# Exploring earliest-arrival paths in large-scale time-dependent networks via combinatorial oracles

**Workshop on Algorithmic Aspects of Temporal Graphs II**

Patras, July 8 2019


Spyros Kontogiannis

kontog@uoi.gr


Joint work with:


G. Papastavrou
**CSE.UoI**



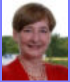
A. Papadopoulos
**CEID.UPatras**



A. Paraskevopoulos
**CEID.UPatras**



C. Zaroliagis
**CEID.UPatras**


D. Wagner
**KIT**

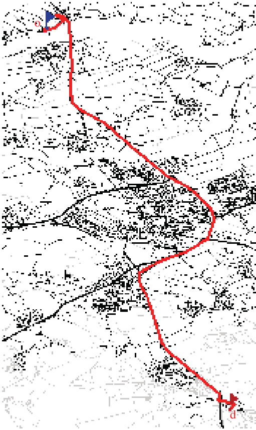
# Time-Dependent Route Planning

## ...problem, assumptions and challenges...

# Shortest Paths

... a fundamental problem, both in theory and in practice...

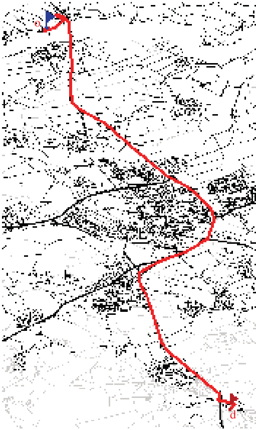
- **Input:**
  - Directed graph $G = (V, E)$.
  - Arc-traversal-time **values**: $D[uv] > 0$.
  - Origin-destination pair: $(o, d) \in V \times V$.



# Shortest Paths

... a fundamental problem, both in theory and in practice...

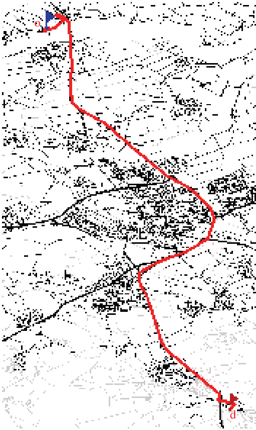
- **Input:**
  - Directed graph $G = (V, E)$.
  - Arc-traversal-time **values**: $D[uv] > 0$.
  - Origin-destination pair: $(o, d) \in V \times V$.
- **Output:** $\pi^* \in \arg\min_{\pi \in P_{o,d}} \{ D[\pi] = \sum_{a \in \pi} D[a] \}$



# Shortest Paths

... a fundamental problem, both in theory and in practice...

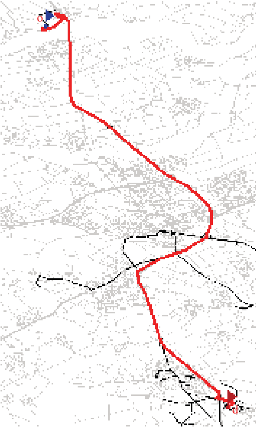
- **Input:**
  - Directed graph $G = (V, E)$.
  - Arc-traversal-time **values**: $D[uv] > 0$.
  - Origin-destination pair: $(o, d) \in V \times V$.
- **Output:** $\pi^* \in \arg\min_{\pi \in P_{o,d}} \{ D[\pi] = \sum_{a \in \pi} D[a] \}$
- MOTIVATION & CHALLENGES: Routing in **road networks**.
  - $V$ = set of intersections, $E$ = set of road segments.
  - *Non-planar*, *sparse* ($|E| \in O(|V|)$) graphs.
  - *Very large* size: $|V|$ = millions of intersections.



# Shortest Paths

... a fundamental problem, both in theory and in practice...

- **Input:**
  - Directed graph $G = (V, E)$.
  - Arc-traversal-time **values**: $D[uv] > 0$.
  - Origin-destination pair: $(o, d) \in V \times V$.
- **Output:** $\pi^* \in \arg\min_{\pi \in P_{o,d}} \{ D[\pi] = \sum_{a \in \pi} D[a] \}$
- MOTIVATION & CHALLENGES: Routing in **road networks**.
  - $V$ = set of intersections, $E$ = set of road segments.
  - *Non-planar*, *sparse* ($|E| \in O(|V|)$) graphs.
  - *Very large* size: $|V|$ = millions of intersections.



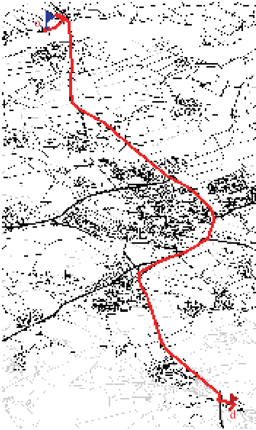
...possibly the most characteristic *success story* of algorithm engineering...

Numerous **oracles** and **speedup techniques** for static road networks.

# Time-Dependent Shortest Paths

...a more challenging problem, both **in theory** and **in practice**...

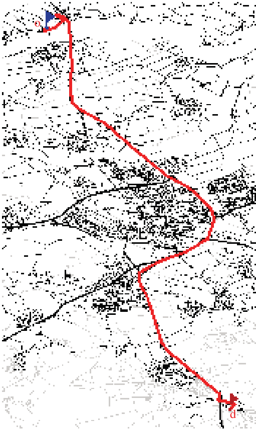
- Input:
  - Directed graph $G = (V, E)$.
  - Arc-traversal-time **functions**: $D[uv] : [0, T) \mapsto \mathbb{R}_{>0}$.
    **Assumption:** Periodic, continuous, piecewise-linear, FIFO-compliant functions...
  - Origin-destination-dep. time triple:
    $(o, d, t_o) \in V \times V \times [0, T)$.



# Time-Dependent Shortest Paths

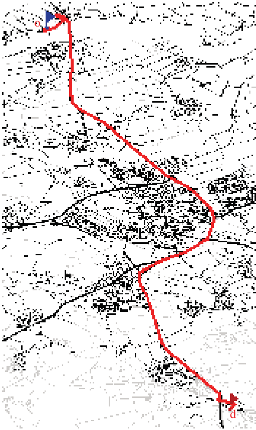
...a more challenging problem, both **in theory** and **in practice**...

- **Input:**
  - Directed graph $G = (V, E)$.
  - Arc-traversal-time **functions**: $D[uv] : [0, T) \mapsto \mathbb{R}_{>0}$.
    **Assumption:** Periodic, continuous, piecewise-linear, FIFO-compliant functions...
  - Origin-destination-dep. time triple:
    $(o, d, t_o) \in V \times V \times [0, T)$.
- **Output:**
  $\pi^* \in \arg\min_{\pi \in P_{o,d}} \{ D[\pi](t_o) = \sum_{a \in \pi} D[a](t_o) \}$



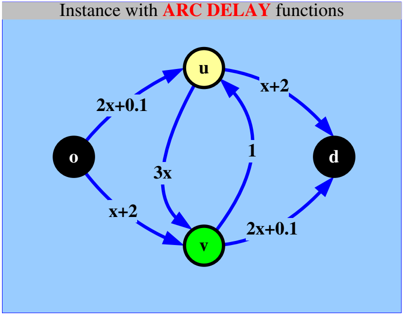
# Time-Dependent Shortest Paths

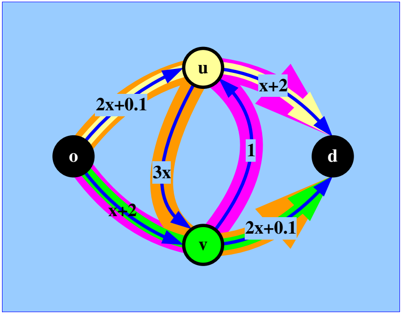
...a more challenging problem, both **in theory** and **in practice**...




- **Input:**
  - Directed graph $G = (V, E)$.
  - Arc-traversal-time **functions**: $D[uv] : [0, T) \mapsto \mathbb{R}_{>0}$.
    **Assumption:** Periodic, continuous, piecewise-linear, FIFO-compliant functions...
  - Origin-destination-dep. time triple:
    $(o, d, t_o) \in V \times V \times [0, T)$.
- **Output:**
  $\pi^* \in \arg\min_{\pi \in P_{o,d}} \{ D[\pi](t_o) = \sum_{a \in \pi} D[a](t_o) \}$
- MOTIVATION & CHALLENGES: Routing in **road networks**.
  - $V$ = set of intersections, $E$ = set of road segments.
  - *Non-planar*, *sparse* ($|E| \in O(|V|)$) graphs.
  - *Very large* size: $|V|$ = millions of intersections.
  - **Time-Dependence**: Computationally harder instances.

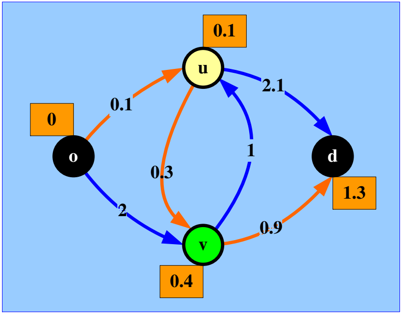
# Time-Dependent Shortest Path: Examples

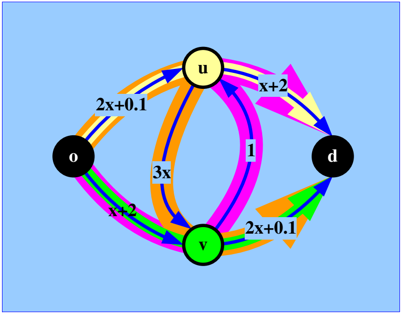



Q1 How would you commute **as fast as possible** from $o$ to $d$, for a given departure time $t_o$ from $o$? E.g.:

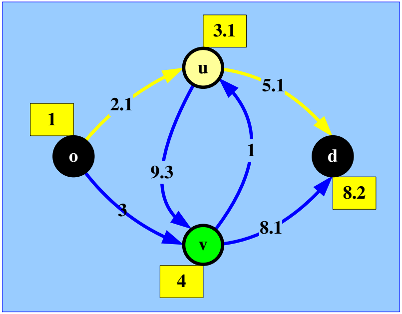
# Time-Dependent Shortest Path: Examples

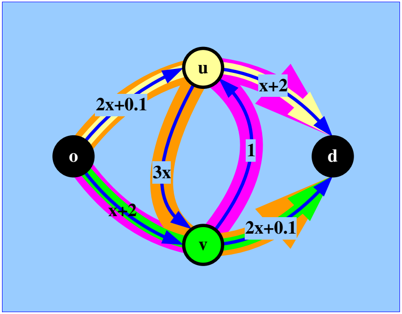



Q1 How would you commute **as fast as possible** from $o$ to $d$, for a given departure time $t_o$ from $o$? E.g.: $t_o = 0$

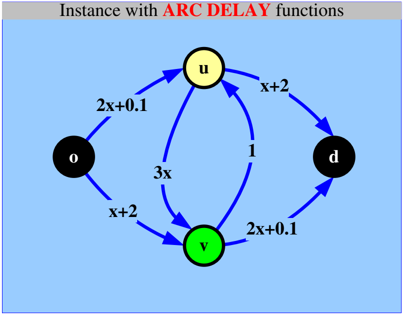
# Time-Dependent Shortest Path: Examples

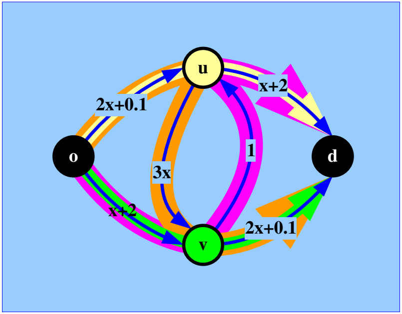



Q1 How would you commute **as fast as possible** from $o$ to $d$, for a given departure time $t_o$ from $o$? E.g.: $t_o = 1$

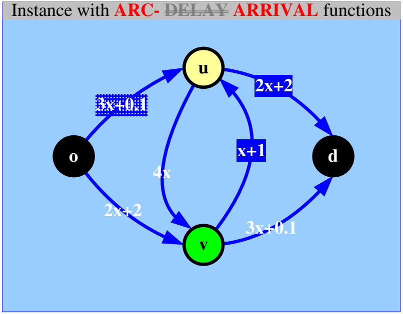
# Time-Dependent Shortest Path: Examples

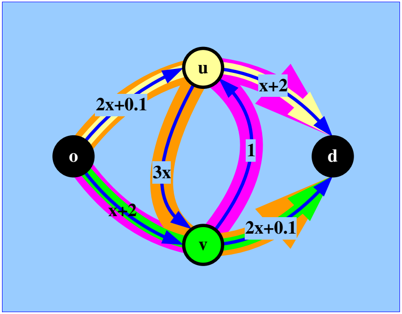



Q1 How would you commute **as fast as possible** from *o* to *d*, for a given departure time $t_o$ from *o*? E.g.:

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

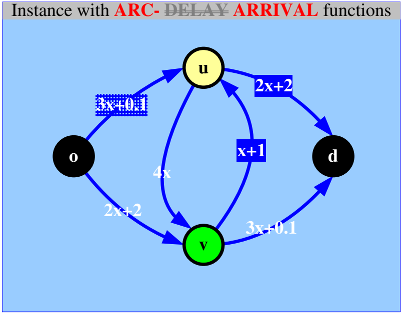


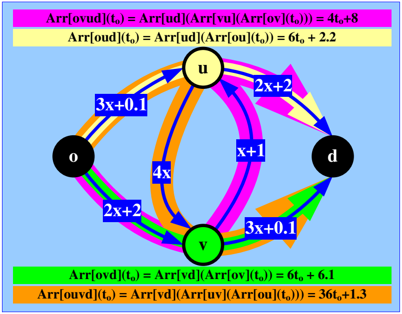




Q1 How would you commute **as fast as possible** from $o$ to $d$, for a given departure time $t_o$ from $o$? E.g.:

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

# Time-Dependent Shortest Path: Examples

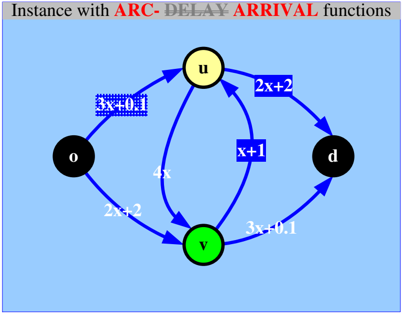


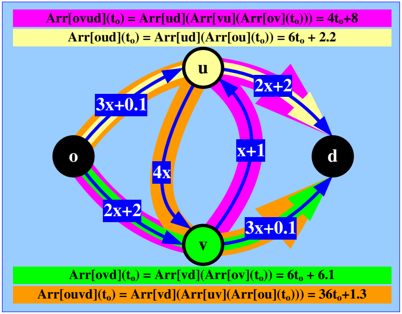




**Q1** How would you commute **as fast as possible** from $o$ to $d$, for a given departure time $t_o$ from $o$? E.g.:

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

# Time-Dependent Shortest Path: Examples




**Q1** How would you commute **as fast as possible** from $o$ to $d$, for a given departure time $t_o$ from $o$? E.g.:

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

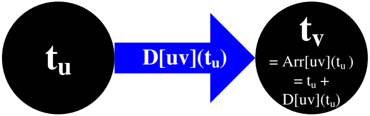
**A**

$$\text{earliest-arrival (path) } \textbf{function} = \begin{cases} \textbf{orange path}, & t_o \in [0, 0.03) \\ \textbf{yellow path}, & t_o \in [0.03, 2.9) \\ \textbf{purple path}, & t_o \in [2.9, +\infty) \end{cases}$$

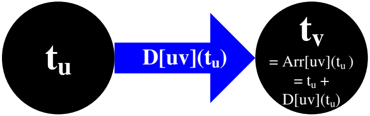
# Time-Dependent Shortest Path: Definitions

INPUT:

- *Directed* graph $G = (V, A)$, $n = |V|$.
- **Arc travel-time / arrival-time** functions:
  $\boxed{D[uv](t_u)}$ $\boxed{Arr[uv](t_u) = t_u + D[uv](t_u)}$



## INPUT:




- *Directed* graph $G = (V, A)$, $n = |V|$.
- **Arc travel-time / arrival-time** functions:
  $D[uv](t_u)$ $Arr[uv](t_u) = t_u + D[uv](t_u)$

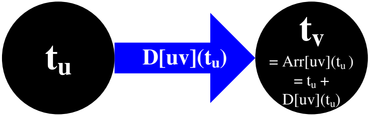
## DEFINITIONS:

- $P_{o,d}$: Set of *od*-paths; $\pi = (a_1, \ldots, a_k) \in P_{o,d}$
- **Path travel-time / arrival-time** functions:
  $Arr[\pi](t_o) = Arr[a_k](Arr[a_{k-1}](\cdots(Arr[a_1](t_o))\cdots))$ /* function composition */
  $D[\pi](t_o) = Arr[\pi](t_o) - t_o$
- **Earliest-arrival / Shortest-travel-time** functions:
  $Arr[o,d](t_o) = \min_{\pi \in P_{o,d}} \{ Arr[\pi](t_o) \}$, $D[o,d](t_o) = Arr[o,d](t_o) - t_o$

# Time-Dependent Shortest Path: Definitions

## INPUT:

- *Directed* graph $G = (V, A)$, $n = |V|$.
- **Arc travel-time / arrival-time** functions:
  $\boxed{D[uv](t_u)}$ $\boxed{Arr[uv](t_u) = t_u + D[uv](t_u)}$




## DEFINITIONS:

- $P_{o,d}$: Set of *od*-paths; $\pi = (a_1, \ldots, a_k) \in P_{o,d}$
- **Path travel-time / arrival-time** functions:
  $Arr[\pi](t_o) = Arr[a_k](Arr[a_{k-1}](\cdots(Arr[a_1](t_o))\cdots))$ /* function composition */
  $D[\pi](t_o) = Arr[\pi](t_o) - t_o$
- **Earliest-arrival / Shortest-travel-time** functions:
  $Arr[o,d](t_o) = \min_{\pi \in P_{o,d}} \{ Arr[\pi](t_o) \}$, $D[o,d](t_o) = Arr[o,d](t_o) - t_o$

## GOALS:

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

# FIFO (a.k.a. Non-Overtaking) Property in TD Networks

## FIFO Arc-Travel-Time Functions

Slopes of all $D[uv](t) \geq -1$ (e.g., for vehicles in road networks).

$\Rightarrow$ **non-decreasing** arc-arrival, path-arrival and earliest-arrival functions.

$\Rightarrow$ No reason to wait at vertices while moving along a path.

# FIFO (a.k.a. Non-Overtaking) Property in TD Networks

## FIFO Arc-Travel-Time Functions

Slopes of all $D[uv](t) \geq -1$ (e.g., for vehicles in road networks).

$\Rightarrow$ **non-decreasing** arc-arrival, path-arrival and earliest-arrival functions.

$\Rightarrow$ No reason to wait at vertices while moving along a path.

## Non-FIFO Arc-Travel-Time Functions

Possibly profitable to *wait at some vertex* (e.g., in public transport).

$\Rightarrow$ **Forbidden waiting:** $\nexists$ subpath optimality; $\mathcal{NP}-$hard. (Orda-Rom (1990))

$\Rightarrow$ **Unrestricted waiting:** Equivalent to FIFO. (Dreyfus (1969))

# FIFO (a.k.a. Non-Overtaking) Property in TD Networks

## FIFO Arc-Travel-Time Functions

Slopes of all $D[uv](t) \geq -1$ (e.g., for vehicles in road networks).

$\Rightarrow$ **non-decreasing** arc-arrival, path-arrival and earliest-arrival functions.

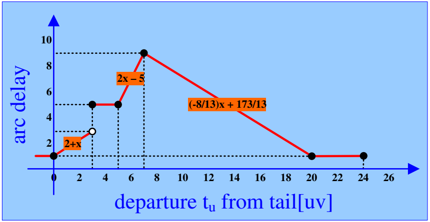
$\Rightarrow$ No reason to wait at vertices while moving along a path.

## Non-FIFO Arc-Travel-Time Functions

Possibly profitable to *wait at some vertex* (e.g., in public transport).

$\Rightarrow$ **Forbidden waiting:** $\nexists$ subpath optimality; $\mathcal{NP}$–hard. (Orda-Rom (1990))

$\Rightarrow$ **Unrestricted waiting:** Equivalent to FIFO. (Dreyfus (1969))




FIFO arc delay example

# FIFO (a.k.a. Non-Overtaking) Property in TD Networks

## FIFO Arc-Travel-Time Functions

Slopes of all $D[uv](t) \geq -1$ (e.g., for vehicles in road networks).

$\Rightarrow$ **non-decreasing** arc-arrival, path-arrival and earliest-arrival functions.

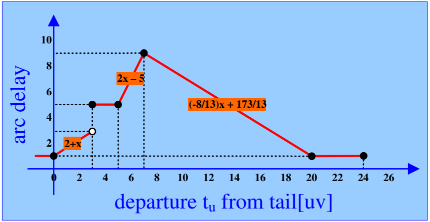
$\Rightarrow$ No reason to wait at vertices while moving along a path.

## Non-FIFO Arc-Travel-Time Functions

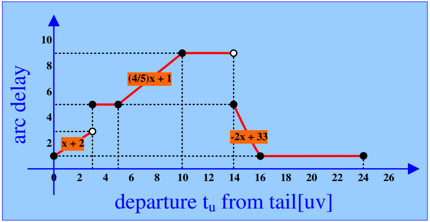
Possibly profitable to *wait at some vertex* (e.g., in public transport).

$\Rightarrow$ **Forbidden waiting:** $\nexists$ subpath optimality; $\mathcal{NP}$−hard. (Orda-Rom (1990))

$\Rightarrow$ **Unrestricted waiting:** Equivalent to FIFO. (Dreyfus (1969))




FIFO arc delay example




Non-FIFO arc delay example

# FIFO (a.k.a. Non-Overtaking) Property in TD Networks

## FIFO Arc-Travel-Time Functions

Slopes of all $D[uv](t) \geq -1$ (e.g., for vehicles in road networks).

$\Rightarrow$ **non-decreasing** arc-arrival, path-arrival and earliest-arrival functions.

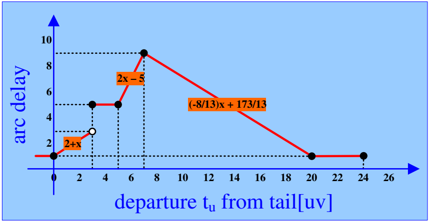
$\Rightarrow$ No reason to wait at vertices while moving along a path.

## Non-FIFO Arc-Travel-Time Functions

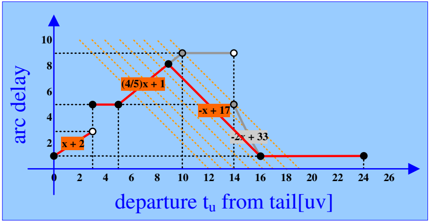
Possibly profitable to *wait at some vertex* (e.g., in public transport).

$\Rightarrow$ **Forbidden waiting:** $\nexists$ subpath optimality; $\mathcal{NP}$−hard. (Orda-Rom (1990))

$\Rightarrow$ **Unrestricted waiting:** Equivalent to FIFO. (Dreyfus (1969))




FIFO arc delay example

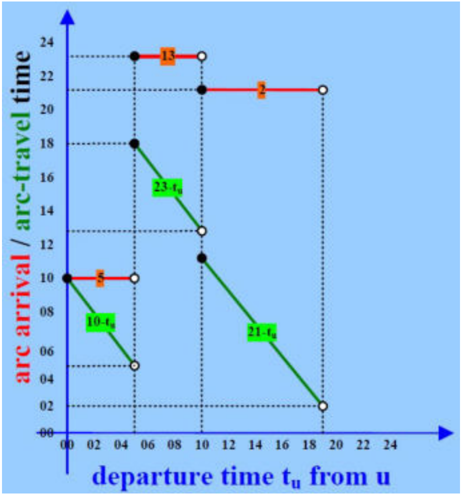



Equivalent FIFO arc delay (arbitrary waiting)

# TDSP vs. vs. EA-Paths in Temporal Graphs

Earliest-arrival paths in **temporal graphs** with **step functions for arc-delays** and **unrestricted waiting** $\approx$ Earliest-arrival paths in **FIFO-abiding TDSP networks** with pwl arc-delay functions.

Earliest-arrival paths in **temporal graphs** with **step functions for arc-delays** and **unrestricted waiting** ≈ Earliest-arrival paths in **FIFO-abiding TDSP networks** with pwl arc-delay functions.



# TDSP in FIFO networks: Complexity

...for piecewise-linear arc-delay functions, with $K$ breakpoints in total...

1. Compute *earliest-arrival-time* at $d$, for given $(o, t_o)$:

2. Compute *succinct representations* of $Arr[o, d]$, for all departure times:

# TDSP in FIFO networks: Complexity

...for piecewise-linear arc-delay functions, with $K$ breakpoints in total...

1. Compute *earliest-arrival-time* at $d$, for given $(o, t_o)$:
   - Time-dependent variant of `Dijkstra` (`TDD`) works for **FIFO instances**. (Dreyfus (1969); Orda-Rom (1990))
   - Time-dependent variant of `Bellman-Ford` (`TDBF`) works for **FIFO instances**. (Orda-Rom (1990))
2. Compute *succinct representations* of $Arr[o, d]$, for all departure times:

# TDSP in FIFO networks: Complexity

...for piecewise-linear arc-delay functions, with $K$ breakpoints in total...

1. Compute *earliest-arrival-time* at $d$, for given $(o, t_o)$:
   - Time-dependent variant of `Dijkstra` (`TDD`) works for **FIFO instances**.
     (Dreyfus (1969); Orda-Rom (1990))
   - Time-dependent variant of `Bellman-Ford` (`TDBF`) works for **FIFO instances**. (Orda-Rom (1990))
2. Compute *succinct representations* of $Arr[o, d]$, for all departure times:
   - Succinct representation of $Arr[o, d]$ may require space $(K + 1) \cdot n^{\Theta(\log(n))}$, even for **sparse** networks with **affine** arc-travel-time functions.
     (Foschini-Hershberger-Suri (2011))
   - $\exists$ polynomial-time point-to-point $(1 + \varepsilon)-$approximation algorithms for $D[o, d]$, requiring space $\mathrm{O}(K + 1)$ per $(o, d)$-pair.
     (Dehne-Omran-Sack (2010); Foschini-Hershberger-Suri (2011))

# Measuring Quality of Algorithms for TDSP...

- IN THEORY: Guaranteed quality of the proposed solution.
  - *OPT*: The cost of a min-travel-time *od*-path.
  - *ACTUAL*: The cost of the proposed *od*-path.
  - **Maximum absolute error**: $MAE = ACTUAL - OPT$.
  - **Relative error** (the $(ACTUAL - OPT)/OPT \leq \epsilon$ value of a $1 + \epsilon$ approximation, as a percentage): $\boxed{RE = 100 \cdot \frac{ACTUAL - OPT}{OPT}\,\%}$

# Measuring Quality of Algorithms for TDSP...

- IN THEORY: Guaranteed quality of the proposed solution.
  - *OPT*: The cost of a min-travel-time *od*-path.
  - *ACTUAL*: The cost of the proposed *od*-path.
  - **Maximum absolute error**: $MAE = ACTUAL - OPT$.
  - **Relative error** (the $(ACTUAL - OPT)/OPT \leq \epsilon$ value of a $1 + \epsilon$ approximation, as a percentage): $RE = 100 \cdot \frac{ACTUAL - OPT}{OPT}\ \%$
- IN PRACTICE: Avg query-response time must be really small, for large-scale instances (e.g., **< 1msec** for instances with millions of arcs).

R1 A relative error of 1% implies an *extra delay* of **at most 36sec per hour** of optimal-travel-time!!!

# Measuring Quality of Algorithms for TDSP...

- IN THEORY: Guaranteed quality of the proposed solution.
  - *OPT*: The cost of a min-travel-time *od*-path.
  - *ACTUAL*: The cost of the proposed *od*-path.
  - **Maximum absolute error**: $MAE = ACTUAL - OPT$.
  - **Relative error** (the $(ACTUAL - OPT)/OPT \leq \epsilon$ value of a $1+\epsilon$ approximation, as a percentage): $RE = 100 \cdot \frac{ACTUAL-OPT}{OPT}\ \%$

- IN PRACTICE: Avg query-response time must be really small, for large-scale instances (e.g., **< 1msec** for instances with millions of arcs).

R1 A relative error of 1% implies an *extra delay* of **at most 36sec per hour** of optimal-travel-time!!!

R2 Hard to compare query times of algorithms *running on different machines*. A `speedup` $= Time(Dij)/Time(Alg)$ over a baseline (time-dependent) Dijkstra implementation might be more meaningful.

# Recap of travel-time oracles

## ...approximation, preprocessing and query algorithms...

# About Oracles for TDSP...

...FIFO abiding, pwl arc-delay functions with $K$ breakpoints in the arc-travel-time functions...

QUESTION: $\exists$ some **data structure** (to precompute), and a **query algorithm** (to approximately answer routing requests on-the-fly) for TDSP that requires **reasonable space** and can answer **arbitrary queries** efficiently?

# About Oracles for TDSP...

...FIFO abiding, pwl arc-delay functions with $K$ breakpoints in the arc-travel-time functions...

**QUESTION:** $\exists$ some **data structure** (to precompute), and a **query algorithm** (to approximately answer routing requests on-the-fly) for TDSP that requires **reasonable space** and can answer **arbitrary queries** efficiently?

- **Trivial solution (I):** Precompute **all** $(1+\varepsilon)-$upper-approximating trravel-time functions (**summaries**) $\Delta[o,d]$ for every *od*-pair.
  - $O(n^2(K+1))$ space — /* store **all** (succinct representations of) $\Delta[o,d]$ */
  - $O(\log\log(K))$ query time — /* look for the **proper leg** in $\Delta[o,d]$ */
  - $(1+\varepsilon)-$stretch

# About Oracles for TDSP...

...FIFO abiding, pwl arc-delay functions with $K$ breakpoints in the arc-travel-time functions...

**QUESTION:** $\exists$ some **data structure** (to precompute), and a **query algorithm** (to approximately answer routing requests on-the-fly) for TDSP that requires **reasonable space** and can answer **arbitrary queries** efficiently?

- **Trivial solution (I):** Precompute **all** $(1+\varepsilon)-$upper-approximating trravel-time functions (**summaries**) $\Delta[o,d]$ for every *od*-pair.
  - $O(n^2(K+1))$ space /* store **all** (succinct representations of) $\Delta[o,d]$ */
  - $O(\log\log(K))$ query time /* look for the **proper leg** in $\Delta[o,d]$ */
  - $(1+\varepsilon)-$stretch
- **Trivial solution (II): No preprocessing**. Respond to queries with `TD-Dijkstra`:
  - $O(n+m+K)$ space /* only store the instance */
  - $O([m+n\log(n)]\times\log\log(K))$ query time /* run TD-Dijkstra */
  - $1-$stretch

# About Oracles for TDSP...

...FIFO abiding, pwl arc-delay functions with $K$ breakpoints in the arc-travel-time functions...

**QUESTION:** $\exists$ some **data structure** (to precompute), and a **query algorithm** (to approximately answer routing requests on-the-fly) for TDSP that requires **reasonable space** and can answer **arbitrary queries** efficiently?

- **Trivial solution (I):** Precompute **all** $(1+\varepsilon)-$upper-approximating trravel-time functions (**summaries**) $\Delta[o, d]$ for every *od*-pair.
  - $O(n^2(K+1))$ space /* store **all** (succinct representations of) $\Delta[o, d]$ */
  - $O(\log\log(K))$ query time /* look for the **proper leg** in $\Delta[o, d]$ */
  - $(1+\varepsilon)-$stretch
- **Trivial solution (II):** **No preprocessing**. Respond to queries with `TD-Dijkstra`:
  - $O(n+m+K)$ space /* only store the instance */
  - $O([m + n\log(n)] \times \log\log(K))$ query time /* run TD-Dijkstra */
  - $1-$stretch

**GOAL:** Can we do better?

- Assure **subquadratic** space & **sublinear** query time.
- Provide a smooth tradeoff among space / query time / stretch.

# Assumptions: Statement (I)

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

# Assumptions: Statement (I)

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

**ASSUMPTION 1:** (bounded travel time slopes)

Slopes of $D[o, d] \in [-\Lambda_{\min}, \Lambda_{\max}]$, for constants $\Lambda_{\max} > 0$, $\Lambda_{\min} \in [0, 1)$.

# Assumptions: Statement (I)

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

**ASSUMPTION 1:** (bounded travel time slopes)

Slopes of $D[o,d] \in [-\Lambda_{\min}, \Lambda_{\max}]$, for constants $\Lambda_{\max} > 0$, $\Lambda_{\min} \in [0,1)$.

**ASSUMPTION 2:** (bounded opposite trips)

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

# Assumptions: Statement (I)

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

**ASSUMPTION 1:** (bounded travel time slopes)

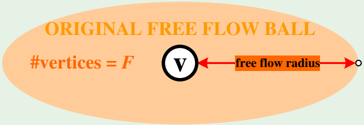
Slopes of $D[o,d] \in [-\Lambda_{\min}, \Lambda_{\max}]$, for constants $\Lambda_{\max} > 0$, $\Lambda_{\min} \in [0,1)$.

**ASSUMPTION 2:** (bounded opposite trips)

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

**ASSUMPTION 3:** (growth of free-flow balls)

The growth of free-flow balls from an origin is at most polylogarithmic.



# Assumptions: Statement (I)

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

**ASSUMPTION 1:** (bounded travel time slopes)

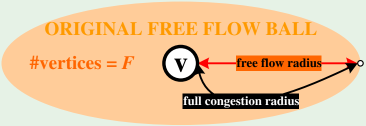
Slopes of $D[o,d] \in [-\Lambda_{\min}, \Lambda_{\max}]$, for constants $\Lambda_{\max} > 0$, $\Lambda_{\min} \in [0,1)$.

**ASSUMPTION 2:** (bounded opposite trips)

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

**ASSUMPTION 3:** (growth of free-flow balls)

The growth of free-flow balls from an origin is at most polylogarithmic.



# Assumptions: Statement (I)

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

**ASSUMPTION 1:** (bounded travel time slopes)

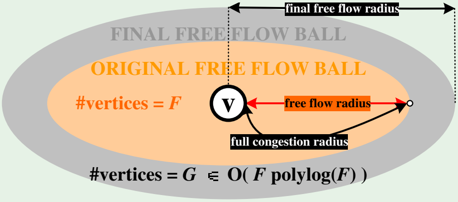
Slopes of $D[o,d] \in [-\Lambda_{\min}, \Lambda_{\max}]$, for constants $\Lambda_{\max} > 0$, $\Lambda_{\min} \in [0,1)$.

**ASSUMPTION 2:** (bounded opposite trips)

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

**ASSUMPTION 3:** (growth of free-flow balls)

The growth of free-flow balls from an origin is at most polylogarithmic.



# Assumptions: Statement (II)

- Necessary assumption for the analysis of the hierarchical oracle (HORN).
- **Dijkstra Rank** $DR[o,d](t_o)$: size of smallest ball from $(o,t_o)$, until $d$ is settled.

**ASSUMPTION 4:** (correlation of travel-times with Dijkstra ranks)

For $\lambda \in o\left(\frac{\log(n)}{\log\log(n)}\right)$ the following hold:

# Assumptions: Statement (II)

- Necessary assumption for the analysis of the hierarchical oracle (HORN).
- **Dijkstra Rank** $DR[o,d](t_o)$: size of smallest ball from $(o, t_o)$, until $d$ is settled.

**ASSUMPTION 4:** (correlation of travel-times with Dijkstra ranks)

For $\lambda \in o\left(\frac{\log(n)}{\log\log(n)}\right)$ the following hold:

1. The **Dijkstra rank** is upper-bounded by a degree-$\lambda$ polynomial of the corresponding **travel-time**: $DR[o,d](t_o) \in \tilde{O}\left((D[o,d](t_o))^{\lambda}\right)$

# Assumptions: Statement (II)

- Necessary assumption for the analysis of the hierarchical oracle (HORN).
- **Dijkstra Rank** $DR[o,d](t_o)$: size of smallest ball from $(o,t_o)$, until $d$ is settled.

**ASSUMPTION 4:** (correlation of travel-times with Dijkstra ranks)

For $\lambda \in \mathrm{o}\left(\frac{\log(n)}{\log\log(n)}\right)$ the following hold:

1. The **Dijkstra rank** is upper-bounded by a degree-$\lambda$ polynomial of the corresponding **travel-time**: $DR[o,d](t_o) \in \tilde{\mathrm{O}}\left((D[o,d](t_o))^{\lambda}\right)$
2. The **travel-time** is upper-bounded by a degree-$\left(\frac{1}{\lambda}\right)$ polynomial of the **Dijkstra-rank**: $D[o,d](t_o) \in \tilde{\mathrm{O}}\left((DR[o,d](t_o))^{1/\lambda}\right)$

# Assumptions: Statement (II)

- Necessary assumption for the analysis of the hierarchical oracle (HORN).
- **Dijkstra Rank** $DR[o,d](t_o)$: size of smallest ball from $(o, t_o)$, until $d$ is settled.

**ASSUMPTION 4:** (correlation of travel-times with Dijkstra ranks)

For $\lambda \in o\left(\frac{\log(n)}{\log\log(n)}\right)$ the following hold:

1. The **Dijkstra rank** is upper-bounded by a degree-$\lambda$ polynomial of the corresponding **travel-time**: $DR[o,d](t_o) \in \tilde{O}\left((D[o,d](t_o))^{\lambda}\right)$
2. The **travel-time** is upper-bounded by a degree-$\left(\frac{1}{\lambda}\right)$ polynomial of the **Dijkstra-rank**: $D[o,d](t_o) \in \tilde{O}\left((DR[o,d](t_o))^{1/\lambda}\right)$

R1 The **doubling dimension** assumption used in **metric embeddings** correlates the distance metric with the Dijkstra-rank metric. For constant $\lambda \geq 1$, we can have an oracle providing a PTAS, for static metrics.

# Assumptions: Statement (II)

- Necessary assumption for the analysis of the hierarchical oracle (HORN).
- **Dijkstra Rank** $DR[o,d](t_o)$: size of smallest ball from $(o, t_o)$, until $d$ is settled.

**ASSUMPTION 4:** (correlation of travel-times with Dijkstra ranks)

For $\lambda \in o\left(\frac{\log(n)}{\log\log(n)}\right)$ the following hold:

1. The **Dijkstra rank** is upper-bounded by a degree-$\lambda$ polynomial of the corresponding **travel-time**: $DR[o,d](t_o) \in \tilde{O}\left((D[o,d](t_o))^{\lambda}\right)$
2. The **travel-time** is upper-bounded by a degree-$\left(\frac{1}{\lambda}\right)$ polynomial of the **Dijkstra-rank**: $D[o,d](t_o) \in \tilde{O}\left((DR[o,d](t_o))^{1/\lambda}\right)$

R1 The **doubling dimension** assumption used in **metric embeddings** correlates the distance metric with the Dijkstra-rank metric. For constant $\lambda \geq 1$, we can have an oracle providing a PTAS, for static metrics.

R2 We have proved that *Assumption 4* $\Rightarrow$ *Assumption 3*

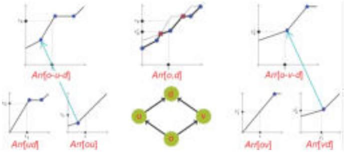
# Overview of Our Theoretical Results

| | preprocessing space/time | query time | recursion depth |
|---|---|---|---|
| (ICALP (2014) ALGORITHMICA (2016)) | $K^* \cdot n^{2-\beta+o(1)}$ | $n^{\delta+o(1)}$ | $R \in O(1)$ |
| TRAPONLY (ISAAC (2016)) | $n^{2-\beta+o(1)}$ | $n^{\delta+o(1)}$ | $R \approx \frac{\delta}{a} - 1$ |
| FLAT (ISAAC (2016)) & CFLAT (ATMOS (2017)) | $n^{2-\beta+o(1)}$ | $n^{\delta+o(1)}$ | $R \approx \frac{2\delta}{a} - 1$ |
| HORN (ISAAC (2016)) | $n^{2-\beta+o(1)}$ | $\approx \Gamma[o,d](t_o)^{\delta+o(1)}$ | $R \approx \frac{2\delta}{a} - 1$ |

- ...assuming TD-instances with period $T = n^a$ for constant $a \in (0,1)$.
- ...achieving approx. guarantee $1 + \varepsilon \cdot \frac{(\varepsilon/\psi)^{R+1}}{(\varepsilon/\psi)^{R+1} - 1}$.
- For all oracles, except for the first, we assume that $\beta \downarrow 0$.

# Recap of travel-time oracles

**...approximation, preprocessing and query algorithms...**



**GIVEN:** Arc-traversal-time (continuous, pwl) functions $D[uv] : [0, T) \mapsto \mathbb{R}_{>0}$.

**GOAL:** Succinct representations of (unknown, pwl,continuous) min-travel-time functions.

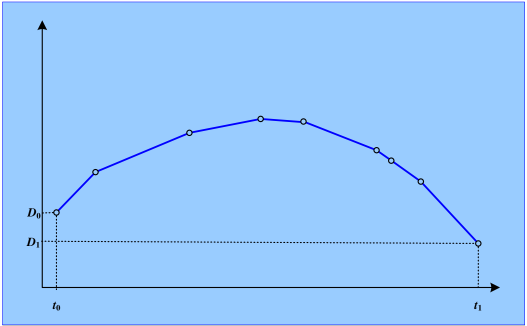
$$D[o, d] = \min_{\pi \in P_{o,d}} \{ D[\pi] \} : [0, T) \mapsto \mathbb{R}_{>0}$$

**PROBLEM:** Superpolynomial time/space complexities.

**SOLUTION:** **Upper-approximations** with polynomial time complexity.

**CHALLENGE:** One-to-all construction of succinct representations for the approximate min-travel-time functions.

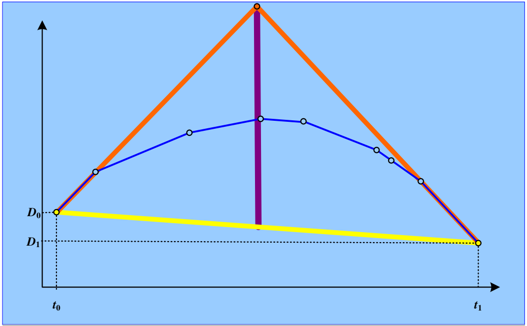
- *Assume concavity* (to be removed later) of the unknown functions $D[o, v]$ in an interval $[t_o, t_1]$ of departure times.




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

- *Bisect on the common departure-times axis:* Recursively keep sampling, **simultaneously for all active destinations** $v \in V$, distance values from $o$, at mid-points of currently unsatisfied intervals (wrt approximation guarantee).

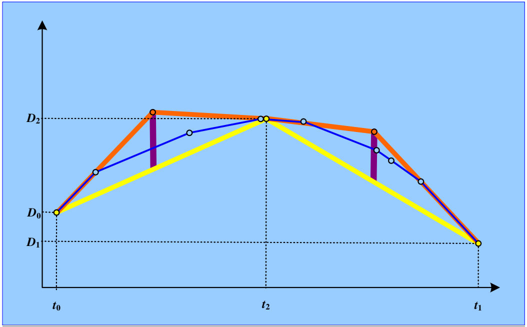
- *Assume concavity* (to be removed later) of the unknown functions $D[o, v]$ in an interval $[t_o, t_1]$ of departure times.




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

- *Bisect on the common departure-times axis:* Recursively keep sampling, **simultaneously for all active destinations** $v \in V$, distance values from $o$, at mid-points of currently unsatisfied intervals (wrt approximation guarantee).

- *Assume concavity* (to be removed later) of the unknown functions $D[o, v]$ in an interval $[t_o, t_1]$ of departure times.



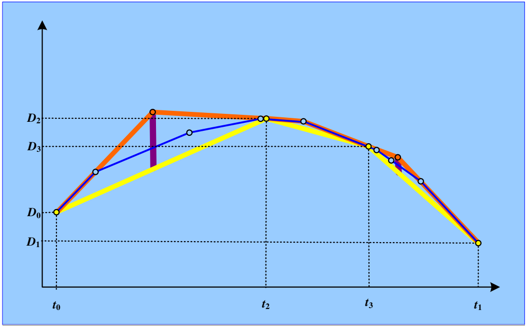
Example of Bisection Execution : Level-1 Recursion

- *Bisect on the common departure-times axis:* Recursively keep sampling, **simultaneously for all active destinations** $v \in V$, distance values from $o$, at mid-points of currently unsatisfied intervals (wrt approximation guarantee).

# Approximation Algorithms for Path-Travel-Time Functions

## First Attempt: Bisection Algorithm (BIS)...

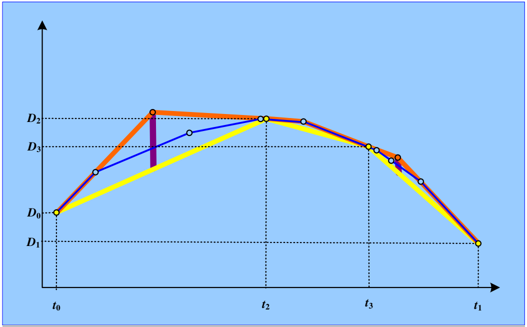
- *Assume concavity* (to be removed later) of the unknown functions $D[o, v]$ in an interval $[t_o, t_1]$ of departure times.




Example of Bisection Execution : Level-2 Recursion

- *Bisect on the common departure-times axis:* Recursively keep sampling, **simultaneously for all active destinations** $v \in V$, distance values from $o$, at mid-points of currently unsatisfied intervals (wrt approximation guarantee).

- *Assume concavity* (to be removed later) of the unknown functions $D[o, v]$ in an interval $[t_o, t_1]$ of departure times.



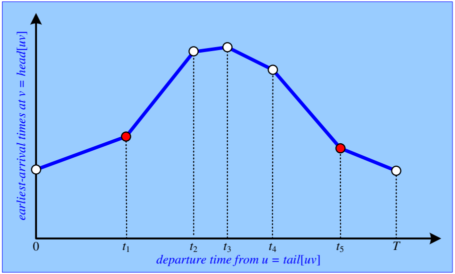

Example of Bisection Execution : Level-2 Recursion

- *Bisect on the common departure-times axis:* Recursively keep sampling, **simultaneously for all active destinations** $v \in V$, distance values from $o$, at mid-points of currently unsatisfied intervals (wrt approximation guarantee).

Remark: Analysis based on *closed-form* expression of **maximum absolute error** (length of purple line).

For each *continuous, pwl, not necessarily concave* arc-delay function:

1. Run `Reverse TD-Dijkstra` on **reversed graph**, to project each **concavity-spoiling PB** to a departure-time (called **primitive image** -- PI) at the origin $o$.




2. For each pair of **consecutive PIs** of $o$ in $[0, T)$, run `Bisection` for the corresponding departure-times interval.

3. Return the **concatenation** of upper-approximating trravel-time summaries.

**Theorem:** Complexity of `Bisection`

For each (common) origin $o \in V$,

- **SPACE**:

$$\mathrm{O}\left((K^* + 1) \cdot n \cdot \frac{1}{\varepsilon} \cdot \max_{v \in V}\left\{\log\left(\frac{D_{\max}[o,v](0,T)}{D_{\min}[o,v](0,T)}\right)\right\}\right)$$

**Theorem:** Complexity of `Bisection`

For each (common) origin $o \in V$,

- **SPACE**:

$$O\left((K^* + 1) \cdot n \cdot \frac{1}{\varepsilon} \cdot \max_{v \in V}\left\{\log\left(\frac{D_{\max}[o,v](0,T)}{D_{\min}[o,v](0,T)}\right)\right\}\right)$$

- **TIME** (in number of TDSP-Probes):

$$O\left((K^* + 1) \cdot \max_{v \in V}\left\{\log\left(\frac{T \cdot (\Lambda_{\max} + 1)}{\varepsilon D_{\min}[o,v](0,T)}\right)\right\} \cdot \frac{1}{\varepsilon} \max_{v \in V}\left\{\log\left(\frac{D_{\max}[o,v](0,T)}{D_{\min}[o,v](0,T)}\right)\right\}\right)$$

**Theorem:** Complexity of `Bisection`

For each (common) origin $o \in V$,

- **SPACE**:

$$O\left((K^* + 1) \cdot n \cdot \frac{1}{\varepsilon} \cdot \max_{v \in V} \left\{ \log\left( \frac{D_{\max}[o,v](0,T)}{D_{\min}[o,v](0,T)} \right) \right\} \right)$$

- **TIME** (in number of TDSP-Probes):

$$O\left((K^* + 1) \cdot \max_{v \in V} \left\{ \log\left( \frac{T \cdot (\Lambda_{\max} + 1)}{\varepsilon D_{\min}[o,v](0,T)} \right) \right\} \cdot \frac{1}{\varepsilon} \max_{v \in V} \left\{ \log\left( \frac{D_{\max}[o,v](0,T)}{D_{\min}[o,v](0,T)} \right) \right\} \right)$$

| PROS | CONS |
|---|---|
| | |

**Theorem:** Complexity of `Bisection`

For each (common) origin $o \in V$,

- **SPACE**:

$$O\left((K^* + 1) \cdot n \cdot \frac{1}{\varepsilon} \cdot \max_{v \in V}\left\{\log\left(\frac{D_{\max}[o,v](0,T)}{D_{\min}[o,v](0,T)}\right)\right\}\right)$$

- **TIME** (in number of TDSP-Probes):

$$O\left((K^* + 1) \cdot \max_{v \in V}\left\{\log\left(\frac{T \cdot (\Lambda_{\max}+1)}{\varepsilon D_{\min}[o,v](0,T)}\right)\right\} \cdot \frac{1}{\varepsilon} \max_{v \in V}\left\{\log\left(\frac{D_{\max}[o,v](0,T)}{D_{\min}[o,v](0,T)}\right)\right\}\right)$$

| PROS | CONS |
|---|---|
| Simplicity.<br>Space-optimal for *concave* functions.<br>First one-to-all approximation. | |

**Theorem: Complexity of `Bisection`**

For each (common) origin $o \in V$,

- **SPACE**:

$$O\left((K^* + 1) \cdot n \cdot \frac{1}{\varepsilon} \cdot \max_{v \in V}\left\{\log\left(\frac{D_{\max}[o,v](0,T)}{D_{\min}[o,v](0,T)}\right)\right\}\right)$$

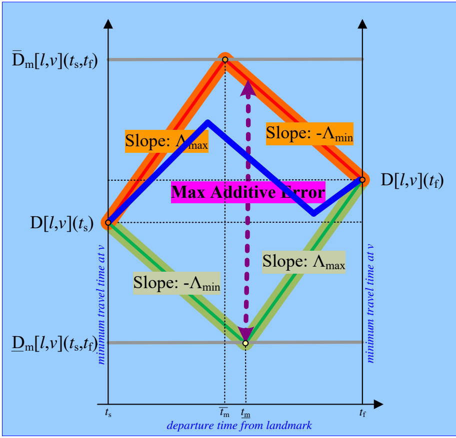
- **TIME** (in number of TDSP-Probes):

$$O\left((K^* + 1) \cdot \max_{v \in V}\left\{\log\left(\frac{T \cdot (\Lambda_{\max}+1)}{\varepsilon D_{\min}[o,v](0,T)}\right)\right\} \cdot \frac{1}{\varepsilon} \max_{v \in V}\left\{\log\left(\frac{D_{\max}[o,v](0,T)}{D_{\min}[o,v](0,T)}\right)\right\}\right)$$

| PROS | CONS |
|---|---|
| Simplicity.<br>Space-optimal for *concave* functions.<br>First one-to-all approximation. | Linear dependence on degree of disconcavity $K^*$. |

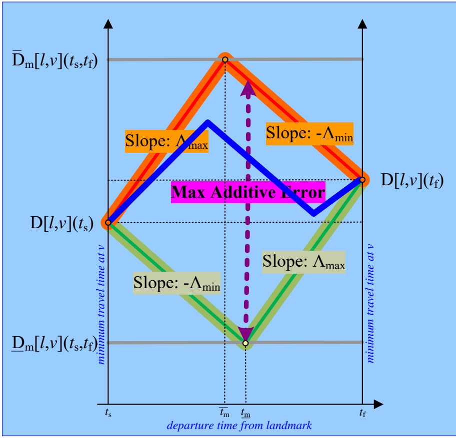
TRAP *samples simultaneously* all **min-travel-time values** from $\ell$, for ever-finer departure-points, until the approximation guarantee is achieved *for all destinations*.

- *Avoids dependence* on the shape of the function to approximate.
- *Exploits knowledge* of min/max slopes $\Lambda_{min}/\Lambda_{max}$ of min-travel-time functions.



TRAP *samples simultaneously* all **min-travel-time values** from $\ell$, for ever-finer departure-points, until the approximation guarantee is achieved *for all destinations*.

- *Avoids dependence* on the shape of the function to approximate.
- *Exploits knowledge* of min/max slopes $\Lambda_{min}/\Lambda_{max}$ of min-travel-time functions.




**Orange line:** Upper-approximating function $\Delta[\ell, v]$.

**Green line:** Lower-approximation.

**Blue line:** The unknown min-travel-time function $D[\ell, v]$.

## Theorem: Complexity of Trapezoidal Method (TRAP)

Split $[0, T)$ into $\left\lceil \frac{T}{\tau} \right\rceil$ length-$\tau$ intervals.

- **TIME & SPACE:** $\Delta[\ell, v] =$ concatenation of approximations by TRAP for all subintervals. $\Rightarrow \mathrm{O}\left(\frac{T}{\tau}\right)$ BPs and TDSP-probes.
- **APPROXIMABILITY**:
  **IF** $\min_{k \in \mathbb{N}: k\tau \in [0,T)} \{ D[\ell, v](k\tau) \} \geq \left(1 + \frac{1}{\epsilon}\right) \Lambda_{\max} \cdot \tau$
  **THEN** $\Delta[\ell, v]$ is $(1 + \epsilon)$-approximation of $D[\ell, v]$ in $[0, T)$.

**Theorem:** Complexity of Trapezoidal Method (TRAP)

Split $[0, T)$ into $\left\lceil \frac{T}{\tau} \right\rceil$ length-$\tau$ intervals.

- **TIME & SPACE:** $\Delta[\ell, v] =$ concatenation of approximations by TRAP for all subintervals. $\Rightarrow \mathrm{O}\left(\frac{T}{\tau}\right)$ BPs and TDSP-probes.
- **APPROXIMABILITY**:
  **IF** $\min_{k \in \mathbb{N}: k\tau \in [0,T)} \{ D[\ell, v](k\tau) \} \geq \left(1 + \frac{1}{\epsilon}\right) \Lambda_{\max} \cdot \tau$
  **THEN** $\Delta[\ell, v]$ is $(1 + \epsilon)$-approximation of $D[\ell, v]$ in $[0, T)$.

| PROS | CONS |
|---|---|
| Simplicity. | |
| One-to-all approximation. | |
| Independence from shape of the function to approximate. | |

# Approximation Algorithms for Path-Travel-Time Functions

Second Attempt: Trapezoidal Algorithm (TRAP)...

**Theorem:** Complexity of Trapezoidal Method (TRAP)

Split $[0, T)$ into $\left\lceil \frac{T}{\tau} \right\rceil$ length-$\tau$ intervals.

- **TIME & SPACE:** $\Delta[\ell, v] =$ concatenation of approximations by TRAP for all subintervals. $\Rightarrow \mathrm{O}\left(\frac{T}{\tau}\right)$ BPs and TDSP-probes.
- **APPROXIMABILITY**:
  **IF** $\min_{k \in \mathbb{N}: k\tau \in [0,T)} \{ D[\ell, v](k\tau) \} \geq \left(1 + \frac{1}{\epsilon}\right) \Lambda_{\max} \cdot \tau$
  **THEN** $\Delta[\ell, v]$ is $(1+\epsilon)$-approximation of $D[\ell, v]$ in $[0, T)$.

**PROS**

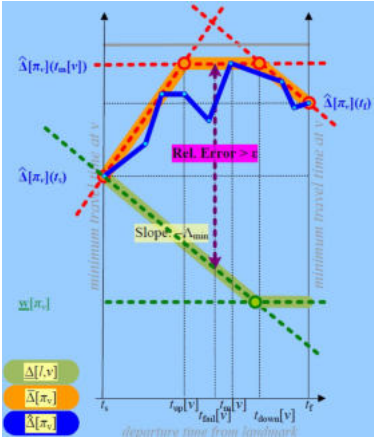
- Simplicity.
- One-to-all approximation.
- Independence from shape of the function to approximate.

**CONS**

- No theoretical guarantee of space-optimality.
- Inappropriate (in theory) for *``nearby'' vertices* around $\ell$.

CTRAP samples and stores *shortest-path trees*, rather than travel-time functions.

- Also *avoids dependence* on the shape of the function to approximate.
- *Exploits knowledge* of min slope $\Lambda_{min}$ of min-travel-time functions.
- Constructs **tighter upper-approximating functions**, by composing approximate arrival-time functions along Shortest-Path trees (in BFS order).



# Recap of travel-time oracles

**...approximation, preprocessing and query algorithms...**

# FLAT: Preprocessing Phase

- **Rationale:**
  - Identify a (small) subset $L$ of allegedly ``important'' vertices (**landmarks**) in the network, which are assumed to be *crucial for almost all optimal paths*.
  - Use the approximation algorithm (BIS, TRAP, or CTRAP) to compute approximate **travel-time summaries** (upper-approximating functions) $\Delta[\ell, v], \forall(\ell, v) \in L \times V$, s.t.:

$$\forall t \in [0, T), \ D[\ell, v](t) \leq \Delta[\ell, v](t) \leq (1 + \epsilon) \cdot D[\ell, v](t)$$

# FLAT: Preprocessing Phase

- **Rationale:**
  - Identify a (small) subset $L$ of allegedly ``important'' vertices (**landmarks**) in the network, which are assumed to be *crucial for almost all optimal paths*.
  - Use the approximation algorithm (BIS, TRAP, or CTRAP) to compute approximate **travel-time summaries** (upper-approximating functions) $\Delta[\ell, v], \forall(\ell, v) \in L \times V$, s.t.:

$$\forall t \in [0, T), \; D[\ell, v](t) \leq \Delta[\ell, v](t) \leq (1 + \epsilon) \cdot D[\ell, v](t)$$

- **Landmark Selection Policies:**
  - **RANDOM** (R): Independent and random selections, without repetitions.
  - **SPARSE-RANDOM** (SR): Sequential and random selections, excluding nearby vertices of already selected landmarks.
  - **SPARSE KAHIP** (SK): Selection of boundary vertices in a given KaHIP partition, excluding nearby vertices of already selected landmarks.
  - **BETWEENESS CENTRALITY** (BC): Sequential selection according to the BC-order, excluding nearby vertices of already selected landmarks.

# CFLAT: Preprocess trees rather than functions

- Challenge for FLAT: **Large preprocessing** requirements (typical for landmark-based algorithms).

# CFLAT: Preprocess trees rather than functions

- Challenge for FLAT: **Large preprocessing** requirements (typical for landmark-based algorithms).

- The *combinatorial structure* of the optimal solution changes over time **less frequently** than the corresponding min-travel-time function.

# CFLAT: Preprocess trees rather than functions

- Challenge for FLAT: **Large preprocessing** requirements (typical for landmark-based algorithms).

- The *combinatorial structure* of the optimal solution changes over time **less frequently** than the corresponding min-travel-time function.

- **Combinatorial FLAT** (CFLAT):
  - Forget about (upper-approximations of) travel-time functions.
  - Store only min-travel-time trees rooted at **time-stamped landmarks** $(\ell, t_\ell)$.
  - Avoid vertex IDs and represent parents in the trees only by their **relative order** in the adjacency lists of incoming arcs.
  - Store two different sequences per vertex $v$ and landmark $\ell$:
    - ★ A sequence of **departure-times** from the landmark.
    - ★ A sequence of **predecessors**, one per departure-time, in the corresponding unique $\ell v$-path.

# CFLAT: Store only sampled trees & avoid duplicates

- The CTRAP approximation algorithm:
  - avoids storing **travel-times**, and only stores the departure-times and the predecessors sequences for all $(\ell, v)$ pairs.
  - avoids storing **intermediate breakpoints**, between pairs of consecutive departure-time samples (saves 10M and 100M breakpoints per landmark in BER and GER, respectively).

# CFLAT: Store only sampled trees & avoid duplicates

- The CTRAP approximation algorithm:
  - avoids storing **travel-times**, and only stores the departure-times and the predecessors sequences for all $(\ell, v)$ pairs.
  - avoids storing **intermediate breakpoints**, between pairs of consecutive departure-time samples (saves 10M and 100M breakpoints per landmark in BER and GER, respectively).
- The CFLAT oracle:
  - merges consecutive breakpoints with **common predecessor**.
  - stores each sequence of departure times **only once** and lets all destinations corresponding to it to just point at it.
  - uses two **random hash functions** for fast recognition of identical sequences of departure times. In case of positive answer, exhaustively check the tautology of the two sequences.

# Recap of travel-time oracles

## ...approximation, preprocessing and query algorithms...

# $\texttt{FCA(N)}$ : A Simple Dijkstra-based Query Algorithm

(Kontogiannis-Papastavrou-Paraskevopoulos-Wagner-Zaroliagis (ALENEX 2016))

**Extended Forward Constant Approximation -- FCA(N)**

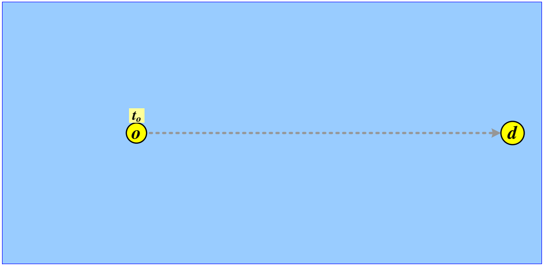
1. Grow TD-Dijkstra ball $B(o, t_o)$ until the $N$ closest landmarks $\ell_o, \ldots, \ell_{N-1}$ (or $d$) are settled.
2. **return** $\min_{i \in \{0,1,\ldots,N-1\}} \{ sol_i = D[o, \ell_i](t_o) + \Delta[\ell_i, d](t_i + D[o, \ell_i](t_o)) \}$

# FCA(N) : A Simple Dijkstra-based Query Algorithm

(Kontogiannis-Papastavrou-Paraskevopoulos-Wagner-Zaroliagis (ALENEX 2016))

**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** $\min_{i \in \{0,1,\ldots,N-1\}} \{ sol_i = D[o, \ell_i](t_o) + \Delta[\ell_i, d](t_i + D[o, \ell_i](t_o)) \}$

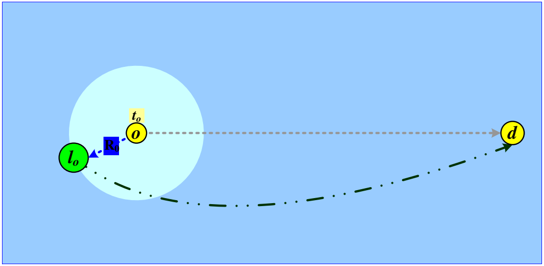


# FCA(N) : A Simple Dijkstra-based Query Algorithm

(Kontogiannis-Papastavrou-Paraskevopoulos-Wagner-Zaroliagis (ALENEX 2016))

**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** $\min_{i \in \{0,1,\ldots,N-1\}} \{ \; sol_i = D[o, \ell_i](t_o) + \Delta[\ell_i, d](t_i + D[o, \ell_i](t_o)) \; \}$

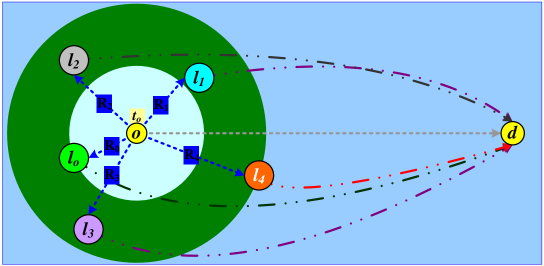


# FCA(N) : A Simple Dijkstra-based Query Algorithm

(Kontogiannis-Papastavrou-Paraskevopoulos-Wagner-Zaroliagis (ALENEX 2016))

**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** $\min_{i \in \{0,1,\ldots,N-1\}} \{ sol_i = D[o, \ell_i](t_o) + \Delta[\ell_i, d](t_i + D[o, \ell_i](t_o)) \}$

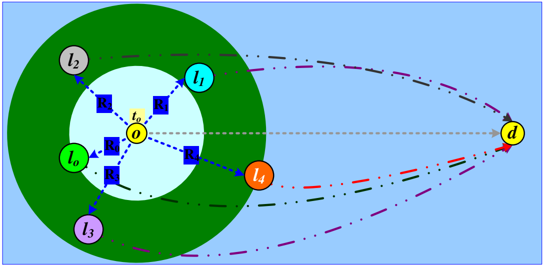


# FCA (N) : A Simple Dijkstra-based Query Algorithm

(Kontogiannis-Papastavrou-Paraskevopoulos-Wagner-Zaroliagis (ALENEX 2016))

**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

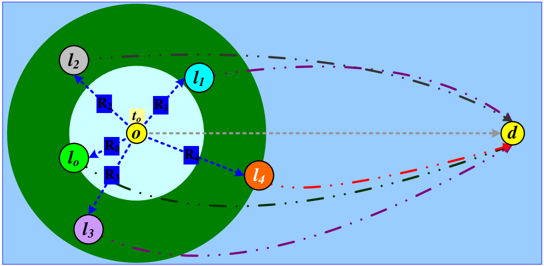
- In theory: Constant-approximation, for a metric-dependent constant.

# FCA (N) : A Simple Dijkstra-based Query Algorithm

(Kontogiannis-Papastavrou-Paraskevopoulos-Wagner-Zaroliagis (ALENEX 2016))

**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:** Constant-approximation, for a metric-dependent constant.
- **In practice:** Fast query-response times, optimal solutions in most cases.

# CFCA(N) : A new query algorithm

**procedure** CFCA(N)

**STEP 1:** A TDD ball is grown from $(o, t_o)$, until $N$ landmarks are settled.

**1.1:** **if** $d$ is already settled **then return** optimal solution.

**1.2:** For each settled landmark $\ell$, $t_\ell = t_o + D[o,\ell](t_o)$.

**STEP 2:** An appropriate subgraph is recursively created from $d$.

**2.1:** $Q = \{ d \}$ /* $Q$ is a FIFO queue */

**2.2:** **while** $\neg Q.Empty()$ **do** :

**2.3:** **if** $v = Q.Pop()$ is not explored from STEP 1's TDD ball **then** :

**2.4:** **for** each settled landmark $\ell$ of STEP 1 **do** :

**2.5:** Mark the arcs $\langle PRED[\ell, v](t_\ell^-), v\rangle$ and $\langle PRED[\ell, v](t_\ell^+), v\rangle$ leading to $v$, where $[t_\ell^-, t_\ell^+)$ is the unique interval in $DEP[\ell, v]$ containing $t_\ell$.

**2.6:** **if** any of the predecessors was not yet visited **then** { $Q.Push(PRED[\ell, v](t_\ell^-))$; $Q.Push(PRED[\ell, v](t_\ell^+))$ }

**2.7:** **endfor**

**2.8:** **endwhile**

**STEP 3:** **return** optimal od-path in the induced subgraph by the TDD ball of STEP 1, and the marked arcs of STEP 2.

# Significance of Path Construction

- For time-independent instances, path construction is quite easy and has essentially **negligible contribution** to the query-time.

# Significance of Path Construction

- For time-independent instances, path construction is quite easy and has essentially **negligible contribution** to the query-time.

- For time-dependent instances, path construction is **not so easy** anymore, and usually contributes a **significant** amount to the query-time:
  - During the (backward) path construction from the destination, one has to deal with evaluations of continuous functions rather than just scalars.
  - For the instance of Germany (see experiments), for both `CFLAT` and `KaTCH`, the path-construction cost contributes more than 30% of the total query-time.

# Significance of Path Construction

- For time-independent instances, path construction is quite easy and has essentially **negligible contribution** to the query-time.

- For time-dependent instances, path construction is **not so easy** anymore, and usually contributes a **significant** amount to the query-time:
  - During the (backward) path construction from the destination, one has to deal with evaluations of continuous functions rather than just scalars.
  - For the instance of Germany (see experiments), for both `CFLAT` and `KaTCH`, the path-construction cost contributes more than 30% of the total query-time.

- Steps 2 and 3 leave more room for (future) algorithm engineering.

# Experimental Evaluation

**...setup, instances, evaluation & comparison...**

# Instances

| Instance | #nodes | #edges |
|---|---|---|
| BER | 473,253 | 1,126,468 |
| GER | 4,692,091 | 10,805,429 |
| EUR | 18,010,173 | 42,188,664 |
| GRID | 5,400,976 | 11,045,894 |

- Real-world Berlin instance (BER) -- kindly provided by TomTom.
- Real-world Germany instance (GER) -- kindly provided by PTV AG.
- Synthetic Europe instance (EUR) -- typical benchmark network of DIMACS challenge.
- Synthetic grid instance (GRID) -- constructed in this work.

# Experiment 1: Preprocessing times of TD-Oracles

- Significant improvement by exploiting a time-dependent variant of the **Delta-Stepping** algorithm (instead of TDD) as an SPT sampling algorithm.
- Exploitation of a careful combination of data-parallelism and algorithmic parallelism.
- Exploitation of the **amorphous-data-parallelism** rationale, which also boosted the preprocessing phase.

# Experiment 1: Preprocessing times of TD-Oracles

- Significant improvement by exploiting a time-dependent variant of the **Delta-Stepping** algorithm (instead of TDD) as an SPT sampling algorithm.
- Exploitation of a careful combination of data-parallelism and algorithmic parallelism.
- Exploitation of the **amorphous-data-parallelism** rationale, which also boosted the preprocessing phase.

| DIJ vs DS @ OFLAT oracle | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| INSTANCE (# landmarks) | BERLIN (1000 landmarks) | | | GERMANY (1000 landmarks) | | | EUROPE (900 landmarks) | | |
| method | OTRAP & 1xDIJsh(24) | OTRAP & 8xDS(3) | speedup | OTRAP & 1xDIJsh(12) | OTRAP & 12xDS(2) | speedup | OTRAP & 1xDIJsh(12) | OTRAP & 4xDS(6) | speedup |
| total time (min) | 5.964 | 7.701 | 0.774 | 123.454 | 89.216 | 1.384 | 5293.333 | 3549.745 | 1.491 |

# Experiment 2: Query Response Times

| QUERY RESPONSE TIMES (msec) | | CFLAT QUERY : : CFCA(N) | OFLAT QUERY : : OFCA(N) | | KaTCH (msec) |
|---|---|---|---|---|---|
| GER | | 4K SR Landmarks | 5K SR Landmarks | | 0.820 |
| | SPT alg | 1xDIJbh(1) | 1xDS(1) [Δ=32] | 1xDIJsh(1) | |
| | N=1 | 0.582 | 0.692 | **0.4972** | |
| | N=2 | 1.242 | 1.087 | 0.9612 | |
| | N=4 | 2.413 | 1.926 | 1.8768 | |
| | N=6 | 3.572 | 2.904 | 2.7515 | |
| EUR | | -- | 700 SR Landmarks | | **1.560** |
| | SPT alg | -- | 1xDS(12) [Δ=128] | 1xDIJsh(1) | |
| | N=1 | -- | 3.332 | 7.998 | |
| | N=2 | -- | 4.678 | 15.463 | |
| | N=4 | -- | 7.688 | 30.008 | |
| | N=6 | -- | 10.444 | 44.652 | |

# Thank You For Your Attention

**Questions**

**algorithms**

an open access journal by MDPI

Covered by EMERGING SOURCES CITATION INDEX

Indexed in: Ei Compendex

**Journal's Aims and Scope:**

*Algorithms* (ISSN 1999-4893) is an international journal, which provides an advanced forum for studies related to algorithms and their applications.

The scope of *Algorithms* includes:

- Theory of algorithms
- Combinatorial optimization and operations research, with applications
- Special data structures
- Distributed and parallel algorithms
- Metaheuristics
- Performance of algorithms and algorithm engineering
- Applications in other areas of computer science
- Algorithms in biology, chemistry, physics, language
- processing, *etc.*
- Image processing with applications
- Machine learning, including grammatical inference
- Educational aspects: how to teach algorithms

**Timescale (in 2018)**

Downloads: 101,907
Page views: 654,126
Published papers: 223

**Journal website:**
mdpi.com/journal/algorithms
**E-Mail:**
algorithms@mdpi.com
**Twitter:** @Algorithms_MDPI

Open Access
Indexed by **Scopus, Ei, and ESCI**

First Decision 19.6 days

**Prof. Dr. Henning Fernau**
Editor-in-Chief

No Space Constraints

MDPI OPEN ACCESS