Distance Oracles for Time-Dependent Networks*

Spyros Kontogiannis[1,2] and Christos Zaroliagis[2,3]

[1] Dept. of Comp. Science & Engineering, U. Ioannina, 45110 Ioannina, Greece
kontog@cs.uoi.gr

[2] Computer Technology Institute & Press "Diophantus", 26504 Patras, Greece

[3] Dept. of Comp. Engineering & Informatics, U. Patras, 26504 Patras, Greece
zaro@ceid.upatras.gr

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, 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. 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.

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., [22,21,17,18,23,24,2]) 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 deferred to [20] and references therein.

In many real-world applications, however, the arc costs may vary as functions of time (e.g., when representing travel-times) giving rise to *time-dependent* network models. A striking example is route planning in road networks where the

* Full version available at [14]. This work was supported by EU FP7/2007-2013 under grant agreements no. 288094 (eCOMPASS) and no. 609026 (MOVESMART), and partially done while both authors were visiting the Karlsruhe Inst. of Technology.

J. Esparza et al. (Eds.): ICALP 2014, Part I, LNCS 8572, pp. 713–725, 2014.

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. For example, TomTom's *LiveTraffic* service 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 connected cars to the service as sampling devices. The crux is how to exploit all this historic traffic information in order to provide *efficiently* route plans that will adapt to the departure-time from the origin. Towards this direction, we 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 long time ago (see e.g., [4,11,16]). The shape of arc-travel-time functions and the waiting policy at vertices may considerably affect the tractability of the problem [16]. 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. Thus, (even if it exists) an optimal route is not computable by well known techniques (Dijkstra or Bellman-Ford) [16]. Additionally, many variants of the problem are also **NP**−hard [19]. 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 [11], 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: 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. 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 useless [16]. Therefore, our focus in this work is on networks with FIFO arc-travel-time functions.

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 [5,7,6], but was solved only recently in [13] 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* approximate distance functions are provided in [13,8]. 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 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 [13]) 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)[1]. 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., *sub-linear*) query-time complexity, while consuming smaller space-complexity (e.g., $\mathrm{o}\big((1 + K^*) \cdot n^2\big)$) for succinctly representing travel-time *functions*, and enjoying a small (e.g., close to 1) approximation guarantee.

We present 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

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

(cf. Assumptions 1 and 2 in Section 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 $m \in o(n^2)$, as it has been recently shown [18,2]; (ii) there is important applied work [10,3,9,15] 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., [22,18,2]): Distance summaries from selected landmarks are precomputed and stored; fast responses to arbitrary real-time queries are provided 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.

Let U be the worst-case number of breakpoints for an $(1+\varepsilon)-$approximation of a *concave* distance function stored in our oracle, and TDP be the maximum number of time-dependent shortest path probes during their construction[2]. The following theorem summarizes our results.

Theorem 1. *For time-dependent instances compliant with Assumptions 1 and 2, a distance oracle is provided storing* $(1+\varepsilon)-$*approximate distance functions from landmarks, which are uniformly and independently selected with probability* ρ*, to all other vertices, and uses a recursion depth (budget)* r *in the query algorithm, guaranteeing expected values of: (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.*

Note that, apart from the choice of landmarks, our algorithms are deterministic. Due to space limitations, proofs and a table with solid examples of the oracle's space/query-time/stretch trade-offs can be found in the full version [14].

[2] As proved in [14], U and TDP are independent of the network size n.

2 Ingredients and Overview of Our Approach

Our input is provided by a directed graph $G = (V, A)$ with n vertices and m 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. 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.

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:

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

The FIFO property is *strict*, if the above inequality is strict. The FIFO property implies that: (i) the slope of any arc-delay function is greater than -1; (ii) the slope of any path-delay or shortest-travel-time function is greater than -1. The *strict* FIFO property implies *subpath optimality* of shortest paths. For formal statements and proofs of these facts, see [14].

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 make two assumptions on the kind of shortest-travel-time functions in the network. Both assumptions are quite natural and justified by a thorough investigation of historic traffic data for the city of Berlin, kindly provided to us by TomTom [12] (see [14] for a more detailed justification). 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 $\Lambda_{\min} > -1$ *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 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:* $\forall (o,d) \in V \times V,\ \forall t \in [0,T),\ D[o,d](t) \leq \zeta \cdot D[d,o](t)$.

As we show in Section 4, the parameters $\Lambda_{\max}$ and ζ allow us to quantify the degree of asymmetry and evolution in time in our distance metric and achieve the aforementioned smooth transition. Another assumption we make and which can be easily guaranteed is that the maximum out-degree is bounded by 2.

Overview of Our Approach. We follow (at a high level) the typical approach adopted for the construction of approximate distance oracles in the static case. In particular, we start by selecting a subset $L \subset V$ of *landmarks*, i.e., vertices which will act as reference points for our distance summaries. For our oracle to work, several ways to choose L would be acceptable. Nevertheless, for the sake of the analysis we assume that this is done by deciding for each vertex randomly and independently with probability $\rho \in (0,1)$ whether it belongs to L. After having L fixed, our approach is deterministic. We start by constructing (concurrently, per landmark) and storing the *distance summaries*, i.e., all landmark-to-vertex $(1+\varepsilon)-$approximate travel-time functions, in time $\mathrm{o}\big((1+K^*)n^2\big)$ and consuming space $\mathrm{o}\big((1+K^*)n^2\big)$ which is indeed asymptotically optimal w.r.t. the required approximation guarantee (cf. Section 3). Then, we provide two approximation algorithms for 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. Section 4). The second (**RQA**) is a recursive algorithm growing small **TDD** outgoing balls from vertices in the vicinity of the origin, until either a satisfactory approximation guarantee is achieved, or an upper bound r on the 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. Section 4). 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.

3 Preprocessing Distance Summaries

We now demonstrate how to construct the preprocessed information that will comprise the *distance summaries* of the oracle, i.e., all landmark-to-vertex shortest-travel-time functions. 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 variant of **TDD** with root (u, t_u) on the *reverse network* $(\overleftarrow{G} = (V, A, (\overleftarrow{D}[a])_{a \in A})$, where $\overleftarrow{D}[uv]$ is the delay of arc uv, measured now as a function of the arrival-time t_v at the tail 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. For each landmark, we repeat the procedure described in the rest of this section for every $K^* + 1$ subinterval of $[0, T)$ determined by *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 $\Delta[\ell, v] : [0, T) \to \mathbb{R}_{>0}$ of the shortest-travel-time functions $D[\ell, v] : [0, T) \to \mathbb{R}_{>0}$. An algorithm providing such functions in a *point-to-point* fashion was proposed in [13]. 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 [13] in two 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 [13]; (ii) within every subinterval of consecutive images of concavity-spoiling breakpoints, it provides asymptotically optimal space per landmark, which is also independent of the network size per landmark-vertex pair, implying that the required preprocessing space per vertex is $\mathcal{O}(|L|)$. This is also claimed in [13], but it is actually true only for their second phase (the bisection). For the first phase of their algorithm, there is no such guarantee. Even for a linear arc-travel-time function, the first phase of that algorithm would still require a number of samples which is logarithmic in the max-to-min travel-time ratio.

Our algorithm, in order to achieve a concurrent one-to-all construction of upper-bounding approximations from a given landmark $\ell \in L$, 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 [13], in order to discard all unnecessary sampling points from being actual breakpoints in the approximate functions.

In a nutshell, we construct two continuous pwl-approximations of the *unknown* shortest-travel-time function $D[\ell, v] : [0, T) \to \mathbb{R}_{>0}$, an upper-bounding approximate function $\overline{D}[\ell, v]$ (playing the role of $\Delta[\ell, v]$) and a lower-bounding approximate function $\underline{D}[\ell, v]$. Our construction guarantees that the exact function is always "sandwiched" between these two approximations. For a given landmark $\ell \in L$ and a subinterval $[t_s, t_f) \subseteq [0, T)$ of departure times from ℓ, in which all the (unknown) shortest-travel-time functions from ℓ are concave, the algorithm proceeds as follows (details are provided in [14]): The current subinterval $[t_s, t_f)$ is bisected in the middle $t_m = \frac{t_s + t_f}{2}$. The result of this bisection is for the lower-approximating function $\underline{D}[\ell, v]$ to be augmented by the new breakpoint t_m, for all still *active* (having not yet met their required approximation guarantee) destination vertices v w.r.t. $[t_s, t_f)$. Our next step is, for each $v \in V$, to check whether the upper-approximating function $\overline{D}[\ell, v]$, consisting of the lower-envelope of the tangents of $D[\ell, v]$ at t_s, t_m and t_f, i.e., at most five breakpoints for the subinterval $[t_s, t_f)$, is already a $(1 + \varepsilon)-$approximation of $\underline{D}[\ell, v]$ within $[t_s, t_m)$ and $[t_m, t_f)$. Each destination vertex that is already satisfied by the current approximation becomes *inactive* for the subsequent subintervals. If any of the two subintervals still has active destination nodes, it is recursively bisected.

$L[\ell, v]$ and $U[\ell, v]$ denote the numbers of breakpoints for $\underline{D}[\ell, v]$ and $\overline{D}[\ell, v]$, $U = \max_{\ell, v}\{U[\ell, v]\}$, and TDP is the number of shortest-path probes during a bisection. By construction it holds that $U[\ell, v] \leq 2 \cdot L[\ell, v]$ (for an explanation see [14]). The expected number of landmarks is $\mathbb{E}\{|L|\} = \rho n$. It is then easy to deduce the required time and space complexity of our entire preprocessing.

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

U and TDP are independent of n (cf. [14]), so we treat them as constants. If all arc-travel-time functions are concave, i.e., $K^* = 0$, then we achieve subquadratic preprocessing space and time $\forall \rho \in \mathcal{O}(n^{-\alpha})$, where $0 < \alpha < 1$. Real data (e.g., TomTom's traffic data for the city of Berlin [12]) demonstrate that: (i) only a small fraction of the arc-travel-time functions exhibit non-constant behaviour; (ii) for the vast majority of these non-constant-delay arcs, their functions are either concave, or can be very tightly approximated by a typical *concave* bell-shaped pwl function. It is only a tiny subset of critical arcs

(e.g., bottleneck road segments) for which it would be meaningful to consider non-concave behaviour. Therefore, $K^* \in \mathrm{o}(n)$ is the typical case. E.g., assuming $K^* \in \mathcal{O}(\mathrm{polylog}(n))$, we can fine-tune ρ and the parameters σ, r (cf. Section 4) so as to achieve *subquadratic* preprocessing space and time. In particular, for $K^* \in \mathcal{O}(\log(n))$ and $K_{\max} \in \mathcal{O}(1)$, $\forall \gamma > \frac{1}{2}$, $\mathbb{E}\{\mathcal{S}\} \in \mathcal{O}\big(n^{2-\varepsilon/(\gamma\psi)} \log(n)\big)$ and $\mathbb{E}\{\mathcal{P}\} \in \mathcal{O}\big(n^{2-\varepsilon/(\gamma\psi)} \log^2(n)\big)$, where $\psi = \psi(\zeta, \Lambda_{\max})$ is a constant that will be specified in Theorem 3. More details are provided in [14].

4 Query Algorithms

Constant-Approximation Query Algorithm. Our next step towards a distance oracle is to provide a fast query algorithm providing constant approximations to the actual shortest-travel-time values of arbitrary queries $(o, d, t_o) \in V \times V \times [0, T)$. Here we propose such a query algorithm, called *Forward Constant Approximation* (**FCA**), which grows an outgoing ball $B_o \equiv B[o](t_o) = \{x \in V : D[o, x](t_o) < D[o, \ell_o](t_o)\}$ around (o, t_o) by running **TDD**, until either d or the closest landmark $\ell_o \in \arg\min_{\ell \in L}\{D[o, \ell](t_o)\}$ is scanned. We call $R_o = D[o, \ell_o](t_o)$ the *radius* of B_o. **FCA** returns either the exact travel-time value, or the approximate travel-time value via ℓ_o. Figure 1 gives an overview of the whole idea. The pseudocode is provided in [14].

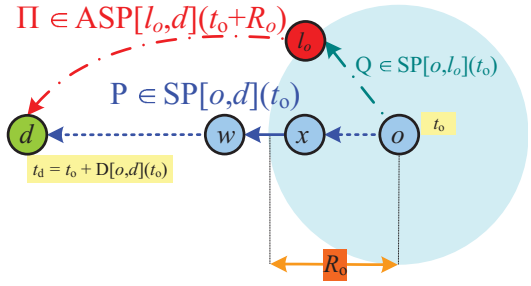

Fig. 1. The rationale of **FCA**. The dashed (blue) path is a shortest $od-$path for query (o, d, t_o). The dashed-dotted (green and red) path is the via-landmark $od-$path indicated by the algorithm, if the destination vertex is out of the origin's **TDD** ball.

Correctness. 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* $\psi = 1{+}\Lambda_{\max}(1{+}\varepsilon)(1{+}2\zeta{+}\Lambda_{\max}\zeta){+}(1{+}\varepsilon)\zeta$.

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 ζ and $\Lambda_{\max}$, towards both *asymmetry* and *time-dependence* of the travel-time metric.

Complexity. 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. L is chosen randomly by selecting each vertex v to become a landmark independently of other vertices, with probability $\rho \in (0,1)$. Clearly $\mathbb{E}\{|B_o|\} = 1/\rho$, and moreover (as a geometrically distributed random variable), $\forall k \geq 1$, $\mathbb{P}\{|B_o| > k\} = (1-\rho)^k \leq e^{-\rho k}$. By setting $k = (1/\rho)\ln(1/\rho)$ we conclude that: $\mathbb{P}\{|B_o| > (1/\rho)\ln(1/\rho)\} \leq \rho$. Since the maximum out-degree is 2, **TDD** will relax at most $2k$ arcs. Hence, for the query-time complexity $\mathcal{Q}_{FCA}$ of **FCA** we conclude that $\mathbb{E}\{\mathcal{Q}_{FCA}\} \in \mathcal{O}((1/\rho)\ln(1/\rho)\log\log(K_{\max}))$, and $\mathbb{P}\left\{\mathcal{Q}_{FCA} \in \Omega\big((1/\rho)\ln^2(1/\rho)\log\log(K_{\max})\big)\right\} \in \mathcal{O}(\rho)$.

$(1+\sigma)-$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 (called the *recursion budget*) of recursive accesses to the preprocessed information, each of which produces (via a call to **FCA**) another candidate $od-$path sol_i. The crux of our approach is the following: We assure that, unless the required approximation guarantee has already been reached by a candidate solution, the recursion budget must be exhausted and the sequence of radii of the consecutive balls that we grow recursively is lower-bounded by a *geometrically increasing* sequence. We prove that this sequence can only have a *constant* number of elements, 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: So long as we have not discovered the destination node within the explored area around the origin, and there is still some remaining recursion budget, we "guess" (by exhaustively searching for it) the next node w_k along the (unknown) 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 $Q_k \bullet \Pi_k$. The pseudocode is provided in [14]. Figure 2 provides an overview of **RQA**'s execution.

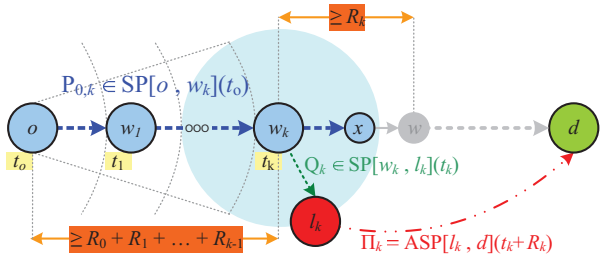

Fig. 2. Overview of the execution of **RQA**

Correctness & Quality. The correctness of **RQA** implies that the algorithm always returns some $od-$path. This is true due to the fact that it either discovers the destination 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 vertices in the graph, it is clear that **RQA** always implies 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 provided stretch $1+\sigma$ guaranteed by **RQA**. Let $\delta > 0$ be a parameter such that $\sigma = \varepsilon + \delta$ and recall the definition of ψ from Theorem 3. In [14] it is shown 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. 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 4. *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* $\sigma = \sigma(r) \leq \frac{\varepsilon \cdot (1+\varepsilon/\psi)^{r+1}}{(1+\varepsilon/\psi)^{r+1}-1}$.

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 assure the same approximation guarantee. That is, at its one extreme ($\Lambda_{\min} = \Lambda_{\max} = 0$, $\zeta = 1$, $\psi = 2$) our approach matches the bounds in [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.

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

Theorem 5. *For networks having $|A|/|V| \in \mathcal{O}(1)$, the expected running time of* **RQA** *is $\mathbb{E}\{\mathcal{Q}_{RQA}\} \in \mathcal{O}\big((1/\rho)^{r+1} \cdot \ln(1/\rho) \cdot \log\log(K_{\max})\big)$, and it holds that:*

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

Continuing the discussion in the paragraph following Theorem 2, we can fine-tune the parameters σ, r so as to achieve, along with subquadratic space and preprocessing time, sublinear query-time complexity $\mathbb{E}\{\mathcal{Q}_{RQA}\} \in \mathcal{O}\big(n^{1/(2\gamma)}\log(n)\big)$, $\forall \gamma > \frac{1}{2}$. More details (and examples) are provided in [14].

References

1. Agarwal, R.: The space-stretch-time trade-off in distance oracles (July 2013) (manuscript)
2. Agarwal, R., Godfrey, P.: Distance oracles for stretch less than 2. In: Proceedings of the 24th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA 2013), pp. 526–538. ACM-SIAM (2013)
3. Batz, G.V., Geisberger, R., Sanders, P., Vetter, C.: Minimum time-dependent travel times with contraction hierarchies. ACM Journal of Experimental Algorithmics 18 (2013)
4. Cooke, K., Halsey, E.: The shortest route through a network with time-dependent intermodal transit times. Journal of Mathematical Analysis and Applications 14(3), 493–498 (1966)
5. Dean, B.C.: Continuous-time dynamic shortest path algorithms. Master's thesis, Massachusetts Institute of Technology (1999)
6. Dean, B.C.: Algorithms for minimum-cost paths in time-dependent networks with waiting policies. Networks 44(1), 41–46 (2004)
7. Dean, B.C.: Shortest paths in fifo time-dependent networks: Theory and algorithms. Technical report, MIT (2004)
8. Dehne, F., Masoud, O.T., Sack, J.-R.: Shortest paths in time-dependent fifo networks. Algorithmica 62(1-2), 416–435 (2012)
9. Delling, D.: Time-Dependent SHARC-Routing. Algorithmica 60(1), 60–94 (2011); Special Issue: European Symposium on Algorithms (2008)
10. Delling, D., Wagner, D.: Time-Dependent Route Planning. In: Ahuja, R.K., Möhring, R.H., Zaroliagis, C.D. (eds.) Robust and Online Large-Scale Optimization. LNCS, vol. 5868, pp. 207–230. Springer, Heidelberg (2009)
11. Dreyfus, S.E.: An appraisal of some shortest-path algorithms. Operations Research 17(3), 395–412 (1969)
12. eCOMPASS Project (2011-2014), `http://www.ecompass-project.eu`
13. Foschini, L., Hershberger, J., Suri, S.: On the complexity of time-dependent shortest paths. Algorithmica 68(4), 1075–1097 (2014); Preliminary version in ACM-SIAM SODA (2011)

14. Kontogiannis, S., Zaroliagis, C.: Distance oracles for time dependent networks. eCOMPASS Technical Report (eCOMPASS-TR-025) / ArXiv Report (arXiv.org > cs > arXiv:1309.4973) (September 2013)
15. Nannicini, G., Delling, D., Liberti, L., Schultes, D.: Bidirectional A* Search on Time-Dependent Road Networks. Networks 59, 240–251 (2012)
16. Orda, A., Rom, R.: Shortest-path and minimum delay algorithms in networks with time-dependent edge-length. Journal of the ACM 37(3), 607–625 (1990)
17. Patrascu, M., Roditty, L.: Distance oracles beyond the Thorup–Zwick bound. In: Proc. of 51th IEEE Symp. on Found. of Comp. Sci. (FOCS 2010), pp. 815–823 (2010)
18. Porat, E., Roditty, L.: Preprocess, set, query! In: Demetrescu, C., Halldórsson, M.M. (eds.) ESA 2011. LNCS, vol. 6942, pp. 603–614. Springer, Heidelberg (2011)
19. Sherali, H.D., Ozbay, K., Subramanian, S.: The time-dependent shortest pair of disjoint paths problem: Complexity, Models, and Algorithms. Networks 31(4), 259–272 (1998)
20. Sommer, C.: Shortest-path queries in static networks. ACM Computing Surveys 46 (2014)
21. Sommer, C., Verbin, E., Yu, W.: Distance oracles for sparse graphs. In: Proc. of 50th IEEE Symp. on Found. of Comp. Sci. (FOCS 2009), pp. 703–712 (2009)
22. Thorup, M., Zwick, U.: Approximate distance oracles. J. of ACM 52, 1–24 (2005)
23. Wulff-Nilsen, C.: Approximate distance oracles with improved preprocessing time. In: Proc. of 23rd ACM-SIAM Symp. on Discr. Alg. (SODA 2012) (2012)
24. Wulff-Nilsen, C.: Approximate distance oracles with improved query time. arXiv abs/1202.2336 (2012)