An Axiomatic Approach to Time-Dependent Shortest Paths

Christos Zaroliagis

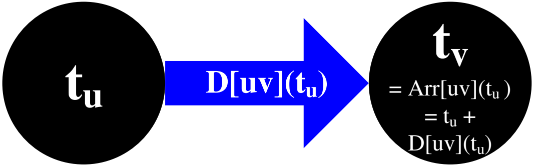
zaro@ceid.upatras.gr

Dept. of Computer Engineering & Informatics
University of Patras, Greece

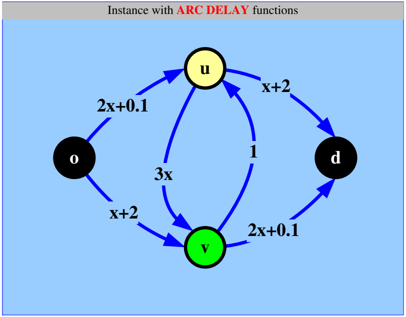
Computer Technology Institute & Press
“Diophantus”

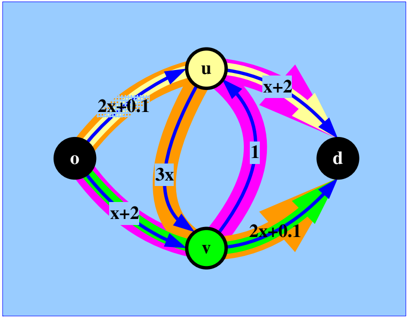
Time-Dependent Arc-Delay and Arrival Functions

- Directed graph $G = (V, A)$, $n = |V|$, $m = |A|$
- Arc (u, v)



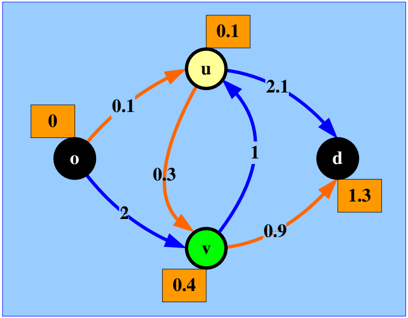
Time-Dependent Shortest Paths

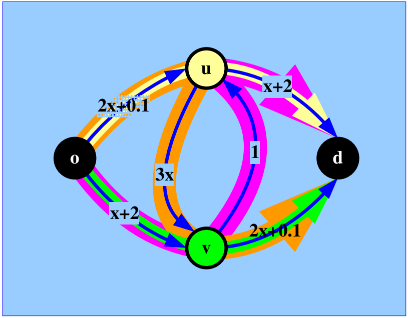




Q1 How would you commute **as fast as possible** from *o* to *d*, for a given departure time (from *o*)?

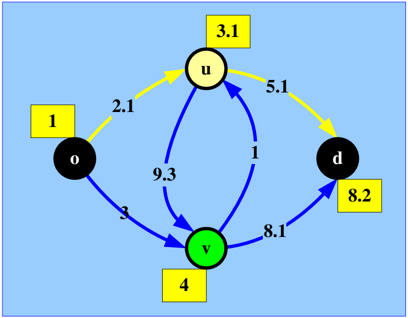
Time-Dependent Shortest Paths

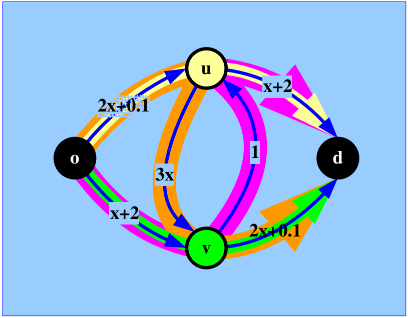




Q1 How would you commute **as fast as possible** from *o* to *d*, for a given departure time (from *o*)? Eg: $t_o = 0$

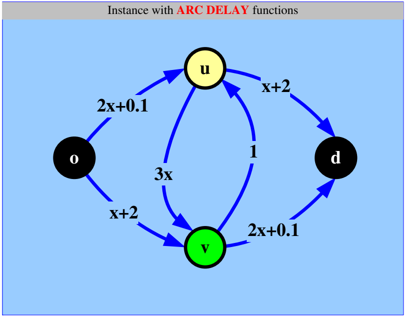
Time-Dependent Shortest Paths

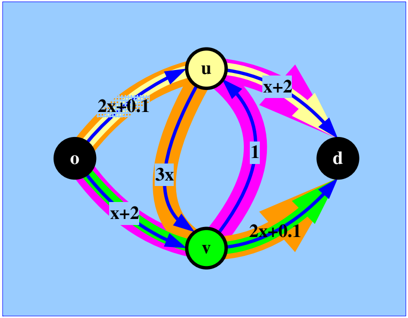




Q1 How would you commute **as fast as possible** from *o* to *d*, for a given departure time (from *o*)? Eg: $t_o = 1$

Time-Dependent Shortest Paths

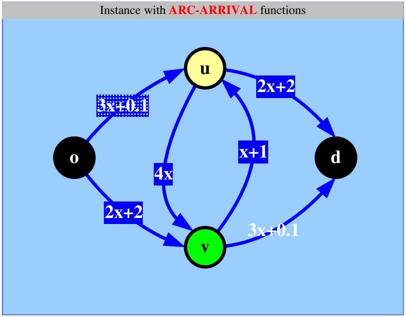


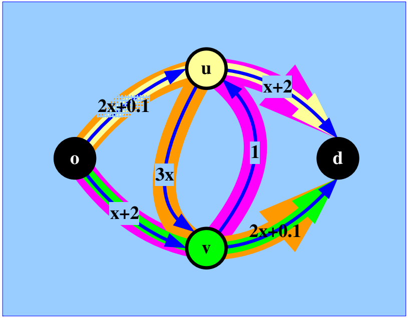


Q1 How would you commute **as fast as possible** from *o* to *d*, for a given departure time (from *o*)?

Q2 What if you are not sure about the departure time?

Time-Dependent Shortest Paths

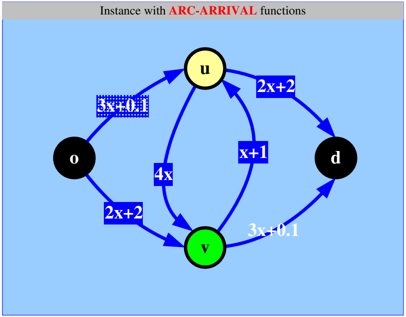


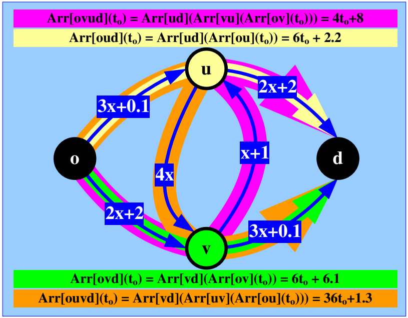


Q1 How would you commute **as fast as possible** from *o* to *d*, for a given departure time (from *o*)?

Q2 What if you are not sure about the departure time?

Time-Dependent Shortest Paths

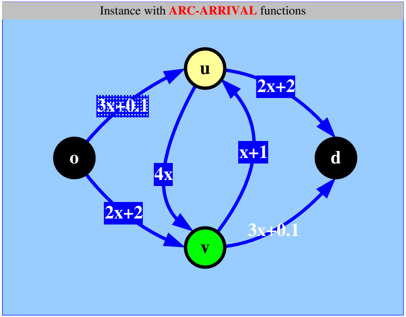


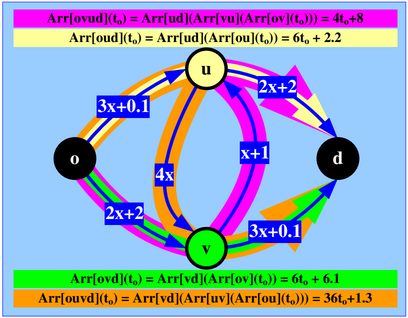


Q1 How would you commute **as fast as possible** from *o* to *d*, for a given departure time (from *o*)?

Q2 What if you are not sure about the departure time?

Time-Dependent Shortest Paths



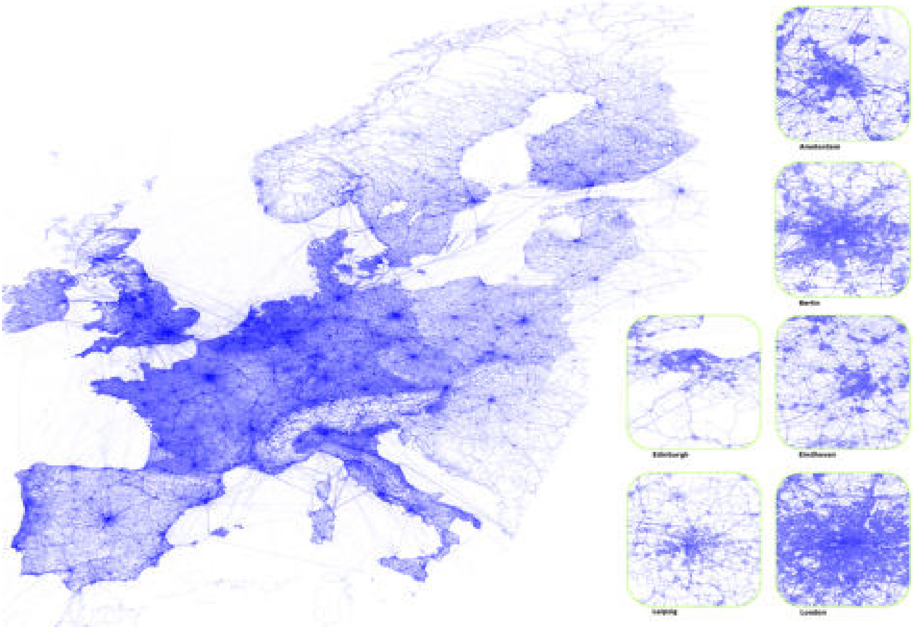


Q1 How would you commute **as fast as possible** from o to d, for a given departure time (from o)?

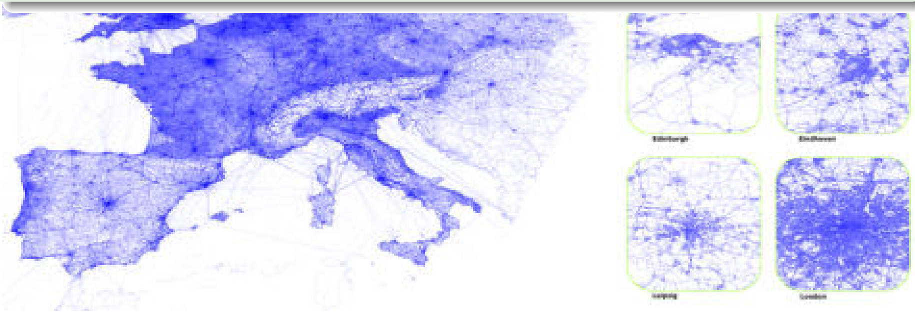
Q2 What if you are not sure about the departure time?

A

$$\text{shortest } od\text{–path} = \begin{cases} \textbf{orange path}, \text{ if } & t_o \in [0, 0.03) \\ \textbf{yellow path}, \text{ if } & t_o \in [0.03, 2.9) \\ \textbf{purple path}, \text{ if } & t_o \in [2.9, +\infty) \end{cases}$$

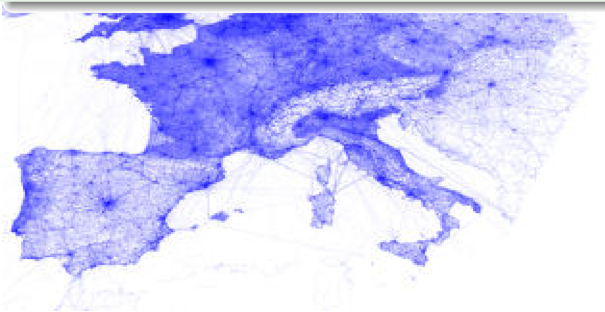


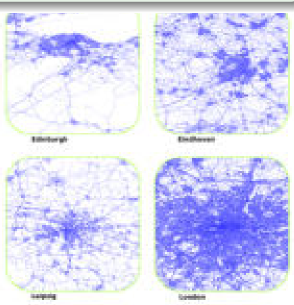
Raw traffic (speed probe) data



Raw traffic (speed probe) data

- **70 Million** contributing users provide periodic measurements



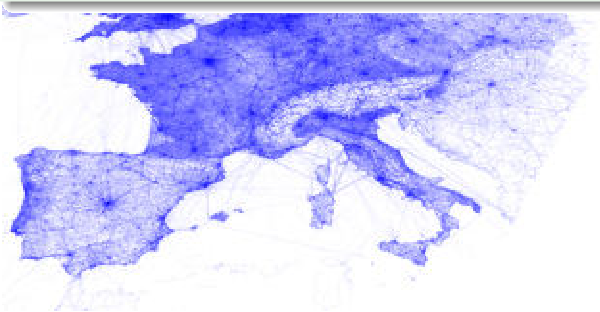


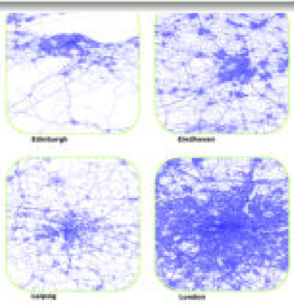
Raw traffic (speed probe) data

- **70 Million** contributing users provide periodic measurements
- Measured speed and position every **5-mins** (for each road segment)

Raw traffic (speed probe) data

- **70 Million** contributing users provide periodic measurements
- Measured speed and position every **5-mins** (for each road segment)
- Every road segment measured $\sim$ 2000 times per week



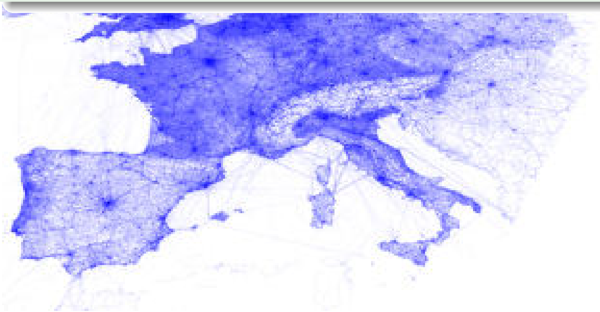


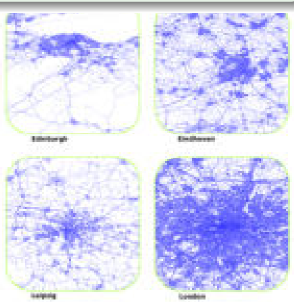
Raw traffic (speed probe) data

- **70 Million** contributing users provide periodic measurements
- Measured speed and position every **5-mins** (for each road segment)
- Every road segment measured ~ 2000 times per week
- **5 Trillion** measurements in **historic data** over 140 Billion Km

Raw traffic (speed probe) data

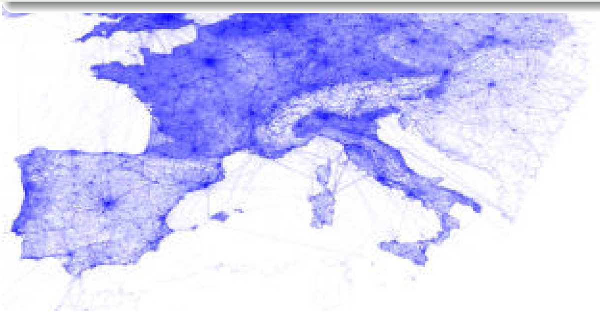
- **70 Million** contributing users provide periodic measurements
- Measured speed and position every **5-mins** (for each road segment)
- Every road segment measured ~ 2000 times per week
- **5 Trillion** measurements in **historic data** over 140 Billion Km
- **4 Billion** new measurements per day

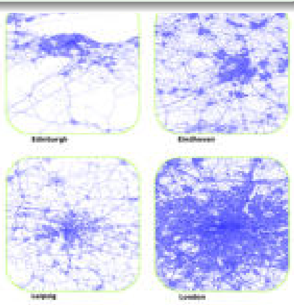




Raw traffic (speed probe) data

- **70 Million** contributing users provide periodic measurements
- Measured speed and position every **5-mins** (for each road segment)
- Every road segment measured ~ 2000 times per week
- **5 Trillion** measurements in **historic data** over 140 Billion Km
- **4 Billion** new measurements per day

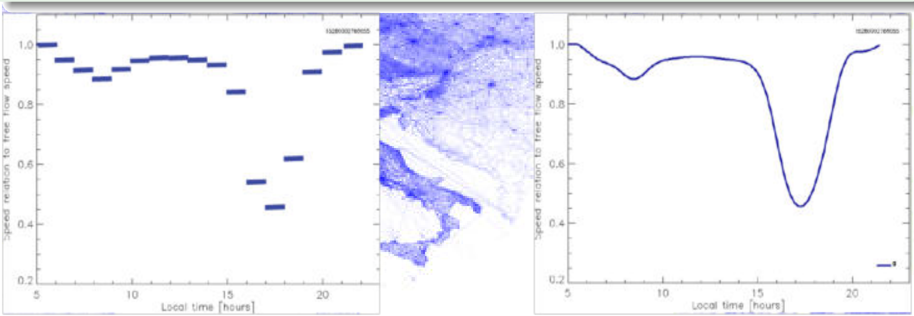




Raw traffic (speed probe) data

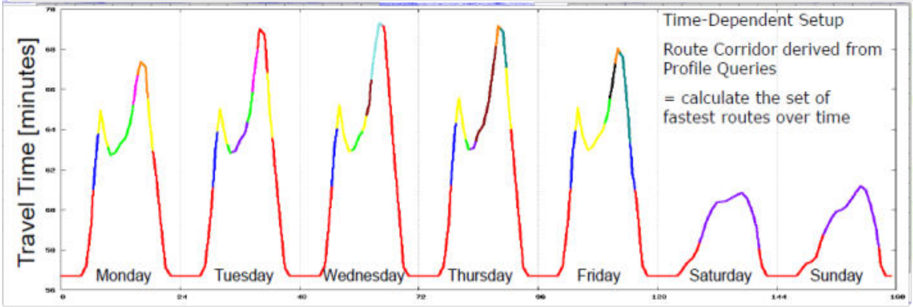
TOMTOM

- **70 Million** contributing users provide periodic measurements
- Measured speed and position every **5-mins** (for each road segment)
- Every road segment measured ~ 2000 times per week
- **5 Trillion** measurements in **historic data** over 140 Billion Km
- **4 Billion** new measurements per day



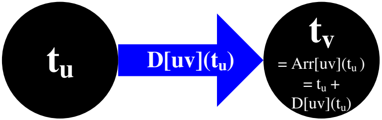
Raw traffic (speed probe) data

- **70 Million** contributing users provide periodic measurements
- Measured speed and position every **5-mins** (for each road segment)
- Every road segment measured $\sim$ 2000 times per week
- **5 Trillion** measurements in **historic data** over 140 Billion Km
- **4 Billion** new measurements per day

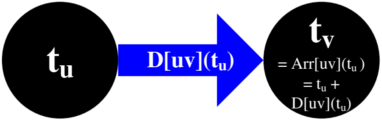


Main Issue: time-dependence

Time-Dependent Shortest Paths

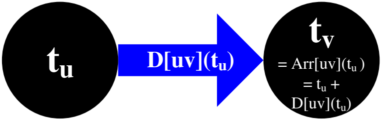


Time-Dependent Shortest Paths



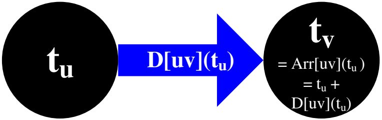
- $P_{o,d}$: *od*-paths; $p = (a_1, \ldots, a_k) \in P_{o,d}$

Time-Dependent Shortest Paths



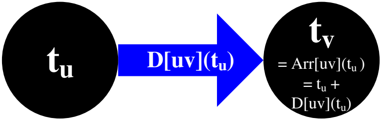
- $P_{o,d}$: *od*-paths; $p = (a_1, \ldots, a_k) \in P_{o,d}$
- **Path arrival / travel-time** functions
  $Arr[p](t_0) = Arr[a_k] \bullet \cdots \bullet Arr[a_1](t_0)$ (function composition)
  $D[p](t_0) = Arr[p](t_0) - t_0$

Time-Dependent Shortest Paths



- $P_{o,d}$: od-paths; $p = (a_1, \ldots, a_k) \in P_{o,d}$
- **Path arrival / travel-time** functions
  $Arr[p](t_0) = Arr[a_k] \bullet \cdots \bullet Arr[a_1](t_0)$ (function composition)
  $D[p](t_0) = Arr[p](t_0) - t_0$
- **Earliest-arrival / Shortest-travel-time** functions
  $Arr[o,d](t_0) = \min_{p \in P_{o,d}} \{ Arr[p](t_0) \}$
  $D[o,d](t_0) = Arr[o,d](t_0) - t_0$

Time-Dependent Shortest Paths



- $P_{o,d}$: od-paths; $p = (a_1, \ldots, a_k) \in P_{o,d}$
- **Path arrival / travel-time** functions
  $Arr[p](t_0) = Arr[a_k] \bullet \cdots \bullet Arr[a_1](t_0)$ (function composition)
  $D[p](t_0) = Arr[p](t_0) - t_0$
- **Earliest-arrival / Shortest-travel-time** functions
  $Arr[o,d](t_0) = \min_{p \in P_{o,d}} \{ Arr[p](t_0) \}$
  $D[o,d](t_0) = Arr[o,d](t_0) - t_0$

Goals

1. For departure-time t_o from o, determine $t_d = Arr[o,d](t_o)$
2. Provide a **succinct representation** of $Arr[o,d]$ (or $D[o,d]$)

FIFO vs non-FIFO Arc Delays

- **FIFO Arc-Delays:** slopes of arc-delay functions ≥ -1
  $\equiv$ non-decreasing arc-arrival functions

FIFO vs non-FIFO Arc Delays

- **FIFO Arc-Delays:** slopes of arc-delay functions ≥ -1
  $\equiv$ non-decreasing arc-arrival functions

- **Non-FIFO Arc-Delays**
  - **Forbidden waiting:** $\nexists$ subpath optimality; NP-hard [Orda-Rom (1990)]
  - **Unrestricted waiting:** $\equiv$ FIFO (arbitrary waiting) [Dreyfus (1969)]

Complexity of TDSP

D: FIFO, piecewise-linear functions; K: total $\#$ of breakpoints

- Given od–pair and departure time t_o from o: **time-dependent** Dijkstra [Dreyfus (1969), Orda-Rom (1990)]

Complexity of TDSP

D: FIFO, piecewise-linear functions; *K*: total # of breakpoints

- Given *od*–pair and departure time t_o from *o*: **time-dependent** Dijkstra [Dreyfus (1969), Orda-Rom (1990)]
- Time-dependent shortest path heuristics: only empirical evidence [Delling & Wagner 2009; Batz etal, 2009]

Complexity of TDSP

D: FIFO, piecewise-linear functions; *K*: total # of breakpoints

- Given *od*−pair and departure time t_o from *o*: **time-dependent** Dijkstra [Dreyfus (1969), Orda-Rom (1990)]
- Time-dependent shortest path heuristics: only empirical evidence [Delling & Wagner 2009; Batz etal, 2009]
- Complexity of computing **succinct representations ?**

Complexity of TDSP

D: FIFO, piecewise-linear functions; *K*: total # of breakpoints

- Given *od*–pair and departure time t_o from *o*: **time-dependent** Dijkstra [Dreyfus (1969), Orda-Rom (1990)]
- Time-dependent shortest path heuristics: only empirical evidence [Delling & Wagner 2009; Batz etal, 2009]
- Complexity of computing **succinct representations ?**
  - Open till recently ...

Complexity of TDSP

D: FIFO, piecewise-linear functions; K: total $\#$ of breakpoints

- Given *od*–pair and departure time t_o from o: **time-dependent** Dijkstra [Dreyfus (1969), Orda-Rom (1990)]
- Time-dependent shortest path heuristics: only empirical evidence [Delling & Wagner 2009; Batz etal, 2009]
- Complexity of computing **succinct representations ?**
  - Open till recently ...
  - $Arr[o, d]$: $O((K+1) \cdot n^{\Theta(\log(n))})$ space [Foschini-Hershberger-Suri (2011)]

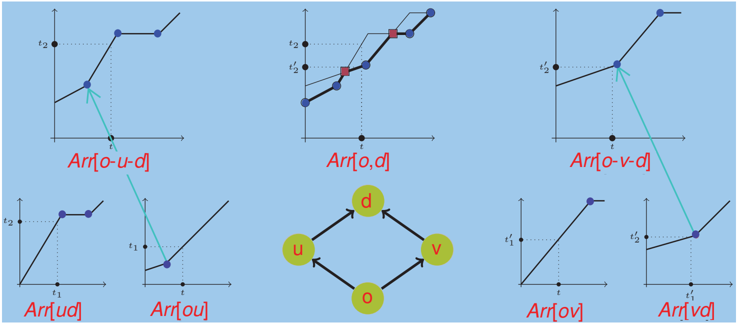
Complexity of TDSP

D: FIFO, piecewise-linear functions; K: total $\#$ of breakpoints

- Given od–pair and departure time t_o from o: **time-dependent** Dijkstra [Dreyfus (1969), Orda-Rom (1990)]
- Time-dependent shortest path heuristics: only empirical evidence [Delling & Wagner 2009; Batz etal, 2009]
- Complexity of computing **succinct representations ?**
  - Open till recently ...
  - $Arr[o,d]$: $O((K+1)\cdot n^{\Theta(\log(n))})$ space [Foschini-Hershberger-Suri (2011)]

Exact Succinct Representation

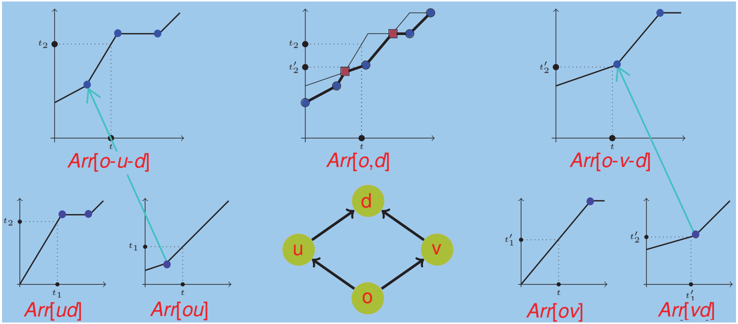
Why so high complexity ?



- **Primitive Breakpoint (PB)**
  Departure-time b_{xy} from x at which $Arr[xy]$ changes slope

Exact Succinct Representation

Why so high complexity ?



- **Primitive Breakpoint (PB)**
  Departure-time b_{xy} from x at which $Arr[xy]$ changes slope
- **Minimization Breakpoint (MB)**
  Departure-time b_x from o s.t. $Arr[o, x]$ changes slope due to **min** operator at x

Complexity of TDSP

D: FIFO, piecewise-linear functions; K: total $\#$ of breakpoints

- Given od–pair and departure time t_o from o: **time-dependent** Dijkstra [Dreyfus (1969), Orda-Rom (1990)]
- Time-dependent shortest path heuristics: only empirical evidence [Delling & Wagner 2009; Batz etal, 2009]
- Complexity of computing **succinct representations ?**
  - Open till recently ...
  - $Arr[o, d]$: $O((K + 1) \cdot n^{\Theta(\log(n))})$ space [Foschini-Hershberger-Suri (2011)]
  - $D[o, d]$: $O(K + 1)$ space for point-to-point $(1 + \varepsilon)$–approximation [Dehne-Omran-Sack (2010), Foschini-Hershberger-Suri (2011)]

Complexity of TDSP

D: FIFO, piecewise-linear functions; *K*: total $\#$ of breakpoints

- **Question 1**: $\exists$ data structure (**oracle**) that
  - requires **reasonable space** ?
  - allows answering **distance queries** efficiently ?

Complexity of TDSP

D: FIFO, piecewise-linear functions; *K*: total # of breakpoints

- **Question 1**: $\exists$ data structure (**oracle**) that
  - requires **reasonable space** ?
  - allows answering **distance queries** efficiently ?
- **Trivial solution I:** Precompute all $(1+\varepsilon)$–approximate distance summaries for every *od*-pair
  - $\mathrm{O}\big(n^2(K+1)\big)$ space
  - $\mathrm{O}(\log\log(K))$ query time
  - $(1+\varepsilon)$–stretch

Complexity of TDSP

D: FIFO, piecewise-linear functions; *K*: total # of breakpoints

- **Question 1**: ∃ data structure (**oracle**) that
  - requires **reasonable space** ?
  - allows answering **distance queries** efficiently ?
- **Trivial solution I:** Precompute all $(1+\varepsilon)$–approximate distance summaries for every *od*-pair
  - $\mathrm{O}\big(n^2(K+1)\big)$ space
  - $\mathrm{O}(\log\log(K))$ query time
  - $(1+\varepsilon)$–stretch
- **Trivial solution II:** No preprocessing, respond to queries with TD-Dijkstra
  - $\mathrm{O}(n+m+K)$ space
  - $\mathrm{O}([m+n\log(n)]\cdot\log\log(K))$ query time
  - 1–stretch

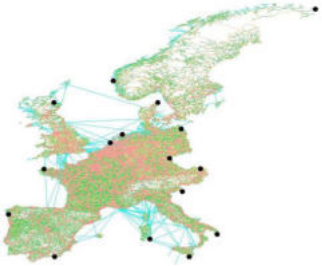
Complexity of TDSP

D: FIFO, piecewise-linear functions; K: total $\#$ of breakpoints

- **Question 1**: $\exists$ data structure (**oracle**) that
  - requires **reasonable space** ?
  - allows answering **distance queries** efficiently ?
- **Trivial solution I:** Precompute all $(1+\varepsilon)-$approximate distance summaries for every *od*-pair
  - $\mathrm{O}\big(n^2(K+1)\big)$ space
  - $\mathrm{O}(\log\log(K))$ query time
  - $(1+\varepsilon)-$stretch
- **Trivial solution II:** No preprocessing, respond to queries with TD-Dijkstra
  - $\mathrm{O}(n+m+K)$ space
  - $\mathrm{O}([m+n\log(n)]\cdot\log\log(K))$ query time
  - $1-$stretch
- **Question 2**: can we do better ?
  - **subquadratic** space & **sublinear** query time
  - $\exists$ smooth tradeoff among space / query time / stretch ?

Towards Time-Dependent Distance Oracles

Generic Framework for Static Landmark-based Oracles



1. Choose a set $L \subset V$ of **landmarks**
2. $\forall \ell \in L$, compute **distance summaries** from ℓ to all $v \in V$
3. Employ a query **algorithm** that uses the pre-computed **distance summaries** to answer **arbitrary** (o, d) distance queries

Towards Time-Dependent Distance Oracles

An Axiomatic Approach – Network Properties

Q Static & undirected world $\longrightarrow$ **time-dependent** & **directed** world ?

Towards Time-Dependent Distance Oracles

An Axiomatic Approach – Network Properties

Q Static & undirected world $\longrightarrow$ **time-dependent** & **directed** world ?

Property 1 (bounded travel time slopes)

Slopes of $D[o, d] \in [-1, \Lambda_{max}]$, for some constant $\Lambda_{max} > 0$

Q Static & undirected world $\longrightarrow$ **time-dependent** & **directed** world ?

Property 1 (bounded travel time slopes)

Slopes of $D[o,d] \in [-1, \Lambda_{\max}]$, for some constant $\Lambda_{\max} > 0$

Property 2 (bounded opposite trips)

$\exists\, \zeta \geq 1 : \forall (o,d) \in V \times V,\ \forall t \in [0,T],\ D[o,d](t) \leq \zeta \cdot D[d,o](t)$

Q Static & undirected world $\longrightarrow$ **time-dependent** & **directed** world ?

Property 1 (bounded travel time slopes)

Slopes of $D[o,d] \in [-1, \Lambda_{\max}]$, for some constant $\Lambda_{\max} > 0$

Property 2 (bounded opposite trips)

$\exists\, \zeta \geq 1 : \forall (o,d) \in V \times V,\ \forall t \in [0,T],\ D[o,d](t) \leq \zeta \cdot D[d,o](t)$

Property 3 (Dij.Rank and TD time are within polynomial factors)

$\exists\, \lambda, c_1, c_2 \in \mathrm{O}(1),\ f(n) \leq \log^{c_1}(n),\ g(n) \leq c_2 \log(n)$:
$\Gamma[o,d](t_o) \leq f(n) \cdot (D[o,d](t_o))^{\lambda}$ and $D[o,d](t_o) \leq g(n) \cdot (\Gamma[o,d](t_o))^{1/\lambda}$

Towards Time-Dependent Distance Oracles

An Axiomatic Approach – Network Properties

Q Static & undirected world $\longrightarrow$ **time-dependent** & **directed** world ?

Property 1 (bounded travel time slopes)

Slopes of $D[o, d] \in [-1, \Lambda_{\max}]$, for some constant $\Lambda_{\max} > 0$

Property 2 (bounded opposite trips)

$\exists\, \zeta \geq 1 : \forall (o, d) \in V \times V,\ \forall t \in [0, T],\ D[o, d](t) \leq \zeta \cdot D[d, o](t)$

Property 3 (Dij.Rank and TD time are within polynomial factors)

$\exists\, \lambda, c_1, c_2 \in \mathrm{O}(1),\ f(n) \leq \log^{c_1}(n),\ g(n) \leq c_2 \log(n)$:
$\Gamma[o, d](t_o) \leq f(n) \cdot (D[o, d](t_o))^{\lambda}$ and $D[o, d](t_o) \leq g(n) \cdot (\Gamma[o, d](t_o))^{1/\lambda}$

Property 4 (no. of arcs linear in no. of vertices)

$m = O(n)$

Validation of Properties

Data Set	Type (source)	n	m	Λ_{max}	ζ_{max}	λ
Berlin	real (TomTom)	480 K	1135 K	0.19	1.19	[1.3,1.6]
Germany	real (PTV)	4690 K	11183 K	0.22	1.05	[1.4,1.7]
WEurope	bench. (PTV)	18010 K	42188 K	3.60	1.13	[1.4,1.7]

First Efficient Time-Dependent Distance Oracle

[Kontogiannis & Zaroliagis, 2014]

1. Choose a set L of **landmarks**

First Efficient Time-Dependent Distance Oracle

[Kontogiannis & Zaroliagis, 2014]

1. Choose a set L of **landmarks**
2. $\forall(\ell, v) \in L \times V$, compute **distance summaries** $\Delta[\ell, v]$, $D[\ell, v] \leq \Delta[\ell, v] \leq (1+\varepsilon) \cdot D[\ell, v]$
    - BIS (bisection-based) approach, one-to-all $(1+\varepsilon)$-approximation

First Efficient Time-Dependent Distance Oracle

[Kontogiannis & Zaroliagis, 2014]

1. Choose a set L of **landmarks**
2. $\forall(\ell, v) \in L \times V$, compute **distance summaries** $\Delta[\ell, v]$, $D[\ell, v] \leq \Delta[\ell, v] \leq (1+\varepsilon) \cdot D[\ell, v]$
   - BIS (bisection-based) approach, one-to-all $(1+\varepsilon)$-approximation
3. Answer **arbitrary** queries (o, d, t_o) using **FCA** & **RQA** query algorithms

First Efficient Time-Dependent Distance Oracle

[Kontogiannis & Zaroliagis, 2014]

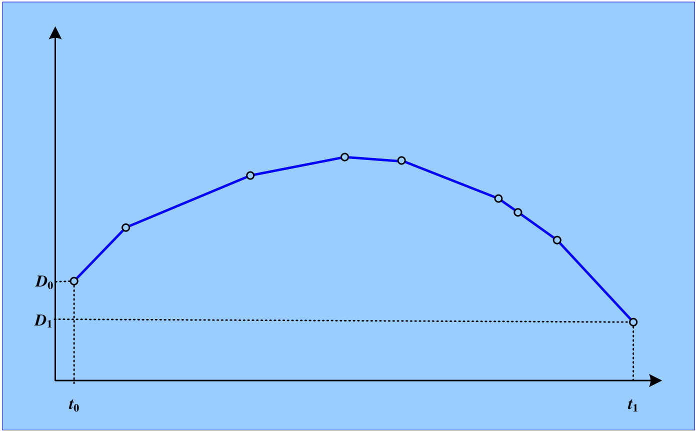
1. Choose a set L of **landmarks**
2. $\forall(\ell, v) \in L \times V$, compute **distance summaries** $\Delta[\ell, v]$, $D[\ell, v] \leq \Delta[\ell, v] \leq (1+\varepsilon) \cdot D[\ell, v]$
   - BIS (bisection-based) approach, one-to-all $(1+\varepsilon)$-approximation
3. Answer **arbitrary** queries (o, d, t_o) using **FCA** & **RQA** query algorithms

	Time	Stretch
Preprocessing	$O(K^* \cdot n^{2-\beta+o(1)})$	
FCA	$O(n^{\delta})$	$1+\varepsilon+\psi$
RQA	$O(n^{\delta+o(1)})$	$1+\varepsilon \cdot \frac{(\varepsilon/\psi)^{r+1}}{(\varepsilon/\psi)^{r+1}-1}$

- K^*: concavity spoiling breakpoints $(0 \leq K^* \leq K)$
- $\beta, \delta \in (0,1)$; $\psi = O(1)$ depends on network characteristics
- $r = O(1)$: recursion depth (budget)

Approximating Distance Functions via Bisection

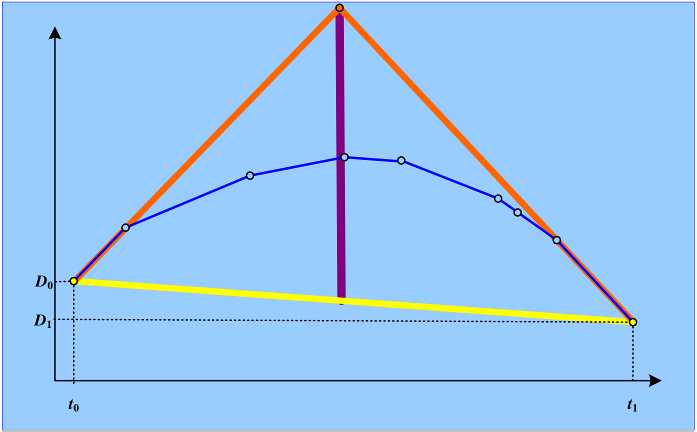
sample simultaneously all distance values from *o*, at mid-points of time intervals, until required approximation guarantee is achieved ∀ destinations



Example of Bisection Execution : INPUT = **UNKNOWN BLUE** function

Approximating Distance Functions via Bisection

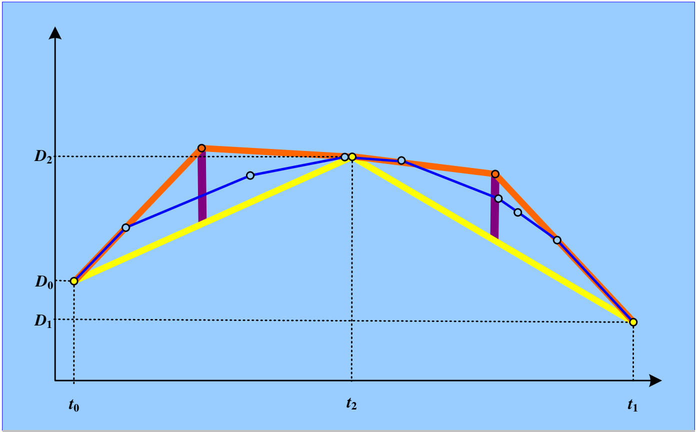
sample simultaneously all distance values from *o*, at mid-points of time intervals, until required approximation guarantee is achieved ∀ destinations



Example of Bisection Execution : **ORANGE** = Upper Bound, **YELLOW** = Lower Bound

Approximating Distance Functions via Bisection

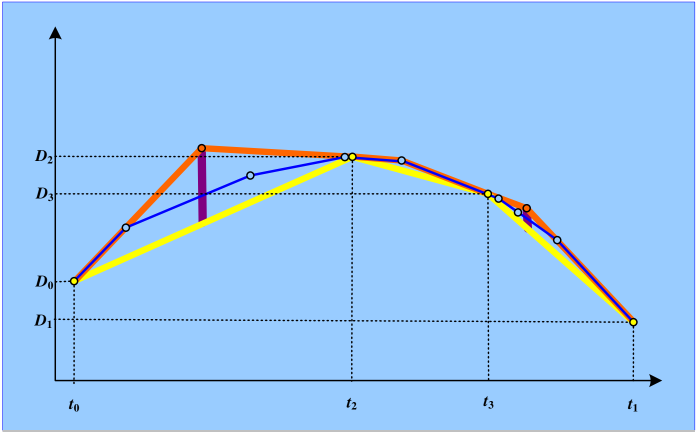
sample simultaneously all distance values from o, at mid-points of time intervals, until required approximation guarantee is achieved $\forall$ destinations



Example of Bisection Execution : Level-1 Recursion

Approximating Distance Functions via Bisection

sample simultaneously all distance values from *o*, at mid-points of time intervals, until required approximation guarantee is achieved ∀ destinations



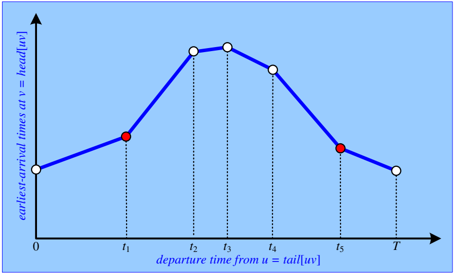
Example of Bisection Execution : Level-2 Recursion

Approximating Distance Functions via Bisection

For continuous, pwl arc-delays

1. Run Reverse TD-Dijkstra to project each concavity-spoiling PB to a PI of the origin *o*

2. For each pair of consecutive PIs at *o*, run BIS for the corresponding departure-times interval



3. Return the concatenation of approximate distance summaries

Landmark Selection and Preprocessing

$K^*(< K)$: total $\#$ of concavity-spoiling breakpoints;

- Landmark selection: $\forall v \in V$, $\Pr[v \in L] = \rho \in (0,1)$, $|L| = \rho \cdot n$
  [correctness is independent of the landmark selection]
- Preprocessing: $\forall \ell \in L$, compute $(1+\varepsilon)$−approximate distance functions $\Delta[\ell, v]$ to all $v \in V$ using **BIS**

Landmark Selection and Preprocessing

$K^*(< K)$: total $\#$ of concavity-spoiling breakpoints;

- Landmark selection: $\forall v \in V$, $\Pr[v \in L] = \rho \in (0,1)$, $|L| = \rho \cdot n$ [correctness is independent of the landmark selection]
- Preprocessing: $\forall \ell \in L$, compute $(1+\varepsilon)-$approximate distance functions $\Delta[\ell, v]$ to all $v \in V$ using **BIS**

Preprocessing complexity ($\rho = n^{-\beta}$)

Landmark Selection and Preprocessing

$K^*(< K)$: total $\#$ of concavity-spoiling breakpoints;

- Landmark selection: $\forall v \in V$, $\Pr[v \in L] = \rho \in (0,1)$, $|L| = \rho \cdot n$
  [correctness is independent of the landmark selection]
- Preprocessing: $\forall \ell \in L$, compute $(1+\varepsilon)$–approximate distance functions $\Delta[\ell, v]$ to all $v \in V$ using **BIS**

Preprocessing complexity ($\rho = n^{-\beta}$)

- Space

$$O\Big((K^* + 1) \cdot |L| \cdot n \cdot \tfrac{1}{\varepsilon} \cdot \log(n/\varepsilon)\Big) = O\Big(K^* \cdot n^{2-\beta+o(1)}\Big)$$

Landmark Selection and Preprocessing

$K^*(< K)$: total $\#$ of concavity-spoiling breakpoints;

- Landmark selection: $\forall v \in V$, $\Pr[v \in L] = \rho \in (0,1)$, $|L| = \rho \cdot n$ [correctness is independent of the landmark selection]
- Preprocessing: $\forall \ell \in L$, compute $(1+\varepsilon)-$approximate distance functions $\Delta[\ell, v]$ to all $v \in V$ using **BIS**

Preprocessing complexity ($\rho = n^{-\beta}$)

- Space

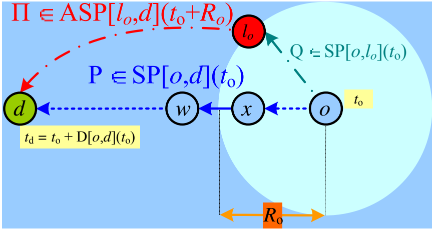
$$O\Big((K^* + 1) \cdot |L| \cdot n \cdot \tfrac{1}{\varepsilon} \cdot \log(n/\varepsilon)\Big) = O\Big(K^* \cdot n^{2-\beta+o(1)}\Big)$$

- Time

$$O\Big(|L| \cdot \tfrac{K^*}{\varepsilon} \log^2(\tfrac{n}{\varepsilon}) \cdot n \log n\Big) = O\Big(K^* \cdot n^{2-\beta+o(1)}\Big)$$

FCA: constant-approximation query algorithm

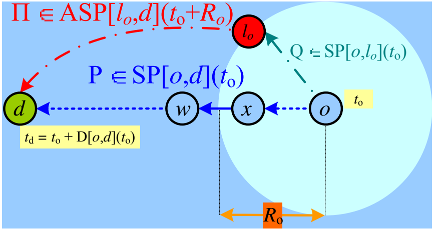
[Kontogiannis & Zaroliagis, 2014]



return $sol_o = D[o, \ell_o](t_o) + \Delta[\ell_o, d](t_o + D[o, \ell_o](t_o))$

FCA: constant-approximation query algorithm

[Kontogiannis & Zaroliagis, 2014]

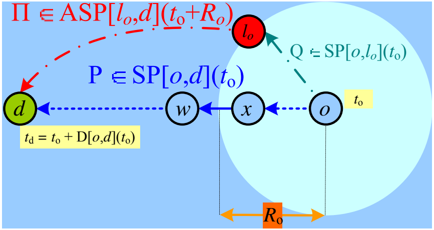


return $sol_o = D[o, \ell_o](t_o) + \Delta[\ell_o, d](t_o + D[o, \ell_o](t_o))$

FCA complexity

FCA: constant-approximation query algorithm

[Kontogiannis & Zaroliagis, 2014]



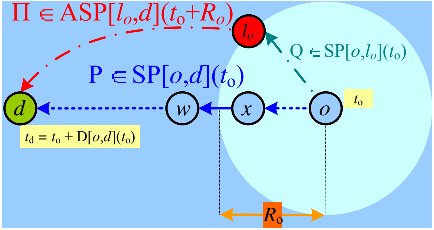
return $sol_o = D[o, \ell_o](t_o) + \Delta[\ell_o, d](t_o + D[o, \ell_o](t_o))$

FCA complexity

- Approximation guarantee: $\leq (1 + \varepsilon + \psi) \cdot D[o, d](t_o)$
  $\psi = 1 + \Lambda_{\max}(1 + \varepsilon)(1 + 2\zeta + \Lambda_{\max}\zeta) + (1 + \varepsilon)\zeta$

FCA: constant-approximation query algorithm

[Kontogiannis & Zaroliagis, 2014]



return $sol_o = D[o,\ell_o](t_o) + \Delta[\ell_o,d](t_o + D[o,\ell_o](t_o))$

FCA complexity

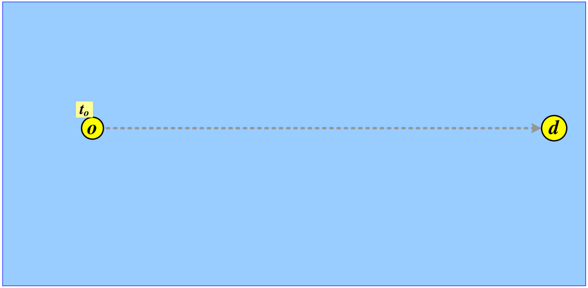
- Approximation guarantee: $\leq (1+\varepsilon+\psi)\cdot D[o,d](t_o)$
  $\psi = 1 + \Lambda_{\max}(1+\varepsilon)(1+2\zeta+\Lambda_{\max}\zeta) + (1+\varepsilon)\zeta$
- Query-time: $O(n^{\delta})$ $(0 < \delta < 1)$

RQA: Boosting the Approximation Guarantee – PTAS

[Kontogiannis & Zaroliagis, 2014]

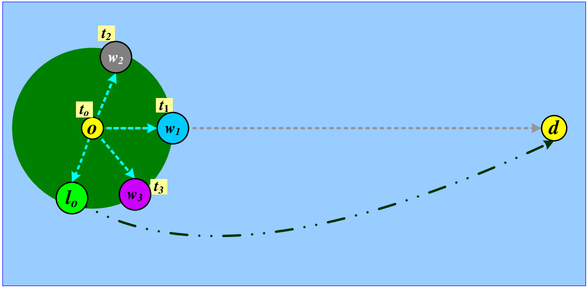
RQA: Boosting the Approximation Guarantee – PTAS

[Kontogiannis & Zaroliagis, 2014]



RQA: Boosting the Approximation Guarantee – PTAS

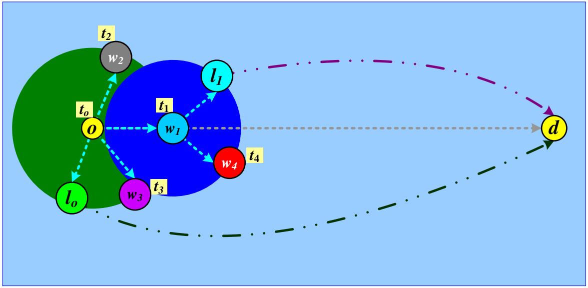
[Kontogiannis & Zaroliagis, 2014]



- Growing level-0 ball...

RQA: Boosting the Approximation Guarantee – PTAS

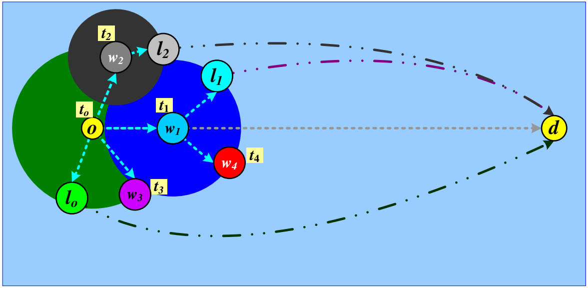
[Kontogiannis & Zaroliagis, 2014]



- Growing level-0 ball...
- Growing level-1 balls...

RQA: Boosting the Approximation Guarantee – PTAS

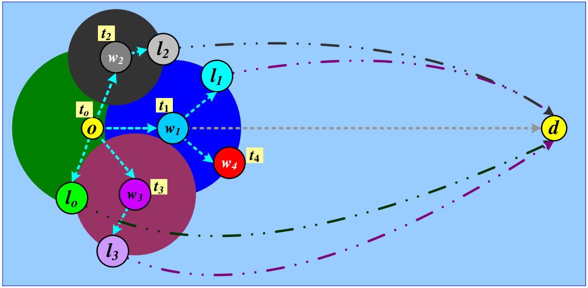
[Kontogiannis & Zaroliagis, 2014]



- Growing level-0 ball...
- Growing level-1 balls...

RQA: Boosting the Approximation Guarantee – PTAS

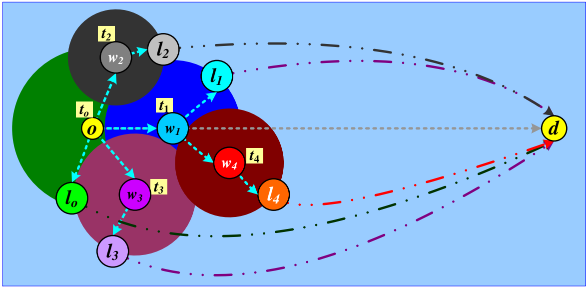
[Kontogiannis & Zaroliagis, 2014]



- Growing level-0 ball...
- Growing level-1 balls...

RQA: Boosting the Approximation Guarantee – PTAS

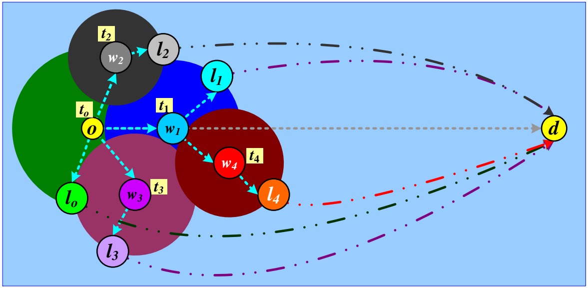
[Kontogiannis & Zaroliagis, 2014]



- Growing level-0 ball...
- Growing level-1 balls...
- Growing level-2 balls...

RQA: Boosting the Approximation Guarantee – PTAS

[Kontogiannis & Zaroliagis, 2014]

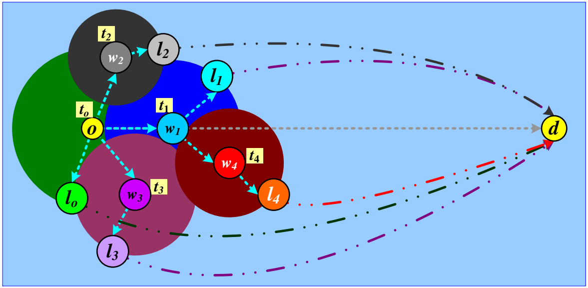


- Growing level-0 ball...
- Growing level-1 balls...
- Growing level-2 balls...
- ... until recursion budget r is exhausted

- return best among
  $sol_i = D[o, w_i](t_o) + D[w_i, \ell_i](t_i) + \Delta[\ell_i, d](t_i + D[w_i, \ell_i](t_i))$

RQA: Boosting the Approximation Guarantee – PTAS

[Kontogiannis & Zaroliagis, 2014]



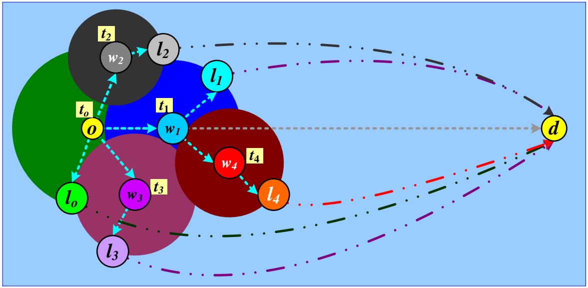
- Growing level-0 ball...
- Growing level-1 balls...
- Growing level-2 balls...
- ... until recursion budget r is exhausted

- return best among
  $sol_i = D[o, w_i](t_o) + D[w_i, \ell_i](t_i) + \Delta[\ell_i, d](t_i + D[w_i, \ell_i](t_i))$

RQA Complexity

RQA: Boosting the Approximation Guarantee – PTAS

[Kontogiannis & Zaroliagis, 2014]



- Growing level-0 ball...
- Growing level-1 balls...
- Growing level-2 balls...
- ... until recursion budget r is exhausted

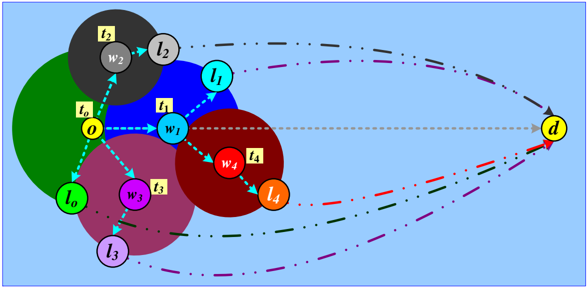
- return best among
  $sol_i = D[o, w_i](t_o) + D[w_i, \ell_i](t_i) + \Delta[\ell_i, d](t_i + D[w_i, \ell_i](t_i))$

RQA Complexity

- Approximation guarantee: $1 + \sigma = 1 + \varepsilon \cdot \frac{(1+\varepsilon/\psi)^{r+1}}{(1+\varepsilon/\psi)^{r+1}-1}$

RQA: Boosting the Approximation Guarantee – PTAS

[Kontogiannis & Zaroliagis, 2014]



- Growing level-0 ball...
- Growing level-1 balls...
- Growing level-2 balls...
- ... until recursion budget r is exhausted

- return best among
  $sol_i = D[o, w_i](t_o) + D[w_i, \ell_i](t_i) + \Delta[\ell_i, d](t_i + D[w_i, \ell_i](t_i))$

RQA Complexity

- Approximation guarantee: $1 + \sigma = 1 + \varepsilon \cdot \frac{(1+\varepsilon/\psi)^{r+1}}{(1+\varepsilon/\psi)^{r+1}-1}$
- Query-time: $O(n^{\delta+o(1)})$; $0 < \delta < 1$

Towards More Efficient Time-Dependent Oracles

- Previous TD oracle efficient only when $K^* \in o(n)$

Towards More Efficient Time-Dependent Oracles

- Previous TD oracle efficient only when $K^* \in o(n)$
- Experimental analysis in Berlin [KMPPWZ, 2015] : $K^* \in \Theta(n)$ (!)

Towards More Efficient Time-Dependent Oracles

- Previous TD oracle efficient only when $K^* \in o(n)$
- Experimental analysis in Berlin [KMPPWZ, 2015] : $K^* \in \Theta(n)$ (!)

$\Downarrow$

Towards More Efficient Time-Dependent Oracles

- Previous TD oracle efficient only when $K^* \in o(n)$
- Experimental analysis in Berlin [KMPPWZ, 2015] : $K^* \in \Theta(n)$ (!)

$\Downarrow$

- Space blow-up

Towards More Efficient Time-Dependent Oracles

- Previous TD oracle efficient only when $K^* \in o(n)$
- Experimental analysis in Berlin [KMPPWZ, 2015] : $K^* \in \Theta(n)$ (!)

$\Downarrow$

- Space blow-up
- Can we avoid dependence on K^* and still maintain

Towards More Efficient Time-Dependent Oracles

- Previous TD oracle efficient only when $K^* \in o(n)$
- Experimental analysis in Berlin [KMPPWZ, 2015] : $K^* \in \Theta(n)$ (!)

$\Downarrow$

- Space blow-up
- Can we avoid dependence on K^* and still maintain
  - **Subquadratic** preprocessing ?

Towards More Efficient Time-Dependent Oracles

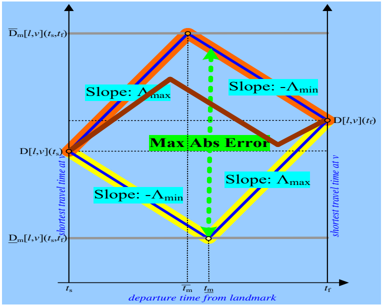
- Previous TD oracle efficient only when $K^* \in o(n)$
- Experimental analysis in Berlin [KMPPWZ, 2015] : $K^* \in \Theta(n)$ (!)

$\Downarrow$

- Space blow-up
- Can we avoid dependence on K^* and still maintain
  - **Subquadratic** preprocessing ?
  - **Sublinear** query time (also on Dijkstra rank) ?

TRAP: New Approximation Method

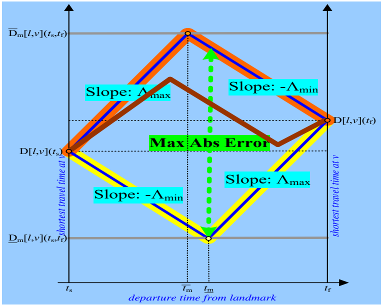
$T \leq n^{\alpha}$ $(0 < \alpha < 1)$: period; [Kontogiannis, Wagner & Zaroliagis, 2016]



- Split $[0, T)$ into $\left\lceil \frac{T}{\tau} \right\rceil$ length-τ subintervals, for a suitable choice of τ
- Compute $(1+\varepsilon)$-upper approximation per subinterval
- $\overline{\Delta}[\ell, v]$ (of $D[o, d] : [0, T) \mapsto \mathbb{R}_{>0}$): concatenation of all upper approximations per subinterval

TRAP: New Approximation Method

$T \leq n^{\alpha}$ $(0 < \alpha < 1)$: period; [Kontogiannis, Wagner & Zaroliagis, 2016]



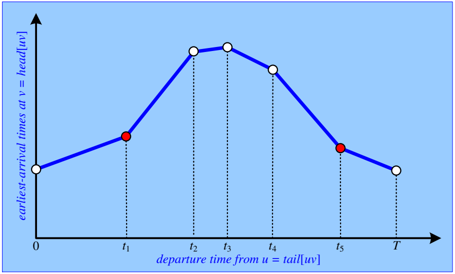
- Split $[0, T)$ into $\left\lceil \frac{T}{\tau} \right\rceil$ length-τ subintervals, for a suitable choice of τ
- Compute $(1 + \varepsilon)$-upper approximation per subinterval
- $\overline{\Delta}[\ell, v]$ (of $D[o, d] : [0, T) \mapsto \mathbb{R}_{>0}$): concatenation of all upper approximations per subinterval

TRAP Complexity

- $O(n^{\alpha})$ TDSP-Calls

BIS vs TRAP Approximation Methods

BIS

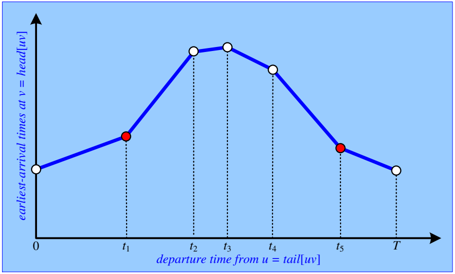


TRAP

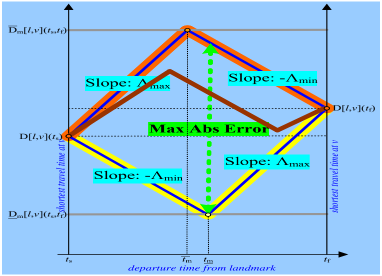
BIS (+)	**BIS (-)**
Simplicity Space-optimal for concave functions First one-to-all approximation	Linear dependence on degree of disconcavity K^*

BIS vs TRAP Approximation Methods

BIS



TRAP



BIS (+)

- Simplicity
- Space-optimal for concave functions
- First one-to-all approximation

BIS (-)

- Linear dependence on degree of disconcavity K^*

TRAP (+)

- Simplicity.
- One-to-all approximation
- Independence from K^*

TRAP (-)

- No guarantee of space-optimality
- Inappropriate for "nearby" vertices around o

TRAPONLY Oracle

[Kontogiannis, Wagner & Zaroliagis, 2016]

Preprocessing

- Compute distance summaries from $\forall \ell \in L$ to all $v \in V$ using **TRAP** (guarantees $(1+\varepsilon)$-approximate distances to "faraway" vertices)

TRAPONLY Oracle

[Kontogiannis, Wagner & Zaroliagis, 2016]

Preprocessing

- Compute distance summaries from $\forall \ell \in L$ to all $v \in V$ using **TRAP** (guarantees $(1+\varepsilon)$-approximate distances to “faraway” vertices)

Query Algorithm

- RQA+

TRAPONLY Oracle

[Kontogiannis, Wagner & Zaroliagis, 2016]

Preprocessing

- Compute distance summaries from $\forall \ell \in L$ to all $v \in V$ using **TRAP** (guarantees $(1+\varepsilon)$-approximate distances to “faraway” vertices)

Query Algorithm

- RQA+
  - Similar to RQA, but in addition ...

TRAPONLY Oracle

[Kontogiannis, Wagner & Zaroliagis, 2016]

Preprocessing

- Compute distance summaries from $\forall \ell \in L$ to all $v \in V$ using **TRAP** (guarantees $(1+\varepsilon)$-approximate distances to "faraway" vertices)

Query Algorithm

- RQA+
  - Similar to RQA, but in addition ...
  - for every $\ell \in L$ discovered by RQA, grow a TD-Dijkstra ball of appropriate size to compute distances to "nearby" vertices

FLAT Oracle

[Kontogiannis, Wagner & Zaroliagis, 2016]

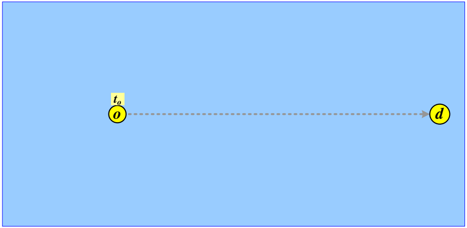
Preprocessing

- compute distance summaries from $\ell \in L$ to all $v \in V$ using **TRAP (BIS)** for "faraway" ("nearby") vertices

Query Algorithms

- Query: FCA, RQA, FCA+(N)

`FCA+(N)` Run FCA until N landmarks are settled. Theory: no better than FCA; practice: **remarkable stretch guarantees**



FLAT Oracle

[Kontogiannis, Wagner & Zaroliagis, 2016]

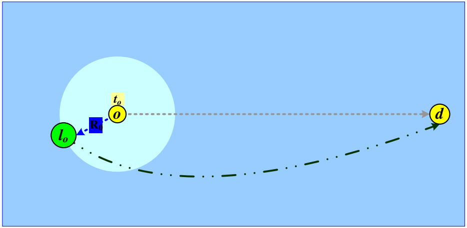
Preprocessing

- compute distance summaries from $\ell \in L$ to all $v \in V$ using **TRAP (BIS)** for "faraway" ("nearby") vertices

Query Algorithms

- Query: FCA, RQA, FCA+(N)

`FCA+(N)` Run FCA until N landmarks are settled. Theory: no better than FCA; practice: **remarkable stretch guarantees**



FLAT Oracle

[Kontogiannis, Wagner & Zaroliagis, 2016]

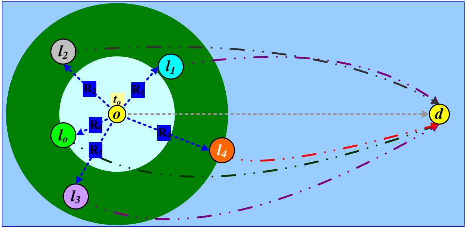
Preprocessing

- compute distance summaries from $\ell \in L$ to all $v \in V$ using **TRAP (BIS)** for "faraway" ("nearby") vertices

Query Algorithms

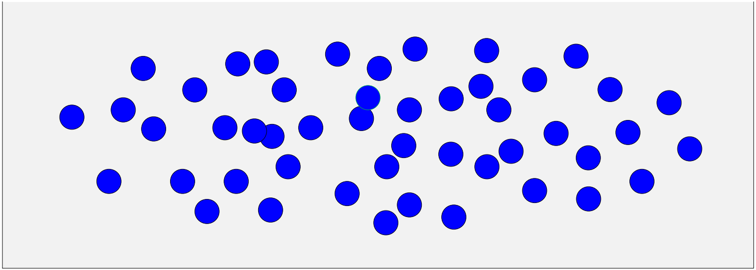
- Query: FCA, RQA, FCA+(N)

FCA+(N) Run FCA until N landmarks are settled. Theory: no better than FCA; practice: **remarkable stretch guarantees**



HORN (**H**ierarchical **OR**acle for TD **N**etworks)

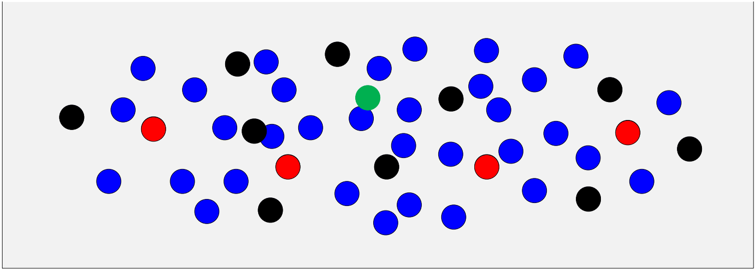
Idea – [Kontogiannis, Wagner & Zaroliagis, 2016]



HORN (**H**ierarchical **OR**acle for TD **N**etworks)

Idea – [Kontogiannis, Wagner & Zaroliagis, 2016]

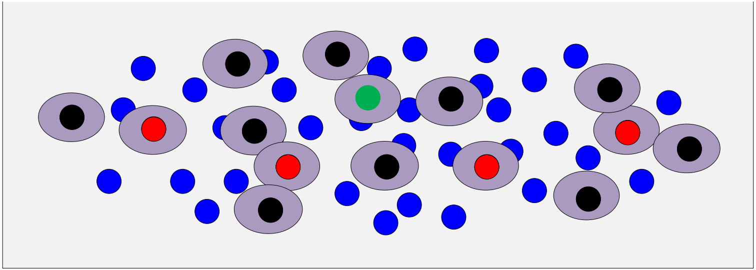
- Selection of landmark sets (colors indicate coverage sizes)



HORN (**H**ierarchical **OR**acle for TD **N**etworks)

Idea – [Kontogiannis, Wagner & Zaroliagis, 2016]

- Selection of landmark sets (colors indicate coverage sizes)
- **Small-coverage** landmarks "learn" travel-time functions to their (only short-range) destinations

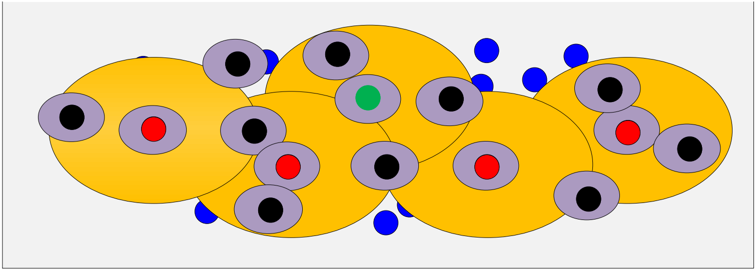


HORN (**H**ierarchical **OR**acle for TD **N**etworks)

Idea – [Kontogiannis, Wagner & Zaroliagis, 2016]

- Selection of landmark sets (colors indicate coverage sizes)
- **Small-coverage** landmarks "learn" travel-time functions to their (only short-range) destinations
- **Medium-coverage** landmarks "learn" travel-time functions to their (up to medium-range) destinations

. . .

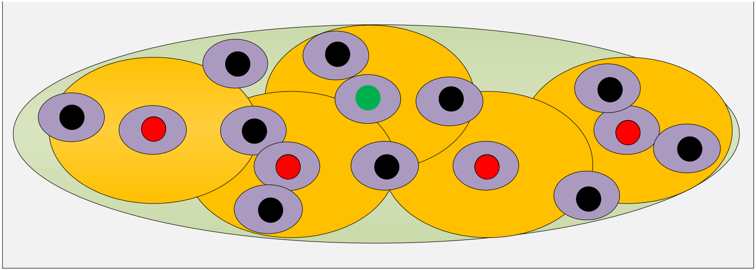


HORN (**H**ierarchical **OR**acle for TD **N**etworks)

Idea – [Kontogiannis, Wagner & Zaroliagis, 2016]

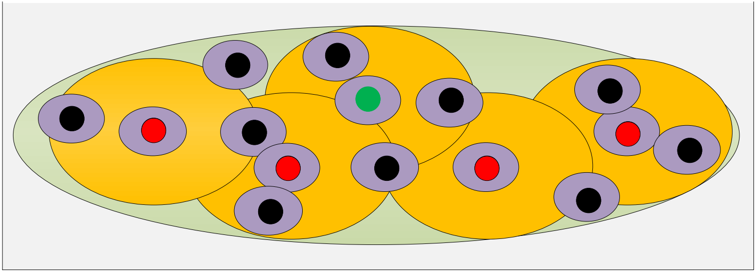
- Selection of landmark sets (colors indicate coverage sizes)
- **Small-coverage** landmarks "learn" travel-time functions to their (only short-range) destinations
- **Medium-coverage** landmarks "learn" travel-time functions to their (up to medium-range) destinations

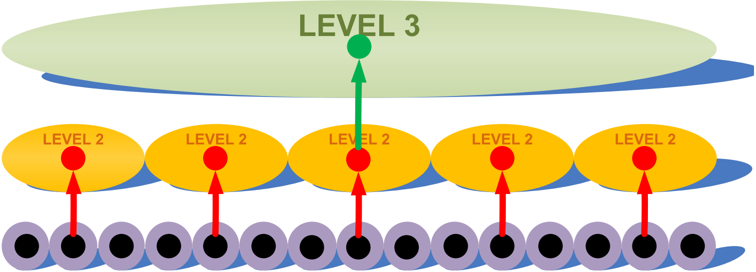
  ...
- **Global-coverage** landmarks "learn" travel-time functions to their (up to long-range) destinations



HORN (**H**ierarchical **OR**acle for TD **N**etworks)

Idea





- Depending on its level, each landmark has its own **coverage**, a given-size set of surrounding vertices for which it is *informed*
- Exponentially decreasing sequence of *landmark set sizes*
- Exponentially increasing sequence of *coverages per landmark*

$\therefore$ $\boxed{O(\log\log(n)) \text{ levels}}$ $\Rightarrow$ **Subquadratic** preprocessing space/time

- Depending on its level, each landmark has its own **coverage**, a given-size set of surrounding vertices for which it is *informed*
- Exponentially decreasing sequence of *landmark set sizes*
- Exponentially increasing sequence of *coverages per landmark*

$\therefore$ $\boxed{O(\log\log(n)) \text{ levels}}$ $\Rightarrow$ **Subquadratic** preprocessing space/time

HORN Preprocessing Complexity

Appropriate construction of the hierarchy ensures **subquadratic** preprocessing space and time $O\left(n^{2-\beta+o(1)}\right)$; $\beta \in (0, 1)$

HORN (**H**ierarchical **OR**acle for TD **N**etworks)

Rationale of the hierarchy

level	targeted DR	Q-time	coverage	TRAP	Ring
1	$N_1 = n^{(\gamma-1)/\gamma}$	N_1^{δ}	$c_1 = N_1 \cdot n^{\xi_1}$	$\sqrt{c_1}$	$N_1^{\delta/(r+1)} \cdot \left(\frac{1}{\ln(n)}, \ln(n)\right]$
2	$N_2 = n^{(\gamma^2-1)/\gamma^2}$	N_2^{δ}	$c_2 = N_2 \cdot n^{\xi_2}$	$\sqrt{c_2}$	$N_2^{\delta/(r+1)} \cdot \left(\frac{1}{\ln(n)}, \ln(n)\right]$
$\vdots$					
k	$N_k = n^{(\gamma^k-1)/\gamma^k}$	N_k^{δ}	$c_k = N_k \cdot n^{\xi_k}$	$\sqrt{c_k}$	$N_k^{\delta/(r+1)} \cdot \left(\frac{1}{\ln(n)}, \ln(n)\right]$
k+1	$N_{k+1} = n$	n^{δ}	$c_{k+1} = n$	$\sqrt{n}$	$\left(N_k^{\delta/(r+1)} \cdot \ln(n), n\right]$

HORN (**H**ierarchical **OR**acle for TD **N**etworks)

Rationale of the hierarchy

level	targeted DR	Q-time	coverage	TRAP	Ring
1	$N_1 = n^{(\gamma-1)/\gamma}$	N_1^δ	$c_1 = N_1 \cdot n^{\xi_1}$	$\sqrt{c_1}$	$N_1^{\delta/(r+1)} \cdot \left(\frac{1}{\ln(n)}, \ln(n)\right]$
2	$N_2 = n^{(\gamma^2-1)/\gamma^2}$	N_2^δ	$c_2 = N_2 \cdot n^{\xi_2}$	$\sqrt{c_2}$	$N_2^{\delta/(r+1)} \cdot \left(\frac{1}{\ln(n)}, \ln(n)\right]$
⋮					
k	$N_k = n^{(\gamma^k-1)/\gamma^k}$	N_k^δ	$c_k = N_k \cdot n^{\xi_k}$	$\sqrt{c_k}$	$N_k^{\delta/(r+1)} \cdot \left(\frac{1}{\ln(n)}, \ln(n)\right]$
k+1	$N_{k+1} = n$	n^δ	$c_{k+1} = n$	$\sqrt{n}$	$\left(N_k^{\delta/(r+1)} \cdot \ln(n), n\right]$

1. Mimic **FLAT** in each level i: all level-i landmarks are informed about c_i destinations around them

HORN (**H**ierarchical **OR**acle for TD **N**etworks)

Rationale of the hierarchy

level	targeted DR	Q-time	coverage	TRAP	Ring
1	$N_1 = n^{(\gamma-1)/\gamma}$	N_1^δ	$c_1 = N_1 \cdot n^{\xi_1}$	$\sqrt{c_1}$	$N_1^{\delta/(r+1)} \cdot \left(\frac{1}{\ln(n)}, \ln(n)\right]$
2	$N_2 = n^{(\gamma^2-1)/\gamma^2}$	N_2^δ	$c_2 = N_2 \cdot n^{\xi_2}$	$\sqrt{c_2}$	$N_2^{\delta/(r+1)} \cdot \left(\frac{1}{\ln(n)}, \ln(n)\right]$
$\vdots$					
k	$N_k = n^{(\gamma^k-1)/\gamma^k}$	N_k^δ	$c_k = N_k \cdot n^{\xi_k}$	$\sqrt{c_k}$	$N_k^{\delta/(r+1)} \cdot \left(\frac{1}{\ln(n)}, \ln(n)\right]$
k+1	$N_{k+1} = n$	n^δ	$c_{k+1} = n$	$\sqrt{n}$	$\left(N_k^{\delta/(r+1)} \cdot \ln(n), n\right]$

1. Mimic **FLAT** in each level i: all level-i landmarks are informed about c_i destinations around them
2. The density of level-i landmarks is such that ALL queries of Dijkstra rank $\leq N_i$ can be answered by using ONLY level-i landmarks

HORN (**H**ierarchical **OR**acle for TD **N**etworks)

Rationale of the hierarchy

level	targeted DR	Q-time	coverage	TRAP	Ring
1	$N_1 = n^{(\gamma-1)/\gamma}$	N_1^δ	$c_1 = N_1 \cdot n^{\xi_1}$	$\sqrt{c_1}$	$N_1^{\delta/(r+1)} \cdot \left(\frac{1}{\ln(n)}, \ln(n)\right]$
2	$N_2 = n^{(\gamma^2-1)/\gamma^2}$	N_2^δ	$c_2 = N_2 \cdot n^{\xi_2}$	$\sqrt{c_2}$	$N_2^{\delta/(r+1)} \cdot \left(\frac{1}{\ln(n)}, \ln(n)\right]$
$\vdots$					
k	$N_k = n^{(\gamma^k-1)/\gamma^k}$	N_k^δ	$c_k = N_k \cdot n^{\xi_k}$	$\sqrt{c_k}$	$N_k^{\delta/(r+1)} \cdot \left(\frac{1}{\ln(n)}, \ln(n)\right]$
k+1	$N_{k+1} = n$	n^δ	$c_{k+1} = n$	$\sqrt{n}$	$\left(N_k^{\delta/(r+1)} \cdot \ln(n), n\right]$

1. Mimic **FLAT** in each level i: all level-i landmarks are informed about c_i destinations around them
2. The density of level-i landmarks is such that ALL queries of Dijkstra rank $\leq N_i$ can be answered by using ONLY level-i landmarks
3. **Fact:** Running **RQA** at the **appropriate level** of the hierarchy would yield a good approximation

HORN (**H**ierarchical **OR**acle for TD **N**etworks)

Rationale of the hierarchy

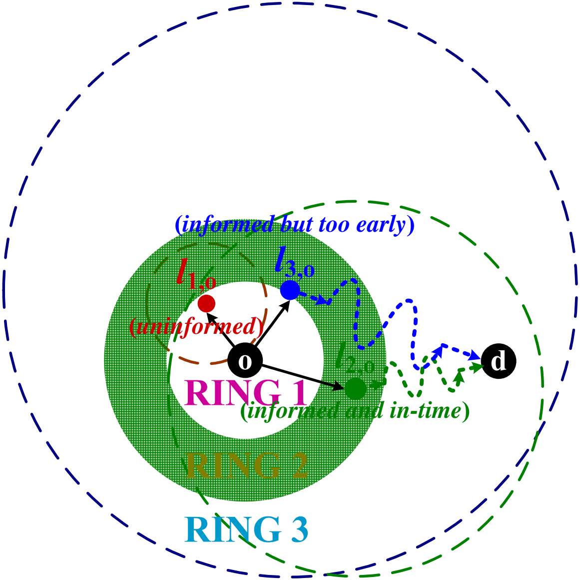
level	targeted DR	Q-time	coverage	TRAP	Ring
1	$N_1 = n^{(\gamma-1)/\gamma}$	N_1^δ	$c_1 = N_1 \cdot n^{\xi_1}$	$\sqrt{c_1}$	$N_1^{\delta/(r+1)} \cdot \left(\frac{1}{\ln(n)}, \ln(n)\right]$
2	$N_2 = n^{(\gamma^2-1)/\gamma^2}$	N_2^δ	$c_2 = N_2 \cdot n^{\xi_2}$	$\sqrt{c_2}$	$N_2^{\delta/(r+1)} \cdot \left(\frac{1}{\ln(n)}, \ln(n)\right]$
⋮					
k	$N_k = n^{(\gamma^k-1)/\gamma^k}$	N_k^δ	$c_k = N_k \cdot n^{\xi_k}$	$\sqrt{c_k}$	$N_k^{\delta/(r+1)} \cdot \left(\frac{1}{\ln(n)}, \ln(n)\right]$
k+1	$N_{k+1} = n$	n^δ	$c_{k+1} = n$	$\sqrt{n}$	$\left(N_k^{\delta/(r+1)} \cdot \ln(n), n\right]$

1. Mimic **FLAT** in each level i: all level-i landmarks are informed about c_i destinations around them
2. The density of level-i landmarks is such that ALL queries of Dijkstra rank $\leq N_i$ can be answered by using ONLY level-i landmarks
3. **Fact:** Running **RQA** at the **appropriate level** of the hierarchy would yield a good approximation
4. **Challenge:** “Guess” the appropriate level; sublinearity on N_i (rather than n) can then be achieved

HORN (**H**ierarchical **OR**acle for TD **N**etworks)

Hierarchical Query Algorithm (HQA)

- level-1 landmark $\ell_{1,o}$ is **uninformed**
- level-3 landmark $\ell_{3,o}$, although informed, came **too early**
- level-2 landmark $\ell_{2,o}$ is **informed** and within the **right distance**

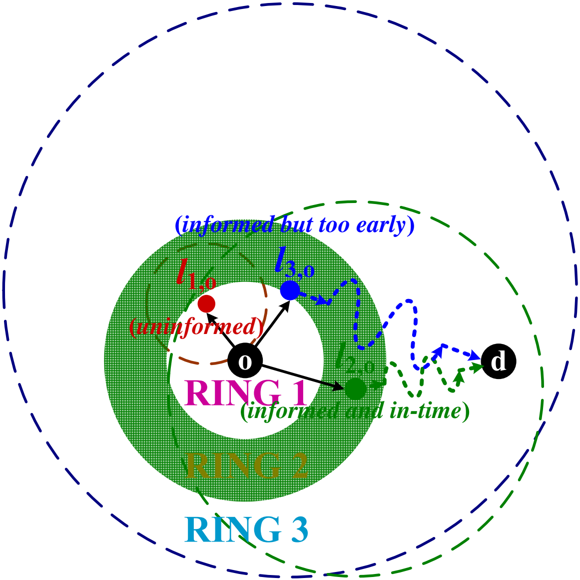


HORN (**H**ierarchical **OR**acle for TD **N**etworks)

Hierarchical Query Algorithm (HQA)

- level-1 landmark $\ell_{1,o}$ is **uninformed**
- level-3 landmark $\ell_{3,o}$, although informed, came **too early**
- level-2 landmark $\ell_{2,o}$ is **informed** and within the **right distance**

∴ **RQA** will use only level-(≥ 2) landmarks from now on



Summary of Time-Dependent Distance Oracles

[Kontogiannis, Wagner & Zaroliagis, 2016]

	preprocessing	query	recursion budget (depth) r
[KZ, 2014]	$K^* \cdot n^{2-\beta+o(1)}$	$n^{\delta+o(1)}$	$r \in O(1)$
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	$n^{2-\beta+o(1)}$	$\approx \Gamma^{\delta+o(1)}$	$r \approx \frac{2\delta}{\alpha} - 1$

- HORN: hierarchical version of FLAT
- Γ: Dijsktra rank
- $T = n^{\alpha}; \alpha, \beta, \delta \in (0, 1)$
- Stretch of all query algorithms: $1 + \varepsilon \cdot \frac{(\varepsilon/\psi)^{r+1}}{(\varepsilon/\psi)^{r+1} - 1}$

Experimental Evaluation

Berlin ($n = 480K$, $m = 1135K$)

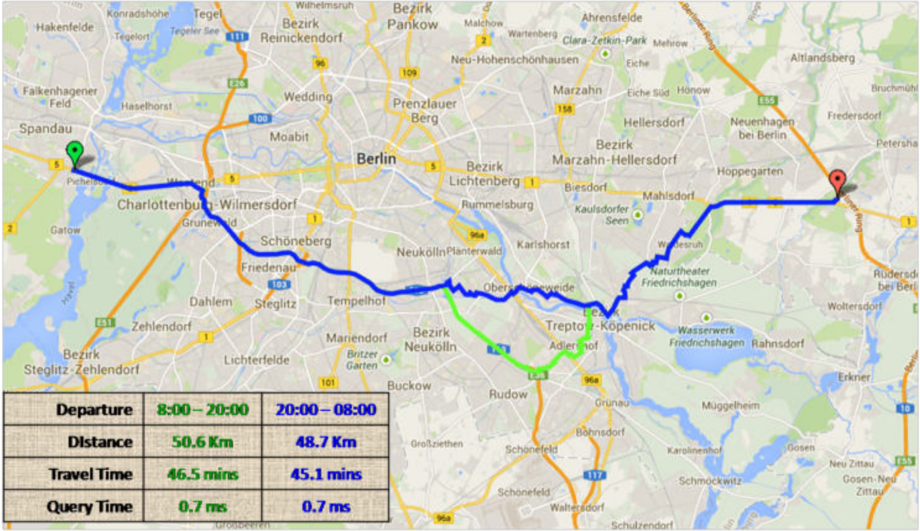
Algorithm	$\|L\|$	Query (ms)	Rel. Error (%)
TDD	–	110.02	0
FLAT	2K	0.081	0.771
CFLAT	4K (1)	0.075	0.521
CFLAT	16K (4)	0.151	0.022

Germany ($n = 4690K$, $m = 11183K$)

Algorithm	$\|L\|$	Query (ms)	Rel. Error (%)
TDD	–	1190.8	0
FLAT	2K	1.269	1.444
CFLAT	4K (1)	0.588	0.791
CFLAT	4K (2)	1.242	0.206

Rel. error 1% $\Rightarrow$ extra delay of 36 sec / 1 hour of optimal travel time

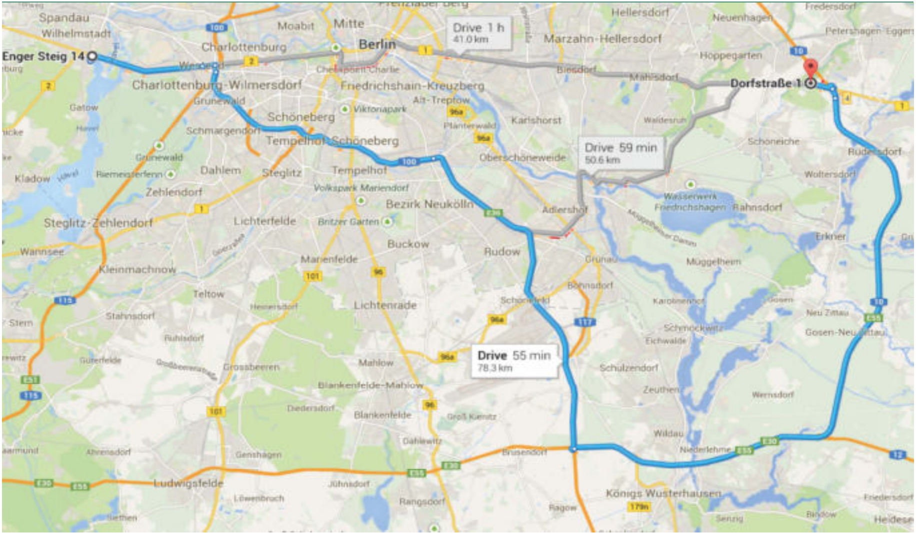
Distance Oracle: Practical Issues



Departure	8:00 – 20:00	20:00 – 08:00
Distance	50.6 Km	48.7 Km
Travel Time	46.5 mins	45.1 mins
Query Time	0.7 ms	0.7 ms

Distance Oracle: Practical Issues

Google Maps, Tuesday 15:45



Conclusions & Future Work

Conclusions & Future Work

Conclusions

- First Time-Dependent Distance Oracles
  - **Subquadratic preprocessing**
  - **Sublinear query** time (also on Dijkstra rank)
  - Provable approximation guarantee
  - **Fully-scalable**; work **well** in practice

Conclusions & Future Work

Conclusions

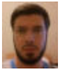
- First Time-Dependent Distance Oracles
  - **Subquadratic preprocessing**
  - **Sublinear query** time (also on Dijkstra rank)
  - Provable approximation guarantee
  - **Fully-scalable**; work **well** in practice

Future Work

- Explore **new landmark sets**
- Improve space through **new compression schemes**
- Exploit **algorithmic parallelism** to further reduce preprocessing time

Publications

CSE Uni Ioan. **CTI & CEID Uni Patras** **KIT**

1. S. Kontogiannis, G. Papastavrou, D. Wagner, C. Zaroliagis: **Improved oracles for time-dependent road networks**. In ATMOS 2017.
2. S. Kontogiannis, D. Wagner, C. Zaroliagis: **Hierarchical Oracles for Time-Dependent Networks**. In ISAAC 2016.
3. S. Kontogiannis, C. Zaroliagis: **Distance Oracles for Time-Dependent Networks**. Algorithmica Vol. 74 (2016), No. 4, pp. 1404-1434. Prel. version in ICALP 2014.
4. K. Giannakopoulou, S. Kontogiannis, G. Papastavrou, and C. Zaroliagis: **A Cloud-based Time-Dependent Routing Service**. In ALGOCLOUD 2016.
5. S. Kontogiannis, G. Michalopoulos, G. Papastavrou, A. Paraskevopoulos, D. Wagner, C. Zaroliagis. **Engineering Oracles for Time-Dependent Road Networks**. In ALENEX 2016.
6. S. Kontogiannis, G. Michalopoulos, G. Papastavrou, A. Paraskevopoulos, D. Wagner, C. Zaroliagis. **Analysis and Experimental Evaluation of Time-Dependent Distance Oracles**. In ALENEX 2015.

Thank you for your attention

Questions