Shortest Paths in FIFO Time-Dependent Networks: Theory and Algorithms

Brian C. Dean
Massachusetts Institute Of Technology

Abstract

We present a concise study of the time-dependent shortest path problem, its theoretical properties, and its solution algorithms. Time-dependent networks, in which the travel time along each arc is a known function of the departure time along the arc, arise in many practical applications, particularly those related to vehicular transportation. Since the general problem is at least NP-hard, we focus entirely on the case of FIFO networks, in which commodities travel through arcs in a First-In-First-Out manner. This special case is very common in practice and enables the development of rich theoretical properties and efficient solution algorithms. Our aim is to present a unified framework which encompasses a wide range of problem variants in both discrete and continuous time, which ties together past work and recent developments.

1 Introduction

The static shortest path problem is one of the most studied problems in algorithmic graph theory. In reality, however, many networks tend to have dynamic characteristics which require more sophisticated approaches for computing shortest paths. There are two common types of dynamic shortest path problems: in the first, one must recompute shortest paths due to frequent, instantaneous, and unpredictable changes in network data. This is essentially the *reoptimization* problem of solving a series of closely-related static shortest path problems. The second type, and the study of this paper, is the *time-dependent shortest path problem*, in which network characteristics change with time in a predictable fashion. Such problems arise frequently in vehicular transportation; the shortest path computed from a snapshot of the network data at the current time may not be optimal if one considers predictable future changes in arc travel times which are bound to occur as one travels through the network, particularly around times such as “rush hour”. In the time-dependent shortest path problem, we assume that the travel time along each arc is a function of the departure time along the arc, and that all such functions are known in advance over all time. The problem initially dates back to 1966, when it was first proposed in discrete time by Cooke and Halsey [5].

The general time-dependent shortest path problem is at least NP-Hard since it may be used to solve a variety of NP-Hard optimization problems such as the knapsack problem. However, depending on how one defines the problem, it may not be in NP since its output is not polynomially bounded; moreover, as shown by Orda and Rom [13] there are even continuous-time instances in which shortest paths consist of an infinite sequence of arcs. In this paper, we study a special class of networks known as *FIFO networks*, in which commodities travel along arcs in a First-In-First-Out manner. In practice many networks, particularly transportation networks, exhibit FIFO behavior. Under the FIFO assumption, time-dependent shortest path problems exhibit many nice structural properties that enable the development of efficient polynomial-time solution algorithms. Our goal in this discussion is to study the theoretical properties and solution algorithms of time-dependent shortest path problems within a simple framework which unifies previous results in both continuous and discrete time. Within this framework, we consider a wide range of problem variants.

The remainder of this paper is structured as follows. The first two sections are devoted to describing the time-dependent shortest path problem and its variants. Time-dependent shortest path problems are plagued by an abundance of problem variants, making a concise problem description rather difficult. We show how to reduce the space of problem variants down to two fundamental variants, and we proceed to give properties and solution algorithms for these two variants. Finally, we briefly discuss results for the non-FIFO case within the context of the results in this paper.

2 Notation and Basic Concepts

The input to a time-dependent shortest path problem is a directed network $G = (N, A)$ and an *arrival time function* $a_{ij}(t)$ for every arc $(i, j) \in A$. The quantity $a_{ij}(t)$ gives the time of arrival at j if one departs from i at time t. It is assumed that $a_{ij}(t) \geq t$. The function $a_{ij}(t) - t$ gives the travel time along arc (i, j) if one departs at time t. Most previous papers work in terms of these arc travel time functions; however, we use arc arrival time functions as this usually leads to simpler and more natural formulations.

If $a_{ij}(t)$ is non-decreasing for all $(i, j) \in A$, we say that G is a *FIFO network*, since in this case commodities will travel through arcs in a First-In-First-Out manner. We henceforth assume that G is a FIFO network. For simplicity, we also assume that $m = \Omega(n)$, where $n = |N|$ and $m = |A|$.

2.1 Discrete vs. Continuous Time

All results in this paper apply equally well if one takes time to be either integer-valued or real-valued. In a discrete-time setting, the a_{ij}'s are integer-valued functions of an integer argument. In continuous time, arbitrary real-valued functions of a real argument are allowed, although we pay particular attention to the case where arc arrival times are piecewise linear functions; we will later see that this greatly simplifies the structure of the output. It is certainly possible to approximate "well-behaved" continuous-time functions as piecewise linear functions; however, a rigorous treatment of this topic is beyond the scope of this paper.

In continuous time, it is assumed that $a_{ij}(t) = \lim_{\tau \to t^+} a_{ij}(\tau)$ in order to avoid difficulties with special cases at points of discontinuity. When dealing with piecewise linear functions, we let P_{ij} denote the number of pieces in a_{ij}, and we let $P^* = \sum P_{ij}$ be the total number of linear pieces across the entire network. In practice, especially in discrete time, one typically focusses only a finite-length window of time from $t = 0$ until a "planning horizon" at $t = T$, where T denotes the length of the window.

2.2 Derived Time-Dependent Quantities

Based on the arc arrival time functions a_{ij}, one can derive several useful time-dependent quantities. We begin with the set of *inverse arc arrival time functions*, $a_{ij}^{-1}(t)$. Since the a_{ij}'s are non-decreasing rather than strictly increasing, they may not be invertible. This difficulty is particularly hard to avoid in discrete time, whereas in continuous time one may wish to conveniently assume strictly increasing functions, yielding what is known as a *strictly FIFO* network. To achieve the greatest generality, we adopt the following natural definition for the inverse of a non-decreasing function $f(t)$:

$$f^{-1}(t) = \sup \{\tau \, : \, f(\tau) \leq t\}.$$

Although this definition does not give us $f(f^{-1}((t)) = t$, it does imply that $(f^{-1})^{-1}(t) = f(t)$ and that f^{-1} is non-decreasing. For a strictly increasing function, the above definition yields its true inverse. Intuitively, $a_{ij}^{-1}(t)$ gives the latest time one may depart i in order to arrive at j by time t by traveling along the arc (i, j). It is always true that $a_{ij}^{-1}(t) \leq t$.

The *path arrival time function* of a path $p = i_1 - i_2 - \ldots - i_k$ is given by the composition of the arc arrival time functions along p:

$$a_p(t) = a_{i_{k-1} i_k} \left(a_{i_{k-2} i_{k-1}} (\ \cdots \ a_{i_1 i_2}(t)) \right).$$

Since they are compositions of non-decreasing functions, path arrival time functions are also non-decreasing. We also have $a_p(t) \geq t$ for all paths p.

Desired Output	Method of Computation
$EA_{s*}(t)$, $EA_{s*}(*)$	These are our two fundamental problems, as which we can express all other variants.
$EA_{sd}(t)$, $EA_{sd}(*)$	As with the static shortest path problem, the single-source single-destination problem seems just as difficult as problems involving either multiple sources or multiple destinations. We therefore solve these as the more general problems $EA_{s*}(t)$ and $EA_{s*}(*)$, respectively.
$EA_{**}(t)$, $EA_{**}(*)$	These are solved by performing n computations of $EA_{s*}(t)$ and $EA_{s*}(*)$, respectively, for every $s \in N$.
$EA_{*d}(t)$	One can show that this problem is no easier than computing $EA_{**}(t)$.
$LD_{sd}(*)$, $LD_{s*}(*)$, $LD_{*d}(*)$, $LD_{**}(*)$	These are computed by solving for and inverting the corresponding earliest arrival time functions. For example, $LD_{s*}(*)$ is found by solving for $EA_{s*}(*)$ and taking inverses.
$LD_{s*}(t)$	One can show that this problem is no easier than computing $LD_{**}(t)$.
$LD_{*d}(t)$	Performing a time-reversal transformation on the network transforms this into the problem of computing $EA_{s*}(t)$.
$LD_{sd}(t)$	Solve the more general problem $LD_{*d}(t)$.
$LD_{**}(t)$	Solve $LD_{*d}(t)$ repeatedly for every $d \in N$.
$EA_{*d}(*)$	Performing a time-reversal transformation on the network transforms this into the problem of computing $LD_{s*}(*)$.

Figure 1: Reduction down to two fundamental problem variants.

The *inverse path arrival time function* of a path p, $a_p^{-1}(t)$, may be defined as the inverse of the corresponding path arrival time function or, equivalently, as the composition of the inverse arc arrival time functions along the path. These functions are non-decreasing and satisfy $a_p^{-1}(t) \leq t$. Intuitively, $a_p^{-1}(t)$ gives the latest time one may depart along p in order to reach the end by time t.

Let $P(s,d)$ denote the set of paths between a source node s and a destination node d. The function $EA_{sd}(t) = \min\{a_p(t) : p \in P(s,d)\}$ gives the *earliest arrival time* at node d if one leaves s at time t. Since it is the minimum of non-decreasing functions, this function is non-decreasing; it also satisfies $EA_{sd}(t) \geq t$. Note that $EA_{sd}(t) - t$ provides the length of the time-dependent shortest path from s to d if one departs at time t.

A similar function, $LD_{sd}(t) = \max\{a_p^{-1}(t) : p \in P(s,d)\}$ gives the *latest departure time* one may leave node s in order to arrive at node d by time t. This function is also non-decreasing, and satisfies $LD_{sd}(t) \leq t$. The latest departure time function LD_{sd} and the earliest arrival time function EA_{sd} are inverses of each-other.

3 Problem Variants

The simplest variant of the time-dependent shortest path problem is to compute $EA_{sd}(t)$ for a single source node s, destination node d, and departure time t. Many other variants are possible if one wishes to consider a range of sources, destinations, or departure times. We specify these using a "wildcard" notation: for example, computing $EA_{s*}(*)$ is the problem of finding the shortest paths from a single source node s to all other nodes and for every departure time, and computing $LD_{*d}(t)$ is the problem of finding the latest possible departure time from every node if one wishes to arrive at node d by a particular time t.

The two most general problem variants are the computation of either $EA_{**}(*)$ or $LD_{**}(*)$. All other variants involve computing some subset of this set of functions as output. One should note that this output gives the lengths of shortest paths rather than the paths themselves. As in the static shortest path problem, the actual structure of the shortest paths can always be computed explicitly within an algorithm or derived subsequently from the path lengths. For simplicity, we focus only on computing shortest path lengths.

3.1 Reducing the Number of Variants

Our notation currently allows us to specify 16 different problem variants based on which elements in either $EA_{sd}(t)$ or $LD_{sd}(t)$ are replaced with wildcards. Most of these variants are similar or highly symmetric in time, so we can reduce this list to only 2 fundamental problems as shown in Figure 1. The first fundamental variant, computing $EA_{s*}(t)$, is the problem of finding shortest paths departing from a single node s at a single departure time t. The second variant, computing $EA_{s*}(*)$, involves finding shortest paths from a single node s at every possible departure time. This variant is the time-dependent analog of the well-known one-to-all static shortest path problem.

3.2 Reversing Time

In the static realm, the one-to-all and all-to-one shortest path problems are made equivalent by reversing all arcs in a network. In the FIFO time-dependent case, a similar feat is possible, except that it is also necessary to reverse the direction of time. The time reversal transformation changes earliest arrival time problems into latest departure time problems as follows:

- Reverse the roles of all sources and destinations.
- Reverse the roles of departure and arrival times, negating all of these times in the process.
- Reverse the direction of all arcs.
- Replace each arc arrival time function $a_{ij}(t)$ with $-a_{ij}^{-1}(-t)$.

When done solving the time-reversed problem instance, $EA_{sd}(t)$ will be the same as $-LD_{ds}(-t)$ prior to the transformation. Similarly, the value of $LD_{sd}(t)$ after the transformation will be equal to $-EA_{ds}(-t)$ prior to the transformation. The reversal of a problem instance over the interval of time $[0, T]$ results in a problem over the interval $[-T, 0]$. As intuition, a particle moving forward in time at time t in the first problem corresponds to a "mirror-image" particle at the same location moving backwards in time at time $-t$ in the reversed problem. One should note that negative values of time are perfectly acceptable. In discrete time, where a finite window of time is required, we assume that all time is to be eventually translated into range from 0 to T for simplicity.

3.3 Optimality Conditions

For our two fundamental time-dependent shortest path problems, optimality conditions are similar to those of the static shortest path problem. The primary difference is that function composition replaces addition.

Lemma 3.1. *The following conditions are necessary and sufficient for optimality for the problems of computing $EA_{s*}(t)$ and $EA_{s*}(*)$. For computing $EA_{s*}(t)$, these conditions need hold only for a particular departure time t, whereas for computing $EA_{s*}(*)$ they must hold for all values of time.*

$$
\begin{array}{lll}
(1) & EA_{ss}(t) = t & \\
(2) & a_{ij}(EA_{si}(t)) \geq EA_{sj}(t) & \forall (i,j) \in A \\
(3) & EA_{si}(t) = a_p(t) \text{ for some } p \in P(s,i) & \forall i \in N
\end{array}
$$

Proof: In order to show necessity, we argue that if condition (2) above fails to hold for any arc (i, j) and time t, then $EA_{sj}(t)$ may be reduced to the quantity $a_{ij}(EA_{si}(t))$, thereby contradicting the optimality of the current earliest arrival time values. It is straightforward to see that conditions (1) and (3) must hold for any optimal solution.

To show sufficiency, assume for the purposes of contradiction that the conditions of the lemma hold for some non-optimal set of earliest arrival time functions. Since they are non-optimal, there exists some destination node d, departure time t, and path p for which $a_p(t) < EA_{sd}(t)$. Let $s - i_1 - i_2 - ... - i_k - d$ denote the nodes along p. By repeatedly composing condition (2) along the arcs in p, we have the following contradiction:

$$\begin{aligned}
a_{si_1}(t) &\geq EA_{si_1}(t) \\
a_{i_1 i_2}(a_{si_1}(t)) &\geq a_{i_1 i_2}(EA_{si_1}(t)) \\
a_{i_1 i_2}(a_{si_1}(t)) &\geq EA_{si_2}(t) \\
a_{i_2 i_3}(a_{i_1 i_2}(a_{si_1}(t))) &\geq a_{i_2 i_3}(EA_{si_2}(t)) \\
a_{i_2 i_3}(a_{i_1 i_2}(a_{si_1}(t))) &\geq EA_{si_3}(t) \\
&\cdots \\
a_p(t) &\geq EA_{sd}(t). \qquad \square
\end{aligned}$$

When computing $EA_{s*}(t)$, the output is similar to the that of a static shortest path problem in that there is a single scalar "distance label", $EA_{si}(t)$, to be computed for every node $i \in N$. For computing $EA_{s*}(*)$, the "distance labels" one must compute for every node $i \in N$ are the functions EA_{si} over all time. In this problem, we can now see why piecewise linear arc arrival time functions are the most practical choice in continuous time. With this choice, the EA_{si}'s are also piecewise linear functions. Moreover, even quadratic arc arrival time functions will result in the EA_{si}'s being piecewise polynomial of degree $O(2^n)$, and manipulating such functions can be quite unwieldy.

3.4 Properties of FIFO Networks

We have already seen many useful properties of FIFO networks. For example, the aforementioned simple optimality conditions and the high degree of symmetry in time between problem variants and between derived time-dependent functions only holds if the FIFO property is respected. In this section, we investigate several other features of FIFO networks. The following lemmas, valid only in FIFO networks, highlight some useful properties of time-dependent shortest paths in such networks.

Lemma 3.2. *In a FIFO network, waiting at nodes is never beneficial (i.e., it will never reduce the arrival time at one's eventual destination).*

Lemma 3.3. *In a FIFO network, one may always find shortest paths which are acyclic.*

Lemma 3.4. *In a FIFO network, one may always find shortest paths whose subpaths are also shortest paths. More precisely, for a given source node s, destination node d, and departure time t, one may always find a path $q \in P(s, d)$ for which $a_q(t) = EA_{sd}(t)$ (i.e. a shortest path) such that every subpath $p \in P(i, j)$ satisfies $a_p(EA_{si}(t)) = EA_{ij}(EA_{si}(t))$.*

Proof: Lemma 3.2 follows directly from the fact that path arrival time functions are non-decreasing, and Lemma 3.3 follows as a consequence since any cycle in a path can be removed and replaced with an equivalent amount of waiting at some node formerly on the cycle. The only way a shortest path may contain a cycle at all is for the cycle to be comprised soley of "zero delay" arcs.

Lemma 3.4 may be established by the following construction: Suppose for some departure time t and some subpath p of q there exists a shorter path p' for which $a_{p'}(EA_{si}(t)) < a_p(EA_{si}(t))$. By splicing p' into q in place of p, we can construct a new path q' of no greater length than q:

$$\begin{aligned}
a_{p'}(EA_{si}(t)) &< a_p(EA_{si}(t)) \\
EA_{jd}(a_{p'}(EA_{si}(t))) &\leq EA_{jd}(a_p(EA_{si}(t))) \\
a_{q'}(t) &\leq a_q(t).
\end{aligned}$$

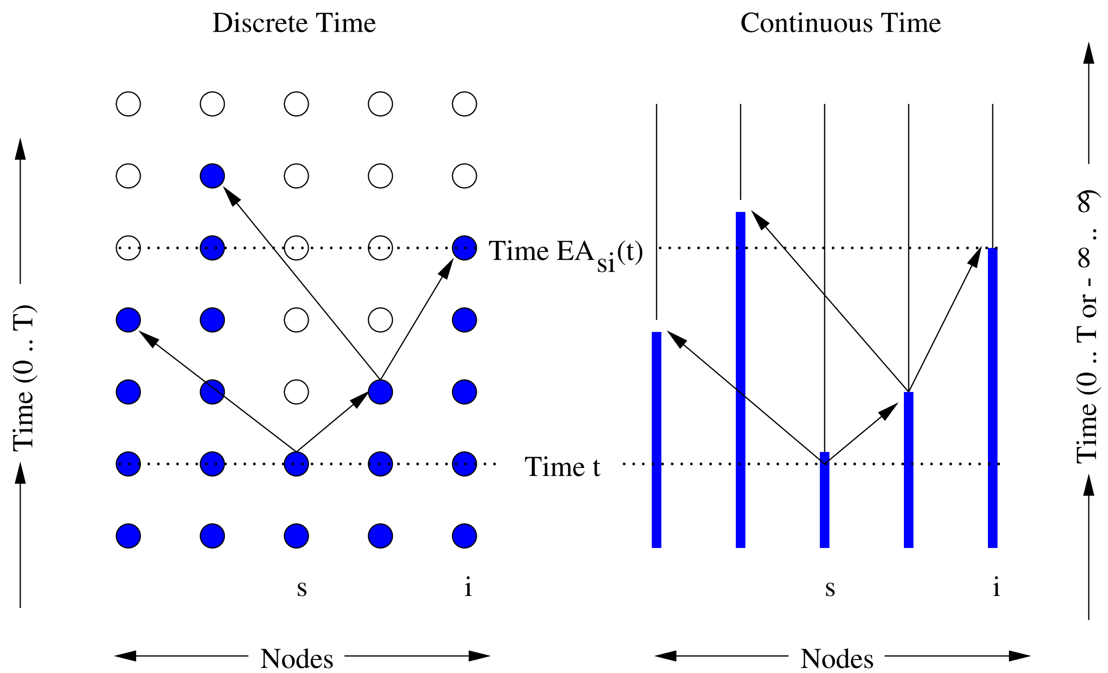

Figure 2: Graphical depiction of shortest path trees on a time-space diagram.

By repeatedly performing this procedure, one will eventually transform q into a shortest path whose subpaths are all shortest paths. It should be noted that in a strictly FIFO network (in which arc arrival time functions are strictly increasing) subpaths of any shortest path are always shortest paths, since the second step above preserves the strict inequality. □

Due to Lemma 3.2 we may safely assume that waiting is forbidden in the network. In non-FIFO networks, one may conceive of elaborate waiting policies which can be a source of considerable complexity. One of these policies, however, that in which unlimited waiting is allowed at all nodes, allows a non-FIFO problem to be transformed into an equivalent FIFO problem by incorporating waiting into the arc arrival time functions as follows

$$a'_{ij}(t) = \min_{\tau \geq t} a_{ij}(\tau).$$

In continuous time, one may construct discontinuous non-FIFO arc arrival time functions for which this minimum does not exist, such as the following:

$$a_{ij}(t) = \begin{cases} t & \text{if } t < 0 \\ 2 - t & \text{if } 0 \leq t < 1 \\ t + 1 & \text{if } t \geq 1. \end{cases}$$

In order to disallow such functions, we usually assume continuity of non-FIFO arc arrival time functions when waiting at nodes is permitted.

Lemma 3.4 shows that as a solution to the problem of computing $EA_{s*}(t)$ one may find a shortest path tree directed out of node s, just as with a static shortest path problem. Similarly, when computing $LD_{*d}(t)$, one may obtain a shortest path tree directed into node d. As shown in Figure 2, we can use time-space diagrams to conveniently visualize such trees. Each of the two diagrams can be thought of as a network expanded by adding a time dimension, where each column represents a single node in N. For the problem of computing $EA_{s*}(t)$, a shortest path tree is shown which spreads forward through time and reaches each node i at time $EA_{si}(t)$. For latest departure time problems, the tree will be “upside down” and directed into a single destination node. One may wish to visualize the time-reversal transformation described in Section 3.2 as a reflection of these diagrams about the $t = 0$ line.

Every shortest path tree rooted at some node s at some time t partitions the time-space realm into two disjoint regions, which we denote $\mathcal{R}_1(s,t) = \{(i,\tau) : \tau > EA_{si}(t)\}$ and $\mathcal{R}_2(s,t) = \{(i,\tau) : \tau \leq EA_{si}(t)\}$. $\mathcal{R}_2(s,t)$ is shaded in Figure 2. This partition will play an important role later in problem decomposition.

Finally, we consider the size of the output when computing $EA_{s*}(*)$, for which we must determine the function $EA_{si}(t)$ for all $i \in N$. In discrete time, these functions can be represented by be vectors with $T+1$ integer components, since we assumed that we focus on a finite window of time of duration T. In continuous time, however, it seems that these functions might in the worst case be quite complicated to describe. As we mentioned before, if arc arrival time functions are not piecewise linear, the $EA_{si}(t)$ functions can be polynomials of exponentially high degree. Moreover, even if the arc arrival time functions are piecewise linear, it is not clear how many pieces will comprise the each $EA_{si}(t)$ function computed as output. This is a currently a significant open question in the study of time-dependent shortest path problems — whether or not there is a polynomial bound on the number of pieces of $EA_{si}(t)$ in the worst case. The author suspects that there is not a polynomial bound, since a superpolynomial worst-case lower bound has been shown [3] for the somewhat related parametric shortest path problem (where the cost of a path is obtained by adding, rather than composing, linear functions along a path).

Conjecture 3.1. *In a FIFO network with piecewise linear arc arrival time functions, some function EA_{sd} can in the worst case contain a superpolynomial number of linear pieces.*

4 Algorithms

In this section we describe serial and parallel algorithms for solving our two fundamental time-dependent shortest path problems.

4.1 Computing $EA_{s*}(t)$

Consider the problem of computing $EA_{s*}(t)$, in which we wish to find shortest paths leaving some source node s at a particular departure time t. The output is to be a single scalar distance label, $EA_{si}(t)$, for every node $i \in N$. This problem is quite similar to the well-known static shortest path problem, and indeed, as as initially shown by Dreyfus [9], it can be solved by a trivially-modified variant of any label-setting or label-correcting static shortest path algorithm (see [1] for a detailed explanation of these static algorithms). Pseudocode containing the necessary modifications for each algorithm is shown in Figure 3. Correctness of these algorithms (and the remaining algorithms in this section) may be argued in a straightforward manner by showing that their output respects the optimality conditions stated in Lemma 3.1.

The asymptotic running times of these modified algorithms will be the same as those of their static counterparts. For the label-setting algorithm, the running time depends on the implementation of the priority queue S; the strongest known running time, $O(m + n\log n)$, is achieved when S is a Fibonacci heap. Performance of the label-correcting algorithm depends on the implementation of the set Q, for which a FIFO queue and a dequeue are popular choices. A polynomial running time of $O(mn)$ is achieved if Q is implemented as a FIFO queue.

The label-correcting algorithm may be interpreted to be acting on distance labels for each node $i \in N$ which are either (i) the scalars $EA_{si}(t)$ for a particular time t, or (ii) the functions EA_{si} over all values of time. For this problem, the distance labels are to be interpreted as scalars in both algorithms. In the label-setting algorithm, labels may only be construed as scalars since the operation of selecting the minimum element from S makes no sense over functions.

<table>
<tr>
<td>

<u>Initialization</u>
For all $i \in N\backslash\{s\}$: $EA_{si}(t) \leftarrow \infty$
$EA_{ss}(t) \leftarrow t$
$S \leftarrow N$

<u>Main Loop</u>
While $S \neq \emptyset$
  Select $i \in S$ minimizing $EA_{si}(t)$
  $S \leftarrow S\backslash\{i\}$
  For all j such that $(i,j) \in A$
    $EA_{sj}(t) \leftarrow \min\{EA_{sj}(t), a_{ij}(EA_{si}(t))\}$

</td>
<td>

<u>Initialization</u>
For all $i \in N\backslash\{s\}$: $EA_{si}(t) \leftarrow \infty$
$EA_{ss}(t) \leftarrow t$
$Q \leftarrow \{s\}$

<u>Main Loop</u>
While $Q \neq \emptyset$
  Select some $i \in Q$
  $Q \leftarrow Q\backslash\{i\}$
  For all j such that $(i,j) \in A$
    $f(t) \leftarrow \min\{EA_{sj}(t), a_{ij}(EA_{si}(t))\}$
    If $EA_{sj}(t) \neq f(t)$ Then
      $EA_{sj}(t) \leftarrow f(t)$
      $Q \leftarrow Q \cup \{j\}$

</td>
</tr>
<tr>
<td>Label-Setting (Dijkstra's) Algorithm</td>
<td>Label-Correcting Algorithm</td>
</tr>
</table>

Figure 3: Pseudocode for time-dependent shortest path algorithms.

4.2 Computing $EA_{s*}(*)$ – Label-Correcting Algorithms

We now turn to the problem of computing shortest paths leaving a single node s departing at all possible times rather than just a single time t. The modified label-correcting algorithm from the previous section (with distance labels construed to be functions rather than scalars) solves this problem. One may view the algorithm as solving an instance of $EA_{s*}(t)$ as before, only now simultaneously for all values of t. In discrete time, this algorithm was proposed by Ziliaskopoulos and Mahmassani [15] to solve the symmetric problem of computing $LD_{*d}(*)$. Since the algorithm operates on $(T+1)$-component vectors rather than scalars, its running time is now $O(mnT)$. In continuous time, Orda and Rom [12, 13] proposed essentially the same algorithm. Not assuming the FIFO property, they were only able to prove termination within a finite amount of time. Given that the FIFO property holds, however, we have a bound of $O(mnF)$, where F denotes the running time required to perform fundamental operations (e.g. addition, comparison, minimum) on our continuous-time functions. In the case of piecewise linear functions, F is essentially the number of pieces that might be present in a given output function $EA_{si}(t)$, so the question of whether or not these algorithms run in polynomial time depends on the resolution of the earlier conjecture regarding whether or not $EA_{si}(t)$ might have a superpolynomial number of pieces.

4.3 Computing $EA_{s*}(*)$ – "Label-Setting" Algorithms

Although it has the advantage of simplicity, the label-correcting algorithm is not the most efficient algorithm for this problem. We proceed to describe what one may describe as analogs to the label-setting algorithm for this problem, in the sense that they compute in small pieces the actual correct values of the output functions rather than iteratively revising these functions.

4.3.1 Discrete Time

A simple algorithm which surpasses the label-correcting algorithm in discrete time is to decompose the computation of $EA_{s*}(*)$ by time into $T+1$ compututations of $EA_{s*}(t)$. This is a valid method since the optimality conditions include no dependence between different values of time for the functions EA_{si}. It achieves a running time of $O(T(m + n\log n))$.

One may develop faster and even simpler algorithms in discrete time by considering a time-dependent

Discrete Time Algorithm	**Piecewise Linear Continuous Time Algorithm**

<u>Initialization</u>
For $t \leftarrow 0$ to T
  $D_{si}(t) \leftarrow \infty$ for all $i \in N \backslash \{s\}$
  $D_{ss}(t) \leftarrow 0$
 	
<u>Main Loop</u>
For $t \leftarrow 1$ to T
  For all $i \in N$
    $D_{si}(t) \leftarrow \min\{D_{si}(t), D_{si}(t-1)\}$
  For all $(i,j) \in A$
    If $a_{ij}(t) \leq T$ and
    $D_{sj}(a_{ij}(t)) > D_{si}(t) + a_{ij}(t) - t$ Then
      $D_{sj}(a_{ij}(t)) \leftarrow D_{si}(t) + a_{ij}(t) - t$
$EA_{si} \leftarrow (t - D_{si}(t))^{-1}$ for all $i \in N$
	<u>Initialization</u>
	For all $(i,j) \in A$: $(\alpha_{ij}, \beta_{ij}, \tau_{ij}) \leftarrow P_{ij}^{th}$ piece of a_{ij}
	Compute (α_i, β_i) for all $i \in N$ using Dijkstra's Algorithm
	Compute X_{ij}, Y_{ij}, and Z_{ij} for all $(i,j) \in A$
	$t \leftarrow \infty$
	
	<u>Main Loop</u>
	While $t > 0$
	  $t_{new} \leftarrow \max Z_{ij}$; $(i,j) \leftarrow \arg\max Z_{ij}$
	  If $t_{new} = X_{ij}$ Then
	    Decrement P_{ij}; $(\alpha_{ij}, \beta_{ij}, \tau_{ij}) \leftarrow P_{ij}^{th}$ piece of a_{ij}
	    Recompute X_{ij}, Y_{ij}, and Z_{ij}
	  Otherwise (here, we know $t_{new} = Y_{ij}$)
	    Record the current piece $(\alpha_j, \beta_j, t_{new})$ of EA_{sj}
	    $(\alpha_j, \beta_j) \leftarrow (\alpha_{ij}\alpha_j, \alpha_{ij}\beta_i + \beta_{ij})$
	    Recompute X_{kj}, X_{kj}, and Z_{kj} for k s.t. $(k,j) \in A$
	  $t \leftarrow t_{new}$
	Record the first piece $(\alpha_i, \beta_i, 0)$ of EA_{si} for all $i \in N$

Figure 4: "Label-Setting" algorithms which compute $EA_{s*}(*)$.

problem to be nothing more than a static shortest path problem cast in the framework of a *time-expanded network*, G_T, formed as shown in Figure 2 by replicating the network along a time dimension. This network contains $n(T+1)$ nodes of the form (i,t), where $i \in N$ and $t \in \{0,1,2,...,T\}$. It contains $O(mT)$ arcs of the form $((i,t),(j,a_{ij}(t)))$, where $(i,j) \in A$, $t \geq 0$, and $a_{ij}(t) \leq T$. Essentially all time-dependent shortest path problems in discrete time can be cast as an equivalent reachability or shortest path problem in G_T. For example, computing $EA_{s*}(*)$ is equivalent to finding the shortest path to every node (i,t) from the set of source nodes $\mathcal{S} = \{(s,\tau) : 0 \leq \tau \leq T\}$. Since G_T is acyclic, static shortest paths may be computed in linear time, $O(mT)$, by examining the nodes of G_T in a topological ordering. If instantaneous travel is possible between nodes, there may be a cycle in G_T; however, an equivalent acyclic network may be formed in linear time by coallescing strongly-connected components.

Assuming that $a_{ij}(t) > t$ for all arcs (i.e., that arc travel times are strictly positive), a topological ordering of the nodes in G_T may be constructed easily by enumerating them one "level" at a time in chronological order; this approach was initially proposed by Cai, Kloks, and Wong [2], and by Chabini [4]. To implement this, we introduce a distance label $D_{si}(t)$ for each node (i,t) in G_T which will give the shortest path distance from $\mathcal{S}$ to (i,t) in G_T. These distance labels will satisfy $LD_{si}(t) = t - D_{si}(t)$, so we can compute EA_{si} by taking the inverse of $t - D_{si}(t)$. We can express the optimality conditions in terms of these new distance labels as follows:

$$
\begin{array}{lll}
(1) & D_{ss}(t) = 0 & 0 \leq t \leq T \\
(2) & D_{si}(t) + a_{ij}(t) - t \geq D_{sj}(a_{ij}(t)) & \forall (i,j) \in A, 0 \leq t \leq T
\end{array}
$$

Using this formulation, the simple pseudocode in Figure 4 may be used to compute optimal distance labels for the nodes in G_T, from which we may then obtain $EA_{s*}(*)$. It is important to note that G_T does not need to be explicitly constructed when using this algorithm. This approach requires only $O(mT)$ time, which matches the lower bound on the complexity for computing $EA_{s*}(*)$ in discrete time.

4.3.2 Continuous Time

In continuous time, for the case where arc arrival times are piecewise linear, Dean [7] developed a "label-setting" algorithm which performs a single chronological scan through time to establish the output functions. The algorithm is somewhat similar to approaches for solving parametric shortest path problems by scanning forward through time.

To describe the algorithm, we first introduce some extra notation: we describe each linear piece of the arc arrival function a_{ij} by a 3-tuple $(\alpha_{ij}, \beta_{ij}, \tau_{ij})$, where $a_{ij}(t) = \alpha_{ij}t + \beta_{ij}$ for the extent of the piece, and where τ_{ij} gives the value of t at the upper boundary of the piece. Similarly, we describe a piece of each output function E_{si} as a 3-tuple $(\alpha_i, \beta_i, \tau_i)$. It is also assumed that $a_{ij}(t) > t$ for all $(i,j) \in A$.

As time approaches $-\infty$, there will eventually be some initial shortest path tree that changes no further as time decreases. This tree may be computed using any static shortest path problem as follows: let $(\alpha_{ij}, \beta_{ij}, \tau_{ij})$ be the parameters for the initial piece of each arc arrival function a_{ij}. As $t \to -\infty$, the travel time along every arc (i,j) is given by the simple linear function $(\alpha_{ij} - 1)t + \beta_{ij}$. There is a total ordering imposed on simple linear functions as $t \to -\infty$, in which two functions are compared by first comparing their linear coefficients, with ties being broken by comparing their constant terms. We can therefore apply any static shortest path algorithm to this problem, regarding path lengths as simple linear functions rather than scalars. As output, we produce the initial linear piece $(\alpha_i, \beta_i, -)$ of each function EA_{si}; note that τ_i has yet to be determined.

Having computed the initial linear pieces of all of the output functions EA_{si}, we now scan forward through time to compute the remainder of these functions, piece by piece. Here we introduce the notion of an *active piece* of a piecewise linear function as being the piece currently under consideration. Initially, the active piece of each a_{ij} is the initial piece; we let $(\alpha_{ij}, \beta_{ij}, \tau_{ij})$ denote this piece. Similarly, we start with the active piece of each output function EA_{si} as the $(\alpha_i, \beta_i, -)$ tuple produced by the preceding shortest path calculation. Based on the active pieces of the input and output functions, we define the following values X_{ij}, Y_{ij}, and Z_{ij} for each $(i,j) \in A$:

$$
\begin{aligned}
X_{ij} &= \begin{cases} \frac{\tau_{ij} - \beta_i}{\alpha_i} & \text{if } \alpha_i > 0 \\ 0 & \text{if } \alpha_i \leq 0. \end{cases} \\
Y_{ij} &= \begin{cases} \frac{\beta_j - \beta_{ij} - \alpha_{ij}\beta_i}{\alpha_{ij}\alpha_i - \alpha_j} & \text{if } \alpha_{ij}\alpha_i > \alpha_j \\ 0 & \text{if } \alpha_{ij}\alpha_i \leq \alpha_j. \end{cases} \\
Z_{ij} &= \min\{X_{ij}, Y_{ij}\}
\end{aligned}
$$

Our goal is now to scan through time starting from $t = -\infty$, such that the optimality condition $a_{ij}(EA_{si}(t)) \geq EA_{sj}(t)$ holds for the active pieces of the input (a_{ij}'s) and output (EA_{si}'s). The X_{ij} and Y_{ij} values help us determine how far ahead in time we can safely scan using the current set of active pieces. Every time we encounter a breakpoint in some arc arrival time function a_{ij} we must stop and move to the next active piece of this function; the value of X_{ij} gives the time at which we must stop to make such an adjustment due to the arc (i,j). The fact that arc arrival time functions are non-decreasing implies that over the entire course of the algorithm we will monotonically sweep over all of their pieces at most once. Additionally, the value of Y_{ij} gives the time at which the arc (i,j) may violate the optimality condition by having improved to the extent that it should be made part of a shortest path. At such a time, we need to perform a "pivot" operation to reroute the shortest path along this arc, which results in a piecewise boundary being added to the output function E_{sj}.

Pseudocode for the complete algorithm appears in Figure 4. Since the Z_{ij} values are stored in a heap, the algorithm achieves a worst-case running time of $O(mP^{**} \log n)$, , where P^{**} is the total number of pieces among all output functions. This slightly improves on the worst-case running time of the label-correcting algorithm in the amount of time required per linear piece of the output. Since the running time is highly-dependent on the number of pieces present in the output, the practical utility of this algorithm depends highly on the extent to which the dynamics of the input are "well-behaved". This is to be contrasted with the discrete-time domain, in which input dynamics have no affect on the previously-mentioned algorithms.

4.4 Computing $EA_{s*}(*)$ in Parallel

Recall from Section 3.4 that a single computation of $EA_{s*}(t)$ (equivalent to solving a static shortest path problem) induces a partition of time and space into two disjoint regions $\mathcal{R}_1(s,t)$ and $\mathcal{R}_2(s,t)$. Note that all shortest paths departing i within $\mathcal{R}_1(s,t)$ are solely contained within $\mathcal{R}_1(s,t)$, and all paths departing within $\mathcal{R}_2(s,t)$ lie solely within $\mathcal{R}_2(s,t)$. We can therefore decompose the computation of $EA_{s*}(*)$ according to these two regions. The two subproblems have no interdependence, and each will likely require less time than the original problem since it involves computing only a small part of the original output. Using this method, initially proposed in [7], we can decompose this problem into an arbitrary number of disjoint smaller pieces for parallel processing.

4.5 Algorithm Performance in Practice

We have implementated of all of the serial algorithms described above, and we wish to briefly summarize our computational experience. As expected, $EA_{s*}(t)$ can be computed many orders of magnitude faster than $EA_{s*}(*)$. We have found that the modified static shortest path algorithms used to compute $EA_{s*}(t)$ exhibit the same performance in practice as their static counterparts.

We next look at the problem of computing $EA_{s*}(*)$. In discrete time, we have found that the label-setting algorithm which operates (implicitly) on the time-expanded network is always more efficient than the label-correcting algorithm; this is the expected result when one considers the operation and theoretical analysis of these algorithms. In continuous time for the piecewise linear case, we have found similar performance among the label-setting and label-correcting algorithms.

Continuous-time algorithms typically have weaker worst-case performance guarantees than their discrete-time counterparts; however in practice, piecewise linear continuous-time algorithms can run substantially faster and consume far less memory than discrete-time algorithms for "well behaved" practical problems, such as for example, transportation problems in which arc arrival time functions consist of few linear pieces. The comparison between continuous-time and discrete-time algorithms depends heavily on T, the number of pieces in the input functions P^*, and network topology. See [7] for a thorough computational comparison of these algorithms.

5 Extensions and Conclusions

In this section we briefly summarize the results of the previous sections, and then proceed to discuss how these results apply to more general problem variants such as non-FIFO problems.

We have seen that the problem of computing $EA_{s*}(t)$ is solved using a trivially-modified static shortest path algorithm, where the modifications have no impact on the running time of the algorithm. We have also investigated several algorithms for computing $EA_{s*}(*)$ in discrete time and in continuous time under the assumption of piecewise linear arc arrival time functions. The running times of these algorithms are summarized in Figure 5.

In practice, recall that the performance of continuous-time algorithms is much more sensitive to the dynamics of the problem instance than that of discrete-time algorithms. Finally, we discussed how the problem of computing $EA_{s*}(*)$ can be decomposed into an arbitrary number of independent subproblems for parallel computation.

	FIFO Running Time		Non-FIFO Running Time	
Algorithm:	**Discrete Time**	**Continuous Time (piecewise linear)**	**Discrete Time**	**Continuous Time (piecewise linear)**
Label Correcting	$O(mnT)$	Not Known	$O(mnT^2)$	Not Polynomial
Repeated $EA_{s*}(t)$	$O(T(m + n \log n))$	Not Applicable	Not Applicable	Not Applicable
Label Setting	$\Theta(mT)$	Not Known	$\Theta(mT)$	Not Polynomial

Figure 5: Running times of algorithms which compute $EA_{s*}(*)$. The "not known" running times depend on the outcome of Conjecture 3.1.

5.1 The Non-FIFO Case

We have focussed on FIFO networks since they are common in practice, and lead to rich theoretical results and efficient algorithms. Most of the theoretical properties such as for example time symmetry, the existence of shortest path trees, and the fact that modified static shortest path algorithms may be used to solve for $EA_{s*}(t)$, all fail to hold in the absence of the FIFO property. In non-FIFO networks, the problem of computing $EA_{s*}(t)$ is usually solved by computing the more general quantity $EA_{s*}(*)$. Other problem variants may be considered as well, such as computing $LD_{*d}(*)$, since the time-reversal transformation no longer applies; however, these variants are similar enough to our two primary variants that essentially all algorithmic results carry over to the new variants. However, even the simplest possible problem variant of computing $EA_{sd}(t)$ can be shown to be at least NP-Hard in non-FIFO networks by a straightforward reduction from the Subset-Sum problem.

The good news is that most of the algorithms described for computing $EA_{s*}(*)$ still work in non-FIFO networks, typically with some degradation in running time. The running time guarantees of these algorithms in non-FIFO networks are shown in Figure 5. No modifications are required to any of the algorithms in order to operate in the non-FIFO case, except for the label-setting algorithm in continuous-time, which becomes more complicated since there are more conditions which interrupt its reverse chronological scan. The worst-case running time of piecewise linear continuous time algorithms can grow quite large in theory, but this is because output size is no longer polynomially-bounded. However, for "well behaved" networks in which shortest paths do not change too frequently, piecewise linear continuous time algorithms typically still exhibit reasonable performance in practice.

5.2 Minimum-Cost Path Problems

One sometimes encounters time-dependent problems where arcs are given time-dependent *travel costs* in addition to (possibly time-dependent) travel times, and one must solve for minimum cost paths. In order to illustrate the complexity of these problems, we point out the following: even if all arc travel times remain static, and only one arc cost changes at only one instant in time, the minimum-cost path problem can be shown to be NP-hard by a reduction from the constrained shortest path problem. Minimum-cost path problems are similar in nature to non-FIFO minimum-time path problems, and all of the algorithms stated in Section 4 can be augmented to solve them except for the modified static algorithms for problems with a single departure time. Single-departure-time problems are solved, as in the non-FIFO case, by multiple-departure-time algorithms. Running times of these augmented algorithms are the same as those of their non-FIFO counterparts, as shown in Figure 5.

5.3 Waiting at Nodes

Recall that in FIFO networks waiting at nodes is not an issue, since waiting is never beneficial. In non-FIFO and minimum-cost path problems, waiting at nodes is sometimes beneficial, and one may construct different problem variants by adopting various policies at nodes. Since single-departure-time problems are solved using multiple-departure-time algorithms in both non-FIFO and minimum-cost path problems, we focus here only on multiple-departure-time problems. An arbitrary waiting policy may be modelled by specifying, for each node $i \in N$, the following functions (in discrete or continuous time):

- The length of allowed waiting, $w_i(t)$, if one arrives at node i at time t.
- For minimum-cost problems, the cost $c_i(t, \tau)$ of waiting τ units of time after arriving at time t.

The label-correcting and label-setting algorithms in both discrete and continuous time may all be augmented to handle arbitrary waiting policies at nodes. Waiting policies are especially straightforward in discrete time since they are modelled by simply adding "waiting arcs" to the time-expanded network G_T. After adding these extra $O(nT^2)$ arcs, G_T will still be acyclic, so shortest paths can be computed in linear time: $O(nT^2 + mT)$. However, in several common special cases, more sophisticated techniques [8] may be used to reduce this running time back to $O(mT)$.

References

[1] R. Ahuja, T. Magnanti, J. Orlin (1993). *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, Englewood Cliffs, NJ.

[2] X. Cai, T. Kloks, C. K. Wong (1997), "Time-Varying Shortest Path Problems with Constraints". *Networks* 29, 141-149.

[3] P.J.Carstensen (1984), "Parametric cost shortest path problems". Unpublished Bellcore memo.

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

[5] L. Cooke and E. Halsey (1966). "The Shortest Route Through a Network with Time-Dependent Internodal Transit Times". *Journal of Mathematical Analysis and Applications* 14, 492-498.

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

[7] B. Dean (1999). *Continuous-Time Dynamic Shortest Path Algorithms.* Master's Thesis, MIT Department of Computer Science.

[8] B. Dean (2004). "Algorithms for Minimum-Cost Paths in Time-Dependent Networks with Waiting Policies". *Networks* 44(1), 41-46.

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

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

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

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

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

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

[15] A. Ziliaskopoulos, H. Mahmassani (1993). "A Time-Dependent Shortest Path Algorithm for Real-Time Intelligent Vehicle Highway System Applications". *Transportation Research Record* 1408, 94-104.