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 – 050

ANALYSIS AND EXPERIMENTAL EVALUATION OF TIME-DEPENDENT DISTANCE ORACLES

SPYROS KONTOGIANNIS, GEORGE MICHALOPOULOS, GEORGIA PAPASTAVROU, ANDREAS PARASKEVOPOULOS, DOROTHEA WAGNER, CHRISTOS ZAROLIAGIS

September 2014

ANALYSIS AND EXPERIMENTAL EVALUATION OF TIME-DEPENDENT DISTANCE ORACLES

SPYROS KONTOGIANNIS$^{\dagger,\diamond}$, GEORGE MICHALOPOULOS$^{\ddagger,\diamond}$, GEORGIA PAPASTAVROU$^{\dagger,\diamond}$, ANDREAS PARASKEVOPOULOS$^{\ddagger,\diamond}$, DOROTHEA WAGNER$^{\between}$, AND CHRISTOS ZAROLIAGIS$^{\ddagger,\diamond}$

Abstract. Urban road networks are typically represented as directed graphs equipped with some metric which assigns distance *functions* (rather than scalars) to the arcs, e.g., representing the traversal times as functions of the departure time from their tails. Such a metric is usually created by sampling the distance values of all the arcs at certain points in time, and then considering the interpolants of these sampled values as the corresponding (periodic, continuous, piecewise linear) distance functions of the arcs. Distance oracles and speedup techniques aim to create appropriate traffic metadata (distance summaries or search profiles) during a preprocessing phase, in order to be able to respond to shortest-path queries significantly faster than a typical Dijkstra run. Distance oracles focus mainly on *provable* worst/average case guarantees on preprocessing time/space complexities, query time complexity and approximation ratio. Speedup techniques emphasize on the thorough experimental evaluation of their performance on real large-scale road network instances.

In this work, we experimentally evaluate the only time-dependent distance oracles existing so far on a truly real-world time-dependent data set (city of Berlin). In particular: (i) We present a new time-dependent distance oracle whose preprocessing phase is based on a new approximation technique for creating approximate distance summaries, that is a quite simple and fast one-to-all approximation method. (ii) We conduct an extensive experimental study with three query algorithms and six different landmark sets, achieving remarkable speedups over the time-dependent variant of Dijkstra'a algorithm.

We describe here all the implementation details concerning the digestion of the raw traffic data, along with several heuristic improvements of both the preprocessing phase and the query algorithms, which are crucial for their experimental analysis. Our results are quite encouraging, achieving speedups by two orders of magnitude versus a typical time-dependent Dijkstra run, while in the vast majority of the queries the exact solution is discovered.

Date: September 24, 2014.

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, route planning.

* Partially supported by EU FP7/2007-2013 under grant agreements no. 288094 (eCOMPASS) and no. 609026 (MOVESMART), and partially done while S. Kontogiannis and C. Zaroliagis were visiting KIT.

† Computer Science & Engineering Dept., University of Ioannina, 45110 Ioannina, GREECE. kontog@cs.uoi.gr, gioulycs@gmail.com.

‡ Computer Engineering & Informatics Dept., University of Patras, 26500 Rio, GREECE. {michalog,paraskevop,zaro}@ceid.upatras.gr.

$^{\diamond}$ Computer Technology Institute and Press "Diophantus", 26504 Rio, GREECE.

$^{\between}$ Karlsruhe Institute of Technology (KIT), Am Fasanengarten 5, 76131 Karlsruhe, GERMANY. dorothea.wagner@kit.edu .

1. Introduction

Distance oracles are succinct data structures encoding shortest path information among a carefully selected subset of pairs of vertices in a graph. The encoding is done in such a way that the oracle can efficiently answer shortest path queries for arbitrary origin-destination pairs, exploiting the preprocessed data and/or local shortest path searches. A distance oracle is exact (resp. approximate) if the returned distances by the accompanying query algorithm are exact (resp. approximate). A bulk of important work (e.g., [26, 25, 21, 22, 27, 28, 3]) is devoted to constructing distance oracles for *static* (i.e., *time-independent*), mostly undirected networks in which the arc-costs are fixed, providing trade-offs between the oracle's space and query time and, in case of approximate oracles, also of the stretch. For an overview of distance oracles for static networks, the reader is deferred to [24] and references therein.

Considerable experimental work on routing in large-scale road networks has also appeared in recent years, with remarkable achievements that have been demonstrated on continental-size road-network instances. The goal is again to preprocess the distance metric and then propose query algorithms (known as *speedup techniques*) for responding to shortest path queries in time that is several orders of magnitude faster than a conventional Dijkstra run. An excellent overview of this line of research is provided in [4]. Again, the bulk of the literature concerns static distance metrics, with only a few exceptions (e.g., [6, 12, 19]).

In many real-world applications, the arc costs may vary as functions of time (e.g., when representing travel-times) giving rise to *time-dependent* network models. A striking example is route planning in road networks where the travel-time for traversing an arc $a = uv$ (modelling a road segment) depends on the temporal traffic conditions while traversing uv, and thus on the departure time from its tail u. The time-dependent model turns out to be more appropriate for exploiting (the vast) historic traffic information in order to provide route plans that will adapt to the departure-time from the origin [14, 17].

Computing a time-dependent shortest path for a triple (o, d, t_o) of an origin-destination pair $(o, d) \in V \times V$, and departure-time t_o from the origin, has been studied since a long time ago (see e.g., [7, 13, 20]). The shape of arc-travel-time functions and the waiting policy at vertices may considerably affect the tractability of the problem [20]. It is customary to consider continuous, piecewise linear (pwl) interpolants of periodically sampled points as *arc-travel-time functions*. Regarding the waiting policy, the customary assumption is that each arc obeys the *FIFO property* (non-FIFO policies may lead to **NP**−hard cases [23]). If arc-travel-time functions possess the FIFO property, then the problem can be solved in polynomial time by a straightforward variant of Dijkstra's algorithm (we call it **TDD**), which relaxes arcs by computing the arc costs "on the fly", when settling their tails [13].

1.1. Related work.

Until recently, most of the previous work on the time-dependent shortest path problem concentrated on computing an optimal origin-destination path providing the earliest-arrival time at destination when departing at a *given* time from the origin, neglecting the computational complexity of providing succinct representations of the entire earliest-arrival-time *functions*, for *all* departure-times from the origin.

The complexity of succinctly representing earliest-arrival-time functions was first questioned in [8, 10, 9], but was solved only recently by a seminal work [15] which, for FIFO-abiding pwl arc-travel-time functions, showed that the problem of succinctly representing such a function for a *single origin-destination pair* has space-complexity $(1 + K) \cdot n^{\Theta(\log n)}$, where n is the number of vertices and K is the total number of breakpoints (or legs) of all the continuous, pwl arc-travel-time functions. Polynomial-time algorithms (or even PTAS) for constructing *point-to-point* $(1 + \varepsilon)$-approximate distance functions are provided in [15, 11], delivering point-to-point travel-time values at most $1 + \varepsilon$ times the true values. Such approximate distance functions possess *succinct representations*, since they require only $\mathcal{O}(1 + K)$

breakpoints per origin-destination pair. It is also easy to verify that K could be substituted by the number K^* of *concavity-spoiling* breakpoints of the arc-travel-time functions (i.e., breakpoints at which the arc-travel-time slopes increase).

Due to the hardness of providing succinct representations of exact shortest-travel-time functions, the only realistic alternative is to use approximations of these functions for computing (in a preprocessing phase) *distance summaries* from central vertices to all other vertices, which is a crucial ingredient for constructing distance oracles in time-dependent networks.

Providing distance oracles for time-dependent networks with *provably* good approximation guarantees, small preprocessing-space complexity and sublinear query time complexity, has only been recently investigated in [16, 17]. In particular, the *first* approximate distance oracle for sparse directed graphs with time-dependent arc-travel-times was presented in [17], which provides $(1+\sigma)-$approximate distances in query-time that is sublinear in the network size, and preprocessing time and space that are subquadratic in the network size, when the total number of concavity-spoiling breakpoints in the instance is sufficiently small, e.g. when $K^* \in \mathcal{O}(\text{polylog}(n))$. The oracle uses a new one-to-all method (called *Bisection* – **BIS**) to produce $(1+\varepsilon)-$approximate landmark-to-vertex distance summaries, for a randomly selected landmark set. It also guarantees either constant approximation ratio (a.k.a *stretch*) via the **FCA** query algorithm, or stretch at most $1+\sigma = 1+\varepsilon\frac{(1+\varepsilon/\psi)^{r+1}}{(1+\varepsilon/\psi)^{r+1}-1}$ via the **RQA** query algorithm, where ψ is a fixed constant depending on the characteristics of the arc-travel-time functions but is independent of the network size. $r \in \mathcal{O}(1)$ is the recursion depth of **RQA**.

A few time-dependent variants of well-known speedup techniques for road networks have also appeared in the literature (e.g., [6, 12, 19]). All of them were experimentally evaluated on synthetic time-dependent instances of the European and German road networks, with impressive performances. For example, in [6] methods are provided that respond to arbitrary queries of the German road network (4.7 million vertices and 10.8 million arcs) in less than $1.5ms$ and preprocessing space requirements of less than 1GB. A point-to-point distance summary can also be constructed in less than $40ms$, when the departure times interval is a single day. For point-to-point approximate distance summaries, with experimentally observed stretch at most 1%, the construction time is less than $3.2ms$. Their approach is based on the so-called time-dependent Contraction Hierarchies [5], along with several heuristic improvements both on the preprocessing step and on the query method.

1.2. Our Contribution.

Our goal in this work is to provide a thorough experimental evaluation of the time-dependent distance oracles that were proposed and analysed in [17]. The main obstacle towards this direction is the dependence of the required preprocessing time and space on the number K^* of concavity-spoiling breakpoints in the raw traffic data.

Our first contribution is a new time-dependent distance oracle, whose preprocessing phase for computing landmark-to-vertex approximate distance summaries is based on a new approximation technique [16], the *trapezoidal* (**TRAP**) method. This method is significantly simpler than **BIS** and reduces dramatically the required space. In particular, **TRAP** avoids any kind of dependence on the number K^* of concavity-spoiling breakpoints, which are completely neglected during the preprocessing and need not be computed at all.

Our second contribution is an extensive experimental study with three query algorithms and six different landmark sets, achieving remarkable speedups over **TDD** on truly real-world time-dependent data sets. More specifically, we experimentally compare **FCA**, **RQA**, and a new quite simple query algorithm, **FCA**$^+$, whose approximation guarantee is similar to that of **FCA**, but which in practice behaves very well, sometimes even better than **RQA**.

We conduct our experimental evaluation on the historic traffic data for the city of Berlin, kindly provided to us by TomTom within [14]. The input instance is a directed graph with $478,989$ vertices and $1,134,489$ arcs. We created six different landmark sets, with either 1000 or 2000 landmarks, and which were chosen either randomly, or as the boundary vertices of appropriate METIS [1] or KaHIP [2] partitions of the Berlin graph. The speedups that we achieve over the average time of a **TDD** run, vary from 397 times (using 1000 randomly chosen landmarks and **FCA**), to 723 times (using 2000 randomly chosen landmarks and **FCA**), for $10.3ms$ resolution in the approximate distance summaries. In both cases the average relative error is less than 1.634%. Analogous speedups are observed if our quality measure is not the computational time, but the (machine-independent) number of settled vertices (a.k.a. Dijkstra rank) of the query algorithms. The best possible observed relative error is indeed much better than the theoretical bounds provided by the analysis of the query algorithms. In particular, it is as small as 0.382% for 1000 KaHIP landmarks, or 0.298% for 2000 KaHIP landmarks, for $10.3ms$ resolution in the approximate distance summaries. The corresponding speedups are more 38 times for the former, and 118 times for the latter.

If we focus on the absolute response times, we manage to provide responses (via **FCA**) to arbitrary queries, in times less than $0.4ms$ for all landmark sets that we used, with relative error no more than 2.201%. For relative error at most 0.701%, we can provide answers in no more than $1.345ms$ using **FCA**$^+$, for all the considered landmark sets.

As for the preprocessed data, we create *all* the $300K$ approximate distance summaries from a given landmark in average sequential time less than $40sec$. That is, the amortized time per approximate distance summary is no more than $0.134ms$.

1.3. **Paper Organization.** Section 2 provides some notation and preliminaries for time-dependent network instances. Section 3 provides an overview of the **TRAP** approximation methods for producing approximate distance summaries, the preprocessing phase for producing all landmark-to-vertex approximate distance summaries, and the query algorithms **FCA**, **FCA**$^+$ and **RQA**, to be experimentally tested. Section 4 presents the results of our experimental experimental evaluation on the real instance of Berlin. Section **??** provides some concluding remarks and directions for further research and experimentation.

2. Preliminaries

We consider directed graphs $G = (V, A)$ with $|V| = n$ vertices and $|A| = m$ arcs, where each arc $a \in A$ is accompanied with a continuous, periodic, piecewise linear (pwl) *arc-travel-time* (or *arc-delay*) function defined as follows: $\forall k \in \mathbb{N}, \forall t \in [0, T),\ D[a](kT + t) = d[a](t)$, where $d[a] : [0, T) \to [1, M_a]$ such that $\lim_{t \uparrow T} d[a](t) = d[a](0)$, for some fixed integer M_a denoting the maximum possible travel time ever seen at arc a. Observe that the minimum arc travel time value ever seen in the entire network is also normalized to 1. Since every arc-travel-time function $D[a]$ is periodic, continuous and pwl, it can be represented succinctly by a number K_a of breakpoints defining $d[a]$. Let $K = \sum_{a\in A} K_a$ be the number of breakpoints to represent all the arc-travel-time functions in G, let $K_{\max} = \max_{a\in A} K_a$, and let K^* be the number of *concavity-spoiling* breakpoints, i.e., those in which the arc-travel-time slopes increase. Clearly, $K^* \leq K$, and $K^* = 0$ for *concave* pwl arc-travel-time functions.

The *arc-arrival-time* functions are defined as $Arr[a](t) = t + D[a](t),\ \forall t \in [0, \infty)$. The *path-arrival-time* function of a given path $p = \langle a_1, \dots, a_k\rangle$ in G (represented as a sequence of arcs) is defined as the composition of the arc-arrival-time functions for the constituent arcs: $Arr[p](t) = Arr[a_k](Arr[a_{k-1}](\cdots (Arr[a_1](t)) \cdots))$. The *path-travel-time* function is then $D[p](t) = Arr[p](t) - t$. Finally, between any origin-destination pair of vertices, $(o, d) \in V \times V$, $\mathcal{P}_{o,d}$ denotes the set of all $od-$paths in G, and the *earliest-arrival-time / shortest-travel-time* functions are defined as follows: $\forall t_o \geq 0,\ Arr[o, d](t_o) = \min_{p\in\mathcal{P}_{o,d}} \{Arr[p](t_o)\}$ and $D[o, d](t_o) = \min_{p\in\mathcal{P}_{o,d}} \{D[p](t_o)\} = Arr[o, d](t_o) - t_o$.

For any arc $a = uv \in A$ and any departure-times subinterval $[t_s, t_f) \subseteq [0, T)$, we consider the free-flow and maximally-congested travel-times for this arc, defined as follows:

- *Free-flow* arc-travel-time: $\underline{D}[uv](t_s, t_f) := \min_{t_u \in [t_s, t_f)} D[uv](t_u)$.
- *Maximally-congested* arc-travel-time: $\overline{D}[uv](t_s, t_f) := \max_{t_u \in [t_s, t_f)} D[uv](t_u)$.

We also denote $\overline{D}[uv] := \overline{D}[uv](0, T)$ and $\underline{D}[uv] := \underline{D}[uv](0, T)$. When $[t_s, t_f) = [0, T)$ then we refer to the *static* free-flow and full-congestion distance metrics $\underline{D}$ and $\overline{D}$, respectively. These definitions also extend naturally to path-travel-times and shortest-travel-times between arbitrary origin-destination pairs of vertices.

For a point $(o, t_o) \in V \times [0, T)$ and $\beta \in \mathbb{N}$, let $B[o](t_o; \beta)$ be the set of the first β vertices settled by **TDD**, when growing a ball from (o, t_o). Analogously, $\underline{B}[o](\beta)$ and $\overline{B}[o](\beta)$ are the corresponding sets under the free-flow and fully-congested metrics $\underline{D}$ and $\overline{D}$, respectively.

For an arbitrary pair $(o, d) \in V \times V$ of origin-destination vertices, a succinctly represented $(1 + \varepsilon)$-*upper-approximation* of $\Delta[o, d]$, is a continuous pwl function, hopefully with a *small* number of breakpoints, such that $\forall t_o \geq 0,\ D[o, d](t_o) \leq \Delta[o, d](t_o) \leq (1 + \varepsilon) \cdot D[o, d](t_o)$.

We adopt two assumptions from [17], and one additional assumption from [16], on the kind of shortest-travel-time functions in the network. All of them are quite natural and justified in urban-traffic road networks. Actually, we conducted an experimental analysis on the real-world instance of Berlin that we had at our disposal, which indeed verified the validity of the assumptions. Technically, they allow the *smooth transition* from static distance metrics on undirected graphs towards time-dependent distance metrics on directed graphs. For a more thorough justification, the reader is deferred to [17, 16].

The first assumption asserts that the partial derivatives of the shortest-travel-time functions between any origin-destination pair are bounded in a fixed interval $[\Lambda_{\min}, \Lambda_{\max}]$.

Assumption 2.1 (Bounded Travel-Time Slopes)**.** *There are* constants $\Lambda_{\min} \in [0, 1)$ *and* $\Lambda_{\max} \geq 0$ *s.t.:* $\forall (o, d) \in V \times V,\ \forall t_1 < t_2,\ \frac{D[o,d](t_1) - D[o,d](t_2)}{t_1 - t_2} \in [-\Lambda_{\min}, \Lambda_{\max}]$.

It is mentioned that the lower-bound of -1 in the shortest-travel-time function slopes is indeed a direct consequence of the FIFO property, which is typically assumed to hold in several time-dependent networks and allows for the use of time-dependent variants of classical shortest-path computation techniques, such as Dijkstra's and Bellman-Ford algorithms. Our experimental analysis on the historic traffic data for the city of Berlin, in which the maximum value of $\Lambda_{\max}$ in a series of 10000 randomly chosen origin-destination pairs was always less than 0.19.

The second assumption asserts that for any given departure time, the shortest-travel-time from o to d is not more than a *constant* $\zeta \geq 1$ times the shortest-travel-time in the opposite direction (but not necessarily along the reverse path). This is quite natural in road networks, and it was indeed justified by our experimental analysis on the historic traffic data for the city of Berlin, in which the maximum value of ζ in a series of 10000 randomly chosen origin-destination pairs was always less than 1.5.

Assumption 2.2 (Bounded Opposite Trips)**.** *There is a constant* $\zeta \geq 1$ *such that:* $\forall (o, d) \in V \times V,\ \forall t \in [0, T),\ D[o, d](t) \leq \zeta \cdot D[d, o](t)$.

An additional assumption, which will only be considred occasionally (and this will be explicitly stated in such cases), concerns the relation of the Dijkstra ranks (i.e., number of settled vertices, up to termination) of cocentric balls in the network, with respect to the (static) *free-flow* metric that implied by our time-dependent instance:

Assumption 2.3 (Growth of Free-Flow Dijkstra Ranks)**.** *For any vertex* $\ell \in V$*, and positive integer* $F \in \mathbb{N}$*, assume growing a (static) Dijkstra ball* $\underline{B}[\ell](F)$ *around* ℓ*. Let* $\underline{R}[\ell] = \max\{\underline{D}[\ell, v] : v \in \underline{B}[\ell](F)\}$ *be the largest free-flow travel-time from* ℓ *to any other vertex in* $\underline{B}[\ell](F)$*. Also let* $\overline{R}[\ell] = \max\{\overline{D}[\ell, v] : v \in \underline{B}[\ell](F)\}$ *be the largest full-congestion travel-time* *from ℓ to any other vertex in $\underline{B}[\ell]$. Finally, let $\underline{B}' = \{v \in V : \underline{D}[\ell, v](0,T) \leq \overline{R}[\ell]\}$ be the free-flow ball around ℓ with radius $\overline{R}[\ell]$. Then it holds that $|\underline{B}'[\ell]| \in \mathcal{O}(F \cdot \mathrm{polylog}(F))$.*

This assumption has also been experimentally tested in the Berlin instance, for various initial ball sizes. In all cases the scaling factor of the ball size was less than 2.

3. Time-Dependent Oracles

In [16] time-dependent distance oracles are proposed and theoretically analysed, which preprocess distances using the novel **TRAP** method for approximating distance summaries and use one of two query algorithms, **FCA**, or **RQA**. The novelty of these oracles is that they assure subquadratic storage space and sublinear query complexity, irrespectively of the degree of disconcavity of the distance metric, measured by the value of K^*. In this work we experimentally evaluate all these oracles, and also experiment with another query algorithm, called **FCA**$^+$, that we propose.

All the oracles start by selecting a subset $L \subset V$ of *landmarks*. This can be done either randomly (e.g., by deciding for each vertex i.u.r with probability $\rho \in (0,1)$ whether it belongs to L), or by selecting L from the vertices in the cut sets provided by some graph partitioning algorithm. In this work we consider appropriate METIS and KaHIP partitions of the Berlin graph. After L is determined, a preprocessing phase is performed in which, $\forall \ell \in L$ and $\forall v \in V$, all ℓ-to-v $(1+\varepsilon)-$upper-approximating travel-time functions (*approximate distance summaries*) are computed and stored, based on the **TRAP** method. Consequently, one of the three different query algorithms, **FCA**, **FCA**$^+$, or **RQA** is used for providing in sublinear time *guaranteed* approximations of the actual shortest travel time values, for arbitrary queries $(o, d, t_o) \in V \times V \times [0,T)$. In a final step, a path-construction step is run to provide an *od*-path with actual path-travel-time that is at most the predicted one. In this section, we briefly review the above mentioned ingredients of our oracles.

3.1. Approximate Distance via the Trapezoidal Method.

TRAP splits the entire period $[0,T)$ into small, consecutive subintervals of length $\tau > 0$ each. It then provides a crude approximation of the unknown distance functions in each interval, solely based on Assumption 2.1 concerning the minimum slope $-\Lambda_{\min}$ and maximum slope $\Lambda_{\max}$ of shortest-travel-time functions in the instance. After sampling the travel-time values of each destination $v \in V$, for a given origin $u \in V$, we consider each pair of consecutive sampling times $t_s < t_f$ and the semilines with slopes $\Lambda_{\max}$ from t_s and $-\Lambda_{\min}$ from t_f. The considered upper-approximating function $\overline{D}[u,v]$ within $[t_s, t_f)$ is then (a refinement of) the lower-envelope of these two lines. Analogously, an lower-approximating function $\underline{D}[u,v]$ is the upper-envelope of the semilines that pass through t_s with slope $-\Lambda_{\min}$, and from t_f with slope $\Lambda_{\max}$. Depending on the value of the absolute error and the minimum possible value of $\underline{D}[u,v]$ in this interval, we can decide whether the $\overline{D}[u,v]$ is a $(1+\varepsilon)$-upper-approximating function of $D[u,v]$. Any destination vertex that has such an approximating function for each subinterval of $[0,T)$, clearly has a $(1+\varepsilon)$-upper-approximating functions for the entire period. The proof of correctness of **TRAP** is provided in Section A, while a more detailed discussion can be found in [16].

The problem with the trapezoidal approximation is that, by construction, it is not possible to provide approximate distance functions of a given approximation guarantee $1+\varepsilon$, for "nearby" destination vertices which are too close to the origin. In [16] these "nearby" vertices of each landmark are either handled by the **BIS** method [17], or are left to be handled on the fly. Here we resolve this issue exclusively with **TRAP**, starting with a large subinterval length, and recursively dividing by 2 the lengths of those subintervals containing vertices which have not been sufficiently approximated yet, until all landmark-to-vertex $(1+\varepsilon)$-approximate distance summaries have been successfully created. This proved to be extremely space- and time-efficient in practice.

3.2. **Query Algorithms.** For computing arbitrary origin/destination/departure-time queries (o, d, t_o), three approximation algorithms are considered. The first one, called **FCA**, is a simple *sublinear*-time constant-approximation algorithm, which works as follows. It grows an outgrowing ball $B_o \equiv B[o](t_o) = \{x \in V : D[o, x](t_o) < D[o, \ell_o](t_o)\}$ from (o, t_o) by running **TDD** until either d or the closest landmark $\ell_o \in \arg\min_{\ell \in L}\{D[o, \ell](t_o)\}$ is settled. **FCA** returns either the exact travel-time value, or the approximate travel-time value via ℓ_o with an $1 + \varepsilon + \psi$ approximation guarantee, where ψ is a constant depending on ε, ζ, and $\Lambda_{\max}$, but not on the size of the network.

The second query algorithm, called **FCA$^+$**, is a variant of **FCA** which keeps growing a **TDD** ball from (o, t_o) until either d or a given number N of landmarks is settled. **FCA$^+$** returns the smallest via-landmark approximate travel-time value, along all these settled landmarks. The approximation guarantee is the same as that of **FCA**, but in practice it performs quite well, in certain cases even better than **RQA**, as it will be demonstrated in the experimental evaluation.

The third algorithm, called **RQA**, improves the approximation guarantee of the chosen $od-$path to $1 + \varepsilon \cdot \frac{(1+\varepsilon/\psi)^{r+1}}{(1+\varepsilon/\psi)^{r+1}-1}$, by exploiting carefully a number $r \in \mathbb{N}$ (called the *recursion budget*) of recursive accesses to the preprocessed information, each of which produces (via calls to **FCA**) additional candidate $od-$paths sol_i. **RQA** works as follows. As long as the destination vertex within the explored area around the origin has not yet been discovered, and there is still some remaining recursion budget, it "guesses" (by exhaustively searching for it) the next vertex w_k of the boundary set of touched vertices (in the priority queue) along the unknown shortest $od-$path. Then, it grows a new outgrowing **TDD** ball from the new center $(w_k, t_k = t_o + D[o, w_k](t_o))$, until it reaches the closest landmark-vertex ℓ_k to it, at distance $R_k = D[w_k, \ell_k](t_k)$. This new landmark offers an alternative $od-$path $sol_k = P_{o,k} \bullet Q_k \bullet \Pi_k$ by a new application of **FCA**, where $P_{o,k} \in SP[o, w_k](t_o)$, $Q_k \in SP[w_k, \ell_k](t_k)$, and $\Pi_k \in ASP[\ell_k, d](t_k + R_k)$ is the approximate suffix subpath provided by the distance oracle (in case of the **TRAP** scenario, it has to be computed "on-the-fly" by growing an additional **TDD** ball from the corresponding landmark). Observe that sol_k uses a longer (optimal) prefix-subpath P_k which is then completed with a shorter approximate suffix-subpath $Q_k \bullet \Pi_k$. **RQA** finally responds with a $(1 + \sigma)-$approximate travel-time to the query in *sublinear* time, for any constant $\sigma > \varepsilon$.

A more detailed presentation of **FCA** and **RQA**, along with the proofs of correctness and their time complexities, are provided in [17]. As for the approximation guarantee of **FCA$^+$**, it is straightforward to observe that, at least theoretically, it is as small as that of **FCA**, whereas its time complexity is comparable to that of **RQA**.

3.3. **Heuristic Improvements.** The **TRAP** approximation method introduces at least one intermediate (possibly two) breakpoint per interval that does not yet meet the required approxmation guarantee. This is certainly unnecessary for intervals in which the actual shortest-travel-time functions are almost constant. To avoid the blow-up of the preprocessing space required, we heuristically make an arbitrary "guess" that we have to deal with an "almost constant" shortest-travel-time function $D[\ell, v]$ within a given interval $[t_s, t_f)$, if the following holds: $D[\ell, v](t_s) = D[\ell, v](t_f) = D[\ell, v]\left(\frac{t_s+t_f}{2}\right)$. This is justified by the fact that $D[\ell, v]$ is a continuous pwl function, along with the fact that already $t_f = t_s + \tau$ for some small value $\tau > 0$. Of course, one could easily construct artificial examples for which this criterion is clearly violated, e.g., by providing a properly chosen periodic function with period τ. On the other hand, one can easily tackle this by considering a *randomly perturbed* sampling period $\tau + \delta$, for some arbitrarily small but positive random variable δ.

Another improvement that we adopt is that, rather than splitting the entire period $[0, T)$ in a flat manner, i.e., into equal-size intervals , we start with a coarse partitioning based on a large length and then in each inteval and for each destination vertex we check for the provided approximation guarantee by **TRAP**. All the vertices which are already satisfied by this guarantee with respect to the current interval, become inactive for this and all its subsequent subintervals. We then proceed by splitting in the middle each subinterval that contains at least one still active destination vertex, and repeating the check for all active vertices within the new subintervals.

4. Experimental Evaluation

4.1. **Preprocessing The Road Instance.** The Berlin instance, kindly provided by TomTom, consists of a directed graph with $478,989$ vertices and $1,134,489$ arcs; $924,254$ of the arcs have constant arc-travel-times. For the remaining $202,214$ arcs, continuous pwl arc-travel-time functions are provided, concerning an entire weekday (Tuesday). The maximum arc-travel-time slope is 0.0166667, whereas the minimum slope is -0.0133333. The succinct representation of these functions requires a total number of $3,234,213$ breakpoints. We substituted each maximal path in the network consisting of intermediate vertices with no intersections (i.e., would have degree 2 in the simple, undirected version of the graph), with a single shortcut arc of arc-travel-time function equal to the corresponding path-travel-time function. This results in a reduced graph consisting of $299,693$ vertices and $950,504$ arcs.

We generated two data formats suitable as input for our query algorithms. The first concerns the arc-travel-time functions and the second the preprocessed approximate distance summaries (i.e., landmark-to-vertex approximate travel-time functions).

4.1.1. *Arc-Travel-Time Functions.* The raw-traffic data set is provided as arrays with average speed estimations. Each row of such an array corresponds to a particular arc indicating a road segment. The columns provide a partition of the entire one-day period into 288 5-min timeslots. The arc-travel-time value of an arc $a = uv$ for a timestlot i is computed as $length/[S_{(a,i)} \times (free\ flow\ speed)_a]$, where $free\ flow\ speed$ denotes the top speed that can be achieved with zero congestion along a, while $S_{(a,i)}$ denotes a scale factor dependent on the road traffic status of timeslot i. Therefore, for arc a a sequence of ($departure\ time, arc\ travel\ time$) breakpoints is created, where the *departure time* is the starting point of the corresponding timeslot.

The corresponding arc-travel-time function $D[a](t)$ was created by running the following preprocessing, for each arc and arc-travel-time values per timeslot. In order to avoid wasting space, consecutive timeslots having the same arc-travel-time value were merged. Optionally, one could perform a broader merging of consecutive timeslots having absolute difference in arc-travel-time values less than a small constant (e.g. $< 1min$ resolution bound). However, in our experiments we chose to preserve the maximum possible resolution of the raw-traffic data. This proved to be extremely efficient by means of approximation guarantees, for different levels of resolution for the approximate distance summaries. The arc-travel-time function $D[a]$ for a particular arc a is the continuous, pwl interpolant of all the ($departure\ time, arc\ travel\ time$) breakpoints in its previously constructed succinct representation. The breakpoints are stored in binary and double-precision floating-point format. The eventual space required for all the raw-traffic data to be provided as input, is roughly 225MB.

4.1.2. *Preprocessed Landmark Information.* In order to create all the landmark-to-vertex approximate distance summaries, we call **TRAP**, which is a one-to-all approximation method, once per landmark. Upon completion of this preprocessing phase, we collect the approximate distance summaries for all the landmark-vertex pairs in the Berlin graph. For each such pair $(\ell, v) \in L \times V$, we store a sequence of breakpoints $(Dep[\ell], Arr[\ell, v])$, where $Dep[\ell]$ denotes the departure-time from landmark ℓ and $Arr[\ell, v]$ denotes the corresponding arrival-time at v. The interpolation of all these breakpoints produces the overall approximate distance summary (continuous, pwl function) $D[\ell, v](t)$. With $|L|$ landmarks and p breakpoints (on average) per approximate distance summary, the preprocessing space required for storing all the landmark-to-vertex approximate distance summaries is $\mathcal{O}(|L|pn)$.

Our approach is focused on achieving a cost effective storage of the approximate distance summaries, while keeping a sufficient precision. The key is that some specific features can be exploited in order to reduce the required space. The main observation is that, for a one-day time period, departure-times and arrival-times have a bounded value range. In particular, when the considered precision of the traffic data is within seconds we handle time-values as integers in the range $[0, 86399]$, for milliseconds as integers in $[0, 86399999]$, etc.

Any (real) time value within a single-day period, as a floating-point number t_f, can be converted to an integer t_i with fewer bytes and a unit of measure. For a unit measure (or scale factor) s, the resulting integer is $t_i = \lceil t_f/s \rceil$. In this manner, t_i needs size $\lceil \log_2(t_f/s)/8 \rceil$ bytes. The division t_f/s has quotient π and remainder v. Thus, $t_f = s \cdot \pi + v$ and $t_i = \lceil (s \cdot \pi + v)/s \rceil = \lceil \pi + v/s \rceil$, with $v < s$. Therefore, converting t_f to t_i results to an absolute error of at most $2s$. In the reverse process, for extracting the stored value, the conversion is $t'_f = t_i \cdot s$. In our experiments, for storing the time-values of approximate distance summaries, we have considered two different resolutions:

(a) $2.64sec$ resolution, corresponding to a scale factor $s = 1.32$ (when counting time in seconds), requiring 2 bytes per time-value, and
(b) $10.3ms$ resolution, corresponding to a scale factor $s = 5.15$ (when counting time in milliseconds), requiring 3 bytes per time-value.

4.2. **Experimental Setup.** All algorithms were implemented using C++ (gcc, version 4.6.3). To support all graph-operations we used the PGL library [18]. All experiments were executed by a CPU of 3.40GHz×8, using 16GB of RAM, on Ubuntu 12.04 LTS. All our algorithms are executed sequentially. Exploitation of parallelism is left for future implementations, and is anticipated to reduce dramatically the execution times, particularly for the preprocessing phase and the query algorithm **RQA** for which parallelism would apply quite naturally.

4.3. Measurements and Evaluation.

4.3.1. *Preprocessing Phase: Creation of Approximate Distance Summaries.* Our preprocessing phase took as input six different landmark sets for the Berlin graph: R_{1000} and R_{2000} correspond to 1000 and 2000 landmarks chosen uniformly at random from the entire vertex set. M_{1000} and M_{2000} correspond to 1021 and 2072 landmarks chosen as the boundary vertices of appropriate METIS partitions. K_{1000} and K_{2000} correspond to 1016 and 2024 landmarks chosen as the boundary vertices of appropriate KaHIP partitions.

For the production of the approximate distance summaries for each of the landmark sets, a total amount of less than 13 hours (for small sets) and 26 hours (for large sets) of sequential computational time was consumed. In particular, the average time per landmark, for producing its approximate distance summaries towards *all* the possible destinations is less than $43sec$, and the amortized time for constructing a single landmark-to-vertex approximate distance summary is less than $0.1435ms$.

The required storage space is less than 35MB per landmark for $2.64sec$ resolution, and 55MB per landmark for the $10.3ms$ resolution.

4.3.2. *Query Phase: Responding to Arbitrary Shortest-Path Queries.* The query execution times and relative errors of the produced solutions, for all possible landmark sets and the two different resolutions that we consider for the approximate distance summaries, are presented in Tables 1 and 2. Moreover, Table 3 presents the speedups of the query algorithms, measured by the machine-independent criterion of Dijkstra-rank, i.e., the number of settled vertices during execution. All reported values are averages over 1000 randomly chosen queries from the Berlin instance. We note that for **RQA** the recursion budget was set to 1. For fairness of comparison, the parameter N (number of landmarks) in $\mathbf{FCA}^+$ was set equal to the number of landmarks settled by **RQA**.

It should be noted that for the query algorithms we only count the required computational time for providing an upper bound on the earliest-arrival-time at the destination. In particular, we exclude the time required for the construction time of a path with the discovered guarantee (which is anyway negligible) and the time required for accessing from the hard disk the approximate distance summaries of the involved landmarks. The latter is done for two reasons: First, we wish our comparison to be as independent as possible of the characteristics of the machine, and in particular of the size of the main memory. For example, the reported times would be as they appear in Tables 1 and 2 in exactly the same machine but with sufficiently large main memory. Second, our main quality measure is the achieved speedup versus the average performance of **TDD**. Clearly, **TDD** produces no disc-IO accesses when being executed, and the comparison would be misleading for the query algorithms, simply due to poor hardware characteristics. We wish to have a clear comparison of the algorithms themselves, which is irrelevant of the hardware platform.

Apart from query-times, we also report the observed *relative errors* of the produced solutions. The relative error for a given $od-$path p is the percentage of surplus from the exact distance (as computed by **TDD**), i.e.:

$$100 \cdot (\text{travel time of } p - \text{exact distance from } o \text{ to } d)/(\text{exact distance from } o \text{ to } d)$$

With respect to the observed query times, in all cases **FCA** is the fastest query method, but with the highest relative error, compared to the other two methods. For example, it returns answers with relative error 1.634% in $0.195ms$ (i.e., a speedup more than 397 over the runtime of **TDD**), for R_{1000} and $10.3ms$-resolution. The response time for R_{2000} and $10.3ms$-resolution is $0.107ms$ (i.e., speedup more than 723) with relative error 1.065%. Similar performance is observed also for the cases of preprocessing with $2.64sec$-resolution. For the other two query algorithms, $\mathbf{FCA}^+$ is always faster than **RQA**, the latter being at most two times slower than the former. This can be justified by the fact that $\mathbf{FCA}^+$ grows a unique Dijkstra ball from the origin, and thus acts like a label-setting algorithm. On the other hand, **RQA** may visit and update the labels of the same vertices more than once, since at the second level of the recursion the labels of the settled nodes are not always shortest path distances from the origin, but shortest path distances via particular parents. On the other hand, it should be noted that **RQA** is amenable to parallelization due to its recursive flavor. This is anticipated to speedup significantly its query time.

With respect to the relative error, we observe that for all the random landmark sets $\mathbf{FCA}^+$ provides smaller values, of 0.449% for 1000 random landmarks and 0.389% for 2000 random landmarks. For the rest of the landmark sets, **RQA** is the best option with respect to the relative error, achieving values 0.314% for 1000 KaHIP landmarks and 0.298% for 2000 KaHIP landmarks.

As for the machine-independent performance of Dijkstra-ranks, we observe that the reported average speedups compared to a typical **TDD** run, are even better. For example, using **FCA** on R_{1000} and R_{2000} produce speedups larger than 429 and 889 times respectively.

A final remark remark is the sensitivity of our algorithms to the choice of resolution for the values of the approximate distance summaries that are created during the preprocessing phase. Observe that if the performance measure is the Dijkstra-rank, then the choice of resolution, which only affects the approximate values of the landmark-to-destination distances, is irrelevant of the rank measure, because the Dijkstra balls grow over the raw traffic data for which we have preserved the maximum possible accuracy. Even when we account for computational times of the query algorithms, we observe that the difference in the relative errors is rather negligible, and in a few cases the coarser resolution of $2.64sec$ results in smaller relative-error values. This is due to the path reconstruction method that we use, which also takes into account the values of the approximate landmark-to-vertex distance values. The main reason for this insensitivity in the chosen resolution is that it is only the last part of the chosen path that is indeed affected, by only a small additive term of few seconds / milliseconds.

	TDD		**FCA**		**FCA$^+$**		**RQA**	
	Time (ms)	Rel.Error (%)	Time (ms)	Rel.Error (%)	Time (ms)	Rel.Error (%)	Time (ms)	Rel.Error (%)
R_{1000}	77.424	0	**0.195**	1.634	1.345	**0.449**	1.692	0.575
M_{1000}			**0.381**	2.201	1.313	0.698	2.349	**0.483**
K_{1000}			**0.362**	2.165	1.223	0.506	2.015	**0.382**
R_{2000}			**0.107**	1.065	0.71	**0.389**	0.771	0.445
M_{2000}			**0.152**	1.115	0.582	0.336	0.7	**0.314**
K_{2000}			**0.148**	1.405	0.599	0.367	0.655	**0.298**

TABLE 1. Query performances for $10.3ms$-resolution of approximate distance summaries.

	TDD		**FCA**		**FCA$^+$**		**RQA**	
	Time (ms)	Rel.Error (%)	Time (ms)	Rel.Error (%)	Time (ms)	Rel.Error (%)	Time (ms)	Rel.Error (%)
R_{1000}	77.424	0	**0.198**	1.634	1.345	**0.449**	1.712	0.574
M_{1000}			**0.381**	2.199	1.287	0.7	2.09	**0.487**
K_{1000}			**0.348**	2.171	1.197	0.512	1.834	**0.381**
R_{2000}			**0.108**	1.065	0.694	**0.382**	0.769	0.442
M_{2000}			**0.156**	1.116	0.589	0.346	0.767	**0.314**
K_{2000}			**0.148**	1.401	0.591	0.366	0.721	**0.295**

TABLE 2. Query performances for $2.64sec$-resolution of approximate distance summaries.

	TDD		**FCA**		**FCA$^+$**		**RQA**	
	Rank	Speedup	Rank	Speedup	Rank	Speedup	Rank	Speedup
R_{1000}	149397	1	**348**	**429.302**	2628	56.848	4261	35.061
M_{1000}			713	209.533	2517	59.355.7	5304	28.167
K_{1000}			657	227.393	2353	63.492	4660	32.059
R_{2000}			**168**	**889.268**	1251	119.422	1820.769	82.086
M_{2000}			252	592.845	1039	143.789	1646	90.764
K_{2000}			247	604.846	1002	149.099	1522	98.158

TABLE 3. Query performances with respect to the numbers of settled vertices.

References

[1] Metis – serial graph partitioning and fill-reducing matrix ordering, 2013. stable version: 5.1.0.

[2] KaHIP – Karlsruhe High Quality Partitioning, May 2014.

[3] Rachit Agarwal and Philip Godfrey. Distance oracles for stretch less than 2. In *Proceedings of the 24th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'13)*, pages 526–538. ACM-SIAM, 2013.

[4] Hannah Bast, Daniel Delling, Andrew V. Goldberg, Mattias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato Werneck. Route planning in transportation networks. Technical Report MSR-TR-2014-4, Microsoft Research, January 2014.

[5] Gernot Veit Batz, Daniel Delling, Peter Sanders, and Christian Vetter. Time-Dependent Contraction Hierarchies. In *Proceedings of the 11th Workshop on Algorithm Engineering and Experiments (ALENEX'09)*, pages 97–105. SIAM, April 2009.

[6] Gernot Veit Batz, Robert Geisberger, Peter Sanders, and Christian Vetter. Minimum time-dependent travel times with contraction hierarchies. *ACM Journal of Experimental Algorithmics*, 18, 2013.

[7] 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.

[8] Brian C. Dean. Continuous-time dynamic shortest path algorithms. Master's thesis, Massachusetts Institute of Technology, 1999.

[9] Brian C. Dean. Algorithms for minimum-cost paths in time-dependent networks with waiting policies. *Networks*, 44(1):41–46, 2004.

[10] Brian C. Dean. Shortest paths in fifo time-dependent networks: Theory and algorithms. Technical report, MIT, 2004.

[11] Frank Dehne, Omran T. Masoud, and Jörg-Rüdiger Sack. Shortest paths in time-dependent FIFO networks. *ALGORITHMICA*, 62(1-2):416–435, 2012.

[12] Daniel Delling. Time-Dependent SHARC-Routing. *Algorithmica*, 60(1):60–94, May 2011. Special Issue: European Symposium on Algorithms 2008.

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

[14] eCOMPASS Project (2011-2014). `http://www.ecompass-project.eu`.

[15] Luca Foschini, John Hershberger, and Subhash Suri. On the complexity of time-dependent shortest paths. *Algorithmica*, 68(4):1075–1097, 2014. Preliminary version in ACM-SIAM SODA 2011.

[16] Spyros Kontogiannis, Dorothea Wagner, and Christos Zaroliagis. Hierarchical distance oracles for time-dependent networks. Technical report, eCOMPASS Project, August 2014.

[17] Spyros Kontogiannis and Christos Zaroliagis. Distance oracles for time-dependent networks. In *J. Esparza et al. (eds.), ICALP 2014, Part I*, volume 8572 of *LNCS*, pages 713–725. Springer-Verlag Berlin Heidelberg, 2014. Full version as eCOMPASS Technical Report (eCOMPASS-TR-025) / ArXiv Report (arXiv.org>cs>arXiv:1309.4973), September 2013.

[18] Georgia Mali, Panagiotis Michail, Andreas Paraskevopoulos, and Christos Zaroliagis. A new dynamic graph structure for large-scale transportation networks. In *Algorithms and Complexity*, volume 7878 of *Lecture Notes in Computer Science*, pages 312–323. Springer, 2013.

[19] Giacomo Nannicini, Daniel Delling, Dominik Schultes, and Leo Liberti. Bidirectional A* Search on Time-Dependent Road Networks. *Networks*, 59:240–251, 2012. Journal version of WEA'08.

[20] 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.

[21] Mihai Patrascu and Liam Roditty. Distance oracles beyond the Thorup–Zwick bound. In *Proc. of 51th IEEE Symp. on Found. of Comp. Sci. (FOCS '10)*, pages 815–823, 2010.

[22] Ely Porat and Liam Roditty. Preprocess, set, query! In *Proc. of 19th Eur. Symp. on Alg. (ESA '11)*, LNCS 6942, pages 603–614. Springer, 2011.

[23] Hanif D. Sherali, Kaan Ozbay, and Sairam Subramanian. The time-dependent shortest pair of disjoint paths problem: Complexity, models, and algorithms. *Networks*, 31(4):259–272, 1998.

[24] Christian Sommer. Shortest-path queries in static networks. *ACM Computing Surveys*, 46, 2014.

[25] Christian Sommer, Elad Verbin, and Wei Yu. Distance oracles for sparse graphs. In *Proc. of 50th IEEE Symp. on Found. of Comp. Sci. (FOCS '09)*, pages 703–712, 2009.

[26] Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. of ACM*, 52:1–24, 2005.

[27] C. Wulff-Nilsen. Approximate distance oracles with improved preprocessing time. In *Proc. of 23rd ACM-SIAM Symp. on Discr. Alg. (SODA '12)*, 2012.

[28] C. Wulff-Nilsen. Approximate distance oracles with improved query time. arXiv abs/1202.2336., 2012.

Appendix A. Trapezoidal Approximation of Distance Summaries

Assume having a landmark vertex $\ell \in L$ and a departure-times subinterval $[t_s, t_f = t_s+\tau) \subseteq [0,T)$, for some small departure-time difference value, compared to the entire period T of departure times. We provide a crude approximation, which we call **TRAP** (the trapezoidal method), for creating distance functions from any landmark $\ell \in L$ towards each posible destination $v \in V$. It is mentioned that, contrary to the approximation method **BIS** proposed in [17], no assumption is made on the shapes of the unknown distance functions to approximate within $[t_s, t_f)$. In particular, no assumption is made on them being concave. **TRAP** will only exploit the fact that τ is indeed small, along with the *Bounded Travel-Time Slopes* Assumption (cf. Assumption 2.1). The approximation guarantee for each of these approximate distance functions actually varies, depending on the minimum (free-flow) travel-time from ℓ to each of the destinations. In particular, by Assumption 2.1, for any departure-time from ℓ, $t \in [t_s, t_f)$ and any destination vertex $v \in V$, the following inequalities hold:

$$\begin{array}{rl} & -\Lambda_{\min} \leq \dfrac{D[\ell,v](t) - D[\ell,v](t_s)}{t - t_s} \leq \Lambda_{\max} \\ \Rightarrow & -\Lambda_{\min}\cdot(t-t_s) + D[\ell,v](t_s) \leq D[\ell,v](t) \leq D[\ell,v](t_s) + \Lambda_{\max}\cdot(t-t_s) \\ \overset{/*\ \tau\geq t-t_s\ */}{\Rightarrow} & \boxed{-\Lambda_{\min}\cdot\tau + D[\ell,v](t_s) \leq D[\ell,v](t) \leq D[\ell,v](t_s) + \Lambda_{\max}\cdot\tau} \end{array}$$

$$\begin{array}{rl} & -\Lambda_{\min} \leq \dfrac{D[\ell,v](t_f) - D[\ell,v](t)}{t_f - t} \leq \Lambda_{\max} \\ \Rightarrow & -\Lambda_{\min}\cdot(t_f-t) \leq D[\ell,v](t_f) - D[\ell,v](t) \leq \Lambda_{\max}\cdot(t_f-t) \\ \overset{/*\ \tau\geq t_f-t\ */}{\Rightarrow} & \boxed{\Lambda_{\min}\cdot\tau + D[\ell,v](t_f) \geq D[\ell,v](t) \geq D[\ell,v](t_f) - \Lambda_{\max}\cdot\tau} \end{array}$$

Combining the two inequalities we get the following bounds: $\forall v \in V, \forall t \in [t_s, t_f)$,

$$\min\left\{\begin{array}{c} D[\ell,v](t_s) + \Lambda_{\max}\tau, \\ D[\ell,v](t_f) + \Lambda_{\min}\tau \end{array}\right\} \geq D[\ell,v](t) \geq \max\left\{\begin{array}{c} D[\ell,v](t_s) - \Lambda_{\min}\tau, \\ D[\ell,v](t_f) - \Lambda_{\max}\tau \end{array}\right\} \tag{1}$$

Exploiting the fact that each shortest-travel-time function from ℓ to any destination $v \in V$ and departure time from $[t_s, t_f)$ respects the above mentioned upper and lower bounds, one could use a simple continuous, pwl approximation of $D[\ell,v]$ within this interval, which is the minimum of four linear functions:

$$\forall t \in [t_s, t_f),\ \overline{D}[\ell,v](t) = \min\left\{\begin{array}{c} D[\ell,v](t_s) + \Lambda_{\max}\tau, \\ D[\ell,v](t_f) + \Lambda_{\min}\tau \\ \Lambda_{\max}t + D[\ell,v](t_s) - \Lambda_{\max}t_s, \\ -\Lambda_{\min}t + D[\ell,v](t_f) + \Lambda_{\min}t_f \end{array}\right\} \tag{2}$$

I.e., we consider the lines passing via the point $(t_s, D[\ell,v](t_s))$ with the maximum slope $\Lambda_{\max}$, until the upper bound in inequality 1 is reached, then follow a constant leg up to the point at which the line passing via $(t_s, D[\ell,v](t_s))$ with the minimum possible slope of $-\Lambda_{\min}$ is met. Analogously, we construct a lower-bounding approximation of $D[\ell,v]$ within $[t_s, t_f)$.

$$\forall t \in [t_s, t_f),\ \underline{D}[\ell,v](t) = \max\left\{\begin{array}{c} D[\ell,v](t_s) - \Lambda_{\min}\tau, \\ D[\ell,v](t_f) - \Lambda_{\max}\cdot\tau \\ \Lambda_{\max}t + D[\ell,v](t_f) - \Lambda_{\max}\cdot t_f, \\ -\Lambda_{\min}t + D[\ell,v](t_s) + \Lambda_{\min}t_s \end{array}\right\} \tag{3}$$

Figure 1 shows the (upper and lower) approximate distance summaries with respect to $D[\ell,v]$ within $[t_s, t_f)$.

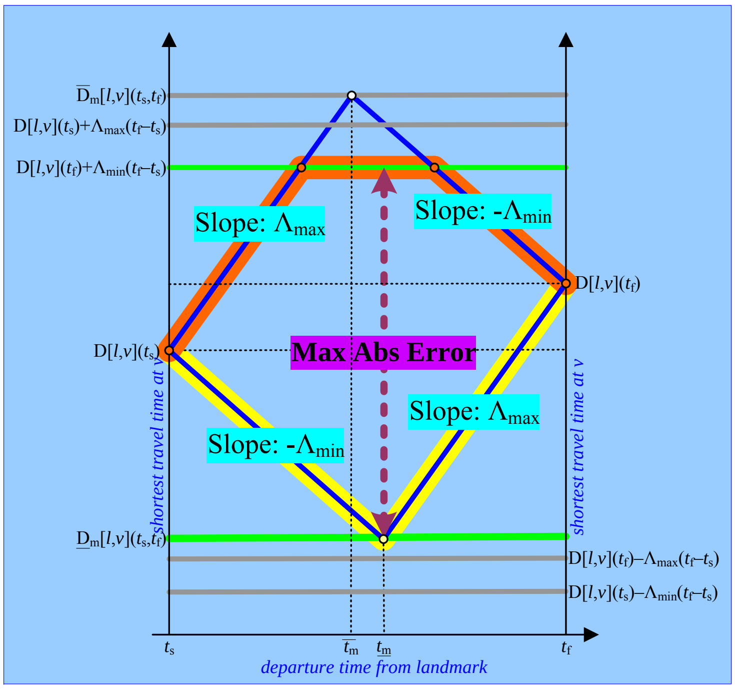

FIGURE 1. The upper-approximating function $\overline{D}[\ell,v]$ (thic orange, upper pwl line), and lower-approximating function $\underline{D}[\ell,v]$ (thic yellow, lower pwl line), of the unknown distance function $D[\ell,v]$ within the interval $[t_s,t_f)$.

Let $(\underline{t}_m,\underline{D}_m)$ be the intersection point of the two non-constant legs involved in the definition of $\underline{D}[\ell,v]$. Then it is easy to observe that:

$$\begin{aligned}\underline{t}_m &= \frac{D[\ell,v](t_s)-D[\ell,v](t_f)}{\Lambda_{\min}+\Lambda_{\max}}+\frac{\Lambda_{\min}t_s+\Lambda_{\max}t_f}{\Lambda_{\min}+\Lambda_{\max}}\\ \underline{D}_m &= \frac{\Lambda_{\max}D[\ell,v](t_s)+\Lambda_{\min}D[\ell,v](t_f)}{\Lambda_{\min}+\Lambda_{\max}}-\frac{\Lambda_{\min}\cdot\Lambda_{\max}}{\Lambda_{\min}+\Lambda_{\max}}\cdot(t_f-t_s)\end{aligned}$$

Similarly, let $(\overline{t}_m,\overline{D}_m)$ be the intersection point of the two non-constant legs involved in the definition of $\overline{D}[\ell,v]$. Then:

$$\begin{aligned}\overline{t}_m &= \frac{D[\ell,v](t_f)-D[\ell,v](t_s)}{\Lambda_{\min}+\Lambda_{\max}}+\frac{\Lambda_{\min}t_f+\Lambda_{\max}t_s}{\Lambda_{\min}+\Lambda_{\max}}\\ \overline{D}_m &= \frac{\Lambda_{\max}D[\ell,v](t_f)+\Lambda_{\min}D[\ell,v](t_s)}{\Lambda_{\min}+\Lambda_{\max}}+\frac{\Lambda_{\min}\Lambda_{\max}}{\Lambda_{\min}+\Lambda_{\max}}(t_f-t_s)\end{aligned}$$

The worst-case maximum (additive) error guaranteed for $\overline{D}[\ell,v]$ within $[t_s,t_f]$ is given by the following closed form:

$$\begin{aligned}MAE(t_s,t_f) &= \max_{t\in[t_s,t_f)}\left\{\overline{D}[\ell,v](t)-\underline{D}[\ell,v](t)\right\}\\ &= \max_{t\in\{\overline{t}_m,\underline{t}_m\}}\left\{\overline{D}[\ell,v](t)-\underline{D}[\ell,v](t)\right\}\end{aligned} \tag{4}$$

The following lemma correlates the value of the maximum absolute error with the minimum possible distance from ℓ, within $[t_s,t_f)$.

Lemma A.1. *For a given landmark vertex $\ell \in L$, any destination vertex $v \in V$ and a given departure-time subinterval $[t_s, t_f) \subseteq [0, T)$, the following hold:*

(1) $MAE[\ell, v](t_s, t_f) \leq \Lambda_{\max} \cdot (t_f - t_s)$.

(2) *If* at least one *of the following conditions hold, then the trapezoidal method guarantees that $\overline{D}[\ell, v]$ is a $(1+\varepsilon)-$approximation of $D[\ell, v]$ within $[t_s, t_f)$.*

  (i) $D[\ell, v](t_s) \geq \left(\Lambda_{\min} + \frac{\Lambda_{\max}}{\varepsilon}\right)(t_f - t_s)$.

  (ii) $D[\ell, v](t_f) \geq \left(\Lambda_{\max} + \frac{\Lambda_{\max}}{\varepsilon}\right)(t_f - t_s)$.

Proof. We start with the upper bound on the maximum absolute error:

$$\begin{aligned}
MAE[\ell, v](t_s, t_f) &\leq \overline{D}_m[\ell, v](t_s, t_f) - \underline{D}_m[\ell, v](t_s, t_f)\\
&= \frac{\Lambda_{\max} D[\ell, v](t_f) + \Lambda_{\min} D[\ell, v](t_s)}{\Lambda_{\min} + \Lambda_{\max}} + \frac{\Lambda_{\min}\Lambda_{\max}}{\Lambda_{\min} + \Lambda_{\max}}(t_f - t_s)\\
&- \frac{\Lambda_{\max} D[\ell, v](t_s) + \Lambda_{\min} D[\ell, v](t_f)}{\Lambda_{\min} + \Lambda_{\max}} + \frac{\Lambda_{\min} \cdot \Lambda_{\max}}{\Lambda_{\min} + \Lambda_{\max}} \cdot (t_f - t_s)\\
&= \frac{(\Lambda_{\max} - \Lambda_{\min}) \cdot (D[\ell, v](t_f) - D[\ell, v](t_s)) + 2\Lambda_{\min}\Lambda_{max}(t_f - t_s)}{\Lambda_{\min} + \Lambda_{\max}}\\
&= \frac{(\Lambda_{\max} - \Lambda_{\min}) \cdot (D[\ell, v](t_f) - D[\ell, v](t_s))/(t_f - t_s) + 2\Lambda_{\min}\Lambda_{max}}{\Lambda_{\min} + \Lambda_{\max}} \cdot (t_f - t_s)\\
&\overset{/*\ As.2.1\ */}{\leq} \frac{(\Lambda_{\max} - \Lambda_{\min}) \cdot \Lambda_{\max} + 2\Lambda_{\min}\Lambda_{max}}{\Lambda_{\min} + \Lambda_{\max}} \cdot (t_f - t_s) = \Lambda_{\max} \cdot (t_f - t_s)
\end{aligned}$$

Recall now about the upper-approximating function $\overline{D}[\ell, v]$ provided by the trapezoidal approximation within $[t_s, t_f)$ that, $\forall t \in [t_s, t_f)$,

$$\begin{aligned}
D[\ell, v](t) \leq \overline{D}[\ell, v](t) &\leq \underline{D}[\ell, v](t) + MAE[\ell, v](t_s, t_f)\\
&\leq \underline{D}[\ell, v](t) + \Lambda_{\max} \cdot (t_f - t_s)\\
&\leq D[\ell, v](t) \cdot \left(1 + \frac{\Lambda_{\max} \cdot (t_f - t_s)}{\underline{D}[\ell, v](t)}\right)
\end{aligned}$$

Our goal is to assure that this last upper bound of $\overline{D}[\ell, v](t)$ is in turn upper bounded by $(1+\varepsilon) \cdot D[\ell, v](t)$. Based on the expression of $\underline{D}[\ell, v](t)$, a *sufficient condition* for this to hold, is the following:

$$\begin{gathered}
D[\ell, v](t_s) \geq \left(\Lambda_{\min} + \tfrac{\Lambda_{\max}}{\varepsilon}\right)(t_f - t_s)\\
\vee\\
D[\ell, v](t_f) \geq \left(\Lambda_{\max} + \tfrac{\Lambda_{\max}}{\varepsilon}\right)(t_f - t_s)
\end{gathered}$$

This sufficient condition is independent of the actual departure time $t \in [t_s, t_f)$, and only depends on the travel-time values at the endpoints and also on the length of the departure-times interval. □

The following theorem distinguishes the "nearby" (for which there is no approximation guarantee) from the "faraway" destinations (for which $\overline{D}[\ell, v]$ is a $(1+\varepsilon)$-approximate distance summary) around ℓ, as a function of the chosen interval length $\tau = t_f - t_s$.

Theorem A.1. *For a given departure-times subinterval length $\tau > 0$ given as input to* **TRAP***, any landmark $\ell \in L$ and any destination $v \in V$, if it holds that:*

$$\min_{k \in \mathbb{N}: k\tau \in [0,T)} \{\ D[\ell, v](k\tau)\ \} \geq \left(1 + \frac{1}{\varepsilon}\right) \cdot \Lambda_{\max} \cdot \tau \tag{5}$$

then $\overline{D}[\ell, v]$ is a $(1+\varepsilon)$-approximation of $D[\ell, v]$ within $[0, T)$.

Proof. The entire period $[0, T)$ is split into $\frac{T}{\tau}$ consecutive intervals of length $\tau > 0$ each. We can easily deduce a sufficient condition for the trapezoidal approximation providing a $(1+\varepsilon)$-upper-approximating distance function for the entire period $[0, T)$. In particular, it suffices to assure that the claimed condition holds with respect to the travel times at its sampled values. Then, Lemma A.1 guarantees that the produced approximating functions within each consecutive interval is indeed a $(1+\varepsilon)$-approximation. $\square$