Hierarchical Time-Dependent Oracles*

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

1 **Comp. Sci. & Eng. Dept., University of Ioannina, Greece; and Computer Technology Institute and Press "Diophantus", Greece**
kontog@cse.uoi.gr
2 **Karlsruhe Institute of Technology, Germany**
dorothea.wagner@kit.edu
3 **Comp. Eng. & Inf. Dept., University of Patras, Greece; and Computer Technology Institute and Press "Diophantus", Greece**
zaro@ceid.upatras.gr

Abstract

We study networks obeying *time-dependent* min-cost path metrics, and present novel oracles for them which *provably* achieve two unique features: (i) *subquadratic* preprocessing time and space, *independent* of the metric's amount of disconcavity; (ii) *sublinear* query time, in either the network size or the actual Dijkstra-Rank of the query at hand.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

Keywords and phrases Time-dependent shortest paths, FIFO property, Distance oracles

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.47

1 Introduction

Concurrent technological infrastructures (e.g., road networks, social networks, e-commerce platforms, energy-management systems) are typically of very large scale and impose as a routine task the computation of min-cost paths in real-time, while their characteristics usually evolve with time. The large-scale and real-time response challenges have been addressed in the last 15 years by means of a new algorithmic trend: the provision of *oracles*. That is, data structures created by appropriately selecting precomputed information (summaries) and which subsequently support query algorithms with real-time responses. The quality of an oracle is assessed by its preprocessing space and time requirements, the time-complexity of the query algorithm and the approximation guarantee (stretch). Numerous oracles have been proposed and analyzed (see e.g., [1, 21, 22, 24, 25, 27, 28, 29] and references therein) for large-scale, mostly undirected networks, accompanied by a *static* arc-cost metric. In tandem with oracles, an equally important effort (with similar characteristics) has also emerged in the last 15 years under the tag of *speedup techniques*, for approaches tailored to work extremely well in real-life instances (see e.g., [3] and references therein).

The temporality of the network characteristics is often depicted by some kind of pre-determined dependence of the metric on the actual time that each resource is used (e.g., traversal speed in road networks, packet-loss rate in IT networks, arc availability in social networks, etc). Perhaps the most typical application scenario, motivating also our work, is

* This work is partially supported by the EU FP7/2007-2013, under grant agreements no. 609026 (project MOVESMART) and no. 621133 (project HoPE), and by DFG under grant agreement no. FOR-2083.

route planning in road networks where the travel-time for traversing an arc $a = uv$ (modeling a road segment) depends on the temporal traffic conditions while traversing uv, and thus on the departure-time from its tail u. This gives rise to *time-varying* network models and to computing min-cost (a.k.a. shortest) paths in such networks. Several variants of this problem try to model time-variation of the underlying graph and/or the arc-cost metric (e.g., dynamic shortest paths, parametric shortest paths, stochastic shortest paths, temporal networks, etc). In this work we assume that the cost variation of each arc a is determined by a *continuous, piecewise linear (pwl) and periodic function* $D[a]$ of the time at which a is actually being traversed[1], as in [7, 8, 12, 20]. When providing route plans in time-dependent road networks, arc-costs are considered as *arc-travel-times*, and time-dependent shortest paths as *minimum-travel-time* paths. The goal is then to determine the cost (*minimum-travel-time*) of a shortest path from an origin o to a destination d, as a function of the *departure-time* t_o from o. Due to the time-dependence of the arc-cost metric, the actual arc-cost value of an arc $a = uv$ is unknown until the exact time $t_u \geq t_o$ at which uv starts being traversed.

Problem setting and related work. Two variants of the *time-dependent shortest path* problem have been considered in the literature: $TDSP(o, d, t_o)$ (resp. $TDSP(o, \star, t_o)$) focuses on the one-to-one (resp. one-to-all) determination of the *scalar cost* of a minimum-travel-time path to d (resp. for all d), when departing from the origin o at time t_o. $TDSP(o, d)$ (resp. $TDSP(o, \star)$) focuses on the one-to-one (resp., one-to-all) succinct representation of the time-dependent minimum-travel-time path *function*(s) $D[o, d]$ from o to d (resp. towards all reachable d), and all departure-times from o. $TDSP(o, d, t_o)$ has been studied as early as [5]. The first work on $TDSP(o, d, t_o)$ for continuous time-axis was [11] where it was proved that, if *waiting-at-nodes* is allowed unconditionally, then $TDSP(o, d, t_o)$ is solvable in quasilinear time via a time-dependent variant of Dijkstra's algorithm (we call it `TDD`), which relaxes arcs by computing the arc costs "on the fly", upon settling their tails. A more complete treatment of the continuous-time case, considering various limitations in the waiting-times at nodes of the network, was provided in [13]; an algorithm was also given for $TDSP(o, d, t_o)$, whose complexity cannot be bounded by a function of the network topology. An excellent overview of the problem is provided in [20]. Among other results, it was proved that for *affine arc-cost* functions possessing the FIFO property (according to which all the arc-cost functions have slopes at least -1), in addition to `TDD`, a time-dependent variant of the label-correcting Bellman-Ford algorithm also works. Moreover, if waiting-at-nodes is *forbidden* and the arc-costs do not preserve the FIFO property, then *subpath-optimality* of shortest paths is not necessarily preserved. In that case, many variants of the problem become NP-hard [23]. Additionally, when shortest path costs 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 [20]. For these reasons, we focus here on instances for which the FIFO property holds, as indeed is the case with most of past and recent works on $TDSP(o, d, t_o)$. The complexity of $TDSP(o, d)$ was first questioned in [6, 7] and remained open until recently, when it was proved in [12] that, even for FIFO-abiding pwl arc-cost functions and a *single* origin-destination pair (o, d), the number of breakpoints for succinctly representing $D[o, d]$ is $(1 + K) \cdot n^{\Theta(\log n)}$, where n is

[1] Major car navigator vendors provide real-time estimations of travel-time values by periodically sampling the average speed of road segments, using the cars connected to the service as sampling devices. The most customary way to represent this historic traffic data, is to consider the continuous pwl interpolants of the sample points as arc-travel-time functions of the corresponding instance.

the number of vertices and K is the number of breakpoints in all the arc-cost functions. Note that K can be substituted by the number K^* of *concavity-spoiling* breakpoints (at which the slopes increase) of the arc-cost functions. Several *output-sensitive* algorithms for the construction of $D[o,d]$ have appeared [7, 8, 12, 20], the most efficient being the ones in [8, 12]. Due to the hardness of $TDSP(o,d)$, and also since the arc-cost functions are typically only (e.g., pwl) approximations of the actual costs, it is quite natural to seek for succinct representations of approximations to $D[o,d]$, which aim at trading-off accuracy for computational effort. Several *one-to-one* $(1+\varepsilon)$-approximation algorithms for $TDSP(o,d)$ have appeared in the literature [8, 12, 19], the most successful being those provided in [19]. The first *one-to-all* $(1+\varepsilon)$-approximation algorithm for $TDSP(o,\star)$, called *bisection* (`BIS`), was given in [17]. It is based on bisecting the (common to all functions) axis of departure-times from o and considers slightly stricter assumptions than just the FIFO property for the arc-cost metric. `BIS` requires $\mathcal{O}\Big(\frac{K^*}{\varepsilon}\cdot\log^2\left(\frac{n}{\varepsilon}\right)\Big)$ calls to $TDSP(o,\star,t_o)$, assuming that the travel-time diameter is upper-bounded by the period $T=n^\alpha$, for some tuning parameter $\alpha\in(0,1)$. Note that all one-to-one approximation algorithms for $TDSP(o,d)$ [8, 12, 19] demand, in worst-case, a comparable amount of calls to $TDSP(o,\star,t_o)$, just for one *od*-pair.

Minimum-travel-time oracles for time-dependent networks (TD-oracles henceforth) had received no attention until recently [17]. A TD-oracle is an offline-precomputed data structure that allows the *efficient evaluation* of an upper-approximation $\overline{\Delta}[o,d](t_o)$ of $D[o,d](t_o)$, for *any* possible query $(o,d,t_o)\in V\times V\times\mathbb{R}_{\geq 0}$ that may appear in an online fashion. One trivial solution would be to provide a succinct representation of $\overline{\Delta}[o,d]$ for all $(o,d)\in V\times V$, for the sake of rapid evaluations in the future but at the expense of superquadratic space. Another trivial solution would be to execute `TDD` "on-the-fly" per query (o,d,t_o), at the expense of superlinear query-time. A non-trivial TD-oracle should aim to trade-off preprocessing requirements with query-times and approximation guarantees. In particular, it should precompute a data structure in *subquadratic* time and space, and also provide a query algorithm which evaluates *efficiently* (i.e., faster than `TDD`) $\overline{\Delta}[o,d](t_o)$, where $\overline{\Delta}[o,d]$ must be a *provably* good approximation of $D[o,d]$. Note that there exists important applied work (*speedup heuristics*) for computing time-dependent shortest paths (e.g., [4, 9, 10, 18]), which however provide mainly empirical evidence on the success of the adopted approaches.

The TD-oracles in [17] require $\mathcal{O}\big(n^{2-\beta}(K^*+1)\big)$ preprocessing space and time, for constant $\beta\in(0,1)$, and can answer queries (under certain conditions) in time $\mathcal{O}\big(n^\delta\big)$, for constant $\delta\in(0,1)$. When $K^*\in o(n)$, the oracles can be fine-tuned to assure query-time $o(n)$ and preprocessing requirements $o\big(n^2\big)$. An extensive experimental evaluation of those oracles on a real-world road network is provided in [14], demonstrating their practicality, at the expense, however, of large memory consumption due to the linear dependence of the preprocessing space on K^* which can be $\Omega(n)$. The main challenge addressed here is to provide TD-oracles that achieve: (i) subquadratic preprocessing requirements, *independently* of K^*; and (ii) query-times sublinear, not just in the network size n, but in the number $\Gamma[o,d](t_o)$ (a.k.a. *Dijkstra-Rank*) of settled vertices when executing $\texttt{TDD}(o,\star,t_o)$ until d is settled.

Our contributions and roadmap. We address positively the aforementioned challenge by providing: (i) A novel and remarkably simple algorithm (`TRAP`) (cf. Section 3) for constructing one-to-many $(1+\varepsilon)$-upper-approximations $\overline{\Delta}[o,d]$ (*summaries*) of minimum-travel-time functions $D[o,d]$, for all "sufficiently distant" destinations d from the origin o. `TRAP` requires $o(n)$ calls to $TDSP(o,\star,t_o)$, which is *independent* of the degree of concavity K^*. Its novelty is that it does not demand the concavity of the unknown function to approximate. (ii) The

Table 1 Achievements of oracles for TD-instances with period $T = n^{\alpha}$, for $\alpha \in (0, 1)$. The stretch of all query algorithms is $1 + \sigma(r) = 1 + \varepsilon \cdot \frac{(1+\varepsilon/\psi)^{r+1}}{(1+\varepsilon/\psi)^{r+1}-1}$. For all but the first oracle, we assume that $\beta \downarrow 0$.

	preprocessing space/time	query time	recursion budget (depth) r
[17]	$K^* \cdot n^{2-\beta+o(1)}$	$n^{\delta+o(1)}$	$r \in \mathcal{O}(1)$
TRAPONLY	$n^{2-\beta+o(1)}$	$n^{\delta+o(1)}$	$r \approx \frac{\delta}{\alpha} - 1$
FLAT	$n^{2-\beta+o(1)}$	$n^{\delta+o(1)}$	$r \approx \frac{2\delta}{\alpha} - 1$
HORN	$n^{2-\beta+o(1)}$	$\approx \Gamma[o, d](t_o)^{\delta+o(1)}$	$r \approx \frac{2\delta}{\alpha} - 1$

TRAPONLY and FLAT oracles (cf. Section 4) which exploit TRAP and BIS to construct minimum-travel-time summaries from randomly selected landmarks to *all* reachable destinations. The preprocessed data structures require subquadratic space and time, independently of K^*. FLAT uses the query algorithms of [17]. TRAPONLY needs to extend them in order to recover missing summaries for local neighborhoods around each landmark. In both cases sublinear query-times are achieved. (iii) The HORN oracle (cf. Section 5) which organizes a hierarchy of landmarks, from many local landmarks possessing summaries only for small neighborhoods of destinations around them, up to a few global landmarks possessing summaries for all reachable destinations. HORN's preprocessing requirements are again subquadratic. We then devise and analyze a novel query algorithm (HQA) to exploit this hierarchy, with query-time *sublinear* in the Dijkstra-Rank of the query at hand. Except for the choice of landmarks, our algorithms are deterministic. An experimental study [15] demonstrates the excellent performance of our oracles in practice, achieving considerable memory savings and query-times about three orders of magnitude faster than TDD, and more than 70% faster than those in [14]. Table 1 summarizes the achievements of the TD-oracles presented here and their comparison with the oracles in [17]. Due to lack of space, all missing proofs are provided in the full version of the paper [16].

2 Preliminaries

Notation and terminology. For any integer $k \geq 1$, let $[k] = \{1, 2, \ldots, k\}$. A *time-dependent network instance* (TD-instance henceforth) consists of a directed graph $G = (V, A)$ with $|V| = n$ vertices and $|A| = m \in \mathcal{O}(n)$ arcs, where each arc $a \in A$ is accompanied with a continuous, pwl *arc-cost* function $D[a] : \mathbb{R}_{\geq 0} \mapsto \mathbb{R}_{\geq 0}$. We assume that all these functions are periodic with period $T > 0$ and are defined as follows: $\forall k \in \mathbb{N}, \forall t \in [0, T)$, $D[a](kT + t) = d[a](t)$, where $d[a] : [0, T) \to (0, M_a]$ is such that $\lim_{t \uparrow T} d[a](t) = d[a](0)$, for some fixed integer M_a denoting the maximum possible cost ever seen for arc a. Let also $M = \max_{a \in A} M_a$ denote the maximum arc-cost ever seen in the entire network. Since $D[a]$ is periodic, continuous and pwl, it can be represented succinctly by a sequence of K_a breakpoints (i.e., pairs of departure-times and arc-cost values) defining $d[a]$. $K = \sum_{a \in A} K_a$ is the number of breakpoints representing all arc-cost functions, $K_{\max} = \max_{a \in A} K_a$, and K^* is the number of *concavity-spoiling* breakpoints (the ones at which the arc-cost slopes increase). Clearly, $K^* \leq K$, and $K^* = 0$ for *concave* arc-cost functions. To ease the exposition and also for the sake of compliance with terminology in previous works (inspired by the primary application scenario of route planning in time-dependent road networks), we consider arc-costs as *arc-travel-times* and time-dependent shortest paths as *minimum-travel-time* paths. This terminology facilitates the following definitions. The *arc-arrival-time* function of $a \in A$ is $Arr[a](t) = t + D[a](t)$, $\forall t \in [0, \infty)$. The *path-arrival-time* function of a path $p = \langle a_1, \ldots, a_k \rangle$ in G (represented as a

sequence of arcs) is the composition $Arr[p](t) = Arr[a_k](Arr[a_{k-1}](\cdots(Arr[a_1](t))\cdots))$ of the arc-arrival-time functions for the constituent arcs. The *path-travel-time* function is then $D[p](t) = Arr[p](t) - t$.

For any $(o,d) \in V \times V$, $\mathcal{P}_{o,d}$ denotes the set of *od*-paths. For any $p \in \mathcal{P}_{o,x}$ and $q \in \mathcal{P}_{x,d}$, $s = p \bullet q \in \mathcal{P}_{o,d}$ is the concatenation of p and q at x. The *earliest-arrival-time* function is $Arr[o,d](t_o) = \min_{p\in\mathcal{P}_{o,d}}\{Arr[p](t_o)\}$, $\forall t_o \geq 0$, while the *minimum-travel-time* function is defined as $D[o,d](t_o) = \min_{p\in\mathcal{P}_{o,d}}\{D[p](t_o)\} = Arr[o,d](t_o) - t_o$. For a given query (o,d,t_o), $SP[o,d](t_o) = \{p \in P_{o,d} : Arr[p](t_o) = Arr[o,d](t_o)\}$ is the set of earliest-arrival-time (equivalently, minimum-travel-time) paths. $ASP[o,d](t_o)$ is the set of *od*-paths whose travel-time values are $(1+\varepsilon)$-approximations of the minimum-travel-time among all *od*-paths.

When we say that we *"grow a* `TDD` *ball from* (o,t_o)*"*, we refer to the execution of `TDD` from $o \in V$ at departure-time $t_o \in [0,T)$ for solving $TDSP(o,\star,t_o)$ (resp. $TDSP(o,d,t_o)$, for a specific destination d). Such a call, denoted as $\texttt{TDD}(o,\star,t_o)$ (resp. $\texttt{TDD}(o,d,t_o)$), takes time $\mathcal{O}(m + n\log(n)[1+\log\log(1+K_{\max})]) = \mathcal{O}(n\log(n)\log\log(K_{\max})])$, using predecessor search for evaluating continuous pwl functions [17]. The *Dijkstra-Rank* $\Gamma[o,d](t_o)$ of (o,d,t_o) is the number of settled vertices up to d, when executing $\texttt{TDD}(o,d,t_o)$.

$\forall a = uv \in A$ and $[t_s,t_f) \subseteq [0,T)$, we define upper- and lower-bounding travel-time metrics: the *minimally-congested* travel-time $\underline{D}[uv](t_s,t_f) := \min_{t_u\in[t_s,t_f)}\{D[uv](t_u)\}$ and the *maximally-congested* travel-time $\overline{D}[uv](t_s,t_f) := \max_{t_u\in[t_s,t_f)}\{D[uv](t_u)\}$. If $[t_s,t_f) = [0,T)$, we refer to the static *free-flow* and *full-congestion* metrics $\underline{D},\overline{D} : A \to [1,M]$, respectively. Each arc $a \in A$ is also equipped with scalars $\underline{D}[a]$ and $\overline{D}[a]$ in these static metrics. For any arc-cost metric D, $diam(G,D)$ is the diameter (largest possible vertex-to-vertex distance) of the graph. For example, $diam(G,\underline{D})$ is the free-flow diameter of G.

In our TD-instance, we assume that $T \geq diam(G,\underline{D})$. If not, we take the minimum number c of copies of each $d[a]$ as a single arc-travel-time function $d'[a] : [0,cT) \mapsto \mathbb{R}_{>0}$ and $D'[a](t+kT') = d'[a](t)$, $\forall t \in [0,T')$ such that $T' = cT \geq diam(G,\underline{D'})$. In addition, we can guarantee that $T = n^\alpha$ for a *constant* $\alpha \in (0,1)$ of our control. If $T \neq n^\alpha$, we scale the travel-time metric by setting $D'' = \frac{n^\alpha}{T}\cdot D$ (e.g., we change the unit by which we measure time from milliseconds to seconds) and use the period $T'' = n^\alpha$, without affecting the structure of the instance at all. From now on we assume w.l.o.g. that $T = n^\alpha \geq diam(G,\underline{D})$.

For any $v \in V$, departure-time $t_v \in \mathbb{R}_{\geq 0}$, integer $F \in [n]$ and $R > 0$, $B[v;F](t_v)$ ($B[v;R](t_v)$) is a ball of size F (of radius R) grown by `TDD` from (v,t_v), in the time-dependent metric. Analogously, $\underline{B}[v;F]$ ($\underline{B}[v;R]$) and $\overline{B}[v;F]$ ($\overline{B}[v;R]$) are, respectively, the size-F (radius-R) balls from v in the free-flow and fully-congested travel-time metrics.

A pair of continuous, pwl, periodic functions $\overline{\Delta}[o,d]$ and $\underline{\Delta}[o,d]$), with a (hopefully) small number of breakpoints, are $(1+\varepsilon)$-*upper-approximation* and $(1+\varepsilon)$-*lower-approximation* of $D[o,d]$, if $\forall t_o \geq 0$, $\frac{D[o,d](t_o)}{1+\varepsilon} \leq \underline{\Delta}[o,d](t_o) \leq D[o,d](t_o) \leq \overline{\Delta}[o,d](t_o) \leq (1+\varepsilon)\cdot D[o,d](t_o)$.

Assumptions on the arc-cost metric. The directedness and time-dependence of the TD-instance imply an asymmetric arc-cost metric, which also evolves with time. To achieve a smooth transition from static and undirected graphs towards time-dependent and directed graphs, we need a quantification of the degrees of asymmetry and evolution of our metric over time. These are captured via a set of parameters depicting the steepness of the minimum-travel-time functions, the ratio of minimum-travel-times in opposite directions, and the relation between graph expansion and travel-times. We make some assumptions on the values of these parameters, which seem quite natural for our main application scenario (route planning in road networks). We only present a qualitative interpretation of them. Their

exact statements, along with their validation on real-world road networks, are presented in [16]. It is noted that Assumptions 1 and 2 were exploited also in the analyses in [17].

▶ **Assumption 1** (Bounded Travel-Time Slopes). *All the minimum-travel-time slopes are bounded in a given interval* $[-\Lambda_{\min}, \Lambda_{\max}]$*, for given constants* $\Lambda_{\min} \in [0,1)$ *and* $\Lambda_{\max} \geq 0$*.*

▶ **Assumption 2** (Bounded Opposite Trips). *The ratio of minimum-travel-times in opposite directions between two vertices, for any specific departure-time but not necessarily via the same path, is upper bounded by a given constant* $\zeta \geq 1$*.*

▶ **Assumption 3** (Growth of Free-Flow Dijkstra Balls). $\forall F \in [n]$*, the free-flow ball* $\underline{B}[v;F]$ *blows-up by at most a polylogarithmic factor, when expanding its (free-flow) radius up to the value of the full-congestion radius within* $\underline{B}[v;F]$*.*

Finally, we need to quantify the correlation between the arc-cost metric and the Dijkstra-Rank metric induced by it. For this reason, inspired by the notion of the doubling dimension (e.g., [2] and references therein), we consider some *scalar* $\lambda \geq 1$ and functions $f, g : \mathbb{N} \mapsto [1,\infty)$, such that the following hold: $\forall (o,d,t_o) \in V \times V \times [0,T)$, (i) $\Gamma[o,d](t_o) \leq f(n) \cdot (D[o,d](t_o))^{\lambda}$, and (ii) $D[o,d](t_o) \leq g(n) \cdot (\Gamma[o,d](t_o))^{1/\lambda}$. This property trivially holds, e.g., for $\lambda = 1$, $f(n) = n$, and $g(n) = \max_{a \in A} \{\overline{D}[a]\}$. Of course, our interest is for the smallest possible values of λ and at the same time the slowest-growing functions $f(n), g(n)$. Our last assumption quantifies the boundedness of this correlation by restricting λ, $f(n)$ and $g(n)$.

▶ **Assumption 4.** *There exist* $\lambda \in \mathrm{o}\left(\frac{\log(n)}{\log\log(n)}\right)$ *and* $f(n), g(n) \in \mathrm{polylog}(n)$ *s.t. the following hold:* (i) $\Gamma[o,d](t_o) \leq f(n) \cdot (D[o,d](t_o))^{\lambda}$*, and* (ii) $D[o,d](t_o) \leq g(n) \cdot (\Gamma[o,d](t_o))^{1/\lambda}$*. Analogous inequalities hold for the free-flow and the full-congestion metrics* $\underline{D}$ *and* $\overline{D}$*.*

Note that static oracles based on the doubling dimension (e.g., [2]) demand a *constant* value for λ. We relax this by allowing λ to be even a (sufficiently slowly) growing function of n. We also introduce some additional slackness, by allowing divergence from the corresponding powers by polylogarithmic factors. In the rest of the paper we consider sparse TD-instances (i.e., $m \in \mathcal{O}(n)$), compliant with Assumptions 1, 2, 3, and 4.

3 The TRAP approximation method

The *trapezoidal* (TRAP) method is a novel algorithm for computing one-to-many $(1+\varepsilon)$-upper-approximations $\overline{\Delta}[\ell,v] : [0,T) \mapsto \mathbb{R}_{>0}$ of $D[\ell,v]$, from a (landmark) vertex ℓ towards all sufficiently distant destinations v. TRAP is remarkably simple and works as follows. First, $[0,T)$ is split into $\lceil \frac{T}{\tau} \rceil$ consecutive length-τ subintervals, where τ is a tuning parameter to be fixed later. Then, for each interval $[t_s, t_f = t_s + \tau) \subseteq [0,T)$, a $(1+\varepsilon)$-upper-approximation of the projection $D[\ell,v] : [t_s,t_f) \mapsto \mathbb{R}_{>0}$ is constructed. Finally, the concatenation of all these $(1+\varepsilon)$-upper-approximations per subinterval constitutes the requested $(1+\varepsilon)$-upper-approximation $\overline{\Delta}[\ell,v]$ of $D[o,d] : [0,T) \mapsto \mathbb{R}_{>0}$. Note that, contrary to BIS, no assumption is made on the shapes of the min-cost functions to approximate within each subinterval; in particular, no assumption is made on them being concave. TRAP only exploits the fact that τ is small, along with Assumption 1 on the boundedness of travel-time slopes. We now describe the upper- and lower-approximations of $D[o,d]$ that we construct in a subinterval $I_k = [t_s = (k-1)\tau, t_f = k\tau) \subset [0,T)$, $k \in \left[\left\lceil \frac{T}{\tau} \right\rceil\right]$, from a vertex $\ell \in V$ towards some destination $v \in V$. The quality of the upper-approximation depends on the value of τ and the delay values at the endpoints of I_k, as we shall explain shortly. TRAP computes the following two functions of $D[\ell,v]$ (cf. Fig. 1): $\forall t \in I_k$, $\overline{\delta}_k[\ell,v](t) = \min\left\{ D[\ell,v](t_f) + \Lambda_{\min} t_f - \Lambda_{\min} t \ ,\ D[\ell,v](t_s) - \Lambda_{\max} t_s + \Lambda_{\max} t \right\}$

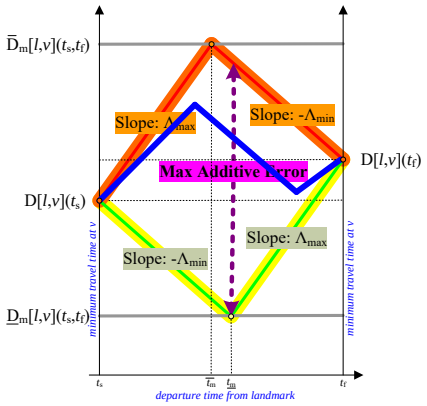

Figure 1 The upper-approximation $\overline{\delta}_k[\ell, v]$ (thick orange, upper pwl line), and lower-approximation $\underline{\delta}_k[\ell, v]$ (thick green, lower pwl line), of the unknown function $D[\ell, v]$ (blue pwl line) within $I_k = [t_s = (k-1)\tau, t_f = k\tau)$.

and $\underline{\delta}_k[\ell, v](t) = \max\{\ D[\ell, v](t_f) - \Lambda_{\max} t_f + \Lambda_{\max} t\ ,\ D[\ell, v](t_s) + \Lambda_{\min} t_s - \Lambda_{\min} t\ \}$ and considers them as the upper- and lower-approximating functions of $D[\ell, v]$ within I_k. The correctness of this choice is proved in the next lemma, which follows by Assumption 1.

▶ **Lemma 5.** *$\overline{\delta}_k[\ell, v](t)$ and $\underline{\delta}_k[\ell, v](t)$ upper- and lower-approximate $D[\ell, v](t)$ within I_k.*

Let $(\underline{t}_m, \underline{D}_m)$ and $(\overline{t}_m, \overline{D}_m)$ be the intersections of the legs in the definitions of $\underline{\delta}_k[\ell, v]$ and $\overline{\delta}_k[\ell, v]$, respectively. The *maximum additive error* in I_k (c.f. Figure 1) is $MAE(I_k) := \max_{t \in I_k} \{\overline{\delta}_k[\ell, v](t) - \underline{\delta}_k[\ell, v](t)\} = \overline{\delta}_k[\ell, v](\underline{t}_m) - \underline{\delta}_k[\ell, v](\underline{t}_m)$. The following lemma proves that, for τ sufficiently small, $MAE(I_k)$ cannot be large. It also provides a *sufficient condition* for the value of τ so that $\overline{\delta}_k[\ell, v]$ is a $(1+\varepsilon)$-upper-approximation of $D[\ell, v]$ in I_k.

▶ **Lemma 6.** *$\forall (\ell, v) \in L \times V$, $\forall k \in \left[\left\lceil \frac{T}{\tau} \right\rceil\right]$ and $I_k = [(k-1)\tau, k\tau)$, the following hold: (1) $MAE[\ell, v](I_k) \leq \Lambda_{\max} \cdot \tau$; (2) $\overline{\delta}_k[\ell, v]$ is a $(1+\varepsilon)$-upper-approximation of $D[\ell, v]$ within I_k, if $\left[\ D[\ell, v](t_s) \geq \left(\Lambda_{\min} + \frac{\Lambda_{\max}}{\varepsilon}\right) \cdot \tau\ \right] \vee \left[\ D[\ell, v](t_f) \geq \left(1 + \frac{1}{\varepsilon}\right) \Lambda_{\max} \cdot \tau\ \right]$*

For given $\tau > 0$ and $\ell \in L$, the set of *faraway destinations* from ℓ is $V[\ell](\tau) = \{v \in V : \tau[\ell, v] > \tau\}$. $\tau[\ell, v] = \frac{\underline{D}[\ell, v]}{(1+1/\varepsilon)\Lambda_{\max}}$ is a sufficient τ-value for $\overline{\delta}_k[\ell, v]$ being $(1+\varepsilon)$-upper-approximation of $D[\ell, v]$ within $I_k = [(k-1)\tau[\ell, v], k\tau[\ell, v])$ (cf. Lemma 6). The next theorem proves that `TRAP` provides a $(1+\varepsilon)$-upper-approximation $\overline{\Delta}[\ell, v]$ for all faraway destinations from ℓ, and also estimates the preprocessing requirements of the algorithm.

▶ **Theorem 7.** *Fix $\ell \in L$, $F > f(n)$, and $\tau \in (0, T)$ s.t. $|V[\ell](\tau)| = n - F$. Let $\tau^* = \min_{v \in V[\ell](\tau)} \left\{ \frac{\underline{D}[\ell, v]}{(1+1/\varepsilon)\Lambda_{\max}} \right\}$. $\forall v \in V[\ell](\tau)$, $\overline{\Delta}[\ell, v]$ is the concatenation of all the upper-approximating functions $\overline{\delta}_k[\ell, v]$ that `TRAP` returns per subinterval $I_k = [\ t_{s_k} = (k-1)\tau^*\ ,\ t_{f_k} = \min\{k\tau^*, T\}\ ) : k \in \left[\left\lceil \frac{T}{\tau^*} \right\rceil\right]$. Then, $\forall v \in V[\ell](\tau)$, $\overline{\Delta}[\ell, v]$ is a $(1+\varepsilon)$-upper-approximation of $D[\ell, v]$ in $[0, T)$, requiring PRE SPACE... at most $2\left\lceil \frac{T}{\tau^*} \right\rceil$ breakpoints. PRE TIME... The number of calls to $TDSP(\ell, \star, t)$ for their construction is $\left\lceil \frac{T}{\tau^*} \right\rceil \leq 1 + \frac{T(1+1/\varepsilon)\Lambda_{\max}}{\min_{v \in V[\ell](\tau)}\{\underline{D}[\ell, v]\}} \in \mathcal{O}(n^\alpha)$.*

Proof of Theorem 7. τ^* is the appropriate length for the subintervals which assures that `TRAP` returns $(1+\varepsilon)$-upper-approximations for all faraway destinations from ℓ. By definition it holds that $\tau^* \geq \tau$. Since $F > f(n)$, it holds that `TRAP` does not consider destinations at free-flow distance less than 1. To see this, fix $v \in V$ s.t. $\underline{D}[\ell, v] \leq 1$. By Assumption 4,

$\underline{\Gamma}[\ell,v] \leq f(n)\cdot \underline{D}[\ell,v]^{\lambda} \leq f(n) < F$. Thus, we can be sure that $v \notin V[\ell](\tau)$. Since $T = n^{\alpha}$, we conclude that $\frac{T}{\tau^*} = \frac{T(1+1/\varepsilon)\Lambda_{\max}}{\min_{v\in V[\ell](\tau)} \underline{D}[\ell,v]} \in \mathcal{O}(n^{\alpha})$. We proceed now with the analysis of TRAP. $[0,T)$ is split into $\lceil \frac{T}{\tau^*} \rceil$ consecutive length-τ^* subintervals. Lemma 5 assures that for each $I_k = [k\tau^*, (k+1)\tau^*)$ an upper-approximating function $\overline{\delta}_k[\ell,v]$ of $D[\ell,v]$ is determined, for each $v \in V[\ell](\tau)$. The concatenation of all these functions constitutes the upper-approximating function $\overline{\Delta}[\ell,v]$ for $D[\ell,v]$ within $[0,T)$. Since $\tau[\ell,v] \geq \tau^* \Rightarrow \underline{D}[\ell,v] \geq \left(1+\frac{1}{\varepsilon}\right)\Lambda_{\max}\tau^*$, we deduce (cf. Lemma 6) that, for all $v \in V[\ell](\tau)$, the produced upper-approximations within the consecutive length-τ^* intervals are $(1+\varepsilon)$-approximations of $D[\ell,v]$. TRAP preprocesses $\ell \in L$ by making $\lceil \frac{T}{\tau^*} \rceil \in \mathcal{O}(n^{\alpha})$ calls to $TDSP(\ell,\star,t)$, to sample the endpoints of all the $\lceil \frac{T}{\tau^*} \rceil$ length-τ^* subintervals. For storing $\overline{\Delta}[\ell,v]$, it needs $2\lceil \frac{T}{\tau^*} \rceil$ breakpoints (at most one intermediate breakpoint $(\overline{t}_m, \overline{D}_m)$ per subinterval). ◀

4 Oracles with fully-informed landmarks

We now describe two novel oracles with landmarks possessing summaries for all reachable destinations, excluding possibly a small neighborhood around them. We start with a random landmark set $L \subset_{\mathbf{uar}(\rho)} V$, i.e., we decide independently and uniformly at random whether each vertex is a landmark, with probability $\rho = n^{-\omega}$ for a constant $\omega \in (0,1)$. We consider as *faraway vertices* from $\ell \in L$, all the vertices at free-flow distance at most $\underline{R} = T^{\theta}$ from it, for a constant $\theta \in (0,1)$ to be determined later. $F = \max_{\ell\in L}\{|\underline{B}[\ell;\underline{R}]|\}$ is the maximum number of faraway vertices from a landmark. The next lemma shows that the main parameters we should consider w.r.t. a TD-instance are λ (cf. Assumption 4) and $\alpha \in (0,1)$ s.t. $T = n^{\alpha}$. All the other parameters essentially adjust their values to them.

▶ **Lemma 8.** *For $\nu \in (0,1)$ s.t. $T = diam(G,\underline{D})^{1/\nu}$, $\theta \in (0,1)$ s.t. $\frac{\nu}{\theta} \in \mathcal{O}(1)$ and λ, f, g defined as in Assumption 4, the following hold:* (i) $\frac{1}{\lambda\nu} = \alpha \pm o(1)$, *and* (ii) $F \in n^{[1\pm o(1)]\theta/\nu}$.

The TRAPONLY oracle. A first attempt towards avoiding the dependency of the preprocessing requirements on K^* is to develop an oracle, called TRAPONLY, whose preprocessing is based solely on TRAP.TRAPONLY PREPROCESSING... The preprocessing of TRAPONLY first considers as subinterval length the value $\tau = \frac{\underline{R}}{(1+1/\varepsilon)\Lambda_{\max}} > 0$. It then calls TRAP for each landmark $\ell \in L$, which guarantees $(1+\varepsilon)$-upper-approximations for all the faraway destinations $v \in V[\ell](\tau)$ (cf. Theorem 7).RQA+... The distances of nearby destinations from ℓ are left to be computed by the query algorithm of TRAPONLY, which is an appropriate variant of RQA (we call it RQA^+) which additionally grows a small TDD ball of size $F\,\text{polylog}(F)$ (cf. Assumption 3) from each newly settled landmark. TRAPONLY COMPLEXITY... The following theorem analyzes the performance of TRAPONLY.

▶ **Theorem 9.** *The expected time of RQA^+ and the preprocessing requirements of* TRAPONLY *are: $\mathbb{E}\{Q_{\text{RQA}^+}\} \in \mathcal{O}\left(n^{\omega r + \max\{\omega, \frac{\theta}{\nu}\}+o(1)}\right)$ and $S_{\text{TRAPONLY}},\ P_{\text{TRAPONLY}} \in \mathcal{O}\left(n^{2+\alpha\cdot(1-\theta)-\omega+o(1)}\right)$.*

Proof of Theorem 9. During the preprocessing, TRAPONLY makes $\lceil \frac{T}{\tau^*} \rceil \leq 1 + \frac{T(1+1/\varepsilon)\Lambda_{\max}}{\underline{R}} = 1 + T^{1-\theta}(1+1/\varepsilon)\Lambda_{\max}$ calls of $\text{TDD}(\ell,t)$, for departure-times $t \in \left\{0, \tau^*, 2\tau^*, \ldots, \lceil \frac{T}{\tau^*} \rceil - 1\right\}$ and landmarks $\ell \in L$, where the equality comes from Lemma 8. Therefore, the preprocessing-time is dominated by the aggregate time for all these TDD probes. Taking into account that each TDD probe takes time $\mathcal{O}(n\log(n)\log\log(K_{\max}))$ and that $|L| = \rho n = n^{1-\omega}$ landmarks, by using Lemma 8 we get the following: $P_{\text{TRAPONLY}} = n^{1-\omega}\cdot n^{\frac{1-\theta}{\nu\lambda}[1+o(1)]}\cdot n\log(n)\log\log(n) \in n^{2-\omega+\frac{1-\theta}{\nu\lambda}[1+o(1)]+\frac{\log\log(n)+\log\log\log(n)}{\log(n)}} = n^{2-\omega+\alpha\cdot(1-\theta)+o(1)}$. The calculations are analogous for the required preprocessing space: For all landmarks $\ell \in L$

and all their faraway destinations $v \in V[\ell](\tau)$, the total number of breakpoints to store is at most $S_{\texttt{TRAPONLY}} = 2\left\lceil \frac{T}{\tau^*} \right\rceil \rho n^2 \in n^{2-\omega+\frac{1-\theta}{\nu\lambda}[1+o(1)]+o(1)} = n^{2-\omega+\alpha\cdot(1-\theta)+o(1)}$. As for the query-time complexity of $\texttt{RQA}^+$, recall that the expected number of TDD balls that it grows is $(1/\rho)^r$. Additionally, $\texttt{RQA}^+$ grows $(1/\rho)^r$ TDD balls from the corresponding closest landmarks. Each ball from a new center costs $\mathcal{O}((1/\rho)\log(1/\rho))$. Each ball from a landmark costs $\mathcal{O}(F\,\mathrm{polylog}(F)) \in n^{[1\pm o(1)]\theta/\nu}$. Thus, the expected query-time is upper-bounded as follows: $\mathbb{E}\{Q_{\texttt{RQA}^+}\} \in \mathcal{O}((1/\rho)^r[(1/\rho)\log(1/\rho) + F\,\mathrm{polylog}(F)]\log\log(K_{\max})) = \mathcal{O}\big(n^{\omega r+\max\{\omega,[1+o(1)]\theta/\nu\}}\big)$. ◀

The next corollaries are parameter-tuning examples showcasing the trade-offs among the sublinearity of query-time, the subquadratic preprocessing requirements and the stretch.

▶ **Corollary 10.** *For $\delta \in (\alpha, 1)$, $\beta \in (0, \alpha^2\nu]$, $\omega = \frac{\delta}{r+1}$, $\theta = \frac{\delta\nu}{r+1}$ and $r = \left\lfloor \frac{\delta\cdot(1+\alpha\nu)}{\alpha+\beta} \right\rfloor - 1$, $S_{\texttt{TRAPONLY}}, P_{\texttt{TRAPONLY}} \in n^{2-\beta+o(1)}$, $\mathbb{E}\{Q_{\texttt{RQA}^+}\} \in n^{\delta+o(1)}$ and the stretch is $1+\sigma(r) = 1+\varepsilon\cdot\frac{(1+\varepsilon/\psi)^{r+1}}{(1+\varepsilon/\psi)^{r+1}-1}$.*

▶ **Corollary 11.** *For any integer $k \geq 2$, let $\eta(k) = \left\lceil \frac{\log(k/(k-1))}{\log(1+\varepsilon/\psi)} \right\rceil - 1$, $\delta \in (0,1)$ and $\beta \in \left(0, \frac{\delta}{\eta(k)+2}\right)$. Then* TRAPONLY *achieves stretch $1+k\cdot\varepsilon$ with $S_{\texttt{TRAPONLY}}, P_{\texttt{TRAPONLY}} \in n^{2-\beta+o(1)}$ and $\mathbb{E}\{Q_{\texttt{RQA}^+}\} \in n^{\delta+o(1)}$, by scaling the TD-instance so that $T = n^\alpha$, for $\alpha = \frac{\delta-[\eta(k)+2]\cdot\beta}{\eta(k)+2-\delta\nu}$.*

The FLAT oracle. Our second attempt, the FLAT oracle, provides preprocessed information for all reachable destinations per landmark, and uses the query algorithm RQA [17]. PRE: BIS+TRAP... FLAT considers again as subinterval length the value $\tau = \frac{R}{(1+1/\varepsilon)\Lambda_{\max}} > 0$. Then, it constructs summaries for all reachable destinations per landmark $\ell \in L$ exploiting both BIS and TRAP: BIS handles all the (at most $F = \max_{\ell\in L}\{|\underline{B}[\ell;\underline{R}]|\}$) nearby destinations in $\underline{B}[\ell;\underline{R}]$, whereas TRAP handles all the faraway destinations of $V \setminus \underline{B}[\ell;\underline{R}]$. The space requirements for the summaries created by TRAP are exactly the same as in TRAPONLY. As for the summaries computed by BIS, we avoid the linear dependence of BIS on K^* by assuring that F is sufficiently small (but not too small) and exploiting Assumption 3 which guarantees that the involved subgraph $\underline{B}'[\ell;F]$ in the preprocessing phase of BIS on behalf of ℓ has size $\mathcal{O}(F\,\mathrm{polylog}(F))$. The next lemma shows that BIS is affected only by the concavity-spoiling breakpoints of arc-travel-time functions in $\underline{B}'[\ell;F]$, rather than the entire graph.

▶ **Lemma 12.** $\forall(\ell,v) \in L\times\underline{B}[\ell;F], \forall u \in V\setminus\underline{B}'[\ell;F], \forall t\in[0,T), D[\ell,v](t) < D[\ell,u](t)$.

Proof of Lemma 12. From the definitions of the involved free-flow and full-congestion Dijkstra balls, the following holds: $D[\ell,v](t) \leq \overline{D}[\ell,v] \leq \overline{R}[\ell] < \underline{D}[\ell,u] \leq D[\ell,u](t)$. ◀

The following theorem summarizes the complexities of the FLAT oracle.

▶ **Theorem 13.** *The query-time $Q_{\texttt{RQA}}$ and the preprocessing time $P_{\texttt{FLAT}}$ and space $S_{\texttt{FLAT}}$ of* FLAT *are: $\mathbb{E}\{Q_{\texttt{RQA}}\} \in \mathcal{O}\big(n^{\omega(r+1)+o(1)}\big)$ and $P_{\texttt{FLAT}}$, $S_{\texttt{FLAT}} \in \mathcal{O}\big(n^{1-\omega+o(1)}\cdot[n^{2\theta/\nu} + n^{1+\alpha\cdot(1-\theta)}]\big)$.*

Proof of Theorem 13. BIS requires space at most $F^2\,\mathrm{polylog}(F)$, since by Lemma 12 the involved graph only contains $F\,\mathrm{polylog}(F)$ vertices and concavity-spoiling breakpoints at the arc-travel-time functions. For the faraway vertices of $V\setminus\underline{B}[\ell;F]$, since $\tau = \frac{R}{(1+1/\varepsilon)\Lambda_{\max}}$, TRAP provides $(1+\varepsilon)$-approximate summaries for all destinations $v \in V\setminus\underline{B}[\ell;\underline{R}]$, because the sufficient condition of Theorem 7 holds: $\underline{D}[\ell,v] > \underline{R} = (1+1/\varepsilon)\,\Lambda_{\max}\tau$. Thus, we conclude that $S_{\texttt{FLAT}} \in \rho n\left[F^2\,\mathrm{polylog}(F) + \frac{T(1+1/\varepsilon)\Lambda_{\max}n}{\underline{R}}\right] \overset{/*\ \text{L.8}\ */}{=} n^{1-\omega}[n^{(2\theta/\nu)\cdot[1+o(1)]} + n^{1+\alpha\cdot(1-\theta)[1+o(1)]}]$ $= n^{1-\omega+[1+o(1)]\cdot\max\{\ 2\theta/\nu\ ,\ 1+\alpha(1-\theta)\ \}+o(1)}$, since $f(n), g(n) \in \mathrm{polylog}(n)$. ◀

The next corollaries are parameter-tuning examples to showcase the effectiveness of FLAT.

▶ **Corollary 14.** *If* $\delta \in (\alpha, 1)$, $\beta \in \left(0, \frac{\alpha\cdot(1+\alpha)}{2/\nu+\alpha}\right]$, $\omega = \frac{\delta}{r+1}$, $r = \left\lfloor \frac{\delta}{\alpha} \cdot \frac{2/\nu+\alpha}{(\beta/\alpha)\cdot(2/\nu+\alpha)+(2/\nu-1)} \right\rfloor - 1$ *and* $\theta = \frac{1+\alpha}{2/\nu+\alpha}$, *then* FLAT *has* $P_{\texttt{FLAT}}, S_{\texttt{FLAT}} \in n^{2-\beta+o(1)}$, $\mathbb{E}\{Q_{\texttt{RQA}}\} \in n^{\delta+o(1)}$ *and stretch* $1 + \sigma(r) = 1 + \varepsilon \cdot \frac{(1+\varepsilon/\psi)^{r+1}}{(1+\varepsilon/\psi)^{r+1}-1}$.

▶ **Corollary 15.** *For integer* $k \geq 2$, *let* $\eta(k) = \left\lceil \frac{\log(k/(k-1))}{\log(1+\varepsilon/\psi)} \right\rceil - 1$ *and* $\delta \in (0, 1)$. FLAT *achieves a target stretch* $1 + k \cdot \varepsilon$ *with preprocessing requirements* $n^{2-o(1)}$ *and expected query-time* $n^{\delta+o(1)}$, *by scaling the TD-instance so that* $T = n^{\alpha}$ *for* $\alpha = \frac{2\delta}{[\eta(k)+2]\cdot(2-\nu)-\delta\nu}$, *as* $\beta \downarrow 0$.

Comparison of TRAPONLY and FLAT. Both TRAPONLY and FLAT depend on the travel-time metric, but are independent of the degree of disconcavity K^*. On one hand, TRAPONLY is a simpler oracle, at least w.r.t. its preprocessing phase. On the other hand, FLAT achieves a better approximation for the same TD-instance and anticipations for sublinear query-time n^{δ} and subquadratic preprocessing requirements $n^{2-\beta}$. This is because, as $\beta \downarrow 0$, FLAT guarantees a recursion budget r of (roughly) $\frac{2\delta}{\alpha} - 1$, whereas TRAPONLY achieves about half this value and r has an exponential effect on the stretch that the query algorithms achieve.

5 The HORN oracle

We now describe and analyze the *Hierarchical ORacle for time-dependent Networks* (HORN), whose query algorithm is highly competitive against TDD, not only for long-range queries (i.e., having Dijkstra-Rank proportional to the network size) but also for medium- and short-range queries, while ensuring *subquadratic* preprocessing space and time. The main idea of HORN is to preprocess: many landmarks, each possessing summaries for a few destinations around them, so that all short-range queries can be answered using only these landmarks; fewer landmarks possessing summaries for more (but still not all) destinations around them, so that medium-range queries be answered by them; and so on, up to only a few landmarks (those required by FLAT) possessing summaries for all reachable destinations. The *area of coverage* $C[\ell] \subset V$ of ℓ is the set of its nearby vertices, for which ℓ possesses summaries. ℓ is called *informed* for each $v \in C[\ell]$, and *uninformed* for each $v \in V \setminus C[\ell]$. The landmarks are organized in a hierarchy, according to the sizes of their areas of coverage. Each level L_i in the hierarchy is accompanied with a *targeted Dijkstra-Rank* $N_i \in [n]$, and the goal of HORN is to assure that L_i should suffice for RQA to successfully address queries (o, d, t_o) with $\Gamma[o, d](t_o) \leq N_i$, in time $o(N_i)$. The difficulty of this approach lies in the analysis of the query algorithm. We want to execute a variant of RQA which, based on a *minimal* subset of landmarks, would guarantee a $(1 + \sigma(r))$-approximate solution for any query (o, d, t_o) (as in TRAPONLY and FLAT), but also time-complexity sublinear in $\Gamma[o, d](t_o)$. We propose the *Hierarchical Query Algorithm* (HQA) which grows an initial TDD ball from (o, t_o) that stops only when it settles an informed landmark ℓ w.r.t. d which is at the "right distance" from o, given the density of landmarks belonging to the same level with ℓ. HQA essentially "guesses" as appropriate level-i in the hierarchy the level that contains ℓ, and continues with the execution of RQA with landmarks having coverage at least equal to that of ℓ (cf. Fig. 2).

Initialization of HORN. We use the following parameters for the hierarchical construction: (i) $k \in \mathcal{O}(\log\log(n))$ determines the number of levels (minus one) comprising the hierarchy of landmarks. (ii) $\gamma > 1$ determines the actual values of the targeted Dijkstra-Ranks, one per level of the hierarchy. For example, as γ gets closer to 1, the targeted Dijkstra-Ranks

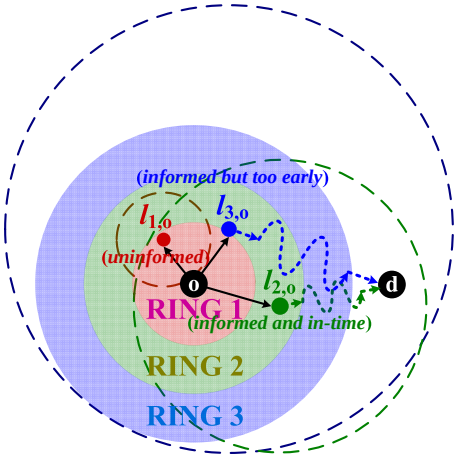

Figure 2 Demonstration of execution of HQA. Dashed circles indicate areas of coverage. Solid circular stripes indicate the rings of the corresponding levels in the hierarchy. Landmark $\ell_{1,o}$ is uninformed and $\ell_{3,o}$, although informed, comes too early. $\ell_{2,o}$ is both informed and within the ring of its own level, leading HQA to deduce that the appropriate level is $i = 2$.

accumulate closer to small- and medium-rank queries. (iii) $\delta \in (0,1)$ is the parameter that quantifies the sublinearity of the query algorithm (HQA), in each level of the hierarchy, compared to the targeted Dijkstra-Rank of this level. In particular, if N_i is the targeted Dijkstra-Rank corresponding to level-i in the hierarchy, then HQA should be executed in time $\mathcal{O}\big((N_i)^{\delta}\big)$, if only the landmarks in this level (or in higher levels) are allowed to be used.

Preprocessing of HORN. $\forall i \in [k]$, we set the targeted Dijkstra-Rank for level-i to $N_i = n^{(\gamma^i-1)/\gamma^i}$. Then, we construct a randomly chosen level-i landmark set $L_i \subset_{\mathbf{uar}(\rho_i)} V$, where $\rho_i = N_i^{-\delta/(r+1)} = n^{-\delta(\gamma^i-1)/[(r+1)\gamma^i]}$. Each $\ell_i \in L_i$ acquires summaries for all (and only those) $v \in C[\ell_i]$, where $C[\ell_i]$ is the smallest *free-flow* ball centered at ℓ_i containing $c_i = N_i \cdot n^{\xi_i} = n^{(\gamma^i-1)/\gamma^i+\xi_i}$ vertices, for a sufficiently small constant $\xi_i > 0$. The summaries to the $F_i = c_i^{\chi}$ nearby vertices around ℓ_i are constructed with BIS; the summaries to the remaining $c_i - F_i$ faraway vertices of ℓ_i are constructed with TRAP, where $\chi = \frac{\theta}{\nu} = \frac{1+\alpha}{2+\alpha\nu} \in \left[\frac{1}{2}, \frac{2}{2+\nu}\right]$ is an appropriate value determined to assure the correctness of FLAT w.r.t. the level-i of the hierarchy. An ultimate level $L_{k+1} \subset_{\mathbf{uar}(\rho_{k+1})} V$ of landmarks, with $\rho_{k+1} = n^{-\frac{\delta}{r+1}}$, assures that HORN is also competitive against queries with Dijkstra-Rank greater than $n^{(\gamma^k-1)/\gamma^k}$. We choose in this case $c_{k+1} = N_{k+1} = n$, $F_{k+1} = n^{\chi}$ and $C[\ell_{k+1}] = V$, $\forall \ell_{k+1} \in L_{k+1}$.

Description of HQA. A TDD ball from (o, t_o) is grown until d is settled, or the *(ESC)*-criterion or the *(ALH)*-criterion is fulfilled (whichever occurs first):

- **Early Stopping Criterion (ESC):** $\ell_o \in L = \cup_{i\in[k+1]} L_i$ is settled, which is informed ($d \in C[\ell_o]$) and, for $\varphi \geq 1$, $\frac{\overline{\Delta}[\ell_o,d](t_o+D[o,\ell_o](t_o))}{D[o,\ell_o](t_o)} \geq (1+\varepsilon)\cdot\varphi\cdot(r+1)+\psi-1$.
- **Appropriate Level of Hierarchy (ALH):** For some level $i \in [k]$ of the hierarchy, the first landmark $\ell_{i,o} \in L_i$ is settled such that: (i) $d \in C[\ell_{i,o}]$ ($\ell_{i,o}$ is "informed"); and (ii) $\frac{N_i^{\delta/(r+1)}}{\ln(n)} \leq \Gamma[o,\ell_{i,o}](t_o) \leq \ln(n)\cdot N_i^{\delta/(r+1)}$ ($\ell_{i,o}$ is at the "right distance"). In that case, HQA concludes that i is the "appropriate level" of the hierarchy to consider. Observe that the level-$(k+1)$ landmarks are always informed. Thus, if no level-$(\leq k)$ informed landmark is discovered at the right distance, then the first level-$(k+1)$ landmark that

will be found at distance larger than $\ln(n) \cdot N_k^{\delta/(r+1)}$ will be considered to be at the right distance, and then HQA concludes that the appropriate level is $k+1$.

If d is settled, an exact solution is returned. If (ESC) causes termination of HQA, the value $D[o,\ell_o](t_o) + \overline{\Delta}[\ell_o,d](t_o + D[o,\ell_o](t_o))$ is reported. Otherwise, HQA stops the initial ball due to the (ALH)-criterion, considering $i \in [k+1]$ as the appropriate level, and then continues executing the variant of RQA, call it $\texttt{RQA}_i$, which uses as its landmark set $M_i = \cup_{j=i}^{k+1} L_j$. Observe that $\texttt{RQA}_i$ may fail constructing approximate solutions via certain landmarks in M_i that it settles, since they may not be informed about d. Eventually, HQA returns the best *od*-path (w.r.t. the approximate travel-times) among the ones discovered by $\texttt{RQA}_i$ via *all* settled and informed landmarks ℓ. Theorem 16 summarizes the performance of HORN.

▶ **Theorem 16.** *Consider any TD-instance with* $\lambda \in \mathrm{o}\Big(\sqrt{\frac{\log(n)}{\log\log(n)}}\Big)$ *and* $g(n), f(n) \in \mathrm{polylog}(n)$ *(cf. Assumption 4). For* $\varphi = \frac{\varepsilon\cdot(r+1)}{\psi\cdot(1+\varepsilon/\psi)^{r+1}-1}$ *and* $k \in \mathcal{O}(\log\log(n))$*, let* $\xi_i \in \Big(\Big[(1+\lambda)\cdot\log\log(n) + \lambda\log\Big(1+\frac{\zeta}{1-\Lambda_{\min}}\Big)\Big]/\log(n)\ ,\ 1-\gamma^{-i}\Big)$*, for all* $i \in [k]$*. Then, for any query* (o,d,t_o) *s.t.* $N_{i^*-1} < \Gamma[o,d](t_o) \le N_{i^*}$ *for some* $i^* \in [k+1]$*, any* $\delta \in (\alpha,1)$*,* $\beta > 0$*, and* $r = \Big\lfloor \frac{\delta}{\alpha}\cdot\frac{(2/\nu+\alpha)(1-\gamma)}{\beta\cdot(2/(\alpha\nu)+1)+2/\nu-1}\Big\rfloor - 1$*,* HORN *achieves* $\mathbb{E}\{Q_{\texttt{HQA}}\} \in (N_{i^*})^{\delta+\mathrm{o}(1)}$*,* $P_{\texttt{HORN}}$ *,* $S_{\texttt{HORN}} \in n^{2-\beta+\mathrm{o}(1)}$ *and stretch* $1+\varepsilon\frac{(1+\varepsilon/\psi)^{r+1}}{(1+\varepsilon/\psi)^{r+1}-1}$*, with probability at least* $1-\mathcal{O}\big(\frac{1}{n}\big)$*.*

References

1 R. Agarwal, P. Godfrey. Distance oracles for stretch less than 2. *ACM-SIAM SODA*:526–538 (2013)

2 Y. Bartal, L. A. Gottlieb, T. Kopelowitz, M. Lewenstein, L. Roditty. Fast, precise and dynamic distance queries. *ACM-SIAM SODA*:840-853 (2011)

3 H. Bast, D. Delling, A. V. Goldberg, M. Müller-Hannemann, T. Pajor, P. Sanders, D. Wagner, R. Werneck. Route planning in transportation networks. Technical Report MSR-TR-2014-4, Microsoft Research, `http://arxiv.org/abs/1504.05140` (2015)

4 G. V. Batz, R. Geisberger, P. Sanders, C. Vetter. Minimum time-dependent travel times with contraction hierarchies. *ACM J. of Exp. Algorithmics*, **18**:1–43 (2013)

5 K. Cooke, E. Halsey. The shortest route through a network with time-dependent intermodal transit times. *Math. Anal. and Appl.*, **14**(3):493-498 (1966)

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

7 B. C. Dean. Shortest paths in FIFO time-dependent networks: Theory and algorithms. Technical Report, MIT (2004)

8 F. Dehne, O. T. Masoud, and J. R. Sack. Shortest paths in time-dependent FIFO networks. *Algorithmica*, **62**(1-2):416–435 (2012)

9 D. Delling. Time-Dependent SHARC-Routing. *Algorithmica*, **60**(1):60–94 (2011)

10 D. Delling, D. Wagner. Time-dependent route planning. *Robust and Online Large-Scale Optimization*, **LNCS 5868**:207–230, Springer (2009)

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

12 L. Foschini, J. Hershberger, S. Suri. On the complexity of time-dependent shortest paths. *Algorithmica*, **68**(4):1075–1097 (2014)

13 J. Halpern. Shortest route with tme dependent length of edges and limited delay possibilities in nodes. *Zeitschrift für Operations Research*, **21**:117–124 (1977)

14 S. Kontogiannis, G. Michalopoulos, G. Papastavrou, A. Paraskevopoulos, D. Wagner, C. Zaroliagis. Analysis and experimental evaluation of time-dependent distance oracles. *ALENEX*:147–158 (2015)

15 S. Kontogiannis, G. Michalopoulos, G. Papastavrou, A. Paraskevopoulos, D. Wagner, C. Zaroliagis. Engineering oracles for time-dependent road networks. *ALENEX*:1–14 (2016)

16 S. Kontogiannis, D. Wagner, C. Zaroliagis. Hierarchical Time-Dependent Oracles. CoRR `http://arxiv.org/abs/1502.05222` (2015)

17 S. Kontogiannis, C. Zaroliagis. Distance oracles for time-dependent networks. *Algorithmica*, **74**(4):1404-1434 (2016).

18 G. Nannicini, D. Delling, L. Liberti, D. Schultes. Bidirectional A* search on time-dependent road networks. *Networks*, **59**:240–251 (2012)

19 M. Omran, J. R. Sack. Improved approximation for time-dependent shortest paths. *COCOON*, **LNCS 8591**:453-464, Springer (2014)

20 A. Orda, R. Rom. Shortest-path and minimum delay algorithms in networks with time-dependent edge-length. *J. of the ACM*, **37**(3):607–625 (1990)

21 M. Patrascu, L. Roditty. Distance oracles beyond the Thorup–Zwick bound. *IEEE FOCS*:815-823 (2010)

22 E. Porat, L. Roditty. Preprocess, set, query! *ESA*, **LNCS 6942**:603-614, Springer (2011)

23 H. Sherali, K. Ozbay, S. Subramanian. The time-dependent shortest pair of disjoint paths problem: Complexity, models, and algorithms. *Networks*, **31**(4):259–272 (1998)

24 C. Sommer. Shortest-path queries in static networks. *ACM Comp. Surv.*, **46** (2014)

25 C. Sommer, E. Verbin, W. Yu. Distance oracles for sparse graphs. *IEEE FOCS*:703–712 (2009)

26 M. Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *J. of the ACM*, **51**(6):993–1024 (2004)

27 M. Thorup, U. Zwick. Approximate distance oracles. *J. of the ACM*, **52**(1):1–24 (2005)

28 C. Wulff-Nilsen. Approximate distance oracles with improved preprocessing time. *ACM-SIAM SODA*:202–208 (2012)

29 C. Wulff-Nilsen. Approximate distance oracles with improved query time. *ACM-SIAM SODA*:539–549 (2013)