Algorithmica (2016) 74:1404–1434
DOI 10.1007/s00453-015-0003-0

Distance Oracles for Time-Dependent Networks

Spyros Kontogiannis[1,2] · **Christos Zaroliagis**[2,3]

Received: 6 August 2014 / Accepted: 23 April 2015 / Published online: 6 May 2015

Abstract We present the first approximate distance oracle for sparse directed networks with *time-dependent* arc-travel-times determined by continuous, piecewise linear, positive functions possessing the FIFO property. Our approach precomputes $(1+\varepsilon)$-approximate distance summaries from selected landmark vertices to all other vertices in the network. Our oracle uses subquadratic space and time preprocessing, 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, for any constant $\sigma > \varepsilon$. Our oracle is based only on the sparsity of the network, along with two quite natural assumptions about travel-time functions which allow the smooth transition towards asymmetric and time-dependent distance metrics.

Keywords Time-dependent shortest paths · FIFO property · Distance oracles

This work was supported by EU FP7/2007-2013 under Grant Agreements Nos. 288094 (eCOMPASS) and 609026 (MOVESMART), and partially done while both authors were visiting the Karlsruhe Instute of Technology.

✉ Christos Zaroliagis
zaro@ceid.upatras.gr

Spyros Kontogiannis
kontog@cs.uoi.gr

1 Department of Computer Science and Engineering, University of Ioannina, 45110 Ioannina, Greece

2 Computer Technology Institute and Press "Diophantus", 26504 Patras, Greece

3 Department of Computer Engineering and Informatics, University of Patras, 26504 Patras, Greece

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., [2,27,28,32,34–36]) 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 (maximum ratio, over all origin–destination pairs, between the distance returned by the oracle and the actual distance). For an overview of distance oracles for static networks, the reader is referred to [31] and references therein.

1.1 Problem Setting and Motivation

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. 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 appropriate with respect to the historic traffic data that the route planning vendors have to digest, in order to provide their customers with fast route plans. To see why it is more appropriate, consider, for example, TomTom's *LiveTraffic* service[1] which provides real-time estimations of average travel-time values, collected by periodically sampling the average speed of each road segment in a city, using the cars connected to the service as sampling devices. The crux is how to exploit all this historic traffic information, in order to *efficiently* provide route plans that will adapt to the departure-time from the origin. A customary way towards this direction is to consider the continuous piecewise linear (pwl) interpolants of these sample points as *arc-travel-time functions* of the corresponding instance.

Computing a time-dependent shortest path for a triple (o, d, t_o) of an origin o, a destination d and a departure-time t_o from the origin, has been studied extensively (see e.g., [6,14,26]). The shape of arc-travel-time functions and the waiting policy at vertices may considerably affect the tractability of the problem [26]. A crucial property is the *FIFO property*, according to which each arc-arrival-time at the head of an arc is a *non-decreasing* function of the departure-time from the tail. If *waiting-at-vertices* is forbidden and the arc-travel-time functions may be non-FIFO, then subpath optimality and simplicity of shortest paths is not guaranteed [26]. Thus, even if it exists, an optimal route is not computable by (extensions of) well known techniques, such as Dijkstra

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

or Bellman–Ford. Additionally, many variants of the problem are also **NP**-hard [30]. On the other hand, 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 (**TDD**), which relaxes arcs by computing the arc costs "on the fly", when scanning their tails. This has been first observed in [14], where the *unrestricted waiting policy* was (implicitly) assumed for vertices, along with the non-FIFO property for arcs. The FIFO property may seem unreasonable in some application scenarios, e.g., when travellers at the dock of a train station wonder whether to take the very next slow train towards destination, or wait for a subsequent but faster train.

Our motivation in this work stems from *route planning* in urban-traffic road networks where the FIFO property seems much more natural, since all cars are assumed to travel according to the same (possibly time-dependent) average speed in 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 [15]. Additionally, when shortest-travel-times are well defined and optimal waiting-times at nodes always exist, a non-FIFO arc with *unrestricted-waiting-at-tail* policy is equivalent to a FIFO arc in which waiting at the tail is not beneficial [26]. Therefore, our focus in this work is on networks with FIFO arc-travel-time functions.

1.2 Related Work and Main Challenge

The study of shortest paths is one of the cornerstone problems in Computer Science and Operations Research. Apart from the well-studied case of instances with static arc weights, several variants towards time-evolving instances have appeared in the literature. We start by mentioning briefly the most characteristic attempts regarding non time-dependent models, and subsequently we focus exclusively on related work regarding time-dependent shortest path models.

In the *dynamic shortest path* problem (e.g., [13,19,29,33]), the arcs are allowed to be inserted to and/or deleted from the graph in an online fashion. The focus is on maintaining and efficiently updating a data structure representing the shortest path tree from a single source, or at least supporting fast shortest path queries between arbitrary vertices, in response to these changes. The main difference with the problem we study is exactly the online fashion of the changes in the characteristics of the graph metric. In *temporal networks* (e.g., [3,18,21]), each arc comes with a vector of discrete arc-labels determining the time-slots of its availability. The goal is then to study the reachability and/or computation of shortest paths for arbitrary pairs of vertices, given that the chosen connecting path must also possess at least one non-decreasing subsequence of arc-labels, as we move from the origin to the destination. This problem is indeed a special case of the time-dependent shortest paths problem, in the sense that the availability patterns may be encoded as distance functions which switch between a finite and an infinite traversal cost. Typically these problems do not possess the FIFO property, but one may exploit the discretization of the time axis, which essentially determines the complexity of the instance to solve. In the *stochastic shortest path* problem (e.g., see [5,24,25]) the uncertainty of the arc weights is modeled

by considering them as random variables. The goal is again the computation of paths with minimum expected weight. This is also a hard problem, but in the time-dependent shortest path there is no uncertainty on the behavior of the arcs. In the *parametric shortest path* problem, the graph comes with two distinct arc-weight vectors. The goal is to determine a shortest path with respect to any possible linear combination of the two weight vectors. It is well known [22] that a shortest *od*-path may change $|V|^{\Omega(|V|)}$ times as the parameter of the linear combination changes. An upper bound of at most $|V|^{\mathcal{O}(|V|)}$ is also well-known [17]. The main difference with the *time-dependent shortest path* problem studied in the present work is that, when computing path lengths, rather than (essentially) composing the arrival-time functions of the constituent arcs, in the parametric shortest path problem the arc-lengths are simply added.

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, and neglected the computational complexity of providing succinct representations of the entire earliest-arrival-time *functions*, for *all* departure-times from the origin. Such representations, 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*, or *search profiles*) from central vertices (e.g., *landmarks* or *hubs*) towards other vertices in the network, providing a crucial ingredient for the construction of distance oracles to support real-time responses to arbitrary queries $(o, d, t_o) \in V \times V \times \mathbb{R}$.

The complexity of succinctly representing earliest-arrival-time functions was first questioned in [7–9], but was solved only recently by a seminal work [16] 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 arc-travel-time functions. Polynomial-time algorithms (or even PTAS) for constructing *point-to-point* $(1+\varepsilon)$-approximate distance functions are provided in [10,16], 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).

To the best of our knowledge, the problem of providing distance oracles for time-dependent networks with *provably* good approximation guarantees, small preprocessing-space complexity and sublinear query time complexity, has not been investigated so far. 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 the distance summaries that will be preprocessed and stored by the oracle. Exploiting a PTAS (such as that in [16]) for computing approximate distance functions, one could provide a trivial oracle with query-time complexity $Q \in \mathcal{O}(\log\log(K^*))$, at the cost of an exceedingly high space-complexity $S \in \mathcal{O}\big((1 + K^*) \cdot n^2\big)$, by storing succinct representations of all the point-to-point $(1 + \varepsilon)$-approximate shortest-travel-time functions. At the other

extreme, one might use the minimum possible space complexity $S \in \mathcal{O}(n+m+K)$ for storing the input, at the cost of suffering a query-time complexity $Q \in \mathcal{O}(m + n\log(n)[1 + \log\log(1 + K_{\max})])$ (i.e., respond to each query by running **TDD** in real-time using a predecessor search structure for evaluating pwl functions).[2] The main challenge considered in this work is to smoothly close the gap between these two extremes, i.e., to achieve a better (e.g., *sublinear*) query-time complexity, while consuming smaller space-complexity (e.g., $o\big((1 + K^*) \cdot n^2\big)$) for succinctly representing travel-time *functions*, and enjoying a small (e.g., close to 1) approximation guarantee (stretch factor).

1.3 Our Contribution

We have successfully addressed the aforementioned challenge by presenting the *first* approximate distance oracle for sparse directed graphs with time-dependent arc-travel-times, which achieves all these goals. Our oracle is based only on the sparsity of the network, plus two assumptions of travel-time functions which are quite natural for route planning in road networks (cf. Assumptions 1 and 2 in Sect. 2). It should be mentioned that: (i) even in static undirected networks, achieving a stretch factor below 2 using subquadratic space and sublinear query time, is possible only when $m \in o\big(n^2\big)$, as it has been recently shown [2,28]; (ii) there is important applied work [4,11,12,23] to develop time-dependent shortest path *heuristics*, which however provide mainly empirical evidence on the success of the adopted approaches.

At a high level, our approach resembles the typical ones used in *static* and *undirected* graphs (e.g., [2,28,34]): Distance summaries from selected 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, 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* and *directed* graphs, since one is confronted with two highly non-trivial challenges: (i) handling directedness, and (ii) dealing with time-dependence, i.e., deciding the arrival-times to grow balls around vertices in the vicinity of 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. A novelty of our query algorithms, contrary to other approaches, is exactly that we achieve the approximation guarantees by growing balls only from vertices around the origin. Managing this was a necessity for our analysis since growing balls around vertices in the vicinity of the destination at the *right* arrival-time is essentially not an option.

Our specific contribution is as follows. Let U be the worst-case number of breakpoints for an $(1+\varepsilon)$-approximation of a *concave* distance function stored in our oracle, and let TDP be the maximum number of time-dependent shortest path probes required for their construction. Then, we are able to construct a distance oracle that efficiently preprocess $(1+\varepsilon)$-approximate distance functions from a set of landmarks, which are uniformly and independently selected with probability ρ, to all other vertices, in order

[2] $K_{\max}$ denotes the maximum number of breakpoints in an arc-travel-time function.

Table 1 Our main result (third row) and its comparison to the straightforward oracles with all-to-all preprocessing and no preprocessing at all, for a given approximation guarantee $1+\varepsilon$ of the preprocessed data

What is preprocessed	Preproc. space	Preproc. time	Query time
All-To-All	$\mathcal{O}\big((K^*+1)n^2U\big)$	$\mathcal{O}\left(\begin{matrix}n^2\log(n)\\ \cdot\log\log(K_{\max})\\ \cdot(K^*+1)TDP\end{matrix}\right)$	$\mathcal{O}\big(\log\log(K^*)\big)$
Nothing	$\mathcal{O}(n+m+K)$	$\mathcal{O}(1)$	$\mathcal{O}\left(\begin{matrix}n\log(n)\cdot\\ \log\log(K_{\max})\end{matrix}\right)$
Landmarks-To-All [This paper]	$\mathcal{O}\big(\rho n^2(K^*+1)U\big)$	$\mathcal{O}\left(\begin{matrix}\rho n^2\log(n)\\ \cdot\log\log(K_{\max})\\ \cdot(K^*+1)TDP\end{matrix}\right)$	$\mathcal{O}\left(\begin{matrix}\left(\frac{1}{\rho}\right)^{r+1}\cdot\log\left(\frac{1}{\rho}\right)\\ \cdot\log\log(K_{\max})\end{matrix}\right)$
$K_{\max}\in\mathcal{O}(1)$ $\rho=n^{-a}$, $U,TDP\in\mathcal{O}(1)$ $K^*\in\mathcal{O}(\text{polylog}(n))$	$\tilde{\mathcal{O}}\big(n^{2-a}\big)$	$\tilde{\mathcal{O}}\big(n^{2-a}\big)$	$\mathcal{O}\left(n^{(r+1)a}\right)$

The fourth row presents an explicit trade-off among preprocessing time/space and query time. $\tilde{\mathcal{O}}(\ )$ hides polylogarithmic factors

to provide real-time responses to arbitrary queries via a recursive query algorithm of recursion depth (budget) r. The specific expected preprocessing and query bounds of our oracle are presented in Table 1 (3rd row) along with a comparison with the best previous approaches (straightforward oracles).

Our oracle guarantees a stretch factor of $1+\varepsilon\frac{(1+\frac{\varepsilon}{\psi})^{r+1}}{(1+\frac{\varepsilon}{\psi})^{r+1}-1}$, where ψ is a fixed constant depending on the characteristics of the arc-travel-time functions, but is independent of the network size. As it is proved in Theorem 1 (Sect. 3), U and TDP are independent of the network size n and thus we can treat them as constants. Similarly, $K_{\max}$ (which is also part of the input) is 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, note that if all arc-travel-time functions are concave, i.e., $K^*=0$, then we clearly achieve subquadratic preprocessing space and time for any $\rho\in\mathcal{O}\big(n^{-\alpha}\big)$, where $0<\alpha<\frac{1}{r+1}$. Real data (e.g., TomTom's traffic data for the city of Berlin [15]) 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 thus 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. Our analysis guarantees that, when $K^*\in o(n)$, one can fine-tune the parameters of the oracles so that both sublinear query times and subquadratic preprocessing space can be guaranteed. For example, assuming $K^*\in\mathcal{O}(\text{polylog}(n))$, we get the trade-off presented in the 4th row of Table 1. We also note that, apart from the choice of landmarks, our algorithms are deterministic.

The rest of the paper is organized as follows. Section 2 gives the ingredients and presents an overview of our approach. Section 3 presents our preprocessing algorithm. Our constant approximation query algorithm is presented in Sect. 4, while our PTAS query algorithm is presented in Sect. 5. Our main results are summarized in Sect. 6. The details on how to compute the actual path from the approximate distance values are presented in Sect. 7. We conclude in Sect. 8. A preliminary version of this work appeared as [20].

2 Ingredients and Overview of Our Approach

2.1 Notation

Our input is provided by a network (directed graph) $G = (V, A)$ with n vertices and $m = O(n)$ arcs. Every arc $uv \in A$ is equipped with a periodic, continuous, piecewise-linear (pwl) *arc-travel-time* (a.k.a. *arc-delay*) function $D[uv] : \mathbb{R} \to \mathbb{R}_{>0}$, such that $\forall k \in \mathbb{Z}, \forall t_u \in [0, T),\ D[uv](k \cdot T + t_u) = D[uv](t_u)$ is the arc-travel-time of uv when the departure-time from u is $k \cdot T + t_u$. $D[uv]$ is represented succinctly as a continuous pwl function, by K_{uv} breakpoints describing its projection to $[0, T)$. $K = \sum_{uv \in A} K_{uv}$ is the number of breakpoints to represent all the arc-delay functions in the network, and $K_{\max} = \max_{uv \in A} K_{uv}$. K^* is the number of *concavity-spoiling* breakpoints, i.e., the ones in which the arc-delay slopes increase. Clearly, $K^* \leq K$, and $K^* = 0$ for *concave* pwl functions. The space to represent the entire network is $\mathcal{O}(n + m + K)$.

The *arc-arrival* function $Arr[uv](t_u) = t_u + D[uv](t_u)$ represents arrival-times at v, depending on the departure-times t_u from u. Note that we can express the same delay function of an arc $a = uv$ as a function of the arrival-time $t_v = t_u + D[uv](t_u)$ at the head v. This is specifically useful when we need to work with the *reverse network* ($\overleftarrow{G} = (V, A, (\overleftarrow{D}[a])_{a \in A})$, where $\overleftarrow{D}[uv]$ is the delay of arc $a = uv$, measured now as a function of the arrival-time t_v at v. For instance, consider an arc $a = uv$ with $D[uv](t_u) = t_u + 1$, $0 \leq t_u \leq 3$. Then, $t_v = 2t_u + 1$ and $1 \leq t_v \leq 7$. Now, the same delay function can be expressed as a function of t_v as $\overleftarrow{D}[uv](t_v) = t_v - t_u = t_v - \frac{t_v - 1}{2} = \frac{t_v + 1}{2}$, for $1 \leq t_v \leq 7$.

For any $(o, d) \in V \times V$, $\mathcal{P}_{o,d}$ is the set of od-paths, and $\mathcal{P} = \cup_{(o,d)} \mathcal{P}_{o,d}$. For a path $p \in \mathcal{P}$, $p_{x \rightsquigarrow y}$ is its subpath from (the first appearance of) vertex x until (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* and *shortest-travel-time* functions from o to d are: $\forall t_o \in [0, T)$, $Arr[o, d](t_o) = \min_{p \in \mathcal{P}_{o,d}} \{Arr[p](t_o)\}$ and $D[o, d](t_o) = Arr[o, d](t_o) - t_o$. Finally, $SP[o, d](t_o)$ (resp. $ASP[o, d](t_o)$) is the set of shortest (resp., with stretch-factor at most $(1 + \varepsilon)$) od-paths for a given departure-time t_o.

2.2 Facts of the FIFO Property

We consider networks $(G = (V, A), (D[a])_{a\in A})$ with continuous arc-delay functions, possessing the *FIFO* (a.k.a. *non-overtaking*) property, according to which all arc-arrival-time functions are non-decreasing:

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

The FIFO property is *strict*, if the above inequality is strict. The following properties (Lemmata 1–3), are, perhaps, more-or-less known. We state them here and provide their proofs only for the sake of completeness.

Lemma 1 (FIFO Property and Arc-Delay Slopes) *If the network satisfies the (strict) FIFO property then any arc-delay function has left and right derivatives with values at least (greater than)* -1.

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

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

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

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

Lemma 2 (FIFO Property for Paths) *If the network satisfies the FIFO property, then* $\forall p \in \mathcal{P}, \forall t_1 \in \mathbb{R}, \forall \delta > 0,\ Arr[p](t_1) \leq Arr[p](t_1 + \delta)$ *. In case of strict FIFO property, the inequality is also strict. The (strict) monotonicity holds also for* $Arr[o, d]$.

Proof To prove the FIFO property for a path $p = \langle a_1, \ldots, a_k\rangle \in \mathcal{P}$, we use a simple inductive argument on the prefixes of p, based on a recursive definition of path-arrival-time functions. $\forall 1 \leq i \leq j \leq k$, 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:

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

The composition of non-decreasing (increasing) functions is well known to also be non-decreasing (increasing). Applying a minimization operation to produce the earliest-arrival-time function $Arr[o, d] = \min_{p\in\mathcal{P}_{o,d}} \{Arr[p]\}$, preserves the same kind of monotonicity. □

It is well-known that in FIFO (or equivalently, non-FIFO with unrestricted-waiting-at-nodes) networks the crucial property of *prefix-subpath optimality* is preserved [14]. We strengthen this observation to the more general (arbitrary) *subpath optimality*, for strict FIFO networks.

Lemma 3 (Subpath Optimality in strict FIFO Networks) *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* $p^* \in SP[u, v](t_u)$, *it holds for every subpath* $q^* \in \mathcal{P}_{x,y}$ *of* p^* *that* $q^* \in SP[x, y](Arr[p^*_{u \rightsquigarrow x}](t_u))$. *In other words,* q^* *is a shortest path between its endpoints* x, y *for the earliest-departure-time from* x, *given* t_u.

Proof 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 $p^*_{y \rightsquigarrow v}$). □

Lemma 3 implies that both Dijkstra's label setting algorithm and Bellman–Ford label-correcting algorithm also work in time-dependent strict FIFO networks, under the usual conventions for static instances (positivity of arc-delays for Dijkstra, and inexistence of negative-travel-time cycles for Bellman–Ford).

In the following, we shall refer to an execution of the time-dependent Dijkstra's algorithm (**TDD**) from origin $o \in V$, with departure time $t_o \in [0, T)$, either as "*a run of* **TDD** *from* (o, t_o)", or as "*growing a* (**TDD**) *ball around (or centered at)* (o, t_o) *(by running* **TDD***)*".

The time-complexity of **TDD** is slightly worse than the corresponding complexity of Dijkstra's algorithm in the static case, since during the relaxation of each arc the actual arc-travel-time of the arc has to be *evaluated* rather than simply retrieved. For example, if each arc-travel-time function $D[uv]$ is periodic, continuous and pwl, represented by at most $K_{\max}$ breakpoints, then the evaluation of arc-travel-times can be done in time $\mathcal{O}(\log\log(K_{\max}))$, e.g. by using a predecessor-search structure to determine the right leg for each function. Therefore, the time complexity for **TDD** would be $\mathcal{O}([m + n\log(n)]\log\log(K_{\max}))$.

2.3 Towards a Time-Dependent Distance Oracle

Our approach for providing a time-dependent distance oracle is inspired by the generic approach for general *undirected* graphs under *static* travel-time metrics. However, we have to tackle the two main challenges of *directedness* and *time-dependence*. Notice that together these two challenges imply an *asymmetric* distance metric, which also *evolves* with time. Consequently, to achieve a smooth transition from the static and undirected world towards the time-dependent and directed world, we have to quantify the *degrees of asymmetry and evolution* in our metric.

Towards this direction, we introduce some metric-related parameters which quantify (i) the steepness of the shortest-travel-time functions (via the parameters $\Lambda_{\min}$ and $\Lambda_{\max}$), and (ii) the degree of asymmetry (via the parameter ζ). We make two assumptions on the values of these parameters, namely, that they have constant (in particular, independent of the network size) values. 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 $[\Lambda_{\min}, \Lambda_{\max}]$.

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

The lower bound $-\Lambda_{\min} > -1$ is justified by the FIFO property (cf. Lemmata 1 and 2 in Sect. 2.2). $\Lambda_{\max}$ represents the maximum possible rate of change of shortest-travel-times in the network, which only makes sense to be bounded (in particular, independent of the network size) in realistic instances such as the ones representing urban-traffic time-dependent road networks.

Towards justifying this assumption, we conducted an experimental analysis with two distinct data sets. The first one is a real-world time-dependent snapshot of two weeks traffic data of the city of Berlin, kindly provided to us by TomTom [15] (consisting of $n = 478{,}989$ vertices and $m = 1{,}134{,}489$ arcs), in which the arc-delay functions are the continuous, pwl interpolants of five-minute samples of the average travel-times in each road segment. The second data set is a benchmark time-dependent instance of Western Europe's (WE) road network (consisting of $n = 18{,}010{,}173$ vertices and $m = 42{,}188{,}664$ arcs) kindly provided by PTV AG for scientific use. The time-dependent arc travel time functions were generated as described in [23], reflecting a high amount of traffic for all types of roads (highways, national roads, urban roads), all of which posses non-constant time-dependent arc travel time functions.

We conducted 10,000 random queries (o, d, t_o) in the Berlin (real-world) instance, focusing on the harder case of rush-hour departure times. We computed the approximate distance functions towards all the destinations using our one-to-all approximation algorithm (cf. Sect. 3) with approximation guarantee $\varepsilon = 0.001$. Our observration was that all the *shortest-travel-time* slopes were $\Lambda_{\max} \leq 0.1843$. That is, the travel-time on every road segment increases at a rate of at most 18.43 % as departure time changes. An analogous experimentation with the benchmark instance of Europe (heavy-traffic variant), again by conducting 10,000 random queries, demonstrated shortest-travel-time slopes at most $\Lambda_{\max} \leq 6.1866$.

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).

Assumption 2 (*Bounded Opposite Trips*) There is a constant $\zeta \geq 1$ such that: $\forall (o, d) \in V \times V,\ \forall t \in [0, T),\ D[o, d](t) \leq \zeta \cdot D[d, o](t)$.

This is also a quite natural assumption in road networks, because it is most unlikely that a trip in one direction would be, say, more than 10 times longer than the trip in the opposite direction (but not necessarily along the reverse path) during the same time period. This was also justified by the two instances at our disposal. In each instance we uniformly selected 10,000 random origin–destination pairs and departure times randomly chosen from the rush-hour period, which is the most interesting and diverging case. For the Berlin-instance the resulting worst-case value was $\zeta \leq 1.5382$. For the WE-instance the resulting worst-case value was $\zeta \leq 1.174$.

A third assumption that we make is that the maximum out-degree of every node is bounded by 2. This can be easily guaranteed 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 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 1 demonstrates an example of such a substitution. For each node $v \in G$ with out-degree $d^+(v) > 2$, the node substitution operation is executed in time $\mathcal{O}\big(d^+(v)\big)$ and introduces $d^+(v) - 1$ new nodes and $d^+(v) - 2$ new arcs (of zero delays). Therefore, in time $\mathcal{O}(|A|)$ we can ensure 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).

2.4 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

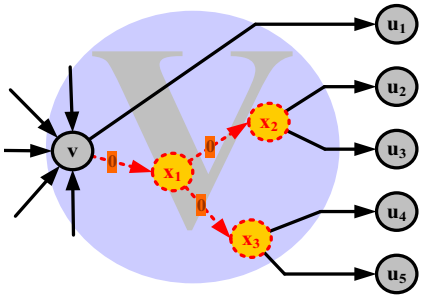


Fig. 1 The node substitution operation for a vertex $v \in V$ with $d^+_G(v) = 5$. The operation ensures an out-degree at most 2 for all the newly inserted vertices in place of v in the graph. The new graph elements (*nodes* and *arcs*) are indicated by *dashed* (*red*) *lines*. The *solid* (*black*) *arcs* and vertices are the ones pre-existing in the graph (Color figure online)

$L \subset V$ 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, that is, we can choose the landmarks randomly among all vertices, or we can choose as landmarks the vertices in the cut sets provided by some graph partitioning algorithm. Nevertheless, for the sake of the analysis we assume that landmark selection 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 time and space $o\big((1 + K^*)n^2\big)$. Then, we provide two approximation algorithms for responding to arbitrary queries $(o, d, t_o) \in V \times V \times [0, T)$. The first (**FCA**) is a simple *sublinear*-time constant-approximation algorithm (cf. Sect. 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 depth of the recursion (the *recursion budget*) has been exhausted. **RQA** finally responds with a $(1+\sigma)$-approximate travel-time to the query in *sublinear* time, for any constant $\sigma > \varepsilon$ (cf. Sect. 5). As it is customary in the distance oracle literature, the query times of our algorithms concern the determination of (upper bounds on) 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 (cf. Sect. 7).

3 Preprocessing Distance Summaries

The purpose of this section is to demonstrate how to construct the preprocessed information that will comprise the *distance summaries* of the oracle, i.e., all landmark-to-vertex $(1 + \varepsilon)$-approximate 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 $uv \in A$) we run a reverse variant of **TDD** (going "back in time") with root (u, t_u) on the *network* $(\overleftarrow{G} = (V, A, (\overleftarrow{D}[a])_{a\in A})$, where $\overleftarrow{D}[uv]$ is the delay of arc $a = uv$, measured now as a function of the arrival-time t_v at the head v. The algorithm proceeds backwards both along the connecting path (from the destination towards the origin) and in time. As a result, we compute all *latest-departure-times* from landmarks that allow us to determine the images (i.e., projections to appropriate departure-times from all possible origins) of concavity-spoiling breakpoints to the spaces of departure-times from each of the landmarks. Then, for each landmark, we repeat the procedure for concave, continuous, pwl arc-delay functions—described in the rest of this section—independently for each of the (at most) $K^* + 1$ consecutive subintervals of $[0, T)$ determined by these consecutive images of concavity-spoiling breakpoints. 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 $\Delta[\ell, v] : [0, T) \to \mathbb{R}_{\geq 0}$ of the shortest-travel-time functions $D[\ell, v] : [0, T) \to \mathbb{R}_{\geq 0}$, i.e., for each $(\ell, v) \in L \times V$ we have to compute a continuous pwl function $\Delta[\ell, v]$ with a *constant* number of breakpoints, such that $\forall t_o \in [0, T)$, $D[\ell, v](t_o) \leq \Delta[\ell, v](t_o) \leq (1+\varepsilon) \cdot D[\ell, v](t_o)$. An algorithm providing such functions in a *point-to-point* fashion was proposed in [16]. For each landmark $\ell \in L$, it has to be executed n times so as to construct all the required landmark-to-vertex approximate functions. The main idea of that algorithm is to keep sampling the travel-time axis of the unknown function $D[\ell, v]$ at a logarithmically growing scale, until its slope becomes less than 1. It then samples the departure-time axis via bisection, until the required approximation guarantee is achieved. All the sample points (in both phases) correspond to breakpoints of a lower-approximating function. The upper-approximating function has at most twice as many points. The number of breakpoints returned may be suboptimal, given the required approximation guarantee: even for an affine shortest-travel-time function with slope 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 function.

Our solution is an improvement of the approach in [16] in three aspects:

(i) It computes *concurrently* all the required approximate distance functions from a given landmark, at a cost equal to that of a single (worst-case with respect to the given origin and all possible destinations) point-to-point approximation of [16].
(ii) Within every subinterval of consecutive images of concavity-spoiling breakpoints, it requires 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 $\mathcal{O}(|L|)$. This is also claimed in [16], but it is actually true only for their second phase (the bisection). For the first phase of their algorithm, there is no such guarantee.
(iii) It provides an exact closed form estimation (see below) of the worst-case absolute error, which guides our method.

In a nutshell, our approach constructs two continuous pwl-approximations of the unknown shortest-travel-time function $D[\ell, v] : [0, T) \to \mathbb{R}_{\geq 0}$, 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 "sandwiched" between these two approximations.

To achieve a concurrent one-to-all construction of upper-bounding approximations from a given landmark $\ell \in L$, our algorithm is purely based on bisection. This is done because the departure-time axis is common for all these unknown functions $(D[\ell, v])_{v \in V}$. In order for this technique to work, despite the fact that the slopes may be greater than one, a crucial ingredient is an *exact closed-form estimation* of the worst-case absolute error that we provide. This helps our construction to indeed consider only the necessary sampling points as breakpoints of the corresponding (concurrently constructed) shortest travel-time functions. It is mentioned that this guarantee could also be used in the first phase of the approximation algorithm in [16],

in order to discard all unnecessary sampling points from being actual breakpoints in the approximate functions. Consequently, we start by providing the closed form estimation of the maximum absolute error and then we present our one-to-all approximation algorithm.

3.1 Absolute Error Estimation

In this section, we provide 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 $[t_s, t_f) \subseteq [0, T)$ that contains no other primitive image, apart possibly from its endpoints.

For an interval $[t_s, t_f) \subseteq [0, T)$, fix an unknown, but amenable to polynomial-time sampling, continuous (not necessarily pwl) *concave* function $D : [t_s, t_f) \to \mathbb{R}_{>0}$, with right and left derivative values at the endpoints $\Lambda^+(t_s)$, $\Lambda^-(t_f)$. Assume that $\Lambda^+(t_s) > \Lambda^-(t_f)$ and $L = t_f - t_s > 0$.

Let $m = \frac{D(t_f)-D(t_s)+t_s\cdot\Lambda^+(t_s)-t_f\cdot\Lambda^-(t_f)}{\Lambda^+(t_s)-\Lambda^-(t_f)}$ and $\overline{D}_m = \Lambda^+(t_s)\cdot(m - t_s) + D(t_s)$.

Lemma 4 *For an interval $[t_s, t_f) \subseteq [0, T)$ and a concanve function $D : [t_s, t_f) \to \mathbb{R}_{>0}$ defined as above, consider the affine function $\underline{D}$ passing via the points $(t_s, D(t_s))$, $(t_f, D(t_f))$. Consider also the pwl function $\overline{D}$ with three breakpoints $(t_s, D(t_s))$, $(m, \overline{D}_m)$, $(t_f, D(t_f))$. Then, $\forall t \in [t_s, t_f)$, $\underline{D}(t) \le D(t) \le \overline{D}(t)$ and the* maximum absolute error *(MAE) between $\underline{D}$ and $\overline{D}$ in $[t_s, t_f)$ is expressed by the following form:*

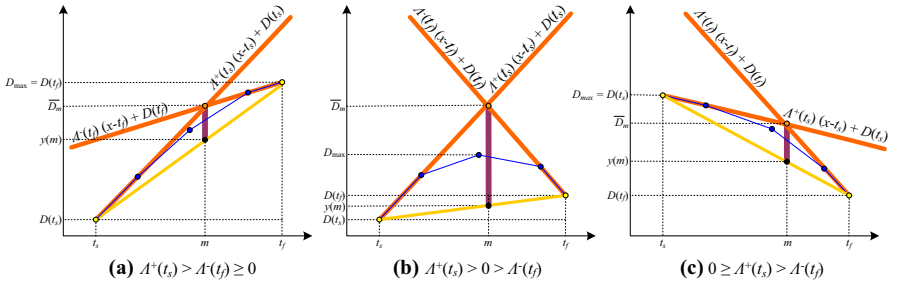
$$MAE(t_s, t_f) = (\Lambda^+(t_s) - \Lambda^-(t_f)) \cdot \frac{(m - t_s)\cdot(t_f - m)}{L} \le \frac{L\cdot(\Lambda^+(t_s) - \Lambda^-(t_f))}{4}.$$

Proof Consider the affine functions (see also Fig. 2):

$$\begin{aligned} y(x) &= \frac{D(t_f) - D(t_f)}{L}\cdot x + \frac{D(t_s)t_f - D(t_f)t_s}{L}, \\ y_s(x) &= \Lambda^+(t_s)\cdot(x - t_s) + D(t_s), \\ y_f(x) &= \Lambda^-(t_f)\cdot(x - t_f) + D(t_f). \end{aligned}$$

The point $\left(m = \frac{D(t_f)-D(t_s)+t_s\cdot\Lambda^+(t_s)-t_f\cdot\Lambda^-(t_f)}{\Lambda^+(t_s)-\Lambda^-(t_f)}, \overline{D}_m = y_s(m) = y_f(m)\right)$ is the intersection point of the lines $y_s(x)$ and $y_f(x)$. As an upper-bounding (pwl) function of D in $[t_s, t_f)$ we consider $\overline{D}(t) = \min\{y_s(t), y_f(t)\}$, 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 $[t_s, t_f)$ the left-derivative value is at least as large as the right-derivative value. Thus, the restriction of D on $[t_s, t_f)$ lies entirely in the area of the triangle $\{(t_s, D(t_s)), (m, \overline{D}_m), (t_f, D(t_f))\}$. The maximum possible distance (additive error) of $\overline{D}$ from $\underline{D}$ is:



(a) $\Lambda^+(t_s) > \Lambda^-(t_f) \geq 0$ **(b)** $\Lambda^+(t_s) > 0 > \Lambda^-(t_f)$ **(c)** $0 \geq \Lambda^+(t_s) > \Lambda^-(t_f)$

Fig. 2 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 (Color figure online)

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

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 Fig. 2). The calculations are identical for the three distinct cases shown in Fig. 2. Let $\underline{\Lambda} = \frac{D(t_f) - D(t_s)}{L}$ be the slope of the line $y(x)$. Observe that:

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

Thus we have:

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

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

3.2 One-To-All Approximation Algorithm

We now present our polynomial-time algorithm which provides asymptotically space-optimal succinct representations of one-to-all $(1 + \varepsilon)$-approximating functions $\overline{\mathbf{D}}[\ell, \star] = (\overline{D}[\ell, v])_{v \in V}$ of $D[\ell, \star] = (D[\ell, v])_{v \in V}$, for a given landmark $\ell \in L$ and all destinations $v \in V$, within a given time interval in which all the travel-time

functions from ℓ are *concave*. Recall our Assumption 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 [16] for point-to-point approximations of distance functions, to the case of a single-origin ℓ and all reachable destinations from it. Our method, which we call **BISECT**, computes *concurrently* (i.e., within the same bisection) all the required breakpoints to describe the (pwl) lower-approximating functions $\underline{\mathbf{D}}[\ell, \star] = \left(\underline{D}[\ell, v]\right)_{v \in V}$, 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 **BISECT**. This is possible due to our closed-form expression for the (worst-case) approximation error between the lower-approximating and the upper-approximating distance function, per destination vertex (cf. Lemma 4). Moreover, all the travel-times from ℓ 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 from (ℓ, t_ℓ).

Let the (unknown) concave travel-time function we wish to approximate be within a subinterval $[t_s, t_f) \subseteq [0, T)$. Let $D_{\min}[\ell, v](t_s, t_f) = \min_{t \in [t_s, t_f]}\{D[\ell, v](t)\}$, and $D_{\max}[\ell, v](t_s, t_f) = \max_{t \in [t_s, t_f]}\{D[\ell, v](t)\}$. Due to the concavity of $D[\ell, v]$ in $[t_s, t_f]$, we have that $D_{\min}[\ell, v](t_s, t_f) = \min\{D[\ell, v](t_s), D[\ell, v](t_f)\}$.

The **BISECT** 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 by Lemma 4) has not been reached yet, and the remaining *inactive*. 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 $\left[t_s, (t_s + t_f)/2\right]$ and $\left[(t_s + t_f)/2, t_f\right]$. 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.

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 breakpoints for $\overline{D}[\ell, v]$: per consecutive pair of breakpoints in $LBP[\ell, v]$ that are added to $UBP[\ell, v]$, we must also add their intermediate breakpoint $(m, \overline{D}_m)$ to $UBP[\ell, v]$ (cf. proof of Lemma 4).

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 $[0, T)$.

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,[3] continuous, pwl arc-travel-time functions, with bounded shortest-travel-time slopes.

Theorem 1 *For a given $\ell \in L$ and any $v \in V$,* **BISECT** *computes an asymptotically optimal, independent of the network size, number of breakpoints*

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

where $D_{\max}[\ell, v](0, T)$ and $D_{\min}[\ell, v](0, T)$ denote the maximum and minimum shortest-travel-time values from ℓ to v within $[0, T)$. The 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:

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

Proof The time complexity of **BISECT** will be asymptotically equal to that of the worst-case point-to-point bisection from ℓ to some destination vertex v. In particular, **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.

We now bound the number of breakpoints produced by **BISECT**. The initial departure-times interval to bisect is $[0, T)$. Assume that we are currently at an 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 1), the absolute error between $\underline{D}[\ell, v]$ and $\overline{D}[\ell, v]$ in this interval is (by Lemma 4) 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

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

destination vertex $v \in V$ 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

$$k_{\max} = \max_{v \in V} \left\{ \left\lceil \log_2 \left(\frac{T \cdot (\Lambda_{\max} + 1)}{\varepsilon D_{\min}[\ell, v](0, T)} \right) \right\rceil \right\} - 2 \tag{3}$$

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

On the other hand, 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 **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 [16], 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 (unknown) 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:

$$L[\ell, v] \leq 2 \cdot \log_{1+\varepsilon} \left(\frac{D_{\max}[\ell, v](0, T)}{D_{\min}[\ell, v](0, T)} \right) \tag{4}$$

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

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

We now proceed with the time-complexity of **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 $[t_s, t_f]$, all the new breakpoints at the new departure-time $t_{mid} = \frac{t_s + t_f}{2}$, 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, taking into account relations (3) and (4), the total number TDP of forward TDSP probes required to construct $LBP[\ell, \star](0, T)$, is upper-bounded by

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

We can 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. □

Let $U = \max_{(\ell, v) \in L \times V}\{U[\ell, v]\}$. Theorem 1 dictates that U and TDP are *independent* of n and they only depend on the degrees of asymmetry and time-dependence of the distance metric. Therefore, they can be treated as constants. Combining the performance of **BISECT** with the fact that the expected number of landmarks is $\mathbb{E}\{|L|\} = \rho n$, it is easy to deduce the required preprocessing time and space complexities for constructing all the $(1 + \varepsilon)$-approximate landmark-to-vertex distance summaries, which is culminated in the next theorem.

Theorem 2 *The preprocessing phase of our time-dependent distance oracle has expected space/time complexities* $\mathbb{E}\{\mathcal{S}\} \in \mathcal{O}\big(\rho n^2(1 + K^*)U\big)$ *and* $\mathbb{E}\{\mathcal{P}\} \in \mathcal{O}\big(\rho n^2 \log(n) \log\log(K_{\max})(1 + K^*)TDP\big)$.

Proof For every landmark $\ell \in L$ and every destination vertex d, there are $(1 + K^*)$ subintervals that need to be bisected and **BISECT** can generate at most U new breakpoints in each such interval. Since there are n destinations, the total number of breakpoints that need to be stored for all landmarks (and all destinations) is $|L|n(1 + K^*)U$. This total number of breakpoints can be computed concurrently for all landmarks and all destinations (cf. proof of Theorem 1) by $|L|(1 + K^*)TDP$ runs of **TDD**. Each such runs takes time $\mathcal{O}((m + n \log(n)) \log\log(K_{\max}))$, where 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 $\mathcal{O}(\log\log(K_{\max}))$. The space and time bounds now follow from the fact that $m = O(n)$ and $\mathbb{E}\{|L|\} = \rho n$. □

4 Constant-Approximation Query Algorithm

Our next step towards a distance oracle is to provide a fast query algorithm providing constant approximation to the shortest-travel-time values of arbitrary queries

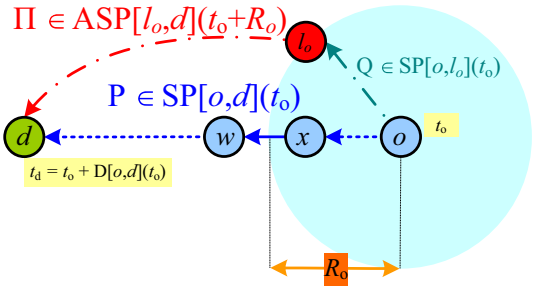


Fig. 3 The rationale of **FCA**. The *dashed* (*blue*) path P is a shortest od-path for (o, d, t_o). The *dashed-dotted* (*green* and *red*) path $Q \bullet \Pi$ is the via-landmark od-path indicated by the algorithm, if the destination vertex is out of the origin's **TDD** ball (Color figure online)

FCA(o, d, t_o)

1. **if** $o \in L$ **then return** $(\Delta[o, d](t_o))$ /* $(1+\varepsilon)-$approximate answer */
2. $B_o =$ **TDD**-ball around (o, t_o) until either d or the first landmark is settled
3. **if** $d \in B_o$ **then return** $(D[o, d](t_o))$ /* exact answer */
4. $\ell_o = B_o \cap L$; $R_o = D[o, \ell_o](t_o)$;
5. **return** $(R_o + \Delta[\ell_o, d](t_o + R_o))$ /* $(1+\varepsilon+\psi)-$approximation */

Fig. 4 The pseudocode describing **FCA**

$(o, d, t_o) \in V \times V \times [0, T)$. The proposed query algorithm, called *Forward Constant Approximation* (**FCA**), grows an outgoing ball

$$B_o := B[o](t_o) = \{x \in V : D[o, x](t_o) \leq \min\{D[o, d](t_o), D[o, \ell_o](t_o)\}\}$$

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. We call $R_o = \min\{D[o, d](t_o), D[o, \ell_o](t_o)\}$ the *radius* of B_o. If $d \in B_o$, then **FCA** returns the *exact* travel-time $D[o, d](t_o)$; otherwise, it returns the approximate travel-time value $R_o + \Delta[\ell_o, d](t_o + R_o)$ via ℓ_o. Figure 3 gives an overview of the whole idea. Figure 4 provides the pseudocode.

4.1 Correctness of FCA

The next theorem demonstrates that **FCA** returns od-paths whose travel-times are constant approximations to the shortest travel-times.

Theorem 3 $\forall (o, d, t_o) \in V \times V \times [0, T)$, **FCA** *returns either an exact path* $P \in SP[o, d](t_o)$, *or a via-landmark od-path* $Q \bullet \Pi$, *s.t.* $Q \in SP[o, \ell_o](t_o)$, $\Pi \in ASP[\ell_o, d](t_o + R_o)$, *and* $D[o, d](t_o) \leq R_o + \Delta[\ell_o, d](t_o + R_o) \leq (1 + \varepsilon) \cdot D[o, d](t_o) + \psi \cdot R_o \leq (1 + \varepsilon + \psi) \cdot D[o, d](t_o)$, *where* $\psi = 1 + \Lambda_{\max}(1 + \varepsilon)(1 + 2\zeta + \Lambda_{\max}\zeta) + (1 + \varepsilon)\zeta$.

Proof 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:

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

□

Note that **FCA** is a generalization of the 3-approximation algorithm in [2] 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. Observe that our algorithm smoothly departs, through the parameters $\Lambda_{\min}$, $\Lambda_{\max}$ and ζ, towards both *asymmetry* and *time-dependence* of the travel-time metric.

4.2 Complexity of FCA

The main cost of **FCA** is to grow the ball $B_o = B[o](t_o)$ by running **TDD**. Therefore, what really matters is the number of vertices in B_o, since the maximum out-degree is 2. Recall that L is chosen randomly by selecting each vertex v to become a landmark independently of other vertices, with probability $\rho \in (0, 1)$. Hence, for any $o \in V$ and any departure-time $t_o \in [0, T)$, the size of the outgoing **TDD**-ball $B_o = B[o](t_o)$ centered at (o, t_o) until the first landmark vertex is settled, behaves as a geometric random variable with success probability $\rho \in (0, 1)$. Consequently, $\mathbb{E}\{|B_o|\} = 1/\rho$, 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 $k = (1/\rho)\ln(1/\rho)$ we conclude that: $\mathbb{P}\{|B_o| > (1/\rho)\ln(1/\rho)\} \le \rho$. Since the maximum out-degree is 2, **TDD** will relax at most $2k$ arcs. Hence, we have established the following.

Theorem 4 *For the query-time complexity $\mathcal{Q}_{FCA}$ of* **FCA** *the following hold:*

$$
\begin{aligned}
&\mathbb{E}\{\mathcal{Q}_{FCA}\} \in \mathcal{O}((1/\rho)\ln(1/\rho)\log\log(K_{\max}))\ . \\
&\mathbb{P}\left\{\mathcal{Q}_{FCA} \in \Omega\left((1/\rho)\ln^2(1/\rho)\log\log(K_{\max})\right)\right\} \in \mathcal{O}(\rho)\ .
\end{aligned}
$$

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

The *Recursive Query Algorithm* (**RQA**) improves the approximation guarantee of the chosen *od*-path provided by **FCA**, by exploiting carefully a number of recursive calls of **FCA**, based on a given bound—called the *recursion budget* r—on the depth of the recursion tree to be constructed. Each of the recursive calls accesses the preprocessed information and produces another candidate *od*-path. The crux of our approach is the following: We ensure 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 from centers lying on the unknown shortest path, is lower-bounded by a *geometrically increasing* sequence. We prove that this sequence can only have a *constant* number of elements until the required approximation guarantee is reached, since the sum of all these radii provides a lower bound on the shortest-travel-time that we seek.

A similar approach was proposed for *undirected* and *static* sparse networks [2], 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 our case the network is both directed and time-dependent. Due to our ignorance of the exact arrival 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 $r - k > 0$ ($k \in \{0, \ldots, r\}$), we "guess" (by exhaustively searching for it) the next node w_k along the (unknown) 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 $Q_k \bullet \Pi_k$. Figure 5 provides an overview of **RQA**'s execution. Figure 6 provides the pseudocode of **RQA**.

5.1 Correctness and Quality of RQA

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 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 landmarks $\ell_o, \ldots, \ell_r$ 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. Since the preprocessed distance summaries stored by the oracle provide approximate travel-times corresponding to actual paths from landmarks to

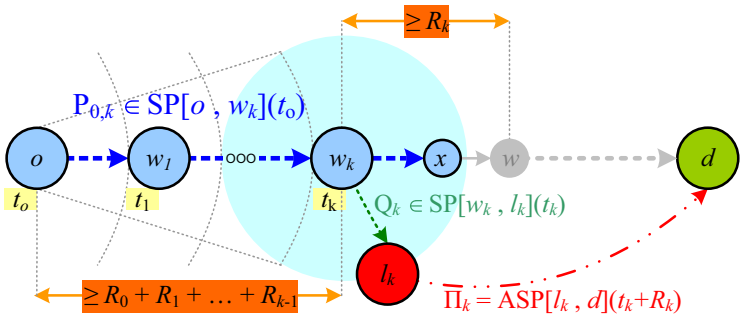


Fig. 5 Overview of the execution of **RQA** (Color figure online)

RQA(o, d, t_o, r)

1. **if** $o \in L$ **then return** $(ASP[o,d](t_o), \Delta[o,d](t_o))$ /* $(1+\varepsilon)-$approximation */
2. $B[o](t_o) :=$ **TDD**-ball from (o, t_o) until either d or a landmark is settled
3. **if** $d \in B_o$ **then return** $(D[o,d](t_o))$ /* exact suffix-subpath */
4. $\ell_0 \in B[o](t_o) \cap L$; $R_0 = D[o, \ell_0](t_o)$
5. $sol_0 = (Q_0 \bullet \Pi_0\ ,\ \Delta[sol_0](t_o) = R_0 + \Delta[\ell_0, d](t_o + R_0))$ /* via-ℓ_o approximation */
6. $k := 0$; $t_k = t_o$;
7. **while** $k < r$ **do**
- 7.1. "guess" the first vertex $w_{k+1} \in SP[w_k, d](t_k) \setminus B[w_k](t_k)$ /* 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. $B[w_{k+1}](t_{k+1}) :=$ **TDD**-ball until d or a landmark is settled
- 7.6 **if** $d \in B[w_{k+1}](t_{k+1})$ **then**
- 7.7. **then** $sol_{k+1} = \begin{pmatrix} P_{0,k+1} \bullet P_{k+1,d}, \\ \Delta[sol_{k+1}](t_o) = t_{k+1} - t_o + D[w_{k+1}, d](t_{k+1}) \end{pmatrix}$
- 7.8. **else**
- 7.8.1 $\ell_{k+1} \in L \cap B[w_{k+1}](t_{k+1})$; $R_{k+1} = D[w_{k+1}, \ell_{k+1}](t_{k+1})$
- 7.8.2 $sol_{k+1} = \begin{pmatrix} P_{0,k+1} \bullet Q_{k+1} \bullet \Pi_{k+1}, \\ \Delta[sol_{k+1}](t_o) = t_{k+1} - t_o + R_{k+1} \\ \quad + \Delta[\ell_{k+1}, d](t_{k+1} + R_{k+1}) \end{pmatrix}$ /* approximate answer via ℓ_{k+1} */
- 7.9. $k = k + 1$
8. **endwhile**
9. **return** $\min_{0 \le k \le r} \{sol_k\}$

Fig. 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

vertices in the graph, it is clear that **RQA** always returns an od-path whose travel-time does not exceed the alleged upper bound on the actual distance.

Our next task is to study the quality of the $1+\sigma$ stretch provided by **RQA**. Let $\delta > 0$ be a parameter such that $\sigma = \varepsilon + \delta$ and recall the definition of ψ from Theorem 3.

The next lemma shows that the sequence of ball radii grown from vertices of the shortest *od*-path $P[o,d](t_o)$ by the recursive calls of **RQA** is lower-bounded by a geometrically increasing sequence.

Lemma 5 *Let $D[o,d](t_o) = t_d - t_o$ and suppose that* **RQA** *does not discover d or any landmark $w_k \in SP[o,d](t_o) \cap L$, $k \in \{0,1,\ldots,r\}$, in the explored area around o. Then, the entire recursion budget r will be consumed and in each round k of recursively constructed balls we have that either $R_k > \left(1+\frac{\varepsilon}{\psi}\right)^k \cdot \frac{\delta}{\psi} \cdot (t_d - t_o)$ or $\exists\, i \in \{0,1,\ldots,k\} : D[sol_i](t_o) \leq (1+\varepsilon+\delta) \cdot D[o,d](t_o)$.*

Proof As 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.

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:

$$\begin{aligned} D[sol_0](t_o) &\leq R_0 + \Delta[\ell_0,d](t_o + R_0) \\ &\overset{/*\ \text{Theorem 3}\ */}{\leq} (1+\varepsilon)\cdot D[o,d](t_o) + \psi \cdot R_0 = \left(1+\varepsilon+\tfrac{\psi R_0}{t_d - t_o}\right)\cdot(t_d - t_o) \end{aligned}$$

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 it 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 $sol_{k+1} = P_{0,k+1} \bullet Q_{k+1} \bullet \Pi_{k+1}$ 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:

$$\begin{aligned} &D[sol_{k+1}](t_o) \\ &\quad\leq\quad t_{k+1} - t_o + R_{k+1} + \Delta[\ell_{k+1},d](t_{k+1} + R_{k+1}) \\ &\overset{/*\ \text{Theorem 3}\ */}{\leq} \quad t_{k+1} - t_o + (1+\varepsilon)\cdot D[w_{k+1},d](t_{k+1}) + \psi \cdot R_{k+1} \\ &\overset{/*\ w_{k+1} \in SP[o,d](t_o)\ */}{=} \quad t_{k+1} - t_o + (1+\varepsilon)\cdot(t_d - t_{k+1}) + \psi \cdot R_{k+1} \\ &\quad=\quad (1+\varepsilon)\cdot(t_d - t_o) - \varepsilon\cdot(t_{k+1} - t_o) + \psi \cdot R_{k+1} \end{aligned}$$

$$
\begin{aligned}
&\overset{/*\ t_{k+1}-t_o \geq R_0+\ldots+R_k\ */}{\leq} && (1+\varepsilon)\cdot(t_d-t_o)-\varepsilon\cdot(R_0+\ldots+R_k)+\psi\cdot R_{k+1}\\
&\overset{/*\ \text{Induction Hypothesis}\ */}{<} && (1+\varepsilon)(t_d-t_o)-\varepsilon\sum_{i=0}^{k}\left(1+\frac{\varepsilon}{\psi}\right)^i\frac{\delta}{\psi}(t_d-t_o)+\psi R_{k+1}\\
&= && \left(1+\varepsilon-\frac{\varepsilon\delta}{\psi}\cdot\sum_{i=0}^{k}\left(1+\frac{\varepsilon}{\psi}\right)^i+\frac{\psi\cdot R_{k+1}}{t_d-t_o}\right)\cdot(t_d-t_o)\\
&= && \left(1+\varepsilon-\delta\cdot\left[\left(1+\frac{\varepsilon}{\psi}\right)^{k+1}-1\right]+\frac{\psi\cdot R_{k+1}}{t_d-t_o}\right)\cdot(t_d-t_o)
\end{aligned}
$$

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. □

The next theorem shows that **RQA** indeed provides $(1+\sigma)$-approximate distances in response to arbitrary queries $(o,d,t_o)\in V\times V\times[0,T)$.

Theorem 5 *For the stretch of* **RQA** *the following hold:*

1. *If* $r=\left\lceil\frac{\ln\left(1+\frac{\varepsilon}{\delta}\right)}{\ln\left(1+\frac{\varepsilon}{\psi}\right)}\right\rceil-1$ *for* $\delta>0$, *then,* **RQA** *guarantees a stretch* $1+\sigma=1+\varepsilon+\delta$.
2. *For a given recursion budget* $r\in\mathbb{N}$, **RQA** *guarantees stretch* $1+\sigma$, *where* $\sigma=\sigma(r)\leq\varepsilon\cdot\frac{(1+\varepsilon/\psi)^{r+1}}{(1+\varepsilon/\psi)^{r+1}-1}$.

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

$$
\begin{aligned}
t_d-t_o\geq R_0+R_1+\cdots+R_r &\overset{/*\ \text{Lemma 5}\ */}{>} \frac{\delta}{\psi}\cdot(t_d-t_o)\cdot\sum_{i=0}^{r}\left(1+\frac{\varepsilon}{\psi}\right)^i\\
&=\frac{\delta}{\psi}\cdot(t_d-t_o)\cdot\frac{\left(1+\frac{\varepsilon}{\psi}\right)^{r+1}-1}{1+\frac{\varepsilon}{\psi}-1}=\frac{\delta}{\varepsilon}\cdot(t_d-t_o)\cdot\left[\left(1+\frac{\varepsilon}{\psi}\right)^{r+1}-1\right]\\
\Rightarrow\frac{\varepsilon}{\delta}>\left(1+\frac{\varepsilon}{\psi}\right)^{r+1}-1 &\Rightarrow\begin{cases}r<\frac{\ln(1+\varepsilon/\delta)}{\ln(1+\varepsilon/\psi)}-1\\ \delta<\frac{\varepsilon}{(1+\varepsilon/\psi)^{r+1}-1}\end{cases}
\end{aligned}
$$

If $r=\left\lceil\frac{\ln(1+\varepsilon/\delta)}{\ln(1+\varepsilon/\psi)}-1\right\rceil\leq\frac{\psi/\delta}{1-\varepsilon/\psi}-1\in\mathcal{O}\left(\frac{\psi}{\delta}\right)$, we have reached a contradiction.[4] 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.

On the other hand, for a given recursion budget $r\in\mathbb{N}$, it holds that $\sigma=\sigma(r)=\varepsilon+\frac{\varepsilon}{(1+\varepsilon/\psi)^{r+1}-1}=\frac{\varepsilon\cdot(1+\varepsilon/\psi)^{r+1}}{(1+\varepsilon/\psi)^{r+1}-1}$ is guaranteed by **RQA**. □

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

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 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 [2] to achieve 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 [2] 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. 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. It is noted that we have recently become aware of an improved bidirectional approximate distance oracle for static undirected graphs [1] which outperforms [2] in the stretch-time-space tradeoff.

5.2 Complexity of RQA

It only remains to determine the query-time complexity $\mathcal{Q}_{RQA}$ of **RQA**. This is provided by the following theorem.

Theorem 6 *For the query-time complexity $\mathcal{Q}_{RQA}$ of* **RQA** *the following hold:*

$$\mathbb{E}\left\{\mathcal{Q}_{RQA}\right\} \in O((1/\rho)^{r+1} \cdot \ln(1/\rho) \cdot \log\log(K_{\max})).$$

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

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

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. Fig. 6) via exhaustive search, would be bounded by a complete $(2\beta^*)$-ary tree of depth r. 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:

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

Assuming that $\log\log((K^*+1)\cdot U) \in \mathcal{O}(\beta^*\log(\beta^*))$, we have that $\mathcal{Q}_{RQA} \in \mathcal{O}\big((\beta^*)^{r+1}\ln(\beta^*)\big)$. 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 $\mathbb{E}\{\mathcal{Q}_{RQA}\} \in \mathcal{O}\left(\left(\frac{1}{\rho}\right)^{r+1}\ln\left(\frac{1}{\rho}\right)\right)$.

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

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

Thus, we conclude that:

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

□

6 Main Results

In this section, we summarize the main result of our paper and establish the tradeoff between preprocessing, query time and stretch. Recall that U is the worst-case number of breakpoints for an $(1+\varepsilon)$-approximation of a *concave* $(1+\varepsilon)$-approximate distance function stored in our oracle, and TDP is the maximum number of time-dependent shortest path probes during their construction.[5] The following theorem summarizes our main result.

[5] As it is proved in Theorem 1, U and TDP are independent of the network size n.

Theorem 7 *For sparse time-dependent network instances compliant with Assumptions 1 and 2, a distance oracle is provided with the following characteristics: (a) it selects among all vertices, uniformly and independently with probability ρ, a set of landmarks; (b) it stores $(1+\varepsilon)$-approximate distance functions (summaries) from every landmark to all other vertices; (c) it uses a query algorithm equipped with a recursion budget (depth) r. Our time-dependent distance oracle achieves the following expected complexities:*

(i) *preprocessing space* $\mathcal{O}\big(\rho n^2(1+K^*)U\big)$;
(ii) *preprocessing time* $\mathcal{O}\big(\rho n^2(1+K^*)\log(n)\log\log(K_{\max})TDP\big)$;
(iii) *query time* $\mathcal{O}\left(\left(\frac{1}{\rho}\right)^{r+1}\log\left(\frac{1}{\rho}\right)\log\log(K_{\max})\right)$.

The guaranteed stretch is $1+\varepsilon\frac{(1+\frac{\varepsilon}{\psi})^{r+1}}{(1+\frac{\varepsilon}{\psi})^{r+1}-1}$, *where ψ is a fixed constant depending on the characteristics of the arc-travel-time functions, but is independent of the network size.*

Proof Immediate consequence of Theorems 2, 5, and 6. □

Note that, apart from the choice of landmarks, our preprocessing and query algorithms are deterministic. The following theorem expresses explicitly the tradeoff between subquadratic preprocessing, sublinear query time and stretch of the proposed oracle.

Theorem 8 *Let $G=(V,A,(D[a])_{a\in A})$ be a sparse time-dependent network instance compliant with Assumptions 1 and 2. Assume that our distance oracle is deployed on G for: (a) creating the landmark set uniformly at random with probability $\rho=n^{-a}$, for some $a\in\left(0,\frac{1}{r+1}\right)$; (b) computing with the* **BISECTION** *method the $(1+\varepsilon)$-approximate distance summaries for landmark-to-vertex distances; (c) running* **RQA** *to respond to arbitrary queries $(o,d,t_o)\in V\times V\times[0,T)$, with approximation guarantee $1+\gamma\varepsilon$, for some constant $\gamma=\frac{(1+\varepsilon/\psi)^{r+1}}{(1+\varepsilon/\psi)^{r+1}-1}>1$. Provided that the degree of disconcavity of G is $K^*\in\text{polylog}(n)$, then the following expected complexities hold:*

$$\begin{aligned} \textit{Preprocessing space:} &\qquad \mathbb{E}\{S\}\in\tilde{\mathcal{O}}\left(n^{2-a}\right)\\ \textit{Preprocessing time:} &\qquad \mathbb{E}\{P\}\in\tilde{\mathcal{O}}\left(n^{2-a}\right)\\ \textit{Query time:} &\qquad \mathbb{E}\{Q\}\in\tilde{\mathcal{O}}\left(n^{(1+r)a}\right)\end{aligned}$$

Proof Since we have assumed that $K^*\in\text{polylog}(n)$ and since by Theorem 1 U and TDP are independent of the network size n and hence can be treated as constants, it follows from Theorem 7 that the expected preprocessing space and time complexities, $\mathbb{E}\{S\}\in\tilde{\mathcal{O}}\left(n^{2-a}\right)$ and $\mathbb{E}\{P\}\in\tilde{\mathcal{O}}\left(n^{2-a}\right)$, are indeed subquadratic. It similarly follows from the same theorem that the expected query time is $\mathbb{E}\left\{Q_{\mathbf{RQA}}\right\}\in\tilde{\mathcal{O}}\left(n^{(1+r)a}\right)$. Hence, to complete the proof is remains to show that $\tilde{\mathcal{O}}\left(n^{(1+r)a}\right)$ is also sublinear, i.e.,

$(r+1)a < 1$. Recall that, by Theorems 7 and 5, a $(1+\gamma\varepsilon)$-approximate solution is returned, for $\gamma = \frac{(1+\varepsilon/\psi)^{r+1}}{(1+\varepsilon/\psi)^{r+1}-1}$ which holds by our assumption. Alternatively, to assure a desired approximation guarantee $1+\gamma\varepsilon$ for arbitrary queries, value $\gamma > 1$ and a given approximation guarantee $1+\varepsilon$ for the preprocessed distance summaries, we should set appropriately the recursion budget to

$$r = \frac{\log\left(\frac{\gamma}{\gamma-1}\right)}{\log\left(1+\frac{\varepsilon}{\psi}\right)} - 1$$

□

7 Approximate Shortest Path Reconstruction

As it is customary in the distance oracles literature, the query-time complexities of our algorithms concern only the determination, for a given query $(o, d, t_o) \in V \times V \times [0, T)$, of an upper bound $\Delta[o, d](t_o)$ on the shortest-travel-time $D[o, d](t_o)$, or equivalently an arrival-time $\tau_d := t_o + \Delta[o, d](t_o)$ at d which guarantees this upper bound on the travel-time.

Our goal in this section is to describe a method for reconstructing an actual *od*-path (roughly) guaranteeing this travel-time bound, in time (additional to the already reported query-time) that is roughly linear in the number of its constituent arcs. Indeed, our goal is only to exploit the precomputed landmarks-to-vertices approximate distance summaries, along with the value τ_d that was computed on the fly, in order to discover such a path. Indeed, the origin-to-landmark path is computed "on-the-fly" and the main challenge is to construct the remaining landmark-to-destination approximate path that would guarantee the reported arrival-time at the destination. A natural approach would be to mimick the path reconstruction from the destination back to the landmark, based only on the (upper bound on the) arrival-time at the destination, as is typically done in the time-independent case. This would indeed be possible, if we had at our disposal *exact* landmark-to-vertices distance summaries. But we can only afford for $(1+\varepsilon)$-approximate distance summaries of the actual travel-time functions and thus the only thing we know is that $\tau_d \in t_o + D[o, d](t_o) \cdot [1, 1+\varepsilon]$. Thus, we cannot be sure that such a reconstruction is indeed possible: It might be the case that $\tau_d = t_d := t_o + D[o, d](t_o)$ while at the same time some of the approximate distances from the landmark to intermediate vertices along the path are indeed inexact.

To resolve this issue, we shall exploit the fact that the approximate distance summaries created during preprocessing, correspond to travel-time functions along a shortest-paths tree from the landmark to all possible destinations, for the given departure time. This tree is actually a valid *approximate* shortest paths tree, not only for the sampled departure time, but also for the entire time-interval of departures till the next sample point. Due to the sparsity of the graph, we can be sure that only a constant number of bits is required per breakpoint in the pwl-approximations, in order for each vertex to memoize its own parent in such a tree (as a function of the departure-time from the landmark). The path reconstruction is then conducted by moving from

the destination towards the landmark, evaluating the right leg of the corresponding approximate distance summary in each intermediate vertex, so that the appropriate parent (and the latest departure time from it) is selected. In overall, the construction time will be almost linear in the number of arcs constituting the required approximate shortest path (times an $\mathcal{O}(\log\log(K_{\max}))$ factor, for evaluating the right leg in the approximate distance summary of each intermediate vertex).

8 Conclusions

We have presented the first time-dependent distance oracle for sparse networks, compliant with Assumptions 1 and 2, that achieves subquadratic preprocessing space and time, sublinear query time, and stretch factor arbitrarily close to 1. Our approach is based on a new algorithm, built upon the bisection method, that computes one-to-all approximate distance summaries from a set of selected landmarks to all other vertices of the network as well as on a new recursive query algorithm. Our assumptions, justified by an experimental analysis of real-world and benchmark data, allow us to achieve a smooth transition, from the undirected (symmetric) and static world to the directed (asymmetric) and time-dependent world, through two parameters that quantify the degree of asymmetry (ζ) and its evolution over time (expressing the steepness of the shortest travel-time functions via $\Lambda_{\min}$ and $\Lambda_{\max}$).

It would be quite interesting to come up with a new method for computing approximate distance summaries, that avoids the dependence of the preprocessing complexities on the number K^* of concavity-spoiling breakpoints.

Finally, almost all distance oracles with provable approximation guarantees in the literature, even for the static case, target at sublinearity in query times with respect to the network size. A very important aspect would be to propose query algorithms that are indeed sublinear not only in worst-case, but also sublinear on the Dijkstra rank of the destination vertex.

References

1. Agarwal, R.: The space-stretch-time trade-off in distance oracles. In: ESA (2014), to appear
2. Agarwal, R., Godfrey, P.: Distance oracles for stretch less than 2. In: SODA 2013, pp. 526–538. ACM-SIAM, (2013)
3. Akrida, E., Gasieniec, L., Mertzios, G., Spirakis, P.: Ephemeral networks with random availability of links: diameter and connectivity. In: 26th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '14), pp. 267–276, (2014)
4. Batz, G.V., Geisberger, R., Sanders, P., Vetter, C.: Minimum time-dependent travel times with contraction hierarchies. ACM J. Exper. Algorithm. **18**, 1–43 (2013)
5. Bertsekas, D., Tsitsiklis, J.: An analysis of stochastic shortest path problems. Math. Oper. Res. **16**(3), 580–595 (1991)
6. Cooke, K., Halsey, E.: The shortest route through a network with time-dependent intermodal transit times. J. Math. Anal. Appl. **14**(3), 493–498 (1966)
7. Dean, B.C.: Continuous-time dynamic shortest path algorithms. MSc thesis. MIT, (1999)
8. Dean, B.C.: Algorithms for minimum-cost paths in time-dependent networks with waiting policies. Networks **44**(1), 41–46 (2004)
9. Dean, B.C.: Shortest paths in FIFO time-dependent networks: theory and algorithms. Technical report, MIT (2004)

10. Dehne, F., Masoud, O.T., Sack, J.R.: Shortest paths in time-dependent fifo networks. Algorithmica **62**(1–2), 416–435 (2012)
11. Delling, D.: Time-dependent SHARC-routing. Algorithmica **60**(1), 60–94 (2011)
12. Delling, D., Wagner, D.: Time-Dependent Route Planning. In: Ahuja, R.K., Möhring, R.H., Zaroliagis, C. (eds.) Robust and Online Large-Scale Optimization, LNCS 5868, pp. 207–230. Springer, Berlin (2009)
13. Demetrescu, C., Italiano, G.F.: Dynamic shortest paths and transitive closure: an annotated bibliography (draft), (2005). http://www.diku.dk/PATH05/biblio-dynpaths
14. Dreyfus, S.E.: An appraisal of some shortest-path algorithms. Oper. Res. **17**(3), 395–412 (1969)
15. eCOMPASS Project (2011–2014). http://www.ecompass-project.eu
16. Foschini, L., Hershberger, J., Suri, S.: On the complexity of time-dependent shortest paths. Algorithmica **68**(4), 1075–1097 (2014). Prelim. version in SODA 2011
17. Gusfield, D.M.: Sensitivity analysis for combinatorial optimization. Phd thesis, University of California, Berkeley, (1980)
18. Kempe, D., Kleinberg, J., Kumar, A.: Connectivity and inference problems for temporal networks. J. Comput. Syst. Sci. **64**, 820–842 (2002)
19. King, V.: Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In: Proceedings of 40th Annual IEEE Symposium on Foundations of Computer Science, pp. 81–89, (1999)
20. Kontogiannis, S., Zaroliagis, C.: Distance oracles for time dependent networks. In: Automata, Languages and Programming (Track A). Lecture Notes in Computer Science 8572, pp. 713–725. Springer (2014)
21. Mertzios, G., Michail, O., Chatzigiannakis, I., Spirakis, P.: Temporal Network Optimization Subject to Connectivity Constraints In: Automata, Languages and Programming (Track C). Lecture Notes in Computer Science 7966, pp. 657–668. Springer (2013)
22. Mulmuley, K., Shah, P.: A lower bound for the shortest path problem. J. Compt. Syst. Sci. **63**, 253–267 (2001)
23. Nannicini, G., Delling, D., Liberti, L., Schultes, D.: Bidirectional A* search on time-dependent road networks. Networks **59**, 240–251 (2012)
24. Nikolova, E., Brand, M., Karger, D.: Optimal route planning under uncertainty. In: International Conference on Automated Planning and Scheduling, (2006)
25. Nikolova, E., Kelner, J., Brand, M., Mitzenmacher, M.: Stochastic shortest paths via quasi-convex maximization. In: 14th European Symposium on Algorithms, pp. 552–563, (2006)
26. Orda, A., Rom, R.: Shortest-path and minimum delay algorithms in networks with time-dependent edge-length. J. ACM **37**(3), 607–625 (1990)
27. Patrascu, M., Roditty, L.: cDistance oracles beyond the Thorup-Zwick bound. In: IEEE Symposium on Foundations of Computer Science, pp. 815–823, (2010)
28. Porat, E., Roditty, L.: Preprocess, set, query! In: European Symposium on Algorithms, LNCS 6942, pp. 603–614. Springer, (2011)
29. Roditty, L., Zwick, U.: On Dynamic Shortest Paths Problems. In: 12th Annual European Symposium on Algorithms. Lecture Notes in Computer Science 3221, pp. 580–591. Springer (2004)
30. Sherali, H., Ozbay, K., Subramanian, S.: The time-dependent shortest pair of disjoint paths problem: complexity, models, and algorithms. Networks **31**(4), 259–272 (1998)
31. Sommer, C.: Shortest-path queries in static networks. ACM Comput. Surv. **46**, 1–31 (2014)
32. Sommer, C., Verbin, E., Yu, W.: Distance oracles for sparse graphs. In: IEEE Symposium on Foundations of Computer Science, pp. 703–712, (2009)
33. Thorup, M.: Worst-case update times for fully-dynamic all-pairs shortest paths. In: 37th Annual ACM Symposium on Theory of Computing, pp. 112–119, (2005)
34. Thorup, M., Zwick, U.: Approximate distance oracles. J. ACM **52**, 1–24 (2005)
35. Wulff-Nilsen, C.: Approximate distance oracles with improved preprocessing time. In: SODA 2012. ACM-SIAM, (2012)
36. Wulff-Nilsen, C.: Approximate distance oracles with improved query time, (2012). arXiv:1202.2336