CONTINUOUS TIME ALGORITHMS FOR A VARIANT OF THE DYNAMIC TRAFFIC ASSIGNMENT PROBLEM

by

Jennifer M. Farver

B.S. Civil Engineering
University of California, Berkeley, 1998

Submitted to the Department of Civil and Environmental Engineering
in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Transportation
at the
Massachusetts Institute of Technology

June 2001

Signature of Author .....
Department of Civil and Environmental Engineering
June 1, 2001

Certified by .....
Ismail Chabini
Assistant Professor of Civil and Environmental Engineering
Thesis Supervisor

Accepted by .....
Oral Buyukozturk
Chairman, Departmental Committee on Graduate Studies

CONTINUOUS TIME ALGORITHMS FOR A VARIANT OF THE DYNAMIC TRAFFIC ASSIGNMENT PROBLEM

by

Jennifer M. Farver

Submitted to the Department of Civil and Environmental Engineering
on June 1, 2001 in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Transportation

ABSTRACT

The Dynamic Traffic Assignment (DTA) Problem is relevant to many transportation contexts, including planning and Intelligent Transportation Systems. Existing models have yielded approximate solutions to this problem using discrete-time methods. We present and implement several algorithms that enable exact continuous-time solution.

The first of these algorithms is a practical continuous-time algorithm for a variant of the Dynamic Network Loading Problem (DNLP) in which input parameters take on a particular functional form. Specifically, path entrance flow rate functions are assumed to be stepwise and arc performance functions are assumed to be affine. These assumptions ensure that the resulting arc flow rate functions will be stepwise and enable computation of an exact solution. The second algorithm is an adaptation of the Bellman-Ford shortest paths algorithm to continuous-time, dynamic networks. We present algorithms for both the one-to-all and all-to-one variants of the dynamic shortest path problem. Both the DNLP and Dynamic Shortest Paths algorithms are implemented and tested on a sample network.

Finally, we present a DTA solution algorithm which uses the DNLP and DSP algorithms to find a continuous-time solution. The DTA algorithm is also implemented and tested on a sample network, thus a continuous-time DTA solution is found.

Thesis Supervisor: Ismail Chabini
Title: Assistant Professor of Civil and Environmental Engineering

ACKNOWLEDGEMENTS

The author would like to thank Ismail Chabini for his guidance, insights and the patience he consistently showed throughout the preparation of this thesis. She would also like to thank her parents, Tom and Phyllis Farver for their love and support and Ronak Bhatt for providing constant encouragement.

Contents

CHAPTER 1

INTRODUCTION

The goal of Intelligent Transportation Systems (ITS) is to use technology to better exploit the physical capacity of existing transportation infrastructure. In particular, many of the technologies employed in ITS serve to collect, process, disseminate, and exploit information about the state of the transportation system. Information technology in ITS has many applications including commercial vehicle operations and non-predictive traveler information, however it is the ability to predict the future state of the transportation network that may provide the most significant advances in transportation efficiency. Predictive transportation models have applications for the transportation planner, the transportation service provider and the system user, providing tools for forecasting as well as for real-time route guidance.

Dynamic Traffic Assignment (DTA) models are predictive tools that, given O-D demands and network characteristics, determine demand on individual network components and the resulting cost of trips within the network. When coupled with demand models, DTA models provide predictive information that can answer various questions such as: "How congested will a given arc be in the future?" and "How long will

my commute be today? Should I take the usual route or an alternate route?" DTA models have many applications, but are particularly important in Advanced Traffic Management Systems in which they can be used to optimize the network via signals, changeable message signs, and other traffic control devices. In such applications, the goal is often to implement systems of real-time monitoring and control, which require DTA models and efficient solution algorithms. In this thesis we explore the properties and promise of a continuous-time DTA model and explore its advantages over discrete-time formulations.

1.1 The DTA Problem

Consider a network composed of arcs and nodes. A path is a set of successive arcs used to travel from some origin node to some destination node in our network. Each path or arc experiences some amount of flow as vehicles travel through the network from many origin nodes to many destination nodes. The DTA problem is to find time-dependent arc travel times, path travel times and arc flows, given a network with time-dependent origin-destination travel demands, a model of the relationship between arc flows and arc travel times and some assumptions regarding user route choice.

The DTA problem can be formulated by either simulation or analytical methods. While simulation has been used extensively to date to obtain DTA solutions, analytical models present several distinct advantages. Firstly, they require only one "run" to obtain a solution and there is no variability in the solution obtained. Secondly, analytical models can be constructed such that they possess theoretical attributes that permit the development of efficient solution algorithms. For these reasons, in this thesis we present an analytical formulation of the DTA problem.

Figure 1 shows a simple DTA framework. Given a set of time-dependent path flows on a network, we calculate flows on the component arcs. From these arc flows, we can determine the total flow on a given arc at any time in the analysis period. Travel time on

an arc is dependent on these total flows, as described by arc performance functions. By composing the exit time functions of arcs along a path, we can then calculate the time-dependent travel time on each path. We then use a shortest path algorithm to determine minimum time paths between each O-D pair and assign O-D demands to these paths to obtain a set of path flows. These flows are then averaged with those of previous iterations. DTA algorithms seek to find a set of path flows such that traversing the DTA loop no longer yields a significant change in the values of the variables.

The DTA problem is composed of several sub-problems for which specialized algorithms are developed. To calculate path costs from arc costs, we use a dynamic variant of a standard shortest path algorithm. Calculating arc travel times from path flow rates is a sub-problem referred to as the Dynamic Network Loading Problem (DNLP) for which DNLP algorithms are developed. In this thesis we present a continuous-time DNLP model and a corresponding continuous-time Dynamic Shortest Paths (DSP) solution algorithm. For each, we first present the theoretical background of the problem before presenting the algorithm itself. We then present a DTA algorithm which uses the above algorithms as subroutines to find a continuous-time DTA solution.

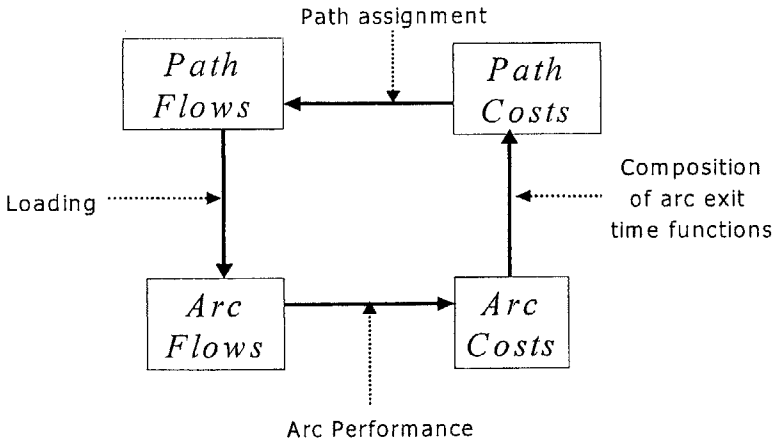


Figure 1: Components of a DTA Model

1.2 Motivation for a Continuous Time Dynamic Traffic Assignment Model

Previous, discrete-time DTA models approximate the behavior of a network, much as numerical integration methods approximate the properties of a function. Time is discretized in small intervals and we make approximations at each interval. Like numerical integration methods, the efficiency of discrete-time models can be improved by decreasing the width of the interval; this is desired particularly in regions of sharp change. From a computational standpoint, the compromise made by increasing accuracy is one of efficiency since a greater number of iterations results in a greater run time. More sophisticated approaches may attempt to balance efficiency and accuracy by performing more computations in regions of rapid change and fewer in less dynamic regions. The success of these approaches however, is limited by the amount of *a priori* knowledge about the network's dynamics.

Continuous-time methods may have two primary advantages over their discrete-time analogs. First, given functional forms that allow us to obtain exact integrals on a computer, continuous-time algorithms may yield exact solutions. Instead of approximating network behavior over an interval by calculating parameters at many intermediate points, a continuous-time algorithm can describe network behavior in functional form. Thus, values of network parameters may be known exactly for all times contained in the interval of interest.

While the continuous-time approach yields algorithms with the advantage of being the most accurate available, it also provides a tool for benchmarking approximate algorithms. Various algorithms can then be compared not only based on run time, but on their error as compared to an exact, continuous-time solution.

A second advantage of continuous-time algorithms is that they are efficient in the sense that they may perform exactly the number of computations required to obtain a solution.

Values of network variables need only be determined at certain time instants; doing so allows us to completely specify the their values at all times in the analysis period. Thus, network behavior is completely and exactly specified by performing the minimum number of computations and a continuous-time model may prove more successful at responding to a network's dynamics than a discrete-time model. This may have implications for the algorithm's efficiency as well as for its data storage requirements, particularly for networks in which certain time intervals or arcs are much more dynamic than others.

1.3 Objectives and Contributions

The objectives of this thesis are to:

- develop an exact, continuous time algorithm for the Dynamic Network Loading Problem, assuming stepwise path flow rate functions, affine arc performance functions and a deterministic, user-optimum users' behavior model;
- illustrate the concepts of the continuous-time Dynamic Shortest Paths problem and provide a simple algorithm for its solution;
- present a Dynamic Traffic Assignment algorithm that uses the above DNLP and DSP algorithms as subroutines to obtain an exact, continuous-time DTA solution; and
- implement these algorithms and test them on a small network in order to verify their correctness and illustrate properties of their solutions.

1.4 Background

In the following section we provide a brief summary of the developments in the literature that are pertinent to our problem. This includes work on discrete-time variants of the component algorithms as well as any existing continuous-time methods.

1.4.1 The DTA Problem

Existing work on the DTA problem is based on the DTA framework shown in Figure 1. Algorithms are developed for the Dynamic Network Loading Problem and the Dynamic Shortest Paths problem and a users' behavior model is adopted to reflect how flow is allocated among paths, once path travel times are known. In a simple users' behavior model, all users may choose the path of minimum travel time from their origin to their destination, while a more complex model may take into account the user's perceptions of travel time and route choice preferences. Inputs to the problem are O-D flow rates and the arc performance functions that model arc travel time as a function of the total flow on an arc.

Existing DTA algorithms have discretized time, calculating network variables for each time interval in the analysis period. Thus, discrete-time Dynamic Network Loading and Dynamic Shortest Paths algorithms exist in the literature. For the continuous-time problem, continuous-time Dynamic Shortest Paths algorithms have been developed, however no practical continuous-time Dynamic Network Loading algorithms exist, to date. In order to generate a continuous-time DTA solution, it is necessary to develop a continuous-time Dynamic Network Loading algorithm and ensure that DTA loop can be traversed using continuous-time variables. The reader is referred to Chabini and He (2000) and Chabini and Kachani (1999) for an overview of analytical DTA models.

1.4.2 The Dynamic Network Loading Problem

The Dynamic Network Loading Problem is to determine arc travel times, given a network with arc performance functions and time-dependent path flows. Previous work on the Dynamic Network Loading problem has focused both on its mathematical solution properties and on the development of solution algorithms.

In Chabini and Kachani (1999), various properties of the DNLP were established for the class of problems in which arc performance functions are continuously differentiable and path flow rates are *Lebesgue Integrable*. The results are generalized for models that are linear or non-linear. The proven properties are:

1. the strong FIFO property is verified (the FIFO property will be discussed in Chapter 2),
2. the exit flow rate function is *Lebesgue Integrable*, nonnegative and bounded from above, and
3. the DNLP possesses a solution and this solution is unique.

The reader is referred to Chabini and Kachani (1999) for details of the proofs including the bounds on various function values.

Existing algorithms for the Dynamic Network Loading problem are of two types: those that iterate over network components and those that move chronologically. The first type, called *I-Load*, operate in a fixed-point manner, searching for a solution that verifies the constraints, using the Method of Successive Averages as a solution approach. The second, *C-Load*, move forward in time, calculating arc parameters as flow is propagated through the network. Discretized time intervals are used in these computations. Computational results of these two algorithms have been found to be similar (Chabini and He 2000), however because neither convergence nor uniqueness of solution for I-Load can be proven, C-Load has been recommended as the preferable network-loading algorithm.

1.4.3 The Dynamic Shortest Paths Problem

The Dynamic Shortest Paths problem is to determine the time-dependent shortest paths in a network, given its time-dependent arc travel times. Like the static shortest paths problem, the dynamic problem has several variants including one-to-all nodes and all-to-one node. Furthermore, in this dynamic problem we may define the problem in terms of

either the departure times of flows, or in terms of their arrival times. The Dynamic Shortest Path problem can also be posed for FIFO or non-FIFO networks, however the networks that verify the FIFO property are of interest in this thesis.

Existing algorithms for the discrete-time Dynamic Shortest Paths problem include dynamic extensions of well-known static shortest paths algorithms including the label-correcting algorithm and the Bellman-Ford algorithm. Additional algorithms, such as Algorithm DOT (Chabini 1997, 1998) have been developed specifically for the dynamic problem. Testing of these algorithms has shown that Algorithm DOT is faster. The Bellman-Ford algorithm also performed well as compared to label-correcting.

For the continuous-time variant of the dynamic shortest paths problem, the label-correcting algorithm and Algorithm DOT have been implemented. In this thesis, we consider a continuous-time adaptation of the Bellman-Ford algorithm. The reader is referred to Dean (1999) for a more exhaustive treatment of the dynamic shortest paths problem, with particular attention given to the continuous-time variant.

1.5 Thesis Outline

The goal of this thesis is to formulate and implement a continuous-time DTA solution algorithm. To do so, we will also formulate and implement continuous-time DNLP and DSP algorithms. All of these developments will rely on networks whose parameters take on functional forms which permit exact continuous-time solutions. Arc performance functions are assumed to be affine; O-D flow rate functions are assumed to be stepwise and all users are assumed to travel along shortest paths.

Chapter 2 concerns the Dynamic Network Loading Problem. We present a variant of the problem in which arc performance functions are affine and path flow rate functions are stepwise. We discuss the advantages of this variant. Furthermore, we present theoretical properties of the problem including proofs of existence and uniqueness of a solution as

well as a proof that the resulting arc travel time functions are piecewise linear. Given these properties, we then present an algorithm for solving the DNLP in continuous time. Chapter 2 concludes with a discussion of the implementation and testing of the DNLP algorithm.

In Chapter 3 we illustrate how the DNLP algorithm can be used to obtain a continuous time solution to the DTA problem. We first discuss the continuous time Dynamic Shortest Paths problem, and formulate its optimality conditions for the one-to-all and all-to-one minimum travel time problems. We then present continuous time DSP algorithms and discuss the implementation of the one-to-all algorithm. Finally we present a DTA algorithm which uses the above algorithms to find a continuous time DTA solution. This algorithm is also implemented and some numerical results are given.

In Chapter 4 we discuss the strengths and weaknesses of the algorithms developed in Chapters 2 and 3 and present some directions for further research.

CHAPTER 2

FORMULATION OF A CONTINUOUS-TIME DYNAMIC NETWORK LOADING MODEL

In Chapter 1 we stated that the Dynamic Network Loading Problem is to determine arc travel times given a network with time-dependent path flows and a model of the relationship between flow and travel time on an arc. This statement of the problem is quite general and any of several models could be used to generate a solution. Having recognized the limitations of discrete-time models, we now seek to develop a model of the DNLP that can be solved in continuous time. To do so, we first make some simplifying assumptions and discuss their validity. We then present a mathematical formulation of the model which defines network parameters and expresses their relationships to one another. We derive various theoretical properties of the model which enable the development of an exact, continuous-time DNLP algorithm. These properties include existence and uniqueness of a solution, as well as functional properties of the unknown variables. Finally, we develop a continuous-time DNLP algorithm and discuss its implementation and testing on a sample network.

2.1 Discussion of Assumptions

In order to develop an exact, continuous-time DNLP algorithm, we make two major simplifying assumptions. First, we assume path flow rate functions to be stepwise. Second, we assume the relationship between an arc's total flow and its travel time (called an arc performance function) to be affine.

Before constructing an algorithm, we consider the strengths and weaknesses of these simplifying assumptions. We first consider the assumption that path flow rate functions are stepwise functions. When selecting inputs to our model, it is likely that we would wish to work from some real world data set to determine origin-destination demands. Recognizing that it is likely that such a data set would be represented as a set of discrete data points, it is advantageous to work with the continuous-time function that naturally follows from discretized data – a stepwise function. Furthermore, it should be recognized that stepwise functions have the additional benefit of easily approximating the shape of any function. Thus, we consider the assumption that path entrance flow rate functions are stepwise to be both realistic and flexible.

With regard to the assumption that arc performance functions are affine, we consider the degree to which the arc performance model represents the real world. Consider the travel time on an arc. In the real world, we know that if the flow on the arc is small enough, a vehicle may traverse the arc without experiencing congestion-related delay. In this case, the vehicle experiences the arc's free flow travel time, τ_{ff}. Suppose, however that a vehicle enters a congested arc. We may wish to model the congestion as a queue that the vehicle must join before exiting the arc. The vehicle's travel time τ, is then equal to τ_{ff}, plus the time spent in queue. For a queue with a deterministic service rate C and X_q enqueued vehicles, we may express this as:

$$\tau = X_q / C + \tau_{ff}.$$

We note that this is equation is an affine function, but that it depends on X_q, not on the total flow on the arc, X. In our model, we wish the arc's travel time to be an affine function of X:

$$\tau = \theta_1 X + \theta_2.$$

If we assume $\theta_2 = \tau_{ff}$ then we note that any amount of flow X, no matter how small, results in a travel time greater than τ_{ff}. This is a weakness of the model, because in real world networks, positive flow rates can indeed experience free flow travel conditions. Furthermore, the assumption that congestion can be adequately represented by a single linear term is overly simplistic. Nevertheless, an affine function is perhaps the most simple way to model the relationship between flow and travel time and possesses theoretical properties which enable an "elegant" solution to the DNLP. In Section 2.4 we will show that by using the linear model above, combined with stepwise path flow rate functions results we can ensure that other network variables maintain functional forms that enable continuous time solution. It is this latter advantage of the linear model that provides the most compelling argument for its use.

Another advantage of this model is its adaptability to future extensions. We recognize that a natural extension of this model is to use piecewise linear arc performance functions to more accurately model the relationship between flow and travel time. This topic is left to further research.

2.2 Notation and Definitions

In order to represent a physical traffic network in a mathematical model we construct a conceptual network, $G = (N, A)$ in which N is the set of nodes and A is the set of arcs. The network consists of a set of paths P, each of which has an origin r and a destination s. We denote by K_{rs} the set of paths connecting r and s. We also denote by K_a the set of paths passing through arc a. For some arc a on path p, we denote by $\bar{a}$ the previous arc on p and by $\vec{a}$ the next arc on p.

Below we present a list of variables used throughout this chapter. Some of the variables express flow rates on each path - we call these path variables; similarly arc variables express flows, flow rates and travel times on each arc. Additionally, we use arc-path

variables to express flows and flow rates on a given arc and path. Finally, we present time variables, which specify the time interval of interest.

Path Variables

$f_p^{rs}(t)$: departure flow rate on path p for O-D pair *(r,s)* at time t;

Arc Variables

$U_a(t)$: cumulative flow that has entered arc a during interval $[0,t]$;

$V_a(t)$: cumulative flow that has exited arc a during interval $[0,t]$;

$X_a(t)$: total flow on arc a at time t;

$D_a(X_a(t))$: travel time function of arc a, where $X_a(t)$ is amount of flow on arc a;

Δ_a : minimum value of $D_a(X_a(t))$ over all times t in the interval $[0,t]$. If $D_a(\cdot)$ is strictly increasing, then Δ_a equals the free flow travel time, $D_a(0)$;

Δ : $\min_{a \in A}\{\Delta_a\}$;

$s_a(t)$: exit time of flow entering arc a at time t. $s_a(t) = t + D_a(X_a(t))$;

Arc-Path Flow Variables

(a, p) : an arc-path pair;

(r, s) : the origin-destination pair of path p;

$u_{ap}^{rs}(t)$: entrance flow rate at time t for arc a and path p;

$v_{ap}^{rs}(t)$: exit flow rate at time t for arc a and path p;

$U_{ap}^{rs}(t)$: cumulative entrance flow at time t for arc a and path p;

$V_{ap}^{rs}(t)$: cumulative exit flow at time t for arc a and path p;

$X_{ap}^{rs}(t)$: total flow on arc a due to path p at time t. The total number of vehicles on an arc at time t is then $X_a(t) = \sum_{p \in K_{rs}} X_{ap}^{rs}(t)$;

Time Variables

t : index for continuous time;

$[0,T]$: O-D traffic demand period; and

$[0,T_\infty]$: analysis period. T_∞ is the lease instant, greater than T after which no flow remains in the network.

FIFO Property

To conclude this section, we present a definition of the First In First Out (FIFO) property which will be referred to in subsequent sections. In this thesis, we consider an arc to verify the FIFO property if:

$$s_a(t_1) < s_a(t_2) \qquad \forall t_1 < t_2 .$$

In the literature this is referred to as the *strict* FIFO property (Chabini and Kachani (1999)). Other definitions of the FIFO property exist, the weakest of which is:

$$s_a(t_1) \le s_a(t_2)\ \forall t_1 \le t_2 \qquad \forall t_1 \le t_2 .$$

Alternately, one may refer to the *strong* FIFO property in which, for $t_1 < t_2$, the difference between $s_a(t_2)$ and $s_a(t_1)$ is at least some positive multiple of the difference between t_2 and t_1. The reader is referred to Chabini and Kachani (1999) for the details of these properties.

2.3 Formulation of the Dynamic Network Loading Problem

In the following section we state important relationships between the network variables that were presented in Section 2.2. These equations are the formulation of the dynamic

network loading model and describe arc dynamics, flow conservation, flow propagation, arc performance and initial/terminal conditions. Each of these equations is an accurate and general representation of how flows move through a transportation network; these equations are common to many dynamic network loading models. In a later section we will choose functions to approximate the relationship between flow and travel time on an arc. By choosing functional forms which ensure various solution properties, we can develop exact, continuous-time solution algorithms.

Arc Dynamics

The arc dynamics equations describe the amount of flow on a given arc as a function of time. They state that the total flow on an arc depends on the arc's entrance and exit flow rates and that the difference between the arc's entrance and exit flow rates is equal to the rate of change of the total flow.

$$\frac{dX_{ap}^{rs}(t)}{dt} = u_{ap}^{rs}(t) - v_{ap}^{rs}(t) \qquad \forall (r,s), \forall p \in K_{rs}, \forall a. \qquad (1)$$

Flow Conservation

Flow conservation equations ensure that along a given path, no flow is lost or gained from one arc to the next on a given path--that is, for two successive arcs along a path, the amount of flow on the path exiting the first arc is equal to the amount of flow entering the second arc at any time t. For the first arc on any path, the entrance flow is equal to the known entrance flow of the path. These conditions are expressed mathematically as:

$$u_{ap}^{rs}(t) = f_{p}^{rs}(t) \qquad \forall (r,s), \forall p \in K_{rs} \qquad (2)$$

for the first arc on any path and as:

$$u_{\bar{a}p}^{rs}(t) = v_{ap}^{rs}(t) \qquad \forall (r,s), \forall p \in K_{rs} \qquad (3)$$

for all other arcs in which arc $\bar{a}$ follows arc a on path p.

Initial and Terminal Conditions

We impose initial conditions such that the network must be empty at time $t = 0$. They are:

$$U_{ap}^{rs}(0) = 0, \quad V_{ap}^{rs}(0) = 0, \quad X_{ap}^{rs}(0) = 0, \quad \forall (r,s), \forall p \in K_{rs}, \forall a \in p. \qquad (4)$$

The above initial conditions are assumed for ease of computation without loss of generality; any set of feasible initial conditions could be used. We likewise denote by $t = T_\infty$ the time at which the network will again be empty.

$$U_{ap}^{rs}(T_\infty) = V_{ap}^{rs}(T_\infty), \quad X_{ap}^{rs}(T_\infty) = 0, \quad T_\infty > 0 \qquad \forall (r,s), \forall p \in K_{rs}, \forall a \in p \qquad (5)$$

Equations (5) are the terminal conditions.

Arc Performance

Arc performance functions relate an arc's travel time to its total flow. Given our discussion of the advantages of a linear arc performance model, we use the equation:

$$D_a(X_a(t)) = \theta_{a1} X_a(t) + \theta_{a2} \qquad \forall a \in A \qquad (6)$$

where θ_{a1} and θ_{a2} are the parameters of the arc performance function of arc a.

Flow Propagation

The following equations relate an arc's outflow to its inflow; they express flow conservation between an arc's origin and destination. In the literature, these equations are referred to as "flow propagation" equations. Though we feel that this terminology does not properly express the meaning of these equations, we retain it for consistency. At some time t, we know that all flows exiting the arc at time t must have entered the arc by time z such that $z + D_a(z) \le t$. If we denote by ω a time at which an entering flow can exit the arc by time t, we can state that the cumulative exit flow on arc a for some path p with origin r and destination s is equal to the integral over all such times ω. Mathematically, we have:

$$V_{ap}^{rs}(t) = \int_{\omega \in \{z \mid z+\tau_a(z) \le t\}} u_{ap}^{rs}(\omega) d\omega \qquad \forall (r,s), \forall p \in K_{rs}, \forall a \in p .$$

This flow propagation equation makes no assumptions about the overtaking behavior of vehicles on the arc. If we assume that the arc operates on a first-in first-out (FIFO) basis, we can state that flows exiting the arc at time t must have entered the arc at time $s_a^{-1}(t)$. Furthermore, we know that the cumulative exit flow at time t is equal to the integral of the entrance flow over the interval $[0, s_a^{-1}(t)]$ (given the initial conditions). The flow propagation equation if FIFO is verified becomes:

$$V_{ap}^{rs}(t) = \int_0^{s_a^{-1}(t)} u_{ap}^{rs}(\omega) d\omega \qquad \forall (r,s), \forall p \in K_{rs}, \forall a \in p . \qquad (7)$$

$$= U_{ap}^{rs}(s_a^{-1}(t)).$$

Summary of the Formulation

We now review the formulation of the model in terms of its known and unknown variables. By initial conditions, $U_{ap}^{rs}(0)$, $V_{ap}^{rs}(0)$, and $X_{ap}^{rs}(0)$ are known values for all arcs, and paths. $f_p^{rs}(t)$ is also a known variable for all paths and times contained in the analysis period. We also know the value of T. The unknown variables are:

- T_∞;
- $U_{ap}^{rs}(t)$, $V_{ap}^{rs}(t)$ and $X_{ap}^{rs}(t)$ $\forall p \in P$, $\forall a \in A$, $t \in [0, T_\infty]$;
- $u_{ap}^{rs}(t)$ and $v_{ap}^{rs}(t)$ $\forall p \in P$, $\forall a \in A$, $\forall t \in [0, T_\infty]$;
- $s_a(t)$ $\forall a$, $\forall t \in [0, T_\infty]$; and
- $U_a(t)$, $V_a(t)$, $X_a(t)$, $D_a(t)$, $u_a(t)$ and $v_a(t)$ $\forall a$, $\forall t \in [0, T_\infty]$.

The above summary lists the known and unknown variables of the dynamic network loading model. In Section 2.4 we rigorously prove that, given stepwise path flow rate functions and affine arc performance functions, a solution to this dynamic network loading models exists and this solution is unique.

2.4 Analysis of the Dynamic Network Loading Problem

The primary focus of the following section is to illustrate that, for a specific class of input functions, solution properties for the DNLP can be established which enable its solution in continuous time. More specifically, by assuming path flow rate functions to be stepwise and arc performance functions to be affine, we will prove that the other network variables take on piecewise forms. Arc and path flow rate functions are proven to be stepwise; cumulative arc and path flow functions and total flow functions are shown to be piecewise linear. Arc exit time functions are also proven to be piecewise linear. In the algorithm, this knowledge of the functional forms will be exploited by calculating problem variables at function "breakpoints". Theorem 1 presents the functional forms of the network variables that result from our choice of input functions.

In order to construct an algorithm to solve the DNLP as described in Section 2.3, we must first establish several mathematical properties of equations (1)-(7). In the following sections, we provide proofs of several lemmas regarding continuous, differentiable functions. These lemmas will assist us in proving the existence and uniqueness of a solution to the DNLP, for single arc and later for a general network. Following the proof of existence of a solution, we prove that for the case of stepwise path flow rate functions and affine arc performance functions, the DNLP possesses a unique solution whose network variables take on a particular functional form. This knowledge leads to an "elegant" construction of a continuous-time dynamic network loading algorithm. Finally, we prove two theorems that assist us in specifying the length of time interval over which to iterate in our algorithm and establish a bound on the number of computations required to solve the DNLP. These theorems ensure that our DNLP algorithm will terminate after a finite number of iterations and provide a basis for discussion of the algorithm's theoretical run-time.

2.4.1 Properties of Inverse Functions

The following two lemmas establish properties of inverse functions. In constructing our algorithm in later sections, we will use $s_a^{-1}(t)$, the inverse of the function $s_a(t)$, thus the following properties will prove useful. The proofs of Lemmas 1 and 2 are borrowed from Chabini and Kachani (1999), with some slight modifications for clarity.

Lemma 1 (Chabini and Kachani, 1999): *Let* $g(\cdot)$ *be a continuously differentiable function on* $[0,T]$. *If for every* $x \in [0,T]$ $g'(x) \neq 0$, *then* $g(\cdot)$ *is invertible on* $[0,T]$, *its inverse function* $g^{-1}(\cdot)$ *is continuously differentiable on* $[\min(g(0),g(T)),\max(g(0),g(T))]$ *and,* $g^{-1\prime}(x) = \dfrac{1}{g'(g^{-1}(x))}$.

Proof of Lemma 1 (Chabini and Kachani, 1999):

Since $g(\cdot)$ is a continuously differentiable function, $g'(\cdot)$ is continuous. Furthermore, since for every $x \in [0,T]$, $g'(x) \neq 0$, we know that $g'(\cdot)$ has a constant sign. Hence, $g(\cdot)$ is either strictly increasing or strictly decreasing. Since every strictly monotonic function is invertible, it follows that $g(\cdot)$ is invertible. Let $g^{-1}(\cdot)$ denote the inverse function of $g(\cdot)$. According to the definition of an inverse function, we have: $g(g^{-1}(x)) = x$. If we differentiate both sides of this equation, we obtain: $g^{-1\prime}(x)g'(g^{-1}(x)) = 1$. Rearranging terms, we have: $g^{-1\prime}(x) = \dfrac{1}{g'(g^{-1}(x))}$. According to the conditions of the lemma, $g'(x) \neq 0$ on $[0,T]$, therefore $g^{-1\prime}(x)$ is defined on $[g(0),g(T)]$ if $g(\cdot)$ is strictly increasing, or on $[g(T),g(0)]$ if $g(\cdot)$ is strictly decreasing. Equivalently, we can say that $g^{-1}(\cdot)$ is continuously differentiable on $[\min(g(0),g(T)),\max(g(0),g(T))]$.

□

Lemma 2 (Chabini and Kachani, 1999): *Let $f(\cdot)$ be a continuous and strictly increasing function on interval $[a,b]$. For $x \in [f(a), f(b)]$, the set $W_x = \{w \in [a,b] \mid f(w) \le x\}$ is the interval $[a, f^{-1}(x)]$.*

Proof of Lemma 2 (Chabini and Kachani, 1999):

Since $f(\cdot)$ is continuous and strictly increasing on $[a,b]$, it then follows that $f(\cdot)$ is invertible and its inverse function $f^{-1}(\cdot)$ is continuous and strictly increasing on $[f(a), f(b)]$.

We first prove that $W_x \subset [a, f^{-1}(x)]$. By assumption, for a given $w \in W_x$, we have: $f(w) \le x$. Since $w \in [a,b]$ and $f(\cdot)$ is increasing, it results that: $f(a) \le f(w)$. Hence, $f(a) \le f(w) \le x$. Since $f^{-1}(\cdot)$ is increasing, we then have: $f^{-1}(f(a)) \le f^{-1}(f(w)) \le f^{-1}(x)$. By the definition of an inverse function, $a \le w \le f^{-1}(x)$. Hence, $w \in [a, f^{-1}(x)]$, and $W_x \subset [a, f^{-1}(x)]$.

We now show that $[a, f^{-1}(x)] \subset W_x$. For a given $w \in [a, f^{-1}(x)]$, we have $w \le f^{-1}(x)$. Since $f(\cdot)$ is increasing, it then follows that $f(w) \le f(f^{-1}(x))$ and $f(w) \le x$. Furthermore, since $x \in [f(a), f(b)]$ and $f^{-1}(\cdot)$ is increasing, it results that $f^{-1}(x) \in [a,b]$. Since $w \in [a, f^{-1}(x)]$, we have $w \in [a,b]$ and $w \in W_x$. Hence, $[a, f^{-1}(x)] \subset W_x$.

Since $W_x \subset [a, f^{-1}(x)]$ and $[a, f^{-1}(x)] \subset W_x$, we have proved that $W_x = [a, f^{-1}(x)]$.

□

2.4.2 Existence and Uniqueness of a Solution to the DNLP: Analysis of One Arc

In order to prove the existence and uniqueness of a solution to the DNLP for a general network, we first prove the existence and uniqueness of a solution for a single arc. In addition to assisting us in understanding the proof for a general network, the results that we establish for the single arc will be used in the analysis of a general network.

This proof relies on several specific network properties, namely that arc performance functions are affine and entrance flow rate functions are stepwise. Proofs of similar results for a more general class of functions can be found in Chabini and Kachani (1999). For the model studied in this chapter the proof below is less complicated. At the heart of the proof is the fact that the solution verifies the FIFO property.

Theorem 1 *For a single arc, if the pair* $(D_a(\cdot), f_p(\cdot))$ *verifies the following properties:*

(i) *the arc performance function* $D_a(\cdot)$ *is affine and* $D'_a(\cdot)$ *is nonnegative; and*

(ii) *the departure flow rate function* $f_a(\cdot)$ *is stepwise, nonnegative and bounded from above by some positive constant M*

then the following properties hold for the DNLP:

(i) *the strong FIFO property is verified; and*

(ii) *the DNLP possesses a solution and this solution is unique.*

Proof of Theorem 1:

We present an induction proof of Theorem 1, which relies on establishing the existence and uniqueness of a solution, as well as the FIFO property for successive time intervals. The induction is over the indices of the time intervals, denoted by i. We define our time intervals such that for some interval $[t_i, t_{i+1})$, t_{i+1} is the first time instant at which a flow entering the arc at time t_i may exit the arc. The sequences of instants t_i is defined by $t_0 = 0$ and $t_{i+1} = t_i + D_a(X_a(t_i))$. Note that $t_{i+1} - t_i \geq D_a(0)$.

The following is the induction hypothesis for some time interval $[t_i, t_{i+1})$.

Induction Hypothesis

For the interval $[t_i, t_{i+1})$, the following properties hold:

(i) $s_a(t)$ *is piecewise linear and continuous over* $[t_i, t_{i+1})$ *and* $s'_a(t) \geq \gamma_i + \alpha_a u_a(t)$ *where* $0 < \gamma_i \leq 1$;

(ii) $V_a(t)$ *is piecewise differentiable over* $[t_i, t_{i+1})$;

(iii) *For every* $t \in [t_i, t_{i+1})$, $v_a(t) \leq \frac{1}{\alpha_a}$; *and*

(iv) *the DNLP has a solution on* $[0, t_{i+1})$ *and this solution is unique.*

Before beginning the induction we establish several properties that hold for all times t in the analysis period. Since we consider a single arc, we know $u_a(\cdot) = f_a(\cdot)$. Thus, $u_a(t)$ has a unique value. Since $u_a(\cdot)$ is stepwise, and therefore piecewise integrable, we can obtain $U_a(\cdot)$ according to:

$$U_a(t) = \int_0^t u(\omega) d\omega .$$

Since $u_a(t)$ is unique and the integral operator is unique, we know $U_a(\cdot)$ to be unique. Furthermore, because $u_a(\cdot)$ is nonnegative, $U_a(\cdot)$ is non-decreasing.

First Base Case: Time interval $[0, t_1)$

Let $t \in [0, t_1)$. We know:

$$\frac{dX_a(t)}{dt} = u_a(t) - v_a(t)$$

and by integration, we have

$$X_a(t) = U_a(t) - V_a(t),$$

for $U_a(0) = 0$, $V_a(0) = 0$ and $X_a(0) = 0$. Additionally since no flow can exit before t_1, $V_a(t) = 0$, so $X_a(t) = U_a(t)$ and

$$X_a(t) = \int_0^t u(\omega)d\omega .$$

Thus, like $U_a(\cdot)$, $X_a(\cdot)$ is continuous and unique on $[0,t_1)$. We now consider $s_a(t)$ which is defined by:

$$s_a(t) = t + D_a(X_a(t)).$$

Since $X_a(\cdot)$ is continuous and unique and $D_a(\cdot)$ is affine, $D_a(X_a(t))$ is continuous and unique, and therefore $s_a(t)$ is continuous and unique. We can also establish that because t is strictly increasing, and $X_a(t)$ is non-decreasing, $D_a(X_a(t))$ is non-decreasing and $s_a(t)$ is strictly increasing. Thus, on the interval $[0,t_1)$, the DNLP has a solution and this solution is unique. Because $s_a(t)$ is strictly increasing, we can also conclude that the solution is FIFO on interval $[0,t_1)$.

In later parts of this proof we will use the properties of $s_a(t)$ to find an upper bound on the exit flow rate function. In particular, we wish to bound the slope of $s_a(t)$. We observe that:

$$s'_a(t) = 1 + D'_a(X_a(t))\frac{dX_a(t)}{dt} = 1 + u_a(t)D'_a(X_a(t)).$$

Since $D_a(\cdot)$ is affine, $D'_a(\cdot)$ is known to have a constant value, which we will denote by α_a. Additionally, we replace the value *1* with the variable γ_0 to yield:

$$s'_a(t) = \gamma_0 + \alpha_a u_a(t).$$

We now construct a second base case. This base case differs from the first because we now examine an interval with positive exit flow. After constructing this second, "more general" base case, we complete the proof with an induction step. We again construct an interval such that vehicles entering at the beginning of the interval will not exit the arc before the end of the interval.

Second Base Case: Time interval $[t_1, t_2)$.

In the first base case, we proved that $s_a(t)$ is increasing on the interval $[0, t_1)$. Since $s_a(0) = t_1$ and $s_a(t_1) = t_2$, then $s_a(t)$ is invertible. Using Lemma 1, we know $s_a^{-1}(t)$ on $[t_1, t_2)$, $s_a^{-1}(t) \in [0, t_1)$ and $(s_a^{-1})'(t) = \frac{1}{s'_a(s_a^{-1}(t))}$. As $s_a^{-1}(t)$ corresponds to the entry time for a flow which exits the link at time t, and $s_a^{-1}(t) \in [0, t_1)$, vehicles that exit arc a before a time t must have entered it before $s_a^{-1}(t)$.

As $V_a(0) = 0$, we then have:

$$V_a(t) = \int_0^{s_a^{-1}(t)} u_a(\omega) d\omega \qquad \forall t \in [t_1, t_2).$$

Since $u_a(\cdot)$ is stepwise and unique on $[0, t_1)$, $V_a(\cdot)$ is piecewise linear and unique (and therefore piecewise differentiable). Additionally, since $u_a(\cdot)$ is nonnegative, $V_a(\cdot)$ must be non-decreasing. We can then compute $v_a(\cdot)$ by differentiating $V_a(\cdot)$. We obtain $v_a(t) = ((s_a^{-1})'(t)) \cdot u_a(s_a^{-1}(t))$, which must be nonnegative because $u_a(\cdot)$ is nonnegative and $s'_a(\cdot)$ is nonnegative on the interval $[0, t_1)$. Given that $(s_a^{-1})'(t) = \frac{1}{s'_a(s_a^{-1}(t))}$, we obtain:

$$v_a(t) = \frac{u_a(s_a^{-1}(t))}{s'_a(s_a^{-1}(t))}.$$

From the first base case, we know that $u_a(s_a^{-1}(t))$ is unique and $s_a(s_a^{-1}(t))$ is unique. $s_a'(s_a^{-1}(t))$ must then also be unique because the derivative operator is unique. Thus, $v_a(\cdot)$ must also be unique. Furthermore, since $s_a(s_a^{-1}(t))$ is strictly increasing, $s'_a(s_a^{-1}(t))$ must be positive, thereby ensuring the positivity and existence of $v_a(\cdot)$. Additionally, from the first base case we proved that :

$$s'_a(t) \geq \gamma_0 + \alpha_a u_a(t) \quad \text{for } t \in [0, t_1)$$

so we have:

$$v_a(t) \le \frac{u_a(s_a^{-1}(t))}{\gamma_0 + \alpha_a u_a(s_a^{-1}(t))}.$$

We also note that given the bound:

$$v_a(t) \le \frac{u_a(s_a^{-1}(t))}{\gamma_i + \alpha_a u_a(s_a^{-1}(t))}$$

used in the proof of Theorem 1, we can obtain the bound:

$$v_a(t) \le \frac{1}{\alpha_a}.$$

We create this bound on $v_a(\cdot)$, because as we shall see later in the proof, the exit time function $s_a(t)$ depends on the difference between the entrance flow rate and the exit flow rate. By bounding the value of the exit flow rate, we can ensure that $s_a(t)$ is strictly increasing and thus that the FIFO property is preserved.

Since $U_a(\cdot)$ and $V_a(\cdot)$ are unique then $X_a(\cdot)$ must also be unique according to:

$$X_a(t) = U_a(t) - V_a(t).$$

Furthermore, since $U_a(\cdot)$ and $V_a(\cdot)$ are piecewise linear, then $X_a(\cdot)$ must be piecewise linear and $D_a(X_a(t))$ must be piecewise linear and unique. Thus, $s_a(t)$ must also be piecewise linear. We have also shown that a solution to the DNLP exists on this interval and this solution is unique.

We now establish that $s_a(t)$ is strictly increasing on the interval . First, we recognize that

$$s_a'(t) = 1 + \alpha_a \frac{dX(t)}{dt}$$

$$= 1 + \alpha_a (u_a(t) - v_a(t)).$$

Since $v_a(t) \le \frac{u_a(s_a^{-1}(t))}{\gamma_0 + \alpha_a u_a(s_a^{-1}(t))}$, we have:

$$s_a'(t) \ge 1 + \alpha_a u_a(t) - \alpha_a \frac{u_a(s^{-1}(t))}{\gamma_0 + \alpha_a u_a(s_a^{-1}(t))}$$

$$\ge \alpha_a u_a(t) + \frac{\gamma_0}{\gamma_0 + \alpha_a u_a(s_a^{-1}(t))}.$$

We recognize that the quantity $u_a(t)$ is bounded from below by zero (because entrance flow rates are nonnegative) and from above by some positive constant M (because by assumption we do not permit flows of infinite flow rate), we have:

$$s_a'(t) \geq \alpha_a u_a(t) + \frac{\gamma_0}{\gamma_0 + \alpha_a M}.$$

We know that α_a and M are nonnegative, so we recognize that $0 \leq \frac{\gamma_0}{\gamma_0 + \alpha_a M} \leq \gamma_0 \leq 1$.

We denote by γ_1 the quantity $\frac{\gamma_0}{\gamma_0 + \alpha_a M}$ and we have:

$$s_a'(t) \geq \gamma_1 + \alpha_a u_a(t).$$

We know that α_a, $u_a(t)$ and γ_1 are nonnegative, so $s_a'(t)$ must be nonnegative also. Consequently, (strict) FIFO is verified on this interval.

To complete the proof, we now prove the existence and uniqueness of a solution, as well as the FIFO property for an interval $[t_{i+1}, t_{i+2})$ where $i \geq 1$.

Induction Step: Time interval $[t_{i+1}, t_{i+2})$

In this interval we proceed in a manner similar to the second base case. By the induction hypothesis, we know that $s_a(\cdot)$ is piecewise differentiable and continuous over $[t_i, t_{i+1})$ and $s'_a(\cdot) > 0$. We recall from Lemma 2 that $s'_a(\cdot)$ must then be continuous over $[s_a(t_i), s_a(t_{i+1})) = [t_{i+1}, t_{i+2})$ as well as differentiable and unique.

We also have:

$$V_a(t) = \int_0^{s_a^{-1}(t)} u(\omega) d\omega \qquad \forall t \in [t_{i+1}, t_{i+2}),$$

and since $s_a^{-1}(\cdot)$ is differentiable and unique on $[t_{i+1}, t_{i+2})$, and the integral operator is unique, $V_a(\cdot)$ must also be differentiable and unique. We can then obtain $v_a(t) = ((s_a^{-1})'(t)) \cdot u_a(s_a^{-1}(t))$, and:

$$v_a(t) = \frac{u_a(s_a^{-1}(t))}{s'_a(s_a^{-1}(t))},$$

as we did in the second base case. We note that $U_a(\cdot)$, $V_a(\cdot)$ and $X_a(\cdot)$ are unique and piecewise linear. Thus, a solution to the DNLP exists and this solution is unique.

From the induction hypothesis, on interval $[t_i, t_{i+1})$ we have:

$$s'_a(t) \geq \gamma_i + \alpha_a u_a(t)$$

Using flow propagation (equation (6)) we then have:

$$v_a(t) \leq \frac{u_a(s_a^{-1}(t))}{\gamma_i + \alpha_a u_a(s_a^{-1}(t))}.$$

This gives:

$$\begin{aligned} s_a'(t) &= 1 + \alpha_a \frac{dX(t)}{dt} \\ &= 1 + \alpha_a(u_a(t) - v_a(t)). \\ &\geq 1 + \alpha_a u_a(t) - \alpha_a \frac{u_a(s^{-1}(t))}{\gamma_i + \alpha_a u_a(s_a^{-1}(t))} \\ &\geq \alpha_a u_a(t) + \frac{\gamma_i}{\gamma_i + \alpha_a u_a(s_a^{-1}(t))}. \end{aligned}$$

We note that $u_a(\cdot)$ is bounded from above by M, so we have:

$$s'_a(t) = \alpha_a u_a(t) + \frac{\gamma_i}{\gamma_i + \alpha_a M}.$$

We denote by γ_{i+1} the quantity $\frac{\gamma_i}{\gamma_i + \alpha_a M}$

and note that $0 < \gamma_{i+1} \leq \gamma_i \leq 1$. This gives:

$$s'_a(t) \geq \alpha_a u_a(t) + \gamma_{i+1}.$$

Furthermore, since α_a and $u_a(t)$ are nonnegative, $s'_a(t) > 0$. Thus, $s_a(t)$ is strictly increasing on $[t_{i+1}, t_{i+2})$. We have now proven that the exit time function is strictly increasing over the analysis period, therefore the FIFO property is verified for all

intervals. The induction stops as the last flow that would enter at $t = T$ will leave during or before the interval indexed $\left\lceil \frac{T}{\Delta} \right\rceil$ where $\Delta = Min(\Delta_a)\ \forall a \in A$.

□

2.4.3 Existence and Uniqueness of a Solution to the DNLP for General Networks

Having established the existence and uniqueness of a FIFO solution to the DNLP for a single arc, we now seek to establish this result for a general network. In order to do so, we first establish an upper bound on the exit flow rate function as shown in the proof below.

This proof is an induction proof over time intervals of length Δ, where $\Delta = \underset{a \in A}{Min}(\Delta_a)$. The first such interval is $[0, \Delta)$ and the ith interval is denoted by $[i\Delta, (i+1)\Delta)$.

Theorem 2 *For a network if for every path p and arc a, the following conditions are verified, then:*

(i) *the arc performance function $D_a(\cdot)$ is affine and $D'_a(\cdot)$ is nonnegative; and*

(ii) *the departure flow rate function $f_p(\cdot)$ is stepwise, nonnegative and bounded from above by some positive constant M_p.*

then the following properties hold for the DNLP:

(i) *the DNLP possesses a solution and that solution is unique;*

(ii) *the FIFO property holds; and*

(iii) *$\forall a \in A$ on $[0, i\Delta)$, $V_a(\cdot)$ is piecewise linear and $v_a(\cdot)$ is stepwise and bounded from above by $\frac{1}{\alpha_a}$.*

Proof of Theorem 2:

In order to bound the exit flow rate function of each arc in the network, we adopt the following notation.

$$M^* = Max(\underset{p \in P}{Max}(M_p), \underset{a \in A}{Max}(\frac{1}{\alpha_a}))$$

$$M^{**} = M^* |P| |A|$$

where $|P|$ is the number of paths and $|A|$ is the number of arcs.

Induction Hypothesis

For the interval $[0, i\Delta)$, the following properties hold for all path p and arcs a:

(i) $s_a(t)$ *is piecewise linear and continuous over* $[0, i\Delta)$ *and* $s'_a(t) \geq \gamma_i + \alpha_a u_a(t)$ *where* $0 < \gamma_i \leq 1$;

(ii) $V_a(t)$ *is piecewise differentiable over* $[(i-1)\Delta, i\Delta)$;

(iii) *For every,* $[0,(i+1)\Delta)$ $v_a(t) \leq \frac{1}{\alpha_a}$; *and*

(iv) *the DNLP has a solution on* $[0, i\Delta)$ *and this solution is unique.*

Base Case: $t \in [0, \Delta)$

In the base case, we know that for every path

$$u_{ap}^{rs}(t) = f_p(t)$$

for all arcs that are the first arc on a path and $u_{ap}^{rs}(t) = v_{ap}^{rs}(t) = 0$ for all other arcs. Since all $f_p(t)$ are bounded from above by M^*, we know $u_{ap}^{rs}(t) \leq M^*$. Using:

$$u_a(t) = \sum_{p \in P} \delta_{ap} u_{ap}^{rs}(t)$$

(where δ_{ap} equals 1 if arc a is on path p and 0 otherwise), we can bound arc flow rate functions for all arcs in the network according to $u_a(t) \leq M^* |P| \leq M^{**}$. We also note that since $v_a(t) = 0$ on this interval, condition *(iii)* of the induction hypothesis is verified.

We have also now verified the conditions of Theorem 1 and thus, the results of Theorem 1 hold on the interval $[0,\Delta)\ \forall a \in A$.

Given that $U_a(0) = 0$, $V_a(0) = 0$ and $X_a(0) = 0$, we have:

$$X_a(t) = U_a(t) - V_a(t) \qquad \forall a \in A$$

Since no flow can exit any arc before Δ, $V_a(t) = 0$, so $X_a(t) = U_a(t)$ and

$$X_a(t) = \int_0^t u_a(\omega)d\omega. \qquad \forall a \in A$$

Thus, $X_a(\cdot)$ is continuous and unique on $[0,\Delta)\ \forall a \in A$. We now consider $s_a(t)$ which is defined by:

$$s_a(t) = t + D_a(X_a(t)) \qquad \forall a \in A.$$

Since $X_a(\cdot)$ is continuous and unique and $D_a(\cdot)$ is affine $\forall a \in A$, $D_a(X_a(t))$ is continuous and unique, and therefore $s_a(t)$ is continuous and unique $\forall a \in A$. We can also establish that because t is strictly increasing, and $X_a(t)$ is non-decreasing, $D_a(X_a(t))$ is non-decreasing and $s_a(t)$ is strictly increasing $\forall a \in A$ on the interval $[0,\Delta)$. Given this, we also know that $s_a^{-1}(t)$ exists on the interval $[s_a(0), s_a(\Delta))$. Since

$$s_a(0) = 0 + D_a(X_s(0)) \geq \Delta$$

and

$$s_a(\Delta) = \Delta + D_a(X_s(\Delta)) \geq 2\Delta$$

we have $[0,2\Delta) \subset [0, s_a(\Delta))$. If $t \in [0, s_a(0))$ then $v_a(t) = 0 \leq M^{**}$. Otherwise, if $t \in [s_a(0), s_a(\Delta))$ then

$$v_a(t) = \sum_{p \in P} \delta_{ap} v_{ap}^{rv}(t) \leq \frac{1}{\alpha_a}\delta_{ap} \leq M^{**}.$$

Thus, for $t \in [0, 2\Delta)$, $v_a(t) \leq M^{**}$ and the conditions of the induction hypothesis are verified.

Induction Step: $t \in [0,(i+1)\Delta)$

We first note that if $t \in [0, i\Delta)$ then $u_{ap}^{rs}(t) \leq M^*$ according to the induction hypothesis. If $t \in [i\Delta, (i+1)\Delta)$ then we have:

$$u_{ap}^{rs}(t) = f_p(t)$$

for all arcs that are the first arc on a path and $u_{ap}^{rs}(t) = v_{\bar{a}p}^{rs}(t)$ for all other arcs. By assumption, we know $f_p(t) \leq M^*$ for all paths. We also know that:

$$v_{\bar{a}p}^{rs}(t) = u_{\bar{a}p}^{rs}(s_a^{-1}(t)).$$

Since $s_a^{-1}(t) \leq i\Delta$, then $u_{\bar{a}p}^{rs}(s_a^{-1}(t))$ exists and is bounded by:

$$u_{\bar{a}p}^{rs}(s_a^{-1}(t)) \leq \frac{1}{\alpha_a} \leq M^*.$$

Thus, we have $v_{\bar{a}p}^{rs}(t) \leq M^*$, $u_{ap}^{rs}(t) \leq M^*$ and $u_a(t) \leq M^{**}$. Thus, the conditions of Theorem 1 are verified on $[0, (i+1)\Delta)$ and according to Theorem 1, a solution to the DNLP exists for all arcs on this interval and this solution is unique. Furthermore, the FIFO property is verified for all arcs on the network. Theorem 1 also proves that condition *(i)* of the induction hypothesis is verified on this interval. To prove that condition *(iii)* is verified on this interval, we note that:

$$[0, (i+2)\Delta) \subset [0, s_a((i+1)\Delta)).$$

Thus if $t \in [0, s_a(i\Delta))$ then $v_a(t) \leq \frac{1}{\alpha_a}$ according to the results of Theorem 1. Otherwise, if $t \in [s_a(i\Delta), s_a((i+1)\Delta))$ then

$$v_a(t) = \sum_{p \in P} \delta_{ap} v_{ap}^{rs}(t) \leq \frac{1}{\alpha_a} \delta_{ap} \leq M^{**}.$$

Thus for all $t \in [0, (i+2)\Delta)$, $v_a(t) \leq \frac{1}{\alpha_a}$ and the conditions of the induction hypothesis are verified for all arcs.

To determine the time instant at which the induction stops, we consider some time $s_p(t)$ which is the exit time a flow on path p. Since $\forall p, \forall t \in [0, T]$, $s_p(t)$ is increasing, we have:

$$s_p(T) \geq Max(s_p(t),\ t \in [0,T])$$

and $T_\infty = \underset{p}{Max}(s_p(T))$. Thus, the induction stops at some index $i = \left\lceil \frac{T_\infty}{\Delta} \right\rceil$.

□

2.4.4 Solution Properties of the DNLP for a Class of Input Functions

In earlier sections, we have stated that if we construct the DNLP such that the entrance flow rate functions are stepwise and the arc performance functions are affine, then the solution will maintain stepwise exit flow rate functions and piecewise linear cumulative flow functions. In the following section, we provide a proof of this property.

Theorem 3 *For a DNLP with stepwise entrance flow rate functions* $f_a(\cdot)$, *and affine arc performance functions* $D_a(\cdot)$, *the solution has the following properties:*

(i) *the cumulative entrance flow functions* $U_a(\cdot)$, *are piecewise linear;*

(ii) *the exit flow rate functions* $v_a(\cdot)$, *are stepwise;*

(iii) *the cumulative exit flow functions* $V_a(\cdot)$, *are piecewise linear;*

(iv) *the total flow functions* $X_a(\cdot)$, *are piecewise linear; and*

(v) *the exit time functions,* $s_a(\cdot)$ *are piecewise linear.*

Proof of Theorem 3:

As in the proofs of the previous section, we present an induction proof of the indices of time intervals denoted by i. Our induction step is that the conditions of Theorem 3 hold for the interval $[0, i\Delta)$. We first present the base case and prove that the conditions of the Theorem hold for the interval $[0, \Delta)$.

Base Case: $t \in [0, \Delta)$

For any arc in the network we know:

$$u_a(t) = \sum_{p \in P} \delta_{ap} u_{ap}^{rs}(t)$$

where δ_{ap} equals 1 if arc a is on path p and 0 otherwise. Thus, $u_a(\cdot)$ is a step function if all of the $u_{ap}(\cdot)$ are also step functions. Indeed, if a is the first arc on path p $u_{ap}^{rs}(\cdot) = f_p(\cdot)$, which is a step function. If it is not, we have $u_{ap}^{rs}(\cdot) = 0$ since by our definition of time intervals, we know that no flow can exit any arc in the network during interval $[0, \Delta)$. Thus we know $v_{\bar{a}p}^{rs}(\cdot) = 0 \quad \forall a \in A, \forall p \in P$. We can then conclude that $u_a(\cdot)$ is a step function on interval $[0, \Delta)$.

Because $u_a(\cdot)$ is stepwise, it is piecewise integrable and:

$$U_a(t) = \int_0^t u(\omega) d\omega$$

and $U_a(\cdot)$ is thus piecewise linear. Finally, $v_{\bar{a}p}^{rs}(\cdot) = 0$ and $v_{\bar{a}}(t) = \sum_{p \in P} \delta_{\bar{a}p} v_{\bar{a}p}^{rs}(t)$ so $v_a(\cdot) = 0$ for all a, hence we know that the cumulative exit flow function $V_a(\cdot)$ equals zero.

Since $X_a(t) = U_a(t) - V_a(t)$, $X_a(t)$ must also be piecewise linear on interval $[0, \Delta)$. $s_a(t)$ is defined according to:

$$s_a(t) = t + D_a(X_a(t)).$$

Since we know that t is linear, $X_a(t)$ is piecewise linear and $D_a(\cdot)$ is an affine function of $X_a(t)$, then $s_a(t)$ is also piecewise linear on interval $[0, \Delta)$.

Induction Step: $t \in [0, (i+1)\Delta)$

According to the induction hypothesis, $u_a(t)$ is stepwise on the interval $[0, i\Delta)$. Since $s_a(t)$ is increasing on $[0, i\Delta)$ for all arcs in the network, hence $t \in [0, (i+1)\Delta)$,

$s_a^{-1}(t) \le t - \Delta \le i\Delta$. Thus, any flow which exits an arc during the interval $[0,(i+1)\Delta)$ must have entered the arc in the interval $[0,i\Delta)$. We can calculate $V_a(\cdot)$ according to the equation

$$V_a(t) = \int_0^{s_a^{-1}(t)} u_a(\omega)d\omega$$

and conclude that since $u_a(\cdot)$ is stepwise on $[0,i\Delta)$, $V_a(\cdot)$ is piecewise linear on $[0,(i+1)\Delta)$. Hence $v_a(\cdot)$ is stepwise on $[0,(i+1)\Delta)$.

For any arc in the network we know:

$$u_a(t) = \sum_{p \in P} \delta_{ap} u_{ap}^{rs}(t).$$

Hence $u_a(\cdot)$ is a step function on $[0,(i+1)\Delta)$ if all of the $u_{ap}(\cdot)$ are also step functions on $[i\Delta,(i+1)\Delta)$. If a is the first arc on path p, $u_{ap}^{rs}(\cdot) = f_p(\cdot)$ which is a step function by assumption. If it is not, we have $u_{ap}^{rs}(\cdot) = v_{\bar{a}p}^{rs}(t)$. Since the network is FIFO, any two paths entering an arc at time $s_a^{-1}(t)$ must both exit the arc at time t. Thus, it must also be true that the fraction of an arc's entrance flow rate that is on path p at time $s_a^{-1}(t)$ is equal to the fraction of the arc's exit flow rate on p at time t. We have:

$$v_{\bar{a}p}^{rs}(t) = \frac{u_{\bar{a}p}^{rs}(s_{\bar{a}}^{-1}(t))}{u_{\bar{a}}(s_{\bar{a}}^{-1}(t))} \times v_{\bar{a}}(t).$$

Here we note that if $u_{\bar{a}}(s_{\bar{a}}^{-1}(t))$ is equal to zero at any instant, then $v_{\bar{a}}(t)$ must also be equal to zero and $v_{\bar{a}p}^{rs}(t)$ is then equal to zero. It can be shown that division or multiplication of stepwise functions yields stepwise functions, thus we have proven that $u_{ap}^{rs}(\cdot)$ and $u_a(\cdot)$ are stepwise on interval $[0,i\Delta)$ and that $v_a(\cdot)$ is stepwise on interval $[0,(i+1)\Delta)$, thus $v_{\bar{a}p}^{rs}(\cdot)$ must also be stepwise on $[0,(i+1)\Delta)$. Since both $f_p(\cdot)$ and $v_{\bar{a}p}^{rs}(\cdot)$ are stepwise on this interval, $u_a(\cdot)$ must also be stepwise on $[0,(i+1)\Delta)$.

Integration of $u_a(\cdot)$ to obtain $U_a(\cdot)$ using

$$U_a(t) = \int_0^t u_a(\omega)d\omega$$

yields a piecewise linear function. Thus $U_a(\cdot)$ is piecewise linear on $[0,(i+1)\Delta)$. Since $X_a(t) = U_a(t) - V_a(t)$, $X_a(t)$ must also be piecewise linear on interval $[0,\Delta)$. $s_a(t)$ is defined according to:

$$s_a(t) = t + D_a(X_a(t)).$$

Since we know that t is linear, $X_a(t)$ is piecewise linear and $D_a(\cdot)$ is an affine function of $X_a(t)$, we know that $s_a(t)$ is also piecewise linear on interval $[0,(i+1)\Delta)$. Thus, we have verified the conditions of the induction hypothesis.

We also note that since $U_a(\cdot)$ and $V_a(\cdot)$ are obtained by integrating stepwise functions, then they are continuous on the time interval $[0,(i+1)\Delta)$. Since we obtain $X_a(t)$ using:

$$X_a(t) = U_a(t) - V_a(t)$$

$X_a(t)$ must also be a continuous piecewise linear function. Thus, given

$$s_a(t) = t + D_a(X_a(t))$$

and since t is linear and $D_a(\cdot)$ is affine, $s_a(t)$ is continuous and piecewise linear.

□

As we will see in later sections concerning the construction of a network-loading algorithm, the above properties are essential in the construction of an exact, continuous-time solution. Each of the functional forms described above is characterized by having "breakpoints" between adjacent pieces of a piecewise-linear or stepwise function. Preserving these functional forms will permit us to construct a DNLP solution by looping over the breakpoints of the relevant functions. To determine the location and number of such breakpoints, we provide two proofs. Theorem 4 establishes how breakpoints in the exit flow rate functions are formed. Theorem 5 provides a bound on the number of breakpoints.

Theorem 4 *For every $a \in A$ with exit flow rate function $v_a(\cdot)$, each breakpoint of $v_a(\cdot)$ is caused by either:*

(i) a breakpoint in the function $u_a(\cdot)$ at $s_a^{-1}(t)$; or

(ii) a breakpoint in the function $v_a(\cdot)$ at $s_a^{-1}(t)$.

Proof of Theorem 4:

From Theorem 3, we know that $v_a(\cdot)$ is a step function, and thus contains breakpoints. From Theorem 3, we also know that $u_a(\cdot)$ is stepwise. We also have:

$$s'_a(t) = 1 + (u_a(t) - v_a(t))D'_a(X_a(t))$$

where $s'_a(t)$ is the slope of $s_a(t)$ to the right of t. Since $D'_a(X_a(t))$ is a constant, we know that $s'_a(t)$ must be a step function with breakpoints occurring at every time t at which either $u_a(\cdot)$ or $v_a(\cdot)$ contains a breakpoint. Thus, $s_a(t)$ must be a piecewise linear function such that for every time t for which $s_a(t)$ contains a breakpoint, $s'_a(t)$ must also contain a breakpoint. From Theorems 1 and 2, we have:

$$v_a(t) = \frac{u_a(s_a^{-1}(t))}{s'_a(s_a^{-1}(t))}.$$

Thus, every breakpoint in $V_a(\cdot)$ corresponds to a breakpoint in $s'_a(t)$ or $u_a(\cdot)$, and therefore to breakpoint in $u_a(\cdot)$ or $v_a(\cdot)$.

□

In later sections we will discuss the efficiency of the continuous-time DNLP algorithm. For this purpose, we will be interested in the maximum number of breakpoints in the arc variables. The proof below provides a bound on the number of breakpoints in an arc exit flow rate function, given the number of breakpoints in its entrance flow rate function.

Theorem 5 *For every* $a \in A$ *with entrance flow rate function* $u_a(\cdot)$ *over interval* $t \in [0,T]$ *the number of breakpoints in the exit flow rate function* $v_a(\cdot)$ *is bounded from above by* $\left\lceil \frac{T}{\Delta} \right\rceil + N_u$ *where* N_u *equals the number of breakpoints in* $u_a(\cdot)$ *and* $\Delta = \min(D_a(t) \mid t \in [0,T])$.

Proof of Theorem 5:

From Theorem 4, we know that a breakpoint in the exit flow rate function $v_a(\cdot)$ at time t' may be caused by either:

(i) a breakpoint in the function $u_a(\cdot)$ at a previous time t; or

(ii) a breakpoint in the function $v_a(\cdot)$ at a previous time t such that $t \in [0,T]$.

Since any flow which enters at some time t cannot exit before time $t + \Delta$, we conclude that any two breakpoints on $v_a(\cdot)$, t and t' must be separated by at least Δ. Thus, on the interval $t \in [0,T]$ there are at most $\left\lceil \frac{T}{\Delta} \right\rceil$ non-differentiable points of $v_a(t)$. Additionally, we know *a priori* the value of N_u, thus, we can bound the number of breakpoints on $v_a(\cdot)$ by $\left\lceil \frac{T}{\Delta} \right\rceil + N_u$ for the interval $t \in [0,T_\infty]$.

□

2.5 Construction of a Continuous-Time Dynamic Network-Loading Algorithm

In the following section we develop a continuous-time algorithm for the Dynamic Network Loading problem, assuming stepwise path flows and affine arc performance functions. We first present an algorithm for a single arc, then extend this algorithm to general networks.

2.5.1 A DNLP Algorithm for a Single Arc

Taking advantage of the theoretical developments in the above proofs, we now wish to construct an algorithm that moves chronologically, computing relevant network parameters. We first consider a single arc. Consider some time interval $[t_{curr}, t_{fut})$ in which $u_a(\cdot)$ and $v_a(\cdot)$ are constant. According to the arc dynamics equation, we know that on this interval, $X_a(\cdot)$ is linear and thus that $s_a(t)$ is linear. We can compute $v_a(\cdot)$ on the interval $[s_a(t_{curr}), s_a(t_{fut}))$ according to the equation:

$$v_a(t) = \frac{u_a(s_a^{-1}(t))}{s'_a(s_a^{-1}(t))}$$

which we obtained in the proof of Theorem 1. We note that since $u_a(\cdot)$ is constant on $[t_{curr}, t_{fut})$ and $s_a(t)$ is linear on $[t_{curr}, t_{fut})$, then $v_a(\cdot)$ is constant on $[s_a(t_{curr}), s_a(t_{fut}))$. Since all flow that enters the link between t_{curr} and t_{fut} will exit the link between $s_a(t_{curr})$ and $s_a(t_{fut})$. we have:

$$\int_{t_{curr}}^{t_{fut}} u_a(y)dy = \int_{s_a(t_{curr})}^{s_a(t_{fut})} v_a(y)dy .$$

Since $u_a(\cdot)$ and $v_a(\cdot)$ are constant (and assuming stepwise functions are specified as $\forall x \in [a,b) \;\; f(x) = \lim_{x \to b} f(x)$ we can rewrite this as:

$$v_a(s_a(t_{curr})) = \frac{u_a(t_{curr})(t_{fut} - t_{curr})}{s_a(t_{fut}) - s_a(t_{curr})} \qquad (8)$$

An illustration of this computation is shown in Figure 2. A_1 and A_2 are equal to the integrals $\int_{t_{curr}}^{t_{fut}} u_a(y)dy$ and $\int_{s_a(t_{curr})}^{s_a(t_{fut})} v_a(y)dy$, respectively and are equal to each other.

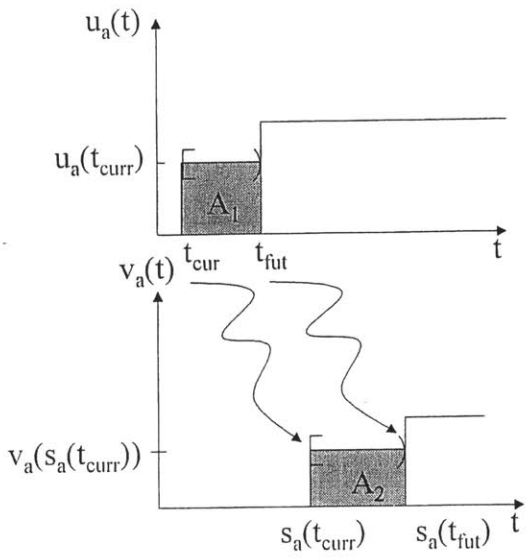


Figure 2: Graphical Interpretation of the DNLP Solution

We note that for any interval in which $u_a(t)$ and $v_a(t)$ are constant, we can compute the value of $v_a(s_a(t))$ using the simple geometric relationship in equation (8). Thus, we construct an algorithm that loops over intervals with this property. Each interval is defined by t_{curr} and t_{fut}, the beginning and end of the interval. At each loop, the algorithm determines the exit time of a flow entering the arc at the beginning of the interval, $s_a(t_{curr})$. It also determines the exit time of a flow entering the arc at the end of the interval, $s_a(t_{fut})$. The algorithm moves forward chronologically, according to the values of t_{curr}. At some later time, the algorithm "arrives" at some interval which begins at $s_a(t_{curr})$ and computes the value of $v_a(s_a(t_{curr}))$.

In Figure 2 time interval $[t_{curr}, t_{fut})$ was defined by two successive breakpoints of the function $u_a(t)$. We note, however that for two successive breakpoints of $u_a(t)$, between which $u_a(\cdot)$ is constant, it is not necessarily true that $v_a(\cdot)$ is constant. Consider Figure

3. Here we note that a flow entering the arc at time t_{curr} will exit the arc at time $s_a(t_{curr})$. At time $s_a(t_{curr})$, the exit flow rate function will then contain a breakpoint. According to the arc dynamics equation, this breakpoint will cause a breakpoint in the function $X_a(\cdot)$, and therefore will also cause a breakpoint in $s_a(t)$. Here, the shortest interval beginning at t_{curr} such that $u_a(t)$ and $v_a(t)$ are constant is $[t_{curr}, s_a(t_{curr}))$.

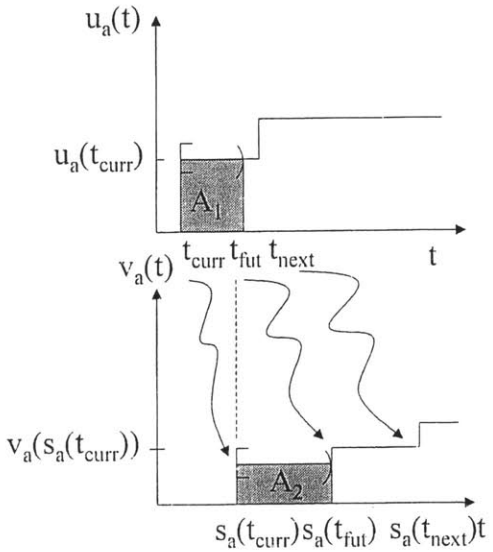


Figure 3: Graphical Interpretation of the DNLP Solution

The above discussion has several important implications for our algorithm. Firstly, we note that it is not sufficient to simply iterate over the intervals defined by the breakpoints of $u_a(t)$; this may not yield a correct solution. Instead, we must iterate over intervals such that both $u_a(t)$ and $v_a(t)$ are constant. A second implication regards the number of breakpoints in our exit flow rate function, $v_a(t)$. We note that a breakpoint in the exit flow rate function of an arc can be caused by either (a) an earlier breakpoint in the corresponding entrance flow rate function or (b) an earlier breakpoint in the exit flow rate function itself. Examination of the arc dynamics equation shows this to be true. Thus,

for any time t at which our algorithm detects a breakpoint in either $u_a(t)$ or $v_a(t)$, there may be a future breakpoint in $v_a(\cdot)$ at time $s_a(t)$.

We propose an algorithm which moves forward chronologically, computing arc variables at each step. Having established that we can completely specify the solution by specifying arc parameters at function breakpoints, we then loop over these breakpoints in an event-based manner. At the end of each loop, we select the time of the next loop based on the first breakpoint of either an entrance or exit flow rate function.

Our algorithm begins by initializing arc variables according to the initial conditions. It also defines two sets, I and E which contain all known breakpoints in the entrance and exit flow rate functions. Set I is initialized with the set of breakpoints contained in the path flow rate function; set E is initially empty. To begin the main loop, we select the breakpoint of I which occurs at the least value of t. For each value of t_{curr} we find the next breakpoint in either the entrance or exit flow rate functions, t_{fut}. These two breakpoints define an interval for which $u_a(t)$ and $v_a(t)$ are constant. We then compute $v_a(t)$ for this interval, based on the value of $u_a(\cdot)$ at $s_a^{-1}(t_{curr})$. By integration, we can then determine the values of $U_a(t)$, $V_a(t)$ and $X_a(t)$. We then calculate the time at which a flow entering the link at time t_{curr} will exit. At this time, $s_a(t_{curr})$ there may be a breakpoint in the function $v_a(\cdot)$, thus we will add this time to E, the set of future breakpoints in the exit flow rate function. To repeat this loop, we then select the next breakpoint in either I or E.

Step 0: (Initialization)

$U_a(0)$, $V_a(0)$, $u_a(0)$, $v_a(0)$, $X_a(0)=0$

Set of entrance breakpoints $I \subset$ *all breakpoints of* $f(\cdot)$

Set of exit breakpoints $E=\varnothing$

$t_{curr} = \min(I)$

Step 1: (Main Loop)

$$t_{fut} = \min(I, E)$$

$$v_a(t) \Leftarrow \frac{u_a(s_a^{-1}(t_{curr}))(s_a^{-1}(t_{fut}) - s_a^{-1}(t_{curr}))}{t_{fut} - t_{curr}} \qquad \forall t \in [t_{curr}, t_{fut})$$

$$V_a(t) \Leftarrow \int_0^t v_a(w)dw \qquad \forall t \in [t_{curr}, t_{fut})$$

$$U_a(t) \Leftarrow \int_0^t u_a(w)dw \qquad \forall t \in [t_{curr}, t_{fut})$$

$$X_a(t) \Leftarrow U_a(t_{curr}) - V_a(t_{curr}) \qquad \forall t \in [t_{curr}, t_{fut})$$

$$s_a(t_{curr}) \Leftarrow t_{curr} + D_a(X_a(t_{curr}))$$

$$s_a(t) \Leftarrow \frac{s_a(t_{fut}) - s_a(t_{curr})}{t_{fut} - t_{curr}} \times (t - t_{curr}) + s_a(t_{curr}) \qquad \forall t \in [t_{curr}, t_{fut})$$

$$E = E \cup \{(s_a(t_{curr}))\}$$

Step 2: (Stopping Criterion)

If the network is empty then stop.

Otherwise,

$t_{curr} = \min(E)$

if($\min(I) < t_{curr}$)

$t_{curr} = \min(I)$

$I = I \setminus \{t_{curr}\}$

else

$E = E \setminus \{t_{curr}\}$

and go to Step 1.

2.5.2 A DNLP Algorithm for a Network

To extend the algorithm of Section 2.5.1 to a general network, we must construct intervals such that the values of $u_a(t)$ and $v_a(t)$ for a given arc are constant. Within each interval, we will then compute the value of all variables on the given arc. The algorithm then loops over the intervals of all arcs in the network. In the following section, we will review the network algorithm, highlighting the differences between it and the single arc algorithm.

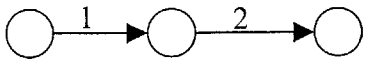
The initialization routine for the network algorithm is similar to that of the single arc algorithm. We initialize the network parameters for all arcs on the network, then place all know entrance flow breakpoints in a set I. In the network algorithm, it is necessary to associate the time at which a breakpoint in a stepwise function occurs with the arc on which it occurs. To do so, we create an element called a "step" and define a step to be the time of a breakpoint in an entrance or exit flow rate function, paired with the arc on which that breakpoint occurs. We represent a step at time t_{curr} on arc a as (t_{curr}, a). When we select the "minimum" step from a set of steps, we select that step whose time t_{curr} is the minimum. The "ArgMin" of t_{curr} is then the arc a. In the algorithm below, I is the set of all future steps in entrance flow rate functions of all arcs in the network.

In the main loop we first determine the value of t_{nxt} by selecting the minimum step in either I or E such that this step occurs on arc a. We then calculate the value of $u_a(t)$ by summing path entrance flow rate functions, $u_{ap}^{rs}(t)$ over all paths that contain arc a. We then compute $v_a(t), U_a(t)$, $V_a(t)$ and $X_a(t)$ in the same manner as in the single link algorithm. It is then necessary to split the arc exit flow rate, $v_a(t)$ among the paths that travel through a. Since the network is FIFO, any two paths entering the arc at time $s_a^{-1}(t)$ must both exit the arc at time t. Thus, it must also be true that the fraction of an arc's entrance flow rate that is on path p at time $s_a^{-1}(t)$ is equal to the fraction of the arc's exit flow rate on p at time t. To split the exit flow rate among paths, we then multiply $v_a(t)$ by the fraction:

$$\frac{u_{ap}^{rs}(s_a^{-1}(t_{curr}))}{u_a(s_a^{-1}(t_{curr}))}.$$

According to flow conservation, we then pass this exit flow rate along to the next arc on the path. To finish the main loop, we then determine the exit time of a flow entering arc a at t_{curr} and add new steps to the sets I and E. A step is added to I to conserve flow between arcs a and $\bar{a}$. A step is added to E because, as shown in Section 5.2.1, any breakpoint in either the entrance of exit flow rate function of an arc can cause a later breakpoint in the exit flow rate function.

In selecting the next step from the sets of entrance and exit flow rate steps, it is important to note that, if the minimum step of I occurs at the same time as the minimum step of E, then we should select the latter step first. To illustrate why, consider the following two successive links on some path:



Consider some time t_{curr} at which there exist a breakpoint in both $v_1(t)$ and $u_2(t)$. Suppose we first select the breakpoint in the function $u_2(t)$ and compute the values of arc 2's variables on some time interval. These computations would be based on the most recently computed value of $u_2(t)$, however this value of $u_2(t)$ will not reflect the breakpoint of $v_1(t)$ that occurs at t_{curr}. According to the flow conservation equation, a breakpoint in $v_1(t)$ will cause a breakpoint in $u_2(t)$, thus to yield correct results, our computations must be based on the value $u_2(t_{curr})$.

To correctly compute network variables, we should select the breakpoint in the function $v_1(t)$ first. We can then set the value of $u_2(t_{curr})$ according to:

$$u_2(t_{curr}) \Leftarrow v_1(t_{curr}).$$

At the next loop, we can then select arc 2 at time t_{curr} and correctly compute arc 2's variables. To avoid incorrect computations, in our algorithm we always select exit flow breakpoints before entrance flow breakpoints if they occur at the same time.

The algorithm follows.

Step 0: (Initialization)

$U_a(0),\ V_a(0),\ u_a(0),\ v_a(0),\ X_a(0)=0$ $\quad \forall a \in A$

Set of entrance steps $I \subset$ *all steps of* $f_p^{rs}(\cdot)$ $\quad \forall (r,s) \in K_{rs}, \forall p \in P$

Set of exit steps $E=\varnothing$

$t_{curr} = \min(I)$

$a = ArgMin(t_{curr})$

Step 1: (Main Loop)

$(t_{fut}, \alpha) = \min(I, E)\, s.t. \alpha = a$

$$u_a(t) \Leftarrow \sum_{p \in K_a} u_{ap}^{rs}(t) \qquad \forall t \in [t_{curr}, t_{fut})$$

$$v_a(t) \Leftarrow \frac{u_a(s_a^{-1}(t_{curr}))(s_a^{-1}(t_{fut}) - s_a^{-1}(t_{curr}))}{t_{fut} - t_{curr}} \qquad \forall t \in [t_{curr}, t_{fut})$$

$$V_a(t) \Leftarrow \int_0^t v_a(w)dw \qquad \forall t \in [t_{curr}, t_{fut})$$

$$U_a(t) \Leftarrow \int_0^t u_a(w)dw \qquad \forall t \in [t_{curr}, t_{fut})$$

$$X_a(t) \Leftarrow U_a(t_{curr}) - V_a(t_{curr}) \qquad \forall t \in [t_{curr}, t_{fut})$$

$$v_{ap}^{rs}(t) \Leftarrow \frac{u_{ap}^{rs}(s_a^{-1}(t_{curr}))}{u_a(s_a^{-1}(t_{curr}))} \times v_a(t_{curr}) \qquad \forall t \in [t_{curr}, t_{fut})$$

$$u_{a'p}^{rs}(t) \Leftarrow v_{ap}^{rs}(t_{curr}) \qquad \forall t \in [t_{curr}, t_{fut})$$

$$s_a(t_{curr}) \Leftarrow t_{curr} + D_a(X_a(t_{curr}))$$

$$s_a(t) \Leftarrow \frac{s_a(t_{fut}) - s_a(t_{curr})}{t_{fut} - t_{curr}} \times (t - t_{curr}) + s_a(t_{curr}) \qquad \forall t \in [t_{curr}, t_{fut})$$

$$I = I \cup \{(t_{curr}, \vec{a})\}$$

$$E = E \cup \{(s_a(t_{curr}), a)\}$$

Step 2: (Stopping Criterion)

If the network is empty then stop.

Otherwise,

$t_{curr} = \min(E)$

$a = ArgMin(t_{curr})$

if($\min(I) < t_{curr}$)

$t_{curr} = \min(I)$

$a = ArgMin(t_{curr})$

$I = I \setminus \{(t_{curr}, a)\}$

else

$E = E \setminus \{(t_{curr}, a)\}$

and go to Step 1.

2.6 Algorithm Implementation

The above algorithm was implemented in Java and tested on a Dell workstation running RedHat Linux 6.2 with a Pentium III 933MHz processor and 256MB RAM. In the following sections we highlight several important aspects of the implementation including input and data storage, the use of methods in the computer code and some details concerning the storage and manipulation of breakpoints.

2.6.1 Input Data Files and Representation of Paths

We first consider the input required by the algorithm. This consists of a list of origin-destination pairs, a list of paths between each of these pairs, a set of arc performance functions for all arcs in the networks, and a set of path flow rate functions. With the exception of the path flow rate functions, all of these inputs are properties of the network itself and it is therefore natural to group the lists of O-D pairs, path and arc performance functions together in a single data file. An example of such a file used in our implementation is included in Appendix A.

Recognizing that many paths in the network share one or more arcs, we use a data structure called a *subpath table* in our implementation to store paths. This data structure has been developed in the literature to reduce the amount of memory required to store paths (see for instance He 1997). In such a table, a path is stored as a set of subpaths, each of which stores a reference to the next arc and an index to a subpath. This is illustrated by the example below. Consider the paths [1,3, 4] and [2, 3, 4] in the small network shown in Figure 4.

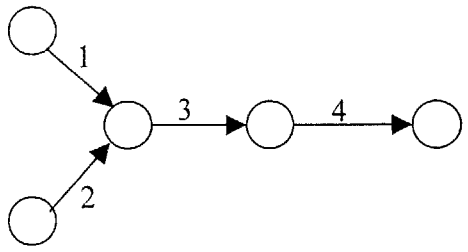


Figure 4: A Small Network

In a subpath table representation, arcs 3 and 4 are stored as subpaths which are referenced by both the subpath containing arc 1, and the subpath containing arc 2. Table 1 shows the resulting subpath array.

Table 1: A Subpath Representation

subpath number	arc number	next subpath
0	1	2
1	2	2
2	3	3
3	4	-1

The last lines of the network data file specify the subpath list used in our implementation. It should be noted that a relationship exists between the list of O-D pairs and the subpath array. Subpaths in the array are listed in such an order that the subpaths pertaining to the first O-D pair appear first, those pertaining to the second pair appear second, and so on. Thus if the first O-D pair is listed in the data file as having 5 paths, then subpaths 0 through 4 are the first subpaths of each of these paths.

Path flow rate data is also a required input. Since our algorithm assumes that this data is stepwise, we can store path flow rate functions by specifying the time of and value at each breakpoint. In our implementation, we specify stepwise functions such that $\forall x \in [a,b) \quad f(x) = \lim_{x \to b} f(x)$. An example of a path flow rate data file used in our implementation is included in Appendix A.

2.6.2 Use of Vectors to Store Functions

In implementing the algorithm, we recognized that the number of breakpoints in a flow rate function (whether for a path, arc or arc-path pair) may vary greatly from path to path, from arc to arc or from O-D pair to O-D pair. Additionally, even if we know the number of breakpoints in the path entrance flow rate functions, we have no *a priori* knowledge of the resulting number of breakpoints in subsequent arc or arc-path entrance and exit flow rate functions. Thus, it necessary that our data structure for flow rate functions be resized throughout the algorithm's execution. While this could be accomplished by resizing arrays, our implementation uses Java vectors to store stepwise and piecewise-linear functions. Vectors are Java objects which can be sized dynamically and are a convenient

way to store a variable number of objects. With respect to ease of implementation, the ease with which vectors can be appended made their use in this implementation natural.

2.6.3 Methods for Piecewise Functions

In reviewing the statement of the algorithm, it is clear that there are several operations on piecewise functions that must be carried out repeatedly during its execution. In particular, our algorithm requires that we add stepwise functions, integrate stepwise functions, and find the inverse of a piecewise linear function. In our implementation, each of these operations was written as a Java method which operates on a piecewise function or functions that have been stored as a vector. Discussion of each of these methods can be found in Appendix B. A pseudocode for each method is included.

2.6.4 Storage and Computation of Functions

In the following section we review how we can compute and store the stepwise and piecewise linear functions used in our algorithm. We note that our implementation often operates on functions, calculating values at all breakpoints. This approach was adopted for ease and clarity but often results in excess computation. More efficient approaches could be adopted by the implementer wishing to test the empirical performance of the algorithm. Our objective was instead to establish the feasibility of this continuous time approach.

In our implementation, the value of $u_a(t)$ is computed at each loop (for the current arc) for all times in the interval $[0, t_{curr})$ according to:

$$u_a(t) \Leftarrow \sum_{p \in K_a} u_{ap}^{ps}(t) .$$

This is done using a method which calculates the sum of stepwise functions over a given interval (Appendix B).

We now note that in order to compute:

$$v_a(t) \Leftarrow \frac{u_a(s_a^{-1}(t_{curr}))(s_a^{-1}(t_{fut}) - s_a^{-1}(t_{curr}))}{t_{fut} - t_{curr}}$$

it is necessary to determine the value of $s_a^{-1}(t)$ at the current and future time instants, and to determine the value of $u_a(t)$ at some previous time instant. In our implementation, $s_a^{-1}(t)$ is found by storing the function $s_a(t)$ for the interval $[0, t_{curr})$, then using a method to compute the function's inverse at a given point (this method is given in Appendix B). $u_a(s_a^{-1}(t_{curr}))$ is computed by storing the function $u_a(t)$ for the interval $[0, t_{curr})$, then using a method to find the function's value at a given point.

In order to compute each of the following two integrals in the algorithm:

$$V_a(t) \Leftarrow \int_0^t v_a(w)dw$$

$$U_a(t) \Leftarrow \int_0^t u_a(w)dw$$

we use the functions $u_a(t)$ and $v_a(t)$ for the interval $[0, t_{curr})$. We then use a method which computes the integral of a stepwise function over some interval to compute the values $U_a(t)$ and $V_a(t)$.

To compute $v_{ap}^{rs}(\cdot)$ we use:

$$v_{ap}^{rs}(t) \Leftarrow \frac{u_{ap}^{rs}(s_a^{-1}(t_{curr}))}{u_a(s_a^{-1}(t_{curr}))} \times v_a(t_{curr}).$$

This computation requires that we determine the value of $u_{ap}^{rs}(t)$ and $u_a(t)$ at previous time instants. We have noted above that in our implementation we compute and the latter function for the interval $[0, t_{curr})$ at each loop. To determine $u_{ap}^{rs}(t)$, we store all arc-path entrance flow breakpoints for the interval $[0, t_{curr})$. $s_a^{-1}(t)$ is computed using a method

that finds a function's inverse at a given point. The value of $v_a(t_{curr})$ is computed at an earlier step in the loop.

Each of the remaining computations in the algorithm is straightforward. In summary we note that in our implementation, the functions $u_{ap}^{rs}(t)$, $v_a(t)$ and $s_a(t)$ are stored for each arc and path for the interval $[0, t_{curr})$. All other functions and values are computed at each loop using methods which manipulate stepwise and piecewise linear functions.

To select the next exit flow rate breakpoint, we maintain a heap of the next steps in the exit flow rate functions of each arc. Since entrance flow rate functions are stored by arc-path pair, if we wished to select the next entrance flow breakpoint in a similar manner, we would need to maintain a larger heap with the next step on all arcs and all paths on each arc. To reduce the computational burden of this selection operation, we instead maintain a heap of the next breakpoints on any path for each arc. To select the next entrance flow rate breakpoint, we then select the element at the top of this smaller heap.

2.6.5 Other Notes on the Implementation

With regard to maintaining the set of future breakpoints in our implementation, we wish to note the following. At some time t_{curr}, the exit time of a flow entering a given arc at time t_{curr} is computed using:

$$s_a(t_{curr}) \Leftarrow t_{curr} + D_a(X_a(t_{curr})).$$

It should be noted, however that at time t_{curr}, we do not know the value of $v_a(s_a(t_{curr}))$. This is because $v_a(s_a(t_{curr}))$ is computed according to:

$$v_a(s_a(t_{curr})) = \frac{u_a(t_{curr})}{s'_a(t_{curr})}.$$

To determine $v_a(s_a(t_{curr}))$ we need the values of $u_a(t_{curr})$ and $s'_a(t_{curr})$. Since $s_a(t_{curr})$ is a piecewise linear function, we the value of $s_a(t_{curr})$ at at least two points in the

interval $[t_{curr}, t_{fut})$ to compute $s'_a(t_{curr})$. At time t_{curr} however, we do have sufficient information to compute $s'_a(t_{curr})$ and thus cannot compute $v_a(s_a(t_{curr}))$. Thus, in our implementation at time t_{curr}, we "mark" time $s_a(t_{curr})$ as a future breakpoint of the exit flow rate function of a, but do not compute its value until the algorithm "arrives" at $s_a(t_{curr})$.

2.7 Algorithm Testing

The algorithm was tested on the sample network of 9 nodes and 12 arcs in Figure 5. This network is equivalent to the network used by Xu et al. (1999) to test a discretized version of a continuous DNLP algorithm. By testing our implementation on the network below, using similar data sets, we can compare and evaluate our results.

We note that throughout this thesis values of time, flow rates and other variables are unitless quantities. The example is not intended to model a real-world transportation system but rather to illustrate the correctness of our algorithms and implementations.

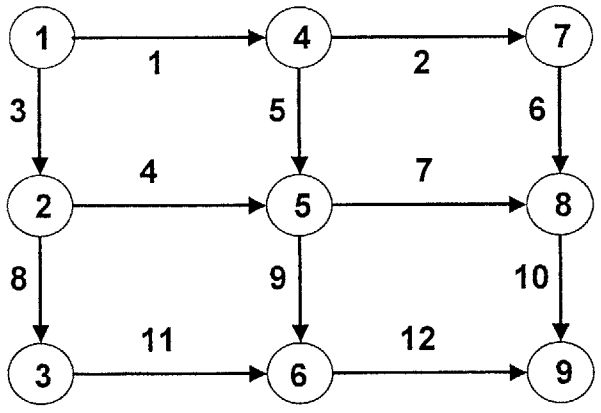


Figure 5: A Sample Network

The network includes 7 O-D pairs and 14 paths. These pairs and paths are listed in Table 2 below.

Table 2: Network Data

O-D Pair	Path
(1,9)	1: (1, 2, 6, 10)
	2: (1, 5, 7, 10)
	3: (1, 5, 9, 12)
	4: (3, 4, 7, 10)
	5: (3, 4, 9, 12)
	6: (3, 8, 11, 12)
(1, 5)	7: (1, 5)
	8: (3, 4)
(5, 9)	9: (7,10)
	10: (9, 12)
(1, 3)	11: (3, 8)
(3, 9)	12: (11, 12)
(1, 7)	13: (1, 2)
(7, 9)	14: (6, 10)

As Xu et al. (1999) implemented a discretized version of a continuous DNLP algorithm, in their implementation it was not necessary to select path flow rate functions and arc performance functions which permit exact continuous-time solution. In particular, Xu et al. (1999) specify the above functions as polynomials. Since it is the goal of this thesis to achieve exact, continuous time implementation, we select affine arc performance functions and stepwise path flow rate functions. To provide a basis for comparison, we selected the above functions to approximate the polynomials used by Xu et al. (1999).

Path flow rate functions in Xu et al (1999) are of the form:

$$f_p^{rs}(t) = \theta_p(5t - t^2)$$

where θ_p is a parameter of the path ranging from 0.1 to 0.35. Step functions containing 5 steps were selected to approximate these functions. Figure 6 shows the relationship of

the original function to the stepwise approximation for values of θ_p of 0.1, 0.15, and 0.2. Table 3 gives the parameter θ_p for each of the paths in the network.

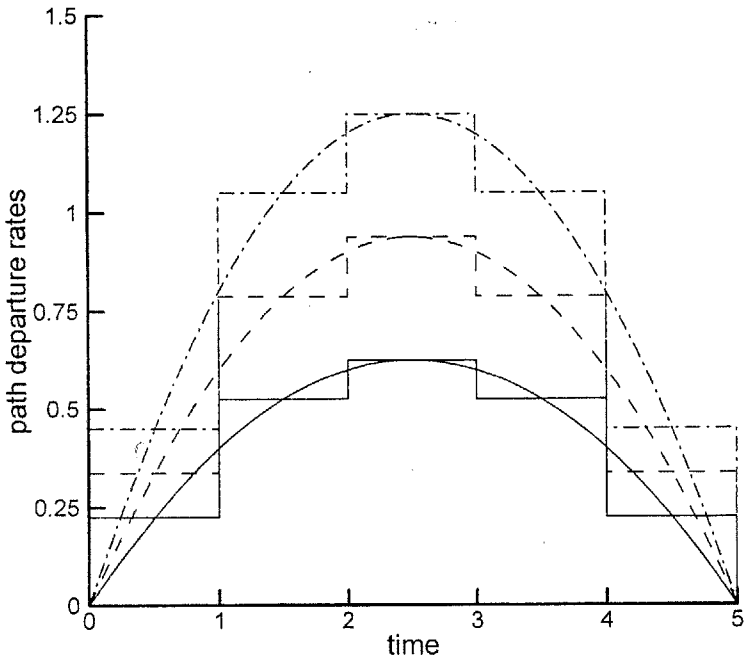


Figure 6: Approximation of Path Flow Rate Functions

Table 3: Parameters for Path Flow Rate Functions

Path	θ_p
1	0.10
2	0.15
3	0.18
4	0.20
5	0.10
6	0.12
7	0.20
8	0.10
9	0.15
10	0.15
11	0.12
12	0.10
13	0.10

14	0.35

Selection of functions to approximate arc performance functions used by Xu et al. (1999) is discussed in Appendix C. Table 4 contains a summary of the parameters of these functions.

Table 4: Parameters for Arc Performance Functions

Arc	Intercept	Slope
1	1.88	0.235
2	1.80	0.443
3	1.39	0.269
4	1.60	0.443
5	1.32	0.408
6	1.92	0.408
7	1.50	0.443
8	1.62	0.408
9	1.80	0.443
10	2.12	0.408
11	2.00	0.443
12	2.32	0.408

2.7.1 DNLP Results

Figure 7 shows the arc exit time functions for each of the twelve arcs in the example problem. Comparison of this figure with the results given in Xu et al. (1999) shows that the continuous time implementation of the Dynamic Network Loading algorithm yields results comparable to those from the discretized version. Small discrepancies do exist; these are most likely due to the linear approximation of the arc performance functions or stepwise approximation of path flow rate functions. The general similarities between the results of the models and algorithms in this thesis an in Xu et al. (1999) indicate the correctness of each.

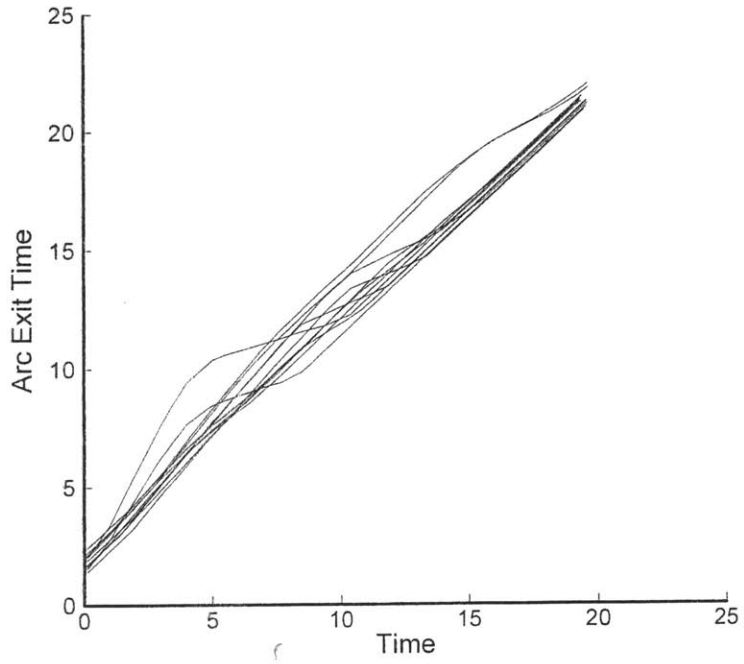


Figure 7: Arc Exit Times

Here it is important to note that all of the exit time functions shown in Figure 7 are strictly increasing functions and are greater than or equal to t for all times in the analysis period. This indicates that the computed arc travel times do indeed verify the FIFO property.

Figure 8 shows the path traversal times for each of the 14 specified paths. These functions show a shape similar to those in Xu et al. (1999) with path travel time increasing as t increases and arcs become congested. The figure also shows that the paths containing more arcs have greater travel times than those containing fewer arcs.

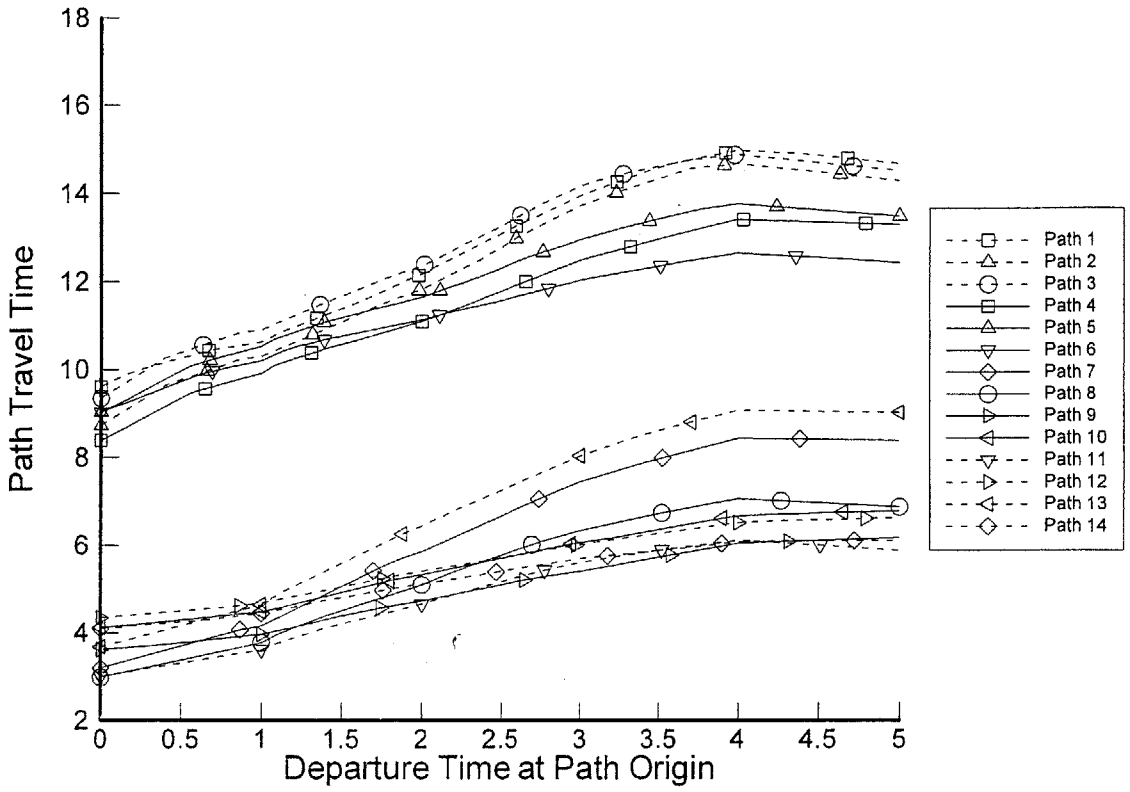


Figure 8: Path Traversal Times

Figure 9 shows arc volumes on each of the 12 arcs throughout the analysis period. This figure is compatible with the results of Theorems 4 and 5. Early in the analysis period, the functions contain relatively few breakpoints and the piecewise linear form can be clearly seen. As t increases, the number of breakpoints in each function increases rapidly until the functions appear smooth (though in fact, they consist of many small linear pieces). This is because, for each arc, the number of breakpoints in the arc's exit flow rate functions may be greater than the number of breakpoints in its entrance flow rate function. The next arc on the path then uses the exit flow rate function of the previous arc as its entrance flow rate function and the number of breakpoints in the function is again increased. This process continues until flows reach the end of their path, thus the number of breakpoints in the functions of each arc tends to increase with time. Furthermore, it should be noted that the number of breakpoints in the entrance flow rate

function of each arc may be as great as the sum of the number of breakpoints in each of the arc's path flow rate functions. This also increases the number of breakpoints The shapes and magnitude of the arc volume functions are in agreement with the results presented by Xu et al. (1999). The results shown in Figure 9 are reproduced in Appendix D for the reader wishing to identify particular arc flows.

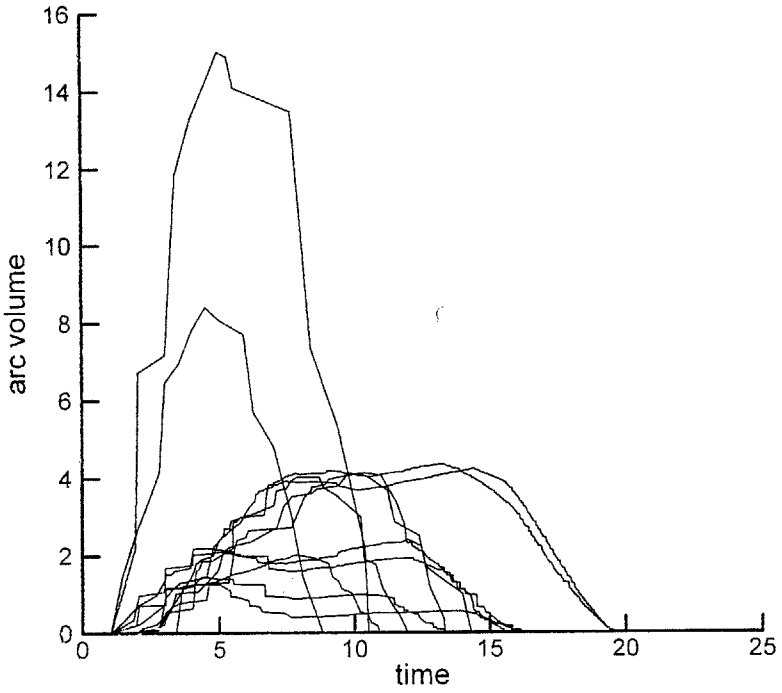


Figure 9: Arc Volumes

2.8 Conclusions

To conclude this chapter we note that we have provided a formulation of the DNLP and proven than we can solve the DNLP in continuous time for the case of affine arc performance functions and stepwise path flow rate functions. We have presented a DNLP algorithm and implemented this algorithm, testing it on a small example network. Results of this testing indicate that this implementation is correct since they are in agreement with those of similar models and with the theoretical developments of this chapter. They also confirm findings regarding the number of breakpoints in the DNLP

solution. We note that for any execution of the DNLP algorithm, the number of breakpoints in the exit flow rate functions can be greater than the number of breakpoints in the input functions. Furthermore, we have noted that as the value of t increases, the number of breakpoints in any given time interval is likely to increase. The implications of these conclusions on the efficiency of a DTA solution will be discussed in Chapter 4 of this thesis.

CHAPTER 3

CONTINUOUS TIME DYNAMIC TRAFFIC ASSIGNMENT

In Chapter 2, we established that, given path flow rate functions, we can use a continuous-time dynamic network loading problem (DNLP) algorithm to calculate time-dependent arc travel times. We now wish to use these arc-based network conditions to determine path flows. To do so, we use a shortest paths algorithm to calculate path travel times based on arc travel times, then assign flows to these shortest paths. Through multiple revisions of our set of path flows, we can obtain a dynamic traffic assignment (DTA) solution.

The objective of this chapter is to present several algorithms which, when combined with the DNLP algorithm of Chapter 2, yield a DTA solution. The first of these algorithms is a dynamic shortest paths (DSP) algorithm which enables us to determine path flows. The formulation of the DSP problem and development of a solution algorithm are given in Section 3.1. In Section 3.2 we present a DTA algorithm which manages the DNLP and DSP subroutines to obtain a DTA solution. Before presenting these developments, we outline the remaining steps required to solve the DTA problem.

The first step is to obtain a set of dynamic shortest paths. In a static network, calculation of path travel times from arc travel times is straightforward. We can traverse paths, adding the cost of each arc along a given path. If we are interested in the minimum travel time from a given origin to a given destination, we can use one of many existing static shortest paths algorithms. In a dynamic network, calculation of path travel times from arc travel times is more complex, since the minimum travel time for a given O-D pair may vary with departure time. Furthermore, because arc travel times are time-dependent, the shortest path itself may vary as a function of departure time.

The next step is to obtain a set of path flows. To do so, we compute the minimum travel time between given O-D pairs and use this knowledge to assign time-dependent flow along each of the paths. Various methods for performing this assignment exist. In a deterministic user-optimum model, all flow between a given origin and destination travels along a shortest path. In other models, however, it is assumed that users have imperfect information regarding the shortest paths in the network, or are unwilling to deviate from their preferred path (He 1997). In such models, system users are grouped into several categories according to their preferences. This grouping, a "Users' Behavior Model," assigns flow along a set of paths for each origin-destination pair. In order to maintain the functional forms of our network variables (which permit continuous-time solution), we assume users have perfect information regarding travel times on all routes and always select the path with the minimum travel time.

Having computed dynamic shortest paths and assigned flows along these paths, we have obtained a revised set of path flows. The final step required to obtain a DTA solution is to continue to revise the set of path flows until a satisfactory solution is obtained. If path flows obtained at successive iterations of the DTA algorithm are sufficiently similar, we conclude with a solution to the DTA problem. If, however, the results are dissimilar, we continue to revise our set of path flow rate functions. In Section 3.2 we present a DTA algorithm which yields a continuous-time solution. This algorithm uses the continuous-time DNLP and DSP algorithms to solve the DNLP and DSP subproblems, respectively.

As in other DTA models, we use a Method of Successive Averages (MSA) algorithm to generate a set of path flow rate functions at each traversal of the DTA loop.

Section 3.1 gives formulations and algorithms for the DSP problem. We first consider the one-to-all problem, then consider the all-to-one problem. Results from implementation and testing of the one-to-all algorithm are also given. In Section 3.2 we discuss the DTA problem and present an algorithm for its solution. Results of DTA testing on a small network are included.

3.1 Continuous-Time Dynamic Shortest Paths

The Dynamic Shortest Paths problem has been studied extensively in the literature, yet because the problem has many variants, each of which may yield different solution properties, further exploration continues to yield interesting results. As with other dynamic problems, most existing developments pertain to a discrete-time version of the problem. Since the goal of this thesis is to present a set of continuous-time algorithms for the DTA problem, in the following section we explore relevant continuous-time variants of the DSP problem.

Like their discrete-time analogs, some continuous-time DSP algorithms are dynamic adaptations of well-known static shortest path algorithms. Orda and Rom (1990, 1991) presents a theoretical algorithm that can be viewed as a dynamic extension of the well-known Bellman-Ford algorithm for static shortest paths. Yadappanavar (2000) and Chabini and Yadappanavar (2000) present a practical interpretation of Orda and Rom's algorithm, given piecewise linear input functions. This algorithm was implemented and tested against other continuous-time DTA algorithms. In Section 3.1.2 we present an algorithm which can be understood as another interpretation of the Bellman-Ford algorithm. The development of this algorithm was largely undertaken prior to the implementation of the Orda and Rom (1990, 1991) algorithm.

While the representation of time (discrete versus continuous) is one dimension along which DSP problems can vary, several other dimensions also exist. Problems may be either deterministic or stochastic; they may be concerned with minimizing travel time or some more generalized cost; they may require that the network verify the FIFO property or they may permit non-FIFO networks; they may allow or prohibit waiting at nodes. The statement of the problem itself may also vary. Some statements require a solution for a single departure time, while others require shortest paths for all departure times. Alternately, some problems require solutions for one or all arrival times. Some may require shortest paths from one node to all nodes in the network while others require those from all nodes to one node.

To narrow our discussion of continuous-time shortest paths algorithms we specify a problem variant which is appropriate for the DTA problem variant of this thesis. We first note that at a given iteration, the DTA will require a solution of the DSP problem between all origin-destination pairs on which there is flow, thus a many-to-many solution of the DSP problem for all departure times is required. To obtain this we may execute either a one-to-all or an all-to-one algorithm repeatedly. We also note that the proofs of Chapter 2 guarantee that the network travel times verify the FIFO property. As a result, waiting at nodes will never improve our shortest paths solutions and need not be considered. We also know the network to be deterministic. We will assume that we wish to minimize path travel time as this is variant of the problem that is relevant to the DTA problem of this thesis. Hereafter in this thesis, the term "shortest path" will refer to the minimum travel time problem.

In Section 3.1.2 we first present a formulation and algorithm for a one-to-all variant of the DSP problem. This algorithm has been implemented and tested on the small sample network of Chapter 2. We note that this algorithm requires that the network verify the FIFO property thus, for greater generality and the benefit of other applications, we will also present an all-to-one version in Section 3.1.3 which permits non-FIFO networks as well.

3.1.1 Notation, Definitions and Assumptions

In the following section we present notation and definitions for the one-to-all dynamic shortest paths problem for all departure times. This notation will first be used in the development of optimality conditions for the problem and again in the statement of the algorithm. The reader may note that this notation (namely that of exit time functions) differs from that of Chapter 2. In both chapters we have chosen notation that is consistent with that which is in use in the literature, though these notations differ from each other.

We consider a network $G = (N, A)$ consisting of a set of nodes N and a set of arcs, A. We designate one node, r as the origin node and assume that there exists a path from r to every other node in N. We also use the notation:

$d_{ij}(t)$: the travel time on an arc between nodes i and j, departing i at time t;

$d_i(t)$: the minimum travel time from r to i, departing r at time t;

$B(j)$: the set of nodes i for which there is some arc (i, j); and

$[0, T_\infty]$: the time interval of interest. T_∞ is defined in Chapter 2; beyond T_∞ we assume the network to be static.

We denote by $a_{ij}(t)$ the arrival time at node j if we depart node i at time t. We have:

$$a_{ij}(t) = d_{ij}(t) + t.$$

We note that if an arc verifies the strict FIFO property (given in Chapter 2 of this thesis), then we know $a_{ij}(t)$ to be a strictly increasing function. Similarly, we also denote by $a_i(t)$ the arrival time at node i if we depart node r at time t. It is defined as $a_i(t) = d_i(t) + t$.

Consider some path $(i_1 \rightarrow i_2 \rightarrow ... \rightarrow i_k)$. According to our notation, if we depart node i_1 at some time t, we will arrive at node i_2 at $a_{i_1,i_2}(t)$. We will then arrive at node i_3 at $a_{i_2,i_3}(a_{i_1,i_2}(t))$. Continuing in this manner, the arrival time at node i_k, given by:

$$a_{i_{k-1},i_k}(a_{i_{k-2},i_{k-1}}(\cdots(a_{i_1,i_2}(t)))).$$

In Chapter 2 we proved that the solution to the DNLP is FIFO on every arc in the network and thus that each of the functions composed in the expression above is strictly increasing. Proofs presented in Chabini and Yadappanavar (2000) provide the properties of the composed functions. A summary of these properties follows:

- If $f(\cdot)$ and $g(\cdot)$ are two non-decreasing functions, then the composition of these functions is also non-decreasing.
- The composition function of a finite number of non-decreasing functions is a non-decreasing function.
- For any path in a FIFO network, the arrival time at the end of the path as a function of the departure time at the start of the path is non-decreasing.
- For any path in a FIFO network, the minimum arrival time at a destination node is an increasing function of the departure time at the origin node.
- Waiting at any node in a FIFO network never decreases the arrival time at the end of any path.

The above properties allow us to assume that no waiting occurs at nodes and therefore that $a_i(t)$ is a non-decreasing function. As we will see in Section 3.1.2, this will allow us to develop and efficiently implement a continuous-time DSP algorithm.

3.1.2 The One-to-All Minimum Travel Time Paths Problem: Formulation and Solution Algorithm

We now present a set of optimality conditions for the one-to-all minimum travel time problem. We note that if $a_j(t)$ is indeed the arrival time on a shortest path from node r to node j, then it must verify the necessary condition:

$$a_r(t) = t \qquad \forall t \in [0, T_\infty]$$

$$a_j(t) = \underset{i \in B(j)}{Min}\left(\underset{s \geq a_i(t)}{Min}(a_{ij}(s))\right) \qquad j \neq r, \forall t \in [0, T_\infty].$$

Since we know all arcs to in the network to be FIFO, the latter equation can be rewritten as:

$$a_j(t) = \underset{i \in B(j)}{Min}(a_{ij}(a_i(t)))$$

If the latter condition is not verified at any point, then there must exist some path to j via some node $k \in B(j)$ such that $a_j(t) > (a_{kj}(a_k(t)))$. If this were true, we could reduce the travel time from r to j by traveling via k, thus contradicting the optimality of $a_j(t)$.

Let us now prove that the optimality conditions are sufficient. We want to prove that for all $i \in N$ and $t \in [0, T_\infty]$, if $a_j(t)$ verifies the optimality conditions, then $a_j(t)$ is the minimum arrival time at node j if one departs node r at time t. To prove this, it suffices to prove that $a_j(t)$ is less than or equal to the arrival time at the end of any path from node r to node j. Let $p = i_1 \to i_2 \to \cdots \to i_q$ be an arbitrary path between r and j. The arrival time at j if one departs node r at time s is:

$$a_{i_{q-1}, i_q}(a_{i_{q-2}, i_{q-1}}(\cdots(a_{i_1, i_2}(t)))).$$

We want to prove that $a_j(t)$ is less than or equal to the above quantity. We have:

$$a_{i_2}(t) \leq a_{i_1 i_2}(t).$$

Hence,

$$a_{i_2 i_3}(a_{i_2}(t)) \leq a_{i_2 i_3}(a_{i_1 i_2}(t)).$$

We know from optimality conditions that

$$a_{i_2 i_3}(a_{i_2}(t)) \geq a_{i_3}(t),$$

hence;

$$a_{i_3}(t) \leq a_{i_2 i_3}(a_{i_1 i_2}(t)).$$

Repeating the above argument for arcs $(i_3, i_4)\cdots(i_{q-1}, i_q)$, we have:

$$a_{i_q}(t) \le a_{i_{q-1}i_q}(a_{i_{q-2}i_{q-1}}(\cdots(a_{i_1,i_2}(t)))).$$

Thus, any path from r to j must have an arrival time greater than or equal to $a_j(t)$. Furthermore, since $a_j(t)$ is the arrival time of some path from r to j, then $a_j(t)$ is the minimum arrival time at j if one departs node r at time t.

□

We now present an algorithm for the minimum time dynamic shortest paths problem that uses the above developments. This algorithm is a dynamic adaptation of the well-known Bellman-Ford algorithm for static shortest paths. The static algorithm operates by determining, at the kth iteration, the shortest paths from some source to all nodes among paths which may have no more than k arcs. Initially, $k = 1$ and only those nodes adjacent to the source have finite shortest path labels. As the value of k is increased, the number of nodes reachable from the origin increases, and when k is equal to the maximum number of arcs on a shortest path, the algorithm terminates with an optimal solution. In the dynamic case the algorithm is similar in structure, however the value of the minimum time path to a given node changes as a function of the time, as does the path itself.

We denote by $a_j^k(t)$ the minimum arrival time at j on a path from r that contains at most k arcs. In accordance with the optimality condition given previously in this section, at every iteration we wish to find the value of $a_j(t) = a_j^k(t)$ that satisfies:

$$a_j^k(t) \le a_{ij}(a_i^{k-1}(t)).$$

To find such a value, we first determine the minimum arrival time function for a path containing exactly k arcs. We then compare this function to the minimum arrival time function of a path containing fewer than k arcs. At any time t, the lesser of these two functions is the minimum arrival time function of a path containing at most k arcs.

We denote by $a^k_{i\to j}(t)$ the arrival time at j on a path with exactly k arcs from the source to node j via a given predecessor node i. We calculate $a^k_{i\to j}(t)$ according to:

$$a^k_{i\to j}(t) = a_{ij}(a^{k-1}_i(t)).$$

We note that the minimum travel time function of a path from node r to node j containing exactly k arcs is equal to:

$$\underset{i\in B(j)}{Min}(a^k_{i\to j}(t)).$$

Thus, we compute $a^k_j(t)$ for each $i \in B(j)$ and select the minimum of these functions. To do so, we loop over the indices i, making a pair-wise comparison at each loop. To find the minimum arrival time function for a path containing at most k arcs, we then compare the resulting function with $a^{k-1}_j(t)$ and select the minimum for all $t \in T_\infty$.

While we could execute the main loop of the algorithm until k is equal to the maximum number of arcs in a shortest path, we recognize that a more efficient termination criterion may exist. We observe that if, for any value of k, $a^k_j(t) = a^{k-1}_j(t)\ \forall j \in N$, then successive iterations will never yield a path with lesser arrival time functions. Thus, we can terminate the algorithm if this condition is ever met. In any case, $k \le (n-1)$ as there always exist optimal paths with at most *(n-1)* arcs.

We present the algorithm below. For clarity, we have omitted the operation of setting predecessor node values. In the implementation setting these values is straightforward and the reader may note that these values may need to be reset each time the "Min" function is called.

One-To-All Bellman-Ford Algorithm for Continuous-Time Dynamic Networks

Given: *G=(N,A)*

source node *r*

Step 0: Initialize

$a^l_j(t) = \infty$ $\qquad \forall t \in [0, T_\infty], \forall j \neq r, l = 0,1$

$a_r^l(t) = t$ $\forall t \in [0, T_\infty], \forall l \geq 0$

$k = 1$

$a_j^k(t) = a_{rj}(t)$ $\forall j \in A(r)$

Step 1: Compute Shortest Paths

While $a_j^k(t) \neq a_j^{k-1}(t) \forall j \in N, \forall t \in [0, T_\infty]$

$k = k + 1$

For every node j

$a_j^k(t) = +\infty$

For every node $i \in B(j)$

$a_{i \to j}(t) \Leftarrow a_{ij}(a_i^{k-1}(t))$ $\forall t \in [0, T_\infty]$

$a_j^k(t) \Leftarrow Min(a_j^k(t), a_{i \to j}(t))$ $\forall t \in [0, T_\infty]$

$a_j^k(t) = Min(a_j^{k-1}(t), a_j^k(t))$ $\forall t \in [0, T_\infty]$

We now consider how the above algorithm relates to that of Orda and Rom (1990, 1991). Both algorithms iterate over the value of k. At each iteration our algorithm compares a optimal path of exactly k arcs from r to j (via some node i), to that of an optimal path of at most k-1 arcs from r to j. In contrast, Orda and Rom's algorithm computes the optimal paths containing at most k arcs from r to j via all intermediate nodes. It then selects the shortest of these paths. Bertsikas and Tsitsiklis (1989) contains a discussion of the difference between these two approaches as it pertains to static networks.

3.1.3 The All-to-One Minimum Travel Time Paths Problem: Formulation and Solution Algorithm

In order to develop optimality conditions and an algorithm for the all-to-one problem, we adopt the following notation change. We denote by s the destination node and denote by $a_i(t)$ the arrival time at s if we depart some node i at time t. Additionally, we denote

by $A(i)$ the set of nodes j such that there exists an arc (i, j). The development of necessary and sufficient optimality conditions for the all-to-one problem is very similar to that of the one-to-all problem. This development follows.

If $a_i(t)$ is the arrival time on the shortest path from node i to node s, then it must verify the necessary condition:

$$a_s(t) = t \qquad \forall t \in [0, T_\infty]$$

$$a_i(t) = \underset{j \in A(i)}{Min}(a_j(a_{ij}(t))) \qquad i \neq s, \forall t \in [0, T_\infty].$$

If this condition is not verified at any point, then there must exist some path from i to s via some node $k \in A(i)$ such that $a_i(t) > (a_k(a_{ik}(t)))$. If this were true, we could reduce the travel time from i to s by traveling via k, thus contradicting the optimality of $a_i(t)$.

Let us now prove that the optimality conditions are sufficient. We want to prove that for all $i \in N$ and $t \in [0, T_\infty]$, if $a_i(t)$ verifies the optimality conditions, then $a_i(t)$ is the minimum arrival time at node s if one departs node i at time t. To prove this, it suffices to prove that $a_i(t)$ is less than or equal to the arrival time at the end of any path from node i to node s. Let $p = i_1 \to i_2 \to \cdots \to i_q$ be an arbitrary path. The arrival time at s if one departs node i at time t is:

$$a_{i_{q-1}, i_q}(a_{i_{q-2}, i_{q-1}}(\cdots(a_{i_1, i_2}(t)))).$$

We want to prove that $a_i(t)$ is less than or equal to the above quantity. We have:

$$a_{i_2}(t) \leq a_{i_1 i_2}(t).$$

Hence,

$$a_{i_2 i_3}(a_{i_2}(t)) \leq a_{i_2 i_3}(a_{i_1 i_2}(t)).$$

We know from optimality conditions that

$$a_{i_2 i_3}(a_{i_2}(t)) \geq a_{i_3}(t),$$

hence;

$$a_{i_3}(t) \leq a_{i_2 i_3}(a_{i_1 i_2}(t)).$$

Repeating the above argument for arcs $(i_3, i_4)\cdots(i_{q-1}, i_q)$, we have:

$$a_{i_q}(t) \le a_{i_{q-1}i_q}(a_{i_{q-2}i_{q-1}}(\cdots(a_{i_1 i_2}(t)))).$$

Thus, any path from i to s must have an arrival time greater than or equal to $a_i(t)$. Furthermore, since $a_i(t)$ is the arrival time of some path from i to s, then $a_i(t)$ is the minimum arrival time at s if one departs node i at time t.

□

Below we present an algorithm for the all-to-one minimum time DSP problem. This algorithm is similar to that of the one-to-all problem. At each iteration, it considers paths of at most k arcs. For each origin node i, it first finds the minimum travel time among paths of exactly k arcs to node s, via any intermediate node, j. This is done by composing the arrival time function of j corresponding to the minimum travel time from j to s using exactly k-1 arcs with that of the arc (i, j). It then compares the obtained arrival time functions to the minimum arrival time functions among paths containing at most k-1 arcs. The minimum of the latter functions is then computed for all values of t in the analysis period. The value of k is increased at each iteration until it is known that subsequent iterations will never yield paths of lesser travel time.

All-to-One Bellman-Ford Algorithm for Continuous-Time Dynamic Networks

Given: $G=(N,A)$

destination node s

Step 0: Initialize

$a_i^l(t) = \infty$ $\quad \forall t \in [0, T_\infty], \forall i \ne s, l = 0, 1$

$a_s^l(t) = t$ $\quad \forall t \in [0, T_\infty], \forall l$

$k = 1$

$a_i^1(t) = a_{is}(t)$ $\quad \forall t \in [0, T_\infty], \forall i \in B(s)$

Step 1: Compute Shortest Paths

While $a_i^k(t) \ne a_i^{k-1}(t) \forall i \in N, \forall t \in T$

$k = k + 1$

For every node i

$$a_i^k(t) = \infty$$

For every node $j \in A(i)$

$$a_{i\to j}(t) \Leftarrow a_j^{k-1}(a_{ij}(t)) \qquad \forall t \in [0, T_\infty]$$

$$a_i^k(t) \Leftarrow Min(a_i^k(t), a_{i\to j}(t)) \qquad \forall t \in [0, T_\infty]$$

$$a_i^k(t) = Min(a_i^{k-1}(t), a_i^k(t)) \qquad \forall t \in [0, T_\infty]$$

We note that in this algorithm, it is not necessary that the FIFO property be verified. In the one-to-all formulation FIFO must be verified in order for the optimality condition:

$$a_j(t) = \underset{i \in B(j)}{Min}(a_{ij}(a_i(t)))$$

to be valid. In the all-to-one algorithm however, the optimality condition:

$$a_i(t) = \underset{j \in A(i)}{Min}(a_j(a_{ij}(t)))$$

does not require FIFO, therefore the all-to-one algorithm does not require the FIFO property to be verified. If $d_{ij} \geq \varepsilon > 0$, there exist optimal paths of at most $\left\lceil \frac{T_\infty}{\varepsilon} \right\rceil + n - 1$ arcs. Thus the algorithm will terminate after k is at most $\left\lceil \frac{T_\infty}{\varepsilon} \right\rceil + n - 1$. If FIFO is verified on the network, then there always exist an optimal path that is acyclic and there will exist a solution with at most *(n-1)* arcs. Thus, if FIFO is verified the algorithm will terminate with k equal to at most *(n-1)*.

3.1.4 Implementation of a Dynamic Shortest Paths Algorithm

In this section we describe the implementation of the one-to-all algorithm. While the applications of this algorithm are more limited than those of the all-to-one algorithm (since the FIFO property must be verified), the one-to-all algorithm is appropriate for the DTA problem since, according to the proofs of Chapter 2, we know the network to verify the FIFO property.

In order to enable continuous-time implementation of the above algorithms, we must be able to perform two operations on piecewise linear functions. Firstly, we must be able to compose any two non-decreasing piecewise linear functions. Secondly, we must be able to find the minimum function of two non-decreasing piecewise linear functions. Our implementation uses subroutines to perform each of these operations. Details of each these subroutines are given in Appendix B.

To ensure that the above subroutines will terminate in a finite number of iterations, we refer to Lemmas 2.1-2.7 of Chabini and Yadappanavar (2000). These lemmas specify the domain and range of the resulting functions and prove that the resulting functions are piecewise linear. Furthermore, they provide a finite bound on the number of linear pieces of the resulting functions.

In particular, Chabini and Yadappanavar (2000) proves that if $f(t)$ and $g(t)$ are non-decreasing piecewise linear functions, then the number of linear pieces of $h(t) = f(g(t))$ is bounded from above by the sum of the number of linear pieces in $f(t)$ and $g(t)$. Chabini and Yadappanavar (2000) also illustrates that the number of linear pieces of $h(t) = Min(f(t), g(t))$ is bounded by twice the sum of the number of linear pieces in $f(t)$ and $g(t)$. An important result however is that Lemma 3.3 of Chabini and Yadappanavar (2000) proves that the number of linear pieces of each arrival time function is less than or equal to twice the total number of linear pieces in all arc travel time functions.

The one-to-all algorithm was implemented in Java and tested on a Dell workstation running RedHat Linux 6.2 with a Pentium III 933MHz processor and 256MB RAM. The algorithm was tested on the sample network presented in Chapter 2 using output data from the DNLP algorithm. The algorithm yields a piecewise-linear, time-dependent minimum time path to each destination node. Since the main objective of this thesis is not the development of efficient DSP algorithms but rather the implementation of a DTA algorithm, the all-to-one algorithm has not yet been implemented. It is, however

anticipated that this implementation will be relatively straightforward for FIFO networks as we already implemented the functional operations needed for its implementation.

Figure 10 shows partial results of our shortest path algorithm for some set of arc travel time functions found on iteration 8 of the DTA algorithm. The graph shows the shortest path between nodes 1 and 9 computed by the algorithm. For comparison, the path travel time on all paths between nodes 1 and 9 of the example network given in Figure 5 are also shown. The reader may note that the shortest path travel time never exceeds the path travel time on any of the paths. At each instant in time, the shortest path travel time is equal to the minimum travel time of all the paths. Consider, for example, the time interval defined by approximately [1.6, 2.3]. On this interval, the travel time on Path 1 is equal to the shortest path travel time and the travel times of other paths are greater. The graph illustrates that our implementation is correct and that over a given time period, the travel time on the shortest path changes as does the path itself.

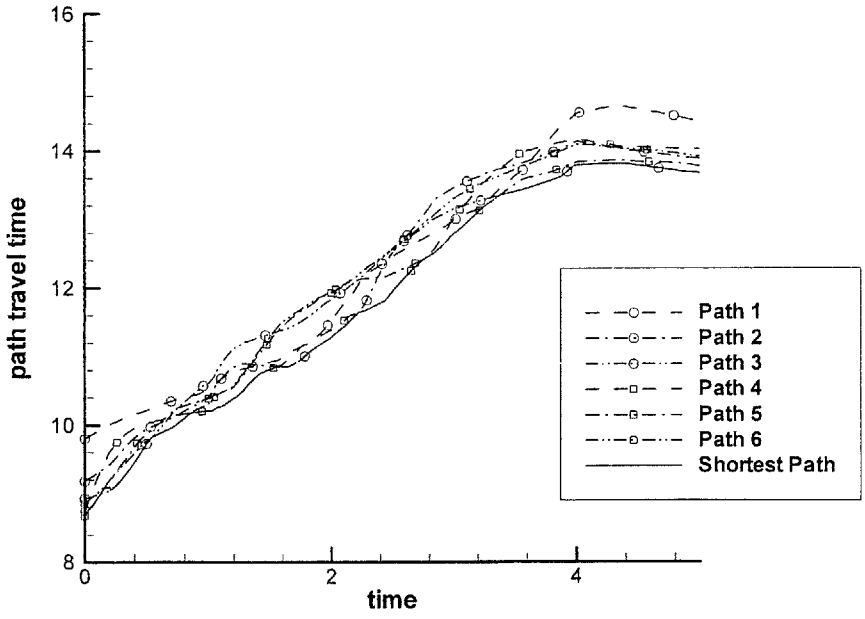


Figure 10: Results of Shortest Paths

3.2 A DTA Solution Algorithm

In Chapter 2 and Section 3.1 we have implemented solution algorithms for the two main sub-problems of the Dynamic Traffic Assignment Problem. The DNLP algorithm presented in Chapter 2 enables us to calculate time-dependent arc travel times from time-dependent path flow rate functions. The Dynamic Shortest Paths algorithms presented in Section 3.1 enable us to use these arc travel time functions to obtain time-dependent minimum time paths.

In this section we describe the tasks that must be completed by a DTA algorithm to obtain a DTA solution. We also discuss how each step may be implemented for the case of the continuous-time problem with stepwise O-D demand functions and affine arc performance functions. Finally, we present a DTA algorithm and discuss the results obtained from its implementation.

The first task of the DTA algorithm is to manage calls to the subroutines for the DNLP and Dynamic Shortest Paths subproblems. This includes using these subroutines to generate an initial assignment, and later calling the subroutines for subsequent, revised sets of path flows. In our implementation we generate an initial set of path flows using a static network in which the network is empty and arc travel times are equal to the constant term of the arc performance function. We use this network to determine shortest paths, then perform an deterministic, user-optimum assignment of our time-dependent O-D flows along these paths. In subsequent iterations, the DTA algorithm calls the DNLP subroutine for a given set of path flows. It then uses the output of this subroutine as input to the Dynamic Shortest Paths subroutine.

Another task of the DTA algorithm is to load flows along shortest paths. In our implementation, this is accomplished by iterating over the shortest paths for each O-D pair. For a given O-D pair, the shortest path varies throughout the analysis period. For some path p which is a shortest path between some O-D pair (r,s) on interval $[a,b)$, we assign the O-D flow during $[a,b)$ to p. Performing this assignment for all O-D pairs and

for all time intervals yields a set of path flows $g_p(t)$. This set of path flows is then averaged with that of the previous iteration.

A third task of the DTA algorithm is to compare path flows yielded in successive iterations. If two sets of path flows are sufficiently similar, we conclude that a solution to the DTA problem has been found and terminate the DTA algorithm. If not, we perform another iteration of the DTA algorithm. In order to compare two successive iterations of the DTA algorithm, we determine the absolute value of the area between path flow rate functions for some path p yielded by successive DTA iterations,

$$\sum_{p \in P} \left(\int_{T_s} \left| f_p^{k+1}(t) - f_p^k(t) \right| dt \right) \leq \varepsilon$$

If the value of this area is below some threshold, ε for all paths p, we conclude that the solutions are sufficiently similar.

If successive assignments are not sufficiently similar, we wish to generate a new assignment. In our implementation, this is accomplished by using the Method of Successive Averages to average path flows. These averaged path flows can then be used by the DNLP and Dynamic Shortest paths subroutines to generate a new revised set of path flows. If we denote by $f_p^k(t)$ the path flow at the kth iteration, and $g_p^k(t)$ the path flow obtained by loading along shortest paths, averaged path flow is:

$$f_p^{k+1}(t) = \frac{g_p^k(t)}{k} + \frac{k-1}{k} f_p^k(t)$$

where $f_p^{k+1}(t)$ is the flow assigned in the current iteration.

3.2.1 The DTA Algorithm

The following is the DTA algorithm. Given a network with time-dependent origin-destination travel demands, we first execute a static shortest paths algorithm and assign these demands to the static shortest paths . This assignment provides the basis for our

first execution of the DNLP algorithm, which calculates time-dependent arc travel times from path flows. We then use a dynamic shortest paths algorithm and load flows on these shortest paths. We average this solution with our initial solution to obtain a revised set of path flows, which is denoted in the algorithm below by Steps 2 through 5. Step 5 illustrates how the flow averaging is performed and gives the algorithm's termination criterion.

The DTA Algorithm

Step 0: Compute Static Shortest Paths

Step 1: Assign O-D demands to shortest paths $\rightarrow f_p^k(t)$

Step 2: Compute piecewise linear arc travel times using DNLP algorithm

Step 3: For all origins, compute dynamic shortest paths

Step 4: Load on shortest paths $\rightarrow g_p^k(t)$

Step 5: (Method of Successive Averages)

$$f_p^{k+1}(t) = \frac{g_p^k(t)}{k} + \frac{k-1}{k} f_p^k(t)$$

$$\text{If } \left(\sum_{p \in P} \left(\int_{T_s} \left(\left| f_p^{k+1}(t) - f_p^k(t) \right| \right) dt \right) \leq \varepsilon \right), \text{ STOP}$$

$k=k+1$;

Return to Step 2;

3.2.2 Some DTA Numerical Results

The above DTA algorithm was implemented in Java, and tested on the sample network presented in Chapter 2. This implementation used the previously developed code for the DNLP and DSP algorithms as subroutines. The DTA implementation then managed the calls to these subroutines and assigned flows to shortest paths.

The assignment of O-D flows to shortest paths was performed by looping over the breakpoints of the shortest path travel time function for each O-D pair. For each

breakpoint, the relevant shortest path was determined using the records of time-dependent predecessor nodes. Flows were then assigned to this shortest path according to the original O-D demand function. The resulting path entrance flow rate function is denoted by $g_p^k(t)$ in the algorithm above. Figures 11-16 show assignments of the flow between nodes 1 and 9 on paths 1 through 6 in iteration 8 of the algorithm. Each figure shows the travel time on a given path, compared with the shortest path travel time between the origin and destination nodes. In areas in which the path travel time is equal to the shortest path travel time, an assignment of O-D flow to the given path is shown. The reader may note that the sum of these path flow rate functions is equal to the total O-D flow rate function shown in Figure 17.

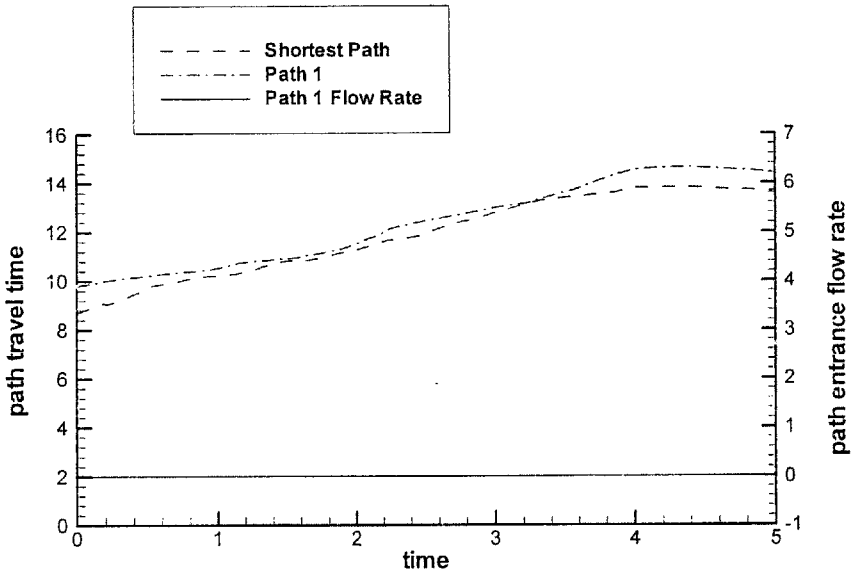


Figure 11: Loading on Shortest Paths

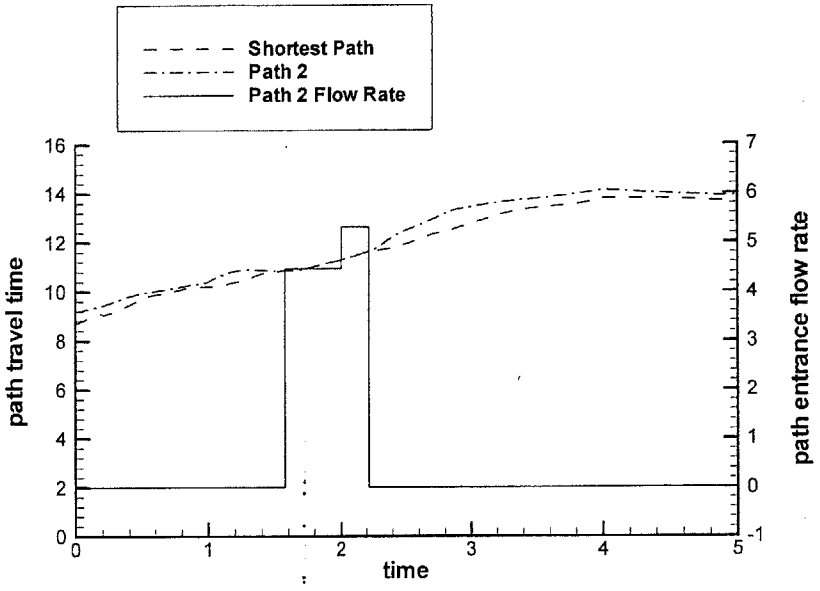


Figure 12: Loading on Shortest Paths

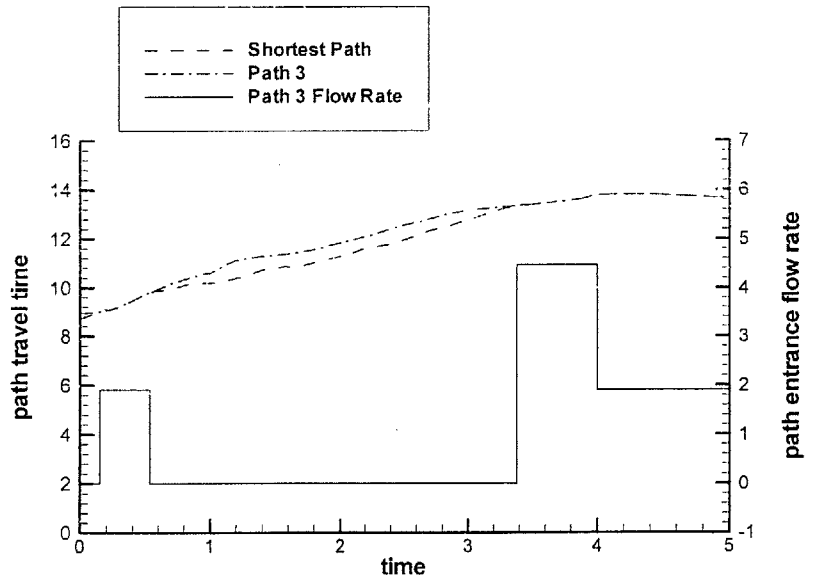


Figure 13: Loading on Shortest Paths

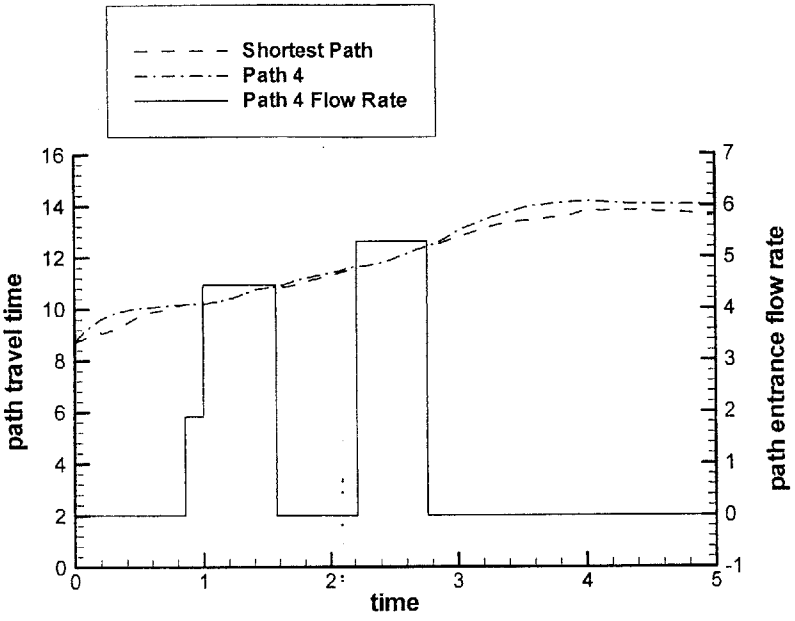


Figure 14: Loading on Shortest Paths

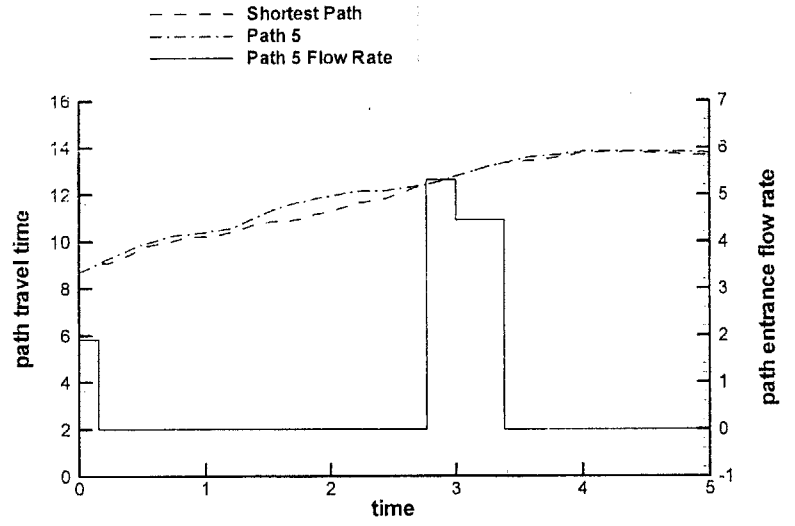


Figure 15: Loading on Shortest Paths

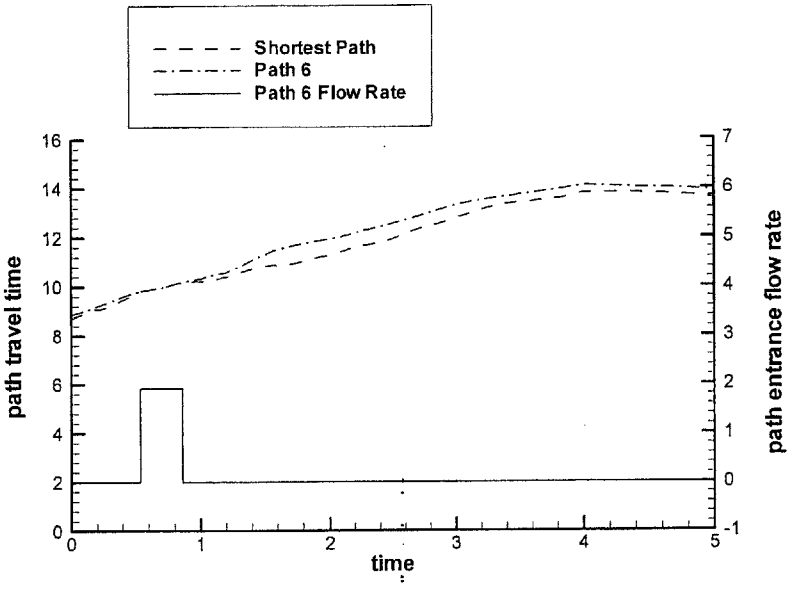


Figure 16: Loading on Shortest Paths

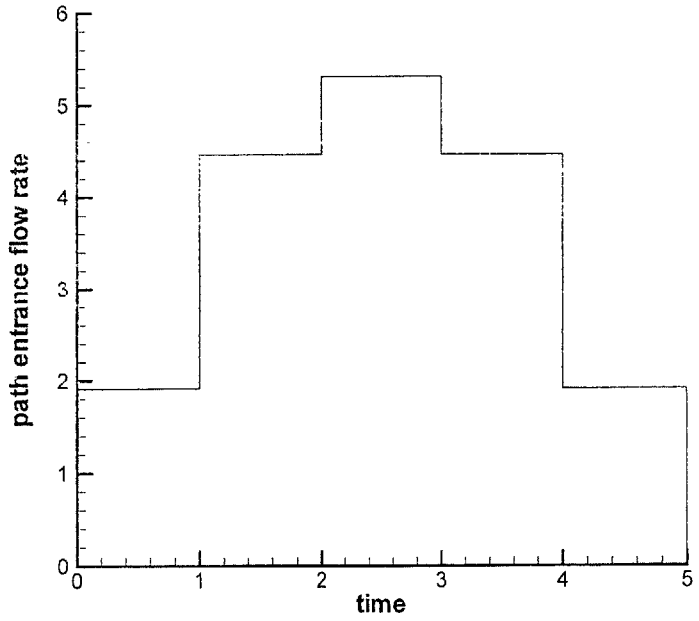


Figure 17: O-D Demand on Between Nodes 1 and 9

The above path flow rate functions were then averaged with the path flow rate function of the previous iteration according to Step 5 of the DTA algorithm to obtain the value of the

path flow rate function for the current iteration. Figure 18 and 19 and show the results of this averaging for path 4 for the first 9 iterations. We note that in the initial assignment, all flow between nodes 1 and 9 is carried on the static shortest path, which is path 4. Thus, path 4's entrance flow rate function is equal to the O-D demand function. In subsequent iterations, the flow is distributed among the other paths between nodes 1 and 9 and the entrance flow rate on path 4 is decreased.

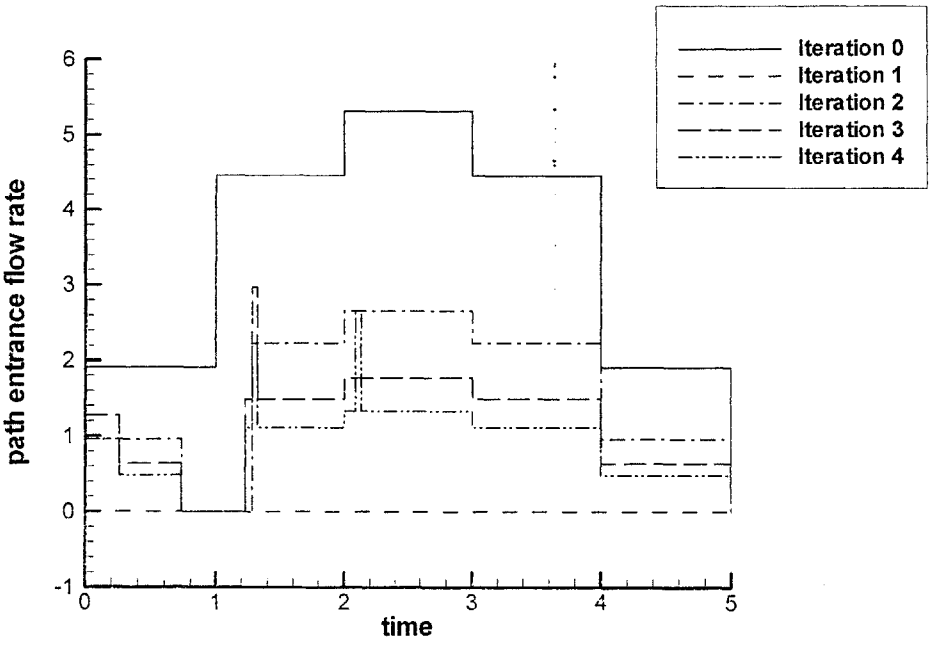


Figure 18: Path 3 Entrance Flow Rate

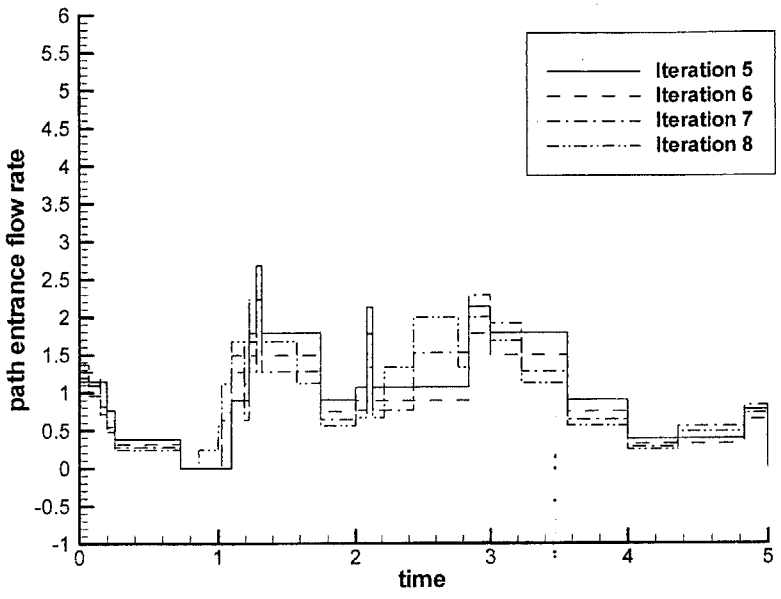


Figure 19: Path 3 Entrance Flow Rate

From Figures 18 and 19 we note that the trend in the difference between the entrance flow rate functions in subsequent iterations is decreasing. While it is not proven that the flows obtained using the MSA method converge to a solution, we can illustrate that in general this difference decreases. Figure 20 shows the value of the difference between successive assignments on all paths in the network as a function of the value of k. The dependent variable is calculated by determining the absolute value of the difference between two assignments for each path and for all times in the analysis period according to:

$$\sum_{p \in P} \left(\int_{T_\infty} \left(\left| f_p^{k+1}(t) - f_p^k(t) \right| \right) dt \right) = \delta$$

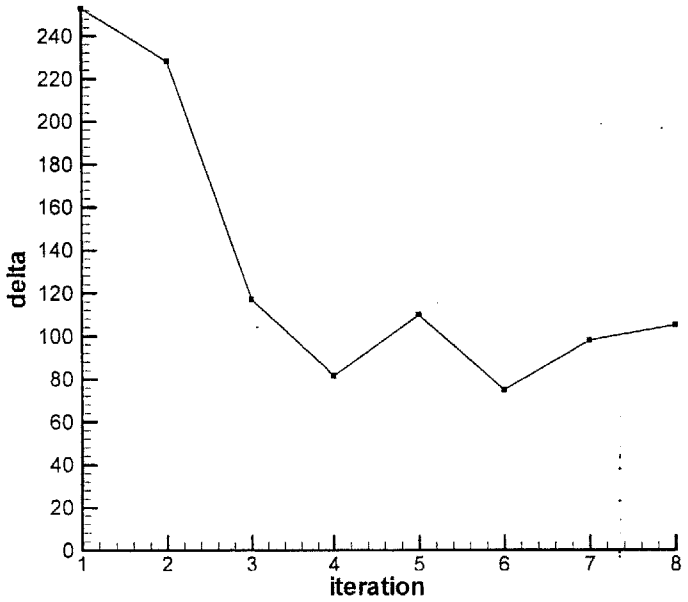


Figure 20: Difference Between Successive Assignments as a Function of the Iteration Number

From Figure 18 and Figure 19 we can also draw some conclusions about the formation of breakpoints in the DTA. We note that the original O-D demand function contains only five breakpoints, but that by iteration 9, the number of breakpoints in Path 4's entrance flow rate function is much more numerous. We note that this is a result of both the formation of breakpoints in the DNLP algorithm, and the formation of breakpoints in the DSP algorithm. Furthermore, we note that the averaging operation performed in the DTA algorithm provides further opportunities to increase the number of breakpoints in the path entrance flow rate functions. Given this, it should not be surprising that the DTA algorithm ran more slowly as the iteration index increases. While a detailed study of the empirical run times of any of the algorithms contained in this thesis has not yet been made, it is reasonable to conjecture that the major limitation of these continuous-time methods would be the relatively high value of the number of breakpoints created as iterations increase.

3.3 Conclusions

To conclude this chapter, we note that we have provided formulations and implementations of algorithms which permit solution of the Dynamic Traffic Assignment Problem in continuous time. Results of these algorithms indicate their implementations to be correct. We note that for a small example network, the difference between successive iterations of the DTA algorithm seems to be decreasing, but that further iterations of the algorithm will have increasingly long run-times due to the increasing number of breakpoints in the network's functions. Further discussion of this topic is given in the concluding chapter.

CHAPTER 4

CONCLUSIONS AND FURTHER DIRECTIONS OF RESEARCH

The Dynamic Traffic Assignment problem has applications in various transportation contexts, from planning to ITS. In each context the requirements of the DTA model may differ: for some applications the run-time of the implemented model may be critical, while for others the accuracy of the solution may be more important. It is thus important to explore various DTA solutions and establish their theoretical and empirical properties.

While various discrete-time DTA models exist in the literature, we have presented a continuous-time model. Recognizing the computational challenges of implementing continuous-time algorithms, we have considered a particular class of functions for which the theoretical properties yield a relatively simple solution. By exploiting these properties, we have developed an exact, continuous-time Dynamic Network Loading algorithm. To complement this algorithm, we have also presented a continuous-time, dynamic extension of an existing static shortest paths algorithm. Finally, we have presented an MSA-based DTA algorithm which uses the above two algorithms to yield a DTA solution.

Each of these algorithms was implemented and tested on a sample network. This testing allowed us to verify that the solutions given by each algorithm are correct. Further testing of these algorithms should be performed on larger networks and should focus on

determining the run-time of the algorithms on networks of various sizes and with various properties. These empirically-determined run-times should then be compared to those of discrete-time algorithms to determine the computational performance of the algorithms.

The strongest point of these algorithms is that they yield an exact solution, unlike discrete-time methods which approximate functions over time. Thus, in contexts in which solution accuracy is more critical than run-time, this continuous-time model could prove more promising than existing methods. In particular, the algorithm may be effective for benchmarking other DTA algorithms. To do this, existing discrete-time algorithms should be applied to the DTA model of this thesis with stepwise O-D flow rate functions, affine arc performance functions and a deterministic, user-optimum users' behavior model. This will then yield an approximate solution to a model for which we have obtained an exact solution and the differences between these solutions should be analyzed.

In the absence of such testing, we note that our theoretical results indicate that the creation of breakpoints in both of the two sub-algorithms presented may be problematic from an efficiency standpoint. We have provided a fairly loose bound on the number of breakpoints created in the DNLP algorithm and a tighter bound on the number of breakpoints created in the DSP algorithm. We have not yet provided a bound on the number of breakpoints created in the DTA algorithm, but have noted the potential for the number of breakpoints to increase with each execution of these algorithm. Thus, we have concluded that the number of breakpoints in each the network's various functions may increase with each iteration of the DTA algorithm. For this reason, we feel that it is unlikely that this continuous-time model will be practical in real-time contexts. Further research should be undertaken to attempt to tighten the bounds on the number of breakpoints created by each algorithm.

An area of further work that may provide greater generality to our model is to permit input functions to take on additional functional forms. In particular, we believe it is possible to formulate and implement an exact solution to the continuous-time DNLP for

which arc performance functions are piecewise linear. If such a model were developed, the transportation modeler would have greater flexibility in how to model the effect of congestion on travel time.

Another area of future work is to test the algorithm's empirical performance on networks in which the dynamics of the network vary greatly from arc to arc or from time interval to time interval. In networks in which most arcs can be assumed to be static for most time intervals, our continuous-time algorithm will result in a solution that contains few breakpoints and may perform well, whereas discrete-time approaches would be slowed by performing many computations on static arcs. Such networks are important as they reflect the nature of many real world transportation networks.

REFERENCES

D.P. Bertsikas and J. N. Tsitsiklis (1989). *Parallel and Distributed Computation: Numerical Methods.* Prentice Hall, Englewood Cliffs, N.J.

I. Chabini (1997). A New Shortest Paths Algorithm for Discrete Time Dynamic Networks. The proceedings of the 8th IFAC Symposium on Transport Systems, pp. 551-556.

I. Chabini (1998). Discrete Dynamic Shortest Path Problems in Transportation Applications: Complexity and Algorithms with Optimal Run-time. Transportation Research Record, pp. 170-175.

I. Chabini and B. Dean (1998). Shortest path Problems in Discrete-Time Dynamic Networks: Complexity, Algorithms, and Implementations. Internal Report.

I. Chabini and Y. He (2000). Analytical Methods for Dynamic Traffic Assignment. Accepted for publication in International Transactions in Operations Research.

I. Chabini and S. Kachani (1999). An Analytical Dynamic Network Loading Model: Formulation, Analysis and Solution Algorithms. Internal Report (Submitted to *Transportation Science).*

I. Chabini and V. Yadappanavar (2000). Advances in Discrete-Time Dynamic Data Representation with Communication and Computation Transportation Applications. Internal Report.

B. Dean (1999). Continuous-Time Shortest Path Algorithms. Masters Thesis, Massachusetts Institute of Technology. (Supervisor: Ismail Chabini).

Y. He (1997). A Flow-Based Approach to the Dynamic Traffic Assignment Problem: Formulations, Algorithms and Computer Implementations. Masters Thesis, Massachusetts Institute of Technology. (Supervisor: Ismail Chabini).

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), pp. 607-625.

A. Orda and R. Rom (1991). Minimum Weight Paths in Time-Dependent Networks. *Networks* 21 (3), 295-320.

Y. W. Xu, J. H. Wu, M. Florian, P. Marcotte and D. L. Zhu (1999). Advances in the Continuous Dynamic Network Loading Problem. *Transportation Science*, pp. 341-353.

V. Yadappanavar (2000). Time Dependent Networks: Data Representation Techniques and Shortest Path Algorithms with Applications to Transportation Problem. Masters Thesis, Massachusetts Institute of Technology. (Supervisor: Ismail Chabini).

APPENDIX A - SAMPLE INPUT AND OUTPUT DATA FILES

The Dynamic Network Loading algorithm requires two input data files. The first contains the specification of the O-D pairs, the arcs and the paths; we call this the network data file. The second is a path flow data file and contains the path entrance flow rate functions. The output file of the Dynamic Network Loading algorithm is the arc exit time data file, which is used as an input to the Dynamic Shortest Paths algorithm. The details of each file are given below.

Network Data File

An example of the network data file are shown below. The first section of the file specifies the number of O-D pairs, an index for each given O-D pair, its origin node, destination node and the number of paths between the two nodes. The total number of paths specified thus far is also contained at the end of each line.

In the second section of the file the details of each arc are given. This includes the total number of arcs and, for each arc its index, origin, destination, and the intercept and slope of its travel time function.

The third section of the network data file specifies the paths in the network using the subpath data structure described in Chapter 2 of this thesis. The first line of this section specifies the total number of subpaths. Each subsequent line contains the number of the next arc, next subpath and an index for the given subpath.

```
#number of OD pairs#
7
#index, origin, destination, number of paths, count#
0     1     9     6     0
1     1     5     2     6
2     5     9     2     8
3     1     3     1     10
4     3     9     1     11
5     7     9     1     12
6     1     7     1     13
#number of arcs#
12
#index, origin, destination, slope, intercept#
0     0     3     .235  1.88
1     3     6     .443  1.80
2     0     1     .269  1.39
3     1     4     .443  1.6
4     3     4     .408  1.32
5     6     7     .408  1.92
6     4     7     .443  1.50
7     1     2     .408  1.62
8     4     5     .443  1.80
9     7     8     .408  2.12
10    2     5     .443  2.00
11    5     8     .408  2.32
#number of subpaths#
26
#this arc, next subpath, index#
0     14    0
0     15    1
0     16    2
2     17    3
2     18    4
2     19    5
0     24    6
2     25    7
6     20    8
8     22    9
2     23    10
10    22    11
5     20    12
0     21    13
1     12    14
4     8     15
4     9     16
3     8     17
3     9     18
7     11    19
9     -1    20
1     -1    21
11    -1    22
7     -1    23
4     -1    24
3     -1    25
```

Path Flow Data File

The first line of the path flow data file contains the number of paths in the network and the number of steps in each path. Data for each path are given in rows. Odd-numbered columns give the time of a breakpoint in the path's entrance flow rate function; even-numbered columns give the function's value at that breakpoint. Path flow rate functions for the first five paths are shown below.

```
#number of breakpoints, number of paths#
6     14
#time, value, time, value, time, value, time, value, time, value    ...
0  0.225  0  0.3375  0  0.405  0  0.45  0  0.225      ...
1  0.525  1  0.7875  1  0.945  1  1.05  1  0.525      ...
2  0.625  2  0.9375  2  1.125  2  1.25  2  0.625      ...
3  0.525  3  0.7875  3  0.945  3  1.05  3  0.525      ...
4  0.225  4  0.3375  4  0.405  4  0.45  4  0.225      ...
5  0      5  0       5  0      5  0     5  0          ...
```

Arc Exit Time File

The output of the Dynamic Network Loading algorithm is an arc exit time file, which specifies the piecewise linear arc exit time functions. This file is in a form similar to that of the path flow data file, but instead of specifying the function's value at each breakpoint, the file specifies its slope. The dynamic shortest paths algorithm can then construct each arc's exit time function, given the following two properties of the exit time function. Firstly, we know that at the first breakpoint of an arc's exit time function, the value of the function is equal to the intercept of the arc performance function. This can be obtained from the network data file. Secondly, we know that the arc exit time functions are continuous. Thus we can obtain the function's value at any time in the analysis period using basic algebra.

The first line of the arc exit time file gives the number of arcs in the network. Subsequent lines give the exit time function for each arc. Odd-numbered columns specify the time at which a given breakpoint occurs. Even-numbered columns give the function's slope to the right of the breakpoint. A portion of this file is shown below.

```
#number of arcs, number of breakpoints#
12  525
#time, value,                  time, value...
0.0   1.5181749999999998      0.0   1.0                        ...
1.0   2.2090749999999995      1.8   0.9999999999999972         ...
2.0   2.098060592899369       3.398174999999999  1.4737673007933205 ...
3.0   1.8677605928993675      3.6   1.473767300793323          ...
.     .                        .     .
.     .                        .     .
.     .                        .     .
.     .                        .     .
```

APPENDIX B - METHODS FOR PIECEWISE FUNCTIONS

The algorithms presented in this thesis have relied on the properties of piecewise functions to obtain exact, continuous time solutions to the Dynamic Network Loading, Dynamic Shortest Path and Dynamic Traffic Assignment Problems. In this appendix we explore in further detail the methods that operate on piecewise functions and are used repeatedly in the execution of these algorithms. They are:

- a method to integrate a stepwise function;
- a method to sum two stepwise linear functions;
- a method to construct the inverse of a strictly increasing, piecewise linear function;
- a method to take the minimum of two piecewise linear functions; and
- a method to compose two piecewise linear functions.

Below, we discuss each of these methods and present pseudocode for its implementation. Each method assumes that stepwise functions are represented as $\forall x \in [a,b)$ $f(x) = \lim_{x \to b} f(x)$.

Integral of a Stepwise Function

Since a stepwise function can be though of as sequence of rectangular blocks, it is straightforward to obtain its exact integral. To do so, we iterate over the breakpoints of the function, adding the area under the curve and between the previous and current breakpoint. Consider $g(x)$ to be a stepwise function and G to be the set of breakpoints

in $g(x)$. Each breakpoint is characterized by its value of x and we denote by $\min(G)$ the breakpoint of G with the least value of x. Our method is then:

$sum = 0;$

$t_0 = \min(G);$

$G = G \setminus \{t_0\};$

while $G \neq \emptyset$

$t = \min(G);$

$sum = sum + (t - t_0) \times g(t_0)$

$t_0 = t;$

end while

return $sum;$

Value of the Inverse of Strictly Increasing Piecewise Linear Functions

We first note that our DNLP algorithm does not require that we determine the inverse of a piecewise function at all points in the interval, but rather that we find its value at a single point. Thus, for ease and efficiency, we present a method that returns the value of the inverse function that corresponds to a particular value of the original function. We denote the original function by $g(x)$ and denote by G the set of breakpoints of $g(x)$, each of which is characterized by its value of x. Our method also assumes that we know the slope of $g(x)$, $g'(x)$ at each breakpoint (that is, the slope of $g(x)$ to the right of x). We define the method "min" to return the time at which the next breakpoint occurs. The method below returns the value of $g^{-1}(\cdot)$ at some point, t.

$x_1 = \min(G);$

while $g(x) \leq t$

if $g(x_1) = t$ *then return x and stop;*

$G = G \setminus \{x_1\}$;

$x_0 = x_1$

$x_1 = \min(G)$;

end while;

$\Delta y = t - g(x_0)$;

$slope = g'(x_0)$;

$\Delta x = \Delta y / slope$;

return $x_0 + \Delta x$;

We note that the above method can operate on strictly increasing functions only. The inverse of a function with a slope of zero at any time within the interval of interest is not well-defined and is therefore not considered here.

Sum of Two Stepwise Functions

This method is of use in the DNLP algorithm in which it is used to calculate the arc entrance flow rate function from arc-path entrance flow rate functions. We consider two stepwise functions $f(x)$ and $g(x)$, each having a set of breakpoints, F and G. These sets contain the x coordinates of the breakpoints in F and G. Our method returns a third stepwise function, $a(x)$ with a set of breakpoints, A. We note that a breakpoint in either $f(x)$ or $g(x)$ corresponds to a breakpoint in $a(x)$, thus our method iterates over the breakpoints of the input functions to determine the breakpoints of the output function. To do so, the method keeps a record of the current breakpoint of each of the functions, f_1 and g_1. It also keeps a record of the smaller of these two breakpoints, x_1. At each iteration, the method uses the values of the functions at these breakpoints to calculate the

slope and value of the output function. The method continues until it reaches some specified endpoint, *end*. The method follows.

$f_1 = \min(F),\ g_1 = \min(G);$

$F = F \setminus \{f_1\},\ G = G \setminus \{g_1\};$

$y_1 = f(f_1) + g(g_1);$

$x_1 = \min(f_1, g_1)$

while$(x_1 < end)$

$A = A \cup \{x_1\},\ a(x_1) = y_1,\ a'(x_1) = slope;$

if$(x_1 \le f_1)$*then* $f_1 = \min(F),\ F = F \setminus \{f_1\},\ x_2 = \min(g_1, F, G);$

if$(x_1 < g_1)$*then* $g_1 = \min(G),\ G = G \setminus \{g_1\},\ x_2 = \min(f_1, F, G);$

$x_1 = \min(f_1, g_1)$

$y_1 = f(x_1) + g(x_1);$

if$(x_1 = x_2), x_2 = \min(F, G);$

return $A, a, a';$

We assume that both functions are defined for the same time interval. The above method returns the set of breakpoints in the function $a(x)$, as well as the functions value and slope at these points. These data completely specify the function at any time in the analysis period.

Minimum of Two Piecewise Linear Functions

In the continuous dynamic shortest paths problem, it necessary to determine the shortest path from a given node for all departure times. To do so, we make multiple pair wise comparisons of travel time functions, generating the minimum function for each pair. When these functions are piecewise linear, it is possible to compute this minimum function by iterating over the function's breakpoints. The following method operates in a

manner similar to the method to sum piecewise functions. We note that in finding the minimum function of two piecewise linear functions, it is possible to fully specify the function by finding its value at, and slope to the right of its breakpoints. These breakpoints are located at the breakpoints of the input functions, or at additional breakpoints created when the input functions intersect.

We again consider two piecewise linear functions $f(x)$ and $g(x)$, each having a set of breakpoints, F and G. We denote by M the set of breakpoints of the minimum function $m(x)$. At each iteration we determine whether the next breakpoint of $m(x)$ will be due to a breakpoint in an input function or due to the intersection of the two input functions. This is done by computing the value of x at which the functions intersect. We call this value *cross*. We assume that both functions are defined for the same time interval. The method follows.

$f_1 = \min(F)$, $g_1 = \min(G)$;

$x_1 = \min(f_1, g_1)$;

$if(x_1 = f_1)\, then\, x_2 = \min(g_1, F, G)$;

$else\, x_2 = \min(f_1, F, G)$;

$while(x_1 < end)$

$M = M \cup \{x_1\}$;

$m(x_1) = \min(f(x_1), g(x_1))$;

$if(f(x_1) = g(x_1), then,\ m'(x_1) = \min(f'(x_1), g'(x_2))$;

$else\ if(m(x_1) = f(x_1))then,\ m'(x_1) = f'(x_1)$;

$else\ if(m(x_1) = g(x_1))then,\ m'(x_1) = g'(x_1)$;

$cross = (g(x_1) - f(x_1)) \Big/ (f'(x_1) - g'(x_1))$;

$if(cross > 0\ and\ (x_1 + cross) < x_2)$

$x_1 = x_1 + cross$;

$else\ x_1 = x_2$;

$if(x_1 > f_1)then\ f_1 = \min(F),\ F = F - f_1,\ x_2 = \min(g_1, F, G)$

$if(x_1 > g_1)then\ g_1 = \min(G),\ G = G - g_1,\ x_2 = \min(f_1, F, G)$

$if(x_1 = x_2), x_2 = \min(F, G);$

$return\ M, m, m';$

Composition of Increasing Piecewise Linear Functions

According to the statement of the dynamic shortest paths algorithm it is necessary to obtain the quantity $a_{ij}(a_i(t))$ where $a_{ij}(t)$ and $a_i(t)$ are piecewise linear functions. To do so, we develop a method that composes two piecewise linear functions, $f(\cdot)$ and $g(\cdot)$ to obtain $g(f(\cdot))$. We denote by $c(\cdot)$ the resulting piecewise linear function, and by F, G and C the sets of breakpoints of the functions $f(\cdot)$, $g(\cdot)$ and $c(\cdot)$, respectively. Our algorithm proceeds by moving forward on the x-axis of $f(\cdot)$. At each loop it selects the next breakpoint of either input function and updates the slope and value of $c(\cdot)$ accordingly. The location of each breakpoint in the composed function corresponds to either the location of a breakpoint in the function $f(\cdot)$, or to the projection of a breakpoint of function $g(\cdot)$ on the x-axis of function $f(\cdot)$. Slope values for the composed function are calculated by multiplying the appropriate values of the slopes of $f(\cdot)$ and $g(\cdot)$. Since we know the function $c(\cdot)$ to be continuous, we can calculate the value of $c(\cdot)$ by basic algebra.

$C = \emptyset;$

$time_f = \min(F);$

$slope_f = f'(time_f);$

$value_f = f(time_f);$

$F = F \setminus \{time_f\};$

$time_g = \min(G)\ s.t.\ time_g > value_f;$

$slope_g = g'(value_f) \times f'(time_f)$;

$value_g = g(value_f)$;

$C = C \cup \{time_f\}$;

$c'(time_f) = slope_g$;

$c(time_f) = value_g$;

while $(time_f < end)$

if $(time_f < f^{-1}(time_g))$

$slope_g = g'(value_f) \times f'(time_f)$;

$value_g = g(value_f)$;

$C = C \cup \{time_f\}$;

$c'(time_f) = slope_g$;

$c(time_f) = value_g$;

$time_f = \min(F)$;

$slope_f = f'(time_f)$;

$value_f = f(time_f)$;

$F = F \setminus \{time_f\}$;

else

$slope_g = g'(time_g) \times f'(f^{-1}(time_g))$;

$ctime = \max(C)$;

$cslope = c'(ctime)$;

$cvalue = c(ctime)$;

$value_g = (f^{-1}(time_g) - ctime) \times cslope + cvalue$;

$C = C \cup \{f^{-1}(time_g)\}$;

$c'(f^{-1}(time_g)) = slope_g$;

$c(f^{-1}(time_g)) = value_g$

$$time_g = \min(G)\, s.t. time_g > time_g ;$$

$return\ c ;$

APPENDIX C - SELECTION OF ARC PERFORMANCE FUNCTIONS

In order to compare the results of our DNLP implementation with those of previous implementations, we wished to select arc performance functions which approximate those used in examples in the literature. Xu et al. (1999) use arc performance functions of the form:

$$D_a(X_a(t)) = \alpha_{a1} + \alpha_{a2} X_a(t) + \alpha_{a3} X_a^2(t)$$

where α_{a1}, α_{a2}, and α_{a3} are the parameters of each arc. Since our algorithm requires the function $D_a(X_a(t))$ to be affine, we wish to determine a linear function of α_{a1}, α_{a2}, and α_{a3} that approximates the above function. The values of α_{a1}, α_{a2}, and α_{a3} used by Xu et al. are given in the following table.

Arc	α_{a1}	α_{a2}	α_{a3}
1	2.0	0.2	0.0138889
2	2.0	0.2	0.038889
3	1.5	0.2	0.011111
4	1.8	0.2	0.038889
5	1.5	0.2	0.033333
6	2.1	0.2	0.033333
7	1.7	0.2	0.038889
8	1.8	0.2	0.033333
9	2.0	0.2	0.038889
10	2.3	0.2	0.033333
11	2.2	0.2	0.038889
12	2.5	0.2	0.033333

As it was observed that the value of total flow on each arc most often falls within the interval (0,5) (Xu et al. 1999), we are most concerned with the goodness of our approximation for values within this interval. Since 2.5 is the middle value of this interval, we estimated the value of the term $\alpha_{a3}X_a^2(t)$ to be equal to:

$$\alpha_{a3}X_a(t)\times 2.5\,.$$

This yielded the affine function:

$$D_a(X_a(t)) = \alpha_{a1} + (\alpha_{a2} + 2.5\alpha_{a3})X_a(t)\,.$$

Graphing the above function and comparing it with the original function showed that in general it slightly overestimated the value of $D_a(X_a(t))$. To compensate for this, we decreased the intercept of the function slightly, subtracting $\alpha_{a3}^{0.5}$. This yielded the function:

$$D_a(X_a(t)) = (\alpha_{a1} - \alpha_{a3}^{0.5}) + (\alpha_{a2} + 2.5\alpha_{a3})X_a(t)\,.$$

This function was found to be a good approximation to the original arc performance function of each arc. The following graphs compare the value of the above affine function with that of the original function for values of $X_a(t)$ within the interval $(0,5)$. For brevity, only the functions of several arcs are shown: the functions of the other arcs are similar.

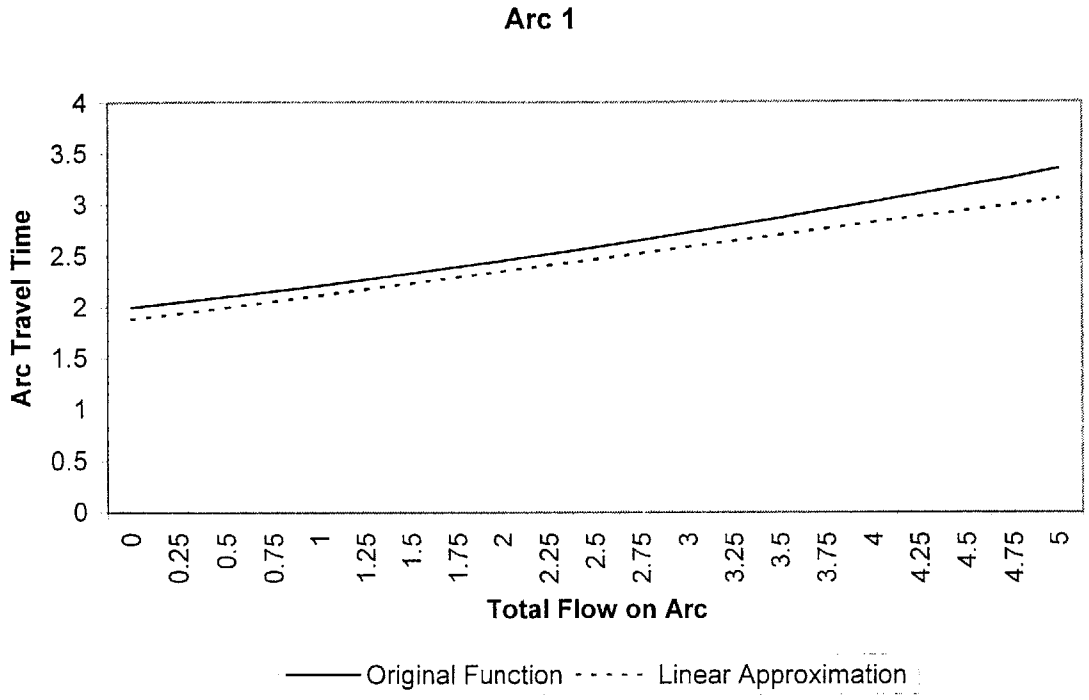


Figure 21: Linear Approximation of the Performance Function of Arc 1

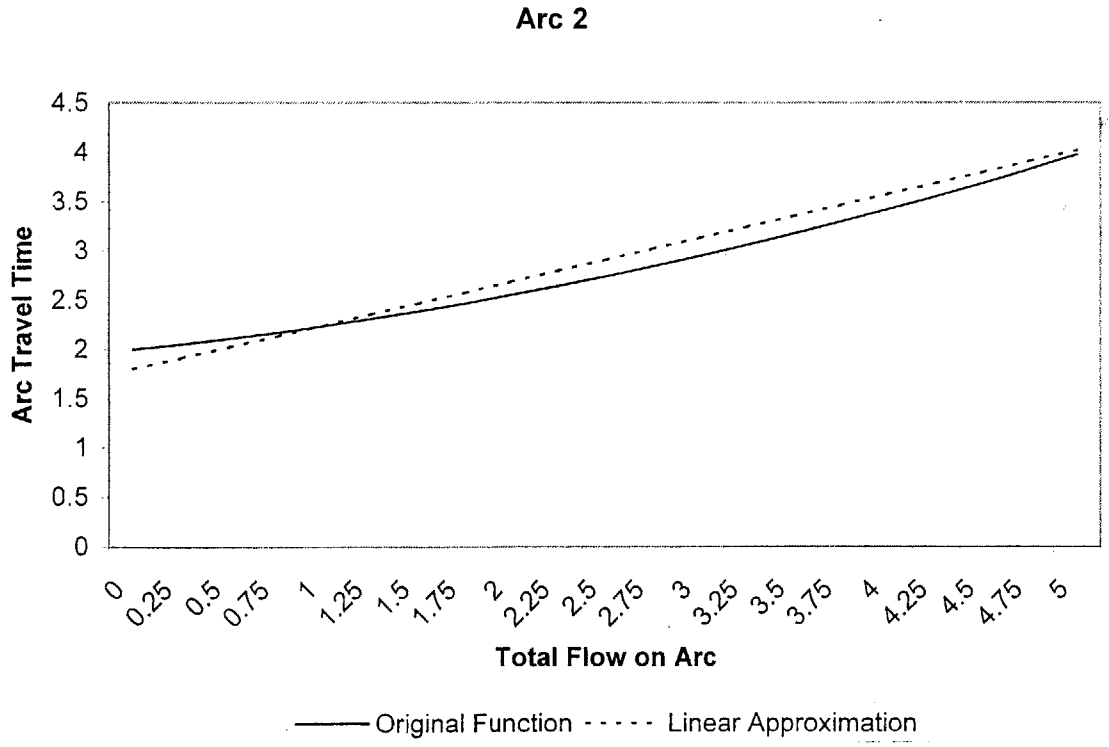


Figure 22: Linear Approximation of the Performance Function of Arc 2

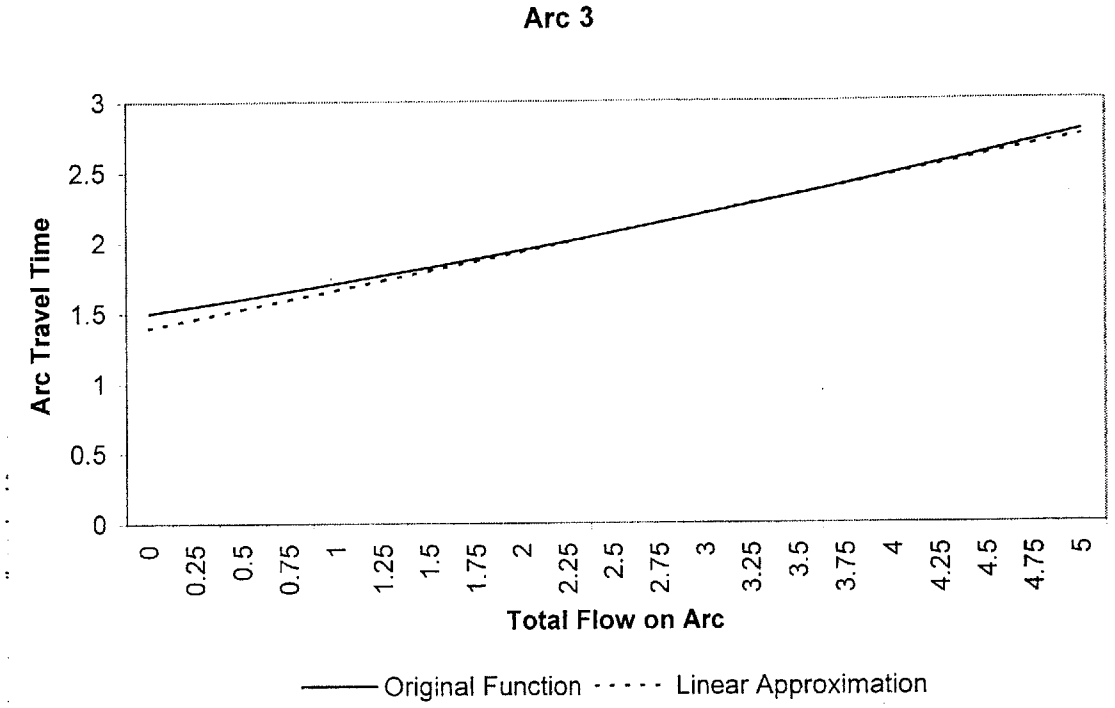


Figure 23: Linear Approximation of the Performance Function of Arc 3

The slope and intercept values resulting from the linear approximation of the original functions are shown in the table below. These values specify the arc performance functions used in our example.

Table 5: Parameters of Affine Arc Performance Functions

Arc	Intercept	Slope
1	1.88	0.235
2	1.80	0.443
3	1.39	0.269
4	1.60	0.443
5	1.32	0.408
6	1.92	0.408
7	1.50	0.443
8	1.62	0.408
9	1.80	0.443
10	2.12	0.408
11	2.00	0.443
12	2.32	0.408

APPENDIX D - DNLP RESULTS

Here we give the arc total flow functions resulting from testing the DNLP algorithm on our small example described in Chapter 2. These figures are included to assist the reader in viewing the DNLP results for a particular arc.

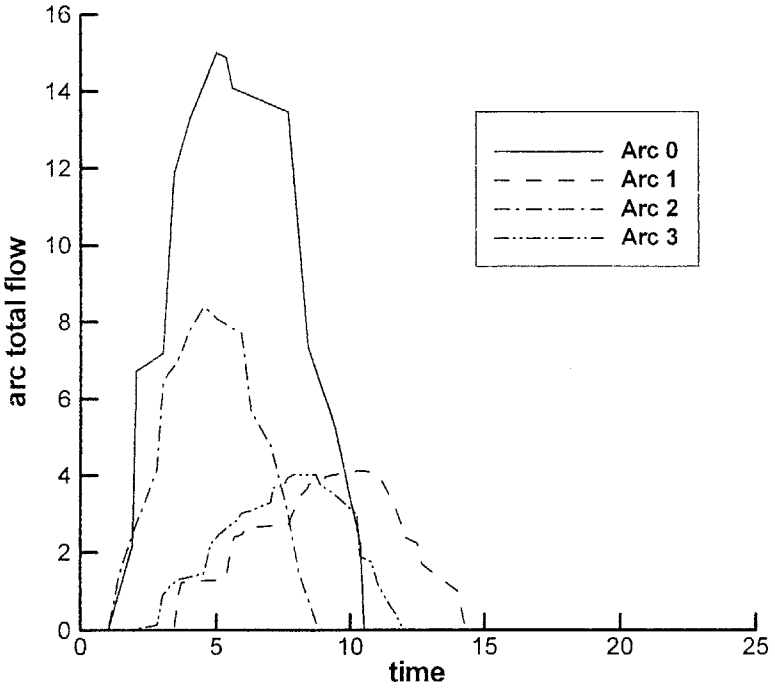


Figure 24: Arc Total Flow

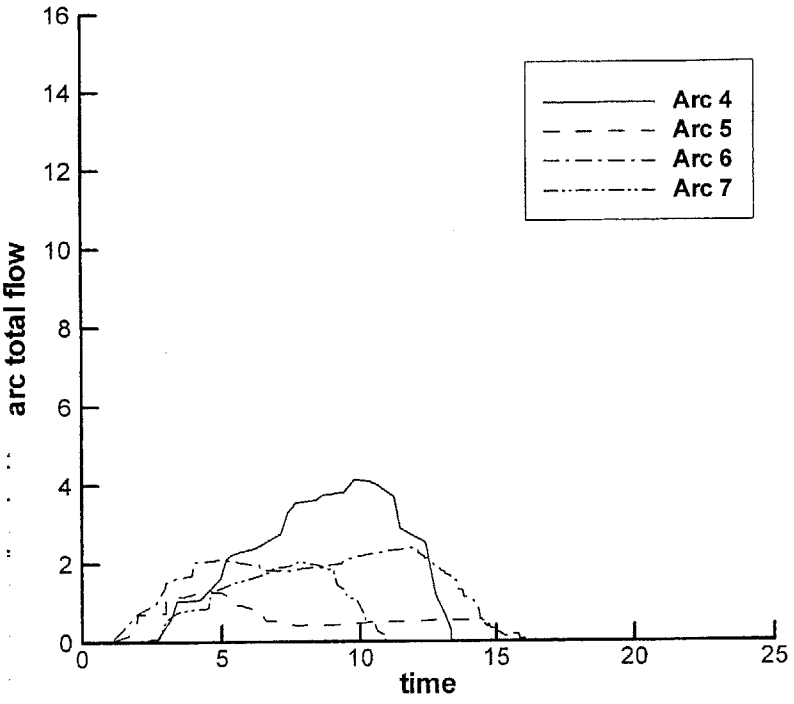


Figure 25: Arc Total Flow

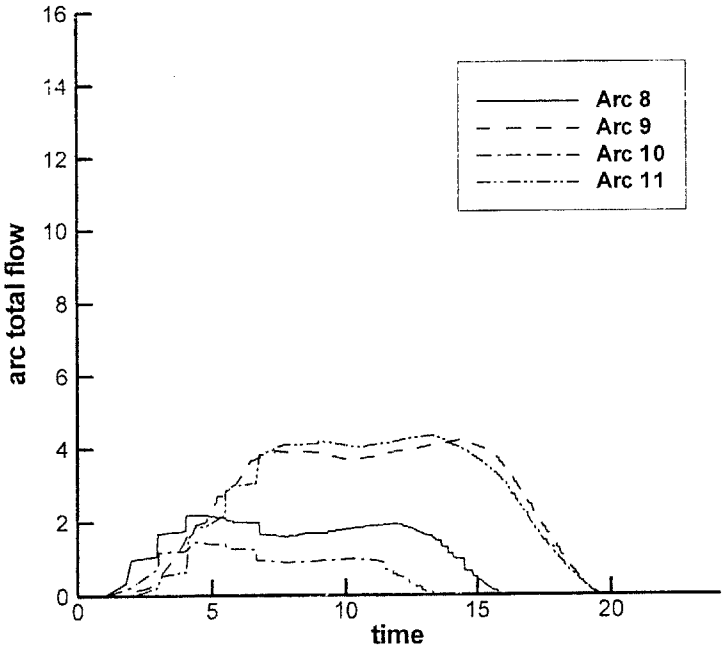


Figure 26: Arc Total Flow