Algorithms for Route Planning

KIT/ITI, Lectures 14-15: Introduction to TDSP & TD-Oracles

Spyros Kontogiannis

Assistant Professor at Department of Computer Science & Engineering

Guest Professor at Institute of Theoretical Informatics (KIT/ITI)

June 11-18, 2014

Time Dependent Shortest Path

Why Time-Dependent Shortest Paths?

Real-life networks: Elements demostrate temporal behavior.

Why Time-Dependent Shortest Paths?

Real-life networks: Elements demostrate temporal behavior.

- Graph elements **added/removed** in real-time. /* Dynamic Shortest Path */
- Metric demonstrates **stochastic behavior**. /* Sthochastic Shortest Path */
- Graph is **fixed**, metric **changes with the value of a parameter** $\gamma \in [0, 1]$ in a predetermined fashion. /* Parametric Shortest Path */
- Graph is **fixed**, metric **changes over time** in a predetermined fashion. /* Time-Dependent Shortest Path */

Why Time-Dependent Shortest Paths?

Real-life networks: Elements demostrate temporal behavior.

- Graph is **fixed**, metric **changes over time** in a predetermined fashion. /* Time-Dependent Shortest Path */

Why Time-Dependent Shortest Paths?

Real-life networks: Elements demostrate temporal behavior.

- Graph is **fixed**, metric **changes over time** in a predetermined fashion. /* Time-Dependent Shortest Path */
  - Arcs are allowed to become **occasionally unavailable** (e.g., due to periodic maintenance, saving consumption of resources, etc), for predetermined unavailability time-intervals (discrete domain).
  - Arc lengths (e.g., traversal-time / consumption) **change with departure-time from tail** which is treated as a real-valued variable (functions with continuous domain, but not necessarily continuous range).

Why Time-Dependent Shortest Paths?

Real-life networks: Elements demostrate temporal behavior.

- Graph is fixed, metric changes with the value of a parameter $\gamma \in [0, 1]$ in a predetermined fashion. /* Parametric Shortest Path */
- Graph is fixed, metric changes over time in a predetermined fashion. /* Time-Dependent Shortest Path */

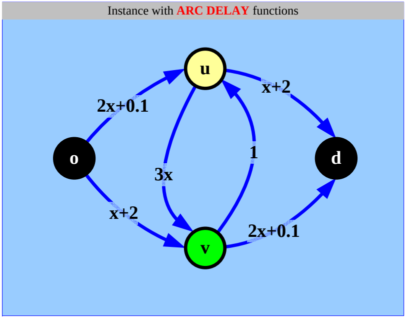
    - Arc lengths (e.g., traversal-time / consumption) change with departure-time from tail which is treated as a real-valued variable (functions with continuous domain, but not necessarily continuous range).

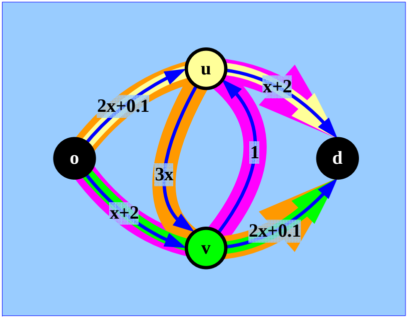
Why Time-Dependent Shortest Paths?

Real-life networks: Elements demostrate temporal behavior.

- Graph is **fixed**, metric **changes over time** in a predetermined fashion. /* Time-Dependent Shortest Path */
  - Arc lengths (e.g., traversal-time / consumption) **change with departure-time from tail** which is treated as a real-valued variable (functions with continuous domain, but not necessarily continuous range).

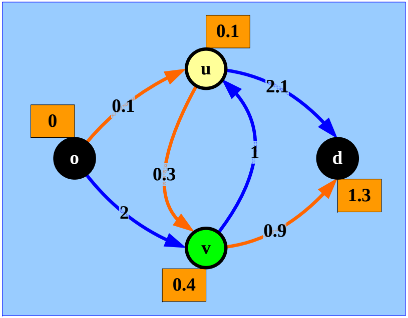
TDSP :: EXAMPLE 1 (Earliest Arrivals)

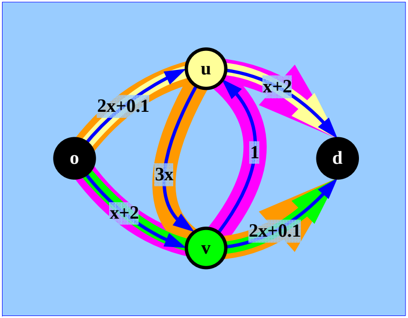




Q1 How would you commute **as fast as possible** from *o* to *d*, for a given departure time (from *o*)?

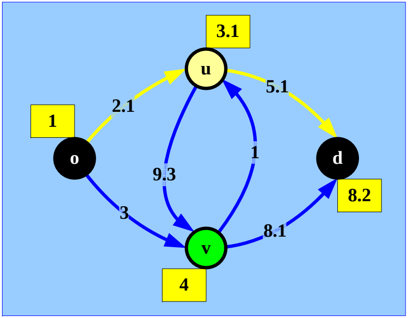
TDSP :: EXAMPLE 1 (Earliest Arrivals)

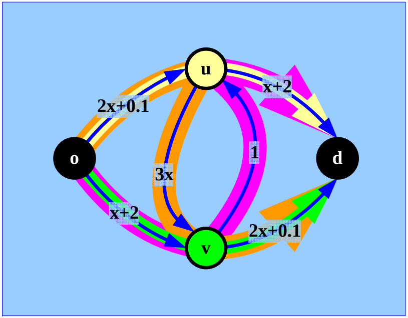




Q1 How would you commute **as fast as possible** from o to d, for a given departure time (from o)? Eg: $\boxed{t_o = 0}$

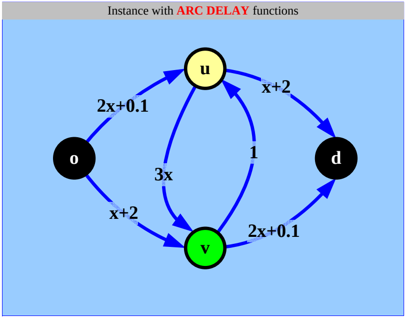
TDSP :: EXAMPLE 1 (Earliest Arrivals)

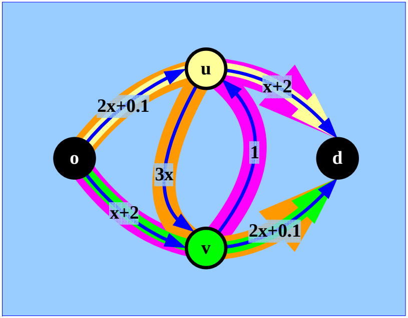




Q1 How would you commute **as fast as possible** from o to d, for a given departure time (from o)? Eg: $\boxed{t_o = 1}$

TDSP :: EXAMPLE 1 (Earliest Arrivals)

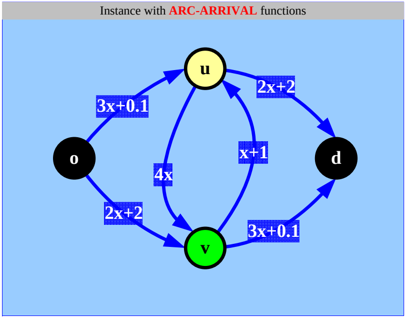


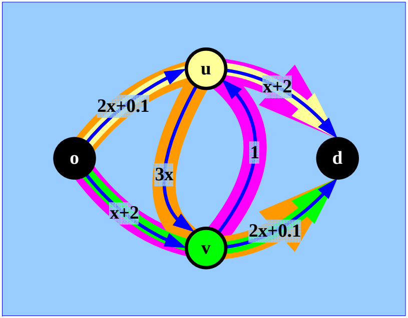


Q1 How would you commute **as fast as possible** from o to d, for a given departure time (from o)?

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

TDSP :: EXAMPLE 1 (Earliest Arrivals)

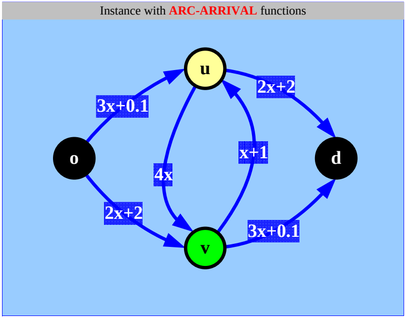


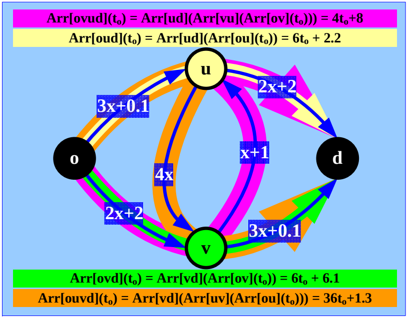


Q1 How would you commute **as fast as possible** from *o* to *d*, for a given departure time (from *o*)?

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

TDSP :: EXAMPLE 1 (Earliest Arrivals)

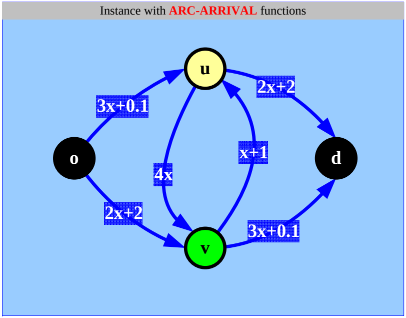


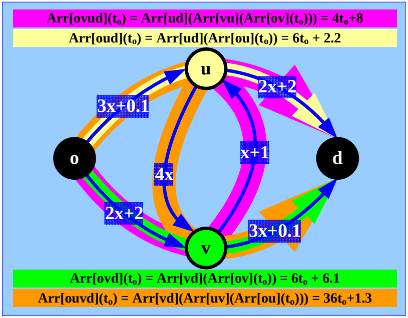


Q1 How would you commute as fast as possible from o to d, for a given departure time (from o)?

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

TDSP :: EXAMPLE 1 (Earliest Arrivals)





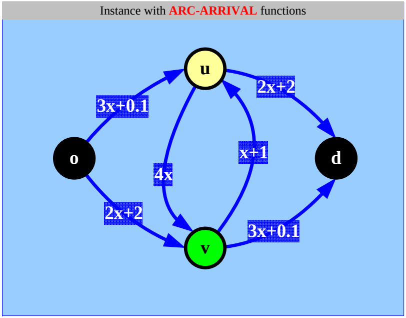
Q1 How would you commute **as fast as possible** from o to d, for a given departure time (from o)?

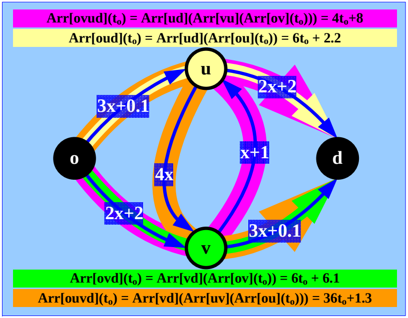
Q2 What if you are **not sure** about the departure time?

A

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

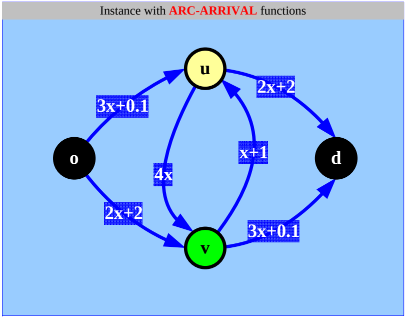
TDSP :: EXAMPLE 2 (Waiting Times)

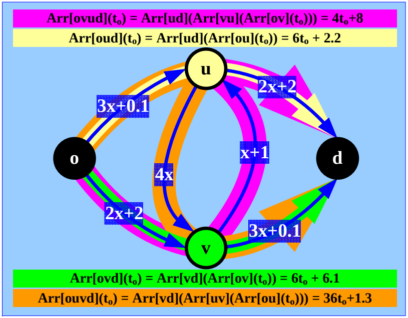




Q1 Would **waiting-at-nodes** be worth it?

TDSP :: EXAMPLE 2 (Waiting Times)

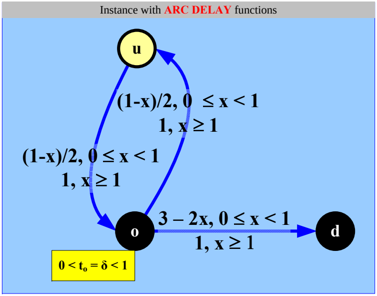




Q1 Would **waiting-at-nodes** be worth it?

A1 NO, since arrival-time functions are *non-decreasing* functions of departure-time from origin.

TDSP :: EXAMPLE 2 (Waiting Times)

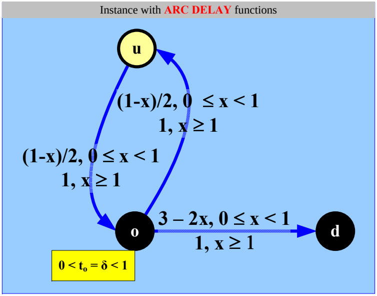


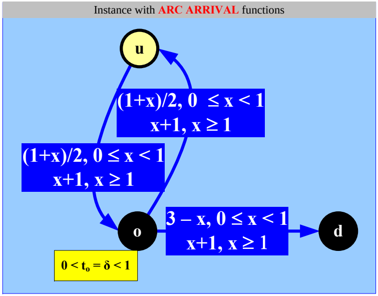
Q1 Would **waiting-at-nodes** be worth it?

A1 NO, since arrival-time functions are *non-decreasing* functions of departure-time from origin.

Q2 Would **waiting-at-nodes** be worth it in this case?

TDSP :: EXAMPLE 2 (Waiting Times)





Q1 Would **waiting-at-nodes** be worth it?

A1 NO, since arrival-time functions are *non-decreasing* functions of departure-time from origin.

Q2 Would **waiting-at-nodes** be worth it in this case?

A2 YES, wait until time 1 and then traverse *od*, if already present at o at time $t_o < 1$. Otherwise, traverse *od* immediately.

Waiting Policies

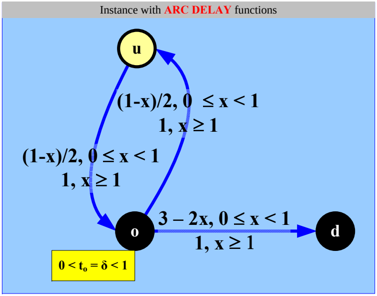
Unrestricted Waiting (UW) Unlimited waiting is allowed at every node along an *od*-path.

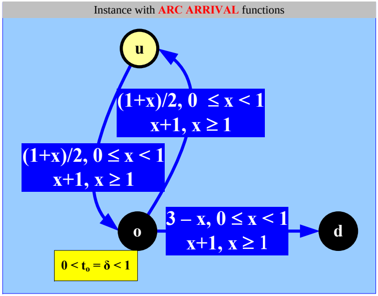
Origin Waiting (OW) Unlimited waiting is only allowed at the origin node of each *od*-path.

Forbidden Waiting (FW) No waiting is allowed at any node of each *od*-path.

> Depending on the *waiting policy*, the scheduler has to decide not only for an optimal connecting path (that assures the earliest arrival at the destination), but also for the appropriate optimal waiting times at the nodes along this path.

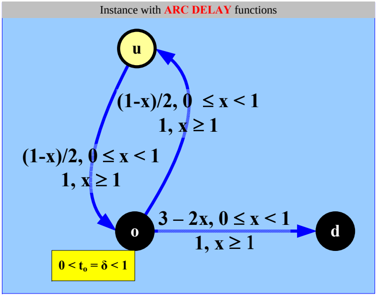
TDSP :: EXAMPLE 2 (Waiting Times) – contd.

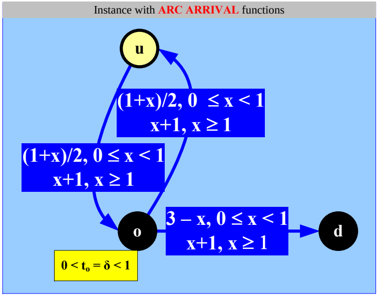




Q3 What if waiting-at-nodes is forbidden?

TDSP :: EXAMPLE 2 (Waiting Times) – contd.



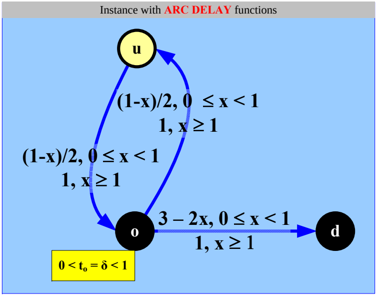


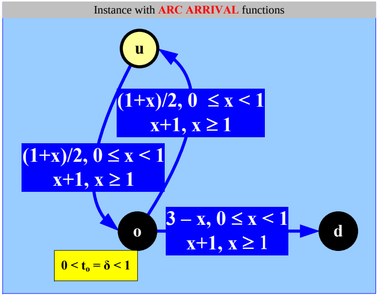
Q3 What if waiting-at-nodes is forbidden?

A3 An infinite, non-simple TD shortest *od*-path with finite delay.

o					d
δ					$3-\delta$

TDSP :: EXAMPLE 2 (Waiting Times) – contd.



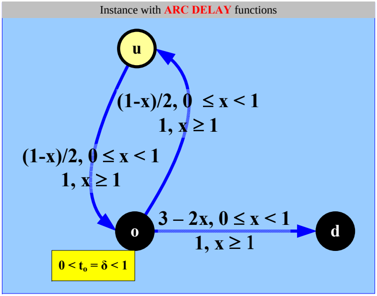


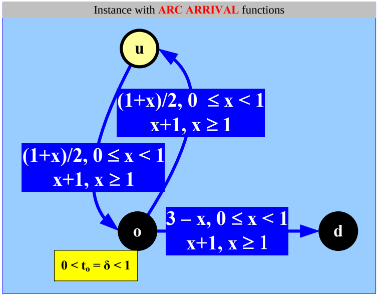
Q3 What if **waiting-at-nodes** is forbidden?

A3 An infinite, non-simple TD shortest *od*-path with finite delay.

o	u	o			d
δ	$\frac{1+\delta}{2}$	$\frac{1}{2} + \frac{1+\delta}{4}$			$3 - \frac{1}{2} - \frac{1+\delta}{4} > 2$

TDSP :: EXAMPLE 2 (Waiting Times) – contd.

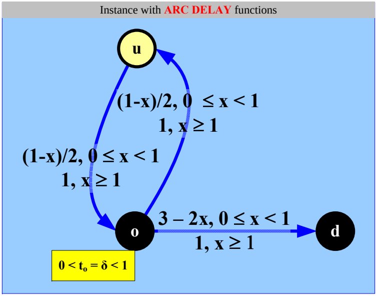


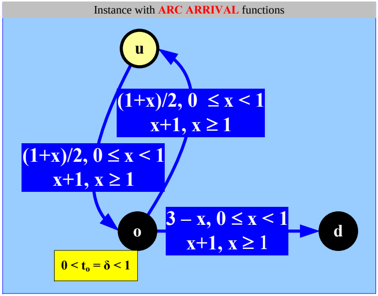


Q3 What if waiting-at-nodes is forbidden?

A3 An infinite, non-simple TD shortest *od*-path with finite delay.

o	u	o	u	o	d
δ	$\frac{1+\delta}{2}$	$\frac{1}{2}+\frac{1+\delta}{4}$	$\frac{1}{2}+\frac{1}{4}+\frac{1+\delta}{8}$	$\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\frac{1+\delta}{16}$	$3-\frac{1}{2}-\frac{1}{4}-\frac{1}{8}-\frac{1+\delta}{16}>2$





Q3 What if **waiting-at-nodes** is forbidden?

A3 An infinite, non-simple TD shortest *od*-path with finite delay.

o	u	o	$<u,o,\ldots,o,u,o>$	d
δ	$\frac{1+\delta}{2}$	$\frac{1}{2}+\frac{1+\delta}{4}$	$\uparrow \sum_{k=1}^{\infty}\left(\frac{1}{2}\right)^k = 1$	$t_d \downarrow 2$

Subpath optimality and *shortest path simplicity* **not guaranteed** for TDSP, if waiting-at-nodes is forbidden.

Inexistence of Optimal Waiting Times

- Do optimal waiting times at nodes always exist?

Inexistence of Optimal Waiting Times

- Do optimal waiting times at nodes always exist?
- Unfortunately NOT! EXAMPLE:

$$
\begin{aligned}
D[od](t) &= \begin{cases} 100, & t \leq 10, \\ 1, & t > 10 \end{cases} \\
\Rightarrow Arr[od](t) &= \begin{cases} t + 100, & t \leq 10, \\ t + 1, & t > 10 \end{cases}
\end{aligned}
$$

Inexistence of Optimal Waiting Times

- Do optimal waiting times at nodes always exist?
- Unfortunately NOT! EXAMPLE:

$$D[od](t) = \begin{cases} 100, & t \leq 10, \\ 1, & t > 10 \end{cases}$$

$$\Rightarrow Arr[od](t) = \begin{cases} t + 100, & t \leq 10, \\ t + 1, & t > 10 \end{cases}$$

- But this is due to the pathological discontinuity of the delay / arrival-time function.

Inexistence of Optimal Waiting Times

- Do optimal waiting times at nodes always exist?
- Unfortunately NOT! EXAMPLE:

$$D[od](t) = \begin{cases} 100, & t \leq 10, \\ 1, & t > 10 \end{cases}$$

$$\Rightarrow Arr[od](t) = \begin{cases} t+100, & t \leq 10, \\ t+1, & t > 10 \end{cases}$$

- But this is due to the pathological discontinuity of the delay / arrival-time function.
- They always exist for **continuous** delay functions, as well as for (possibly discontinuous) **pwl** functions for which:

$$\lim_{t \downarrow t_u} D[uv](t) < \lim_{t \uparrow t_u} D[uv](t) \Rightarrow D[uv](t_u) = \lim_{t \downarrow t_u} D[uv](t)$$

Inexistence of Optimal Waiting Times

- Do optimal waiting times at nodes always exist?
- Unfortunately NOT! EXAMPLE:

$$D[od](t) = \begin{cases} 100, & t \leq 10, \\ 1, & t > 10 \end{cases}$$

$$\Rightarrow Arr[od](t) = \begin{cases} t + 100, & t \leq 10, \\ t + 1, & t > 10 \end{cases}$$

- But this is due to the pathological discontinuity of the delay / arrival-time function.
- They always exist for **continuous** delay functions, as well as for (possibly discontinuous) **pwl** functions for which:

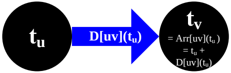
$$\lim_{t \downarrow t_u} D[uv](t) < \lim_{t \uparrow t_u} D[uv](t) \Rightarrow D[uv](t_u) = \lim_{t \downarrow t_u} D[uv](t)$$

> From now on we assume that optimal waiting times at nodes exist (and are polynomial-time computable).

FIFO vs non-FIFO Arc Delays

- **(Strict) FIFO Arc-Delays:** The slopes of all the *arc-delay* functions are at least equal to (greater than) -1.

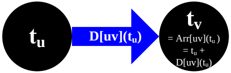
Equivalently: *Arc-arrival* functions are **non-decreasing** (aka **no-overtaking** property).



FIFO vs non-FIFO Arc Delays

- **(Strict) FIFO Arc-Delays:** The slopes of all the *arc-delay* functions are at least equal to (greater than) -1.

Equivalently: *Arc-arrival* functions are **non-decreasing** (aka **no-overtaking** property).

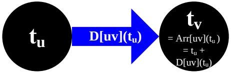


- **Non-FIFO Arc-Delays:** Possibly preferrable to wait for some period at the tail of an arc, before trespassing it. E.g.:
  - Wait for the next *(faster) IC train*, than use the (immediately available) *(slower) local train*.

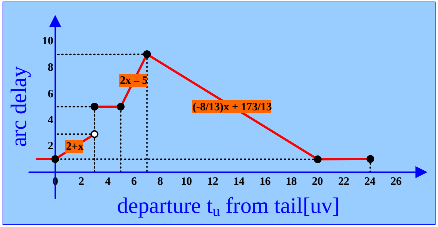
FIFO vs non-FIFO Arc Delays

- **(Strict) FIFO Arc-Delays:** The slopes of all the *arc-delay* functions are at least equal to (greater than) -1.

Equivalently: *Arc-arrival* functions are **non-decreasing** (aka **no-overtaking** property).



- **Non-FIFO Arc-Delays:** Possibly preferrable to wait for some period at the tail of an arc, before trespassing it. E.g.:
  - Wait for the next *(faster) IC train*, than use the (immediately available) *(slower) local train*.

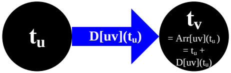


FIFO arc delay example

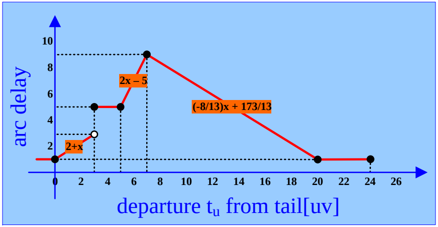
FIFO vs non-FIFO Arc Delays

- **(Strict) FIFO Arc-Delays:** The slopes of all the *arc-delay* functions are at least equal to (greater than) -1.

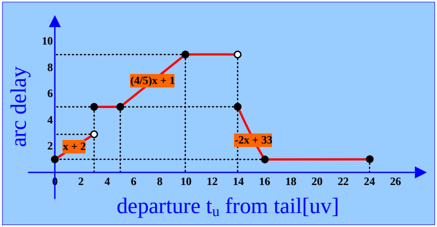
Equivalently: *Arc-arrival* functions are **non-decreasing** (aka **no-overtaking** property).



- **Non-FIFO Arc-Delays:** Possibly preferrable to wait for some period at the tail of an arc, before trespassing it. E.g.:
  - Wait for the next *(faster) IC train*, than use the (immediately available) *(slower) local train*.

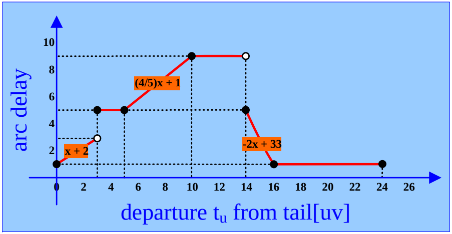


FIFO arc delay example

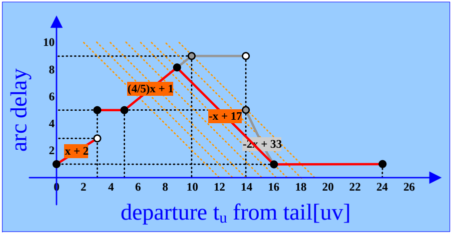


Non-FIFO arc delay example

Non-FIFO+UW Network ⇔ FIFO Network

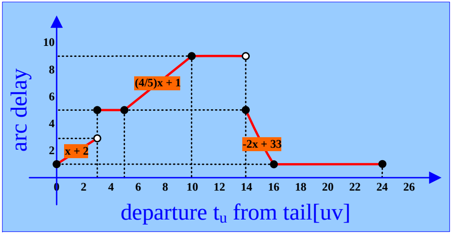


Non-FIFO+UW arc delay function

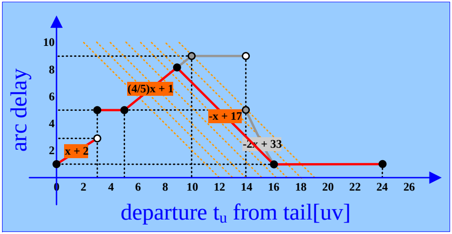


Equivalent FIFO (+FW) arc delay function

Non-FIFO+UW Network $\Leftrightarrow$ FIFO Network



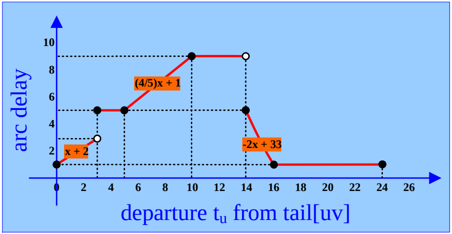
Non-FIFO+UW arc delay function



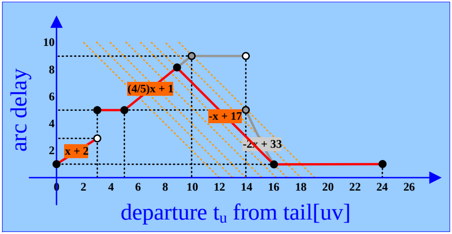
Equivalent FIFO (+FW) arc delay function

- A "scan" of the line with slope -1 *from right to left* suffices.

Non-FIFO+UW Network $\Leftrightarrow$ FIFO Network



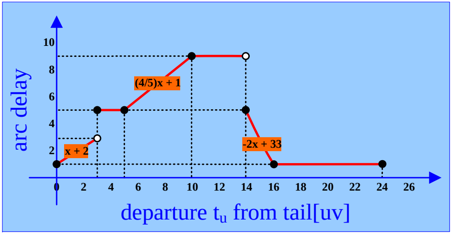
Non-FIFO+UW arc delay function



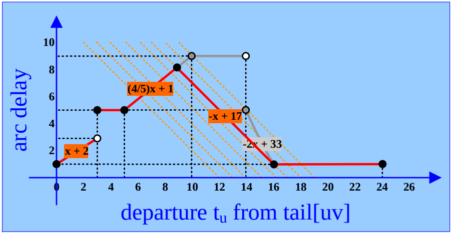
Equivalent FIFO (+FW) arc delay function

- A "scan" of the line with slope -1 *from right to left* suffices.
  - **Shortcircuit** pieces of the arc-delay function lying above the line of slope -1.

Non-FIFO+UW Network $\Leftrightarrow$ FIFO Network



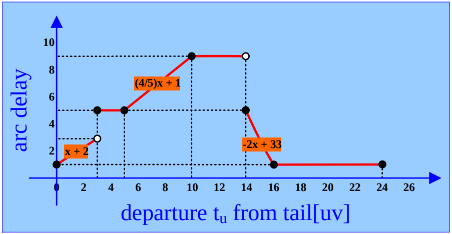
Non-FIFO+UW arc delay function



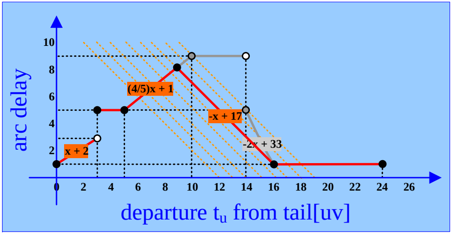
Equivalent FIFO (+FW) arc delay function

- A "scan" of the line with slope -1 *from right to left* suffices.
  - **Shortcircuit** pieces of the arc-delay function lying above the line of slope -1.
- *Identical arrival-times* in **Non-FIFO+UW** and **FIFO** instances.

Non-FIFO+UW Network $\Leftrightarrow$ FIFO Network



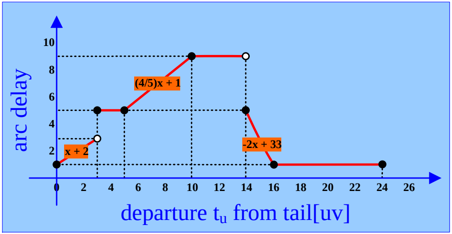
Non-FIFO+UW arc delay function



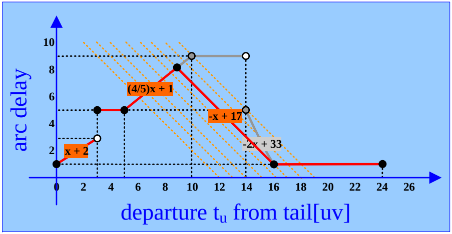
Equivalent FIFO (+FW) arc delay function

- A "scan" of the line with slope -1 *from right to left* suffices.
  - **Shortcircuit** pieces of the arc-delay function lying above the line of slope -1.
- *Identical arrival-times* in **Non-FIFO+UW** and **FIFO** instances.
- Need to consider *latest departures* given the arrival times, in order to compute the optimal waiting times in the original **Non-FIFO+UW** instance.

Non-FIFO+UW Network $\Leftrightarrow$ FIFO Network



Non-FIFO+UW arc delay function



Equivalent FIFO (+FW) arc delay function

- A "scan" of the line with slope -1 *from right to left* suffices.
  - **Shortcircuit** pieces of the arc-delay function lying above the line of slope -1.
- *Identical arrival-times* in **Non-FIFO+UW** and **FIFO** instances.
- Need to consider *latest departures* given the arrival times, in order to compute the optimal waiting times in the original **Non-FIFO+UW** instance.

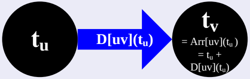
Interested in programming the transformation? Let me know!

Variants of Time-Dependent Shortest Path

DEFINITION: Time-Dependent Shortest Paths

INPUT:

- *Directed* graph $G = (V, A)$ with **succinctly represented** **arc-travel-time** functions $(D[\alpha])_{\alpha\in A}$. $(Arr[\alpha] = ID + D[\alpha])_{\alpha\in A}$.

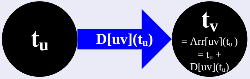


Variants of Time-Dependent Shortest Path

DEFINITION: Time-Dependent Shortest Paths

INPUT:

- *Directed* graph $G = (V, A)$ with **succinctly represented arc-travel-time** functions $(D[\alpha])_{\alpha \in A}$. $(Arr[\alpha] = ID + D[\alpha])_{\alpha \in A}$.



DEFINITIONS:

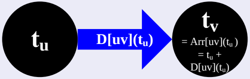
- **Path arrival / travel-time** functions: $\forall p = (\alpha_1, \ldots, \alpha_k) \in P_{o,d}$, $Arr[p] = Arr[\alpha_k] \circ \cdots \circ Arr[\alpha_1]$ (*composition* of the involved arc-arrivals). $D[p] = Arr[p] - ID$.
- **Earliest-arrival / Shortest-travel-time** functions: $Arr[o,d] = \min_{p \in P_{o,d}} \{ \; Arr[p] \; \}$, $D[o,d] = Arr[o,d] - ID$.

Variants of Time-Dependent Shortest Path

DEFINITION: Time-Dependent Shortest Paths

INPUT:

- *Directed* graph $G = (V, A)$ with **succinctly represented arc-travel-time** functions $(D[\alpha])_{\alpha\in A}$. $(Arr[\alpha] = ID + D[\alpha])_{\alpha\in A}$.



DEFINITIONS:

- **Path arrival / travel-time** functions: $\forall p = (\alpha_1, \ldots, \alpha_k) \in P_{o,d}$, $Arr[p] = Arr[\alpha_k] \circ \cdots \circ Arr[\alpha_1]$ (*composition* of the involved arc-arrivals). $D[p] = Arr[p] - ID$.
- **Earliest-arrival / Shortest-travel-time** functions: $Arr[o,d] = \min_{p\in P_{o,d}} \{ Arr[p] \}$, $D[o,d] = Arr[o,d] - ID$.

GOAL1: For departure-time t_o from o, determine $t_d = Arr[o,d](t_o)$.

GOAL2: Provide a **succinct representation** of $Arr[o,d]$ (or $D[o,d]$).

Why Care for Both Goals?

1. Not always sure when to depart (still think about it)! Possessing the entire *distance function* $D[o,d]$ allows for easy answers (e.g., via look-ups) in several queries for varying departure times, or even finding the *minimum travel / ealriest-arrival time* within a window of possible departure times.

Why Care for Both Goals?

1. Not always sure when to depart (still think about it)! Possessing the entire *distance function* $D[o,d]$ allows for easy answers (e.g., via look-ups) in several queries for varying departure times, or even finding the *minimum travel / ealriest-arrival time* within a window of possible departure times.

2. Need to respond **efficiently** (in theory: sublinear time, in practice: micro/miliseconds) to arbitrary queries in *large-scale nets*, for any departure time and $od-$pair.

Why Care for Both Goals?

1. Not always sure when to depart (still think about it)! Possessing the entire *distance function* $D[o,d]$ allows for easy answers (e.g., via look-ups) in several queries for varying departure times, or even finding the *minimum travel / ealriest-arrival time* within a window of possible departure times.

2. Need to respond **efficiently** (in theory: sublinear time, in practice: micro/miliseconds) to arbitrary queries in *large-scale nets*, for any departure time and $od-$pair.

- **Preprocess** (offline) towards GOAL2 (succinct representations of *selected* $D[o,d]$ functions) in order to support *real-time* responses to **queries** of GOAL1.

Why Care for Both Goals?

1. Not always sure when to depart (still think about it)! Possessing the entire *distance function* $D[o,d]$ allows for easy answers (e.g., via look-ups) in several queries for varying departure times, or even finding the *minimum travel / ealriest-arrival time* within a window of possible departure times.
2. Need to respond **efficiently** (in theory: sublinear time, in practice: micro/miliseconds) to arbitrary queries in *large-scale nets*, for any departure time and $od-$pair.

- **Preprocess** (offline) towards GOAL2 (succinct representations of *selected* $D[o,d]$ functions) in order to support *real-time* responses to **queries** of GOAL1.
- **Preprocessing** of distance summaries (as in static case) requires to precompute functions instead of scalars.

Consequencies of Different Network Models

- [Dreyfus (1969)] Prefix-subpath optimality holds in **Non-FIFO+UW** networks (given that optimal waiting times *exist*). The same applies for the **FIFO** networks.

[1] This means that: $\forall t_u, \lim_{t\uparrow t_u} D[uv](t) \geq \lim_{t\downarrow t_u} D[uv](t))$

Consequencies of Different Network Models

- [Dreyfus (1969)] Prefix-subpath optimality holds in **Non-FIFO+UW** networks (given that optimal waiting times *exist*). The same applies for the **FIFO** networks.
- [Orda-Rom (1990)] Prefix-subpath optimality does NOT hold in **non-FIFO+FW** networks (cf. EXAMPLE of Slide 7).

[1] This means that: $\forall t_u, \lim_{t \uparrow t_u} D[uv](t) \geq \lim_{t \downarrow t_u} D[uv](t)$)

Consequencies of Different Network Models

- [Dreyfus (1969)] Prefix-subpath optimality holds in **Non-FIFO+UW** networks (given that optimal waiting times *exist*). The same applies for the **FIFO** networks.
- [Orda-Rom (1990)] Prefix-subpath optimality does NOT hold in **non-FIFO+FW** networks (cf. EXAMPLE of Slide 7).
- [Orda-Rom (1990)] If arc-delay functions are *continuous*, or *piecewise continuous with negative discontinuities*[1], then the solution (path+waiting policy) in non-FIFO+UW network induces a solution in **non-FIFO+OW** network using the **same path** and appropriate waiting time **only at the origin**.

[1] This means that: $\forall t_u, \lim_{t \uparrow t_u} D[uv](t) \geq \lim_{t \downarrow t_u} D[uv](t)$)

Consequencies of Different Network Models

- [Dreyfus (1969)] Prefix-subpath optimality holds in **Non-FIFO+UW** networks (given that optimal waiting times *exist*). The same applies for the **FIFO** networks.
- [Orda-Rom (1990)] Prefix-subpath optimality does NOT hold in **non-FIFO+FW** networks (cf. EXAMPLE of Slide 7).
- [Orda-Rom (1990)] If arc-delay functions are *continuous*, or *piecewise continuous with negative discontinuities*[1], then the solution (path+waiting policy) in non-FIFO+UW network induces a solution in **non-FIFO+OW** network using the **same path** and appropriate waiting time **only at the origin**.
- [Kontogiannis-Zaroliagis (2013)] In **strict-FIFO** networks, (general) subpath optimality holds also in the time-dependent case.

[1] This means that: $\forall t_u, \lim_{t \uparrow t_u} D[uv](t) \geq \lim_{t \downarrow t_u} D[uv](t))$

Consequencies of Different Network Models

- [Dreyfus (1969)] Prefix-subpath optimality holds in **Non-FIFO+UW** networks (given that optimal waiting times *exist*). The same applies for the **FIFO** networks.
- [Orda-Rom (1990)] Prefix-subpath optimality does NOT hold in **non-FIFO+FW** networks (cf. EXAMPLE of Slide 7).
- [Orda-Rom (1990)] If arc-delay functions are *continuous*, or *piecewise continuous with negative discontinuities*[1], then the solution (path+waiting policy) in non-FIFO+UW network induces a solution in **non-FIFO+OW** network using the **same path** and appropriate waiting time **only at the origin**.
- [Kontogiannis-Zaroliagis (2013)] In **strict-FIFO** networks, (general) subpath optimality holds also in the time-dependent case.
- [Foschini-Hershberger-Suri (2011)] In **(strict) FIFO** networks, $Arr[o,d]$ is non-decreasing (increasing).

[1] This means that: $\forall t_u, \lim_{t\uparrow t_u} D[uv](t) \geq \lim_{t\downarrow t_u} D[uv](t))$

Algorithms for TDSP

- For arbitrary (o, d, t_o) queries (**GOAL1**):

Algorithms for TDSP

- For arbitrary (o, d, t_o) queries (**GOAL1**):
  - TD variants of `Dijkstra` and `Bellman-Ford` algorithms work correctly in **FIFO** networks, and in **non-FIFO+UW** networks. Time complexity slightly worse (when updating arc labels, some arc-delay *functions* are evaluated).
  - TD variants of `Dijkstra` and `Bellman-Ford` algorithms do NOT work correctly in **non-FIFO+FW** networks. Time complexity slightly worse (when updating arc labels, some arc-delay *functions* are evaluated). Determining existence of a finite-hop solution is $\mathcal{NP}-$hard.

Algorithms for TDSP

- For arbitrary (o, d, t_o) queries (**GOAL1**):
  - TD variants of `Dijkstra` and `Bellman-Ford` algorithms work correctly in **FIFO** networks, and in **non-FIFO+UW** networks. Time complexity slightly worse (when updating arc labels, some arc-delay *functions* are evaluated).
  - TD variants of `Dijkstra` and `Bellman-Ford` algorithms do NOT work correctly in **non-FIFO+FW** networks. Time complexity slightly worse (when updating arc labels, some arc-delay *functions* are evaluated). Determining existence of a finite-hop solution is $\mathcal{NP}-$hard.
- For arbitrary (o, d) queries (**GOAL2**):
  - [Orda-Rom (1990)] Propose a TD-variant of `Bellman-Ford`, for **non-FIFO+UW** networks.

Algorithms for TDSP

- For arbitrary (o, d, t_o) queries (**GOAL1**):
  - TD variants of `Dijkstra` and `Bellman-Ford` algorithms work correctly in **FIFO** networks, and in **non-FIFO+UW** networks. Time complexity slightly worse (when updating arc labels, some arc-delay *functions* are evaluated).
  - TD variants of `Dijkstra` and `Bellman-Ford` algorithms do NOT work correctly in **non-FIFO+FW** networks. Time complexity slightly worse (when updating arc labels, some arc-delay *functions* are evaluated). Determining existence of a finite-hop solution is $\mathcal{NP}-$hard.

- For arbitrary (o, d) queries (**GOAL2**):
  - [Orda-Rom (1990)] Propose a TD-variant of `Bellman-Ford`, for **non-FIFO+UW** networks.
    - ☹ Complexity is **polynomial** on number of "elementary" *functional operations*. (EVAL, LINEAR COMBINATION, MIN, COMPOSITION)
    - ☹ Not so "elementary" operations after all (see next slides)!!!

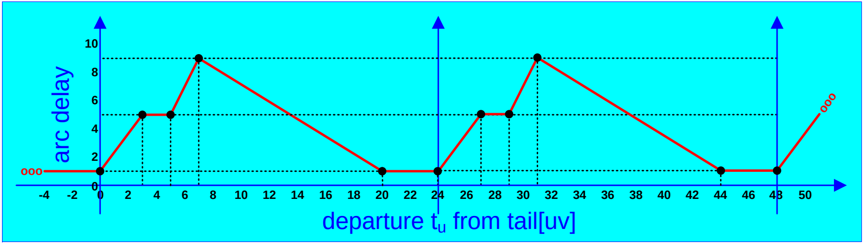
Algorithms for TDSP

in FIFO, Continuous, Pwl Instances

Input/Output Data

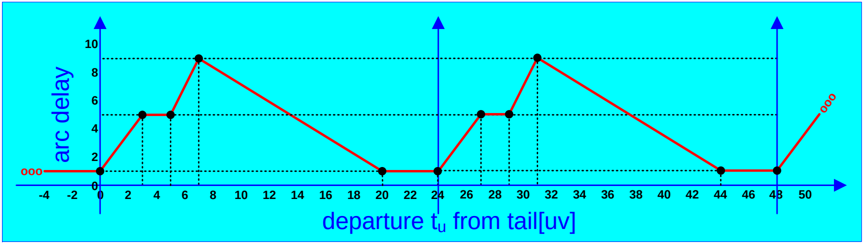
PWL Arc Delays

Forward Description (as function of departure times from origin)

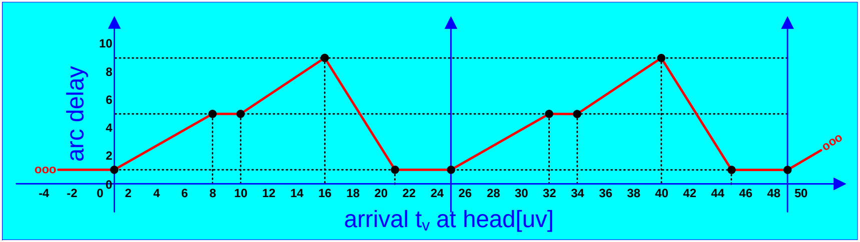


PWL Arc Delays

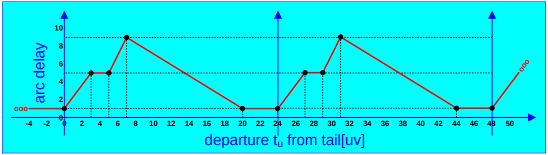
Forward Description (as function of departure times from origin)



Reverse Description (as function of arrival times at destination)



How to Store/Access PWL Arc Delays



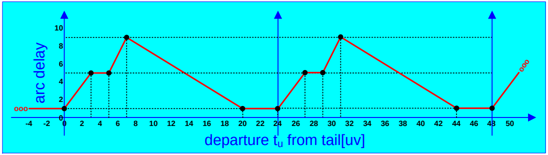
- Exploit *periodicity* and *piecewise-linearity*:

$$\forall t_u \in \mathbb{R},\quad \overrightarrow{D}[uv](t_u) = \begin{cases} \frac{4}{3}t_u + 1, & 0 \leq t_u \mod T \leq 3 \\ 5, & 3 \leq t_u \mod T \leq 5 \\ 2t_u - 5, & 5 \leq t_u \mod T \leq 7 \\ -\frac{8}{13}t_u + \frac{173}{13}, & 7 \leq t_u \mod T \leq 20 \\ 1, & 20 \leq t_u \mod T \leq 24 \end{cases}$$

- Representation: Array of **(slope-constant) triples** equipped with advanced (eg, binary / predecessor) *search capabilities*.

$\left(\frac{4}{3}, 1, 3\right)$	$(0, 5, 5)$	$(2, -5, 7)$	$\left(-\frac{8}{13}, \frac{173}{13}, 20\right)$	$(0, 1, 24)$

How to Store/Access PWL Arc Delays



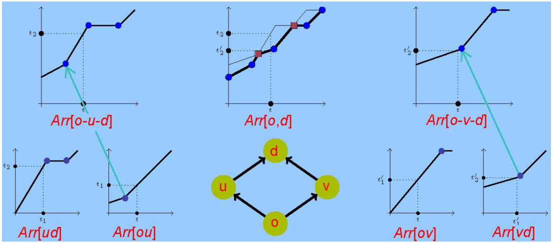
- Exploit *periodicity* and *piecewise-linearity*:

$$\forall t_u \in \mathbb{R},\ \overrightarrow{D}[uv](t_u) = \begin{cases} \frac{4}{3}t_u + 1, & 0 \le t_u \mod T \le 3 \\ 5, & 3 \le t_u \mod T \le 5 \\ 2t_u - 5, & 5 \le t_u \mod T \le 7 \\ -\frac{8}{13}t_u + \frac{173}{13}, & 7 \le t_u \mod T \le 20 \\ 1, & 20 \le t_u \mod T \le 24 \end{cases}$$

- Representation: Array of **(dep.time - delay) pairs** equipped with advanced (eg, binary / predecessor) *search capabilities*.

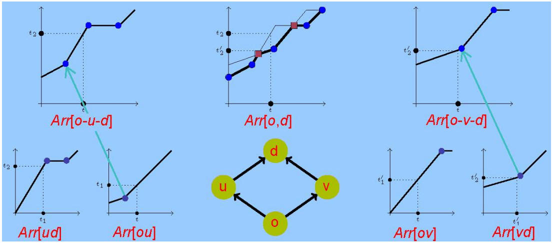
$(0,1)$	$(3,5)$	$(5,5)$	$(7,9)$	$(20,1)$

Piecewise Linearity of Path / Earliest Arrivals



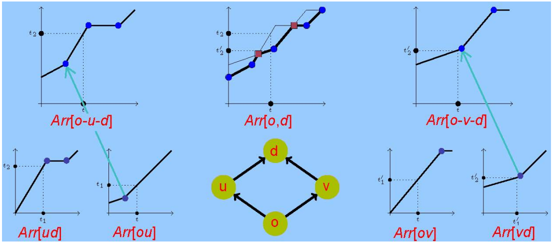
- **Primitive Breakpoint (PB):** Departure-time b'_e from $head[e]$ at which $D[e]$ changes slope (assume $K \in O(m)$ PBs in total).

Piecewise Linearity of Path / Earliest Arrivals



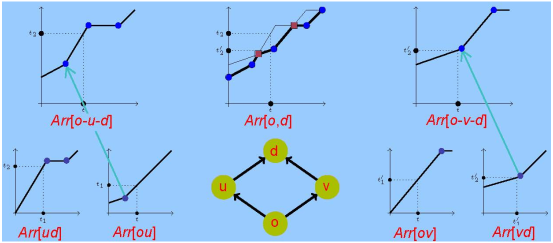
- **Primitive Breakpoint (PB):** Departure-time b'_e from $head[e]$ at which $D[e]$ changes slope (assume $K \in O(m)$ PBs in total).
- **Primitive Image (PI):** **Latest departure-time** b_e from origin o s.t. earliest-arrival-time $b'_e = Arr[o, tail(e)](b_e)$ coincides with a breakpoint for $D[e]$.

Piecewise Linearity of Path / Earliest Arrivals



- **Primitive Breakpoint (PB):** Departure-time b'_e from $head[e]$ at which $D[e]$ changes slope (assume $K \in O(m)$ PBs in total).
- **Primitive Image (PI):** **Latest departure-time** b_e from origin o s.t. earliest-arrival-time $b'_e = Arr[o, tail(e)](b_e)$ coincides with a breakpoint for $D[e]$.
- **Minimization Breakpoint (MB):** Departure-time b_v from origin o s.t. $Arr[o, v]$ **changes slope** due to application of MIN.

Piecewise Linearity of Path / Earliest Arrivals



- **Primitive Breakpoint (PB):** Departure-time b'_e from $head[e]$ at which $D[e]$ changes slope (assume $K \in O(m)$ PBs in total).
- **Primitive Image (PI):** **Latest departure-time** b_e from origin o s.t. earliest-arrival-time $b'_e = Arr[o, tail(e)](b_e)$ coincides with a breakpoint for $D[e]$.
- **Minimization Breakpoint (MB):** Departure-time b_v from origin o s.t. $Arr[o, v]$ **changes slope** due to application of *MIN*.
- Periodicity of arc-delays implies periodicity of earliest-arrival function $Arr[o, d]$.

Known Issues wrt Representations

- **Same representation** both for arc-arrival (or delay) functions and earliest-arrival (or shortest-travel-time) functions.
  - Convenient for handling artificial arcs (representing shortest-travel-time functions) in *overlay abstractions* of the road network.

Known Issues wrt Representations

- **Same representation** both for arc-arrival (or delay) functions and earliest-arrival (or shortest-travel-time) functions.
    - Convenient for handling artificial arcs (representing shortest-travel-time functions) in *overlay abstractions* of the road network.

- Too many (worst case: $n^{\Theta(\log(n))}$) breakpoints to store $Arr[o,d]$ (or $D[o,d]$), even for **linear** arc-delays and **planar** graphs.

Known Issues wrt Representations

- **Same representation** both for arc-arrival (or delay) functions and earliest-arrival (or shortest-travel-time) functions.
  - Convenient for handling artificial arcs (representing shortest-travel-time functions) in *overlay abstractions* of the road network.

- Too many (worst case: $n^{\Theta(\log(n))}$) breakpoints to store $Arr[o,d]$ (or $D[o,d]$), even for **linear** arc-delays and **planar** graphs.

- We need only $O\left(\frac{1}{\varepsilon} \cdot \log\left(\frac{D_{\max}[o,d]}{D_{\min}[o,d]}\right)\right)$ breakpoints for a $(1+\varepsilon)$ **upper approximation** $\overline{D}[o,d]$ of $D[o,d]$, for the case of linear arc-delays.

Complexity of TDSP

Lower Bound: $|BP(Arr_{\text{pwl}}[o,d])| = n^{\Omega(\log n)}$ (I)

A Useful Observation (L2.1-2.2 in FHS11)

For any pair of **monotone, pwl** functions f and g, both their composition $f \circ g$ and their minimum $\min\{f, g\}$ are **monotone, pwl** functions as well.

Lower Bound: $|BP(Arr_{\mathrm{pwl}}[o,d])| = n^{\Omega(\log n)}$ (I)

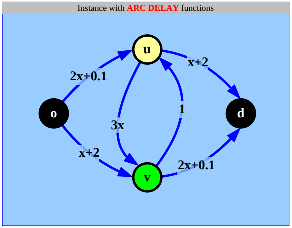
A Useful Observation (L2.1-2.2 in FHS11)

For any pair of **monotone, pwl** functions f and g, both their composition $f \circ g$ and their minimum $\min\{f,g\}$ are **monotone, pwl** functions as well.

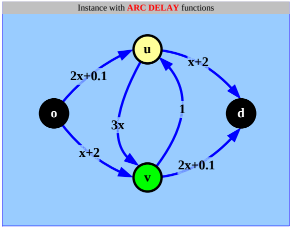
Parametric Shortest Path (PSP): A Similar Problem

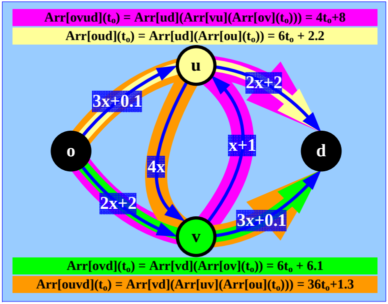
- INPUT: $G = (V,A)$, $o,d \in V$. A *linear length function* $\ell[\alpha](\gamma) = \lambda[\alpha] \cdot \gamma + \mu[\alpha]$ per edge $\alpha \in A$ (negative lengths are **allowed**).
- DEFINITIONS:
  - **Path-length:** $\forall p \in G, L[p](\gamma) = \sum_{\alpha \in p} \ell[\alpha](\gamma)$.
  - **Min-length:** $\forall x,y \in V, L[x,y](\gamma) = \min_{p \in P_{x,y}}\{L[p](\gamma)\}$.
- GOAL1: Compute $L[o,d]$ for a **given value** of γ.
- GOAL2: Compute $L[o,d]$ for **all (real) values** of γ.

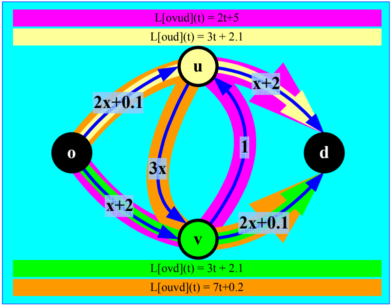
TDSP vs PSP?



TDSP vs PSP?







TDSP: Delay composition along paths

PSP: Delay addition along paths

Lower Bound: $|BP(Arr_{\mathrm{pwl}}[o,d])| = n^{\Omega(\log n)}$ (II)

Known Fact [Carstensen (1984), Mulmuley-Shah (2000)]

There exists (linear) PSP-instance with $n^{\Omega(\log n)}$ BPs in $L[o,d]$.

Lower Bound: $|BP(Arr_{\mathrm{pwl}}[o,d])| = n^{\Omega(\log n)}$ (II)

Known Fact [Carstensen (1984), Mulmuley-Shah (2000)]

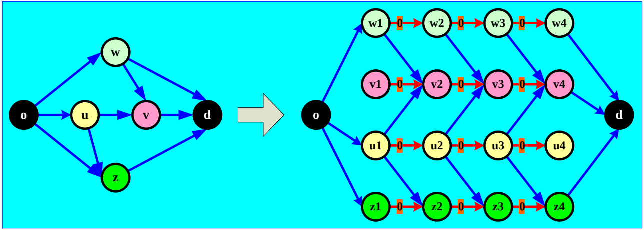
There exists (linear) PSP-instance with $n^{\Omega(\log n)}$ BPs in $L[o,d]$.

Main Steps for TDSP Lower Bound:

1. Assure *non-negativity of lengths* in the PSP instance, in the **departure-time interval of interest**.
2. Scale properly the PSP instance.
3. Consider the corresponding TDSP instance, with parameter γ handled as time.
4. Prove that $L[o,d]$ (for PSP instance) and $D[o,d]$ (for TDSP instance) have (almost) the **same number** of BPs.

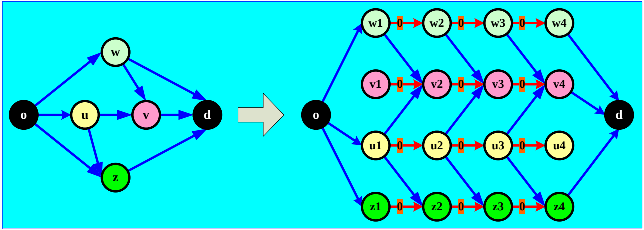
Lower Bound: $|BP(Arr_{\text{pwl}}[o,d])| = n^{\Omega(\log n)}$ (III)

1. Construct a **layered-graph**, in a **path-length-preserving** manner:



Lower Bound: $|BP(Arr_{\mathrm{pwl}}[o,d])| = n^{\Omega(\log n)}$ (III)

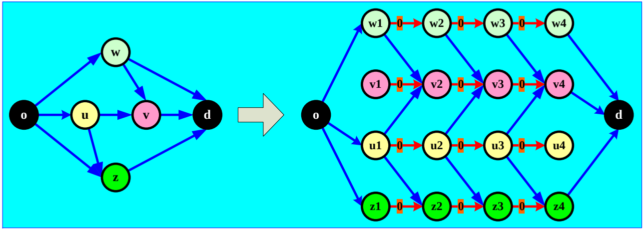
1. Construct a **layered-graph**, in a **path-length-preserving** manner:



Assure non-negativity of arc-lengths in PSP: For the sequence $\langle \gamma_1, \gamma_2, \ldots, \gamma_N \rangle$ of **breakpoints** (BPs) wrt $L[o,d]$, shift arc lengths by $\max\{0, -L_{\min}\}$, $\boxed{L_{\min} = \min_{\gamma \in [\gamma_1, \gamma_N], \alpha \in A(G)} \{L[\alpha](\gamma)\}}$.

Lower Bound: $|BP(Arr_{\mathrm{pwl}}[o,d])| = n^{\Omega(\log n)}$ (III)

1. Construct a **layered-graph**, in a **path-length-preserving** manner:

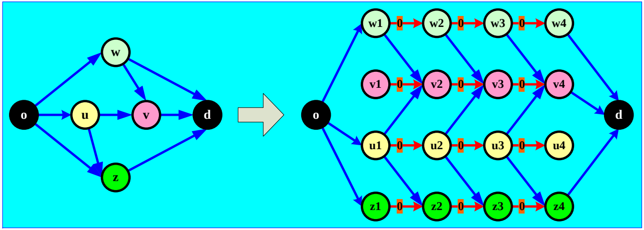


Assure non-negativity of arc-lengths in PSP: For the sequence $\langle \gamma_1, \gamma_2, \ldots, \gamma_N \rangle$ of **breakpoints** (BPs) wrt $L[o,d]$, shift arc lengths by $\max\{0, -L_{\min}\}$, $\boxed{L_{\min} = \min_{\gamma \in [\gamma_1, \gamma_N], \alpha \in A(G)} \{L[\alpha](\gamma)\}}$.

2. Scale arc-lengths in PSP by a proper positive constant μ.

Lower Bound: $|BP(Arr_{\text{pwl}}[o,d])| = n^{\Omega(\log n)}$ (III)

1. Construct a **layered-graph**, in a **path-length-preserving** manner:



Assure non-negativity of arc-lengths in PSP: For the sequence $\langle \gamma_1, \gamma_2, \ldots, \gamma_N \rangle$ of **breakpoints** (BPs) wrt $L[o,d]$, shift arc lengths by $\max\{0, -L_{\min}\}$, $\boxed{L_{\min} = \min_{\gamma \in [\gamma_1, \gamma_N], \alpha \in A(G)} \{L[\alpha](\gamma)\}}$.

2. Scale arc-lengths in PSP by a proper positive constant μ.

3. For the TDSP resulting from the scaled PSP when considering γ as departure time, prove that $\forall j \in \{1, \ldots, N-1\}$, at **"time"** $\bar{\gamma}_j \equiv \frac{\gamma_j + \gamma_{j+1}}{2}$ both instances return *the same* shortest $od-$path p_j.

Lower Bound: $|BP(Arr_{\text{pwl}}[o,d])| = n^{\Omega(\log n)}$ (IV)

How it works: At given $j \in \{1,\ldots,N-1\}$:

- $\bar{\gamma}_j = \frac{\gamma_j + \gamma_{j+1}}{2}$, $\bar{L}_j = L[p_j](\bar{\gamma}_j) = L[o,d](\bar{\gamma}_j)$.
- $L'_j = \min_{q \in P_{s,d} - \{p_j\}} \{L[q](\bar{\gamma}_j)$, $\Delta_j = L'_j - \bar{L}_j > 0$.
- $\boxed{\Delta_{\min} = \min_{j\in[N-1]} \Delta_j \;\Big|\; \varepsilon^* = \frac{\Delta_{\min}}{2n} \;\Big|\; \delta^* = \min_{\alpha\in A:\lambda[\alpha]\neq 0} \left\{ \frac{\Delta_{\min}}{2n|\lambda[\alpha]|} \right\}}$

Lower Bound: $|BP(Arr_{\text{pwl}}[o,d])| = n^{\Omega(\log n)}$ (IV)

How it works: At given $j \in \{1, \ldots, N-1\}$:

- $\bar{\gamma}_j = \frac{\gamma_j + \gamma_{j+1}}{2}$, $\bar{L}_j = L[p_j](\bar{\gamma}_j) = L[o,d](\bar{\gamma}_j)$.
- $L'_j = \min_{q \in P_{s,d} - \{p_j\}} \{L[q](\bar{\gamma}_j)$, $\Delta_j = L'_j - \bar{L}_j > 0$.
$\Delta_{\min} = \min_{j \in [N-1]} \Delta_j$	$\varepsilon^* = \frac{\Delta_{\min}}{2n}$	$\delta^* = \min_{\alpha \in A : \lambda[\alpha] \neq 0} \left\{ \frac{\Delta_{\min}}{2n\lvert\lambda[\alpha]\rvert} \right\}$
- **Arc-delay perturbations:** Small-enough so as *not to affect optimality* of p_j in PSP instance: $\forall \varepsilon_\alpha \in (0, \varepsilon^*]$,

$$\boxed{\sum_{\alpha \in p_j} \ell[\alpha](\bar{\gamma}_j + \varepsilon_\alpha) \leq \bar{L}_j + \frac{\Delta_j}{2} < \frac{\bar{L}_j + L'_j}{2} < L'_j \leq \sum_{\alpha \in q} \ell[\alpha](\bar{\gamma}_j), \ \ \forall q \neq p_j}$$

Lower Bound: $|BP(Arr_{\mathrm{pwl}}[o,d])| = n^{\Omega(\log n)}$ (IV)

How it works: At given $j \in \{1,\ldots,N-1\}$:

- $\bar{\gamma}_j = \frac{\gamma_j+\gamma_{j+1}}{2}$, $\bar{L}_j = L[p_j](\bar{\gamma}_j) = L[o,d](\bar{\gamma}_j)$.
- $L'_j = \min_{q\in P_{s,d}-\{p_j\}}\{L[q](\bar{\gamma}_j)$, $\Delta_j = L'_j - \bar{L}_j > 0$.
- $\boxed{\Delta_{\min} = \min_{j\in[N-1]} \Delta_j}$ $\boxed{\varepsilon^* = \frac{\Delta_{\min}}{2n}}$ $\boxed{\delta^* = \min_{\alpha\in A:\lambda[\alpha]\neq 0}\left\{\frac{\Delta_{\min}}{2n|\lambda[\alpha]|}\right\}}$
- **Arc-delay perturbations:** Small-enough so as *not to affect optimality* of p_j in PSP instance: $\forall \varepsilon_\alpha \in (0,\varepsilon^*]$,

$$\boxed{\sum_{\alpha\in p_j}\ell[\alpha](\bar{\gamma}_j+\varepsilon_\alpha) \le \bar{L}_j + \frac{\Delta_j}{2} < \frac{\bar{L}_j+L'_j}{2} < L'_j \le \sum_{\alpha\in q}\ell[\alpha](\bar{\gamma}_j),\ \ \forall q\neq p_j}$$

- **Departure-time perturbations:** Small-enough so as to cause *not too large arc-delay perturbations*: $\forall\alpha\in A$, $\forall\delta_\alpha\in(0,\delta^*]$,

$$\boxed{D[\alpha](\bar{\gamma}_j+\delta_\alpha) = \ell[\alpha](\bar{\gamma}_j+\delta_\alpha) \le \ell[\alpha](\bar{\gamma}_j)+\varepsilon^*}$$

Lower Bound: $|BP(Arr_{\text{pwl}}[o,d])| = n^{\Omega(\log n)}$ (V)

How it works (continued): At given $j \in \{1, \ldots, N-1\}$:

- Scale-invariance of time-perturbations: **Scaling** of all arc-delays by a **positive** number $\mu > 0$ does not affect at all the range of allowed **time-perturbations** $\delta^* = \min_{\alpha \in A: \lambda[\alpha] \neq 0} \left\{ \frac{\Delta_{\min}}{2n|\lambda[\alpha]|} \right\}$.

Lower Bound: $|BP(Arr_{\mathrm{pwl}}[o,d])| = n^{\Omega(\log n)}$ (V)

How it works (continued): At given $j \in \{1, \ldots, N-1\}$:

- Scale-invariance of time-perturbations: **Scaling** of all arc-delays by a **positive** number $\mu > 0$ does not affect at all the range of allowed **time-perturbations** $\delta^* = \min_{\alpha \in A : \lambda[\alpha] \neq 0} \left\{ \frac{\Delta_{\min}}{2n|\lambda[\alpha]|} \right\}$.
- TDSP-instance: Scale the PSP-instance by $\mu = \frac{\delta^*}{2(L_{\max} + \Delta_{\min})}$. Handle the PSP-parameter γ as time.

Lower Bound: $|BP(Arr_{\text{pwl}}[o,d])| = n^{\Omega(\log n)}$ (V)

How it works (continued): At given $j \in \{1, \ldots, N-1\}$:

- Scale-invariance of time-perturbations: **Scaling** of all arc-delays by a **positive** number $\mu > 0$ does not affect at all the range of allowed **time-perturbations** $\delta^* = \min_{\alpha \in A : \lambda[\alpha] \neq 0} \left\{ \frac{\Delta_{\min}}{2n|\lambda[\alpha]|} \right\}$.
- TDSP-instance: Scale the PSP-instance by $\mu = \frac{\delta^*}{2(L_{\max} + \Delta_{\min})}$. Handle the PSP-parameter γ as time.
- **Proper scaling** guarantees sufficiently small **departure-time perturbations**: $Arr[p_j](\bar{\gamma}_j) = \bar{\gamma}_j + D[p_j](\bar{\gamma}_j) < \bar{\gamma}_j + \delta^*$.

Lower Bound: $|BP(Arr_{\mathrm{pwl}}[o,d])| = n^{\Omega(\log n)}$ (V)

How it works (continued): At given $j \in \{1,\ldots,N-1\}$:

- Scale-invariance of time-perturbations: **Scaling** of all arc-delays by a **positive** number $\mu > 0$ does not affect at all the range of allowed **time-perturbations** $\delta^* = \min_{\alpha\in A:\lambda[\alpha]\neq 0}\left\{\frac{\Delta_{\min}}{2n|\lambda[\alpha]|}\right\}$.
- TDSP-instance: Scale the PSP-instance by $\mu = \frac{\delta^*}{2(L_{\max}+\Delta_{\min})}$. Handle the PSP-parameter γ as time.
- **Proper scaling** guarantees sufficiently small **departure-time perturbations**: $Arr[p_j](\bar{\gamma}_j) = \bar{\gamma}_j + D[p_j](\bar{\gamma}_j) < \bar{\gamma}_j + \delta^*$.

$\therefore$ Small time-perturbations guarantee sufficiently small **arc-delay perturbations**, and thus, optimality of p_j:

$$\boxed{\begin{aligned} D[p_j](\bar{\gamma}_j) &\leq \mu\cdot\bar{L}_j + \mu\cdot\frac{(n-1)\Delta_{\min}}{2n} \\ &< \mu\cdot L'_j - \mu\frac{(n-1)\Delta_{\min}}{2n} \leq D[q](\bar{\gamma}_j),\ \forall q\neq p_j \end{aligned}}$$

QED

Upper Bound: $|BP(Arr_{\text{pwl}}[o,d])| = K \cdot n^{O(\log n)}$ (I)

Observation: (L4.1 in FHS11)

Between any two consecutive Primitive Images (PIs) $t_j < t_{j+1}$, $Arr[o,d]$ forms a concave chain.

Upper Bound: $|BP(Arr_{\mathrm{pwl}}[o,d])| = K \cdot n^{O(\log n)}$ (I)

Observation: (L4.1 in FHS11)

Between any two consecutive Primitive Images (PIs) $t_j < t_{j+1}$, $Arr[o,d]$ forms a concave chain.

WHY?

- Any arc-delay is **linear** (no primitive breakpoints occur at edges), if the departure-time domain is restricted to (t_j, t_{j+1}).
- Any path-arrival $Arr[p](t)$ function is a *composition* of linear functions, thus **linear**.
- $Arr[o,d]$ is the application of the min operator among linear functions, thus **concave**.

Upper Bound: $|BP(Arr_{\mathrm{pwl}}[o,d])| = K \cdot n^{O(\log n)}$ (I)

Observation: (L4.1 in FHS11)

Between any two consecutive Primitive Images (PIs) $t_j < t_{j+1}$, $Arr[o,d]$ forms a concave chain.

WHY?

- Any arc-delay is linear (no primitive breakpoints occur at edges), if the departure-time domain is restricted to (t_j, t_{j+1}).
- Any path-arrival $Arr[p](t)$ function is a *composition* of linear functions, thus linear.
- $Arr[o,d]$ is the application of the min operator among linear functions, thus **concave**.

- Corollary: $|BP(Arr_{\mathrm{pwl}}[o,d])| \leq$ #different path slopes

Upper Bound: $|BP(Arr_{\mathrm{pwl}}[o,d])| = K \cdot n^{O(\log n)}$ (I)

Observation: (L4.1 in FHS11)

Between any two consecutive Primitive Images (PIs) $t_j < t_{j+1}$, $Arr[o,d]$ forms a concave chain.

WHY?

- Any arc-delay is linear (no primitive breakpoints occur at edges), if the departure-time domain is restricted to (t_j, t_{j+1}).
- Any path-arrival $Arr[p](t)$ function is a *composition* of linear functions, thus linear.
- $Arr[o,d]$ is the application of the min operator among linear functions, thus **concave**.

- Corollary: $|BP(Arr_{\mathrm{pwl}}[o,d])| \leq$ #different path slopes
- Is this enough?

Upper Bound: $|BP(Arr_{\text{pwl}}[o,d])| = K \cdot n^{O(\log n)}$ (I)

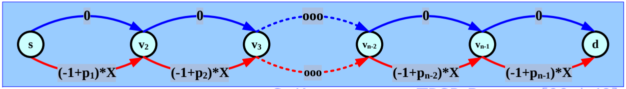
Observation: (L4.1 in FHS11)

Between any two consecutive Primitive Images (PIs) $t_j < t_{j+1}$, $Arr[o,d]$ forms a concave chain.

WHY?

- Any arc-delay is linear (no primitive breakpoints occur at edges), if the departure-time domain is restricted to (t_j, t_{j+1}).
- Any path-arrival $Arr[p](t)$ function is a *composition* of linear functions, thus linear.
- $Arr[o,d]$ is the application of the min operator among linear functions, thus concave.

- Corollary: $|BP(Arr_{\text{pwl}}[o,d])| \leq$ #different path slopes
- Is this enough?
- *NO!!!*



Upper Bound: $|BP(Arr_{\mathrm{pwl}}[o,d])| = K \cdot n^{O(\log n)}$ (II)

OBSERVATION II: (L4.2 in FHS11)

$|BP(Arr_{\mathrm{pwl}}[o,d])| \leq K \cdot |BP(Arr_{\mathrm{lin}}[o,d])|.$

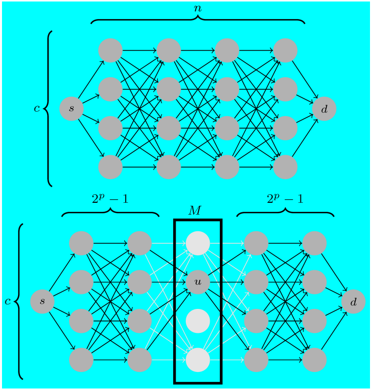
Upper Bound: $|BP(Arr_{\mathrm{pwl}}[o,d])| = K \cdot n^{O(\log n)}$ (II)

OBSERVATION II: (L4.2 in FHS11)

$|BP(Arr_{\mathrm{pwl}}[o,d])| \leq K \cdot |BP(Arr_{\mathrm{lin}}[o,d])|$.

Lemma 4.3 (FHS11)

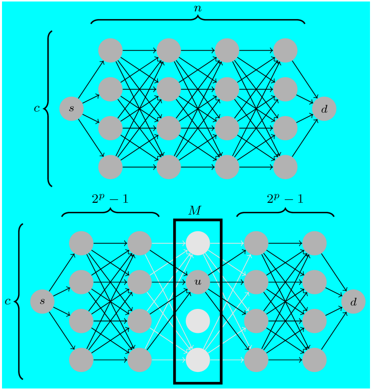
$|BP(Arr_{\mathrm{lin}}[o,d])| \leq \frac{(2n+1)^{1+\log c}}{2}$ in a layered graph with c layers of n nodes each.



Upper Bound: $|BP(Arr_{\mathrm{pwl}}[o,d])| = K \cdot n^{O(\log n)}$ (II)

OBSERVATION II: (L4.2 in FHS11)

$|BP(Arr_{\mathrm{pwl}}[o,d])| \leq K \cdot |BP(Arr_{\mathrm{lin}}[o,d])|$.



Lemma 4.3 (FHS11)

$|BP(Arr_{\mathrm{lin}}[o,d])| \leq \frac{(2n+1)^{1+\log c}}{2}$ in a **layered graph** with c layers of n nodes each.

THM4.4 (FHS11)

$|BP(Arr_{\mathrm{lin}}[o,d])| = n^{O(\log n)}$ in any graph G and pair of nodes $o, d \in V(G)$.

(Exact) Output Sensitive Algorithm for Earliest-Arrival Functions

Why Do We Need the Output Sensitive Algorithm?

Why Do We Need the Output Sensitive Algorithm?

1. It gives exactly the distance functions in question, ie, **functional descriptions of earliest-arrivals**, that we would ideally like to have from/to any origin/destination vertex.

Why Do We Need the Output Sensitive Algorithm?

1. It gives exactly the distance functions in question, ie, **functional descriptions of earliest-arrivals**, that we would ideally like to have from/to any origin/destination vertex.

2. We may need to compute *exact distance summaries* for **special pairs of vertices** (eg, from/to hubs, all superhub-to-superhub connections, etc).

Why Do We Need the Output Sensitive Algorithm?

1. It gives exactly the distance functions in question, ie, **functional descriptions of earliest-arrivals**, that we would ideally like to have from/to any origin/destination vertex.

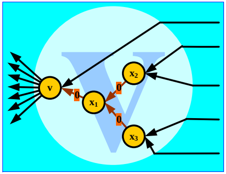
2. We may need to compute *exact distance summaries* for **special pairs of vertices** (eg, from/to hubs, all superhub-to-superhub connections, etc).

3. Interesting to discover whether the complexity of the earliest-arrival functions is indeed so bad **in real (e.g., road) networks**.

The Output-Sensitive Algorithm (I)

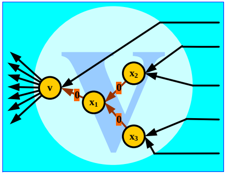
The Output-Sensitive Algorithm (I)

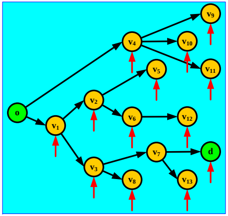
- ASSUMPTION: The in-degree of every node in the graph is at most 2.



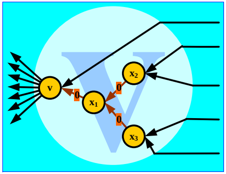
The Output-Sensitive Algorithm (I)

- ASSUMPTION: The in-degree of every node in the graph is at most 2.
- Given an arbitrary point in time (*"current time"*) $t_0 \geq 0$ as departure time from origin o, compute a **TDSP tree**.

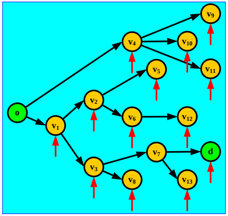




The Output-Sensitive Algorithm (I)



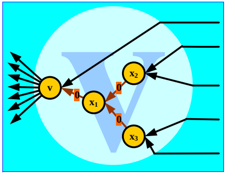
- ASSUMPTION: The in-degree of every node in the graph is at most 2.
- Given an arbitrary point in time (*"current time"*) $t_0 \geq 0$ as departure time from origin o, compute a **TDSP tree**.
- Discover *until when* the TDSP tree is **valid**.



  - $\forall v \in V$, two short alternatives when departing from o at time t_0: Earliest-arrival to each parent, plus delay of corresponding incoming arc.
  - **Minimization (vertex) Certificate** $t_{fail}[v]$: Earliest departure time from o at which the two alternatives of v become **equivalent**.

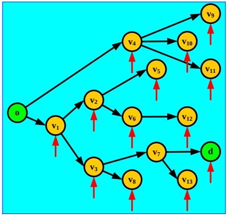
    Primitive (arc) Certificate $t_{fail}[e]$: Primitive image of the next (ie, after t_0) breakpoint of the arc to come.

The Output-Sensitive Algorithm (I)



- ASSUMPTION: The in-degree of every node in the graph is at most 2.
- Given an arbitrary point in time (*"current time"*) $t_0 \geq 0$ as departure time from origin o, compute a **TDSP tree**.
- Discover *until when* the TDSP tree is **valid**.
  - $\forall v \in V$, two short alternatives when departing from o at time t_0: Earliest-arrival to each parent, plus delay of corresponding incoming arc.
  - **Minimization (vertex) Certificate** $t_{fail}[v]$: Earliest departure time from o at which the two alternatives of v become **equivalent**.

    Primitive (arc) Certificate $t_{fail}[e]$: Primitive image of the next (ie, after t_0) breakpoint of the arc to come.



- All $(m+n)$ certificates temporarily stored in a *priority queue*.

The Output-Sensitive Algorithm (II)

When current time $t_1 > t_0$ matches the earliest failure-time of a certificate in the queue:

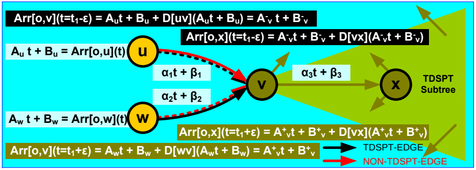
if minimization-certificate failure, at node $v \in V$:

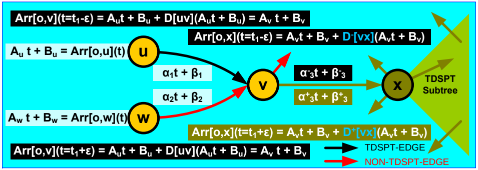
then (1) Update shortest $ov-$path

/* **ONE-BIT** change in combinatorial structure */

(2) Update $Arr[o, x]$ and $t_{fail}[x]$, $\forall x \in T_v$.

(3) Update $t_{fail}[e]$, $\forall e \in E : x = tail[e] \in T_v$.

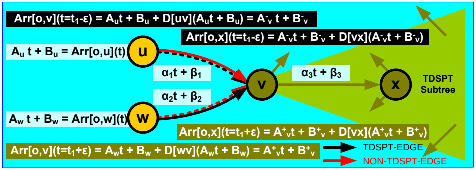




The Output-Sensitive Algorithm (II)

When current time $t_1 > t_0$ matches the earliest failure-time of a certificate in the queue:

if minimization-certificate failure, at node $v \in V$:

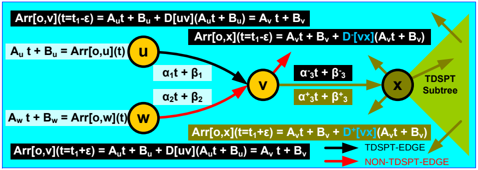


then (1) Update shortest $ov-$path

/∗ **ONE-BIT** change in combinatorial structure ∗/

(2) Update $Arr[o, x]$ and $t_{fail}[x]$, $\forall x \in T_v$.

(3) Update $t_{fail}[e]$, $\forall e \in E : x = tail[e] \in T_v$.



else /∗ primitive-certificate failure, at arc $e = vx \in E$ ∗/

(1) Update $Arr[o,y]$ and $t_{fail}[y]$, $\forall y \in T_x$.

(2) Update $t_{fail}[e']$, $\forall e' \in E : tail[e'] \in T_x$.

The Output-Sensitive Algorithm (III)

- What to keep in memory:
  - Breakpoint triples for earliest-arrival functions, plus ONE bit (indicating the parent).
  - Advanced search structures, if number of BPs is large.
  - Only temporarily store certificates in a priority queue.

The Output-Sensitive Algorithm (III)

- What to keep in memory:
  - Breakpoint triples for earliest-arrival functions, plus ONE bit (indicating the parent).
  - Advanced search structures, if number of BPs is large.
  - Only temporarily store certificates in a priority queue.
- *Response-time* per certificate failure at $c \in V \cup E$:
  - In the *in-degrees-2 graph* (or any constant-in-degree graph): $\mathrm{O}(|E_c| \cdot \log n)$. E_c is the set of arcs whose tails are in T_c, or $T_{head[c]}$. Logarithmic factor is due to **priority-queue operations**.
  - In the *original graph* (in worst-case): $\mathrm{O}\left(m \times \log^2 n\right)$. Second logarithmic factor is due to **updates of tournament trees** implementing the MIN operator at a particular node, upon emergence of a single certificate failure.

The Output-Sensitive Algorithm (III)

- What to keep in memory:
  - Breakpoint triples for earliest-arrival functions, plus ONE bit (indicating the parent).
  - Advanced search structures, if number of BPs is large.
  - Only temporarily store certificates in a priority queue.
- *Response-time* per certificate failure at $c \in V \cup E$:
  - In the *in-degrees-2 graph* (or any constant-in-degree graph): $O(|E_c| \cdot \log n)$. E_c is the set of arcs whose tails are in T_c, or $T_{head[c]}$. Logarithmic factor is due to **priority-queue operations**.
  - In the *original graph* (in worst-case): $O\left(m \times \log^2 n\right)$. Second logarithmic factor is due to **updates of tournament trees** implementing the MIN operator at a particular node, upon emergence of a single certificate failure.
- *Worst-case time-complexity* of output-sensitive algorithm:

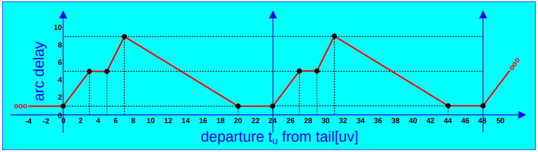
$$O\left(m \times \log^2 n \times (\text{PRIMBPs} + \text{MINBPs})\right)$$

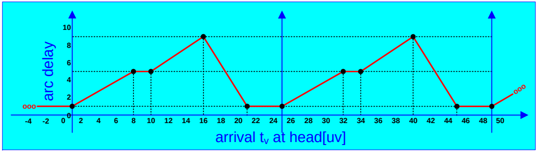
Poly-time Approximation Algorithms

$(1+\varepsilon)$–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!!!

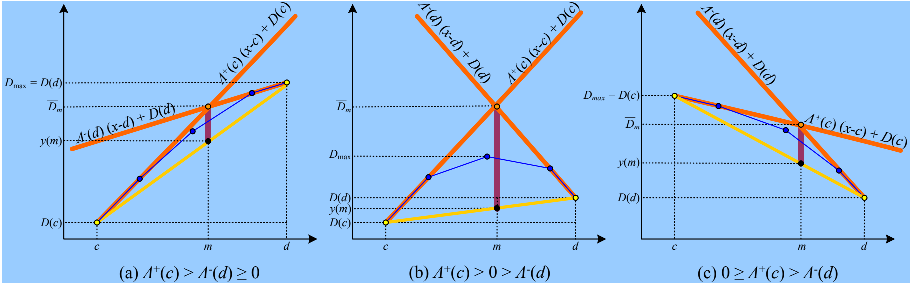




- $t_o = \overleftarrow{Arr}[o,v](t_v) = t_v - \overleftarrow{D}[o,v](t_v)$: **Latest-departure-time** from o to v, as a function of the arrival time t_v at v.

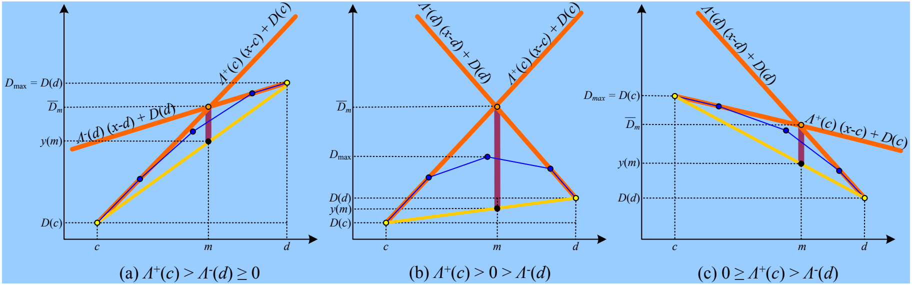
Approximating $D[o,d]$: Quality

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



Approximating $D[o,d]$: Quality

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



LEMMA: Closed Form of Maximum Absolute Error

[Kontogianis-Zaroliagis (2013)]

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

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

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

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

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 $(1+\epsilon)-$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 $\forall t \in [0,T)$,

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

- FACT: **if** $D[o,d]$ was a priori known **then** a **linear scan** gives a space-optimal $(1+\epsilon)-$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 $\forall t \in [0,T)$,

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

- FACT: **if** $D[o,d]$ was a priori known **then** a **linear scan** gives a space-optimal $(1+\epsilon)-$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.

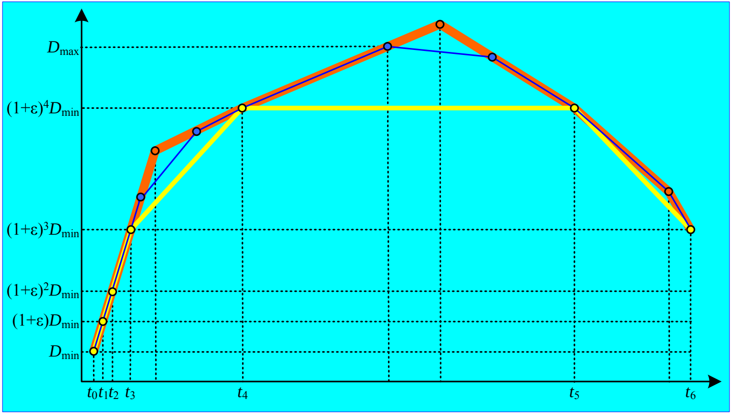
Approximating $D[o,d]$: Basic Idea (I)

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

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

- FACT: **if** $D[o,d]$ was a priori known **then** a **linear scan** gives a space-optimal $(1+\epsilon)-$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**

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

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

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

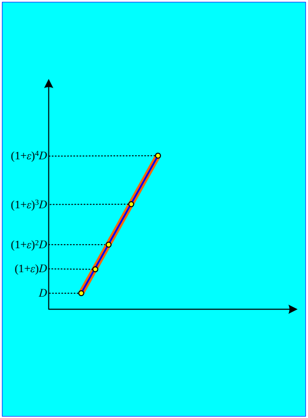
$$\vdots$$

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)] :

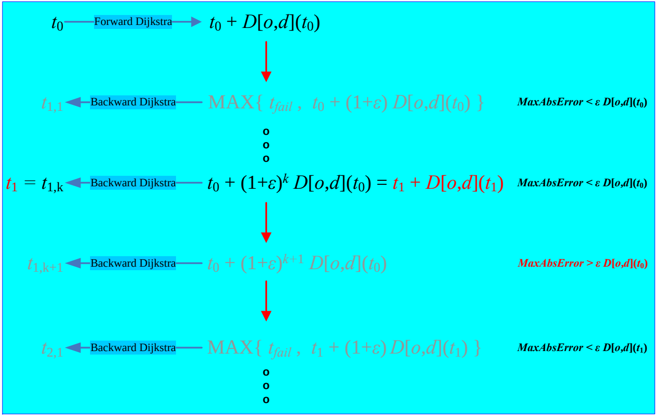


One-To-One Approximation: PHASE-1

[Foschini-Hershberger-Suri (2011)]

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

[Kontogiannis-Zaroliagis (2013)] :



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 (2013)]

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 (2013)]

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

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

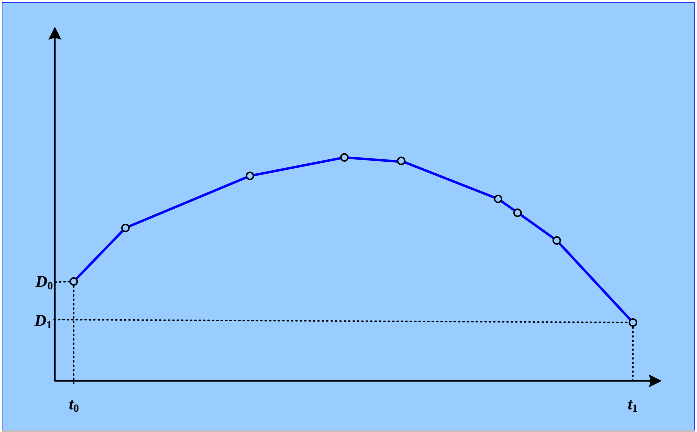
ASSUMPTION 2: Bounded Travel-Time Slopes. Small slopes of the (pwl) arc-delay functions.

- **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*.

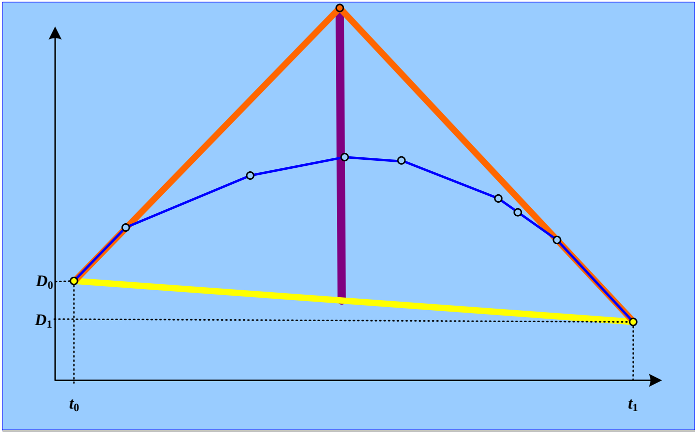


Example of Bisection Execution : INPUT = **UNKNOWN BLUE** function

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*.

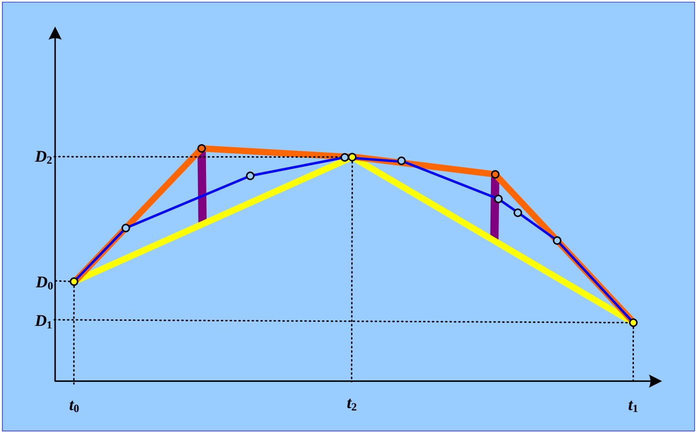


Example of Bisection Execution : **ORANGE** = Upper Bound, **YELLOW** = Lower Bound

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*.

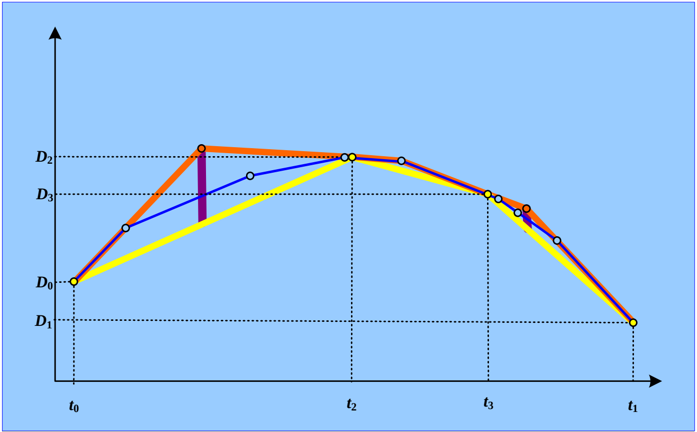


Example of Bisection Execution : Level-1 Recursion

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*.



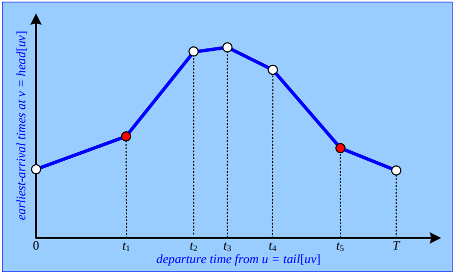
Example of Bisection Execution : Level-2 Recursion

One-To-All Approximaton via Bisection (III)

[Kontogiannis-Zaroliagis (2013)]

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 $D[o,d]$: Space/Time Complexity

THEOREM: Space Complexity [Kontogianis-Zaroliagis (2013)]

Let K^* be the total number of concavity-spoiling BPs among all the arc-delay functions in the instance.

Space Complexity: For a *given orign* $o \in V$ and *all* possible destinations $d \in V$, the following complexity bounds hold for creating all the approximation functions $\overline{D}[o,*] = (\overline{D}[o,d])_{d \in V}$:

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

Time Complexity: The number of *shortest-path probes* executed for the computation of the approximate distance functions is:
$TDSP[o,d] \in O\left(\log\left(\frac{T}{\varepsilon \cdot D_{\min}[o,d]}\right) \cdot \frac{K^*}{\varepsilon} \log\left(\frac{D_{\max}[o,*](0,T)}{D_{\min}[o,*](0,T)}\right)\right)$

Implementation Issues wrt One-To-All Bisection

- `One-To-All Bisection` of [Kontogiannis-Zaroliagis (2013)] is a **label-setting** approximation method that provably works *space/time optimally* (within constant factors) wrt **concave** continuous pwl arc-delay functions.

Implementation Issues wrt One-To-All Bisection

- One-To-All Bisection of [Kontogiannis-Zaroliagis (2013)] is a **label-setting** approximation method that provably works *space/time optimally* (within constant factors) 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 (2013)] suffer from **linear dependence** in the degree of disconcavity (value of K^*) in the TD Instance.

Implementation Issues wrt One-To-All Bisection

- `One-To-All Bisection` of [Kontogiannis-Zaroliagis (2013)] is a **label-setting** approximation method that provably works *space/time optimally* (within constant factors) 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 (2013)] suffer from **linear dependence** in the degree of disconcavity (value of K^*) in the TD Instance.

- A novel **one-to-all** (again **label-setting**) approximation technique, called the `Trapezoidal` method ([Kontogiannis-Wagner-Zaroliagis (2014)]) avoids entirely the dependence of the required space from the network structure (and, of course, the degree of disconcavity).

Any Questions?

Next Lecture
Time-Dependent Oracles
Wednesday, June 18 2014, 11:30

Distance Oracles

Distance Oracles

A Success Story in Static Graphs

CHALLENGE: Given a *large scale* graph with arc-travel-times, create a data structure (**oracle**) that requires *reasonable space* requirements and allows answering **distance queries** *efficiently*.

CHALLENGE: Given a *large scale* graph with arc-travel-times, create a data structure (**oracle**) that requires *reasonable space* requirements and allows answering **distance queries** *efficiently*.

- Trivial solution: Preprocess by executing and storing **APSP**.
  - $O(n^2)$ size.
  - $O(1)$ query time.
  - $1-$stretch.

CHALLENGE: Given a *large scale* graph with arc-travel-times, create a data structure (**oracle**) that requires *reasonable space* requirements and allows answering **distance queries** *efficiently*.

- Trivial solution: Preprocess by executing and storing **APSP**.
  - $O(n^2)$ size.
  - $O(1)$ query time.
  - $1-$stretch.

- Trivial solution: No preprocessing, respond to queries by running `Dijkstra`.
  - $O(n+m)$ size.
  - $O(m+n\log(n))$ query time.
  - $1-$stretch.

CHALLENGE: Given a *large scale* graph with arc-travel-times, create a data structure (**oracle**) that requires *reasonable space* requirements and allows answering **distance queries** *efficiently*.

- Trivial solution: Preprocess by executing and storing **APSP**.
  - ☹ $O(n^2)$ size.
  - ☺ $O(1)$ query time.
  - ☺ $1-$stretch.

- Trivial solution: No preprocessing, respond to queries by running `Dijkstra`.
  - ☺ $O(n+m)$ size.
  - ☹ $O(m+n\log(n))$ query time.
  - ☺ $1-$stretch.

💡 Try to provide smooth tradeoffs among space / query time / stretch!!!

Distance Oracles: Generic Idea

1. Metric-independent preprocessing: Split the graph, essentially **ignoring** the distance metric.

Distance Oracles: Generic Idea

1. Metric-independent preprocessing: Split the graph, essentially **ignoring** the distance metric.
   - Roughly **equal size** per cell (in each level, if *recursive division* is applied).
   - Boundary vertices/arcs much less than the graph size (e.g., $O(\sqrt{n})$ in total).
   - Each cell may be required to be a **weakly connected** subgraph.

Distance Oracles: Generic Idea

1. Metric-independent preprocessing: Split the graph, essentially **ignoring** the distance metric.
   - Roughly **equal size** per cell (in each level, if *recursive division* is applied).
   - Boundary vertices/arcs much less than the graph size (e.g., $O(\sqrt{n})$ in total).
   - Each cell may be required to be a **weakly connected** subgraph.
2. Metric-dependent preprocessing: Equip the network with selective **distance summaries**, e.g., boundary-to-boundary / boundary-to-cell / boundary-to-all distances.

Distance Oracles: Generic Idea

1. Metric-independent preprocessing: Split the graph, essentially **ignoring** the distance metric.
   - Roughly **equal size** per cell (in each level, if *recursive division* is applied).
   - Boundary vertices/arcs much less than the graph size (e.g., $O(\sqrt{n})$ in total).
   - Each cell may be required to be a **weakly connected** subgraph.
2. Metric-dependent preprocessing: Equip the network with selective **distance summaries**, e.g., boundary-to-boundary / boundary-to-cell / boundary-to-all distances.
3. Query Algorithm: Respond fast to queries, based on the (hierarchical?) *distance-independent* division and/or the *distance-dependent* summaries.

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 extended 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 extended to time-dependent case.

IN THIS TALK

The focus is on **time-dependent oracles**, with **provably good** preprocessing-space / query-time / stretch tradeoffs.

Theoretical Bounds for Static Graphs

Reference	Setting	Stretch	Query	Space
[TZ05]	weighted graph	$2k-1$, $k \geq 2$	$O(k)$	$O\left(kn^{1+1/k}\right)$
[WN13]	weighted graph	$2k-1$, $k \geq 2$	$O(\log(k))$	$O\left(kn^{1+1/k}\right)$
[Che13]	weighted graph	$2k-1$, $k \geq 2$	$O(1)$	$O\left(kn^{1+1/k}\right)$
[AG13]	sparse weighted graph	$1+\epsilon$	$o(n)$	$o\left(n^2\right)$
[Kle02] [Tho04]	planar weighted digraph	$1+\epsilon$	$O\left(\epsilon^{-1}\right)$	$O\left(\frac{n\log(n)}{\epsilon}\right)$
[MN06]	metric	$O(k)$	$O(1)$	$O\left(kn^{1+1/k}\right)$
[BGKRL11]	Doubling metric, dynamic	$1+\epsilon$	$O(1)$	$\epsilon^{-O(ddim)}n + 2^{O(ddim\log(ddim))}n$

CHALLENGE: Given a *large scale* 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).

CHALLENGE: Given a *large scale* 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: Precompute all the $(1+\epsilon)-$approximate distance summaries from every origin to every destination.
  - $O(n^3)$ size ($O(n^2)$, if all arc-delay functions concave).
  - $O(\log\log(n))$ query time.
  - $(1+\epsilon)-$stretch.

CHALLENGE: Given a *large scale* 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: Precompute all the $(1+\epsilon)-$approximate distance summaries from every origin to every destination.
  - ☹ $O(n^3)$ size ($O(n^2)$, if all arc-delay functions concave).
  - ☺ $O(\log\log(n))$ query time.
  - ☺ $(1+\epsilon)-$stretch.

- Trivial solution: No preprocessing, respond to queries by running `TD-Dijkstra`.
  - ☺ $O(n+m+K)$ size (K = total number of PBs of arc-delays).
  - ☹ $O([m+n\log(n)]\times\log\log(K))$ query time.
  - ☺ $1-$stretch.

Distance Oracles

Is it a Success Story in Time-Dependent Graphs?

CHALLENGE: Given a *large scale* 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: Precompute all the $(1+\epsilon)-$approximate distance summaries from every origin to every destination.
  - ☹ $O(n^3)$ size ($O(n^2)$, if all arc-delay functions concave).
  - ☺ $O(\log\log(n))$ query time.
  - ☺ $(1+\epsilon)-$stretch.
- Trivial solution: No preprocessing, respond to queries by running `TD-Dijkstra`.
  - ☺ $O(n+m+K)$ size (K = total number of PBs of arc-delays).
  - ☹ $O([m+n\log(n)]\times\log\log(K))$ query time.
  - ☺ $1-$stretch.

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

FLAT TD-Oracle

FLAT TD-Oracle: Overall Idea

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

FLAT TD-Oracle: Overall Idea

1. Choose a set L of **landmarks**.
   - In theory: Each vertex $v \in V$ is chosen, *independently of other vertices*, to be included in the landmark set L w.p. $\rho \in (0, 1)$.
   - In practice: Selection of landmark set 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

1. Choose a set L of **landmarks**.
   - In theory: Each vertex $v \in V$ is chosen, *independently of other vertices*, to be included in the landmark set L w.p. $\rho \in (0, 1)$.
   - In practice: Selection of landmark set 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.p. $\rho \in (0, 1)$ for the **landmark set** $L \subseteq V$.
- `Preprocessing:` $\forall \ell \in L$, precompute $(1 + \epsilon)-$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.p. $\rho \in (0,1)$ for the **landmark set** $L \subseteq V$.
- `Preprocessing:` $\forall \ell \in L$, precompute $(1+\epsilon)-$approximate distance functions $\Delta[\ell, v]$ to all destinations $v \in V$.

THEOREM: [Kontogiannis-Zaroliagis (2013)]

Using `Bisection` for computing approximate distance summaries:

- Pre-Space:

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

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

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

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 (2014)]

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

- Pre-Space:

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

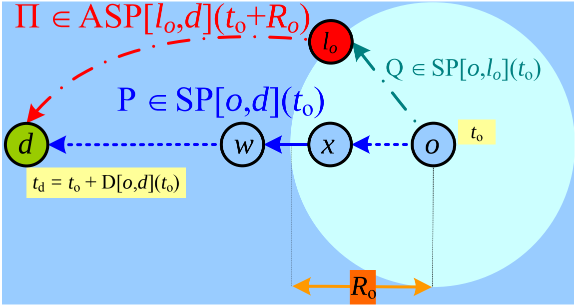
- Pre-Time:

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

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

Under ASSUMPTIONS 2-3, and any route planning request (o,d,t_o), `FCA` achieves the following performance:

- Approximation guarantee:

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

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

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

Under ASSUMPTIONS 2-3, and any route planning request (o,d,t_o), `FCA` achieves the following performance:

- Approximation guarantee:

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

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

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

FLAT TD-Oracle

RQA: boosting approximation guarantee

RQA: Overview

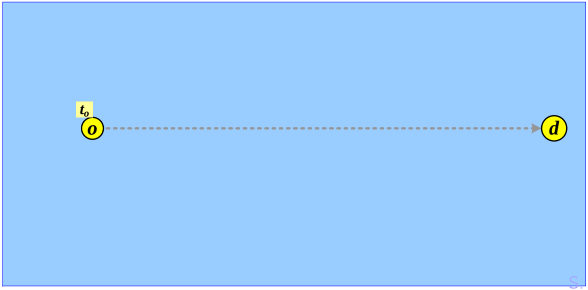
Recursive Query Approximation: $RQA\left(o, d, t_o, (\Delta[\ell, v])_{(\ell,v)\in L\times V}, R\right)$

1. **while** recursion budget R not exhausted **do**
2. Exploration: Grow a TD-Dijkstra forward-ball $B(w_i, t_i)$ until the 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: $RQA\left(o, d, t_o, (\Delta[\ell, v])_{(\ell,v)\in L\times V}, R\right)$

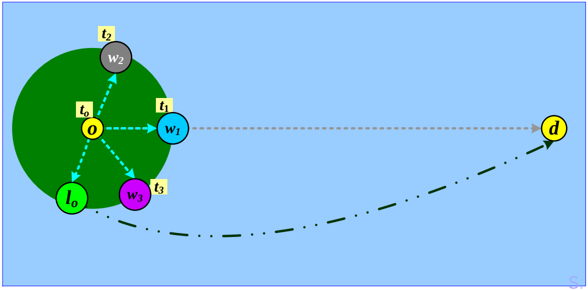
1. **while** recursion budget R not exhausted **do**
2. Exploration: Grow a TD-Dijkstra forward-ball $B(w_i, t_i)$ until the closest landmark ℓ_i is settled.
3. $sol_i = D[o, w_i](t_o) + D[w_i, \ell_i](t_i) + \Delta[\ell_i, d](t_i + D[w_i, \ell_i](t_i))$.
4. Recursion: Execute RQA centered at *each boundary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

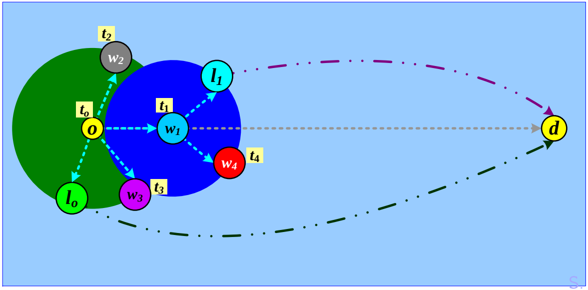


- Growing level-0 ball...

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 the 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.

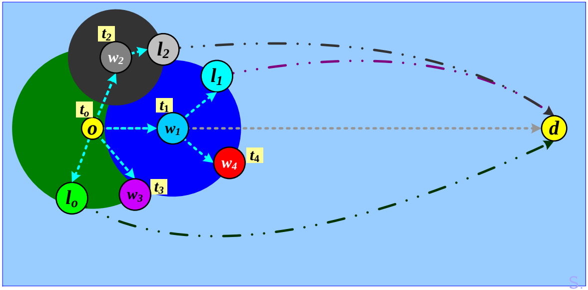


- Growing level-0 ball...
- Growing level-1 balls...

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 the 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.

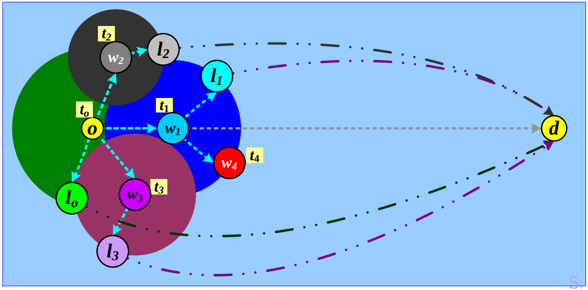


- Growing level-0 ball...
- Growing level-1 balls...

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 the 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.

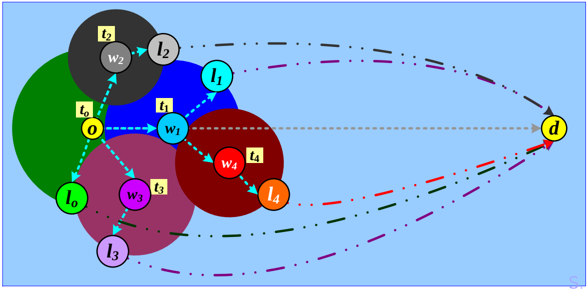


- Growing level-0 ball...
- Growing level-1 balls...

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 the 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.

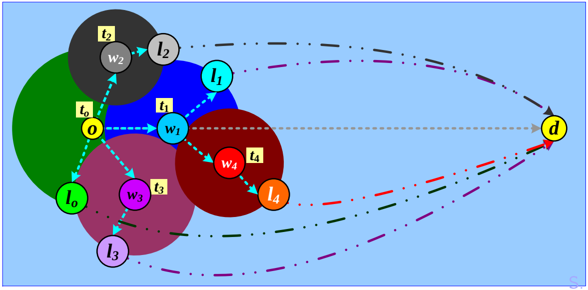


- Growing level-0 ball...
- Growing level-1 balls...
- Growing level-2 balls...

RQA: Overview

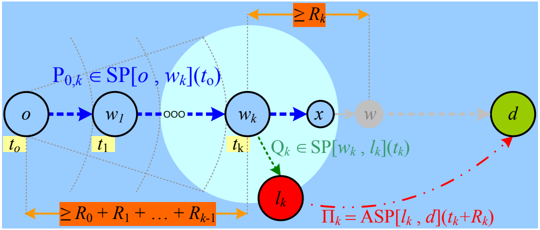
Recursive Query Approximation: $RQA\left(o, d, t_o, (\Delta[\ell, v])_{(\ell,v)\in L\times V}, R\right)$

1. **while** recursion budget R not exhausted **do**
2. Exploration: Grow a TD-Dijkstra forward-ball $B(w_i, t_i)$ until the 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.

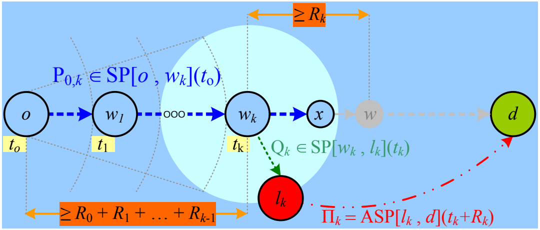


- Growing level-0 ball...
- Growing level-1 balls...
- Growing level-2 balls...
- ...

RQA: Why Does Recursion Boost Approximation?

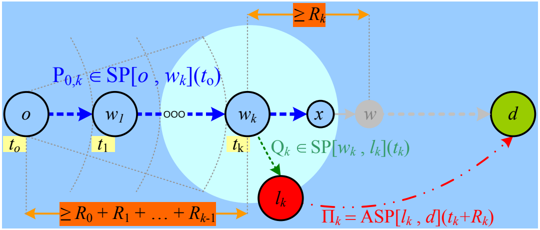


RQA: Why Does Recursion Boost Approximation?



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

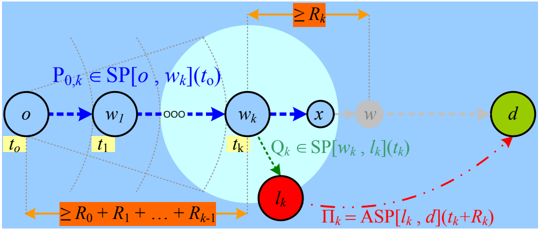
RQA: Why Does Recursion Boost Approximation?



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?

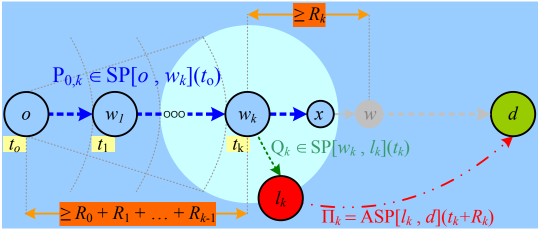


1. One of the discovered approximate *od*−paths has all its ball centers at *nodes of the (unknown) shortest od-path*.
2. Optimal **prefix subpaths** improve approximation guarantee:

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

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 $1+\epsilon$.

RQA: Performance

THEOREM: Complexity of RQA

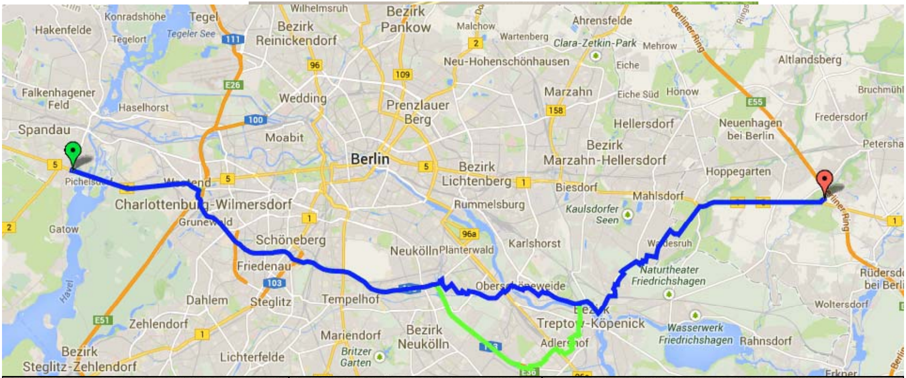
For sparse networks (i.e., having $\mu = |A|/|V| \in O(1)$), the complexity of `RQA` with recursion budget R for obtaining $(1+\sigma)-$approximate distances (for any constant $\sigma > \epsilon$) to arbitrary (o, d, t_o) queries, is:

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

TD Distance Oracle: Recap

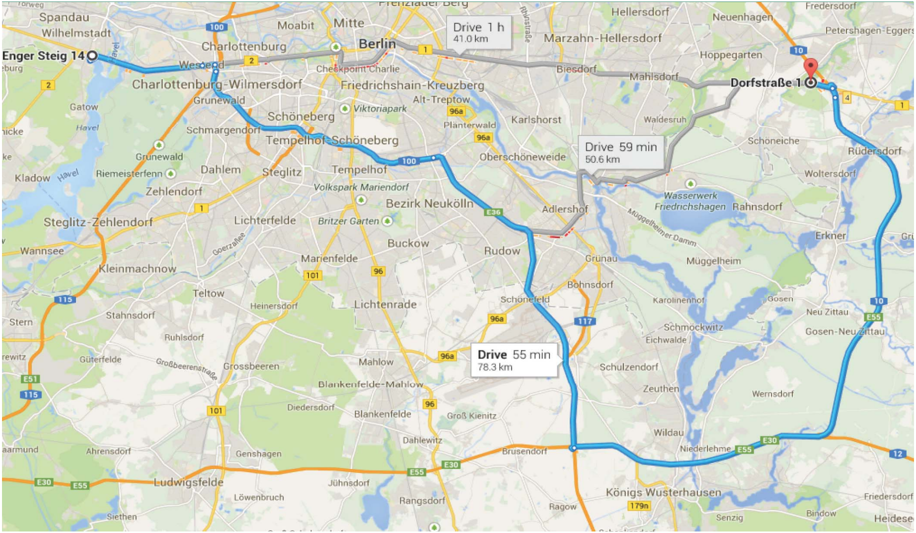
what is preprocessed	space : $\mathbb{E}[\mathcal{S}]$	preprocessing : $\mathbb{E}[\mathcal{P}]$	query : $\mathbb{E}[\mathcal{Q}_{RQA}]$
All-To-All	$O\left((K^* + 1)n^2 U\right)$	$O\left(\begin{array}{c} n^2 \log(n) \\ \cdot \log\log(K_{\max}) \\ \cdot(K^* + 1)TDP \end{array}\right)$	$O(\log\log(K^*))$
Nothing	$O(n + m + K)$	$O(1)$	$O\left(\begin{array}{c} n\log(n) \cdot \\ \log\log(K_{\max}) \end{array}\right)$
Landmarks-To-All	$O\left(\rho n^2 (K^* + 1)U\right)$	$O\left(\begin{array}{c} \rho n^2 \log(n) \\ \cdot \log\log(K_{\max}) \\ \cdot(K^* + 1)TDP \end{array}\right)$	$O\left(\begin{array}{c} \left(\frac{1}{\rho}\right)^{R+1} \cdot \log\left(\frac{1}{\rho}\right) \\ \cdot \log\log(K_{\max}) \end{array}\right)$
$K_{\max} \in O(1)$ $\rho = n^{-\alpha}$ $U, TDP \in O(1)$ $K^* \in O(\mathrm{polylog}(()n))$	$\tilde{O}\left(n^{2-\alpha}\right)$	$\tilde{O}\left(n^{2-\alpha}\right)$	$\tilde{O}\left(n^{(r+1)\cdot\alpha}\right)$

TD Distance Oracle: Towards Implementation...



Departure	**8:00 – 20:00**	**20:00 – 08:00**
Travel Time	**46.5 mins**	**45.1 mins**
FCA Response Time	**< 0.2 ms**	**< 0.2 ms**
RQA(1) Repsponse Time	**<1 ms**	**<1 ms**
P2P Dist.Summary Construction	**120 ms**	
One-To-All Dist.Summary Construction	**< 40 sec**	

TD Distance Oracle: Towards Implementation...



Related Literature

- [Dreyfus (1969)] S. E. Dreyfus. An appraisal of some shortest-path algorithms. In Operations Research, 17(3):395–412, 1969.
- [Orda-Rom (2000)] A. Orda, R. Rom. Shortest-path and minimum delay algorithms in networks with time-dependent edge-length. In J. ACM, 37(3):607–625, 1990.
- [Dean (2004)] B. C. Dean. Shortest paths in FIFO time-dependent networks: Theory and algorithms. Technical report. MIT, 2004.
- [Dehne-Omran-Sack (2010)] F. Dehne, O. T. Masoud, J. R. Sack. Shortest paths in time-dependent FIFO networks. In Algorithmica, 62(1-2):416–435, 2012.
- [Foschini-Hershberger-Suri (2011)] L. Foschini, J. Hershberger, S. Suri. On the complexity of time-dependent shortest paths. In Algorithmica, 68(4), pp. 1075–1097, 2014. Prelim. version in SODA 2011.
- [Kontogiannis-Zaroliagis (2013)] 14. S. Kontogiannis, C. Zaroliagis. Distance oracles for time dependent networks. In Automata, Languages and Programming (ICALP-A 2014), Springer, 2014.

Any Questions?

Next Lecture
Time-Dependent Speed-up Techniques
Wednesday, July 2 2014, 11:30