Project Number 288094

eCOMPASS

eCO-friendly urban Multi-modal route PlANning Services for mobile uSers

STREP
Funded by EC, INFSO-G4(ICT for Transport) under FP7

eCOMPASS – TR – 017

Approximation Algorithms for Time-Dependent Shortest Paths

Spyros Kontogiannis and Christos Zaroliagis

April 2013

APPROXIMATING TIME-DEPENDENT SHORTEST PATHS IN ROAD NETWORKS*

SPYROS KONTOGIANNIS AND CHRISTOS ZAROLIAGIS

Abstract. We present efficient algorithms for approximating time-dependent shortest travel-time functions, in directed graphs representing road networks.

1. Introduction

Computing shortest paths in graphs is a core task in many real-world applications, such as route planning in transportation networks, routing in communication infrastructures, etc. Typically the underlying graph is accompanied with an arc-cost function, assigning a *fixed* cost value to every arc, representing average travel-time, distance, fuel consumption, etc. The path of a particular cost is then the aggregation of arc costs along it.

However, in real-world applications the cost of each arc should not be considered as a fixed value, since it undergoes frequent updates. These updates may be instantaneous, unpredictable changes (e.g., due to a sudden change of weather conditions, or a car accident that blocks a road segment or junction), or anticipated updates due to periodic changes of the network characteristics over time. For example, the traversal-time of a road segment may depend on the real-time congestion upon traversal, and thus on the departure time from its tail: In rush hours it is anticipated that it will be much longer than the free-ride traversal-time which is usually valid only for particular departure times (e.g., during the weekend, or at night). Such networks in which the characteristics of the network change in a predictable fashion over time, are called *time-dependent networks*. In the present work we focus on such networks in which it is the behavior of the arc-cost functions that are described by time-dependent functions, whose exact shape comes from statistical analysis of historical traffic information. For example, the traversal time of a particular road segment may be sampled at particular times during a day from the historical traffic information, say per 5 minutes during rush hours and more rarely for the remaining periods of the day; the corresponding arc-cost function is then considered to be the (continuous) interpolant of all these sample points. Simply taking a snapshot of the entire network (if possible) and solving the corresponding Static Shortest Path problem is clearly not the proper way to provide a route plan in this case. In the following we shall consider as arc-cost functions the traversal-time (or delay) functions when we start traversing them at particular times. The problem is then to compute a truly shortest path between an origin-vertex and a destination vertex in the network taking into account also, not only the departure time from the origin, but also the consequent departure time of any other arc that is to be used by a shortest path towards the destination. The problem was introduced in [1].

1.1. Notation.

Consider a directed graph $G = (V, A)$, with nonnegative, continuous, piecewise linear (pwl) *arc-delay* functions $\forall a \in A, \overrightarrow{D}[a] : \mathbb{R} \to \mathbb{R}_{\geq 0}$ providing the arrival time at the destination $head[a]$ as a function of the departure time from the origin $tail[a]$. Such a function could for example be the interpolant of average arc-delays for particular departure times from a

Date: September 1, 2013.

1991 *Mathematics Subject Classification.* 05C85: Graph algorithms; 05C12: Distance in graphs; 68W25: Approximation algorithms; 68Q25: Analysis of algorithms and problem complexity.

Key words and phrases. Time-dependent shortest paths, FIFO property, distance oracles.

* Partiallly supported by the EU FP7/2007-2013 (DG CONNECT.H5-Smart Cities & Sustainability), under grant agreement no. 288094 (project eCOMPASS).

S. Kontogiannis: University of Ioannina and Computer Technology Institute & Press "Diophantus", kontog@cs.uoi.gr.

C. Zaroliagis: University of Patras and Computer Technology Institute & Press "Diophantus", zaro@ceid.upatras.gr.

given time period $\Pi = [0, T]$, such that $\forall k \in \mathbb{Z}, \forall t_u \in \Pi, \forall a \in A, \overrightarrow{D}[a](t_u + k \cdot T) = \overrightarrow{D}[a](t_u)$. An example of such an arc-delay function is the following (its plot is shown in figure 1):

$$\forall t_u \in \mathbb{R},\ \overrightarrow{D}[uv](t_u) \quad = \quad \begin{cases} \frac{4}{3}t_u + 1, & 0 \leq t_u \mod T \leq 3 \\ 5, & 3 \leq t_u \mod T \leq 5 \\ 2t_u - 5, & 5 \leq t_u \mod T \leq 7 \\ -\frac{8}{13}t_u + \frac{173}{13}, & 7 \leq t_u \mod T \leq 20 \\ 1, & 20 \leq t_u \mod T \leq 24 \end{cases}$$

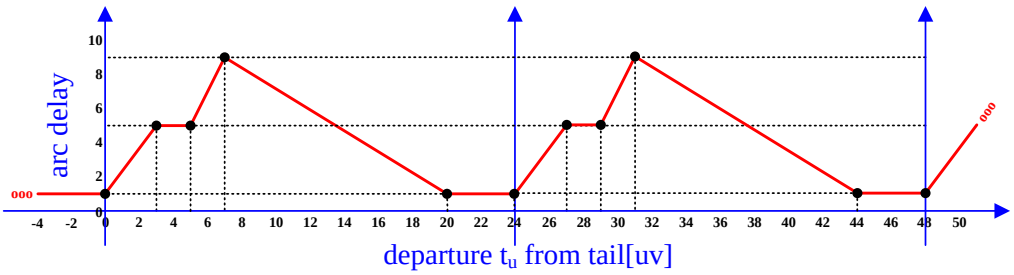


FIGURE 1. Example of a continuous pwl (forward) arc delay function (on the left) and the corresponding reverse-arc-delay (on the right), for an arc $a = uv \in A$, whose period is $T = 24h$.

For notational reasons we assume that $\forall t_u \in \Pi, \forall u \in V, \overrightarrow{D}[uu](t_u) = 0$ and $\forall uv \notin A \Rightarrow \overrightarrow{D}[uv](t_u) = +\infty$. Moreover, rather than defining the arc-delay functions as functions of *departure-time from the tail*, we may also prefer to express them as functions of *arrival-times at the heads*. We use the notation $\overleftarrow{D}[uv] : \mathbb{R} \to \mathbb{R}_{\geq 0}$ for these *reverse arc-delay* functions. For example, the reverse arc-delay function corresponding to the forward arc-delay of figure 1 is the following (see figure 2):

$$\forall t_v \in \mathbb{R},\ \overleftarrow{D}[uv](t_v) \quad = \quad \begin{cases} \frac{4}{7}t_v + \frac{3}{7}, & 1 \leq t_v \mod T \leq 8 \\ 5, & 8 \leq t_v \mod T \leq 10 \\ \frac{2}{3}t_v - \frac{5}{3}, & 10 \leq t_v \mod T \leq 16 \\ -\frac{8}{5}t_v + \frac{173}{5}, & 16 \leq t_v \mod T \leq 21 \\ 1, & 21 \leq t_v \mod T \leq 24 \vee 0 \leq t_v \mod T \leq 1 \end{cases}$$

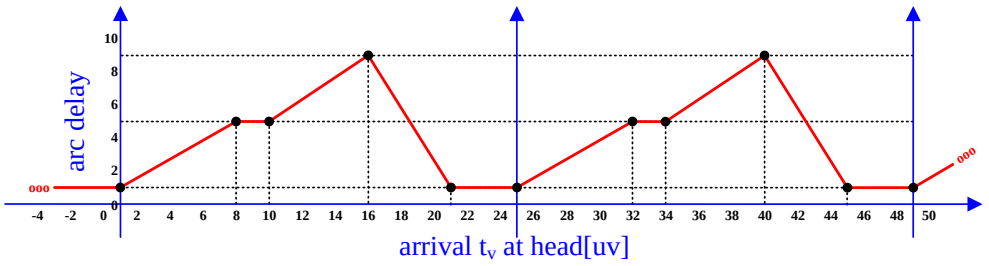


FIGURE 2. The reverse arc-delay function corresponding to the (forward) arc delay function of figure 1.

Remark: It is mentioned that for the reverse expression of the arc-delay function to exist, it must be the case that the original (forward) arc-delay does not have any leg of slope less or equal to -1. In particular, when this is the case, we can invert the (monotone in this case) arrival-time function $t_v = Arr[uv](t_u) = t_u + \overrightarrow{D}[uv](t_u)$ to get $Dep[uv] = (Arr[uv])^{-1}$ and then compute

$\overleftarrow{D}[uv](t_v) = t_v - Dep[uv](t_v) = Arr[uv](t_u) - t_u = \overrightarrow{D}[uv](t_u)$. As we shall explain later, we indeed demand that all the slopes in any forward arc-delay function have value strictly greater than this value, and this is not only for the computation of the reverse arc delays.

Analogously, $\overleftarrow{G} = (V, \overleftarrow{A})$ where $\overleftarrow{A} = \{vu \in V \times V : uv \in A\}$ is the graph produced by G if we reverse the directions of all the arcs in it.

For an arbitrary origin-destination pair of vertices, $(o, d) \in V \times V$, let $\mathcal{P}_{o,d}(G)$ be the set of all (directed) walks from o to d in G, while $\mathcal{P}(G) = \bigcup_{(o,d)\in V\times V} \mathcal{P}_{o,d}(G)$. For arbitrary vertices $u, v, z \in V$ and any walks $p \in \mathcal{P}_{u,v}$ and $q \in \mathcal{P}_{v,z}$, $p \oplus q \in \mathcal{P}_{u,z}$ is the walk resulting as the concatenation of p and q at vertex v. Any walk $p \in \mathcal{P}(G)$ that does not repeat any vertex is a (sometimes redundantly called simple) *path*. For sake of simplicity, we shall skip reference to the graph, when this is clear from the context. Any particular walk will mostly be considered as an *ordered set* of arcs such that for any pair of consecutive arcs, the head of the first arc is identical to the tail of the second arc. Occassionally we may want to declare a subwalk of p from (the first appearance in p of) a vertex $x \in V$ to (the fist appearance in p of) a vertex $y \in V$. This subwalk will be denoted by $p_{x\rightsquigarrow y}$.

For a walk (path) $p = \langle a_1, \cdots a_k\rangle \in \mathcal{P}_{o,d}$ and $\forall 1 \le i \le j \le k$, let $p_{i,j}$ be the subwalk (subpath) of p starting with the i^{th} arc a_i and ending with the j^{th} arc a_j in the order. We define the *walk/path-delay* function of p recursively as a function of the departure time t_o from its own origin $tail(p) = tail(a_1)$, as follows:

$$\begin{aligned} &\forall t_o \in \mathbb{R}, \forall 1 \le i \le k, \quad \overrightarrow{D}[p_{i,i}](t_o) = \overrightarrow{D}[a_i](t_o) \\ &\forall t_o \in \mathbb{R}, \forall 1 \le i < j \le k, \quad \overrightarrow{D}[p_{1,j}](t_o) = \overrightarrow{D}[p_{1,i}](t_o) + \overrightarrow{D}[p_{i+1,j}]\left(t_o + \overrightarrow{D}[p_{1,i}](t_o)\right) \end{aligned} \tag{1}$$

We may also express a similar recursive definition of the reverse-path-delays:

$$\begin{aligned} &\forall t_d \in \mathbb{R}, \forall 1 \le i \le k, \quad \overleftarrow{D}[p_{i,i}](t_d) = \overleftarrow{D}[a_i](t_d) \\ &\forall t_d \in \mathbb{R}, \forall 1 \le i < j \le k, \quad \overleftarrow{D}[p_{i,k}](t_d) = \overleftarrow{D}[p_{j,k}](t_d) + \overleftarrow{D}[p_{i,j-1}]\left(t_d - \overleftarrow{D}[p_{j,k}](t_d)\right) \end{aligned} \tag{2}$$

Similarly, we define the *arrival-time* function of p at its end-vertex $head(p) = head(a_k)$, as a function of the departure-time $t_o \in \mathbb{R}$ from its start-vertex $tail(p) = tail(a_1)$:

$$\forall t_o \in \mathbb{R}, Arr[p](t_o) = t_o + \overrightarrow{D}[p](t_o) \tag{3}$$

It is easily seen that the path-arrival-time functions are indeed compositions of the corresponding arc-arrival-time functions of the arcs comprising them:

$$\begin{aligned} Arr[p_{1,k}](t_o) &= t_o + \overrightarrow{D}[p_{1,k}](t_o) = t_o + \overrightarrow{D}[p_{1,1}](t_o) + \overrightarrow{D}[p_{2,k}](t_o + \overrightarrow{D}[p_{1,1}](t_o)) \\ &= Arr[p_{2,k}]\left(Arr[p_{1,1}](t_o)\right) = \left(Arr[p_{2,k}] \circ Arr[p_{1,1}]\right)(t_o) = \cdots \\ &= \left(Arr[a_k] \circ \cdots \circ Arr[a_1]\right)(t_o) \end{aligned} \tag{4}$$

Analogously, the path-departure-time function of p from $tail(p) = tail(a_1)$, given the arrival-time $t_d \in \mathbb{R}$ at $head(p) = head(a_k)$, is defined as follows:

$$Dep[p](t_d) = t_d - \overleftarrow{D}[p](t_d) \tag{5}$$

Again, the path-departure-time functions are compositions of the corresponding arc-departure functions of the arcs comprising them:

$$\begin{aligned} Dep[p_{1,k}](t_d) &= t_d - \overleftarrow{D}[p_{1,k}](t_d) = t_d - \overleftarrow{D}[p_{k,k}](t_d) - \overleftarrow{D}[p_{1,k-1}]\left(t_d - \overleftarrow{D}[p_{k,k}](t_d)\right) \\ &= Dep[p_{1,k-1}]\left(Dep[p_{k,k}](t_d)\right) = \left(Dep[p_{1,k-1}] \circ Dep[p_{k,k}]\right)(t_d) = \cdots \\ &= \left(Dep[a_1] \circ \cdots \circ Dep[a_k]\right)(t_d) \end{aligned} \tag{6}$$

For any pair of vertices $(o, d) \in V \times V$, the *earliest-arrival-time* function from o to d is defined as follows:

$$\forall t_o \in \mathbb{R},\ Arr[o, d](t_o) = \min_{p\in\mathcal{P}_{o,d}} \{Arr[p](t_o)\} \tag{7}$$

Assuming that the strict FIFO property holds (to be determined later), we know that at least one of the optimal $od-$walks assuring the earliest-arrival at the destination is indeed a path. Recall

also that self-loops are assumed to have zero delay, while inexistent loops are assumed to have infinite delay. Therefore, we could also write (7) as a continuous optimization problem as follows:

$$Arr[o,d](t_o) = \min_{v_1,\ldots,v_{n-2}\in V} \{Arr[v_{n-2},d](\cdots(Arr[o,v_1](t_o)))\}$$

The *earliest-arrival-time* at the destination, between a given pair of origin-destination nodes and a given departure time from the origin, is stated as follows:

Definition 1.1. Earliest-arrival-times to destinations, for given Origin (SOEAT)

INPUT :	*Directed graph* $G = (V, A)$. $o \in V \times V$*: An origin-vertex.* $t_o \in \Pi$*: The departure-time from the origin.* $\forall a \in A,\ \overrightarrow{D}[a] : \mathbb{R} \to \mathbb{R}_{\geq 0}$. *The* forward arc-delay *functions.*
GOAL :	*The earliest-arrival-time values* $Arr[o,d](t_o)$ *to all possible destinations* $d \in V$ *reachable from* o*, given the departure-time* t_o *from* o*. Moreover, a tree of* $od-$*paths path* $p[o,d] \in \mathcal{P}_{o,d}$ *achieving these arrival-times, i.e., such that* $Arr[p[o,d]](t_o) = Arr[o,d](t_o)$.

2. FIFO Property in Time Dependent Networks

A fundamental property of time-dependent networks is the *FIFO* (a.k.a. *non-overtaking*) property which states the following:

$$\forall t_u, t'_u \in \mathbb{R}, \forall uv \in A, t_u > t'_u \Rightarrow Arr[uv](t_u) \geq Arr[uv](t'_u) \tag{8}$$

That is, all the arc-arrival-time functions in the network are non-decreasing. The following proposition is a characterization of the FIFO property for networks with continuous arc-delay functions:

Proposition 2.1. *Assume a graph* $G = (V, A)$ *with continuous arc-delay functions, satisfying the (strict) FIFO property. Then any arc-delay function must have left and right derivatives with values at least (greater than)* -1.

Proof. Observe that, by the FIFO property: $\forall a \in A, \forall t_u \in \mathbb{R}, \forall \delta > 0$,

$$\begin{aligned} & Arr[a](t_u) \leq Arr[a](t_u + \delta) \Leftrightarrow t_u + \overrightarrow{D}[a](t_u) \leq t_u + \delta + \overrightarrow{D}[a](t_u + \delta) \\ \overset{/*\ \delta>0\ */}{\Leftrightarrow} \quad & \frac{\overrightarrow{D}[a](t_u + \delta) - \overrightarrow{D}[a](t_u)}{\delta} \geq -1 \end{aligned}$$

This immediately implies that the left and right derivatives of $D[a]$ are lower bounded (strictly, in case of strict FIFO property) by -1. □

It is also easy to verify that the FIFO property, only assumed for arc-arrival-time functions, also holds for arbitrary path-arrival-time functions, and earliest-arrival-time functions in the graph:

Proposition 2.2. *Assume a graph* $G = (V, A)$ *with continuous arc-delay functions, satisfying the FIFO property. Then, for any path* $p = \langle a_1, \ldots, a_k \rangle \in \mathcal{P}(G)$ *it holds that:*

$$\forall t_1 \in \mathbb{R}, \forall \delta > 0,\ Arr[p](t_1) \leq Arr[p](t_1 + \delta)$$

In case of strict FIFO property, the inequality is also strict. FIFO property holds also for every earliest-arrival-time function in G.

Proof. The explanation for the FIFO property on an arbitrary path p in G is provided by a simple inductive argument on the prefixes of p, based on the recursive definition of path-arrival-time functions (cf. equation (4)).

As for the earliest-arrival-time function $Arr[o,d]$, since this is the minimization operator over non-decreasing (increasing) path-arrival-time functions, it is also itself a nondecreasing (increasing) function of departure-time from o. □

When moving from an origin to a destination in a time-dependent network, a traveler may possibly have the option to wait at a node for certain amounts of time, prior to traversing an arc emanating from it. We consider the following cases of waiting policies (see also [5]).

Unrestricted Waiting (UW): A traveler may wait at any node for an arbitrary amount of time, prior to traversing an arc emanating from it.

Forbidden Intermediate Waiting (FIW): A traveler may wait only at the origin, for an arbitrary amount of time, prior to starting the journey towards the destination (without any other waiting at a node).

Forbidden Waiting (FW): No waiting is allowed, at any node in the network.

The general problem SOEAT was proved to be **NP**−hard, if the FW-policy is adopted and arc-delays are allowed *not* to possess the FIFO property. It may even be the case that there is no optimal-waiting (i.e., one that minimizes the earliest-arrival-time at the destination) at a node [5]. On the other hand, it is well known [2] that SOEAT is polynomial-time solvable when the UW-policy is adopted and the optimal-waiting time always exists for every node, independently of the shape of the arc-delay functions. Indeed, such a scenario is also known [5] to be equivalent to an appropriate FIFO network with the FW-policy.

For instances in which the FIFO property holds, the crucial property of subpath optimality holds:

Proposition 2.3. *Assume a graph with arc-delays satisfying the strict FIFO property. Then, for all vertices $u, v \in V$, any departure-time $t_u \in \mathbb{R}$ from u, and any optimal path*

$$p^* \in \arg\min_{p \in \mathcal{P}_{u,v}} \{Arr[p](t_u)\}$$

it holds that every subpath $q^* \in \mathcal{P}_{x,y}$ of p^* is a shortest path between its endpoints x, y for departure time from x equal to $t_x^* = Arr[p^*_{u \rightsquigarrow x}](t_u)$.

Proof. For sake of contradiction, assume that: $\exists q \in \mathcal{P}_{x,y} : D[q](t_x) < D[q^*](t_x)$. Then, $p = p^*_{u \rightsquigarrow x} \oplus q \oplus p^*_{y \rightsquigarrow v}$ suffers smaller delay than p^* for departure time t_u. Indeed, let $t_y \equiv t_x^* + D[q](t_x^*)$ and $t_y^* \equiv t_x^* + D[p^*_{x \rightsquigarrow y}](t_x^*)$. Due to the alleged suboptimality of $p^*_{x \rightsquigarrow y}$ when departing at time t_x^*, it holds that $t_y < t_y^*$. Then:

$$\begin{aligned} Arr[p](t_u) &= t_u + D[p](t_u) \\ &= \underbrace{t_u + D[p^*_{u \rightsquigarrow x}](t_u)}_{=t_x^*} + D[q](t_x^*) + D[p^*_{y \rightsquigarrow v}](t_x^* + D[q](t_x^*)) \\ &= \underbrace{t_x^* + D[q](t_x^*)}_{=t_y} + D[p^*_{y \rightsquigarrow v}](t_x^* + D[q](t_x^*)) = t_y + D[p^*_{y \rightsquigarrow v}](t_y) \\ &< t_y^* + D[p^*_{y \rightsquigarrow v}](t_y^*) = Arr[p^*](t_u) \end{aligned}$$

violating the optimality of p^* for the given departure-time t_u (the inequality is due to the strict FIFO property). □

Therefore, both Dijkstra's label setting algorithm and label-correcting algorithms for shortest uv−path computations in time-independent graphs, also work (with slight modifications in the label updates, e.g., see figure **??**) in time-dependent strictly FIFO networks, under the usual conventions that we consider for the static instances (positivity of arc-delays for the label setting, and inexistence of negative-delay cycles for the label correcting approaches).

A similar problem is to compute the value of the *latest-departure-time* function from the origin, between a given pair of origin-destination nodes and a given arrival time: Again, we could express the required function as the result of a continuous optimization problem as follows:

$$\begin{aligned} (9) \qquad \forall t_d \in \mathbb{R}, Dep[o,d](t_d) &= \max_{p \in \mathcal{P}_{o,d}} \{Dep[p](t_d)\} \\ &= \max_{v_1,\ldots,v_{n-2} \in V} \{Dep[o, v_1](\cdots(Dep[v_{n-2}, d](t_d)))\} \end{aligned}$$

where the second equality also holds FIFO networks. The corresponding combinatorial optimization problem is thus the following:

Definition 2.1. Latest Departure Times towards a Single Destination (SDLDT)

INPUT :	*Directed graph* $G = (V, A)$. *A unique destination vertex.* $d \in V \times V$: $t_d \in \mathbb{R}$: *The arrival-time at the destination vertex.* $\forall a \in A, \overleftarrow{D}[a] : \mathbb{R} \to \mathbb{R}_{\geq 0}$. *The* reverse arc-delay *functions.*
GOAL :	*The latest-departure-time values* $Dep[o, d](t_d)$ *from all possible origins* $o \in V$, *given the arrival-time* t_d *at* d. *Moreover, a tree of shortest-travel-time paths* $p^*[o, d] \in \mathcal{P}_{o,d}$ *achieving these departure-times, i.e., such that* $Dep[p^*[o, d]](t_d) = Dep[o, d](t_d)$.

3. Succinct Representations of Earliest-Arrival Functions

In case of huge networks, as is the case for either continental road networks, metropolitan-size urban networks, or social networks, it is rather impractical to use Dijkstra or a label-setting algorithm for every individual shortest path query, even in the stricter case of planar embedded graphs. For the time-independent case the issue has been tackled quite successfully both theoretically (using distance oracles) and in practice (using speed-up techniques). The main idea in both cases is to afford a costly preprocessing phase, that is nevertheless polynomial-time tractable, space efficient and amenable to relatively fast updates in case of dynamic changes in the graph, so that in real-time one can support extremely fast (in sub-linear / polylogarithmic / constant time theoretically, within microseconds in practice) arbitrary shortest path queries.

Both the theoretical and the practical approaches precompute distance-related information from / to specific subsets of nodes in the network. This precomputed information is stored and then used either as part of the direct shortest path calculations between arbitrary pairs of vertices, or in order to provide good lower bounds that are used to direct the search of a shortest path in a Dijkstra-like query algorithm. When applied to time-dependent instances, rather than storing shortest-path distances, one has to keep in memory the earliest-arrival-time / latest-departure-time functions from / to these particular nodes. Therefore, an important problem to be solved (and consequently used as a subroutine) is the following:

Definition 3.1. Earliest-Arrival-Time Functions from Single Origin (SOEAF)

INPUT :	*Directed graph* $G = (V, A)$. $o \in V$: *A particular vertex considered as the origin.* $\forall a \in A, \overrightarrow{D}[a] : \mathbb{R} \to \mathbb{R}_{\geq 0}$. *The* arc-delay *functions.*
GOAL :	*A* succinct representation *of the entire earliest-arrival-time functions* $Arr[o, d](t_o)$, *for any possible destination vertex* $d \in V$ *and all possible departure-times* $t_o \in \mathbb{R}$ *from* o.

The analogue of SOEAF for latest departures from various origins towards a single destination is called *Latest-Departure-Time Functions towards a Single Destination (SDLDF)* and simply considers the reverse arc-delay functions $\overleftarrow{D}[a] : \mathbb{R} \to \mathbb{R}_{\geq 0}$ and the latest-departure-time functions $Dep[o, d]$ from various origins to the single destination d. Alternatively, we might be interested in computing all the earliest-arrival-time functions from a given set of origins $O \subseteq V$, to any other vectex in the graph:

Definition 3.2. Earliest Arrival Time Functions from Multiple Origins (MOEAF)

INPUT :	*Directed graph* $G = (V, A)$. $R \subseteq V$: *A particular subset of potential origin-vertices.* $\forall a \in A, \overrightarrow{D}[a] : \mathbb{R} \to \mathbb{R}_{\geq 0}$. *The* forward arc-delay *functions.*
GOAL :	*A succinct representation of the entire earliest-arrival-time functions* $Arr[o, d](t_o)$, *for any possible origin-vertex* $o \in O$, *any destination-vertex* $d \in V$ *and arbitrary departure-times* $t_o \in \mathbb{R}$ *from the origin* o.

Once more, the analogue of MOEAF for the case of latest-departure-time functions from all origins to any vertex from a given set of destinations, is called *Latest-Departure-Time Functions towards Multiple Destinations (MDLDF).*

In the particular case where the arc-delay functions are continuous, piecewise-linear functions, we shall add the prefix "PWL-" to the identifier of the corresponding problem to be solved. Similarly, if all the arc-delay functions are linear, we shall denote this by a "LIN-" prefix. For example, PWL-SOEAT, LIN-SDLDT, PWL-SOEAF, LIN-MOEAF, etc.

3.1. **Linear and PWL Arc-Delays.** In this work we consider each arc-delay function is a piecewise liner, continuous, nonnegative function of the departure time from the arc's tail. Due to the assumption for periodicity, we are only provided with the restriction of the function within a single period. That is:

$$\forall a = uv \in A, \forall t_u \in \Pi, \overrightarrow{D}[a](t_u) = \begin{cases} \lambda_1^a \cdot t_u + \mu_1^a, & t_u \in [0, t_1^a] \\ \lambda_2^a \cdot t_u + \mu_2^a, & t_u \in [t_1^a, t_2^a] \\ & \vdots \\ \lambda_{K_a}^a \cdot t_u + \mu_{K_a}^a, & t_u \in [t_{K_a-1}^a, t_{K_a}^a = T] \end{cases} \tag{10}$$

For any other departure time $t_u \notin \Pi$, we assume that $\overrightarrow{D}[a](t_u) = \overrightarrow{D}[a](t_u \mod T)$. We denote by $\overrightarrow{\lambda}[a](t)$ and $\overrightarrow{\mu}[a](t)$ the corresponding *step functions* assigning the proper constant values for $\overrightarrow{\lambda}^a$ and $\overrightarrow{\mu}^a$, in order to express the pwl-function $\overrightarrow{D}[uv](t) = \overrightarrow{\lambda}[a](t) \cdot t + \overrightarrow{\mu}[a](t)$. The corresponding (pwl) arc-arrival-time function is $Arr[uv](t_u) = (1 + \overrightarrow{\lambda}[a](t_u)) \cdot t_u + \overrightarrow{\mu}[a](t_u)$. For a walk (path) $p = \langle a_1, \cdots a_k \rangle \in \mathcal{P}_{o,d}$ from the recursive definition, cf. equation (4), we know that $\forall 1 \leq i \leq k$:

$$Arr[p_{1,i}](t) = \begin{cases} Arr[a_1](t) = (1 + \overrightarrow{\lambda}[a_1](t)) \cdot t + \overrightarrow{\mu}[a_1](t), & i = 1 \\ \\ Arr[a_i](\underbrace{Arr[p_{1,i-1}](\tau)}_{\equiv t_{1,i-1}}), & i \geq 2 \\ \quad = \; [1 + \overrightarrow{\lambda}[a_i](t_{1,i-1})] \cdot Arr[p_{1,i-1}](\tau) + \overrightarrow{\mu}[a](t_{1,i-1}) \\ \quad = \; [1 + \overrightarrow{\lambda}[a_i](t_{1,i-1})] \cdot ([1 + \overrightarrow{\Lambda}[p_{1,i-1}](t)] \cdot t + \overrightarrow{M}[p_{1,i-1}](t)) + \overrightarrow{\mu}[a_i](t_{1,i-1}) \\ \quad = \; \underbrace{[1 + \overrightarrow{\lambda}[a_i](t_{1,i-1})] \cdot [1 + \overrightarrow{\Lambda}[p_{1,i-1}](t)]}_{\equiv\, 1 + \overrightarrow{\Lambda}[p_{1,i}](t)} \cdot t \\ \qquad + \underbrace{[1 + \overrightarrow{\lambda}[a_i](t_{1,i-1})] \cdot \overrightarrow{M}[p_{1,i-1}](t) + \overrightarrow{\mu}[a_i](t_{1,i-1})}_{\equiv\, \overrightarrow{M}[p_{1,i}](t)} \end{cases} \tag{11}$$

The succinct representation of $\overrightarrow{D}[a]$ is given by a collection (ordered list, by increasing time values) of triples:

$$\langle (\overrightarrow{\lambda}_i^a, \overrightarrow{\mu}_i^a, \overrightarrow{t}_i^a) : i \in \{1, \ldots, K_a\} \rangle$$

where the linear function describing the $i-$th leg of $\overrightarrow{D}[a]$ is $\overrightarrow{\lambda}_i^a \cdot t + \overrightarrow{\mu}_i^a$ and its valid interval is $[\overrightarrow{t}_{i-1}^a, \overrightarrow{t}_i^a]$ (we assume that $\overrightarrow{t}_0^a = 0$ and $\overrightarrow{t}_{K_a}^a = T$).

It is now relatively simple to determine the reverse arc-delay functions. Assume an arbitrary leg of the forward arc-delay function $\overrightarrow{D}[a]$ of $a = uv$ corresponding to departure time t_u from u, that is determined by $(\overrightarrow{\lambda}[a](t_u), \overrightarrow{\mu}[a](t_u))$. Then, assuming t_v is the arrival-time at v, it must hold that $t_v - t_u = \overrightarrow{D}[a](t_u) = \overrightarrow{\lambda}[a](t_u)t_u + \overrightarrow{\mu}[a](t_u)$. The corresponding reverse arc-delay function $\overleftarrow{D}[a]$ is given by another linear function, but still indicates the same delay of the very same arc: $t_v - t_u = \overleftarrow{D}[a](t_v) = \overleftarrow{\lambda}[a](t_v)t_v + \overleftarrow{\mu}[a](t_v)$. Therefore we have:

$$\begin{aligned} & t_v = t_u + \overrightarrow{D}[a](t_u) = (1 + \overrightarrow{\lambda}[a](t_u))t_u + \overrightarrow{\mu}[a](t_u) \\ \overset{/*\ 1+\overrightarrow{\lambda}[a](t_u)>0\ */}{\Rightarrow} \quad & t_u = \frac{t_v - \overrightarrow{\mu}[a](t_u)}{1 + \overrightarrow{\lambda}[a](t_u)} \end{aligned}$$

Consequently, we can compute the reverse arc-delay functions:

$$\begin{aligned}\overleftarrow{D}[a](t_v) &= t_v - t_u = t_v - \frac{t_v - \overrightarrow{\mu}[a](t_u)}{1+\overrightarrow{\lambda}[a](t_u)} = \frac{\overrightarrow{\lambda}[a](t_u)t_v + \overrightarrow{\mu}[a](t_u)}{1+\overrightarrow{\lambda}[a](t_u)} \\ &= \underbrace{\frac{\overrightarrow{\lambda}[a](t_u)}{1+\overrightarrow{\lambda}[a](t_u)}}_{\equiv \overleftarrow{\lambda}[a](t_v)} \cdot t_v + \underbrace{\frac{\overrightarrow{\mu}[a](t_u)}{1+\overrightarrow{\lambda}[a](t_u)}}_{\equiv \overleftarrow{\mu}[a](t_v)} = \overleftarrow{\lambda}[a](t_v)\cdot t_v + \overleftarrow{\mu}[a](t_v)\end{aligned} \tag{12}$$

This implies that for each leg of $\overrightarrow{D}[a]$ there is a corresponding leg of $\overleftarrow{D}[a]$, which is easily computed according to equation (12) in time linear in the representation of $\overrightarrow{D}[a]$: It suffices to check the $\overrightarrow{\lambda}[a]$ and $\overrightarrow{\mu}[a]$ values only at the breakpoints of $\overrightarrow{D}[a]$, in order to build $\overleftarrow{D}[a]$. The continuity of $\overleftarrow{D}[a]$, along with the FIFO property, assure that also $\overleftarrow{D}[a]$ is a continuous pwl function expressing exactly the same delays of a, only now expressed as functions of arrival-times at the head. An example for the reversion of the pwl continuous (periodic) function of figure 1 was demonstrated in figure 2.

4. How to Solve PWL-SOEAT and PWL-SDLDT

As already mentioned (cf. Proposition 2.3), when the (strict) FIFO property holds, *subpath optimality* holds also in time-dependent instances. Simple variants of Dijkstra indeed work also for the computation of shortest $od-$paths and earliest-arrival-time values (for given departure time from origin) in any time-dependent network possessing the FIFO property [2]. We denote such a time-dependent variant by **TDD**. To avoid tricky situations in which the algorithm (even for static networks) might fail, we suppose that all the arc-delay functions are always *non-negative*. Put it differently, we consider as the actual arc-delay to be the maximum of zero and the declared arc-delay function, for any departure time from the tail.

4.1. **Evaluating Arc Delays.** During the execution of **TDD** in a FIFO network with (non-negative) arc-delay functions, the delay value of every arc has to be estimated upon its (unique) relaxation, when its head is settled. For LIN-SOEAT this would definitely have cost $\mathcal{O}(1)$. The case of PWL-SOEAT is a little bit more complicated: When referring to the description ($\lambda-$ and $\mu-$ values) of an arc-delay function for the arc $a = uv$ that is currently being relaxed for a given departure time $t_u = Arr[o,u](t_o)$, the arc-delay evaluation operation is not constant anymore, but costs either $\mathcal{O}(\log(K_a))$ (e.g., by maintaining a binary search tree of breakpoints) or even $\mathcal{O}(\log(\log(K_a)))$ if one employs more advanced data structures (e.g., fast tries of breakpoints) in order to determine the appropriate leg of the (pwl) arc-delay function $D[a]$ which is appropriate for t_u. K_a is the space-complexity (i.e., the number of breakpoints) of $D[a]$. Since every arc is relaxed at most once, in overall **TDD** will have time-complexity $\mathcal{O}(n\log(n) + m\cdot\log(\log(K_{\max})))$ to solve PWL-SOEAT, where $K_{\max} = \max_{a\in A} K_a$.

4.2. **Solving SDLDT.** In order to solve either LIN-SDLDT or PWL-SDLDT, the arc delay functions, (i.e., we consider and then run **TDD**, the only differences being that:

- Within Q the objects are ordered in decreasing departure-times from the tails of the arcs.
- Each $Q.pop()$ operation retrieves the object with the maximum key.
- Relaxation of arc $vu \in \overleftarrow{A}$ occurs during the settlement of the *tail vertex* v, when the *subtraction* of the reverse-arc-delay value $\overleftarrow{D}[uv](t_v)$ from the actual arrival time t_v at v is *greater than* the current value of vertex v in Q.

4.3. **Instantaneous Descriptions of Earliest Arrival Functions at Sampled Points.** As mentioned in [2], a slight modification of Dijkstra that relaxes vertex labels according to the *temporal* arc-travel times of their incoming arcs, depending on the departure-times from their heads, works perfect in time-dependent networks possessing the FIFO property, for arbitrary (but given) departure-times t_o from the origin o. Eventually, the labels of the vertices reachable from o denote the earliest-arrival-time *values*, when one departs from o at the given departure time t_o.

Our purpose in this subsection is to explain how one can gather additional information, concerning the instantaneous functional descriptions of the earliest-arrival-time *functions* at nodes

reachable from the origin o, with respect to an arbitrary sampling departure-time t_o. In particular, our goal is to provide the description of the (affine) earliest-arrival-time functions:

$$\begin{aligned} Arr^-[o,v](x) &= A^-[o,v](t_o)\cdot x + B^-[o,v](t_o),\ x\in(t_o-\delta,t_o] \\ Arr^+[o,v](x) &= A^+[o,v](t_o)\cdot x + B^+[o,v](t_o),\ x\in[t_o,t_o+\delta) \end{aligned}$$

to each node $v\in V$, for arbitrarily small $\delta>0$.

The main idea is, after having executed $\mathbf{TDD}(G,D,o,t_o)$, which created not only the earliest-arrival-time values at the final vertex labels $L[v]$, but also the shortest-paths tree T assuring them, to execute a BFS scan in T (starting from the root o) in order to recursively compute the above mentioned functional descriptions of earliest-arrival-time functions. Before describing the appropriate formula, we need some additional notation. The set

$$P[o,v](t_o)=\left\{u\in V:(u,v)\in A\wedge L[v]=L[u_v^i]+D[u_v^iv](L[u_v^i])\right\}$$

contains all the parents of v in shortest $ov-$paths with departure time t_o.

$$A^{\pm}[o,o](t_o)=1,\ B^{\pm}[o,o](t_o)=0$$

$\underline{P[o,v](t_o)=\{u\},\ \text{for some } u\in V:}$ /* unique shortest $ov-$path parent for departure-time t_o */

$$\begin{aligned} A^{\pm}[o,v](t_o) &= (1+\lambda^{\pm}[uv](Arr^{\pm}[o,u](t_o)))\cdot A^{\pm}[o,u](t_o) \\ B^{\pm}[o,v](t_o) &= (1+\lambda^{\pm}[uv](Arr^{\pm}[o,u](t_o)))\cdot B^{\pm}[o,u](t_o)+\mu^{\pm}[uv](Arr^{\pm}[o,u](t_o)) \end{aligned}$$

$\underline{|P[o,v](t_o)|\geq 2:}$ /* multiple shortest $ov-$path parents for departure-time t_o */

$$\begin{aligned} A^-[o,v](t_o) &= \max_{u\in P[o,v](t_o)}\{(1+\lambda^-[uv](Arr^-[o,u](t_o)))\cdot A^-[o,u](t_o)\} \\ B^-[o,v](t_o) &= \min_{u\in P[o,v](t_o)}\{(1+\lambda^-[uv](Arr^-[o,u](t_o)))\cdot B^-[o,u](t_o)+\mu^-[uv](Arr^-[o,u](t_o))\} \\ A^+[o,v](t_o) &= \min_{u\in P[o,v](t_o)}\{(1+\lambda^+[uv](Arr^+[o,u](t_o)))\cdot A^+[o,u](t_o)\} \\ B^+[o,v](t_o) &= \max_{u\in P[o,v](t_o)}\{(1+\lambda^+[uv](Arr^+[o,u](t_o)))\cdot B^+[o,u](t_o)+\mu^+[uv](Arr^+[o,u](t_o))\} \end{aligned}$$

where the arc-travel-time function of an arc $a=uv$ may also have a prior-description $(\lambda^-[uv],\mu^-[uv])$ and a post-description $(\lambda^+[uv],\mu^+[uv])$, if we depart from u at a breakpoint time of a.

Proposition 4.1. *Consider a time-dependent instance $\langle G=(V,A),(D[a]:[0,T]\to\mathbb{R}_{>0})_{a\in A}\rangle$ with* strictly positive *arc-travel-time functions. Assume that the instantaneous functional descriptions of earliest-arrival-time functions are computed as described, after the completion of the time-dependent Dijkstra run with departure time t_o from the origin, by considering the vertices of the graph exactly in the same order as they were settled (i.e., as if we scan the shortest paths tree in bfs order, with the vertices of each level ordered by increasing settling times). Then it holds that:*

$$\begin{aligned} Arr^-[o,v](x) &= A^-[o,v](t_o)\cdot x + B^-[o,v](t_o),\ x\in(t_o-\delta,t_o] \\ Arr^+[o,v](x) &= A^+[o,v](t_o)\cdot x + B^+[o,v](t_o),\ x\in[t_o,t_o+\delta) \end{aligned}$$

to each node $v\in V$, for arbitrarily small $\delta>0$.

Proof. The explanation of this proposition is based on an inductive argument on the vertices whose functional descriptions have already been computed. The basis of the induction concerns the root itself, whose functional description is trivially correct. Assume now that all the vertices that have been processed so far already have the correct instantaneous descriptions of their earliest-arrival functions. Consider the next vertex $v_k\in V$ to process. Let $V_k=\{o=v_1,v_2,\ldots,v_{k-1}\}$ be the set of already processed vertices. Clearly, $\forall 1\leq i<k<j\leq n$ it holds that $L[v_i]\leq L[v_k]\leq L[v_j]$, by correctness of time-dependent Dijkstra. Due to the *positivity* of the arc-delay values, it certainly holds that $P[o,v_k](t_o)$ may only contain nodes from V_k, whose instantaneous descriptions of earliest-arrival-time functions. In case that $P[o,v_k](t_o)=\{u\}$ for some vertex $u\in V$, there is a sufficiently small $\delta>0$ such that u is the unique shortest-path parent of v_k, for all departure times $t\in(t_o-\delta,t_o+\delta)$, by continuity of the earliest-arrival functions. Thus, by the (inductively assumed) correctness of the instantaneous functional description for vertex u's earliest arrival, it certainly holds that v_k's functional description is also correct. When $|P[o,v_k](t_o)|\geq 2$, then for each $u\in P[o,v_k](t_o)$ we have a different earliest-arrival-time-via-u function, $Arr^{\pm}[o,v|u](t)$. The recursive description of these functions, based on their unique choice for a parent of v_k, are given

in equation (11). All these are *affine* functions that meet at the point $(t_o, L[v])$. We want to express $Arr^{\pm}[o,v]$ around t_o as the minimum over these affine functions. Clearly, $Arr^{-}[o,v](t)$ has the largest slope and the smallest constant, whereas $Arr^{+}[o,v](t)$ has the smallest slope and the larger constant among these affine functions, as shown also in the following figure: □

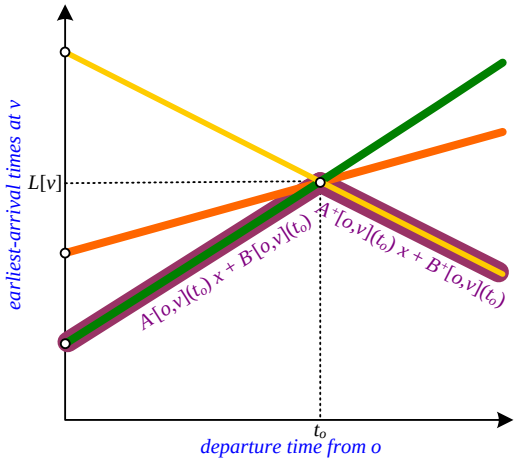


FIGURE 3. Instantaneous functional descriptions of the earliest-arrival-time function around the sampling departure-time t_o from the origin.

4.4. **Time Horizon of Combinatorial Structures of Given Departure Times.** After running **TDD** for a given departure time t_o and consequently computing the instantaneous functional descriptions of the earliest-arrival functions to destinations, our last task is to determine the *time horizon* $\hat{t}_o > t_o$ until which this information will remain valid. The reasons for such a change might be either a future arc-delay breakpoint activation, or the effect of a minimization operation at a destination vertex. Our approach for this computation is inspired by the output-sensitive algorithm of Foschini et al. **??**, which introduced the related notion of *certificates*. The time-horizon that we seek is exactly the earliest certificate failure that we shall discover in the graph.

In order to discover these discrete points at which the combinatorial structure (shortest path tree) and / or some earliest-arrival-time functions change (due to appearance of new breakpoints), we compute a set of *certificates* (one per vertex and edge) to indicate the next failure time of some earliest-arrival function, triggered by a particular element of the graph as if nothing else would change in the future.

The notion of *minimization (vertex) certificates*, one per vertex $v \in V$, provides estimations on the earliest future departure-time $t_{fail}[v](t_o)$ from the origin with respect to t_o, at which the shortest $ov-$path would change, due to the application (at v) of the minimization operation at the earliest arrival functions via different parents, assuming that no other functional description would change in the graph. Similarly, the notion of *primitive (arc) certificates*, one per arc $a = uv \in A$, indicates the projection $t_{fail}[a]$ (to a departure-time from the origin) of the next breakpoint-time (after $Arr[o,u](t_o)$) to the arc-travel-time function $D[a]$.

In particular, assume that for every vertex $v \in V$ its in-neighborhood is $IN[v] = \{u \in V : uv \in A\}$ and let for convenience $u_v = p[v](t_o)$ be the current parent of v in the shortest paths tree $SPT[o](t_o)$ for departure time t_o. Recall that the instantaneous *earliest-arrival-time-via* functions (starting from departure-time t_o) are determined as follows:

$$\begin{aligned} \forall u \in IN[v],\ Arr^{+}[o,v|u](t) &= Arr^{+}[o,u](t) + D^{+}[uv](Arr^{+}[o,u](t)) \\ = \quad & (1 + \lambda^{+}[uv](Arr^{+}[o,u](t)) \cdot A^{+}[o,u](t_o) \cdot t \\ & + (1 + \lambda^{+}[uv](Arr^{+}[o,u](t))) \cdot B^{+}[o,u](t_o) + \mu^{+}[uv](Arr^{+}[o,u](t)) \end{aligned}$$

where, $Arr^{+}[o,v](t_o) = Arr^{+}[o,v|u_v](t_o) < Arr^{+}[o,v|u](t_o),\ \forall u \in IN[v] \setminus \{u_v\}$. Assuming that the recursively called earliest-arrival-via functions at $Arr^{+}[o,v|u](t) : u \in IN[v]$ would remain affine from t_o and beyond, the next *minimization certificate* at vertex v is relatively simple to

compute: It is the earliest point $t_{fail}[v] > t_o$ (if any) at which any of the (suboptimal at t_o) alternative earliest-travel-time functions become equal to the value of $Arr[o,v](t_{fail}[v])$. For simplicity in notation we drop the dependence of all the functional descriptions from the departure-time from the orign o (for earliest-arrival functions), or from the departure-time $Arr^+[o,u](t)$ from the tail u of arc uv: $\forall u \in IN[v] \setminus \{u_v\}$,

$$t_{fail}[v|u](t_o) \quad = \quad \begin{cases} +\infty, & \begin{array}{c} A^+[o,v] \leq (1+\lambda^+[uv]) \cdot A^+[o,u] \\ \vee \\ B^+[o,v] \geq (1+\lambda^+[uv]) \cdot B^+[o,u] + \mu^+[uv] \end{array} \\ \frac{(1+\lambda^+[uv]) \cdot B^+[o,u] + \mu^+[uv] - B^+[o,v]}{A^+[o,v] - (1+\lambda^+[uv]) \cdot A^+[o,u]}, & \text{otherwise.} \end{cases}$$

The certificate failure of vertex v is then the earliest failure indication from all the possible alternative routes to reach v:

$$t_{fail}[v](t_o) = \inf_{u \in IN[v] \setminus \{u_v\}} \{\ t_{fail}[v|u](t_o)\ \} \tag{13}$$

We must also deal with the future primitive (arc) breakpoints of the earliest-arrival functions, which are indeed caused by the fact that the arc-travel-time functions are themselves piecewise linear. For this reason, for every arc $a \in A$ and departure-time t_o from o, we must know the closest future departure time from o (if any) for which the earliest arrival time at $tail[a]$ is the time-coordinate of a breakpoint in $D[a]$, *assuming* that no other breakpoint would happen to affect the earliest arrival function at $tail[a]$. We call this departure time from o a *primitive certificate* for arc a. This is defined as follows:

$$t_{fail}[a](t_o) \tag{14}$$

$$= \begin{cases} +\infty, & A[tail[a]](t_o) \cdot t_o + B[tail[a]](t_o) > t^a_{k_a} \\ \min\left\{t \geq t_o : A[tail[a]](t_o) \cdot t + B[tail[a]](t_o) \in \{t^a_1, \ldots, t^a_{k_a}\}\right\}, & \text{otherwise.} \end{cases}$$

It is mentioned that, in order to compute the instantaneous earliest-arrival-time functional descriptions and the tentative certificate failure times, one has to perform two sequential passes (in BFS order) over the vertices of the shortest paths tree produced by the execution of the time-dependent Dijkstra. This is because the first pass will compute the earliest-arrival-time functions, and only then can one (in the second pass of the same tree, in any order) recalculate the correct values of all the tails of arcs headed from vertices of the subtree. Figure 4 gives a brief explanation of the necessity for two passes.

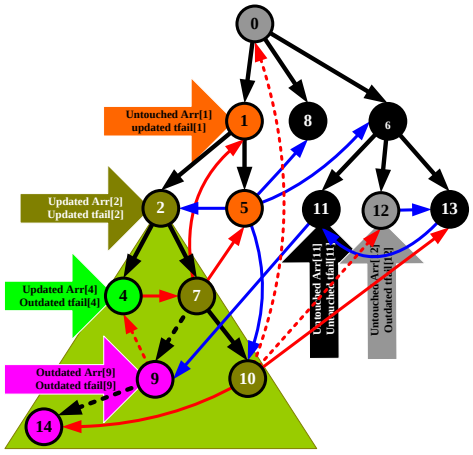


FIGURE 4. Explanation of the necessity for two passes of the acrs headed by vertices in the subtree of the graph element causing the current certificate failure. Black lines indicate tree-arcs in $SPT[o](t_o)$. Red/Blue arcs indicate non-tree arcs. The assumption is that the next certificate failure occurs due to vertex 2. The (green) subtree routed at this vertex is the one whose combinatorial structure is affected. Red (non-tree) arcs have their tail in this subtree rooted at vertex 2, and blue (non-tree) arcs have their tail outside this tree. Therefore, the functional descriptions and/or certificate at the read arcs and at their heads have to be updated, but no update is necessary due to blue arcs. Solid arcs indicate that the earliest-arrival functional descriptions at the heads have already been updated, or remain untouched. Dotted arcs indicate that, despite the necessity for update, this has not been done yet by the (first) BFS pass in $SPT[o]$. A node can update its certificate only when all the incoming arcs are solid. Otherwise, some of the functional descriptions are outdated.

5. Approximation Algorithms for Shortest-Travel-Time (Delay) Functions

In the present section we shall deal with the problem of providing (the explicit representations of) shortest-travel-time functions in a time-dependent instance. In particular, we shall deal with the following problems:

Definition 5.1. Time Dependent Delay Approximation Functions (TDDA)

INPUT :	*Directed graph* $G = (V, A)$. $(o, d) \in V \times V$*: An origin-destination pair of vertices.* $O \subseteq V$*: A subset of potential origin vertices.* $\forall a \in A, \overrightarrow{D}[a] : \mathbb{R} \rightarrow \mathbb{R}_{\geq 0}$. *The* forward arc-delay *(pwl) functions.*
SPTDDA :	*Provide (the explicit representation of) an approximate shortest-travel-time (delay) function* $\overline{D}[o, d]$ *for* $D[o, d]$*, that will assure the following approximation guarantee:* $\forall t_o \in [0, T], D[o, d](t_o) \leq \overline{D}[o, d](t_o) \leq (1 + \varepsilon) \cdot D[o, d](t_o)$.
SOTDDA :	*Provide (explicit representations of) approximate shortest-travel-time (delay) functions* $\overline{D}[o, v]$ *for all* $v \in V$*, that will assure the following approximation guarantee:* $\forall v \in V,\ \forall t_o \in [0, T], D[o, v](t_o) \leq \overline{D}[o, v](t_o) \leq (1 + \varepsilon) \cdot D[o, v](t_o)$. *Notation: The representation of the entire output will be denoted by* $\overline{\mathbf{D}}[o, \star]$.
MOTDDA :	*Provide (explicit representations of) all the approximate shortest-travel-time (delay) functions* $\overline{D}[o, d]$ *for any origin-destination pair of vertices* $(o, v) \in O \times V$*, that will assure the following approximation guarantees:* $\forall o \in O,\ \forall v \in V,\ \forall t_o \in [0, T], D[o, v](t_o) \leq \overline{D}[o, v](t_o) \leq (1 + \varepsilon) \cdot D[o, v](t_o)$. *Notation: The representation of the entire output will be denoted by* $\overline{\mathbf{D}}[O, \star]$.

The main criteria for the quality of solutions to the above mentioned problems are: (i) the polynomial-time construction of the required functions, and (ii) the required storage for the produced representations. Focusing only on (ii), one could have asymptotically optimal solutions, assuming prior knowledge (or, paying for the required computational cost and space to construct them) of the exact delay functions. For example, the technique of Imai and Iri [4] assures that the space-complexity (number of breakpoints) of the produced approximation function of a pwl-function is asymptotically optimal. Nevertheless, it is indeed required that the original function to approximate is a priori given. This is not the case in our setting and it would probably be prohibitively expensive to construct the exact delay functions before space-optimally approximating them. Therefore, we have to be based only on (polynomial-time computable) samplings of the exact functions, in order to produce (as fast as possible) the required upper approximations, using as little space and assuring as good approximation guarantee as possible.

Since we wish to avoid computing the exact shape of $D[o, d]$ for a given od-pair, we need also a *lower-bounding point-to-point approximate distance* function for the same time-window:

$$\forall t_o \in [0, T], (1 - \varepsilon) \cdot D[o, d](t_o) \leq \underline{D}[o, d](t_o) \leq D[o, d](t_o) \tag{15}$$

Having two (guaranteed upper and lower) approximations $\overline{D}[o, d]$ and $\underline{D}[o, d]$ of $D[o, d]$ in the time-window of interest and an approximation guarantee between them:

$$\forall t_o \in [0, T],\ \overline{D}[o, d](t_o) \leq (1 + \varepsilon) \cdot \underline{D}[o, d](t_o) \tag{16}$$

would assure us that $\overline{D}[o, d] \leq (1 + \varepsilon) D[o, d]$ is the required upper-approximation and $\underline{D}[o, d] \geq \left(1 - \frac{\varepsilon}{1+\varepsilon}\right) D[o, d]$ is the required lower-approximation of $D[o, d]$, without even knowing (except for the explicitly sampled values) the actual shape of the function that is approximated.

As in Foschini et al. [3], we start by presenting an approximation algorithm for subintervals $[t_s, t_f] \subseteq [0, T]$ in which $D[o, d]$ is a concave function. Foschini et al. then project all the K arc-delay function primitive breakpoints to (latest) departure-times (the so-called *primitive images*) from the origin o in $[0, T]$, and apply the proposed approximation algorithm for every subinterval between consecutive primitive images. In this case, in each subinterval all the arc-delay functions are *affine* and thus path-delay functions, as compositions of affine functions, are also affine functions. Finally, shortest-travel-time function $D[o, d]$ is a concave function, as the minimization

operator on a set of affine (path-delay) functions. Indeed, in order to preserve concavity of $D[o,d]$, one only has to project to departure-times from the origin only those arc (primitive) breakpoints that might spoil the concavity of arc-delay functions. Figure 5 demonstrates such breakpoints of an arc-delay function.

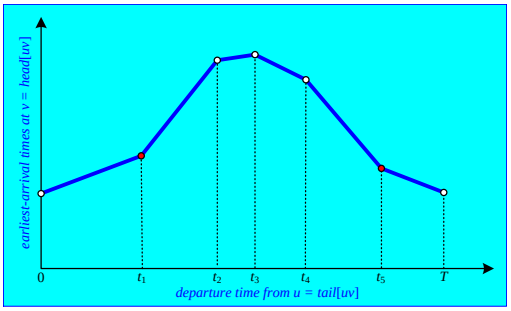


FIGURE 5. Concavity-spoiling breakpoints of arc-delay functions. Only the (red) primitive breakpoints t_1, t_5 spoil the concavity of $D[uv]$, and possibly the concavity of $D[o,d]$. These breakpoints have to be projected (as primitive images) to departure-times from the origin o, for every arc in the network. Observe that the (positive) slope at time t_1 increases, while the (negative) slope at t_5 also increases. In all other (non-concavity-spoiling) breakpoints the arc-delay slopes decrease.

This information (of concavity-spoiling primitive images) can be preprocessed and then be considered as part of the input, or can be computed prior to approximating the unknown function $D[o,d]$ by applying backwards time-dependent Dijkstra probes from all the tails of arcs possessing breakpoints in their delay functions. From now on we focus on time-subintervals $[t_s,t_f] \subseteq [0,T]$ that are between *consecutive* primitive images of all these concavity-spoiling arc-delay breakpoint times, for all arcs in the network. For any such subinterval, during which any arc-delay function is indeed an affine function, we shall provide the required upper-approximation of $D[o,d]$. The union of all these approximating functions will then be the upper-approximation of $D[o,d]$ in the entire domain $[0,T]$. So, fix a departure-time interval $[t_s,t_f]$ from the origin, during which all the arc-delay functions are *affine* (rather than pwl). For any vertex $v \in V$ reachable from an origin-vertex o, let $D_{\max}[o,v] = \max_{t\in[t_s,t_f]}\{D[o,v](t)\}$ and $D_{\min}[o,v] = \min_{t\in[t_s,t_f]}\{D[o,v](t)\}$. It is easy to observe that, due to the linearity of the arc-delays, the (actual) shortest-travel-time functions $D[o,v]$ to be approximated are pwl, continuous, *concave* functions in $[t_s,t_f]$. The concavity is due to the existence solely of minimization breakpoints (at vertices), and no primitive (arc-delay) breakpoints at all. Therefore, the sequence of partial derivates (i.e., constant slopes of affine legs) of each function $D[o,v]$ is non-increasing within $[t_s,t_f]$. Also recall that the strict FIFO property is assumed to hold (cf. 8, with strict inequality holding).

It is not hard to see that the maximum value of $D[o,d]$ within $[t_s,t_f]$ cannot exceed the value $2 \cdot D[o,d]\left(\frac{t_s+t_f}{2}\right)$, with respect to mid-point of the time-window of interest[1]:

Proposition 5.1. $\forall (o,d) \in V \times V,\ D_{\max}[o,d] \leq 2 \cdot D[o,d]\left(\frac{t_s+t_f}{2}\right)$.

Proof. We exploit the assumptions for $D[o,d]$'s non-negativity and concavity. Observe that, for any $t \in [t_s,t_f]$, its symmetric point $\bar{t} = t_s + t_f - t$ with respect to $\frac{t_s+t_f}{2}$ is also in $[t_s,t_f]$. Therefore, by the definition of concavity and the positivity of $D[o,d]$ we conclude that $\forall t, \bar{t} = t_s + t_f - t \in$

[1]This fact was also mentioned, without any justification, by Foschini et al. Here we confirm it by providing a quite simple and clear proof.

$[t_s, t_f]$,

$$\begin{aligned}
\max\left\{\frac{D[o,d](t)}{2}, \frac{D[o,d](\bar{t})}{2}\right\} &\leq \frac{D[o,d](t) + D[o,d](\bar{t})}{2} \\
&\leq D[o,d]\left(\frac{t+\bar{t}}{2}\right) = D[o,d]\left(\frac{t_s+t_f}{2}\right) \\
\implies \max\{D[o,d](t), D[o,d](\bar{t})\} &\leq 2 \cdot D[o,d]\left(\frac{t_s+t_f}{2}\right).
\end{aligned}$$

□

Due to being a concave chain, the minimum $D_{\min}[o,d]$ within $[t_s, t_f]$ is attained at one of the endpoints. Therefore, although we do not want to compute explicitly $D[o,d]$, we already know that it is enclosed in the following bounding box:

$$BB[o,d](t_s,t_f) = [t_s,t_f] \times \left[\min\left\{D[o,d](t_s), D[o,d](t_f)\right\}, 2 \cdot D[o,d]\left(\frac{t_s+t_f}{2}\right)\right] \tag{17}$$

This bounding box can be determined by just 3 calls of **TDD** per od-pair. On the other hand, if we wish to do the same for a particular origin vertex $o \in V$ and all the vertices reachable from it, we still need only 3 calls of **TDD**. Recall also that each call of **TDD** from a given origin-vertex $o \in V$ does not only provide the earliest-arrival-times (and also the shortest-travel-times) for a given departure-time t_o from o, but also the instantaneous functional descriptions of $Arr[o,v]$ per vertex in the shortest paths tree rooted at o, $SPT[o](t_o)$. In particular, for some sufficiently small $\delta > 0$ and each vertex $v \in V$, it holds that:

$$\forall t \in [t_o - \delta, t_o], \qquad Arr[o,v](t) = A^-[o,v](t_o) \cdot t + B^-[o,v](t_o)$$

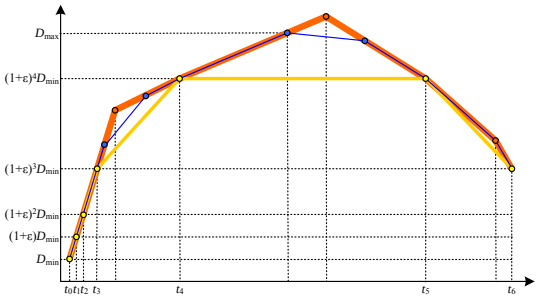
$$\forall t \in [t_o, t_o + \delta], \qquad Arr[o,v](t) = A^+[o,v](t_o) \cdot t + B^+[o,v](t_o)$$

These functional descriptions are recursively determined after the execution of **TDD** for departure-time t_o (cf. subsection 4.3). Finally, by the certificates structure we are also able to know the time-horizon $T_{fail}(t_o)$ up to which the current solution is valid (cf. subsection 4.4).

5.1. **Approximating Delay Functions for a Particular OD-Pair.** We shall now present an algorithm for computing an upper and a lower approximation of $D[o,d]$, $\overline{D}[o,d]$ and $\underline{D}[o,d]$ which are within a factor of $1+\varepsilon$ of each other, for a *given* OD-pair $(o,d) \in V \times V$. It is based on an algorithm proposed by Foschini et al. [3, Section 5.3]. Our algorithm is a refinement of that approximation technique, based on a careful and detailed analysis that will help us in being more accurate in the description of the algorithm itself. More importantly, this accurate estimation of the maximum error in the approximation guarantee will help us also assure that the space complexity of the returned upper-approximation is at most double the space complexity of any pwl-approximation of $D[o,d]$ that maintains the same approximation guarantee. This is guaranteed by the approach of Foschini et al. only for one part (the bisection) of their approximation algorithm. There is no guarantee at all about the first part of the algorithm which samples fast-growing delay functions. Indeed, it is straightforward to create instances in which the approximation algorithm of Foschini et al. fails arbitrarily bad with respect to the minimum required number of breakpoints, when for example $D[o,d]$ is an "almost linear", rapidly growing function with very small starting delay value. We achieve a global guarantee for our space complexity, exactly due to the careful analysis and measurement of this absolute error which allows us to discard all the unnecessary sample points. The time-complexity of our algorithm is the same as that of Foschini et al., since the set of sampled points is produced similarly.

5.1.1. *Upper Bound on the Required Space Complexity.* A simple way to get an upper bound on the minimum number of required sample points for a guaranteed $(1+\varepsilon)-$approximation of $D[o,d]$, is the following: If we knew the explicit description of $D[o,d]$, we could use as sample points only the intersection points of $D[o,d]$ with parallels of the time axis at travel-time values $D_{\min}[o,d]$, $(1+\varepsilon)D_{\min}[o,d]$, $(1+\varepsilon)^2 D_{\min}[o,d], \ldots$ (which intersect $D[o,d]$ in at most two points). The interpolation of these sample points would then provide $\underline{D}[o,d]$, whereas $\overline{D}[o,d]$ would be the lower envelope of the tangents of $D[o,d]$ at the sample points. Clearly the required guarantee expressed in (16) is fulfilled, by construction of $\overline{D}[o,d]$ and $\underline{D}[o,d]$. The following figure shows

such an example, where $D[o,d]$ is shown by the blue line, $\underline{D}[o,d]$ is the yellow line, and $\overline{D}[o,d]$ is the orange line:



Therefore, we conclude that a space-optimal upper approximation $\overline{D}^*[o,d]$ of $D[o,d]$ would require at most:

$$|BP(\overline{D}^*[o,d])| \leq 4 \cdot \left\lceil \log_{1+\varepsilon} \left(\frac{D_{\max}[o,d]}{D_{\min}[o,d]} \right) \right\rceil \tag{18}$$

interpolation points ($\overline{D}^*[o,d]$ may have at most twice as many breakpoints as the sampled points).

5.1.2. *The Technique of Foschini et al.* Clearly, we do not wish to explicitly compute $D[o,d]$ because this is already too expensive. The main idea proposed in Foschini et al. [3] is that, since we can easily determine the bounding box $BB[o,d](0,T)$ of $D[o,d]$, it would suffice to *sample* $D[o,d]$ at particular points of either the horizontal axis $[0,T]$, or the vertical axis $[D_{\min}[o,d], D_{\max}[o,d]]$, in such a way that an overall approximation ratio of $1+\varepsilon$ be assured between the produced approximations $\overline{D}[o,d]$ and $\underline{D}[o,d]$. The choice of axis to sample depends on the slope of the function being sampled. The goal is always to sample the *faster changing* axis, in order to assure the approximation guarantee. Therefore, so long as the shortest-travel-time slope (i.e., of function $D[o,d]$) is greater than 1 we shall keep sampling the vertical (travel-time) axis, as follows: Departing from o at time $t_s = t_0$, we compute a forward call of **TDD**, to get an earliest-arrival-time at d, $t_0 + D[o,d](t_0)$. We then carefully delay the arrival-time, ie, we consider the arrival-time $t_1 + D[o,d](t_1) = t_0 + \sqrt{1+\varepsilon} \cdot D[o,d](t_0)$ and we perform a backward call of **TDD**, in order to determine the proper (latest) departure-time t_1 from o. This procedure is repeated until the slope of $D[o,d]$ drops below 1, in which case the first phase of the approximation algorithm is stopped. Figure 6 demonstrates exactly this phase. The second phase of Foschini et al.'s approximation

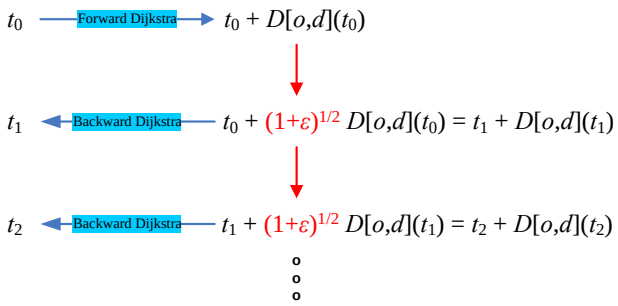


FIGURE 6. The first phase of Foschini et al.'s aproximation algorithm. The forward call determines an earliest-arrival-time at d (for given departure-time from o) and each backward call determines the latest-departure-time from o (for given arrival-times at d).

algorithm is a typical bisection that keeps bisecting the time-axis until the required approximation guarantee is achieved.

When the sampling procedure is complete, we consider $\underline{D}[o,d]$ to be just the interpolant of the sampled points of $D[o,d]$, while $\overline{D}[o,d]$ is the lower envelope (minimum) of all the supporting lines (tangents) of $D[o,d]$ at the sampled points. Figure 7 provides an example of a shortest-travel time function and the two bounding approximations. The facts that, with respect to $D[o,d]$, (i)

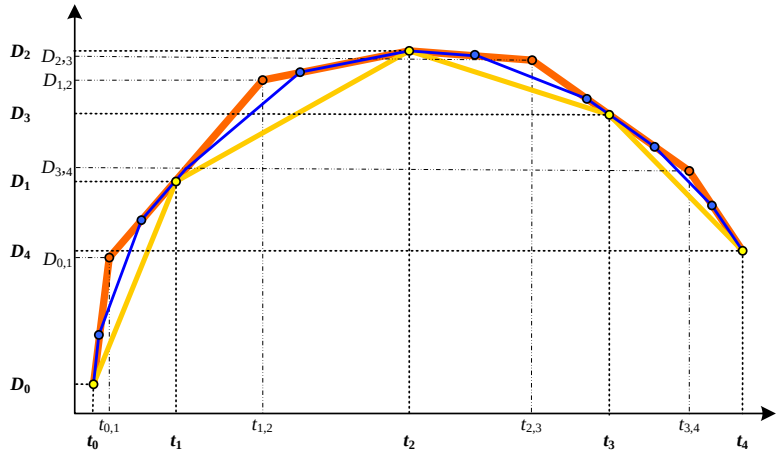


FIGURE 7. An example of the considered pwl approximations $\overline{D}[o,d]$ (orange line) and $\underline{D}[o,d]$ (yellow line) of the pwl function $D[o,d]$ (blue line). The sampled points of $D[o,d]$ are $\{(t_0,D_0),(t_1,D_1),(t_2,D_2),(t_3,D_3),(t_4,D_4)\}$ and they are as the interpolation points of $\underline{D}[o,d]$. $\overline{D}[o,d]$ also has some additional interpolation points $\{(t_{0,1},D_{0,1}),(t_{1,2},D_{1,2}),(t_{2,3},D_{2,3}),(t_{3,4},D_{3,4})\}$, at the intersections of consecutive tangets of the sampled points.

$\underline{D}[o,d]$ is the required lower bound (never exceeding it), and (ii) $\overline{D}[o,d]$ is the required upper bound (always above it), are simple consequences of the concavity of $D[o,d]$ (since all arc-delays are linear).

5.1.3. *Overview of our Refined Approximation Technique.* We are now ready to describe our sampling procedure, namely the algorithms **OD_PWL1** (see figure **??**) that provides the explicit representation of the breakpoints of $\underline{D}[o,d]$, and **OD_PWL2** (see figure **??**) that provides the breakpoints of $\overline{D}[o,d]$, in such a way that both the required approximation ratio of eq. (16) is guaranteed and the output is indeed asymptotically space-optimal. **OD_PWL1** is again split in two distinct (sampling) phases, depending on the slope of $D[o,d]$ at the last sampled point. The first phase again samples the delay-axis. But this time, based on the exact comptution of the worst-case absolute error that may occur, we roughly make half the samples compared to the method of Foschini et al., while keeping (as breakpoints of $\underline{D}$) among them only those that are really necessary in order to assure the approximation guarantee. Additionally, we are only interested in candidate breakpoints which are beyond the next certificate failure t_{fail}, a quantity that can also be computed during each Dijkstra-run for the next sample point. This is safe because until t_{fail} we already know that $\overline{D}[o,d]$ will be identical to $D[o,d]$. Figure 8 provides an overview of the basic idea. The second phase samples the time-axis using bisection, but again we use the exact expression of the maximum absolute error, in order to determine whether to stop or continue bisecting the time axis. We provide here a complete, self-contained analysis, whose crucial element is of course the exact estimation of the maximum absolute error (ie, maximum distance between $\overline{D}[o,d]$ and $\underline{D}[o,d]$) during our construction.

5.1.4. *Correctness of Our Approximation Technique.* We shall now demonstrate that it is indeed the case that $\overline{D}[o,d]$ is an upper approximation of $D[o,d]$, with the required approximation guarantee. For sake of simplicity, we shall drop in this subsection the reference to the OD-pair (o,d) that we consider and we shall write D, $\underline{D}$ and $\overline{D}$ respectively from now on. We shall assure that the linear interpolant $\overline{D}$ is indeed a $(1+\varepsilon)-$upper bound, not of $D[o,d]$ (whose exact shape we do not know), but of of the linear interpolant $\underline{D}$, in the time-window $[t_s,t_f]$. In particular, we shall demonstrate inequality (16) holds.

We start by upper-bounding the absolute error between $\underline{D}$ and $\overline{D}$, and consequently we study the performance guarantee of the algorithm. The following claim is used:

Proposition 5.2. *Fix any continuous (not necessarily pwl) concave function $D:[c,d]\to\mathbb{R}_{\geq 0}$ on a (nonempty) time interval $[c,d]\subset\mathbb{R}_{\geq 0}$, with right and left derivatives at the endpoints denoted*

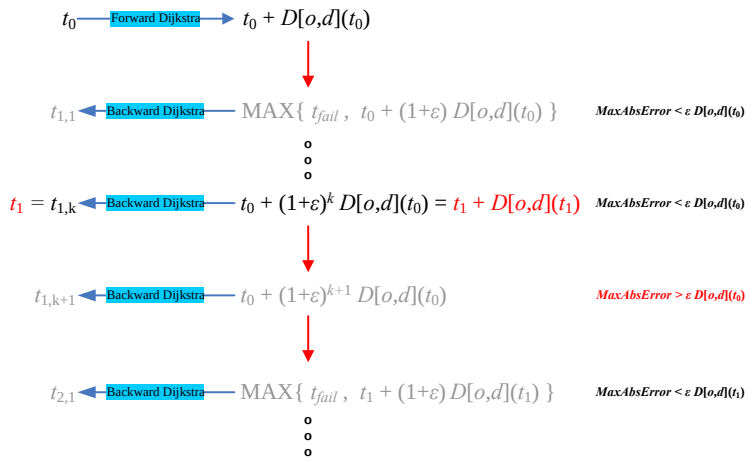
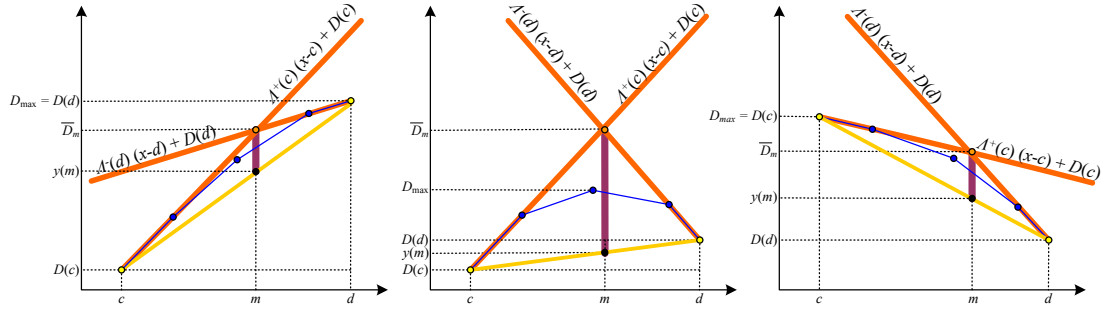


FIGURE 8. Overview of the first phase of our refined approximation technique.



(a) $\Lambda^+(c) > \Lambda^-(d) \geq 0$ (b) $\Lambda^+(c) > 0 > \Lambda^-(d)$ (c) $0 \geq \Lambda^+(c) > \Lambda^-(d)$

FIGURE 9. Three distinct cases for upper-bounding the absolute error between two consecutive interpolation points. The (upper bound on the) absolute error considered is shown by the vertical (purple) line segment at point m of the time axis.

by $\Lambda^+(c), \Lambda^-(d)$. *Assume that* $\Lambda^+(c) > \Lambda^-(d)$ *and* $L = d - c > 0$. *Consider the affine functions (see also figure 9):*

$$\begin{aligned} y(x) &= \tfrac{D(d)-D(c)}{L} \cdot x + \tfrac{D(c)d - D(d)c}{L}\,, \\ y_c(x) &= \Lambda^+(c) \cdot (x-c) + D(c)\,, \\ y_d(x) &= \Lambda^-(d) \cdot (x-d) + D(d)\,. \end{aligned}$$

Let Let $\left(m = \frac{D(d)-D(c)+c\cdot\Lambda^+(c)-d\cdot\Lambda^-(d)}{\Lambda^+(c)-\Lambda^-(d)}, \overline{D}_m = y_c(m) = y_d(m)\right)$ *be the intersection point of the lines* $y_c(x)$ *and* $y_d(x)$. *For the pwl upper bound of* D *in* $[c,d]$, $\overline{D}(t) = \min\{y_c(t), y_d(t)\}$ *and the linear lower bound of* D *in* $c,d]$, $\underline{D}(t) = y(t)$, *the* maximum absolute error *is at most:*

$$MAE(c,d) = (\Lambda^+(c) - \Lambda^-(d)) \cdot \frac{(m-c)\cdot(d-m)}{L} \leq \frac{L \cdot (\Lambda^+(c) - \Lambda^-(d))}{4}\,.$$

Proof. By concavity and continuity of D, we know that the partial derivatives' values may only decrease with time, and at any given point in $[c,d]$ the left-derivative value is at least as large as the right-derivative value. Thus, the restriction of D on $[c,d]$ lies entirely in the area of the triangle $\{(c, D(c)), (m, \overline{D}_m), (d, D(d))\}$. The maximum possible distance (additive error) of $\overline{D}$ from $\underline{D}$

$$MAE(c,d) = \max_{c \leq t \leq d} \{\overline{D}(t) - \underline{D}(t)\}$$

is at most equal to the *vertical distance* of the two approximation functions, namely, at most equal to the length of the line segment connecting the points $(m, y(m))$ and $(m, \overline{D}_m)$(denoted by purple

color in figure 9). The calculations are identical for the three distinct cases shown in figure 9. Let $\underline{\Lambda} = \frac{D(d)-D(c)}{L}$ be the slope of the line $y(x)$. Observe that:

$$\begin{aligned}\underline{\Lambda} &= \frac{D(d)-D(c)}{L} = \frac{(\overline{D}_m - D(c)) - (\overline{D}_m - D(d))}{L} \\ &= \frac{m-c}{L}\cdot\frac{\overline{D}_m - D(c)}{m-c} - \frac{d-m}{L}\cdot\frac{\overline{D}_m - D(d)}{d-m} \\ &= \frac{m-c}{L}\cdot\Lambda^+(c) + \frac{d-m}{L}\cdot\Lambda^-(d)\,.\end{aligned}$$

Thus we have:

$$\begin{aligned}MAE(c,d) &= \overline{D}_m - y(m) = (\overline{D}_m - D(c)) - (y(m) - D(c)) \\ &= \Lambda^+(c)\cdot(m-c) - \underline{\Lambda}\cdot(m-c) = (\Lambda^+(c) - \underline{\Lambda})\cdot(m-c) \\ &= (\Lambda^+(c) - \Lambda^-(d))\cdot\frac{(m-c)\cdot(d-m)}{L} \le \frac{L\cdot(\Lambda^+(c)-\Lambda^-(d))}{4}\,,\end{aligned}$$

since $(m-c)+(d-m) = d-c = L$ and the product $(m-c)\cdot(d-m)$ is maximized at $m = \frac{c+d}{2}$. □

We shall now prove the claimed quality of approximation provided by $\underline{D}$ and $\overline{D}$, by demonstrating that $\overline{D}$ is indeed a $(1+\varepsilon)-$approximation of $\underline{D}$ between any pair of consecutive breakpoints in $\underline{D}$ ($\overline{D}$ may have at most one more breakpoint between them). We shall prove this seperately for consecutive points produced by Phase-1 of the sampling procedure (ie, while **OD_PWL1.1** is sampling the delay-axis), and then for the points produced by the Phase-2 (ie, while **OD_PWL1.2** is sampling the time-axis).

CASE A: Consecutive breakpoints produced **OD_PWL1.1**. During this phase all the produced sample points have positive partial derivatives (indeed, greater than 1), meaning that D is indeed strictly increasing. Consider two consecutive breakpoints of $\underline{D}$ (t_k, D_k) and (t_{k+1}, D_{k+1}) produced during **OD_PWL1.1**. Recall that the sampling procedure created a sequence of sample points from (t_k, D_k), before eventually choosing one of them as the next breakpoint (t_{k+1}, D_{k+1}) of $\underline{D}$:

$$\begin{aligned}D_{1,k} &= t_k + (1+\varepsilon)D_k - t_{1,k} && \overset{/*\ t_{1,k}\ge t_k\ */}{\Longrightarrow} && MAE_{1,k} \le D_{1,k} - D_k \le \varepsilon D_k \\ &\vdots && \vdots \\ D_{j_{\max},k} &= t_k + (1+\varepsilon)^{j_{\max}}D_k - t_{j_{\max},k} && \Longrightarrow && MAE_{j_{\max},k} \le \varepsilon D_k \\ D_k^{j_{\max}+1} &= t_k + (1+\varepsilon)^{j_{\max}+1}D_k - t_k^{j_{\max}+1} && \Longrightarrow && MAE_k^{j_{\max}+1} > \varepsilon D_k\end{aligned}$$

where, $MAE_{j,k}$ is the value of the max absolute error between the (confirmed) breakpoint (t_k, D_k) and the sample point $(t_{j,k}, D_{j,k})$. The first sample point $(t_{1,k}, D_{1,k})$ always satisfies the required quality of approximation, by construction. We keep looking for sample points, until the last one $(t_{j_{\max},k}, D_{j_{\max},k})$ that satisfies the required quality of approximation. Then we store it as the next breakpoint (t_{k+1}, D_{k+1}) of both $\underline{D}$ (in LBP) and $\overline{D}$ (in UBP). Additionally, in UBP (before this new breakpoint) we also store the intermediate breakpoint $(m, \overline{D}_m)$ as described in Proposition 5.2 for $c = t_k$ and $d = t_{k+1}$. It is then straigthforward to assure that for any point $t \in (t_k, t_{k+1})$ it holds that:

$$\begin{aligned}\overline{D}(t) - \underline{D}(t) &\le MAE \le \varepsilon D_k \le \varepsilon\underline{D}(t) \\ \Rightarrow \overline{D}(t) &\le (1+\varepsilon)\underline{D}(t)\end{aligned}$$

where the second inequality is by the fact that $\min_{t\in[t_k,t_{k+1}]}\underline{D}(t) = D_k$ (see figure 9(a)).

CASE B: Consecutive breakpoints produced by **OD_PWL1.2**. Assume now that we are given two consecutive breakpoints (t_k, D_k), (t_{k+1}, D_{k+1}) of $\underline{D}$ with respect to the departure times from the origin (which are also breakpoints of $\overline{D}$) produced by the bisection phase, along with the additional intermediate breakpoint $(m, \overline{D})$ for $\overline{D}$. By construction the bisection did not produce another breakpoint for $\underline{D}$ between them, exactly because the maximum absolute error produced satisfies the required approximation guarantee:

$$\begin{aligned}\forall t \in [t_k, t_{k+1}],\ \overline{D}(t) - \underline{D}(t) &\le MAE(t_k, t_{k+1}) \le \varepsilon\cdot\min\{D_k, D_{k+1}\} \le \varepsilon\underline{D}(t) \\ \Rightarrow \forall t \in [t_k, t_{k+1}],\ \overline{D}(t) &\le (1+\varepsilon)\underline{D}(t)\end{aligned}$$

CASE C: Last breakpoint of **OD_PWL1.1** and first breakpoint of **OD_PWL1.2**. In this case observe that in the last round of **OD_PWL1.1** (after having produced (t_k, D_k), the last sample point before termination of **OD_PWL1.2**, say $(t_{j,k}, D_{j,k})$ for some $j \geq 1$, still assures an absolute error at most $\varepsilon \cdot D_k$ but it happens that the slope of D at that point falls below 1 for the first time. This point is accepted both by **OD_PWL1.1** and by **OD_PWL1.2** (as the first endpoint of the bisection, the other endpoint being (t_f, D_f)). Therefore, between (t_k, D_k) and $(t_{k+1}, D_{k+1}) = (t_{j,k}, D_{j,k})$ **OD_PWL1.1** assures that $\forall t \in [t_k, t_{k+1}]$, $\overline{D}(t) \leq (1+\varepsilon)\underline{D}(t)$, as required.

5.1.5. *Time and Space Complexity of* **OD_PWL1**. We now explore the time and space complexity of **OD_PWL1**. We measure the space complexity by the *number of breakpoints* (i.e., $|UBP[o,d]|$) that eventually have to be stored. The time complexity is measured by the *number of time-dependent shortest-path calls* (e.g., calls of **TDD**) involved. We do this because on one hand it is clearly that parameter which dominates the time-complexity of the algorithm, while on the other hand its exact execution-time depends both on the implementation (Dikstra / speed-up heuristics, etc.) and the characteristics of the graph instance (planar / bounded-genus graphs, directed / undirected graphs, integer weights, etc.) that we consider.
Number of breakpoints produced by **OD_PWL1.1**. Assume that

$$LBP1[o,d] = \langle (t_0, D_0) = (t_s, D_s), (t_1, D_1), \ldots, (t_{\ell_1}, D_{\ell_1}) \rangle$$

is the ordered list of $\ell_1 + 1$ breakpoints for $\underline{D}[o,d]$ during this phase. Recall that for all the stored sample points in the list the left and right slopes of $Arr[o,d]$ are greater than 2 (except possibly for the right slope at time t_{ℓ_1}). Therefore:

$$\forall 0 \leq k \leq \ell_1 - 1, \exists j_k \geq 1: \qquad t_{k+1} + D_{k+1} = t_k + (1+\varepsilon)^{j_k} \cdot D_k \geq t_k + D_k + 2 \cdot (t_{k+1} - t_k)$$

$$\Rightarrow \quad t_{k+1} - t_k \leq \frac{\left[(1+\varepsilon)^{j_k} - 1\right] \cdot D_k}{2} \tag{19}$$

We know (by the correctness of **OD_PWL1.1**) that $\forall 0 \leq k \leq \ell_1 - 1, D_{k+1} \leq D_{\max}$. We shall now provide also lower bounds, that will determine the required number of breakpoints: $\forall 0 \leq k \leq \ell_1 - 1, \exists j_k \geq 1$,

$$\begin{aligned} & D_{k+1} = (1+\varepsilon)^{j_k} \cdot D_k - (t_{k+1} - t_k) \\ \overset{/* \text{ eq. (19) } */}{\Rightarrow} \quad & D_{k+1} > \frac{(1+\varepsilon)^{j_k} + 1}{2} \cdot D_k \overset{/* \, j_k \geq 1 \, */}{\geq} \frac{2+\varepsilon}{2} \cdot D_k \\ \Rightarrow \quad & D_{k+1} \geq \left(\frac{2+\varepsilon}{2}\right)^2 \cdot D_{k-1} \geq \cdots \geq \left(1 + \frac{\varepsilon}{2}\right)^{k+1} \cdot D_0 \end{aligned}$$

Therefore, we conclude that:

$$D_{\max}[o,d] \geq \left(1 + \frac{\varepsilon}{2}\right)^{\ell_1} \cdot D_{\min}[o,d] \qquad \Rightarrow$$

$$|LBP1[o,d]| = 1 + \ell_1 \leq 1 + \log_{1+\varepsilon/2}\left(\frac{D_{\max}[o,d]}{D_{\min}[o,d]}\right) \in \mathcal{O}\left(\frac{1}{\varepsilon} \cdot \log\left(\frac{D_{\max}[o,d]}{D_{\min}[o,d]}\right)\right) \tag{20}$$

It is now straightforward to conclude that the number of breakpoints of $\overline{D}[o,d]$ created during **OD_PWL1.1** is:

$$|UBP1[o,d]| \leq 2|LBP1[o,d]| - 1 \leq 1 + 2 \cdot \log_{1+\varepsilon/2}\left(\frac{D_{\max}[o,d]}{D_{\min}[o,d]}\right) \tag{21}$$

As for the time-complexity of this phase, the number $TDSP1$ of time-dependent shortest-path probes (either forward or reverse) involved in **OD_PWL1.1** is at most equal to the upper bound on $|LBP1[o,d]|$ computed in equation (20). This is because this upper bound was computed under the assumption that all sample points were producing breakpoints of $\underline{D}$, ie, $j_k = 1$ for each sample point.

Finally, the space-complexity of **OD_PWL1.1** is asymptotically optimal, due to the fact that we *choose to store* in $LBP1[o,d]$ only those sample points that are really necessary for guaranteeing the claimed approximation ratio. This implies that any upper-approximation of $D[o,d]$ for this particular time subinterval handled by **OD_PWL1.1** has to employ at least one breakpoint for every two consecutive breakpoints of $LBP1[o,d]$, otherwise it will not be able to guarantee the

claimed approximation ratio. We therefore conclude that our number of stored breakpoints in $UBP1$ is at most 4 times the minimum number of breakpoints required. Of course, one could also store only the breakpoints of $LBP1$ and each time compute on-the-fly the additional intermediate breakpoints of $UBP1$, in order to assure a factor of 2 away of space-optimality.

Number of sample points by **OD_PWL1.2**. Each bisection halves the time interval under consideration. For initial interval (at the beginning of phase **OD_PWL1.2**) equal to $L(0) \leq t_f - t_s$, we know that at the $k-$th level of the recursion tree the intervals have length $L(k) = L(0)/2^k$. Since $0 \leq \Lambda^+(c) - \Lambda^-(d) \leq 2$ (recall that the partial derivatives of $D[o,d]$ in this time window are in $[-1,1]$), the absolute error is upper bounded by $\frac{L(k)\cdot(\Lambda^+(c)-\Lambda^-(d))}{4} \leq \frac{L(0)}{2^{k+1}}$. In each interval (t_1,t_2) it holds that $t_2 - t_1 \geq D^{1,2}_{\max} - D^{1,2}_{\min}$, where $D^{1,2}_{\max}, D^{1,2}_{\min}$ are the corresponding max- and min-delay values in the time-interval under consideration. This implies that the bisection will certainly stop at a level ℓ_2 of the recursion tree at which

$$MAE(\ell_2) \leq \frac{L(0)}{2^{\ell_2+1}} \leq \varepsilon D_{\min}[o,d] \Rightarrow \ell_2 \geq \log_2\left(\frac{L(0)}{\varepsilon D_{\min}[o,d]}\right) - 1$$

i.e., when the absolute error falls below the safety level. We thus conclude that the *depth* of the recursion tree produced by Phase-2 is at most equal to $\ell_2 = \left\lfloor \log_2\left(\frac{L(0)}{\varepsilon \cdot D_{\min}[o,d]}\right)\right\rfloor - 1$.

On the other hand, the parents of consecutive leaves $(t_1,D_1),(t_2,D_2),(t_3,D_3)$ at the recursion tree correspond to intervals $[t_1,t_3]$ for which the maximum absolute error guarantee is greater than $\varepsilon D^{1,3}_{\min}[o,d] \geq \varepsilon D_{\min}[o,d]$, indicating that (in worst case) no pwl $(1+\varepsilon)-$approximation may avoid placing at least one interpolation point in the interval $[t_1,t_3]$. Therefore, the proposed bisection method of phase **OD_PWL1.2** samples again at most 2 times the minimum number of interpolation points required for any $(1+\varepsilon)-$approximation of $D[o,d]$, when the maximum slope of $D[o,d]$ is upper bounded by 1:

$$|LBP2[o,d]| \leq 2 \cdot \log_{1+\varepsilon}\left(\frac{D_{\max}[o,d]}{D_{\min}[o,d]}\right) \in \mathcal{O}\left(\frac{1}{\varepsilon}\log\left(\frac{D_{\max}[o,d]}{D_{\min}[o,d]}\right)\right). \tag{22}$$

As for the interpolation points of $\overline{D}$, these may be at most twice as many:

$$|UBP2[o,d]| \leq 2 \cdot |LBP2[o,d]| - 1 \leq 4 \cdot \log_{1+\varepsilon}\left(\frac{D_{\max}[o,d]}{D_{\min}[o,d]}\right) \tag{23}$$

We now proceed with the time-complexity, we shall count the number $TDSP2$ of time-dependent shortest-path probes to compute all the breakpoints of $\underline{D}$ produced during the bisection. For each breakpoint in $LBP2[o,d]$, the corresponding number of (forward) TDSP-probes is upper-bounded by the path-length leading to consideration of this point in the recursion tree of the bisection. Any root-to-node path in this tree has length at most ℓ_2, therefore each breakpoint of $LBP2[o,d]$ requires at most ℓ_2 TDSP-probes up to its own sampling. In overall, the total number of TDSP-probes to compute $LBP2[o,d]$ is upper-bounded by the following inequality:

$$TDSP2 \leq \ell_2 \cdot |LBP2[o,d]| \in \mathcal{O}\left(\log\left(\frac{L(0)}{\varepsilon \cdot D_{\min}[o,d]}\right) \cdot \frac{1}{\varepsilon}\log\left(\frac{D_{\max}[o,d]}{D_{\min}[o,d]}\right)\right) \tag{24}$$

time-dependent shortest-path calls. **OD_PWL2** constructs $UBP[o,d]$ from $LBP[o,d]$ without any execution of a TDSP-probe, it just sweeps $LBP[o,d]$ once and adds the intermediate breakpoints required. The time-complexity of this procedure is $\mathcal{O}(|LBP[o,d]|)$ and this is clearly dominated by the time-complexity (number of TDSP probes) for constructing $LBP[o,d]$ itself.

5.2. **Approximating shortest-travel-times from a given origin.** In this subsection we study how to approximate the problem SOTDDA. That is, we want to provide an upper-approximation $\overline{\mathbf{D}}[o,\star]$ of a vector function $\mathbf{D}[o,\star]$, in such a way that each coordinate $\overline{D}[o,v]$ is a $(1+\varepsilon)$-upper-approximation of $D[o,v]$, for every vertex v reachable from the origin. Analogously, $\underline{\mathbf{D}}[o,\star]$ is the corresponding lower-approximating vector function that we wish to produce, in order to have a measure of comparison between the upper and the lower approximation of the unknown vector function $\mathbf{D}[o,\star]$ of shortest travel times from o. Our goal will be to provide the following bounds, which act as our performance guarantee for the upper-approximating functions that we produce:

$$\forall v \in V, \forall t_o \geq 0,\ \underline{D}[o,v](t_o) \leq D[o,v](t_o) \leq \overline{D}[o,v](t_o) \leq (1+\varepsilon)\cdot \underline{D}[o,v](t_o) \tag{25}$$

Of course, one might consider the case of exploiting the previous methodology of point-to-point approximations of shortest-travel-time functions, but this would clearly be a waste of time. Our objective is to manage to produce the required upper-approximation with time complexity that is comparable to that of a single-pair upper-approximation construction.

Again in this subsection we focus on the case of linear arc-delays. The extension to the case of pwl arc delays is once more done by projecting all the primitive (arc-delay) breakpoints to departure-times from the origin, and then approximating the required functions per subinterval between consecutive primitive images.

5.2.1. *The Case of Bounded Slopes.* We first consider the case in which all shortest-travel-time functions in the network have bounded slopes. In particular, we make the following assumptions, which seem to be quite reasonable in realistic scenarios, as in road networks:

Assumption 5.1. *The following assumptions are made for shortest-travel-time functions of the network:*

(1) *All the arc-delay slopes are from a real interval* $(-1, \lambda_{\max})$*, for a given* constant $\lambda_{\max} > 0$.
(2) *The slopes of the shortest-travel-time functions are upper-bounded by a* known (constant) value $\Lambda_{\max}$*. In particular:*

$$\forall v \in V, \forall t_1, t_2 \geq 0 : t_1 \neq t_2, -1 < \frac{D[o,v](t_1) - D[o,v](t_2)}{t_1 - t_2} \leq \Lambda_{\max}$$

We start with a generalization of the bisection method for single-pairs (cf. **OD_PWL1.2**) to the case of a single-origin o and all reachable destinations from it. Our method creates *concurrently* (i.e., within the same bisection) all the required approximate functions of $\underline{\mathbf{D}}[o, \star]$ and $\overline{\mathbf{D}}[o, \star]$. This is possible because the bisection is done on the (common for all travel-time functions to approximate) departure-time axis from the common origin o. The other important observation is that, for each destination node $v \in V$, we only keep as breakpoints of $\underline{D}[o, v]$ those sample points which are indeed necessary for the required approximation guarantee per particular vertex, thus achieving an asymptotically optimal space-complexity of our method, as we shall explain in the analysis of our approach. This is possible due to the fact that we have an exact expression for the evaluation of the (worst-case) error of approximation (cf. Proposition 5.2) per destination vertex. Moreover, all the delays to be sampled at a particular bisection point are calculated by a single time-dependent shortest-path (e.g., **TDD**) execution. In particular, we keep bisecting the departure-time axis, until the two bounding approximations $\overline{\mathbf{D}}[o, \star]$ and $\underline{\mathbf{D}}[o, \star]$ are guaranteed to be within a relative error of $\varepsilon D_{\min}[o, \star]$, for all possible destination vertices. The time complexity of this apprach will be asymptotically equal to that of **OD_PWL1.2** for the more demanding origin-destination pair.

SO_PWL1.2 is just a recursive call of the new bisection method **SO_BISECT**, which manages to concurrently compute new breakpoints for all the shortest-travel-time approximations that really need it. As already mentioned, the main trick is that we keep sampling the time-axis (for departure times from the origin o) which is common for all the sampled shortest-travel-time functions. Additionally, the sampling of all these functions is conducted by a single (forward) time-dependent shortest-path probe.

Number of sample points by **SO_PWL1.2**. Assume that we start with an initial interval $L(0) = t_f - t_s$. Each bisection again halves the current time subinterval under consideration. Therefore, at the $k-$th level of the recursion tree all the (still active) subintervals have length $L(k) = L(0)/2^k$. Since, for any shortest-travel-time function and any subinterval $[t_1, t_2]$ of departure-times from o, it holds that $0 \leq \Lambda^+[o,v](t_1) - \Lambda^-[o,v](t_2) \leq \Lambda_{\max} + 1$, the absolute error between $\underline{D}[o, v]$ and $\overline{D}[o, v]$ in this interval is at most $\frac{L(k)\cdot(\Lambda_{\max}+1)}{4} \leq \frac{L(0)\cdot(\Lambda_{\max}+1)}{2^{k+2}}$. This implies that the bisection will certainly stop at a level ℓ at which

$$\forall v \in V, MAE[o,v](\ell) \leq \frac{T \cdot (\Lambda_{\max} + 1)}{2^{\ell+2}} \leq \varepsilon D_{\min}[o,v](t_s, t_f) \Rightarrow \ell \geq \log_2 \left(\frac{T \cdot (\Lambda_{\max} + 1)}{\varepsilon D_{\min}[o,v](t_s, t_f)} \right) - 2$$

Our assumption at this point is that any vertex v which is removed from the set M of active vertices (exactly because its own absolute error is already small enough) keeps this upper bound on its absolute error also in the subsequent levels of the recursive tree, although it is inactive at

them. We thus conclude that the height of the recursion tree is indeed:

$$\ell = \left\lfloor \max_{v \in V} \left\{ \log_2 \left(\frac{T \cdot (\Lambda_{\max} + 1)}{\varepsilon D_{\min}[o, v](t_s, t_f)} \right) \right\} \right\rfloor - 2$$

On the other hand, the parents of the leaves at the recursion tree correspond to subintervals $[t_1, t_2] \subset [t_s, t_f]$ for which the absolute error of at least one vertex $v \in V$ is greater than $\varepsilon D_{\min}[o, v](t_1, t_2)$, indicating that (in worst case) no pwl $(1+\varepsilon)-$approximation may avoid placing at least one interpolation point in this subinterval. Therefore, the proposed bisection method **SO_PWL1.2** produces at most twice as many interpolation points (to determine the lower-approximating vector function $\underline{\mathbf{D}}[o, \star]$) required for any $(1 + \varepsilon)$-upper-approximation of $\mathbf{D}[o, \star]$. The produced $(1 + \varepsilon)$-upper-approximation by **SO_PWL1.2** uses at most twice the number of sampled points:

(26)

$$\forall v \in V,\ |UBP[o, v](t_s, t_f)| \leq 4 \cdot \log_{1+\varepsilon} \left(\frac{D_{\max}[o, v](t_s, t_f)}{D_{\min}[o, v](t_s, t_f)} \right) \in \mathcal{O}\left(\frac{1}{\varepsilon} \log \left(\frac{D_{\max}[o, v](t_s, t_f)}{D_{\min}[o, v](t_s, t_f)} \right) \right).$$

We now proceed with the time-complexity of **SO_PWL1.2**. Again we shall count the number $SOTDSP2$ of time-dependent shortest-path probes to compute all the sample points during the bisection. The crucial observation is that the bisection is applied on the departure-time axis. Additionally, in each recursive call from $[t_1, t_2]$, all the new sample points at the new departure-time $t_{mid} = \frac{t_1+t_2}{2}$, to be added to the breakpoint lists of the active vertices, are computed by a *single* (forward) TDSP-probe. Moreover, for each vertex v, every breakpoint of $LBP[o, v](t_s, t_f)$ requires again a number of (forward) TDSP-probes that is upper bounded by the path-length leading to the consideration of this point in the recursion tree of the bisection. Any root-to-node path in this tree has length at most ℓ, therefore each sample point of $LBP2[o, v](t_s, t_f)$ requires at most ℓ TDSP-probes. In overall, the total number $SOTDSP2$ of shortest-path-tree probes required to construct $LBP2[o, \star](t_s, t_f)$, is upper-bounded by the following inequality:

$$SOTDSP2 \in \mathcal{O}\left(\max_{v \in V} \left\{ \log_2 \left(\frac{T \cdot (\Lambda_{\max} + 1)}{\varepsilon D_{\min}[o, v](t_s, t_f)} \right) \right\} \frac{1}{\varepsilon} \log \left(\frac{D_{\max}[o, v](t_s, t_f)}{D_{\min}[o, v](t_s, t_f)} \right) \right) \tag{27}$$

forward TDSP-probes. We can also construct $UBP2[o, \star](t_s, t_f)$ from $LBP[o, \star](t_s, t_f)$ without any execution of a TDSP-probe, by just sweeping once for every vertex $v \in V$ $LBP2[o, v](t_s, t_f)$ and adding all the intermediate breakpoints required. The time-complexity of this procedure is $\mathcal{O}(|LBP2[o, \star](t_s, t_f)|)$ and this is clearly dominated by the time-complexity (number of TDSP-probes) for constructing $LBP2[o, \star](t_s, t_f)$ itself.

5.2.2. *Shortest-Travel-Time Functions with Arbitrary Slopes.* We close our discussion on approximating $\mathbf{D}[o, \star]$ by providing a method (we call it **SO_PWL1.1**, in analogy to the point-to-point approximations) that covers the case when some of the partial derivatives (slopes of affine legs in the pwl functions) are only restricted to satisfy the FIFO property. In other words, there is no upper bound in the arc-delay and shortest-travel-time functions in the network.

In the previous section we demonstrated how it is indeed possible to create (asymptotically space-optimal) *point-to-point* pwl approximations to $D[o, d]$, even when this function is rapidly increasing. The main ingredient was a sequence of *backward* TDSP probes that allowed sampling the (faster-growing) delay axis of the unknown function $D[o, d]$. Unfortunately, in the case of a single origin and multiple destinations it is not possible to do the same, since we are now sampling various axes of shortest-travel-time values (one per destination). But sampling on the values of different functions would be equivalent to computing point-to-point approximations separately, which would be unnecessarily expensive. Our goal is to manage to create all the required sample points of shortest-travel-time values *concurrently*, with time-complexity equivalent to the *worst* time-complexity of a point-to-point approximation of a shortest-travel-time function from o to one of the possible destination nodes.

What we do is the following: We create again a sequence of sample points for different departure-times from the origin in such a way that, for each sampled departure-time, at least one destination node would need it in order to assure the desired approximation guarantee. On the other hand, for each possible destination node we store only those sample points that are indeed necessary for their approximation guarantee, and only so long as this node still has an instantaneous (i.e.,

around the sampling departure-time) shortest-travel-time function slope greater than 1. Since we are guaranteed that these slopes may only decrease with time, we *deactivate* each destination node v whose shortest-travel-time function $D[o,v]$ reaches a right-slope $\Lambda^+[o,v]$ of at most 1, and finish this approximation phase when all the destination nodes have become deactivated. The next phase (bisection) of the approximation algorithm would have to start at the departure-time of the first node deactivation during this first phase.

In particular, starting from a departure-time $t_0 \geq 0$, we run a forward TDSP probe to determine the earliest-arrival values (and the instantaneous functional descriptions of these functions) at destination nodes:

$$\Big(t'_{0,v} = t_0 + \overrightarrow{D}[o,v](t_0)\Big)_{v\in V} = \big(t'_{0,v} = A^{\pm}_{0,v} \cdot t_0 + B^{\pm}_{0,v}\big)_{v\in V}$$

Each destination node that has $A^+_{o,v} \leq 2$ is removed from the set M of active nodes, in order to become deactivated. Consequently, we delay the entire shortest-travel-times (vector) function $\mathbf{D}[o,\star]$ by $(1+\varepsilon)$, only for the active nodes:

$$\Big(t'_{1,v} = t_0 + (1+\varepsilon)\cdot \overrightarrow{D}[o,v](t_0)\Big)_{v\in M} = \big(t'_{0,v} = t_0 + (1+\varepsilon)(A^+_{0,v}-1)\cdot t_0 + (1+\varepsilon)\cdot B^+_{0,v}\big)_{v\in M}$$

Rather than computing (via backward TDSP probes) the actual latest-departure times $t_{1,v}$ from the origin to meet all the arrival-times $t'_{1,v}$ at active destinations, we estimate *lower bounds* $\underline{t}_{1,v} \leq t_{1,v}$ by reversing the delay functions and computing the corresponding latest departure times from o, as if no further breakpoint would occur. Clearly, these estimated departure times can only be earliear than the actual latest departure times, due to the concavity of all the shortest-travel-time functions in the subinterval $[t_s, t_f]$. The minimum (over all active nodes) of all these lower bounds is a new sampling departure-time, for which we can be sure that will not violate any of the approximation guarantees for all the shortest-travel-time functions. In particular, we consider the next sampling departure-time:

$$t_1 \;=\; \min_{v\in M}\big\{\ \underline{t}_{1,v}\ \big\} = t_0 + \varepsilon \cdot \min_{v\in M}\left\{\frac{(A^+_{0,v}-1)\cdot t_0 + B^+_{0,v}}{A^+_{0,v}}\right\} > t_0 \tag{28}$$

and we create a new set of sampled shortest-travel-time points at time t_1 by running a single forward TDSP probe. From all these sample points, only those that are indeed necessary for the required approximation guarantee and involve still active destination nodes are stored. The rest are simply skipped. An overview of the algorithm is given in figure 10.

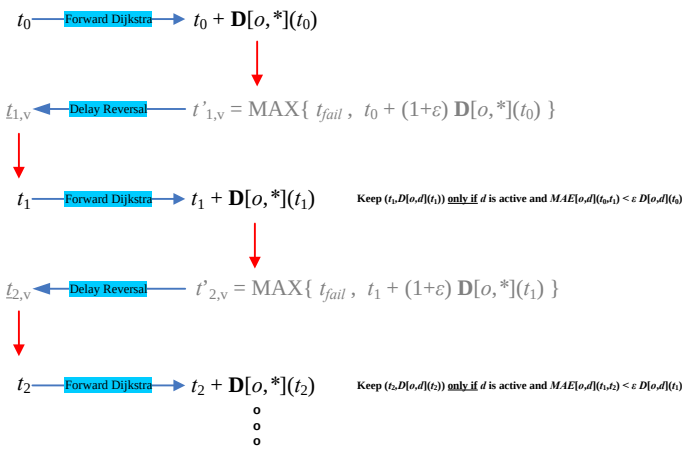


FIGURE 10. Overview of the first phase of our refined approximation technique for SOTDDA.

Correctness of Phase 1. We exploit the fact that all the arc-delay functions are assumed to be linear in the subdomain of interest $[t_s, t_f] \subset [0,T]$, and thus the unknown shortest-travel-times vector function $\mathbf{D}[o,\star]$ is a continuous, pwl concave chain in every coordinate. Recall that, at any sample point $(t_o, \mathbf{D}[o,\star](t_o))$, we know not only the particular delay values, but also the instantaneous functional descriptions of all the functions $D[o,v]$ in a small ball around t_o (assuming

no two minimization breakpoints at nodes never happen to occur exactly at the same departure time). From a given sample point t_0, we choose the next sample point (cf. equation 28) in such a way that the gap between each upper-approximating and each lower-approximating function does not violate the required approximation guarantee (as was the case also in our phase-1 for the point-to-point approximation). The main difference is that, rather than executing individual backward TDSP probes (from any possible destination of interest) in order to discover the latest next departure time that would not violate any of these guarantees, we compute directly the corresponding departure times for the reverse shortest-travel-time upper-approximating functions, and choose the earliest of them as the next sample point. If we had executed the backward TDSP probes instead, we might have even later sample points, but our sampling sequence is only a pessimistic one (with more sample points than the one with backward TDSP executions). Consequently, we run again a forward TDSP probe to get the functional description and delay values of $\mathbf{D}[o,\star](t_1)$.

Therefore, the correctness of our approach is a simple consequence of the correctness for the point-to-point approximation presented in subsection 5.1.4.

Time/Space Complexity of Phase 1. The space-complexity of our algorithm is assured by the fact that each destination nodes only keeps as breakpoints those sample points that are actually necessary for its own approximation guarantee. It is straightforward to see that any other approximation algorithm would require at least one breakpoint per two (stored) breakpoints of our approach.

Therefore, we focus our interest on the time-complexity of our approach. Our main argument is that in each sample point and each active destination v, either a significant progress is done on the delay value (until we reach $D_{\max}[o,v]$) or the earliest-arrival slope is significantly reduced. In particular, the following property holds:

Proposition 5.3. *Let $t_0 < t_1$ two sampling departure-times of* **SO_PWL1.1***, and let v be the departure-vertex that remains active up to time t_1 and is responsible for the value of t_1 (cf. equation 28). Assume knowledge of the instantaneous functional descriptions of the earliest-arrival function $Arr[o,v]$ given by $A_0^{\pm}x + B_0^{\pm}$ at time t_0 and $A_1^{\pm}x + B_1^{\pm}$ at time t_1. Then the following hold:*

$$\begin{aligned} D[o,v](t_1) &\geq \left(1 + \frac{1-\delta}{2}\varepsilon\right) \cdot D[o,v](t_0) \\ A_1^+ &\leq (A_0^+ - 1) \cdot (1-\delta) - 1 \end{aligned}$$

where $D_1 = (1-\delta)\overline{D}_1 + \delta D_0$ for some $\delta \in (0,1]$, $D_1 = D[o,v](t_1) = (A_1^+ - 1)t_1 + B_1^+$ is the actual delay for departure time t_1, $D_0 = D[o,v](t_0) = (A_0^+ - 1)t_0 + B_0^+$ is the actual delay for departure time t_0, and $\overline{D}_1 = (A_0^+ - 1)t_1 + B_0^+$ is the hypothetical delay for departure time t_1, if no further breakpoint would occur in $(t_0, t_1]$.

Before providing the proof of this proposition, we should mention that it assures that in each sampling point, for at least one active destination vertex (the one determining the value of $\underline{t}_1$) either the actual delay is significantly increased by a constant factor, or the instantaneous earliest-arrival slope decreases by (roughly) a constant factor.

Proof. Our first observation has to do with the difference of the two sampling departure times:

$$\begin{aligned} & t_1 + \overline{D}_1 = t_0 + (1+\varepsilon)[(A_0^+ - 1)t_0 + B_0^+] \\ \Rightarrow \quad & t_1 - t_0 = (1+\varepsilon)[(A_0^+ - 1)t_0 + B_0^+] - \overline{D}_1 \\ \Rightarrow \quad & t_1 - t_0 = (1+\varepsilon)[(A_0^+ - 1)t_0 + B_0^+] - (A_0^+ - 1)t_1 - B_0^+ \\ \Rightarrow \quad & A_0^+ \cdot (t_1 - t_0) = \varepsilon[(A_0^+ - 1)t_0 + B_0^+] = \varepsilon D_0 \\ \Rightarrow \quad & t_1 - t_0 = \frac{\varepsilon D_0}{A_0^+} \end{aligned}$$

Recall now that, by the determination of $t_1 = \underline{t}_{1,v}$, the following holds:

$$\begin{aligned} \overline{D}_1 &= (A_0^+ - 1)t_1 + B_0^+ = D_0^+ + (A_0^+ - 1) \cdot (t_1 - t_0) \\ &= (1+\varepsilon)D_0 - (t_1 - t_0) \\ &= \left(1 + \varepsilon - \frac{\varepsilon}{A_0^+}\right) D_0 \end{aligned}$$

We shall now quantify the actual increase (as a function of δ) in the actual delay value, when considering t_0 and t_1:

$$\begin{array}{rcl} D_1 & = & (1-\delta)\overline{D}_1 + \delta D_0 \\ & = & \left[(1-\delta)\cdot\left(1+\varepsilon-\frac{\varepsilon}{A_0^+}\right)+\delta\right]\cdot D_0 \\ & = & \left[1+(1-\delta)\cdot\left(\varepsilon-\frac{\varepsilon}{A_0^+}\right)\right]\cdot D_0 \\ & \overset{/*\ A_0^+\geq 2\ */}{\geq} & \left(1+\frac{1-\delta}{2}\cdot\varepsilon\right)\cdot D_0 \end{array}$$

We conclude our proof by quantifying the actual decrease of the earliest-arrival slope for $Arr[o,v]$:

$$\begin{array}{rcl} A_1^+ & \leq & \frac{t_1+D_1-(t_0+D_0)}{t_1-t_0} \\ & = & \frac{D_1-D_0}{t_1-t_0}-1 \\ & = & \frac{(1-\delta)\cdot\left(\varepsilon-\frac{\varepsilon}{A_0^+}\right)\cdot D_0}{\frac{\varepsilon D_0}{A_0^+}}-1 \\ & = & (A_0^+-1)\cdot(1-\delta)-1 \end{array}$$

□

References

[1] K. Cooke and E. Halsey. The shortest route through a network with time-dependent intermodal transit times. *Journal of Mathematical Analysis and Applications*, 14(3):493–498, 1966.

[2] Stuart E. Dreyfus. An appraisal of some shortest-path algorithms. *Operations Research*, 17(3):395–412, 1969.

[3] Luca Foschini, John Hershberger, and Subhash Suri. On the complexity of time-dependent shortest paths. In *Proc. of 22nd ACM-SIAM Symp. on Discr. Alg. (SODA '11)*, pages 327–341. ACM-SIAM, 2011.

[4] H. Imai and Masao Iri. An optimal algorithm for approximating a piecewise linear function. *Journal of Information Processing*, 9(3):159–162, 1987.

[5] Ariel Orda and Raphael Rom. Shortest-Path and Minimum Delay Algorithms in Networks with Time-Dependent Edge-Length. *Journal of the ACM*, 37(3):607–625, 1990.

S. Kontogiannis: Computer Science & Engineering Department, Dourouti University Campus, 45110 Ioannina, GREECE.

C. Zaroliagis: Computer Engineering & Informatics Dept., University of Patras, 26500 Rion, GREECE.

E-mail address: kontog@cs.uoi.gr and zaro@ceid.upatras.gr