# Analysis and Experimental Evaluation of Time-Dependent Distance Oracles*

Spyros Kontogiannis[†,◇] George Michalopoulos[‡,◇] Georgia Papastavrou[†,◇]
Andreas Paraskevopoulos[‡,◇] Dorothea Wagner[◊] Christos Zaroliagis[‡,◇]

**Abstract**

Urban road networks are represented as directed graphs, accompanied by a metric which assigns cost *functions* (rather than scalars) to the arcs, e.g. representing time-dependent arc-traversal-times. In this work, we present oracles for providing *time-dependent* min-cost route plans, and conduct their experimental evaluation on a real-world data set (city of Berlin). Our oracles are based on precomputing all landmark-to-vertex shortest travel-time functions, for properly selected landmark sets. The core of this preprocessing phase is based on a novel, quite efficient and simple one-to-all approximation method for creating approximations of shortest travel-time functions. We then propose three query algorithms, including a PTAS, to efficiently provide min-cost route plan responses to arbitrary queries. Apart from the purely algorithmic challenges, we deal also with several implementation details concerning the digestion of raw traffic data, and we provide heuristic improvements of both the preprocessing phase and the query algorithms. We conduct an extensive, comparative experimental study with all query algorithms and six landmark sets. Our results are quite encouraging, achieving remarkable speedups (at least by two orders of magnitude) and quite small approximation guarantees, over the time-dependent variant of Dijkstra's algorithm.

## 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., [38, 36, 31, 32, 39, 40, 3]) is devoted to constructing distance oracles for *static* (i.e., *time-independent*), mostly undirected networks in which the arc-costs are fixed scalars, providing trade-offs between the oracle's space and query time and, in case of approximate oracles, also of the stretch. For an overview of distance oracles for static networks, the reader is deferred to [35] and references therein.

Considerable experimental work on routing in large-scale road networks has also appeared in recent years, with remarkable achievements that have been demonstrated on continental-size road-network instances. The goal is again to preprocess the distance metric and then propose query algorithms (known as *speedup techniques* in this framework) for responding to shortest path queries in time that is several orders of magnitude faster than a conventional Dijkstra run. An excellent overview of this line of research is provided in [5]. Once more, the bulk of the literature concerns static distance metrics, with only a few exceptions (e.g., [7, 14, 27]) that will be discussed later in more detail.

### 1.1 Modelling the Time-Variance of the Cost Metric.

In many real-world applications, the arc costs may vary as functions of time (e.g., when representing the variability of arc traversal times, temporal unavailability of a particular arc, etc.) giving rise to *time-varying* network models. A striking example is route planning in road networks where the travel-time for traversing an arc $uv$ (modelling a road segment) depends on the temporal traffic or availability conditions while attempting to traverse $uv$, and thus on the departure time from its tail $u$. Consequently, the min-cost path from an origin $o$ to a destination $d$ may vary with the departure-time $t_o$ from the origin. In this work, we consider the *time-dependent network model*, in which every arc $uv$ comes with an arc-traversal-time function $D[uv]$, whereas each path-traversal-time function is simply the *composition* of the corresponding arc-traversal-time functions of its constituent arcs. The *Time Dependent Shortest Path* (TDSP) problem concerns computing an $od-$path attaining the *earliest arrival time* at $d$,

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

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

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

◇ Computer Technology Institute and Press "Diophantus", 26504 Rio, GREECE.

◊ Karlsruhe Institute of Technology (KIT), Am Fasanengarten 5, 76131 Karlsruhe, GERMANY. dorothea.wagner@kit.edu

for an arbitrary triple $(o, d, t_o)$ of an origin-destination pair of vertices $(o, d) \in V \times V$ and departure-time $t_o \in \mathbb{R}$ from the origin, in a time-dependent network model $(G = (V, A), (D[a] : \mathbb{R} \to \mathbb{R}_{>0})_{a \in A})$ . The problem has been studied since a long time ago (see e.g., [9, 16, 30]). The shape of arc-travel-time functions and the waiting policy at vertices may considerably affect the tractability of the problem [30]. It is customary to consider as arc-travel-time functions the *continuous, piecewise linear (pwl) interpolants* of periodically sampled arc-travel-times. Regarding the waiting policy, a crucial assumption is that each arc obeys the *FIFO property*, according to which the earliest-arrival-time function of an arc $uv$ is an increasing function of its departure time $t_u$ from the tail $u$. Non-FIFO policies may lead to **NP**−hard cases [34]. On the other hand, in FIFO network models in which all the arc-travel-time functions possess the FIFO property, there is no need for waiting at either the origin or at intermediate nodes of the chosen path. Then, the problem can be solved in polynomial time by a straightforward variant of Dijkstra's algorithm (we call it `TDD`), which relaxes arcs by computing the arc costs "on the fly", when settling their tails [16].

Apart from the theoretical challenge, the time-dependent network model with FIFO-abiding, continuous, pwl arc-travel-time functions, is also much more appropriate with respect to handling the historic traffic data that the route planning vendors have to digest in order to provide their customers with fast route plans within milliseconds. For example, TomTom's *LiveTraffic* service[1] 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.

**1.2 Alternative time-varying network models.** We now briefly mention the most characteristic attempts, apart from the time-dependent shortest paths, to model shortest-path computations in time-varying network models.

In the *dynamic shortest path* problem (e.g., [15, 21, 33, 37]), the arcs are allowed to be inserted to and/or deleted from the graph in an online fashion. The focus is on maintaining and efficiently updating a data structure representing the shortest path tree from a single source, or at least supporting fast shortest path queries between arbitrary vertices, in response to these changes. The main difference with TDSP is exactly the online fashion of the changes in the characteristics of the metric.

In a *temporal network model* (e.g., [20, 25, 4]), each arc comes with a vector of discrete arc-labels determining the time-slots of its availability. The goal is then to study the reachability and/or computation of shortest paths for arbitrary pairs of vertices, given that the chosen connecting path must possess at least one non-decreasing subsequence of arc-labels, as we move from the origin to the destination. This problem is indeed a special case of TDSP, in the sense that the availability patterns may be encoded as arc-travel-time functions which switch between a finite and an infinite traversal cost. Typically these problems do not possess the FIFO property, but one may exploit the discretization of the time axis, which essentially determines the complexity of the instance to solve.

In the *stochastic shortest path* problem (e.g., see [8, 28, 29]) the uncertainty of the arc weights is modeled by considering them as random variables. The goal is again the computation of paths minimizing, not just the expected cost, but possibly more complex functions of the travel-time variables' moments. This is certainly a hard problem, but in the time-dependent shortest path there is no uncertainty on the behavior of the arcs and this is exactly what is meant to be exploited.

In the *parametric shortest path* (PSP) problem, the network comes with two distinct (but fixed) arc-cost metrics. The goal is to provide succinct representations, for any possible convex combination of the two metrics, of shortest paths between arbitrary origin-destination pairs. It is well known [26] that a shortest $od$−path may change $|V|^{\Omega(|V|)}$ times as the parameter of the convex combination on the two metrics varies. An upper bound of at most $|V|^{\mathcal{O}(|V|)}$ is also well-known [19]. The main difference with the *time-dependent shortest path* problem studied in the present work is that, when computing path costs, rather than (essentially) composing the arrival-time functions of the constituent arcs, in PSP the arc-costs are simply added.

**1.3 Related work.** Until recently, most of the previous work on the time-dependent shortest path problem concentrated on computing an optimal origin-destination path providing the earliest-arrival time at destination when departing at a *given* time from the origin, neglecting the computational complexity of providing succinct representations of the entire earliest-arrival-time (or equivalently for FIFO networks, shortest-travel-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

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

valuable for the construction of *travel-time 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 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 [10, 12, 11], but was solved only recently by a seminal work [18]. In particular, it was shown that, for FIFO-abiding pwl arc-travel-time functions, the problem has space-complexity $(1 + K) \cdot n^{\Theta(\log n)}$ for a *single* origin-destination pair, where $n$ is the number of vertices and $K$ is the total number of breakpoints of all the continuous, pwl arc-travel-time functions. Polynomial-time algorithms (or even PTAS) for constructing *point-to-point* $(1 + \varepsilon)$-approximate shortest-travel-time functions are provided in [18, 13], delivering point-to-point travel-time values at most $1 + \varepsilon$ times the true values. These functions indeed 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). Of course, the succinctness in this representation heavily depends on the value of $K^*$. E.g., for $K^* \in \mathcal{O}(\text{polylog}(n))$, clearly these point-to-point approximation methods would work very well. Things become harder though for instances with more concavity-spoiling breakpoints, e.g. when $K^* \in \Omega(n)$.

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

Exploiting a PTAS (such as that in [18]) for computing travel-time summaries, 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 precomputing and storing travel-time summaries from all possible origins. At the other extreme, one might use the minimum possible space complexity $S \in \mathcal{O}(n + m + K)$ for just 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 continuous, pwl functions). $K_{\max}$ denotes the maximum number of breakpoints in an arc-travel-time function. The main challenge for a time-depenent oracle is thus to smoothly close the gap between these two extremes, i.e., to achieve a better (e.g., *sublinear*) query-time complexity, while consuming smaller space-complexity, e.g., $o\big(n^2\big)$, for succinctly representing travel-time summaries, while enjoying a small, e.g., close to 1, approximation guarantee (stretch factor). It would also be crucial to avoid the dependence on the amount of discncavity in the travel-time metric, as expressed by the value of $K^*$, at least for instances in which $K^* \in \Omega(n)$.

Providing distance oracles for time-dependent networks with *provably* good approximation guarantees, small preprocessing-space complexity and sublinear query time complexity, has only been recently investigated in [22, 23]. In particular, the *first* approximate distance oracle for sparse directed graphs with time-dependent arc-travel-times was presented in [23], providing $(1 + \sigma)-$approximate travel-times in query-time that is *sublinear* in the network size, and preprocessing time and space that are *subquadratic* in the network size, when the total number of concavity-spoiling breakpoints in the instance is sufficiently small, e.g. when $K^* \in \mathcal{O}(\text{polylog}(n))$. The oracle uses a novel *one-to-all* method (called *Bisection* – `BIS`) to produce $(1 + \varepsilon)-$approximate landmark-to-vertex travel-time summaries, for a randomly selected landmark set. It also guarantees either constant approximation ratio (a.k.a *stretch*) via the `FCA` query algorithm, or stretch at most $1 + \sigma = 1 + \varepsilon\frac{(1+\varepsilon/\psi)^{r+1}}{(1+\varepsilon/\psi)^{r+1}-1}$ via the `RQA` query algorithm, where $\psi$ is a fixed constant depending on the characteristics of the arc-travel-time functions but is independent of the network size, and $r \in \mathcal{O}(1)$ is the recursion depth of `RQA`. In [22], another oracle is proposed, providing both constant and $(1 + \sigma)-$approximate travel-times in query-time that is *sublinear* in the network size, and preprocessing time and space that are *subquadratic* in the network size, independently of the amount of disconcavity $K^*$ in the network instance at hand. This is achieved by combining `BIS` with another *one-to-all* method (called *Trapezoidal* – `TRAP`) to produce $(1 + \varepsilon)-$approximate landmark-to-vertex travel-time summaries.

A few time-dependent variants of well-known speedup techniques for road networks have also appeared in the literature (e.g., [7, 14, 27]). All of them were experimentally evaluated on synthetic time-dependent instances of the European and German road networks, with impressive performances. For example, in [7] methods are provided that respond to arbitrary queries of the German road network (4.7 million vertices and 10.8 million arcs) in less than $1.5ms$ and preprocessing space requirements of less than 1GB. A point-to-point travel-time summary (a.k.a. search pro-

file) can also be constructed in less than $40ms$, when the departure times interval is a single day. For point-to-point approximate travel-time summaries, with experimentally observed stretch at most 1%, the construction time is less than $3.2ms$. Their approach is based on the so-called *time-dependent Contraction Hierarchies* [6], along with several heuristic improvements both on the preprocessing step and on the query method.

**1.4 Our Contribution.** Our goal in this work is to provide a thorough experimental evaluation of the time-dependent distance oracles that were proposed and analysed in [23]. The main obstacle towards this direction is the dependence of the required preprocessing time and space on the number $K^*$ of concavity-spoiling breakpoints in the raw traffic data.

Inspired by the theoretical analysis of [22], our first contribution is to propose a new time-dependent distance oracle whose preprocessing phase for computing landmark-to-vertex approximate travel-time summaries is solely based on a new approximation technique [22], the *trapezoidal* (`TRAP`) method. This method is significantly simpler than `BIS` and reduces dramatically the required space. In particular, `TRAP` avoids any kind of dependence on the number $K^*$ of concavity-spoiling breakpoints, which are completely neglected during the preprocessing and need not be computed at all. Based on `TRAP`, we build new time-dependent distance oracles, which preprocess landmark-to-vertex approximate travel-time summaries for various landmark sets, and again employ the `FCA` and `RQA` query algorithms that were proposed in [23]. Additionally, we propose another quite simple query algorithm, $\texttt{FCA}^+$. Although the theoretical guarantee of its stretch factor is analogous to that of `FCA`, in practice it behaves very well, sometimes even better than `RQA`.

Our second contribution is an extensive experimental study of the above mentioned query algorithms for six different landmark sets, achieving remarkable speedups over `TDD` on truly real-world time-dependent data sets. In particular, we conduct our experimental evaluation on the historic traffic data for the city of Berlin, kindly provided to us by TomTom within [17]. The input instance is a directed graph with $478,989$ vertices and $1,134,489$ arcs. The provided raw traffic data for the arcs were stored as integer values for two different levels of resolution, one considering $10.3msec$, and another using $2.64sec$, as the time unit. We created six different landmark sets with 1000 or 2000 landmarks, which were chosen either randomly, or as the boundary vertices of appropriate METIS [1] or KaHIP [2] partitions of the Berlin graph. The speedups that we observed for our query algorithms over the average time of a `TDD` run, vary from 397 times (using 1000 randomly chosen landmarks and `FCA`), to 723 times (using 2000 randomly chosen landmarks and `FCA`), for $10.3ms$ resolution in the approximate travel-time summaries. In both cases the average relative error is less than 1.634%. Analogous speedups are observed if our quality measure is not the computational time, but the (machine-independent) number of settled vertices (a.k.a. Dijkstra rank) of the query algorithms. The best possible observed relative error is indeed much better than the theoretical bounds provided by the analysis of the query algorithms. In particular, it is as small as 0.382% for 1000 KaHIP landmarks, or 0.298% for 2000 KaHIP landmarks, for $10.3ms$ resolution in the approximate travel-time summaries. The corresponding speedups are 38 for the former, and 118 for the latter.

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

As for the preprocessed data, we create and succinctly store roughly $300K$ approximate travel-time summaries from a given landmark, in average sequential time less than $40sec$. That is, the amortized sequential time per approximate travel-time summary is no more than $0.134ms$.

**1.5 Paper Organization.** Section 2 provides some notation and preliminaries for time-dependent network instances. Section 3 provides an overview of the `TRAP` approximation method for producing approximate travel-time summaries, the preprocessing phase for producing all landmark-to-vertex summaries, and the query algorithms `FCA`, $\texttt{FCA}^+$ and `RQA`, to be experimentally tested. Section 4 presents the results of our experimental experimental evaluation on the real instance of Berlin. Section 5 provides some concluding remarks and directions for further research and experimentation.

## 2 Preliminaries

We consider directed graphs $G = (V, A)$ with $|V| = n$ vertices and $|A| = m$ arcs, where each arc $a \in A$ is accompanied with a continuous, periodic, piecewise linear (pwl) *arc-travel-time* (or *arc-delay*) function defined as follows: $\forall k \in \mathbb{N}, \forall t \in [0, T),\ D[a](kT + t) = d[a](t)$, where $d[a] : [0, T) \to [1, M_a]$ such that $\lim_{t \uparrow T} d[a](t) = d[a](0)$, for some fixed integer $M_a$ denoting the maximum possible travel time ever observed at arc $a$. Notice that the minimum arc travel time value in the entire network is also normalized to 1. Each arc-travel-time

function $D[a]$ can be represented succinctly as a list of $K_a$ breakpoints defining $d[a]$. Let $K = \sum_{a \in A} K_a$ be the number of breakpoints to represent all of them, $K_{\max} = \max_{a \in A} K_a$, and $K^*$ be the number of *concavity-spoiling* breakpoints, i.e., those in which the arc-travel-time slopes increase. Clearly, $K^* \leq K$, and $K^* = 0$ for *concave* pwl arc-travel-time functions.

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

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

- *Free-flow* arc-travel-time:

$$\underline{D}[uv](t_s, t_f) := \min_{t_u \in [t_s, t_f)} D[uv](t_u)\,.$$

- *Maximally-congested* arc-travel-time:

$$\overline{D}[uv](t_s, t_f) := \max_{t_u \in [t_s, t_f)} D[uv](t_u)\,.$$

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

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

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

We adopt two assumptions from [23] and one additional assumption from [22], on the kind of shortest-travel-time functions that may appear in the time-dependent network instance at hand. All of them are quite natural and justified in urban-traffic road networks. Indeed, we conducted an experimental analysis on the real-world instance of Berlin that we had at our disposal, which verified the validity of the assumptions. Technically, these assumptions allow the *smooth transition* from static metrics on undirected graphs towards time-dependent metrics on directed graphs. For a more thorough justification, the reader is deferred to [23, 22].

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

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

It is mentioned that the lower-bound of $-1$ in the shortest-travel-time function slopes is indeed a direct consequence of the strict FIFO property, which is typically assumed to hold in several time-dependent networks and allows for the use of time-dependent variants of classical shortest-path computation techniques, such as Dijkstra's and Bellman-Ford algorithms. Our experimental analysis on the historic traffic data for the city of Berlin, in which the maximum value of $\Lambda_{\max}$ in a series of $10,000$ randomly chosen origin-destination pairs was always less than 0.19.

The second assumption asserts that for any given departure time, the shortest-travel-time from $o$ to $d$ is not more than a *constant* $\zeta \geq 1$ times the shortest-travel-time in the opposite direction (but not necessarily along the reverse path). This is quite natural in road networks. E.g., it is most unlikely that a trip in one direction is more than, say, 10 times longer than the trip in the opposite direction for the same departure time. The assumption was also confirmed by our experimental analysis on the historic traffic data for the city of Berlin, in which the maximum value of $\zeta$ in a series of 10000 randomly chosen origin-destination pairs was always less than 1.5.

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

One last assumption concerns the relation of the Dijkstra ranks (i.e., number of settled vertices, up to

termination) of cocentric balls in the network, with respect to the (static) *free-flow* metric implied by the time-dependent instance at hand:

ASSUMPTION 2.3. (GROWTH OF FREE-FLOW BALLS) *For any vertex $\ell \in V$ and positive integer $F \in \mathbb{N}$, assume growing a free-flow Dijkstra ball $\underline{B}[\ell](F)$ around $\ell$, of size $F$. Let $\underline{R}[\ell] = \max\{\underline{D}[\ell, v] : v \in \underline{B}[\ell](F)\}$ be the* free-flow radius *in $\underline{B}[\ell](F)$. Also let $\overline{R}[\ell] = \max\{\overline{D}[\ell, v] : v \in \underline{B}[\ell](F)\}$ be the* full-congestion radius *in $\underline{B}[\ell]$. Finally, $\underline{B}' = \{v \in V : \underline{D}[\ell, v](0, T) \leq \overline{R}[\ell]\}$ is the free-flow ball with radius $\overline{R}[\ell]$ around $\ell$. Then it holds that $|\underline{B}'[\ell]| \in \mathcal{O}(F \cdot \text{polylog}(F))$.*

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

## 3 Time-Dependent Oracles

In [22] time-dependent distance oracles are proposed and theoretically analysed, which preprocess the travel-times metric using both the `BIS` method (for nearby destinations) proposed in [23] and the novel `TRAP` method (for faraway destinations) to approximate shortest travel-time functions, and then use one of two query algorithms (`FCA` or `RQA`) for efficiently responding to arbitrary queries. The novelty of these oracles is that they assure subquadratic storage space and sublinear query complexity, irrespectively of the degree of disconcavity of the travel-time metric, measured by the value of $K^*$. In this work we experimentally evaluate these oracles exploiting exclusively the `TRAP` method for creating travel-time summaries, and also experiment with an additional query algorithm, called `FCA`$^+$, that we propose.

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

**3.1 Approximate Travel-Time Functions via the Trapezoidal Method.** We briefly present here the novel preprocessing step of our oracles which, based on the `TRAP` method, constructs $(1 + \varepsilon)-$upper-approximations of shortest travel-time functions (cf. [22] for a detailed presentation and analysis). The performance of this new preprocessing phase is practically *independent* of the degree of disconcavity of the instance as expressed by $K^*$.

`TRAP` splits the entire period $[0, T)$ into small, consecutive subintervals of length $\tau > 0$ each. It then provides a crude approximation of the unknown shortest-travel-time functions in each interval, solely based on Assumption 2.1 concecning the boundedness of the shortest travel-time slopes in the instance. After sampling the travel-time values of each destination $v \in V$, for a given origin $u \in V$, we consider each pair of consecutive sampling times $t_s < t_f$ and the semilines with slopes $\Lambda_{\max}$ from $t_s$ and $-\Lambda_{\min}$ from $t_f$. The considered upper-approximating function $\overline{D}[u, v]$ within $[t_s, t_f)$ is then (a refinement of) the lower-envelope of these two lines. Analogously, a lower-approximating function $\underline{D}[u, v]$ is the upper-envelope of the semilines that pass through $t_s$ with slope $-\Lambda_{\min}$, and from $t_f$ with slope $\Lambda_{\max}$. Depending on the value of the absolute error and the minimum possible value of $\underline{D}[u, v]$ in this interval, we can decide whether $\overline{D}[u, v]$ is a $(1+\varepsilon)$-upper-approximating function of $D[u, v]$. Any destination vertex that has such a $(1 + \varepsilon)$-upper-approximating function for each subinterval of $[0, T)$, clearly has a $(1 + \varepsilon)$-upper-approximating function for the entire period as well. The proof of correctness of `TRAP` is provided in [22].

The problem with the trapezoidal approximation is that, by construction, it is not possible to provide $(1 + \varepsilon)$-approximate travel-time functions for "nearby" destination vertices, which are too close to the origin. In [22] these "nearby" vertices of each landmark are either handled by the `BIS` method [23], or are left to be handled by local `TDD` searches "on the fly". Here we resolve this issue exclusively with `TRAP`, starting with a large subinterval length, and then recursively dividing by 2 the lengths of those subintervals containing vertices which have not been sufficiently approximated yet, until all landmark-to-vertex $(1 + \varepsilon)$-approximate travel-time summaries have been successfully created. This proved to be extremely space- and time-efficient in practice.

**3.2 Query Algorithms.** For efficiently responding to arbitrary origin/destination/departure-time queries $(o, d, t_o)$, three approximation algorithms are consid-

ered. The first one, called FCA, is a simple *sublinear*-time constant-approximation algorithm, which works as follows. It grows a ball $B_o \equiv B[o](t_o) = \{x \in V : D[o,x](t_o) \leq D[o,\ell_o](t_o)\}$ from $(o,t_o)$, by running TDD until either $d$ or the closest landmark $\ell_o \in \arg\min_{\ell \in L}\{D[o,\ell](t_o)\}$ is settled. It then returns either the exact travel-time value, or the approximate travel-time value via $\ell_o$, achieving a $1+\varepsilon+\psi$ approximation guarantee as was shown in [23], where $\psi$ is a constant depending on $\varepsilon, \zeta$, and $\Lambda_{\max}$, but not on the size of the network.

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

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

A more detailed presentation of FCA and RQA, along with the proofs of correctness and their time complexities, are provided in [23]. As for the approximation guarantee of FCA$^+$, it is straightforward to observe that, at least theoretically, it is as small as that of FCA, whereas its time complexity is comparable to that of RQA.

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

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

## 4 Experimental Evaluation

The purpose of this section is to provide all the details of the experimental evaluation that we conducted on the three query algorithms that we propose, and for the TRAP-based preprocessing phase executed on six properly chosen landmark sets, of either $1,000$ or $2,000$ landmarks each.

**4.1 Preprocessing The Road Instance.** The Berlin instance, kindly provided by TomTom within [17], consists of a directed graph with $478,989$ vertices and $1,134,489$ arcs. We focused only on the strongly

connected component of this input graph, consisting of 473,253 vertices and 1,126,468 arcs. 924,254 of the arcs have constant arc-travel-times. For the remaining 202,214 arcs, continuous pwl arc-travel-time functions are provided, concerning an entire weekday (Tuesday). The maximum arc-travel-time slope is 0.0166667, whereas the minimum slope is $-0.0133333$. The succinct representation of these functions requires a total number of 3,234,213 breakpoints. We substituted each maximal path in the network consisting of intermediate vertices with no intersections (i.e., would have degree 2 in the simple, undirected version of the graph), with a single shortcut arc of arc-travel-time function equal to the corresponding (exact) path-travel-time function. This resulted in a reduced graph consisting of 299,693 vertices and 950,504 arcs.

We generated two data formats suitable as input for our query algorithms. The first concerns the arc-travel-time functions and the second the preprocessed travel-time summaries (i.e., landmark-to-vertex $(1+\varepsilon)$-approximate shortest travel-time functions).

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

In order to avoid wasting space, for each arc and arc-travel-time value per timeslot, consecutive timeslots having the same arc-travel-time value were merged. Optionally, one could perform a broader merging of consecutive timeslots having absolute difference in arc-travel-time values less than a small constant (e.g. $< 1min$ resolution bound). However, in our experiments we chose to preserve the maximum possible resolution of the raw-traffic data. This proved to be extremely efficient by means of approximation guarantees, for different levels of resolution for the approximate travel-time summaries. The eventual space required for all the raw-traffic data provided as input, is roughly 225MB.

The arc-travel-time function $d[a](t)$ is simply the continuous, pwl interpolant of all the breakpoints corresponding to arc $a$. $D[a](t)$ is then the periodic repetition of $d[a](t)$.

**4.1.2 Preprocessed Landmark Information.** In order to create all the landmark-to-vertex $(1+\varepsilon)$-approximate shortest travel-time summaries, we call `TRAP`, which is a one-to-all approximation method, once per landmark. Upon completion of this preprocessing phase, we collect the $(1+\varepsilon)$-approximate travel-time summaries for all the landmark-vertex pairs in the Berlin graph. For each such pair $(\ell, v) \in L \times V$, we store a sequence $\langle$ ( $Dep[\ell]_i$ , $Arr[\ell, v]_i$ )$_i$ $\rangle$ of breakpoints, where $Dep[\ell]_i$ denotes a departure-time from landmark $\ell$ and $Arr[\ell, v]_i$ denotes the corresponding earliest-arrival-time at $v$. The interpolation of all these breakpoints produces the overall $(1+\varepsilon)$-upper-approximating travel-time summary (continuous, pwl function) $\Delta[\ell, v](t)$. With $|L|$ landmarks and $p$ breakpoints (on average) per approximate travel-time summary, the preprocessing space required for storing all the landmark-to-vertex approximate travel-time summaries is $\mathcal{O}(|L|pn)$.

Our approach is focused on achieving a cost-effective storage of these summaries, while keeping a sufficient precision. The key is that some specific features can be exploited in order to reduce the required space. The main observation is that, for a one-day time period, departure-times and arrival-times have a bounded value range. In particular, when the considered precision of the traffic data is within seconds we handle time-values as integers in the range [0 , 86,399], for milliseconds as integers in [0 , 86,399,999], etc.

Any (real) time value within a single-day period, represented as a floating-point number $t_f$, can thus be converted to an integer $t_i$ with fewer bytes and a given unit of measure. For a unit measure (or scale factor) $s$, the resulting integer is $t_i = \lceil t_f/s \rceil$. In this manner, $t_i$ needs size $\lceil \log_2(t_f/s)/8 \rceil$ bytes. The division $t_f/s$ has quotient $\pi$ and remainder $\upsilon$. Thus, $t_f = s \cdot \pi + \upsilon$ and $t_i = \lceil (s \cdot \pi + \upsilon)/s \rceil = \lceil \pi + \upsilon/s \rceil$, with $\upsilon < s$. Therefore, converting $t_f$ to $t_i$ results to an absolute error of at most $2s$. In the reverse process, for extracting the stored value, the conversion is $t_f^{'} = t_i \cdot s$. In our experiments, for storing the time-values of approximate travel-time summaries, we have considered two different resolutions:

(a) $2.64sec$ resolution, corresponding to a scale factor $s = 1.32$ (when counting time in seconds), requiring 2 bytes per time-value, and

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

**4.2 Experimental Setup.** All algorithms were implemented using C++ (gcc, version 4.6.3). To support all graph-operations we used the PGL library [24]. All experiments were executed by a CPU of 3.40GHz×8, using 16GB of RAM, on Ubuntu 12.04 LTS. All our algorithms are executed sequentially. Exploitation of parallelism is left for future implementations and is anticipated to reduce dramatically the execution times, particularly for the preprocessing phase and the query algorithm RQA for which parallelism would apply quite naturally.

**4.3 Measurements and Evaluation.** We now proceed with the presentation and discussion of our findings in the experimental evaluation that we conducted on the data set of Berlin, for the three query algorithms and the six landmark sets that we considered.

**4.3.1 Preprocessing Phase: Creation of Approximate Travel Time Summaries.** Our preprocessing phase took as input six different landmark sets for the Berlin graph: $R_{1000}$ and $R_{2000}$ correspond to $1,000$ and $2,000$ landmarks chosen uniformly at random from the entire vertex set. $M_{1000}$ and $M_{2000}$ correspond to $1,021$ and $2,072$ landmarks chosen as the boundary vertices of appropriate METIS partitions. $K_{1000}$ and $K_{2000}$ correspond to $1,016$ and $2,024$ landmarks chosen as the boundary vertices of appropriate KaHIP partitions.

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

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

**4.3.2 Query Phase: Responding to Arbitrary Shortest-Path Queries.** The query execution times and relative errors of the produced solutions, for all possible landmark sets and the two different resolutions that we consider for the approximate travel-time summaries, are presented in Tables 1 and 2. Moreover, Table 3 presents the speedups of the query algorithms, measured by the machine-independent criterion of Dijkstra-rank, i.e., the number of settled vertices during execution. All reported values are averages over $1,000$ randomly chosen queries from the Berlin instance. We note that for RQA the recursion budget was set to 1. For fairness of comparison, the parameter $N$ (number of landmarks) in FCA$^+$ was set equal to the number of landmarks settled by RQA.

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

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

$$100 \cdot \frac{\text{travel time of } p - \text{shortest travel time from } (o, t_o) \text{ to } d}{\text{shortest travel time from } (o, t_o) \text{ to } d}$$

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

travel-times from the origin, but shortest travel-times via particular parents. On the other hand, it should be noted that RQA is amenable to parallelization due to its recursive flavor. This is anticipated to speedup significantly the average query time in forthcoming implementations of RQA, which will then be comparable to that of FCA.

With respect to the relative error, we observe that for all the random landmark sets $\texttt{FCA}^+$ provides smaller values, of 0.449% for 1000 random landmarks and 0.389% for 2000 random landmarks. For the rest of the landmark sets, RQA is the best option with respect to the relative error, achieving values 0.314% for 1000 KaHIP landmarks and 0.298% for 2000 KaHIP landmarks. That is, the oriented expansion of the Dijkstra tree provided by RQA performs better in cases of landmark sets created from well structured partitions, whereas the brute-force expansion of $\texttt{FCA}^+$ is better for randomly chosen landmarks in the network.

As for the machine-independent performance of Dijkstra-ranks (cf. Table 3), we observe that the reported average speedups of our query algorithms, compared to a typical TDD run, are even better. For example, using FCA on $R_{1000}$ and $R_{2000}$ produce speedups larger than 429 and 889 times respectively. This is indeed quite encouraging, since all the proposed query algorithms are based on label-settings, just as TDD, and this performance measure is truly machine independent.

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

## 5 Conclusions and Future Work

In this work we have proposed novel distance oracles for sparse time dependent network instances corresponding mainly to road networks, which achieve sublinear average query time and subquadratic preprocessing space and time requirements. All the proposed techniques are sequential, and no attempt has been made so far to exploit the inherent potential of parallelization in them, which would significantly speed-up the execution times. In particular, parallelizing the preprocessing phase is straightforward and would significantly improve the adaptivity of our oracle to live-traffic reports of unforeseen disruptions (e.g., temporal congestion, or even blockage of a particular road segment). Moreover, there are still many possibilities of improving the required preprocessing space, by exploiting the one-to-all flavour of the constructed travel-time summaries. All these issues are part of our ongoing research towards truly efficient time-dependent distance oracles.

## References


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

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

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

[4] E. Akrida, L. Gasieniec, G. Mertzios, and P. Spirakis. Ephemeral networks with random availability of links: Diameter and connectivity. In *Proc. of 26th ACM Symp. on Par. Alg. and Archit. (SPAA '14)*, 2014.

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

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

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

[8] Dimitris Bertsekas and John Tsitsiklis. An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, 16(3):580–595, 1991.

[9] K. Cooke and E. Halsey. The shortest route through a network with time-dependent intermodal transit times. *Journal of Mathematical Analysis and Applications*, 14(3):493–498, 1966.

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

Table 1: Query performances for 10.3$ms$-resolution of the raw traffic data.

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

Table 2: Query performances for 2.64$sec$-resolution of the raw traffic data.

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

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

| | TDD | | FCA | | FCA$^+$ | | RQA | |
|---|---|---|---|---|---|---|---|---|
| | Rank | Speedup | Rank | Speedup | Rank | Speedup | Rank | Speedup |
| $R_{1000}$ | 149397 | 1 | **348** | **429.302** | 2628 | 56.848 | 4261 | 35.061 |
| $M_{1000}$ | | | 713 | 209.533 | 2517 | 59.355 | 5304 | 28.167 |
| $K_{1000}$ | | | 657 | 227.393 | 2353 | 63.492 | 4660 | 32.059 |
| $R_{2000}$ | | | **168** | **889.268** | 1251 | 119.422 | 1820 | 82.086 |
| $M_{2000}$ | | | 252 | 592.845 | 1039 | 143.789 | 1646 | 90.764 |
| $K_{2000}$ | | | 247 | 604.846 | 1002 | 149.099 | 1522 | 98.158 |

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

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

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

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

[15] Camil Demetrescu and Giuseppe Italiano. Dynamic shortest paths and transitive closure: An annotated bibliography (draft). `http://www.diku.dk/PATH05/biblio-dynpaths.pdf`, 2005.

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

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

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

[19] D.M. Gusfield. *Sensitivity analysis for combinatorial optimization*. PhD thesis, University of California, Berkeley, 1980.

[20] David Kempe, Jon Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. *J. Comp. Sys. Sci.*, 64:820–842, 2002.

[21] V. King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *Proc. of 40th IEEE Symp. on Found. of Comp. Sci. (FOCS '99)*, 1999.

[22] Spyros Kontogiannis, Dorothea Wagner, and Christos Zaroliagis. Hierarchical distance oracles for time-dependent networks. Technical Report, eCOMPASS Project, December 2014.

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

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

[25] George Mertzios, Othon Michail, Ioannis Chatzigiannakis, and Paul Spirakis. Temporal network optimization subject to connectivity constraints. In *Int. Col. on Aut., Lang. and Progr.*, 2013.

[26] Ketan Mulmuley and Pradyut Shah. A lower bound for the shortest path problem. *J. Comp. Sys. Sci.*, 63:253–267, 2001.

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

[28] Evdokia Nikolova, Matthew Brand, and David Karger. Optimal route planning under uncertainty. In International Conference on Automated Planning and Scheduling, 2006.

[29] Evdokia Nikolova, Jonathan Kelner, Matthew Brand, and Michael Mitzenmacher. Stochastic shortest paths via quasi-convex maximization. In *Proceedings of European Symposium of Algorithms*, pages 552–563. Springer, 2006.

[30] Ariel Orda and Raphael Rom. Shortest-path and minimum delay algorithms in networks with time-dependent edge-length. *Journal of the ACM*, 37(3):607–625, 1990.

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

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

[33] Liam Roditty and Uri Zwick. On Dynamic Shortest Paths Problems. In *Proceedings of the 12th Annual European Symposium on Algorithms (ESA'04)*, volume 3221 of *Lecture Notes in Computer Science*, pages 580–591, 2004.

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

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

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

[37] Mikkel Thorup. Worst-case update times for fully-dynamic all-pairs shortest paths. In *Proc. of 37th ACM Symp. on Th. of Comp. (STOC '05)*, pages 112–119, 2005.

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

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

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