant exception is the aforementioned minimum-time one-to-all problem in a FIFO network, which is solvable in the same amount of time as a static shortest path problem). A discrete-time solution algorithm will typically need to examine a significant portion of the nodes and arcs in the time-expanded network, leading to a worst-case lower bound of $\Omega(nT + mT)$ on running time. This lower bound is also often close to the average-case running time of most discrete-time solution algorithms. Since T usually takes values in the hundreds or thousands, discrete-time approaches can have very high computational demands. Furthermore, the amount of memory required by these algorithms is typically $\Omega(nT)$ in the worst (and often average) case, as one needs to assign cost labels to the nodes of the time-expanded network in order to compute shortest paths through this network.

Although discrete-time solution algorithms are straightforward to design and implement, they have many drawbacks, including primarily their high storage and processing time requirements. Additionally, the high running time of a discrete-time algorithm often doesn't reflect the true complexity of the dynamics of the underlying problem. For example, consider a problem in which the travel cost of only one arc changes at exactly one point in time. In fact, this is a common scenario, since time-constrained minimum-cost static shortest path problems may be modeled as dynamic problems in which one arc changes its travel cost (to infinity) at exactly one point in time. In order to obtain a solution to this problem using discrete-time methods, one must often do as much work as in a fully-dynamic problem in which all network data is constantly changing – simplicity in the time-dependent network data very rarely translates to a lower running time. Also, in most dynamic networks, a high percentage of the network is actually static. For example, in a transportation network, major roads exhibit dynamic characteristics due to changes in congestion over the course of a day, but lesser-traveled minor roads, comprising a large percentage of the network, are for the most part unaffected by congestion delays and remain essentially static. Discrete-time algorithms typically offer no corresponding decrease in running time as a reward for this simplicity, as their performance is determined for the most part only by n, m, and T, and not by the structure of the time-dependent network data. In contrast, we will see in forthcoming chapters that the performance of the new continuous-time algorithms developed in this thesis will be determined primarily by the complexity of the network data, so that their running time is more closely correlated with the true complexity of a problem, and so that they may solve simple dynamic problems with much greater efficiency.

2.3 Previous Results for Continuous-Time Problems

The only significant previous algorithmic results in the literature related to the continuous-time dynamic shortest path problem are due to Halpern [12] in 1977, and Orda and Rom [15], [16] in 1990 and 1991. These papers consider the problem variant of computing optimal paths through a continuous-time dynamic network originating at a single source node, where waiting is allowed within certain fixed intervals of time at each node. Halpern considers only the case of computing paths of minimum time, while Orda and Rom consider the more general case of computing paths of minimum cost. They each propose solution algorithms, and argue their correctness and termination within a finite amount of time. The work of Orda and Rom is much more comprehensive than that of Halpern, and identifies problems for which the algorithm of Halpern will fail to produce an optimal solution within a finite amount of time.

The primary shortcoming of the algorithms of Halpern and those of Orda and Rom is that they are presented as theoretical, rather than as practical results. These algorithms rely on operations on general continuous-time functions as their fundamental operations; hence, these algorithms as stated can be prohibitively complicated to implement, let alone implement efficiently. To date no authors have reported on any computational evaluation of these algorithms, except for one case in which the algorithm of Halpern was evaluated for a static network with time windows (feasible regions of time during which one's visit to a node must fall).

There are two primary approaches for creating a feasible implementation of the algorithms of Halpern and of Orda and Rom: either one must discretize time, in which case one enters the discrete-time domain, in which superior algorithms exist, or alternatively one may impose a simplifying piece-wise linearity assumption on the continuous-time network data as is done in this thesis. Under piece-wise linearity assumptions, the methodology underlying the algorithms of Halpern and of Orda and Rom forms the basis for the second of two classes of solution algorithms discussed in this thesis, which we call *label-correcting* algorithms. In Chapter 7, we describe this broad class of solution algorithms, including the special cases developed by these authors, and provide a thorough theoretical analysis of their performance. We show that these algorithms will run in polynomial time in when used to compute minimum-time paths through FIFO networks. The other class of solution algorithms, which we call *chronological scan* algorithms, is presented in Chapter 6; chronological scan algorithms are shown to have stronger worst-case theoretical running times than label-correcting algorithms.

Ioachim, Gelinas, Soumis, and Desrosiers [13] address the question of computing optimal static shortest paths through an acyclic network in the presence of time windows. Waiting at nodes is allowed during these windows, and accrues a cost which is linear in the amount of time spent waiting. The authors devise a solution algorithm which is relevant to the work in this thesis because it produces piece-wise linear path travel costs. This result may be seen as a very special case of the general framework developed in this thesis, since it is restricted in focus only to static, acyclic networks. The continuous-time dynamic models developed in this thesis provide a more general approach for representing and solving a much wider class of problems, both dynamic and static, involving time windows.

Chapter 3

Continuous-Time Models

In this chapter, we discuss notation for continuous-time models, and describe the different variants of continuous-time dynamic shortest path problems.

3.1 Piece-Wise Linear Function Notation

The results in this thesis depend strongly on the simplifying assumption that network data functions are given as piece-wise linear functions of time. We assume that all piece-wise linear network data functions have a finite number of pieces, and by convention we treat all points on the boundary between two linear pieces of a function as belonging to the piece left of the boundary, unless the boundary point is itself a linear piece of zero extent. We assume, however, that all network data functions given as input to a dynamic shortest path algorithm will contain only linear pieces of strictly positive extent.

Since there may be several piece-wise linear functions involved with the specification of a single dynamic shortest path problem, we adopt the following general notation for a piece-wise linear function *f(t)*:

$P(f)$	:	Number of pieces into which f is divided.
$B(f, k)$	:	The right boundary of the k^{th} piece of f. By definition, let $B(f, 0) = -\infty$ and $B(f, P(f)) = +\infty$.
$\alpha(f, k)$	:	Linear coefficient of the k^{th} linear piece of f.
$\beta(f, k)$	:	Constant term in the k^{th} linear piece of f.

We may therefore specify an arbitrary piece-wise linear function *f(t)* as follows.

$$f(t) = \begin{cases} f^{(1)}(t) & \text{if } -\infty < t \le B(f,1) \\ f^{(2)}(t) & \text{if } B(f,1) < t \le B(f,2) \\ f^{(3)}(t) & \text{if } B(f,2) < t \le B(f,3) \\ \vdots & \vdots \\ f^{(P(f))}(t) & \text{if } B(f,P(f)-1) < t < +\infty \end{cases} \tag{3.1}$$

$$f^{(k)}(t) = \alpha(f,k)t + \beta(f,k) \tag{3.2}$$

If some time t marks the boundary between two linear pieces of a function, we say that this function *improves* at time t if either it is discontinuous and drops to a lower value after time t, or if it is continuous and its derivative drops to a lower value after time t. Likewise, we say that a function *worsens* at a boundary time t if either it is discontinuous and jumps to a higher value after time t, or if it is continuous and its derivative jumps to a higher value after time t. This terminology is used because in this thesis we will seek the minimum value of a piece-wise linear objective function, and a decrease of this function or its derivative will correspond to an improvement in the objective. It is possible for a piece-wise linear function to both improve and worsen at a boundary point, if both the function and its derivative are discontinuous at that point. For example, the function could improve by jumping to a lower value at time t and simultaneously improve by its derivative jumping to a higher value at time t. Finally, we say that a function is *simple* over a region if it contains no piece-wise boundaries within that region.

In all of our models, there is no particular significance attached to the time $t = 0$, or to times which are positive. We will always consider functions of time to be defined over all real values of time, and we will compute solutions which represent network conditions over all values of time. The choice of origin in time can therefore be arbitrary.

3.2 Network Data Notation

Let $G = (N, A)$ be a directed network with $n = |N|$ nodes and $m = |A|$ arcs. Let A_i denote the set $\{j \mid (i, j) \in A\}$ of nodes after node i, and let $B_j = \{i \mid (i, j) \in A\}$ denote the set of nodes before node j. For each arc $(i, j) \in A$, we denote by $d_{ij}(t)$ the travel time of a commodity entering the arc at time t. It is assumed that all arc travel time functions are piece-wise linear, finite-valued, and greater than some arbitrarily-small positive value. On occasion, instead of arc travel time functions, it will be more convenient to work with arc arrival time functions, defined as $a_{ij}(t) = t + d_{ij}(t)$. We define the quantity P^* to be the total number of linear pieces present among all arc travel time functions:

$$P^* = \sum_{(i,j)\in A} P(d_{ij}) . \qquad (3.3)$$

We denote by the piece-wise linear function $c_{ij}(t)$ the general cost of traveling along an arc $(i, j) \in A$, departing at time t. As time approaches $+\infty$, we assume that all travel time and travel cost functions either increase linearly or remain static and non-negative; otherwise, least-cost paths of unbounded duration may occur. For the same reason, it is also assumed that the initial linear pieces of each arc travel time and arc cost function are constant (and also non-negative, in the case of travel cost); that is, there must be some point in time before which the network is entirely static. This assumption is generally not restrictive in practice, because we are typically only interested in computing optimal paths within the region of time which extends from the present forward.

We say that a function $f(t)$ satisfies the *First-In-First-Out*, or *FIFO property* if the function $g(t) = t + f(t)$ is non-decreasing. A piece-wise linear function $f(t)$ will satisfy the FIFO property if and only if $\alpha(f, k) \geq -1$ for every $k \in \{1, 2, \ldots, P(f)\}$ and there are no discontinuities where at which $f(t)$ drops to a lower value. We describe an arc $(i, j) \in A$ as a *FIFO arc* if $d_{ij}(t)$ satisfies the FIFO property, or equivalently if $a_{ij}(t)$ is non-decreasing. If all arcs $(i, j) \in A$ are FIFO arcs, we say that G is a *FIFO network.* Commodities will exit from a FIFO arc in the same order as they entered, and commodities traveling along the same path in a FIFO network will exit from the path in the same order as they entered. As we shall see, this special characteristic enables the application of efficient techniques to compute minimum-time paths in FIFO networks. Finally, if $g(t)$ represents the "arrival time" function $t + f(t)$ of a FIFO travel time function f, we define the *FIFO inverse* of $g(t)$ as:

$$g^{-1}(t) = \max_{\{\tau | g(\tau) \leq t\}} \{\tau\}. \tag{3.4}$$

For any arc $(i, j) \in A$ in a FIFO network, the arc arrival time inverse function $a_{ij}^{-1}(t)$ gives the latest possible departure time along the arc for which arrival occurs at or before time t. In any FIFO network, arc arrival time inverse functions will always exist, and will themselves be non-decreasing functions of time; we therefore can define the inverse of the arc travel time function $d_{ij}(t)$ of an arc as $d_{ij}^{-1}(t) = a_{ij}^{-1}(t) - t$. It can be easily shown that each $d_{ij}^{-1}(t)$ function will satisfy the FIFO property and will have the same form as the original $d_{ij}(t)$ function – it will be piece-wise linear with a finite number of pieces (no more than twice as many pieces as the original function), where each piece spans a duration of time of strictly positive extent. Taking the FIFO inverse of an arrival time function twice yields, as would be expected, the original function.

Depending on the situation being modeled, it may be permissible to wait at some locations in the network during travel along a path. In order to model a general waiting policy in a dynamic network, we must specify two time-dependent quantities: bounds on the length of time one may wait at each node, and the costs of waiting for different lengths of time at each node. Let $ubw_i(t)$ be a non-negative-valued piece-wise linear function which denotes the upper bound on the amount of time one may wait at node i following an arrival at time t. We assume that these functions satisfy the FIFO property, because it is sensible for the latest time until one may wait at a node, $t + ubw_i(t)$, to be a non-decreasing function of t. It is not necessary to specify lower bounds on waiting times, since any amount of required waiting time at a node may be incorporated directly into the arc travel time functions of the arcs directed out of that node. Finally, for each node $i \in N$, we denote by $WT_i(t)$ the set of feasible waiting times $\{\tau \mid 0 \leq \tau \leq ubw_i(t)\}$ if waiting begins at time t.

In general, the cost of waiting at a node is a function of two parameters: the starting time and the duration of the interval of waiting. For simplicity, however, we focus in this thesis only on waiting costs which adhere to a simpler structure: we say waiting costs are *memoryless* if the cost of waiting beyond a particular time is not dependent on the amount of waiting which has occurred up until that time. In this case, one can write the cost of waiting at node i from time t until time $t + \tau$ as the difference between two cumulative waiting costs, $cwc_i(t + \tau) - cwc_i(t)$, where

$$cwc_i(t) = \int_0^t w_i(t)dt\,, \qquad (3.5)$$

and where $w_i(t)$ gives the unit-time cost of waiting density function of node i. We assume that the $w_i(t)$ functions are piece-wise constant and non-negative, so that the cumulative waiting cost functions $cwc_i(t)$ will be piece-wise linear, continuous, and non-decreasing, and so the cost of waiting for any interval of time will always be non-negative.

Memoryless waiting costs are very common in dynamic network models. If the objective of a dynamic shortest path problem is to find paths of minimum travel time, then waiting costs will be memoryless because they will satisfy $cwc_i(t) = t$. As a final restriction, if waiting is allowed anywhere in the network, and if G is not a FIFO network or if minimum-cost paths are desired, then we require continuity of all network data functions. If this restriction is not met, then an optimal solution may not exist. An example of such a problem is a single arc network, where unbounded waiting is allowed at no cost at the source, and where the travel cost of the single arc (i, j) is given by:

$$c_{ij}(t) = \begin{cases} 1 & \text{if } t \in (-\infty, 0] \\ t & \text{if } t \in (0, +\infty) \end{cases} \tag{3.6}$$

In this case, there is no optimal solution. In practice, the continuity restriction is not troublesome, since any discontinuity in a piece-wise linear function may be replaced by an additional linear piece with an arbitrarily steep slope. For problems in which waiting is prohibited, continuity of network data functions is not required.

3.3 Description of Problem Variants

Dynamic shortest path problems may be categorized into different variants based on the criteria used to select optimal paths and on the desired configuration of source and destination nodes and times. As an objective, it is often the case that we wish to find paths of minimum travel time; in this case, $c_{ij}(t) = d_{ij}(t)$, and we call the problem a *minimum-time* dynamic shortest path problem. We refer to the more general problem of finding paths of least cost a *minimum-cost* problem. Optimal paths through a dynamic network may contain cycles, depending on the network data. In FIFO networks, though, we shall see that minimum-time paths will always be acyclic.

As with static shortest path problems, the fundamental dynamic shortest path problems involve the computation of optimal paths between all nodes and either a single source or single destination node. The addition of a time dimension, however, complicates the description of dynamic problems since a "source" now comprises both a source node and a set of feasible departure times from that node, and a "destination" consists of a node along with the set of feasible arrival times at that node. Additionally, there is a greater distinction in dynamic problems between computing optimal paths emanating from a source node and computing optimal paths into a destination node. In the realm of static networks, these two problems are made equivalent by reversing all arcs in the network; in dynamic networks, the problems are still related but more remotely so, as there is often no trivial way to transform one to the other. In FIFO networks, however, there is greater degree of symmetry between these two types of problems. We will discuss methods of transforming between symmetric problems in FIFO networks in Section 4.4.

In this thesis, we consider two fundamental variants of the dynamic shortest path problem, which we call the *all-to-one* and *one-to-all* problems. The all-to-one problem involves the computation of optimal paths from all nodes and all departure times to a single destination node (and an associated set of feasible arrival times at that destination node). Symmetrically, the one-to-all problem entails the computation of optimal paths from a single source node (and a set of feasible departure times) to all other nodes. There are two common flavors of the one-to-all problem. The most common, which we simply call the one-to-all problem, involves computing a single shortest path from the source node to every other node, where departure time from the source is possibly constrained. The other variant, which we call the *one-to-all problem for all departure times* involves computing separate shortest paths from the source to every other node for every feasible departure time. An important result, which we discuss in Section 4.4, is the fact that in a FIFO network, the minimum-time all-to-one problem is computationally equivalent to the minimum-time one-to-all problem for all departure times.

The all-to-one problem is of benefit primarily to applications which control and optimize the system-wide performance of a dynamic network, as its solution allows for commodities located anywhere in time and space to navigate to a destination along an optimal route. The one-to-all problem, conversely, is well-suited for applications which compute optimal routes from the viewpoint of individual commodities traveling through a dynamic network from specific starting locations. In this thesis we will develop solution methods for both problems.

3.4 Solution Characterization

For the all-to-one problem, we designate a node $d \in N$ as the destination node. We can then characterize a solution to the all-to-one problem by a set of functions which describe the cost and topology of optimal paths to d from all nodes and departure times. The following functions will constitute the set of decision variables used when computing an optimal solution to the all-to-one problem:

$C_i^{(w)}(t)$	:	Cost of an optimal path to d departing from node i at time t, where waiting is allowed at node i before departure.
$C_i^{(nw)}(t)$	:	Cost of an optimal path to d departing from node i at time t, where no waiting is allowed at node i before departure.
$N_i(t)$	:	Next node to visit along an optimal path to d from node i, time t.
$W_i(t)$	:	Amount of time to wait at node i, starting at time t, before departing for node $N_i(t)$ on an optimal path to d.

Based on the information provided by these functions, it is a simple matter to trace an optimal route from any starting node and departure time to the destination, and to determine the cost of this route.

For the one-to-all problem, we designate one node $s \in N$ as a source node. We will use an equivalent but symmetric form of notation as in the all-to-one problem in order to describe a solution to the one-to-all problem. Specifically, we characterize a solution by a set of functions which specify the structure and cost of optimal paths from s to all nodes and all arrival times:

$C_i^{(w)}(t)$	:	Cost of an optimal path from s to node i and time t, where waiting is allowed at node i after arrival and up until time t.
$C_i^{(nw)}(t)$	:	Cost of an optimal path from s arriving at node i at exactly time t.
$PN_i(t)$	:	Preceding node along an optimal path from s to node i and time t.
$PT_i(t)$	:	Departure time from the preceding node $PN_i(t)$ along an optimal path from s to node i and time t.
$W_i(t)$	:	Amount of waiting time to spend after arrival at node i up to time t on an optimal path from s.

Specification of both the previous node and previous departure time functions is necessary in order to trace a path from any arbitrary destination node and arrival time back to the source. It is almost always the case that in the one-to-all problem, we are concerned with finding the best path to each node $i \in N$ irrespective of arrival time. In this case, we can therefore specify for each node $i \in N$ the arrival time opt_i at that node which yields the best possible path travel cost, where

$$opt_i = \arg\min_t \{C_i^{(w)}(t)\} \qquad \forall i \in N. \qquad (3.7)$$

Given any destination node $i \in N$, one may use the preceding solution information to easily determine the cost and structure of an optimal path from s to i.

Under the assumption of piece-wise linear network data functions, we shall see that the path travel cost functions $C_i^{(w)}(t)$ and $C_i^{(nw)}(t)$, the waiting time functions $W_i(t)$, and the previous node departure time functions $PT_i(t)$ which comprise the solution to the one-to-all and all-to-one problems will also be piece-wise linear. To describe the piece-wise linear behavior of these solution functions, we define a *linear node interval*, or *LNI*, to be a 3-tuple (i, t_a, t_b), where $i \in N$, and where $(t_a, t_b]$ is an interval of time of maximal possible extent during which $N_i(t)$ (or $PN_i(t)$) does not change with time, and during which the remaining solution functions change as simple linear functions of time. For the all-to-one problem and for all problems in FIFO networks, we shall see that LNIs will always span a duration of time of strictly positive extent. However, LNIs comprising the solution to a one-to-all problem may in some cases span only a single point in time – we will call such LNIs *singular*. The LNI (i, t_a, t_b) chronologically preceding a singular LNI (i, t_b, t_b) at some node $i \in N$ is taken to represent the interval of time (t_a, t_b) which contains neither of its two endpoints.

For each node, it will be possible to partition time so as to divide the piece-wise behavior of departures from that node into a finite number of LNIs. The output of an all-to-one or one-to-all dynamic shortest path algorithm will therefore consist of an enumeration of the complete set of LNIs for every node $i \in N$, along with a specification of the solution functions during each LNI. During the time extent of a particular LNI at some node i, we say that node i exhibits *linear departure behavior*, as arrivals or departures to or from the node during this interval follow the same incoming or outgoing arc experience an optimal path cost which changes as a simple linear function of time.

Chapter 4

Problem Formulation and Analysis

This chapter contains a mathematical formulation and optimality conditions for the all-to-one and one-to-all continuous-time dynamic shortest path problems. Following this, we establish some of the fundamental mathematical properties of solutions to these problems, and we discuss several key properties of FIFO networks. We show that a solution always exists for all variants of the dynamic shortest path problem, and that this solution always consists of a finite number of linear node intervals. In the case of a FIFO network, we will develop stronger bounds on the number of LNIs comprising the solution to a minimum-time path problem. Finally, we discuss symmetries between different variants of the minimum-time dynamic shortest path problem in a FIFO network – we will show that the minimum-time all-to-one problem and the minimum-time one-to-all problem for all departure times are computationally equivalent, so it will be necessary in future algorithmic chapters only to develop solution algorithms for one of these two problems.

4.1 Mathematical Formulation

In order to simplify our formulation of the all-to-one problem, we wish for the destination node d to appear at the end of all optimal paths, but not as an intermediate node along these paths. For minimum-time problems, this is not an issue. In minimum-cost problems, however, we enforce this desired behavior by replacing node d with the designation d' and by adding a new artificial destination node d along with a zero-cost arc (d', d). We assume that arrivals to d at any point in time are allowed; if there are restrictions on the set of feasible arrival times at d, we may model these by setting the time-dependent travel cost of the arc (d', d) so it incurs infinite travel cost at prohibited arrival times. In general, one may only constrain the arrival time for minimum-cost or non-FIFO minimum-time problems, since it is usually not possible to add such a constraint while satisfying the FIFO property.

The optimality conditions for the general minimum-cost all-to-one dynamic shortest path problem are as follows. A proof of necessity and sufficiency of these optimality conditions appears at the end of this section. In order to ensure optimality, the solution functions $C_i^{(w)}(t)$ and $C_i^{(nw)}(t)$ must satisfy:

$$C_i^{(w)}(t) = \min_{\tau \in WT_i(t)} \{cwc_i(t+\tau) - cwc_i(t) + C_i^{(nw)}(t+\tau)\} \qquad \forall i \in N, \qquad (4.1)$$

$$C_i^{(nw)}(t) = \begin{cases} 0 & \text{if } i = d \\ \min\limits_{j \in A_i}\{c_{ij}(t) + C_j^{(w)}(t + d_{ij}(t))\} & \text{if } i \neq d \end{cases} \qquad \forall i \in N. \qquad (4.2)$$

At optimality, the solution functions $W_i(t)$ and $N_i(t)$ are given by arguments respectively minimizing equations *(4.1)* and *(4.2)*. These functions are not necessarily unique, since there may be several paths and waiting schedules which attain an optimal travel cost.

$$W_i(t) = \arg\min_{\tau \in WT_i(t)}\{cwc_i(t+\tau) - cwc_i(t) + C_i^{(nw)}(t+\tau)\} \qquad \forall i \in N, \qquad (4.3)$$

$$N_i(t) = \arg\min_{j \in A_i}\{c_{ij}(t) + C_j^{(w)}(t + d_{ij}(t))\} \qquad \forall i \in N - \{d\}. \qquad (4.4)$$

For the one-to-all problem, we take similar steps as in the all-to-one case in order to prevent the source node s from appearing as an intermediate node in any optimal path, in order to simplify the formulation of the problem. For minimum-cost one-to-all problems, we therefore replace s with the designation s' and add a new artificial source node s and a zero-cost arc (s, s'). In order to restrict the set of feasible departure times from s we use the same method as with the all-to-one problem, in which the time-dependent travel time and cost of the arc (s, s') is used to determine the set of feasible departure times. As in the all-to-one problem, restriction of departure times in this fashion only applies to minimum-cost or non-FIFO minimum-time problems. For the minimum-time one-to-all problem in a FIFO network, it is necessary to consider only the single earliest possible departure time, since we shall see in Section 4.3 that departures from later times will never lead to earlier arrivals at any destination.

Optimality conditions for the one-to-all problem are similar to those of the all-to-one problem. Since network data is stored in a so-called "forward-star" representation, from which we cannot easily determine the set of departure times along a link that result in a particular arrival time, we write the optimality conditions as set of inequalities rather than minimization expressions. If $C_i^{(w)}(t)$ and $C_i^{(nw)}(t)$ represent a feasible set of path costs, then they must satisfy the following conditions for optimality:

$$C_i^{(w)}(t)+c_{ij}(t) \geq C_j^{(nw)}(t+d_{ij}(t)) \qquad \forall (i,j) \in A, \qquad (4.5)$$
$$C_i^{(nw)}(t)+cwc_i(t+\tau)-cwc_i(t) \geq C_i^{(w)}(t+\tau) \qquad \forall \tau \in WT_i(t),\ \forall i \in N-\{s\}, \qquad (4.6)$$
$$C_s^{(w)}(t) = C_s^{(nw)}(t) = 0. \qquad (4.7)$$

The functions $W_i(t)$, $PN_i(t)$, and $PT_i(t)$ must at optimality satisfy the following conditions in a minimum-cost one-to-all problem:

$$C_i^{(nw)}(t)+cwc_i(t+W_i(t))-cwc_i(t) = C_i^{(w)}(t+W_i(t)) \qquad \forall i \in N, \qquad (4.8)$$
$$C_{\tilde{i}}^{(w)}(PT_i(t))+c_{\tilde{i}i}(PT_i(t)) = C_i^{(nw)}(t),\ where\ \tilde{i} = PN_i(t) \qquad \forall i \in N-\{s\}. \qquad (4.9)$$

Proposition 4.1: *Conditions (4.1) and (4.2) are necessary and sufficient for optimality for the all-to-one dynamic shortest path problem, and conditions (4.5), (4.6), and (4.7) are necessary and sufficient for optimality for the one-to-all problem.*

Proof: It is a simple matter to show that these conditions are necessary for optimality. If any one of these conditions is violated, then the particular arc or waiting time causing the violation may be utilized as part of a new path in order to construct a better solution. For example, if condition *(4.2)* fails to hold for some node $i \in N$ and time t, then this means there is some outgoing node $j \in A_i$ which will lead to a better path than $N_i(t)$. Since violation of these conditions implies non-optimality, the conditions are therefore necessary.

In order to show sufficiency, we examine first the case of the all-to-one problem. Assume conditions *(4.1)* and *(4.2)* are satisfied by some functions $C_i^{(w)}(t)$ and $C_i^{(nw)}(t)$ which represent a feasible solution. Consider any feasible path from some node $i \in N$ and some departure time t to the destination node d, having the form $P = (i_1, t_1, w_1)$ - (i_2, t_2, w_2) - ... - (i_k, t_k, w_k), where each ordered triple represents a node, an initial time at that node, and a waiting time before departure from that node. We have $i_1 = i$, $t_1 = t$, $i_k = d$, $w_k = 0$, and for any valid path we must have $t_{j+1} = t_j + w_j + d_{i_j i_{j+1}}(t_j + w_j)$ for all $j \in \{1, 2, \ldots, k-1\}$. Assuming *(4.1)* and *(4.2)* are satisfied, we have, respectively,

$$C_{i_1}^{(w)}(t_1) \leq cwc_{i_1}(t_1 + w_1) - cwc_{i_1}(t_1) + C_{i_1}^{(nw)}(t_1 + w_1) \qquad (4.10)$$
$$C_{i_1}^{(w)}(t_1) \leq cwc_{i_1}(t_1 + w_1) - cwc_{i_1}(t_1) + c_{i_1 i_2}(t_1 + w_1) + C_{i_2}^{(w)}(t_2) \qquad (4.11)$$

In general, after expanding the $C_{i_2}^{(w)}(t_2)$ term in *(4.11)*, followed by $C_{i_3}^{(w)}(t_3)$, $C_{i_4}^{(w)}(t_4)$, etc. along the entire path, since $C_{i_k}^{(w)}(t) = 0$ the right-hand side of the inequality in *(4.11)* will expand into the cost of the entire path *P*. Hence, $C_i^{(w)}(t)$ is a lower bound on the cost of any arbitrary path from node *i* and time *t* to the destination, and since $C_i^{(w)}(t)$ represents the cost of some feasible path, we conclude that the solution represented by $C_i^{(w)}(t)$ and $C_i^{(nw)}(t)$ is optimal. A similar, symmetric argument applies to show that conditions *(4.5)*, *(4.6)*, and *(4.7)* are sufficient for optimality for the one-to-all problem. □

4.2 Solution Properties

We proceed to establish several useful mathematical properties of solutions to the all-to-one and one-to-all dynamic shortest path problem in a piece-wise linear continuous-time network.

Lemma 4.1: *There always exists a finite time t^+, beyond which all nodes exhibit linear departure behavior. Similarly, there always exists a finite time t^-, prior to which all nodes exhibit linear departure behavior.*

Proof: We prove the lemma for the all-to-one problem; the same argument applies to the one-to-all problem. Consider first the case of constructing a value of t^+. We focus on the region of time greater than all piece-wise boundaries of all network data functions $d_{ij}(t)$, $c_{ij}(t)$, $w_i(t)$, and $cwc_i(t)$, so these functions will behave as simple linear functions for the interval of time under consideration. We further restrict our focus within this interval to a subinterval of time $t > t_0$, where t_0 is chosen such that $c_{ij}(t) \geq 0$ for all arcs $(i, j) \in A$. Such a subinterval always exists since we have assumed that all arc travel cost functions must be either increasing or static and non-negative as time approaches infinity. For any departure at a time $t > t_0$, there exists an acyclic optimal path to the destination which involves no waiting, since the introduction of waiting or a cycle can never decrease the cost of any path departing at a time $t > t_0$. For any node $i \in N$, we can therefore enumerate the entire set of feasible acyclic paths from i to d. There will be a finite number of such paths, and each of these paths will have a simple linear time-dependent path travel cost for departure times $t > t_0$. The optimal path travel cost from i to d for departures within this interval will be the minimum of this finite set of linear functions, and since the minimum of such a set of functions will be a piece-wise linear function with a finite number of pieces, there must exist some departure time t_i^+ after which the same path remains optimal forever from i to d. We can therefore pick t^+ to be the maximum of these t_i^+ values.

The existence of the time t^- is argued from the fact that we know by assumption that there is some time t_{static} before which the network data functions remain static. One may always find acyclic optimal paths which involve no waiting that arrive within this region of time, since the presence of waiting or a cycle in any path arriving within this time interval can never decrease the cost of the path. We thus have an upper bound of $\sum_{(i,j)\in A} \beta(d_{ij},1)$ on the length of any path which arrives at a time prior to t_{static}, so we can accordingly set t^- to the value $t_{static} - \sum_{(i,j)\in A} \beta(d_{ij},1)$. □

Lemma 4.2: *The time-dependent function giving the optimal travel cost of any particular path P as a function of departure time will always be piece-wise linear, and consist of a finite number of linear pieces, each spanning an extent of time of strictly positive duration. The function giving the optimal travel cost of any particular path P as a function of arrival time will always be piece-wise linear and consist of a finite number of linear pieces, where some of these pieces may span an extent of time of zero duration.*

Proof: Suppose that a path P consists of the sequence of nodes $i_1, i_2, \ldots, i_k$. The time-dependent travel cost along path P as a function of departure time will depend on the waiting times spent at the nodes along the path, and the optimal time-dependent travel cost function is obtained by minimizing over the set of all such feasible waiting times. For simplicity, we will say that a function is *special* if it is piece-wise linear and consists of a finite number of linear pieces, each spanning an extent of time of strictly positive duration. We prove the fact that the optimal time-dependent travel cost function (as a function of departure time) along any path P will be special by induction on the terminal sub-paths of P of increasing number of arcs: i_k, $i_{k-1} - i_k$, $i_{k-2} - i_{k-1} - i_k$, etc. We claim that each of these sub-paths is special. The induction hypothesis trivially holds for the path consisting of the single node i_k, since this path has zero travel cost. Suppose now that the induction hypothesis holds for some sub-path $P_{r+1} = i_{r+1}, \ldots, i_k$. We can write the time-dependent optimal travel cost of the sub-path $P_r = i_r, i_{r+1}, \ldots, i_k$ as a function of departure time using *(4.1)* and *(4.2)* as follows. We denote the optimal travel cost of such a sub-path at departure time t by $TC_{p,r}(t)$.

$$TC_{p,r}(t) = \min_{\tau \in WT_i(t)} \{cwc_i(t+\tau) - cwc_i(t) + c_{i_r i_{r+1}}(t+\tau) + TC_{p,r+1}(a_{i_r i_{r+1}}(t+\tau))\} \qquad (4.12)$$

Denote the composition of the path travel cost function TC_{r+1} and the arc arrival time function $a_{i_r i_{r+1}}$ as $TC'_{p,r+1}(t)$. This composed function will be the composition of two special functions, so $TC'_{p,r+1}(t)$ will also be special. We can then write Equation *(18)* in the following simple form

$$TC_{p,r}(t) = \min_{\tau \in WT_i(t)} \{f_r(t+\tau)\} - cwc_i(t), \qquad (4.13)$$

where $f_r(t) = cwc_i(t) + c_{i_r i_{r+1}}(t) + TC'_{p,r+1}(t)$ is a sum of special functions, and therefore also special. If waiting is prohibited at all nodes, then we will have $TC_{p,r}(t) = f_r(t) - cwc_i(t)$, which will be special. If waiting is allowed as some nodes, then by the continuity assumption $f_r(t)$ will be continuous, and the minimum of $f_r(t + \tau)$ over the window $[t, t + ubw_i(t)]$, where the endpoints of this window are continuous non-decreasing functions if t, will also be a special function. We therefore have the fact that $TC_{p,r}(t)$ satisfies the induction hypothesis, and by induction, the optimal time-dependent travel cost of the entire path $TC_p(t) = TC_{p,1}(t)$ must be special.

Once the optimal time-dependent path travel cost function $TC_p(t)$ is determined as a function of departure time, we can define an associated travel time function $TT_p(t)$ which gives the time required to transverse path P, departing from time t, using an appropriate waiting schedule such that we can achieve the minimal path transversal cost $TC_p(t)$. It can be shown using induction in the same way as in the previous argument, that the $TT_p(t)$ function should be special. We can then write the optimal time-dependent path travel cost function of path P as a function of arrival time as follows. We denote by $TC_p^{(arr)}(t)$ the optimal path cost as a function of an arrival time t.

$$TC_p^{(arr)}(t) = \min_{\{\tau | \tau + TT_p(\tau) = t\}} \{TC_p(\tau)\} \tag{4.14}$$

Due to the structure of this minimization and the fact that $TC_p(t)$ and $TT_p(t)$ are special, the function $TC_p^{(arr)}(t)$ will be piece-wise linear with a finite number of pieces, but some of these pieces may span an extent of time of zero duration. □

Proposition 4.2: *An optimal solution always exists for both the all-to-one and one-to-all dynamic shortest path problems. Furthermore, this solution will always consist of a finite number of LNIs, and for the all-to-one problem all LNIs will be nonsingular.*

Proof: We prove the proposition for the all-to-one problem; the same argument symmetrically applies to the one-to-all problem. We know that there exist finite times t^+ and t^- as defined in Lemma 4.1. Optimal paths departing at times $t > t^+$ will consist of no more than $n - 1$ arcs, since optimal paths are acyclic within this region of time. Similarly, any optimal path which arrives at the destination at a time $t < t^-$ will consist of no more than $n - 1$ arcs. By assumption, there exists a positive ε such that $\varepsilon \leq d_{ij}(t)$ for all $(i, j) \in A$. We therefore have a finite bound of $L = 2n + (t^+ - t^-)/\varepsilon$ arcs in any optimal path. Consider any particular node $i \in N$, and consider the finite set of all paths of L or fewer arcs from i to the destination node d. For every path P within this set, by Lemma 4.2 we know that the optimal time-dependent travel cost along P will be a piece-wise linear function of the departure time from node i consisting of a finite number of pieces, each spanning a positive extent of time. For the all-to-one problem, the function $C_i^{(w)}(t)$ giving the optimal time-dependent path travel cost from i to d as a function of departure time is the minimum of a finite number of such piece-wise linear functions, and we can therefore conclude that the optimal path travel cost functions $C_i^{(w)}(t)$ always a) exists, b) consists of only a finite number of linear pieces, and c) consists only of pieces which span an extent of time of strictly positive duration. Also by Lemma 4.2, and the time-dependent optimal travel cost of P as a function of arrival time is piece-wise linear with a finite number of pieces, but some of these pieces may have zero extent. For the one-to-all problem, the function $C_i^{(w)}(t)$ is the minimum of a finite set of functions which are piece-wise linear with a finite number of pieces, and therefore $C_i^{(w)}(t)$ will be piece-wise linear and contain a finite number of linear pieces. □

As with the static shortest path problem, the solution to a dynamic shortest path problem will yield unique path costs $C_i^{(w)}(t)$ and $C_i^{(nw)}(t)$, but the actual paths which constitute such a solution may not necessarily be unique.

4.3 Properties of FIFO Networks

It is possible to develop efficient algorithms to solve dynamic shortest path problems in FIFO networks because these networks satisfy a number of important properties which simplify many of the computations related to finding shortest paths. We describe some of these properties as lemmas below.

Lemma 4.3: *For any path through a FIFO network, the function giving the arrival time at the end of the path as a function of departure time at the start of the path is non-decreasing.*

Proof: The arrival time function of a path is equal to the composition of the arrival time functions of the arcs comprising the path. Since arc arrival time functions are non-decreasing in a FIFO network, so are path travel time functions, since the composition of any set of non-decreasing functions yields a non-decreasing function. □

Lemma 4.4: *Waiting in a FIFO network never decreases the arrival time at the end of any path.*

Proof: Since waiting is equivalent to a delay in the departure time along some path, and since path arrival time is a non-decreasing function of departure time by Lemma 4.3, we see that waiting can never lead to a decrease in the arrival time of any path. □

Lemma 4.5: *In a FIFO network, one can always find minimum-time paths which are acyclic.*

Proof: Suppose a minimum-time path contains a cycle. This implies that some node $i \in N$ is visited at two different times t_1 and t_2, where $t_1 < t_2$. However, a path of equivalent travel time is obtained by removing the cycle, and by simply waiting at node i from time t_1 until time t_2 before continuing along the remainder of the path. Using Lemma 4.4, we conclude that the same path with the waiting interval removed, which is equivalent to the original path with the cycle removed, will have an arrival time no later than that of the original path. Since the removal of any cycle in a path never leads to an increase in arrival time, we can always compute acyclic minimum-time paths. □

Lemma 4.6: *A minimum-time problem in a non-FIFO network can be transformed into an equivalent FIFO minimum-time problem, if unbounded waiting is allowed everywhere in the network.*

Proof: The following transformation of the arc travel time functions produces the desired effect by selecting the most optimal arrival time attainable by the combination of waiting followed by travel along an arc:

$$d_{ij}^{*}(t) = \min_{\tau \geq 0}\{\tau + d_{ij}(t+\tau)\} \qquad \forall (i, j) \in A \qquad (4.15)$$

A minimum is guaranteed to exist for Equation *(4.15)* since we have assumed that arc travel times are continuous in non-FIFO networks in which waiting is allowed. In the absence of this assumption, the minimum may not be attainable. □

Lemma 4.7: *For any arrival time t_a at a the end of a path, there will exist a corresponding contiguous interval of departure times which will result in an arrival at exactly time t_a.*

Proof: Follows directly from non-decreasing property of path travel time functions, shown in Lemma 4.3. □

Lemma 4.8: *For any given departure time t_d at a source node s in a FIFO network G, one can find a shortest path tree embedded in G directed out of s which consists of shortest paths originating at s at time t_d.*

Corollary: *For any arrival time t_a at a destination node d in a FIFO network G, one can find a shortest path tree embedded in G directed into d which consists of shortest paths arriving to d at time t_d.*

Proof: Follows from the fact that one can always find acyclic shortest paths, due to Lemma 4.5. Note that for the corollary, the tree of shortest paths arriving at a destination node at a particular time t_a may not include all nodes $i \in N$, since for some nodes there may not exist a departure time from which an optimal path to the destination arrives at time t_a. Sections 4.4 and 5.1 discuss techniques and results which may be applied to compute these shortest path trees. □

For the next lemma, we will argue bounds on the number of LNI boundaries at each node which may be caused by each piece-wise linear boundary of each arc travel time function. In saying that a piece-wise boundary of an arc travel-time function *causes* an LNI boundary, we mean that removal of the arc travel time boundary would cause the removal of the LNI boundary. One can intuitively visualize this causality by considering a linear piece of an arc travel time function to be a taut string. Pulling on a point in the middle of the string results in the addition of a new piece-wise boundary at that point, and any LNI boundary which is created or affected by this pulling is considered to be caused by this new piece-wise boundary.

Lemma 4.9: *In a FIFO network, a piece-wise boundary time t at which some arc travel time function $d_{ij}(t)$ improves may cause at most one LNI boundary at each node $i \in N$. A piece-wise boundary t at which some arc travel time function $d_{ij}(t)$ worsens may cause at most two LNI boundaries at each node.*

Proof: Define P_{ab} to be the set of paths connecting nodes a and b. Let $TT_p(t)$ denote the travel time along path p departing at time t, and let $TT_{ab}(t)$ denote travel time of a minimal-time path from node *a* to node *b*, departing from node *a* at time *t*. Consider an arbitrary node $k \in N$, and an arbitrary arc $(i, j) \in A$ where the arc travel time $d_{ij}(t)$ either improves or worsens at some time t^*. The optimal path travel time from *k* to the destination node *d*, $C_k^{(nw)}(t)$ can be expressed as the minimum travel time over paths containing (i, j) and over paths which do not contain (i, j), which we define as $C_k^{(1)}(t)$ and $C_k^{(2)}(t)$ respectively:

$$C_k^{(nw)}(t) = \min\{C_k^{(1)}(t), C_k^{(2)}(t)\} \tag{4.16}$$

$$C_k^{(1)}(t) = \min_{\substack{p \in P_{kd} \\ (i,j) \in p}} \{TT_p(t)\} \tag{4.17}$$

$$C_k^{(2)}(t) = \min_{\substack{p \in P_{kd} \\ (i,j) \notin p}} \{TT_p(t)\} \tag{4.18}$$

Since the FIFO property is satisfied, we know by Lemma 4.5 that all minimum-time paths are acyclic, so the arc (i, j) appears at most once in any optimal path, and we can rewrite $C_k^{(1)}(t)$ as:

$$C_k^{(1)}(t) = TT_{ki}(t) + d_{ij}(t + TT_{ki}(t)) + TT_{jd}(t + TT_{ki}(t) + d_{ij}(t + TT_{ki}(t)))` \tag{4.19}$$

The piece-wise linear function $t + TT_{ki}(t)$ is equal to the minimum over the set of paths $p \in P_{ki}$ of the function $t + TT_p(t)$, which by the FIFO property must be non-decreasing; hence, the function $t + TT_{ki}(t)$ is also non-decreasing, since the minimum of a set of non-decreasing functions is also a non-decreasing function. The piece-wise linear function $f(t) = t + TT_{ki}(t) + d_{ij}(t + TT_{ki}(t))$ is also non-decreasing since it is the composition of two non-decreasing functions: $t + d_{ij}(t)$ and $t + TT_{ki}(t)$. The presence of a piece-wise boundary at time t^* where $d_{ij}(t)$ improves or worsens will cause at most one piece-wise boundary where $f(t)$ respectively improves or worsens, where this boundary is at a time τ determined by

$$\tau = (t + TT_{ki}(t))^{-1}(t^*), \tag{4.20}$$

which is nothing more than the FIFO inverse of the function $t + TT_{ki}(t)$ evaluated at t^*. The piece-wise linear arrival time function $t + C_k^{(1)}(t)$ is given by the composition of the functions $t + TT_{kd}(t)$ and $f(t)$, both non-decreasing, and is therefore also non-decreasing. Furthermore, an improvement or worsening of $f(t)$ at time $t = \tau$ will cause at most one respective boundary where $t + C_k^{(1)}(t)$ respectively improves or worsens, also at time $t = \tau$. If the function $t + C_k^{(1)}(t)$ improves or worsens at some time $t = \tau$, then so will the function $C_k^{(1)}(t)$ by itself. We therefore conclude that the presence of a piece-wise boundary at time t^* where $d_{ij}(t)$ improves or worsens will cause at most one piece-wise boundary where $C_k^{(1)}(t)$ respectively either improves or worsens, where this boundary is at a time τ determined by Equation *(4.20)*. The function $C_k^{(2)}(t)$ will be unaffected by all piece-wise boundaries of $d_{ij}(t)$.

A single piece-wise boundary at which the function $C_k^{(1)}(t)$ improves will cause at most one LNI boundary (an improvement) at node k. At the point in time $t = \tau$ where the improvement of $C_k^{(1)}(t)$ occurs, either

(i) $C_k^{(1)}(\tau) < C_k^{(2)}(\tau)$, or
(ii) $C_k^{(1)}(\tau) \geq C_k^{(2)}(\tau)$.

In case (i), due to Equation *(4.16)*, $C_k^{(nw)}(t)$ will improve at time τ with $N_k(t)$ remaining unchanged. In case (ii), the piece-wise improvement of $C_k^{(1)}(t)$ might cause $C_k^{(1)}(t)$ to begin decreasing until it eventually overtakes $C_k^{(2)}(t)$, becoming more minimal at a later point in time, and resulting in an improvement in $C_k^{(nw)}(t)$ at that later time. In both cases (i) and (ii), the improvement of $C_k^{(1)}(t)$ causes at most only one LNI boundary at node k because, after this boundary, $C_k^{(1)}(t)$ will be linearly decreasing below $C_k^{(2)}(t)$, and an additional piece-wise boundary (an improvement) of some arc travel time function will be necessary in order to cause another improvement to $C_k^{(1)}(t)$, and therefore also to $C_k^{(nw)}(t)$.

If the function $C_k^{(1)}(t)$ worsens at some point in time $t = \tau$, we can apply the same reasoning. In case (ii), an LNI boundary at node k will not be caused as a result of $C_k^{(1)}(t)$ worsening at $t = \tau$, since this can only result in $C_k^{(1)}(\tau)$ becoming even less optimal. In case (i), a worsening of $C_k^{(1)}(t)$ at $t = \tau$ will cause no more than two LNI boundaries at node k: one of these boundaries occurs immediately at time $t = \tau$ since $C_k^{(nw)}(t)$ will worsen at $t = \tau$. The second LNI boundary may occur due to the fact that $C_k^{(1)}(t)$ will start increasing faster or decreasing slower after the first LNI boundary, resulting in $C_k^{(2)}(t)$ eventually becoming more minimal at some later point in time, and resulting in corresponding improvement in $C_k^{(nw)}(t)$ at this later point in time. Therefore, we have shown that a single piece-wise boundary of an arbitrary arc travel time function can cause at most three LNI boundaries at every node $k \in N$, since a single piece-wise arc travel time function boundary may consist of both an improvement and a worsening of the arc travel time function. □

Lemma 4.10: *For each node $i \in N$, there will be at most $3P^*$ LNIs comprising a solution to the minimum-time all-to-one dynamic shortest path problem in a FIFO network.*

Proof: Due to Lemma 4.9, we can bound the number of LNI boundaries at each node in terms of the number of piece-wise boundaries in the arc travel time functions of each arc $(i, j) \in A$. There are $P^* - m$ piece-wise arc travel time boundaries, each of which represents an improvement, a worsening, or both. By applying Lemma 4.9 we establish the fact that there must be no more than $3P^*$ LNIs at any given node which are caused by these piece-wise arc travel time boundary points.

However, we argue that every boundary between two LNIs must be caused by some arc travel time boundary point. The presence of an LNI boundary at some point in time t_0 indicates that there is an intersection at this time where two path travel time functions cross. At this intersection there is a switch from one optimal path to another. In order for such an intersection to occur, the slope of at least one of the path travel time functions must be nonzero. However, since we assume that all path travel time functions are initially constant (at time $t = -\infty$), the cause for the intersection can be attributed to some piece-wise boundary point in some arc travel time function which arranged the slopes of two path travel times in a configuration such that they would intersect at time t_0. Since all of the LNI boundaries in the solution functions are caused by piece-wise arc travel time boundaries, the lemma is therefore established. □

4.4 Symmetry and Equivalence Among Minimum-Time FIFO Problems

Within the realm of minimum-time problems in a FIFO networks, there is a greater symmetry between the all-to-one problem and the one-to-all problem. In this section we discuss methods of transforming between minimum-time FIFO problems which are symmetric in time. Some of the transformations presented below have been independently proposed by Daganzo [8].

An alternative way of describing the minimum-time one-to-all problem is to call it a *minimum arrival time problem*, for which one must find a set of paths which leave a designated source node $s \in N$ at or after a particular departure time t_d, which reach each node $i \in N$ at the earliest possible arrival time. While the all-to-one problem is not completely symmetric in time with respect to this problem, a close variant, which we call the *maximum departure time* problem, is exactly symmetric. The maximum departure time problem involves finding a set of paths which arrive at a designated destination node $d \in N$ at or before some particular arrival time t_a which leave each node $i \in N$ at the latest possible departure time. The discussion of these two problems in this subsection will apply only within the context of FIFO networks.

We now develop a simplified alternative formulation for the minimum arrival time problem and the maximum departure time problem. For the minimum arrival time problem, we let $EA_i(t_d)$ denote the earliest arrival time at node i if one departs from the source node s at or after time t_d, and we let $PN_i(t_d)$ denote the previous node along a path to i which arrives at time $EA_i(t_d)$. Similarly, for the maximum departure time problem, we let $LD_i(t_a)$ denote the latest possible departure time from node i if one wishes to arrive at the destination node d at or before time t_a, and let $N_i(t_a)$ denote the next node along such a path. One can write optimality conditions for the minimum arrival time problem as follows. These conditions are well-known, as the minimum arrival time (i.e. minimum-time one-to-all) problem in a FIFO network has been well-studied in the literature.

$$EA_j(t_d) = \begin{cases} t_d & \text{if } j = s \\ \min\limits_{i \in B_j}\{a_{ij}(EA_i(t_d))\} & \text{if } j \neq s \end{cases} \qquad \forall j \in N, \qquad (4.21)$$

$$PN_j(t_d) = \arg\min_{i \in B_j}\{a_{ij}(EA_i(t_d))\} \qquad \forall j \in N - \{s\}. \qquad (4.22)$$

Similarly, one can write the symmetric optimality conditions for the maximum departure time problem as:

$$LD_i(t_a) = \begin{cases} t_a & \text{if } i = d \\ \max\limits_{j \in A_i}\{a_{ij}^{-1}(LD_j(t_a))\} & \text{if } i \neq d \end{cases} \qquad \forall i \in N, \qquad (4.23)$$

$$N_i(t_a) = \arg\max_{j \in A_i}\{a_{ij}^{-1}(LD_j(t_a))\} \qquad \forall i \in N - \{d\}. \qquad (4.24)$$

In the above expressions, the function $a_{ij}^{-1}(t)$ denotes the FIFO inverse of the arc arrival time function $a_{ij}(t)$. We now discuss the possibility of transformations between the one-to-all (minimum arrival time), maximum departure time, and all-to-one problems.

Lemma 4.11: *It is possible to transform between instances (and solutions) of the minimum arrival time and maximum departure time problems.*

Corollary: *In a FIFO network, it is possible to transform between instances (and solutions) of the minimum arrival time problem for all departure times (i.e. the minimum-time one-to-all problem for all departure times), and the maximum departure time problem for all arrival times.*

Proof: The following operations will achieve a transformation between the minimum arrival time problem and its solution and the maximum departure time problem:

(i) Reverse the direction of all arcs in the network
(ii) Swap the source node s and the destination node d
(iii) Swap the source departure time t_d with the destination arrival time t_a
(iv) Let $PN_i(t) = N_i(t)$ for all $i \in N$
(v) Let $EA_i(t) = -LD_i(t)$ for all $i \in N$
(vi) Replace each arc arrival time function $a_{ij}(t)$ with $-a_{ij}^{-1}(-t)$ for all $(i, j) \in A$

Steps *(v)* and *(vi)* above actually represent a reversal in the direction of time. The preceding steps will transform the optimality conditions *(4.21)* and *(4.22)* of the minimum arrival time problem into optimality conditions *(4.23)* and *(4.24)* of the maximum departure time problem and vice versa. Therefore, any solution which satisfies the optimality conditions for one of the problems prior to transformation will satisfy the optimality conditions for the other problem after transformation. As the minimum arrival time problem for all departure times and the maximum departure time problem for all arrival times only involves computing $EA_i(t)$ or $LD_i(t)$ respectively as functions of time rather than for a particular value of time, the above transformation still applies to these problems. □

Lemma 4.12: *In a FIFO network, it is possible to transform between instances (and solutions) of the maximum departure time problem for all arrival times, and the minimum-time all-to-one problem.*

Proof: A transformation between these two problems will not change the arc travel time functions. For the minimum-time all-to-one problem in a FIFO network, the function $C_i^{(nw)}(t)$ gives the travel time of the optimal path to the destination from node i, departing at time t. Since by Lemma 4.4, waiting is never beneficial in FIFO networks, we will have $C_i^{(w)}(t) = C_i^{(nw)}(t)$ and $W_i(t) = 0$, so only the functions $C_i^{(nw)}(t)$ and $N_i(t)$ are necessary to specify a solution to the all-to-one problem. Additionally, in a FIFO network, since path travel times satisfy the FIFO property, the function $f(t) = t + C_i^{(nw)}(t)$ will be non-decreasing and we can therefore construct its FIFO inverse $f^{-1}(t)$. Similarly, since the function $LD_i(t)$ will for all nodes be a non-decreasing functions, we can construct its FIFO inverse $LD_i^{-1}(t)$. In order for the solution to a minimum-time all-to-one problem to be consistent with the solution of a maximum-departure time problem for all arrival times, we must satisfy $LD_i(t + C_i^{(nw)}(t)) = C_i^{(nw)}(t)$. This naturally leads to the following transformation between the decision variables representing solutions to the two problems:

$$LD_i(t) = f_i^{-1}(t), where\ f(t) = t + C_i^{(nw)}(t) \qquad \forall i \in N \qquad (4.25)$$

$$C_i^{(nw)}(t) = LD_i^{-1}(t) - t \qquad \forall i \in N \qquad (4.26)$$

It can be shown that decision variables satisfying the optimality conditions of one problem prior to transformation will satisfy the optimality conditions of the other problem after transformation. □

Proposition 4.3: *Any algorithm which solves the minimum-time all-to-one problem in a FIFO network can be used to solve the minimum-time one-to-all problem for all departure times in a FIFO network.*

Corollary: *Any algorithm which solves the minimum-time one-to-all problem for all departure times in a FIFO network can be used to solve the minimum-time all-to-one problem in a FIFO network.*

Proof: Follows directly from Lemmas 4.11 and 4.12. One may transform between instances of the minimum-time one-to-all problem and instances of the all-to-one problem, solve a problem, and then transform the solution back into the form desired for the original problem. Alternatively, one may modify the workings of an all-to-one algorithm or a one-to-all for all departure times algorithm so that these transformations are implicitly represented within the algorithm. □

Chapter 5

Preliminary Algorithmic Results for Minimum-Time FIFO Problems

A historic result in the development of dynamic shortest path algorithms has been the fact that the minimum-time one-to-all problem in FIFO networks is solvable by a trivial adaptation of any static label-setting or label-correcting shortest path algorithm. Details of this solution technique are given in this chapter. Additionally, we develop a simple, generic method for adapting any minimum-time FIFO all-to-one problem so that it may be solved efficiently in parallel using existing serial algorithms. Since the minimum-time all-to-one problem has been shown in Section 4.4 to be computationally equivalent to the minimum-time one-to-all problem for all departure times in a FIFO network, the parallel techniques derived within this chapter will apply to any algorithm which solves either of these two problems.

5.1 FIFO Minimum-Time One-to-All Solution Algorithm

To solve the FIFO minimum-time one-to-all problem for a single departure time, we must compute the most optimal path to every node $i \in N$ originating from a single source node $s \in N$ and a given departure time t_d. This problem is the same as the earliest arrival time problem discussed in Section 4.4. In this section, we present a well-known result from the literature, due to Dreyfus [10], Ahn and Shin [1], and Kaufman and Smith [14], which allows this problem to be solved as efficiently as a static shortest path problem.

In order to describe the solution to a minimum-time one-to-all problem in a FIFO network, it is only necessary to specify a single scalar label for each node rather than a label which is a function of time, for the following reasons. Even if waiting is allowed at the source node, we know that departure from the source must occur at time exactly t_d since by Lemma 4.4 waiting at the source node will never decrease the travel time of any path. By the same reasoning, one can argue that it is only necessary to keep track of the earliest possible arrival time at each node, which we denote EA_i in keeping with the notation from Section 4.4. Commodities traveling along minimum-time paths from the source node at time t_d will only visit each node $i \in N$ at time EA_i. As discussed in Section 4.1, the union of the set of optimal paths originating at node s and time t_d will form a shortest path tree embedded in G. We denote by PN_j the predecessor node of a node $j \in N$ along this tree.

Equations *(4.21)* and *(4.22)* from Section 4.4 give the optimality conditions for the minimum-time one-to-all problem. These conditions are quite similar to the optimality conditions of a static shortest path problem, and indeed, it is possible to modify any labeling static shortest path algorithm so that it is based on this extended dynamic formulation. Algorithm 1 is an example of such a modified static algorithm. For the results of computational testing of the performance of dynamic adaptations of different types of static algorithms, the reader is referred to [5].

```
Initialization:
$\forall i \quad opt_i \leftarrow \infty,\ PT_i \leftarrow \emptyset$
$opt_s \leftarrow t_d,\ UnSetNodes = N$

Main Loop:
For $k \leftarrow 1 .. n-1$
  $i \leftarrow Select\_Mincost\_Node(\ UnSetNodes,\ opt\ )$
  $UnSetNodes \leftarrow UnSetNodes - \{i\}$
  For all $j \in A_i$
    If $opt_i + d_{ij}(opt_i) < opt_j$ then
      $opt_j \leftarrow opt_i + d_{ij}(opt_i)$
      $PN_j \leftarrow i$
```

Algorithm 1: A modified version of Dijkstra's static shortest path algorithm which computes minimum-time one-to-all dynamic shortest paths in a FIFO network.

Using the transformation given in Lemma 4.11, one may use dynamically-adapted static solution algorithms to solve the maximum departure time problem described Section 4.4 in a FIFO network. Additionally, by employing the transformation given in Lemma 4.6, one can use dynamically-adapted static solution algorithms to solve minimum-time one-to-all problems in non-FIFO networks if unbounded waiting is allowed at all nodes. In

[15], Orda and Rom describe a further extension of this approach which efficiently solves non-FIFO minimum-time one-to-all problems in which waiting is allowed only at the source node. We will show in Chapters 6 and 7 methods for solving the minimum-time all-to-one problem, which due to Proposition 4.3 may also be used to solve the minimum-time one-to-all problem for all departure times in a FIFO network.

5.2 Parallel Algorithms for Problems in FIFO Networks

One particular advantage of FIFO networks is that it is possible to partition any minimum-time FIFO all-to-one dynamic shortest path problem into completely disjoint pieces, so that distributed computation may be performed on these pieces without any overhead due to message passing between parallel processing units. Since by Proposition 4.3, the minimum-time FIFO one-to-all problem for all departure times is computationally equivalent to the FIFO minimum-time all-to-one problem, either one of these problems may be solved in parallel by trivially adapting any serial solution algorithm for the minimum-time all-to-one problem. Chapters 6 and 7 present different solution algorithms for the all-to-one problem; either of these may be easily converted to run in a parallel environment. The parallel decomposition methods we describe below are suitable for either a distributed-memory or a shared-memory parallel system; we will for simplicity assume a distributed-memory system for the following discussion.

Our parallel approach to solving the minimum-time all-to-one problem actually involves solving instead the closely-related maximum departure time problem for all arrival times, described in Section 4.4. By Lemma 4.12, a solution to this problem can be easily transformed into a solution to the all-to-one problem – this transformation may be performed very efficiently either in serial or in parallel, using only a single linear sweep through the linear pieces of each solution function. We must therefore as an objective compute the solution functions $LD_i(t)$ and $N_i(t)$ for all nodes $i \in N$, as defined in Section 4.4.

The optimality conditions we must satisfy for the maximum departure time problem for all arrival time problem are given by *(4.23)* and *(4.24)*. These conditions are somewhat similar in form to the minimum-time all-to-one optimality conditions, except the computation of the solution functions at a particular arrival time t_a is no longer dependent on the value of the solution functions at any other point in time except time t_a. Based on this observation, we propose a method of partitioning a problem up into disjoint sub-problems by arrival time. Suppose we have K processing units available. We will assign the k^{th} processor, $1 \leq k \leq K$, to an interval of arrival times $(t_k^{(min)}, t_k^{(max)}]$, such that all of these intervals are disjoint and such that their union is the entirety of time. It will be the responsibility of the k^{th} processor to compute the values of the solution functions $LD_i(t_a)$ for $t_a \in (t_k^{(min)}, t_k^{(max)}]$ and for all nodes $i \in N$. Since there is no dependence between solution functions at different values of time, the tasks of each of the parallel processing units will be completely disjoint, and each may therefore be performed independently with no inter-processor communication.

Initially, the network G and a specification of the FIFO inverse arc arrival time functions $a_{ij}^{-1}(t)$ for all arcs $(i, j) \in A$ are to be broadcast to all processing units. Due to the FIFO property, the solution functions $LD_i(t_a)$ will be non-decreasing for all nodes $i \in N$. Using the transformation given in Lemma 4.11, and the solution techniques of the preceding subsection, it is possible to solve the maximum departure time problem for a particular arrival time t_a very efficiently using a modified static shortest path algorithm. Each processor is to perform two such computations, in order to determine the respective values of $LD_i(t_k^{(min)})$ and $LD_i(t_k^{(max)})$ for all $i \in N$. Since these solution functions are non-decreasing, we will then have for the k^{th} processor:

$$LD_i(t_k^{(\min)}) \leq LD_i(t_a) \leq LD_i(t_k^{(\max)}), where\ t_a \in (t_k^{(\min)}, t_k^{(\max)}) \qquad \forall i \in N. \qquad (5.1)$$

Since we know bounds on the values of the solution functions assigned to each processor, each processor can "clip" the inverse arc arrival time functions $a_{ij}^{-1}(t)$ so that they remain static for all values of time not within the interval of possible values of the solution function $LD_i(t_a)$ for that processor. This operation has no affect on the optimal solution computed by each processor, since at optimality the conditions *(4.23)* and *(4.24)* will never reference any inverse arc arrival time function $a_{ij}^{-1}(t)$ at any value of time t which was clipped away. For the k^{th} processor, the clipping operation utilizes the following transformation:

$$\bar{a}_{ij}^{-1}(t) = \begin{cases} a_{ij}^{-1}(LD_i(t_k^{(\min)})) & \text{if } t \le LD_i(t_k^{(\min)}) \\ a_{ij}^{-1}(t) & LD_i(t_k^{(\min)}) < t \le LD_i(t_k^{(\max)}) \\ a_{ij}^{-1}(LD_i(t_k^{(\max)})) & \text{if } t > LD_i(t_k^{(\max)}) \end{cases} \quad \forall (i,j) \in A. \qquad (5.2)$$

The resulting clipped inverse arc arrival time functions will in general, for each processor, consist of far fewer linear pieces than the original inverse arc arrival time functions. By employing the transformations given in Lemma 4.12, each processing unit can solve this simplified maximum departure time problem for all arrival times as a minimum-time all-to-one problem, using any all-to-one solution algorithm. This computation is likely to proceed quickly due to the smaller number of pieces present in the clipped problem.

We have therefore devised a generic method of partitioning up any minimum-time all-to-one problem into several disjoint all-to-one problems, each of which is simpler to solve in the sense that it contains only a small portion of the linear pieces of the arc travel time functions provided as input to the original problem. One can adopt one of many reasonable policies for determining the boundaries of the intervals of time to assign to each parallel processor. These intervals of time should in general be assigned such that the number of solution LNIs computed by each processor is expected to be equal. A perfectly equal partition may not be possible, but intelligent heuristic partitioning schemes should be able to balance the load reasonably well.

5.2.1 Parallel Techniques for Discrete-Time Problems

It is interesting to note that the approach outlined above for development of parallel algorithms applies not only to continuous-time problems, but to discrete-time problems as well. Ganugapati [11] has investigated different strategies of parallel decomposition based on network topology applied to discrete-time problems. Topology-based decomposition methods were shown to work well on shared-memory parallel platforms. However, their performance on distributed-memory platforms suffers from high inter-processor communication overhead, since the parallel tasks are highly dependent on each-other. The only method proposed to date for partitioning a single discrete-time dynamic shortest path problem into completely disjoint sub-problems is due to Chabini, Florian, and Tremblay [6]. This method solves the all-to-one minimum-time problem in a FIFO network by solving the equivalent latest departure time problem for all arrival times. Each parallel processor is assigned a set of arrival times, and applies a modified static algorithm (as described in Section 5.1) to solve the latest departure time problem for each of its assigned arrival times in sequence. We propose an alternative parallelization technique for FIFO problems, based on the method developed in the previous subsection, which gives a stronger theoretical running time.

Based on the discussion in Section 2.2, we see that all discrete-time problems can be viewed as the application of a static shortest path algorithm to compute shortest paths within a static time-expanded network. For a minimum-time FIFO problem, the time-expanded network can be partitioned into completely disjoint pieces along the time dimension as follows. Consider the all-to-one problem with a destination node $d \in N$, and consider a particular arrival time t_a. By solving a single latest departure time problem, described Section 4.4, one can compute for each node $i \in N$ the latest departure time $LD_i(t_a)$ for which arrival at the destination is possible by time t_a. These departure times may be efficiently computed using a modified static algorithm, as described in Section 5.1. The time-space network may then be partitioned into two disjoint pieces, where one processor is assigned the computation of optimal paths leaving from the set node-time pairs $S_1 = \{(i, t) \mid i \in N, t \leq LD_i(t_a)\}$, and a second processor is assigned the remaining node-time pairs $S_2 = \{(i, t) \mid i \in N, t > LD_i(t_a)\}$. Any arc in the time-expanded network crossing from S_1 to S_2 will never be part of an optimal path, because from any node-time pair in S_1, it is possible to reach the destination by time t_a, whereas any optimal path departing from any node-time pair in S_2 must necessarily reach the destination after time t_a. Therefore, this partition effectively divides the time-expanded network into two completely disjoint pieces; if two processors are assigned the computation of optimal paths departing from each region, then no communication will be necessary between the processors during the process of computation. Within each partition, one may apply discrete-time algorithms with optimal running time, such as those in [5], leading to stronger running times for each processor than the approach proposed by Chabini, Florian, and Tremblay.

This technique generalizes to any number of processors. In the case of multiple processors, one should choose a set of arrival times which partitions the time-expanded network into pieces of relatively equal size. Similarly, one may solve the minimum-time one-to-all for all departure times problem in a discrete-time FIFO network using this technique, only the partitions are generated in this case by solving minimum arrival time problems. Time-based parallelization techniques should enable the efficient computation of both discrete-time and continuous-time dynamic shortest paths in FIFO networks.

Chapter 6

Chronological and Reverse-Chronological Scan Algorithms

In this section, we introduce a new class of continuous-time dynamic shortest path algorithms, which we refer to as chronological scan algorithms. These algorithms will in general compute a solution by scanning either forward or backward through time, in a similar fashion to optimal discrete-time algorithms such as those described in [5]. Although some of these algorithms utilize a reverse-chronological scan through time, we will often refer to all algorithms which scan sequentially through time inclusively as chronological scan algorithms, regardless of the direction one moves through time.

In order to provide a gradual introduction to the methodology underlying these approaches, we first consider the simple, yet extremely common case of computing minimum-time all-to-one paths through a FIFO network. For this problem, we will present a solution algorithm and prove that its running time is strongly polynomial. Using the result of Proposition 4.3, this algorithm can be used as well to solve the minimum-time one-to-all problem for all departure times in a FIFO network, and it can be made to run in parallel using the techniques given in Section 5.2.

We then proceed to develop extended chronological scan algorithms for solving general minimum-cost all-to-one and one-to-all dynamic shortest path problems in FIFO or non-FIFO networks. Although these problems are shown to be NP-Hard, it is argued that for most reasonable problem instances, the running time should be quite efficient. Finally, we discuss the possibility of hybrid continuous-discrete variants of these algorithms which compute an approximate solution in a shorter amount of time.

6.1 FIFO Minimum-Time All-to-One Solution Algorithm

As the minimum-time one-to-all problem in a FIFO network has been optimally solved, as discussed in Section 5.1, we devote our attention to solving the minimum-time all-to-one problem in a FIFO network. We will introduce an reverse-chronological scan algorithm for solving this problem, and prove a polynomial bound on its worst-case running time. The all-to-one algorithm which we give for the minimum-time FIFO case will actually be a subset of the more complicated general minimum-cost all-to-one algorithm presented in Section 6.2. For minimum-time FIFO problems, the computation performed by the general algorithm will actually be equivalent to that of the simplified FIFO algorithm; we will therefore sometimes refer to these two algorithms together as a unit as simply "the all-to-one chronological scan algorithm". Due to Proposition 4.3, this solution algorithm may be used to solve a minimum-time one-to-all problem for all departure times in a FIFO network. Furthermore, the techniques given in Section 5.2 may be used to perform parallel computation.

Due to Lemma 4.4, we find that all waiting costs and constraints may be ignored when computing minimum-time paths through a FIFO network, as waiting is never beneficial in FIFO networks. This fact allows us to considerably simplify the formulation of the all-to-one problem. In the minimum-time FIFO case, we will have $C_i^{(w)}(t) = C_i^{(nw)}(t)$ and $W_i(t) = 0$, so it is only necessary to specify the $C_i^{(nw)}(t)$ and $N_i(t)$ functions in order to characterize a solution to the all-to-one problem. A linear node interval in this case therefore reduces to a region of time during which $N_i(t)$ does not change and during which only $C_i^{(nw)}(t)$ changes as a simple linear function of time. The optimality conditions for the FIFO minimum-time all-to-one problem are simplified accordingly to:

$$C_i^{(nw)}(t) = \begin{cases} 0 & \text{if } i = d \\ \min\limits_{j \in A_i}\{d_{ij}(t) + C_j^{(nw)}(t + d_{ij}(t))\} & \text{if } i \neq d \end{cases} \qquad \forall i \in N, \qquad (6.1)$$

$$N_i(t) = \arg\min_{j \in A_i}\{d_{ij}(t) + C_j^{(nw)}(t + d_{ij}(t))\} \qquad \forall i \in N - \{d\}. \qquad (6.2)$$

To specify a solution, we must therefore enumerate for each node $i \in N$ the complete set of LNIs which determine the piece-wise behavior of the functions $C_i^{(nw)}(t)$ and $N_i(t)$ for that node over all departure times.

The all-to-one chronological scan solution algorithm proceeds in two phases, which we briefly outline. Due to Lemma 4.1, as time approaches $+\infty$ eventually all nodes will exhibit linear departure behavior, and the set of optimal paths emanating from all nodes will form a static shortest path tree directed into the destination node d. By computing the structure of this "infinite-time" shortest path tree, one can determine the values of the solution functions $C_i^{(nw)}(t)$ and $N_i(t)$ for sufficiently large values of t.

Phase two of the algorithm uses this partial solution as a starting point, and analytically computes how far one can move backward in time, starting from time $t = +\infty$, such that all nodes maintain their current linear departure behavior, stopping at the first LNI boundary it encounters. The solution functions are then adjusted to reflect the change in linear behavior at this boundary, and the algorithm continues its backward scan through time until it discovers each LNI boundary in sequence, terminating once all LNIs are discovered.

6.1.1 Computing the Infinite-Time Shortest Path Tree

Eventually, due to Lemma 4.1, any dynamic network with piece-wise linear characteristics will reach a steady state where all nodes exhibit linear departure behavior and in which the set of optimal paths to the destination will form a static shortest path tree directed into the destination node, which we denote by T_∞. We will give a modified static shortest path algorithm which determines the topology of T_∞ and the simple linear behavior of $C_i^{(nw)}(t)$ for all nodes $i \in N$ as time approaches infinity.

Proposition 6.1: *Commodities following optimal routes to the destination departing from any node as departure time approaches infinity will travel along a common static shortest path tree* T_∞

Proof: The proposition readily follows from Lemma 4.1 since, if every node $i \in N$ experiences linear departure behavior during the interval of time which extends from some time t^+ to infinity, then by definition the value of all next node functions $N_i(t)$ will be constant over this region. Furthermore, due to Lemma 4.5, we can find acyclic shortest paths. Hence, the next node pointers will define a shortest path tree directed into the destination node which is static over the interval. □

In determining T_∞, we will only need to focus on the behavior of all arc travel time functions as time approaches infinity. We may therefore concern ourselves only with the final linear piece of each arc travel time function. For simplicity of notation, let $\alpha_{ij} = \alpha(d_{ij}, P(d_{ij}))$ and $\beta_{ij} = \beta(d_{ij}, P(d_{ij}))$, so the final linear piece of each arc travel time function $d_{ij}(t)$ is given simply by $\alpha_{ij}\, t + \beta_{ij}$. Since we have linear departure behavior at each node as time approaches infinity, the optimal path travel time functions $C_i^{(nw)}(t)$ must in this limiting case be simple linear functions of time, which we denote by $C_i^{(nw)}(t) = \alpha_i t + \beta_i$. We can now rewrite the all-to-one optimality conditions based on these simple linear path travel time functions, in the limiting case as time approaches infinity. By substituting into the minimum-time all-to-one problem optimality conditions for a FIFO network given in *(6.1)*, we obtain

$$\alpha_i t + \beta_i = \begin{cases} 0 & \text{if } i = d \\ \min\limits_{j \in A_i}\{\alpha_{ij} t + \beta_{ij} + \alpha_j (t + \alpha_{ij} t + \beta_{ij}) + \beta_j\} & \text{if } i \neq d \end{cases} \qquad \forall i \in N. \qquad (6.3)$$

The optimality conditions given in *(6.3)* can be written as separate optimality conditions for each arc. For sufficiently large values of t, all simple linear node path cost functions must satisfy:

$$\alpha_i t + \beta_i \le (\alpha_{ij} + \alpha_j \alpha_{ij} + \alpha_j) t + (\beta_{ij} + \alpha_j \beta_{ij} + \beta_j) \qquad \forall (i, j) \in A,\ i \neq d, \qquad (6.4)$$

$$\alpha_d t + \beta_d = 0. \qquad (6.5)$$

Based on either of the preceding sets of optimality conditions, one can easily modify any label-setting or label-correcting static shortest path algorithm so that it computes T_∞. Instead of assigning scalar distance labels to each node, we instead represent the distance label of each node in this case by a simple linear function. There is a total ordering imposed on the set of all simple linear functions evaluated at $t = +\infty$, whereby we can compare any two linear functions by comparing first their linear coefficients, and by breaking ties by comparing constant terms. To illustrate the computation of T_∞ and its simple linear node labels, Algorithm 2 shows pseudo-code for an adapted generic label-correcting algorithm which performs this task. Other algorithms, such as Dijkstra's label-setting algorithm, may also be adapted for this purpose.

```
Initialization:
$\forall i \;\; \alpha_i \leftarrow \infty,\ \beta_i \leftarrow \infty,\ N_i \leftarrow \varnothing$
$\alpha_d \leftarrow 0,\ \beta_d \leftarrow 0$

Main Loop:
Repeat {
  Locate an arc $(i, j) \in A$ such that either:
      (i) $\alpha_i > \alpha_{ij} + \alpha_j\alpha_{ij} + \alpha_j$, or
      (ii) $\alpha_i = \alpha_{ij} + \alpha_j\alpha_{ij} + \alpha_j$ and $\beta_i > \beta_{ij} + \alpha_j\beta_{ij} + \beta_j$
  If such an arc exists, then
      $\alpha_i \leftarrow \alpha_{ij} + \alpha_j\alpha_{ij} + \alpha_j$
      $\beta_i \leftarrow \beta_{ij} + \alpha_j\beta_{ij} + \beta_j$
      $N_i \leftarrow j$
  If no such arc exists, then terminate the algorithm
}
```

Algorithm 2: Adaptation of a generic static label-correcting algorithm to compute T_∞. After termination, the topology of T_∞ is given by the next node pointers stored in the N_i values, and the simple linear travel cost from each node $i \in N$ to the destination as time approaches infinity is given by $C_i^{(nw)}(t) = \alpha_i t + \beta_i$.

6.1.2 Computing Linear Node Intervals

We proceed to show how to compute all LNIs comprising a solution using a backward scan through time. Suppose that, for some time $t_{current}$, we know the value of $C_i^{(nw)}(t)$ and $N_i(t)$ for all times $t > t_{current}$, and furthermore we know the simple linear behavior of $C_i^{(nw)}(t)$ for all times $t \in (t_{current} - \delta, t_{current}]$, where δ is taken to be small enough such that all nodes exhibit linear departure behavior over the region $(t_{current} - \delta, t_{current}]$. We know due to Proposition 4.2 that all LNIs for the all-to-one problem will span a non-zero extent of time, so it is always possible to find some such range of time $(t_{current} - \delta, t_{current}]$ during which the solution functions exhibit linear departure behavior. Using the algorithmic result of the preceding section, we can compute the simple linear behavior of functions $C_i^{(nw)}(t)$ and the constant behavior of $N_i(t)$ for sufficiently large values of t, so we initially set $t_{current}$ to $+\infty$.

The first goal of this section will be to analytically determine an earlier time $t_{new} < t_{current}$ to which one may scan backward in time, such that all nodes are guaranteed to maintain their current linear departure behavior over the region of time $t \in (t_{new}, t_{current}]$. The time t_{new} will often mark the piece-wise boundary of one or more of the functions $C_i^{(nw)}(t)$ or $N_i(t)$, and therefore also the boundary of one or more LNIs. For each node with an LNI that is closed off at time t_{new}, we will record this LNI and compute the simple linear behavior of the next LNI beginning at that node (having an upper boundary at time t_{new}). After this computation, we will know the value of $C_i^{(nw)}(t)$ and $N_i(t)$ for all $t > t_{new}$, and the simple linear behavior of these functions for all $t \in (t_{new} - \delta, t_{new}]$, and we can therefore then set $t_{current}$ to the value of t_{new} and repeat this process until t_{new} reaches $-\infty$, at which point we will have generated the complete set of LNIs characterizing an optimal solution to the minimum-time all-to-one dynamic shortest path problem in a FIFO network.

We turn our attention now to the calculation of t_{new}, given the value of $t_{current}$ and the simple linear behavior of $C_i^{(nw)}(t)$ for all $t \in (t_{current} - \delta, t_{current}]$. Based on the optimality conditions given in Equations *(6.1)* and *(6.2)*, we see that there are three possible factors which can determine the value of t_{new}. All $C_i^{(nw)}(t)$ functions will retain their simple linear behavior while reducing the value of t from $t_{current}$ as long as:

(i) the value of t remains within the same linear piece of $d_{ij}(t)$, for each arc $(i, j) \in A$,
(ii) the value of $t + d_{ij}(t)$ remains within the same linear piece of $C_j^{(nw)}(t)$, for each arc $(i, j) \in A$, and
(iii) the same optimal outgoing arc is selected for each node $i \in N$ by the minimization expression in Equations *(6.1)* and *(6.2)*.

In order to write these conditions mathematically, our algorithm will maintain as part of its current state several variables containing indices of the current linear pieces of the network data and solution functions at time $t = t_{current}$. These index variables are:

CP_i	:	Index of the current linear piece of $C_i^{(nw)}(t)$, at $t = t_{current}$,
CP_{ij}	:	Index of the current linear piece of $d_{ij}(t)$, at $t = t_{current}$,
CDP_{ij}	:	Index of the current "downstream" linear piece of $C_j^{(nw)}(t)$, at $t = a_{ij}(t_{current})$.

In terms of these index variables, we will have over the region $t \in (t_{new}, t_{current}]$:

$$C_i^{(nw)}(t) = \alpha(C_i^{(nw)}, CP_i)t + \beta(C_i^{(nw)}, CP_i) \qquad \forall i \in N \qquad (6.6)$$

$$d_{ij}(t) = \alpha(d_{ij}, CP_{ij})t + \beta(d_{ij}, CP_{ij}) \qquad \forall (i,j) \in A \qquad (6.7)$$

$$C_j^{(nw)}(a_{ij}(t)) = \alpha(C_j^{(nw)}, CDP_{ij})(a_{ij}(t)) + \beta(C_j^{(nw)}, CDP_{ij}) \qquad \forall (i,j) \in A \qquad (6.8)$$

One can then write mathematical bounds on t_{new} based on conditions *(i)*, *(ii)*, and *(iii)* as:

$$(i)\ \ t_{new} \geq B(d_{ij}, CP_{ij} - 1) \qquad \forall (i,j) \in A,\ i \neq d \qquad (6.9)$$

$$(ii)\ t_{new} \geq \left(\frac{1}{\rho_{ij}}\right)\left(B(C_j^{(nw)}, CDP_{ij} - 1) - \beta(d_{ij}, CP_{ij})\right) \qquad \forall (i,j) \in A,\ i \neq d,\ \text{and}\ \rho_{ij} > 0 \qquad (6.10)$$

$$\text{with}\ \rho_{ij} = 1 + \alpha(d_{ij}, CP_{ij})$$

$$(iii)\ t_{new} \geq \left(\frac{1}{\sigma_{ij}}\right)\begin{pmatrix}\beta(C_i^{(nw)}, CP_i) - \beta(C_j^{(nw)}, CDP_{ij}) - \\ \beta(d_{ij}, CP_{ij})(1 + \alpha(C_j^{(nw)}, CDP_{ij}))\end{pmatrix} \qquad \forall (i,j) \in A,\ i \neq d,\ \text{and}\ \sigma_{ij} > 0 \qquad (6.11)$$

$$\text{with}\ \sigma_{ij} = \alpha(d_{ij}, CP_{ij}) - \alpha(C_i^{(nw)}, CP_i) + \alpha(C_j^{(nw)}, CDP_{ij})(1 + \alpha(d_{ij}, CP_{ij}))$$

Each of the bounds given in *(i)*, *(ii)*, and *(iii)* above is a linear inequality relating the value of t to known quantities. The denominators ρ_{ij} and σ_{ij} of *(ii)* and *(iii)* can potentially be zero or negative for some arcs; if this is the case, then these conditions will impose no lower bounds on t_{new} for these arcs. A simple example of such a situation is the case where the entire network is static, in which case σ_{ij} will be zero for all arcs. If we let $\Gamma_{ij}^{(i)}$, $\Gamma_{ij}^{(ii)}$, and $\Gamma_{ij}^{(iii)}$ represent the lower bound obtainable from each of the respective conditions *(i)*, *(ii)*, and *(iii)* above for a particular arc $(i, j) \in A$, and we let $\Gamma_{ij} = max\{\Gamma_{ij}^{(i)}, \Gamma_{ij}^{(ii)}, \Gamma_{ij}^{(iii)}\}$, then can write the value of t_{new} as

$$t_{new} = \max_{(i,j)\in A}\{\Gamma_{ij}\}. \qquad (6.12)$$

We therefore have a method for computing t_{new}. In order to facilitate rapid computation, we store the Γ_{ij} value associated with each arc in a heap data structure, so the determination of t_{new} using Equation *(6.12)* requires only *O(1)* time.

After the determination of t_{new}, the behavior of the solution functions $C_i^{(nw)}(t)$ and $N_i(t)$ will be known for the region of time $t > t_{new}$. At time t_{new}, we may encounter the closure of existing LNIs at some nodes and the subsequent creation of new LNIs at these nodes to reflect the changes in the structure and linear cost of optimal paths departing at times $t \in$

$(t_{new} - \delta, t_{new}]$. The problem we now consider is that of computing the piece-wise changes to $C_i^{(nw)}(t)$ and $N_i(t)$ at time t_{new}.

Only nodes in the set $S = \{i \in N \mid \exists j \in A_i \text{ such that } \Gamma_{ij} = t_{new}\}$ are possible candidates for piece-wise linear changes at time t_{new}, since conditions imposed on arcs emanating from only these nodes were responsible for the value of t_{new}. For each node $i \in S$, we will re-compute the simple behavior of the functions $N_i(t)$ and $C_i^{(nw)}(t)$ over the interval $(t_{new} - \delta, t_{new}]$, and also update the linear piece indices for arcs emanating from these nodes to reflect the decrease in time to t_{new}. This computation requires three steps:

1. We first update the values of the current indices CP_{ij}, and CDP_{ij} for each arc $(i, j) \in A$ where $i \in S$, so that these quantities give the index of the current linear pieces of $d_{ij}(t)$ and $C_j^{(nw)}(t + d_{ij}(t))$ respectively, only at time $t = t_{new}$ rather than at $t = t_{current}$. Due to the FIFO property, these indices will not increase as t decreases from $t_{current}$ to t_{new}. We therefore successively decrement CP_{ij} and CDP_{ij} until the desired pieces of $d_{ij}(t)$ and $C_j^{(nw)}(t + d_{ij}(t))$ containing time t_{new} are found.

2. For all nodes $i \in S$ we compute the optimal outgoing arc $N_i(t)$ for $t \in (t_{new} - \delta, t_{new}]$ using the optimality conditions of Equation *(6.2)* and the fact that the optimal solution function $C_i^{(nw)}(t)$ is already known for all nodes $i \in N$ and all future times $t > t_{new}$ due to our backward transversal through time. Equation *(6.2)* cannot be directly applied, however, since a degenerate situation arises if there is a tie among several arcs emanating from the same node for the optimal outgoing arc at exactly time t_{new}. In this case, one must break the tie by selecting an arc which is also optimal at time $t_{new} - \delta$, where δ is infinitesimally small, so that this outgoing arc is then optimal for the entire region of time $(t_{new} - \delta, t_{new}]$. This method is equivalent to breaking ties based on the linear coefficients of the simple linear path travel times experienced by following each potential outgoing arc. The following equation performs this computation (we denote by *arg min** the set of all arguments which minimize an expression):

$$N_i(t) = \arg\max_{j \in \gamma_i} \{\alpha(d_{ij}, CP_{ij}) + \alpha(C_j^{(nw)}, CDP_{ij})(1+\alpha(d_{ij}, CP_{ij}))\},$$
$$where\ \gamma_i = \arg\min_{j \in A_i}{}^{*}\{d_{ij}(t_{new}) + C_j^{(nw)}(t_{new} + d_{ij}(t_{new}))\} \qquad \forall i \in S. \qquad (6.13)$$

3. After the determination of $N_i(t)$ for the interval $(t_{new} - \delta, t_{new}]$, optimal path costs may be easily obtained for this interval of time as follows:

$$C_i^{(nw)}(t) = d_{ij}(t) + C_j^{(nw)}(t + d_{ij}(t)), where\ j = N_i(t) \qquad \forall i \in S. \qquad (6.14)$$

We have therefore devised a method for computing $N_i(t)$ and $C_i^{(nw)}(t)$ for all nodes over the time interval $(t_{new} - \delta, t_{new}]$. For any node $i \in S$ which transitions to a new LNI at time t_{new}, our algorithm must record the completed LNI and initialize the new LNI at node i. Only a small amount of overhead is required for this task, as LNIs are generated in reverse chronological order for each node, and therefore may be stored and updated efficiently in either a linked list or an array data structure. Additionally, the Γ_{ij} values, maintained in a heap, must be re-computed for all $(i, j) \in A$ such that either $i \in S$ or $j \in S$, so that a value of t_{new} may be determined on the next iteration of the algorithm.

In the event that t_{new} reaches $-\infty$, we may terminate our algorithm since the functions $C_i^{(nw)}(t)$ and $N_i(t)$ will have been determined for all values of time, giving us a complete solution to the all-to-one problem. As long as t_{new} is finite, however, we compute the necessary piece-wise changes of the $C_i^{(nw)}(t)$ and $N_i(t)$ functions at time t_{new}, then set $t_{current}$ to the value of t_{new} and repeat our backward chronological scan until a complete solution is determined. Algorithm 3 gives the pseudo-code for the complete FIFO all-to-one algorithm.

```
Function Compute-Γ(i, j):
$\Gamma_{ij}^{(i)} \leftarrow \begin{cases} -\infty & if\ i = d \\ B(d_{ij}, CP_{ij} - 1) & otherwise \end{cases}$
$\rho_{ij} \leftarrow 1 + \alpha(d_{ij}, CP_{ij})$
$\Gamma_{ij}^{(ii)} \leftarrow \begin{cases} -\infty & if\ i = d\ or\ \rho_{ij} = 0 \\ (1/\rho_{ij})[B(C_j^{(nw)}, CDP_{ij} - 1) - \beta(d_{ij}, CP_{ij})] & otherwise \end{cases}$
$\sigma_{ij} \leftarrow \alpha(d_{ij}, CP_{ij}) - \alpha(C_i^{(nw)}, CP_i) + \alpha(C_j^{(nw)}, CDP_{ij})(1 + \alpha(d_{ij}, CP_{ij}))$
```

```
$\Gamma_{ij}^{(iii)} \leftarrow \begin{cases} -\infty & \text{if } i = d \text{ or } \sigma_{ij} \le 0 \\ \left(\frac{1}{\sigma_{ij}}\right)\begin{pmatrix} \beta(C_i^{(nw)}, CP_i) - \beta(C_j^{(nw)}, CDP_{ij}) - \\ \beta(d_{ij}, CP_{ij})(1+\alpha(C_j^{(nw)}, CDP_{ij})) \end{pmatrix} & \text{otherwise} \end{cases}$
$\Gamma_{ij} \leftarrow max\{\Gamma_{ij}^{(i)}, \Gamma_{ij}^{(ii)}, \Gamma_{ij}^{(iii)}\}$

Function Update-Piece-Indices(i, j):
While($t_{new} \notin (B(d_{ij}, CP_{ij} - 1), B(d_{ij}, CP_{ij})]$)
  Decrement $CP_{ij}$
While($(1+\alpha(d_{ij}, CP_{ij}))\, t_{new} + \beta(d_{ij}, CP_{ij}) \notin (B(C_j^{(nw)}, CDP_{ij} - 1), B(C_j^{(nw)}, CDP_{ij})]$)
  Decrement $CDP_{ij}$

Function Compute-New-LNI(i):
$\gamma_i \leftarrow \underset{j \in A_i}{\arg\min^*}\{d_{ij}(t_{new}) + C_j^{(nw)}(t_{new} + d_{ij}(t_{new}))\}$,
  where $d_{ij}(t) = \alpha(d_{ij}, CP_{ij})\, t + \beta(d_{ij}, CP_{ij})$,
  and $C_j^{(nw)}(t) = \alpha(C_j^{(nw)}, CDP_{ij})\, t + \beta(C_j^{(nw)}, CDP_{ij})$
$j \leftarrow \underset{j \in \gamma_i}{\arg\max}\{\alpha(d_{ij}, CP_{ij}) + \alpha(C_j^{(nw)}, CDP_{ij})(1+\alpha(d_{ij}, CP_{ij}))\}$
$a \leftarrow \alpha(d_{ij}, CP_{ij}) + \alpha(C_j^{(nw)}, CDP_{ij})(1 + \alpha(d_{ij}, CP_{ij}))$
$b \leftarrow (1 + \alpha(C_j^{(nw)}, CDP_{ij}))\beta(d_{ij}, CP_{ij}) + \beta(C_j^{(nw)}, CDP_{ij})$
If $j \ne N_i(B(C_i^{(nw)}, CP_i))$ or $a \ne \alpha(C_i^{(nw)}, CP_i)$ or $b \ne \beta(C_i^{(nw)}, CP_i)$ then {
  Decrement $CP_i$
  $B(C_i^{(nw)}, CP_i) \leftarrow t_{new}$, $B(C_i^{(nw)}, CP_i - 1) \leftarrow -\infty$
  $N_i(t_{new} < t \le t_{current}) \leftarrow j$, $\alpha(C_i^{(nw)}, CP_i) \leftarrow a$, $\beta(C_i^{(nw)}, CP_i) \leftarrow b$
  For all $i' \in B_i$ : Compute-$\Gamma(i', i)$, Update-Heap($H$, $\Gamma_{i'i}$)
}
For all $j \in A_i$ : Compute-$\Gamma(i, j)$, Update-Heap($H$, $\Gamma_{ij}$)

Initialization:
$\forall i \in N : CP_i \leftarrow 0$
$\forall (i, j) \in A : CDP_{ij} \leftarrow 0,\ CP_{ij} \leftarrow P(d_{ij})$
$\{\alpha(C^{(nw)}, 0), \beta(C^{(nw)}, 0), N(t = +\infty)\} \leftarrow$ Algorithm 2
$\forall i \in N : B(C_i^{(nw)}, -1) \leftarrow -\infty,\ B(C_i^{(nw)}, 0) \leftarrow +\infty$
$\forall (i, j) \in A$ : Compute-$\Gamma(i, j)$
$H \leftarrow$ Create-Heap($\Gamma$)
$t_{current} \leftarrow +\infty$
```

```
Main Loop:
While($t_{current} \neq -\infty$) {
  $t_{new} \leftarrow$ Heap-Max(H)
  $S \leftarrow \{i \in N \mid \exists j \in A_i$ such that $\Gamma_{ij} = t_{new}\}$
  For all $i \in S, j \in A_i$
    Update-Piece-Indices(i, j)
  For all $i \in S$
    Compute-New-LNI(i)
  $t_{current} \leftarrow t_{new}$
}

Termination:
For all $i \in N$
  $P(C_i^{(nw)}) \leftarrow 1 - CP_i$
  Renumber the pieces of $C_i^{(nw)}$ from 1.. $P(C_i^{(nw)})$
```

Algorithm 3: Reverse-chronological scan algorithm which solves the minimum-time all-to-one dynamic shortest path problem in a FIFO network.

During the execution Algorithm 3, the linear pieces of the $C_i^{(nw)}(t)$ functions are indexed by the decreasing integers $\{0, -1, -2, \ldots\}$ as they are generated, and renumbered from 1 through $P(C_i^{(nw)})$ at the end of the algorithm once $P(C_i^{(nw)})$ has finally been computed. Standard auxiliary heap functions are assumed to exist, in order to maintain a heap containing the values of the Γ_{ij} bounds.

Proposition 6.2: *The all-to-one chronological scan algorithm correctly solves the minimum-time all-to-one dynamic shortest path problem in a FIFO network.*

Proof: To argue correctness, we assert that the algorithm, by design, produces only pieces of the solution functions $C_i^{(nw)}(t)$ and $N_i(t)$ which satisfy the FIFO optimality conditions given in Equations *(6.1)* and *(6.2)*. The algorithm maintains the invariant that, for $t > t_{current}$, the optimal values of both of these solution functions have already been computed. Each iteration of the main loop decreases $t_{current}$ by some positive quantity (due to Proposition 4.2), until $t_{current}$ reaches $-\infty$ and a complete solution has been determined. □

An analysis of the worst-case and average-case running time of the FIFO minimum-time all-to-one chronological scan algorithm appears in Section 6.4, and the results of computational testing of this algorithm may be found in Chapter 8.

6.2 General All-to-One Solution Algorithm

In this section and the next we develop generalizations of the reverse-chronological scan algorithm which solve any all-to-one or one-to-all dynamic shortest path problem within the framework under consideration in this thesis. These more general algorithms are based on the same methodology as the simpler FIFO all-to-one algorithm, and consequentially we will simplify the discussion of these new algorithms by referring back at times to results presented in the previous subsection.

We begin by developing a general all-to-one chronological scan algorithm which solves any all-to-one dynamic shortest path problem. The algorithm proceeds in two phases, exactly as its simpler FIFO counterpart. Initially, an "infinite-time" tree is computed which gives the structure and cost of optimal paths departing at sufficiently large values of t. Next, a backward scan through time is performed in order to compute the LNIs comprising the solution functions – it is this backward scan through time that is substantially more complicated for the general algorithm. The full set of solution functions $C_i^{(w)}(t)$, $C_i^{(nw)}(t)$, $N_i(t)$, and $W_i(t)$ will be computed as output of this algorithm. In Section 6.4, we will analyze the worst-case and expected performance of the general all-to-one chronological scan algorithm. If the general all-to-one chronological scan algorithm is used to compute minimum-time paths through a FIFO network, then the operations of the algorithm will reduce exactly to those performed by the simpler FIFO variant of the algorithm, since the simpler algorithm is merely as subset of the general algorithm. The performance of the general algorithm applied to find minimum-time paths in a FIFO network will therefore be the same as that of the minimum-time FIFO all-to-one algorithm.

6.2.1 Simplification of Waiting Constraints

We briefly discuss a network transformation which will simplify the computation of optimal paths in the presence of waiting constraints and costs. This subsection only applies to problems which involve waiting. Recall that in this case we assume that all network data functions are continuous. By re-writing the "waiting" optimality condition *(4.1)* of the all-to-one problem, we have

$$C_i^{(w)}(t) = \min_{\tau \in WT_i(t)} \{cwc_i(t+\tau) + C_i^{(nw)}(t+\tau)\} - cwc_i(t) \qquad \forall i \in N. \qquad (6.15)$$

If we let $WCF_i(t)$ denote the *waiting cost function* $cwc_i(t) + C_i^{(nw)}(t)$, we can write *(6.15)* as

$$C_i^{(w)}(t) = \min_{0 \le \tau \le ubw_i(t)} \{WCF_i(t+\tau)\} - cwc_i(t) \qquad \forall i \in N. \qquad (6.16)$$

Since the waiting cost function of each node is continuous and piece-wise linear, the minimum of this function in Equation *(6.16)* will occur either:

(i) at one of the endpoints of the region of minimization, where $\tau = 0$ or $\tau = ubw_i(t)$, or

(ii) at a time $t + \tau$ which is a piece-wise boundary of $WCF_i(t + \tau)$, where $t < t + \tau < t + ubw_i(t)$.

In order to eliminate the possibility of option *(i)*, we perform the following network transformation: replace each node i at which waiting is allowed with 3 virtual nodes i_{nw}, i_{mw}, and i_w, which represent respectively a "non-waiting node", a "maximum waiting node", and a "waiting node" corresponding to node i. Each of these virtual nodes are to be linked to the same neighboring nodes as the original node i. We want the non-waiting node i_{nw} to behave like node i, except that commodities traveling through this virtual node are prohibited from waiting and must immediately depart along an outgoing arc. Similarly, commodities traveling through the maximum waiting node i_{mw} are required to wait for the maximum possible allowable length of time before departure on an outgoing arc – this restriction is enforced by prohibiting waiting at the node i_{mw} and by adding $ubw_i(t)$ to the travel time function of each outgoing arc. Commodities traveling through the waiting node i_w are required to wait and depart only at times $t + \tau$ which are piece-wise boundaries of the function $WCF_i(t + \tau)$, as in option *(ii)* above. For each waiting node, we only allow departures from times which are within the closed interval $[t, t + ubw_i(t)]$ following an arrival at time t. Although option *(ii)* specifies an open interval, the fact that we allow departures from waiting nodes from a closed interval of time will not affect the final solution produced by the algorithm.

This transformation maps each node i at which waiting is allowed into three nodes, two of which no longer permit waiting, and one of which permits a restricted form of waiting; however, for the virtual waiting node i_w which permits waiting, only option *(ii)* above needs to be considered when selecting a minimum waiting time, since the two other virtual nodes account for the possibilities comprising option *(i)*. The optimal path travel cost function for node i as a whole, $C_i^{(w)}(t)$, will be given by the minimum of $C_{i_{nw}}^{(nw)}(t), C_{i_{mw}}^{(nw)}(t)$, and $C_{i_w}^{(w)}(t)$. We can therefore simplify the computation of $C_i^{(w)}(t)$, since these three virtual optimal path functions will be easier to compute individually.

After the transformation, our network will contain two types of nodes: nodes i for which waiting is prohibited, for which $C_i^{(w)}(t) = C_i^{(nw)}(t)$, and nodes j where waiting is allowed, but for which $C_j^{(w)}(t)$ can be computed in a simple fashion based on option *(ii)* above. We will call the set of transformed nodes where waiting is prohibited *non-waiting nodes*, and denote this set by N_{nw}. Similarly, we call the set of nodes at which special restricted waiting is allowed *waiting nodes*, and we denote this set by N_w. Additionally, we describe each piece-wise boundary time of the function $WCF_i(t + \tau)$ where $\tau \in [0, ubw_i(t)]$ as a *feasible waiting departure time*, and we let $FWDT_i(t)$ denote the set of all feasible waiting departure times from a waiting node i, where waiting begins at time t. The optimal path cost $C_i^{(w)}(t)$ for each waiting node $i \in N_w$ is obtained by selecting the best feasible waiting departure time:

$$C_i^{(w)}(t) = \min_{\tau \in FWDT_i(t)} \{WCF_i(\tau)\} - cwc_i(t) \qquad \forall i \in N_w. \qquad (6.17)$$

It will be of importance to our chronological scan approaches that the minimization expression in *(6.17)* yields the same minimum value as the value of t increases or decreases slightly, provided that the set of feasible waiting departure times $FWDT_i(t)$ does not change.

We henceforth assume that we are dealing with a transformed network, in which each node is either a non-waiting node or a waiting node. From an implementation standpoint, the transformation may be performed either explicitly or implicitly. The latter of these choices may be preferable since it will reduce storage requirements. After the execution of the algorithms described in the next two sections, it will be necessary to perform a straightforward reverse transformation to recover the solution functions $C_i^{(w)}(t)$, $C_i^{(nw)}(t)$, $N_i(t)$, and $W_i(t)$ for the original network.

6.2.2 Computing the Infinite-Time Shortest Path Tree

Using Lemma 4.1 and the same reasoning as in Section 6.1.1, one may argue that as time approaches infinity, the union of optimal paths departing from each node $i \in N$ will form a static shortest path tree, which we denote T_∞. We will develop an algorithm for computing T_∞. This algorithm will be very similar to the analogous algorithm presented in Section 6.1.1.

Since we may restrict our focus to only the final linear pieces of each arc travel time and travel cost function, for simplicity of notation let $\alpha_{ij}^{(c)} = \alpha(c_{ij}, P(c_{ij}))$ and $\beta_{ij}^{(c)} = \beta(c_{ij}, P(c_{ij}))$, and also let $\alpha_{ij}^{(d)} = \alpha(d_{ij}, P(d_{ij}))$ and $\beta_{ij}^{(d)} = \beta(d_{ij}, P(d_{ij}))$. As t approaches infinity we will then have $c_{ij}(t) = \alpha_{ij}^{(c)}t + \beta_{ij}^{(c)}$ and $d_{ij}(t) = \alpha_{ij}^{(d)}t + \beta_{ij}^{(d)}$. Since waiting is never beneficial along paths departing at times larger than t^+ (given by Lemma 4.1), we will have $W_i(t) = 0$ and $C_i^{(w)}(t) = C_i^{(nw)}(t)$ for all times $t > t^+$. Furthermore, since each node will exhibit linear departure behavior over this interval, we can write the optimal path travel time functions for $t > t^+$ as simple linear functions of time, which we denote by $C_i^{(w)}(t) = C_i^{(nw)}(t) = \alpha_i t + \beta_i$. Substituting into the general all-to-one optimality conditions given in Equation *(4.2)*, we obtain the following set of new optimality conditions for determining the topology and simple linear node costs of T_∞:

$$\alpha_i t + \beta_i = \begin{cases} 0 & \text{if } i = d \\ \min\limits_{j \in A_i}\{\alpha_{ij}^{(c)}t + \beta_{ij}^{(c)} + \alpha_j(t + \alpha_{ij}^{(d)}t + \beta_{ij}^{(d)}) + \beta_j\} & \text{if } i \neq d \end{cases} \quad \forall i \in N. \qquad (6.18)$$

The optimality conditions given in *(6.18)* can be written as separate optimality conditions for each arc. For sufficiently large values of t, all simple linear node path cost functions must satisfy:

$$\alpha_i t + \beta_i \leq (\alpha_{ij}^{(c)} + \alpha_j \alpha_{ij}^{(d)} + \alpha_j)t + (\beta_{ij}^{(c)} + \alpha_j \beta_{ij}^{(d)} + \beta_j) \qquad \forall (i,j) \in A,\ i \neq d \quad (6.19)$$

$$\alpha_d t + \beta_d = 0 \quad (6.20)$$

Based on either of the preceding sets of optimality conditions, one can easily modify any label-setting or label-correcting static shortest path algorithm so that it computes T_∞. The pseudo-code of Algorithm 2 from Section 6.1.1 may be trivially modified to perform this computation.

6.2.3 Computing Linear Node Intervals

As in the FIFO all-to-one algorithm, we proceed to compute all LNIs using a backward chronological scan through time. Suppose that, for some time $t_{current}$, we know the value of $C_i^{(w)}(t)$, $C_i^{(nw)}(t)$, $N_i(t)$, and $W_i(t)$ for all values of time $t > t_{current}$. Furthermore suppose that we know the simple behavior of these functions for all times $t \in (t_{current} - \delta, t_{current}]$, where δ is taken to be small enough such that all nodes exhibit linear departure behavior over the region $(t_{current} - \delta, t_{current}]$. Initially, we can take $t_{current}$ to be $+\infty$, and we can set the initial values of all solution functions using the topology and linear node costs obtained from the computation if T_∞ in the previous section.

As in the FIFO case, we proceed to compute an earlier time $t_{new} < t_{current}$ to which one can move in time, such that all nodes are guaranteed to maintain their current linear departure behavior over the region of time $t \in (t_{new}, t_{current}]$. For any node with an LNI that is closed off at time t_{new}, we will record this LNI and compute the next LNI beginning at that node. After this computation, we will know the value of the solution functions for all $t > t_{new}$, and their simple behavior for all $t \in (t_{new} - \delta, t_{new}]$, and we can then set $t_{current}$ to the value of t_{new}. Repeating this process until $t_{new} = -\infty$ will generate the complete set of LNIs characterizing an optimal solution.

Based on the optimality conditions *(4.1)* and *(4.2)* of the all-to-one problem, and the discussion of waiting cost simplification given in Section 6.2.1, we can show that there are seven conditions which determine bounds on the value of t_{new}. All solution functions will retain their simple linear behavior while reducing the value of t from $t_{current}$ as long as:

<ins>Basic Conditions:</ins>

(i) the value of t remains within the same linear piece of $d_{ij}(t)$, for each arc $(i, j) \in A$,

(ii) the value of t remains within the same linear piece of $c_{ij}(t)$, for each arc $(i, j) \in A$,

(iii) the value of $t + d_{ij}(t)$ remains within the same linear piece of $C_j^{(w)}(t)$, for each arc $(i, j) \in A$,

(iv) the optimal immediate departure outgoing arc $(i, N_i(t_{current}))$ is selected for each node $i \in N$ by the minimization expressions in Equation *(4.2)*,

<ins>Waiting Conditions:</ins>

(v) the value of t remains within the same linear piece of $ubw_i(t)$ for all waiting nodes,

(vi) the value of t remains within the same linear piece of $cwc_i(t)$ for all waiting nodes, and

(vii) for all waiting nodes, the interval of feasible waiting at that node must contain the same set of feasible waiting departure times as $FDWT_i(t_{current})$, so Equation *(6.17)* will produce a simple linear function.

As with the FIFO all-to-one algorithm, we maintain a set of variables for each node and arc which will hold indices into the current linear pieces of various network data and solution functions. These index variables are listed below; the set of such variables we must maintain in the general case is quite a bit more extensive than that in the FIFO case.

Node Index Variables:

$CP_i^{(w)}$	:	Index of the current linear piece of $C_i^{(w)}(t)$, at $t = t_{current}$.
$CP_i^{(nw)}$	:	Index of the current linear piece of $C_i^{(nw)}(t)$, at $t = t_{current}$.
$CP_i^{(ubw)}$	:	Index of the current linear piece of $ubw_i(t)$, at $t = t_{current}$.
$CP_i^{(cwc)}$	:	Index of the current linear piece of $cwc_i(t)$, at $t = t_{current}$.
$CWP_i^{(C)}$	:	Index of the current "maximal waiting" piece of $C_i^{(nw)}(t + ubw_i(t))$ at $t = t_{current}$.
$CWP_i^{(cwc)}$	:	Index of the current "maximal waiting" piece of $cwc_i(t + ubw_i(t))$ at $t = t_{current}$.

Arc Index Variables:

$CP_{ij}^{(d)}$	:	Index of the current linear piece of $d_{ij}(t)$, at $t = t_{current}$.
$CP_{ij}^{(c)}$	:	Index of the current linear piece of $c_{ij}(t)$, at $t = t_{current}$.
CDP_{ij}	:	Index of the current "downstream" linear piece of $C_j^{(w)}(t)$ at $t = t_{current} + d_{ij}(t_{current})$.

In addition to these indices, we will maintain for each node i the set of feasible waiting departure times $FWDT_i(t)$ at time $t_{current}$. This set of waiting times is implemented as a queue for the following reason: the set $FWDT_i(t)$ represents all piece-wise boundary points of the waiting cost function $C_i^{(nw)}(\tau) + cwc_i(\tau)$ over the contiguous window of time $\tau \in [t, t + ubw_i(t)]$. As we decrease t over the course of the algorithm, the lower endpoint of this window will decrease, and the upper endpoint will either remain constant or decrease in time; as the algorithm progresses, we will therefore add new piece-wise boundaries to $FWDT_i(t)$ as they enter the lower end of this window, and we will remove these boundaries when they exit from the upper end of the window. In addition to feasible waiting departure times, the value at each boundary point of the waiting cost function $C_i^{(nw)}(\tau) + cwc_i(\tau)$ is stored along with its corresponding waiting departure time τ in the queue $FWDT_i(t)$.

We can write the following mathematical bounds on t_{new} based on conditions *(i)...(vii)* above.

$$(i)\quad t_{new} \geq B(d_{ij}, CP_{ij}^{(d)} - 1) \qquad \forall (i, j) \in A,\ i \neq d \qquad (6.21)$$

$$(ii)\quad t_{new} \geq B(c_{ij}, CP_{ij}^{(c)} - 1) \qquad \forall (i, j) \in A,\ i \neq d \qquad (6.22)$$

$$(iii)\ t_{new} \geq \begin{cases} (1/\rho_{ij})[B(C_j^{(w)}, CDP_{ij}-1) - \beta(d_{ij}, CP_{ij}^{(d)})] & if\ \rho_{ij} > 0 \\ (1/\rho_{ij})[B(C_j^{(w)}, CDP_{ij}) - \beta(d_{ij}, CP_{ij}^{(d)})] & if\ \rho_{ij} < 0 \end{cases} \quad \forall (i,j) \in A,\ i \neq d,\ and\ \rho_{ij} \neq 0 \qquad (6.23)$$

$$with\ \rho_{ij} = 1 + \alpha(d_{ij}, CP_{ij}^{(d)})$$

$$(iv)\ t_{new} \geq \left(\frac{1}{\sigma_{ij}} \left(\begin{array}{l} \beta(C_i^{(nw)}, CP_i^{(nw)}) - \beta(C_j^{(nw)}, CDP_{ij}) - \\ \beta(d_{ij}, CP_{ij}^{(d)}) \alpha(C_j^{(w)}, CDP_{ij}) - \beta(c_{ij}, CP_{ij}^{(c)}) \end{array} \right) \right) \quad \forall (i,j) \in A,\ i \neq d,\ and\ \sigma_{ij} > 0 \qquad (6.24)$$

$$with\ \sigma_{ij} = \alpha(c_{ij}, CP_{ij}^{(c)}) - \alpha(C_i^{(nw)}, CP_i^{(nw)}) + \alpha(C_j^{(w)}, CDP_{ij})(1 + \alpha(d_{ij}, CP_{ij}^{(d)}))$$

$$(v)\ t_{new} \geq B(ubw_i, CP_i^{(ubw)} - 1) \qquad \forall i \in N_w \qquad (6.25)$$

$$(vi)\ t_{new} \geq B(cwc_i, CP_i^{(cwc)} - 1) \qquad \forall i \in N_w \qquad (6.26)$$

$$(vii)\ t_{new} \geq \frac{\max\left\{ \begin{array}{l} B(C_i^{(nw)}, CWP_i^{(C)} - 1), \\ B(cwc_i, CWP_i^{(cwc)} - 1) \end{array} \right\} - \beta(ubw_i, CP_i^{(ubw)})}{\mu_i} \qquad \forall i \in N_w,\ and\ \mu_i > 0 \qquad (6.27)$$

$$with\ \mu_i = 1 + \alpha(ubw_i, CP_i^{(ubw)})$$

Each of the bounds given in *(i)...(vii)* above is a linear inequality relating the value of t_{new} to known quantities. If the denominator ρ_{ij} in *(iii)* is zero or if the denominator σ_{ij} in *(iv)* is non-positive for some arcs, then these arcs impose no lower bound on t_{new}. Similarly, if the denominator μ_i in *(vii)* is zero for some nodes, then these nodes will impose no lower bound on t_{new}. We denote by $\Gamma_{ij}^{(i)}...\Gamma_{ij}^{(iv)}$ the tightest lower bounds obtainable from each of the respective conditions *(i)...(iv)* above for a particular arc $(i, j) \in A$, and we let Γ_{ij} be equal to $max\{\Gamma_{ij}^{(i)}...\Gamma_{ij}^{(iv)}\}$. Similarly, we denote by $\Gamma_i^{(v)}...\Gamma_i^{(vii)}$ the tightest lower bounds obtainable from each of the respective conditions *(v)...(vii)* above for a particular waiting node $i \in N_w$, and we let Γ_i denote the expression $max\{\Gamma_i^{(v)}...\Gamma_i^{(vii)}\}$. We can then write the value of t_{new} as

$$t_{new} = \max\{\max_{(i,j)\in A}\{\Gamma_{ij}\}, \max_{i \in N_w}\{\Gamma_i\}\}. \qquad (6.28)$$

In order to facilitate rapid computation, we store the Γ_{ij} and Γ_i values associated with each arc and waiting node in a heap data structure, so the determination of t_{new} requires only *O(1)* time. After the determination of t_{new}, the behavior of the solution functions $C_i^{(w)}(t)$, $C_i^{(nw)}(t)$, $N_i(t)$, and $W_i(t)$ will be known for the region of time $t > t_{new}$. At time

t_{new}, we may encounter the closure of existing LNIs at some nodes and the subsequent creation of new LNIs at these nodes to reflect the changes in the structure and linear cost of optimal paths departing at times $t \in (t_{new} - \delta, t_{new}]$, for δ sufficiently small. We now consider the problem of computing the piece-wise changes to the solution functions at time t_{new}.

Only nodes in the set $S = \{i \in N \mid \Gamma_i = t_{new}, \text{ or } \exists j \in A_i \text{ such that } \Gamma_{ij} = t_{new}\}$ are possible candidates for piece-wise linear changes at time t_{new}, since conditions imposed on these nodes or on arcs emanating from only these nodes were responsible for the value of t_{new}. For each node $i \in S$, we therefore re-compute the simple behavior of the solution functions over the interval $(t_{new} - \delta, t_{new}]$ using the following five steps:

1. We first update the values of the arc index variables $CP_{ij}^{(d)}$, $CP_{ij}^{(c)}$, and CDP_{ij} for all arcs $(i, j) \in A$ where $i \in S$, so that they correspond to time $t = t_{new}$ rather than at $t = t_{current}$. The update will involve at most a single decrement operation for the indices $CP_{ii}^{(d)}$, $CP_{ij}^{(c)}$; however, updating the index CDP_{ij} may involve several successive decrements or increments until the desired linear piece of $C_j^{(nw)}(t + d_{ij}(t))$ is located, depending on whether or not $d_{ij}(t)$ is discontinuous at time t_{new}. In a FIFO network, it will only be necessary to decrement these indices.

2. For all nodes $i \in S$ we compute the optimal outgoing arc $N_i(t)$ for $t \in (t_{new} - \delta, t_{new}]$ using the optimality conditions of Equation *(4.2)* and the fact that the optimal solution functions $C_i^{(w)}(t)$ is already known for all nodes $i \in N$ and all future times $t > t_{new}$. As in the FIFO case, a degenerate situation arises if there is a tie between several arcs emanating from the same node for the optimal outgoing arc at exactly time t_{new}, and this tie must be broken by the linear coefficients of the simple linear path travel times experienced by following each potential outgoing arc:

$$\begin{aligned} &N_i(t) = \arg\max_{j \in \gamma_i}\{\alpha(c_{ij}, CP_{ij}^{(c)}) + \alpha(C_j^{(w)}, CDP_{ij})(1 + \alpha(d_{ij}, CP_{ij}^{(d)}))\}, \\ &\textit{where } \gamma_i = \arg\min_{j \in A_i}{}^{*}\{c_{ij}(t_{new}) + C_j^{(w)}(t_{new} + d_{ij}(t_{new}))\} \end{aligned} \qquad \forall i \in S. \qquad (6.29)$$

3. After the determination of $N_i(t)$ for the interval $(t_{new} - \delta, t_{new}]$, optimal path costs for immediate departure, $C_i^{(nw)}(t)$, may be easily obtained for this interval of time as follows:

$$C_i^{(nw)}(t) = d_{ij}(t) + C_j^{(w)}(t + d_{ij}(t)), where\ j = N_i(t) \qquad \forall i \in S. \qquad (6.30)$$

4. For each waiting node $i \in S \cap N_w$, we compute changes to the set of feasible departure waiting times $FDWT_i(t)$ at time t_{new}. Due to condition *(vii)*, this set of times must remain unchanged over the interval of departure times $(t_{new}, t_{current}]$. There may, however, be changes to this set at time t_{new}: the time t_{new} is added as a new feasible departure waiting time if $\Gamma_i^{(v)} = t_{new}$ or $\Gamma_i^{(vi)} = t_{new}$, and time $t_{new} + ubw_i(t_{new})$ is removed if $\Gamma_i^{(vii)} = t_{new}$. This update takes $O(1)$ computation time since the set $FDWT_i(t)$ is implemented as a queue.

5. Finally, for each waiting node $i \in S \cap N_w$ we can determine optimal path costs where waiting is allowed, $C_i^{(w)}(t)$, for $t \in (t_{new} - \delta, t_{new}]$ and the corresponding optimal waiting times $W_i(t)$ over this interval using equation *(6.17)*. To do this, for each node $i \in S$ we select the most optimal feasible departure waiting time τ_i from the set of possibilities given in the set $FWDT_i(t_{new})$:

$$\tau_i = \underset{\tau \in FWDT_i(t_{new})}{\arg\min} \{cwc_i(\tau) + C_i^{(nw)}(\tau)\} \qquad \forall i \in S \cap N_w \qquad (6.31)$$

This minimization operation can be made to take only $O(1)$ amortized time if each queue of feasible waiting departure times is implemented as an augmented queue which we call a *min-queue*, described in Appendix A. For the interval of time $t \in (t_{new} - \delta, t_{new}]$, we will then have:

$$W_i(t) = \tau_i - t \qquad \forall i \in S \cap N_w, \qquad (6.32)$$
$$C_i^{(w)}(t) = cwc_i(t + W_i(t)) - cwc_i(t) + C_i^{(nw)}(t + W_i(t)) \qquad \forall i \in S \cap N_w. \qquad (6.33)$$

We have therefore a method for computing $C_i^{(w)}(t)$, $C_i^{(nw)}(t)$, $N_i(t)$, and $W_i(t)$ for all nodes over the time interval $(t_{new} - \delta, t_{new}]$. For any node $i \in S$ which transitions to a new LNI at time t_{new}, our algorithm must record the completed LNI and initialize a new LNI at node i. Only a small amount of overhead is required for this task, as LNIs are generated in chronological order for each node, and therefore may be stored and updated efficiently in either a linked list or an array data structure. Additionally, the Γ_{ij} values must be re-computed for all $(i, j) \in A$ such that $i \in S$ or $j \in S$, and the Γ_i values must be re-computed for all $i \in S$, so that a new value of t_{new} may be determined on the next iteration of the algorithm.

In the event that t_{new} reaches $-\infty$, we may terminate our algorithm. As long as t_{new} is finite, however, we compute the necessary piece-wise changes of the solution functions at time t_{new}, then set $t_{current}$ to the value of t_{new} and repeat our backward chronological scan until a complete solution is determined.

Proposition 6.3: *The all-to-one chronological scan algorithm correctly solves the minimum-cost all-to-one dynamic shortest path problem.*

Justification of Proposition 6.3 is identical to that of Proposition 6.2, and is therefore omitted.

6.3 General One-to-All Solution Algorithm

We next develop a symmetric chronological scan algorithm for solving any general minimum-cost one-to-all dynamic shortest path problem. In contrast with the all-to-one algorithm, the one-to-all algorithm works be scanning forward through time, starting from an initial static shortest path tree computed at time $t = -\infty$. A fundamental difference in the operation of the algorithm is that it is necessary to compute the solution functions $C_i^{(w)}(t)$, $C_i^{(nw)}(t)$, $PN_i(t)$, $PT_i(t)$, and $W_i(t)$ not at the source end of each arc and at the current time $t_{current}$ maintained by the chronological scan, but at the destination (downstream) end of each arc and at some time in the future determined by the arc arrival time functions. This situation adds considerably more complexity to the solution algorithm, since it allows for LNIs to be generated and modified in an arbitrary order rather than in chronological order in non-FIFO networks. The general one-to-all problem also presents the only occasion in which singular LNIs may exist – this fact will further complicate our solution algorithm.

6.3.1 Simplification of Waiting Constraints

In order to simply the computation of optimal paths in the case where waiting is allowed, we assume that the network transformation described in Section 6.2.1 has been performed, so that there are now two types of nodes: non-waiting nodes, from which immediate departure is required, and waiting nodes, which permit a restricted form of waiting. Recall also that all network data functions are assumed to be continuous in networks for which waiting is allowed. As discussed in Section 6.2.1, when determining an optimal waiting time at a waiting node for the all-to-one algorithm, one need only consider a finite set of feasible waiting departure times which constitute the piece-wise boundaries of the waiting cost function $C_i^{(nw)}(t) + cwc_i(t)$ within a feasible range of waiting times. In the one-to-all algorithm, we will instead be minimizing over previous arrival times, so we maintain a set of feasible waiting arrival times, constructed as follows: for any particular time $t_{current}$, the optimal waiting path cost value $C_i^{(w)}(t)$ at the time $t = t_{current}$ is given by the following minimization expression over previous times:

$$C_i^{(w)}(t) = \min_{\{\tau | 0 \le t-\tau \le ubw_i(\tau)\}} \{C_i^{(nw)}(\tau) - cwc_i(\tau)\} + cwc_i(t) \qquad \forall i \in N_w. \qquad (6.34)$$

Due to the FIFO property, the set $\{\tau \mid 0 \le t - \tau \le ubw_i(\tau)\}$ will be a contiguous window of time, and the endpoints of this window will be non-decreasing functions of t. For convenience, we will compute an auxiliary function $EWAT_i(t)$ for each $i \in N_w$, which represents the *earliest waiting arrival time* from which one may start waiting at node i, such that one may depart by or after time t:

$$EWAT_i(t) = \min_{\{\tau | 0 \le t-\tau \le ubw_i(\tau)\}} \{\tau\} \qquad \forall i \in N_w. \qquad (6.35)$$

Computation of the $EWAT_i(t)$ functions for all nodes is performed in much the same way as a FIFO inverse is computed, by a single scan over the linear pieces of the $ubw_i(t)$ functions for each node. After this computation, the set $\{\tau \mid 0 \le t - \tau \le ubw_i(\tau)\}$ over which minimization takes place in *(6.34)* may be expressed as the interval $[EWAT_i(t), t]$. Due to continuity of network data functions, the minimum in *(6.34)* over this interval will be attained either at an endpoint of this interval or at a piece-wise boundary point within this interval of the *waiting cost function*, defined as $WCF_i(t) = C_i^{(nw)}(t) - cwc_i(t)$ for the one-to-all problem. However, due to the simplifying transformation of Section 6.2.1, it is only necessary to consider piece-wise boundary points when computing this minimum for each waiting node. If we denote by $FWAT_i(t)$ the set of *feasible waiting arrival times* for node i and time t, defined as all piece-wise boundaries of $WCF_i(t)$ within the interval of time $[EWAT_i(t), t]$, then we can re-write *(6.34)* as follows:

$$C_i^{(w)}(t) = \min_{\tau \in FWAT_i(t)} \{WCF_i(\tau)\} + cwc_i(t) \qquad \forall i \in N_W. \qquad (6.36)$$

It will be important to our chronological scan approach that the minimization expression in *(6.36)* evaluates to the same result upon slight increases of the value of t, provided that the set of feasible waiting arrival times $FWAT_i(t)$ does not change. This will ensure that $C_i^{(w)}(t)$ will behave as a simple linear function as we scan over any duration of time during which $FWAT_i(t)$ does not change.

6.3.2 Computing the Infinite-Time Shortest Path Tree

As in the all-to-one chronological scan algorithm, we initialize all solution functions by computing a static shortest path tree as time approaches infinity. In the one-to-all, case, however, this tree will be computed at $t = -\infty$, since we will then scan forward through time to generate the remainder of the solution. Since we assume the network becomes static as time approaches $-\infty$, we may generate this tree, which we call $T_{-\infty}$, and its associated constant node cost functions, by simply applying a static shortest path algorithm using the initial static travel cost $\beta(c_{ij}, 1)$ as the cost of each arc $(i, j) \in A$. Since waiting is never beneficial along paths arriving at times earlier than t^- (given by Lemma 4.1), we have $W_i(t) = 0$ and $C_i^{(w)}(t) = C_i^{(nw)}(t)$ for sufficiently small t. The solution functions $PN_i(t)$ and $PT_i(t)$ are both easily determined from the structure and optimal path costs of $T_{-\infty}$. We can therefore determine the behavior of all solution functions for sufficiently small (negative) values of t.

6.3.3 Computing Linear Node Intervals

The one-to-all chronological scan algorithm computes the LNIs comprising all solution functions using a forward scan through time. We will keep track of the current time during the scan, $t_{current}$, initially $-\infty$, and each iteration we will increase $t_{current}$ by some positive quantity until it reaches $+\infty$ and a complete optimal solution has been determined. We will assume prior to each chronological time-step by induction that the functions $C_i^{(w)}(t)$, $C_i^{(nw)}(t)$, $W_i(t)$, $PN_i(t)$, and $PT_i(t)$ have been correctly computed for all times $t \leq t_{current}$.

In addition to computing the solution function $C_i^{(nw)}(t)$ for each node $i \in N$, we will compute additional solution functions for each arc $(i, j) \in A$, denoted by $C_{ij}^{(nw)}(t)$, where $C_{ij}^{(nw)}(t)$ represents the cost of an optimal path arriving at exactly time t into node j via arc (i, j). For all nodes we will then have

$$C_j^{(nw)}(t) = \min_{i \in B_j} \{C_{ij}^{(nw)}(t)\} \qquad \forall j \in N. \qquad (6.37)$$

For any node $j \in N$ and time t, the value $PN_j(t)$, the preceding node along an optimal path arriving at time t, is therefore given as

$$PN_j(t) = \arg\min_{i \in B_j} \{C_{ij}^{(nw)}(t)\} \qquad \forall j \in N. \qquad (6.38)$$

Furthermore, we will compute for each arc the solution function $PT_{ij}(t)$, where $PT_{ij}(t)$ gives the departure time at node i of the best path from the source which arrives at node j exactly at time t, traveling via arc (i, j). We will then have:

$$PT_j(t) = PT_{ij}(t), where\ i = PN_j(t) \qquad \forall j \in N. \qquad (6.39)$$

At a particular given time t, we will say that an arc $(i, j) \in A$ is *favorable* if the cost of an optimal path from the source node to node i and time t, extended by the cost $c_{ij}(t)$ of traveling along arc (i, j) at time t, provides the least cost of any path we know of so far from the source to node j, arriving at time $t + d_{ij}(t)$. If two or more arcs directed into node j satisfy this condition, we may arbitrarily designate only one of them as favorable; if arc (i, j) is designated as favorable, then node i will be the "back-pointer" of node j for an arrival at time $t + d_{ij}(t)$; that is, we will have $PN_j(t + d_{ij}(t)) = i$ and $PT_{ij}(t + d_{ij}(t)) = t$. The following is a partial restatement of the one-to-all optimality conditions, and must be satisfied for all arcs $(i, j) \in A$. For any favorable arc, the condition is satisfied as an equality:

$$C_{ij}^{(nw)}(t + d_{ij}(t)) \leq C_i^{(w)}(t) + c_{ij}(t) \qquad \forall (i, j) \in A \qquad (6.40)$$

The primary difference between the one-to-all algorithm and the all-to-one algorithm is that we will be adjusting the solution functions $C_{ij}^{(nw)}(t)$, $PN_i(t)$, and $PT_{ij}(t)$ not at time $t_{current}$, but at future times which correspond to the arrival times one attains when departing along favorable arcs at time $t_{current}$. Since this operation will be performed continually for all arcs during the forward scan through time, and since we will inductively assume that the optimal values of all solution functions will have been determined for all times $t \leq t_{current}$, we will maintain the invariant that the solution functions $C_{ij}^{(nw)}(t)$, $PN_i(t)$, and $PT_{ij}(t)$ for all values of time t later than $t_{current}$ represent the best attainable solution obtained by extending an optimal path arriving prior to or at time $t_{current}$ by a single arc. Therefore, we can ensure that these future solution function values will be optimal when they are finally reached by the chronological scan.

In a similar fashion as in the all-to-one algorithm, the one-to-all algorithm will maintain during its execution a set of state variables, most of them linear piece index variables, described as follows. In general, for any time $t_{current}$, we will keep track of these state variables for a time just beyond time $t_{current}$.

Node Variables:

$CP_i^{(w)}$	:	Index of the current linear piece of $C_i^{(w)}(t)$, for t just past $t_{current}$
$CP_i^{(nw)}$	:	Index of the current linear piece of $C_i^{(nw)}(t)$, for t just past $t_{current}$
$CP_i^{(EWAT)}$	:	Index of the current linear piece of $EWAT_i(t)$ (described in Section 6.3.1), for t just past $t_{current}$
$CP_i^{(cwc)}$	:	Index of the current linear piece of $cwc_i(t)$, for t just past $t_{current}$
$CWP_i^{(nw)}$	:	Index of the current "minimal waiting" piece of $C_i^{(nw)}(t)$, at $t = EWAT_i(t_{current} + \varepsilon)$, where ε is very small.
$CWP_i^{(cwc)}$	:	Index of the current "minimal waiting" piece of $cwc_i(t)$, at $t = EWAT_i(t_{current} + \varepsilon)$, where ε is very small.
$FWAT_i$	:	Set of all feasible waiting arrival times (as described in Section 6.3.1) for node i and a time t just past $t_{current}$, maintained as a min-queue (see Appendix A).
OPN_i	:	Optimal preceding node of node i along an optimal path from the source which arrives just past time $t_{current}$ at node i.
$BDRY_i$	:	Boolean variable which is true if time $t_{current}$ marks the boundary between LNIs for node i.

Arc Variables:

$CP_{ij}^{(d)}$	:	Index of the current linear piece of $d_{ij}(t)$, for t just past $t_{current}$.
$CP_{ij}^{(c)}$	:	Index of the current linear piece of $c_{ij}(t)$, for t just past $t_{current}$.
CDP_{ij}	:	Index of the current "downstream" linear piece of $C_j^{(w)}(t)$ at $t = a_{ij}(t_{current} + \varepsilon)$, where ε is very small.
$CP_{ij}^{(nw)}$	:	Index of the current linear piece of $C_{ij}^{(nw)}(t)$, for t just past $t_{current}$.
$CP_{ij}^{(PT)}$	:	Index of the current linear piece of $PT_{ij}(t)$, for t just past $t_{current}$.
F_{ij}	:	Boolean variable which is true if (i, j) is a favorable arc at a departure time t just past $t_{current}$.
$INIT_{ij}$	:	If F_{ij} is true, then this variable holds the value of $a_{ij}(t_0)$ where t_0 is earliest point in the current interval of time for which arc (i, j) is favorable, as explained below.

For the purposes of the one-to-all algorithm, we will say that an arc (i, j) is favorable over an interval of departure times if the arc travel time and cost functions $d_{ij}(t)$ and $c_{ij}(t)$ are simple linear functions over this interval. Therefore, we can say that any arc which is favorable over an interval of time between two times t_0 to t_1 will cause an adjustment which improves the solution functions $C_{ij}^{(nw)}(t)$, $PT_{ij}(t)$, and $PN_j(t)$ over the contiguous interval of arrival times between $a_{ij}(t_0)$ and $a_{ij}(t_1)$. If, while advancing through time, we cross into a new linear piece of either $d_{ij}(t)$ or $c_{ij}(t)$ at some time t_0 but the arc (i, j) remains favorable, we say that in this case the initial point at which (i, j) became favorable, $INIT_{ij}$, is reset to t_0. Under this convention, $INIT_{ij}$ always gives the starting endpoint of a contiguous region of arrival times during which solution functions are being modified by a favorable arc (i, j).

Assume now that for some time $t_{current}$, we know the correct optimal values of all solution functions for all times $t \leq t_{current}$. Furthermore, suppose all state variables are determined so as to be valid for a time which is infinitesimally greater than $t_{current}$. We try to determine a time $t_{new} > t_{current}$ to which one can advance such that all solution functions maintain linear departure behavior and such that the set of favorable arcs remains unchanged during the interval of time $(t_{current}, t_{new})$.

By examining equations *(6.37)*, *(6.38)*, and *(6.39)*, we see that the solution functions $C_i^{(nw)}(t)$ and $PT_i(t)$ will maintain simple linear behavior and the function $PN_i(t)$ will maintain simple behavior while increasing the value of t over the region of time $(t_{current}, t_{new})$ as long as the following conditions are satisfied:

- *(i)* the value of t remains within the same linear piece of $C_{ij}^{(nw)}(t)$ for each arc $(i, j) \in A$.
- *(ii)* the same optimal incoming arc is selected by the minimization expression in *(6.38)*.
- *(iii)* the value of t remains within the same linear piece of $PT_{ij}(t)$ for each arc $(i, j) \in A$.

Further explanation is necessary to describe some subtle details related to conditions *(i)*, *(ii)*, and *(iii)*. If an arc (i, j) is not a favorable arc (that is, if F_{ij} is false), then these conditions pose no difficulty. However, if the arc is favorable, then as one advances the value of a time t during the interval $(t_{current}, t_{new})$, we know that the solution functions $C_{ij}^{(nw)}(t)$, $PT_{ij}(t)$, and $PN_j(t)$ are being simultaneously adjusted at future points in time given by $a_{ij}(t)$, in order to reflect the presence of a better path to node j via a departure along (i, j) at time t. However, we will actually perform this adjustment of these functions *after* the value of t_{new} is discovered. Hence, for the purposes of evaluating conditions *(i)...(iii)*, we must take into account the fact that these functions are supposedly being simultaneously modified during the scan through time. In the event that (i, j) is favorable over an interval of departure times between two times t_0 and t_1, where $t_0 \leq t_1$, we will be able to lower the solution function $C_{ij}^{(nw)}(t)$ over the interval of arrival times $a_{ij}(t_0)$ to $a_{ij}(t_1)$. Using the state variable $INIT_{ij}$, we keep track of the endpoint $a_{ij}(t_0)$. In a non-FIFO network, this may be either the upper or lower endpoint of the window of time over which $C_{ij}^{(nw)}(t)$ is modified. When evaluating the conditions above, we must therefore ensure that for times t during this window, we use the anticipated modified values of the functions $C_{ij}^{(nw)}(t)$ and $PT_{ij}(t)$. Fortunately, due to our assumption of a positive lower bound on all arc travel time functions, the only scenario for which we will need to use anticipated values of $C_{ij}^{(nw)}(t)$ and $PT_{ij}(t)$ is the case where $INIT_{ij}$ is less than or equal to $t_{current}$. In this situation, the value of t will be contained within the window of modification of these solution functions. Otherwise, if $INIT_{ij}$ is greater than $t_{current}$, then this will not be the case, and the remaining conditions which we impose on the computation of t_{new} will prevent the value of t from advancing into a window of modification of the solution functions. In short, if for any favorable arc (i, j) we have $INIT_{ij} < t_{current}$, then we must use the "anticipated" values of $C_{ij}^{(nw)}(t)$ and $PT_{ij}(t)$ rather than the actual values of these functions when evaluating conditions *(i)*, *(ii)*, and *(iii)*.

Next, for the solution functions which involve waiting, we will have over the interval of time $(t_{current}, t_{new})$:

$$C_i^{(w)}(t) = \min_{\tau \in FWAT_i(t)} \{C_i^{(nw)}(\tau) - cwc_i(\tau)\} + cwc_i(t) \qquad \forall i \in N, \qquad (6.41)$$

$$W_i(t) = \arg\min_{\tau \in FWAT_i(t)} \{C_i^{(nw)}(\tau) - cwc_i(\tau)\} + cwc_i(t) \qquad \forall j \in N. \qquad (6.42)$$

Equations *(6.41)* and *(6.42)* indicate that the solution functions $C_i^{(w)}(t)$ and $W_i(t)$ will exhibit simple linear behavior while increasing t within the interval $(t_{current}, t_{new})$ as long as $C_i^{(nw)}(t)$ remains a simple linear function, which is covered by the above conditions, and as long as:

(iv) the value of t remains within the same linear piece of $cwc_i(t)$ for waiting node $i \in N_w$.

(v) no new entries are added to the feasible set of waiting arrival times $FWAT_i$ for all $i \in N_w$. (Removals from this set are covered by the preceding cases). No new entries are added to this set as long as t remains within the same linear piece of $EWAT_i(t)$, and as long as $EWAT_i(t)$ remains within the same linear pieces of $cwc_i(t)$ and $C_i^{(nw)}(t)$.

Finally, Equation *(6.40)* must be satisfied by all arcs, and satisfied as an equality for all favorable arcs. The same set of arcs will therefore remain favorable while increasing t within the interval $(t_{current}, t_{new})$ as long as:

(vi) the value of t remains within the same linear piece of $d_{ij}(t)$ for arc $(i, j) \in A$.
(vii) the value of t remains within the same linear piece of $c_{ij}(t)$ for arc $(i, j) \in A$.
(viii) the value of $t + d_{ij}(t)$ remains within the same linear piece of $C_{ij}^{(nw)}(t)$ for arc $(i, j) \in A$.
(ix) Equation *(6.40)* is satisfied by all arcs $(i, j) \in A$, and satisfied as equality for all favorable arcs.

We proceed to write the above conditions mathematically.

$$(i)\ t_{new} \le B(C_{ij}^{(nw)}, CP_{ij}^{(nw)}) \qquad \begin{array}{l} \forall (i,j) \in A, j \neq s \\ \text{and } F_{ij} \text{ false, or} \\ INIT_{ij} > t_{current} \end{array} \qquad (6.43)$$

$$(ii)\ t_{new} \le (1/\rho_{ij})[b - \beta(C_{ij}^{(nw)}, CP_{ij}^{(nw)})] \qquad \begin{array}{l} \forall (i,j) \in A, j \neq s \\ \text{where } \rho_{ij} > 0 \end{array} \qquad (6.44)$$

$$\text{with } \rho_{ij} = \alpha(C_{ij}^{(nw)}, CP_{ij}^{(nw)}) - a,\ \tilde{i} = OPN_j,$$

$$a = \begin{cases} [\alpha(C_i^{(w)}, CP_i^{(w)}) + \alpha(c_{ij}, CP_{ij}^{(c)})]/[1 + \alpha(d_{ij}, CP_{ij}^{(d)})] & \text{if } F_{ij} \text{ and } INIT_{ij} \le t_{current} \\ \alpha(C_{\tilde{i}j}^{(nw)}, CP_{\tilde{i}j}^{(nw)}) & \text{otherwise} \end{cases},$$

$$b = \begin{cases} [\beta(C_i^{(w)}, CP_i^{(w)}) + \beta(c_{ij}, CP_{ij}^{(c)}) - a\beta(d_{ij}, CP_{ij}^{(d)}) & \text{if } F_{ij} \text{ and } INIT_{ij} \le t_{current} \\ \beta(C_{\tilde{i}j}^{(nw)}, CP_{\tilde{i}j}^{(nw)}) & \text{otherwise} \end{cases}$$

$$(iii)\ t_{new} \le B(PT_{ij}, CP_{ij}^{(PT)}) \qquad \begin{array}{l} \forall (i,j) \in A, j \neq s \\ \text{and } F_{ij} \text{ false, or} \\ INIT_{ij} > t_{current} \end{array} \qquad (6.45)$$

$$(iv)\ t_{new} \le B(cwc_i, CP_i^{(cwc)}) \qquad \forall i \in N_w \qquad (6.46)$$

$$(v.a)\ t_{new} \le B(EWAT_i, CP_i^{(EWAT)}) \qquad \forall i \in N_w \qquad (6.47)$$

$$(v.b)\ t_{new} \le \frac{B(C_i^{(nw)}, CWP_i^{(nw)}) - \beta(EWAT_i, CP_i^{(EWAT)})}{\alpha(EWAT_i, CP_i^{(EWAT)})} \qquad \begin{array}{l} \forall i \in N_w, \\ \alpha(EWAT_i, CP_i^{(EWAT)}) > 0 \end{array} \qquad (6.48)$$

$$(v.c)\ t_{new} \le \frac{B(C_i^{(cwc)}, CWP_i^{(cwc)}) - \beta(EWAT_i, CP_i^{(EWAT)})}{\alpha(EWAT_i, CP_i^{(EWAT)})} \qquad \begin{array}{l} \forall i \in N_w, \\ \alpha(EWAT_i, CP_i^{(EWAT)}) > 0 \end{array} \qquad (6.49)$$

$$(vi)\ t_{new} \le B(d_{ij}, CP_{ij}^{(d)}) \qquad \forall (i,j) \in A, j \neq s \qquad (6.50)$$

$$(vii)\ t_{new} \le B(c_{ij}, CP_{ij}^{(c)}) \qquad \forall (i,j) \in A, j \neq s \qquad (6.51)$$

(viii)

$$t_{new} \le \begin{cases} (1/\sigma_{ij})[B(C_j^{(w)}, CDP_{ij}) - \beta(d_{ij}, CP_{ij}^{(d)})] & \text{if } \sigma_{ij} > 0 \\ (1/\sigma_{ij})[B(C_j^{(w)}, CDP_{ij} - 1) - \beta(d_{ij}, CP_{ij}^{(d)})] & \text{if } \sigma_{ij} < 0 \end{cases} \quad \forall (i,j) \in A,\ j \ne s, \text{ where } \rho_{ij} \ne 0 \tag{6.52}$$

with $\sigma_{ij} = 1 + \alpha(d_{ij}, CP_{ij}^{(d)})$

$$(ix.a)\ t_{new} \le \frac{1}{\mu_{ij}} \begin{pmatrix} \beta(C_{ij}^{(nw)}, CDP_{ij}) - \beta(C_i^{(w)}, CP_i^{(w)}) - \beta(c_{ij}, CP_{ij}^{(c)}) + \\ \alpha(C_{ij}^{(nw)}, CDP_{ij})\beta(d_{ij}, CP_{ij}^{(d)}) \end{pmatrix} \quad \forall (i,j)\in A,\ j \ne s,\ F_{ij} \text{ true},\ \mu_{ij} > 0 \tag{6.53}$$

$$(ix.b)\ t_{new} \le \frac{-1}{\mu_{ij}} \begin{pmatrix} \beta(C_i^{(w)}, CP_i^{(w)}) + \beta(c_{ij}, CP_{ij}^{(c)}) - \beta(C_{ij}^{(nw)}, CDP_{ij}) - \\ \alpha(C_{ij}^{(nw)}, CDP_{ij})\beta(d_{ij}, CP_{ij}^{(d)}) \end{pmatrix} \quad \forall (i,j)\in A,\ j \ne s,\ F_{ij} \text{ false},\ \mu_{ij} < 0 \tag{6.54}$$

with $\mu_{ij} = \alpha(C_i^{(w)}, CP_i^{(w)}) + \alpha(c_{ij}, CP_{ij}^{(c)}) - \alpha(C_{ij}^{(nw)}, CDP_{ij})(1 + \alpha(d_{ij}, CP_{ij}^{(d)}))$

Each of the above conditions is a linear inequality relating the value of t_{new} to known quantities. Let $\Gamma_{ij}^{(i)}$, $\Gamma_{ij}^{(ii)}$, $\Gamma_{ij}^{(iii)}$, $\Gamma_{ij}^{(vi)}$, $\Gamma_{ij}^{(vii)}$, $\Gamma_{ij}^{(viii)}$, $\Gamma_{ij}^{(ix.a)}$, and $\Gamma_{ij}^{(ix.b)}$ be the smallest upper bounds respectively given by conditions *(i)*, *(ii)*, *(iii)*, *(vi)*, *(vii)*, *(viii)*, *(ix.a)*, and *(ix.b)* above for a particular arc $(i, j) \in A$, and let Γ_{ij} denote the minimum of all of these bounds for a particular arc. Similarly, let $\Gamma_i^{(iv)}$, $\Gamma_i^{(v.a)}$, $\Gamma_i^{(v.b)}$, and $\Gamma_i^{(v.c)}$ be the smallest upper bounds given for each waiting node $i \in N_w$ for the respective conditions *(iv)*, *(v.a)*, *(v.b)*, and *(v.c)*, and let Γ_i denote the minimum of all of these bounds. We then have the following means of computing t_{new}:

$$t_{new} = \min\{\min_{(i,j)\in A}\{\Gamma_{ij}\}, \min_{i\in N_w}\{\Gamma_i\}\}. \tag{6.55}$$

As in the all-to-one algorithm, we store the Γ_{ij} and Γ_i values in a heap, so this computation can be performed in *O(1)* time.

Recall that the invariant we maintain is that all solution functions are optimally determined for $t \le t_{current}$ and that the current or "anticipated" functions $C_{ij}^{(nw)}(t)$, $PN_i(t)$, and $PT_{ij}(t)$ for all values of time t later than $t_{current}$ represent the best attainable solution obtained by extending an optimal path arriving prior to or at time $t_{current}$ by a single arc. Since all arc travel times are by assumption greater than some positive constant ε, it follows from the invariant that we will know the optimal values of $C_{ij}^{(nw)}(t)$, $PN_i(t)$, and

$PT_{ij}(t)$ for all values of time $t \in (t_{current}, t_{current} + \varepsilon)$. However, since all solution functions are guaranteed to exhibit linear departure behavior over the interval $(t_{current}, t_{new})$, we will therefore know the optimal values of $C_{ij}^{(nw)}(t)$, $PN_i(t)$, and $PT_{ij}(t)$ for times in the entire interval $t \in (t_{current}, t_{new})$, and therefore for all times $t < t_{new}$. In order to advance the value of $t_{current}$ while maintaining the invariant above and while maintaining the necessary set of node and arc state variables, we perform the following 3 steps:

1. Compute the values of all solution functions $C_i^{(nw)}(t)$, $C_i^{(w)}(t)$, $PN_i(t)$, $PT_i(t)$, and $W_i(t)$ for the interval of time $t \in (t_{current}, t_{new})$. The time $t_{current}$ may mark the boundary between LNIs at one or more nodes; however, only nodes $i \in N$ for which $BDRY_i$ is true will transition to a new LNI at time $t_{current}$. All other nodes will carry over their previous simple linear behavior from $t = t_{current}$ into the region of time $(t_{current}, t_{new})$, so we therefore only need to re-compute solution functions for nodes for which $BDRY_i$ is true. We denote this set of nodes by N_B. For any arc (i, j), where $j \in N_B$, which is favorable over the interval $(t_{current}, t_{new})$ we must first update the solution functions $C_{ij}^{(nw)}(t)$ and $PT_{ij}(t)$ so as to satisfy Equation *(6.40)* over this interval. Hence, these functions reflect the presence of the better path to node j via arc (i, j). The following set of equations is then used to compute all solution functions over the interval. Each of these equations takes only $O(1)$ amortized time to apply, due to the amortized performance of the min-queue data structure representing the set $FWAT_i$ for each node:

$$C_j^{(nw)}(t) = C_{ij}^{(nw)}(t), where\ i = OPN_j \qquad \forall j \in N_B \qquad (6.56)$$

$$PT_j(t) = PT_{ij}(t), where\ i = OPN_j \qquad \forall j \in N_B \qquad (6.57)$$

$$PN_j(t) = OPN_j \qquad \forall j \in N_B \qquad (6.58)$$

$$C_j^{(nw)}(t) = \min_{\tau \in FWAT_i} \{C_j^{(nw)}(\tau) - cwc_j(\tau)\} + cwc_j(t) \qquad \forall j \in N_B \qquad (6.59)$$

$$W_j(t) = \arg\min_{\tau \in FWAT_i} \{C_j^{(nw)}(\tau) - cwc_j(\tau)\} \qquad \forall j \in N_B \qquad (6.60)$$

Before Equations *(6.59)* and *(6.60)* are evaluated, the time $t_{current}$ and the corresponding value of the waiting cost functions $C_j^{(nw)}(t_{current}) - cwc_j(t_{current})$ should for all $j \in N_B$ be added to the set of feasible waiting arrival times, since time $t_{current}$ will be a piece-wise boundary point of the waiting cost function.

2. Compute the values of all solution functions $C_i^{(nw)}(t)$, $C_i^{(w)}(t)$, $PN_i(t)$, $PT_i(t)$, and $W_i(t)$ at exactly time t_{new}, and adjust the values of all future solution functions $C_{ij}^{(nw)}(t)$, $PN_i(t)$, and $PT_{ij}(t)$ so that they represent the best attainable solution obtained by extending an optimal path arriving exactly at time t_{new} by a single arc. This computation need only be performed for nodes within the set $S = \{i \in N \mid \Gamma_i = t_{new}$, or $\exists j \in A_i$ *such that* $\Gamma_{ij} = t_{new}$ *or* $\exists i' \in B_i$ *such that* $\Gamma_{i'i} = t_{new}\}$, since only nodes within this set were responsible for the current value of t_{new}. After this computation, the state variables $BDRY_i$ for all nodes will be set to true only for nodes within the set S, since these nodes will have potentially undergone a transition to a new LNI at time t_{new}. The same set of equations given in *(6.56)...(6.60)* are used to perform the solution function update at exactly time t_{new}. It may be necessary, however, to adjust some of the linear piece index variables prior to this computation.

3. Update all state variables so that they reflect a time just after t_{new}. This step is straightforward, and performed in the same manner as the corresponding update step in the all-to-one algorithm. Linear piece index variables are successively incremented or decremented until they are found to contain a desired point in time.

After the completion of step 3, we may set the value of $t_{current}$ to t_{new} and repeat the entire sequence of steps again – the invariant described above will be maintained for the new value of $t_{current}$. In the event that t_{new} reaches $+\infty$, we may terminate the algorithm, having computed a complete optimal solution.

Proposition 6.4: *The one-to-all chronological scan algorithm correctly solves the minimum-cost one-to-all dynamic shortest path problem.*

Proof: We assert that the algorithm maintains the invariant that for any time $t_{current}$, all node and arc state variables are correctly maintained, all solution functions are optimally determined for $t \leq t_{current}$, and that the functions $C_{ij}^{(nw)}(t)$, $PN_i(t)$, and $PT_{ij}(t)$ for all values of time t later than $t_{current}$ represent the best attainable solution obtained by extending an optimal path arriving prior to or at time $t_{current}$ by a single arc. Each iteration of the algorithm increases $t_{current}$ by some positive amount, and since by Proposition 4.2 the solution to the one-to-all problem always consists of a finite number of LNIs, the algorithm will terminate, and it will terminate with an optimal solution. □

6.4 Running Time Analysis

In this section we analyze the worst-case and average-case running time of the previous chronological scan algorithms. For the case of computing minimum-time all-to-one paths through a FIFO network, we will prove a strongly polynomial bound of $O(mP^*\log n)$ on worst-case running time. As with many other network optimization algorithms, this polynomial bound is expected to be very loose – we will argue that the average-case running time for sparse planar networks, such as transportation networks, should be $O(n^{1.5}\log n)$ under certain reasonable assumptions. By a *planar* network, in this thesis we mean a network model which represents a problem that has a two-dimensional character with respect to the spatial orientation of nodes. Arcs may conceivably cross each-other, but it is assumed that arc travel times are generally highly-correlated with Euclidean distance.

The general all-to-one and one-to-all problems are shown to be NP-Hard, although we will argue that for most practical cases running time should be quite reasonable. This theoretical analysis may be compared with the results of computation testing, presented in Chapter 8.

It is often reasonable to make the assumption that each arc travel time function has at most a constant number of pieces, since for most applications the complexity of these functions should not necessarily scale with the size of the network. Under this assumption, we will be able to show strong polynomial bounds on running time for the minimum-time all-to-one algorithm in a FIFO network.

Proposition 6.5: *The worst-case asymptotic running time of the all-to-one chronological scan algorithm, when used to compute minimum-time paths in a FIFO network, is* $O(mP^*\log n)$.

Corollary: *We have the following:*

- *If $P(d_{ij})$ is $O(1)$ for all $(i, j) \in A$, then the worst-case running time is $O(m^2 \log n)$.*
- *If G is a sparse, bounded-degree network and if $P(d_{ij}) = O(1)$ for all $(i, j) \in A$, then the worst-case running time is $O(n^2 \log n)$.*
- *Since the all-to-one label-correcting algorithm can be applied to solve the minimum-time one-to-all problem for all departure times by Proposition 4.3, the above results hold for this problem as well.*

Proof: Initialization, including the computation of T_∞, requires the same amount of running time as a static shortest path algorithm, which is absorbed by the total $O(mP^* \log n)$ running time bound as long as a suitable polynomial modified static shortest path algorithm, such as Dijkstra's algorithm based on a heap, is used. During each subsequent iteration of the main loop, $O(1)$ time is required to determine a new value of t_{new}, and then $O(\log n)$ time is required to update in the heap the Γ_{ij} value for any arc (i, j) directed into or out of a node for which an LNI boundary or arc travel time boundary occurs at time t_{new}. Since, by Lemma 4.10, there is a bound of $O(P^*)$ on the number of LNIs at each node, there will be a bound of $O(P^*)$ on the number of heap updates required for the values of Γ_{ij} corresponding to each arc $(i, j) \in A$. The total number of arc updates is therefore $O(mP^*)$, each requiring $O(\log n)$ time, for a total running time of $O(mP^* \log n)$ in the worst case. If the transformation techniques given by Proposition 4.3 are used to solve a one-to-all problem for all departure times as an all-to-one problem, then the cost of transformation will always be dominated by the running time of the all-to-one solution algorithm. □

Proposition 6.6: *For sparse, bounded-degree FIFO networks, the total running time of the all-to-one and one-to-all chronological scan algorithms is $O((I + S)\log n)$, where I gives the number of linear pieces present among all network data functions provided as input, and where S gives the total number of LNIs among all solution functions produced as output.*

Corollary: *For sparse, bounded-degree FIFO networks, the running time of the all-to-one and one-to-all chronological scan algorithms is no more than a factor of log n larger than the best possible achievable running time for any solution algorithm.*

Proof: For a FIFO network, LNIs for each node will be chronologically generated, and linear pieces of each network data function will be examined chronologically. Upon the generation of each LNI or the switch to a new linear piece of any network data function, at most $O(1)$ of the Γ_i and Γ_{ij} values will be updated since the degree of the network is bounded. Since these values are stored in a heap, there is a cost of $O(\log n)$ in running time for each update, for a total running time of $O((I + S)\log n)$. The presence of waiting constraints and costs will not influence this asymptotic running time, due to the amortized performance of the min-queue data structure.

Since any solution algorithm must examine all I linear pieces of the network data functions provided as input, and compute a total of S LNIs as output, we have a lower bound of $\Omega(I + S)$ on running time, and we therefore conclude that if there were a better algorithm for sparse bounded-degree FIFO networks, its asymptotic running time could be no more than a factor of *log n* faster than that of the chronological scan algorithms. □

Proposition 6.7: *The minimum-cost all-to-one and one-to-all problems, and the minimum-time all-to-one and one-to-all problems in a non-FIFO network, are all NP-Hard.*

Proof: The reader is referred to a well-written proof by Sherali, Ozbay, and Subramanian in [18] which a reduction is given from the subset-sum problem, known to be NP-Hard. The subset-sum problem is stated thus: given a set of N positive integers $A_1 \ldots A_N$, partition these integers (if possible) into two subsets such that the sums of the elements in both subsets are equal. Sherali, Ozbay, and Subramanian construct a "ladder network" consisting of a series of $N+1$ primary nodes and a destination node, such that two possible feasible routes connect primary node i to primary node $i + 1$: one route with a static travel time of one, the other with a static travel time of $1 + A_i$. Waiting in this network is forbidden. The final primary node is connected to the destination node with a time-dependent arc, the travel time of which is given by:

$$d(t) = \begin{cases} S+2 & \text{if } t \leq N+S-\frac{1}{10}, \text{or } t > N+S+\frac{1}{10}, \\ 1 & \text{otherwise} \end{cases} \qquad (6.61)$$
$$\text{where } S = \left(\frac{1}{2}\right)\sum_{i=1}^{N} A_i.$$

If a subset exists with sum equal to exactly *S*, then a path through the ladder network corresponding the this subset (i.e. following the routes corresponding to the elements in the subset), departing at time $t = 0$,will arrive at exactly the right moment at the last node so that a travel time of $d(t) = 1$ is available to continue to the destination, for an arrival time of $t = N + S + 1$. Any other subset sum will result in an arrival to the last node at a time such that the remaining travel time to the destination is $S + 2$, yielding an arrival time later than $N + S + 1$. Therefore, a solution to the dynamic shortest path problem through this ladder network is equivalent to a solution to the subset-sum problem. □

We now argue that average-case running time for a broad class of problems, including many transportation problem, should be quite efficient. If *G* is a sparse, bounded-degree planar network, if $P(d_{ij}) = O(1)$ for all $(i, j) \in A$, and if arc travel times are highly-correlated with Euclidean distance, then the expected running time of the one-to-all and all-to-one chronological scan algorithms is $O(n^{1.5} \log n)$. This result is expected to hold even for most minimum-cost problems and for problems in non-FIFO networks.

The number of LNI boundaries at a particular node $i \in N$ is equal to the number of times the optimal path from *i* to the destination *d* either transitions to a new path or undergoes a piece-wise change in path travel time. This number is expected to be equal approximately to the product of the average number of arcs in any such optimal path and the number of linear pieces in the travel time functions of each of these arcs. For a planar network in which arc travel times are highly correlated with Euclidean distance, the average number of arcs in an optimal path is expected to be on the order of $n^{0.5}$, giving $O(n^{0.5})$ expected LNIs at each node in the network. Using the reasoning from the proof of Proposition 6.5, we therefore arrive at an expected running time of $O(n^{1.5} \log n)$.

Although in the non-FIFO case, there may be more overhead required to maintain the CDP_{ij} arc index variables (in the FIFO case, these index variables decrease monotonically over the course of the algorithm, but in the non-FIFO case these variables may be both incremented and decremented substantially more over the course of the algorithm), for most "reasonable" problems this extra overhead is expected to be negligible.

Since transportation networks, one of the main areas of application for dynamic network algorithms, almost always satisfy the conditions of the previous claim, the all-to-one chronological scan algorithm is expected to run in $O(n^{1.5} \log n)$ time for these networks. Since the best-known static shortest path algorithms applied to sparse, bounded degree networks run in $O(n \log n)$ time, we conclude that our chronological scan algorithms should for many problems be a factor of only $n^{0.5}$ slower asymptotically than a static shortest path algorithm. Although there is typically a larger hidden constant in the running time of the dynamic algorithm, these results are very encouraging for the prospect of scaling these algorithms to large networks. The parallel techniques given in Section 5.2 may be employed to further reduce computation time for minimum-time problems in FIFO networks.

6.5 Hybrid Continuous-Discrete Methods

The performance of chronological scan algorithms, and in fact of any continuous-time dynamic shortest path algorithm, is highly sensitive to the complexity of the dynamics represented by the time-dependent network data functions. For very large networks with time-dependent data functions containing many linear pieces, it is conceivable that the solution functions may consist of very many linear pieces, leading to slower computation times and greater memory demands. However, the factor one should blame for this decline in performance is not that the solution algorithms are slow, but rather the fact that these algorithms are required to compute the exact solution to a problem for which the solution is simply very complex. Since algorithmic performance is primarily determined by solution complexity, there are intrinsic bounds imposed on the performance of any continuous-time dynamic shortest path algorithm which computes an exact solution.

Aside from parallel computation, the only remedy for this difficulty is to devise algorithms which compute only approximate solutions. Discrete-time algorithms can be viewed as one such avenue for computing approximate solutions – these algorithms effectively ignore dynamic solution behavior which takes place at a finer time scale than the granularity of the global discretization applied to time. In this subsection, we consider a slight modification of chronological scan algorithms such that they assume the behavior of discrete-time algorithms in the event that solution dynamics reach such a fine time scale. We will call such approaches *hybrid continuous-discrete* algorithms.

The general idea behind these algorithms is to try to force all solution LNIs to be longer than certain minimum duration in time. This effect is not difficult to achieve – during the chronological scan through time during which $t_{current}$ either increases or decreases between positive and negative infinity, we will impose a minimum step-size Δ so that $t_{current}$ must either increase or decrease by at least Δ; that is, we require for each step through time that $|t_{new} - t_{current}| \geq \Delta$. This requirement is imposed by adjusting the value of t_{new} appropriately as it is computed. For example, consider the case of the all-to-one chronological scan algorithm. If at any iteration of the algorithm we compute a value of t_{new} where $t_{new} > t_{current} - \Delta$, then we set the value of t_{new} to $t_{current} - \Delta$ in order to enforce the minimum step-size requirement. The set of nodes to update after a *Δ-step* will be given by the set $S = \{i \in N \mid \Gamma_i \geq t_{new} \text{ or } \exists j \in A_i \text{ such that } \Gamma_{ij} \geq t_{new}\}$. In principle, no other changes should be necessary to the remainder of the algorithm. One must ensure that this is the case, however, since for any particular implementation there are certain assumptions which one may make in the exact case which are no longer valid in the hybrid case. For example, in the exact algorithm, the linear piece indices $CP_{ij}^{(d)}$, $CP_{ij}^{(c)}$, $CP_i^{(cwc)}$, and $CP_i^{(ubw)}$ are guaranteed to decrease by at most one during any particular iteration of the algorithm. In the hybrid case, it may be necessary to decrement these variables several times until they reach their desired values. Accordingly, several values may enter the queue of feasible waiting departure times $FWDT_i$ per iteration in the hybrid case, whereas at most one value may enter per iteration in the exact case.

Recall that for discrete-time problems, only integer-valued times within the finite window *{0, 1, 2, ..., T}* are considered during computation of a solution. If this is deemed to be an acceptable approximation, then the hybrid algorithm discussed above may be further modified so that each iteration must compute a discrete value of t_{new} which is in the window *{0, 1, 2, ..., T}*, where the algorithm terminates once t_{new} is determined to be less than *0*. In this case, a bucket approach can be used to determine the next value of t_{new} rather than a heap, so that the factor of $\log n$ will be effectively removed from the running time of the algorithm. The worst-case running time of the hybrid algorithm in a FIFO network should therefore improve to $O(mT + SSP)$, assuming that the number of linear pieces present in the network data functions is absorbed by this asymptotic quantity. However, this worst-case running time is identical to the worst-case and average-case running time of an optimal discrete-time algorithm for the all-to-one problem. Therefore, in a FIFO network, approximate chronological scan algorithms can be made to perform asymptotically equal to or better than optimal discrete-time solution algorithms.

It is possible to construct a continuous-discrete hybrid of the one-to-all chronological scan algorithm as well, using the same type of modifications as described above.

Chapter 7

Label-Correcting Algorithms

In this Chapter, we describe a second class of continuous-time dynamic shortest path algorithms, which we call label-correcting algorithms. These algorithms have the advantage of being simple to describe, due to their similarity to well-known existing static label-correcting algorithms. Although their worst-case theoretical running time is worse than that of chronological scan algorithms, these algorithms perform quite well in practice for almost all reasonable problem instances.

The label-correcting methods we present in this Chapter are all based on the underlying methodology of algorithms proposed by Halpern [12] and Orda and Rom [16] for respectively solving the minimum-time and minimum-cost one-to-all problem, in which waiting is allowed during certain fixed regions of time. The algorithm of Halpern solves only the minimum-time problem, so we focus below on the more general algorithm of Orda and Rom. Both papers consider general continuous-time arc travel time and travel cost functions, and propose algorithms which produce an optimal solution within a finite amount of time. However, as we shall see in this chapter, implementation of these algorithms can be a non-trivial task due to the fact that they are based on operations on general continuous-time functions. Under the piece-wise linearity assumptions of this thesis, we will be able both to implement these algorithms and to prove strong bounds on their worst-case and average-case running times, including polynomial worst-case running time bounds for minimum-time algorithms in FIFO networks.

The results of computational testing of label-correcting algorithms, and a comparison between their performance and that of chronological scan algorithms, appears in Chapter 8. The all-to-one and one-to-all label-correcting algorithms are described in Sections 7.1 and 7.2 respectively, and an analysis of their theoretical performance is contained in Section 7.3.

7.1 All-to-One Solution Algorithm

We proceed to develop a label-correcting solution algorithm for the minimum-cost all-to-one continuous-time dynamic shortest path problem. Although the performance of this algorithm is sensitive to whether or not the FIFO condition holds, the statement of the algorithm itself is not simplified if the FIFO condition holds.

Dynamic label-correcting algorithms are essentially identical in operation to static label correcting algorithms, only node labels are in this case given as functions of time rather than as simple scalars. Using the notation developed in this thesis, these node labels are the solution functions $C_i^{(w)}(t)$ and $C_i^{(nw)}(t)$ for each node $i \in N$. For simplicity of discussion, we first assume that waiting is forbidden, so that only the function $C_i^{(nw)}(t)$ is necessary to specify optimal path costs. The optimality conditions for the general minimum-cost all-to-one problem where waiting is forbidden are:

$$C_i^{(nw)}(t) \le c_{ij}(t) + C_j^{(nw)}(t + d_{ij}(t)) \qquad \forall (i, j) \in A,\ i \ne d, \qquad (7.1)$$

$$C_d^{(nw)}(t) = 0. \qquad (7.2)$$

Initially, we can set the node label functions to zero for the destination node, or infinity otherwise. If the node label function $C_i^{(nw)}(t)$ for some arc $(i, j) \in A$ violates one of the optimality conditions given in *(7.1)*, we can "correct" the label by assigning the label the function $min\{\ C_i^{(nw)}(t),\ c_{ij}(t) + C_j^{(nw)}(t + d_{ij}(t))\ \}$. This correction process replaces the parts of the solution function $C_i^{(nw)}(t)$ corresponding to values of time which violate optimality. Each iteration of the algorithm involves locating an arc $(i, j) \in A$ which violates *(7.1)* and performing a correction of the node cost label of node i. If at a certain point all optimality conditions are satisfied, then the algorithm terminates with a complete solution.

If waiting is allowed, the same principle applies, only the process of correcting labels is more complicated. In this case, we take the solution function $C_i^{(w)}(t)$ to be the label of each node $i \in N$. At optimality the following conditions must be satisfied for every arc $(i, j) \in A$, obtained from the all-to-one optimality conditions *(4.1)* and *(4.2)*:

$$C_i^{(w)}(t) \le \min_{\tau \in [t, t+ubw_i(t)]} \{cwc_i(\tau) + c_{ij}(\tau) + C_j^{(w)}(a_{ij}(\tau))\} - cwc_i(t) \qquad \forall (i, j) \in A,\ i \ne d, \qquad (7.3)$$

$$C_d^{(w)}(t) = 0. \qquad (7.4)$$

Again, we initialize the node label functions to zero for the destination node and infinity for all other nodes. If the optimality conditions above are satisfied for all arcs $(i, j) \in A$, then we have an optimal solution. Otherwise, we must identify an arc (i, j) for which *(7.3)* does not hold, and correct the label $C_i^{(w)}(t)$ by assigning to it the function $min\{ C_i^{(w)}(t), f_{ij}(t) - cwc_i(t) \}$, where $f_{ij}(t)$ represents the function given by the result of the minimization expression in *(7.3)*. Computing $f_{ij}(t)$ is not a trivial matter, but under our assumptions of piece-wise linearity and continuity of all network data functions, this computation is feasible. The minimum given by $f_{ij}(t)$ is in this case achieved at either a value of τ which is an endpoint of the window $[t, t + ubw_i(t)]$, or at a value of τ which is a boundary between two linear pieces of the piece-wise linear function $cwc_i(t) + c_{ij}(t) + C_j^{(w)}(a_{ij}(t))$ within this window.

The all-to-one label-correcting algorithm is easy to state, and straightforward to implement, provided that the following fundamental routines are available:

- Addition or subtraction of two functions,
- Computation of the minimum of two functions,
- Computation of the composition of two functions,
- Determination if a function is less than or equal to function at all points in time,
- Computation of a minimum of the form given in *(7.3)*, if waiting is allowed.

In the absence of piece-wise linearity assumptions, some of these operations, in particular the composition operation and the minimization operation, may be prohibitively complicated to implement. Moreover, even if implementation is possible, these operations may require excessive amounts of computation time to perform if the functions under consideration are not sufficiently simple.

The worst-case and average-case theoretical running time of the all-to-one label-correcting algorithm are discussed in Section 7.3. We now assert the correctness and termination of the algorithm.

Proposition 7.1: *The all-to-one label-correcting algorithm correctly solves the minimum-cost all-to-one dynamic shortest path problem and terminates within a finite amount of time.*

Proof: Proof is given for the case where the "scan-eligible" list of the label-correcting algorithm is stored as a FIFO queue; it should be possible to extend this proof to cover arbitrary variants for utilization of the scan-eligible list. Since optimality is required for termination, we must simply prove that the algorithm terminates in a finite amount of time. To do this, we employ the same reasoning as in a well-known proof of termination for static label-correcting algorithms. Suppose that the algorithm proceeds in stages, where in each stage a pass is made over all arcs $(i, j) \in A$, and any arc violating the optimality conditions is used to correct the label of the node at the upstream end of the arc. After the first stage, the algorithm will correctly identify all optimal paths which consist of at most a single arc. After two stages, the algorithm will discover all optimal paths consisting of two or fewer arcs, and in general, after N stages, the all optimal paths of N or fewer arcs will have been discovered. We then claim, using the same reasoning as the proof of Proposition 4.2, that all optimal paths consist of at most a finite number of arcs, so that the all-to-one algorithm should terminate within a finite number of stages. □

7.2 One-to-All Solution Algorithm

We attribute the one-to-all label-correcting algorithm to Orda and Rom [16]. This algorithm proceeds in much the same fashion as the all-to-one label-correcting algorithm discussed in the preceding subsection. The cost label of each node $i \in N$ is taken to be the function $C_i^{(w)}(t)$, and at optimality we must satisfy the following conditions for each arc $(i, j) \in A$, which are derived from the one-to-all optimality conditions *(4.5)*, *(4.6)*, and *(4.7)*:

$$C_j^{(nw)}(a_{ij}(t)) \leq \min_{\{\tau | a_{ij}(\tau)=t\}} \{C_i^{(w)}(\tau) - cwc_i(\tau)\} + cwc_i(t) + c_{ij}(t) \qquad \forall (i, j) \in A, j \neq s, \qquad (7.5)$$

$$C_s^{(w)}(t) = 0. \qquad (7.6)$$

In order to simplify *(7.5)*, we let $f_i(t)$ be the result of the minimization expression over τ. Given our assumptions of piece-wise linearity (and continuity, in the event that waiting is allowed), the computation of $f(t)$ is difficult, but feasible. One possible method of performing this computation involves building $f_i(t)$ using a forward scan through the linear pieces of $C_i^{(w)}(t) - cwc_i(t)$, in a similar manner as that of the one-to-all chronological scan algorithm.

To begin with, we will initialize the node label functions of all nodes to infinity, except for that of the source node, which we set to zero. If at some point in time these node label functions satisfy the optimality conditions *(7.5)* and *(7.6)*, we terminate the algorithm. Otherwise, we locate an arc $(i, j) \in A$ for which *(7.5)* is not satisfied, and we correct the node label function $C_j^{(w)}(t)$ by performing the assignment:

$$C_j^{(w)}(a_{ij}(t)) \leftarrow \min\{C_j^{(w)}(a_{ij}(t)), f_i(t) + cwc_i(t) + c_{ij}(t)\} \qquad (7.7)$$

If $a_{ij}(t)$ is strictly increasing (i.e. if the arc travel time $d_{ij}(t)$ satisfies the so-called *strict FIFO condition*), then we have the following closed-form expression for the assignment:

$$C_j^{(w)}(t) \leftarrow \min\{C_j^{(w)}(t), f_i(a_{ij}^{-1}(t)) + cwc_i(a_{ij}^{-1}(t)) + c_{ij}(a_{ij}^{-1}(t))\}. \qquad (7.8)$$

In this case $a_{ij}^{-1}(t)$ denotes the inverse of the function $a_{ij}(t)$, which will be equivalent to the FIFO inverse of $a_{ij}(t)$ under the strict FIFO condition. If the strict FIFO condition is not satisfied, however, then it is still possible to perform the assignment given in *(7.7)*, only in this case the function $C_j^{(w)}(t)$ must be constructed by scanning over the linear pieces of the piece-wise linear function $min\{C_j^{(w)}(a_{ij}(t)), f_i(t) + cwc_i(t)+c_{ij}(t)\}$, in much the same fashion as the one-to-all chronological scan algorithm. Under piece-wise linearity assumptions of network data, implementation of the one-to-all label-correcting algorithm is straightforward, provided that the following operations are available:

- Addition or subtraction of two piece-wise linear functions,
- Computation of the minimum of two piece-wise linear functions,
- Computation of the composition of two piece-wise linear functions,
- Determination if a piece-wise linear function is less than or equal to another piece-wise linear function,
- Solution of the minimization of the form given in *(7.5)*, if waiting is allowed,
- Determination of the result of the "composed assignment" operation given in *(7.7)*.

Proposition 7.2: *The one-to-all label-correcting algorithm correctly solves the minimum-cost one-to-all dynamic shortest path problem and terminates within a finite amount of time.*

Proof of the proposition follows the same line of reasoning as the proof of Proposition 7.1, and is omitted.

7.3 Running Time Analysis

We proceed to analyze the theoretical worst-case and average-case running time of the one-to-all and all-to-one label-correcting algorithms. This analysis will make extensive use of the running time analysis performed for chronological scan algorithms. Due to Proposition 6.5, the minimum-cost and non-FIFO minimum-time problems are NP-Hard. We will therefore focus only on the average-case running time for label-correcting algorithms applied to these problems. For minimum-time FIFO problems, however, we are able to establish strong polynomial bounds.

Proposition 7.3: *The worst-case asymptotic running time of the all-to-one label-correcting algorithm, when used to compute all-to-one minimum-time paths in a FIFO network, is $O(nmP^*)$.*

Corollary: *We have the following:*

- *If $P(d_{ij})$ is $O(1)$ for all $(i, j) \in A$, then the worst-case running time is $O(m^2n)$.*
- *If G is a sparse, bounded-degree network and if $P(d_{ij}) = O(1)$ for all $(i, j) \in A$, then the worst-case running time is $O(n^3)$.*
- *Since the all-to-one label-correcting algorithm can be applied to solve the minimum-time one-to-all problem for all departure times by Proposition 4.3, the above results hold for this problem as well.*

Proof: The strongest known polynomial asymptotic running time for a label-correcting algorithm is achieved when its list of "scan-eligible" nodes is maintained as a FIFO queue. In this case, the running time the algorithm will be equal to the number of arcs times the maximum number of possible corrections of each node label times the amount of time required to correct a single label. Since by Lemma 4.5 minimum-time paths in a FIFO network will contain at most $n - 1$ arcs, each label will be corrected at most $n - 1$ times, and since by Lemma 4.10 there are $O(P^*)$ LNIs generated for each node, the time required to correct each label will be $O(P^*)$, since any of the operations performed by the algorithm in a label correction may be performed in time proportional to the number of linear pieces in the node label functions. The total running time of the minimum-time all-to-one algorithm for computing optimal paths in a FIFO network is therefore bounded in the worst case by $O(nmP^*)$. As with most label-correcting algorithms, this bound is expected to be very loose. □

We proceed to argue the expected running time of label-correcting algorithms for simple problems. If G is a sparse, bounded-degree planar (defined as in Section 6.4) network, if $P(d_{ij}) = O(1)$ for all $(i, j) \in A$, and if arc travel times are highly-correlated with Euclidean distance, then the expected running time of the one-to-all and all-to-one label-correcting algorithms is $O(n^{0.5}LC(n, m))$, where $LC(n, m)$ denotes the expected running time of a static label-correcting algorithm applied to a network with n nodes and m arcs. This result is expected to hold even for most minimum-cost problems and for problems in non-FIFO networks. The reasoning behind this claim is similar to the reasoning behind the average-case running time analysis for the chronological scan algorithms performed in Section 6.4. Since we expect on average $O(n^{0.5})$ LNIs comprising the solution for a particular node, it will take $O(n^{0.5})$ time on average to correct each node label, and therefore we have a total expected running time of $O(n^{0.5}LC(n, m))$. This expected running time is very close to the expected running time of chronological scan algorithms, and indeed, the results of computational testing show that for most problem instances the two algorithms are very similar in running time.

In a FIFO network, parallel adaptations of the minimum-time all-to-one algorithm may be constructed using the techniques outlined in Section 5.2, and these adaptations may be used to efficiently solve either the minimum-time all-to-one problem or the minimum-time one-to-all for all departure times. Unfortunately, it is not possible to apply the same methods as in Section 6.5 to develop hybrid continuous-discrete approximation algorithms of label-correcting algorithms, so the development of label-correcting approximation algorithms is an area open to future research.

Chapter 8

Computational Results

This chapter contains the results of extensive computational testing of the continuous-time dynamic shortest path algorithms presented in this thesis. In order to fully assess the performance of these algorithms as a feasible solution technique, we compare their performance with that of the best known discrete-time dynamic shortest path algorithm. Appendix B contains all of the raw data collected from computational testing which is interpreted in this Chapter.

8.1 Network Generation

Since transportation networks constitute one of the primary areas of application for dynamic shortest path algorithms, we have selected most of the networks used for computational testing to be networks with a high degree of resemblance to transportation networks. Two real networks are considered: the Amsterdam A10 outer beltway network, a large ring-shaped network with 196 nodes and 310 arcs, and an urban network model of the city of Montreal, consisting of 6906 nodes and 17157 arcs. Additionally, random networks of two types have been generated. The first type of these random networks, which we refer to as *random planar networks*, were generated by placing random points in a two-dimensional plane, and by randomly adding arcs until a desired density is achieved. As explained in Section 6.4, a *planar* network in this thesis is allowed to have arcs which cross, as long as the network has a general two-dimensional character in which arcs primarily connect spatially close nodes. In keeping with this notion, the probability that an arc is added between two randomly-selected nodes in a random planar network was taken to be proportional to $1/\sqrt{d}$, where d is the distance between the two nodes, since arcs in planar networks typically connect pairs of nodes which are in close proximity. The second type of random network, which we refer to as a *completely random network*, contains arcs which connect between randomly chosen pairs of nodes, where the "length" of each arc is chosen randomly from a uniform distribution from 1.0 to 100.0. All random networks are built from an initial spanning tree of arcs in order to ensure strong connectivity.

Time-dependent arc travel times were generated for each network in several ways, in order to test the effects of varying types of dynamic situations. In each testing scenario, a certain designated percentage of the arcs are assigned time-varying characteristics, since many dynamic networks are expected to have a significant amount of arcs whose behavior does not change substantially over time. For testing, we consider the entire spectrum of networks from completely static networks up through networks in which 100% of the arcs exhibit dynamic characteristics. All time-varying arc travel time functions are generated such that they are dynamic for a two-hour window of time, and static for all times before this window. These functions are generated by one of two possible methods. The first method, which we say generates *triangular* arc travel time functions, attempts to simulate the tendency of many networks (in particular, transportation networks) to experience rising congestion which peaks at some sort of "rush-hour", and then recedes. To generate a triangular arc travel time function, a random hour-long interval is selected within the two-hour dynamic window of time. The arc travel time function is then assigned a triangle-shaped profile over this hour-long interval, whereby it rises linearly for 30 minutes, and then falls linearly for the remaining 30 minutes. We make certain that the linear slopes involved are shallow enough such that the FIFO condition is satisfied. Triangular arc travel time functions therefore always have exactly four linear pieces. The second method we use to generate arc travel time functions is to simply generate a fixed number of totally random linear pieces within the two-hour window of dynamic behavior. We say this methods generates *random* arc travel-time functions; these functions will always be continuous, and consist of a designated number of linear pieces. Both FIFO and non-FIFO sets of random arc travel time functions were generated and tested using this method.

8.2 Description of Implementations

We have constructed implementations of the all-to-one chronological scan and label-correcting algorithms, both written in C++, and run on a 333 megahertz Pentium-based system with 128Mb of memory. The performance of the one-to-all algorithms is expected to be nearly identical to that of the corresponding one-to-all algorithms. We consider only the computation of minimum-time paths, in either a FIFO or non-FIFO network. Solution algorithms which compute minimum-cost paths are expected to perform similarly. For simplicity, waiting at nodes was not considered for any testing scenario. The presence of waiting constraints is expected to scale the algorithmic running time by a small constant. All execution times (except for a few involving very large problems) were obtained by averaging the execution times of 10 different all-to-one computations, each with a new random destination. Correctness of algorithms was ensured by verifying small test networks by hand, and by cross-checking the output of the chronological scan and label-correcting algorithms for larger problems. For all problem instances checked, the output of the two methods was identical.

The scan-eligible list of nodes for the all-to-one label-correcting algorithm was implemented both as a FIFO queue and as a double-ended queue. Results are reported for both implementations.

Furthermore, the best-known all-to-one discrete-time dynamic shortest path algorithm for the case where waiting at nodes is not permitted, described in [5], was implemented in order to make a comparison between continuous-time and discrete-time methods. For the discrete-time algorithm, we consistently use a discretization of time into 1000 levels; for a two hour total period of analysis, this corresponds to a discretization of time into intervals of 7.2 seconds in duration. For the Montreal network, only 100 levels were used due to memory constraints – the running time for 1000 levels should be 10 times the running time measured with 100 levels, since the discrete-time solution algorithm has a running time which is linear in the number of time intervals present. Dynamic arc travel time data are always stored as piece-wise linear functions continuous-time functions. In order to apply discrete-time solution algorithms, these functions are sampled at 1000 evenly-spaced points in time. We sometimes report the running time of this discretization process separately from the running time of the discrete-time solution algorithm; in general, this discretization process often requires as much or more processing time compared to the discrete-time solution algorithm. Whether or not these two running times should be considered jointly depends on the underlying means of storing time-dependent network data – if network data is stored in the space-efficient form of piece-wise linear functions, then extra processing time will be required for sampling this data. Alternatively, network data can conceivably be stored in discrete sampled form, but this in turn can require a great deal of memory.

8.3 Assessment of Algorithmic Performance

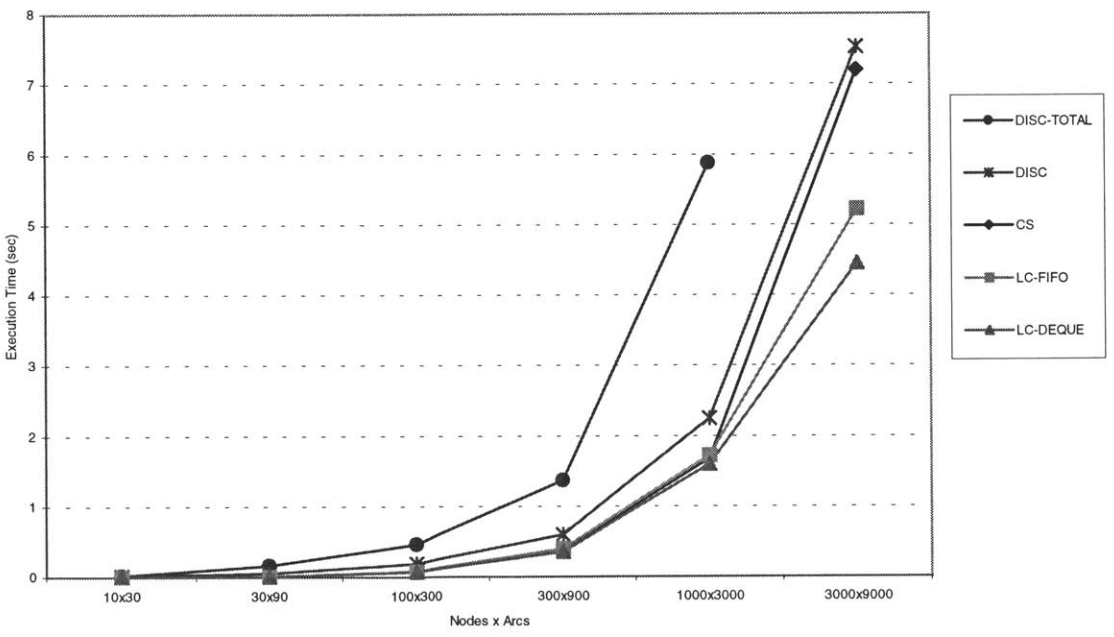
There are many parameters one may vary when experimentally evaluating the performance of dynamic shortest path algorithms. In the analysis that follows, we will illustrate the effects upon computation time and memory of varying each significant parameter associated with the problem in turn, while holding the remaining parameters constant. From these results, one may extrapolate with reasonable certainty the performance of the algorithms under consideration for many common problem instances.

Figure 1 displays the relative performance of each algorithm tested as a function of network size. The chronological scan (CS) algorithm, the label-correcting algorithm based on a FIFO queue and on a double-ended queue (LC-FIFO and LC-DEQUEUE respectively), and the best-known discrete-time all-to-one algorithm are compared. The DISC data series gives the running time of only the discrete-time algorithm, whereas the DISC-TOTAL series gives this running time plus the time required to sample the piece-wise linear travel time functions provided as input.

Each of the tests shown in Figure 1 was performed in a random planar network, where 25% of the arc travel time functions in the network varied with time. For these arcs, travel time functions consisting of 10 random linear pieces satisfying the FIFO property were generated. The number of nodes and arcs is scaled uniformly in the figure, so that average node degree remains constant, and only network size changes between sample executions.

For each of the network sizes tested, the continuous-time algorithms displayed similar performance, and outperformed the discrete-time algorithm by a significant factor in speed, especially when the time required to sample data is considered. Additionally, the average number of LNIs per node produced as output was observed to be no higher than 16, leading to more than an order of magnitude of savings in memory space compared with the vectors of 1000 distance labels produced for each node by the discrete-time algorithm. It is important to bear in mind, however, that these results for the continuous-time algorithms are very sensitive to the remaining input conditions, such as the number of linear pieces used as input, the percentage of the network which is dynamic, and on factors such as the FIFO property and the planar nature of the network. One must carefully consider all results presented in this section as a whole in order to be able to predict the general performance of the continuous-time algorithms.

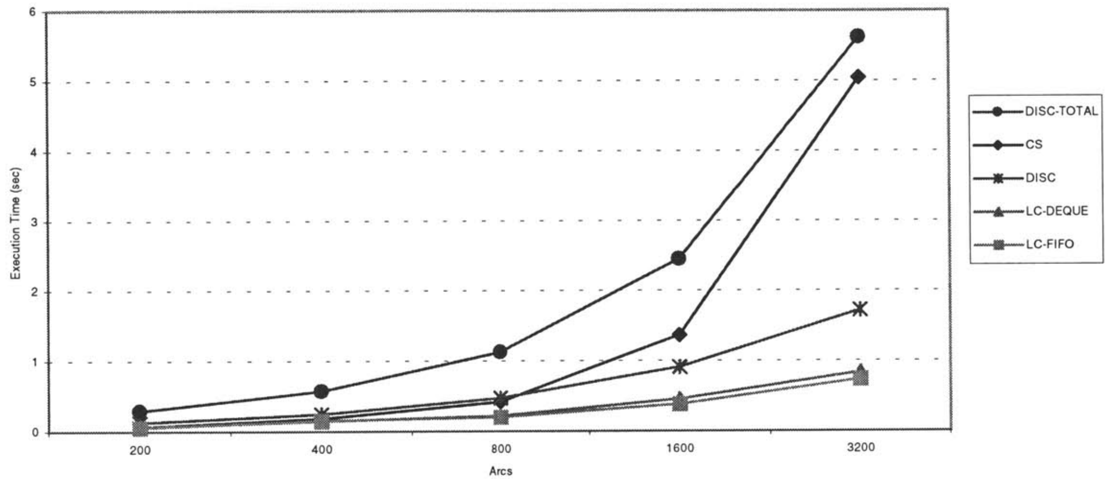
Figure 1: Execution Time vs. Network Size (Planar, FIFO, 25% Dynamic, 10 Pieces / Arc)



Note that the horizontal axis in Figure 1 and in the remaining figures does not follow a linear scale. The running time of the DISC-TOTAL algorithm for the large 3000x9000 network was 20.11 seconds, and was omitted for better scaling of the entire graph.

Continuing with our analysis, Figure 2 shows the relationship between running time and number of arcs, where the number of nodes is held constant at 100. As before, the network is planar in nature, and 25% of the arcs have been assigned time-varying travel time functions with 10 linear pieces which satisfy the FIFO property. According to this result, the label-correcting algorithms tend to scale better with increasing network density. The running time of the discrete-time solution algorithm scales linearly with density, as is expected.

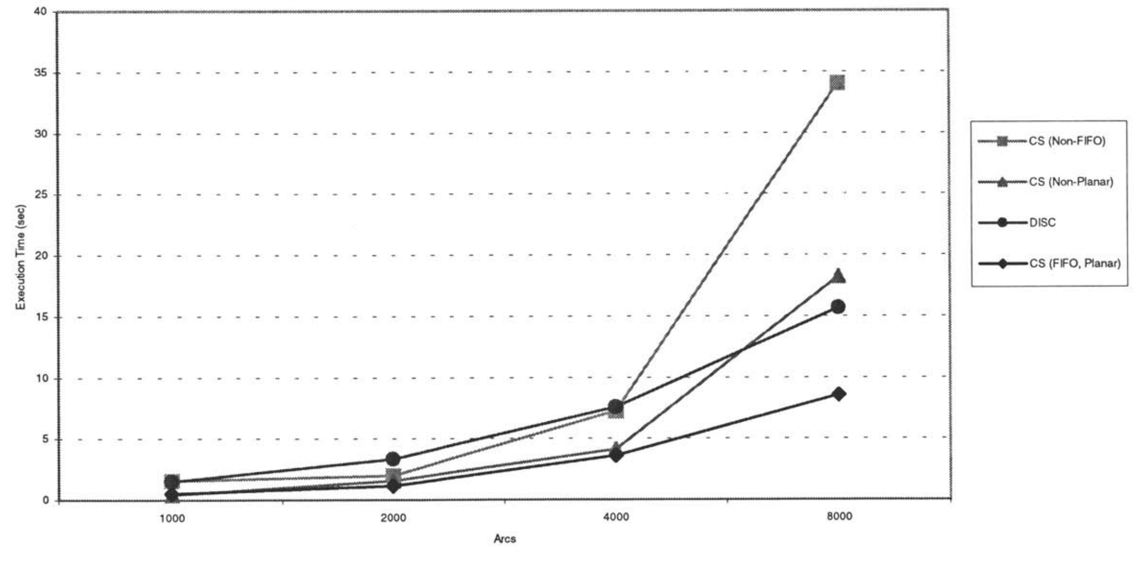
Figure 2: Execution Time vs. Number of Arcs (Planar, FIFO, 100 Nodes, 25% Dynamic, 10 Pieces / Arc)



The average number of LNIs per node encountered while performing the sample runs for Figure 2 was observed to fall between 8 and 16, for more than an order of magnitude of savings in space compared with the discrete-time algorithm.

Figures 3 and 4 demonstrate the negative effects on running time which can occur as a result of non-FIFO or non-planar networks. In non-FIFO networks, it is possible that solution functions may consist of a very large number of LNIs due to cycles within shortest paths. In non-planar networks, the length of a shortest path, in terms of arcs, may possibly be very large, which will also result in a large number of LNIs within each solution function. Each network tested in Figures 3 and 4 was randomly generated with 500 nodes, 25% dynamic behavior, and 10 random linear pieces in each dynamic arc travel time function. Results indicate that the performance of label-correcting algorithms is degraded far more than that of chronological scan algorithms by non-FIFO and non-planar network conditions. This behavior is expected, as the cost of correcting labels is proportional to the number of LNIs present in the solution functions, and if there are cycles in optimal paths or if optimal paths contain many arcs, then these labels are likely to be corrected many times, leading to a decrease in performance for label-correcting algorithms. The running time of the discrete-time algorithm, shown in Figure 3, is shown not to depend on properties of network data, as is expected. Finally, the anomalous shape of the result curves in Figure 4 is due to the fact that some of the random networks tested happened to exhibit high numbers of LNIs – there was an average of 72 LNIs per node over all networks generated with 1000 arcs. This average number was the maximum observed for any of the samples cases in Figures 3 and 4, and still represents an order of magnitude in savings in terms of memory as compared to the discrete-time algorithm.

Figure 3: Effect of Non-FIFO and Non-Planar Conditions on Execution Time: Chronological Scan Algorithms (500 Nodes, 25% Dynamic, 10 Pieces / Arc)



In Figures 5 and 6, we examine the impact of complexity of the time-dependent network data functions on the running time of solution algorithms. In Figure 5, random planar networks with 500 nodes and 1500 arcs were generated, where each time-dependent arc travel time function has 10 linear pieces and satisfies the FIFO condition. Running time is then shown as a function of the percentage of arcs in the network randomly selected to exhibit dynamic behavior (the remaining arcs have static travel time functions).

Figure 4: Effect of Non-FIFO and Non-Planar Conditions on Execution Time: Label-Correcting Algorithms (500 Nodes, 25% Dynamic, 10 Pieces / Arc)

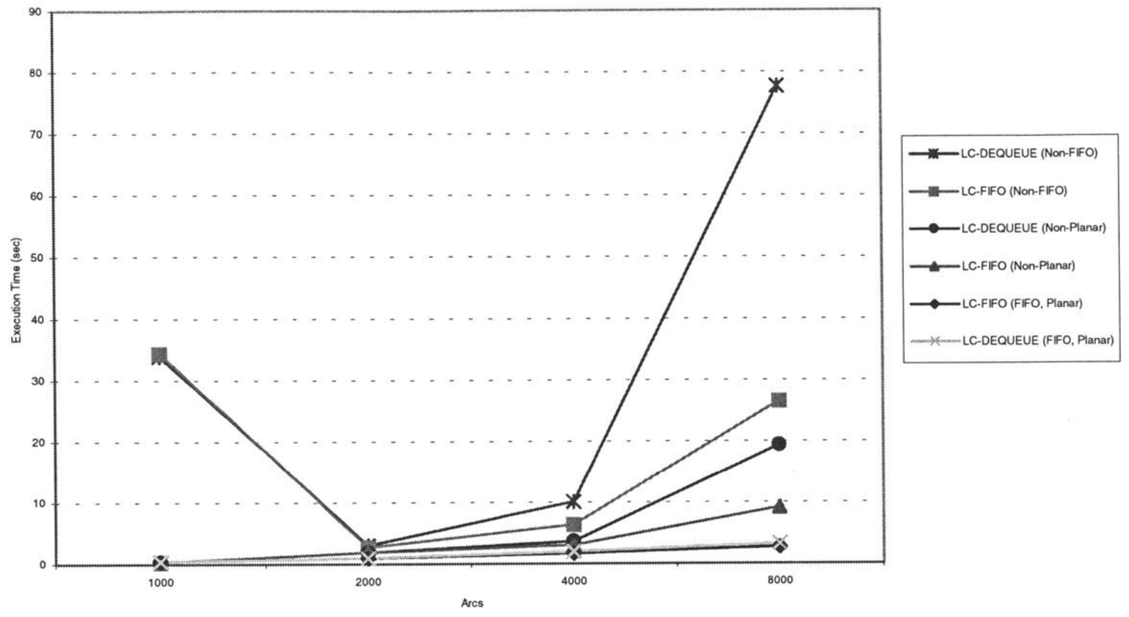
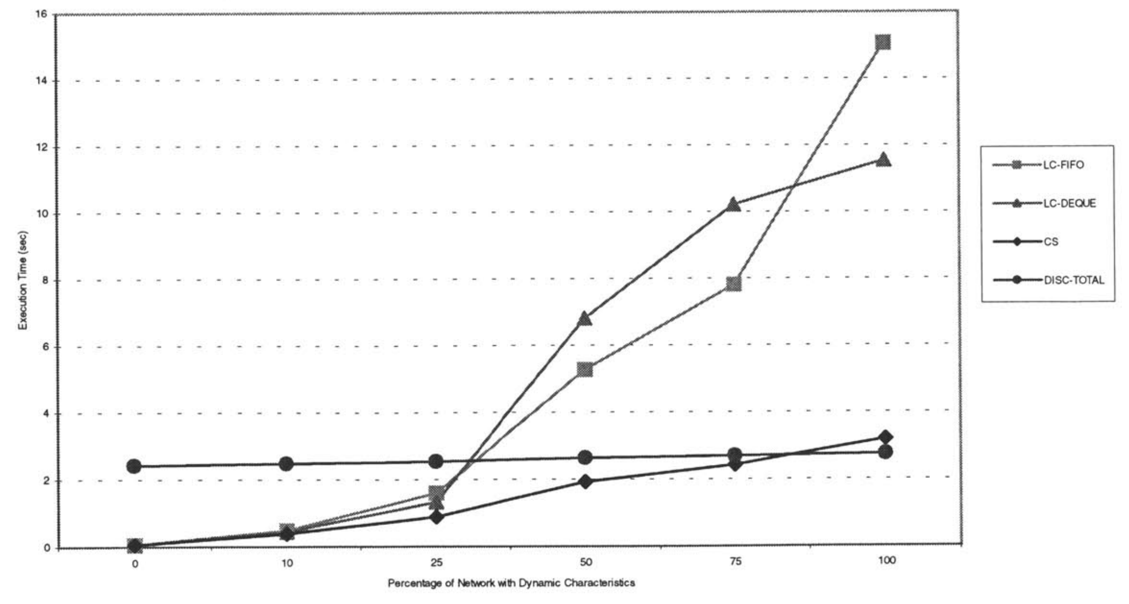


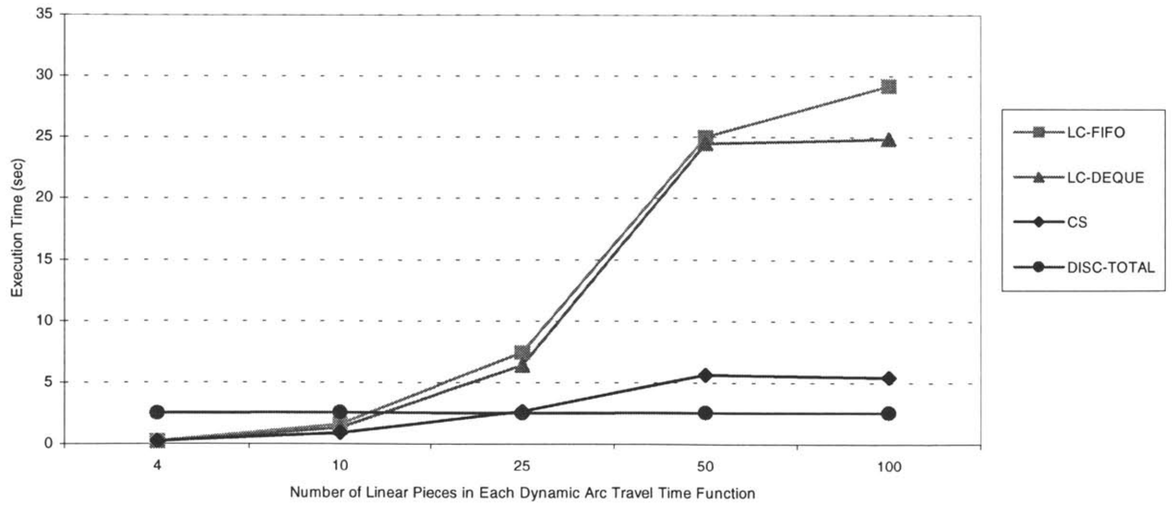
Figure 5: Execution Time vs. Dynamic Fraction of Network (Planar, FIFO, 500 Nodes, 1500 Arcs, 10 Pieces / Arc)



The running time of the discrete-time algorithm is shown above not to depend on complexity of network data functions, as is expected. The performance of the chronological scan algorithm appears to scale far better than that both variants of the label-correcting algorithm, which is expected based on the previous discussion of the impact of complex solution functions upon the label-correcting algorithms. The average number of LNIs comprising the solution functions was, as would be expected, greatest for the case where 100% of the network was dynamic – in this case, there were 70 LNIs per node on average. There is still an order of magnitude in memory savings, even in this case, compared with the discrete-time algorithm.

In Figure 6, we generate random planar networks with 500 nodes and 1500 arcs, each with 25% dynamic, FIFO behavior. The parameter we vary in this case is the number of linear pieces comprising those arc travel time functions which are chosen to exhibit dynamic behavior. Again, the chronological scan algorithm scales in running time much better than either of the label-correcting algorithms. An average of 130 LNIs per solution function (the maximum observed for this test) was observed for the sample case where there were 50 pieces in each dynamic arc travel time function. Additionally, the discrete-time algorithm is shown not to be sensitive to the complexity of input data, as expected.

Figure 6: Execution Time vs. Linear Pieces in Network Data Functions



In addition to networks with random topology, computational testing was performed on two models of real networks. The Amsterdam A10 outer beltway is a ring network with 196 nodes and 310 arcs. For this network, we assigned random FIFO dynamic travel time functions with a 4-piece triangular profile to 25% of the arcs. For this case, each continuous-time algorithm ran in 0.05 seconds, and the discrete-time algorithm (with sampling included) required 0.5 seconds. On average, there were 12 LNIs per solution node. Additionally, we ran the same test with 25% of the arcs in the network assigned random 10-piece FIFO travel time functions. In this instance, the chronological scan algorithm required 0.11 seconds, the label-correcting algorithms required 0.22 seconds, the discrete-time algorithm (with sample) required 0.44 seconds, and there were on average 26 LNIs per solution function. Finally, a very large network model of the city of Montreal, consisting of 6906 nodes and 17157 arcs, was tested, where 5% of arcs in the network were assigned 4-piece FIFO dynamic travel time functions. For this case, the discrete-time algorithm required 53.27 seconds, the label-correcting algorithm based on a FIFO queue required 24.01 seconds, the dequeue-based label-correcting algorithm required 10.88 seconds, the chronological scan algorithm required 3.35 seconds, and there were 6 LNIs per solution function on average.

In summary, continuous-time methods appear to offer substantial improvements both in terms of running time and memory requirements over existing discrete-time algorithms, provided that dynamic network conditions are sufficiently well-behaved. For FIFO planar networks of low degree, where a relatively small percentage of the network is actually dynamic, and where the dynamic part of the network exhibits relatively simple behavior, continuous-time approaches appear to be a good deal more efficient in both time and space than their discrete-time counterparts. This analysis supports the conclusion that continuous-time methods may be more favorable to incorporate in network optimization systems designed around many different types of dynamic network scenarios, particularly those related to transportation networks.

Chapter 9

Conclusions

Below, we give a summary of the results and contributions developed in this thesis, followed by suggestions for future research.

9.1 Summary of Results

The developments in this thesis originated from a need for more efficient models and algorithms for computing shortest paths in dynamic networks than existing discrete-time methods. In order to escape the inherent restrictions of discrete-time models, we considered instead the use of continuous-time models, and argued that computation within this domain was practical only with a simplifying piece-wise linearity assumption on all time-dependent network data functions. Adopting this assumption, we discussed the classification of all common problem variants of the dynamic shortest path problem and formulated these problems mathematically in continuous time.

Mathematical properties of dynamic networks and dynamic shortest path problems, especially those involving FIFO networks, were discussed. Initially, we proved the existence and finiteness of a solution for all dynamic shortest path problem variants. In addition to several other important properties of FIFO networks, we showed strong polynomial bounds of $O(P^*)$ linear pieces in the solution function of each node, and it was shown that the minimum-time one-to-all problem for all departure times is computationally equivalent to the minimum-time all-to-one problem in a FIFO network.

The vast majority of the discussion in this thesis is centered around the development of two broad classes of solution algorithms, either of which can be applied to solve all variants of the continuous-time dynamic shortest path problem. One of these classes, the set of label-correcting algorithms presented in Chapter 7, represents a generalization of previous theoretical algorithmic results by Orda and Rom [16]. These algorithms have the advantage that they are simple to state and efficient in practice. The second class of solution algorithms, the chronological scan algorithms presented in Chapter 6, are slightly more difficult to state, but perform equally well in practice and have stronger theoretical running times. These algorithms, while designed for dynamic networks, are extremely well-equipped to handle other types of extended static problems. For instance, static shortest path problems with time windows and time-constrained minimum-cost static shortest path problems are likely to be solved extremely efficiently by the methods outlined in this thesis. In fact, the dynamic algorithms developed in this thesis may well represent one of the most effective solution methods for these types of extended static problems.

For the minimum-time all-to-one problem and the minimum-time one-to-all problem for all departure times in a FIFO network, we derived strong polynomial worst-case running time bounds of $O(mP^* \log n)$ and $O(nmP^*)$ respectively for the chronological scan and label-correcting algorithms. Although we show that the more general minimum-cost and non-FIFO minimum-time dynamic shortest path problems are NP-Hard, we argue that the expected running time required to solve most common problems should be quite reasonable. For instance, for problems in sparse, bounded degree, networks with a planar character which involve "reasonable" dynamics consisting of at most a constant number of linear pieces, the expected asymptotic running time for the chronological scan algorithm and for the label-correcting algorithm should be on the order of only $n^{0.5}$ times greater than that of an efficient static shortest path algorithm applied to the same network. Furthermore, for any FIFO problem in a bounded-degree network, we show that the running time of the chronological-scan algorithm is only a factor of $\log n$ greater than the combined size of the network data functions specified as input and the solution functions produced as output by the algorithm, and therefore only a factor of $\log n$ away from the best possible running time for which one could ever hope. For all minimum-time problems in a FIFO network, we additionally proposed methods to partition a problem into disjoint sub-problems for parallel computation, and proposed approximation techniques for further speeding up the chronological scan algorithms.

The computational study of this thesis showed that the performance of continuous-time methods is comparable to that of discrete-time methods, especially for cases in which a relatively fine discretization is applied to time. For certain problems in which the dynamic nature of the network is not excessively complicated, we have observed as much as an order of magnitude of savings in running time, and often more than an order of magnitude of savings in terms of memory requirements by using continuous-time methods.

9.2 Future Research Directions

There are numerous research directions one may wish to follow as an extension of the work of this thesis. Although all common variants of the dynamic shortest path problem are addressed and solved within this thesis, there are several possibilities for future work relating to this problem. Since the running time of the chronological scan algorithms developed in this thesis are a factor of *log n* away from optimality for solving problems in bounded-degree FIFO networks, and arguably far from optimality for non-FIFO networks, there exists the possibility that stronger solution algorithms may exist. The brief treatment devoted to hybrid continuous-discrete chronological scan algorithms can certainly be expanded to encompass a more rigorous treatment of approximation algorithms for continuous-time dynamic shortest path problems, and approximation algorithms derived from label-correcting methods may be possible to develop. Finally, the one-to-all problem for all departure times has only been addressed within the context of FIFO networks; no solution techniques currently exist for this problem variant for non-FIFO networks or minimum-cost problems.

Another exciting prospect for future research is the possibility of applying the algorithmic methodology developed in this thesis to other continuous-time dynamic network optimization problems. The general approach employed by the chronological scan algorithms of this thesis may be well-suited to solving many other problems in dynamic networks with piece-wise linear characteristics, such as the maximum-flow problem, the minimum-cost flow problem, the multi-commodity flow problem, and the traveling salesman problem.

Finally, continuous-time dynamic shortest path algorithms may be used as a component within larger network optimization systems. For example, in the rapidly developing field of *Intelligent Transportation Systems (ITS)*, there is a growing need for algorithms which can efficiently compute route guidance information for drivers on a dynamic transportation network. Dynamic shortest path problems are a component within larger network optimization problems such as dynamic routing and dynamic traffic assignment problems, and with the advent of continuous-time dynamic shortest path algorithms, advances may be possible to some of these larger problems. In particular, since there is no discretization of data, the exact mapping from arc travel time and costs to optimal path travel times and costs given by continuous-time models preserves the analytical properties of mathematical solution algorithms, allowing for more robust mathematical analysis of solution algorithms to these large-scale problems.

Appendices

Appendix A. The Min-Queue Data Structure

The minimum-cost all-to-one and one-to-all chronological scan algorithms presented in this thesis can achieve greater efficiency if the set of feasible waiting departure times at each waiting node is represented using an augmented queue data structure, which we call a *min-queue*. A min-queue is a queue which supports the following operations, and which is able to perform any set of N of these operations in $O(N)$ time, so that each individual operation requires $O(1)$ amortized time.

- Insertion of a value at the head of the queue,
- Removal of the value at the tail of the queue,
- Computation of the minimum value and the index of its position in the queue,
- Addition of a constant to every value in the queue.

One can represent a min-queue as a 6-tuple $Q = (V, P, H, T, M, K)$, where the values in the queue are given by V_i, and H and T give the indices of the head and tail of the queue, respectively (H actually points to the index directly ahead of the first element, where the next inserted element should be placed). The queue is empty if $H = T$. An index to the minimum element of the queue is stored in M at all times. Additionally, the values P_i contain the index of the previous minimum value in the queue, before the value V_i was added. The value K holds a number, initially zero, which is to be added to all elements of the min-queue. The following pseudo-code provides a straightforward implementation of the create, insert, find minimum, and global add operations:

```
MinQueue-Create ()
$T \leftarrow 1$
$H \leftarrow 1$
$M \leftarrow \varnothing$
$K \leftarrow 0$
Return Q

MinQueue-InsertElement (Q, Value)
$V_H \leftarrow$ Value - K
$P_H \leftarrow M$
If Value < $V_M$ Then $M \leftarrow H$
$H \leftarrow H + 1$
```

```
MinQueue-MinElement (Q)
Return $V_M + K$

MinQueue-MinIndex (Q)
Return H - M

MinQueue-AddToAllElements (Q, Value)
$K \leftarrow K + Value$
```

Removal of the last element is a bit more difficult. We must to update *M* such that it still points to the minimal element of *Q*. Normally, we can set *M* to be the value of P_T. However, if P_T happens to point backward to an index no longer in the queue (that is to say, $P_T < T$), then we must go through the entire queue in reverse to reconstruct the values of P_i such that they all point forward to the next-smallest element in the min-queue. We implement this reconstruction operation and the remove operation as follows:

```
Reconstruct (Q)
$M \leftarrow \varnothing$
For $i \leftarrow H-1 .. T$
  $P_i \leftarrow M$
  If $M = \varnothing$ or $V_i < V_M$ Then $M \leftarrow i$

MinQueue-RemoveLastElement (Q)
If $P_T < T$ Then Reconstruct(Q)
$M \leftarrow P_T$
$T \leftarrow T + 1$
```

Each of the above operations require *O(1)* time to complete, except for the removal of the last element in the queue – this removal may trigger a reconstruct operation which requires *O(N)* time, where there are *N* elements in the queue. However, after a reconstruct operation is performed, the elements in the min-queue will have forward-pointing P_i pointers, and will therefore not cause a reconstruct operation upon their removal, so that a reconstruct operation may only occur after another *N* removals, thus justifying the amortized running time of *O(1)* time per operation.

Appendix B. Raw Computational Results

The following table contains all data collected during computational testing. Each line contains the result of running all implemented algorithms in turn on a single test network. The types of networks and exact implementations of the algorithms used for these tests are described in Chapter 8.

The *Network Type* column gives the particular network used for testing: *A* for the Amsterdam outer beltway, *M* for the network model of Montreal, *RP* for a random planar network, and *R* for a completely random network. If the *Triangular $d_{ij}(t)$ Functions?* column is checked, then random arc travel time functions were generated with a triangular profile, described in Chapter 8; otherwise, random piece-wise linear arc travel time functions were generated. The *% Dynamic* column indicates the percentage of the arcs in the network with time-varying travel time functions; all other arcs are given static travel times. For those arcs with dynamic travel time functions, the *# Linear Pieces* column gives the number of linear pieces they contain. The two discrete-time running time columns on the far right of the table give running times for discretizing and solving a discrete-time dynamic shortest path problem using 1000 discrete levels of time; Chapter 8 contains further comments regarding these algorithms. All running times are given in seconds.

Network Type	# Nodes	# Arcs	FIFO?	"Triangular" $d_{ij}(t)$ Functions?	% Dynamic	# Linear Pieces	Average # of Solution LNIs per Node	(FIFO queue) Label-Correcting Running Time	Dequeue) Label-Correcting Running Time	Chronological Scan Running Time	Discretization Running Time	Discrete-Time Dynamic SP Running Time
RP	10	30	•		25	10	3	0.000	0.005	0.005	0.000	0.016
RP	30	90	•		25	10	5	0.011	0.011	0.011	0.110	0.055
RP	100	300	•		25	10	9	0.082	0.077	0.071	0.275	0.187
RP	300	900	•		25	10	13	0.401	0.357	0.379	0.769	0.599
RP	1000	3000	•		25	10	16	1.720	1.593	1.676	3.626	2.253
RP	3000	9000	•		25	10	9	5.220	4.450	7.200	12.58	7.530
RP	100	200	•		25	10	16	0.055	0.060	0.066	0.165	0.126
RP	100	400	•		25	10	13	0.159	0.148	0.181	0.330	0.247
RP	100	800	•		25	10	9	0.203	0.214	0.418	0.659	0.473
RP	100	1600	•		25	10	9	0.379	0.451	1.368	1.538	0.907
RP	100	3200	•		25	10	8	0.742	0.835	5.044	3.901	1.720

RP	500	1000	●		25	10	21	0.385	0.407	0.522	0.897	0.709
RP	500	1000			25	10	72	34.335	33.918	1.560	0.879	0.709
R	500	1000	●		25	10	18	0.324	0.330	0.429	0.879	0.637
RP	500	2000	●		25	10	14	0.962	0.912	1.137	2.033	1.379
RP	500	2000			25	10	25	2.659	2.989	1.987	2.033	1.374
R	500	2000	●		25	10	21	1.830	1.868	1.588	2.088	1.231
RP	500	4000	●		25	10	14	1.692	1.929	3.610	5.275	2.676
RP	500	4000			25	10	29	6.346	10.137	7.269	5.220	2.665
R	500	4000	●		25	10	16	3.020	3.630	4.120	5.220	2.360
RP	500	8000	●		25	10	9	2.750	3.190	8.570	11.10	5.270
RP	500	8000			25	10	38	26.480	77.750	34.070	11.15	5.330
R	500	8000	●		25	10	20	9.180	19.340	18.240	11.21	4.440
A	196	310	●	●	25	4	12	0.050	0.050	0.050	0.275	0.220
A	196	310	●		25	10	26	0.220	0.220	0.110	0.275	0.160
M	6906	17157	●	●	5	4	6	24.010	10.880	3.350	28.57	13.700
RP	500	1500	●		0	10	1	0.050	0.050	0.050	1.319	1.100
RP	500	1500	●		10	10	9	0.490	0.440	0.380	1.374	1.100
RP	500	1500	●		25	10	21	1.590	1.320	0.880	1.429	1.100
RP	500	1500	●		50	10	43	5.270	6.810	1.920	1.538	1.100
RP	500	1500	●		75	10	53	7.800	10.220	2.420	1.538	1.150
RP	500	1500	●		100	10	70	15.050	11.540	3.190	1.648	1.100
RP	500	1500	●	●	25	4	7	0.270	0.220	0.270	1.429	1.100
RP	500	1500	●		25	10	21	1.590	1.320	0.930	1.429	1.150
RP	500	1500	●		25	25	62	7.470	6.430	2.690	1.429	1.100
RP	500	1500	●		25	50	130	25.000	24.450	5.660	1.429	1.150
RP	500	1500	●		25	100	112	29.180	24.840	5.440	1.484	1.100

References

[1] B. H. Ahn, J. Y. Shin (1991), "Vehicle routing with time windows and time-varying congestion". *J. Opl. Res. Soc.* 42, 393-400.

[2] R. Ahuja, T. Magnanti, J. Orlin (1993). *Network flows: Theory, algorithms, and applications.* Prentice Hall, Englewood Cliffs, NJ.

[3] I. Chabini (1997). "A new algorithm for shortest paths in discrete dynamic networks". *Proceedings of the IFAC Symposium on Transportation Systems.* Chania, Greece.

[4] I. Chabini (1998). "Discrete Dynamic Shortest Path Problems in Transportation Applications: Complexity and Algorithms with Optimal Run Time". *Transportation Research Record* 1645.

[5] I. Chabini and B. Dean (1999). "The Discrete-Time Dynamic Shortest Path Problem: Complexity, Algorithms, and Implementations". Submitted for Publication.

[6] I. Chabini, M. Florian, N. Tremblay (1998). "Parallel Implementations of Time-Dependent Shortest Path Algorithms". Spring INFORMS National Meeting, Montreal, Quebec, April 1998.

[7] L. Cooke and E. Halsey (1966). "The shortest route through a network with time-dependent internodal transit times". *Journal of Mathematical Analysis and Applications* 14, 492-498.

[8] C. Daganzo (1998). "Symmetry Properties of the Time-Dependent Shortest Path Problem With FIFO". Private Communication with Ismail Chabini.

[9] B. Dean (1997). "Optimal Algorithms for Computing Shortest Paths in Dynamic Networks with Bounded Waiting at Vertices". Internal Report.

[10] S. Dreyfus (1969). "An appraisal of some shortest-path algorithms". *Operations Research* 17, 395-412.

[11] S. Ganugapati (1998). "Dynamic Shortest Path Algorithms: Parallel Implementations and Application to the Solution of Dynamic Traffic Assignment Models". Master's Thesis, MIT Center for Transportation Studies.

[12] J. Halpern (1977). "Shortest Route with Time-Dependent Length of Edges and Limited Delay Possibilities in Nodes". *Zeitschrift fur Operations Research* 21, 117-124.

[13] I. Ioachim, S. Gelinas, F. Soumis, J. Desrosiers (1998). "A Dynamic Programming Algorithm for the Shrotest Path Problem with Time Windows and Linear Node Costs". *Networks* 31, 193-204.

[14] D. E. Kaufman, R. L. Smith (1993). "Fastest paths in time-dependent networks for intelligent-vehicle-highway systems application". *IVHS Journal* 1, 1-11.

[15] A. Orda and R. Rom (1990). "Shortest-path and minimum-delay algorithms in networks with time-dependent edge length". *Journal of the ACM* 37 (3), 607-625.

[16] A. Orda and R. Rom (1991). "Minimum weight paths in time-dependent networks". *Networks* 21 (3), 295-320.

[17] S. Pallottino and M. Scutella (1998). "Shortest Path Algorithms in Transportation Models: Classical and Innovative Aspects". In (P. Marcotte and S. Nguyen, Eds.) Equilibrium and Advanced Transportation Modelling. Kluwer 245-281.

[18] H. Sherali, K. Ozbay, S. Subramanian (1998). "The Time-Dependent Shortest Pair of Disjoint Paths Problem: Complexity, Models, and Algorithms". *Networks* 31, 259-272.