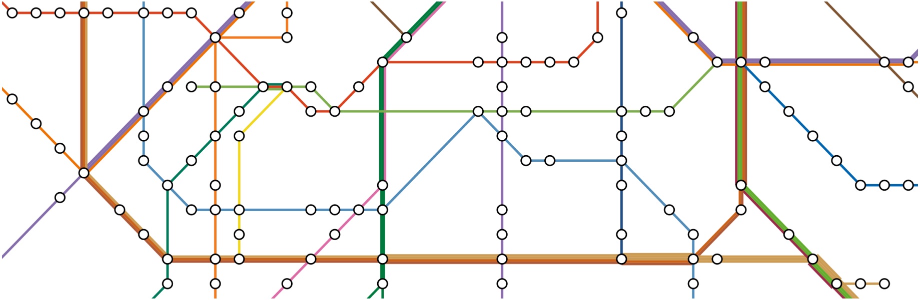
Mathematical Aspects of Public Transportation Networks

Niels Lindner



May 7, 2018

Chapter 2

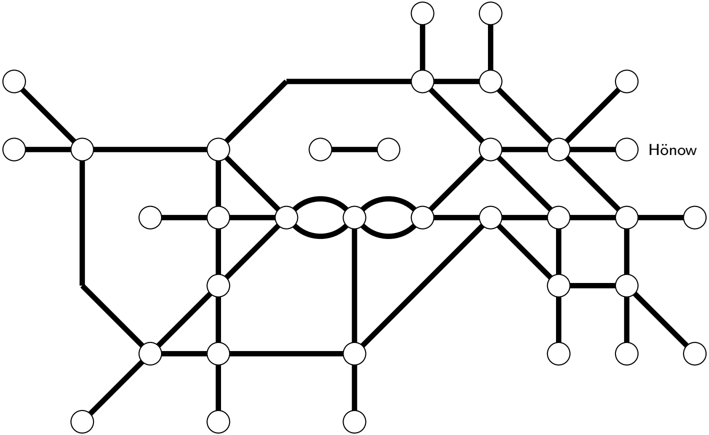
Shortest Routes in Public Transportation Networks

§2.2 Graph Methods

Time-Dependent Dijkstra

Example (shortest path tree)

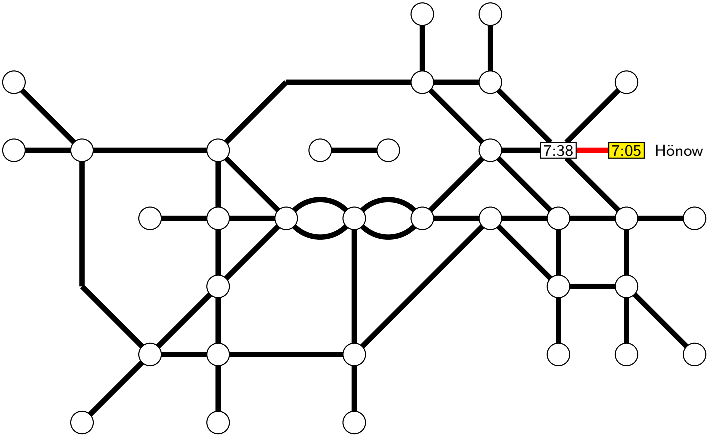
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

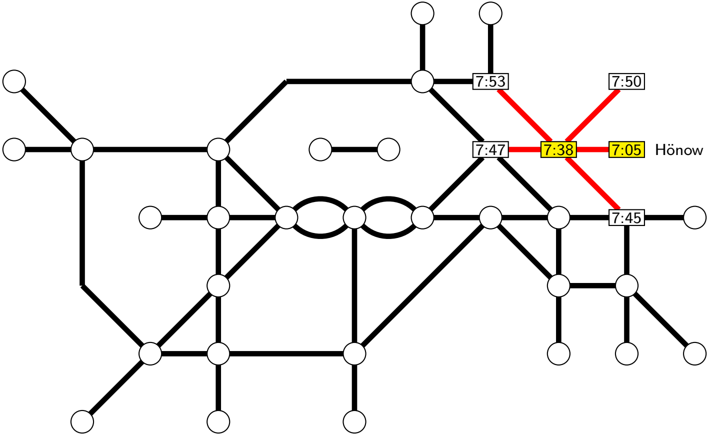
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

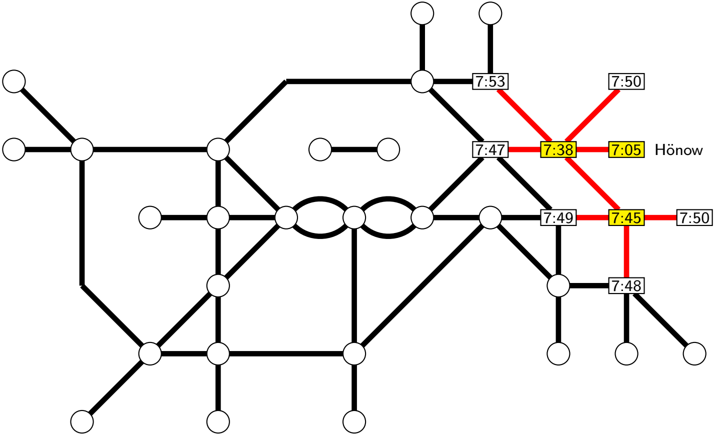
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

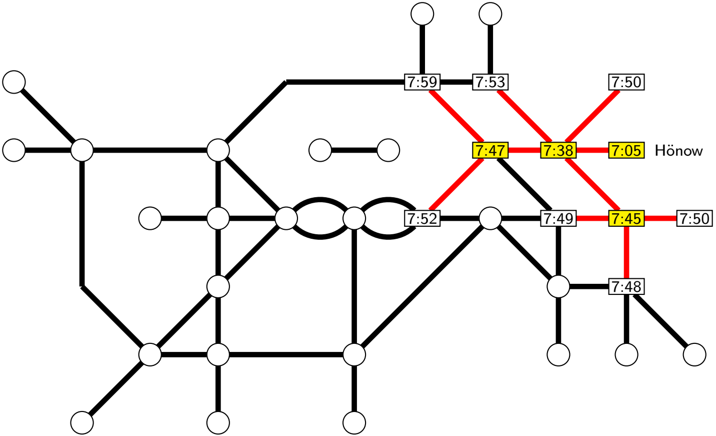
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

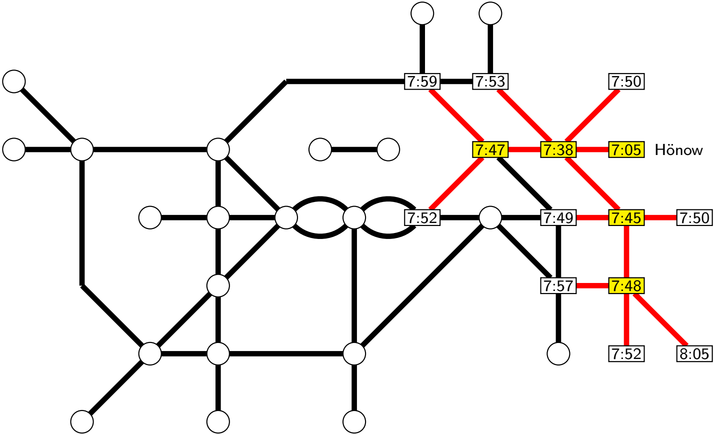
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

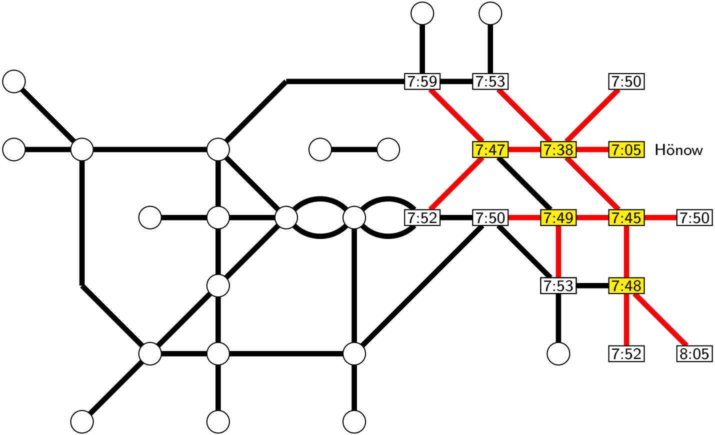
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

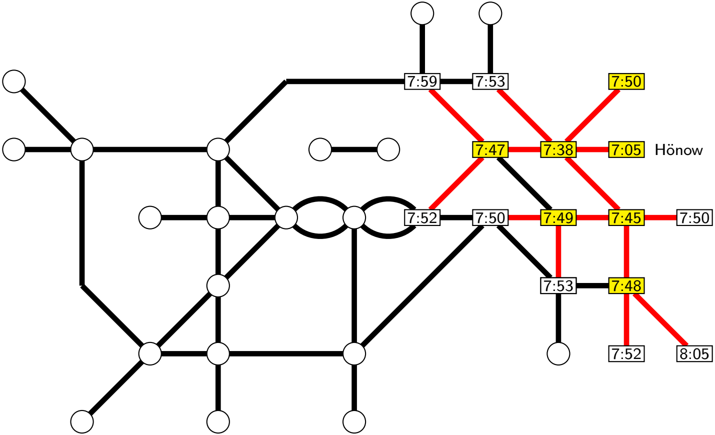
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

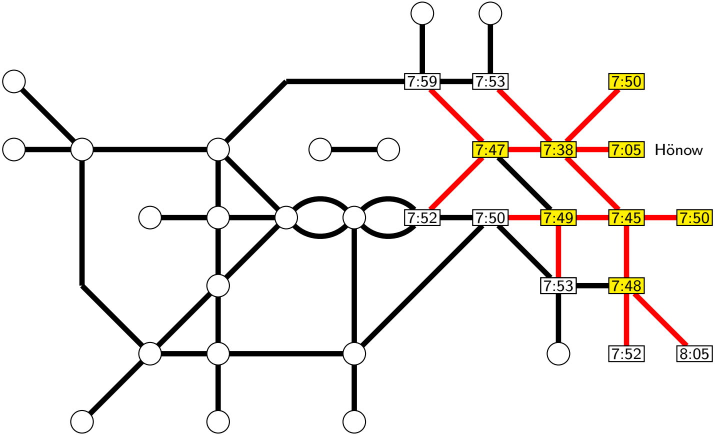
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

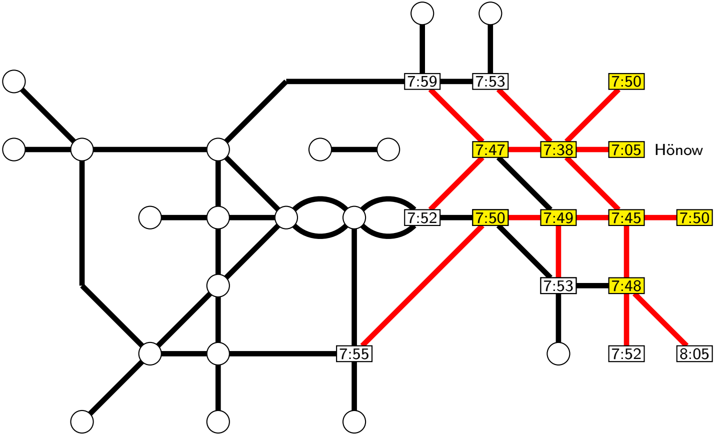
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

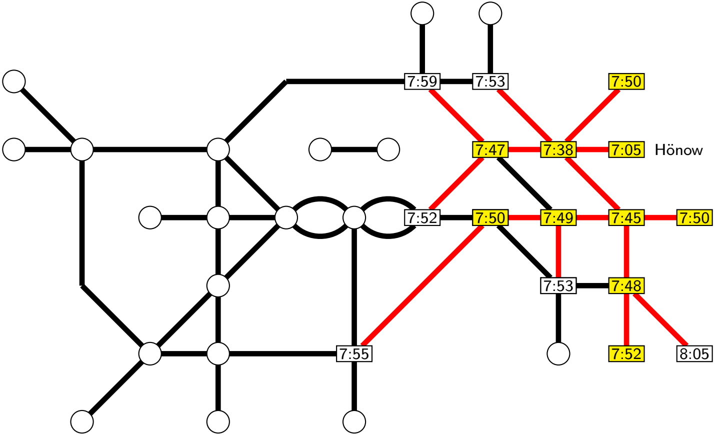
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

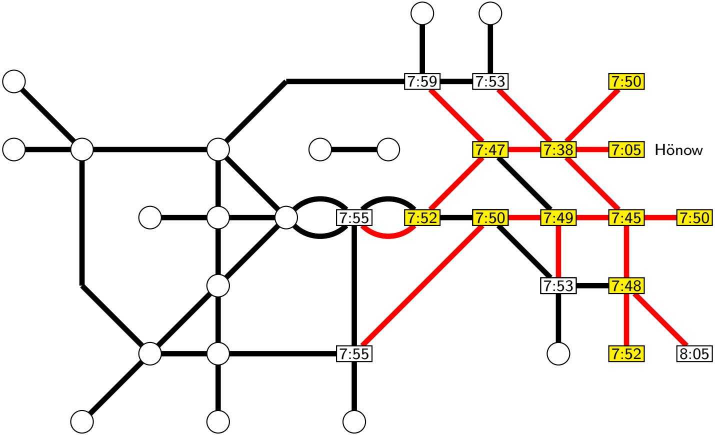
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

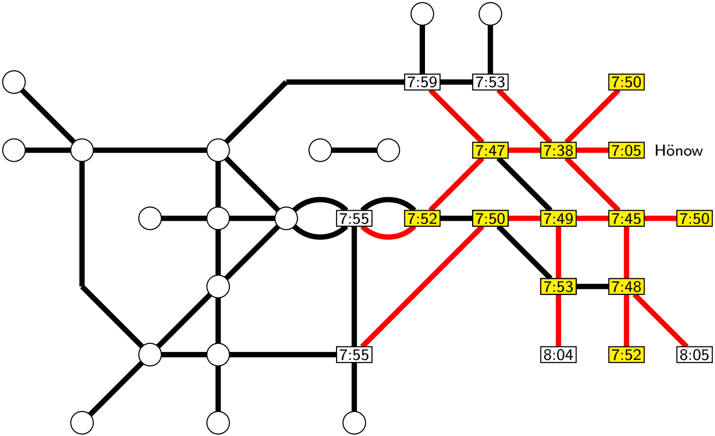
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

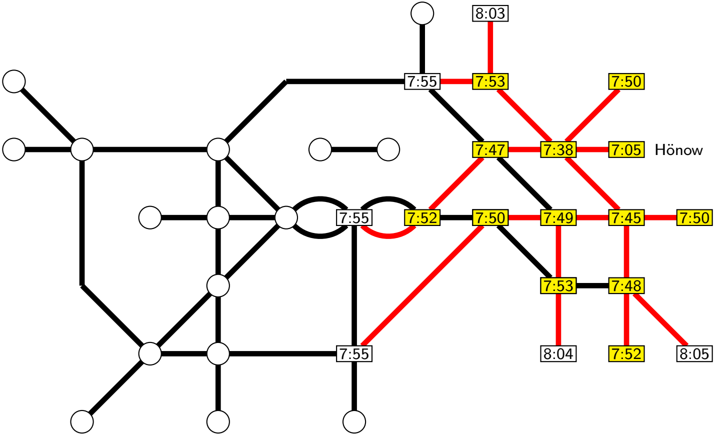
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

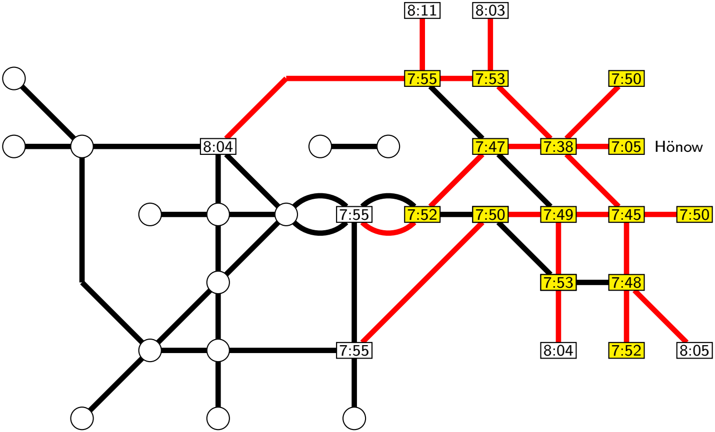
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

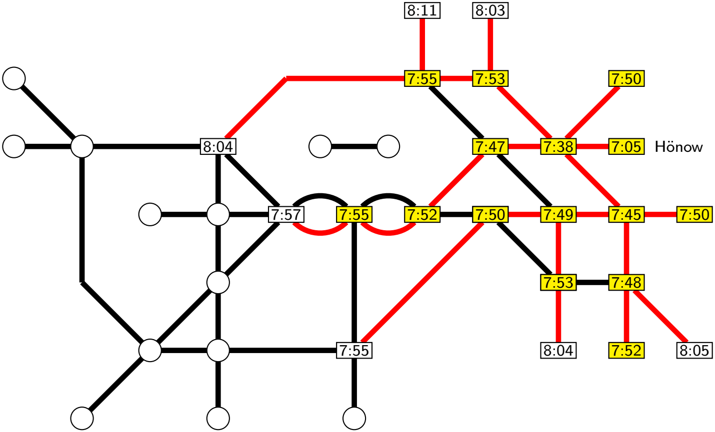
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

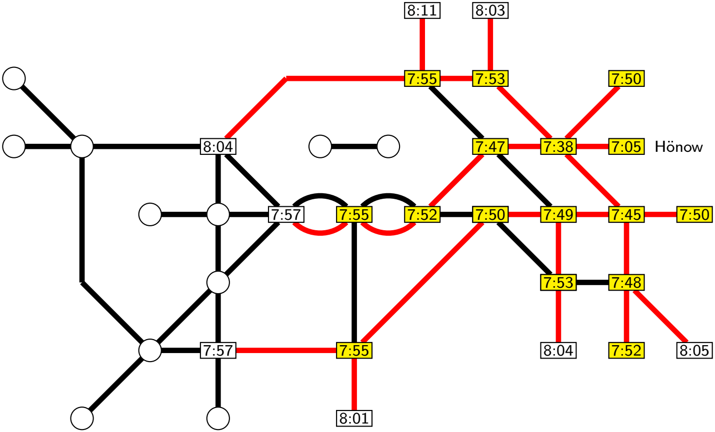
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

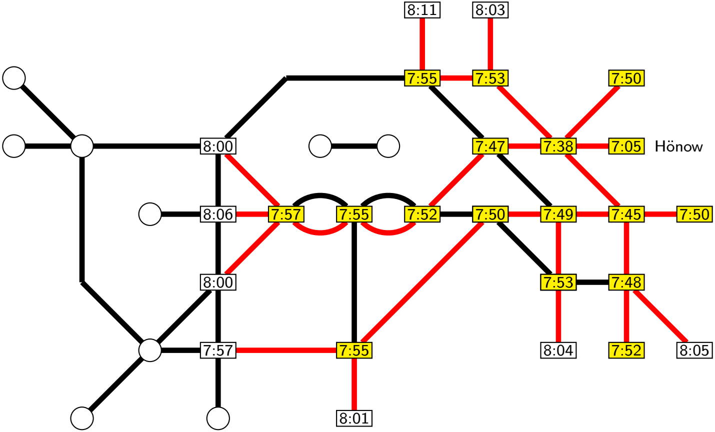
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

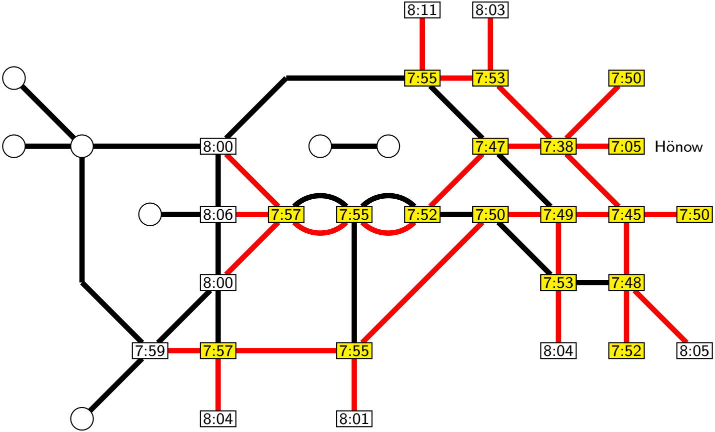
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

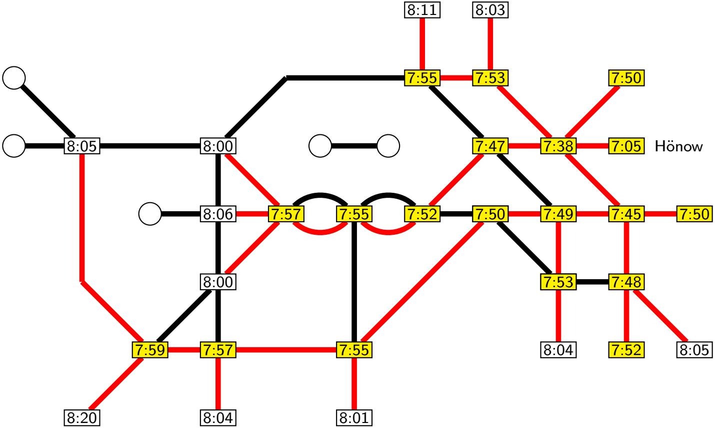
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

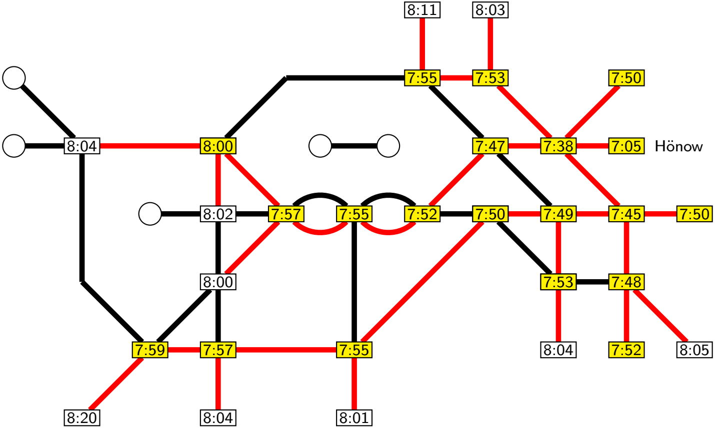
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

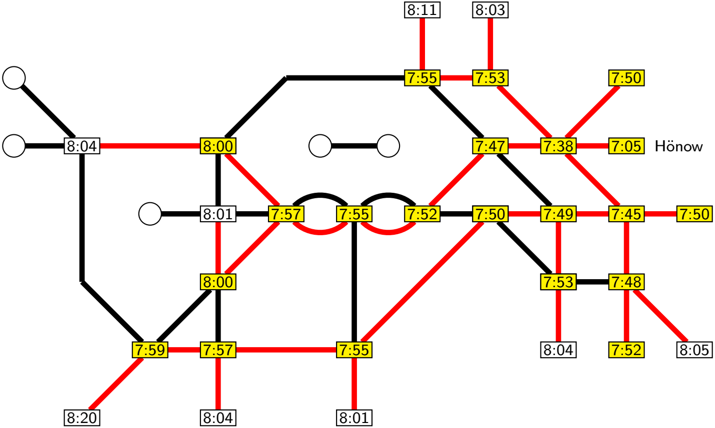
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

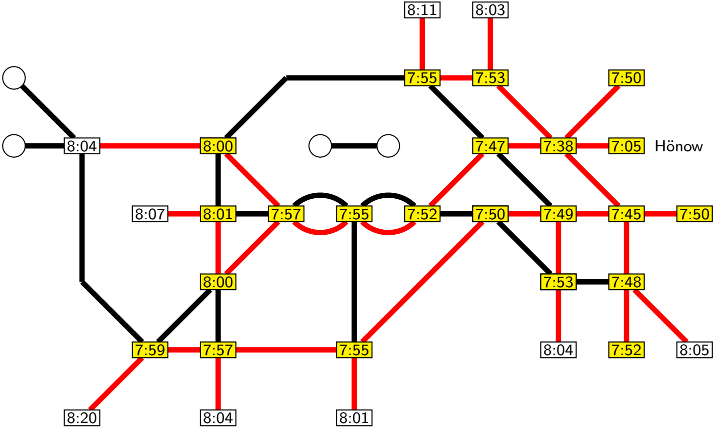
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

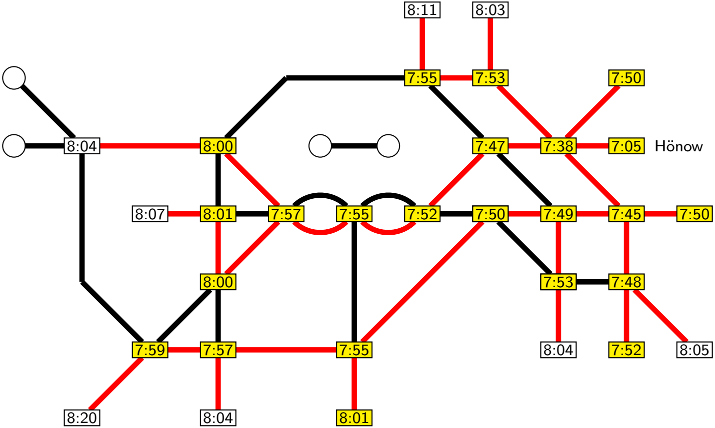
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

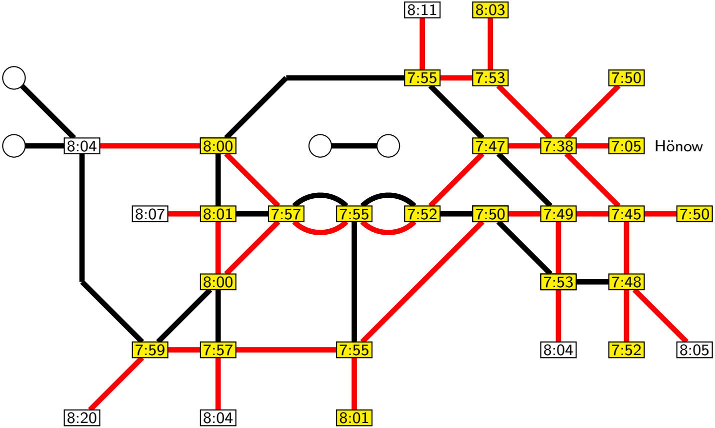
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

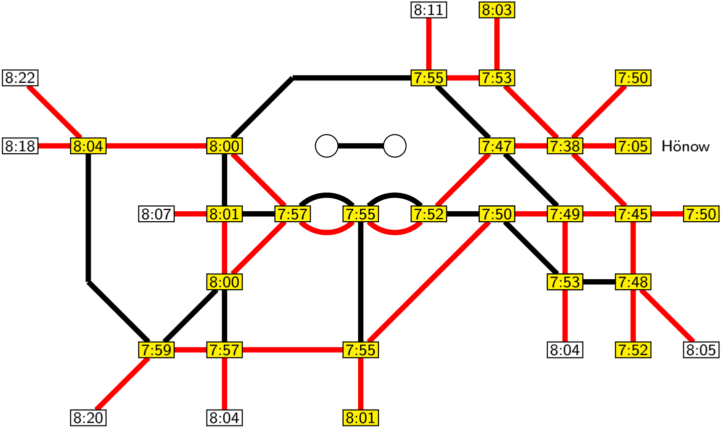
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

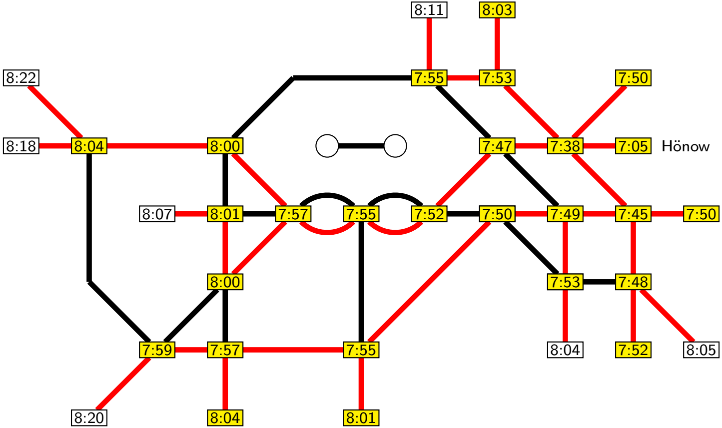
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

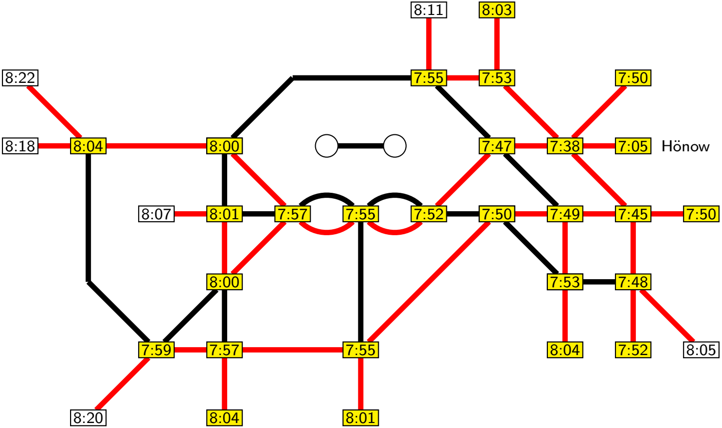
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

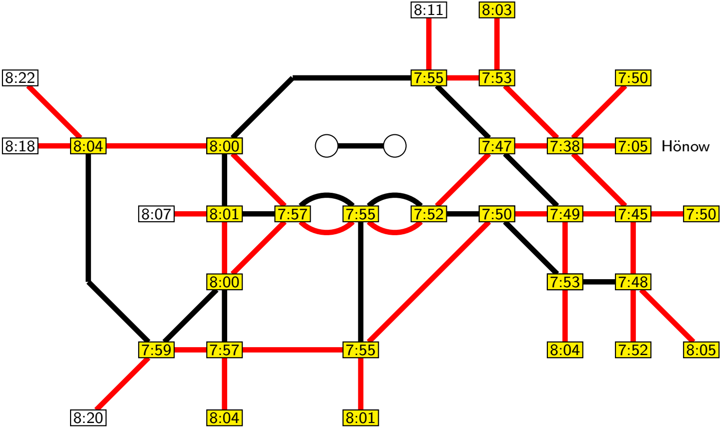
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

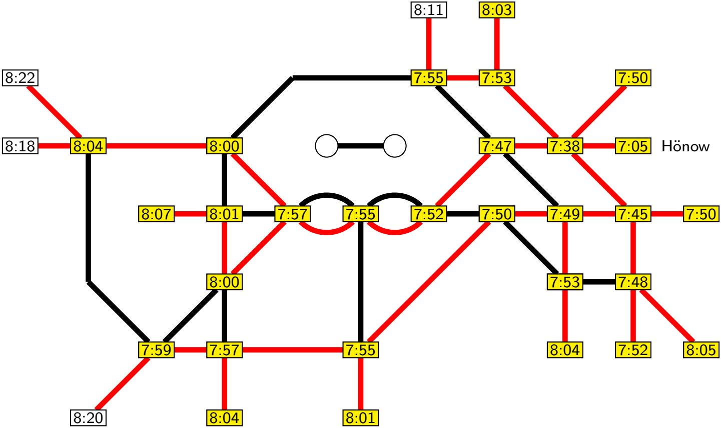
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

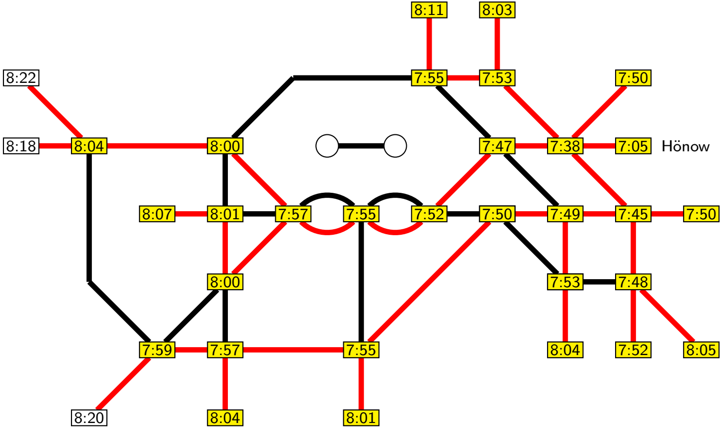
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

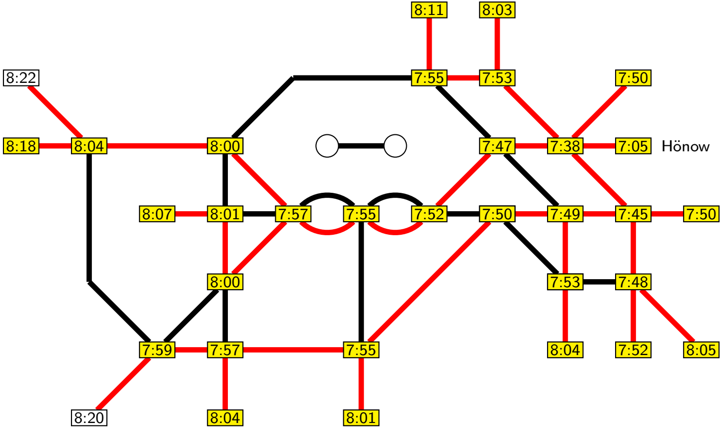
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

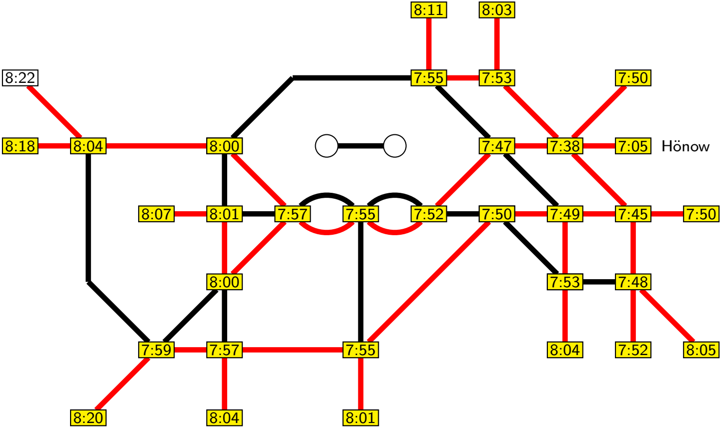
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

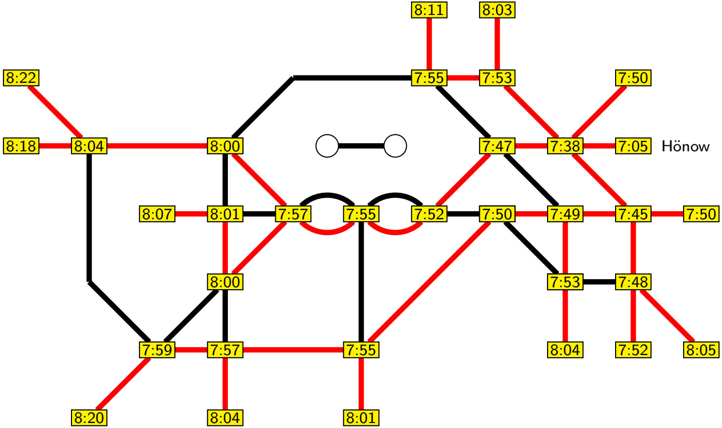
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

Query: Hönow @ 07:03 → all stations, without minimum change times



Chapter 2

Shortest Routes in Public Transportation Networks

§2.3 Advanced Methods

Topological orders

Let $G = (V, E)$ be a digraph.

Definition

A **topological order** of G is a total order $\leq$ on V such that for each edge $(v, w) \in E$ holds $v < w$.

Lemma

G has a topological order if and only if G contains no directed circuits.

Proof.

($\Rightarrow$) Let $\leq$ be a topological order. If $(v_1, \dots, v_k, v_1)$ is a directed circuit in G, then $v_1 < \dots < v_k < v_1$, contradicting the reflexivity of $\leq$.
($\Leftarrow$) Induction on the number of vertices: If $|V| = 1$ and G has no loops, then take $\{v \leq v\}$ as order. Now suppose $|V| > 1$. Since G has no directed circuit, there is a vertex v with out-degree 0. Removing v yields a graph having a topological order $\leq$ by the induction hypothesis. Append v to $\leq$ as largest element, i.e., $v > w$ for all $w \in V \setminus \{v\}$. □

Shortest Paths in DAGs

Definition

A **directed acyclic graph (DAG)** is a directed graph without directed circuits.

Example

Our (aperiodic) time expansions are DAGs if every driving activity has positive length.

Theorem

Let G be a directed acyclic graph on n vertices and m edges. The single-source shortest path problem in G – i.e., finding all shortest paths from a fixed source to all target vertices – can be solved in $\mathcal{O}(n + m)$ time.

Observation

Let $\leq$ be a topological order on a DAG. If there is a directed path from v to w, then $v < w$.

Shortest Paths in DAGs: Algorithm

Let $G = (V, E)$ be a DAG with an arbitrary length function $\ell : E \to \mathbb{R}$, and let $s \in V$ be a source vertex.

DAG Single-Source Shortest Path Algorithm

1. For all $v \in V$:
   - $\texttt{path}(v) := [s]$
   - $\texttt{distance}(v) := \begin{cases} 0 & \text{if } v = s, \\ \infty & \text{else} \end{cases}$
2. Compute a topological order $\leq$ on V.
3. For all $v \geq s$ sorted in ascending order w.r.t. $\leq$:
   - For all successors w of v:
     - If $\texttt{distance}(v) + \ell(v, w) < \texttt{distance}(w)$:
       $\texttt{distance}(w) := \texttt{distance}(v) + \ell(v, w)$
       $\texttt{path}(w) := \texttt{path}(v) + [w]$.
4. Return path, distance.

Shortest Paths in DAGs: Correctness

Claim

After v is scanned in Step 3, $\texttt{distance}(v)$ = length of a shortest s-v-path.

Proof.

- This is clear for $v = s$.
- Suppose $v > s$. Let $p = (s, u_1, \ldots, u_k, v)$ be a shortest s-v-path.
- Since $\texttt{distance}(v)$ is clearly the length of the path given by $\texttt{path}(v)$, we have $\texttt{distance}(v) \geq \ell(p)$.
- Moreover

$$\begin{aligned} \ell(p) &= \ell(s, u_1) + \sum_{i=1}^{k-1} \ell(u_i, u_{i+1}) + \ell(u_k, v) \\ &\geq \texttt{distance}(u_k) + \ell(u_k, v) && \text{induction hyp.} \\ &\geq \min_{(u,v)\in E} (\texttt{distance}(u) + \ell(u, v)) \\ &= \texttt{distance}(v). \end{aligned}$$

Shortest Paths in DAGs: Runtime

Claim

If G has n vertices and m edges, then the algorithm runs in $\mathcal{O}(n+m)$ time.

Proof.

Asymptotic complexity of the individual steps:

1. $\mathcal{O}(n)$
2. $\mathcal{O}(n+m)$ – compute a topological order by depth-first search (includes sorting)
3. $\mathcal{O}(m)$ – inner for-loop visits each edge at most once □

Remark

This yields a linear-time algorithm for shortest paths in time-expanded networks. Recall that building the graph is very expensive.

Idea

Use topological orders, but do not build the graph.

$\rightsquigarrow$ Connection Scan Algorithm (Dibbelt/Pajor/Strasser/Wagner, 2013)

Elementary connections

Consider a line network $\mathcal{N}$. Let $t_1, \ldots, t_k$ be the trips of a timetable for $\mathcal{N}$.

Definition

- An **elementary connection** on an edge $e = (v_{\text{dep}}, v_{\text{arr}}) \in E(\mathcal{N})$ is a 5-tuple $(v_{\text{dep}}, v_{\text{arr}}, \tau_{\text{dep}}(v_{\text{dep}}), \tau_{\text{arr}}(v_{\text{arr}}), i)$, where $t_i = (\tau_{\text{dep}}, \tau_{\text{arr}})$ is a trip of a line using e.
- If c is an elementary connection, we will write $v_{\text{dep}}(c), v_{\text{arr}}(c), \tau_{\text{dep}}(c), \tau_{\text{arr}}(c), \text{trip}(c)$ for its entries.

Observation

There is a one-to-one correspondence between

- elementary connections,
- driving activities of the time expansion,
- departure events of the time expansion,
- arrival events of the time expansion.

Aside: Dominant elementary connections

Definition

Let c, c' be elementary connections. Then c **dominates** c' if

(1) $v_{\text{dep}}(c) = v_{\text{dep}}(c')$ and $v_{\text{arr}}(c) = v_{\text{arr}}(c')$,

(2) $\tau_{\text{dep}}(c) \geq \tau_{\text{dep}}(c')$ and $\tau_{\text{arr}}(c) \leq \tau_{\text{arr}}(c')$.

Application to Time-Dependent Dijkstra

- In order to compute the time function, the time-dependent Dijkstra algorithm needs for every edge (v, w) a list of elementary connections on (v, w) sorted by τ_{dep}.
- The FIFO property means that connections with the earliest departures have the earliest arrivals.
- In particular, finding the connection with the earliest arrival is a simple binary search on the sorted list of elementary connections.
- If FIFO does not hold, one needs to find the first *dominant* elementary connection.

Connection Scan Algorithm (CSA)

Let $s@\tau \to t$ be an earliest arrival query, $s \neq t$.

Basic Connection Scan Algorithm (minimum transfer time τ_{min})

Preprocessing

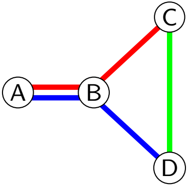
1. Sort all elementary connections by τ_{dep} in ascending order.

Query

1. $\texttt{time}(v) := \infty$ for all stops v, $\texttt{time}(s) := \tau$
   $\texttt{path}(v) := []$ for all stops v
   $\texttt{trip\_used}(i) := \mathit{false}$ for all trips i
2. For all elementary connections c increasing by $\tau_{\text{dep}}(c)$:
   - If $\texttt{trip\_used}(\mathit{trip}(c))$ or $\texttt{time}(v_{\text{dep}}(c)) \leq \tau_{\text{dep}}(c)$:
     - $\texttt{trip\_used}(\mathit{trip}(c)) := \mathit{true}$
     - If $\tau_{\text{arr}}(c) + \tau_{\text{min}} < \texttt{time}(v_{\text{arr}}(c))$:
       $\texttt{time}(v_{\text{arr}}(c)) := \tau_{\text{arr}}(c) + \tau_{\text{min}}$
       $\texttt{path}(v_{\text{arr}}(c)) := \texttt{path}(v_{\text{dep}}(c)) + [c]$
3. Return $\texttt{path}(t)$, $\texttt{time}(t) - \tau_{\text{min}}$.

CSA: Example

Query: A@7:00 $\rightarrow$ D, $\tau_{\text{min}} = 2$ minutes

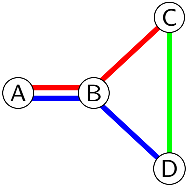


$v_{\text{dep}}, v_{\text{arr}}, \tau_{\text{dep}}, \tau_{\text{arr}},$ trip

```
A,B,7:00,7:05,1    C,B,7:05,7:12,8
B,C,7:05,7:12,1    B,A,7:12,7:18,8
A,B,7:10,7:15,2    C,B,7:15,7:22,9
B,C,7:15,7:22,2    B,A,7:22,7:28,9
A,B,7:05,7:10,3    D,B,7:05,7:16,10
B,D,7:10,7:21,3    B,A,7:16,7:21,10
A,B,7:15,7:20,4    D,B,7:15,7:26,11
B,D,7:20,7:31,4    B,A,7:26,7:31,11
C,D,7:04,7:10,5    D,C,7:04,7:10,12
C,D,7:14,7:20,6    D,C,7:14,7:20,13
C,D,7:24,7:30,7    D,C,7:24,7:30,14
```

CSA: Example

Query: A@7:00 → D, $\tau_{min} = 2$ minutes

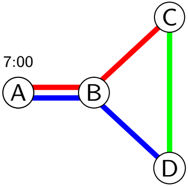


$v_{dep}, v_{arr}, \tau_{dep}, \tau_{arr}$, trip

A,B,7:00,7:05,1	D,C,7:14,7:20,13
C,D,7:04,7:10,5	B,C,7:15,7:22,2
D,C,7:04,7:10,12	A,B,7:15,7:20,4
B,C,7:05,7:12,1	C,B,7:15,7:22,9
A,B,7:05,7:10,3	D,B,7:15,7:26,11
C,B,7:05,7:12,8	B,A,7:16,7:21,10
D,B,7:05,7:16,10	B,D,7:20,7:31,4
A,B,7:10,7:15,2	B,A,7:22,7:28,9
B,D,7:10,7:21,3	C,D,7:24,7:30,7
B,A,7:12,7:18,8	D,C,7:24,7:30,14
C,D,7:14,7:20,6	B,A,7:26,7:31,11

CSA: Example

Query: A@7:00 → D, $\tau_{\min} = 2$ minutes



used trips:

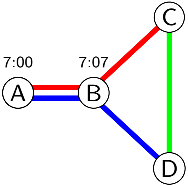
$\emptyset$

$v_{dep}, v_{arr}, \tau_{dep}, \tau_{arr}$, trip

```
A,B,7:00,7:05,1
C,D,7:04,7:10,5
D,C,7:04,7:10,12
B,C,7:05,7:12,1
A,B,7:05,7:10,3
C,B,7:05,7:12,8
D,B,7:05,7:16,10
A,B,7:10,7:15,2
B,D,7:10,7:21,3
B,A,7:12,7:18,8
C,D,7:14,7:20,6
D,C,7:14,7:20,13
B,C,7:15,7:22,2
A,B,7:15,7:20,4
C,B,7:15,7:22,9
D,B,7:15,7:26,11
B,A,7:16,7:21,10
B,D,7:20,7:31,4
B,A,7:22,7:28,9
C,D,7:24,7:30,7
D,C,7:24,7:30,14
B,A,7:26,7:31,11
```

CSA: Example

Query: A@7:00 → D, $\tau_{\min} = 2$ minutes



used trips:

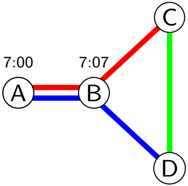
1

$v_{\text{dep}}, v_{\text{arr}}, \tau_{\text{dep}}, \tau_{\text{arr}}$, trip

~~A,B,7:00,7:05,1~~	D,C,7:14,7:20,13
C,D,7:04,7:10,5	B,C,7:15,7:22,2
D,C,7:04,7:10,12	A,B,7:15,7:20,4
B,C,7:05,7:12,1	C,B,7:15,7:22,9
A,B,7:05,7:10,3	D,B,7:15,7:26,11
C,B,7:05,7:12,8	B,A,7:16,7:21,10
D,B,7:05,7:16,10	B,D,7:20,7:31,4
A,B,7:10,7:15,2	B,A,7:22,7:28,9
B,D,7:10,7:21,3	C,D,7:24,7:30,7
B,A,7:12,7:18,8	D,C,7:24,7:30,14
C,D,7:14,7:20,6	B,A,7:26,7:31,11

CSA: Example

Query: A@7:00 → D, $\tau_{\text{min}} = 2$ minutes



used trips:

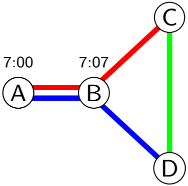
1

$v_{\text{dep}}, v_{\text{arr}}, \tau_{\text{dep}}, \tau_{\text{arr}}$, trip

~~A,B,7:00,7:05,1~~
~~C,D,7:04,7:10,5~~
D,C,7:04,7:10,12
B,C,7:05,7:12,1
A,B,7:05,7:10,3
C,B,7:05,7:12,8
D,B,7:05,7:16,10
A,B,7:10,7:15,2
B,D,7:10,7:21,3
B,A,7:12,7:18,8
C,D,7:14,7:20,6
D,C,7:14,7:20,13
B,C,7:15,7:22,2
A,B,7:15,7:20,4
C,B,7:15,7:22,9
D,B,7:15,7:26,11
B,A,7:16,7:21,10
B,D,7:20,7:31,4
B,A,7:22,7:28,9
C,D,7:24,7:30,7
D,C,7:24,7:30,14
B,A,7:26,7:31,11

CSA: Example

Query: A@7:00 → D, $\tau_{\text{min}} = 2$ minutes



used trips:

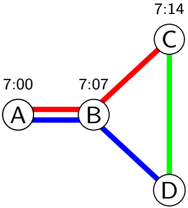
1

$v_{\text{dep}}, v_{\text{arr}}, \tau_{\text{dep}}, \tau_{\text{arr}}$, trip

~~A,B,7:00,7:05,1~~
~~C,D,7:04,7:10,5~~
~~D,C,7:04,7:10,12~~
B,C,7:05,7:12,1
A,B,7:05,7:10,3
C,B,7:05,7:12,8
D,B,7:05,7:16,10
A,B,7:10,7:15,2
B,D,7:10,7:21,3
B,A,7:12,7:18,8
C,D,7:14,7:20,6
D,C,7:14,7:20,13
B,C,7:15,7:22,2
A,B,7:15,7:20,4
C,B,7:15,7:22,9
D,B,7:15,7:26,11
B,A,7:16,7:21,10
B,D,7:20,7:31,4
B,A,7:22,7:28,9
C,D,7:24,7:30,7
D,C,7:24,7:30,14
B,A,7:26,7:31,11

CSA: Example

Query: A@7:00 → D, $\tau_{\min} = 2$ minutes



used trips:

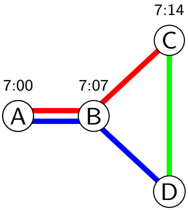
1

$v_{\text{dep}}, v_{\text{arr}}, \tau_{\text{dep}}, \tau_{\text{arr}}$, trip

~~A,B,7:00,7:05,1~~
~~C,D,7:04,7:10,5~~
~~D,C,7:04,7:10,12~~
~~B,C,7:05,7:12,1~~
A,B,7:05,7:10,3
C,B,7:05,7:12,8
D,B,7:05,7:16,10
A,B,7:10,7:15,2
B,D,7:10,7:21,3
B,A,7:12,7:18,8
C,D,7:14,7:20,6
D,C,7:14,7:20,13
B,C,7:15,7:22,2
A,B,7:15,7:20,4
C,B,7:15,7:22,9
D,B,7:15,7:26,11
B,A,7:16,7:21,10
B,D,7:20,7:31,4
B,A,7:22,7:28,9
C,D,7:24,7:30,7
D,C,7:24,7:30,14
B,A,7:26,7:31,11

CSA: Example

Query: A@7:00 → D, $\tau_{\text{min}} = 2$ minutes



used trips:

1, 3

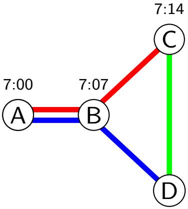
$v_{\text{dep}}, v_{\text{arr}}, \tau_{\text{dep}}, \tau_{\text{arr}}$, trip

~~A,B,7:00,7:05,1~~
~~C,D,7:04,7:10,5~~
~~D,C,7:04,7:10,12~~
~~B,C,7:05,7:12,1~~
~~A,B,7:05,7:10,3~~
C,B,7:05,7:12,8
D,B,7:05,7:16,10
A,B,7:10,7:15,2
B,D,7:10,7:21,3
B,A,7:12,7:18,8
C,D,7:14,7:20,6

D,C,7:14,7:20,13
B,C,7:15,7:22,2
A,B,7:15,7:20,4
C,B,7:15,7:22,9
D,B,7:15,7:26,11
B,A,7:16,7:21,10
B,D,7:20,7:31,4
B,A,7:22,7:28,9
C,D,7:24,7:30,7
D,C,7:24,7:30,14
B,A,7:26,7:31,11

CSA: Example

Query: A@7:00 → D, $\tau_{\min} = 2$ minutes



used trips:

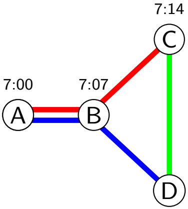
1, 3

$v_{\text{dep}}, v_{\text{arr}}, \tau_{\text{dep}}, \tau_{\text{arr}},$ trip

~~A,B,7:00,7:05,1~~
~~C,D,7:04,7:10,5~~
~~D,C,7:04,7:10,12~~
~~B,C,7:05,7:12,1~~
~~A,B,7:05,7:10,3~~
~~C,B,7:05,7:12,8~~
D,B,7:05,7:16,10
A,B,7:10,7:15,2
B,D,7:10,7:21,3
B,A,7:12,7:18,8
C,D,7:14,7:20,6
D,C,7:14,7:20,13
B,C,7:15,7:22,2
A,B,7:15,7:20,4
C,B,7:15,7:22,9
D,B,7:15,7:26,11
B,A,7:16,7:21,10
B,D,7:20,7:31,4
B,A,7:22,7:28,9
C,D,7:24,7:30,7
D,C,7:24,7:30,14
B,A,7:26,7:31,11

CSA: Example

Query: A@7:00 → D, $\tau_{\text{min}} = 2$ minutes



used trips:

1, 3

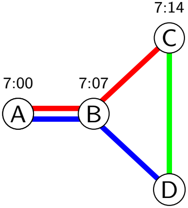
$v_{\text{dep}}, v_{\text{arr}}, \tau_{\text{dep}}, \tau_{\text{arr}}$, trip

~~A,B,7:00,7:05,1~~
~~C,D,7:04,7:10,5~~
~~D,C,7:04,7:10,12~~
~~B,C,7:05,7:12,1~~
~~A,B,7:05,7:10,3~~
~~C,B,7:05,7:12,8~~
~~D,B,7:05,7:16,10~~
A,B,7:10,7:15,2
B,D,7:10,7:21,3
B,A,7:12,7:18,8
C,D,7:14,7:20,6

D,C,7:14,7:20,13
B,C,7:15,7:22,2
A,B,7:15,7:20,4
C,B,7:15,7:22,9
D,B,7:15,7:26,11
B,A,7:16,7:21,10
B,D,7:20,7:31,4
B,A,7:22,7:28,9
C,D,7:24,7:30,7
D,C,7:24,7:30,14
B,A,7:26,7:31,11

CSA: Example

Query: A@7:00 → D, $\tau_{\min} = 2$ minutes



used trips:

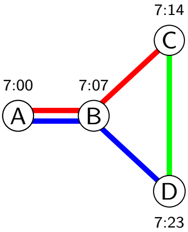
1, 3, 2

$v_{\text{dep}}, v_{\text{arr}}, \tau_{\text{dep}}, \tau_{\text{arr}}$, trip

~~A,B,7:00,7:05,1~~
~~C,D,7:04,7:10,5~~
~~D,C,7:04,7:10,12~~
~~B,C,7:05,7:12,1~~
~~A,B,7:05,7:10,3~~
~~C,B,7:05,7:12,8~~
~~D,B,7:05,7:16,10~~
~~A,B,7:10,7:15,2~~
B,D,7:10,7:21,3
B,A,7:12,7:18,8
C,D,7:14,7:20,6
D,C,7:14,7:20,13
B,C,7:15,7:22,2
A,B,7:15,7:20,4
C,B,7:15,7:22,9
D,B,7:15,7:26,11
B,A,7:16,7:21,10
B,D,7:20,7:31,4
B,A,7:22,7:28,9
C,D,7:24,7:30,7
D,C,7:24,7:30,14
B,A,7:26,7:31,11

CSA: Example

Query: A@7:00 → D, $\tau_{\text{min}} = 2$ minutes



used trips:

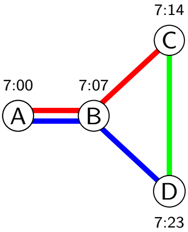
1, 3, 2

$v_{\text{dep}}, v_{\text{arr}}, \tau_{\text{dep}}, \tau_{\text{arr}}$, trip

~~A,B,7:00,7:05,1~~
~~C,D,7:04,7:10,5~~
~~D,C,7:04,7:10,12~~
~~B,C,7:05,7:12,1~~
~~A,B,7:05,7:10,3~~
~~C,B,7:05,7:12,8~~
~~D,B,7:05,7:16,10~~
~~A,B,7:10,7:15,2~~
~~B,D,7:10,7:21,3~~
B,A,7:12,7:18,8
C,D,7:14,7:20,6
D,C,7:14,7:20,13
B,C,7:15,7:22,2
A,B,7:15,7:20,4
C,B,7:15,7:22,9
D,B,7:15,7:26,11
B,A,7:16,7:21,10
B,D,7:20,7:31,4
B,A,7:22,7:28,9
C,D,7:24,7:30,7
D,C,7:24,7:30,14
B,A,7:26,7:31,11

CSA: Example

Query: A@7:00 → D, $\tau_{\min} = 2$ minutes



used trips:

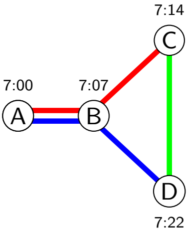
1, 3, 2, 8

$v_{\text{dep}}, v_{\text{arr}}, \tau_{\text{dep}}, \tau_{\text{arr}}$, trip

~~A,B,7:00,7:05,1~~	D,C,7:14,7:20,13
~~C,D,7:04,7:10,5~~	B,C,7:15,7:22,2
~~D,C,7:04,7:10,12~~	A,B,7:15,7:20,4
~~B,C,7:05,7:12,1~~	C,B,7:15,7:22,9
~~A,B,7:05,7:10,3~~	D,B,7:15,7:26,11
~~C,B,7:05,7:12,8~~	B,A,7:16,7:21,10
~~D,B,7:05,7:16,10~~	B,D,7:20,7:31,4
~~A,B,7:10,7:15,2~~	B,A,7:22,7:28,9
~~B,D,7:10,7:21,3~~	C,D,7:24,7:30,7
~~B,A,7:12,7:18,8~~	D,C,7:24,7:30,14
C,D,7:14,7:20,6	B,A,7:26,7:31,11

CSA: Example

Query: A@7:00 → D, $\tau_{\text{min}} = 2$ minutes



used trips:

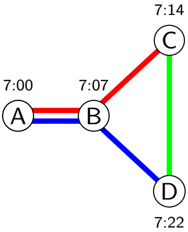
1, 3, 2, 8, 6

$v_{\text{dep}}, v_{\text{arr}}, \tau_{\text{dep}}, \tau_{\text{arr}},$ trip

~~A,B,7:00,7:05,1~~
~~C,D,7:04,7:10,5~~
~~D,C,7:04,7:10,12~~
~~B,C,7:05,7:12,1~~
~~A,B,7:05,7:10,3~~
~~C,B,7:05,7:12,8~~
~~D,B,7:05,7:16,10~~
~~A,B,7:10,7:15,2~~
~~B,D,7:10,7:21,3~~
~~B,A,7:12,7:18,8~~
~~C,D,7:14,7:20,6~~
D,C,7:14,7:20,13
B,C,7:15,7:22,2
A,B,7:15,7:20,4
C,B,7:15,7:22,9
D,B,7:15,7:26,11
B,A,7:16,7:21,10
B,D,7:20,7:31,4
B,A,7:22,7:28,9
C,D,7:24,7:30,7
D,C,7:24,7:30,14
B,A,7:26,7:31,11

CSA: Example

Query: A@7:00 → D, $\tau_{\text{min}} = 2$ minutes



used trips:

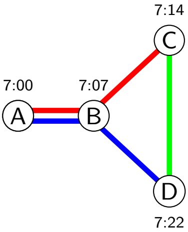
1, 3, 2, 8, 6

$v_{\text{dep}}, v_{\text{arr}}, \tau_{\text{dep}}, \tau_{\text{arr}}$, trip

~~A,B,7:00,7:05,1~~
~~C,D,7:04,7:10,5~~
~~D,C,7:04,7:10,12~~
~~B,C,7:05,7:12,1~~
~~A,B,7:05,7:10,3~~
~~C,B,7:05,7:12,8~~
~~D,B,7:05,7:16,10~~
~~A,B,7:10,7:15,2~~
~~B,D,7:10,7:21,3~~
~~B,A,7:12,7:18,8~~
~~C,D,7:14,7:20,6~~
~~D,C,7:14,7:20,13~~
B,C,7:15,7:22,2
A,B,7:15,7:20,4
C,B,7:15,7:22,9
D,B,7:15,7:26,11
B,A,7:16,7:21,10
B,D,7:20,7:31,4
B,A,7:22,7:28,9
C,D,7:24,7:30,7
D,C,7:24,7:30,14
B,A,7:26,7:31,11

CSA: Example

Query: A@7:00 → D, $\tau_{\text{min}} = 2$ minutes



used trips:

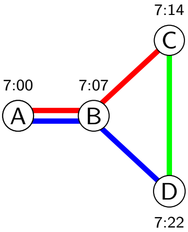
1, 3, 2, 8, 6

$v_{\text{dep}}, v_{\text{arr}}, \tau_{\text{dep}}, \tau_{\text{arr}}$, trip

~~A,B,7:00,7:05,1~~
~~C,D,7:04,7:10,5~~
~~D,C,7:04,7:10,12~~
~~B,C,7:05,7:12,1~~
~~A,B,7:05,7:10,3~~
~~C,B,7:05,7:12,8~~
~~D,B,7:05,7:16,10~~
~~A,B,7:10,7:15,2~~
~~B,D,7:10,7:21,3~~
~~B,A,7:12,7:18,8~~
~~C,D,7:14,7:20,6~~
~~D,C,7:14,7:20,13~~
~~B,C,7:15,7:22,2~~
A,B,7:15,7:20,4
C,B,7:15,7:22,9
D,B,7:15,7:26,11
B,A,7:16,7:21,10
B,D,7:20,7:31,4
B,A,7:22,7:28,9
C,D,7:24,7:30,7
D,C,7:24,7:30,14
B,A,7:26,7:31,11

CSA: Example

Query: A@7:00 → D, $\tau_{\min} = 2$ minutes



used trips:

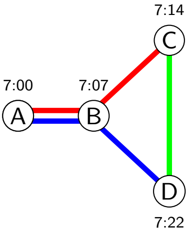
1, 3, 2, 8, 6, 4

$v_{\text{dep}}, v_{\text{arr}}, \tau_{\text{dep}}, \tau_{\text{arr}}$, trip

~~A,B,7:00,7:05,1~~
~~C,D,7:04,7:10,5~~
~~D,C,7:04,7:10,12~~
~~B,C,7:05,7:12,1~~
~~A,B,7:05,7:10,3~~
~~C,B,7:05,7:12,8~~
~~D,B,7:05,7:16,10~~
~~A,B,7:10,7:15,2~~
~~B,D,7:10,7:21,3~~
~~B,A,7:12,7:18,8~~
~~C,D,7:14,7:20,6~~
~~D,C,7:14,7:20,13~~
~~B,C,7:15,7:22,2~~
~~A,B,7:15,7:20,4~~
C,B,7:15,7:22,9
D,B,7:15,7:26,11
B,A,7:16,7:21,10
B,D,7:20,7:31,4
B,A,7:22,7:28,9
C,D,7:24,7:30,7
D,C,7:24,7:30,14
B,A,7:26,7:31,11

CSA: Example

Query: A@7:00 → D, $\tau_{\min} = 2$ minutes



used trips:

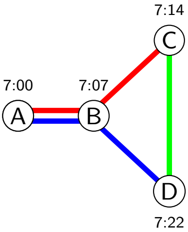
1, 3, 2, 8, 6, 4, 9

$v_{\text{dep}}, v_{\text{arr}}, \tau_{\text{dep}}, \tau_{\text{arr}},$ trip

~~A,B,7:00,7:05,1~~
~~C,D,7:04,7:10,5~~
~~D,C,7:04,7:10,12~~
~~B,C,7:05,7:12,1~~
~~A,B,7:05,7:10,3~~
~~C,B,7:05,7:12,8~~
~~D,B,7:05,7:16,10~~
~~A,B,7:10,7:15,2~~
~~B,D,7:10,7:21,3~~
~~B,A,7:12,7:18,8~~
~~C,D,7:14,7:20,6~~
~~D,C,7:14,7:20,13~~
~~B,C,7:15,7:22,2~~
~~A,B,7:15,7:20,4~~
~~C,B,7:15,7:22,9~~
D,B,7:15,7:26,11
B,A,7:16,7:21,10
B,D,7:20,7:31,4
B,A,7:22,7:28,9
C,D,7:24,7:30,7
D,C,7:24,7:30,14
B,A,7:26,7:31,11

CSA: Example

Query: A@7:00 → D, $\tau_{min} = 2$ minutes



used trips:

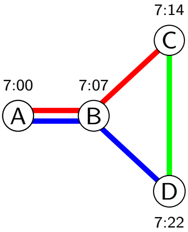
1, 3, 2, 8, 6, 4, 9

$v_{dep}, v_{arr}, \tau_{dep}, \tau_{arr}$, trip

~~A,B,7:00,7:05,1~~
~~C,D,7:04,7:10,5~~
~~D,C,7:04,7:10,12~~
~~B,C,7:05,7:12,1~~
~~A,B,7:05,7:10,3~~
~~C,B,7:05,7:12,8~~
~~D,B,7:05,7:16,10~~
~~A,B,7:10,7:15,2~~
~~B,D,7:10,7:21,3~~
~~B,A,7:12,7:18,8~~
~~C,D,7:14,7:20,6~~
~~D,C,7:14,7:20,13~~
~~B,C,7:15,7:22,2~~
~~A,B,7:15,7:20,4~~
~~C,B,7:15,7:22,9~~
~~D,B,7:15,7:26,11~~
B,A,7:16,7:21,10
B,D,7:20,7:31,4
B,A,7:22,7:28,9
C,D,7:24,7:30,7
D,C,7:24,7:30,14
B,A,7:26,7:31,11

CSA: Example

Query: A@7:00 → D, $\tau_{\text{min}} = 2$ minutes



used trips:

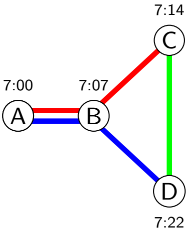
1, 3, 2, 8, 6, 4, 9, 10

$v_{\text{dep}}, v_{\text{arr}}, \tau_{\text{dep}}, \tau_{\text{arr}}$, trip

~~A,B,7:00,7:05,1~~
~~C,D,7:04,7:10,5~~
~~D,C,7:04,7:10,12~~
~~B,C,7:05,7:12,1~~
~~A,B,7:05,7:10,3~~
~~C,B,7:05,7:12,8~~
~~D,B,7:05,7:16,10~~
~~A,B,7:10,7:15,2~~
~~B,D,7:10,7:21,3~~
~~B,A,7:12,7:18,8~~
~~C,D,7:14,7:20,6~~
~~D,C,7:14,7:20,13~~
~~B,C,7:15,7:22,2~~
~~A,B,7:15,7:20,4~~
~~C,B,7:15,7:22,9~~
~~D,B,7:15,7:26,11~~
~~B,A,7:16,7:21,10~~
B,D,7:20,7:31,4
B,A,7:22,7:28,9
C,D,7:24,7:30,7
D,C,7:24,7:30,14
B,A,7:26,7:31,11

CSA: Example

Query: A@7:00 → D, $\tau_{\text{min}} = 2$ minutes



used trips:

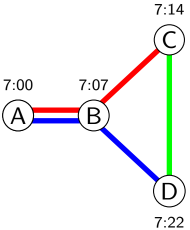
1, 3, 2, 8, 6, 4, 9, 10

$v_{\text{dep}}, v_{\text{arr}}, \tau_{\text{dep}}, \tau_{\text{arr}}$, trip

~~A,B,7:00,7:05,1~~
~~C,D,7:04,7:10,5~~
~~D,C,7:04,7:10,12~~
~~B,C,7:05,7:12,1~~
~~A,B,7:05,7:10,3~~
~~C,B,7:05,7:12,8~~
~~D,B,7:05,7:16,10~~
~~A,B,7:10,7:15,2~~
~~B,D,7:10,7:21,3~~
~~B,A,7:12,7:18,8~~
~~C,D,7:14,7:20,6~~
~~D,C,7:14,7:20,13~~
~~B,C,7:15,7:22,2~~
~~A,B,7:15,7:20,4~~
~~C,B,7:15,7:22,9~~
~~D,B,7:15,7:26,11~~
~~B,A,7:16,7:21,10~~
~~B,D,7:20,7:31,4~~
B,A,7:22,7:28,9
C,D,7:24,7:30,7
D,C,7:24,7:30,14
B,A,7:26,7:31,11

CSA: Example

Query: A@7:00 → D, $\tau_{\text{min}} = 2$ minutes



used trips:

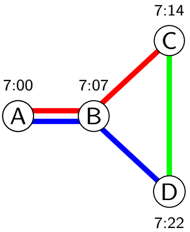
1, 3, 2, 8, 6, 4, 9, 10, 7, 14, 11

$v_{\text{dep}}, v_{\text{arr}}, \tau_{\text{dep}}, \tau_{\text{arr}}$, trip

~~A,B,7:00,7:05,1~~
~~C,D,7:04,7:10,5~~
~~D,C,7:04,7:10,12~~
~~B,C,7:05,7:12,1~~
~~A,B,7:05,7:10,3~~
~~C,B,7:05,7:12,8~~
~~D,B,7:05,7:16,10~~
~~A,B,7:10,7:15,2~~
~~B,D,7:10,7:21,3~~
~~B,A,7:12,7:18,8~~
~~C,D,7:14,7:20,6~~
~~D,C,7:14,7:20,13~~
~~B,C,7:15,7:22,2~~
~~A,B,7:15,7:20,4~~
~~C,B,7:15,7:22,9~~
~~D,B,7:15,7:26,11~~
~~B,A,7:16,7:21,10~~
~~B,D,7:20,7:31,4~~
~~B,A,7:22,7:28,9~~
~~C,D,7:24,7:30,7~~
~~D,C,7:24,7:30,14~~
~~B,A,7:26,7:31,11~~

CSA: Example

Query: A@7:00 → D, $\tau_{\text{min}} = 2$ minutes



used trips:
1, 3, 2, 8, 6, 4, 9, 10, 7, 14, 11

$v_{\text{dep}}, v_{\text{arr}}, \tau_{\text{dep}}, \tau_{\text{arr}}$, trip

~~A,B,7:00,7:05,1~~
~~C,D,7:04,7:10,5~~
~~D,C,7:04,7:10,12~~
~~B,C,7:05,7:12,1~~
~~A,B,7:05,7:10,3~~
~~C,B,7:05,7:12,8~~
~~D,B,7:05,7:16,10~~
~~A,B,7:10,7:15,2~~
~~B,D,7:10,7:21,3~~
~~B,A,7:12,7:18,8~~
~~C,D,7:14,7:20,6~~
~~D,C,7:14,7:20,13~~
~~B,C,7:15,7:22,2~~
~~A,B,7:15,7:20,4~~
~~C,B,7:15,7:22,9~~
~~D,B,7:15,7:26,11~~
~~B,A,7:16,7:21,10~~
~~B,D,7:20,7:31,4~~
~~B,A,7:22,7:28,9~~
~~C,D,7:24,7:30,7~~
~~D,C,7:24,7:30,14~~
~~B,A,7:26,7:31,11~~

⇝ earliest arrival time at D: 7:20

CSA: Interpretation

Interpretation

- Scanning the elementary connections in ascending order by departure time is equivalent to scanning departure events of the time-expanded graph w.r.t. the topological order given by departure time.
- The successor of a departure event is a single arrival event, whose successors are in turn several departure events. Hence it makes sense to contract the arrival event and to view the next departure events as direct successors.
- The CSA does the following: If a connection c departs at $v_{\mathrm{dep}}(c)$ at time $\tau_{\mathrm{dep}}(c)$, then set the time at the next departure event at $v_{\mathrm{arr}}(c)$ to $\tau_{\mathrm{arr}}(c) + \tau_{\mathrm{min}}$ if this improves the value in `time`.
- This is done if the $\tau_{\mathrm{dep}}(c) \geq \mathtt{time}(v_{\mathrm{dep}}(c)) = \tau_{\mathrm{arr}}(c') + \tau_{\mathrm{min}}$ for some connection c' – i.e., including a transfer – or if the same trip has been used before, disregarding the transfer time.
- The transfer time to the last departure event should be subtracted.

CSA: More

More Remarks

- $\texttt{path}(t)$ contains the elementary connections on the computed route.
- In fact, CSA solves the earliest arrival problem for *every* target vertex.

Optimizations

- Starting criterion: A connection c needs only to be scanned if $\tau_{\mathsf{dep}}(c) \geq \tau$. $\rightsquigarrow$ Determine the first connection departing after τ by binary search.
- Stopping criterion: In an $s\texttt{@}\tau \to t$ earliest arrival query, CSA can be stopped if it scans a connection c with $\tau_{\mathsf{dep}}(c) \geq \texttt{time}(t)$, because $\texttt{time}(t)$ then cannot be improved further.

CSA: Footpaths

Definition

A **footpath graph** for a line network $\mathcal{N}$ is a digraph F with a length function $\ell : E(F) \to \mathbb{R}_{\geq 0}$ such that

(1) $V(F) = V(\mathcal{N})$,

(2) F is transitively closed, i.e., $(u, v), (v, w) \in E(F) \Rightarrow (u, w) \in E(F)$,

(3) ℓ satisfies the triangle inequality.

Application to transfers

- Transfers between stops of the line network are modeled by edges of the footpath graph, and their duration is given by ℓ.
- Minimum transfer times at a stop correspond to loops in the footpath graph.
- In more detailed line networks, e.g., when each platform or bus stop location corresponds to a vertex, there might also be loops of length 0.

CSA: Algorithm with footpaths

Connection Scan Algorithm (with footpath graph (F, ℓ))

Preprocessing

1. Sort all elementary connections by τ_{dep} in ascending order.

Query

1. $\texttt{time}(v) := \infty$ and $\texttt{path}(v) := []$ for all stops v,
   $\texttt{time}(v) := \tau + \ell(s, v)$ and $\texttt{path}(v) := [(s, v)]$ for all $(s, v) \in E(F)$,
   $\texttt{trip\_used}(i) :=$ *none* for all trips i
2. For all elementary connections c with $\tau_{\text{dep}}(c) \geq \tau$, in asc. order:
   If $\tau_{\text{dep}}(c) \geq \texttt{time}(t)$, go to Step 3.
   If $\texttt{trip\_used}(\textit{trip}(c)) \neq \textit{none}$ or $\texttt{time}(v_{\text{dep}}(c)) \leq \tau_{\text{dep}}(c)$:
   If $\texttt{trip\_used}(\textit{trip}(c)) = \textit{none}$:
   $\texttt{trip\_used}(\textit{trip}(c)) := c$
   If $\tau_{\text{arr}}(c) < \texttt{time}(v_{\text{arr}}(c))$:
   For all $f = (v_{\text{arr}}(c), w) \in E(F)$:
   If $\tau_{\text{arr}}(c) + \ell(f) < \texttt{time}(w)$:
   $\texttt{time}(w) := \tau_{\text{arr}}(c) + \ell(f)$
   $\texttt{path}(w) := \texttt{path}(v_{\text{dep}}(c)) + [(\texttt{trip\_used}(\textit{trip}(c)), c), f]$
3. Return $\texttt{path}(t)$, $\texttt{time}(t)$.

CSA: Algorithm with footpaths

Remarks

- Every *journey* computed by CSA (`path`) is an alternating sequence $(f_1, l_1, f_2, l_2, \dots, f_k, l_k, f_{k+1})$, where the f_i are footpaths and l_i are *legs*, i.e., subsequent elementary connections of the same trip.
- The field `path` returns the legs by specifying the enter and exit connections of each trip.
- The footpath graph needs loops at every station so that connections can be reached at all. Another variant would be to use a detailed model with *parent stations* connected to several platforms/bus stop locations/... In this model, there are footpaths between a parent station and its individual platforms. Queries run from parent station to parent station.
- This CSA version includes the starting and stopping criterion.
- Moreover, it uses the following *limited walking* strategy: Footpaths with $\tau_{\text{arr}}(c) \geq \texttt{time}(v_{\text{arr}}(c))$ are not considered.

CSA: Limited walking

Lemma

Suppose $\tau_{arr}(c) \geq \texttt{time}(v_{\text{arr}}(c))$. *Then no footpath* $(v_{arr}(c), w) \in E(F)$ *improves* $\texttt{time}(w)$.

Proof.

- Let $v := v_{\text{arr}}(c)$. Assume there is a footpath $(v, w) \in E(F)$ improving $\texttt{time}(w)$, i.e., $\texttt{time}(w) > \tau_{\text{arr}}(c) + \ell(v, w)$.
- Then by hypothesis, $\texttt{time}(w) > \texttt{time}(v) + \ell(v, w)$.
- If $\texttt{time}(v) = \infty$, then this is not improving.
- Otherwise $\texttt{time}(v) = \tau_{\text{arr}}(c') + \ell(u, v)$ for some connection c' arriving at some stop u and $(u, v) \in E(F)$.
- Thus $\texttt{time}(w) > \tau_{\text{arr}}(c') + \ell(u, v) + \ell(v, w)$. Since the footpath graph is transitive and ℓ satisfies the triangle inequality, there is an edge $(u, w) \in E(F)$ such that $\texttt{time}(w) > \tau_{\text{arr}}(c') + \ell(u, w)$.
- This is impossible, as CSA scanned c' and (u, w) before and would have set $\texttt{time}(w)$ accordingly.

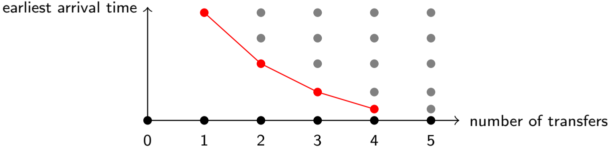
CSA: Extensions

Final remarks

- The main advantage of CSA is that it does not need to build the graph. Moreover, elementary connections are accessed sequentially, only the footpath graph needs random access.
- However, it scans *every* connection, also very distant and unlikely ones. Correcting this is one of the ideas of *accelerated CSA*.
- There is a profile version of CSA.
- There is a version finding the Pareto optimal solutions w.r.t. earliest arrival time and minimum number of transfers.

RAPTOR: Overview

- The next algorithm is called RAPTOR (Delling/Pajor/Werneck, 2012), which is short for *Round-based public transit routing*.
- The RAPTOR algorithm is designed to solve the two-criteria problem w.r.t. earliest arrival time and minimum number of transfers in a Pareto sense: For $k \in \mathbb{N}_0$, it computes the earliest arrival time w.r.t. a journey with at most k transfers.



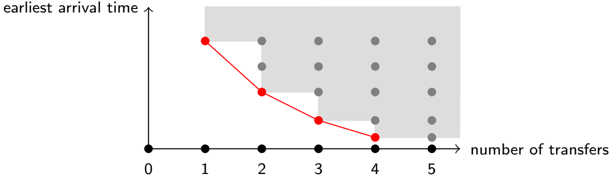
- The algorithm works therefore with rounds, and each round increases the number of transfers by 1, so it is a *dynamic program*.
- RAPTOR does not use graphs as underlying data structure.

Pareto optimization

Definition

Let X be a set and let $f_1, \ldots, f_k : X \to \mathbb{R}$ be functions.

- The problem $\min_{x \in X}(f_1(x), \ldots, f_k(x))$ is called a **multi-criteria minimization problem**.
- An element $x \in X$ **dominates** $y \in X$ if $f_i(x) \leq f_i(y)$ for all i.
- An element $x \in X$ is **Pareto-optimal** if x is not dominated by another element of X.
- The **Pareto set** or **Pareto front** is the set of all Pareto-optimal elements of X.



RAPTOR: Pareto optimization

Application to RAPTOR

- A *journey* is a sequence of elementary connections and footpaths in the order of travel.
- A journey J dominates a journey J' if J arrives no later than J' and J does not use more than transfers than J'.
- Given an earliest arrival problem $s@\tau \to t$, a Pareto set is a maximal set of pairwise non-dominating journeys from s to t not departing before τ.

RAPTOR: Model

Input data structures

RAPTOR works in principle on a directed line network with a timetable and a footpath graph. However, the information is usually organized in lists:

- `RouteStops`: for each line L a sorted list of the stops served by L,
- `Trips`: for each line L a sorted list of trips on L,
- `StopTimes`: for each trip a sorted list of departures and arrivals,
- `StopRoutes`: for each stop v a list of the lines serving v,
- `Transfers`: for each stop v a list of footpaths from v.

Comparison to the CSA model

The models for RAPTOR and CSA differ in the notion of journeys: There is not necessarily a footpath between trips. RAPTOR does hence not respect minimum transfer times if arrival and departure are at the same stop. This has to be modelled by introducing a stop for each platform/location.

RAPTOR: Algorithm

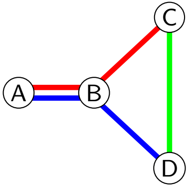
Basic RAPTOR Algorithm (K rounds)

1. $\texttt{time}(k, v) := \infty$ for each stop v and $k = 0, 1, \dots, K$, $\texttt{time}(0, s) := \tau$
2. For $k = 1, 2, \dots, K$:
   - $\texttt{time}(k, v) := \texttt{time}(k-1, v)$ for all v
   - For all lines $L = (v_1, \dots, v_r)$:
   - $\texttt{curr\_trip} := \emptyset$
     - For $j = 1, \dots, r$:
       - If $\texttt{curr\_trip} \neq \emptyset$ and $\tau_{\text{arr}}(v_j) < \texttt{time}(k, v_j)$:
         - $(\tau_{\text{dep}}, \tau_{\text{arr}}) := \texttt{curr\_trip}$
         - $\texttt{time}(k, v_j) := \tau_{\text{arr}}(v_j)$
         - $\texttt{path}(k, v_j) := \texttt{path}(k-1, v_i) + [(v_i, v_j, \tau_{\text{dep}}(v_i), \tau_{\text{arr}}(v_j), \texttt{curr\_trip})]$
       - If $\texttt{curr\_trip} = \emptyset$ or $\tau_{\text{dep}}(v_j) \geq \texttt{time}(k-1, v_j)$:
         - $\texttt{curr\_trip} := \texttt{trip}(k, v, L)$
   - For all footpaths (v, w):
     - If $\texttt{time}(k, v) + \ell(v, w) < \texttt{time}(k, w)$:
       - $\texttt{time}(k, w) := \texttt{time}(k, v) + \ell(v, w)$
       - $\texttt{path}(k, w) := \texttt{path}(k, v) + [(v, w)]$
3. Return $\texttt{path}(\cdot, t)$, $\texttt{time}(\cdot, t)$.

$\texttt{trip}(k, v, L) :=$ earliest trip $(\tau_{\text{dep}}, \tau_{\text{arr}})$ on L s.t. $\tau_{\text{dep}}(v) \geq \texttt{time}(k-1, v)$ if exists, else $\emptyset$

RAPTOR: Example

Query: A@7:00 → D, $K = 2$, no footpaths



current trip: ∅

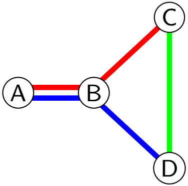
	0	1	2
A			
B			
C			
D			

$v_{dep}, v_{arr}, \tau_{dep}, \tau_{arr}$, trip

```
A,B,7:00,7:05,1
B,C,7:05,7:12,1
A,B,7:10,7:15,2
B,C,7:15,7:22,2
A,B,7:05,7:10,3
B,D,7:10,7:21,3
A,B,7:15,7:20,4
B,D,7:20,7:31,4
C,D,7:04,7:10,5
C,D,7:14,7:20,6
C,D,7:24,7:30,7
C,B,7:05,7:12,8
B,A,7:12,7:18,8
C,B,7:15,7:22,9
B,A,7:22,7:28,9
D,B,7:05,7:16,10
B,A,7:16,7:21,10
D,B,7:15,7:26,11
B,A,7:26,7:31,11
D,C,7:04,7:10,12
D,C,7:14,7:20,13
D,C,7:24,7:30,14
```

RAPTOR: Example

Query: A@7:00 → D, $K = 2$, no footpaths



current trip: ∅

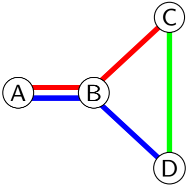
	0	1	2
A	7:00		
B	∞		
C	∞		
D	∞		

$v_{dep}, v_{arr}, \tau_{dep}, \tau_{arr}$, trip

```
A,B,7:00,7:05,1
B,C,7:05,7:12,1
A,B,7:10,7:15,2
B,C,7:15,7:22,2
A,B,7:05,7:10,3
B,D,7:10,7:21,3
A,B,7:15,7:20,4
B,D,7:20,7:31,4
C,D,7:04,7:10,5
C,D,7:14,7:20,6
C,D,7:24,7:30,7
C,B,7:05,7:12,8
B,A,7:12,7:18,8
C,B,7:15,7:22,9
B,A,7:22,7:28,9
D,B,7:05,7:16,10
B,A,7:16,7:21,10
D,B,7:15,7:26,11
B,A,7:26,7:31,11
D,C,7:04,7:10,12
D,C,7:14,7:20,13
D,C,7:24,7:30,14
```

RAPTOR: Example

Query: A@7:00 → D, $K = 2$, no footpaths



current trip: ∅

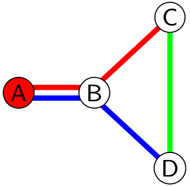
	0	1	2
A	7:00	7:00	
B	∞	∞	
C	∞	∞	
D	∞	∞	

$v_{dep}, v_{arr}, \tau_{dep}, \tau_{arr}$, trip

```
A,B,7:00,7:05,1    C,B,7:05,7:12,8
B,C,7:05,7:12,1    B,A,7:12,7:18,8
A,B,7:10,7:15,2    C,B,7:15,7:22,9
B,C,7:15,7:22,2    B,A,7:22,7:28,9
A,B,7:05,7:10,3    D,B,7:05,7:16,10
B,D,7:10,7:21,3    B,A,7:16,7:21,10
A,B,7:15,7:20,4    D,B,7:15,7:26,11
B,D,7:20,7:31,4    B,A,7:26,7:31,11
C,D,7:04,7:10,5    D,C,7:04,7:10,12
C,D,7:14,7:20,6    D,C,7:14,7:20,13
C,D,7:24,7:30,7    D,C,7:24,7:30,14
```

RAPTOR: Example

Query: A@7:00 → D, $K = 2$, no footpaths



current trip: 1

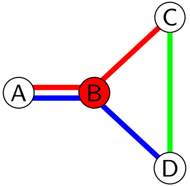
	0	1	2
A	7:00	7:00	
B	∞	∞	
C	∞	∞	
D	∞	∞	

$v_{\text{dep}}, v_{\text{arr}}, \tau_{\text{dep}}, \tau_{\text{arr}}$, trip

```
A,B,7:00,7:05,1
B,C,7:05,7:12,1
A,B,7:10,7:15,2
B,C,7:15,7:22,2
A,B,7:05,7:10,3
B,D,7:10,7:21,3
A,B,7:15,7:20,4
B,D,7:20,7:31,4
C,D,7:04,7:10,5
C,D,7:14,7:20,6
C,D,7:24,7:30,7
C,B,7:05,7:12,8
B,A,7:12,7:18,8
C,B,7:15,7:22,9
B,A,7:22,7:28,9
D,B,7:05,7:16,10
B,A,7:16,7:21,10
D,B,7:15,7:26,11
B,A,7:26,7:31,11
D,C,7:04,7:10,12
D,C,7:14,7:20,13
D,C,7:24,7:30,14
```

RAPTOR: Example

Query: A@7:00 → D, $K = 2$, no footpaths



current trip: 1

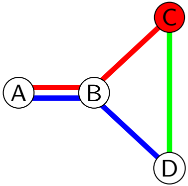
	0	1	2
A	7:00	7:00	
B	∞	7:05	
C	∞	∞	
D	∞	∞	

$v_{\text{dep}}, v_{\text{arr}}, \tau_{\text{dep}}, \tau_{\text{arr}}$, trip

```
A,B,7:00,7:05,1
B,C,7:05,7:12,1
A,B,7:10,7:15,2
B,C,7:15,7:22,2
A,B,7:05,7:10,3
B,D,7:10,7:21,3
A,B,7:15,7:20,4
B,D,7:20,7:31,4
C,D,7:04,7:10,5
C,D,7:14,7:20,6
C,D,7:24,7:30,7
C,B,7:05,7:12,8
B,A,7:12,7:18,8
C,B,7:15,7:22,9
B,A,7:22,7:28,9
D,B,7:05,7:16,10
B,A,7:16,7:21,10
D,B,7:15,7:26,11
B,A,7:26,7:31,11
D,C,7:04,7:10,12
D,C,7:14,7:20,13
D,C,7:24,7:30,14
```

RAPTOR: Example

Query: A@7:00 → D, $K = 2$, no footpaths



current trip: 1

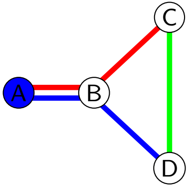
	0	1	2
A	7:00	7:00	
B	∞	7:05	
C	∞	7:12	
D	∞	∞	

$v_{dep}, v_{arr}, \tau_{dep}, \tau_{arr}$, trip

```
A,B,7:00,7:05,1
B,C,7:05,7:12,1
A,B,7:10,7:15,2
B,C,7:15,7:22,2
A,B,7:05,7:10,3
B,D,7:10,7:21,3
A,B,7:15,7:20,4
B,D,7:20,7:31,4
C,D,7:04,7:10,5
C,D,7:14,7:20,6
C,D,7:24,7:30,7
C,B,7:05,7:12,8
B,A,7:12,7:18,8
C,B,7:15,7:22,9
B,A,7:22,7:28,9
D,B,7:05,7:16,10
B,A,7:16,7:21,10
D,B,7:15,7:26,11
B,A,7:26,7:31,11
D,C,7:04,7:10,12
D,C,7:14,7:20,13
D,C,7:24,7:30,14
```

RAPTOR: Example

Query: A@7:00 → D, $K = 2$, no footpaths



current trip: 3

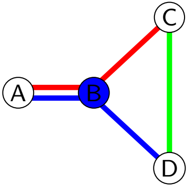
	0	1	2
A	7:00	7:00	
B	∞	7:05	
C	∞	7:12	
D	∞	∞	

$v_{dep}, v_{arr}, \tau_{dep}, \tau_{arr}$, trip

```
A,B,7:00,7:05,1
B,C,7:05,7:12,1
A,B,7:10,7:15,2
B,C,7:15,7:22,2
A,B,7:05,7:10,3
B,D,7:10,7:21,3
A,B,7:15,7:20,4
B,D,7:20,7:31,4
C,D,7:04,7:10,5
C,D,7:14,7:20,6
C,D,7:24,7:30,7
C,B,7:05,7:12,8
B,A,7:12,7:18,8
C,B,7:15,7:22,9
B,A,7:22,7:28,9
D,B,7:05,7:16,10
B,A,7:16,7:21,10
D,B,7:15,7:26,11
B,A,7:26,7:31,11
D,C,7:04,7:10,12
D,C,7:14,7:20,13
D,C,7:24,7:30,14
```

RAPTOR: Example

Query: A@7:00 → D, $K = 2$, no footpaths



current trip: 3

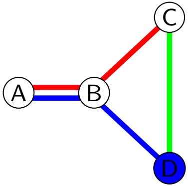
	0	1	2
A	7:00	7:00	
B	∞	7:05	
C	∞	7:12	
D	∞	∞	

$v_{\text{dep}}, v_{\text{arr}}, \tau_{\text{dep}}, \tau_{\text{arr}}$, trip

```
A,B,7:00,7:05,1
B,C,7:05,7:12,1
A,B,7:10,7:15,2
B,C,7:15,7:22,2
A,B,7:05,7:10,3
B,D,7:10,7:21,3
A,B,7:15,7:20,4
B,D,7:20,7:31,4
C,D,7:04,7:10,5
C,D,7:14,7:20,6
C,D,7:24,7:30,7
C,B,7:05,7:12,8
B,A,7:12,7:18,8
C,B,7:15,7:22,9
B,A,7:22,7:28,9
D,B,7:05,7:16,10
B,A,7:16,7:21,10
D,B,7:15,7:26,11
B,A,7:26,7:31,11
D,C,7:04,7:10,12
D,C,7:14,7:20,13
D,C,7:24,7:30,14
```

RAPTOR: Example

Query: A@7:00 → D, $K = 2$, no footpaths



current trip: 3

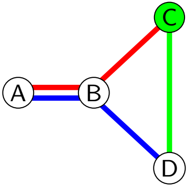
	0	1	2
A	7:00	7:00	
B	∞	7:05	
C	∞	7:12	
D	∞	7:21	

$v_{\text{dep}}, v_{\text{arr}}, \tau_{\text{dep}}, \tau_{\text{arr}}$, trip

```
A,B,7:00,7:05,1     C,B,7:05,7:12,8
B,C,7:05,7:12,1     B,A,7:12,7:18,8
A,B,7:10,7:15,2     C,B,7:15,7:22,9
B,C,7:15,7:22,2     B,A,7:22,7:28,9
A,B,7:05,7:10,3     D,B,7:05,7:16,10
B,D,7:10,7:21,3     B,A,7:16,7:21,10
A,B,7:15,7:20,4     D,B,7:15,7:26,11
B,D,7:20,7:31,4     B,A,7:26,7:31,11
C,D,7:04,7:10,5     D,C,7:04,7:10,12
C,D,7:14,7:20,6     D,C,7:14,7:20,13
C,D,7:24,7:30,7     D,C,7:24,7:30,14
```

RAPTOR: Example

Query: A@7:00 → D, $K = 2$, no footpaths



current trip: ∅

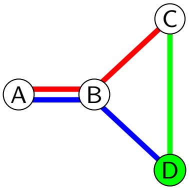
	0	1	2
A	7:00	7:00	
B	∞	7:05	
C	∞	7:12	
D	∞	7:21	

$v_{dep}, v_{arr}, \tau_{dep}, \tau_{arr}$, trip

```
A,B,7:00,7:05,1
B,C,7:05,7:12,1
A,B,7:10,7:15,2
B,C,7:15,7:22,2
A,B,7:05,7:10,3
B,D,7:10,7:21,3
A,B,7:15,7:20,4
B,D,7:20,7:31,4
C,D,7:04,7:10,5
C,D,7:14,7:20,6
C,D,7:24,7:30,7
C,B,7:05,7:12,8
B,A,7:12,7:18,8
C,B,7:15,7:22,9
B,A,7:22,7:28,9
D,B,7:05,7:16,10
B,A,7:16,7:21,10
D,B,7:15,7:26,11
B,A,7:26,7:31,11
D,C,7:04,7:10,12
D,C,7:14,7:20,13
D,C,7:24,7:30,14
```

RAPTOR: Example

Query: A@7:00 → D, $K = 2$, no footpaths



current trip: ∅

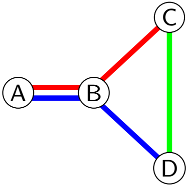
	0	1	2
A	7:00	7:00	
B	∞	7:05	
C	∞	7:12	
D	∞	7:21	

$v_{dep}, v_{arr}, \tau_{dep}, \tau_{arr}$, trip

```
A,B,7:00,7:05,1
B,C,7:05,7:12,1
A,B,7:10,7:15,2
B,C,7:15,7:22,2
A,B,7:05,7:10,3
B,D,7:10,7:21,3
A,B,7:15,7:20,4
B,D,7:20,7:31,4
C,D,7:04,7:10,5
C,D,7:14,7:20,6
C,D,7:24,7:30,7
C,B,7:05,7:12,8
B,A,7:12,7:18,8
C,B,7:15,7:22,9
B,A,7:22,7:28,9
D,B,7:05,7:16,10
B,A,7:16,7:21,10
D,B,7:15,7:26,11
B,A,7:26,7:31,11
D,C,7:04,7:10,12
D,C,7:14,7:20,13
D,C,7:24,7:30,14
```

RAPTOR: Example

Query: A@7:00 → D, $K = 2$, no footpaths



current trip: ∅

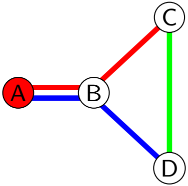
	0	1	2
A	7:00	7:00	7:00
B	∞	7:05	7:05
C	∞	7:12	7:12
D	∞	7:21	7:21

$v_{\text{dep}}, v_{\text{arr}}, \tau_{\text{dep}}, \tau_{\text{arr}}$, trip

```
A,B,7:00,7:05,1
B,C,7:05,7:12,1
A,B,7:10,7:15,2
B,C,7:15,7:22,2
A,B,7:05,7:10,3
B,D,7:10,7:21,3
A,B,7:15,7:20,4
B,D,7:20,7:31,4
C,D,7:04,7:10,5
C,D,7:14,7:20,6
C,D,7:24,7:30,7
C,B,7:05,7:12,8
B,A,7:12,7:18,8
C,B,7:15,7:22,9
B,A,7:22,7:28,9
D,B,7:05,7:16,10
B,A,7:16,7:21,10
D,B,7:15,7:26,11
B,A,7:26,7:31,11
D,C,7:04,7:10,12
D,C,7:14,7:20,13
D,C,7:24,7:30,14
```

RAPTOR: Example

Query: A@7:00 → D, $K = 2$, no footpaths



current trip: 1

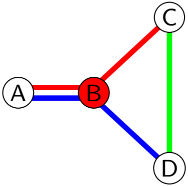
	0	1	2
A	7:00	7:00	7:00
B	∞	7:05	7:05
C	∞	7:12	7:12
D	∞	7:21	7:21

$v_{dep}, v_{arr}, \tau_{dep}, \tau_{arr}$, trip

```
A,B,7:00,7:05,1
B,C,7:05,7:12,1
A,B,7:10,7:15,2
B,C,7:15,7:22,2
A,B,7:05,7:10,3
B,D,7:10,7:21,3
A,B,7:15,7:20,4
B,D,7:20,7:31,4
C,D,7:04,7:10,5
C,D,7:14,7:20,6
C,D,7:24,7:30,7
C,B,7:05,7:12,8
B,A,7:12,7:18,8
C,B,7:15,7:22,9
B,A,7:22,7:28,9
D,B,7:05,7:16,10
B,A,7:16,7:21,10
D,B,7:15,7:26,11
B,A,7:26,7:31,11
D,C,7:04,7:10,12
D,C,7:14,7:20,13
D,C,7:24,7:30,14
```

RAPTOR: Example

Query: A@7:00 → D, $K = 2$, no footpaths



current trip: 1

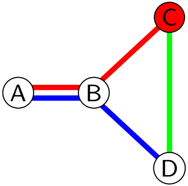
	0	1	2
A	7:00	7:00	7:00
B	∞	7:05	7:05
C	∞	7:12	7:12
D	∞	7:21	7:21

$v_{\text{dep}}, v_{\text{arr}}, \tau_{\text{dep}}, \tau_{\text{arr}}$, trip

```
A,B,7:00,7:05,1
B,C,7:05,7:12,1
A,B,7:10,7:15,2
B,C,7:15,7:22,2
A,B,7:05,7:10,3
B,D,7:10,7:21,3
A,B,7:15,7:20,4
B,D,7:20,7:31,4
C,D,7:04,7:10,5
C,D,7:14,7:20,6
C,D,7:24,7:30,7
C,B,7:05,7:12,8
B,A,7:12,7:18,8
C,B,7:15,7:22,9
B,A,7:22,7:28,9
D,B,7:05,7:16,10
B,A,7:16,7:21,10
D,B,7:15,7:26,11
B,A,7:26,7:31,11
D,C,7:04,7:10,12
D,C,7:14,7:20,13
D,C,7:24,7:30,14
```

RAPTOR: Example

Query: A@7:00 → D, $K = 2$, no footpaths



current trip: 1

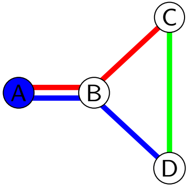
	0	1	2
A	7:00	7:00	7:00
B	∞	7:05	7:05
C	∞	7:12	7:12
D	∞	7:21	7:21

$v_{dep}, v_{arr}, \tau_{dep}, \tau_{arr}$, trip

```
A,B,7:00,7:05,1
B,C,7:05,7:12,1
A,B,7:10,7:15,2
B,C,7:15,7:22,2
A,B,7:05,7:10,3
B,D,7:10,7:21,3
A,B,7:15,7:20,4
B,D,7:20,7:31,4
C,D,7:04,7:10,5
C,D,7:14,7:20,6
C,D,7:24,7:30,7
C,B,7:05,7:12,8
B,A,7:12,7:18,8
C,B,7:15,7:22,9
B,A,7:22,7:28,9
D,B,7:05,7:16,10
B,A,7:16,7:21,10
D,B,7:15,7:26,11
B,A,7:26,7:31,11
D,C,7:04,7:10,12
D,C,7:14,7:20,13
D,C,7:24,7:30,14
```

RAPTOR: Example

Query: A@7:00 → D, $K = 2$, no footpaths



current trip: 3

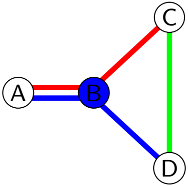
	0	1	2
A	7:00	7:00	7:00
B	∞	7:05	7:05
C	∞	7:12	7:12
D	∞	7:21	7:21

$v_{dep}, v_{arr}, \tau_{dep}, \tau_{arr}$, trip

A,B,7:00,7:05,1	C,B,7:05,7:12,8
B,C,7:05,7:12,1	B,A,7:12,7:18,8
A,B,7:10,7:15,2	C,B,7:15,7:22,9
B,C,7:15,7:22,2	B,A,7:22,7:28,9
A,B,7:05,7:10,3	D,B,7:05,7:16,10
B,D,7:10,7:21,3	B,A,7:16,7:21,10
A,B,7:15,7:20,4	D,B,7:15,7:26,11
B,D,7:20,7:31,4	B,A,7:26,7:31,11
C,D,7:04,7:10,5	D,C,7:04,7:10,12
C,D,7:14,7:20,6	D,C,7:14,7:20,13
C,D,7:24,7:30,7	D,C,7:24,7:30,14

RAPTOR: Example

Query: A@7:00 → D, $K = 2$, no footpaths



current trip: 3

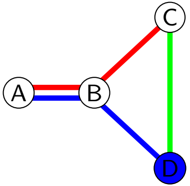
	0	1	2
A	7:00	7:00	7:00
B	∞	7:05	7:05
C	∞	7:12	7:12
D	∞	7:21	7:21

$v_{\text{dep}}, v_{\text{arr}}, \tau_{\text{dep}}, \tau_{\text{arr}}$, trip

```
A,B,7:00,7:05,1     C,B,7:05,7:12,8
B,C,7:05,7:12,1     B,A,7:12,7:18,8
A,B,7:10,7:15,2     C,B,7:15,7:22,9
B,C,7:15,7:22,2     B,A,7:22,7:28,9
A,B,7:05,7:10,3     D,B,7:05,7:16,10
B,D,7:10,7:21,3     B,A,7:16,7:21,10
A,B,7:15,7:20,4     D,B,7:15,7:26,11
B,D,7:20,7:31,4     B,A,7:26,7:31,11
C,D,7:04,7:10,5     D,C,7:04,7:10,12
C,D,7:14,7:20,6     D,C,7:14,7:20,13
C,D,7:24,7:30,7     D,C,7:24,7:30,14
```

RAPTOR: Example

Query: A@7:00 $\rightarrow$ D, $K = 2$, no footpaths



current trip: 3

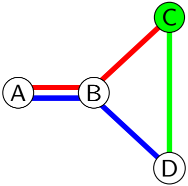
	0	1	2
A	7:00	7:00	7:00
B	∞	7:05	7:05
C	∞	7:12	7:12
D	∞	7:21	7:21

$v_{\text{dep}}, v_{\text{arr}}, \tau_{\text{dep}}, \tau_{\text{arr}},$ trip

```
A,B,7:00,7:05,1
B,C,7:05,7:12,1
A,B,7:10,7:15,2
B,C,7:15,7:22,2
A,B,7:05,7:10,3
B,D,7:10,7:21,3
A,B,7:15,7:20,4
B,D,7:20,7:31,4
C,D,7:04,7:10,5
C,D,7:14,7:20,6
C,D,7:24,7:30,7
C,B,7:05,7:12,8
B,A,7:12,7:18,8
C,B,7:15,7:22,9
B,A,7:22,7:28,9
D,B,7:05,7:16,10
B,A,7:16,7:21,10
D,B,7:15,7:26,11
B,A,7:26,7:31,11
D,C,7:04,7:10,12
D,C,7:14,7:20,13
D,C,7:24,7:30,14
```

RAPTOR: Example

Query: A@7:00 → D, $K = 2$, no footpaths



current trip: 6

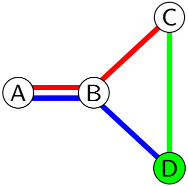
	0	1	2
A	7:00	7:00	7:00
B	∞	7:05	7:05
C	∞	7:12	7:12
D	∞	7:21	7:21

$v_{\text{dep}}, v_{\text{arr}}, \tau_{\text{dep}}, \tau_{\text{arr}}$, trip

```
A,B,7:00,7:05,1
B,C,7:05,7:12,1
A,B,7:10,7:15,2
B,C,7:15,7:22,2
A,B,7:05,7:10,3
B,D,7:10,7:21,3
A,B,7:15,7:20,4
B,D,7:20,7:31,4
C,D,7:04,7:10,5
C,D,7:14,7:20,6
C,D,7:24,7:30,7
C,B,7:05,7:12,8
B,A,7:12,7:18,8
C,B,7:15,7:22,9
B,A,7:22,7:28,9
D,B,7:05,7:16,10
B,A,7:16,7:21,10
D,B,7:15,7:26,11
B,A,7:26,7:31,11
D,C,7:04,7:10,12
D,C,7:14,7:20,13
D,C,7:24,7:30,14
```

RAPTOR: Example

Query: A@7:00 → D, $K = 2$, no footpaths



current trip: 6

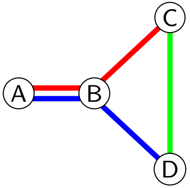
	0	1	2
A	7:00	7:00	7:00
B	∞	7:05	7:05
C	∞	7:12	7:12
D	∞	7:21	7:20

$v_{\text{dep}}, v_{\text{arr}}, \tau_{\text{dep}}, \tau_{\text{arr}},$ trip

```
A,B,7:00,7:05,1
B,C,7:05,7:12,1
A,B,7:10,7:15,2
B,C,7:15,7:22,2
A,B,7:05,7:10,3
B,D,7:10,7:21,3
A,B,7:15,7:20,4
B,D,7:20,7:31,4
C,D,7:04,7:10,5
C,D,7:14,7:20,6
C,D,7:24,7:30,7
C,B,7:05,7:12,8
B,A,7:12,7:18,8
C,B,7:15,7:22,9
B,A,7:22,7:28,9
D,B,7:05,7:16,10
B,A,7:16,7:21,10
D,B,7:15,7:26,11
B,A,7:26,7:31,11
D,C,7:04,7:10,12
D,C,7:14,7:20,13
D,C,7:24,7:30,14
```

RAPTOR: Example

Query: A@7:00 → D, $K = 2$, no footpaths



$v_{\text{dep}}, v_{\text{arr}}, \tau_{\text{dep}}, \tau_{\text{arr}}$, trip

```
A,B,7:00,7:05,1    C,B,7:05,7:12,8
B,C,7:05,7:12,1    B,A,7:12,7:18,8
A,B,7:10,7:15,2    C,B,7:15,7:22,9
B,C,7:15,7:22,2    B,A,7:22,7:28,9
A,B,7:05,7:10,3    D,B,7:05,7:16,10
B,D,7:10,7:21,3    B,A,7:16,7:21,10
A,B,7:15,7:20,4    D,B,7:15,7:26,11
B,D,7:20,7:31,4    B,A,7:26,7:31,11
C,D,7:04,7:10,5    D,C,7:04,7:10,12
C,D,7:14,7:20,6    D,C,7:14,7:20,13
C,D,7:24,7:30,7    D,C,7:24,7:30,14
```

current trip:

	0	1	2
A	7:00	7:00	7:00
B	∞	7:05	7:05
C	∞	7:12	7:12
D	∞	7:21	7:20

⇝ earliest arrival at D: 7:20

(Remark: omitted scanning the lines in the other direction)

RAPTOR: Discussion

- $\texttt{time}(k, v)$ is the earliest arrival time at v using at most k trips.
- $\texttt{path}(k, v)$ contains the corresponding journey as sequence of elementary connections and footpaths.
- In a round, each line L is scanned exactly once.
- When a line is scanned, the algorithm begins with the first stop v on the line where a trip departs after $\texttt{time}(k-1, v)$. The arrival times on the following stops are set following this trip (`curr_trip`). If a stop with an earlier arrival is found, reset `curr_trip` to the earlier trip and continue.
- Footpaths are considered at the end of each round.
- There is no preprocessing, except for the organization of timetable data in arrays.

RAPTOR: Optimization

- The algorithm can be stopped after round k if for all stops v holds $\texttt{time}(k, v) = \texttt{time}(k-1, v)$.
- *Marking:* During round k, it suffices to consider routes containing a stop reached with *exactly* $k-1$ trips. The arrival times at the stops of all other routes have not improved in round $k-1$, hence they can only be improved by *another* route in round k.
  This adds the following to RAPTOR: If $\texttt{time}(k, v)$ changed, then *mark* v. Instead of scanning all routes in round $k+1$, consider pairs (L, v), where L is a route and v is the earliest marked stop on L. Before updating the arrival times, every stop gets unmarked. Mark s at the beginning.
- *Stopping criterion:* In an $s@\tau \to t$ query, there is no need to consider arrival times $\tau_{\text{arr}}(v_j) > \texttt{time}(k, t)$.

RAPTOR: Extensions

Range queries (rRAPTOR)

- Suppose we want to find a set of Pareto-optimal journeys departing at s in a time range R.
- *rRAPTOR Algorithm*:
  - Let T be a list of departure times of trips leaving s within R, sorted in descending order.
  - Run RAPTOR for each $\tau \in T$, but keep `time` and `path` for the next τ.

Multi-criteria problems (McRAPTOR)

- The *McRAPTOR* algorithm stores a set of parwise non-dominating labels for each round k and stop v.
- If fare zones are to be integrated, a potential label could be (arrival time, set of touched fare zones).
- When traversing a route, the labels are updated. Finally, dominated labels are discarded.

Transfer Patterns

Overview

- *Transfer Patterns* are a speedup technique for large public transportation networks (Bast et. al., 2010).
- The key observation is that, independent of time, many optimal journeys from s to t make the same transfers.
- In a huge preprocessing step, for each pair (s, t) of stops some sequences of potentially optimal transfers is computed – these are the actual *transfer patterns*.
- At query time, these transfer patterns are merged into a small *query graph*, where Dijkstra's algorithm is fast enough to solve the shortest path problem in reasonable time.
- Transfer Patterns are used e.g. by Google Transit.

Transfer Patterns: Definition

For a line network $\mathcal{N}$ with a timetable, consider its realistic time-expanded network $\mathcal{E}$ (i.e., with transfer events).

Definition

- The **transfer pattern** of a path in $\mathcal{E}$ is the sequence of the stations in $\mathcal{N}$ corresponding to the first event, each arrival event whose successor is a transfer event, and the last event.
- An **optimal set of transfer patterns** for a pair $(s, t) \in V(\mathcal{N})$ is a set S of transfer patterns such that:
  - for all earliest arrival queries $s@\tau \to t$, there is an optimal set of s-t-journeys whose transfer patterns are contained in S,
  - every element in S is the transfer pattern of an optimal s-t-journey for some earliest arrival query $s@\tau \to t$.

In this context, an s-t-journey for an earliest arrival query $s@\tau \to t$ is a path from the first transfer event at s after τ to some arrival event of τ.

Transfer Patterns: Direct Connections

Consider the following data structures:

- For each line L, sort its trips by first departure and organize them in a table:

Line L1	v_1	v_2	v_3	...
Trip 1	07:12	07:22 07:23	07:44 07:46	...
Trip 2	09:14	09:22 09:23	09:44 09:46	...

- For each station, compute the list of lines incident to it and its position on the line:

Stop v_1	(L1, 1)	(L2, 5)	...
Stop v_2	(L1, 2)	(L3, 6)	...

A *direct connection query* $s@\tau \to t$ (i.e., no transfers allowed), can be answered in the following way:

- Intersect the list of incident lines of s and t.
- For each occurence of s before t on a line, read off the cost of the earliest feasible trip (FIFO), and take the optimal cost among all lines.

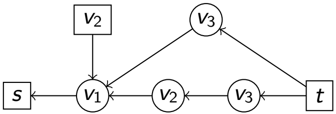
Transfer Patterns: Preprocessing

Preprocessing

For every station s of the line network, compute the transfer patterns (s, t) for all stations t reachable from s.

Details

- Run a shortest-path algorithm from all transfer events of s to compute transfer patterns $(s, v_1, \ldots, v_k, t)$ for all reachable stations t.
- Store the transfer patterns in a DAG with reversed arrows:



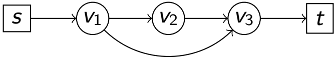
  Each directed path between rectangular vertices corresponds to a transfer pattern. A reachable station occurs as the label of a rectangular vertex and possibly in several circular vertices.
- There is also a multi-criteria version.

Transfer Patterns: Query graph

Query graph

For a query $s@\tau \to t$, the *query graph* $Q_{s,t}$ is constructed as follows:

- Fetch the transfer pattern DAG for s.
- Let L be the set of (distinct) labels of the successors of t in the DAG. Add edges (ℓ, t) to $Q_{s,t}$ for each $\ell \in L$.
- Recursively perform Step 2 for each successor $\neq s$.



Remark

The DAG has reversed arrows because it is easier to look for successors than for predecessors.

Transfer Patterns: Algorithm

Let $s@\tau \to t$ be an earliest arrival query.

Basic Transfer Patterns Algorithm

Preprocessing

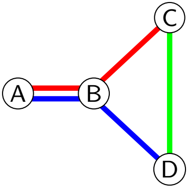
1. Create the data structures for direct connection queries.
2. Compute the transfer patterns DAG for every station s of the line network.

Query

1. Build the query graph $Q_{s,t}$.
2. Run the time-dependent Dijkstra algorithm on $Q_{s,t}$ using the direct connection data structures.

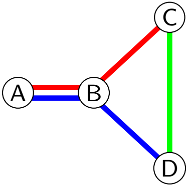
Transfer Patterns: Example

Query: A@7:00 → D, no footpaths



Transfer Patterns: Example

Query: A@7:00 → D, no footpaths



Direct connection tables (backward direction omitted):

red	A	B	C
1	7:00	7:05 7:05	7:12
2	7:10	7:15 7:15	7:22

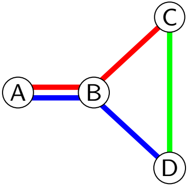
green	C	D
5	7:04	7:10
6	7:14	7:20
7	7:24	7:30

blue	A	B	D
3	7:05	7:10 7:10	7:21
4	7:15	7:20 7:20	7:31

A	red, 1	blue, 1
B	red, 2	blue, 2
C	red, 3	green, 1
D	blue, 3	green, 2

Transfer Patterns: Example

Query: A@7:00 → D, no footpaths



Direct connection tables (backward direction omitted):

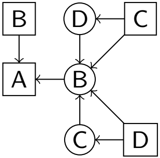
red	A	B	C
1	7:00	7:05 7:05	7:12
2	7:10	7:15 7:15	7:22

green	C	D
5	7:04	7:10
6	7:14	7:20
7	7:24	7:30

blue	A	B	D
3	7:05	7:10 7:10	7:21
4	7:15	7:20 7:20	7:31

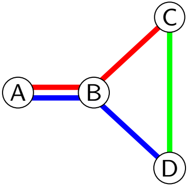
A	red, 1	blue, 1
B	red, 2	blue, 2
C	red, 3	green, 1
D	blue, 3	green, 2

Transfer Pattern DAG for A:



Transfer Patterns: Example

Query: A@7:00 → D, no footpaths



Direct connection tables (backward direction omitted):

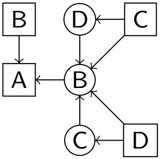
red	A	B	C
1	7:00	7:05 7:05	7:12
2	7:10	7:15 7:15	7:22

green	C	D
5	7:04	7:10
6	7:14	7:20
7	7:24	7:30

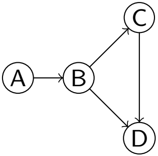
blue	A	B	D
3	7:05	7:10 7:10	7:21
4	7:15	7:20 7:20	7:31

A	red, 1	blue, 1
B	red, 2	blue, 2
C	red, 3	green, 1
D	blue, 3	green, 2

Transfer Pattern DAG for A:

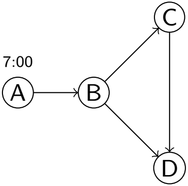


Query graph for (A, D):



Transfer Patterns: Example

Query: A@7:00 → D, no footpaths



Direct connection tables (backward direction omitted):

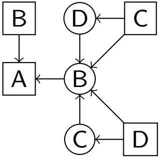
red	A	B	C
1	7:00	7:05 7:05	7:12
2	7:10	7:15 7:15	7:22

green	C	D
5	7:04	7:10
6	7:14	7:20
7	7:24	7:30

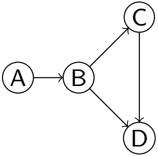
blue	A	B	D
3	7:05	7:10 7:10	7:21
4	7:15	7:20 7:20	7:31

A	red, 1	blue, 1
B	red, 2	blue, 2
C	red, 3	green, 1
D	blue, 3	green, 2

Transfer Pattern DAG for A:

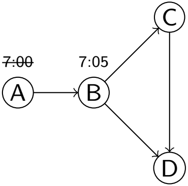


Query graph for (A, D):



Transfer Patterns: Example

Query: A@7:00 → D, no footpaths



Direct connection tables (backward direction omitted):

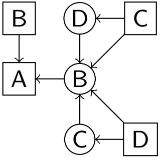
red	A	B	C
1	7:00	7:05 7:05	7:12
2	7:10	7:15 7:15	7:22

green	C	D
5	7:04	7:10
6	7:14	7:20
7	7:24	7:30

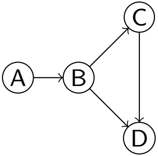
blue	A	B	D
3	7:05	7:10 7:10	7:21
4	7:15	7:20 7:20	7:31

A	red, 1	blue, 1
B	red, 2	blue, 2
C	red, 3	green, 1
D	blue, 3	green, 2

Transfer Pattern DAG for A:

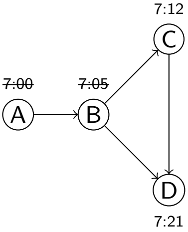


Query graph for (A, D):



Transfer Patterns: Example

Query: A@7:00 → D, no footpaths



Direct connection tables (backward direction omitted):

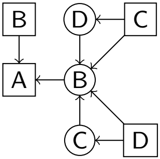
red	A	B	C
1	7:00	7:05 7:05	7:12
2	7:10	7:15 7:15	7:22

green	C	D
5	7:04	7:10
6	7:14	7:20
7	7:24	7:30

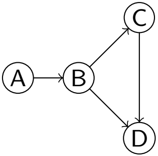
blue	A	B	D
3	7:05	7:10 7:10	7:21
4	7:15	7:20 7:20	7:31

A	red, 1	blue, 1
B	red, 2	blue, 2
C	red, 3	green, 1
D	blue, 3	green, 2

Transfer Pattern DAG for A:

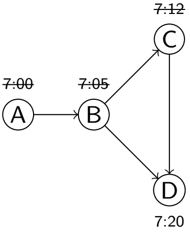


Query graph for (A, D):



Transfer Patterns: Example

Query: A@7:00 → D, no footpaths



Direct connection tables (backward direction omitted):

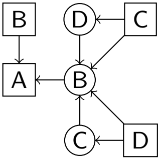
red	A	B	C
1	7:00	7:05 7:05	7:12
2	7:10	7:15 7:15	7:22

green	C	D
5	7:04	7:10
6	7:14	7:20
7	7:24	7:30

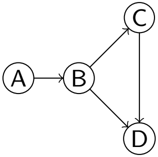
blue	A	B	D
3	7:05	7:10 7:10	7:21
4	7:15	7:20 7:20	7:31

A	red, 1	blue, 1
B	red, 2	blue, 2
C	red, 3	green, 1
D	blue, 3	green, 2

Transfer Pattern DAG for A:

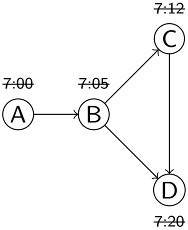


Query graph for (A, D):



Transfer Patterns: Example

Query: A@7:00 → D, no footpaths



Direct connection tables (backward direction omitted):

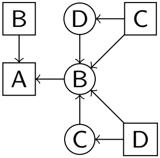
red	A	B		C
1	7:00	7:05	7:05	7:12
2	7:10	7:15	7:15	7:22

green	C	D
5	7:04	7:10
6	7:14	7:20
7	7:24	7:30

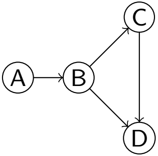
blue	A	B		D
3	7:05	7:10	7:10	7:21
4	7:15	7:20	7:20	7:31

A	red, 1	blue, 1
B	red, 2	blue, 2
C	red, 3	green, 1
D	blue, 3	green, 2

Transfer Pattern DAG for A:



Query graph for (A, D):



⇝ earliest arrival at D: 7:20

Transfer Patterns: Optimization

- Remember that time-expanded networks are huge. This makes the preprocessing for Transfer Patterns a real challenge.
- Solution: Store only transfer patterns to *hubs*.
- Hubs are selected using a heuristic strategy, e.g., by random sampling of earliest arrival queries and taking the stations with the highest number of journeys passing through it.
- The query graph needs to evaluate the transfer patterns from, to and between all relevant hubs for s and t.
- This strategy is still correct and reduces memory usage significantly.
- There are more optimizations, some of them are heuristic.

Routing: Comparison

Earliest arrival query benchmarks (Bast et. al., 2015)

Algorithm	Instance	Elem. conn.	Query time
Time-Expanded Dijkstra	London	$5 \cdot 10^6$	44.8 ms
Time-Dependent Dijkstra	London	$5 \cdot 10^6$	11.0 ms
Connection Scan	London	$5 \cdot 10^6$	2.0 ms
RAPTOR*	London	$5 \cdot 10^6$	5.4 ms
Transfer Patterns**	Germany	$90 \cdot 10^6$	0.4 ms

*RAPTOR: Pareto optimization **Transfer Patterns: 22 d 13 h preprocessing time

Range+Pareto query benchmarks (Bast et. al., 2015)

Algorithm	Instance	Elem. conn.	Query time
Connection Scan	London	$5 \cdot 10^6$	466.0 ms
rRAPTOR	London	$5 \cdot 10^6$	922.0 ms
Transfer Patterns*	Germany	$90 \cdot 10^6$	39.6 ms

*Transfer Patterns: 23 d 14 h preprocessing time

Appendix: GTFS

Google's *General Transit Feed Specification* is a de facto standard for exchanging timetable data of public transportation networks.
It is a zip archive containing several plain text files. Each text file represents a table by *c*omma-*s*eparated *v*alues (csv).

Contents

- *agency.txt* – information about transport companies
- *stops.txt* – list of stops with coordinates
- *routes.txt* – list of routes (lines)
- *trips.txt* – trips for each route
- *stop_times.txt* – departure and arrival times for each trip
- *calendar.txt* – days on which trips run
- several optional files (transfers, calendar exceptions, frequencies, ...)

Appendix: GTFS stops

stops.txt

```
stop_id,stop_name,stop_lat,stop_lon,location_type,parent_station
900000051303,"U Dahlem-Dorf",52.457695,13.290011,1,
070101000015,"U Dahlem-Dorf",52.457695,13.290011,0,900000051303
070101000067,"U Dahlem-Dorf",52.457695,13.290011,0,900000051303
070101000679,"U Dahlem-Dorf",52.457695,13.290011,0,900000051303
070101000843,"U Dahlem-Dorf",52.457695,13.290011,0,900000051303
070201034001,"U Dahlem-Dorf",52.457695,13.290011,0,900000051303
070201034002,"U Dahlem-Dorf",52.457695,13.290011,0,900000051303
```

- Each stop location has a unique ID (`stop_id`), e.g., 070201034001 is the southbound track of subway line U3
- All stop locations belong to a meta-stop (`parent_station`).
- Stops are equipped with WGS84 coordinates (`stop_lat`, `stop_lon`).

Appendix: GTFS trips

routes.txt

```
route_id,agency_id,route_short_name,route_type
17515_400,796,"U3",400
```
← urban railway service route U3 with ID 17515_400

trips.txt

```
route_id,service_id,trip_id,trip_headsign
17515_400,913,74049828,"U Krumme Lanke"
```
← trip 74049828 on southbound U3

stop_times.txt

```
trip_id,arrival_time,departure_time,stop_id,stop_sequence
74049828,3:12:00,3:12:00,070201042103,0
74049828,3:14:00,3:14:00,070201013101,1
...
74049828,3:27:00,3:27:00,070201033901,9
74049828,3:28:30,3:28:30,070201034001,10
74049828,3:30:30,3:30:30,070201034101,11
74049828,3:32:00,3:32:00,070201034201,12
74049828,3:34:00,3:34:00,070201034301,13
74049828,3:36:00,3:36:00,070201034402,14
```
← stop at U Dahlem-Dorf

Appendix: GTFS resources

- The GTFS documentation is available at `https://developers.google.com/transit/gtfs`.
- Several transport associations provide their timetable as open data in GTFS format, e.g., VBB (Berlin-Brandenburg), VRN (Rhein-Neckar).