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 – 025

Distance Oracles for Time-Dependent Networks

Spyros Kontogiannis and Christos Zaroliagis

July 2013

DISTANCE ORACLES FOR TIME-DEPENDENT NETWORKS

SPYROS KONTOGIANNIS AND CHRISTOS ZAROLIAGIS

Abstract. We present the first approximate distance oracles for sparse directed networks with *time-dependent* arc-travel-times determined by continuous, piecewise linear, positive functions possessing the FIFO property. Our approach precomputes, in subquadratic time and space, $(1+\varepsilon)-$approximate distance summaries from selected vertices to all other vertices in the network, and provides two sublinear time query algorithms that deliver constant and $(1+\sigma)-$approximate shortest-travel-times, respectively, for arbitrary origin-destination pairs in the network. More specifically, our contributions are threefold: (i) we present a one-to-all polynomial-time algorithm for computing $(1+\varepsilon)-$approximations of the shortest-travel-travel time functions from a specific origin to all other vertices in the network, using space per approximate travel-time function that is *asymptotically optimal* and *independent* of the network size; (ii) we give a constant-approximation query algorithm of sublinear time complexity for arbitrary origin-destination pairs based on distance summaries precomputed from a subset of vertices (landmarks) to all other vertices using the one-to-all algorithm; (iii) we give a query algorithm that recursively uses the precomputed distance summaries and the constant-approximation algorithm, in order to achieve a $(1+\sigma)$-stretch factor, for any $\sigma > \varepsilon$.

Date: July 8, 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.

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, by querying the preprocessed data and/or applying *local* shortest path searches. A distance oracle is exact (resp. approximate) if the shortest paths discovered by the accompanying query algorithm are exact (resp. approximate). A bulk of important work (e.g., [18, 17, 14, 15, 19, 20, 1]) is devoted to constructing exact or approximate distance oracles in the case of *static* or *time-independent* (mostly) undirected general networks where the arc costs are fixed, providing trade-offs between oracle space and query time and, in the case of approximate oracles, also of the stretch (the max ratio, over all origin-destination pairs, between the path-length returned by the oracle and the true distance). For an overview of distance oracles for static networks, the reader is deferred to the survey article [16] and the references therein.

In many real-world applications, however, 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 $a = uv$ (modeling a road segment) depends on the real-time traffic conditions and thus on the departure time from its tail u. Consequently, the optimal origin-destination path may vary with the departure-time from the origin. Apart from the theoretical challenge, the time-dependent model is also much more realistic concerning the actual traffic data that the route planning vendors have to digest, in order to provide their customers with fast route plans. For example, TomTom's LiveTraffic service[1] provides real-time estimations collected by periodically sampling the status (availability, average speed, etc) of each road segment in a city, using the connected cars to the service as sampling devices. The crucial challenge is how to exploit all this temporal information in order to provide fast route plans that will vary with the departure-time from the origin. A way towards this direction is to construct continuous piecewise linear (pwl) functions (the interpolants of the averaged sampled points) that may then be considered as travel-time functions of the arcs.

Computing time-dependent shortest paths for a given triple (o, d, t_o) of an origin o, a destination d and a departure-time t_o from the origin, has been studied long ago (see e.g., [3, 9, 13]). It turns out that the shape of arc-travel-time functions and the waiting policy at vertices can considerably affect the solvability of the problem [13]. A crucial property is the FIFO property, according to which each arc in the network behaves as a FIFO queue, in the sense that the arrival-time at the head is a *non-decreasing* function of the departure-time from the tail. If a *forbidden-waiting-at-vertices* policy is adopted and the arc-travel-time functions do not possess the FIFO property, then the problem becomes **NP**−hard [13]. On the other hand, if arc-travel-time functions possess the FIFO property, then (regardless of the waiting policy at vertices) the problem can be solved in polynomial time by a variation of Dijkstra's algorithm (time-dependent Dijkstra – **TDD**), which scans arcs and settles vertices by computing the arc costs "on the fly" (i.e., the travel-time of arc $a = uv$ is computed when vertex u is settled). This has been first observed in [9], where the unrestricted waiting policy was (implicitly) assumed for vertices along with the non-FIFO property for arcs, which is equivalent to the FIFO case with forbidden-waiting-at-vertices since one can always wait at the tail of an arc to optimize the arrival-time at its head. The FIFO property may seem unreasonable in some application scenarios, e.g., for a traveler waiting at the dock of a train station and wondering whether to take the very next (slow) train towards destination, or wait for a later (faster) train that will take him much earlier to his destination.

Our motivation in this work stems from route planning in urban-traffic metropolitan road networks. In this case the FIFO property seems much more natural, since all cars are assumed

[1] `http://www.tomtom.com/livetraffic/`

to travel according to the same (average) speed along each road segment, and overtaking is not considered as an option when choosing a route plan. Indeed, the raw traffic data for arc-travel-time functions by TomTom for the city of Berlin are compliant with this assumption [10]. In any case, it is well known (see e.g., [11, 13]) that, when shortest-travel-times are well defined, a non-FIFO arc with *unrestricted-waiting-at-tail* policy is equivalent to a FIFO arc in which waiting at the tail is useless. Therefore, our focus in this work is on networks with FIFO arc-travel-time functions.

Until recently, most of the previous works 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 particular time from the origin, and neglected the computational complexity of providing succinct representations of the entire earliest-arrival-time (or equivalently, the shortest-travel-time) *functions*, for any departure-time from the origin. Such a representation, apart from allowing rapid answers to several queries for selected origin-destination pairs but for varying departure times, would also be valuable for the construction of *distance summaries* (a.k.a. route planning maps) from some central vertices (called landmarks or hubs) towards any other vertex in the network. This would in turn be a crucial ingredient for the construction of distance oracles to support real-time responses to arbitrary origin-destination-departure time queries in time-dependent networks.

The complexity issue of succinctly representing earliest-arrival-time functions was first raised by Dean [4, 6, 5], but was solved only recently by a seminal work [11] 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 $(K+1)\cdot n^{\Theta(\log n)}$, where n is the number of vertices and K is the total number of breakpoints (legs) of all the arc-travel-time functions. Among other results, a $(1+\varepsilon)$-approximation polynomial-time algorithm (PTAS) is provided in [11] that computes an approximate arrival-time function providing point-to-point travel-time values at most $1+\varepsilon$ times the true values. The importance of this result is that this function has a *succinct representation* requiring only $\mathcal{O}(K+1)$ breakpoints per origin-destination pair. Also, it is easy to verify that in this case 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).

To the best of our knowledge, the problem of providing distance oracles for time-dependent networks has not been investigated so far. Due to the hardness of providing succinct representations of exact shortest-travel-time functions between selected origin-destination pairs of vertices [11], the only realistic alternative is to use approximations of these functions for the distance summaries that will be preprocessed and stored by the oracle. Having a PTAS (as that in [11, Section 5.3]) for computing point-to-point distances, one could provide a trivial oracle with query-time complexity $Q \in \mathcal{O}(1)$, at the cost of an exceedingly high space-complexity $S \in \mathcal{O}\big((K^*+1)n^2\big)$, by storing the succinct representations of all the point-to-point $(1+\varepsilon)-$approximate shortest-travel-time functions. At the other extreme, one might only use the minimum possible space complexity $S \in \mathcal{O}(n+m+K)$ at the cost of suffering a query-time complexity $Q \in \mathcal{O}(m + n\log(n)[1+\log\log(1+K_{\max})])$ (e.g., only store the graph itself and the arc-travel-time functions, and respond to each query by running **TDD** in real-time). Here, $K_{\max}$ denotes the maximum number of breakpoints in any of the arc-travel-time functions[2]. The main challenge is, as in the time-independent case, to smoothly close this gap, i.e., to achieve a better (e.g., sublinear) query-time complexity, while consuming sub-quadratic space-complexity and enjoying a small (e.g., less than 2) approximation guarantee (stretch factor).

[2]The extra $\log\log(K_{\max})$ term in the Dijkstra-time is due to the fact that the arc-travel-times are *continuous* pwl functions of the departure-time from their tails, represented as collections of breakpoints. A predecessor-search structure would allow the evaluation of such a function to be achieved in time $\mathcal{O}(\log\log(K_{\max}))$.

Our goal in this work is to provide efficient distance oracles for time-dependent networks along with a solid *theoretical* analysis for the claimed complexity bounds. In particular, we present the first approximate distance oracle for sparse directed networks with *time-dependent* arc-travel-times, which achieves subquadratic time and space preprocessing complexity, sublinear query time, and a stretch factor close to 1. Note that: (i) even in static undirected networks, achieving a stretch factor below 2 using subquadratic space and sublinear query time, is only possible when $m \in o(n^2)$, and has been achieved, at least to our knowledge, only by two very recent works [15, 1]; (ii) there is important applied work [8, 2, 7, 12] to develop time-dependent shortest path *heuristics*, which however provide only empirical evidence of the used approaches.

At a high level, our approach resembles the typical ones used in *time-independent* and *undirected* graphs (e.g., [18, 15, 1]), where all distance summaries from selected vertices (landmarks) are precomputed and stored so as to support fast responses to arbitrary real-time queries by growing small distance balls around the origin and the destination vertices and then closing the gap between the prefix subpath (from the origin) and the suffix subpath (towards the destination). However, it is not at all straightforward how this generic approach can be extended to *time-dependent directed* graphs, since one is confronted with two highly non-trivial challenges: (i) handling directedness, and (ii) managing time-dependency, i.e., deciding the specific times to grow balls at the destination because we simply do ***not*** know the earliest-arrival-time at destination – actually, this is what the original query to the oracle asks for, since the earliest-arrival-time at destination minus the departure-time from the origin is exactly the requested shortest-travel-time.

More specifically, our contributions are threefold: (i) we present a *concurrent* (i.e., one-to-all) polynomial-time algorithm for computing $(1+\varepsilon)-$approximations of the shortest-travel-travel time functions from a specific origin to all possible destinations in the network, using space per approximate travel-time function that is *asymptotically optimal* and *independent* of the network size; (ii) we give a constant-approximation, sublinear-time query algorithm for arbitrary origin-destination pairs based on distance summaries precomputed from a predetermined set of landmarks to all other vertices using the one-to-all algorithm; (iii) we give a query algorithm that recursively uses the precomputed distance summaries and the constant-approximation algorithm, in order to achieve a $(1+\sigma)$-stretch factor, $\forall \sigma > \varepsilon$. Due to lack of space, several proofs are given in the appendix, as well as complexity trade-offs (Section F).

2. Ingredients and Oveview of Our Approach

Our input is provided by a directed graph $G = (V, A)$ with $|V| = n$ vertices and $|A| = m$ arcs, in which every arc $a = uv \in A$ is equipped with a periodic, continuous, piecewise-linear (pwl) *arc-travel-time* (or *arc-delay*) function $D[a] : \mathbb{R} \to \mathbb{R}_{>0}$, such that $\forall k \in \mathbb{Z}, \forall t_u \in [0, T]$, $D[a](k \cdot T + t_u) = D[a](t_u)$ is the arc-travel-time for traversing $a = uv$ when the departure-time from u is $k \cdot T + t_u$. Moreover, $D[a]$ is represented succinctly by a (constant) number K_a of breakpoints, or equivalently, as a collection of $K_a + 1$ affine legs that comprise this pwl function. Let $K = \sum_{a \in A} K_a$ be the total number of breakpoints to represent all the arc-delay functions in the network, and let $K_{\max} = \max_{a \in A} K_a$. Also, let K^* be the number of *concavity-spoiling breakpoints*, i.e., those breakpoints in which the arc-delay slopes increase. Clearly, $K^* < K$, and $K^* = 0$ for concave pwl functions. The space to represent the entire network is $\mathcal{O}(n + m + K)$. The *arc-arrival* function $Arr[a](t_u) = t_u + D[a](t_u)$ represents arrival-times at the head v of $a = uv$, depending on the departure-times t_u from its tail u.

Between any pair $(o, d) \in V \times V$ in G, $\mathcal{P}_{o,d}$ is the set of $od-$paths in G, and $\mathcal{P} = \cup_{(o,d)} \mathcal{P}_{o,d}$. For a path $p \in \mathcal{P}$, we denote by $p_{x \rightsquigarrow y}$ its subpath that starts from (the first appearance of) vertex x and ends at (the subsequent first appearance of) vertex y. For any pair of paths $p \in \mathcal{P}_{o,v}$ and $q \in \mathcal{P}_{v,d}$, $p \bullet q$ is the $od-$path produced as the concatenation of p

and q at v. For any path (represented as a sequence of arcs) $p = \langle a_1, a_2, \cdots, a_k \rangle \in \mathcal{P}_{o,d}$, the *path-arrival* function is the composition of the constituent arc-arrival functions: $\forall t_o \in [0,T],\ Arr[p](t_o) = Arr[a_k](Arr[a_{k-1}](\cdots(Arr[a_1](t_o))\cdots))$. The *path-travel-time* function is $D[p](t_o) = Arr[p](t_o) - t_o$. The *earliest-arrival-time* function from o to d is defined as follows: $\forall t_o \in [0,T], Arr[o,d](t_o) = \min_{p\in\mathcal{P}_{o,d}}\{Arr[p](t_o)\}$. The *shortest-travel-time* function is $D[o,d](t_o) = Arr[o,d](t_o) - t_o$. Finally, $SP[o,d](t_o)$ (resp. $ASP[o,d](t_o)$) is the set of all shortest (resp., with stretch-factor at most $(1+\varepsilon)$) $od-$paths for a given departure-time t_o.

2.1. **Some facts on the FIFO property.** We consider networks $G = (V, A, (D[a])_{a\in A})$ with continuous arc-delay functions, possessing the *FIFO* (or *non-overtaking*) *property*, which states that all arc-arrival-time functions in the network are non-decreasing:

$$\forall t_u, t'_u \in \mathbb{R}, \forall uv \in A, t_u > t'_u \Rightarrow Arr[uv](t_u) \geq Arr[uv](t'_u) \tag{1}$$

The FIFO property is *strict*, if the above inequality is strict. The following properties (Lemmata 2.1–2.3), are (perhaps) more-or-less known. We state them here (and provide their proofs in the Appendix) only for the sake of completeness.

Lemma 2.1. *If the network satisfies the (strict) FIFO property then any arc-delay function must have left and right derivatives with values at least (greater than) -1.*

It is easy to verify that the (assumed for arc-arrival-time functions) FIFO property also holds for arbitrary path-arrival-time functions and earliest-arrival-time functions.

Lemma 2.2. *If the network satisfies the FIFO property, then $\forall p = \langle a_1, \ldots, a_k \rangle \in \mathcal{P}_{o,d}$ 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. The (strict) monontonicity holds also for $Arr[o,d]$.*

The strict FIFO property implies also the crucial property of *subpath optimality*.

Lemma 2.3. *If the network possesses the strict FIFO property, then $\forall (u,v) \in V\times V$, $\forall t_u \in \mathbb{R}$ and any optimal path $p^* \in \arg\min_{p\in\mathcal{P}_{u,v}}\{Arr[p](t_u)\} = SP[u,v](t_u)$, it holds for every subpath $q^* \in \mathcal{P}_{x,y}$ of p^* that $q^* \in SP[x,y](Arr[p^*_{u\rightsquigarrow x}](t_u))$, i.e., q^* is a shortest path between its endpoints x, y for the earliest-departure-time from x, given t_u.*

Lemma 2.3 implies that both Dijkstra's label setting algorithm and the label-correcting algorithm also work in time-dependent strict FIFO networks, under the usual conventions for static instances (positivity of arc-delays and inexistence of negative-travel-time cycles).

2.2. **Towards a time-dependent distance oracle.** Our main assumption is that we are provided with a polynomial-time algorithm that provides, for an arbitrary pair $(o,d) \in V\times V$ of origin-destination vertices, a *succinctly represented* upper-bounding approximate distance function, $\Delta[o,d]$, i.e., a continuous pwl function with a *constant* number of breakpoints, such that $\forall t_o \in \mathbb{R},\ D[o,d](t_o) \leq \Delta[o,d](t_o) \leq (1+\varepsilon)\cdot D[o,d](t_o)$. Such a polynomial-time algorithm for *point-to-point* $(1+\varepsilon)-$approximate distance functions is, e.g., provided in [11, Section 5.3].

We demonstrate in this work an alternative approach to construct *one-to-all* approximate distance functions in the same time-complexity as for a single point-to-point construction, and with an asymptotically optimal space complexity. This will be crucially based on an exact expression of the maximum (worst-case) additive error when we "sandwitch" an unknown concave function between two properly chosen continuous pwl functions.

We make two assumptions on the kind of shortest-travel-time functions in the network, one that measures the steepness of the time-dependence, and one that quantifies the asymmetry of the distances in the network due to directedness. These assumptions seem quite natural in realistic time-dependent route planning instances, such as urban-traffic metropolitan road networks. The first assumption, called *Bounded Travel-Time Slopes*, asserts that the partial derivatives of the shortest-travel-time functions between any pair of origin-destination vertices are bounded in a given fixed interval $[\Lambda_{\min}, \Lambda_{\max}]$ for *constants* $\Lambda_{\max} \geq 0$ and $\Lambda_{\min} > -1$.

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

The lower bound -1 is justified by Lemmata 2.1 and 2.2 (strict FIFO property), while $\Lambda_{\max}$ represents the maximum possible rate of shortest-travel-time change in a road network, which only makes sense to be bounded. The second assumption, called *Bounded Opposite Trips*, 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 same path). This is quite natural in road networks, a fact that is indeed verified by real data, e.g., the traffic data provided by TomTom for the city of Berlin [10] convey that the maximum value of ζ is always less than 1.5.

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

Another assumption that we make, which nevertheless can be guaranteed w.l.o.g. by a simple, standard transformation that at most doubles the size of the network, is that the maximum out-degree is bounded by 2 (cf. Appendix, Section B for details).

2.3. **Overview of our approach.** We follow (at a high level) the typical approach adopted for the construction of approximate distance oracles in the static case. In particular, we start by selecting a subset $L \subset V$ of *landmarks*, i.e., vertices which will act as reference points for our distance summaries. For our oracle to work, several ways to choose L would be acceptable. Nevertheless, for the sake of the analysis we assume that this is done by deciding for each vertex randomly and independently with probability $\rho \in (0,1)$ whether it belongs to L. After having L fixed, our approach is deterministic. We start by constructing (concurrently, per landmark) and storing the *distance summaries*, i.e., all landmark-to-vertex $(1+\varepsilon)-$approximate travel-time functions, in subquadratic time and consuming subquadratic space which is indeed asymptotically optimal w.r.t. the required approximation guarantee (cf. Section 3). Then, we provide two approximation algorithms for arbitrary origin-destination-departure time queries (o,d,t_o). The first (**FCA**) is a simple *sublinear*-time constant-approximation algorithm (cf. Section 4). The second (**RQA**) is a recursive algorithm growing small **TDD** outgoing balls from vertices in the vicinity of the origin, until either a satisfactory approximation guarantee is achieved, or an upper bound r on the number of recursions (the *recursion budget*) has been exhausted. This algorithm responds with a $(1+\sigma)-$approximate travel-time to the query in *sublinear* time, for any constant $\sigma > \varepsilon$ (cf. Section 5). As it is customary in the distance oracle literature, the query times of our algorithms concern the determination of (upper bounds on) the shortest-travel-time from o to d. An actual path guaranteeing this bound can be reported in additional time that is linear in the number of its arcs.

3. Preprocessing Distance Summaries

The purpose of this section is to demonstrate how one can construct the necessary preprocessed information that will comprise the distance summaries of the oracle, i.e., all landmark-to-vertex shortest-travel-time functions. Our focus is on instances with *concave*, continuous, pwl arc-delay functions possessing the strict FIFO property. If there exist $K^* \geq 1$ *concavity-spoiling* breakpoints among the arc-delay functions, then we do the following: For each of them (which is a departure-time t_u from the tail u of an arc $a = uv \in A$) we run a variant of **TDD** with root (u, t_u) on the *reverse network* $(\overleftarrow{G} = (V, A, (\overleftarrow{D}[a])_{a \in A})$, where $\overleftarrow{D}[xy]$ describes the delay of arc $a = xy$, measured now as a function of the arrival-time t_y at the tail y. The algorithm proceeds backwards both along the connecting path(s) (from the destination towards the origin) and in time. As a result, we compute all *latest-departure-times* from

landmarks to catch the corresponding concavity-spoiling breakpoints, i.e., their projections to appropriate departure-times from the landmarks. Then, for each given orign-landmark we repeat the procedure described in the rest of this section independently for each of the (at most) K^*+1 consecutive subintervals of $[0,T]$ determined by these images. Within each subinterval all arc-travel-time functions are concave, as required in our analysis.

We must construct in polynomial time, for all $(\ell, v) \in L \times V$ succinctly represented upper-bounding $(1+\varepsilon)-$approximations $\Delta[\ell, v] : [0,T] \to \mathbb{R}_{>0}$ of the shortest-travel-time functions $D[\ell, v] : [0,T] \to \mathbb{R}_{>0}$. An algorithm providing such functions in a *point-to-point* fashion was proposed in [11, Section 5.3]. For each landmark vertex $\ell \in L$, it has to be executed n times so as to construct all the required landmark-to-vertex approximate functions. Additionally, the number of breakpoints returned by that algorithm may not be asymptotically optimal, given the required approximation guarantee: even for an affine (or almost affine) shortest-travel-time function with fixed slope(s) in $(1,2]$ it would require a number of points logarithmic in the ratio of max-to-min travel-time values from ℓ to v, despite the fact that we could avoid all intermediate breakpoints for the upper-approximating travel-time function.

Our solution is an improvement of the approach in [11] in two aspects: (i) it computes *concurrently* all the required approximate distance functions from a given landmark at a cost of a single point-to-point approximation of [11], and (ii) within every subinterval of consecutive images of concavity-spoiling breakpoints, our construction provides asymptotically optimal space per landmark, which is also independent of the network size per landmark-vertex pair, implying that the required preprocessing space per vertex is $\mathcal{O}(|L|)$.

In a nutshell, we construct two continuous pwl-approximations of the unknown shortest-travel-time function $D[\ell, v] : [0,T] \to \mathbb{R}_{>0}$, an upper-bounding approximate function $\overline{D}[\ell, v]$ and a lower-bounding approximate function $\underline{D}[\ell, v]$. $\overline{D}[\ell, v]$ plays the role of $\Delta[\ell, v]$. Our construction guarantees that the exact function is always "sandwitched" between these two approximations. In particular, for a given landmark $\ell \in L$, the algorithm proceeds as follows: for any subinterval $[t_s, t_f] \subseteq [0,T]$ we distinguish the destination vertices into *active*, i.e., the ones for which the desired value $\varepsilon \cdot D_{\min}[\ell, v](t_s, t_f)$ of the maximum absolute error within $[t_s, t_f]$ (whose closed form is provided in Lemma C.1) has not been reached yet, and the remaining *inactive* vertices[3]. Starting from $[t_s, t_f] = [0,T]$, as long as there is at least one active destination vertex for $[t_s, t_f]$, we bisect this time interval and recur on the subintervals $[t_s, (t_s+t_f)/2]$ and $[(t_s+t_f)/2, t_f]$. Prior to recurring to the two new subintervals, every destination vertex $v \in V$ that is active for $[t_s, t_f]$ stores the bisection point $(t_s+t_f)/2$ (and the corresponding sampled travel-time) in a list $LBP[\ell, v]$ of breakpoints for $\underline{D}[\ell, v]$. All inactive vertices just ignore this bisection point. The bisection procedure is terminated as soon as all vertices have become inactive. A linear scan of $LBP[\ell, v]$ allows also the construction of the list $UBP[\ell, v]$ of breakpoints for $\overline{D}[\ell, v]$. Lemma C.1 explains which is the extra breakpoint for $UBP[\ell, v]$, for each pair of consecutive breakpoints in $LBP[\ell, v]$. We denote by $L[\ell, v]$ ($U[\ell, v]$, resp.) the number of breakpoints we used for $\underline{D}[\ell, v]$ ($\overline{D}[\ell, v]$, resp.). By construction it holds that $U[\ell, v] \leq 2 \cdot L[\ell, v]$ (see more details in Section C).

The following theorem summarizes the space-complexity and time-complexity of our bisection method for providing concurrently one-to-all shortest-travel-time approximate travel-time functions in time-dependent instances with concave[4], continuous, pwl arc-travel-time functions, with bounded shortest-travel-time slopes. Let $U^*[\ell, v]$ denote the minimum possible number of breakpoints for any $(1+\varepsilon)-$approximating function of $D[\ell, v]$.

[3] $D_{\min}[\ell, v](t_s, t_f) = \min_{t \in [t_s, t_f]}\{D[\ell, v](t)\} = \min\{D[\ell, v](t_s), D[\ell, v](t_f)\}$, due to concavity of $D[\ell, v]$ in $[t_s, t_f]$. Similarly, $D_{\max}[\ell, v](t_s, t_f) = \max_{t \in [t_s, t_f]}\{D[\ell, v](t)\}$.

[4] If concavity is not assured, then these numbers must be multiplied by K^*+1, since the proposed approximation procedure has to be repeated per subinterval of consecutive images of concavity-spoiling breakpoints.

Theorem 3.1. *For a given $\ell \in L$ and any $v \in V$, our preprocessing algorithm computes an asymptotically optimal and independent of the network size number of breakpoints*

$$U[\ell, v] \leq 4U^*[\ell, v] \leq 4\log_{1+\varepsilon}\left(\frac{D_{\max}[\ell,v](0,T)}{D_{\min}[\ell,v](0,T)}\right) \in \mathcal{O}\Big(\frac{1}{\varepsilon}\log\left(\frac{D_{\max}[\ell,v](0,T)}{D_{\min}[\ell,v](0,T)}\right)\Big)$$

where $D_{\max}[\ell, v](0, T)$ and $D_{\min}[\ell, v](0, T)$ are the maximum and minimum shortest-travel-time values within $[0, T]$. Moreover, the total number TDP of time-dependent (forward) shortest-path-tree probes for the construction of all the lists of breakpoints for $(\underline{D}[\ell, v])_{v\in V}$, is:

$$TDP \in \mathcal{O}\Big(\max_{v\in V}\left\{\log\left(\frac{T\cdot(\Lambda_{\max}+1)}{\varepsilon D_{\min}[\ell,v](0,T)}\right)\right\}\cdot\frac{1}{\varepsilon}\cdot\max_{v\in V}\left\{\log\left(\frac{D_{\max}[\ell,v](0,T)}{D_{\min}[\ell,v](0,T)}\right)\right\}\Big)\ .$$

Let $U = \max_{\ell,v} U[\ell, v]$. Since the expected number of landmarks is $\mathbb{E}\{|L|\} = \rho n$, it is easy to deduce the required time and space complexity of our entire preprocessing.

Corollary 3.1. *The expected complexities for space $\mathcal{S}$ and time $\mathcal{P}$ of the preprocessing phase are: $\mathbb{E}\{\mathcal{S}\} \in \mathcal{O}\big(\rho n^2(K^*+1)U\big)$ and $\mathbb{E}\{\mathcal{P}\} \in \mathcal{O}\big(\rho n^2\log(n)\cdot\log\log(K_{\max})\cdot(K^*+1)TDP\big)$.*

U and TDP are independent of n (Theorem 3.1), so we consider them as constants. If all arc-travel-time functions are concave, i.e., $K^* = 0$, then we achieve subquadratic preprocessing space and time $\forall\rho \in \mathcal{O}(n^{-\alpha})$, where $0 < \alpha < 1$. Real data (e.g., TomTom's traffic data for the city of Berlin [10]) demonstrate that: (i) only a small fraction of the arc-travel-time functions exhibit non-constant behavior; (ii) for the vast majority of these non-constant-delay arcs, their functions are either concave, or can be very tightly approximated by a typical *concave* bell-shaped pwl function. It is only a tiny subset of critical arcs (e.g., bottleneck road segments) for which it would be meaningful to consider non-concave behavior. Therefore, $K^* \in o(n)$ is the typical case, and indeed, assuming e.g., $K^* \in \mathcal{O}(\text{polylog}(n))$, we can easily fine-tune ρ and the parameters σ, r (cf. Section 5) so as to achieve subquadratic preprocessing space and time. In particular, for $K^* \in \mathcal{O}(\log(n))$ and $K_{\max} \in \mathcal{O}(1)$, $\forall\gamma > \frac{1}{2}$, $\mathbb{E}\{\mathcal{S}\} \in \mathcal{O}\big(n^{2-\varepsilon/(\gamma\psi)}\log(n)\big)$ and $\mathbb{E}\{\mathcal{P}\} \in \mathcal{O}\big(n^{2-\varepsilon/(\gamma\psi)}\log^2(n)\big)$, where $\psi = \psi(\zeta, \Lambda_{\max})$ is a constant that will be specified in Lemma 4.1. More details are provided in Section F.

4. Constant-Approximation Query Algorithm

Based on the preprocessed distance summaries, the next step towards a distance oracle is to provide a fast query algorithm providing constant-approximations to the actual shortest-travel-time values of arbitrary queries $(o, d, t_o) \in V \times V \times [0, T]$. In this section, we propose such a query algorithm, called *Forward Constant Approximation* (**FCA**), which grows an outgoing ball $B_o \equiv B[o](t_o) = \{x \in V : D[o, x](t_o) < D[o, \ell_o](t_o)\}$ around (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 reached. We call $R_o = D[o, \ell_o](t_o)$ the *radius* of B_o. Then, the algorithm either returns the exact travel-time value, or it returns the approximate travel-time value via ℓ_o. The pseudocode is in Figure 5, while Figure 1 gives an overview of the whole idea.

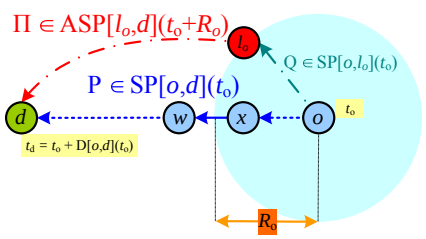


Figure 1. The rationale of **FCA**. The dashed (blue) path is a shortest $od-$path for query (o, d, t_o). The dashed-dotted (green and red) path is the via-landmark $od-$path indicated by the algorithm, if the destination vertex is out of the origin's **TDD** ball.

4.1. Correctness.

First we demonstrate that **FCA** indeed returns $od-$paths whose travel-times are constant approximations to the shortest travel-times.

Lemma 4.1. *For any time-dependent network* $(V, A, (D[a])_{a\in A})$ *and any query* $(o, d, t_o) \in V \times V \times [0, T]$, **FCA** *returns either a path* $P \in SP[o, d](t_o)$ *with the shortest travel-time* $D[o, d](t_o)$*, or it returns a via-landmark* $od-$*path* $Q \bullet \Pi$ *such that* $Q \in SP[o, \ell_o](t_o)$ *and* $\Pi \in ASP[\ell_o, d](t_o + R_o)$ *(the approximate path provided by the oracle), for which the following holds:* $D[o, d](t_o) \le R_o + \Delta[\ell_o, d](t_o + R_o) \le (1+\varepsilon)\cdot D[o, d](t_o) + \psi \cdot R_o \le (1+\varepsilon+\psi)\cdot D[o, d](t_o)$, *where* $\psi = 1 + \Lambda_{\max}(1 + \varepsilon)(1 + 2\zeta + \Lambda_{\max}\zeta) + (1 + \varepsilon)\zeta$.

Observe that **FCA** is a generalization of the $3-$approximation algorithm in [1] for symmetric (i.e., $\zeta = 1$) and time-independent (i.e., $\Lambda_{\min} = \Lambda_{\max} = 0$) network instances, the only difference being that the stored distance summaries we consider are $(1 + \varepsilon)-$approximations of the actual shortest-travel-times. Of course, our algorithm smoothly departs (through the parameters ζ and $\Lambda_{\max}$) towards both asymmetry (directedness) and time-dependence.

4.2. **Complexity.** The main cost of **FCA** is to grow the out-ball from (o, t_o) by running **TDD** until either d (the destination) or ℓ_o (the first landmark) is settled. Therefore, what really matters is the order of (number of vertices in) B_o, since the maximum out-degree is 2. L is chosen randomly by selecting each vertex v to become a landmark independently of other vertices, with probability $\rho \in (0, 1)$. Clearly $\mathbb{E}\{|B_o|\} = \frac{1}{\rho}$, and moreover (as a geometrically distributed random variable), $\forall k \ge 1$, $\mathbb{P}\{|B_o| > k\} = (1 - \rho)^k \le e^{-\rho k}$. By setting $k = \frac{\ln(1/\rho)}{\rho}$ we conclude that: $\mathbb{P}\left\{|B_o| > \frac{\ln(1/\rho)}{\rho}\right\} \le \rho$. Since the maximum out-degree is 2, **TDD** will relax at most $2k$ arcs. Hence, for the query-time complexity $\mathcal{Q}_{FCA}$ of **FCA** we conclude that $\mathbb{E}\{\mathcal{Q}_{FCA}\} \in \mathcal{O}\left(\frac{\ln(1/\rho)}{\rho}\right)$, and $\mathbb{P}\left\{\mathcal{Q}_{FCA} \in \Omega\left(\frac{\ln^2(1/\rho)}{\rho}\right)\right\} \in \mathcal{O}(\rho)$.

5. $(1 + \sigma)-$Approximate Query Algorithm

In this section we present the *Recursive Query Algorithm* (**RQA**) that improves the approximation guarantee of the chosen $od-$path provided by **FCA** by exploiting carefully a number (that we call the *recursion budget*) of recursive accesses to the preprocessed information, each of which produces (via a call to **FCA**) one more candidate $od-$path sol_i. The crux of our approach is the following: We assure that, unless the required approximation guarantee has already been reached by a candidate solution, the recursion budget must be exhausted and the sequence of radii of the consecutive balls that we grow recursively is lower-bounded by a geometrically increasing sequence. Then, we demonstrate that this sequence can only have a *constant* number of elements, since the sum of all these radii provides also a lower bound on the shortest-travel-time that we seek.

A somewhat similar approach was proposed recently for undirected and time-independent sparse networks [1], in which a number of recursively growing balls (up to the recursion budget) is used in the vicinities of both the origin and the destination nodes, before eventually applying a constant-approximation algorithm to close the gap, so as to achieve improved approximation guarantees. In order for their recursive argument to work, the method in [1] needs to assure that all the radii of the recursively growing balls are at least as large as the last ball-radius to be considered by the final recursive call in which the constant-approximation algorithm is indeed used. This simply provides a good lower bound on the actual origin-destination distance, along with an acceptable upper bound on the performance of the approximation algorithm that closes the gap between the discovered prefix-subpath (from the origin) and suffix-subpath (towards the destination). Growing balls both around the origin and the destination node is actually crucial for their recursive argument to assure the required lower bound on the unknown distance.

In our case the network is both directed and time-dependent, and thus one is confronted with two main challenges: (i) overcoming the asymmetry due to directedness, and (ii) managing carefully time-dependency. In particular, the latter challenge is hard, since we do not

know the earliest-arrival time t_d corresponding to the query (o, d, t_o) so as to start growing balls from the side of the destination vertex d (actually, t_d is the sought answer to the query!).

Due to this ignorance of the exact departure time at the destination, it is difficult (if at all possible) to grow incoming balls in the vicinity of the destination node. Hence, our only choice is to build a recursive argument that grows outgoing balls in the vicinity of the origin, since we only know the requested departure-time from it. This is exactly what we do: as long as we have not discovered the destination node within the explored area around the origin, and there is still some remaining recursion budget, we "guess" (by exhaustively searching for it) the next node w_k along the (unknown to our algorithm) shortest $od-$path. We then grow a new out-ball from the new center $(w_k, t_k = t_o + D[o, w_k](t_o))$, until we reach 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. Observe that sol_k uses a longer (optimal) prefix-subpath P_k which is then completed with a shorter approximate suffix-subpath $Q_k \bullet \Pi_k$ provided by the preprocessed information kept in the oracle. The pseudocode in Figure 6 provides the details of our approach. Figure 2 provides an overview of **RQA**'s execution.

5.1. **Correctness & Quality.** The correctness of **RQA** implies that the algorithm always returns some $od-$path. This is true due to the fact that it either discovers the destination

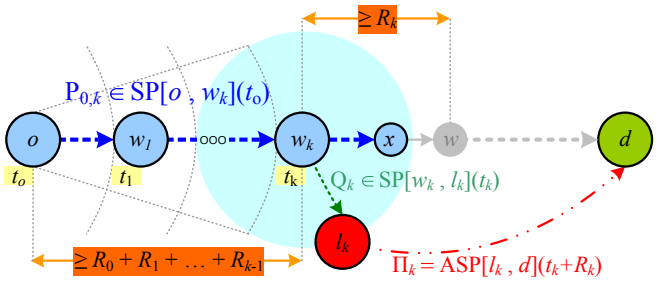


FIGURE 2. Overview of the execution of **RQA**.

node d as it explores new nodes in the vicinity of the origin node o, or it returns the shortest of the approximate $od-$paths $sol_0, \ldots, sol_r$ via one of the closest landmark nodes $\ell_o, \ldots, \ell_r$ to "guessed" nodes $w_0 = o, w_1, \ldots, w_r$ along the shortest $od-$path $P \in SP[o, d](t_o)$, where r is the recursion budget. Based on the fact that the preprocessed time-dependent distance oracle provides not only approximate distances, but also actual paths from landmarks to vertices in the graph guaranteeing these distances, it is clear that **RQA** always returns a connecting path whose actual delay does not exceed the alleged upper bound on the actual distance.

Our next task is to study the quality of the provided approximation guarantees. Recall that we are looking for an $(1+\sigma)-$approximation. Then, let $\delta > 0$ be a parameter such that $\sigma = \varepsilon + \delta$ and recall the definition of ψ from Lemma 4.1. The next lemma shows that the sequence of ball radii grown by the recursive calls of **RQA** is lower-bounded by a geometrically increasing sequence.

Lemma 5.1. *Let $D[o, d](t_o) = t_d - t_o$, and suppose that* **RQA** *does not discover d or any landmark $w_k \in SP[o, d](t_o) \cap L$ in the explored area around o. Then, the entire recursion budget r will be consumed and in each round $k \in \{0, 1, \ldots, r\}$ of recursively constructed balls we have:* $R_k > \left(1 + \frac{\varepsilon}{\psi}\right)^k \cdot \frac{\delta}{\psi} \cdot (t_d - t_o) \vee \exists i \in \{0, 1, \ldots, k\} : D[sol_i](t_o) \leq (1+\varepsilon+\delta) \cdot D[o, d](t_o)$.

The following theorem proves that **RQA** provides $(1+\sigma)-$approximations to the actual distances that should be returned in response to an arbitrary query $(o, d, t_o) \in V \times V \times [0, T]$.

Theorem 5.1. *Assume that* $r = \left\lceil \frac{\ln\left(1+\frac{\varepsilon}{\delta}\right)}{\ln\left(1+\frac{\varepsilon}{\psi}\right)} \right\rceil - 1$, *where* ψ *is the constant in* **FCA***'s approximation guarantee,* $\varepsilon \geq 0$ *is the approximation-guarantee of the preprocessed information kept by the oracle, and* $\delta > 0$ *is a parameter such that* $\sigma = \varepsilon + \delta$. *Then, for arbitrary query* $(o, d, t_o) \in V \times V \times [0, T]$, **RQA** *returns an* $od-$*path* Π *such that* $D[\Pi](t_o) \leq (1+\sigma) \cdot D[o, d](t_o)$.

Proof. If none of the returned via-landmark solutions is a $(1 + \varepsilon + \delta)-$approximation, then:

$$\begin{aligned} t_d - t_o &\geq R_0 + R_1 + \ldots + R_r \overset{/* \text{ Lemma 5.1 } */}{>} \frac{\delta}{\psi} \cdot (t_d - t_o) \cdot \sum_{i=0}^{r} \left(1 + \frac{\varepsilon}{\psi}\right)^i \\ &= \frac{\delta}{\psi} \cdot (t_d - t_o) \cdot \frac{\left(1 + \frac{\varepsilon}{\psi}\right)^{r+1} - 1}{1 + \frac{\varepsilon}{\psi} - 1} = \frac{\delta}{\varepsilon} \cdot (t_d - t_o) \cdot \left[\left(1 + \frac{\varepsilon}{\psi}\right)^{r+1} - 1\right] \\ \Rightarrow \frac{\varepsilon}{\delta} &> \left(1 + \frac{\varepsilon}{\psi}\right)^{r+1} - 1 \Rightarrow r < \frac{\ln\left(1 + \frac{\varepsilon}{\delta}\right)}{\ln\left(1 + \frac{\varepsilon}{\psi}\right)} - 1 \end{aligned}$$

Since $r \leq \frac{\psi/\delta}{1-\varepsilon/\psi} - 1 \in \mathcal{O}\left(\frac{\psi}{\delta}\right)$, we have reached a contradiction[5]. For this value of the recursion budget **RQA** either discovers the destination node, or at least a landmark node that also belongs to $SP[o, d](t_o)$, or else it returns a via-landmark path that is a $(1 + \varepsilon + \delta)-$approximation of the required shortest $od-$path. □

Note that for time-independent, undirected-graphs (for which $\Lambda_{\min} = \Lambda_{\max} = 0$ and $\zeta = 1$) it holds that $\psi = 2 + \varepsilon$. If we additionally equip our oracle with *exact* rather than $(1 + \varepsilon)-$approximate landmark-to-vertex distances (i.e., $\varepsilon = 0$), then in order to achieve $\sigma = \delta = \frac{2}{t+1}$ for some positive integer t, our recursion budget r is *upper bounded* by $\frac{\psi}{\delta} - 1 = t$. This is exactly the amount of recursion required by the approach in [1] to assure the same approximation guarantee. That is, at its one extreme ($\Lambda_{\min} = \Lambda_{\max} = 0,\ \zeta = 1,\ \psi = 2$) our approach matches the bounds in [1] for the same class of graphs, without the need to grow balls from both the origin and destination vertices. Moreover, our approach allows for a smooth transition from static and undirected-graphs to directed-graphs with FIFO arc-delay functions. Of course, the required recursion budget now depends not only on the targeted approximation guarantee, but also on the degree of asymmetry (the value of $\zeta \geq 1$) and the steepness of the shortest-travel-time functions (the value of $\Lambda_{\max}$) for the time-dependent case.

5.2. **Complexity.** It only remains to determine the query-time complexity $\mathcal{Q}_{RQA}$ of **RQA**. The following theorem provides this remaining query-time complexity.

Theorem 5.2. *For sparse networks (i.e., having* $\mu = |A|/|V| \in \mathcal{O}(1)$*) the expected running time of* **RQA** *is* $\mathbb{E}\{\mathcal{Q}_{RQA}\} \in \mathcal{O}\left(\left(\frac{1}{\rho}\right)^r \cdot \ln\left(\frac{1}{\rho}\right)\right)$. *Moreover, it holds that:*

$$\mathbb{P}\left\{\mathcal{Q}_{RQA} \in \mathcal{O}\left(\left(\frac{\ln(n)}{\rho}\right)^r \cdot \left[\ln\ln(n) + \ln\left(\frac{1}{\rho}\right)\right]\right)\right\} \in 1 - \mathcal{O}\left(\frac{1}{n}\right).$$

Continuing the discussion in the paragraph following Corollary 3.1, we can fine-tune the parameters σ, r so as to achieve, along with subquadratic space and preprocessing time, a sublinear query-time complexity $\mathbb{E}\{\mathcal{Q}_{RQA}\} \in \mathcal{O}\left(n^{1/(2\gamma)} \log(n)\right), \forall \gamma > \frac{1}{2}$. More details (and examples) are provided in Section F.

[5]The inequality $r \leq \frac{\psi/\delta}{1-\varepsilon/\psi} - 1$ holds due to the following bound: $\forall z \geq -\frac{1}{2}, z - z^2 \leq \ln(1 + z) \leq z$.

6. Conclusions and Future Work

In this work we provide, at least to our knowledge, the first approximate distance oracle for time-dependent FIFO networks with continuous, pwl arc-delay functions possessing the FIFO property. Our analysis exploits two quite natural assumptions that allow us to smoothly move from static and undirected instances to time-dependent and directed ones. Our construction works well for sparse graphs. It would be interesting to consider approximate distance oracles for special network classes that often appear in practice, such as planar graphs.

References

[1] 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.

[2] Gernot Veit Batz, Robert Geisberger, Sabine Neubauer, and Peter Sanders. Time-Dependent Contraction Hierarchies and Approximation. In Paola Festa, editor, *Proceedings of the 9th International Symposium on Experimental Algorithms (SEA'10)*, volume 6049 of *Lecture Notes in Computer Science*, pages 166–177. Springer, May 2010.

[3] 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.

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

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

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

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

[8] Daniel Delling and Dorothea Wagner. Time-Dependent Route Planning. In Ravindra K. Ahuja, Rolf H. Möhring, and Christos Zaroliagis, editors, *Robust and Online Large-Scale Optimization*, volume 5868 of *Lecture Notes in Computer Science*, pages 207–230. Springer, 2009.

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

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

[11] 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.

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

[13] 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.

[14] 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.

[15] 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.

[16] Christian Sommer. Shortest-path queries in static networks, 2012. submitted.

[17] 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.

[18] Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. of ACM*, 52:124, 2005.

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

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

Appendix A. Missing proofs of Section 2

The properties stated in Lemmata 2.1, 2.2, and 2.3 are (perhaps) more-or-less known. We provide their proofs here for the sake of completeness.

A.1. **Proof of Lemma 2.1.** Observe that, by the FIFO property: $\forall a \in A, \forall t_u \in \mathbb{R}, \forall \delta > 0$,

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

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

□

A.2. **Proof of Lemma 2.2.** 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 a recursive definition of path-arrival-time functions. For a path (represented as a sequence of arcs) $p = \langle a_1, \cdots a_k \rangle \in \mathcal{P}_{o,d}$ and $\forall 1 \leq i \leq j \leq k$, let $p_{i,j}$ be the subpath of p starting with the i^{th} arc a_i and ending with the j^{th} arc a_j in order. Then:

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

The composition of non-decreasing (increasing) functions is well known to also be non-decreasing (increasing). As for the earliest-arrival-time function $Arr[o,d] = \min_{p \in \mathcal{P}_{o,d}} \{Arr[p]\}$, as a minimization operation over non-decreasing (increasing) functions, it is also itself a non-decreasing (increasing) function of departure-time from o.

□

A.3. **Proof of Lemma 2.3.** Let $t^*_x = Arr[p^*_{u \rightsquigarrow x}](t_u)$. For sake of contradiction, assume that $\exists q \in \mathcal{P}_{x,y} : D[q](t^*_x) < D[q^*](t^*_x)$. Then, $p = p^*_{u \rightsquigarrow x} \bullet q \bullet p^*_{y \rightsquigarrow v}$ 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 of the suffix-subpath $p^*_{y \rightsquigarrow v}$).

□

Appendix B. Boundedness of Out-Degrees

Our focus in this work is on *sparse* networks, in which $\mu = \frac{m}{n} \in \mathcal{O}(1)$. Moreover, we need to guarantee also that the out-degree of every node is bounded by a constant (say, at most 2). This will be crucial for the query-time complexity of our approximation algorithms for arbitrary origin-destination pairs. Indeed, it is not hard to assure that the maximum out-degree is 2, by using an equivalent network of at most double size (number of vertices and number of arcs). This is achieved by substituting every vertex of the original graph (V, A) with out-degree greater than 2 with a complete binary tree whose leaf-edges are indeed the outgoing edges from v in (V, A), and each internal level consists of a maximal number

of nodes with two children from the lower level, until a 1-node level is reached. This root node inherits all the incoming arcs from v in the original graph. All the newly inserted arcs (except for the original arcs outgoing from v) get zero delay functions. Figure 3 demonstrates an example of such a substitution. For each node $v \in G : d^+(v) > 2$, the node substitution

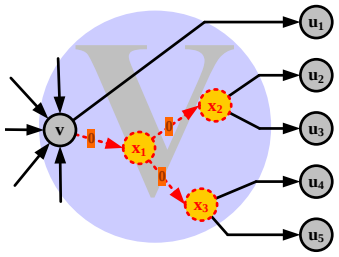


FIGURE 3. The node substitution operation for a vertex $v \in V$ with out-degree $d^+_G(v) = 5$. The operation assures an out-degree at most 2 for all the newly inserted vertices in place of v in the graph. The new graph elements (nodes and arcs) are indicated by dashed (red) lines. The solid (black) arcs and vertices are the ones pre-existing in the graph.

operation is executed in time $\mathcal{O}(d^+(v))$ and introduces $d^+(v)-1$ new nodes and $d^+(v)-2$ new arcs (of zero delays). Therefore, in time $\mathcal{O}(|A|)$ we can assure out-degree at most 2 and the same time-dependent travel-time characteristics, by at most doubling the size of the graph ($\sum_{v\in V:d^+(v)>2}(d^+(v)-1) < |A|$ new nodes and $\sum_{v\in V:d^+(v)>2}(d^+(v)-2) < |A|$ new arcs).

APPENDIX C. DETAILED PRESENTATION OF THE PREPROCESSING PHASE

We start by providing a closed form for the maximum absolute error between the upper-approximating and the lower-approximating functions of a generic shortest-travel-time function D within a time interval $[c, d] \subseteq [0, T]$[6].

Lemma C.1. *For $[c, d] \subseteq [0, T]$, fix an unknown but amenable to polynomial-time sampling continuous (not necessarily pwl) concave function $D : [c, d] \to \mathbb{R}_{>0}$, with right and left derivatives at the endpoints denoted by $\Lambda^+(c), \Lambda^-(d)$. Assume that $\Lambda^+(c) > \Lambda^-(d)$ and $L = d - c > 0$. Let $m = \frac{D(d)-D(c)+c\cdot\Lambda^+(c)-d\cdot\Lambda^-(d)}{\Lambda^+(c)-\Lambda^-(d)}$ and $\overline{D}_m = \Lambda^+(c)\cdot(m-c)+D(c)$. Consider the affine function $\underline{D}$ passing via the points $(c, D(c))$, $(d, D(d))$. Consider also the pwl function $\overline{D}$ with three breakpoints $(c, D(c))$, $(m, \overline{D}_m)$, $(d, \underline{D}(d))$. Then $\forall t \in [c, d], \underline{D}(t) \le D(t) \le \overline{D}(t)$ and the* maximum absolute error *between $\underline{D}$ and $\overline{D}$ in $[c, d]$ is at most:*

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

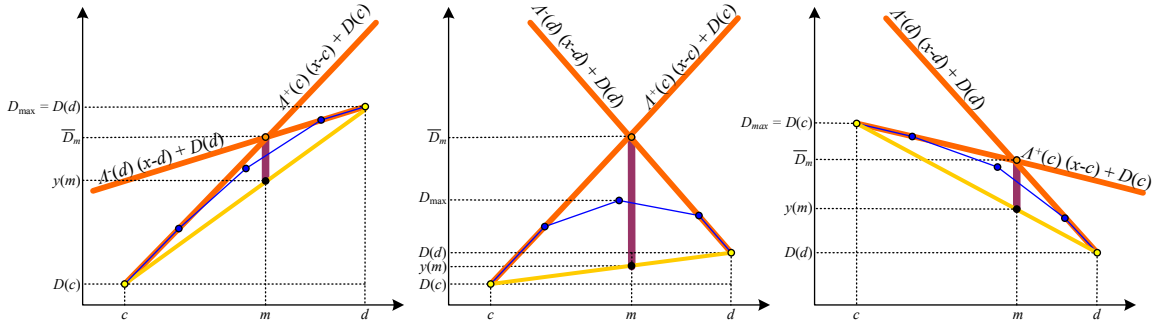
Proof. Consider the affine functions (see also figure 4):

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

The point $\left(m = \frac{D(d)-D(c)+c\cdot\Lambda^+(c)-d\cdot\Lambda^-(d)}{\Lambda^+(c)-\Lambda^-(d)}, \overline{D}_m = y_c(m) = y_d(m)\right)$ is the intersection point of the lines $y_c(x)$ and $y_d(x)$. As an upper-bounding (pwl) function of D in $[c, d]$ we consider $\overline{D}(t) = \min\{y_c(t), y_d(t)\}$, whereas the lower-bounding (affine) function of D is $\underline{D}(t) = y(t)$.

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

[6]In this section the variables c, d denote departure times for the generic shortest-travel-time function D, which we wish to approximate. In particular, d has nothing to do with a destination vertex, but is only a departure-time from the origin of D, just like c.



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

FIGURE 4. Three distinct cases for upper-bounding the absolute error between two consecutive interpolation points. The maximum absolute error (MAE) considered is shown by the vertical (purple) line segment at point m of the time axis.

the triangle $\{(c, D(c)), (m, \overline{D}_m), (d, D(d))\}$. The maximum possible distance (additive error) of $\overline{D}$ from $\underline{D}$ is:

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

This value 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 4). The calculations are identical for the three distinct cases shown in figure 4. 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:

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

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 now proceed by presenting our polynomial-time algorithm which provides asymptotically space-optimal succinct representations of one-to-all $(1+\varepsilon)-$approximating funnctions $\overline{D}[\ell, \star] = (\overline{D}[\ell, v])_{v \in V}$ of $D[\ell, \star] = (D[\ell, v])_{v \in V}$, for a given landmark $\ell \in L$ and all destinations $v \in V$. Recall our Assumption 2.1 concerning the boundedness of the shortest-travel-time function slopes. Given this assumption, we are able to construct a generalization of the bisection method proposed in [11] for point-to-point approximations of distance functions, to the case of a single-origin ℓ and all reachable destinations from it. Our method (we call it **SO_BISECT**) computes *concurrently* (i.e., within the same bisection) all the required breakpoints to describe the (pwl) lower-approximating functions $\underline{\mathbf{D}}[\ell, \star] = (\underline{D}[\ell, v])_{v \in V}$, and finally, via a linear scan of it, the upper-approximating functions $\overline{\mathbf{D}}[\ell, \star] = \left(\overline{D}[\ell, v]\right)_{v \in V}$. This is

possible because the bisection is done on the (common for all travel-time functions to approximate) axis of departure-times from the origin ℓ. The other crucial observation is that for each destination vertex $v \in V$ we keep as breakpoints of $\underline{D}[\ell, v]$ only 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) approximation error between the lower-approximating and the upper-approximating distance function per destination vertex (cf. Lemma C.1). Moreover, all the delays to be sampled at a particular bisection point $t_\ell \in [0, T]$ are calculated by a single time-dependent shortest-path-tree (e.g., **TDD**) execution for (ℓ, t_ℓ).

The algorithm proceeds as follows: For any subinterval $[t_s, t_f] \subseteq [0, T]$ we split the destination vertices into *active*, i.e., the ones for which the desired value $\varepsilon \cdot D_{\min}[\ell, v](t_s, t_f)$ of the maximum absolute error within $[t_s, t_f]$ has not been reached yet, and the remaining *inactive* vertices[7]. As long as there is at least one active destination vertex for the interval $[t_s, t_f]$, we split it in the middle, and recur our bisection on the subintervals $\left[t_s, \frac{t_s+t_f}{2}\right]$ $\left[\frac{t_s+t_f}{2}, t_f\right]$. The bisection point is stored as a breakpoints only of the lists $LBP[\ell, v]$ of still active vertices v for $[t_s, t_f]$. The inactive vertices of $[t_s, t_f]$ just skip this point. The bisection procedure is terminated as soon as all vertices have become inactive. Apart from the list $LBP[\ell, v]$ of breakpoints for $\underline{D}[\ell, v]$, a linear scan of this list allows also the construction of the list $UBP[\ell, v]$ of the required breakpoints for $\overline{D}[\ell, v]$ (cf. Lemma C.1).

The time-complexity and space-complexity of the preprocessing phase are provided in Theorem 3.1, whose proof is given in the following subsection. In what follows, $L[\ell, v] = |LBP[\ell, v]|$ is the number of breakpoints for $\underline{D}[\ell, v]$, $U[\ell, v] = |UBP[\ell, v]|$ is the number of breakpoints for $\overline{D}[\ell, v]$ and, finally, $U^*[\ell, v]$ is the minimum number of breakpoints of any $(1+\varepsilon)-$upper approximating function of $D[\ell, v]$, within the time-interval $[0, T]$.

C.1. **Proof of Theorem 3.1.** The time complexity of **SO_BISECT** will be asymptotically equal to that of the worst-case point-to-point bisection from ℓ to some destination vertex v. In particular, **SO_BISECT** *concurrently* computes the new breakpoints for the lower-bounding approximate distance functions of all the active nodes, within the same **TDD**-run. This is because the departure-time axis is common for all the shortest-travel-time functions from the common origin ℓ. Moreover, due to being able to (exactly) calculate the worst-case maximum absolute error per destination vertex in each interval of the bisection, the algorithm is able to deactivate (and thus, stop producing breakpoints for) those vertices which have already reached the required approximation guarantee. The already deactivated node will remain so until the end of the algorithm. Nevertheless, the bisection continues as long as there exists at least one active destination vertex.

Number of breakpoints produced by **SO_BISECT**. The initial departure-times interval to bisect is $[0, T]$. Assume that we are currently at an initial interval $[t_s, t_f] \subseteq [0, T]$, of length $t_f - t_s$. A new bisection halves this subinterval and creates new breakpoints at $\frac{t_s+t_f}{2}$, one for each vertex that remains active. Thus, at the $k-$th level of the recursion tree all the subintervals have length $L(k) = T/2^k$. Since for any shortest-travel-time function and any subinterval $[t_s, t_f]$ of departure-times from ℓ it holds that $0 \leq \Lambda^+[\ell, v](t_s) - \Lambda^-[\ell, v](t_f) \leq \Lambda_{\max} + 1$ (cf. Assumption 2.1), the absolute error between $\underline{D}[\ell, v]$ and $\overline{D}[\ell, v]$ in this interval is at most $\frac{L(k)\cdot(\Lambda_{\max}+1)}{4} \leq \frac{T\cdot(\Lambda_{\max}+1)}{2^{k+2}}$. This implies that the bisection will certainly stop at a level $k_{\max}$ of the recursion tree at which for any subinterval $[t_s, t_f] \subseteq [0, T]$ and any destination vertex

[7] $D_{\min}[\ell, v](t_s, t_f) = \min_{t\in[t_s,t_f]}\{D[\ell, v](t)\} = \min\{D[\ell, v](t_s), D[\ell, v](t_f)\}$, due to concavity of $D[\ell, v]$ in $[t_s, t_f]$. Similarly, $D_{\max}[\ell, v](t_s, t_f) = \max_{t\in[t_s,t_f]}\{D[\ell, v](t)\}$.

$v \in V$ the following holds:

$$MAE[\ell, v](t_s, t_f) \leq \frac{T \cdot (\Lambda_{\max} + 1)}{2^{k_{\max}+2}} \leq \varepsilon D_{\min}[\ell, v](t_s, t_f) \leq \varepsilon D_{\min}[\ell, v](0, T)$$

From this we conclude that setting

$$k_{\max} = \max_{v \in V} \left\{ \left\lceil \log_2 \left(\frac{T \cdot (\Lambda_{\max} + 1)}{\varepsilon D_{\min}[\ell, v](0, T)} \right) \right\rceil \right\} - 2$$

is a safe upper bound on the depth of the recursion tree.

On the other hand, observe that the parents of the leaves in the recursion tree correspond to subintervals $[t_s, t_f] \subset [0, T]$ for which the absolute error of at least one vertex $v \in V$ is greater than $\varepsilon D_{\min}[\ell, v](t_s, t_f)$, 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_BISECT** produces at most twice as many interpolation points (to determine the lower-approximating vector function $\underline{\mathbf{D}}[\ell, \star]$) required for any $(1+\varepsilon)$-upper-approximation of $\mathbf{D}[\ell, \star]$. But, as suggested in [11], by taking as breakpoints the (at most two) intersections of the horizontal lines $(1+\varepsilon)^j \cdot D_{\min}[\ell, v](0, T)$ with the (unkown) function $D[\ell, v]$, one would guarantee the following upper bound on the minimum number of breakpoints for any $(1+\varepsilon)-$approximation of $D[\ell, v]$ within $[0, T]$:

$$U^*[\ell, v] \leq \left\lceil \log_{1+\varepsilon} \left(\frac{D_{\max}[\ell, v](0, T)}{D_{\min}[\ell, v](0, T)} \right) \right\rceil - 1$$

Therefore, $\forall v \in V$ it holds that:

$$L[\ell, v] \leq 2 \cdot \log_{1+\varepsilon} \left(\frac{D_{\max}[\ell, v](0, T)}{D_{\min}[\ell, v](0, T)} \right)$$

The produced list $UBP[\ell, v]$ of breakpoints for the $(1+\varepsilon)$-upper-approximation $\overline{D}[\ell, v]$ produced by **SO_BISECT** uses at most one extra breakpoint for each pair of consecutive breakpoints in $LBP[\ell, v]$ for $\underline{D}[\ell, v]$. Therefore, $\forall v \in V$:

$$U[\ell, v] \leq 4 \cdot \log_{1+\varepsilon} \left(\frac{D_{\max}[\ell, v](0, T)}{D_{\min}[\ell, v](0, T)} \right) \in \mathcal{O}\left(\frac{1}{\varepsilon} \log \left(\frac{D_{\max}[\ell, v](0, T)}{D_{\min}[\ell, v](0, T)} \right) \right). \tag{3}$$

We now proceed with the time-complexity of **SO_BISECT**. We shall count the number TDP of time-dependent shortest-path (TDSP) probes (e.g., **TDD** runs) to compute all the candidate breakpoints during the entire bisection. The crucial observation is that the bisection is applied on the common departure-time axis: In each recursive call from $[c, d]$, all the new breakpoints at the new departure-time $t_{mid} = \frac{c+d}{2}$, 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 $LBP[\ell, v](0, T)$ requires a number of (forward) TDSP-probes that is upper bounded by the path-length leading to the consideration of this point for bisection, in the recursion tree. Any root-to-node path in this tree has length at most $k_{\max}$, therefore each breakpoint of $LBP[\ell, v](0, T)$ requires at most $k_{\max}$ TDSP-probes, to be computed. In overall, the total number TDP of TDSP probes required to construct $LBP[\ell, \star](0, T)$, is upper-bounded by the following inequality:

$$\begin{aligned} TDP &\leq k_{\max} \cdot \max_{v \in V} |LBP[\ell, v](0, T)| \\ &\in \mathcal{O}\left(\max_{v \in V} \left\{ \left\lceil \log \left(\frac{T \cdot (\Lambda_{\max} + 1)}{\varepsilon D_{\min}[\ell, v](0, T)} \right) \right\rceil \right\} \cdot \frac{1}{\varepsilon} \cdot \max_{v \in V} \left\{ \log \left(\frac{D_{\max}[\ell, v](0, T)}{D_{\min}[\ell, v](0, T)} \right) \right\} \right) \end{aligned} \tag{4}$$

forward TDSP-probes. We can also construct $UBP[\ell, \star](0, T)$ from $LBP[\ell, \star](0, T)$ without any execution of a TDSP-probe, by just sweeping once for every vertex $v \in V$ $LBP[\ell, v](0, T)$

and adding all the intermediate breakpoints required. The time-complexity of this procedure is $\mathcal{O}(|LBP[\ell,\star](0,T)|)$ and this is clearly dominated by the time-complexity (number of TDSP-probes) for constructing $LBP[\ell,\star](0,T)$ itself.

□

Appendix D. Missing Proofs of Section 4

D.1. **Proof of Lemma 4.1.** In case that $d \in B_o$, there is nothing to prove since **FCA** returns the exact distance. So, assume that $d \notin B_o$, implying that $D[o,d](t_o) \geq R_o$. As for the returned distance value $R_o + \Delta[\ell_o,d](t_o+R_o)$, it is not hard to see that this is indeed an overestimation of the actual distance $D[o,d](t_o)$. This is because $\Delta[\ell_o,d](t_o+R_o)$ is an overestimation (implying also a connecting $\ell_o d-$path) of $D[\ell_o,d](t_o+R_o)$, and of course $R_o = D[o,\ell_o](t_o)$ corresponds to a (shortest) $o\ell_o-$path that was discovered by the algorithm on the fly. Therefore, $R_o + \Delta[\ell_o,d](t_o+R_o)$ is an overestimation of an actual $od-$path for departure time t_o, and cannot be less than $D[o,d](t_o)$. We now prove that it is not arbitrarily larger than this shortest distance:

$$
\begin{aligned}
R_o + \Delta[\ell_o,d](t_o+R_o) &\leq R_o + (1+\varepsilon)D[\ell_o,d](t_o+R_o) \\
&\overset{\text{/* triangle */}}{\leq} R_o + (1+\varepsilon)[D[\ell_o,o](t_o+R_o) + D[o,d](t_o+R_o+D[\ell_o,o](t_o+R_o))] \\
&\overset{\text{/* Assumption 2.1 */}}{\leq} R_o + (1+\varepsilon)[(1+\Lambda_{\max})D[\ell_o,o](t_o+R_o) + \Lambda_{\max}R_o + D[o,d](t_o)] \\
&\overset{\text{/* Assumption 2.2 */}}{\leq} R_o + (1+\varepsilon)[(1+\Lambda_{\max})\zeta D[o,\ell_o](t_o+R_o) + \Lambda_{\max}R_o + D[o,d](t_o)] \\
&\overset{\text{/* Assumption 2.1 */}}{\leq} R_o + (1+\varepsilon)[(1+\Lambda_{\max})\zeta(R_o+\Lambda_{\max}R_o) + \Lambda_{\max}R_o + D[o,d](t_o)] \\
&= \left[1+(1+\varepsilon)(1+\Lambda_{\max})^2\zeta + (1+\varepsilon)\Lambda_{\max}\right]\cdot R_0 + (1+\varepsilon)\cdot D[o,d](t_o) \\
&= \underbrace{\left[1+\Lambda_{\max}(1+\varepsilon)(1+2\zeta+\Lambda_{\max}\zeta)+(1+\varepsilon)\zeta\right]}_{=\psi}\cdot R_0 + (1+\varepsilon)\cdot D[o,d](t_o) \\
&= (1+\varepsilon)\cdot D[o,d](t_o) + \psi\cdot R_o
\end{aligned}
$$

□

Appendix E. Missing Proofs of Section 5

E.1. **Proof of Lemma 5.1.** So long as none of the discovered nodes $o = w_0, w_1, \ldots, w_k$ is a landmark node and the recursion budget has not been consumed yet, **RQA** continues guessing new nodes of $P \in SP[o,d](t_o)$. If any of these nodes (say, w_k) is a landmark node, the $(1+\varepsilon)-$approximate solution $P_{0,k} \bullet \Pi[w_k,d](t_k)$ is returned and we are done. Otherwise, **RQA** will certainly have to consume the entire recursion budget.

Moreover, for any $k \in \{0,1,\ldots,r\}$, if $\exists i \in \{0,1,\ldots,k\} : D[sol_i](t_o) \leq (1+\varepsilon+\delta)\cdot D[o,d](t_o)$ then there is nothing to prove from that point on. The required disjunction trivially holds for all rounds $k, k+1, \ldots, r$. We therefore consider the case in which up to round $k-1$ of the recursion no good approximation has been discovered, and we shall prove inductively that either sol_k is a $(1+\varepsilon+\delta)-$ approximation, or else $R_k > \left(1+\frac{\varepsilon}{\psi}\right)^k \cdot \frac{\delta}{\psi}\cdot(t_d - t_o)$.

BASIS: Recall that **FCA** is used to provide the suffix-subpath of the returned solution sol_0, whose prefix (from o to ℓ_o) is indeed a shortest path. Therefore:

$$
\begin{aligned}
D[sol_0](t_o) &\leq R_0 + \Delta[\ell_0,d](t_o+R_0) \\
&\overset{\text{/* Lemma 4.1 */}}{\leq} (1+\varepsilon)\cdot D[o,d](t_o) + \psi\cdot R_0 = \left(1+\varepsilon+\frac{\psi R_0}{t_d-t_o}\right)\cdot(t_d-t_o)
\end{aligned}
$$

Clearly, either $\frac{\psi R_0}{t_d - t_o} \leq \delta \Leftrightarrow R_0 \leq \frac{\delta}{\psi} \cdot (t_d - t_o)$, which then implies that we already have a $(1+\varepsilon+\delta)-$approximate solution, or else it holds that $R_0 > \frac{\delta}{\psi} \cdot (t_d - t_o)$.
HYPOTHESIS: We assume inductively that $\forall 0 \leq i \leq k$, no $(1+\varepsilon+\delta)-$approximate solution has been discovered up to round k, and thus holds that $R_i > \left(1+\frac{\delta}{\psi}\right)^i \cdot \frac{\delta}{\psi} \cdot (t_d - t_o)$.
STEP: We prove that for the $(k+1)-$st recursive call, either the new via-landmark solution $sol_{k+1} = P_{0,k+1} \bullet Q_{k+1} \bullet \Pi_{k+1}$ (see figures 6 and 2 for explanation of the notation) is a $(1+\varepsilon+\delta)-$approximate solution, or else $R_{k+1} > \left(1+\frac{\delta}{\psi}\right)^{k+1} \cdot \frac{\delta}{\psi} \cdot (t_d - t_o)$. For the travel-time along this path we have:

$$
\begin{aligned}
& D[sol_{k+1}](t_o) \\
& \qquad \leq && t_{k+1} - t_o + R_{k+1} + \Delta[\ell_{k+1}, d](t_{k+1} + R_{k+1}) \\
& \qquad \text{/* Lemma 4.1 */} \\
& \qquad \leq && t_{k+1} - t_o + (1+\varepsilon) \cdot D[w_{k+1}, d](t_{k+1}) + \psi \cdot R_{k+1} \\
& \qquad \text{/* } w_{k+1} \in SP[o,d](t_o) \text{ */} \\
& \qquad = && t_{k+1} - t_o + (1+\varepsilon) \cdot (t_d - t_{k+1}) + \psi \cdot R_{k+1} \\
& \qquad = && (1+\varepsilon) \cdot (t_d - t_o) - \varepsilon \cdot (t_{k+1} - t_o) + \psi \cdot R_{k+1} \\
& \qquad \text{/* } t_{k+1} - t_o \geq R_0 + \ldots + R_k \text{ */} \\
& \qquad \leq && (1+\varepsilon) \cdot (t_d - t_o) - \varepsilon \cdot (R_0 + \ldots + R_k) + \psi \cdot R_{k+1} \\
& \qquad \text{/* Ind. Hypothesis */} \\
& \qquad < && (1+\varepsilon) \cdot (t_d - t_o) - \varepsilon \cdot \sum_{i=0}^{k} \left(1+\frac{\varepsilon}{\psi}\right)^i \cdot \frac{\delta}{\psi} \cdot (t_d - t_o) + \psi \cdot R_{k+1} \\
& \qquad = && \left(1 + \varepsilon - \frac{\varepsilon\delta}{\psi} \cdot \sum_{i=0}^{k} \left(1+\frac{\varepsilon}{\psi}\right)^i + \frac{\psi \cdot R_{k+1}}{t_d - t_o}\right) \cdot (t_d - t_o) \\
& \qquad = && \left(1 + \varepsilon - \delta \cdot \left[\left(1+\frac{\varepsilon}{\psi}\right)^{k+1} - 1\right] + \frac{\psi \cdot R_{k+1}}{t_d - t_o}\right) \cdot (t_d - t_o)
\end{aligned}
$$

Once more, it is clear that either $D[sol_{k+1}](t_o) \leq (1+\varepsilon+\delta) \cdot D[o,d](t_o)$, or else it must hold that $R_{k+1} > \left(1+\frac{\varepsilon}{\psi}\right)^{k+1} \cdot \frac{\delta}{\psi} \cdot (t_d - t_o)$ as required.

□

E.2. **Proof of Theorem 5.2.** Recall that for any vertex $w \in V$ and any departure-time $t_w \in [0,T]$, the size of the outgoing **TDD**$-$ball $B_w = B[w](t_w)$ centered at (w, t_w) until the first landmark vertex is settled, behaves as a geometric random variable with success probability $\rho \in (0,1)$. Thus, $\mathbb{E}\{|B_w|\} = \frac{1}{\rho}$ and $\forall \beta \in \mathbb{N}$, $\mathbb{P}\{|B_w| > \beta\} \leq \exp(-\rho \cdot \beta)$. By applying the trivial union bound, one can then deduce that:

$$\forall W \subseteq V, \mathbb{P}\{\exists w \in W : |B_w| > \beta\} \leq |W| \exp(-\rho\beta) = \exp\left(-\rho\beta + \ln(|W|)\right)$$

Assume now that we somehow could guess an upper bound β^* on the number of vertices in every ball grown by an execution of **RQA**. Then, since the out-degree is upper bounded by 2, we know that the boundary ∂B of each ball B will have size $|\partial B| \leq 2|B|$. This in turn implies that the branching tree that is grown in order to implement the "guessing" of step 7.1 in **RQA** (cf. figure 6) via exhaustive search, would be bounded by a complete $(2\beta^*)-$ary tree of depth $r-1$. For each node in this branching tree we have to grow a new **TDD**$-$ball outgoing from the corresponding center, until a landmark vertex is settled. The size of this ball will once more be upper-bounded by β^*. Due to the fact that the out-degree is bounded by 2, at most $2\beta^*$ arcs will be relaxed. Therefore, the running time of growing each ball is $\mathcal{O}(\beta^* \ln(\beta^*))$. At the end of each **TDD** execution, we query the oracle for the

distance of the newly discovered landmark to the destination node. This will have a cost of $\mathcal{O}(\log\log((K^*+1)\cdot U))$, where U is the maximum number of required breakpoints between two concavity-spoiling arc-delay breakpoints in the network, since all the breakpoints of the corresponding shortest-travel-time function are assumed to be organized in a predecessor-search data structure. The overall query-time complexity of **RQA** would thus be bounded as follows:

$$\begin{aligned}\mathcal{Q}_{RQA} &\leq \frac{(2\beta^*)^r-1}{2\beta^*-1}\cdot\mathcal{O}(\beta^*\ln(\beta^*)+\log\log((K^*+1)\cdot U))\\ &\in \mathcal{O}\big((\beta^*)^r\ln(\beta^*)+\beta^{r-1}\log\log((K^*+1)\cdot U)\big)\end{aligned}$$

Assuming that $\log\log((K^*+1)\cdot U)\in\mathcal{O}(\beta^*\log(\beta^*))$, we have that $\mathcal{Q}_{RQA}\in\mathcal{O}((\beta^*)^r\ln(\beta^*))$. If we are only interested on the expected running time of the algorithm, then each ball has expected size $\mathcal{O}\left(\frac{1}{\rho}\right)$ and thus $\mathbb{E}\left\{\mathcal{Q}_{RQA}\right\}\in\mathcal{O}\left(\left(\frac{1}{\rho}\right)^r\ln\left(\frac{1}{\rho}\right)\right)$.

In general, if we set $\beta^*=\frac{r\ln(n)}{\rho}$, then we know that **RQA** will grow $|W|\in\mathcal{O}\left(\left(\frac{r\ln(n)}{\rho}\right)^{r-1}\right)$ balls, and therefore:

$$\begin{aligned}\mathbb{P}\left\{\forall w\in W,|B_w|\leq\frac{r\ln(n)}{\rho}\right\} &\geq 1-\exp\left(-\rho\frac{r\ln(n)}{\rho}+(r-1)\cdot[\ln\ln(n)+\ln(1/\rho)]\right)\\ &\in 1-\mathcal{O}\left(\frac{1}{|V|}\right)\end{aligned}$$

Thus, we conclude that:

$$\mathbb{P}\left\{\mathcal{Q}_{RQA}\in\mathcal{O}\left(\left(\frac{\ln(n)}{\rho}\right)^r\cdot\left[\ln\ln(n)+\ln\left(\frac{1}{\rho}\right)\right]\right)\right\}\in 1-\mathcal{O}\left(\frac{1}{n}\right)\ .$$

□

Appendix F. Preprocessing-Space-Query Time Tradeoffs of Our Oracle

In this section we demonstrate the tradeoffs achieved by our oracle among the expected preprocessing time $\mathbb{E}\left\{\mathcal{P}\right\}$, the required space $\mathbb{E}\left\{\mathcal{S}\right\}$, and the query-time complexity $\mathbb{E}\left\{\mathcal{Q}_{RQA}\right\}$ respectively. Let $U=\max_{\ell,v}U[\ell,v]=\max_{\ell,v}|UBP[\ell,v]|$, and recall that TDP is the maximum number of forward TDSP probes required for computing the preprocessed data for a particular landmark vertex (cf. Theorem 3.1).

Table 1 summarizes some characteristic examples of the trade-offs. The first two rows present the two extremes of our oracle: the first with all-to-all preprocessing and the second with no preprocessing at all. From the third row on we provide demonstrating examples of the main results in this paper.

Since U and TDP are independent of the network size n, we consider them as constants from the fourth row on. Similarly, $K_{\max}$ which denotes the maximum number of breakpoints of an arc-travel-time function in the network (which is part of the input) is also considered to be independent of the network size. But even if it was the case that $K_{\max}\in\Theta(K)$, this would only have a doubly-logarithmic multiplicative effect in the preprocessing-time and query-time complexities, which is indeed acceptable.

Regarding the number K^* of *concavity-spoiling* breakpoints of arc-travel-time functions, if all arc-travel-time functions are concave, i.e., $K^*=0$, then we clearly achieve subquadratic preprocessing space and time for any $\rho\in\mathcal{O}(n^{-\alpha})$, where $0<\alpha<1$. Real data (e.g., TomTom's traffic data for the city of Berlin [10]) demonstrate that: (i) only a small fraction of the arc-travel-time functions exhibit non-constant behavior; (ii) for the vast majority of these non-constant-delay arcs, the arc-travel-time functions are either concave, or can be very tightly approximated by a typical *concave* bell-shaped pwl function. It is only a tiny subset

of critical arcs (e.g., bottleneck-segments in a large city) for which it would be indeed meaningful to consider also non-concave behavior. Therefore, $K^* \in \mathrm{o}(n)$ is the typical case, and indeed, assuming e.g., $K^* \in \mathcal{O}(\mathrm{polylog}(n))$, we can easily fine-tune ρ and the parameters σ, r (cf. Section 5) so as to achieve subquadratic preprocessing space and time. In particular, for $K^* \in \mathcal{O}(\log(n))$ and $\forall \gamma > \frac{1}{2}$, $\mathbb{E}\{\mathcal{S}\} \in \mathcal{O}\big(n^{2-\varepsilon/(\gamma\psi)} \log(n)\big)$ and $\mathbb{E}\{\mathcal{P}\} \in \mathcal{O}\big(n^{2-\varepsilon/(\gamma\psi)} \log^2(n)\big)$, along with sublinear query-time complexity $\mathbb{E}\{\mathcal{Q}_{RQA}\} \in \mathcal{O}\big(n^{1/(2\gamma)} \log(n)\big), \forall \gamma > \frac{1}{2}$. Recall that $\psi = \psi(\zeta, \Lambda_{\max})$ is a constant that is specified in Lemma 4.1.

The constant γ is only used to demonstrate the trade-off between subquadratic preprocessing space and time on one hand, and the sublinear query time on the other hand. Case 1 allows only a slight deterioration of the approximation guarantee provided by the preprocessed data (from $1+\varepsilon$ to $1+2\varepsilon$). Case 2 considers a more severe deterioration in the accuracy of the query algorithm (from $1+\varepsilon$ to $1+10\varepsilon$).

what is preprocessed	$\mathbb{E}\{\mathcal{S}\}$	$\mathbb{E}\{\mathcal{P}\}$	$\mathbb{E}\{\mathcal{Q}_{RQA}\}$
All-To-All	$\mathcal{O}\big((K^*+1)n^2U\big)$	$\mathcal{O}\left(\begin{array}{c} n^2\log(n) \\ \cdot \log\log(K_{\max}) \\ \cdot (K^*+1)TDP \end{array}\right)$	$\mathcal{O}(\log\log(U))$
Nothing	$\mathcal{O}(n+m+K)$	$\mathcal{O}(1)$	$\mathcal{O}\left(\begin{array}{c} n\log(n)\cdot \\ \log\log(K_{\max}) \end{array}\right)$
Landmarks-To-All	$\mathcal{O}\big(\rho n^2(K^*+1)U\big)$	$\mathcal{O}\left(\begin{array}{c} \rho n^2\log(n) \\ \cdot \log\log(K_{\max}) \\ \cdot (K^*+1)TDP \end{array}\right)$	$\mathcal{O}\left(\begin{array}{c} \left(\frac{1}{\rho}\right)^r \cdot \log\left(\frac{1}{\rho}\right) \\ \cdot \log\log(K_{\max}) \end{array}\right)$
$K_{\max} \in \mathcal{O}(1)$ $U, TDP \in \mathcal{O}(1)$ $K^* \in \mathcal{O}(\log(n))$	$\mathcal{O}\big(\rho n^2\log(n)\big)$	$\mathcal{O}\big(\rho n^2\log^2(n)\big)$	$\mathcal{O}\Big(\left(\frac{1}{\rho}\right)^r \log\left(\frac{1}{\rho}\right)\Big)$
1. $\gamma > 1+\frac{\varepsilon}{\psi}$ $\rho = n^{-\varepsilon/(\gamma\psi)}$ $\sigma = 2\varepsilon$ $\Rightarrow r \le \frac{\psi}{\varepsilon}+1$	$\mathcal{O}\big(n^{2-\varepsilon/(\gamma\psi)}\log(n)\big)$	$\mathcal{O}\big(n^{2-\varepsilon/(\gamma\psi)}\log^2(n)\big)$	$\mathcal{O}\big(n^{1/\gamma+\varepsilon/(\gamma\psi)}\log(n)\big)$
2. $\gamma > \frac{1}{2}$ $\rho = n^{-\varepsilon/(\gamma\psi)}$ $\sigma = 10\varepsilon$ $\Rightarrow r \le \frac{\psi}{2\varepsilon}$	$\mathcal{O}\big(n^{2-\varepsilon/(\gamma\psi)}\log(n)\big)$	$\mathcal{O}\big(n^{2-\varepsilon/(\gamma\psi)}\log^2(n)\big)$	$\mathcal{O}\big(n^{1/(2\gamma)}\log(n)\big)$

TABLE 1. List of trade-off examples of the proposed distance oracle, for a given approximation guarantee $1+\varepsilon$ for preprocessed data. For sake of comparison, also the two extremes of the oracle are provided, with all-to-all preprocessing, and no preprocessing at all.

Appendix G. Pseudocodes of proposed algorithms

In this section we provide the pseudocodes for the query-response algorithms **FCA** and **RQA** proposed in this work.

FCA(o, d, t_o)

1. **if** $o \in L$ **then return** $(\Delta[o,d](t_o))$ /* $(1+\varepsilon)-$approximate answer */
2. $B_o = B[o](t_o) := \{\ x \in V : D[o,x](t_o) < \min_{\ell \in L}\{D[o,\ell](t_o)\}\ \}$ /* **TDD**−run till ℓ_o is settled */
3. $\ell_o = \ell[o](t_o) \in \arg\min_{\ell \in L}\{D[o,\ell](t_o)\};\ R_o = D[o,\ell_o](t_o)$
4. **if** $d \in B_o$ **then return** $(D[o,d](t_o))$ /* exact answer */
5. **return** $(R_o + \Delta[\ell_o, d](t_o + R_o))$ /* $(1+\varepsilon+\psi)-$approximate answer */

Figure 5. The pseudocode describing **FCA**.

RQA(o, d, t_o, r)

1. **if** $\{o,d\} \cap L \neq \emptyset$
2. **then return** $(ASP[o,d](t_o), \Delta[o,d](t_o))$ /* $(1+\varepsilon)-$approximate answer */
3. $B_0 = B[o](t_o) := \{x \in V : D[o,x](t_o) < \min_{\ell \in L}\{D[o,\ell](t_o)\}\}$ /* **TDD**−run till ℓ_o is settled */
4. $\ell_0 \in \arg\min_{\ell \in L}\{D[o,\ell](t_o)\};\ R_0 = D[o,\ell_0](t_o)$
5. $sol_0 = (Q_0 \bullet \Pi_0\ ,\ \Delta[sol_0](t_o) = R_0 + \Delta[\ell_0, d](t_o + R_0))$ /* approximate answer via ℓ_o */
6. $k := 0;\ t_k = t_o;$
7. **while** $k < r$ **do**
7.1. "guess" the first vertex w_{k+1} out of B_k on $SP[w_k, d](t_k)$ /* exhaustive search */
7.2. $t_{k+1} = t_k + D[w_k, w_{k+1}](t_k);$
7.3. **if** $w_{k+1} \in L$
7.4. **then return** $(P_{0,k+1} \bullet \Pi[w_{k+1}, d](t_{k+1}), t_{k+1} - t_0 + \Delta[w_{k+1}, d](t_{k+1}))$ /* approximate answer via w_{k+1} */
7.5. $B_{k+1} = B[w_{k+1}](t_{k+1}) := \{x \in V : D[w_{k+1}, x](t_{k+1}) < \min_{\ell \in L}\{D[w_{k+1}, \ell](t_{k+1})\}\}$
7.6. $\ell_{k+1} \in \arg\min_{\ell \in L}\{D[w_{k+1}, \ell](t_{k+1})\};\ R_{k+1} = D[w_{k+1}, \ell_{k+1}](t_{k+1})$
7.7. $sol_{k+1} = \left(\begin{array}{c} P_{0,k+1} \bullet Q_{k+1} \bullet \Pi_{k+1}, \\ \Delta[sol_{k+1}](t_o) = t_{k+1} - t_o + R_{k+1} + \Delta[\ell_{k+1}, d](t_{k+1} + R_{k+1}) \end{array} \right)$ /* approximate answer via ℓ_{k+1} */
7.8. $k = k + 1$
8. **endwhile**
9. **return** $\min_{0 \le k \le r}\{sol_k\}$

Figure 6. The recursive algorithm **RQA** providing $(1+\sigma)-$approximate time-dependent shortest paths. $Q_k \in SP[w_k, \ell_k](t_k)$ is the shortest path connecting w_k to its closest landmark w.r.t. departure-time t_k. $P_{0,k} \in SP[o, w_k](t_o)$ is the prefix of the shortest $od-$path that has been already discovered, up to vertex w_k. $\Pi_k = ASP[\ell_k, d](t_k + R_k)$ denotes the $(1+\varepsilon)-$approximate shortest $\ell_k d-$path precomputed by the oracle.

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

C. Zaroliagis: Computer Engineering & Informatics Department, University of Patras, 26504 Rion, GREECE.

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