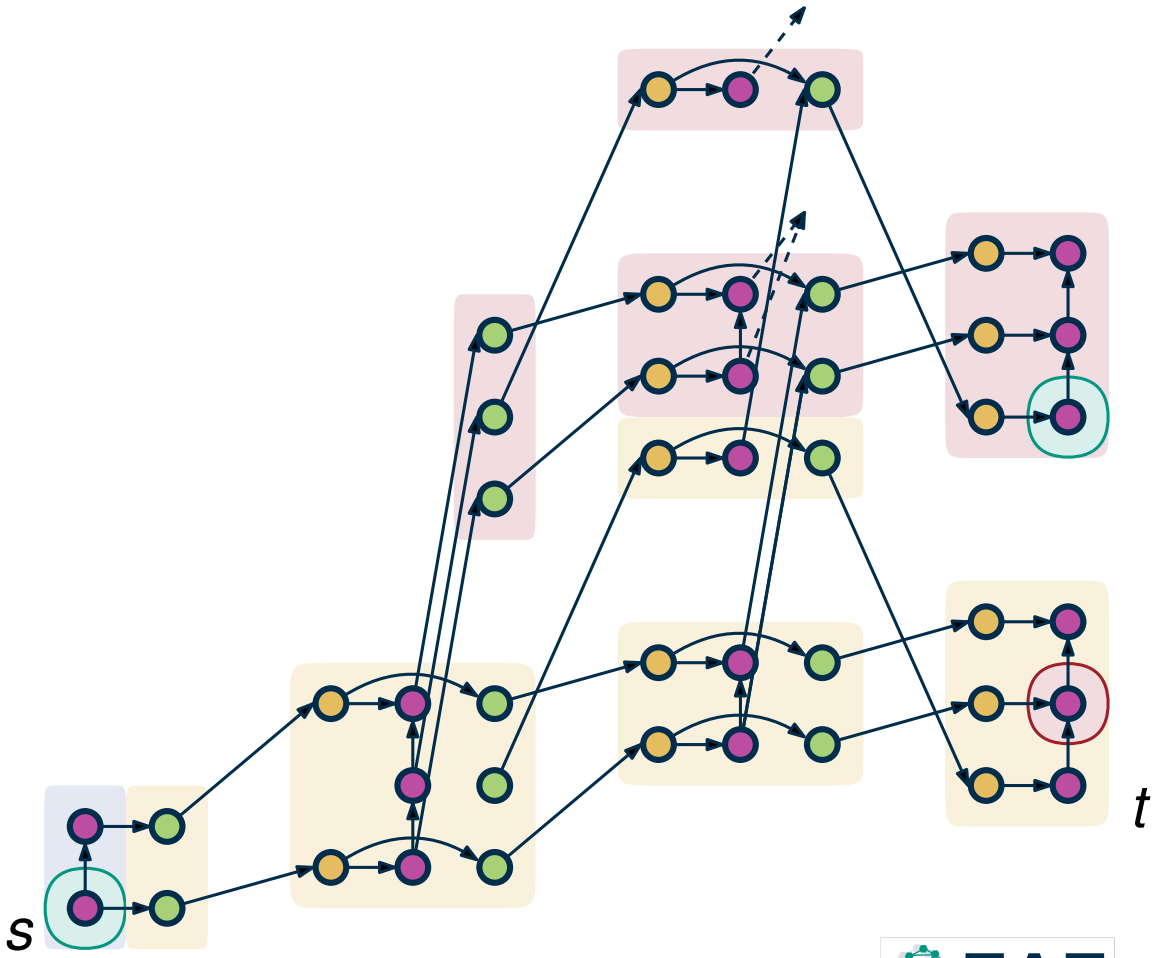
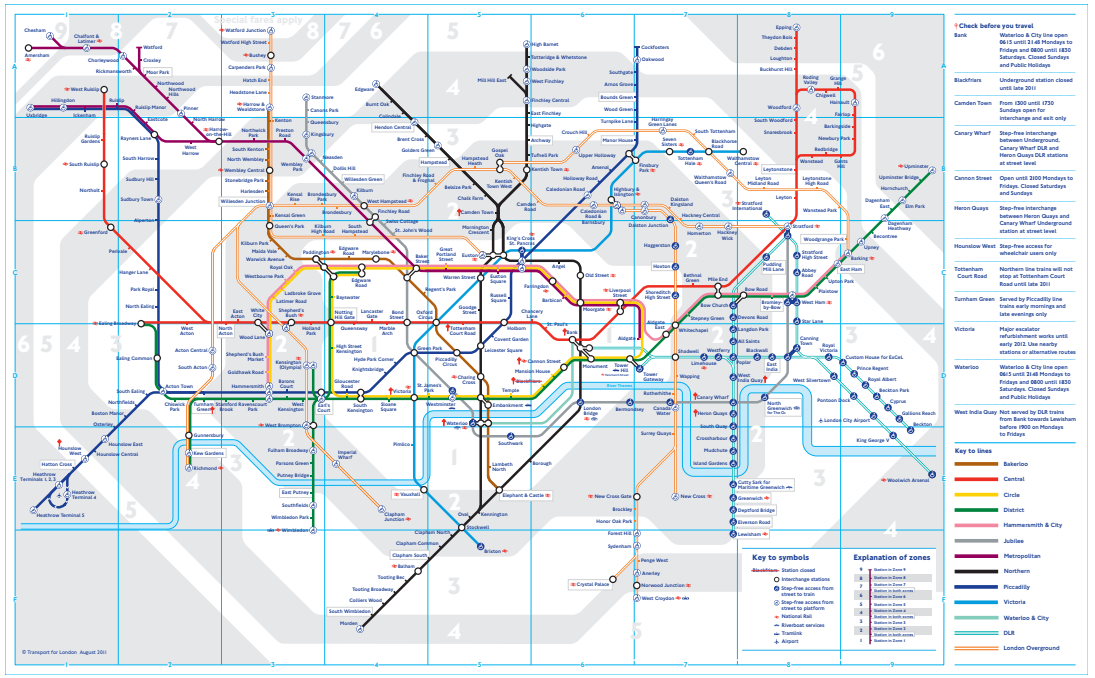
Journey planning in public transportation networks



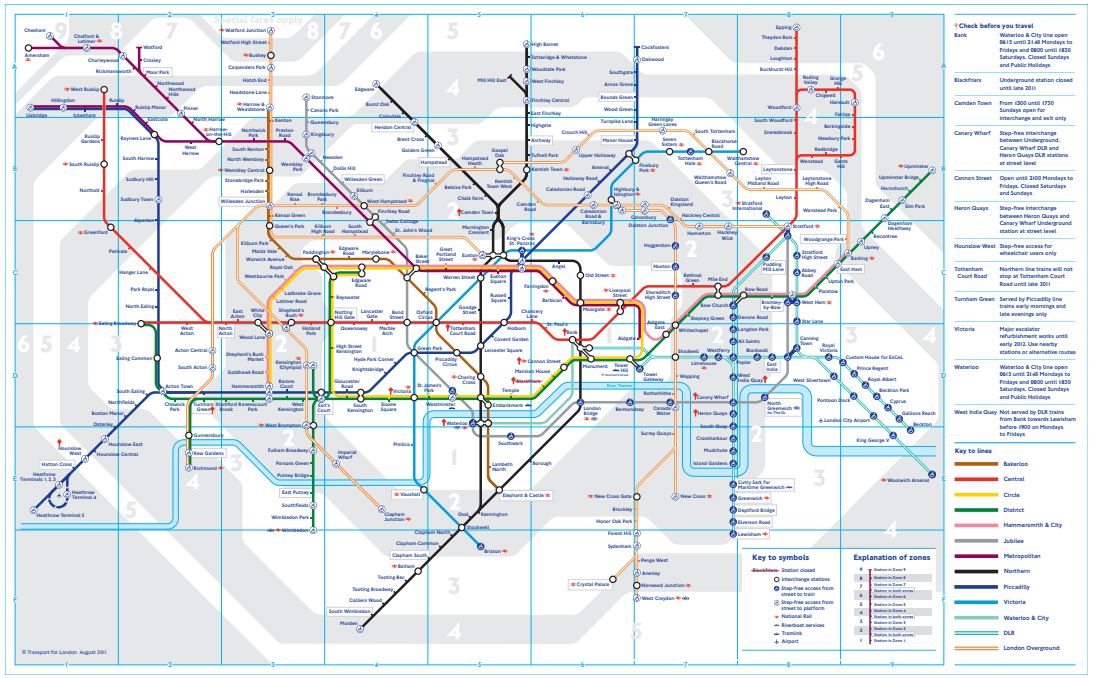
An application of temporal graph reachability

Jonas Sauer | July 6, 2026

Motivation

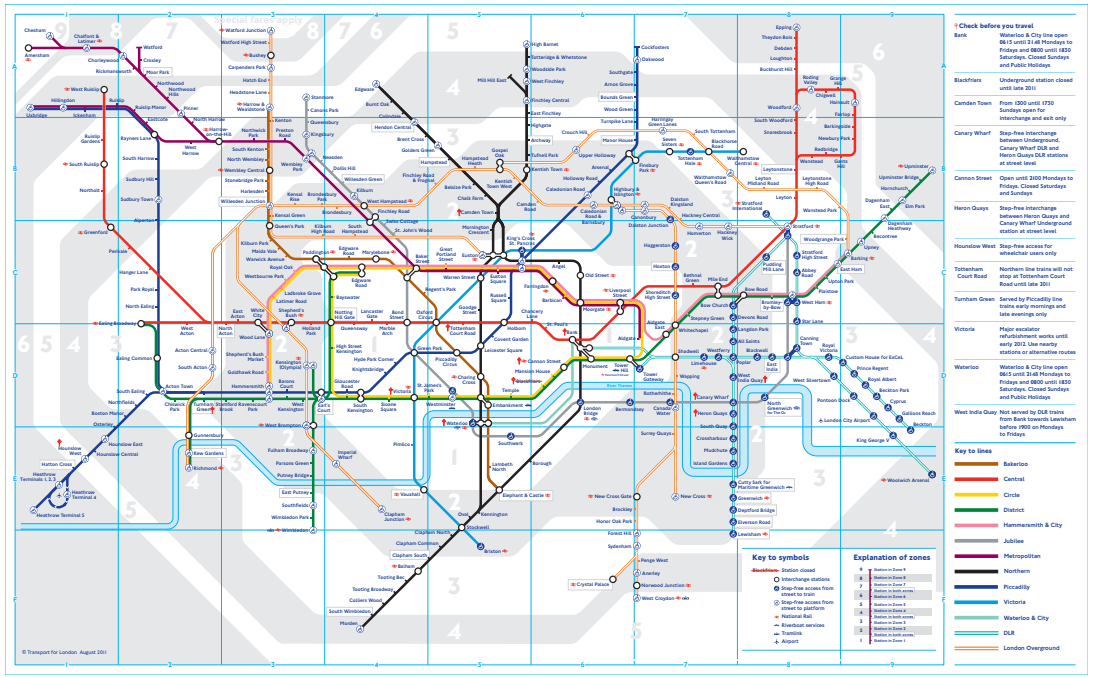


Motivation



Usually framed as a shortest path problem, not a temporal graph problem

Motivation

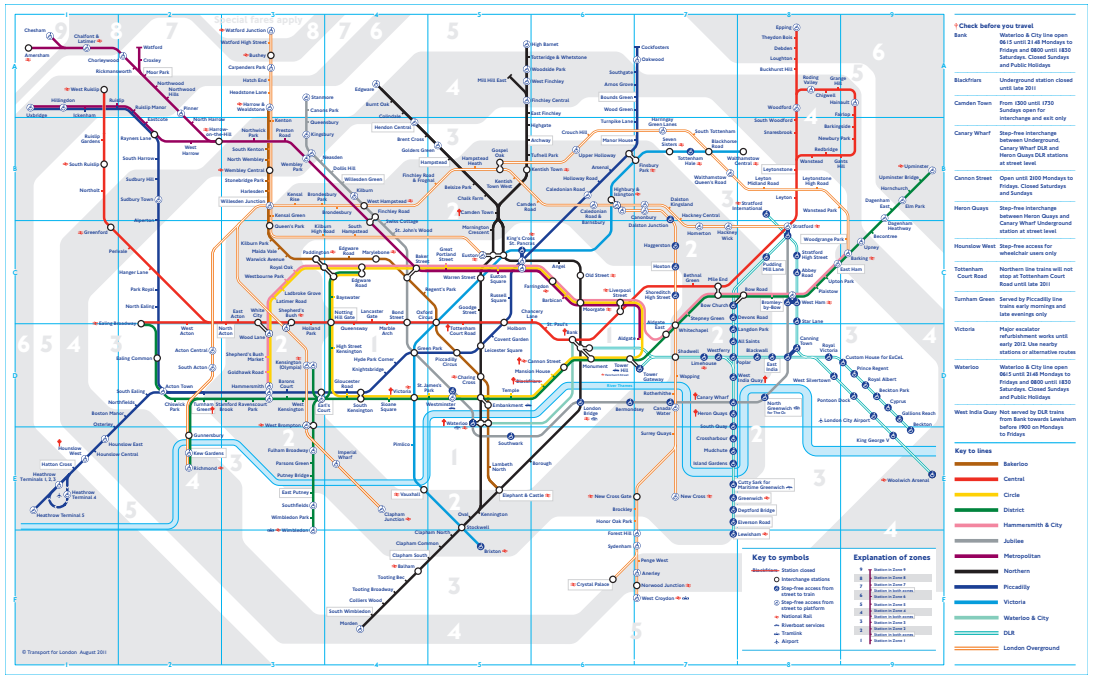


Usually framed as a shortest path problem, not a temporal graph problem

Agenda:

1. Why temporal reachability is the right way to think about this

Motivation

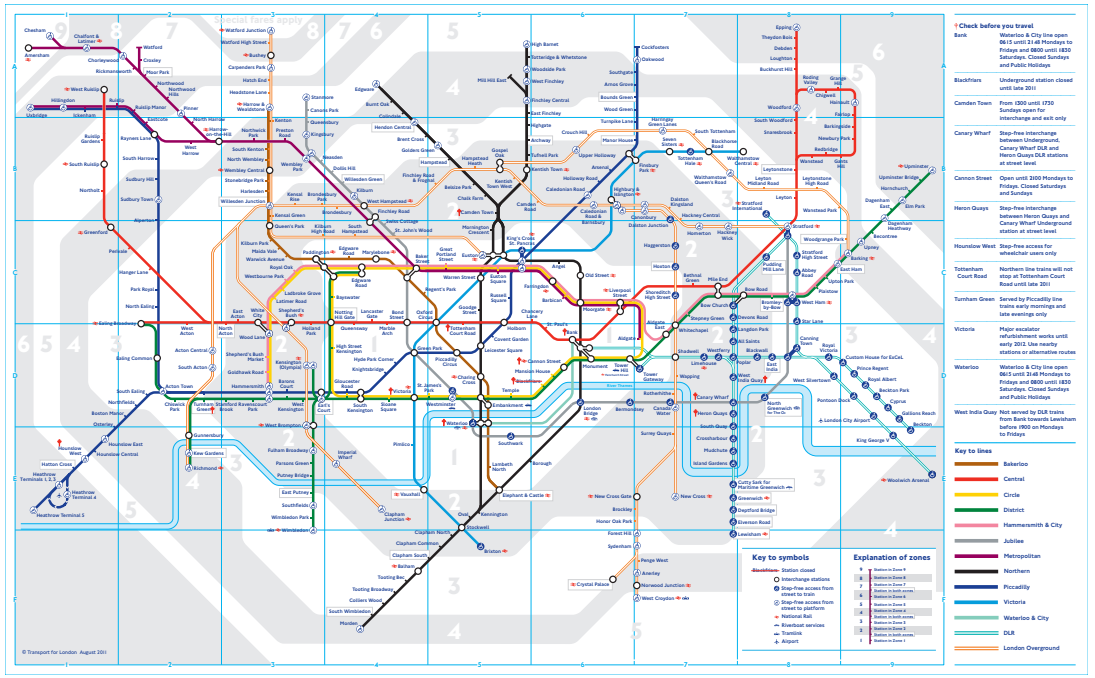


Usually framed as a shortest path problem, not a temporal graph problem

Agenda:

1. Why temporal reachability is the right way to think about this
2. How we make these algorithms really, really fast in practice

Motivation



Usually framed as a shortest path problem, not a temporal graph problem

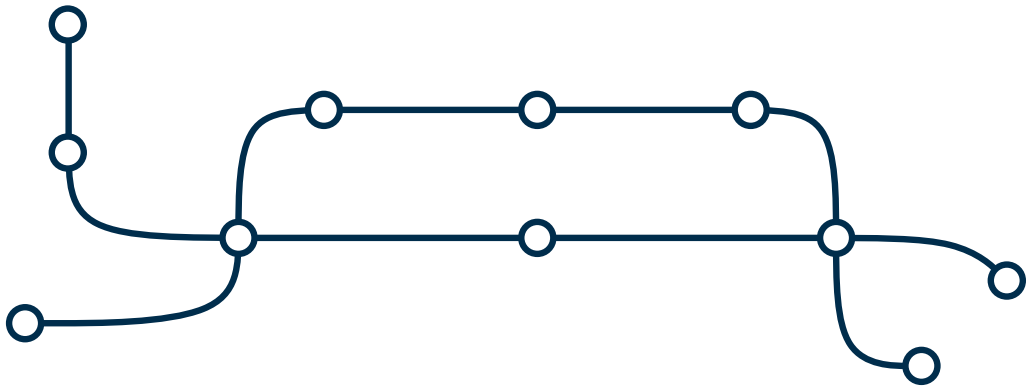
Agenda:

1. Why temporal reachability is the right way to think about this
2. How we make these algorithms really, really fast in practice
3. Why both communities can learn from each other

Public Transit Network

Elements:

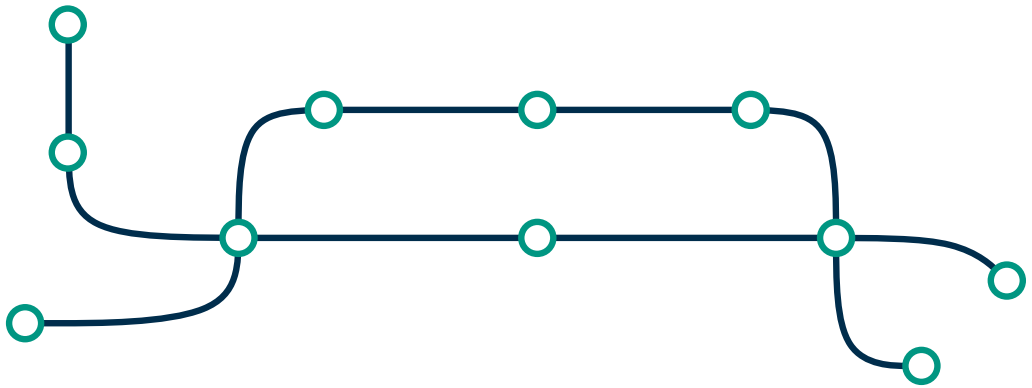
- Stops (bus/train stations)
- Connections between events, forming trips
- Trips with the same stop sequence grouped into lines



Public Transit Network

Elements:

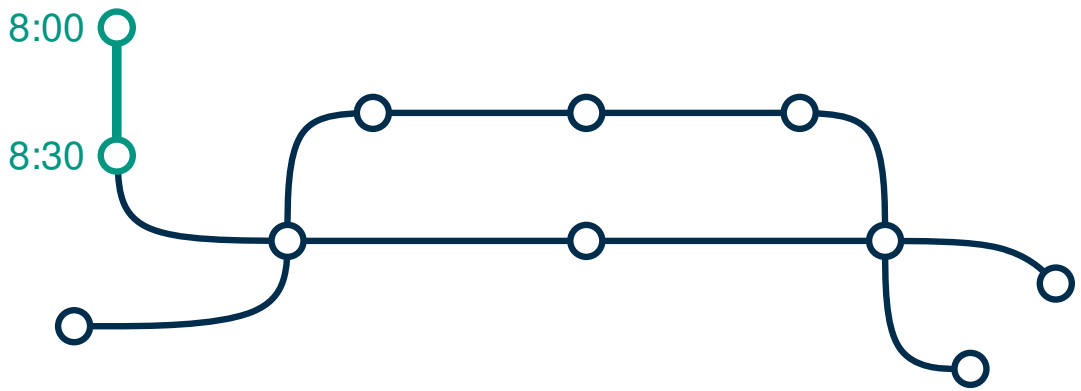
- Stops (bus/train stations)
- Connections between events, forming trips
- Trips with the same stop sequence grouped into lines



Public Transit Network

Elements:

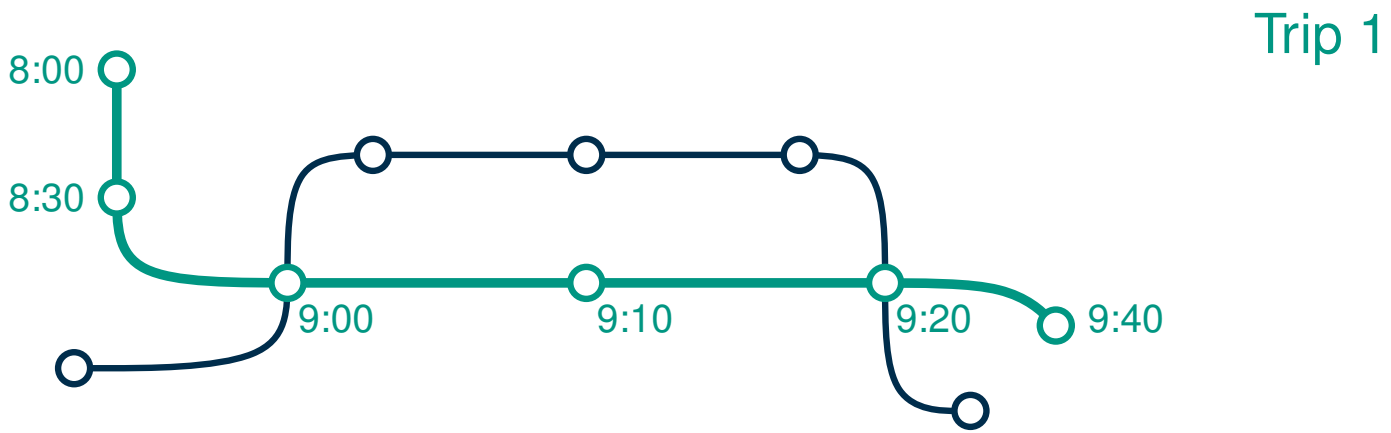
- Stops (bus/train stations)
- Connections between events, forming trips
- Trips with the same stop sequence grouped into lines



Public Transit Network

Elements:

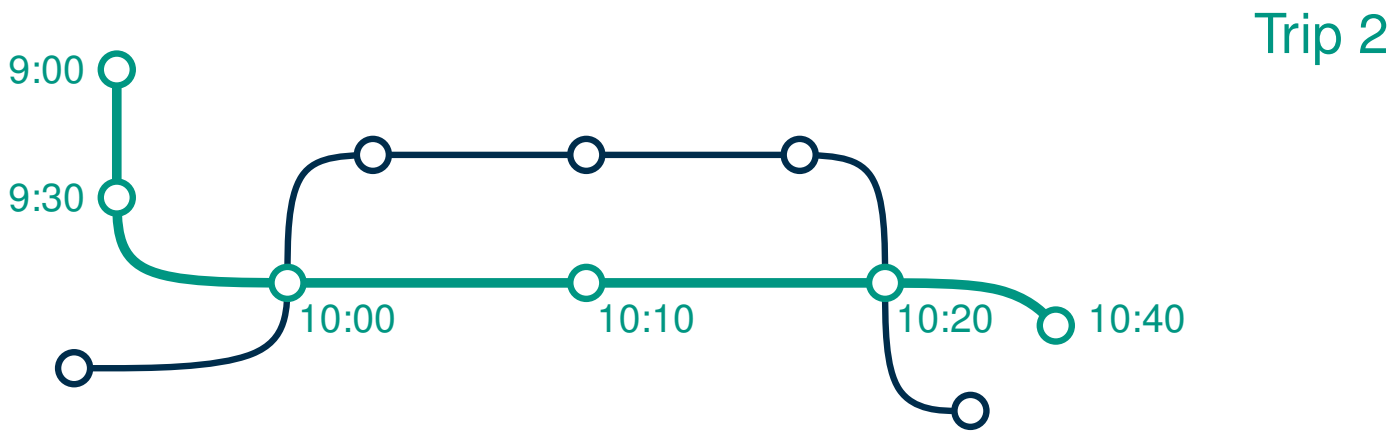
- Stops (bus/train stations)
- Connections between events, forming trips
- Trips with the same stop sequence grouped into lines



Public Transit Network

Elements:

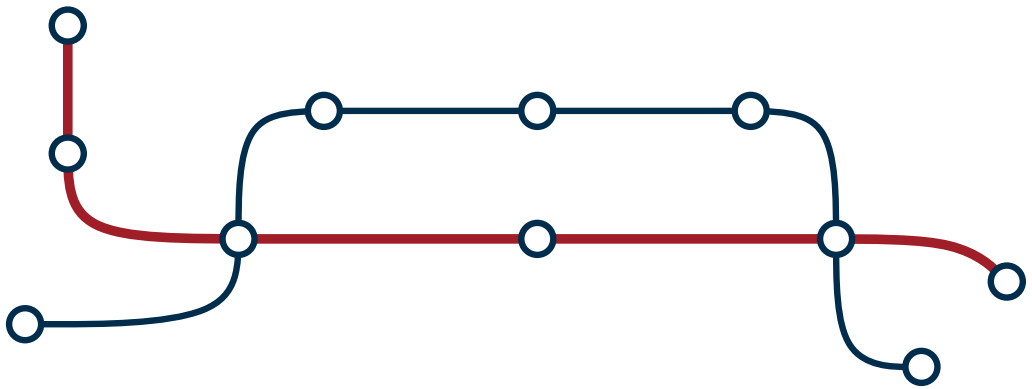
- Stops (bus/train stations)
- Connections between events, forming trips
- Trips with the same stop sequence grouped into lines



Public Transit Network

Elements:

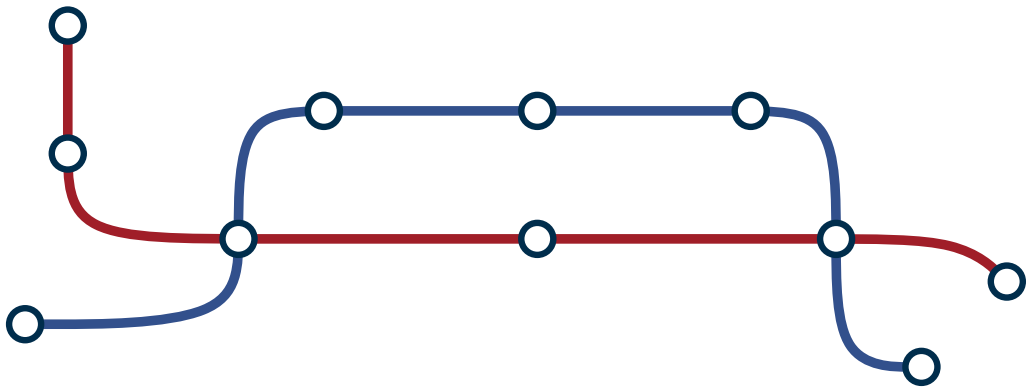
- Stops (bus/train stations)
- Connections between events, forming trips
- Trips with the same stop sequence grouped into lines



Public Transit Network

Elements:

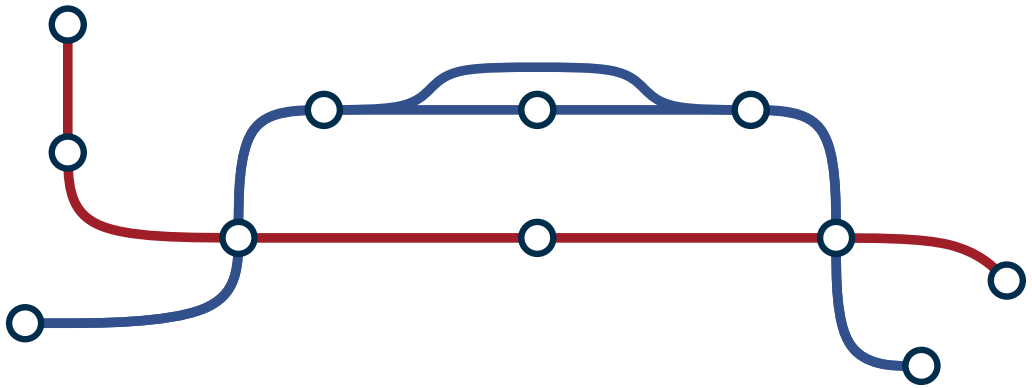
- Stops (bus/train stations)
- Connections between events, forming trips
- Trips with the same stop sequence grouped into lines



Public Transit Network

Elements:

- Stops (bus/train stations)
- Connections between events, forming trips
- Trips with the same stop sequence grouped into lines

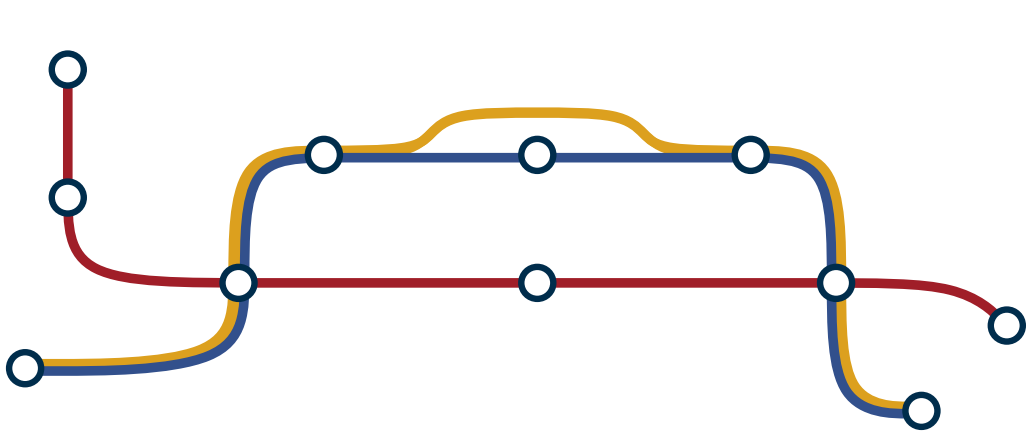


Line 1
not a line!

Public Transit Network

Elements:

- Stops (bus/train stations)
- Connections between events, forming trips
- Trips with the same stop sequence grouped into lines

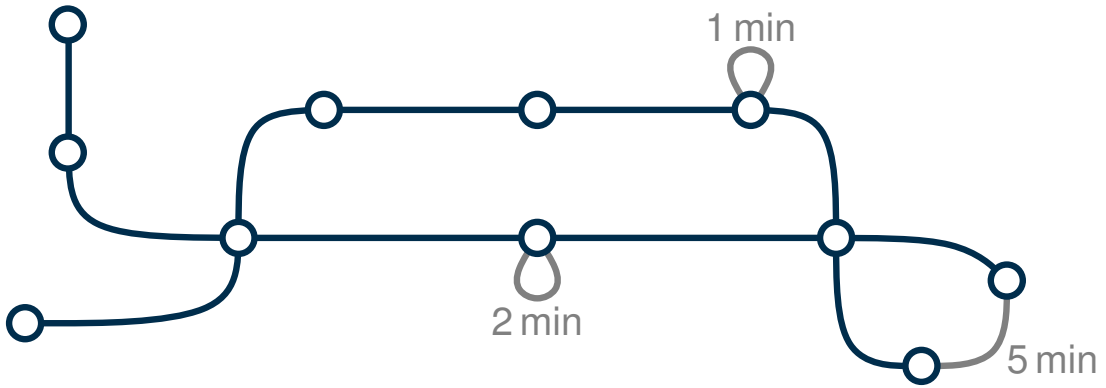


Line 1
Line 2
Line 3

Public Transit Network

Elements:

- Stops (bus/train stations)
- Connections between events, forming trips
- Trips with the same stop sequence grouped into lines
- Footpaths: Ignored here for simplicity



Journey Planning Problem

Input:

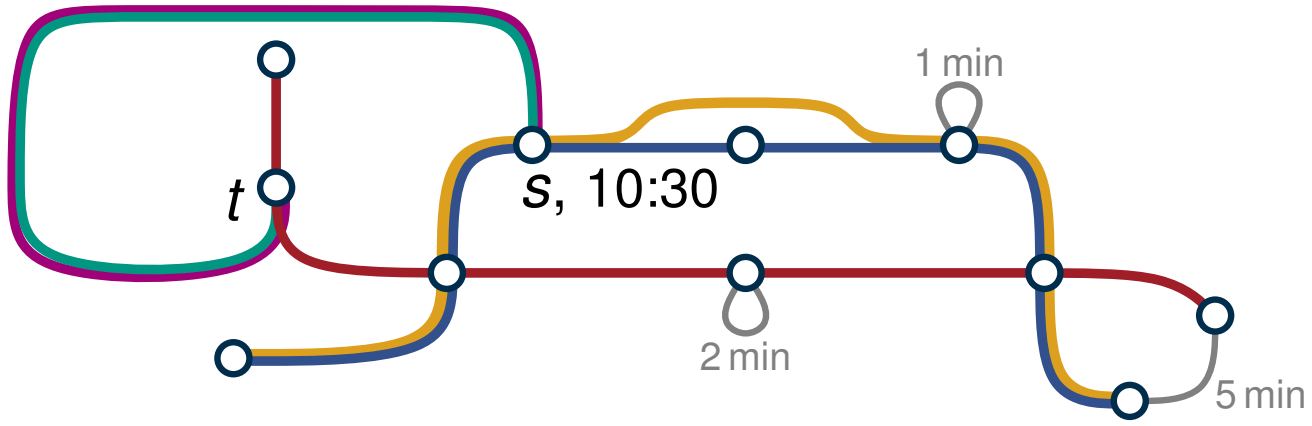
- Source stop s
- Target stop t
- Earliest possible departure time τ_{dep}

Criteria to minimize:

- Arrival time
- Number of used trips
- (maybe more)

Output:

- Pareto front
- One representative journey for each entry in the Pareto front



$$\mathcal{L} = \{ \quad \}$$

Journey Planning Problem

Input:

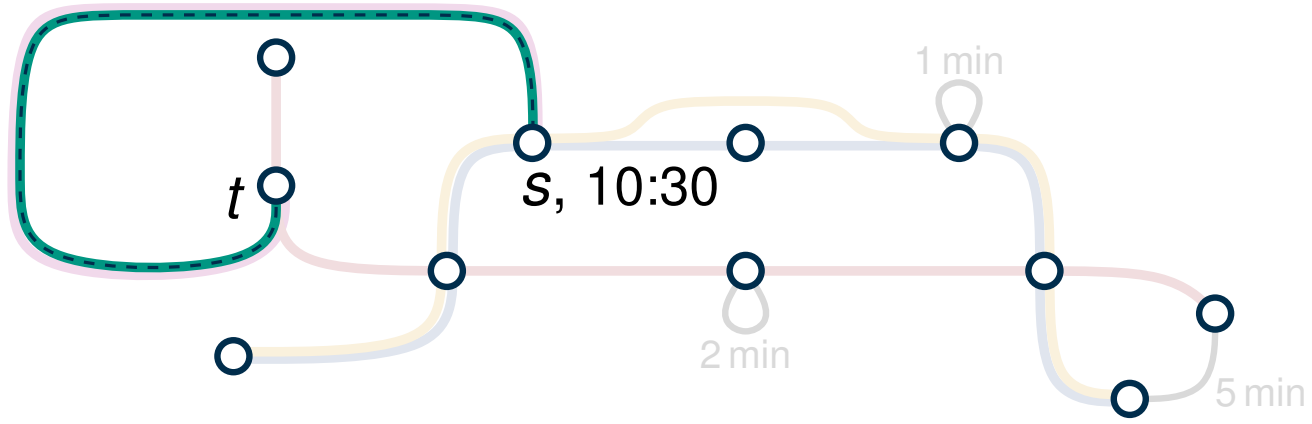
- Source stop s
- Target stop t
- Earliest possible departure time τ_{dep}

Criteria to minimize:

- Arrival time
- Number of used trips
- (maybe more)

Output:

- Pareto front
- One representative journey for each entry in the Pareto front



$$\mathcal{L} = \{(12{:}00, 1),$$

$$\}$$

Journey Planning Problem

Input:

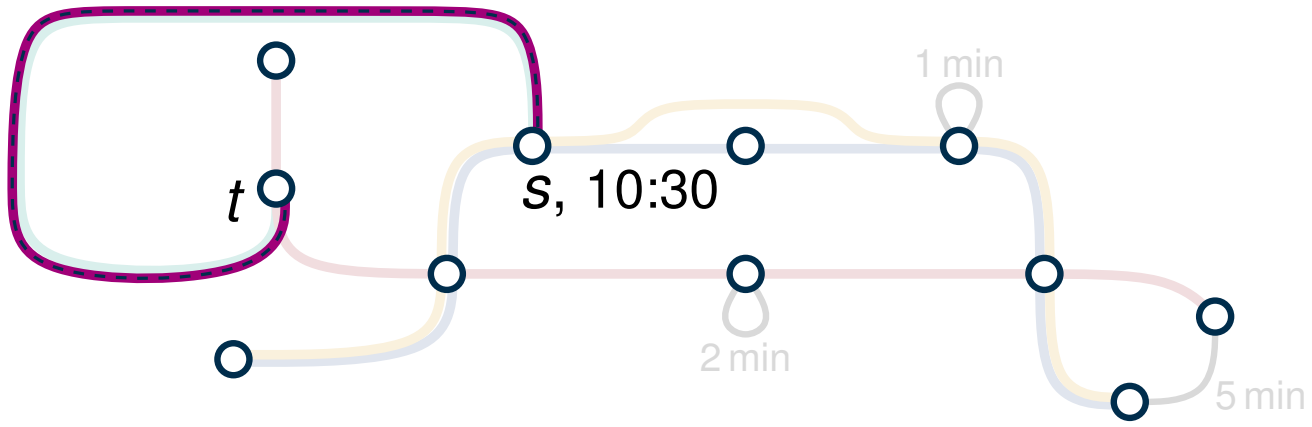
- Source stop s
- Target stop t
- Earliest possible departure time τ_{dep}

Criteria to minimize:

- Arrival time
- Number of used trips
- (maybe more)

Output:

- Pareto front
- One representative journey for each entry in the Pareto front



$$\mathcal{L} = \{(12{:}00, 1), (13{:}00, 1), \}$$

Journey Planning Problem

Input:

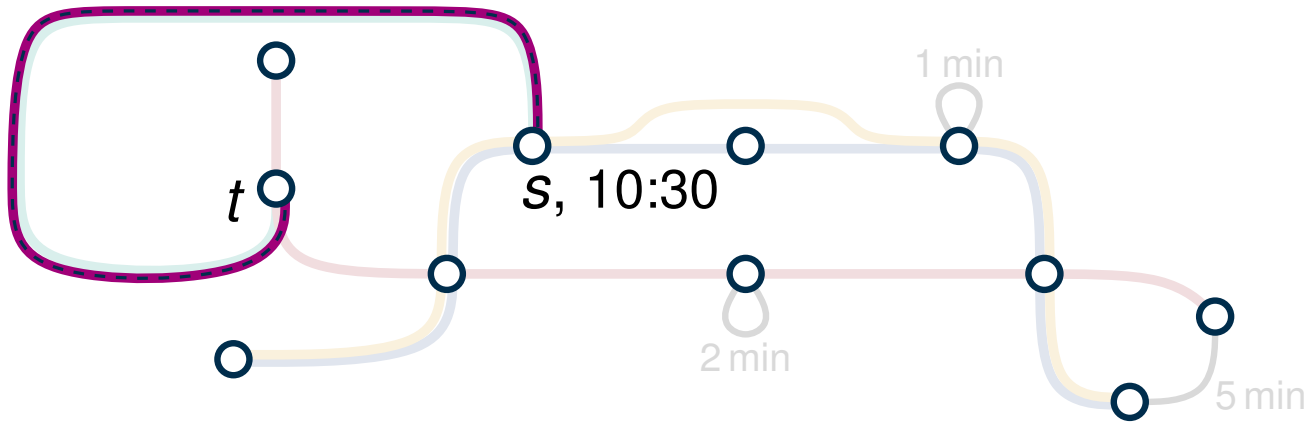
- Source stop s
- Target stop t
- Earliest possible departure time τ_{dep}

Criteria to minimize:

- Arrival time
- Number of used trips
- (maybe more)

Output:

- Pareto front
- One representative journey for each entry in the Pareto front



$$\mathcal{L} = \{(12{:}00, 1), \cancel{(13{:}00, 1),} \}$$

Journey Planning Problem

Input:

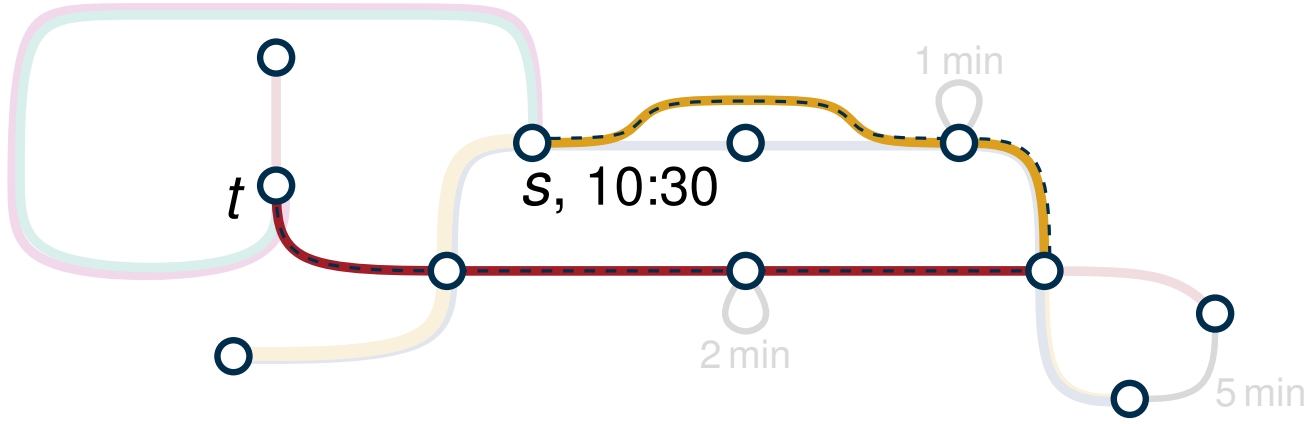
- Source stop s
- Target stop t
- Earliest possible departure time τ_{dep}

Criteria to minimize:

- Arrival time
- Number of used trips
- (maybe more)

Output:

- Pareto front
- One representative journey for each entry in the Pareto front



$$\mathcal{L} = \{(12{:}00, 1), \\ \cancel{(13{:}00, 1)}, \\ (11{:}30, 2), \\ \}$$

Journey Planning Problem

Input:

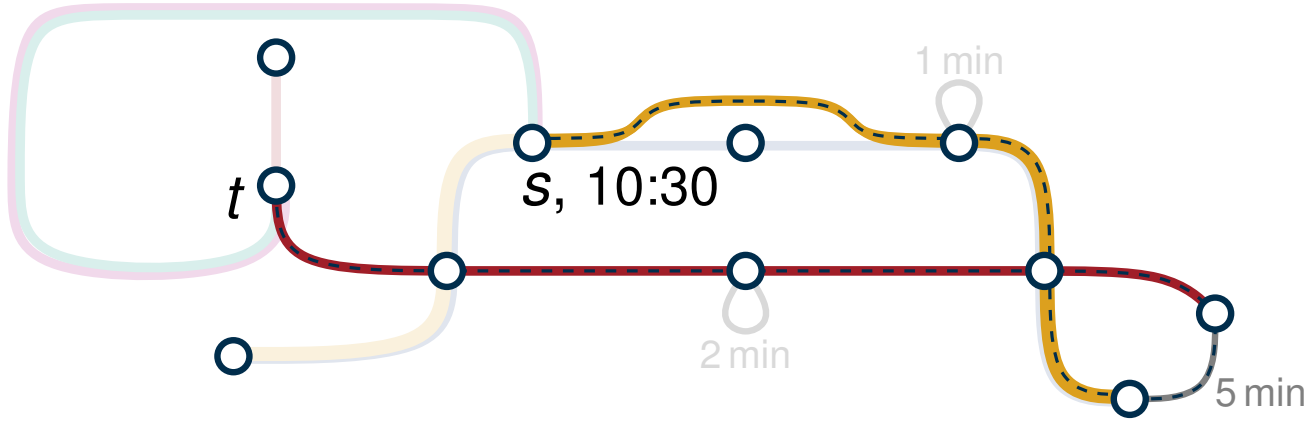
- Source stop s
- Target stop t
- Earliest possible departure time τ_{dep}

Criteria to minimize:

- Arrival time
- Number of used trips
- (maybe more)

Output:

- Pareto front
- One representative journey for each entry in the Pareto front



$$\mathcal{L} = \{(12{:}00, 1), \cancel{(13{:}00, 1),} (11{:}30, 2), (11{:}30, 2)\}$$

Journey Planning Problem

Input:

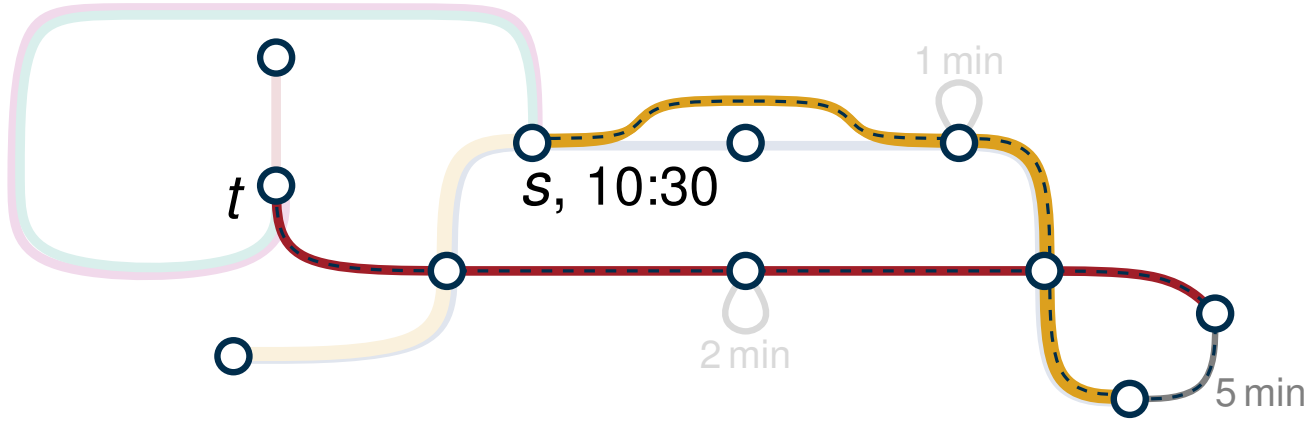
- Source stop s
- Target stop t
- Earliest possible departure time τ_{dep}

Criteria to minimize:

- Arrival time
- Number of used trips
- (maybe more)

Output:

- Pareto front
- One representative journey for each entry in the Pareto front



$$\mathcal{L} = \{(12{:}00, 1), \cancel{(13{:}00, 1)}, (11{:}30, 2), \cancel{(11{:}30, 2)}\}$$

Time-Expanded Graph

Other terms: Static expansion, event graph

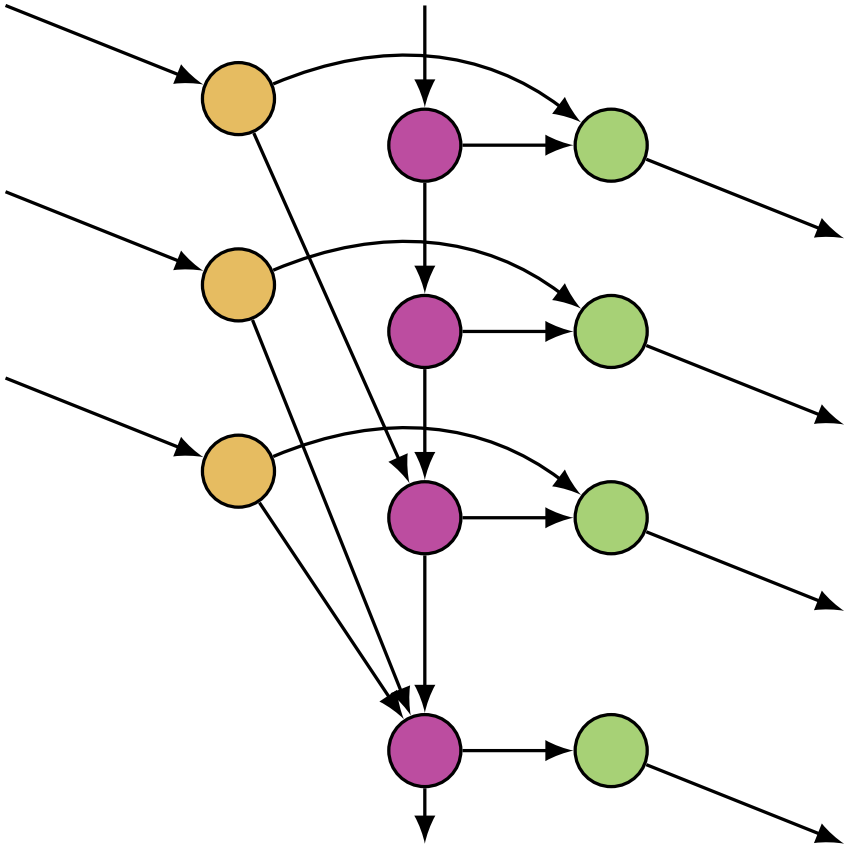
Nodes:

- Three per event: arrival, transfer, departure
- Labeled with event time

Edges:

- Transfer $\rightarrow$ departure edges: Cost 1
- Others: Cost 0

Size: $O(m)$ (m = number of connections)



Earliest-Arrival Path (EAP)

Early public transit literature: [Pyrga, Schulz, Wagner, Zaroliagis 2008]

- Dijkstra's algorithm on the TEG
- Temporal structure not exploited
- Runtime: $O(m \log m)$

Earliest-Arrival Path (EAP)

Early public transit literature: [Pyrga, Schulz, Wagner, Zaroliagis 2008]

- Dijkstra's algorithm on the TEG
- Temporal structure not exploited
- Runtime: $O(m \log m)$

Temporal graph literature: [Wu, Cheng, Huang, Ke, Lu, Xu 2014]

- Exploit that the TEG is a DAG
- Sweep over temporal edges in ascending order of time
- Runtime: $O(m)$

Earliest-Arrival Path (EAP)

Early public transit literature: [Pyrga, Schulz, Wagner, Zaroliagis 2008]

- Dijkstra's algorithm on the TEG
- Temporal structure not exploited
- Runtime: $O(m \log m)$

Temporal graph literature: [Wu, Cheng, Huang, Ke, Lu, Xu 2014]

- Exploit that the TEG is a DAG
- Sweep over temporal edges in ascending order of time
- Runtime: $O(m)$

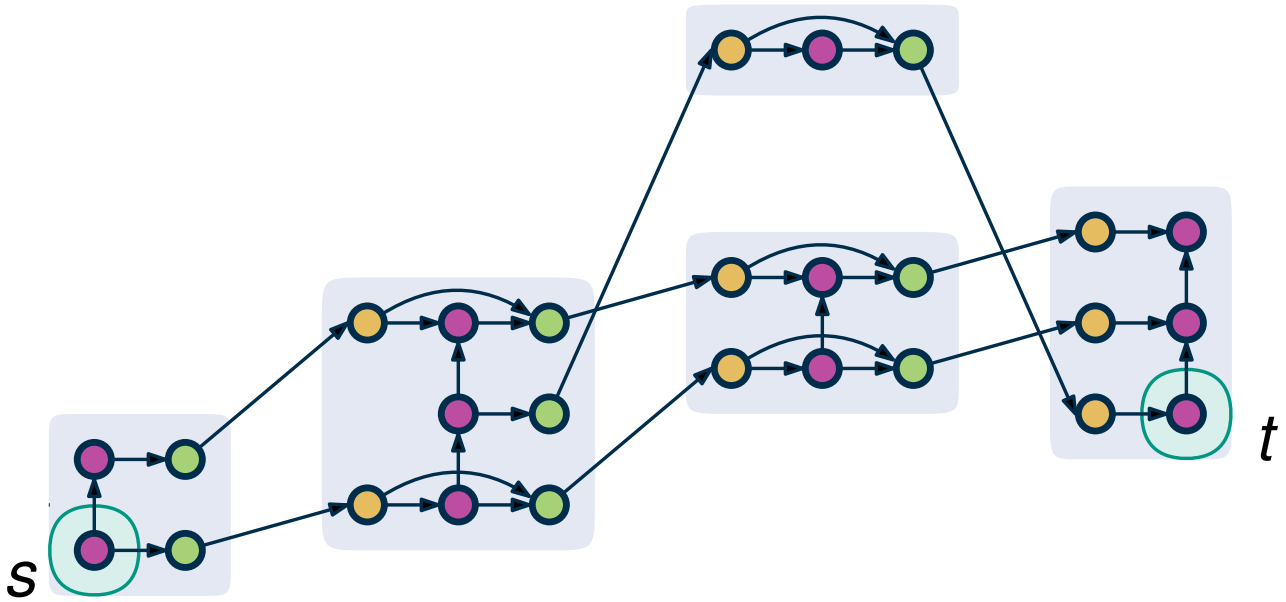
Connection Scan Algorithm: [Dibbelt, Pajor, Strasser, Wagner 2013]

- Same algorithm as [Wu, Cheng, Huang, Ke, Lu, Xu 2014]
- But published earlier!
- Both results were arrived at independently

BFS on the TEG

Idea:

- Start from earliest transfer node at s with time $\geq \tau_{\text{dep}}$
- Find earliest reachable transfer node at t
- Runtime: $O(m)$



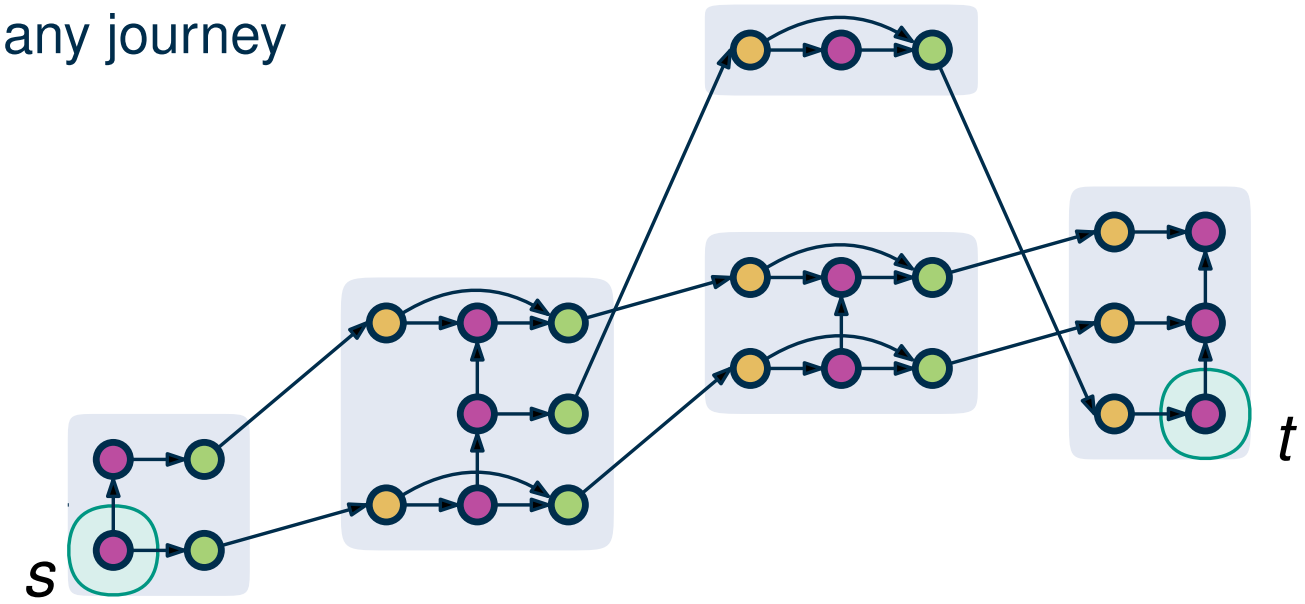
Minimizing Number of Trips

As a temporal path problem:

- Min-cost temporal path with 0-1 costs
- Dijkstra's algorithm with bucket-based priority queue
- Runtime: $O(m)$

Alternate view:

- $k =$ Upper bound on number of trips in any journey
- Create k copies (layers) of the TEG
- Cost-1 edges connect to next layer
- Bucket-Dijkstra $\hat{=}$ layer-by-layer BFS
- Runtime: $O(km)$ (k is small in practice)



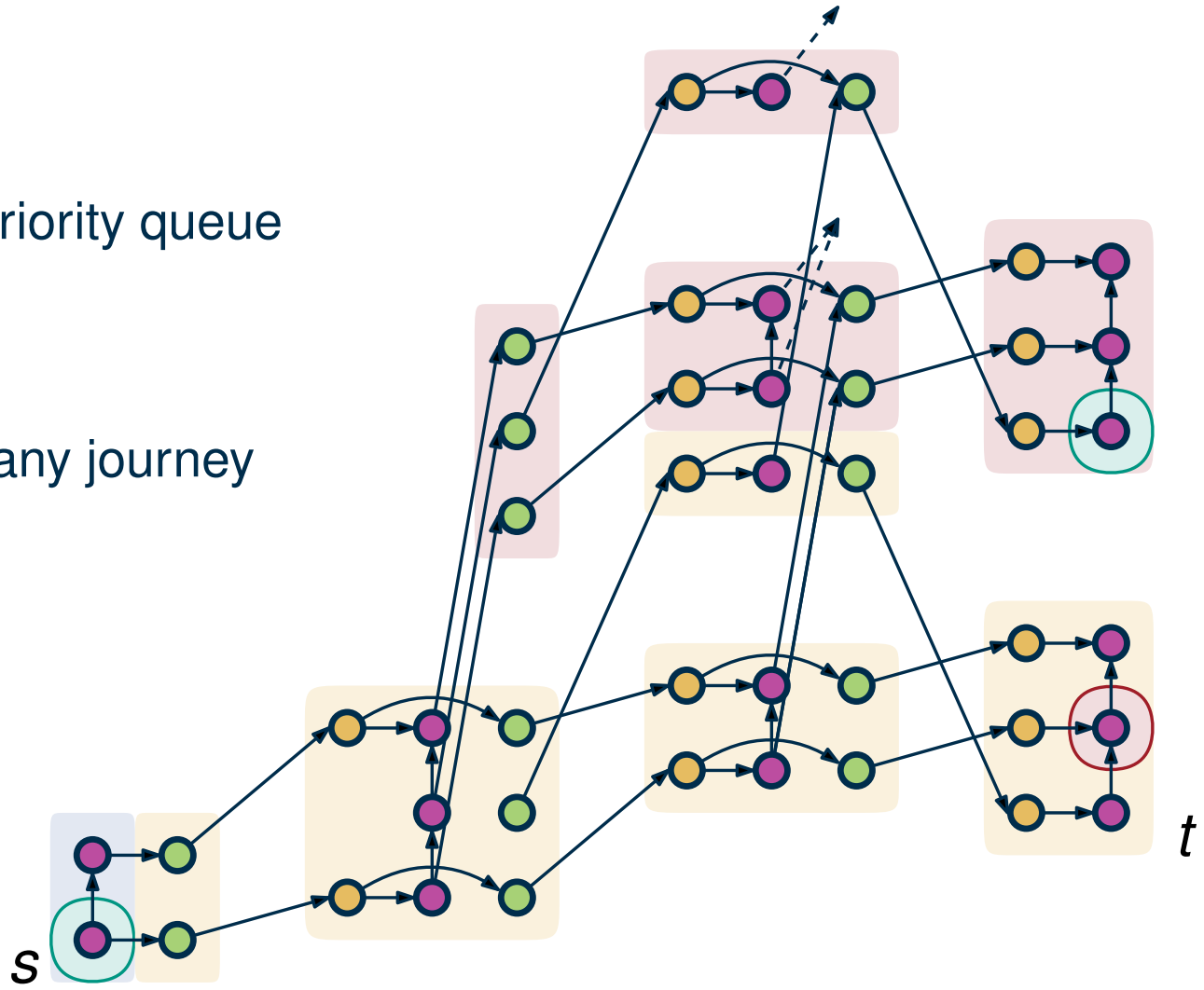
Minimizing Number of Trips

As a temporal path problem:

- Min-cost temporal path with 0-1 costs
- Dijkstra's algorithm with bucket-based priority queue
- Runtime: $O(m)$

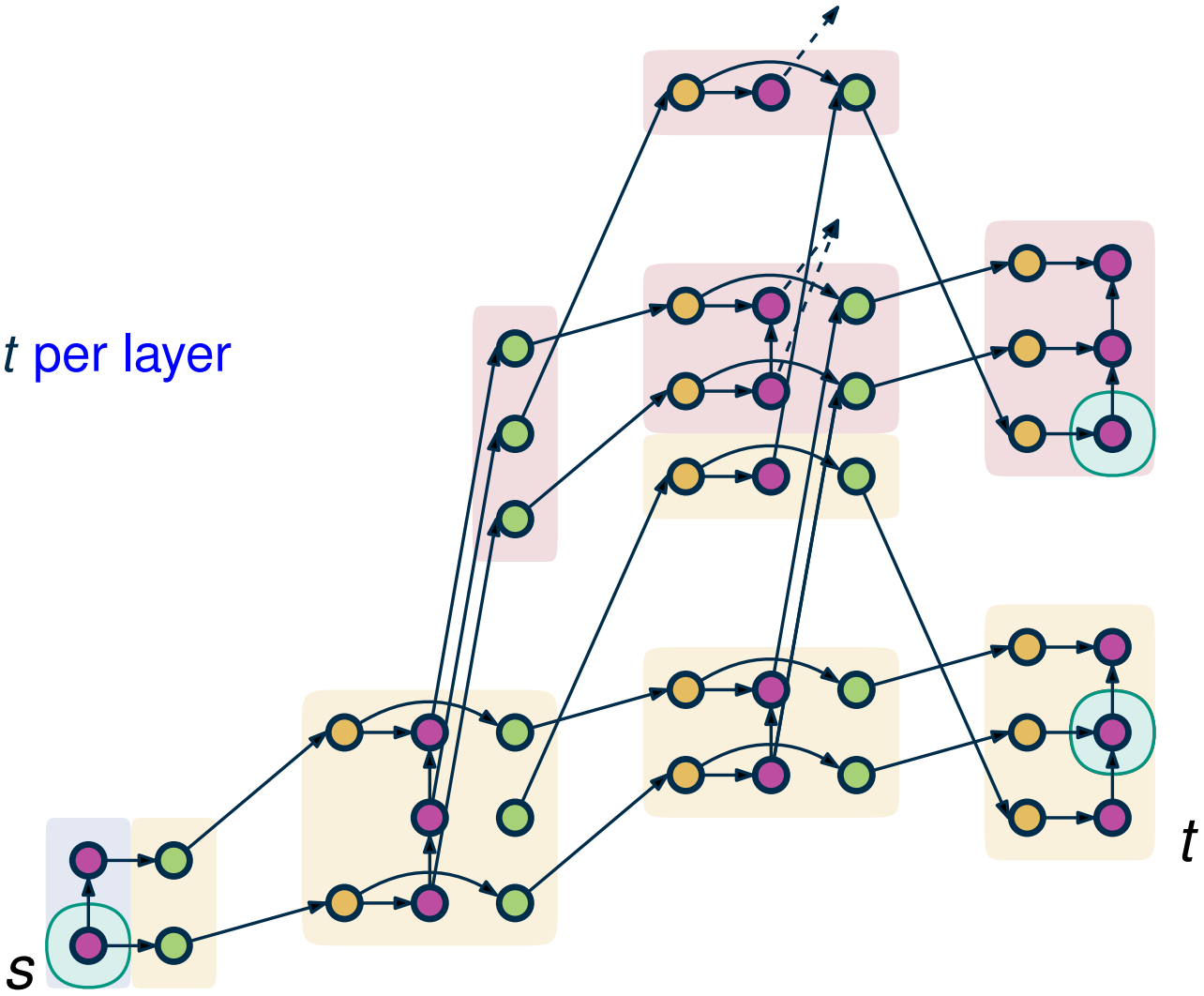
Alternate view:

- k = Upper bound on number of trips in any journey
- Create k copies (layers) of the TEG
- Cost-1 edges connect to next layer
- Bucket-Dijkstra $\hat{=}$ layer-by-layer BFS
- Runtime: $O(km)$ (k is small in practice)



Pareto Optimization

- Find earliest reachable transfer node at t per layer
- Layer-BFS still works!



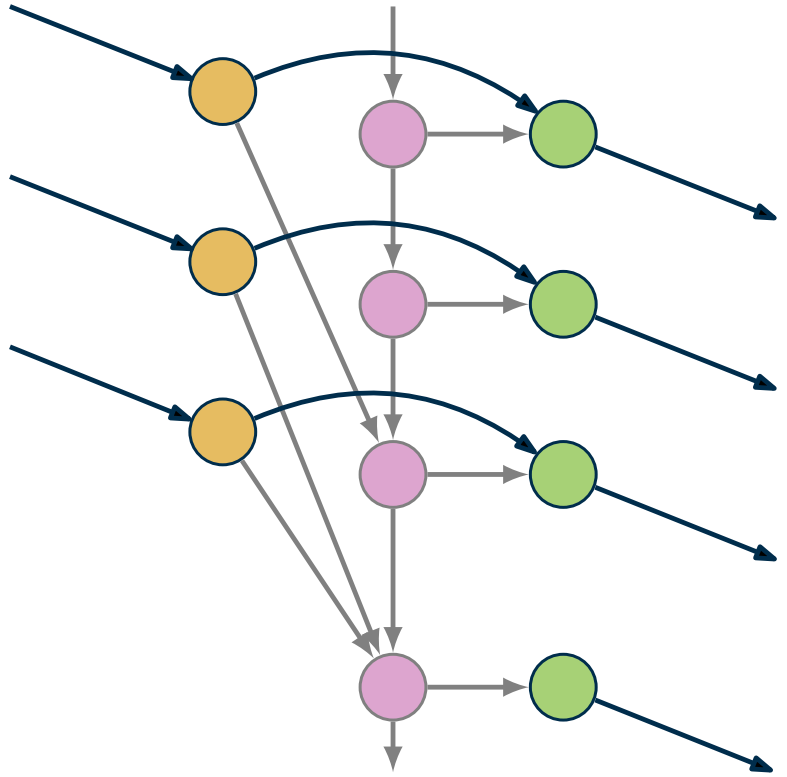
Trip-Based Routing (TB) [Witt 2015]

Transfer shortcuts:

- Connect arrival nodes to departure nodes at the same stop
- Only if actually needed in optimal paths
- Can be precomputed quickly

Query:

- Transfer nodes only needed at the beginning
- After that: Layer-BFS where each trip is treated as a node
- Prune irrelevant events with reached index
- Only scan previously unreached segments of trips



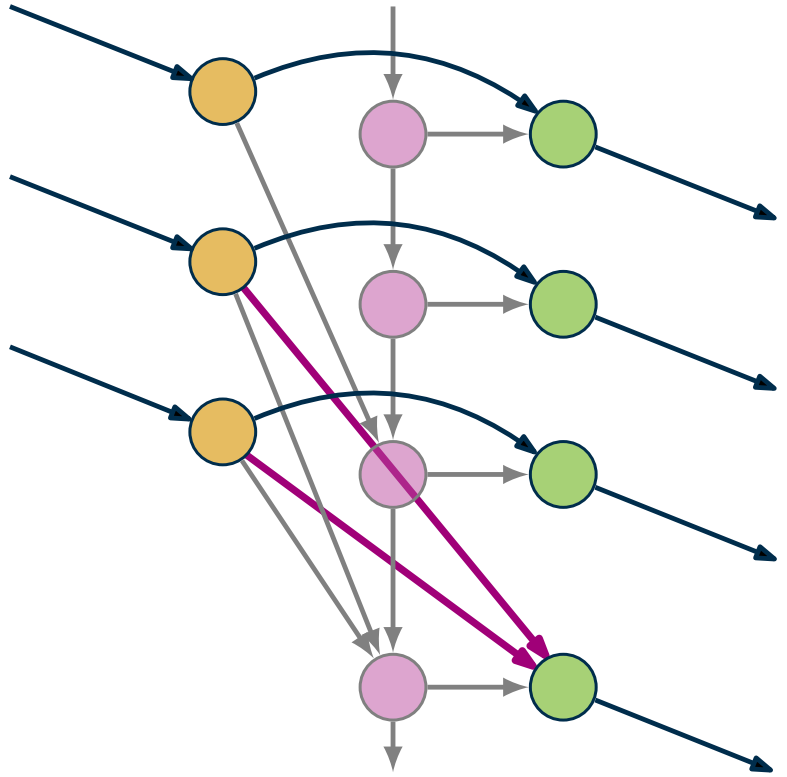
Trip-Based Routing (TB) [Witt 2015]

Transfer shortcuts:

- Connect arrival nodes to departure nodes at the same stop
- Only if actually needed in optimal paths
- Can be precomputed quickly

Query:

- Transfer nodes only needed at the beginning
- After that: Layer-BFS where each trip is treated as a node
- Prune irrelevant events with reached index
- Only scan previously unreached segments of trips



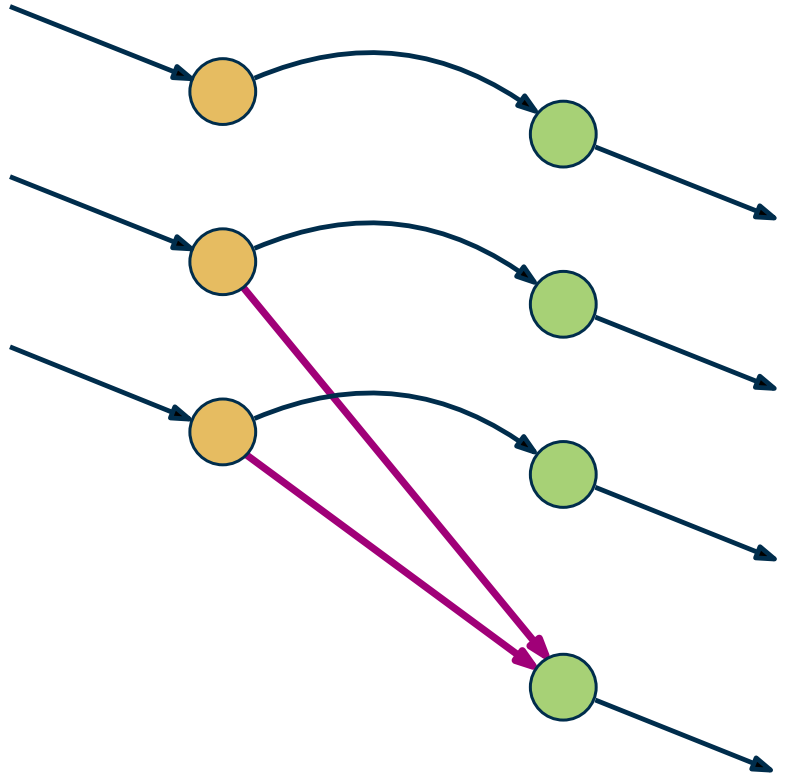
Trip-Based Routing (TB) [Witt 2015]

Transfer shortcuts:

- Connect arrival nodes to departure nodes at the same stop
- Only if actually needed in optimal paths
- Can be precomputed quickly

Query:

- Transfer nodes only needed at the beginning
- After that: Layer-BFS where each trip is treated as a node
- Prune irrelevant events with reached index
- Only scan previously unreached segments of trips



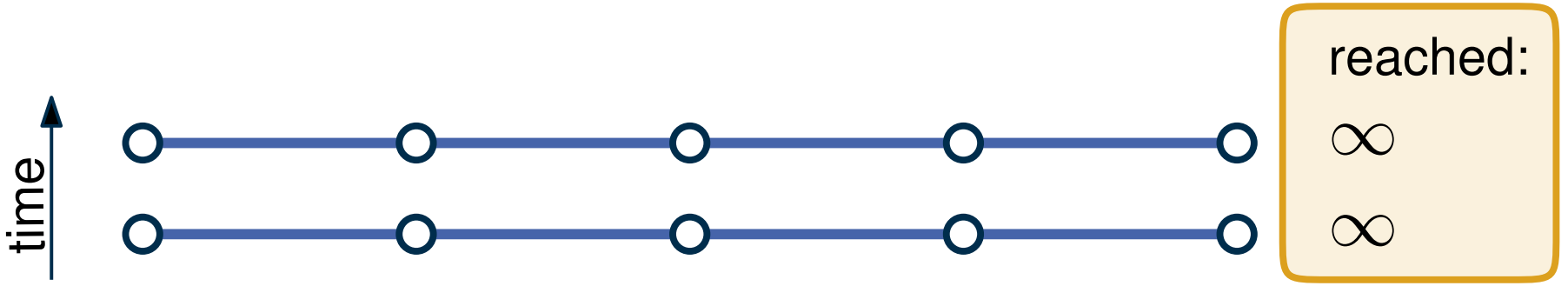
Trip-Based Routing (TB) [Witt 2015]

Transfer shortcuts:

- Connect arrival nodes to departure nodes at the same stop
- Only if actually needed in optimal paths
- Can be precomputed quickly

Query:

- Transfer nodes only needed at the beginning
- After that: Layer-BFS where each trip is treated as a node
- Prune irrelevant events with reached index
- Only scan previously unreached segments of trips



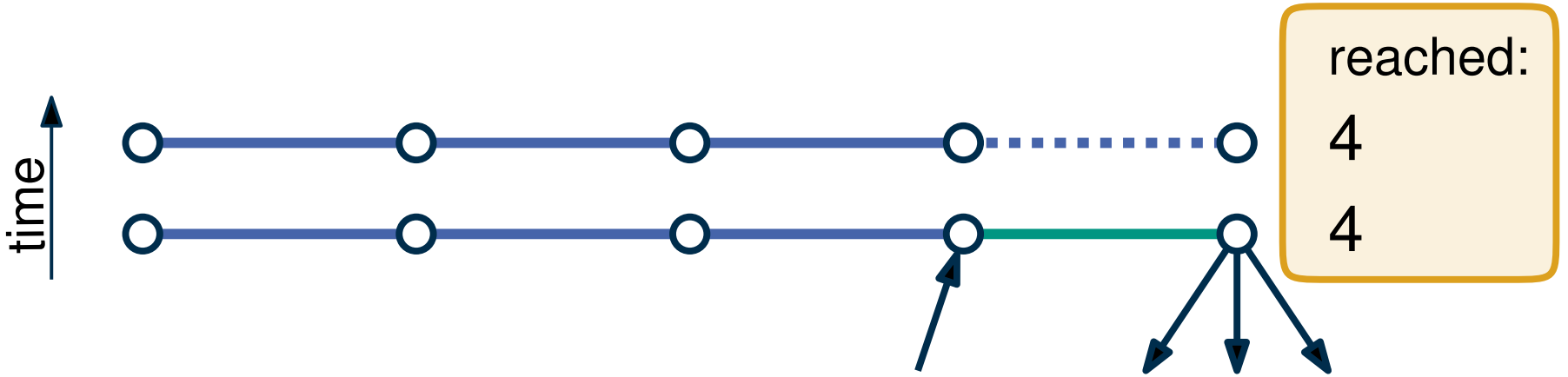
Trip-Based Routing (TB) [Witt 2015]

Transfer shortcuts:

- Connect arrival nodes to departure nodes at the same stop
- Only if actually needed in optimal paths
- Can be precomputed quickly

Query:

- Transfer nodes only needed at the beginning
- After that: Layer-BFS where each trip is treated as a node
- Prune irrelevant events with reached index
- Only scan previously unreached segments of trips



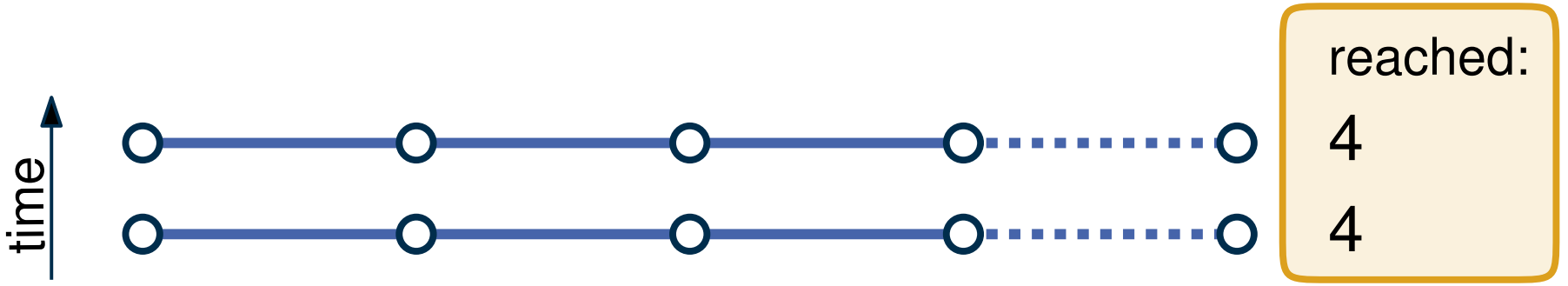
Trip-Based Routing (TB) [Witt 2015]

Transfer shortcuts:

- Connect arrival nodes to departure nodes at the same stop
- Only if actually needed in optimal paths
- Can be precomputed quickly

Query:

- Transfer nodes only needed at the beginning
- After that: Layer-BFS where each trip is treated as a node
- Prune irrelevant events with reached index
- Only scan previously unreached segments of trips



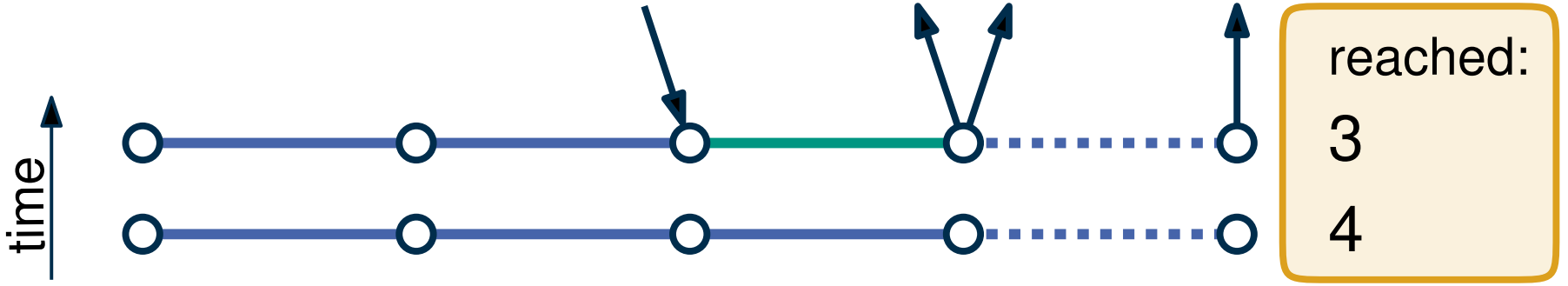
Trip-Based Routing (TB) [Witt 2015]

Transfer shortcuts:

- Connect arrival nodes to departure nodes at the same stop
- Only if actually needed in optimal paths
- Can be precomputed quickly

Query:

- Transfer nodes only needed at the beginning
- After that: Layer-BFS where each trip is treated as a node
- Prune irrelevant events with reached index
- Only scan previously unreached segments of trips



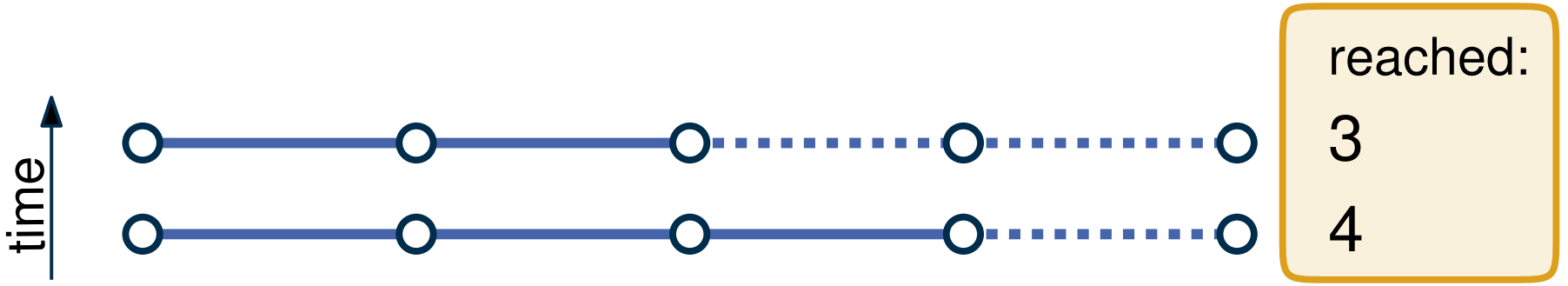
Trip-Based Routing (TB) [Witt 2015]

Transfer shortcuts:

- Connect arrival nodes to departure nodes at the same stop
- Only if actually needed in optimal paths
- Can be precomputed quickly

Query:

- Transfer nodes only needed at the beginning
- After that: Layer-BFS where each trip is treated as a node
- Prune irrelevant events with reached index
- Only scan previously unreached segments of trips



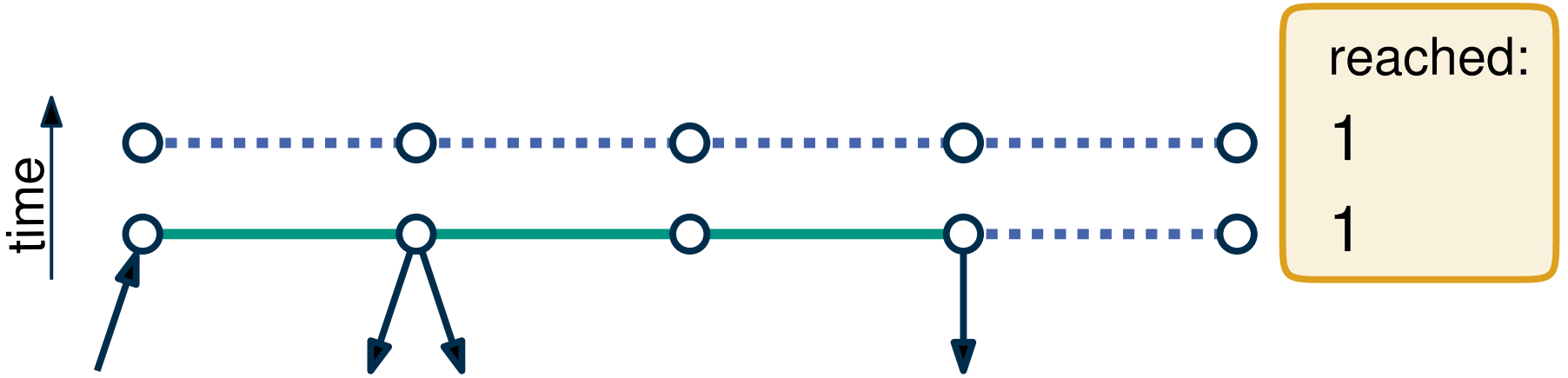
Trip-Based Routing (TB) [Witt 2015]

Transfer shortcuts:

- Connect arrival nodes to departure nodes at the same stop
- Only if actually needed in optimal paths
- Can be precomputed quickly

Query:

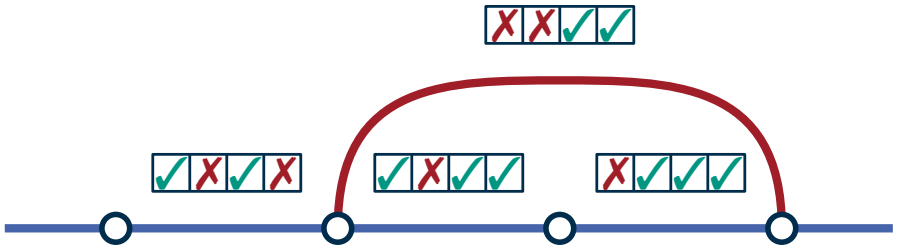
- Transfer nodes only needed at the beginning
- After that: Layer-BFS where each trip is treated as a node
- Prune irrelevant events with reached index
- Only scan previously unreached segments of trips



Indexing

Idea:

- Precompute auxiliary information
- Use it for pruning during query
- Common ingredient: Edge annotations



Types of annotations:

- Goal-directed:

  “Do I need this edge if the target is in region *x*?”
- Hierarchical:

  “Do I need this edge if it is far away from source and target?”

Applied to:

- Road networks: Highly successful [Geisberger, Sanders, Schultes, Vetter 2008]
- Public transit: Not so much [Bast 2009]

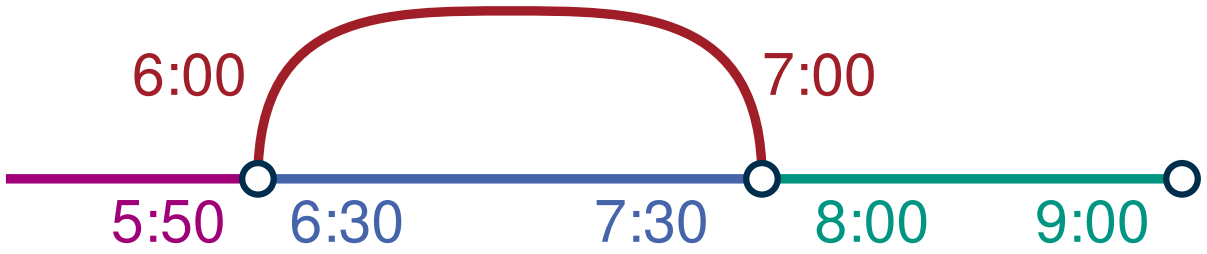
What's the problem?

One reason:

- Indexing usually relies on subpath property:

  Every subpath of a shortest path is itself a shortest path

- Pruning information can be "glued together" from subpaths
- Subpath property does not hold in temporal EAP



- Previous attempts either sacrificed optimality or weakened the pruning

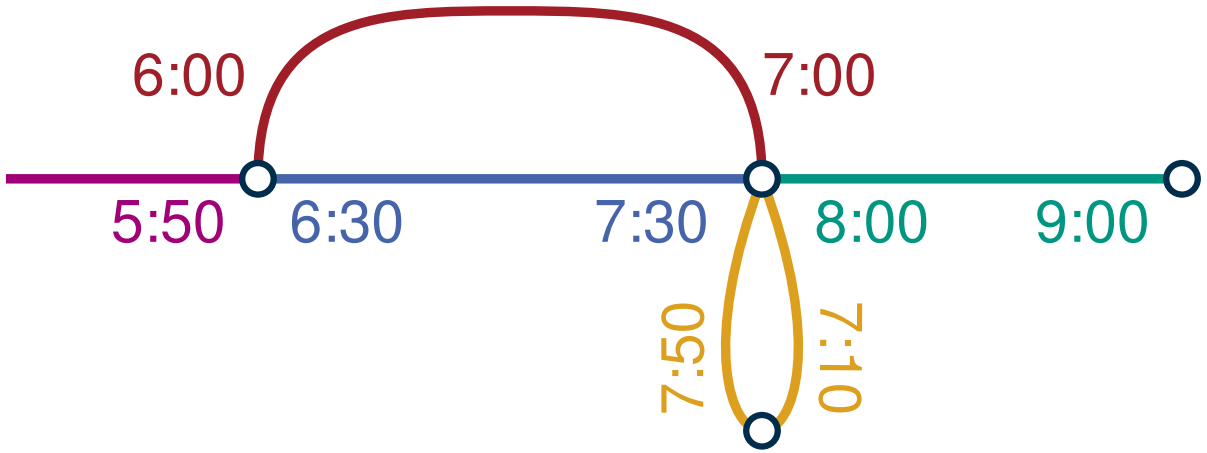
What's the problem?

One reason:

- Indexing usually relies on subpath property:

  Every subpath of a shortest path is itself a shortest path

- Pruning information can be "glued together" from subpaths
- Subpath property does not hold in temporal EAP

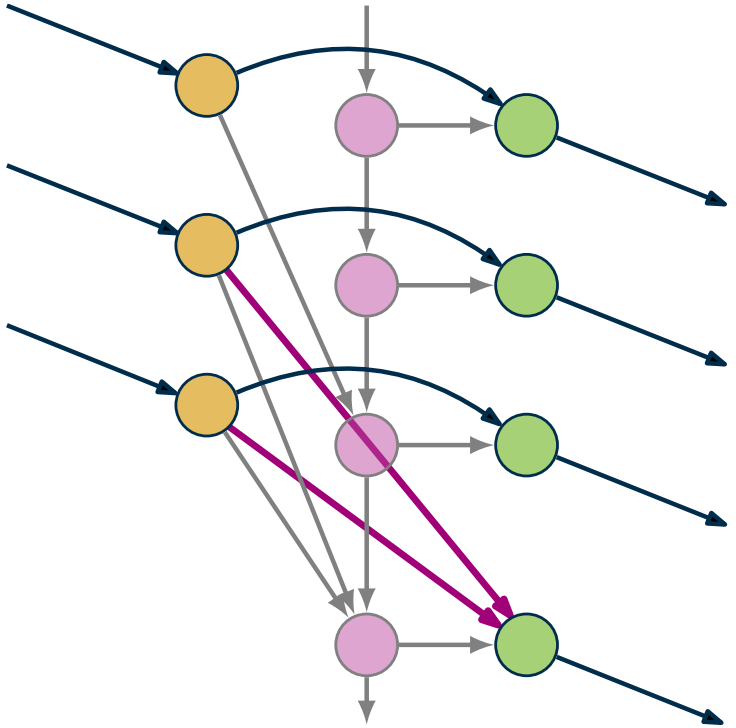


- Previous attempts either sacrificed optimality or weakened the pruning

Indexing for TB

Restoring subpath property:

- For each entry in the Pareto front:
  Choose one canonical representative path
- Such that every (u, v)-subpath is itself canonical
- Modify the search algorithm
  such that it finds this canonical path
- Not obvious that this is even possible



Indexing:

- Annotate only the transfer shortcuts
- They offer a stronger pruning effect than the other edges

Our Algorithms

FLASH-TB: TB + goal direction

[Großmann, Schulz, Steil, S., Witt 2025]

++ Extremely fast queries

\+ Configurable memory consumption

– Quadratic preprocessing effort

T-REX: Hierarchical TB

[Steil, S., Witt 2026]

\+ Very fast queries

\+ Moderate memory consumption

++ Fast preprocessing

++ Supports real-time updates

Performance: Germany

- Fastest: ≈ 0.1 ms with 18.8 GB
- Moderate: < 1 ms with < 0.4 GB
- Preprocessing time: 30 h
  (does not scale to Europe)

Performance: Europe

- Fastest: < 10 ms with 27.8 GB
- Moderate: < 20 ms with 8.5 GB
- Preprocessing time: 2 min

Summary

Public transit journey planning:

- More about temporal reachability than shortest paths
- Turn everything into a cache-friendly dynamic program
- We are only just starting to get the hang of indexing

Summary

Public transit journey planning:

- More about temporal reachability than shortest paths
- Turn everything into a cache-friendly dynamic program
- We are only just starting to get the hang of indexing

Two (largely) independent communities:

- Temporal graphs and journey planning
- They often have the same insights independently

Summary

Public transit journey planning:

- More about temporal reachability than shortest paths
- Turn everything into a cache-friendly dynamic program
- We are only just starting to get the hang of indexing

Two (largely) independent communities:

- Temporal graphs and journey planning
- They often have the same insights independently

My pitch

Let’s work together!