Temporal graphs

Laurent Viennot

MPRI – Theory of practical graph algorithms

Temporal graphs

Definition : a graph that changes with time.

Examples :

- Mobile and sensor networks.
- Social networks (co-athorship, contact traces).
- Transportation networks (buses, road).

Formal treatment still in its infancy :

- natural notions (connectivity),
- temporal generalization of classical notions ? (Menger)

Temporal graphs

Definition : a graph that changes with time.

Examples :

- Mobile and sensor networks.
- Social networks (co-athorship, contact traces).
- Transportation networks (buses, road).

Formal treatment still in its infancy :

- natural notions (connectivity),
- temporal generalization of classical notions ? (Menger)

Temporal graphs

Definition : a graph that changes with time.

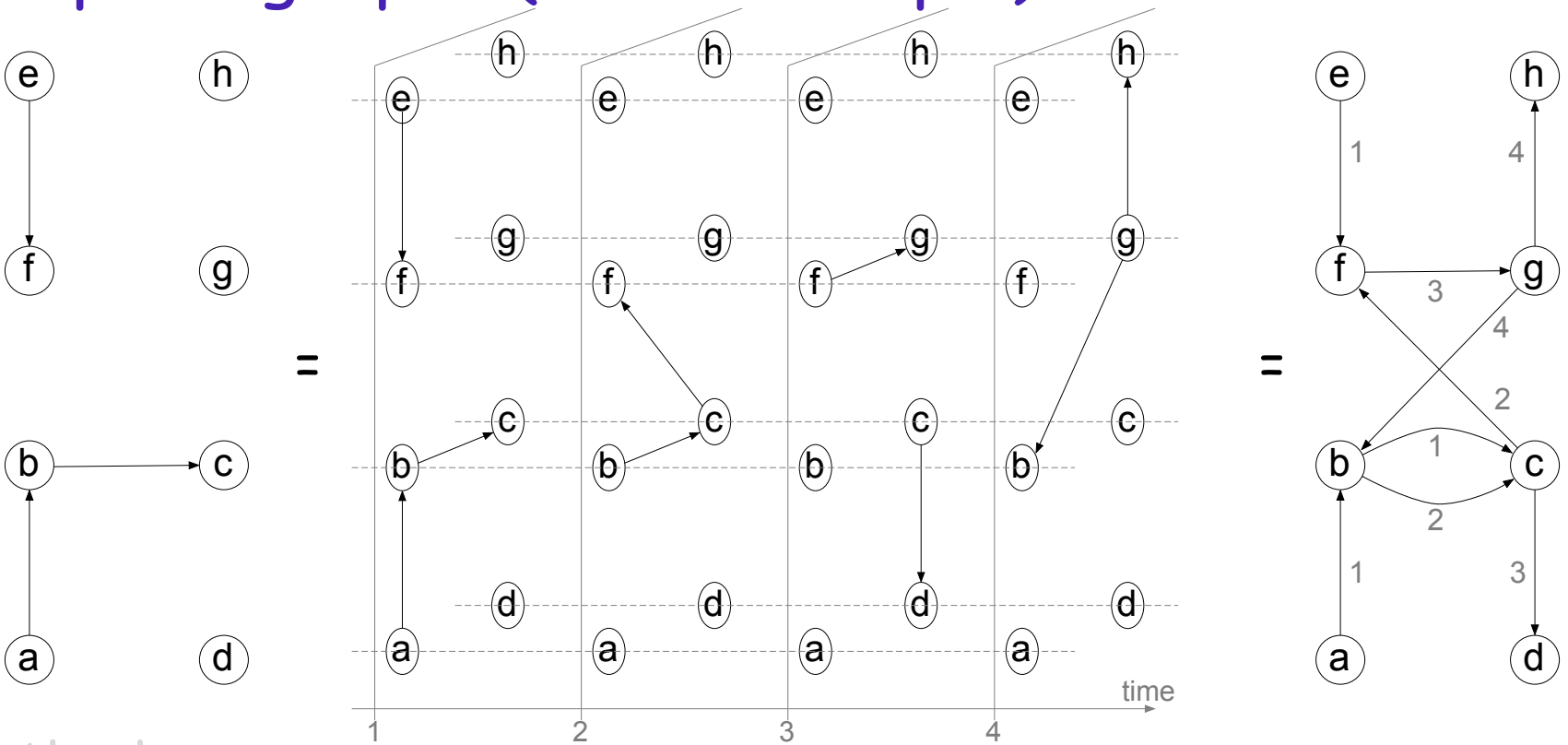
Examples :

- Mobile and sensor networks.
- Social networks (co-athorship, contact traces).
- Transportation networks (buses, road).

Formal treatment still in its infancy :

- natural notions (connectivity),
- temporal generalization of classical notions? (Menger)

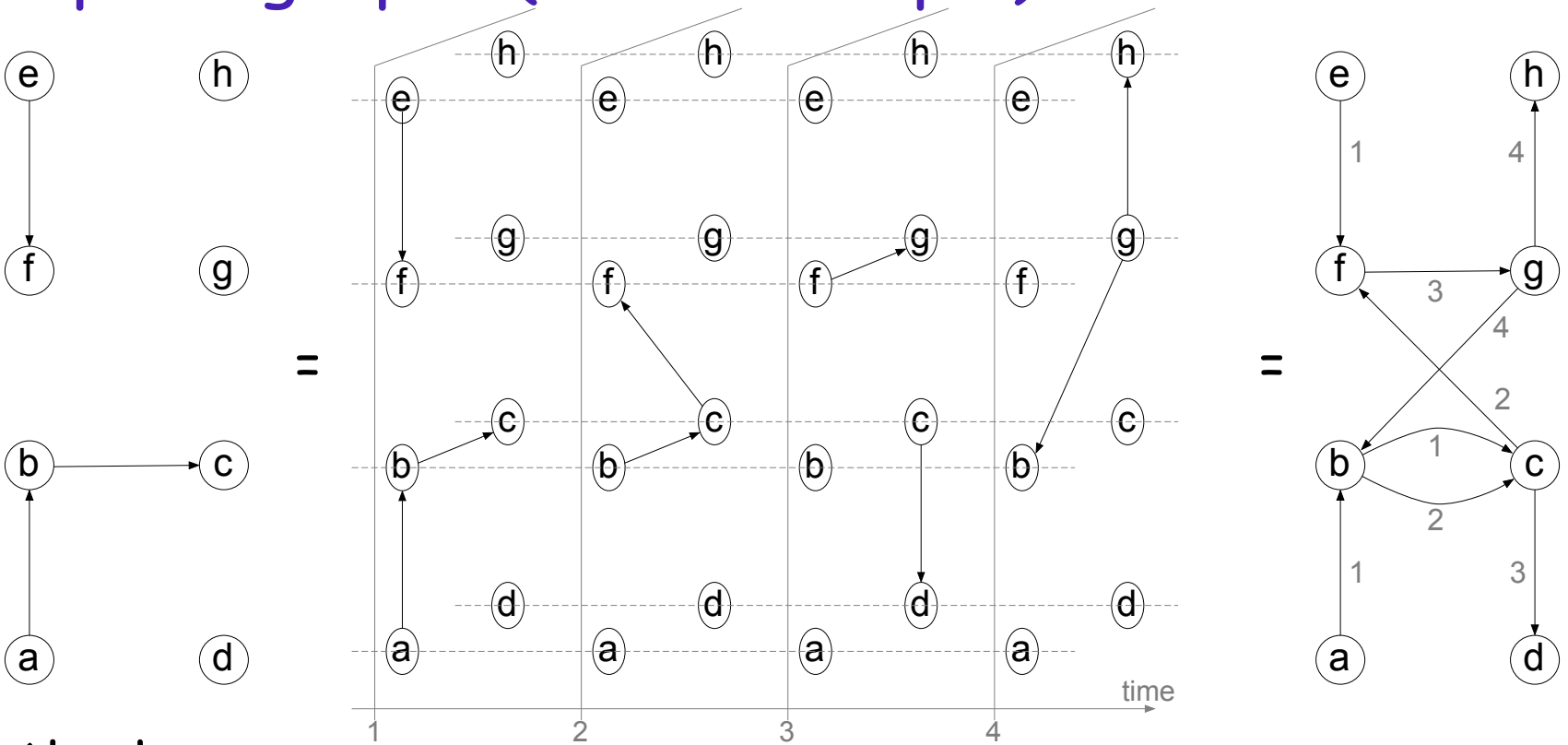
Temporal graphs (basic example)



Also known as :

- time-dependent networks [Cooke, Halsey 1966],
- edge scheduled networks [Berman 1996], dynamic graphs [Harary, Gupta 1997], temporal networks [Kempe, Kleinberg, Kumar 2002], evolving graphs [Bhadra, Ferreira 2003],
- time-varying graphs [Casteigts, Flocchini, Quattrociocchi, Santoro 2012],
- link streams [Latapy, Viard, Magnien 2018],...

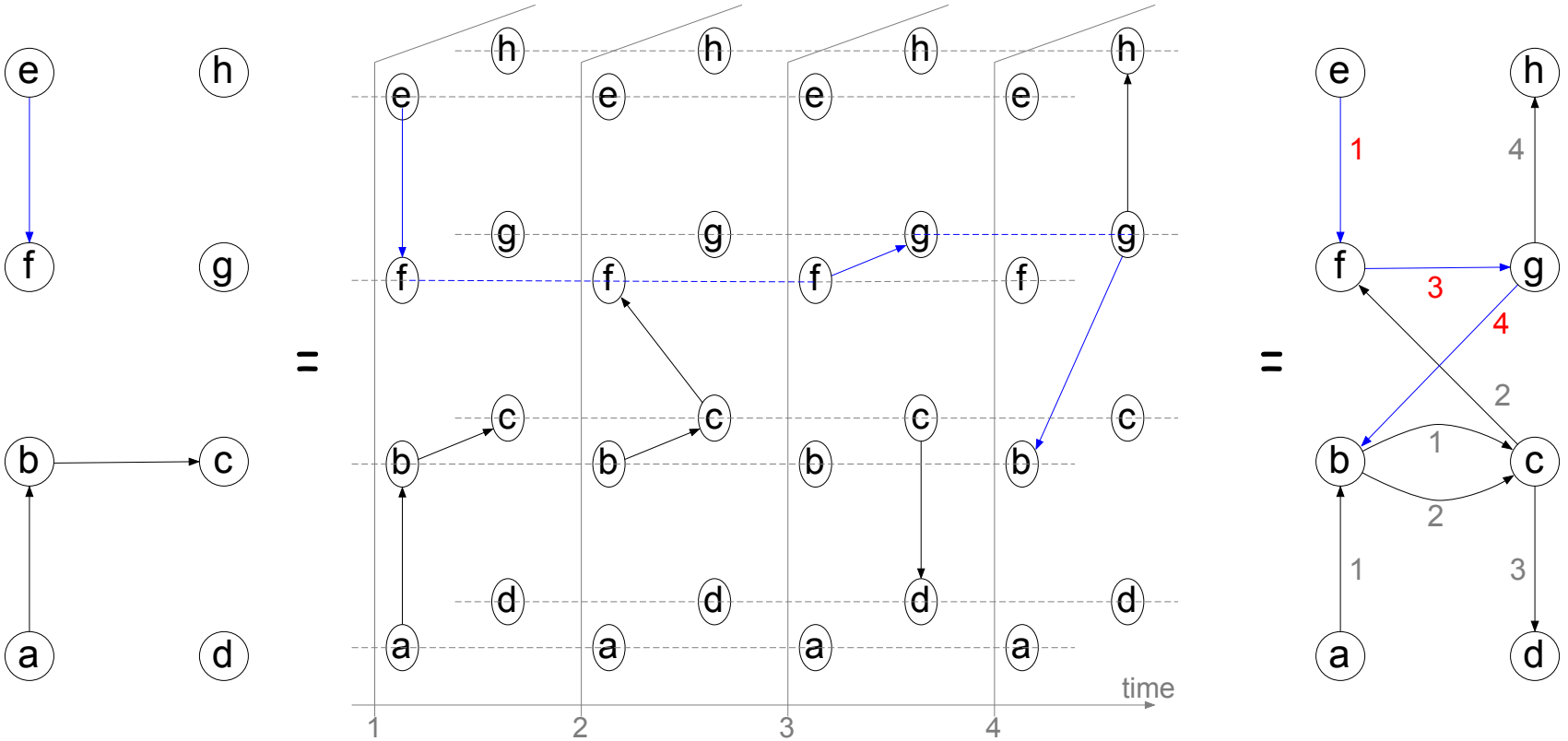
Temporal graphs (basic example)



Also known as :

- time-dependent networks [Cooke, Halsey 1966],
- edge scheduled networks [Berman 1996], dynamic graphs [Harary, Gupta 1997], temporal networks [Kempe, Kleinberg, Kumar 2002], evolving graphs [Bhadra, Ferreira 2003],
- time-varying graphs [Casteigts, Flocchini, Quattrociocchi, Santoro 2012],
- link streams [Latapy, Viard, Magnien 2018],...

Temporal path/walk



Strict temporal walk : a walk with increasing time labels.

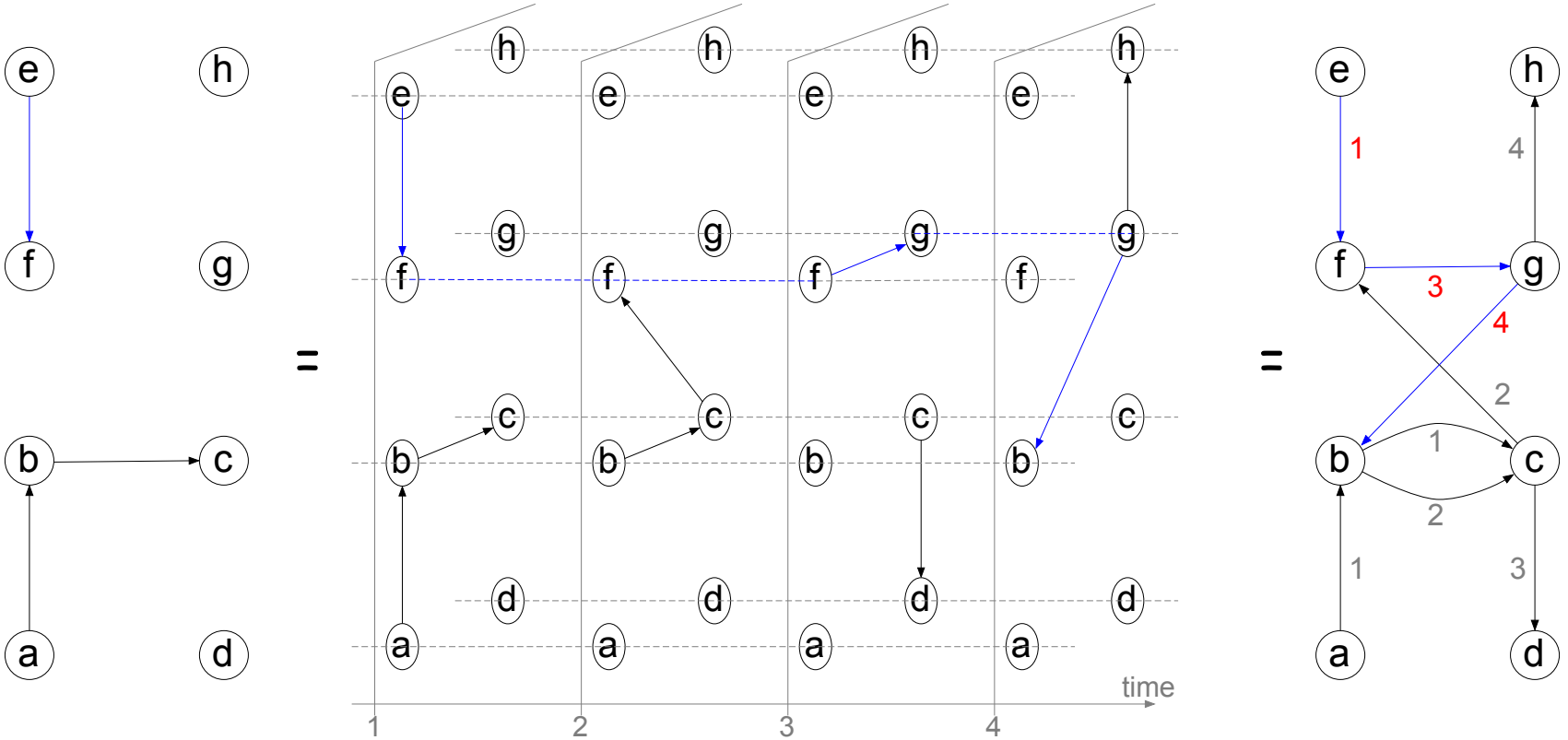
Non-strict temp. walk : a walk with non-decreasing time labels.

A path is walk visiting at most once a vertex.

Usually waiting is allowed.

Above, we have : $e \longrightarrow b, b \longrightarrow h, \neg a \longrightarrow f, \neg a \longrightarrow h$

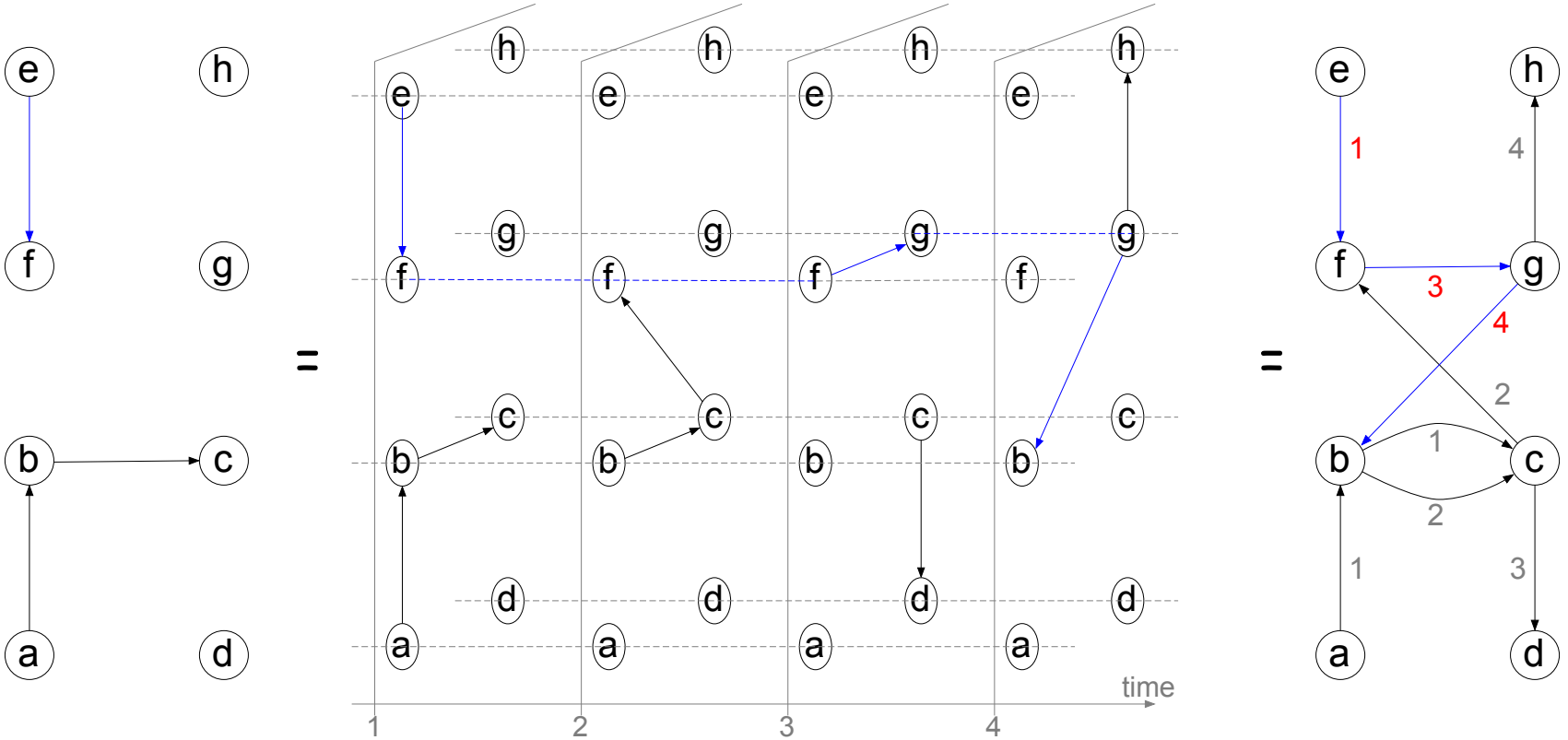
Temporal path/walk



Strict temporal walk : a walk with increasing time labels.
Non-strict temp. walk : a walk with non-decreasing time labels.

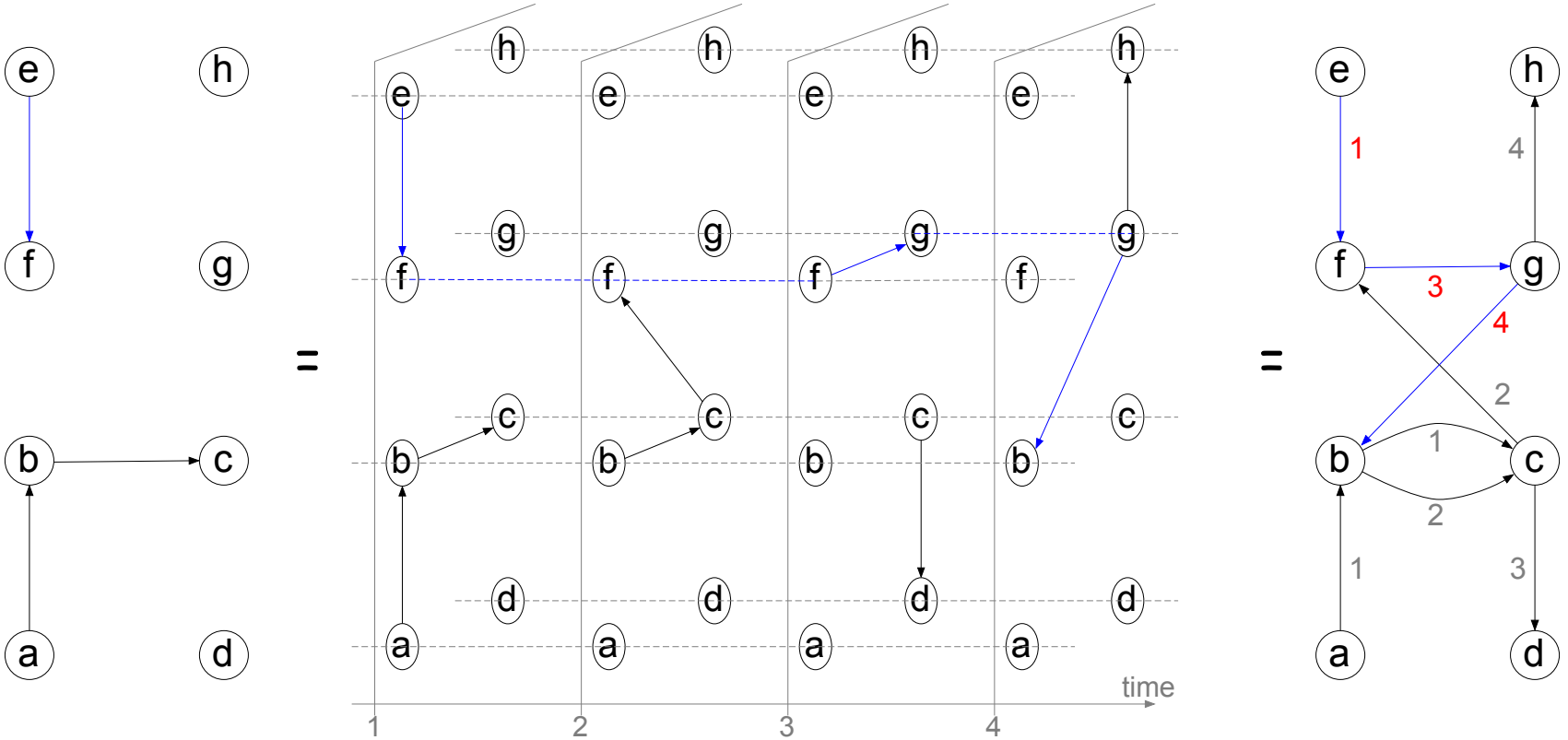
A path is walk visiting at most once a vertex.
Usually waiting is allowed.
Above, we have : $e \longrightarrow b, b \longrightarrow h, \neg a \longrightarrow f, \neg a \longrightarrow h$

Temporal path/walk



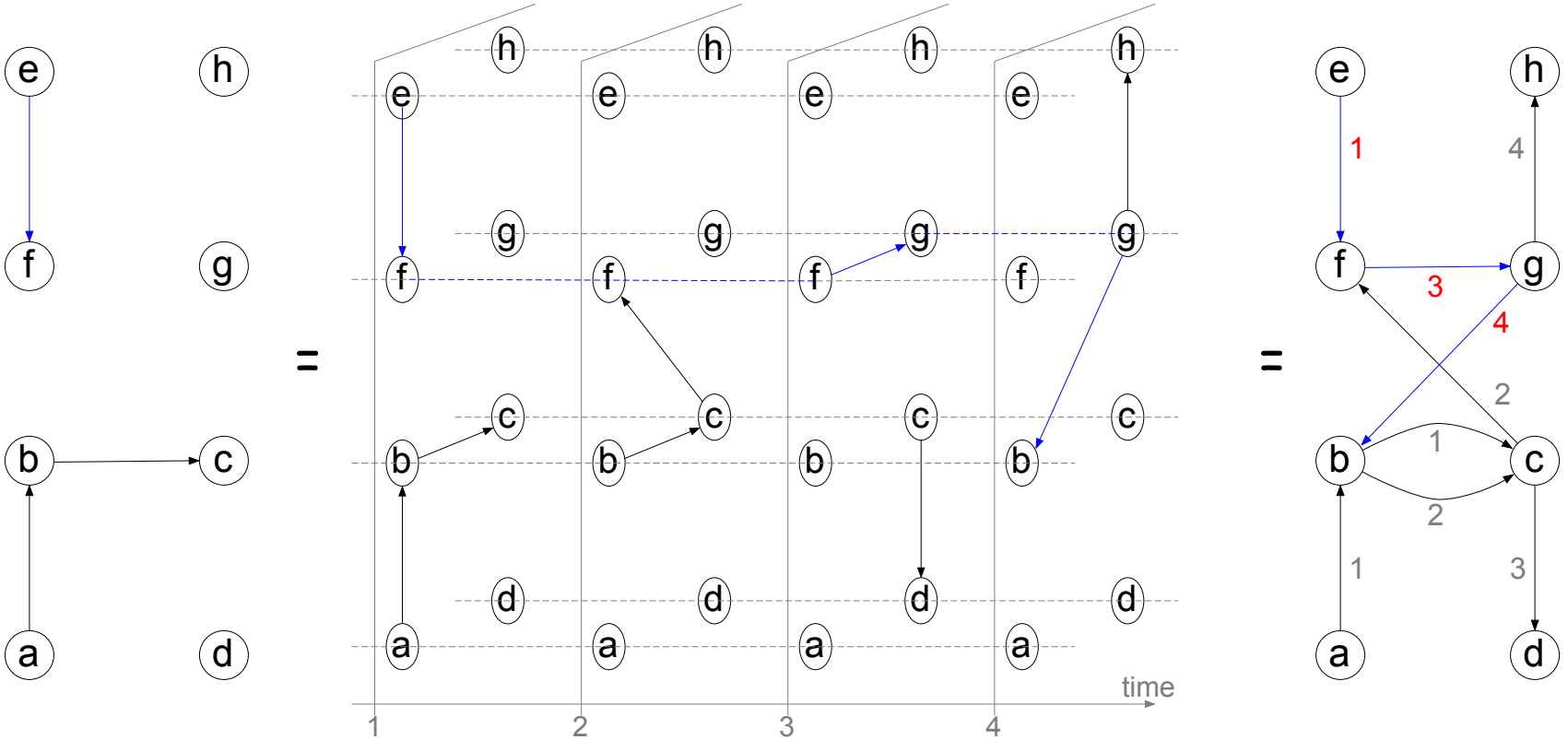
Strict temporal walk : a walk with increasing time labels.
Non-strict temp. walk : a walk with non-decreasing time labels.
A path is walk visiting at most once a vertex.
Usually waiting is allowed.
Above, we have : $e \longrightarrow b, b \longrightarrow h, \neg a \longrightarrow f, \neg a \longrightarrow h$

Temporal path/walk



Strict temporal walk : a walk with increasing time labels.
Non-strict temp. walk : a walk with non-decreasing time labels.
A path is walk visiting at most once a vertex.
Usually waiting is allowed.
Above, we have : $e \longrightarrow b, b \longrightarrow h, \neg a \longrightarrow f, \neg a \longrightarrow h$

Temporal path/walk



Strict temporal walk : a walk with increasing time labels.
Non-strict temp. walk : a walk with non-decreasing time labels.
A path is walk visiting at most once a vertex.
Usually waiting is allowed.
Above, we have : $e \longrightarrow b, b \longrightarrow h, \neg a \longrightarrow f, \neg a \longrightarrow h$

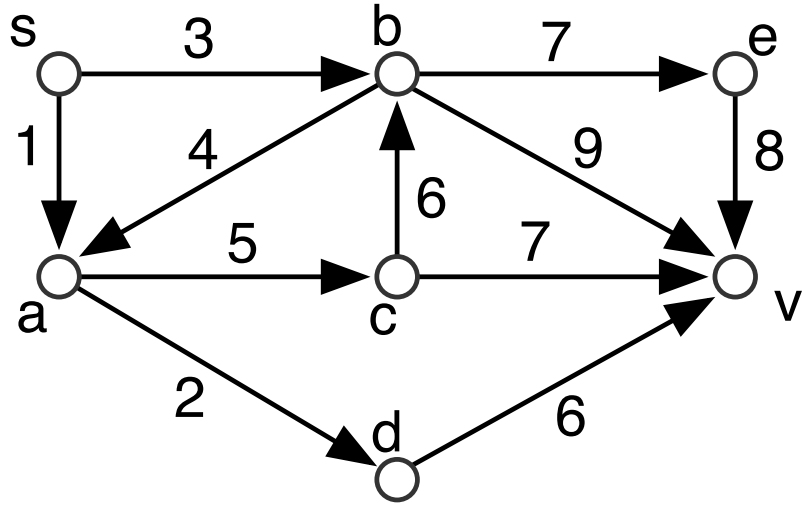
Various natural "shortest" uv-walks

Earliest arrival (or foremost) : starting at τ from u, minimize arrival time in v.

Fewest hops (or shortest) : minimize the number k of edges.

Shortest duration (or fastest) : minimize arrival time - departure time.

Min. waiting time : minimize the sum of waiting times at intermediate nodes.



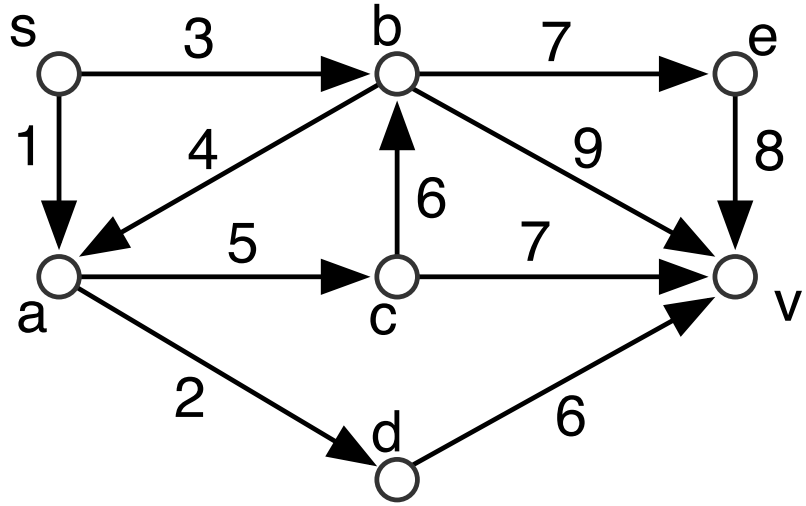
Various natural "shortest" uv-walks

Earliest arrival (or foremost) : starting at τ from u, minimize arrival time in v.

Fewest hops (or shortest) : minimize the number k of edges.

Shortest duration (or fastest) : minimize arrival time - departure time.

Min. waiting time : minimize the sum of waiting times at intermediate nodes.



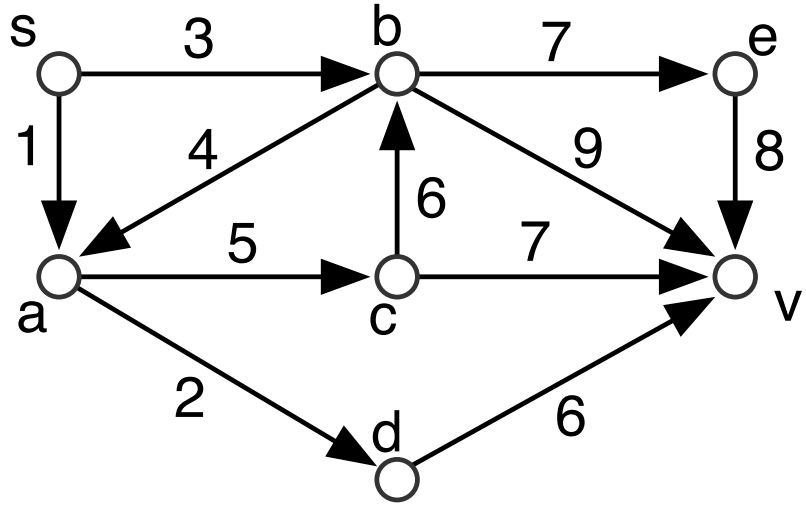
Various natural "shortest" uv-walks

Earliest arrival (or foremost) : starting at τ from u, minimize arrival time in v.

Fewest hops (or shortest) : minimize the number k of edges.

Shortest duration (or fastest) : minimize arrival time - departure time.

Min. waiting time : minimize the sum of waiting times at intermediate nodes.



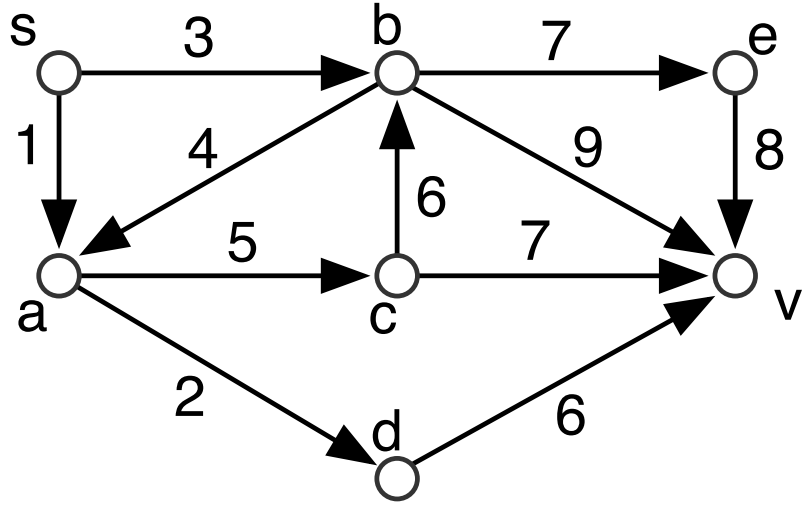
Various natural "shortest" uv-walks

Earliest arrival (or foremost) : starting at τ from u, minimize arrival time in v.

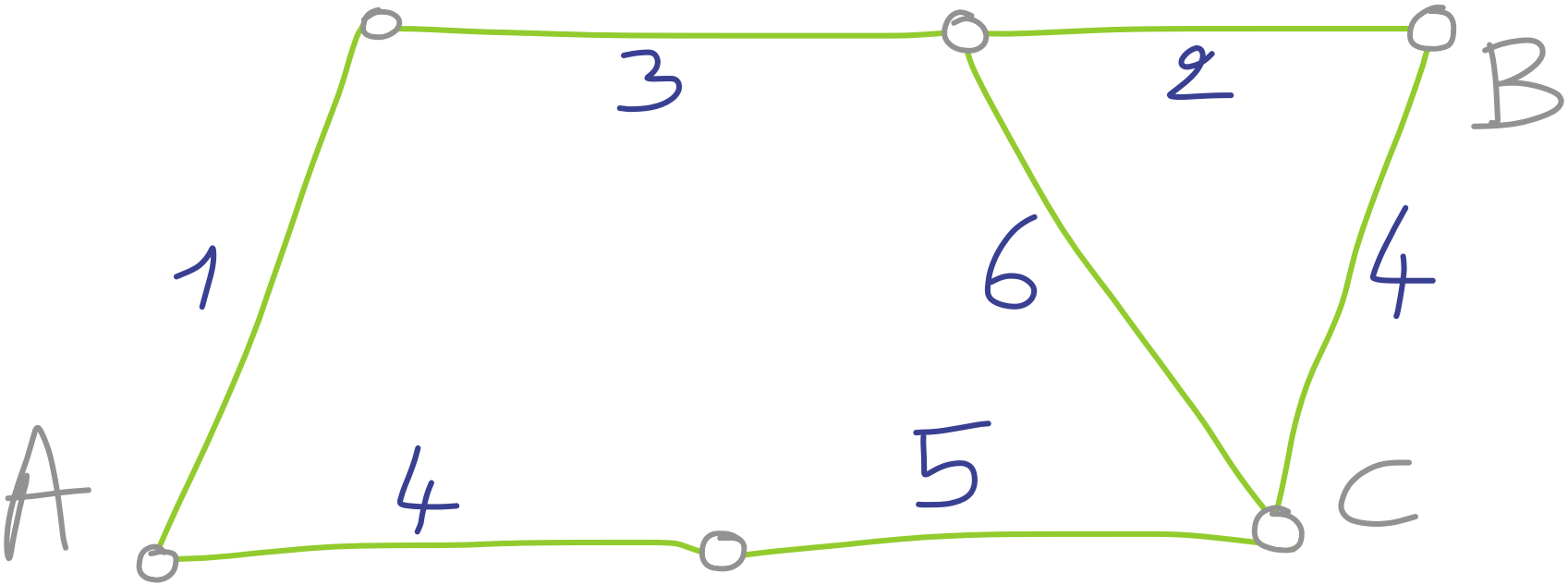
Fewest hops (or shortest) : minimize the number k of edges.

Shortest duration (or fastest) : minimize arrival time - departure time.

Min. waiting time : minimize the sum of waiting times at intermediate nodes.



Undirected temporal graphs still have time orientation!



Path between A and B?

Path between A and C? between C and A?

Main models

Time-dependent network : a graph where edge delay depends on time. [Cooke, Halsey 1966]
$\mathcal{G} = ((V, E), \delta)$ with $\delta : E \to \mathbb{R}^{\mathbb{R}}$ ($\delta(e)(\tau)$ is the delay of $e \in E$ at time τ).

Pice-wise constant-delay : each $\delta(e)$ is picewise constant. [Bui-Xuan, Ferreira, Jarry 2003], [Dehne, Omran, Sack 2012], (link-stream [Latapy, Viard, Magnien 2018] with delay 0)

Evolving graph : sequence of (static) graphs with same vertices.
$\mathcal{G} = (V, E_1, \ldots, E_\ell)$ [Bhadra, Ferreira 2003] or equivalently $\mathcal{G} = ((E, V), \lambda)$ with $\lambda : E \to 2^{\mathbb{N}}$ [Michail 2016]. (Delay of edges is 1/0 for strict/non-strict temporal paths.)

Simple : $\mathcal{G} = ((E, V), \lambda)$ with $\lambda : E \to \mathbb{N}$ [Kempe, Kleinberg, Kumar 2002].

Proper : edges incident to a node have pairwise disjoint labels (strict and non-strict then coincide) [Casteigts, Corsini, Sarkar 2022].

Globally proper : edge schedule.

Main models

Time-dependent network : a graph where edge delay depends on time. [Cooke, Halsey 1966]
$\mathcal{G} = ((V, E), \delta)$ with $\delta : E \to \mathbb{R}^{\mathbb{R}}$ ($\delta(e)(\tau)$ is the delay of $e \in E$ at time τ).
Pice-wise constant-delay : each $\delta(e)$ is picewise constant. [Bui-Xuan, Ferreira, Jarry 2003], [Dehne, Omran, Sack 2012], (link-stream [Latapy, Viard, Magnien 2018] with delay 0)
Evolving graph : sequence of (static) graphs with same vertices.
$\mathcal{G} = (V, E_1, \ldots, E_\ell)$ [Bhadra, Ferreira 2003] or equivalently $\mathcal{G} = ((E, V), \lambda)$ with $\lambda : E \to 2^{\mathbb{N}}$ [Michail 2016]. (Delay of edges is 1/0 for strict/non-strict temporal paths.)
Simple : $\mathcal{G} = ((E, V), \lambda)$ with $\lambda : E \to \mathbb{N}$ [Kempe, Kleinberg, Kumar 2002].
Proper : edges incident to a node have pairwise disjoint labels (strict and non-strict then coincide) [Casteigts, Corsini, Sarkar 2022].
Globally proper : edge schedule.

Main models

Time-dependent network : a graph where edge delay depends on time. [Cooke, Halsey 1966]
$\mathcal{G} = ((V, E), \delta)$ with $\delta : E \to \mathbb{R}^{\mathbb{R}}$ ($\delta(e)(\tau)$ is the delay of $e \in E$ at time τ).
Pice-wise constant-delay : each $\delta(e)$ is picewise constant. [Bui-Xuan, Ferreira, Jarry 2003], [Dehne, Omran, Sack 2012], (link-stream [Latapy, Viard, Magnien 2018] with delay 0)
Evolving graph : sequence of (static) graphs with same vertices.
$\mathcal{G} = (V, E_1, \ldots, E_\ell)$ [Bhadra, Ferreira 2003] or equivalently $\mathcal{G} = ((E, V), \lambda)$ with $\lambda : E \to 2^{\mathbb{N}}$ [Michail 2016]. (Delay of edges is 1/0 for strict/non-strict temporal paths.)
Simple : $\mathcal{G} = ((E, V), \lambda)$ with $\lambda : E \to \mathbb{N}$ [Kempe, Kleinberg, Kumar 2002].
Proper : edges incident to a node have pairwise disjoint labels (strict and non-strict then coincide) [Casteigts, Corsini, Sarkar 2022].
Globally proper : edge schedule.

Main models

Time-dependent network : a graph where edge delay depends on time. [Cooke, Halsey 1966]
$\mathcal{G} = ((V, E), \delta)$ with $\delta : E \to \mathbb{R}^{\mathbb{R}}$ ($\delta(e)(\tau)$ is the delay of $e \in E$ at time τ).
Pice-wise constant-delay : each $\delta(e)$ is picewise constant. [Bui-Xuan, Ferreira, Jarry 2003], [Dehne, Omran, Sack 2012], (link-stream [Latapy, Viard, Magnien 2018] with delay 0)
Evolving graph : sequence of (static) graphs with same vertices.
$\mathcal{G} = (V, E_1, \ldots, E_\ell)$ [Bhadra, Ferreira 2003] or equivalently $\mathcal{G} = ((E, V), \lambda)$ with $\lambda : E \to 2^{\mathbb{N}}$ [Michail 2016]. (Delay of edges is 1/0 for strict/non-strict temporal paths.)
Simple : $\mathcal{G} = ((E, V), \lambda)$ with $\lambda : E \to \mathbb{N}$ [Kempe, Kleinberg, Kumar 2002].
Proper : edges incident to a node have pairwise disjoint labels (strict and non-strict then coincide) [Casteigts, Corsini, Sarkar 2022].
Globally proper : edge schedule.

What is time?

Time domain : $\mathbb{N}$ or $\mathbb{R}$ or $\mathbb{T}$?

Discrete :

- time is discrete,
- and/or edges are available at some given points in time,
- and/or traversal takes time 1 (or 0).

Continous :

- time is continuous,
- and/or edges are available during given intervals of time,
- and/or traversal takes constant time.

What is time?

Time domain : $\mathbb{N}$ or $\mathbb{R}$ or $\mathbb{T}$?

Discrete :

- time is discrete,
- and/or edges are available at some given points in time,
- and/or traversal takes time 1 (or 0).

Continous :

- time is continuous,
- and/or edges are available during given intervals of time,
- and/or traversal takes constant time.

What is time?

Time domain : $\mathbb{N}$ or $\mathbb{R}$ or $\mathbb{T}$?

Discrete :

- time is discrete,
- and/or edges are available at some given points in time,
- and/or traversal takes time 1 (or 0).

Continous :

- time is continuous,
- and/or edges are available during given intervals of time,
- and/or traversal takes constant time.

What is time?

Time domain : $\mathbb{N}$ or $\mathbb{R}$ or $\mathbb{T}$?

Discrete :

- time is discrete,
- and/or edges are available at some given points in time,
- and/or traversal takes time 1 (or 0).

Continous :

- time is continuous,
- and/or edges are available during given intervals of time,
- and/or traversal takes constant time.

Data structures

Adjacency lists : each u has for each neighbor v a representation of $\delta(uv)$, typically a sorted list of availability times with associated delays.

(Sorted) list of temporal edges.

Time-expanded graph.

Time tables (public transit networks) : for all edges on a line, the sequence of departure and arrival times, plus footpaths.

Data structures

Adjacency lists : each u has for each neighbor v a representation of $\delta(uv)$, typically a sorted list of availability times with associated delays.

(Sorted) list of temporal edges.

Time-expanded graph.

Time tables (public transit networks) : for all edges on a line, the sequence of departure and arrival times, plus footpaths.

Data structures

Adjacency lists : each u has for each neighbor v a representation of $\delta(uv)$, typically a sorted list of availability times with associated delays.

(Sorted) list of temporal edges.

Time-expanded graph.

Time tables (public transit networks) : for all edges on a line, the sequence of departure and arrival times, plus footpaths.

Data structures

Adjacency lists : each u has for each neighbor v a representation of $\delta(uv)$, typically a sorted list of availability times with associated delays.

(Sorted) list of temporal edges.

Time-expanded graph.

Time tables (public transit networks) : for all edges on a line, the sequence of departure and arrival times, plus footpaths.

Data structure : temporal edges

Temporal edge (u, v, τ) : arc (u, v) at time τ (delay 1 or 0),

(with delay δ) (u, v, τ, δ) : taking arc (u, v) at time τ from u leads to v at time $\tau + \delta$,

(with availability interval $[\tau_1, \tau_2]$) $(u, v, \tau_1, \tau_2, \delta)$: arc (u, v) is available for any $\tau \in [\tau_1, \tau_2]$ with constant-delay δ,

(with affine delay) $(u, v, \tau_1, \tau_2, \gamma, \beta)$: taking arc (u, v) at $\tau \in [\tau_1, \tau_2]$ leads to v at $\tau + \delta(\tau)$ where $\delta(\tau) = \gamma\tau + \beta$,

(with arbitrary delay function δ) (u, v, δ) : taking arc (u, v) at τ leads to v at $\tau + \delta(\tau)$.
Example : speed profile in road networks (20k profiles of 100 speeds/day for 10m edges).

Data structure : temporal edges

Temporal edge (u, v, τ) : arc (u, v) at time τ (delay 1 or 0),

(with delay δ) (u, v, τ, δ) : taking arc (u, v) at time τ from u leads to v at time $\tau + \delta$,

(with availability interval $[\tau_1, \tau_2]$) $(u, v, \tau_1, \tau_2, \delta)$: arc (u, v) is available for any $\tau \in [\tau_1, \tau_2]$ with constant-delay δ,

(with affine delay) $(u, v, \tau_1, \tau_2, \gamma, \beta)$: taking arc (u, v) at $\tau \in [\tau_1, \tau_2]$ leads to v at $\tau + \delta(\tau)$ where $\delta(\tau) = \gamma\tau + \beta$,

(with arbitrary delay function δ) (u, v, δ) : taking arc (u, v) at τ leads to v at $\tau + \delta(\tau)$.
Example : speed profile in road networks (20k profiles of 100 speeds/day for 10m edges).

Data structure : temporal edges

Temporal edge (u, v, τ) : arc (u, v) at time τ (delay 1 or 0),

(with delay δ) (u, v, τ, δ) : taking arc (u, v) at time τ from u leads to v at time $\tau + \delta$,

(with availability interval $[\tau_1, \tau_2]$) $(u, v, \tau_1, \tau_2, \delta)$: arc (u, v) is available for any $\tau \in [\tau_1, \tau_2]$ with constant-delay δ,

(with affine delay) $(u, v, \tau_1, \tau_2, \gamma, \beta)$: taking arc (u, v) at $\tau \in [\tau_1, \tau_2]$ leads to v at $\tau + \delta(\tau)$ where $\delta(\tau) = \gamma\tau + \beta$,

(with arbitrary delay function δ) (u, v, δ) : taking arc (u, v) at τ leads to v at $\tau + \delta(\tau)$.
Example : speed profile in road networks (20k profiles of 100 speeds/day for 10m edges).

Data structure : temporal edges

Temporal edge (u, v, τ) : arc (u, v) at time τ (delay 1 or 0),

(with delay δ) (u, v, τ, δ) : taking arc (u, v) at time τ from u leads to v at time $\tau + \delta$,

(with availability interval $[\tau_1, \tau_2]$) $(u, v, \tau_1, \tau_2, \delta)$: arc (u, v) is available for any $\tau \in [\tau_1, \tau_2]$ with constant-delay δ,

(with affine delay) $(u, v, \tau_1, \tau_2, \gamma, \beta)$: taking arc (u, v) at $\tau \in [\tau_1, \tau_2]$ leads to v at $\tau + \delta(\tau)$ where $\delta(\tau) = \gamma\tau + \beta$,

(with arbitrary delay function δ) (u, v, δ) : taking arc (u, v) at τ leads to v at $\tau + \delta(\tau)$.
Example : speed profile in road networks (20k profiles of 100 speeds/day for 10m edges).

Data structure : temporal edges

Temporal edge (u, v, τ) : arc (u, v) at time τ (delay 1 or 0),

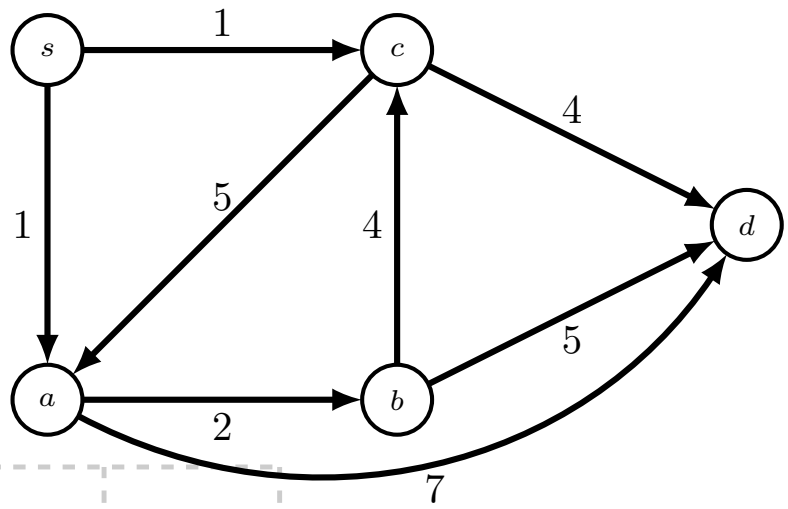
(with delay δ) (u, v, τ, δ) : taking arc (u, v) at time τ from u leads to v at time $\tau + \delta$,

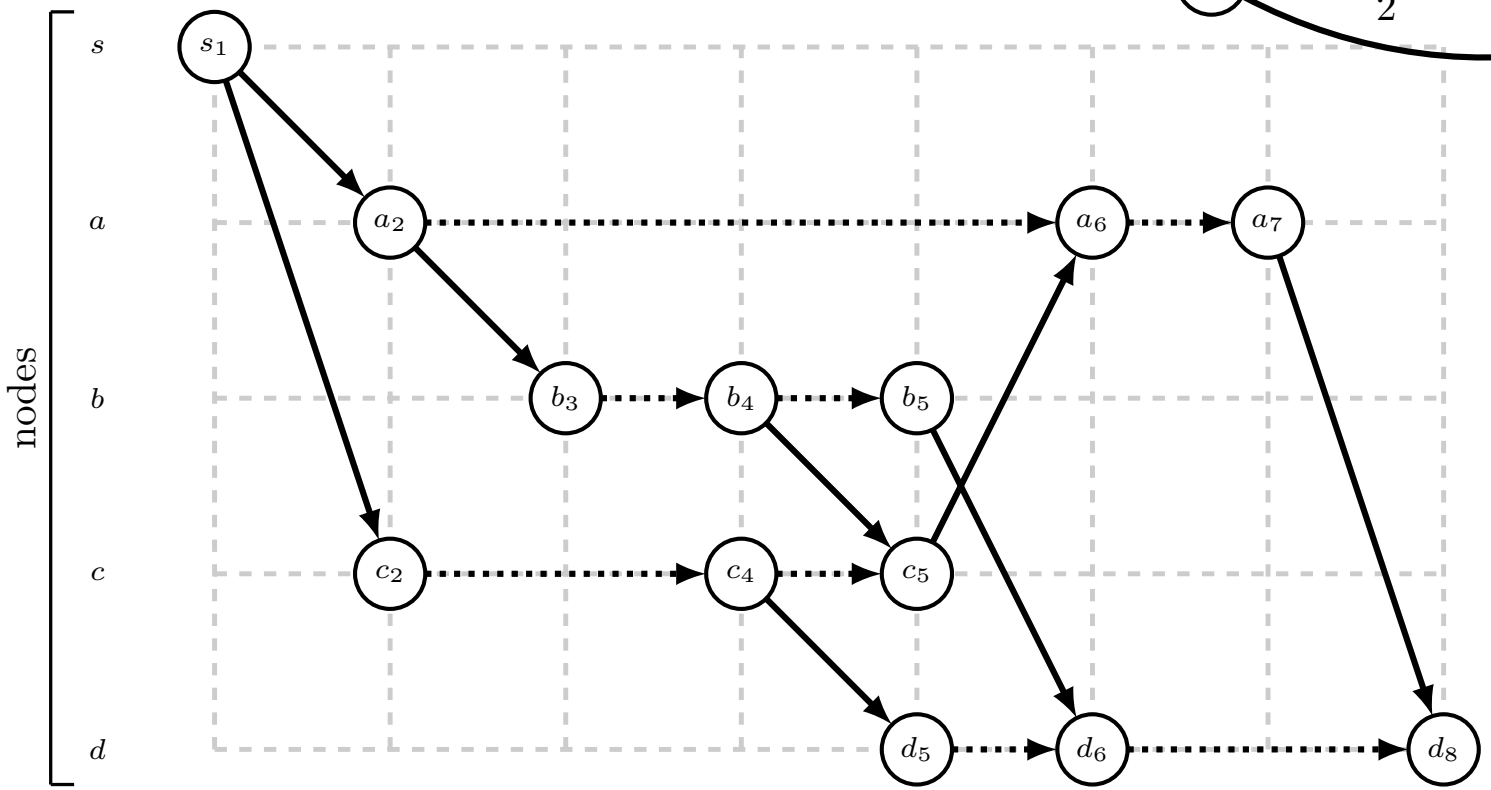
(with availability interval $[\tau_1, \tau_2]$) $(u, v, \tau_1, \tau_2, \delta)$: arc (u, v) is available for any $\tau \in [\tau_1, \tau_2]$ with constant-delay δ,

(with affine delay) $(u, v, \tau_1, \tau_2, \gamma, \beta)$: taking arc (u, v) at $\tau \in [\tau_1, \tau_2]$ leads to v at $\tau + \delta(\tau)$ where $\delta(\tau) = \gamma\tau + \beta$,

(with arbitrary delay function δ) (u, v, δ) : taking arc (u, v) at τ leads to v at $\tau + \delta(\tau)$.
Example : speed profile in road networks (20k profiles of 100 speeds/day for 10m edges).

Data structure : time-expanded graph (point availability)





Data structure : time tables (point availability)

Nom du train	GOTA	NORA	GOTA	NORA	GATA	GOTA	NORA	GATA	NORA	GATA	GOTA	NORA	GATA	NORA	GATA
Bibliothèque François Mitterrand	5:31	5:45	6:01	6:15	6:23	6:30	6:32	6:38	6:46	6:53	7:01	7:02	7:08	7:16	7:24
Paris Austerlitz	5:34	5:48	6:04	6:18	6:26	6:33	6:35	6:41	6:50	6:56	7:04	7:05	7:11	7:20	7:28
Saint-Michel Notre Dame	5:38	5:52	6:08	6:22	6:30	6:38	6:39	6:45	6:54	7:00	7:09	7:09	7:15	7:24	7:32
Musée d'Orsay	5:42	5:56	6:12	6:26	6:34	6:41	6:43	6:48	6:58	7:03	7:12	7:13	7:18	7:28	7:35
Invalides	5:44	5:59	6:14	6:29	6:37	6:44	6:45	6:51	7:00	7:06	7:15	7:15	7:21	7:30	7:38
Pont de l'Alma	5:46	6:01	6:16	6:31	6:39	6:46	6:48	6:53	7:02	7:08	7:17	7:18	7:23	7:33	7:40
Champ de Mars Tour Eiffel	5:49	6:03	6:19	6:33	6:42	6:49	6:50	6:56	7:04	7:11	7:19	7:20	7:26	7:35	7:42
Avenue du Président Kennedy	5:51	6:06	6:21	6:36	6:44	6:51	6:52	6:58	7:07	7:13	7:22	7:22	7:28	7:37	7:45
Boulainvilliers	5:53	6:08	6:23	6:38	6:46	6:53	6:54	7:00	7:09	7:15	7:24	7:24	7:30	7:39	7:47
Avenue Henri Martin	5:56	6:10	6:26	6:40	6:48	6:55	6:57	7:02	7:11	7:17	7:26	7:27	7:32	7:41	7:49
Avenue Foch	5:58	6:12	6:28	6:42	6:50	6:57	6:58	7:04	7:13	7:19	7:28	7:28	7:34	7:43	7:51
Neuilly Porte Maillot	6:00	6:15	6:30	6:45	6:53	7:00	7:01	7:07	7:16	7:22	7:31	7:31	7:37	7:46	7:54
Péreire Levallois	6:03	6:18	6:33	6:48	6:56	7:02	7:04	7:10	7:18	7:25	7:33	7:34	7:40	7:49	7:56
Porte de Clichy	6:06	6:21	6:36	6:51	6:59	7:06	7:07	7:13	7:21	7:28	7:36	7:37	7:43	7:52	8:00

Temporal graph parameters

Number n of nodes.

Number m of edges.

Number μ of availability points/intervals.

We assume $n \leq m \leq \mu$.

$\mu \leq m\Delta_e$ where Δ_e is the maximum number of time events at an edge.

$\mu \leq n\Delta_n$ where Δ_n is the maximum number of time events at a node.

Temporal graph parameters

Number n of nodes.

Number m of edges.

Number μ of availability points/intervals.

We assume $n \leq m \leq \mu$.

$\mu \leq m\Delta_e$ where Δ_e is the maximum number of time events at an edge.

$\mu \leq n\Delta_n$ where Δ_n is the maximum number of time events at a node.

Temporal graph parameters

Number n of nodes.

Number m of edges.

Number μ of availability points/intervals.

We assume $n \leq m \leq \mu$.

$\mu \leq m\Delta_e$ where Δ_e is the maximum number of time events at an edge.

$\mu \leq n\Delta_n$ where Δ_n is the maximum number of time events at a node.

Temporal graph parameters

Number n of nodes.

Number m of edges.

Number μ of availability points/intervals.

We assume $n \leq m \leq \mu$.

$\mu \leq m\Delta_e$ where Δ_e is the maximum number of time events at an edge.

$\mu \leq n\Delta_n$ where Δ_n is the maximum number of time events at a node.

Temporal graph parameters

Number n of nodes.

Number m of edges.

Number μ of availability points/intervals.

We assume $n \leq m \leq \mu$.

$\mu \leq m\Delta_e$ where Δ_e is the maximum number of time events at an edge.

$\mu \leq n\Delta_n$ where Δ_n is the maximum number of time events at a node.

Temporal graph parameters

Number n of nodes.

Number m of edges.

Number μ of availability points/intervals.

We assume $n \leq m \leq \mu$.

$\mu \leq m\Delta_e$ where Δ_e is the maximum number of time events at an edge.

$\mu \leq n\Delta_n$ where Δ_n is the maximum number of time events at a node.

Connectivity : temporal paths

Definition : a **timed edge** is a triple (τ, u, v) such that edge uv is available at time τ ($Arr_{uv}(\tau) < \infty$).

Definition : a **temporal walk** $\mathcal{P}$ from u at τ_{dep} to v at τ_{arr} is a sequence of timed edges $(\tau_1, u_1, v_1), \ldots, (\tau_k, u_k, v_k)$ such that :

- $u_1 = u$, $\tau_1 \tau_{dep}$,
- for $i = 1, \ldots, k-1$, $v_i = u_{i+1}$ and $\tau_i + \delta_{u_i v_i}(\tau_i) \tau_{i+1}$,
- $v_k = v$, $Arr_{u_k v_k}(\tau_k) \tau_{arr}$.

Strict : for $i = 1, \ldots, k-1$, $\tau_i < \tau_{i+1}$.

Temporal path : a temporal walk visiting a node at most once ($|\{u_1, v_1, \ldots, v_k\}| = k+1$).

Connectivity : temporal paths

Definition : a **timed edge** is a triple (τ, u, v) such that edge uv is available at time τ ($Arr_{uv}(\tau) < \infty$).

Definition : a **temporal walk** $\mathcal{P}$ from u at τ_{dep} to v at τ_{arr} is a sequence of timed edges $(\tau_1, u_1, v_1), \ldots, (\tau_k, u_k, v_k)$ such that :

- $u_1 = u$, $\tau_1 \geq \tau_{dep}$,
- for $i = 1, \ldots, k-1$, $v_i = u_{i+1}$ and $\tau_i + \delta_{u_i v_i}(\tau_i) \leq \tau_{i+1}$,
- $v_k = v$, $Arr_{u_k v_k}(\tau_k) \leq \tau_{arr}$.

Strict : for $i = 1, \ldots, k-1$, $\tau_i < \tau_{i+1}$.

Temporal path : a temporal walk visiting a node at most once ($|\{u_1, v_1, \ldots, v_k\}| = k+1$).

Connectivity : temporal paths

Definition : a **timed edge** is a triple (τ, u, v) such that edge uv is available at time τ ($Arr_{uv}(\tau) < \infty$).

Definition : a **non-waiting temporal walk** $\mathcal{P}$ from u at τ_{dep} to v at τ_{arr} is a sequence of timed edges $(\tau_1, u_1, v_1), \ldots, (\tau_k, u_k, v_k)$ such that :

- $u_1 = u$, $\tau_1 = \tau_{dep}$,
- for $i = 1, \ldots, k-1$, $v_i = u_{i+1}$ and $\tau_i + \delta_{u_i v_i}(\tau_i) = \tau_{i+1}$,
- $v_k = v$, $Arr_{u_k v_k}(\tau_k) = \tau_{arr}$.

Strict : for $i = 1, \ldots, k-1$, $\tau_i < \tau_{i+1}$.

Temporal path : a temporal walk visiting a node at most once ($|\{u_1, v_1, \ldots, v_k\}| = k+1$).

Connectivity : temporal paths

Definition : a **timed edge** is a triple (τ, u, v) such that edge uv is available at time τ ($Arr_{uv}(\tau) < \infty$).

Definition : a **temporal walk** $\mathcal{P}$ from u at τ_{dep} to v at τ_{arr} is a sequence of timed edges $(\tau_1, u_1, v_1), \ldots, (\tau_k, u_k, v_k)$ such that :

- $u_1 = u$, $\tau_1 \geq \tau_{dep}$,
- for $i = 1, \ldots, k-1$, $v_i = u_{i+1}$ and $\tau_i + \delta_{u_i v_i}(\tau_i) \leq \tau_{i+1}$,
- $v_k = v$, $Arr_{u_k v_k}(\tau_k) \leq \tau_{arr}$.

Strict : for $i = 1, \ldots, k-1$, $\tau_i < \tau_{i+1}$.

Temporal path : a temporal walk visiting a node at most once ($|\{u_1, v_1, \ldots, v_k\}| = k+1$).

Connectivity : temporal paths

Definition : a timed edge is a triple (τ, u, v) such that edge uv is available at time τ ($Arr_{uv}(\tau) < \infty$).

Definition : a temporal walk $\mathcal{P}$ from u at τ_{dep} to v at τ_{arr} is a sequence of timed edges $(\tau_1, u_1, v_1), \ldots, (\tau_k, u_k, v_k)$ such that :

- $u_1 = u$, $\tau_1 \geq \tau_{dep}$,
- for $i = 1, \ldots, k-1$, $v_i = u_{i+1}$ and $\tau_i + \delta_{u_i v_i}(\tau_i) \leq \tau_{i+1}$,
- $v_k = v$, $Arr_{u_k v_k}(\tau_k) \leq \tau_{arr}$.

Strict : for $i = 1, \ldots, k-1$, $\tau_i < \tau_{i+1}$.

Temporal path : a temporal walk visiting a node at most once ($|\{u_1, v_1, \ldots, v_k\}| = k+1$).

A natural problem : earliest arrival time

Definition : $EAT_{uv}(\tau) := \min_{\mathcal{P} \text{ dep. at } \tau} \tau_{arr}(\mathcal{P})$.

A path $\mathcal{P}$ arriving at $EAT_{uv}(\tau)$ is a foremost path.

EAT Problem : given u, v, τ, compute $EAT_{uv}(\tau)$.

Natural solution : temporal Dijkstra [Berman 1996] in $O(n \log n + m \log \Delta_e)$ (adjacency lists) (also $O(n \log n + \mu)$ with a sorted list of temporal edges).

Question : what makes temporal Dijkstra correct ?

A natural problem : earliest arrival time

Definition : $EAT_{uv}(\tau) := \min_{\mathcal{P} \text{ dep. at } \tau} \tau_{arr}(\mathcal{P})$.

A path $\mathcal{P}$ arriving at $EAT_{uv}(\tau)$ is a foremost path.

EAT Problem : given u, v, τ, compute $EAT_{uv}(\tau)$.

Natural solution : temporal Dijkstra [Berman 1996] in $O(n \log n + m \log \Delta_e)$ (adjacency lists) (also $O(n \log n + \mu)$ with a sorted list of temporal edges).

Question : what makes temporal Dijkstra correct ?

A natural problem : earliest arrival time

Definition : $EAT_{uv}(\tau) := \min_{\mathcal{P}} \text{ dep. at } \tau \ \tau_{arr}(\mathcal{P})$.

A path $\mathcal{P}$ arriving at $EAT_{uv}(\tau)$ is a foremost path.

EAT Problem : given u, v, τ, compute $EAT_{uv}(\tau)$.

Natural solution : temporal Dijkstra [Berman 1996] in $O(n \log n + m \log \Delta_e)$ (adjacency lists) (also $O(n \log n + \mu)$ with a sorted list of temporal edges).

Question : what makes temporal Dijkstra correct ?

A natural problem : earliest arrival time

Definition : $EAT_{uv}(\tau) := \min_{\mathcal{P}} \text{ dep. at } \tau \ \tau_{arr}(\mathcal{P})$.

A path $\mathcal{P}$ arriving at $EAT_{uv}(\tau)$ is a foremost path.

EAT Problem : given u, v, τ, compute $EAT_{uv}(\tau)$.

Natural solution : temporal Dijkstra [Berman 1996] in $O(n \log n + m \log \Delta_e)$ (adjacency lists) (also $O(n \log n + \mu)$ with a sorted list of temporal edges).

Question : what makes temporal Dijkstra correct ?

A natural problem : earliest arrival time

Definition : $EAT_{uv}(\tau) := \min_{\mathcal{P}} \text{ dep. at } \tau \ \tau_{arr}(\mathcal{P})$.

A path $\mathcal{P}$ arriving at $EAT_{uv}(\tau)$ is a foremost path.

EAT Problem : given u, v, τ, compute $EAT_{uv}(\tau)$.

Natural solution : temporal Dijkstra [Berman 1996] in $O(n \log n + m \log \Delta_e)$ (adjacency lists) (also $O(n \log n + \mu)$ with a sorted list of temporal edges).

Question : what makes temporal Dijkstra correct ?

The power of waiting

Assumption :
FIFO property : for all edge uv $\in$ E and $\tau < \tau'$,
$\mathsf{Arr}_{uv}(\tau) \leq \mathsf{Arr}_{uv}(\tau')$ (where $\mathsf{Arr}_{uv}(\tau) = \tau + \delta_{uv}(\tau)$).

Consequence :

- loops are useless,
- any foremost path is concatenated : it can be obtained by extending a foremost path.

Lemma : waiting implies FIFO.

The power of waiting

Assumption :
FIFO property : for all edge uv $\in$ E and $\tau < \tau'$,
$\mathsf{Arr}_{uv}(\tau) \leq \mathsf{Arr}_{uv}(\tau')$ (where $\mathsf{Arr}_{uv}(\tau) = \tau + \delta_{uv}(\tau)$).

Consequence :

- loops are useless,
- any foremost path is concatenated : it can be obtained by extending a foremost path.

Lemma : waiting implies FIFO.

The power of waiting

Assumption :
FIFO property : for all edge uv $\in$ E and $\tau < \tau'$,
$\mathsf{Arr}_{uv}(\tau) \leq \mathsf{Arr}_{uv}(\tau')$ (where $\mathsf{Arr}_{uv}(\tau) = \tau + \delta_{uv}(\tau)$).

Consequence :

- loops are useless,
- any foremost path is concatenated : it can be obtained by extending a foremost path.

Lemma : waiting implies FIFO.

Infinite shortest path (non-waiting)

A temporal graph can have foremost temporal walks with infinitely many edges. [Orda, Rom 1991]

Exercise what if waiting is allowed?

Theorem [Orda, Rom 1991] : finding a non-waiting temporal walk is NP-hard.

Remark Becomes polynomial in the point availability model...

Infinite shortest path (non-waiting)

A temporal graph can have foremost temporal walks with infinitely many edges. [Orda, Rom 1991]

Exercise what if waiting is allowed?

Theorem [Orda, Rom 1991] : finding a non-waiting temporal walk is NP-hard.

Remark Becomes polynomial in the point availability model...

Infinite shortest path (non-waiting)

A temporal graph can have foremost temporal walks with infinitely many edges. [Orda, Rom 1991]

Exercise what if waiting is allowed?

Theorem [Orda, Rom 1991] : finding a non-waiting temporal walk is NP-hard.

Remark Becomes polynomial in the point availability model...

Infinite shortest path (non-waiting)

A temporal graph can have foremost temporal walks with infinitely many edges. [Orda, Rom 1991]

Exercise what if waiting is allowed?

Theorem [Orda, Rom 1991] : finding a non-waiting temporal walk is NP-hard.

Remark Becomes polynomial in the point availability model...

Temporal graph hierarchy

Profile problem : given u, v, compute $EAT_{uv}(.)$ assuming FIFO.

model	profile size π	profile complexity
piecewise linear-delay	$\pi = \mu n^{O(\log n)}$ [Foschini et al 14]	$O(nm\pi)$ [Orda, Rom 90]
piecewise constant-delay	$\pi = O(\mu)$	$O((m + n \log n)\mu)$ [Dehne et al 12]
point-availability		
unit-delay point-availability		$O(\mu)$ [Kossinets et al 08]

Rq1 : for a walk $P = e_1, \ldots, e_k$, $Arr_P(\tau) = Arr_{e_k} \circ \cdots \circ Arr_{e_1}(\tau)$

Rq2 : $EAT_{uv}(.) = \min_{uv\text{-walk } P} Arr_P(.)$

Open pb : fine-grained complexity of profile (interval) or even shortest duration.

Temporal graph hierarchy

Profile problem : given u, v, compute $EAT_{uv}(.)$ assuming FIFO.

model	profile size π	profile complexity
piecewise linear-delay	$\pi = \mu n^{O(\log n)}$ [Foschini et al 14]	$O(nm\pi)$ [Orda, Rom 90]
piecewise constant-delay	$\pi = O(\mu)$	$O((m + n \log n)\mu)$ [Dehne et al 12]
point-availability		
unit-delay point-availability		$O(\mu)$ [Kossinets et al 08]

Rq1 : for a walk $P = e_1, \ldots, e_k$, $Arr_P(\tau) = Arr_{e_k} \circ \cdots \circ Arr_{e_1}(\tau)$

Rq2 : $EAT_{uv}(.) = \min_{uv\text{-walk } P} Arr_P(.)$

Open pb : fine-grained complexity of profile (interval) or even shortest duration.

Temporal graph hierarchy

Profile problem : given u, v, compute $EAT_{uv}(.)$ assuming FIFO.

model	profile size π	profile complexity
piecewise linear-delay	$\pi = \mu n^{O(\log n)}$ [Foschini et al 14]	$O(nm\pi)$ [Orda, Rom 90]
piecewise constant-delay	$\pi = O(\mu)$	$O((m + n \log n)\mu)$ [Dehne et al 12]
point-availability		
unit-delay point-availability		$O(\mu)$ [Kossinets et al 08]

Rq1 : for a walk $P = e_1, \ldots, e_k$, $Arr_P(\tau) = Arr_{e_k} \circ \cdots \circ Arr_{e_1}(\tau)$

Rq2 : $EAT_{uv}(.) = \min_{uv\text{-walk } P} Arr_P(.)$

Open pb : fine-grained complexity of profile (interval) or even shortest duration.

Temporal graph hierarchy

Profile problem : given u, v, compute $EAT_{uv}(.)$ assuming FIFO.

model	profile size π	profile complexity
piecewise linear-delay	$\pi = \mu n^{O(\log n)}$ [Foschini et al 14]	$O(nm\pi)$ [Orda, Rom 90]
piecewise constant-delay	$\pi = O(\mu)$	$O((m + n \log n)\mu)$ [Dehne et al 12]
point-availability		$O(\mu \log \mu)$ [Dibbelt et al 13, ...]
unit-delay point-availability		$O(\mu)$ [Kossinets et al 08]

Rq1 : for a walk $P = e_1, \ldots, e_k$, $Arr_P(\tau) = Arr_{e_k} \circ \cdots \circ Arr_{e_1}(\tau)$

Rq2 : $EAT_{uv}(.) = \min_{uv\text{-walk } P} Arr_P(.)$

Open pb : fine-grained complexity of profile (interval) or even shortest duration.

Temporal graph hierarchy

Profile problem : given u, v, compute $EAT_{uv}(.)$ assuming FIFO.

model	profile size π	profile complexity
piecewise linear-delay	$\pi = \mu n^{O(\log n)}$ [Foschini et al 14]	$O(nm\pi)$ [Orda, Rom 90]
piecewise constant-delay	$\pi = O(\mu)$	$O((m + n \log n)\mu)$ [Dehne et al 12]
point-availability		$O(\mu \log \mu)$ [Dibbelt et al 13, ...]
unit-delay point-availability		$O(\mu)$ [Kossinets et al 08]

Rq1 : for a walk $P = e_1, \ldots, e_k$, $Arr_P(\tau) = Arr_{e_k} \circ \cdots \circ Arr_{e_1}(\tau)$

Rq2 : $EAT_{uv}(.) = \min_{uv\text{-walk } P} Arr_P(.)$

Open pb : fine-grained complexity of profile (interval) or even shortest duration.

Temporal graph hierarchy

Profile problem : given u, v, compute $EAT_{uv}(.)$ assuming FIFO.

model	profile size π	profile complexity
piecewise linear-delay	$\pi = \mu n^{O(\log n)}$ [Foschini et al 14]	$O(nm\pi)$ [Orda, Rom 90]
piecewise constant-delay	$\pi = O(\mu)$	$O((m + n \log n)\mu)$ [Dehne et al 12]
point-availability		$O(\mu)$ [Brunelli et al 23]
unit-delay point-availability		$O(\mu)$ [Kossinets et al 08]

Rq1 : for a walk $P = e_1, \ldots, e_k$, $Arr_P(\tau) = Arr_{e_k} \circ \cdots \circ Arr_{e_1}(\tau)$

Rq2 : $EAT_{uv}(.) = \min_{uv\text{-walk } P} Arr_P(.)$

Open pb : fine-grained complexity of profile (interval) or even shortest duration.

Temporal graph hierarchy

Profile problem : given u, v, compute $EAT_{uv}(.)$ assuming FIFO.

model	profile size π	profile complexity
piecewise linear-delay	$\pi = \mu n^{O(\log n)}$ [Foschini et al 14]	$O(nm\pi)$ [Orda, Rom 90]
piecewise constant-delay	$\pi = O(\mu)$	$O((m + n \log n)\mu)$ [Dehne et al 12]
point-availability		$O(\mu)$ [Brunelli et al 23]
unit-delay point-availability		$O(\mu)$ [Kossinets et al 08]

Rq1 : for a walk $P = e_1, \ldots, e_k$, $Arr_P(\tau) = Arr_{e_k} \circ \cdots \circ Arr_{e_1}(\tau)$

Rq2 : $EAT_{uv}(.) = \min_{uv\text{-walk } P} Arr_P(.)$

Open pb : fine-grained complexity of profile (interval) or even shortest duration.

Temporal graph hierarchy

Profile problem : given u, v, compute $EAT_{uv}(.)$ assuming FIFO.

model	profile size π	profile complexity
piecewise linear-delay	$\pi = \mu n^{O(\log n)}$ [Foschini et al 14]	$O(nm\pi)$ [Orda, Rom 90]
piecewise constant-delay	$\pi = O(\mu)$	$O((m + n \log n)\mu)$ [Dehne et al 12]
point-availability		$O(\mu)$ [Brunelli et al 23]
unit-delay point-availability		$O(\mu)$ [Kossinets et al 08]

Rq1 : for a walk $P = e_1, \ldots, e_k$, $Arr_P(\tau) = Arr_{e_k} \circ \cdots \circ Arr_{e_1}(\tau)$

Rq2 : $EAT_{uv}(.) = \min_{uv\text{-walk } P} Arr_P(.)$

Open pb : fine-grained complexity of profile (interval) or even shortest duration.

Temporal graph hierarchy

Profile problem : given u, v, compute $EAT_{uv}(.)$ assuming FIFO.

model	profile size π	profile complexity
piecewise linear-delay	$\pi = \mu n^{O(\log n)}$ [Foschini et al 14]	$O(nm\pi)$ [Orda, Rom 90]
piecewise constant-delay	$\pi = O(\mu)$	$O((m + n \log n)\mu)$ [Dehne et al 12]
point-availability		$O(\mu)$ [Brunelli et al 23]
unit-delay point-availability		$O(\mu)$ [Kossinets et al 08]

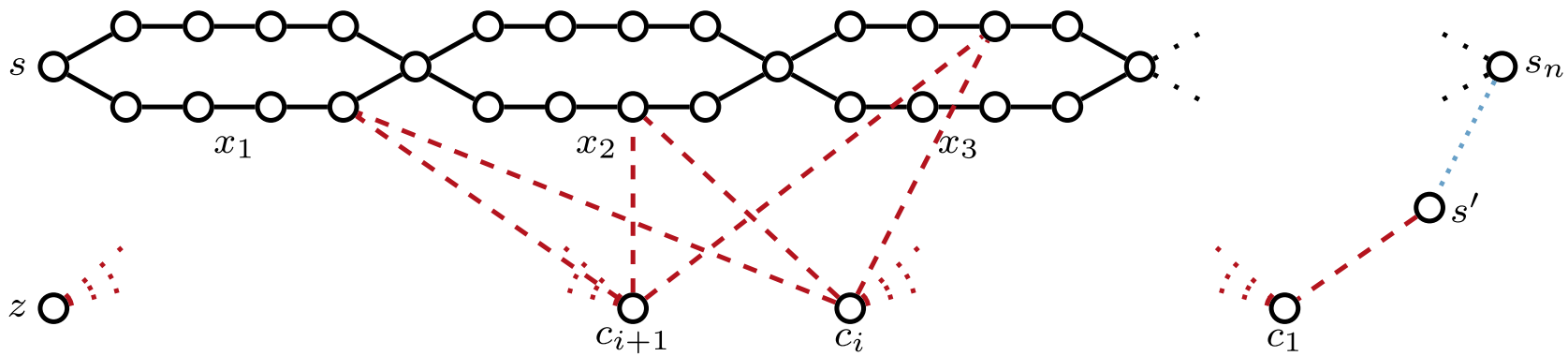
Rq1 : for a walk $P = e_1, \ldots, e_k$, $Arr_P(\tau) = Arr_{e_k} \circ \cdots \circ Arr_{e_1}(\tau)$

Rq2 : $EAT_{uv}(.) = \min_{uv\text{-walk } P} Arr_P(.)$

Open pb : fine-grained complexity of profile (interval) or even shortest duration.

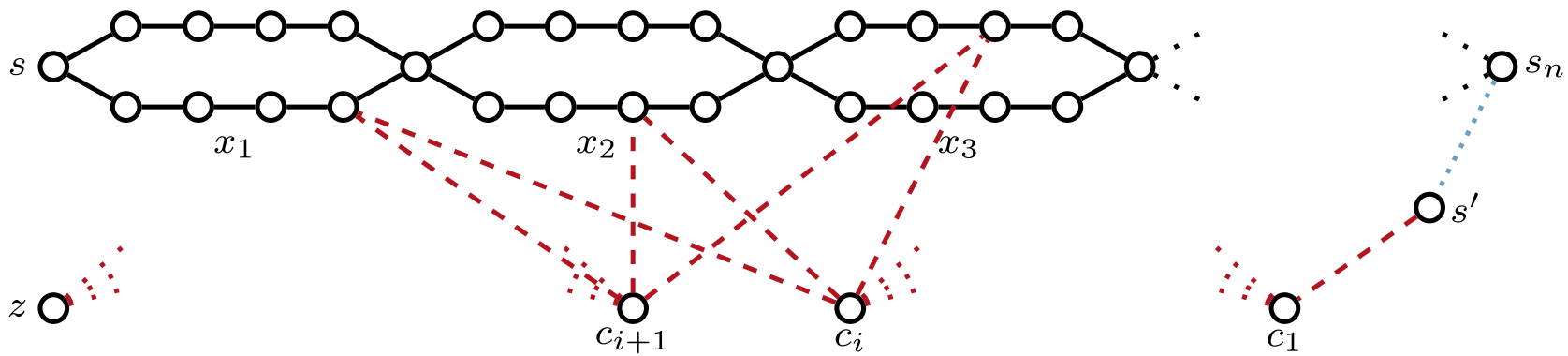
Point-availability : non-waiting temporal paths are hard

Theorem [Casteigts, Himmel, Molter, Zschoche 2021] Testing if a β-waiting temporal path connects a given pair of nodes is NP-hard (even W[1]-hard parametrized by pathwidth).

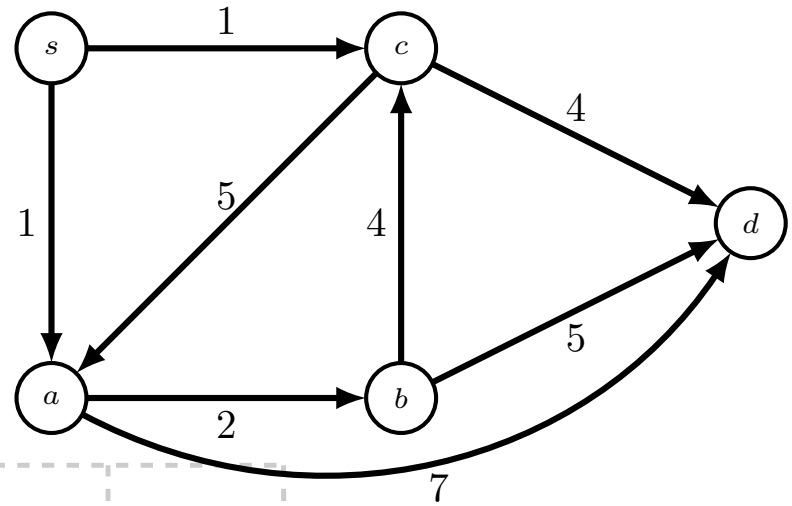


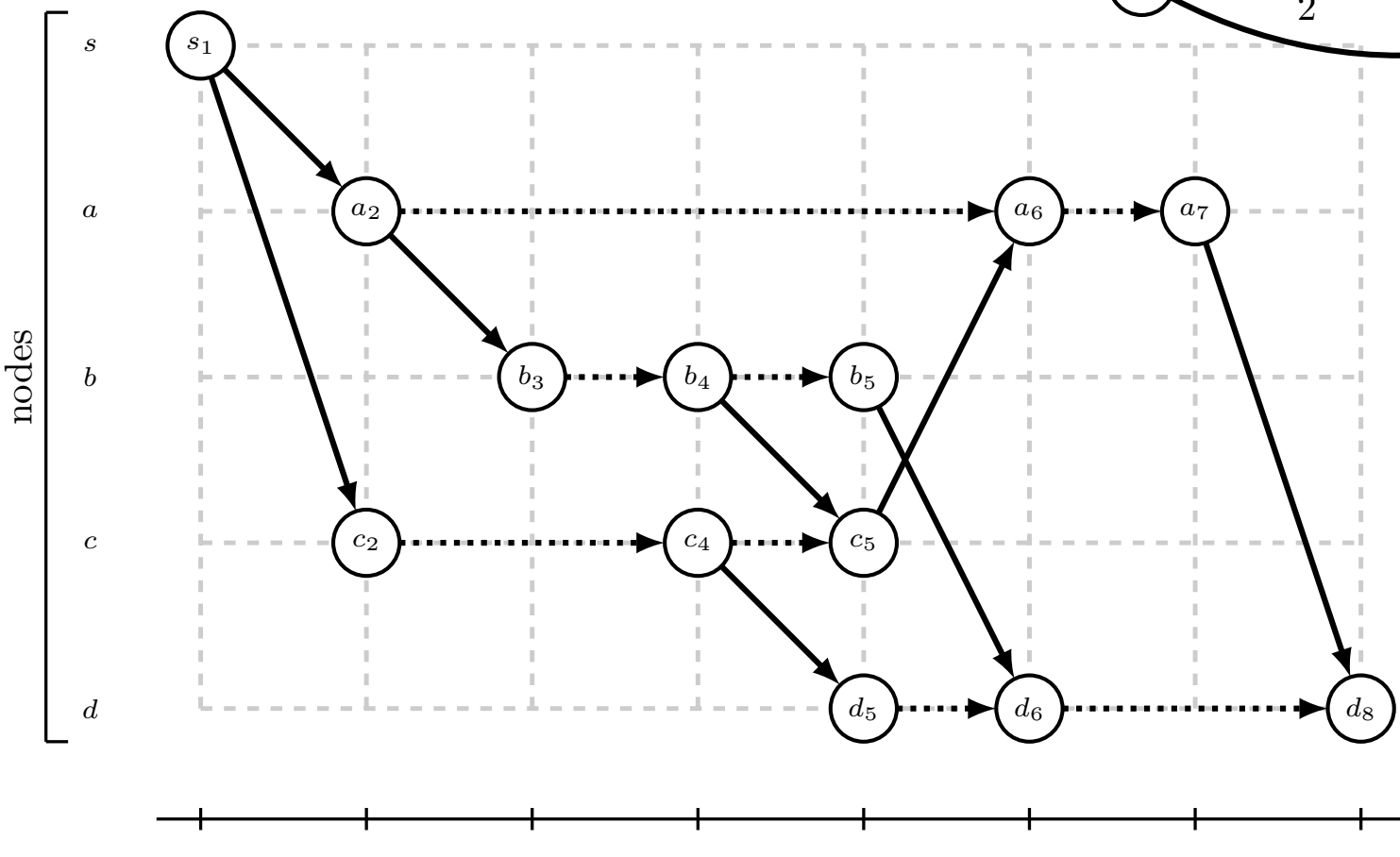
Point-availability : non-waiting temporal paths are hard

Theorem [Casteigts, Himmel, Molter, Zschoche 2021] Testing if a β-waiting temporal path connects a given pair of nodes is NP-hard (even W[1]-hard parametrized by pathwidth).



β-waiting temp. walks in lin. time [Brunelli et al. 2023]





Various natural "shortest" uv-walks

Earliest arrival (or foremost) : starting at τ from u, minimize arrival time in v (concatenated).

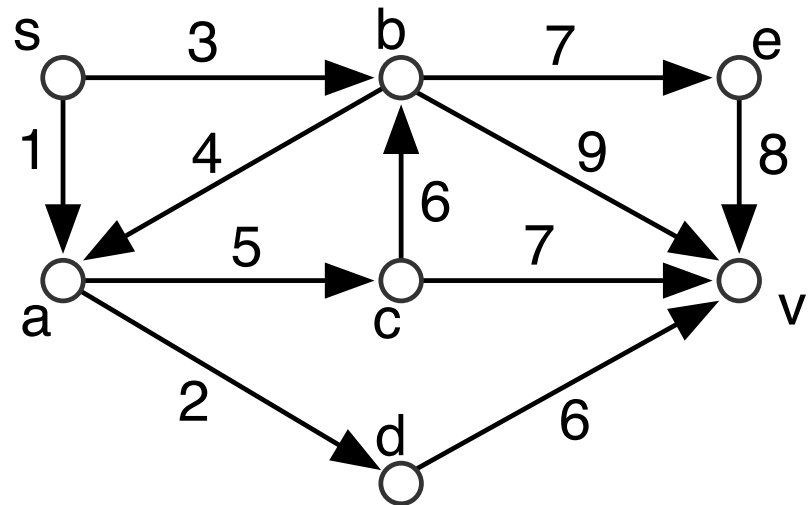
Concatenated : any optimal walk can be obtained by concatenating an edge to an optimal walk.

Fewest hops (or shortest) : minimize the number k of edges (not concat.).

Shortest duration (or fastest) : minimize arrival time - departure time (not concatenated).

Shortest delay : minimize $\sum_{i=1}^{k} Arr_{u_i v_i}(\tau_i) - \tau_i$ (not concat.) over walks $\mathcal{P} = u_1 v_1, \ldots, u_k v_k$.

Min. waiting time : minimize $\sum_{i=1}^{k-1} \tau_{i+1} - Arr_{u_i v_i}(\tau_i)$ (not concat.).



Various natural "shortest" uv-walks

Earliest arrival (or foremost) : starting at τ from u, minimize arrival time in v (concatenated).

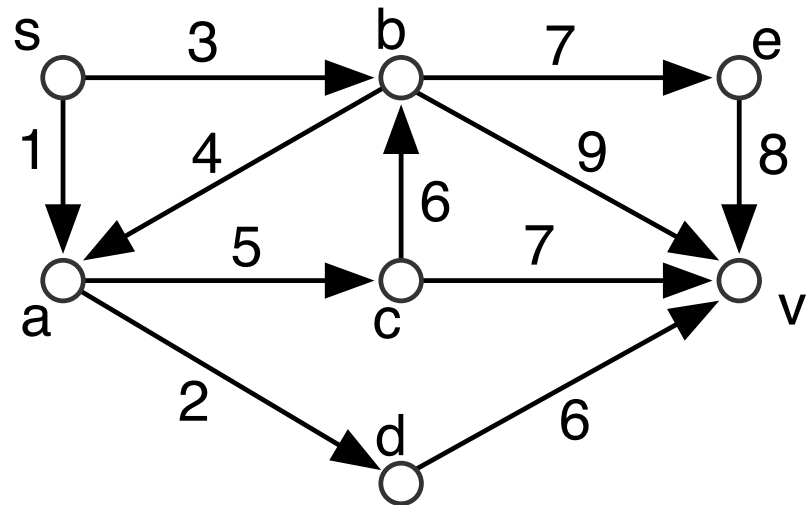
Concatenated : any optimal walk can be obtained by concatenating an edge to an optimal walk.

Fewest hops (or shortest) : minimize the number k of edges (not concat.).

Shortest duration (or fastest) : minimize arrival time - departure time (not concatenated).

Shortest delay : minimize $\sum_{i=1}^{k} Arr_{u_i v_i}(\tau_i) - \tau_i$ (not concat.) over walks $\mathcal{P} = u_1 v_1, \ldots, u_k v_k$.

Min. waiting time : minimize $\sum_{i=1}^{k-1} \tau_{i+1} - Arr_{u_i v_i}(\tau_i)$ (not concat.).



Various natural "shortest" uv-walks

Earliest arrival (or foremost) : starting at τ from u, minimize arrival time in v (concatenated).

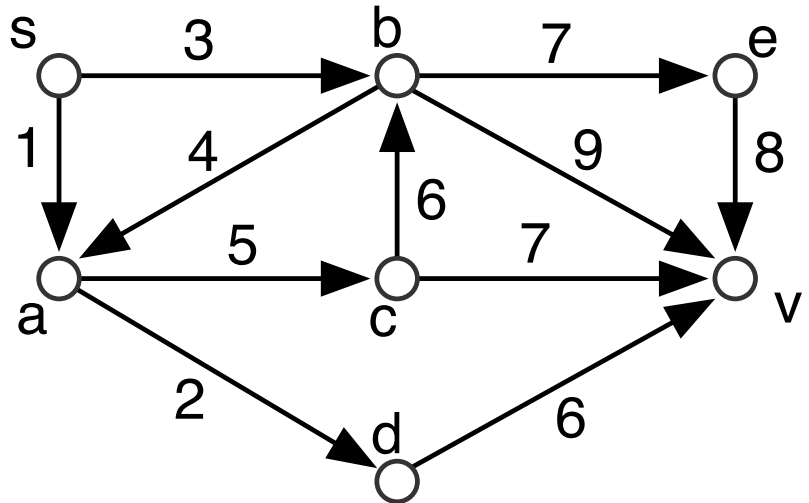
Concatenated : any optimal walk can be obtained by concatenating an edge to an optimal walk.

Fewest hops (or shortest) : minimize the number k of edges (not concat.).

Shortest duration (or fastest) : minimize arrival time - departure time (not concatenated).

Shortest delay : minimize $\sum_{i=1}^{k} Arr_{u_i v_i}(\tau_i) - \tau_i$ (not concat.) over walks $\mathcal{P} = u_1 v_1, \ldots, u_k v_k$.

Min. waiting time : minimize $\sum_{i=1}^{k-1} \tau_{i+1} - Arr_{u_i v_i}(\tau_i)$ (not concat.).



Various natural "shortest" uv-walks

Earliest arrival (or foremost) : starting at τ from u, minimize arrival time in v (concatenated).

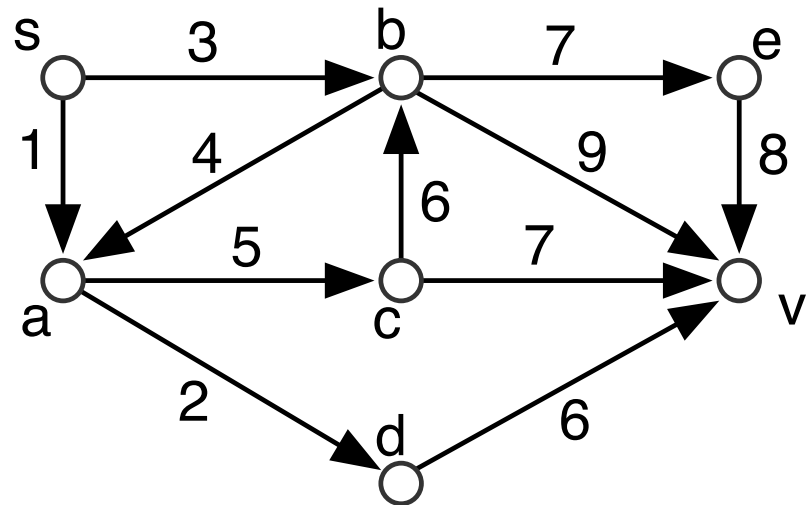
Concatenated : any optimal walk can be obtained by concatenating an edge to an optimal walk.

Fewest hops (or shortest) : minimize the number k of edges (not concat.).

Shortest duration (or fastest) : minimize arrival time - departure time (not concatenated).

Shortest delay : minimize $\sum_{i=1}^{k} Arr_{u_i v_i}(\tau_i) - \tau_i$ (not concat.) over walks $\mathcal{P} = u_1 v_1, \ldots, u_k v_k$.

Min. waiting time : minimize $\sum_{i=1}^{k-1} \tau_{i+1} - Arr_{u_i v_i}(\tau_i)$ (not concat.).



Various natural "shortest" uv-walks

Earliest arrival (or foremost) : starting at τ from u, minimize arrival time in v (concatenated).

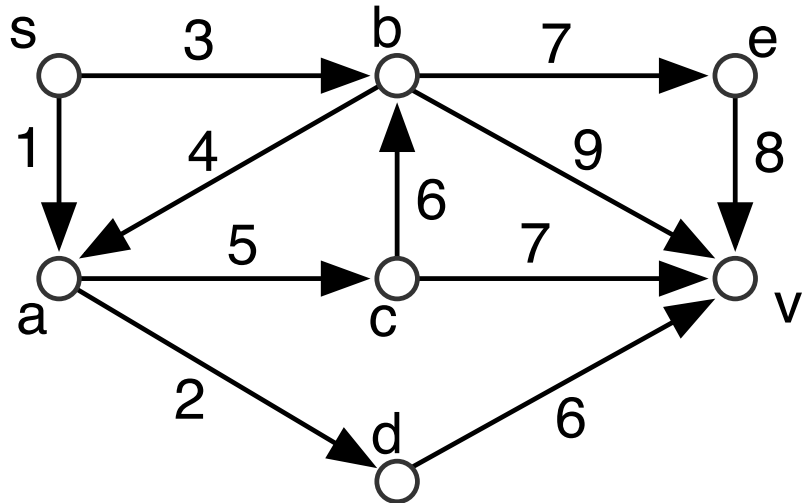
Concatenated : any optimal walk can be obtained by concatenating an edge to an optimal walk.

Fewest hops (or shortest) : minimize the number k of edges (not concat.).

Shortest duration (or fastest) : minimize arrival time - departure time (not concatenated).

Shortest delay : minimize $\sum_{i=1}^{k} Arr_{u_i v_i}(\tau_i) - \tau_i$ (not concat.) over walks $\mathcal{P} = u_1v_1, \ldots, u_kv_k$.

Min. waiting time : minimize $\sum_{i=1}^{k-1} \tau_{i+1} - Arr_{u_i v_i}(\tau_i)$ (not concat.).



Min-cost vs arrival time

Temporal path cost : $\mathsf{cost}(\mathcal{P}, (\tau, u, v))) = \mathsf{cost}(\mathcal{P}) \oplus \mathsf{cost}_{uv}(\tau)$

Theorem [Brunelli et al. 2023] Given the time-expanded representation and a fixed node s, minimum-cost strict se-walks can be computed for all temporal edges e in linear time when costs are isotone ($c_1 \oplus c \leq c_2 \oplus c$ when $c_1 \leq c_2$).

Lemma : Minimum-cost walks are concatenated.

Idea : Use the β-waiting algorithm plus intervals at each node for grouping node copies with same optimal cost.

Min-cost :

- edge cost of 1 : shortest (fewest hops),
- edge cost of -1 : longest (most hops),
- cost of $-\tau_{dep}$: profile and fastest (shortest duration).

Exercise : define isotone costs allowing to compute minimum-waiting walks.

Min-cost vs arrival time

Temporal path cost : $\mathsf{cost}(\mathcal{P}, (\tau, \mathsf{u}, \mathsf{v}))) = \mathsf{cost}(\mathcal{P}) \oplus \mathsf{cost}_{\mathsf{uv}}(\tau)$

Theorem [Brunelli et al. 2023] Given the time-expanded representation and a fixed node s, minimum-cost strict se-walks can be computed for all temporal edges e in linear time when costs are isotone ($c_1 \oplus c \leq c_2 \oplus c$ when $c_1 \leq c_2$).

Lemma : Minimum-cost walks are concatenated.

Idea : Use the β-waiting algorithm plus intervals at each node for grouping node copies with same optimal cost.

Min-cost :

- edge cost of 1 : shortest (fewest hops),
- edge cost of -1 : longest (most hops),
- cost of $-\tau_{\mathsf{dep}}$: profile and fastest (shortest duration).

Exercise : define isotone costs allowing to compute minimum-waiting walks.

Min-cost vs arrival time

Temporal path cost : $\mathsf{cost}(\mathcal{P}, (\tau, u, v))) = \mathsf{cost}(\mathcal{P}) \oplus \mathsf{cost}_{uv}(\tau)$

Theorem [Brunelli et al. 2023] Given the time-expanded representation and a fixed node s, minimum-cost strict se-walks can be computed for all temporal edges e in linear time when costs are isotone ($c_1 \oplus c \leq c_2 \oplus c$ when $c_1 \leq c_2$).

Lemma : Minimum-cost walks are concatenated.

Idea : Use the β-waiting algorithm plus intervals at each node for grouping node copies with same optimal cost.

Min-cost :

- edge cost of 1 : shortest (fewest hops),
- edge cost of -1 : longest (most hops),
- cost of $-\tau_{dep}$: profile and fastest (shortest duration).

Exercise : define isotone costs allowing to compute minimum-waiting walks.

Min-cost vs arrival time

Temporal path cost : $\mathsf{cost}(\mathcal{P}, (\tau, \mathsf{u}, \mathsf{v}))) = \mathsf{cost}(\mathcal{P}) \oplus \mathsf{cost}_{\mathsf{uv}}(\tau)$

Theorem [Brunelli et al. 2023] Given the time-expanded representation and a fixed node s, minimum-cost strict se-walks can be computed for all temporal edges e in linear time when costs are isotone ($c_1 \oplus c \leq c_2 \oplus c$ when $c_1 \leq c_2$).

Lemma : Minimum-cost walks are concatenated.

Idea : Use the β-waiting algorithm plus intervals at each node for grouping node copies with same optimal cost.

Min-cost :

- edge cost of 1 : shortest (fewest hops),
- edge cost of -1 : longest (most hops),
- cost of $-\tau_{\mathsf{dep}}$: profile and fastest (shortest duration).

Exercise : define isotone costs allowing to compute minimum-waiting walks.

Min-cost vs arrival time

Temporal path cost : $\mathsf{cost}(\mathcal{P}, (\tau, \mathsf{u}, \mathsf{v}))) = \mathsf{cost}(\mathcal{P}) \oplus \mathsf{cost}_{\mathsf{uv}}(\tau)$

Theorem [Brunelli et al. 2023] Given the time-expanded representation and a fixed node s, minimum-cost strict se-walks can be computed for all temporal edges e in linear time when costs are isotone ($c_1 \oplus c \leq c_2 \oplus c$ when $c_1 \leq c_2$).

Lemma : Minimum-cost walks are concatenated.

Idea : Use the β-waiting algorithm plus intervals at each node for grouping node copies with same optimal cost.

Min-cost :

- edge cost of 1 : shortest (fewest hops),
- edge cost of -1 : longest (most hops),
- cost of $-\tau_{\mathsf{dep}}$: profile and fastest (shortest duration).

Exercise : define isotone costs allowing to compute minimum-waiting walks.

Min-cost vs arrival time

Temporal path cost : $\mathsf{cost}(\mathcal{P}, (\tau, \mathsf{u}, \mathsf{v}))) = \mathsf{cost}(\mathcal{P}) \oplus \mathsf{cost}_{\mathsf{uv}}(\tau)$

Theorem [Brunelli et al. 2023] Given the time-expanded representation and a fixed node s, minimum-cost strict se-walks can be computed for all temporal edges e in linear time when costs are isotone ($c_1 \oplus c \leq c_2 \oplus c$ when $c_1 \leq c_2$).

Lemma : Minimum-cost walks are concatenated.

Idea : Use the β-waiting algorithm plus intervals at each node for grouping node copies with same optimal cost.

Min-cost :

- edge cost of 1 : shortest (fewest hops),
- edge cost of -1 : longest (most hops),
- cost of $-\tau_{\mathsf{dep}}$: profile and fastest (shortest duration).

Exercise : define isotone costs allowing to compute minimum-waiting walks.

Generalizing static graph concepts

Basic temporal graphs

Single-labeled temporal graphs :

each edge uv appears at a single time $\lambda(uv) \in \mathbb{N}$.

Basic temporal graphs

Single-labeled temporal graphs :

each edge uv appears at a single time $\lambda(uv) \in \mathbb{N}$.

Strongly connected components

Definition 1 : $U \subseteq V$ is (strongly) open-connected if for any $u, v \in U$ there exists a temporal path from u to v.

Definition 2 : $U \subseteq V$ is (strongly) closed-connected if for any $u, v \in U$ there exists a temporal path from u to v in $G_{|U}$.

Theorem [Bhadra, Ferreira 03] : Deciding whether there exists a strongly open/closed-connected set of size k is W[1]/NP-hard (in parameter k).

IDea : Reduction of k-Clique : for each edge uv with $u < v$: create $u \xrightarrow{-2} (u, v) \xrightarrow{-3} v$ and $v \xrightarrow{-2} (v, u) \xrightarrow{-3} u$.

Open pb : Show that finding a closed-connected set of size k is W[1]-hard (for parameter k).

Strongly connected components

Definition 1 : $U \subseteq V$ is (strongly) open-connected if for any $u, v \in U$ there exists a temporal path from u to v.

Definition 2 : $U \subseteq V$ is (strongly) closed-connected if for any $u, v \in U$ there exists a temporal path from u to v in $G_{|U}$.

Theorem [Bhadra, Ferreira 03] : Deciding whether there exists a strongly open/closed-connected set of size k is W[1]/NP-hard (in parameter k).

IDea : Reduction of k-Clique : for each edge uv with $u < v$: create $u \xrightarrow{-2} (u, v) \xrightarrow{-3} v$ and $v \xrightarrow{-2} (v, u) \xrightarrow{-3} u$.

Open pb : Show that finding a closed-connected set of size k is W[1]-hard (for parameter k).

Strongly connected components

Definition 1 : $U \subseteq V$ is (strongly) open-connected if for any $u, v \in U$ there exists a temporal path from u to v.

Definition 2 : $U \subseteq V$ is (strongly) closed-connected if for any $u, v \in U$ there exists a temporal path from u to v in $G_{|U}$.

Theorem [Bhadra, Ferreira 03] : Deciding whether there exists a strongly open/closed-connected set of size k is W[1]/NP-hard (in parameter k).

IDea : Reduction of k-Clique : for each edge uv with $u < v$: create $u \xrightarrow{-2} (u, v) \xrightarrow{-3} v$ and $v \xrightarrow{-2} (v, u) \xrightarrow{-3} u$.

Open pb : Show that finding a closed-connected set of size k is W[1]-hard (for parameter k).

Strongly connected components

Definition 1 : $U \subseteq V$ is (strongly) open-connected if for any $u, v \in U$ there exists a temporal path from u to v.

Definition 2 : $U \subseteq V$ is (strongly) closed-connected if for any $u, v \in U$ there exists a temporal path from u to v in $G_{|U}$.

Theorem [Bhadra, Ferreira 03] : Deciding whether there exists a strongly open/closed-connected set of size k is W[1]/NP-hard (in parameter k).

IDea : Reduction of k-Clique : for each edge uv with $u < v$: create $u -2\rightarrow (u, v) -3\rightarrow v$ and $v -2\rightarrow (v, u) -3\rightarrow u$.

Open pb : Show that finding a closed-connected set of size k is W[1]-hard (for parameter k).

Strongly connected components

Definition 1 : $U \subseteq V$ is (strongly) open-connected if for any $u, v \in U$ there exists a temporal path from u to v.

Definition 2 : $U \subseteq V$ is (strongly) closed-connected if for any $u, v \in U$ there exists a temporal path from u to v in $G_{|U}$.

Theorem [Bhadra, Ferreira 03] : Deciding whether there exists a strongly open/closed-connected set of size k is $W[1]$/NP-hard (in parameter k).

IDea : Reduction of k-Clique : for each edge uv with $u < v$: create $u -2\rightarrow (u, v) -3\rightarrow v$ and $v -2\rightarrow (v, u) -3\rightarrow u$.

Open pb : Show that finding a closed-connected set of size k is $W[1]$-hard (for parameter k).

Minimum spanner

Sparse spanner : keep a minimum number of edges preserving connectivity.

Spanner of a (connected) static graph ?

Spanner of a temporal graph [Kempe et al 02] ?

Exercise : minimum spanner of the hypercube of dim. k with temporal edges $b_1 \cdots b_i \cdots b_k - i \to b_1 \cdots \overline{b_i} \cdots b_k$.

Theorem [Axiotis, Fotakis 16] : there exist connected temporal graphs of size $\Theta(n^2)$ without any $o(n^2)$ spanner.

Theorem [Casteigts et al 19] : a temporal clique with pairwise distinct time labels admits an $O(n \log n)$ spanner.

Open pb : is this tight ?

Idea (dismountability) : if there exist u, v, w sucht that $\lambda(uv) = \min_{v'} \lambda(uv')$ and $\lambda(vw) = \max_{v'} \lambda(v'w)$, then return Spanner$(G - v) \cup \{uv, vw\}$.

Minimum spanner

Sparse spanner : keep a minimum number of edges preserving connectivity.

Spanner of a (connected) static graph ?

Spanner of a temporal graph [Kempe et al 02] ?

Exercise : minimum spanner of the hypercube of dim. k with temporal edges $b_1 \cdots b_i \cdots b_k - i \rightarrow b_1 \cdots \overline{b_i} \cdots b_k$.

Theorem [Axiotis, Fotakis 16] : there exist connected temporal graphs of size $\Theta(n^2)$ without any $o(n^2)$ spanner.

Theorem [Casteigts et al 19] : a temporal clique with pairwise distinct time labels admits an $O(n \log n)$ spanner.

Open pb : is this tight ?

Idea (dismountability) : if there exist u, v, w sucht that $\lambda(uv) = \min_{v'} \lambda(uv')$ and $\lambda(vw) = \max_{v'} \lambda(v'w)$, then return Spanner$(G - v) \cup \{uv, vw\}$.

Minimum spanner

Sparse spanner : keep a minimum number of edges preserving connectivity.

Spanner of a (connected) static graph?

Spanner of a temporal graph [Kempe et al 02]?

Exercise : minimum spanner of the hypercube of dim. k with temporal edges $b_1 \cdots b_i \cdots b_k - i \to b_1 \cdots \overline{b_i} \cdots b_k$.

Theorem [Axiotis, Fotakis 16] : there exist connected temporal graphs of size $\Theta(n^2)$ without any $o(n^2)$ spanner.

Theorem [Casteigts et al 19] : a temporal clique with pairwise distinct time labels admits an $O(n \log n)$ spanner.

Open pb : is this tight?

Idea (dismountability) : if there exist u, v, w sucht that $\lambda(uv) = \min_{v'} \lambda(uv')$ and $\lambda(vw) = \max_{v'} \lambda(v'w)$, then return Spanner$(G - v) \cup \{uv, vw\}$.

Minimum spanner

Sparse spanner : keep a minimum number of edges preserving connectivity.

Spanner of a (connected) static graph?

Spanner of a temporal graph [Kempe et al 02]?

Exercise : minimum spanner of the hypercube of dim. k with temporal edges $b_1 \cdots b_i \cdots b_k - i \rightarrow b_1 \cdots \overline{b_i} \cdots b_k$.

Theorem [Axiotis, Fotakis 16] : there exist connected temporal graphs of size $\Theta(n^2)$ without any $o(n^2)$ spanner.

Theorem [Casteigts et al 19] : a temporal clique with pairwise distinct time labels admits an $O(n \log n)$ spanner.

Open pb : is this tight?

Idea (dismountability) : if there exist u, v, w sucht that $\lambda(uv) = \min_{v'} \lambda(uv')$ and $\lambda(vw) = \max_{v'} \lambda(v'w)$, then return Spanner$(G - v) \cup \{uv, vw\}$.

Minimum spanner

Sparse spanner : keep a minimum number of edges preserving connectivity.

Spanner of a (connected) static graph ?

Spanner of a temporal graph [Kempe et al 02] ?

Exercise : minimum spanner of the hypercube of dim. k with temporal edges $b_1 \cdots b_i \cdots b_k - i \rightarrow b_1 \cdots \overline{b_i} \cdots b_k$.

Theorem [Axiotis, Fotakis 16] : there exist connected temporal graphs of size $\Theta(n^2)$ without any $o(n^2)$ spanner.

Theorem [Casteigts et al 19] : a temporal clique with pairwise distinct time labels admits an $O(n \log n)$ spanner.

Open pb : is this tight ?

Idea (dismountability) : if there exist u, v, w sucht that $\lambda(uv) = \min_{v'} \lambda(uv')$ and $\lambda(vw) = \max_{v'} \lambda(v'w)$, then return Spanner$(G - v) \cup \{uv, vw\}$.

Minimum spanner

Sparse spanner : keep a minimum number of edges preserving connectivity.

Spanner of a (connected) static graph ?

Spanner of a temporal graph [Kempe et al 02] ?

Exercise : minimum spanner of the hypercube of dim. k with temporal edges $b_1 \cdots b_i \cdots b_k - i \rightarrow b_1 \cdots \overline{b_i} \cdots b_k$.

Theorem [Axiotis, Fotakis 16] : there exist connected temporal graphs of size $\Theta(n^2)$ without any $o(n^2)$ spanner.

Theorem [Casteigts et al 19] : a temporal clique with pairwise distinct time labels admits an $O(n \log n)$ spanner.

Open pb : is this tight ?

Idea (dismountability) : if there exist u, v, w sucht that $\lambda(uv) = \min_{v'} \lambda(uv')$ and $\lambda(vw) = \max_{v'} \lambda(v'w)$, then return Spanner$(G - v) \cup \{uv, vw\}$.

Minimum spanner

Sparse spanner : keep a minimum number of edges preserving connectivity.

Spanner of a (connected) static graph?

Spanner of a temporal graph [Kempe et al 02]?

Exercise : minimum spanner of the hypercube of dim. k with temporal edges $b_1 \cdots b_i \cdots b_k - i \to b_1 \cdots \overline{b_i} \cdots b_k$.

Theorem [Axiotis, Fotakis 16] : there exist connected temporal graphs of size $\Theta(n^2)$ without any $o(n^2)$ spanner.

Theorem [Casteigts et al 19] : a temporal clique with pairwise distinct time labels admits an $O(n \log n)$ spanner.

Open pb : is this tight?

Idea (dismountability) : if there exist u, v, w sucht that $\lambda(uv) = \min_{v'} \lambda(uv')$ and $\lambda(vw) = \max_{v'} \lambda(v'w)$, then return Spanner$(G - v) \cup \{uv, vw\}$.

Minimum spanner

Sparse spanner : keep a minimum number of edges preserving connectivity.

Spanner of a (connected) static graph ?

Spanner of a temporal graph [Kempe et al 02] ?

Exercise : minimum spanner of the hypercube of dim. k with temporal edges $b_1 \cdots b_i \cdots b_k - i \rightarrow b_1 \cdots \overline{b_i} \cdots b_k$.

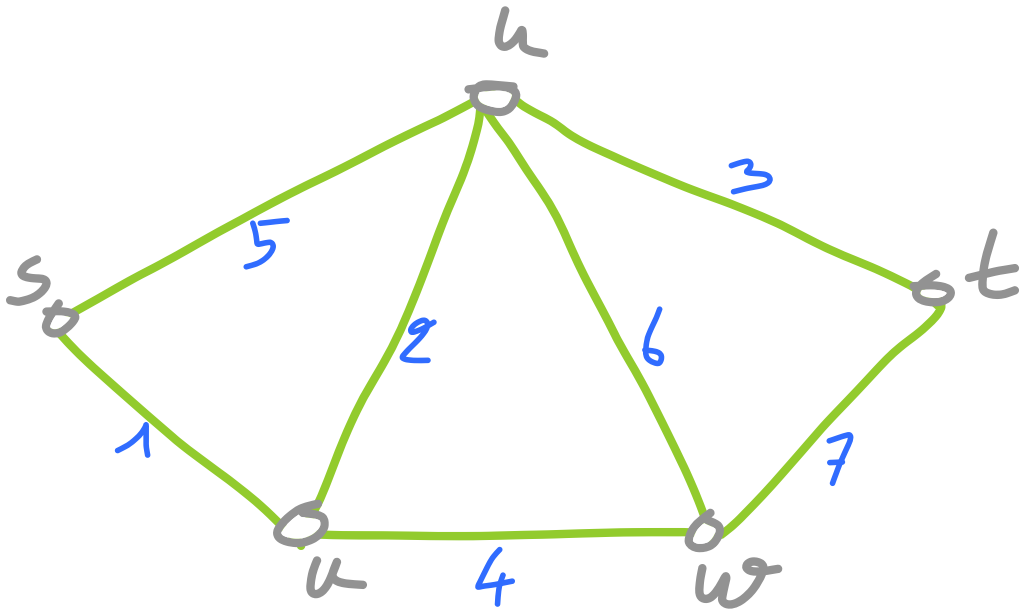
Theorem [Axiotis, Fotakis 16] : there exist connected temporal graphs of size $\Theta(n^2)$ without any $o(n^2)$ spanner.

Theorem [Casteigts et al 19] : a temporal clique with pairwise distinct time labels admits an $O(n \log n)$ spanner.

Open pb : is this tight ?

Idea (dismountability) : if there exist u, v, w sucht that $\lambda(uv) = \min_{v'} \lambda(uv')$ and $\lambda(vw) = \max_{v'} \lambda(v'w)$, then return Spanner$(G - v) \cup \{uv, vw\}$.

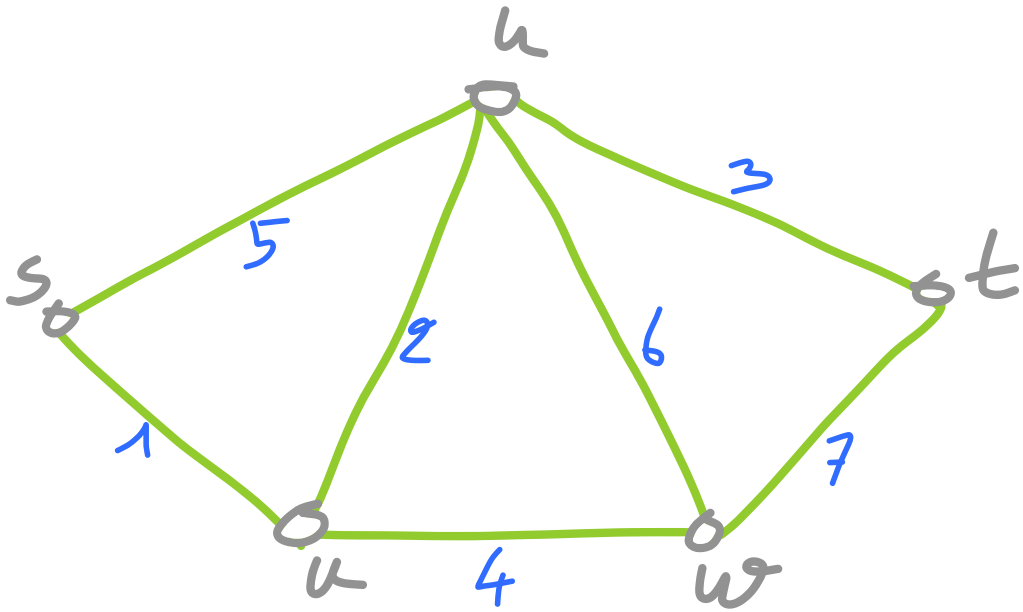
No natural node-version of Menger



There exists temporal graphs where the number of node-disjoint uv-temporal-paths is less than the minimum number of node deletions disconnecting u and v [Kempe et al 02].

Theorem [Kempe et al 02] : computing two node-disjoint uv-temporal-paths is NP-hard.

No natural node-version of Menger

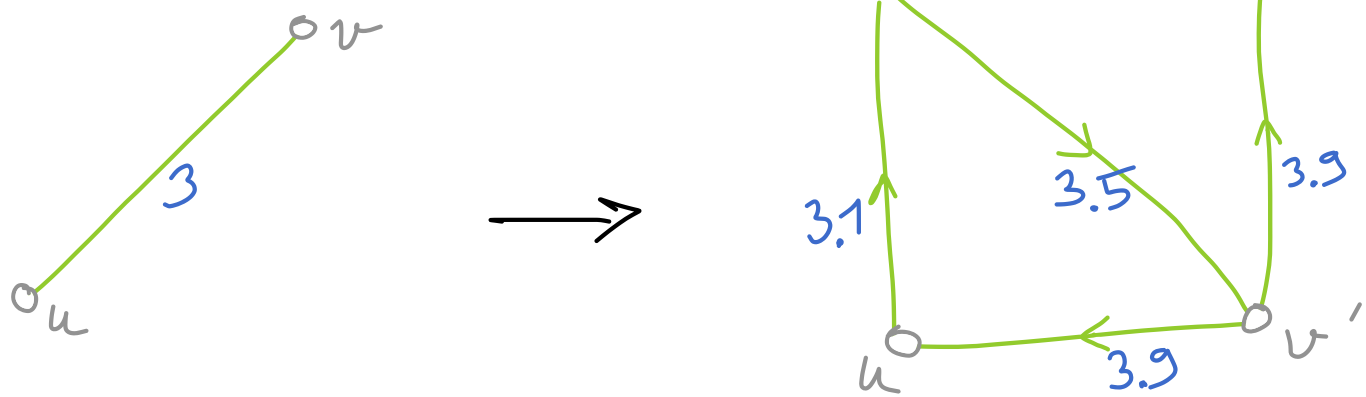


There exists temporal graphs where the number of node-disjoint uv-temporal-paths is less than the minimum number of node deletions disconnecting u and v [Kempe et al 02].

Theorem [Kempe et al 02] : computing two node-disjoint uv-temporal-paths is NP-hard.

Other versions of Menger

The number of temporal-edge-disjoint uv-temporal-paths always equals the minimum number of edge deletions disconnecting u and v.

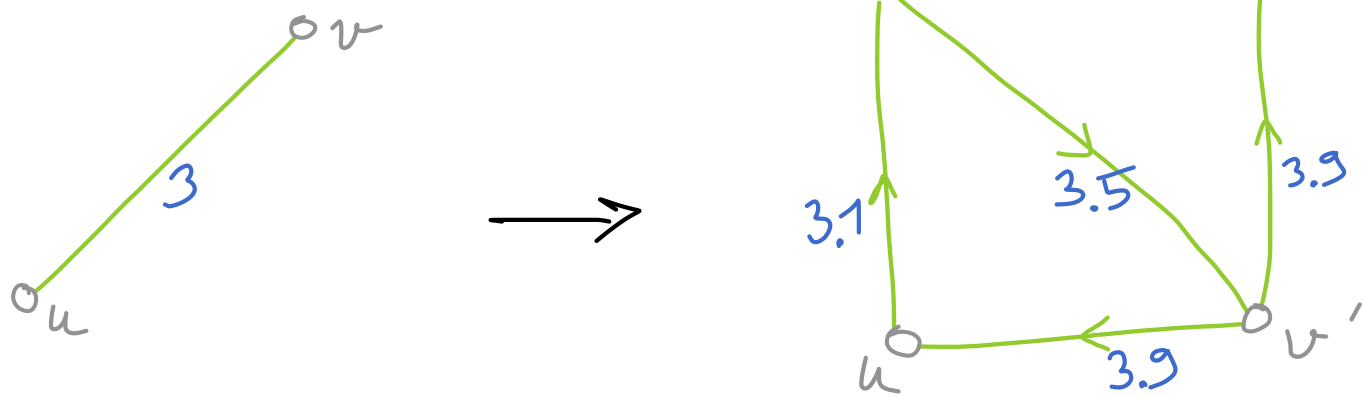


The number of node-departure-disjoint uv-temporal-paths always equals the minimum number of node-departure deletions disconnecting u and v [Mertzios et al 13].

Other problems : see [Michail 16].

Other versions of Menger

The number of temporal-edge-disjoint uv-temporal-paths always equals the minimum number of edge deletions disconnecting u and v.

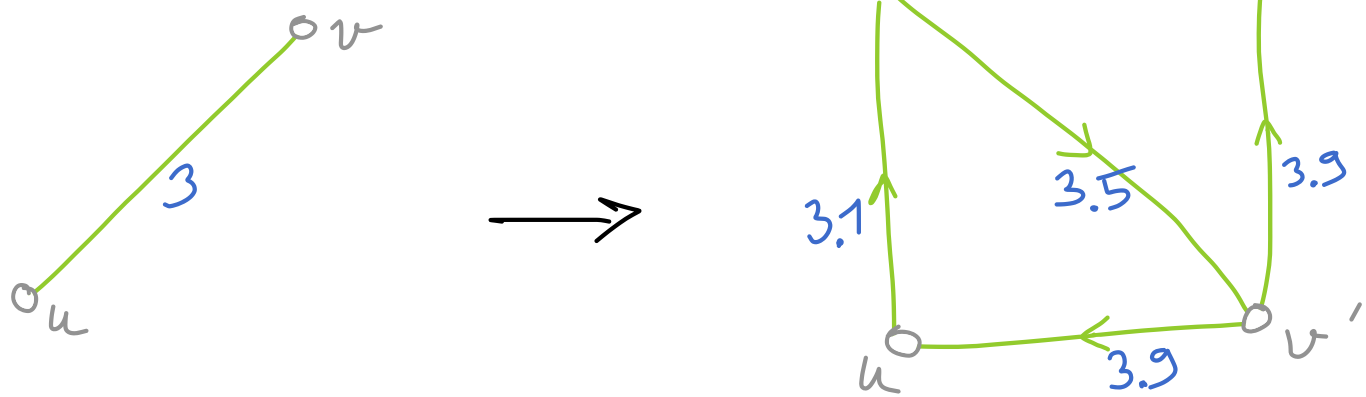


The number of node-departure-disjoint uv-temporal-paths always equals the minimum number of node-departure deletions disconnecting u and v [Mertzios et al 13].

Other problems : see [Michail 16].

Other versions of Menger

The number of temporal-edge-disjoint uv-temporal-paths always equals the minimum number of edge deletions disconnecting u and v.

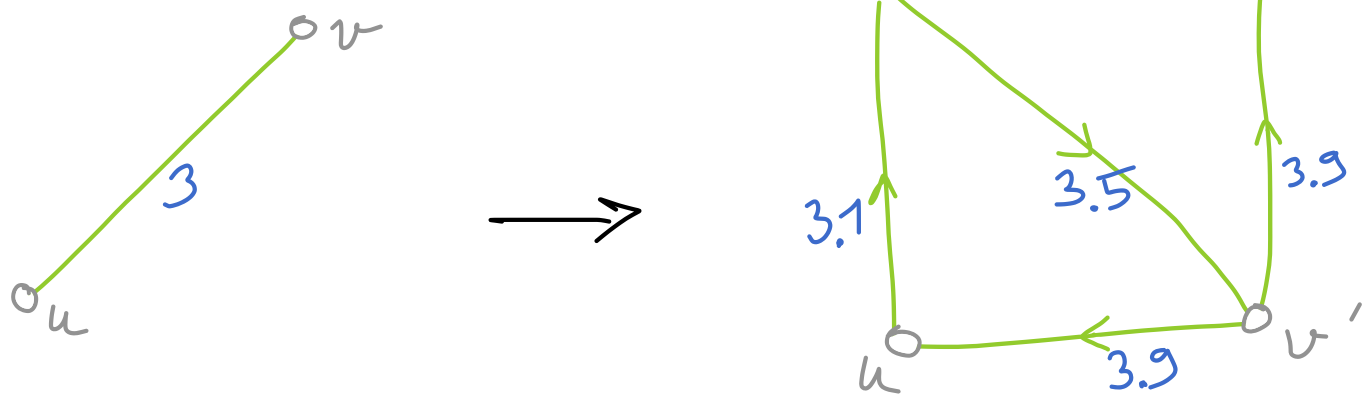


The number of node-departure-disjoint uv-temporal-paths always equals the minimum number of node-departure deletions disconnecting u and v [Mertzios et al 13].

Other problems : see [Michail 16].

Other versions of Menger

The number of temporal-edge-disjoint uv-temporal-paths always equals the minimum number of edge deletions disconnecting u and v.



The number of node-departure-disjoint uv-temporal-paths always equals the minimum number of node-departure deletions disconnecting u and v [Mertzios et al 13].

Other problems : see [Michail 16].

Public transit networks

Connection Scan Algorithm (CSA) [Dibbelt, Pajor, Strasser, Wagner '13]

Observation : the time expanded graph is acyclic.

Sort connections by departure time.

For a (earliest arrival time) query from s to t at time τ_{dep} :
- $EAT(s) := \tau_{dep}$ (earliest arrival time at a stop);
- for each connection (u, v, τ, δ), if $EAT(u) \leq \tau$ then :
  - $EAT(v) := \min(EAT(v), \tau_{arr}$ where $\tau_{arr} = \tau + \delta)$,
  - for each footpath (v, w, δ'),

$EAT(w) := \min(EAT(z), \tau_{arr} + \delta')$.

Quite fast (few ms) when few footpaths.

Exercise : propose simple optimization for reducing the number of connections scanned.

Connection Scan Algorithm (CSA) [Dibbelt, Pajor, Strasser, Wagner '13]

Observation : the time expanded graph is acyclic.

Sort connections by departure time.

For a (earliest arrival time) query from s to t at time τ_{dep} :

- $EAT(s) := \tau_{dep}$ (earliest arrival time at a stop);
- for each connection (u, v, τ, δ), if $EAT(u) \leq \tau$ then :
  - $EAT(v) := \min(EAT(v), \tau_{arr}$ where $\tau_{arr} = \tau + \delta)$,
  - for each footpath (v, w, δ'),

$EAT(w) := \min(EAT(z), \tau_{arr} + \delta')$.

Quite fast (few ms) when few footpaths.

Exercise : propose simple optimization for reducing the number of connections scanned.

Connection Scan Algorithm (CSA) [Dibbelt, Pajor, Strasser, Wagner '13]

Observation : the time expanded graph is acyclic.

Sort connections by departure time.

For a (earliest arrival time) query from s to t at time τ_{dep} :

- $EAT(s) := \tau_{dep}$ (earliest arrival time at a stop);
- for each connection (u, v, τ, δ), if $EAT(u) \leq \tau$ then :
  - $EAT(v) := \min(EAT(v), \tau_{arr}$ where $\tau_{arr} = \tau + \delta)$,
  - for each footpath (v, w, δ'),

$EAT(w) := \min(EAT(z), \tau_{arr} + \delta')$.

Quite fast (few ms) when few footpaths.

Exercise : propose simple optimization for reducing the number of connections scanned.

Connection Scan Algorithm (CSA) [Dibbelt, Pajor, Strasser, Wagner '13]

Observation : the time expanded graph is acyclic.

Sort connections by departure time.

For a (earliest arrival time) query from s to t at time τ_{dep} :
- $EAT(s) := \tau_{dep}$ (earliest arrival time at a stop);
- for each connection (u, v, τ, δ), if $EAT(u) \leq \tau$ then :
  - $EAT(v) := \min(EAT(v), \tau_{arr}$ where $\tau_{arr} = \tau + \delta)$,
  - for each footpath (v, w, δ'),

$EAT(w) := \min(EAT(z), \tau_{arr} + \delta')$.

Quite fast (few ms) when few footpaths.

Exercise : propose simple optimization for reducing the number of connections scanned.

Connection Scan Algorithm (CSA) [Dibbelt, Pajor, Strasser, Wagner '13]

Observation : the time expanded graph is acyclic.

Sort connections by departure time.

For a (earliest arrival time) query from s to t at time τ_{dep} :

- $EAT(s) := \tau_{dep}$ (earliest arrival time at a stop);
- for each connection (u, v, τ, δ), if $EAT(u) \leq \tau$ then :
  - $EAT(v) := \min(EAT(v), \tau_{arr}$ where $\tau_{arr} = \tau + \delta)$,
  - for each footpath (v, w, δ'),

$EAT(w) := \min(EAT(z), \tau_{arr} + \delta')$.

Quite fast (few ms) when few footpaths.

Exercise : propose simple optimization for reducing the number of connections scanned.

Round-Based Public Transit Routing (RAPTOR) [Delling, Pajor, Werneck '12]

Idea : follow the trips.

For a (earliest arrival time) query from u to v at time t :

- for each line ℓ stopping at u,
- find the first trip of ℓ stopping at u after t,
- update earliest arrival time at following stops in the trip.

Repeat :
In k rounds, compute earliest arrival time with $k-1$ transfers at most.

Quite fast (few ms) when few footpaths.

Bonus : it provides a Pareto set of solutions :
A solution arriving at t' in v with k' transfers is pareto optimal if there is no solution t'', k'' with $t'' < t'$ and $k'' < k'$.

Round-Based Public Transit Routing (RAPTOR) [Delling, Pajor, Werneck '12]

Idea : follow the trips.

For a (earliest arrival time) query from u to v at time t :

- for each line ℓ stopping at u,
- find the first trip of ℓ stopping at u after t,
- update earliest arrival time at following stops in the trip.

Repeat :
In k rounds, compute earliest arrival time with $k-1$ transfers at most.

Quite fast (few ms) when few footpaths.

Bonus : it provides a Pareto set of solutions :
A solution arriving at t' in v with k' transfers is pareto optimal if there is no solution t'', k'' with $t'' < t'$ and $k'' < k'$.

Round-Based Public Transit Routing (RAPTOR) [Delling, Pajor, Werneck '12]

Idea : follow the trips.

For a (earliest arrival time) query from u to v at time t :

- for each line ℓ stopping at u,
- find the first trip of ℓ stopping at u after t,
- update earliest arrival time at following stops in the trip.

Repeat :
In k rounds, compute earliest arrival time with $k-1$ transfers at most.

Quite fast (few ms) when few footpaths.

Bonus : it provides a Pareto set of solutions :
A solution arriving at t' in v with k' transfers is pareto optimal if there is no solution t'', k'' with $t'' < t'$ and $k'' < k'$.

Round-Based Public Transit Routing (RAPTOR) [Delling, Pajor, Werneck '12]

Idea : follow the trips.

For a (earliest arrival time) query from u to v at time t :

- for each line ℓ stopping at u,
- find the first trip of ℓ stopping at u after t,
- update earliest arrival time at following stops in the trip.

Repeat :
In k rounds, compute earliest arrival time with $k-1$ transfers at most.

Quite fast (few ms) when few footpaths.

Bonus : it provides a Pareto set of solutions :
A solution arriving at t' in v with k' transfers is pareto optimal if there is no solution t'', k'' with $t'' < t'$ and $k'' < k'$.

Round-Based Public Transit Routing (RAPTOR) [Delling, Pajor, Werneck '12]

Idea : follow the trips.

For a (earliest arrival time) query from u to v at time t :

- for each line ℓ stopping at u,
- find the first trip of ℓ stopping at u after t,
- update earliest arrival time at following stops in the trip.

Repeat :
In k rounds, compute earliest arrival time with $k-1$ transfers at most.

Quite fast (few ms) when few footpaths.

Bonus : it provides a Pareto set of solutions :
A solution arriving at t' in v with k' transfers is pareto optimal if there is no solution t'', k'' with $t'' < t'$ and $k'' < k'$.

Transfer Patterns [Bast, Carlsson, Eigenwillig, Geisberger, Harrelson, Raychev, Viger '16]

Pre-compute for each station u :

- lines going through u and position on line,
- transfer pattern DAG : union of transfer paths

$u \to x \to y \to v$ for all v (and departure time τ).

For a query from u to v :

- try direct connection (common line?),
- otherwise try all possible paths in the DAG of u.

Optimization : compute global transfer pattern DAG only for a selected set of (common) "landmarks".
Prune transfer pattern DAG of u at landmarks (local computation).
For a query, use the union of transfer pattern DAGs of u and v.

Experimental results (New York) : precomput. 800h, 1.5Go, 5-20 transf. pat. / station pair, query time 10ms.

Transfer Patterns [Bast, Carlsson, Eigenwillig, Geisberger, Harrelson, Raychev, Viger '16]

Pre-compute for each station u :

- lines going through u and position on line,
- transfer pattern DAG : union of transfer paths

$u \to x \to y \to v$ for all v (and departure time τ).

For a query from u to v :

- try direct connection (common line?),
- otherwise try all possible paths in the DAG of u.

Optimization : compute global transfer pattern DAG only for a selected set of (common) "landmarks".
Prune transfer pattern DAG of u at landmarks (local computation).
For a query, use the union of transfer pattern DAGs of u and v.

Experimental results (New York) : precomput. 800h, 1.5Go, 5-20 transf. pat. / station pair, query time 10ms.

Transfer Patterns [Bast, Carlsson, Eigenwillig, Geisberger, Harrelson, Raychev, Viger '16]

Pre-compute for each station u :

- lines going through u and position on line,
- transfer pattern DAG : union of transfer paths

$u \to x \to y \to v$ for all v (and departure time τ).

For a query from u to v :

- try direct connection (common line?),
- otherwise try all possible paths in the DAG of u.

Optimization : compute global transfer pattern DAG only for a selected set of (common) "landmarks".
Prune transfer pattern DAG of u at landmarks (local computation).
For a query, use the union of transfer pattern DAGs of u and v.

Experimental results (New York) : precomput. 800h, 1.5Go, 5-20 transf. pat. / station pair, query time 10ms.

Transfer Patterns [Bast, Carlsson, Eigenwillig, Geisberger, Harrelson, Raychev, Viger '16]

Pre-compute for each station u :

- lines going through u and position on line,
- transfer pattern DAG : union of transfer paths

$u \to x \to y \to v$ for all v (and departure time τ).

For a query from u to v :

- try direct connection (common line?),
- otherwise try all possible paths in the DAG of u.

Optimization : compute global transfer pattern DAG only for a selected set of (common) "landmarks".
Prune transfer pattern DAG of u at landmarks (local computation).
For a query, use the union of transfer pattern DAGs of u and v.

Experimental results (New York) : precomput. 800h, 1.5Go, 5-20 transf. pat. / station pair, query time 10ms.

Public Transit Labeling [Delling, Dibbelt, Pajor, Werneck '15]

HL on the time expanded graph :

- a node for each event at a station,
- waiting arcs between consecutive nodes of a station,
- connection arcs for each connection.

Query from u to v at τ :

- find (by dichotomy) the first event u′ at u after τ,
- the first event v′ at v s.t. $H_{u'} \cap H_{v'} \neq \emptyset$,
- expand the path through the best hub in $H_{u'} \cap H_{v'}$.

Experimental results (London, earliest arrival – multicriteria) : precomput. 1-49h, 1.3-28Go, hub sets 70-700/event, query time 10-30μs.

Similar : direct approach [Wang, Lin, Yang, Xiao, Zhou '15] : HHL + timetable rel. opt.

Open : what if paths are repeatedly the same ?

Public Transit Labeling [Delling, Dibbelt, Pajor, Werneck '15]

HL on the time expanded graph :

- a node for each event at a station,
- waiting arcs between consecutive nodes of a station,
- connection arcs for each connection.

Query from u to v at τ :

- find (by dichotomy) the first event u′ at u after τ,
- the first event v′ at v s.t. $H_{u'} \cap H_{v'} \neq \emptyset$,
- expand the path through the best hub in $H_{u'} \cap H_{v'}$.

Experimental results (London, earliest arrival – multicriteria) : precomput. 1-49h, 1.3-28Go, hub sets 70-700/event, query time 10-30μs.

Similar : direct approach [Wang, Lin, Yang, Xiao, Zhou '15] : HHL + timetable rel. opt.

Open : what if paths are repeatedly the same ?

Public Transit Labeling [Delling, Dibbelt, Pajor, Werneck '15]

HL on the time expanded graph :

- a node for each event at a station,
- waiting arcs between consecutive nodes of a station,
- connection arcs for each connection.

Query from u to v at τ :

- find (by dichotomy) the first event u′ at u after τ,
- the first event v′ at v s.t. $H_{u'} \cap H_{v'} \neq \emptyset$,
- expand the path through the best hub in $H_{u'} \cap H_{v'}$.

Experimental results (London, earliest arrival – multicriteria) : precomput. 1-49h, 1.3-28Go, hub sets 70-700/event, query time 10-30μs.

Similar : direct approach [Wang, Lin, Yang, Xiao, Zhou '15] : HHL + timetable rel. opt.

Open : what if paths are repeatedly the same ?

Public Transit Labeling [Delling, Dibbelt, Pajor, Werneck '15]

HL on the time expanded graph :

- a node for each event at a station,
- waiting arcs between consecutive nodes of a station,
- connection arcs for each connection.

Query from u to v at τ :

- find (by dichotomy) the first event u′ at u after τ,
- the first event v′ at v s.t. $H_{u'} \cap H_{v'} \neq \emptyset$,
- expand the path through the best hub in $H_{u'} \cap H_{v'}$.

Experimental results (London, earliest arrival – multicriteria) : precomput. 1-49h, 1.3-28Go, hub sets 70-700/event, query time 10-30μs.

Similar : direct approach [Wang, Lin, Yang, Xiao, Zhou '15] : HHL + timetable rel. opt.

Open : what if paths are repeatedly the same ?

Public Transit Labeling [Delling, Dibbelt, Pajor, Werneck '15]

HL on the time expanded graph :

- a node for each event at a station,
- waiting arcs between consecutive nodes of a station,
- connection arcs for each connection.

Query from u to v at τ :

- find (by dichotomy) the first event u′ at u after τ,
- the first event v′ at v s.t. $H_{u'} \cap H_{v'} \neq \emptyset$,
- expand the path through the best hub in $H_{u'} \cap H_{v'}$.

Experimental results (London, earliest arrival – multicriteria) : precomput. 1-49h, 1.3-28Go, hub sets 70-700/event, query time 10-30μs.

Similar : direct approach [Wang, Lin, Yang, Xiao, Zhou '15] : HHL + timetable rel. opt.

Open : what if paths are repeatedly the same ?

Further reading

[Holme 2015]
Modern temporal network theory : a colloquium.
The European Physical Journal B 2015
https://arxiv.org/abs/1508.01303

[Michail 2016]
An Introduction to Temporal Graphs : An Algorithmic Perspective.
Internet Mathematics 2016
https://arxiv.org/abs/1503.00278

Further reading

[Holme 2015]
Modern temporal network theory : a colloquium.
The European Physical Journal B 2015
https://arxiv.org/abs/1508.01303

[Michail 2016]
An Introduction to Temporal Graphs : An Algorithmic Perspective.
Internet Mathematics 2016
https://arxiv.org/abs/1503.00278

Exercise 4

Show that WL and 2-OWL have the disciminative power (see detailed exercise on the course page).

Exercise 5 (optional)

Show that finding a closed-connected set of size k (for non-strict temporal paths) is W[1]-hard for parameter k.

Suggestion : use a parameterized reduction of k-multicolored-clique : given a graph G, an integer k, and a partition $V_1, \ldots, V_k$ of $V(G)$, decide if there exists a k-clique containing exactly one vertex from each set V_i.

Exercise 5 (optional)

Show that finding a closed-connected set of size k (for non-strict temporal paths) is W[1]-hard for parameter k.

Suggestion : use a parameterized reduction of k-multicolored-clique : given a graph G, an integer k, and a partition $V_1, \ldots, V_k$ of $V(G)$, decide if there exists a k-clique containing exactly one vertex from each set V_i.

Some open problems

Hardness in P :

- Certificate for approximated diameter?
- Other sub-quad./sub-cubic certificates?

Hub labeling :

- Best hub labeling for a grid?
- Lower bounds for hopset-based distance oracles?

Graph neural networks :

- GNNs for graphs with structural properties?
- Generative models?

Temporal graphs :

- Interesting structural properties of temporal graphs (ex : temporal planarity)?
- Conditional lower-bound for shortest duration (with interval availability)?
- Distance/adjacency labeling?

Some open problems

Hardness in P :

- Certificate for approximated diameter?
- Other sub-quad./sub-cubic certificates?

Hub labeling :

- Best hub labeling for a grid?
- Lower bounds for hopset-based distance oracles?

Graph neural networks :

- GNNs for graphs with structural properties?
- Generative models?

Temporal graphs :

- Interesting structural properties of temporal graphs (ex : temporal planarity)?
- Conditional lower-bound for shortest duration (with interval availability)?
- Distance/adjacency labeling?

Some open problems

Hardness in P :

- Certificate for approximated diameter?
- Other sub-quad./sub-cubic certificates?

Hub labeling :

- Best hub labeling for a grid?
- Lower bounds for hopset-based distance oracles?

Graph neural networks :

- GNNs for graphs with structural properties?
- Generative models?

Temporal graphs :

- Interesting structural properties of temporal graphs (ex : temporal planarity)?
- Conditional lower-bound for shortest duration (with interval availability)?
- Distance/adjacency labeling?

Some open problems

Hardness in P :

- Certificate for approximated diameter?
- Other sub-quad./sub-cubic certificates?

Hub labeling :

- Best hub labeling for a grid?
- Lower bounds for hopset-based distance oracles?

Graph neural networks :

- GNNs for graphs with structural properties?
- Generative models?

Temporal graphs :

- Interesting structural properties of temporal graphs (ex : temporal planarity)?
- Conditional lower-bound for shortest duration (with interval availability)?
- Distance/adjacency labeling?