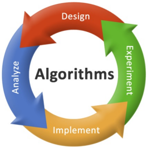
Algorithms for Route Planning

KIT (SS 2016)

Lecture: **Time Dependent Route Planning – II**



Christos Zaroliagis

zaro@ceid.upatras.gr

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

Computer Technology Institute & Press
“Diophantus”

Time Dependent
Shortest Paths – II

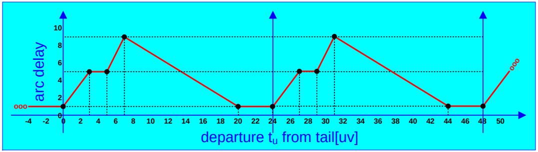
a more realistic and more involved problem

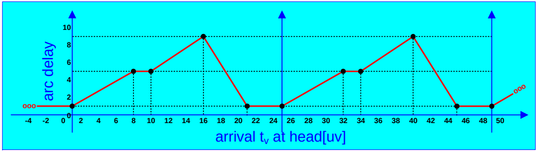
Poly-time Approximation Algorithms

$(1+\varepsilon)$–approximation of $D[o,d]$: Preliminaries

- Why focus on **shortest-travel-time** (delays) functions, and not on **earliest-arrival-time** functions ?

- Arc/Path Delay Reversal: Easy task

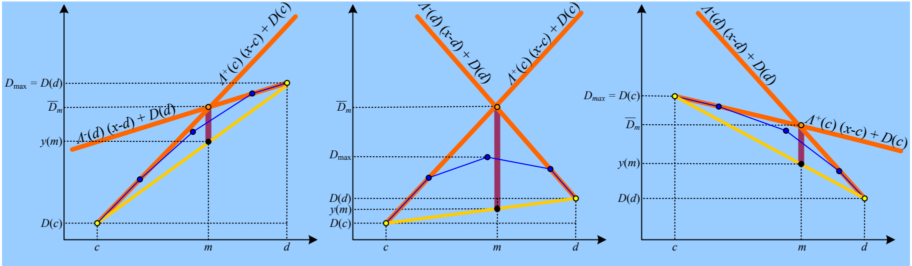




- $t_o = \overleftarrow{Arr}[o,v](t_v) = t_v - \overleftarrow{D}[o,v](t_v)$: **Latest-departure-time** from o to v, as a function of the arrival time t_v at v

Approximating $D[o, d]$: Quality

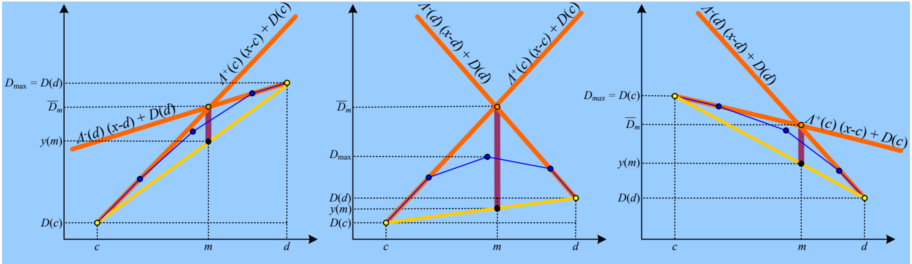
- **Maximum Absolute Error:** A crucial quantity for the time and space complexity of the algorithm



(a) $\Lambda^+(c) > \Lambda^-(d) \geq 0$ (b) $\Lambda^+(c) > 0 > \Lambda^-(d)$ (c) $0 \geq \Lambda^+(c) > \Lambda^-(d)$

Approximating $D[o, d]$: Quality

- **Maximum Absolute Error:** A crucial quantity for the time and space complexity of the algorithm



(a) $\Lambda^+(c) > \Lambda^-(d) \geq 0$ (b) $\Lambda^+(c) > 0 > \Lambda^-(d)$ (c) $0 \geq \Lambda^+(c) > \Lambda^-(d)$

LEMMA: Maximum Absolute Error [Kontogianis-Zaroliagis (2014)]

$$MAE(c, d) = (\Lambda^+(c) - \Lambda^-(d)) \cdot \frac{(m-c)\cdot(d-m)}{d-c} \leq \frac{(d-c)\cdot(\Lambda^+(c)-\Lambda^-(d))}{4}$$

Approximating $D[o, d]$: Basic Idea (I)

- Approximations of $D[o, d]$: For given $\varepsilon > 0$, and $\forall t \in [0, T)$,

$$\boxed{\underline{D}[o, d](t) \leq D[o, d](t) \leq \overline{D}[o, d](t) \leq (1 + \varepsilon) \cdot \underline{D}[o, d](t)}$$

Approximating $D[o, d]$: Basic Idea (I)

- Approximations of $D[o, d]$: For given $\varepsilon > 0$, and $\forall t \in [0, T)$,

$$\boxed{\underline{D}[o, d](t) \leq D[o, d](t) \leq \overline{D}[o, d](t) \leq (1 + \varepsilon) \cdot \underline{D}[o, d](t)}$$

- FACT: **if** $D[o, d]$ was a priori known **then** a **linear scan** gives a space-optimal $(1 + \varepsilon)$–upper-approximation (i.e., with the MIN #BPs)

Approximating $D[o,d]$: Basic Idea (I)

- Approximations of $D[o,d]$: For given $\varepsilon > 0$, and $\forall t \in [0, T)$,

$$\boxed{\underline{D}[o,d](t) \leq D[o,d](t) \leq \overline{D}[o,d](t) \leq (1+\varepsilon) \cdot \underline{D}[o,d](t)}$$

- FACT: **if** $D[o,d]$ was a priori known **then** a **linear scan** gives a space-optimal $(1+\varepsilon)$–upper-approximation (i.e., with the MIN #BPs)
- PROBLEM: Prohibitively expensive to compute/store $D[o,d]$ before approximating it. We must be based only on a few **samples** of $D[o,d]$

Approximating $D[o,d]$: Basic Idea (I)

- Approximations of $D[o,d]$: For given $\varepsilon > 0$, and $\forall t \in [0,T)$,

$$\boxed{\underline{D}[o,d](t) \leq D[o,d](t) \leq \overline{D}[o,d](t) \leq (1+\varepsilon)\cdot\underline{D}[o,d](t)}$$

- FACT: **if** $D[o,d]$ was a priori known **then** a **linear scan** gives a space-optimal $(1+\varepsilon)$–upper-approximation (i.e., with the MIN #BPs)
- PROBLEM: Prohibitively expensive to compute/store $D[o,d]$ before approximating it. We must be based only on a few **samples** of $D[o,d]$
- FOCUS: **Linear** arc-delays. Later extend to pwl arc-delays

Approximating $D[o,d]$: Basic Idea (I)

- Approximations of $D[o,d]$: For given $\varepsilon > 0$, and $\forall t \in [0,T)$,

$$\boxed{\underline{D}[o,d](t) \leq D[o,d](t) \leq \overline{D}[o,d](t) \leq (1+\varepsilon) \cdot \underline{D}[o,d](t)}$$

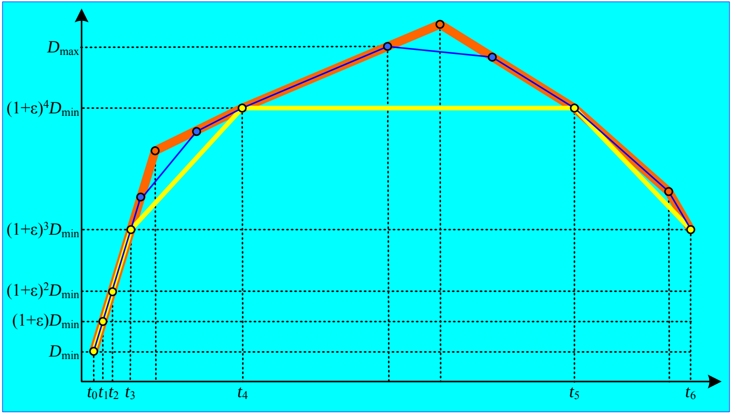
- FACT: **if** $D[o,d]$ was a priori known **then** a **linear scan** gives a space-optimal $(1+\varepsilon)$–upper-approximation (i.e., with the MIN #BPs)

- PROBLEM: Prohibitively expensive to compute/store $D[o,d]$ before approximating it. We must be based only on a few **samples** of $D[o,d]$

- FOCUS: **Linear** arc-delays. Later extend to pwl arc-delays

- $D[o,d]$ lies entirely in a **bounding box** that we can easily determine, with only 3 TD-Djikstra probes

Approximating $D[o, d]$: Basic Idea (II)



- Make the sampling so that $\forall t \in [0, T],\ \overline{D}[o, d](t) \leq (1 + \varepsilon) \cdot \underline{D}[o, d](t)$
- Keep sampling always the **fastest-growing axis** wrt to $D[o, d]$

One-To-One Approximation: PHASE-1

[Foschini-Hershberger-Suri (2011)]

while slope of $D[o,d] \geq 1$ **do**

$t_0 \xrightarrow{\text{Forward Dijkstra}} t_0 + D[o,d](t_0)$

$t_1 \xleftarrow{\text{Backward Dijkstra}} t_0 + (1+\varepsilon)^{1/2} D[o,d](t_0) = t_1 + D[o,d](t_1)$

$t_2 \xleftarrow{\text{Backward Dijkstra}} t_1 + (1+\varepsilon)^{1/2} D[o,d](t_1) = t_2 + D[o,d](t_2)$

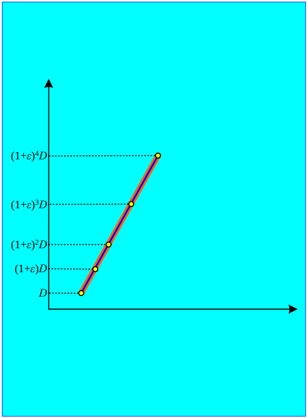
$\vdots$

One-To-One Approximation: PHASE-1

[Foschini-Hershberger-Suri (2011)]

while slope of $D[o, d] \geq 1$ **do**

Bad Case for [Foschini-Hersberger-Suri (2011)] :



One-To-One Approximation: PHASE-1

[Foschini-Hershberger-Suri (2011)]

while slope of $D[o,d] \geq 1$ **do**

[Kontogiannis-Zaroliagis (2014)] :

$t_0 \xrightarrow{\text{Forward Dijkstra}} t_0 + D[o,d](t_0)$

$t_{1,1} \xleftarrow{\text{Backward Dijkstra}} \mathrm{MAX}\{\, t_{fail} \,,\; t_0 + (1+\varepsilon)\, D[o,d](t_0) \,\}$ — *MaxAbsError* $< \varepsilon\, D[o,d](t_0)$

o
o
o

$t_1 = t_{1,k} \xleftarrow{\text{Backward Dijkstra}} t_0 + (1+\varepsilon)^k\, D[o,d](t_0) = t_1 + D[o,d](t_1)$ — *MaxAbsError* $< \varepsilon\, D[o,d](t_0)$

$t_{1,k+1} \xleftarrow{\text{Backward Dijkstra}} t_0 + (1+\varepsilon)^{k+1}\, D[o,d](t_0)$ — *MaxAbsError* $> \varepsilon\, D[o,d](t_0)$

$t_{2,1} \xleftarrow{\text{Backward Dijkstra}} \mathrm{MAX}\{\, t_{fail} \,,\; t_1 + (1+\varepsilon)\, D[o,d](t_1) \,\}$ — *MaxAbsError* $< \varepsilon\, D[o,d](t_1)$

o
o
o

One-To-One Approximation: PHASE-2

[Foschini-Hershberger-Suri (2011)]

Slope of $D[o,d] \leq 1$:

repeat

Apply **BISECTION** to the remaining time-interval(s)

until desired approximation guarantee (wrt **Max Absolute Error**) is achieved

One-To-All Approximaton via Bisection (I)

[Kontogiannis-Zaroliagis (2014)]

ASSUMPTION 1: Concavity of arc-delays. /* to be removed later */

- Implies concavity of the *unknown* function $D[o, d]$

One-To-All Approximaton via Bisection (I)

[Kontogiannis-Zaroliagis (2014)]

ASSUMPTION 1: Concavity of arc-delays. /* to be removed later */

- Implies concavity of the *unknown* function $D[o, d]$

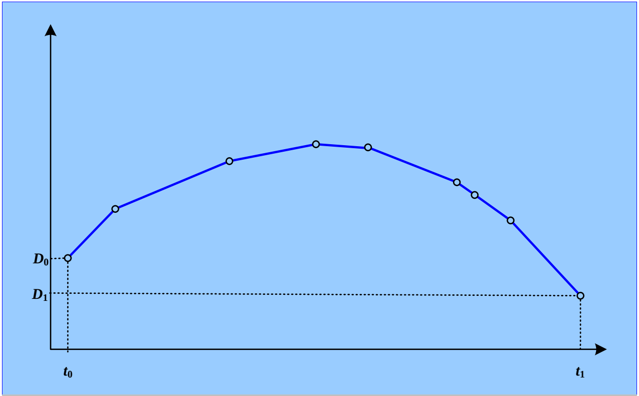
ASSUMPTION 2: Bounded Travel-Time Slopes. Small slopes of the (pwl) arc-delay functions

- **Verified** by TD-traffic data for road network of Berlin [TomTom (February 2013)] that all arc-delay slopes are in $[-0.5, 0.5]$.
- Slopes of *shortest-travel-time* function $D[o, d]$ from $[-\Lambda_{\min}, \Lambda_{\max}]$, for some constants $\Lambda_{\max} > 0$, $\Lambda_{\min} \in [0, 1)$.

One-To-All Approximaton via Bisection (II)

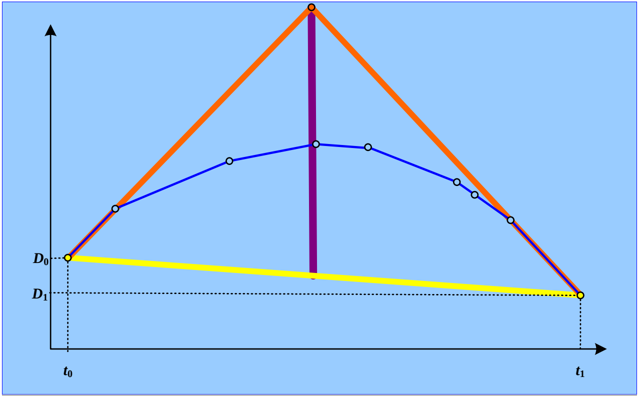
[Kontogiannis-Zaroliagis (2013)]

Under ASSUMPTIONS 1-2: Execute `Bisection` to *sample simultaneously* all distance values from o, at mid-points of time intervals, until required approximation guarantee is achieved *for each destination node*



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

Under ASSUMPTIONS 1-2: Execute `Bisection` to *sample simultaneously* all distance values from o, at mid-points of time intervals, until required approximation guarantee is achieved *for each destination node*

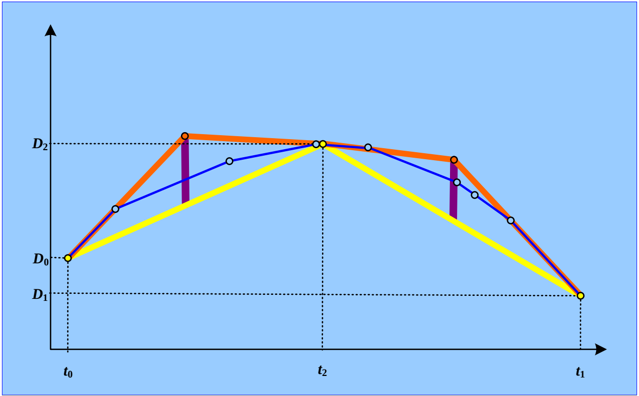


Example of Bisection Execution : **ORANGE** = Upper Bound, **YELLOW** = Lower Bound

One-To-All Approximaton via Bisection (II)

[Kontogiannis-Zaroliagis (2013)]

Under ASSUMPTIONS 1-2: Execute `Bisection` to *sample simultaneously* all distance values from o, at mid-points of time intervals, until required approximation guarantee is achieved *for each destination node*

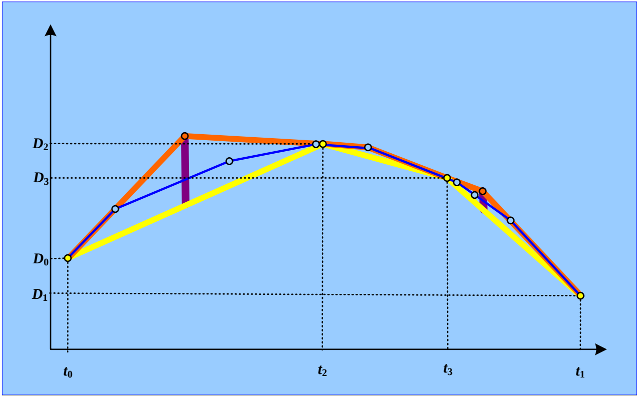


Example of Bisection Execution : Level-1 Recursion

One-To-All Approximaton via Bisection (II)

[Kontogiannis-Zaroliagis (2013)]

Under ASSUMPTIONS 1-2: Execute `Bisection` to *sample simultaneously* all distance values from o, at mid-points of time intervals, until required approximation guarantee is achieved *for each destination node*



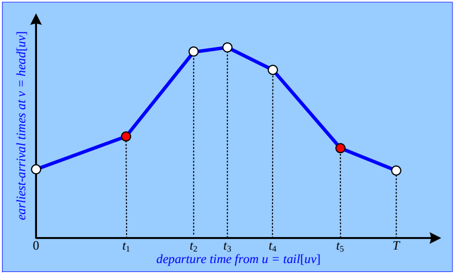
Example of Bisection Execution : Level-2 Recursion

One-To-All Approximaton via Bisection (III)

[Kontogiannis-Zaroliagis (2014)]

Only under ASSUMPTION 2: For continuous, pwl arc-delays

1. Call `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 `Bisection` for the corresponding departure-times interval



3. Return the *concatenation* of approximate distance summaries

Approximating $D[o, \star]$: Space & Time Complexity

THEOREM: Space/Time Complexity [Kontogianis-Zaroliagis (2014)]

K^*: total number of concavity-spoiling BPs among all arc-delay functions
Approximating $\overline{D}[o,\star] = (\overline{D}[o,d])_{d\in V}$ (for *given* $o \in V$ and *all* $d \in V$)

Space Complexity:

1. $\mathrm{O}\left(n\frac{K^*}{\varepsilon}\log\left(\frac{D_{\max}[o,\star](0,T)}{D_{\min}[o,\star](0,T)}\right)\right)$
2. In each interval of *consecutive* PIs,
   $|UBP[o,d]| \leq 4 \cdot$ (minimum #BPs for any $(1+\varepsilon)$–approximation)

Time Complexity: number of *shortest-path probes*

$$\in \mathrm{O}\left(\log\left(\frac{T}{\varepsilon \cdot D_{\min}[o,d]}\right) \cdot \frac{K^*}{\varepsilon}\log\left(\frac{D_{\max}[o,\star](0,T)}{D_{\min}[o,\star](0,T)}\right)\right)$$

Implementation Issues wrt One-To-All Bisection

- `One-To-All Bisection` of [Kontogiannis-Zaroliagis (2014)] is a **label-setting** approximation method that provably works *optimally* wrt **concave** continuous pwl arc-delay functions

Implementation Issues wrt One-To-All Bisection

- `One-To-All Bisection` of [Kontogiannis-Zaroliagis (2014)] is a **label-setting** approximation method that provably works *optimally* wrt **concave** continuous pwl arc-delay functions
- Both `One-To-One Approximation` of [Foschini-Hershberger-Suri (2011)] and `One-To-All Bisection` of [Kontogiannis-Zaroliagis (2014)] suffer from **linear dependence** on K^* (degree of disconcavity)

Implementation Issues wrt One-To-All Bisection

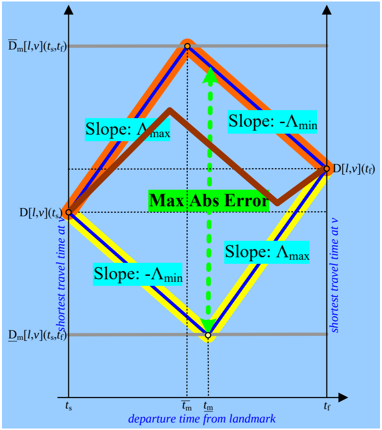
- `One-To-All Bisection` of [Kontogiannis-Zaroliagis (2014)] is a **label-setting** approximation method that provably works *optimally* wrt **concave** continuous pwl arc-delay functions

- Both `One-To-One Approximation` of [Foschini-Hershberger-Suri (2011)] and `One-To-All Bisection` of [Kontogiannis-Zaroliagis (2014)] suffer from **linear dependence** on K^* (degree of disconcavity)

- A novel **one-to-all** (again **label-setting**) approximation technique, called the `Trapezoidal` method ([Kontogiannis-Wagner-Zaroliagis (2016)]), avoids entirely the dependence on K^*

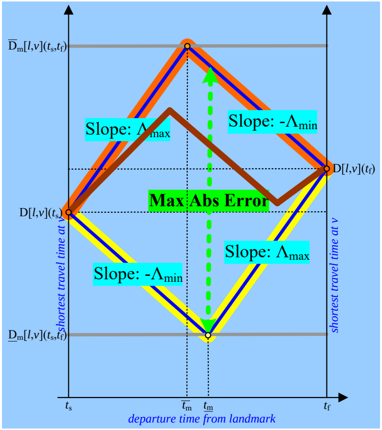
The Trapezoidal One-To-All Approximation Method

- **Sample travel-times** to all destinations, from coarser to finer departure-times from the (common) origin
- Between consecutive samples of the same resolution, the unknown function is *bounded within a given trapezoidal*
- "Freeze" destinations within intervals with satisfactory approximation guarantee



The Trapezoidal One-To-All Approximation Method

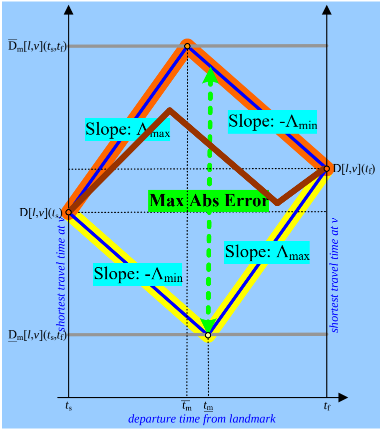
- **Sample travel-times** to all destinations, from coarser to finer departure-times from the (common) origin
- Between consecutive samples of the same resolution, the unknown function is *bounded within a given trapezoidal*
- "Freeze" destinations within intervals with satisfactory approximation guarantee



☺ Avoids dependence on concavity-spoiling BPs of the metric

The Trapezoidal One-To-All Approximation Method

- **Sample travel-times** to all destinations, from coarser to finer departure-times from the (common) origin
- Between consecutive samples of the same resolution, the unknown function is *bounded within a given trapezoidal*
- “Freeze” destinations within intervals with satisfactory approximation guarantee



- Avoids dependence on concavity-spoiling BPs of the metric
- Cannot provide good approximations for “nearby” destinations around the origin

Time-Dependent Oracles

Distance Oracles

- Extremely successful theme in static graphs
  - In theory:
    - ★ P-Space: Subquadratic (sometimes quasi-linear)
    - ★ Q-Time: Constant
    - ★ Stretch: Small (sometimes PTAS)
  - In practice:
    - ★ P-Space: A few GBs (sometimes less than 1 GB)
    - ★ Q-Time: Miliseconds (sometimes microseconds)
    - ★ Stretch: Exact distances (in most cases)

Distance Oracles

- Extremely successful theme in static graphs
  - In theory:
    - P-Space: Subquadratic (sometimes quasi-linear)
    - Q-Time: Constant
    - Stretch: Small (sometimes PTAS)
  - In practice:
    - P-Space: A few GBs (sometimes less than 1 GB)
    - Q-Time: Miliseconds (sometimes microseconds)
    - Stretch: Exact distances (in most cases)
- Some practical algorithms extend to time-dependent case

Distance Oracles

- Extremely successful theme in static graphs
  - In theory:
    - P-Space: Subquadratic (sometimes quasi-linear)
    - Q-Time: Constant
    - Stretch: Small (sometimes PTAS)
  - In practice:
    - P-Space: A few GBs (sometimes less than 1 GB)
    - Q-Time: Miliseconds (sometimes microseconds)
    - Stretch: Exact distances (in most cases)
- Some practical algorithms extend to time-dependent case

FOCUS (rest of talk)

Time-dependent oracles with **provably good** preprocessing-space / query-time / stretch tradeoffs

Distance Oracles

Is it a Success Story in Time-Dependent Graphs?

CHALLENGE: Given a *large scale* TD graph with continuous, pwl, FIFO arc-delay functions, create a data structure (**oracle**) that requires reasonable *(subquadratic)* space and allows answering **distance queries** efficiently (in *sublinear* time)

Distance Oracles

Is it a Success Story in Time-Dependent Graphs?

CHALLENGE: Given a *large scale* TD graph with continuous, pwl, FIFO arc-delay functions, create a data structure (**oracle**) that requires reasonable *(subquadratic)* space and allows answering **distance queries** efficiently (in *sublinear* time)

- Trivial solution 1: Precompute all $(1+\epsilon)$–approximate distance summaries from every origin to every destination
  - ☹ $\mathrm{O}\left(n^3\right)$ size ($\mathrm{O}\left(n^2\right)$, if all arc-delay functions concave)
  - ☺ $\mathrm{O}(\log\log(n))$ query time
  - ☺ $(1+\epsilon)$–stretch

Distance Oracles

Is it a Success Story in Time-Dependent Graphs?

CHALLENGE: Given a *large scale* TD graph with continuous, pwl, FIFO arc-delay functions, create a data structure (**oracle**) that requires reasonable *(subquadratic)* space and allows answering **distance queries** efficiently (in *sublinear* time)

- Trivial solution 1: Precompute all $(1+\epsilon)$–approximate distance summaries from every origin to every destination
  - $\mathrm{O}\left(n^3\right)$ size ($\mathrm{O}\left(n^2\right)$, if all arc-delay functions concave)
  - $\mathrm{O}(\log\log(n))$ query time
  - $(1+\epsilon)$–stretch
- Trivial solution 2: No preprocessing, respond to queries by running `TD-Dijkstra`
  - $\mathrm{O}(n+m+K)$ size (K = total number of PBs of arc-delays)
  - $\mathrm{O}([m+n\log(n)]\cdot\log\log(K))$ query time.
  - 1–stretch

Distance Oracles

Is it a Success Story in Time-Dependent Graphs?

CHALLENGE: Given a *large scale* TD graph with continuous, pwl, FIFO arc-delay functions, create a data structure (**oracle**) that requires reasonable *(subquadratic)* space and allows answering **distance queries** efficiently (in *sublinear* time)

- Trivial solution 1: Precompute all $(1+\epsilon)$–approximate distance summaries from every origin to every destination
  - $\mathrm{O}(n^3)$ size ($\mathrm{O}(n^2)$, if all arc-delay functions concave)
  - $\mathrm{O}(\log\log(n))$ query time
  - $(1+\epsilon)$–stretch
- Trivial solution 2: No preprocessing, respond to queries by running `TD-Dijkstra`
  - $\mathrm{O}(n+m+K)$ size (K = total number of PBs of arc-delays)
  - $\mathrm{O}([m+n\log(n)]\cdot\log\log(K))$ query time.
  - 1–stretch

Is there a smooth tradeoff among space / query time / stretch?

FLAT TD-Oracle

FLAT TD-Oracle: Overall Idea

[Kontogiannis-Zaroliagis (2014)]

1. Choose a set L of **landmarks**
   - In theory: Each vertex $v \in V$ is chosen *independently* w.prob. $\rho \in (0, 1)$
   - In practice: Select landmarks either randomly, or as the set of *boundary vertices* of a given graph partition

FLAT TD-Oracle: Overall Idea

[Kontogiannis-Zaroliagis (2014)]

1. Choose a set L of **landmarks**
   - In theory: Each vertex $v \in V$ is chosen *independently* w.prob. $\rho \in (0, 1)$
   - In practice: Select landmarks either randomly, or as the set of *boundary vertices* of a given graph partition

2. Preprocess $(1 + \epsilon)$-approximate **distance summaries** (functions) $\overline{D}[\ell, v]$ *from* every **landmark** $\ell \in L$ *towards* each destination $v \in V$
   - Label-setting approach
   - One-to-all approximation, for any given landmark $\ell \in L$

FLAT TD-Oracle: Overall Idea

[Kontogiannis-Zaroliagis (2014)]

1. Choose a set L of **landmarks**
   - In theory: Each vertex $v \in V$ is chosen *independently* w.prob. $\rho \in (0, 1)$
   - In practice: Select landmarks either randomly, or as the set of *boundary vertices* of a given graph partition
2. Preprocess $(1 + \epsilon)$-approximate **distance summaries** (functions) $\overline{D}[\ell, v]$ *from* every **landmark** $\ell \in L$ *towards* each destination $v \in V$
   - Label-setting approach
   - One-to-all approximation, for any given landmark $\ell \in L$
3. Provide **query algorithms** (FCA/RQA) that return constant / $(1 + \sigma)$-approximate distance values, for arbitrary query (o, d, t_o)

FLAT TD-Oracle

selection & preprocessing of landmarks

Landmark Selection and Preprocessing (I)

- Select each vertex *independently and uniformly at random* w.prob. $\rho \in (0, 1)$ for the **landmark set** $L \subseteq V$
- `Preprocessing:` $\forall \ell \in L$, precompute $(1+\varepsilon)$–approximate distance functions $\Delta[\ell, v]$ to all destinations $v \in V$

Landmark Selection and Preprocessing (I)

- Select each vertex *independently and uniformly at random* w.prob. $\rho \in (0,1)$ for the **landmark set** $L \subseteq V$
- `Preprocessing:` $\forall \ell \in L$, precompute $(1+\varepsilon)$–approximate distance functions $\Delta[\ell, v]$ to all destinations $v \in V$

THEOREM: [Kontogiannis-Zaroliagis (2014)]

Using `Bisection` for computing approximate distance summaries:

- Pre-Space:

$$\boxed{\mathrm{O}\left(\frac{K^*\cdot|L|\cdot n}{\varepsilon}\cdot \max_{(\ell,v)\in L\times V}\left\{\log\left(\frac{\overline{D}[\ell,v](0,T)}{\underline{D}[\ell,v](0,T)}\right)\right\}\right)}$$

- Pre-Time (in number of TDSP-Probes):

$$\boxed{\mathrm{O}\left(\max_{(\ell,v)}\left\{\log\left(\frac{T\cdot(\Lambda_{\max}+1)}{\varepsilon\underline{D}[\ell,v](0,T)}\right)\right\}\cdot\frac{K^*\cdot|L|}{\varepsilon}\max_{(\ell,v)}\left\{\log\left(\frac{\overline{D}[\ell,v](0,T)}{\underline{D}[\ell,v](0,T)}\right)\right\}\right)}$$

Landmark Selection and Preprocessing (II)

A recent development: Improved preprocessing time/space

Landmark Selection and Preprocessing (II)

A recent development: Improved preprocessing time/space

THEOREM: [Kontogiannis-Wagner-Zaroliagis (2016)]

Using both `Bisection` (for *nearby* nodes) and `Trapezoidal` (for *faraway* nodes):

- Pre-Space:

$$\boxed{\mathbb{E}\left[S_{\mathrm{BIS+TRAP}}\right] \in O\left(T\left(1+\frac{1}{\varepsilon}\right)\Lambda_{\max}\cdot\rho n^2\,\mathrm{polylog}(n)\right)}$$

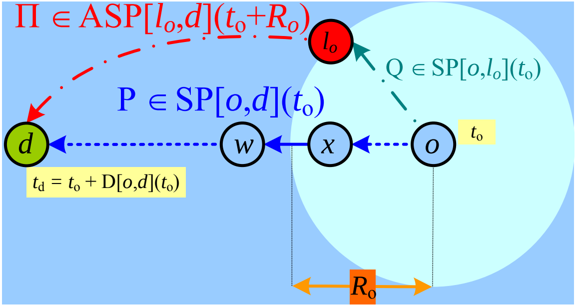
- Pre-Time:

$$\boxed{\mathbb{E}\left[P_{\mathrm{BIS+TRAP}}\right] \in O\left(T\left(1+\frac{1}{\varepsilon}\right)\Lambda_{\max}\cdot\rho n^2\,\mathrm{polylog}(n)\log\log(K_{\max})\right)}$$

FLAT TD-Oracle

FCA: constant-approximation query

FCA: A constant-approximation query algorithm (I)



Forward Constant Approximation: $FCA\left(o, d, t_o, (\Delta[\ell, v])_{(\ell,v)\in L\times V}\right)$

1. Exploration: Grow a TD-Dijkstra forward ball $B(o, t_o)$ until the closest landmark ℓ_o is settled
2. **return** $sol_o = D[o, \ell_o](t_o) + \Delta[\ell_o, d](t_o + D[o, \ell_o](t_o))$

FCA: A constant-approximation query algorithm (II)

- ASSUMPTION 3: Bounded Opposite Trips.
  $\exists \zeta \geq 1 : \forall (o,d) \in V \times V,\ \forall t \in [0,T],\ D[o,d](t) \leq \zeta \cdot D[d,o](t_o)$

FCA: A constant-approximation query algorithm (II)

- ASSUMPTION 3: Bounded Opposite Trips.
  $\exists \zeta \geq 1 : \forall (o,d) \in V \times V,\ \forall t \in [0,T],\ D[o,d](t) \leq \zeta \cdot D[d,o](t_o)$

THEOREM: FCA Performance

For any route planning request (o,d,t_o), `FCA` achieves

- Approximation guarantee:

$$\begin{aligned} D[o,d](t_o) &\leq R_o + \Delta[\ell_o,d](t_o+R_o) \leq (1+\varepsilon)D[o,d](t_o) + \psi R_o \\ &\leq \left(1+\varepsilon+\psi\cdot\frac{R_o}{D[o,d](t_o)}\right)\cdot D[o,d](t_o) \end{aligned}$$

  where $\psi = 1 + \Lambda_{\max}(1+\varepsilon)(1+2\zeta+\Lambda_{\max}\zeta) + (1+\varepsilon)\zeta$

FCA: A constant-approximation query algorithm (II)

- ASSUMPTION 3: Bounded Opposite Trips.
  $\exists \zeta \geq 1 : \forall (o,d) \in V \times V,\ \forall t \in [0,T],\ D[o,d](t) \leq \zeta \cdot D[d,o](t_o)$

THEOREM: FCA Performance

For any route planning request (o, d, t_o), `FCA` achieves

- Approximation guarantee:

$$\begin{aligned} D[o,d](t_o) &\leq R_o + \Delta[\ell_o, d](t_o + R_o) \leq (1+\varepsilon)D[o,d](t_o) + \psi R_o \\ &\leq \left(1 + \varepsilon + \psi \cdot \frac{R_o}{D[o,d](t_o)}\right) \cdot D[o,d](t_o) \end{aligned}$$

  where $\psi = 1 + \Lambda_{\max}(1+\varepsilon)(1 + 2\zeta + \Lambda_{\max}\zeta) + (1+\varepsilon)\zeta$

- Query-time complexity:
  - $\mathbb{E}\left[Q_{FCA}\right] \in \mathrm{O}\left(\frac{1}{\rho} \cdot \ln\left(\frac{1}{\rho}\right)\right)$
  - $\mathbb{P}\left[Q_{FCA} \in \Omega\left(\frac{1}{\rho} \cdot \ln^2\left(\frac{1}{\rho}\right)\right)\right] \in \mathrm{O}(\rho)$

FLAT TD-Oracle

RQA: boosting approximation guarantee

RQA: Overview

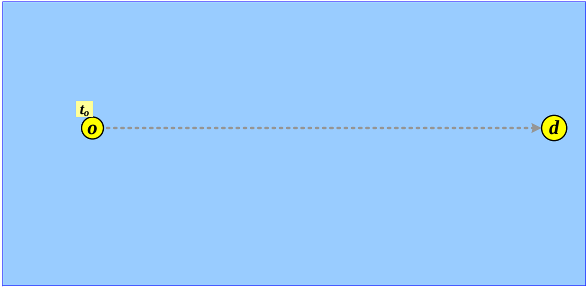
Recursive Query Approximation: $RQA\left(o, d, t_o, (\Delta[\ell, v])_{(\ell,v)\in L\times V}, R\right)$

1. **while** recursion budget R not exhausted **do**
2. Exploration: Grow a TD-Dijkstra forward-ball $B(w_i, t_i)$ until closest landmark ℓ_i is settled
3. $sol_i = D[o, w_i](t_o) + D[w_i, \ell_i](t_i) + \Delta[\ell_i, d](t_i + D[w_i, \ell_i](t_i))$
4. Recursion: Execute RQA centered at *each boundary node* of $B(w_i, t_i)$ with recursion budget $R - 1$
5. **endwhile**
6. **return** best possible solution found

RQA: Overview

Recursive Query Approximation: $RQA\left(o, d, t_o, (\Delta[\ell, v])_{(\ell,v)\in L\times V}, R\right)$

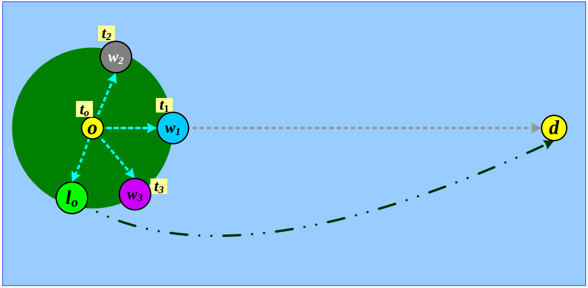
1. **while** recursion budget R not exhausted **do**
2. Exploration: Grow a TD-Dijkstra forward-ball $B(w_i, t_i)$ until closest landmark ℓ_i is settled
3. $sol_i = D[o, w_i](t_o) + D[w_i, \ell_i](t_i) + \Delta[\ell_i, d](t_i + D[w_i, \ell_i](t_i))$
4. Recursion: Execute RQA centered at *each boundary node* of $B(w_i, t_i)$ with recursion budget $R - 1$
5. **endwhile**
6. **return** best possible solution found



RQA: Overview

Recursive Query Approximation: $RQA\left(o, d, t_o, (\Delta[\ell, v])_{(\ell,v)\in L\times V}, R\right)$

1. **while** recursion budget R not exhausted **do**
2. Exploration: Grow a TD-Dijkstra forward-ball $B(w_i, t_i)$ until closest landmark ℓ_i is settled
3. $sol_i = D[o, w_i](t_o) + D[w_i, \ell_i](t_i) + \Delta[\ell_i, d](t_i + D[w_i, \ell_i](t_i))$
4. Recursion: Execute RQA centered at *each boundary node* of $B(w_i, t_i)$ with recursion budget $R - 1$
5. **endwhile**
6. **return** best possible solution found

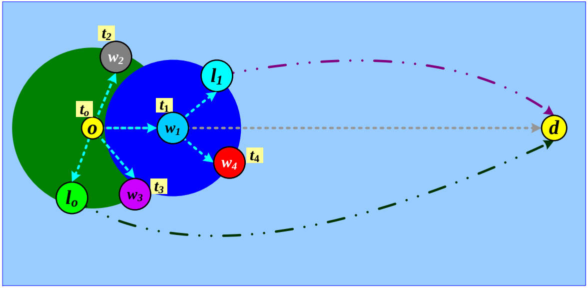


- Growing level-0 ball...

RQA: Overview

Recursive Query Approximation: $RQA\left(o, d, t_o, (\Delta[\ell, v])_{(\ell,v)\in L\times V}, R\right)$

1. **while** recursion budget R not exhausted **do**
2. Exploration: Grow a TD-Dijkstra forward-ball $B(w_i, t_i)$ until closest landmark ℓ_i is settled
3. $sol_i = D[o, w_i](t_o) + D[w_i, \ell_i](t_i) + \Delta[\ell_i, d](t_i + D[w_i, \ell_i](t_i))$
4. Recursion: Execute RQA centered at *each boundary node* of $B(w_i, t_i)$ with recursion budget $R - 1$
5. **endwhile**
6. **return** best possible solution found

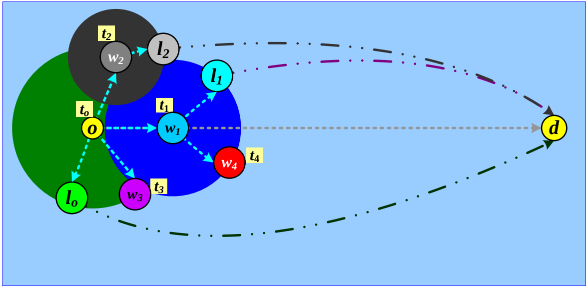


- Growing level-0 ball...
- Growing level-1 balls...

RQA: Overview

Recursive Query Approximation: $RQA\left(o, d, t_o, (\Delta[\ell, v])_{(\ell,v)\in L\times V}, R\right)$

1. **while** recursion budget R not exhausted **do**
2. Exploration: Grow a TD-Dijkstra forward-ball $B(w_i, t_i)$ until closest landmark ℓ_i is settled
3. $sol_i = D[o, w_i](t_o) + D[w_i, \ell_i](t_i) + \Delta[\ell_i, d](t_i + D[w_i, \ell_i](t_i))$
4. Recursion: Execute RQA centered at *each boundary node* of $B(w_i, t_i)$ with recursion budget $R - 1$
5. **endwhile**
6. **return** best possible solution found

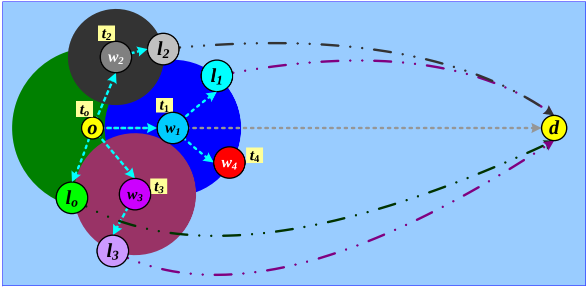


- Growing level-0 ball...
- Growing level-1 balls...

RQA: Overview

Recursive Query Approximation: $RQA\left(o, d, t_o, (\Delta[\ell, v])_{(\ell,v)\in L\times V}, R\right)$

1. **while** recursion budget R not exhausted **do**
2. Exploration: Grow a TD-Dijkstra forward-ball $B(w_i, t_i)$ until closest landmark ℓ_i is settled
3. $sol_i = D[o, w_i](t_o) + D[w_i, \ell_i](t_i) + \Delta[\ell_i, d](t_i + D[w_i, \ell_i](t_i))$
4. Recursion: Execute RQA centered at *each boundary node* of $B(w_i, t_i)$ with recursion budget $R - 1$
5. **endwhile**
6. **return** best possible solution found

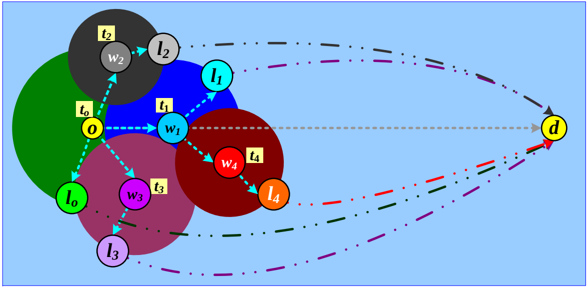


- Growing level-0 ball...
- Growing level-1 balls...

RQA: Overview

Recursive Query Approximation: $RQA\left(o, d, t_o, (\Delta[\ell, v])_{(\ell,v)\in L\times V}, R\right)$

1. **while** recursion budget R not exhausted **do**
2. Exploration: Grow a TD-Dijkstra forward-ball $B(w_i, t_i)$ until closest landmark ℓ_i is settled
3. $sol_i = D[o, w_i](t_o) + D[w_i, \ell_i](t_i) + \Delta[\ell_i, d](t_i + D[w_i, \ell_i](t_i))$
4. Recursion: Execute RQA centered at *each boundary node* of $B(w_i, t_i)$ with recursion budget $R - 1$
5. **endwhile**
6. **return** best possible solution found

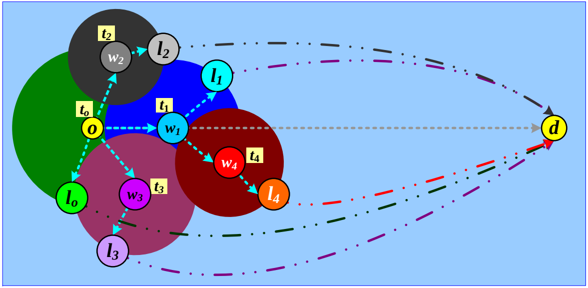


- Growing level-0 ball...
- Growing level-1 balls...
- Growing level-2 balls...

RQA: Overview

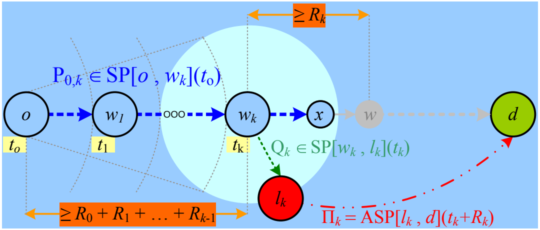
Recursive Query Approximation: $RQA\left(o, d, t_o, (\Delta[\ell, v])_{(\ell,v)\in L\times V}, R\right)$

1. **while** recursion budget R not exhausted **do**
2. Exploration: Grow a TD-Dijkstra forward-ball $B(w_i, t_i)$ until closest landmark ℓ_i is settled
3. $sol_i = D[o, w_i](t_o) + D[w_i, \ell_i](t_i) + \Delta[\ell_i, d](t_i + D[w_i, \ell_i](t_i))$
4. Recursion: Execute RQA centered at *each boundary node* of $B(w_i, t_i)$ with recursion budget $R - 1$
5. **endwhile**
6. **return** best possible solution found

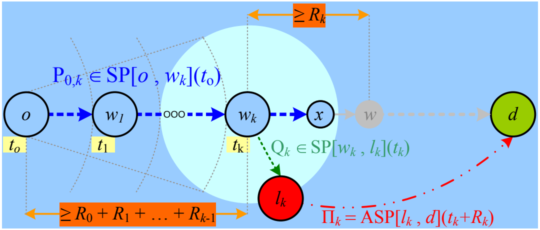


- Growing level-0 ball...
- Growing level-1 balls...
- Growing level-2 balls...
- ...

RQA: Why Does Recursion Boost Approximation?

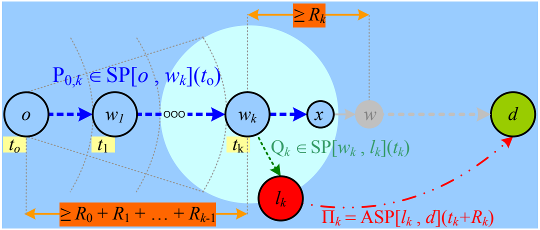


RQA: Why Does Recursion Boost Approximation?



1. One of the discovered approximate *od*–paths has all its ball centers at *nodes of the (unknown) shortest od-path*

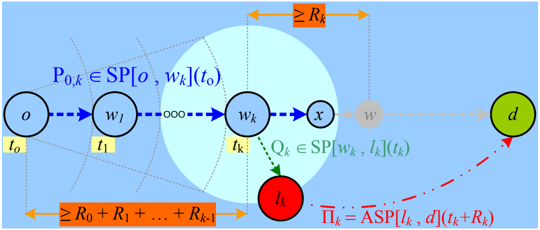
RQA: Why Does Recursion Boost Approximation?



1. One of the discovered approximate *od*–paths has all its ball centers at *nodes of the (unknown) shortest od-path*
2. Optimal **prefix subpaths** improve approximation guarantee:

$$\boxed{\forall\beta > 1,\ \forall\lambda \in (0,1),\ \lambda \cdot OPT + (1-\lambda)\cdot\beta\cdot OPT < \beta\cdot OPT}$$

RQA: Why Does Recursion Boost Approximation?

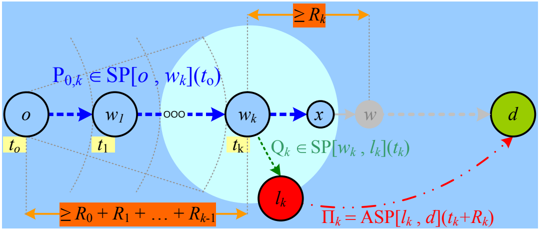


1. One of the discovered approximate *od*–paths has all its ball centers at *nodes of the (unknown) shortest od-path*
2. Optimal **prefix subpaths** improve approximation guarantee:

$$\boxed{\forall \beta > 1,\ \forall \lambda \in (0,1),\ \lambda \cdot OPT + (1-\lambda)\cdot \beta \cdot OPT < \beta \cdot OPT}$$

3. Quality of approximation guarantee of `FCA` (per ball) for remaining **suffix subpath** to the destination depends on *ball radius* (distance from the closest landmark to the ball center)

RQA: Why Does Recursion Boost Approximation?



1. One of the discovered approximate *od*–paths has all its ball centers at *nodes of the (unknown) shortest od-path*
2. Optimal **prefix subpaths** improve approximation guarantee:

$$\boxed{\forall \beta > 1,\ \forall \lambda \in (0,1),\ \lambda \cdot OPT + (1-\lambda) \cdot \beta \cdot OPT < \beta \cdot OPT}$$

3. Quality of approximation guarantee of `FCA` (per ball) for remaining **suffix subpath** to the destination depends on *ball radius* (distance from the closest landmark to the ball center)
4. A constant number of recursion depth R suffices to assure guarantee close to $1 + \varepsilon$

RQA: Performance

THEOREM: Complexity of RQA

The complexity of `RQA` with recursion budget R for obtaining $(1+\sigma)$–approximate distances (for any constant $\sigma > \varepsilon$) to arbitrary (o, d, t_o) queries, is

- $\mathbb{E}\left[Q_{RQA}\right] \in \mathrm{O}\left(\left(\frac{1}{\rho}\right)^{R+1} \cdot \ln\left(\frac{1}{\rho}\right)\right)$
- $\mathbb{P}\left[Q_{RQA} \in \mathrm{O}\left(\left(\frac{\ln(n)}{\rho}\right)^{R+1} \cdot \left[\ln\ln(n) + \ln\left(\frac{1}{\rho}\right)\right]\right)\right] \in 1 - \mathrm{O}\left(\frac{1}{n}\right)$

FCA+: A natural extension of FCA

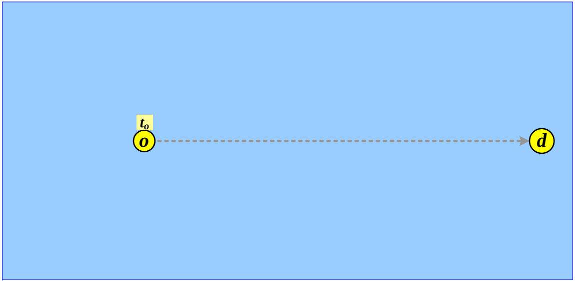
Extended Forward Constant Approximation FCA+(N)

1. Grow TD-Dijkstra ball $B(o, t_o)$ until the N closest landmarks $\ell_o, \ldots, \ell_{N-1}$ (or d) are settled
2. **return** $\min_{i \in \{0,1,\ldots,N-1\}} \{ \, sol_i = D[o, \ell_i](t_o) + \Delta[\ell_i, d](t_i + D[o, \ell_i](t_o)) \, \}$

FCA+: A natural extension of FCA

Extended Forward Constant Approximation FCA+(N)

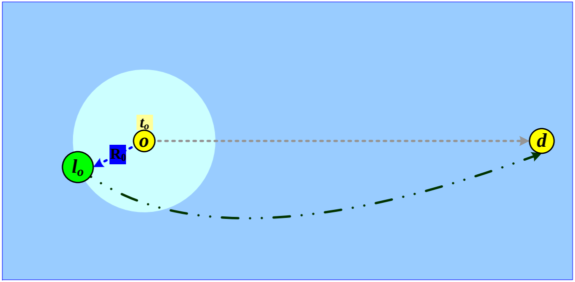
1. Grow TD-Dijkstra ball $B(o, t_o)$ until the N closest landmarks $\ell_o, \ldots, \ell_{N-1}$ (or d) are settled
2. **return** $\min_{i \in \{0,1,\ldots,N-1\}} \{ sol_i = D[o, \ell_i](t_o) + \Delta[\ell_i, d](t_i + D[o, \ell_i](t_o)) \}$



FCA+: A natural extension of FCA

Extended Forward Constant Approximation FCA+(N)

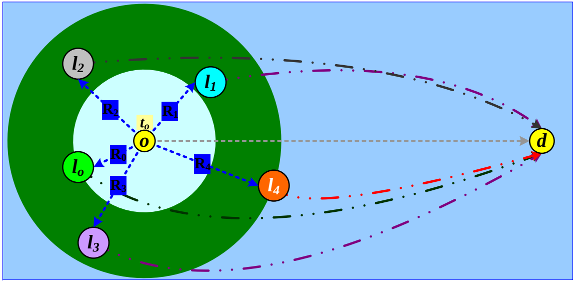
1. Grow TD-Dijkstra ball $B(o, t_o)$ until the N closest landmarks $\ell_o, \ldots, \ell_{N-1}$ (or d) are settled
2. **return** $\min_{i \in \{0,1,\ldots,N-1\}} \{ \, sol_i = D[o, \ell_i](t_o) + \Delta[\ell_i, d](t_i + D[o, \ell_i](t_o)) \, \}$



FCA+: A natural extension of FCA

Extended Forward Constant Approximation FCA+(N)

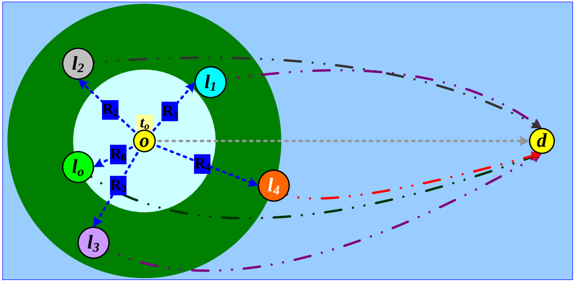
1. Grow TD-Dijkstra ball $B(o, t_o)$ until the N closest landmarks $\ell_o, \ldots, \ell_{N-1}$ (or d) are settled
2. **return** $\min_{i \in \{0,1,\ldots,N-1\}} \{\ sol_i = D[o, \ell_i](t_o) + \Delta[\ell_i, d](t_i + D[o, \ell_i](t_o))\ \}$



FCA+: A natural extension of FCA

Extended Forward Constant Approximation FCA+(N)

1. Grow TD-Dijkstra ball $B(o, t_o)$ until the N closest landmarks $\ell_o, \ldots, \ell_{N-1}$ (or d) are settled
2. **return** $\min_{i \in \{0,1,\ldots,N-1\}} \{ \; sol_i = D[o, \ell_i](t_o) + \Delta[\ell_i, d](t_i + D[o, \ell_i](t_o)) \; \}$



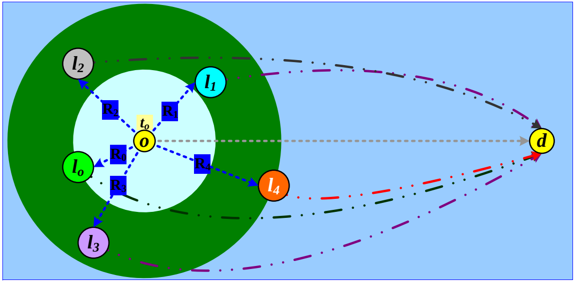
Performance of FCA+(N) for random landmarks

- In theory: Analogous to that of FCA

FCA+: A natural extension of FCA

Extended Forward Constant Approximation FCA+(N)

1. Grow TD-Dijkstra ball $B(o, t_o)$ until the N closest landmarks $\ell_o, \ldots, \ell_{N-1}$ (or d) are settled
2. **return** $\min_{i \in \{0,1,\ldots,N-1\}} \{\ sol_i = D[o, \ell_i](t_o) + \Delta[\ell_i, d](t_i + D[o, \ell_i](t_o))\ \}$



Performance of FCA+(N) for random landmarks

- In theory: Analogous to that of FCA
- In practice: Remarkable performance, analogous to that of RQA

HQA: The Query Algorithm of HORN

Main Goal: Achieve query time **sublinear** in actual Dijkstra Rank (DR)

Constraint: Keep preprocessing space **subquadratic**

HQA: The Query Algorithm of HORN

Main Goal: Achieve query time **sublinear** in actual Dijkstra Rank (DR)

Constraint: Keep preprocessing space **subquadratic**

Hierarchical Query Algorithm (HQA)

1. Grow a unique TD-ball from (o, t_o), until the first **informed landmark** ℓ_o discovered *at the right distance* (not too close, not too far) from o
2. Execute an appropriate variant of RQA, using only landmarks of level at least as high as that of ℓ_o
3. Return the best approximate solution, among all discovered informed landmarks

HQA: The Query Algorithm of HORN

Main Goal: Achieve query time **sublinear** in actual Dijkstra Rank (DR)

Constraint: Keep preprocessing space **subquadratic**

Hierarchical Query Algorithm (HQA)

1. Grow a unique TD-ball from (o, t_o), until the first **informed landmark** ℓ_o discovered *at the right distance* (not too close, not too far) from o
2. Execute an appropriate variant of RQA, using only landmarks of level at least as high as that of ℓ_o
3. Return the best approximate solution, among all discovered informed landmarks

Performance of HQA for random landmarks

With high probability the query-complexity of HQA is $o(N_i)$, where $i \in [k+1]$ is such that $N_{i-1} < DR[o,d](t_o) \leq N_i$

TD Distance Oracles: Recap

what is preprocessed	space : $\mathbb{E}[\mathcal{S}]$	preprocessing : $\mathbb{E}[\mathcal{P}]$	query : $\mathbb{E}[Q_{RQA}]$
All-To-All	$\mathrm{O}\left((K^* + 1)n^2 U\right)$	$\mathrm{O}\left(\begin{array}{c} n^2 \log(n) \\ \cdot \log\log(K_{\max}) \\ \cdot (K^* + 1) TDP \end{array}\right)$	$\mathrm{O}(\log\log(K^*))$
Nothing	$\mathrm{O}(n + m + K)$	$\mathrm{O}(1)$	$\mathrm{O}\left(\begin{array}{c} n \log(n) \cdot \\ \log\log(K_{\max}) \end{array}\right)$
BIS-only Preprocessing			
Landmarks-To-All	$\mathrm{O}\left(\rho n^2 (K^* + 1) U\right)$	$\mathrm{O}\left(\begin{array}{c} \rho n^2 \log(n) \\ \cdot \log\log(K_{\max}) \\ \cdot (K^* + 1) TDP \end{array}\right)$	$\mathrm{O}\left(\begin{array}{c} \left(\frac{1}{\rho}\right)^{R+1} \cdot \log\left(\frac{1}{\rho}\right) \\ \cdot \log\log(K_{\max}) \end{array}\right)$
$K_{\max} \in \mathrm{O}(1)$ $\rho = n^{-a}$ $U, TDP \in \mathrm{O}(1)$ $K^* \in \mathrm{O}(\mathrm{polylog}(()n))$	$\tilde{\mathrm{O}}\left(n^{2-a}\right)$	$\tilde{\mathrm{O}}\left(n^{2-a}\right)$	$\tilde{\mathrm{O}}\left(n^{(R+1)\cdot a}\right)$
BIS+TRAP Preprocessing			
`FLAT`	$\tilde{\mathrm{o}}\left(n^2\right)$	$\tilde{\mathrm{o}}\left(n^2\right)$	$\tilde{\mathrm{o}}(n)$
`HORN`	$\tilde{\mathrm{o}}\left(n^2\right)$	$\tilde{\mathrm{o}}\left(n^2\right)$	$\tilde{\mathrm{o}}(DijkstraRank(0, d))$

Experimental Evaluation

[Kontogiannis et al (ALENEX 2016)]

Identities of Instances

PARAMETER \ INSTANCE	Berlin (TomTom)	Germany (PTV AG)
#Nodes	473,253	4,692,091
#Edges	1,126,468	11,183,060
Time Period	24h (Tue)	24h (Tue-Wed-Thu)
λ_{max}	0.017	0.130
$-\lambda_{\text{min}}$	-0.013	-0.130
#Arcs with constant traversal-times	924,254	10,310,234
#Arcs with non-constant traversal-times	20,2214	872,826
Min #Breakpoints	4	5
Avg #Breakpoints	10.4	16.3
Max #Breakpoints	125	52
Total #Breakpoints	3,234,213	25,424,506

Experimental Evaluation

[Kontogiannis et al (ALENEX 2016)]

Preprocessing and Live-Traffic Updates

- Preprocessing of FLAT @ BERLIN:

	BERLIN		GERMANY	
Parallelism	1 thread	6 threads	1 thread	6 threads
Time per landmark	69.5sec	11.5sec	481sec	80.2sec
Space per landmark	13.8MB		25.7MB	

- Responsiveness to live-traffic reporting: Averaging $1,000$ random disruptions of 15-min duration

	BERLIN		GERMANY	
	#Affected Landmarks	Update Time (sec)	#Affected Landmarks	Update Time (sec)
SR_{2000}	32	21.4	3	37.2
SK_{2000}	36	28.8	4	39.1

Experimental Evaluation

[Kontogiannis et al (ALENEX 2016)]

Query-Time Performance: Speedup $> 1,146$ for Berlin and > 902 for Germany

Berlin: $n = 473,253$ vertices, $m = 1,126,468$ arcs

Germany: $n = 4,692,091$ vertices, $m = 11,183,060$ arcs

- **BERLIN:** 2.64sec resolution and 10,000 *random* queries

	TDD		FCA		FCA$^+$(6)		RQA	
	Time (msec)	Rel.Error %	Time (msec)	Rel.Error %	Time (msec)	Rel.Error %	Time (msec)	Rel.Error %
R_{2000}	92.906	0	0.100	0.969	0.527	0.405	0.519	0.679
K_{2000}			0.115	1.089	0.321	0.405	0.376	0.523
H_{2000}			0.102	0.886	0.523	0.332	0.445	0.602
IR_{2000}			0.086	0.923	0.489	0.379	0.473	0.604
SR_{2000}			0.081	0.771	0.586	0.317	0.443	0.611
SK_{2000}			0.083	0.781	0.616	0.227	0.397	0.464
R_{541}			0.326	1.854	1.887	0.693	1.904	1.610
SR_{541}			0.451	1.638	3.252	0.614	2.856	1.531
R_{270}			0.639	2.583	3.707	0.881	3.842	2.482
SR_{270}			0.730	2.198	4.491	0.745	4.271	2.336

- **GERMANY:** 17.64sec resolution and 10,000 *random* queries

	TDD		FCA		FCA$^+$(6)		RQA	
	Time (msec)	Rel.Error %	Time (msec)	Rel.Error %	Time (msec)	Rel.Error %	Time (msec)	Rel.Error %
R_{2000}	1,145.060	0	1.532	1.567	8.529	0.742	9.219	1.502
K_{2000}			10.455	2.515	15.209	1.708	30.577	2.343
SR_{2000}			1.275	1.444	9.952	0.662	9.011	1.412
SK_{2000}			1.269	1.534	9.689	0.676	7.653	1.475

Experimental Evaluation

[Kontogiannis et al (ALENEX 2016)]

Dijkstra-Rank Performance: Speedup $> 1,570$ for Berlin and $> 1,531$ for Germany

Berlin: $n = 473,253$ vertices, $m = 1,126,468$ arcs

Germany: $n = 4,692,091$ vertices, $m = 11,183,060$ arcs

- **BERLIN:** 2.64sec resolution and 10,000 *random* queries

	TDD		FCA		$FCA^{+}(6)$		RQA	
	Rank	Speedup	Rank	Speedup	Rank	Speedup	Rank	Speedup
R_{2000}	146,022	1	150	973.480	877	166.502	925	157.862
K_{2000}			190	768.537	866	168.616	670	217.943
H_{2000}			154	948.195	851	171.589	777	187.931
IR_{2000}			135	1,081.644	823	177.426	839	174.043
SR_{2000}			119	1,227.075	952	153.384	776	188.173
SK_{2000}			93	1,570.129	755	193.406	501	291.461
R_{541}			545	267.930	3,178	45.947	3,406	42.872
SR_{541}			638	228.874	3,684	39.637	3,950	36.967
R_{270}			1,075	135.834	6,198	23.559	6,702	21.788
SR_{270}			1,195	122.194	7,362	19.835	7,398	19.738

- **GERMANY:** 17.64sec resolution and 10,000 *random* queries

	TDD		FCA		$FCA^{+}(6)$		RQA	
	Rank	Speedup	Rank	Speedup	Rank	Speedup	Rank	Speedup
R_{2000}	1,717,793	1	1,659	1,035.439	10,159	169.091	11,045	155.527
K_{2000}			9,302	184.669	15,373	111.741	30,137	56.999
SR_{2000}			1,277	1,345.178	9,943	172.764	9,182	187.082
SK_{2000}			1,122	1,531.010	9,000	190.866	7,975	215.397

- Landmark hierarchies for HORN, with HR and HSR landmark sets

Level	Size of Levels		Area of coverage	Excluded Ball Size (for *HSR*)	
	$\lvert L\rvert = 10,256$	$\lvert L\rvert = 20,513$		$\lvert L\rvert = 10,256$	$\lvert L\rvert = 20,513$
L_1	7,685	15,370	1,274	35	15
L_2	1,604	3,208	29,243	150	80
L_3	697	1,394	154,847	350	180
L_4	270	541	292,356	800	400

- HQA at 2.64sec resolution and 10,000 *random* queries

	TDD				HQA			
	Time (msec)	Rel.Error %	Rank	Speedup	Time (msec)	Rel.Error %	Rank	Speedup
HR_{10256}	92.906	0	146,022	1	0.354	1.499	636	229.594
HSR_{10256}					0.436	1.409	721	202.527
HR_{20513}					0.217	1.051	324	450.685
HSR_{20513}					0.314	0.919	378	386.302

- HQA vs. FLAT/FCA in Berlin

	Improvement in			Deterioration in
	Query Times (%)	Worst-case Relative Error (%)	Dijkstra Ranks (%)	Space (times)
R_{270} vs HR_{10256}	44.60	41.96	40.83	6.089
SR_{270} vs HSR_{10256}	40.27	35.89	39.66	6.407
R_{541} vs HR_{20513}	33.43	43.31	40.55	6.195
SR_{541} vs HSR_{20513}	30.37	43.89	40.75	6.438

Recap and Open Issues

Recap

- Experimented extensively on landmark-based oracles for TD-nets
- Observed remarkable speedups with reasonable space requirements, both for urban and for national road networks
- Experimented on digesting live-traffic reporting within a few seconds
- Observed **full scalability** in trade-offs between space and query-responses
- Can achieve query-response times 0.73msec, relative error 2.198%, for the Berlin instance, consuming space 3.72GB

Future Work

- Explore new landmark sets that will achieve even better speedups and/or approximation guarantees
- Explore further improvements in the compression schemes to reduce required space
- Exploit algorithmic parallelism to further reduce preprocessing time and responsiveness to live-traffic reports

Related Literature

1. [Dreyfus (1969)] S. E. Dreyfus. **An appraisal of some shortest-path algorithms**. *Operations Research*, 17(3):395–412, 1969.
2. [Orda-Rom (2000)] A. Orda, R. Rom. S**hortest-path and minimum delay algorithms in networks with time-dependent edge-length**. *J. ACM*, 37(3):607–625, 1990.
3. [Dean (2004)] B. C. Dean. **Shortest paths in FIFO time-dependent networks: Theory and algorithms**. Technical report. MIT, 2004.
4. [Dehne-Omran-Sack (2010)] F. Dehne, O. T. Masoud, J. R. Sack. Shortest paths in time-dependent FIFO networks. *ALGORITHMICA*, 62(1-2):416–435, 2012.
5. [Foschini-Hershberger-Suri (2011)] L. Foschini, J. Hershberger, S. Suri. **On the complexity of time-dependent shortest paths**. *ALGORITHMICA*, 68(4), pp. 1075–1097, 2014.
6. [Kontogiannis-Zaroliagis (2014)] S. Kontogiannis, C. Zaroliagis. **Distance oracles for time dependent networks**. *ALGORITHMICA*, 74(4), pp. 1404–1434, 2016.
7. [Kontogiannis et al. (2015)] S. Kontogiannis, G. Michalopoulos, G. Papastavrou, A. Paraskevopoulos, D. Wagner, C. Zaroliagis. **Analysis and Experimental Evaluation of Time-Dependent Distance Oracles**. *Algorithm Engineering and Experiments* (ALENEX 2015), SIAM, 2015.
8. [Kontogiannis et al. (2016)] S. Kontogiannis, G. Michalopoulos, G. Papastavrou, A. Paraskevopoulos, D. Wagner, C. Zaroliagis. **Engineering Oracles for Time-Dependent Road Networks**. *Algorithm Engineering and Experiments* (ALENEX 2016), SIAM, 2016.
9. [Kontogiannis, Wagner, Zaroliagis (2016)] S. Kontogiannis, D. Wagner, C. Zaroliagis. **Hierarchical Oracles for Time-Dependent Road Networks**. arXiv:1502.05222 (2015).