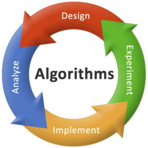
Algorithms for Route Planning

KIT (SS 2016)

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



Christos Zaroliagis

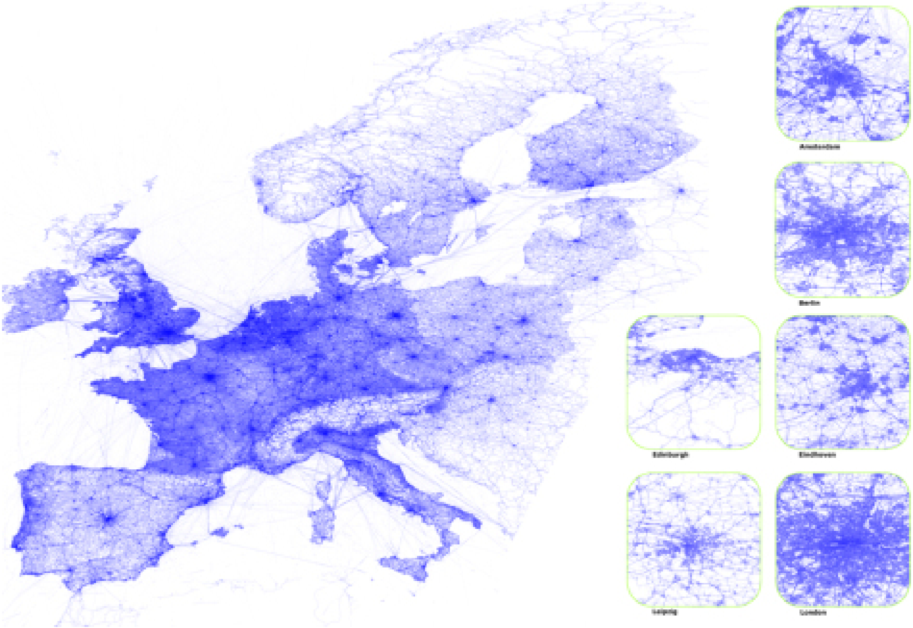
zaro@ceid.upatras.gr

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

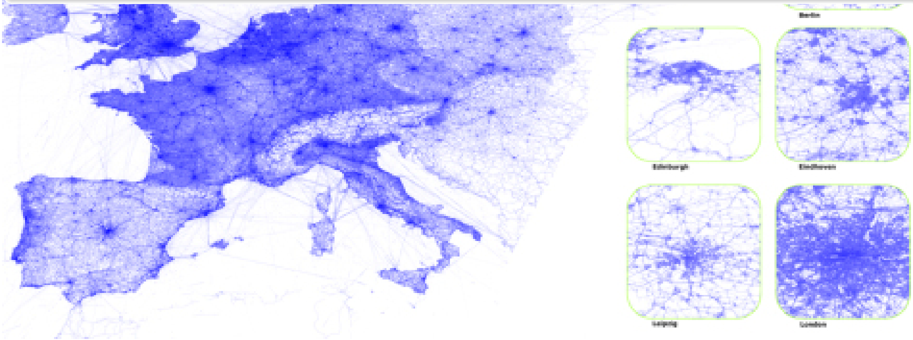
Dept. of Computer Engineering & Informatics
University of Patras, Greece

CTI "DIOPHANTUS" computer technology institute & press

Computer Technology Institute & Press
"Diophantus"

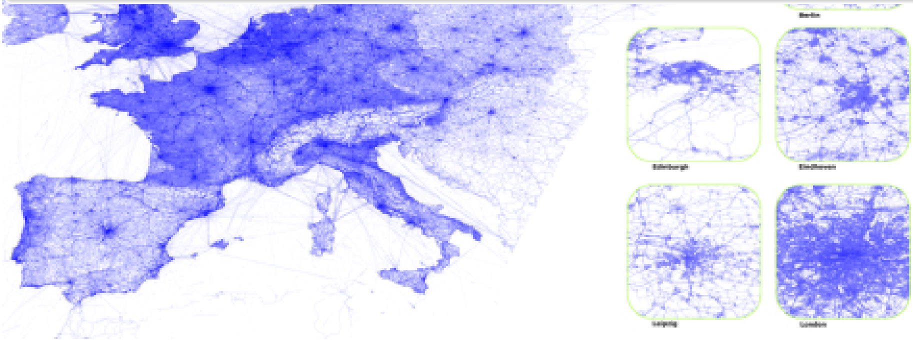


Raw traffic (speed probe) data



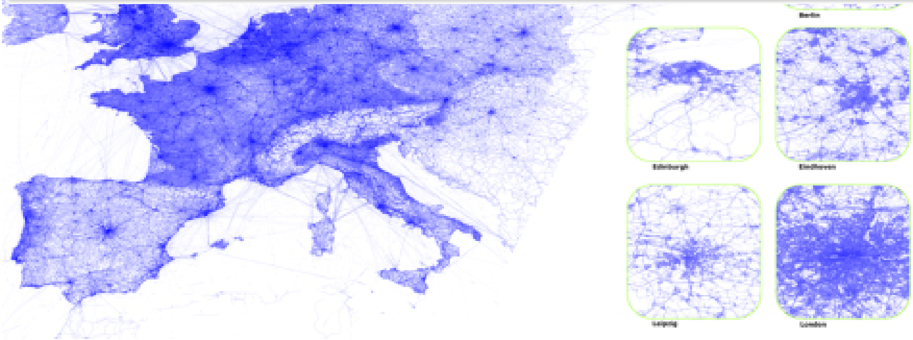
Raw traffic (speed probe) data

- **70 Million** contributing users



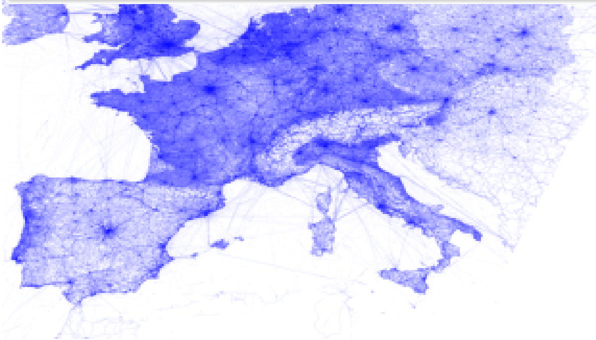
Raw traffic (speed probe) data

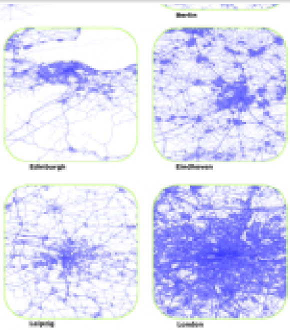
- **70 Million** contributing users
- **4 Billion** measurements per day



Raw traffic (speed probe) data

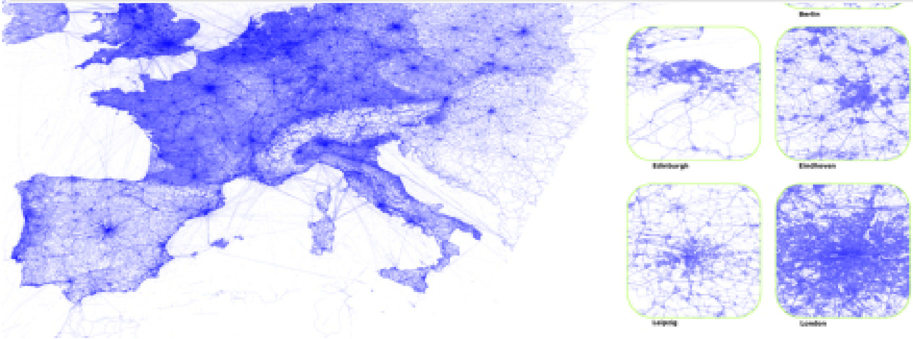
- **70 Million** contributing users
- **4 Billion** measurements per day
- **5 Trillion** measurements over *140 Billion Km*





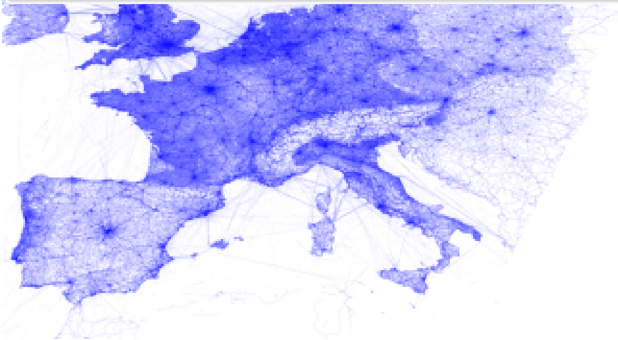
Raw traffic (speed probe) data

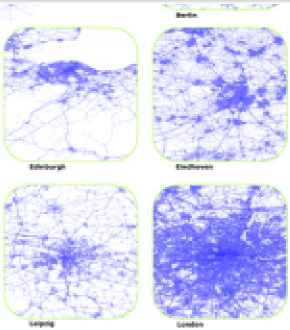
- **70 Million** contributing users
- **4 Billion** measurements per day
- **5 Trillion** measurements over *140 Billion Km*
- every road segment measured 2000 times on average



Raw traffic (speed probe) data

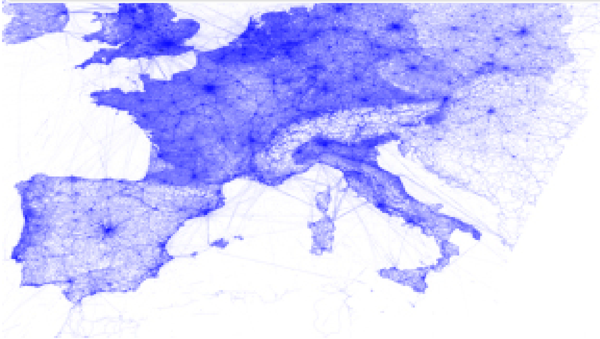
- **70 Million** contributing users
- **4 Billion** measurements per day
- **5 Trillion** measurements over *140 Billion Km*
- every road segment measured 2000 times on average
- measured speeds in 5-min intervals

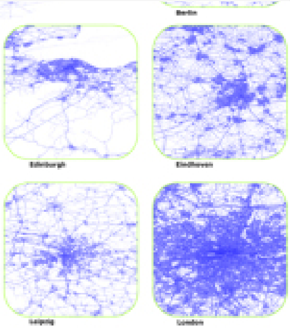




Raw traffic (speed probe) data

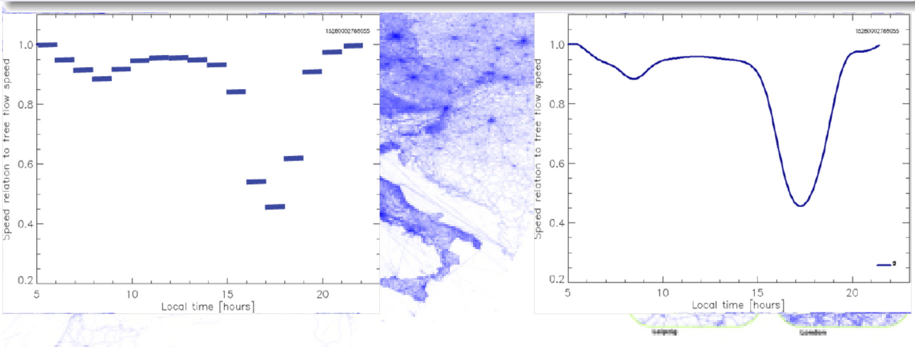
- **70 Million** contributing users
- **4 Billion** measurements per day
- **5 Trillion** measurements over *140 Billion Km*
- every road segment measured 2000 times on average
- measured speeds in 5-min intervals





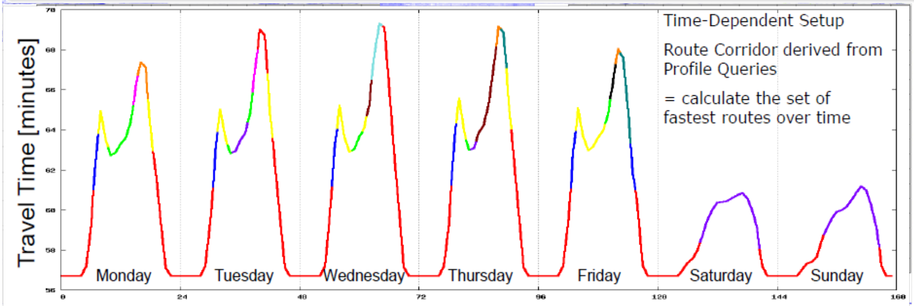
Raw traffic (speed probe) data

- **70 Million** contributing users
- **4 Billion** measurements per day
- **5 Trillion** measurements over *140 Billion Km*
- every road segment measured 2000 times on average
- measured speeds in 5-min intervals



Raw traffic (speed probe) data

- **70 Million** contributing users
- **4 Billion** measurements per day
- **5 Trillion** measurements over *140 Billion Km*
- every road segment measured 2000 times on average
- measured speeds in 5-min intervals



Main Issue: time-dependence

Time Dependent Shortest Paths – I

a more realistic and more involved problem

Why Time-Dependent Shortest Paths?

Real-life networks: Elements demonstrate temporal behavior

Why Time-Dependent Shortest Paths?

Real-life networks: Elements demonstrate temporal behavior

- Graph elements **added/removed** in real-time /∗ Dynamic Shortest Path ∗/
- Metric demonstrates **stochastic behavior** /∗ Stochastic 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 demonstrate temporal behavior

- Graph is **fixed**, metric **changes over time** in a predetermined fashion

  /* Time-Dependent Shortest Path */

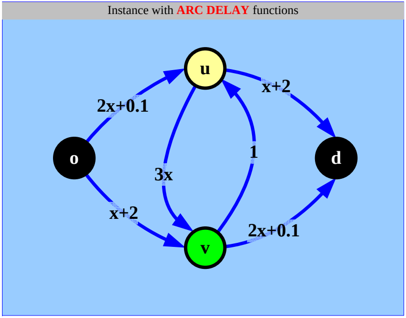
Why Time-Dependent Shortest Paths?

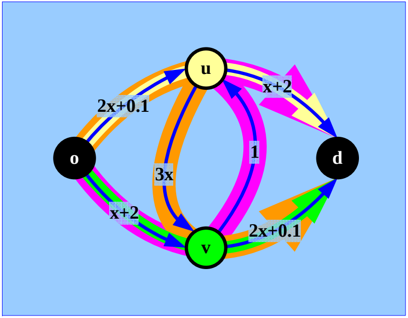
Real-life networks: Elements demonstrate temporal behavior

- Graph is **fixed**, metric **changes over time** in a predetermined fashion

  /∗ Time-Dependent Shortest Path ∗/

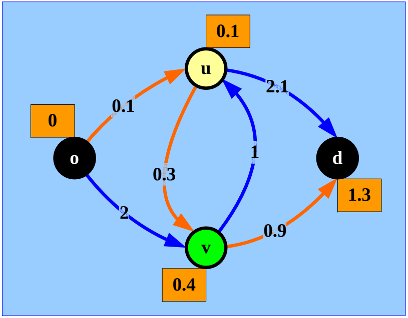
  - Arcs are allowed to become **occasionally unavailable** (e.g., due to periodic maintenance, saving consumption of resources, etc), for predetermined unavailability time-intervals (discrete domain)
  - Arc lengths (e.g., traversal-time / consumption) **change with departure-time from tail** which is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

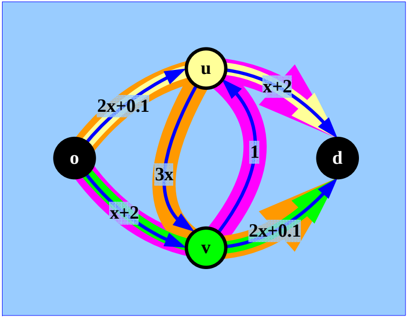




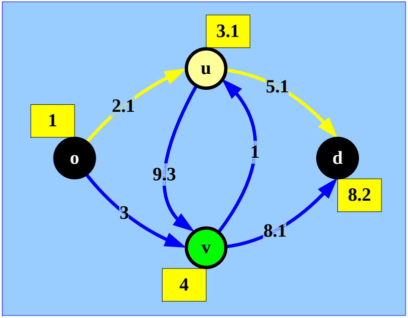
Q1 How would you commute **as fast as possible** from *o* to *d*, for a given departure time (from *o*)?

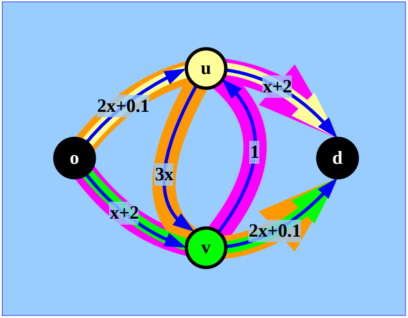
TDSP :: EXAMPLE 1 (Earliest Arrivals)



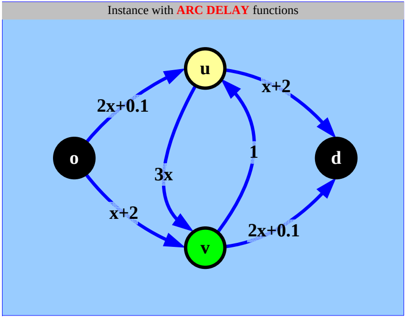


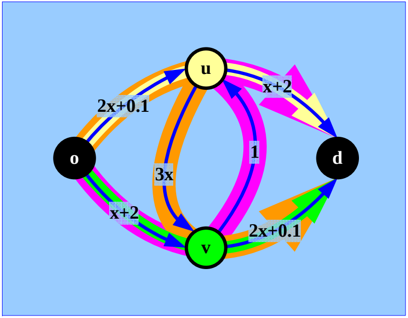
Q1 How would you commute **as fast as possible** from *o* to *d*, for a given departure time (from *o*)? Eg: $\boxed{t_o = 0}$





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

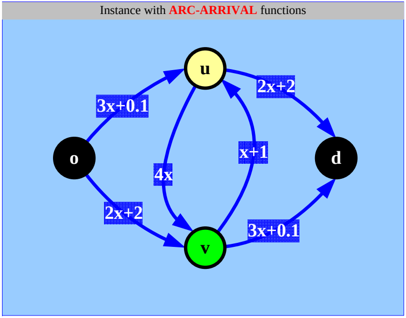


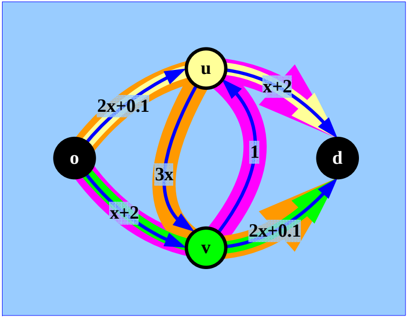


Q1 How would you commute **as fast as possible** from o to d, for a given departure time (from o)?

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

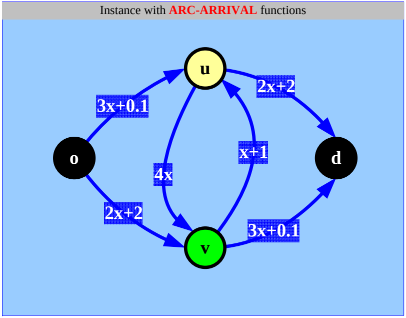
TDSP :: EXAMPLE 1 (Earliest Arrivals)

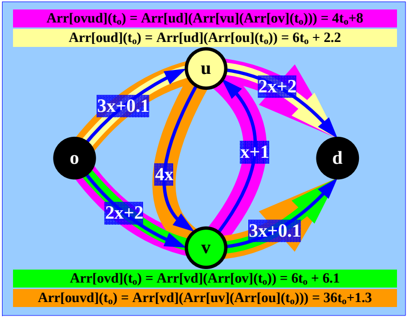




Q1 How would you commute **as fast as possible** from *o* to *d*, for a given departure time (from *o*)?

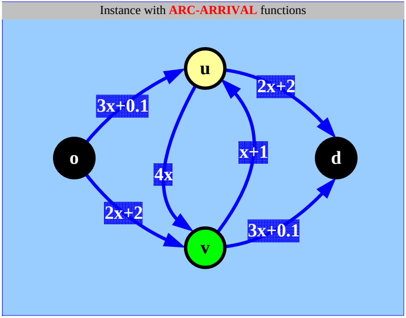
Q2 What if you are not sure about the departure time?

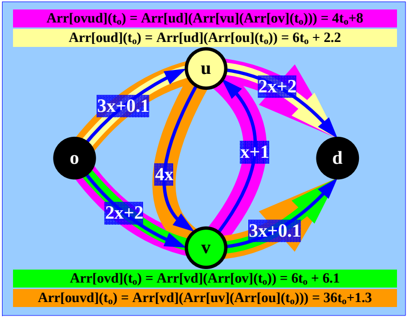




Q1 How would you commute **as fast as possible** from *o* to *d*, for a given departure time (from *o*)?

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



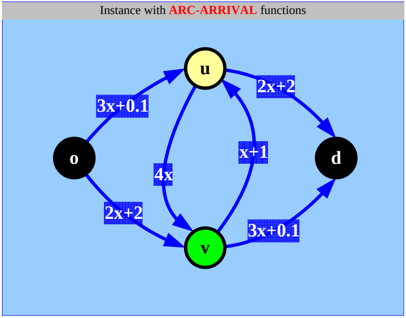


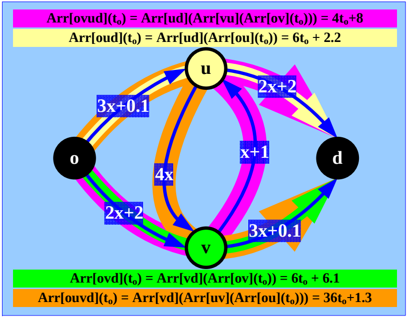
Q1 How would you commute **as fast as possible** from *o* to *d*, for a given departure time (from *o*)?

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

A

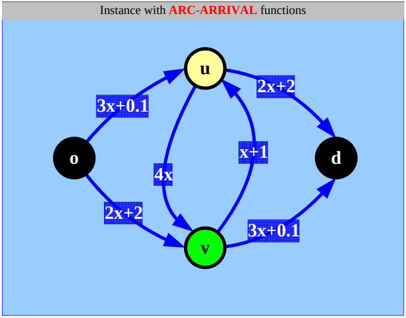
$$\text{shortest } od\text{–path} = \begin{cases} \textbf{orange path}, \text{ if} & t_o \in [0, 0.03] \\ \textbf{yellow path}, \text{ if} & t_o \in [0.03, 2.9] \\ \textbf{purple path}, \text{ if} & t_o \in [2.9, +\infty) \end{cases}$$

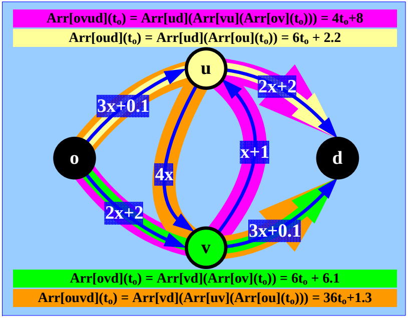




Q1 Would **waiting-at-nodes** be worth it?

TDSP :: EXAMPLE 2 (Waiting Times)

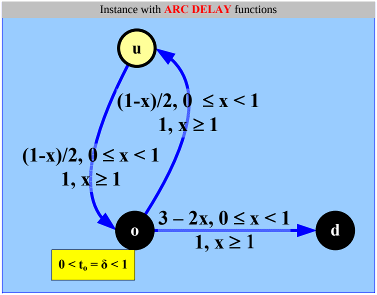




Q1 Would **waiting-at-nodes** be worth it?

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

TDSP :: EXAMPLE 2 (Waiting Times)

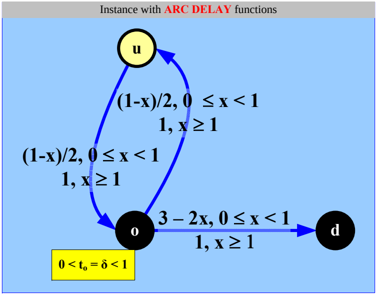


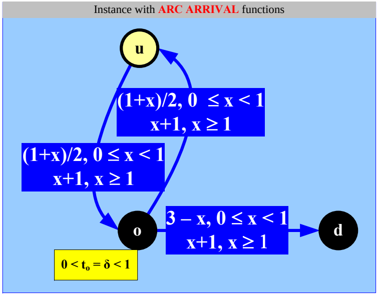
Q1 Would **waiting-at-nodes** be worth it?

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

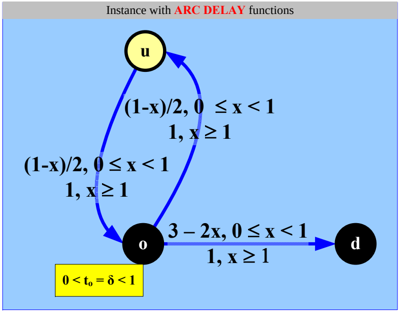
Unrestricted Waiting (UW) Unlimited waiting is allowed at every node along an *od*-path

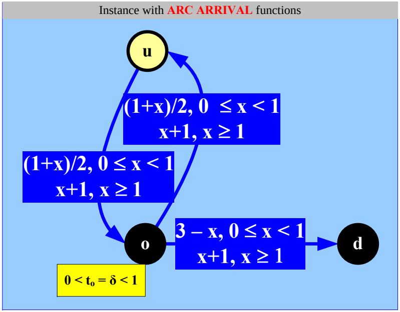
Origin Waiting (OW) Unlimited waiting is only allowed at the origin node of each *od*-path

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 (ensuring earliest arrival at destination), but also for the appropriate optimal waiting times at the nodes along this path

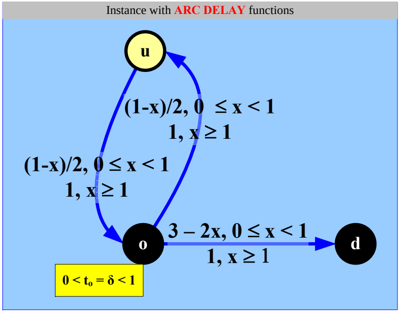
TDSP :: EXAMPLE 2 (Waiting Times) – contd.

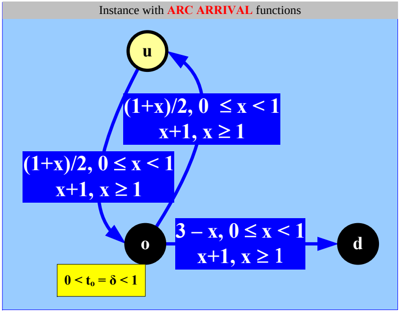




Q3 What if **waiting-at-nodes** is forbidden?

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



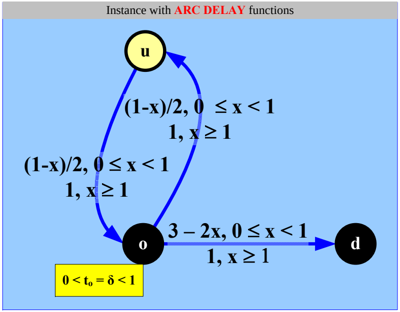


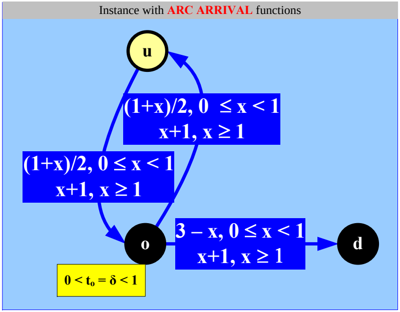
Q3 What if waiting-at-nodes is forbidden?

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

o					d
δ					$3 - \delta$

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



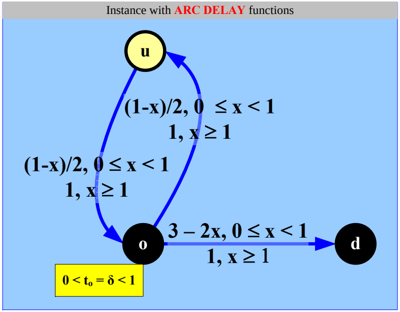


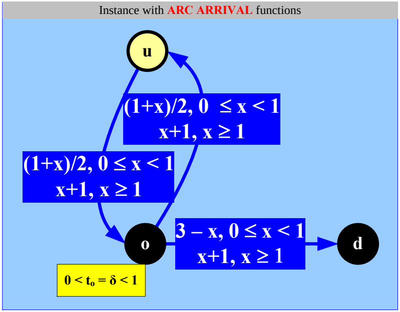
Q3 What if waiting-at-nodes is forbidden?

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

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

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



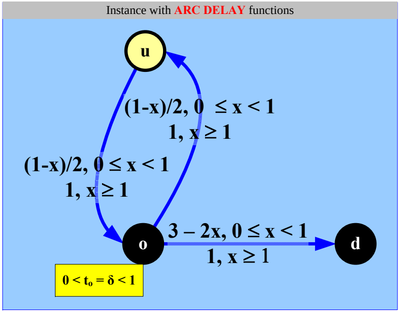


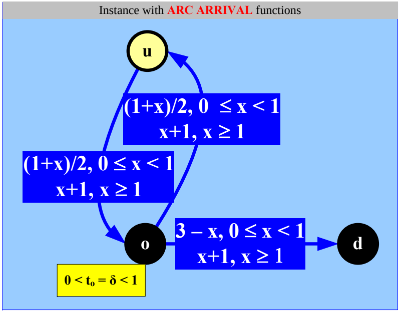
Q3 What if **waiting-at-nodes** is forbidden?

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

o	u	o	u	o	d
δ	$\frac{1+\delta}{2}$	$\frac{3+\delta}{4}$	$\frac{7+\delta}{8}$	$\frac{15+\delta}{16}$	$3 - \frac{15+\delta}{16} > 2$

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

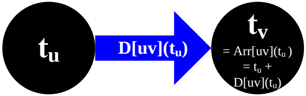
o	u	o	presence at o after $k \uparrow \infty$ visits of u	d
δ	$\frac{1+\delta}{2}$	$\frac{3+\delta}{4}$	$\lim_{k\uparrow\infty} \frac{2^{2k}-1+\delta}{2^{2k}} = 1$	$t_d \downarrow 2$

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

FIFO vs non-FIFO Arc Delays

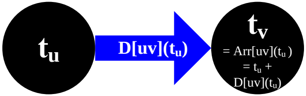
- **(Strict) FIFO Arc-Delays:** The slopes of all the *arc-delay* functions are at least equal to (greater than) -1

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



- **(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)

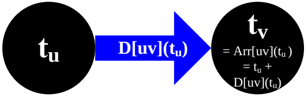


- **Non-FIFO Arc-Delays:** Possibly preferable to wait for some period at the tail of an arc, before trespassing it. E.g.:
  - Wait for the next *(faster) IC train*, than use the (immediately available) *(slower) local train*

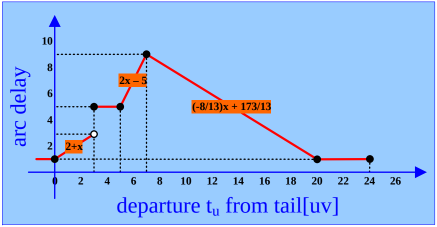
FIFO vs non-FIFO Arc Delays

- **(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 preferable 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*

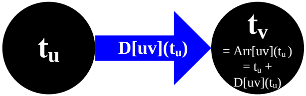


FIFO arc delay example

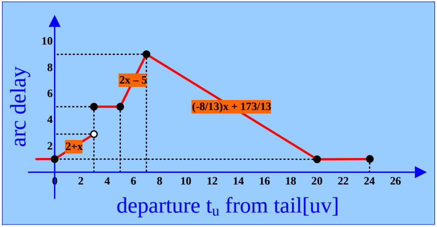
FIFO vs non-FIFO Arc Delays

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

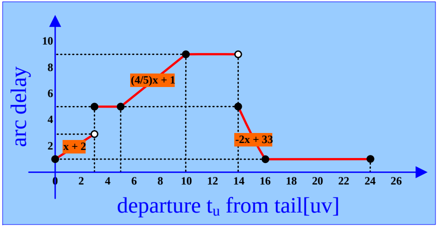
Equivalently: *Arc-arrival* functions are **non-decreasing** (aka **no-overtaking** property)



- **Non-FIFO Arc-Delays:** Possibly preferable 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*

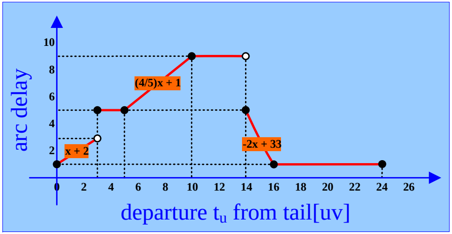


FIFO arc delay example

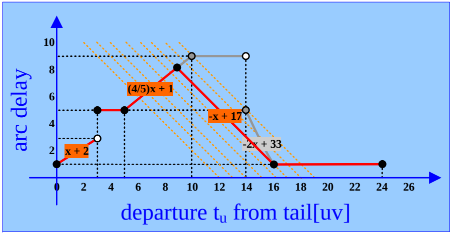


Non-FIFO arc delay example

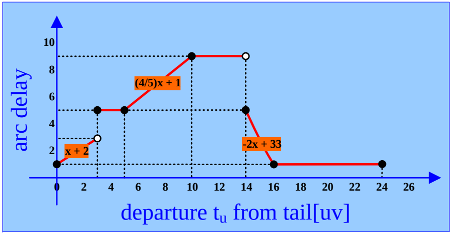
Non-FIFO+UW Arc ⇔ FIFO Arc



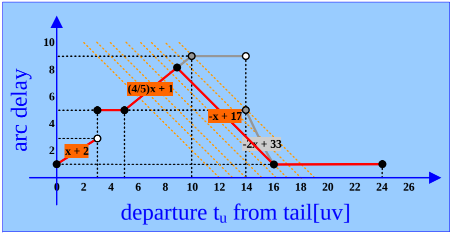
Non-FIFO+UW arc delay function



Equivalent FIFO (+FW) arc delay function

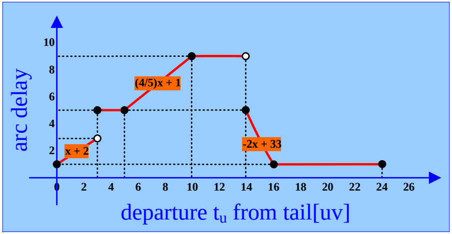


Non-FIFO+UW arc delay function

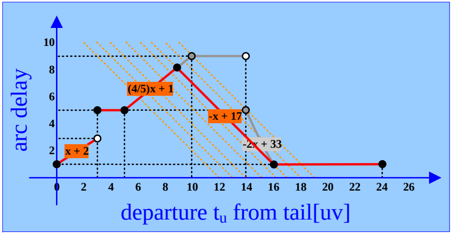


Equivalent FIFO (+FW) arc delay function

- Sweeping a line with slope -1 *from right to left* suffices



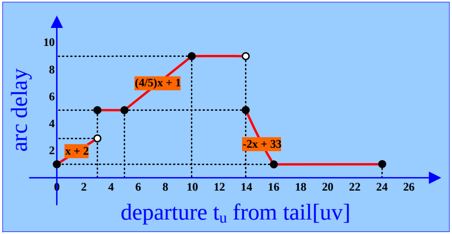
Non-FIFO+UW arc delay function



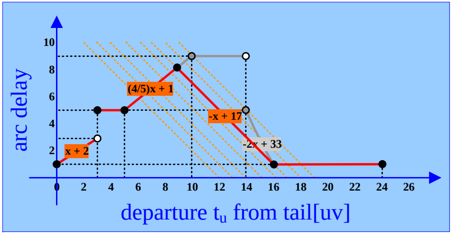
Equivalent FIFO (+FW) arc delay function

- Sweeping a 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 Arc ⇔ FIFO Arc



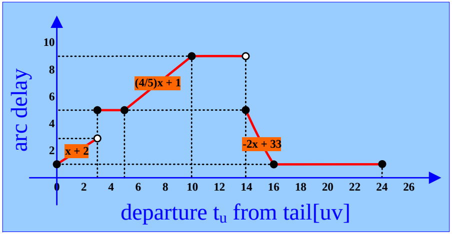
Non-FIFO+UW arc delay function



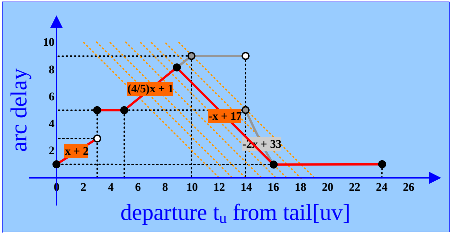
Equivalent FIFO (+FW) arc delay function

- Sweeping a 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 Arc ⇔ FIFO Arc



Non-FIFO+UW arc delay function



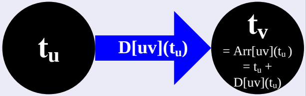
Equivalent FIFO (+FW) arc delay function

- Sweeping a 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

Variants of Time-Dependent Shortest Path

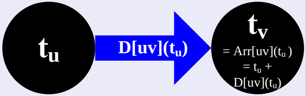
Time-Dependent Shortest Path

- *Directed* graph $G = (V, A)$ with **arc-travel-time** function $D[uv](t)$, $\forall uv \in A$
  $Arr[uv](t) = t + D[uv](t)$



Variants of Time-Dependent Shortest Path

Time-Dependent Shortest Path



- *Directed* graph $G = (V, A)$ with **arc-travel-time** function $D[uv](t)$, $\forall uv \in A$
  $Arr[uv](t) = t + D[uv](t)$

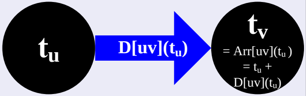
- $P_{o,d}$: od-paths; $p = (a_1, \ldots, a_k) \in P_{o,d}$
- **Path arrival** / **travel-time** functions:
  $Arr[p](t) = Arr[a_k] \circ \cdots \circ Arr[a_1](t)$ (*composition*)
  $D[p](t) = Arr[p](t) - t$

- **Earliest-arrival** / **Shortest-travel-time** functions:
  $Arr[o, d](t) = \min_{p \in P_{o,d}} \{ Arr[p](t) \}$
  $D[o, d](t) = Arr[o, d](t) - t$

Variants of Time-Dependent Shortest Path

Time-Dependent Shortest Path

- *Directed* graph $G = (V, A)$ with **arc-travel-time** function $D[uv](t)$, $\forall uv \in A$
  $Arr[uv](t) = t + D[uv](t)$



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

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** (theory: in sublinear time; practice: in micro/milliseconds) to arbitrary queries in *large-scale nets*, for arbitrary departure-times and od–pairs.

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** (theory: in sublinear time; practice: in micro/milliseconds) to arbitrary queries in *large-scale nets*, for arbitrary departure-times and od–pairs.

- **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** (theory: in sublinear time; practice: in micro/milliseconds) to arbitrary queries in *large-scale nets*, for arbitrary departure-times and od–pairs.

- **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 **FIFO** networks.

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 **FIFO** networks.
- [Orda-Rom (1990)] Prefix-subpath optimality does NOT hold in **non-FIFO+FW** networks (cf. EXAMPLE of Slide 7).

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 **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: $\forall t_u, D[uv](t_u) \geq \lim_{t \downarrow t_u} D[uv](t))$

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 **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 (2014)] In **strict-FIFO** networks, (general) subpath optimality holds also in the time-dependent case.

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

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 **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 (2014)] 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: $\forall t_u, D[uv](t_u) \geq \lim_{t \downarrow t_u} D[uv](t))$

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. 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. 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. 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** in the number of “elementary” *functional operations*. i.e., (EVAL, LINEAR COMBINATION, MIN, COMPOSITION)
    - ☹ Not so “elementary” operations after all (see next slides)!

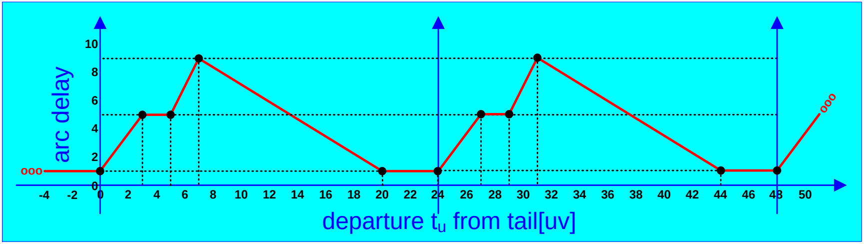
Algorithms for TDSP

... in FIFO, continuous, pwl instances

Input/Output Data

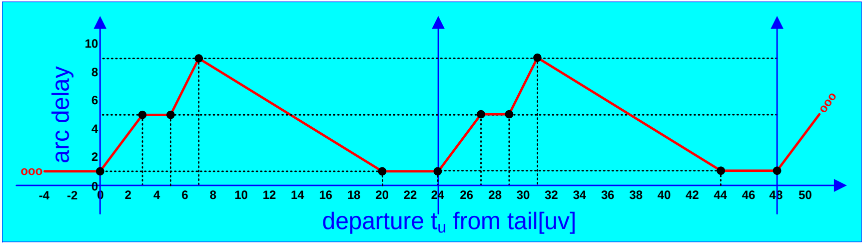
PWL Arc Delays

Forward Description (as function of departure times from origin)

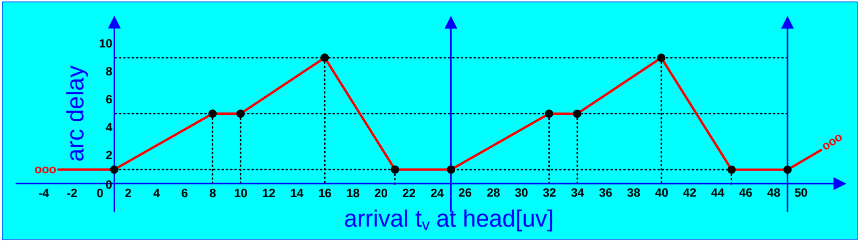


PWL Arc Delays

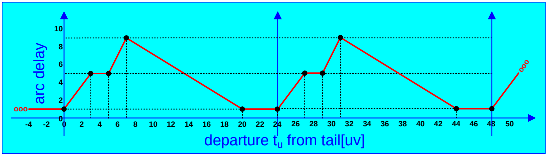
Forward Description (as function of departure times from origin)



Reverse Description (as function of arrival times at destination)



How to Store/Access PWL Arc Delays



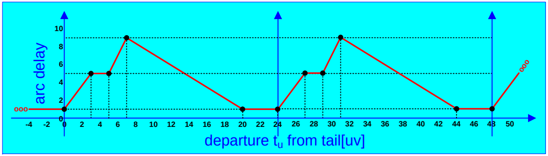
- Exploit *periodicity* and *piecewise-linearity*:

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

- Representation: Array of **(slope, constant, dep.time UB) triples** equipped with advanced (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



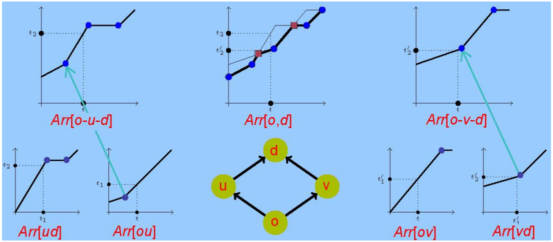
- Exploit *periodicity* and *piecewise-linearity*:

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

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

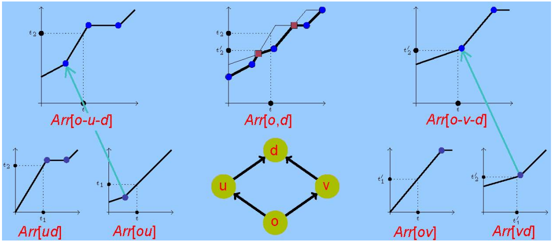
$(0,1)$	$(3,5)$	$(5,5)$	$(7,9)$	$(20,1)$

Piecewise Linearity of Path / Earliest Arrivals



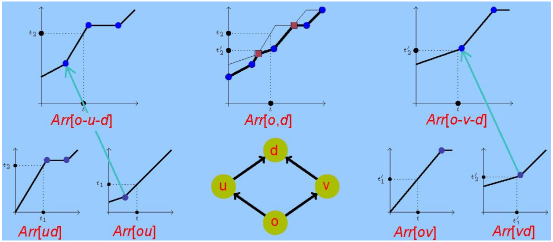
- **Primitive Breakpoint (PB):** Departure-time b'_e from $head[e]$ at which $D[e]$ changes slope (assume $K \in O(m)$ PBs in total)

Piecewise Linearity of Path / Earliest Arrivals



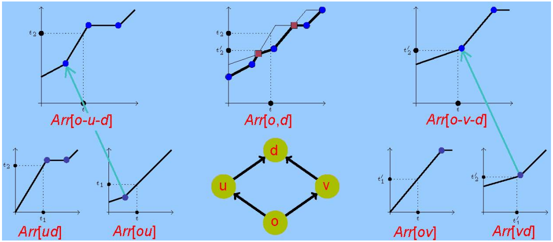
- **Primitive Breakpoint (PB):** Departure-time b'_e from $head[e]$ at which $D[e]$ changes slope (assume $K \in \mathrm{O}(m)$ PBs in total)
- **Primitive Image (PI):** **Latest departure-time** b_e from origin o s.t. earliest-arrival-time $b'_e = Arr[o, tail(e)](b_e)$ coincides with a breakpoint for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

Piecewise Linearity of Path / Earliest Arrivals



- **Primitive Breakpoint (PB):** Departure-time b'_e from $head[e]$ at which $D[e]$ changes slope (assume $K \in \mathcal{O}(m)$ 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 $K \in \mathrm{O}(m)$ 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 **very sparse** 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 **very sparse** graphs

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

(Exact) Output Sensitive Algorithm for Earliest-Arrival Functions

Why Do We Need an Output Sensitive Algorithm?

Why Do We Need an 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 an 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 an 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

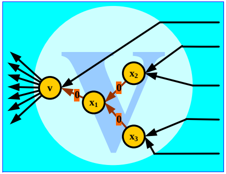
2. We may need to compute *exact distance summaries* for **special pairs of vertices** (eg, from/to hubs, all superhub-to-superhub connections, etc)

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)

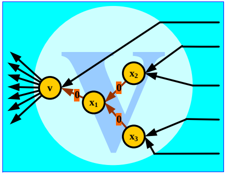
The Output-Sensitive Algorithm (I)

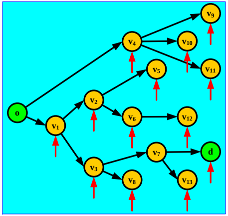
- ASSUMPTION: The in-degree of every node in the graph is at most 2



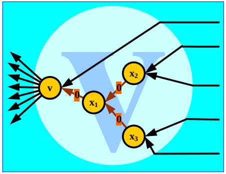
The Output-Sensitive Algorithm (I)

- 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**

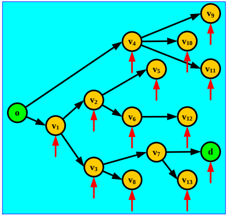




The Output-Sensitive Algorithm (I)

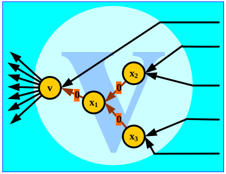


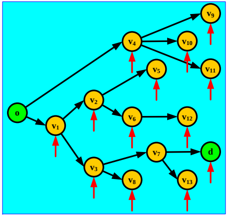
- ASSUMPTION: The in-degree of every node in the graph is at most 2
- Given an arbitrary point in time (*"current time"*) $t_0 \geq 0$ as departure time from origin o, compute a **TDSP tree**
- Discover *until when* the TDSP tree is **valid**:
  - $\forall v \in V$, two short alternatives when departing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

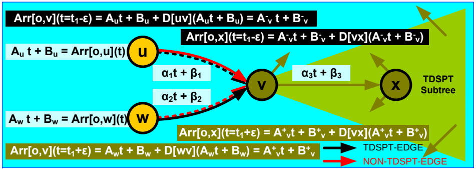
- ASSUMPTION: The in-degree of every node in the graph is at most 2
- Given an arbitrary point in time (*“current time”*) $t_0 \geq 0$ as departure time from origin o, compute a **TDSP tree**
- Discover *until when* the TDSP tree is **valid**:
  - $\forall v \in V$, two short alternatives when departing 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 priority queue:



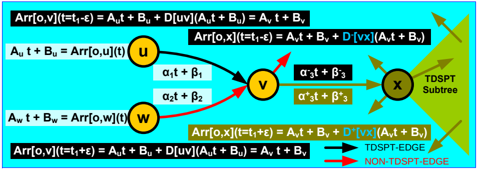
if minimization-certificate failure, at node $v \in V$:

then (1) Update shortest ov–path

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

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

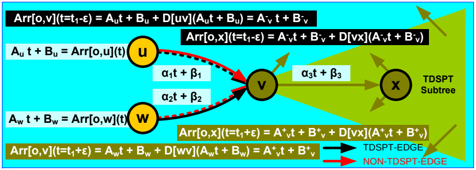
(3) Update $t_{fail}[e]$, $\forall e \in E : x = tail[e] \in T_v$



The Output-Sensitive Algorithm (II)

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

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

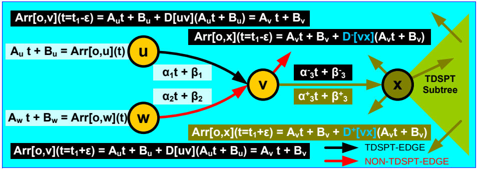


then (1) Update shortest ov–path

/* **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): $\mathrm{O}(|E_c| \cdot \log n)$. 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
- *Worst-case time-complexity* of output-sensitive algorithm:

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