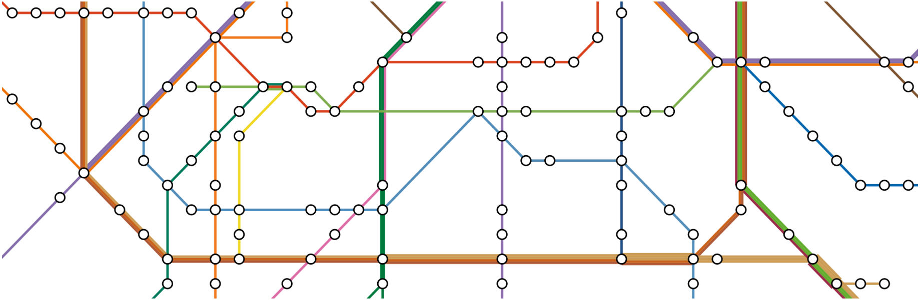
Mathematical Aspects of Public Transportation Networks

Niels Lindner



April 30, 2018

Chapter 1

S-Bahn Challenge

§1.5 Public Transportation Networks

Line Networks

Definition

A **line network** is a graph G together with a **line cover** $\mathcal{L}$, i.e., $\mathcal{L}$ is a set of walks in G such that $E(G) = \bigcup_{L \in \mathcal{L}} E(L)$.

Line Networks and Event-Activity Networks

Remarks

- Depending on the application, line networks may be undirected or directed.
- The vertices of a line network are *stations* or *stops*.
- The elements of $\mathcal{L}$ are *lines* or *routes*.
- The two directions of a classical path-shaped line can be modeled by two separated walks or by a closed walk.

Definition

An **event-activity network (EAN)** is a directed graph $\mathcal{E}$ whose vertices are called *events* and whose edges are called *activities*.

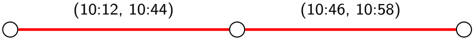
Timetables for Line Networks

Definition

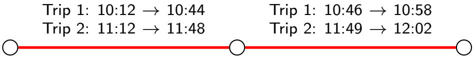
Let $\mathcal{N} = (G, \mathcal{L})$ be a line network.

- A **trip** of a line $L = (e_1, \dots, e_k) \in \mathcal{L}$ is a pair $(\tau_{\text{dep}}, \tau_{\text{arr}})$ of maps $\tau_{\text{dep}}, \tau_{\text{arr}} : \{1, \dots, k\} \to \mathbb{R}$ such that

$$\begin{aligned} \tau_{\text{dep}}(i) &\leq \tau_{\text{arr}}(i), && i = 1, \dots, k \\ \tau_{\text{arr}}(i) &\leq \tau_{\text{dep}}(i+1), && i = 1, \dots, k-1. \end{aligned}$$



- A **schedule** for L is a collection of trips of L.



- A **timetable** for $\mathcal{N}$ assigns a schedule to each line.

Time Expansion

Definition

Consider a timetable $\mathcal{T}$ for a line network $\mathcal{N}$. The **time expansion** of $\mathcal{N}$ w.r.t. $\mathcal{T}$ is the event-activity network $\mathcal{E}$, together with the length function $\ell : E(\mathcal{E}) \to \mathbb{R}_{\geq 0}$, constructed as follows:

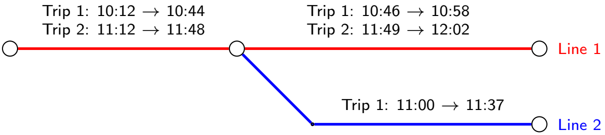
1. For each trip $\tau = (\tau_{\text{dep}}, \tau_{\text{arr}})$ of a line $L = (e_1, \ldots, e_k)$ in $\mathcal{N}$:
   - Add *departure events* (L, τ, i, dep) for $i = 1, \ldots, k$.
   - Add *arrival events* (L, τ, i, arr) for $i = 1, \ldots, k$.
   - Add *driving activities* $(L, \tau, i, \text{dep}) \to (L, \tau, i, \text{arr})$ with length $\tau_{\text{arr}}(i) - \tau_{\text{dep}}(i)$, $i = 1, \ldots, k$.
   - Add *waiting activities* $(L, \tau, i, \text{arr}) \to (L, \tau, i+1, \text{dep})$ with length $\tau_{\text{dep}}(i+1) - \tau_{\text{arr}}(i)$, $i = 1, \ldots, k-1$.
2. Add a *transfer activity* $(L, \tau, i, \text{arr}) \to (L', \tau', i', \text{dep})$ with length $\tau'_{\text{dep}}(i') - \tau_{\text{arr}}(i)$ for each pair of trips (τ, τ') associated to a pair of lines (L, L') whenever:
   - $\tau'_{\text{dep}}(i') - \tau_{\text{arr}}(i) \geq 0$, and
   - the $(i+1)$-st vertex of L and the i'-th vertex of L' coincide in $\mathcal{N}$,
   - (L, τ, i, arr) and $(L', \tau', i', \text{dep})$ are not connected by a waiting activity.

Time Expansion

Remarks

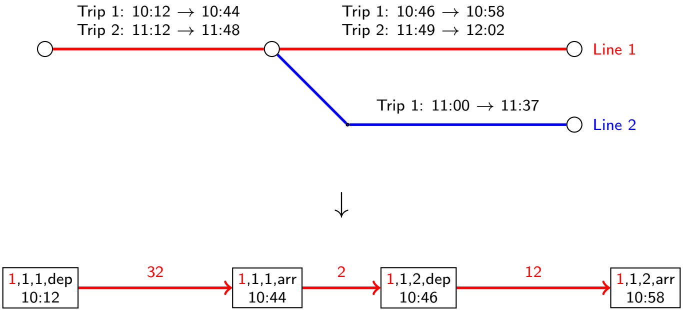
- Trips correspond to certain disjoint directed paths in the EAN.
- The EAN is bipartite, as there are no departure-departure and no arrival-arrival activities.
- No activity goes "backward in time": Circuits can only have length 0.
- The number of driving and waiting activities is linear in the number of trips, whereas the number of transfer activities is quadratic.
- A transfer activity between two trips of a line at one of its endpoints is called a *turnaround activity*.
- Often there is no point in a transfer between trips of parallel lines, and the corresponding transfer activities can be removed.
- Sometimes we want to establish a minimum transfer time, and hence only add transfer activities where $\tau'_{\mathrm{dep}}(i') - \tau_{\mathrm{arr}}(i)$ is large enough.
- Footpath information can also be included using transfer activities.

Time Expansion: Example

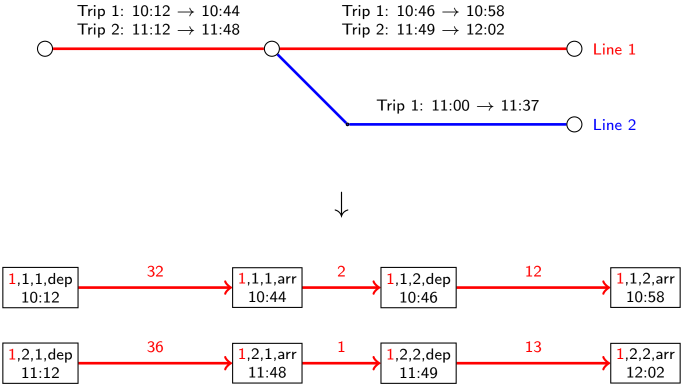


$\downarrow$

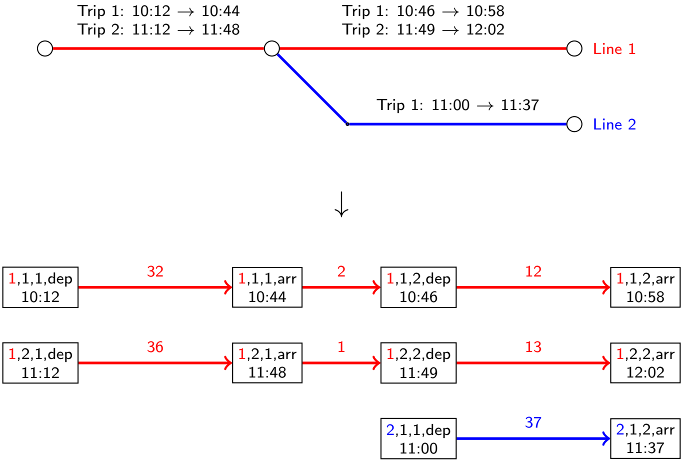
Time Expansion: Example



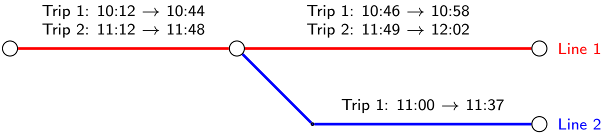
Time Expansion: Example



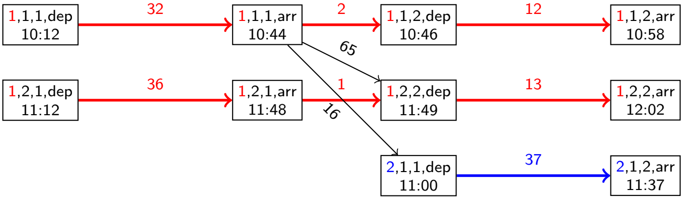
Time Expansion: Example



Time Expansion: Example



$\downarrow$



Timetables for EAN

Definition

A timetable for an EAN $\mathcal{E}$ with length function ℓ is a map $\pi : V(\mathcal{E}) \to \mathbb{R}$ such that

$$\forall\,(v, w) \in E(\mathcal{E}) : \quad \pi(w) - \pi(v) = \ell(v, w).$$

Remarks

- By construction, a timetable of a line network yields an equivalent timetable on the time expansion.
- The timetable for the example on the previous slide is the time written in the vertex labels.
- In other terms: A timetable is a potential for $\mathcal{E}$.

Periodic Timetables

Definition

- A **periodic timetable** for a line network $\mathcal{N}$ assigns to each line a collection of **periodic trips**, i.e., triples $(\tau_{\text{dep}}, \tau_{\text{arr}}, f)$, where τ_{dep}, τ_{arr} are defined as before and $f \in \mathbb{N}$ is a **frequency**.
- A **periodic timetable** with period time $T \in \mathbb{N}$ for an EAN $\mathcal{E}$ with length function ℓ is a map $\pi : V(\mathcal{E}) \to [0, T)$ such that

$$\forall\, (v, w) \in E(\mathcal{E}) : \quad \pi(w) - \pi(v) \equiv \ell(v, w) \mod T.$$

Intuition

- A periodic trip $(\tau_{\text{dep}}, \tau_{\text{arr}}, f)$ corresponds to aperiodic trips $(\tau_{\text{dep}} + i \cdot f, \tau_{\text{arr}} + i \cdot f)$, $i \in \mathbb{Z}$.
- The period time of a periodic time expansion should be an integer multiple of all trip frequencies.

Periodic Time Expansion

Definition

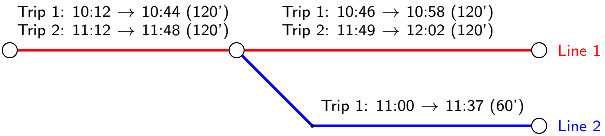
Given a line network with a periodic timetable, its **periodic time expansion** is an EAN, together with a non-negative length function ℓ and a period time $T \in \mathbb{N}$, constructed as follows:

- T is the least common multiple of all trip frequencies.
- For each periodic trip with frequency f, we add T/f directed paths for the pairs $(\tau_{\mathsf{dep}} + i \cdot f, \tau_{\mathsf{arr}} + i \cdot f)$, $i = 0, \ldots, T/f - 1$, as before.
- Connect *any two* arrival and departure events with the same underlying station of the line network by a transfer activity (if there is no waiting activity). Add a suitable integer multiple of T so that the length lies in $[0, T)$.

Remark

Taking departure and arrival times modulo T yields a periodic timetable for the EAN.

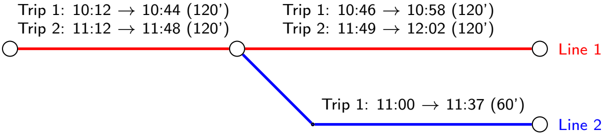
Periodic Time Expansion: Example



$$\downarrow$$

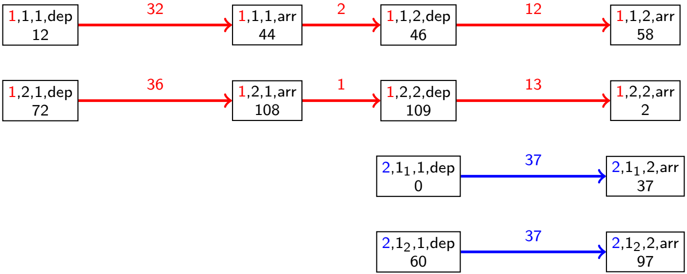
$$T = 120$$

Periodic Time Expansion: Example

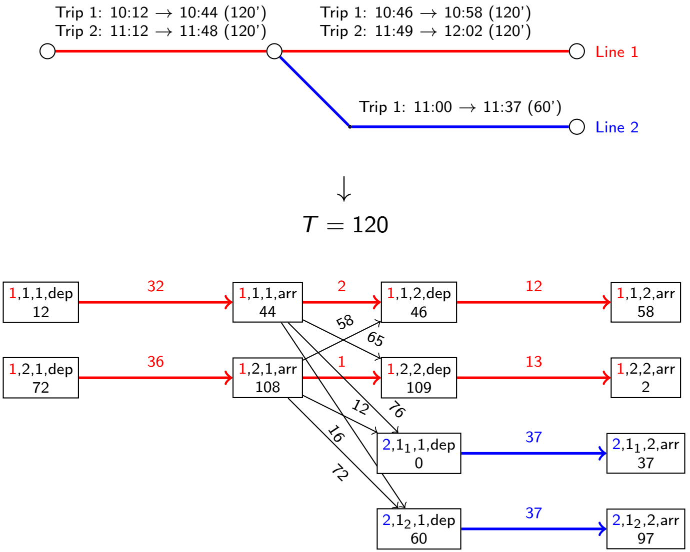


$$\downarrow$$

$$T = 120$$



Periodic Time Expansion: Example



Periodic Time Expansion

Remarks

- The EAN is still bipartite.
- However, the network may now contain circuits of positive length.
- Minimum transfer time analogue: Add a penalty of some multiple of T to transfers that are too short.
- Travel times along driving activities should not be reduced modulo T.
- The periodic time expansion contains many transfer activities: If a station of the line network has a arrivals and d departures, there will be $a \cdot d$ transfer (or waiting) activities. The transfer and waiting activities at such a station form a complete bipartite graph $K_{a,d}$.
- However, for networks with many trips but short frequencies, there are much less events than in an aperiodic time expansion.

S-Bahn Challenge Revisited

Let $(\mathcal{E}, \ell, T)$ be a periodic time expansion of a line network $(G, \mathcal{L})$.

S-Bahn Challenge 1

Define for $v \in V(G)$ the set V_v of all arrival and departure events in $\mathcal{E}$ associated to v.

Problem: Solve the GATSP on $(\mathcal{E}, \ell, \{V_v \mid v \in V(G)\})$.

S-Bahn Challenge 2

Define for $e \in E(G)$ the set E_e of all driving activities in $\mathcal{E}$ coming from e, i.e., the set of all $(L, \tau, i, \text{dep}) \to (L, \tau, i+1, \text{arr}) \in E(\mathcal{E})$ where τ is a trip of a line $L \in \mathcal{L}$ that has e as its i-th edge.

Problem: Solve the GDRPP on $(\mathcal{E}, \ell, \{E_e \mid e \in E(G)\})$.

Exercise

How to compute walks with start $\neq$ end?

S-Bahn Challenge Revisited

Data of graphs involved in solving the GDRPP S-Bahn Challenge:

Problem		# vertices	# edges	opt. tour
CPP	line network	36	45	754
GDRPP	periodic expansion	530	3 554	940
GATSP	splitting edges	795	3 819	940
ATSP	Noon/Bean	265	69 960	381 685
TSP	Jonker/Volgenant	530	140 185	2 673 935

- The GDRPP has 45 clusters containing 265 driving activities.
- The optimal closed walk in the GDRPP graph takes 940 minutes.
- The optimal non-closed walk takes 839 minutes.
- Computing a TSP tour which is relatively close to the optimum is not sufficient: A TSP tour whose value is 1.0001 times the optimum leads to a GDRPP tour which is more than 4 hours longer.

Chapter 2

Shortest Routes in Public Transportation Networks

§2.1 Overview

Basic Problems

Consider a line network $\mathcal{N}$ with a timetable.

Definition

Let s and t be stops in $\mathcal{N}$.

- The **earliest arrival problem** asks for a journey departing from s no earlier than a given departure time τ and arriving at t as early as possible. Short notation: $s@\tau \to t$.
- The **latest departure problem** asks for a journey arriving at t no later than a given arrival time τ and departing at s as late as possible.
- The **profile** or **range earliest arrival problem** asks for a set of journeys departing from s within a specified *range* and arriving at t as early as possible.

Remark

The latest departure problem can be transformed into an earliest arrival problem by going backwards in time.

Challenges: Time

Example (BVG – Berliner Verkehrsbetriebe)

BVG had 1 064 million passengers in 2017. *Fahrinfo*, the trip planner of BVG, received 332.8 million requests. This is an average of approx. 633 queries per minute.

- Therefore, shortest route algorithms need to have a very short running time.
- Usually, the algorithms are divided into a *preprocessing phase* and a *query phase*. This trade-off enables query times of at most a few milliseconds, whereas preprocessing may take days.
- Asymptotic complexity like Dijkstra's $\mathcal{O}(|E| + |V| \log |V|)$ is not suitable to measure exact query times.

Challenges: Space

Example (24 hours of VBB)

Building the time expansion for a normal Tuesday of the Berlin-Brandenburg area produces the following (numbers are rounded):

- 2.4 million events (from 12 000 stops)
- 1.2 million driving activities (from 58 000 trips)
- 1.1 million waiting activities
- 78.8 million transfer activities
- $>$ 32 GB memory usage
  (naive python/networkx implementation – this has a big overhead)

Conclusion

Time expansions are large graphs. However, they are still sparse: the complete digraph on $2.4 \cdot 10^6$ vertices has $\approx 5.76 \cdot 10^{12}$ edges.

Challenges: More

Models

- A good model is crucial for performance – in both speed and space.
- Although a graph model seems to be natural, there might be better data structures.

Comparison to road networks

Unlike road networks, . . .

- public transportation networks are inherently *time-dependent*.
- public transportation networks have a poor structure: Shortest routes in road networks “converge” to highways – this is not the case for transportation networks within a city.

Optimization criteria

Usually, finding a journey solving the earliest arrival problem does not suffice.

More criteria

- minimize the number of transfers
- find the cheapest route
- find a robust route (delays)
- find a generic route that works for most departure times (guidebook routing)

Multi-criteria optimization

Search for all journeys that are *Pareto-optimal*, i.e., journeys where a single criterion cannot be improved without worsening another criterion.
Caveat: There might be exponentially many Pareto-optimal journeys.

Chapter 2

Shortest Routes in Public Transportation Networks

§2.2 Graph Methods

Time-Expanded Dijkstra 1

The easiest approach to solve an earliest arrival query $s@\tau \to t$ is:

Time-Expanded Dijkstra Algorithm – Version 1

Preprocessing

1. Compute the time expansion $\mathcal{E}$ and its timetable π for a sufficiently long time.

Query

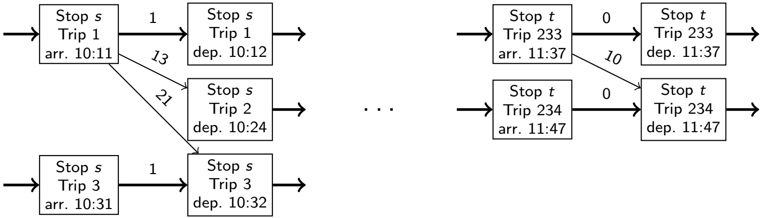
1. Add a *start* vertex to $\mathcal{E}$ and add activities of length 0 to all departure events of s with departure time $\geq \tau$.
2. Invoke Dijkstra's algorithm with *start* as source. Stop when the first arrival event of t is labeled permanently. Return the result.

Drawbacks

- We have to insert the start vertex at query time.
- Dijkstra tends to visit a lot of vertices – there are way too many transfer activities.

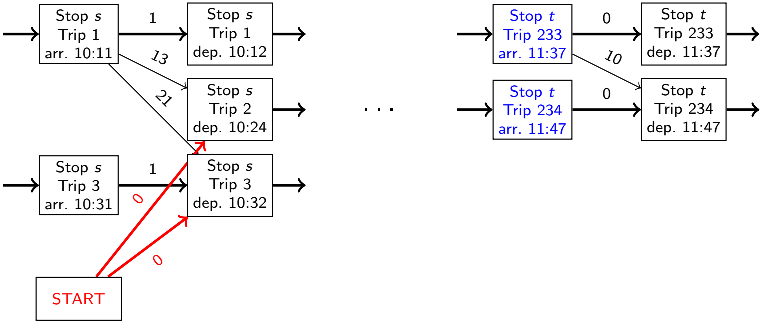
Time-Expanded Dijkstra 1

Query: s@10:15 $\rightarrow t$
full time-expanded graph



Time-Expanded Dijkstra 1

Query: s@10:15 $\to$ t
full time-expanded graph
Dijkstra from START to any arrival event of t



Time-Expanded Dijkstra 2

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

Time-Expanded Dijkstra Algorithm – Version 2

Preprocessing

1. Compute the time expansion $\mathcal{E}$ and its timetable π *without transfer and waiting activities* for a sufficiently long time.
2. For each stop, let $v_1, \dots, v_k$ be its events in ascending order w.r.t. the timetable π. Introduce activities (v_i, v_{i+1}) with length $\pi(v_{i+1}) - \pi(v_i)$ for $i = 1, \dots, k-1$.

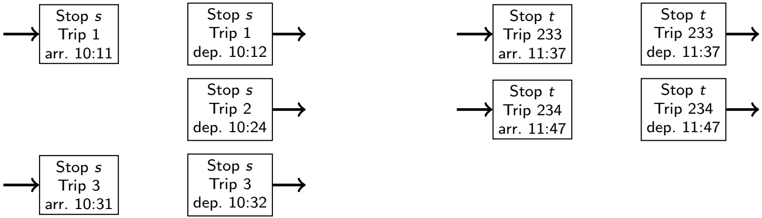
Query

1. Invoke Dijkstra's algorithm, the source being the first departure event v of s with $\pi(v) \geq \tau$. Stop when the first arrival event of t is labeled permanently. Return the result.

Time-Expanded Dijkstra 2

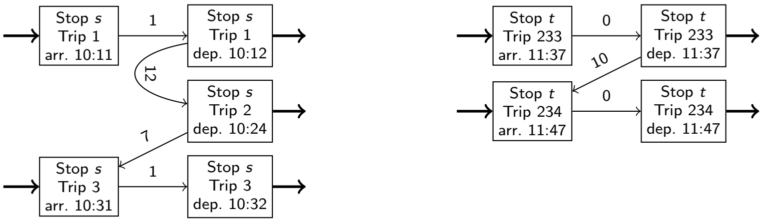
Query: $s@10{:}15 \to t$

time-expanded graph without transfer and waiting activities



Time-Expanded Dijkstra 2

Query: s@10:15 $\rightarrow t$
time-expanded graph without transfer and waiting activities
new edges inside stops



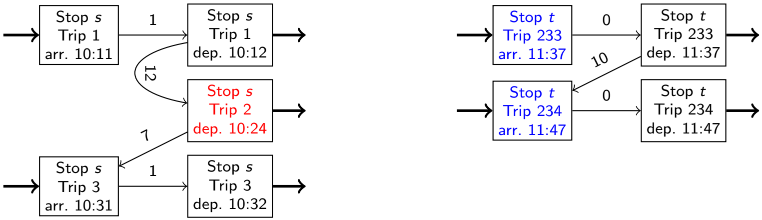
Time-Expanded Dijkstra 2

Query: s@10:15 $\to t$

time-expanded graph without transfer and waiting activities

new edges inside stops

Dijkstra from first departure event of s after τ to any arrival event of t



Time-Expanded Dijkstra 2

Observation

Version 2 uses a linear amount of transfer activities – but all transfer information has gone.

Correction of Version 2 $\rightarrow$ Version 3

- If each stop in the line network has a minimum change time, we can incorporate it by using *transfer events*.
- This is sometimes called the *realistic* time-expanded graph.
- Adding transfer activities back in enables variable change times as well.

Time-Expanded Dijkstra 3

Time-Expanded Dijkstra Algorithm – Version 3

Preprocessing

1. Compute the time expansion $\mathcal{E}$ and its timetable π *without transfer, but with waiting activities* for a sufficiently long time.
2. For each stop:
   - Let $\tau_{\min}$ be the minimum change time.
   - For each dep. w add a *transfer event* x with $\pi(x) := \pi(w) - \tau_{\min}$.
   - Let $x_1, \dots, x_k$ be the transfer events of the stop in ascending order w.r.t. π. Introduce activities (x_i, x_{i+1}) with length $\pi(x_{i+1}) - \pi(x_i)$ for $i = 1, \dots, k-1$.
   - For each arrival event v, add an activity (v, x) of length $\pi(x) - \pi(v)$, where x is the first transfer event with $\pi(x) \geq \pi(v)$.

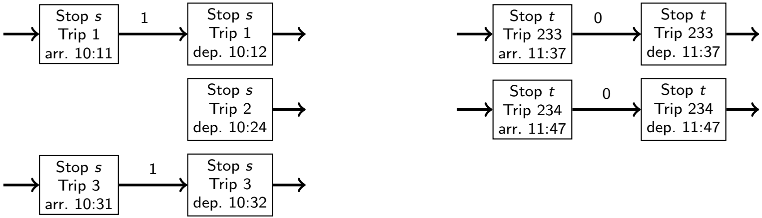
Query

1. Invoke Dijkstra's algorithm, the source being the first transfer event x of s with $\pi(x) \geq \tau$. Stop when the first arrival event of t is labeled permanently. Return the result.

Time-Expanded Dijkstra 3

Query: s@10:15 $\rightarrow t$

time-expanded graph without transfer activities

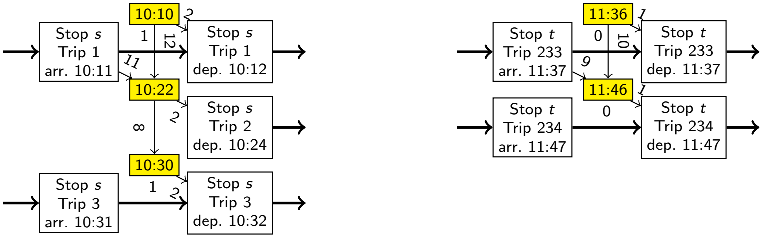


Time-Expanded Dijkstra 3

Query: s@10:15 $\rightarrow$ t

time-expanded graph without transfer activities

transfer vertices



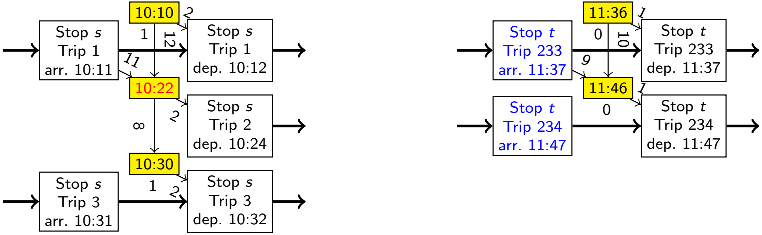
Time-Expanded Dijkstra 3

Query: s@10:15 $\rightarrow t$

time-expanded graph without transfer activities

transfer vertices

Dijkstra from first transfer vertex of s after τ to any arrival event of t



Time-Expanded Dijkstra 4-

Observation

The source vertex for Dijkstra is now a transfer event, the target vertex is an arrival event. Therefore we can contract departure events. This reduces the number of vertices by a third.

Further speed-ups

- A^* search, using geographical distance divided by top speed as heuristic
- bidirectional search
- road network techniques: landmarks, geometric containers, arc flags, contraction hierarchies, ...

Time-Dependent Dijkstra

Idea

Since time expansions are large, it could be more efficient not to expand. The length of the activities then has to be computed at query time.

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

Time-Dependent Dijkstra Algorithm

Preprocessing

1. Construct a graph G as follows: Take all stops from the line network. Add a directed edge (v, w) whenever there is a trip using (v, w).
2. Label each edge (v, w) with a time function $f_{(v,w)}$ such that for any departure time τ_v at v, $f_{(v,w)}(\tau_v)$ is the earliest arrival time at w.

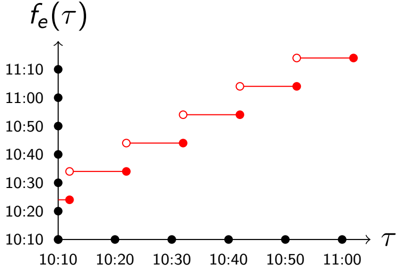
Query

1. Run Dijkstra's algorithm on pairs (v, τ_v), the queue being initialized with (s, τ). Stop if (t, τ_t) is permanently labeled for some time τ_t. Return the result.

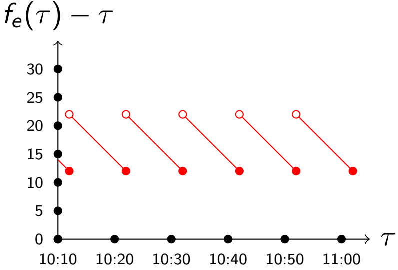
Time-Dependent Dijkstra

Example

Suppose we have trips 10:12 → 10:24, 10:22 → 10:34, . . . repeating every 10 minutes on an edge e of the line network.
The corresponding time and length functions are piecewise linear:



time function f_e



length function $f_e - \tau$

Time-Dependent Dijkstra

Time-Dependent Dijkstra Algorithm – Details

1. $queue := [(s, \tau)]$, for $v \in V(G)$:
   $$time(v) := \begin{cases} \tau & \text{if } v = s, \\ \infty & \text{otherwise.} \end{cases}, \; visited(v) := false, \; path(v) := [v].$$
2. While $queue \neq \emptyset$:
   - Pop minimal element (u, τ_u) of *queue* w.r.t. second entry
   - $visited(u) := true$
   - If $u = t$: break
   - For all successors of v of u with $visited(v) = false$:
     - $\tau_v := f_{(u,v)}(time(u))$.
     - If $\tau_v < time(v)$: Insert (v, τ_v) into *queue*, remove $(v, time(v))$ if $time(v) \neq \infty$, and set $time(v) := \tau_v$, $path(v) := path(u) + [v]$.
3. Return $(path(t), time(t))$.

Time-Dependent Dijkstra

Correctness

The algorithm is correct as long as the FIFO principle holds: Vehicles on the same edge in the line network are not allowed to overtake each other.

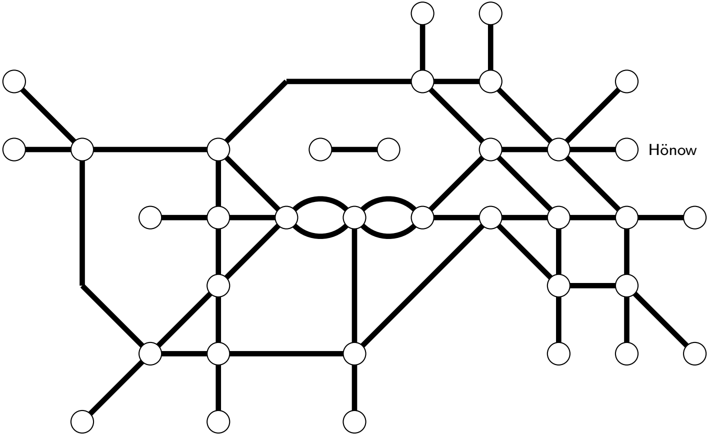
Adjustments

- One can also keep track of the trips.
- The function f does not need to be computed explicitly: During preprocessing, create a sorted list of all trips on all edges. Computing $f_{(u,v)}$ at time $time(u)$ reduces to find the first trip departing after $time(u)$ on (u, v) (binary search).
- Minimum change times (*realistic* time-dependent graph): At each stop v, introduce *route vertices* for each line stopping at v. There are three types of directed edges:
  - stop to route vertex: length = minimum transfer time
  - route vertex to route vertex: time function f as before
  - route vertex to stop: length = 0

Time-Dependent Dijkstra

Example (shortest path tree)

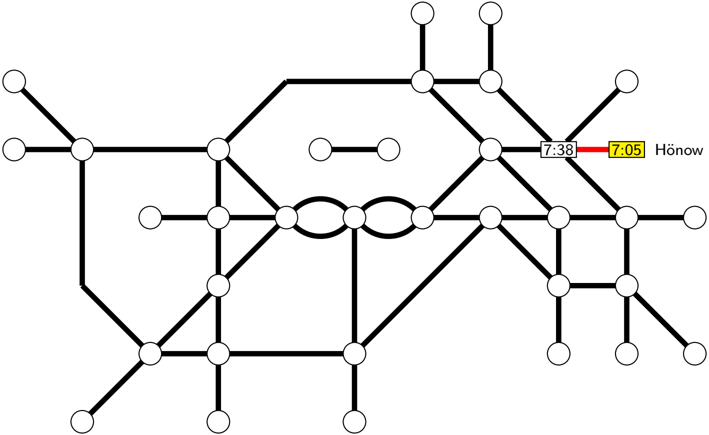
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

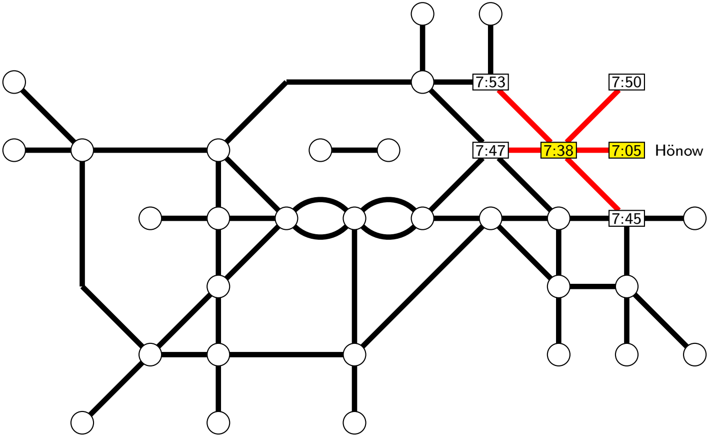
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

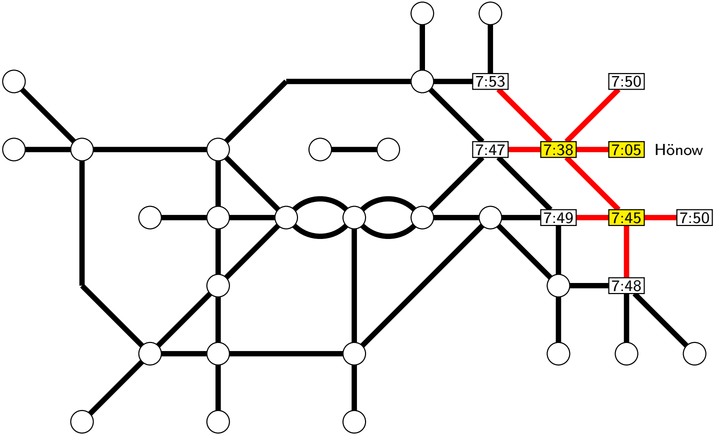
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

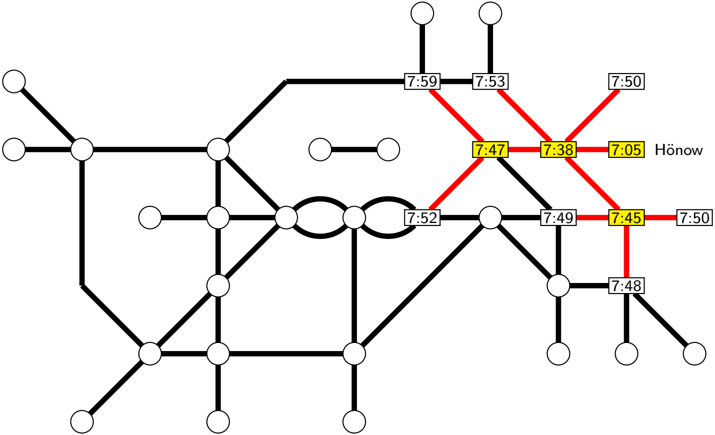
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

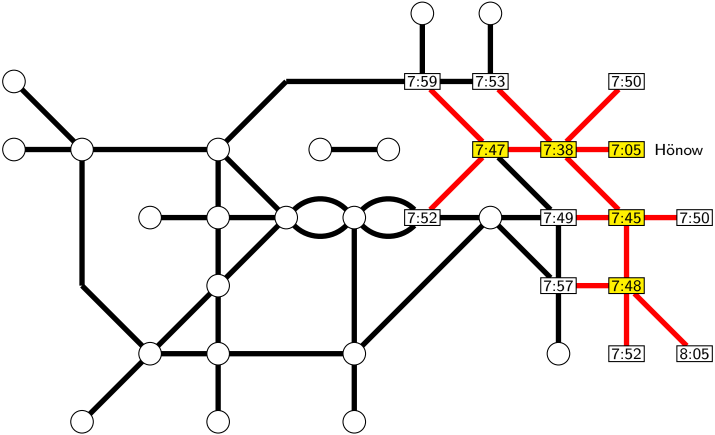
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

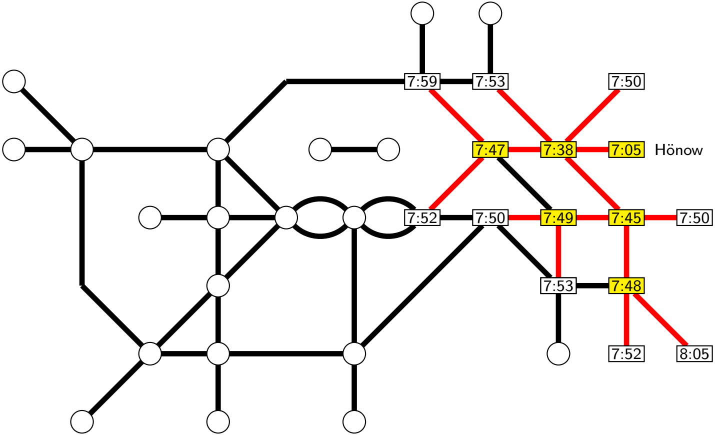
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

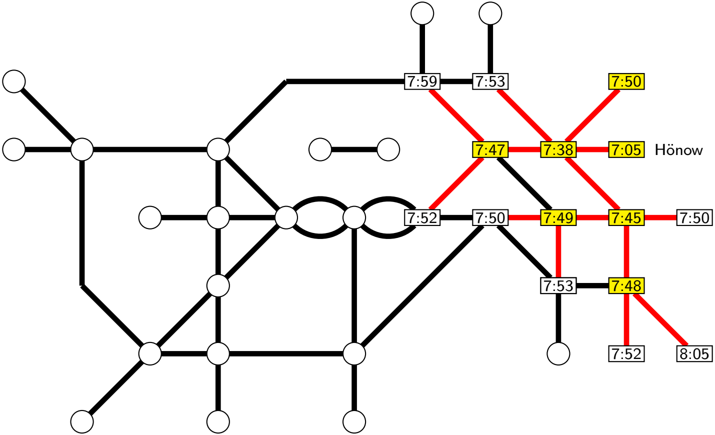
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

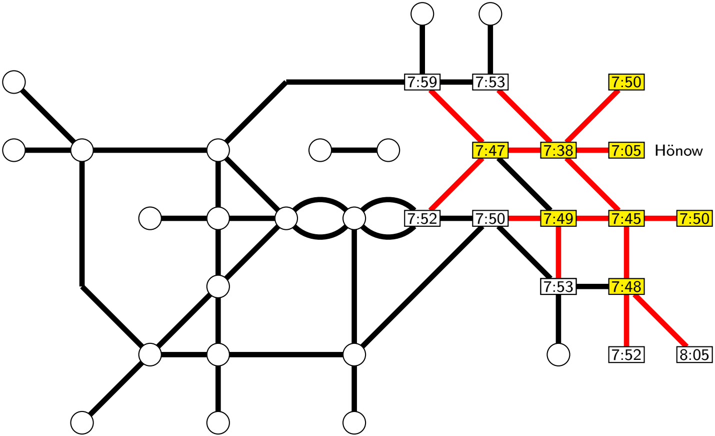
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

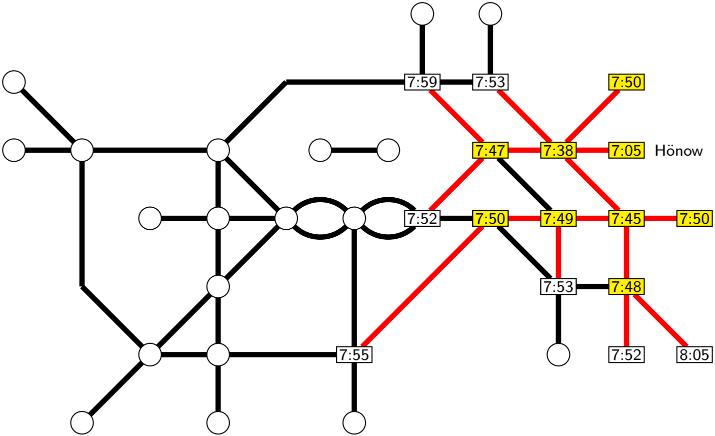
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

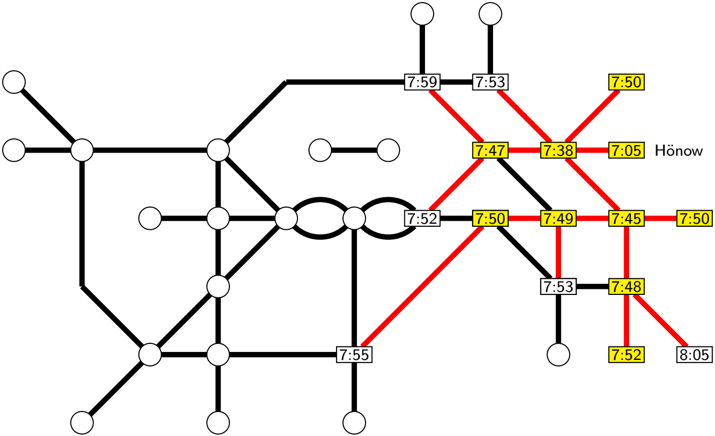
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

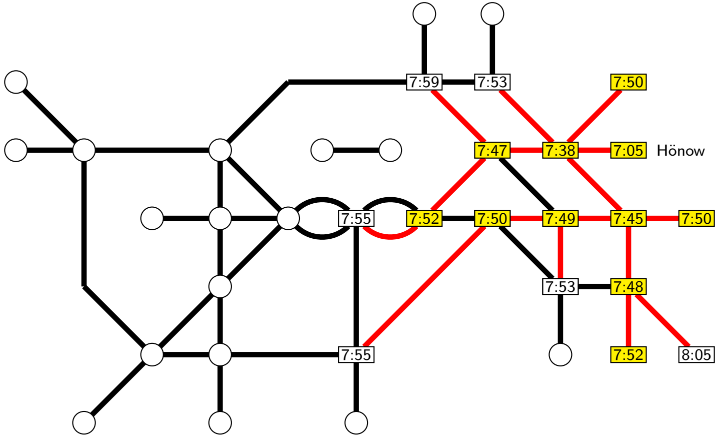
Query: Hönow @ 07:03 $\rightarrow$ all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

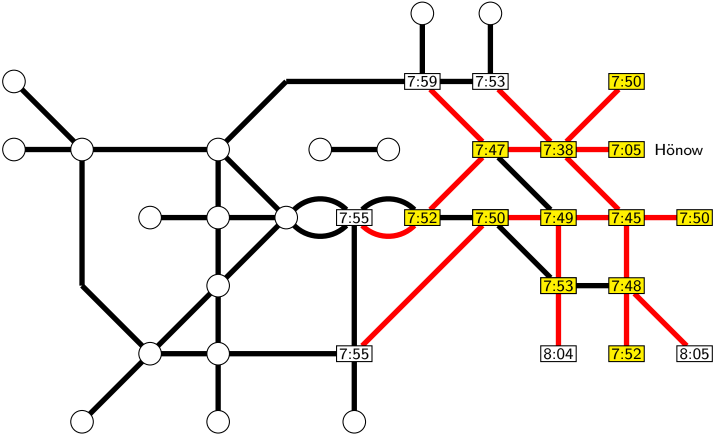
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

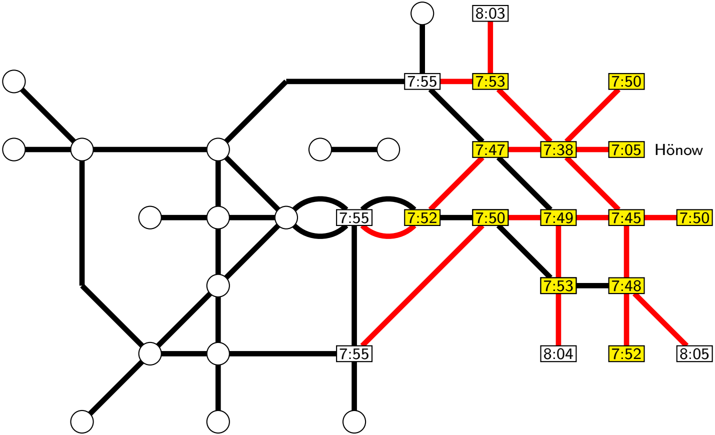
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

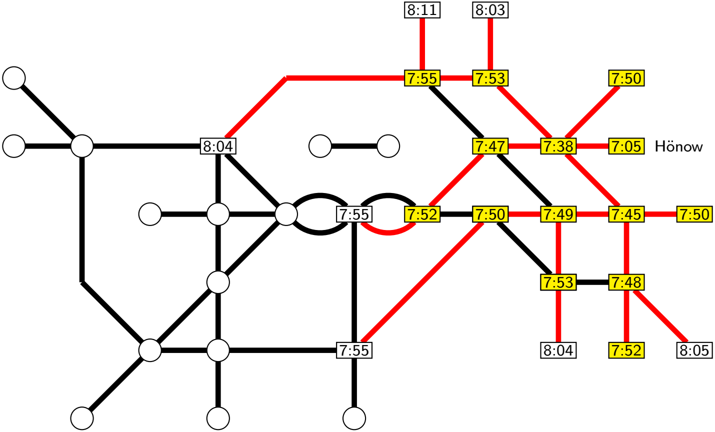
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

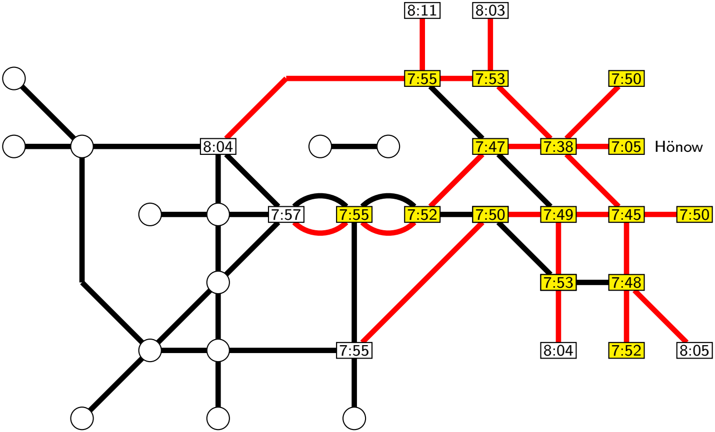
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

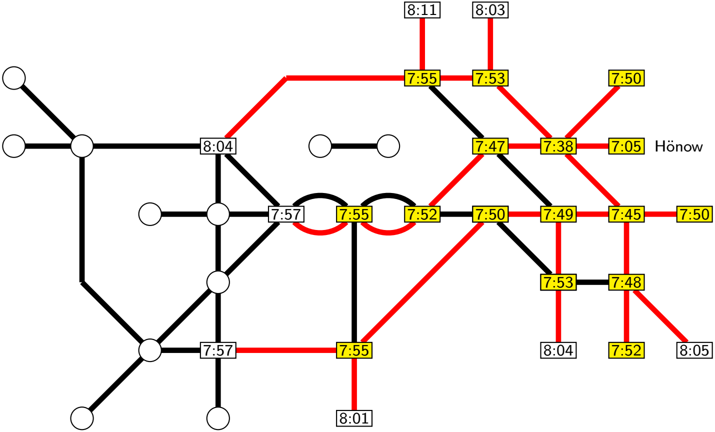
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

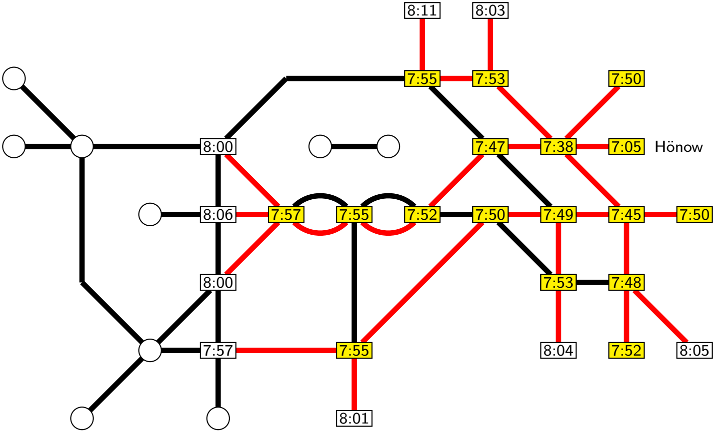
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

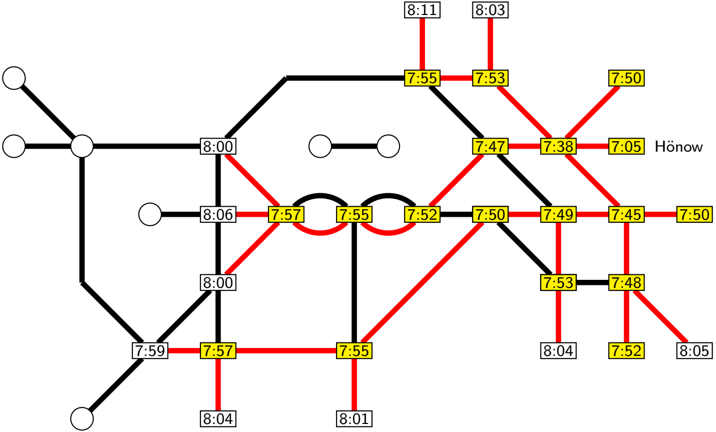
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

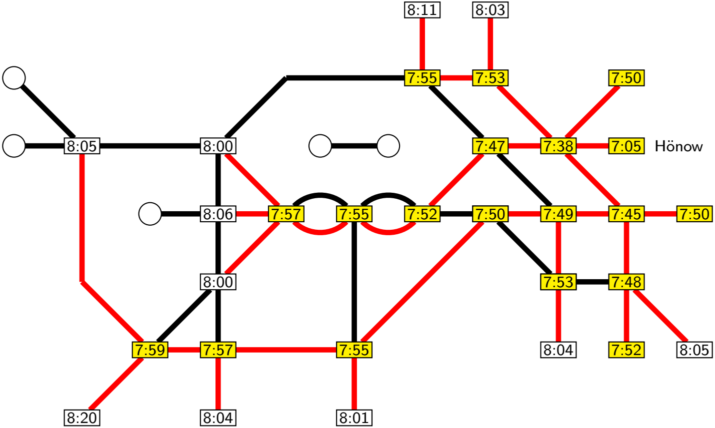
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

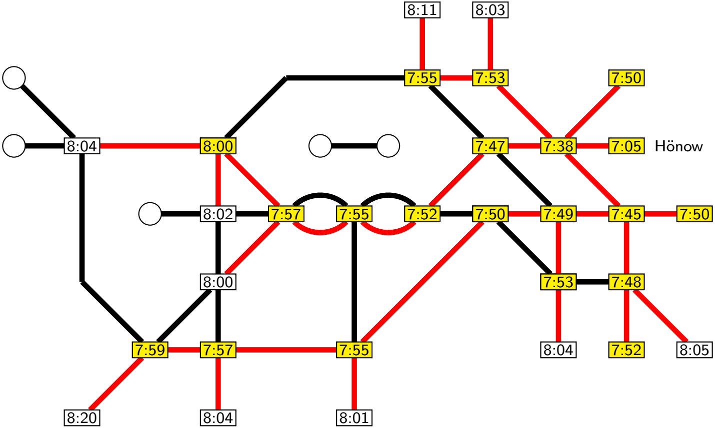
Query: Hönow @ 07:03 $\rightarrow$ all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

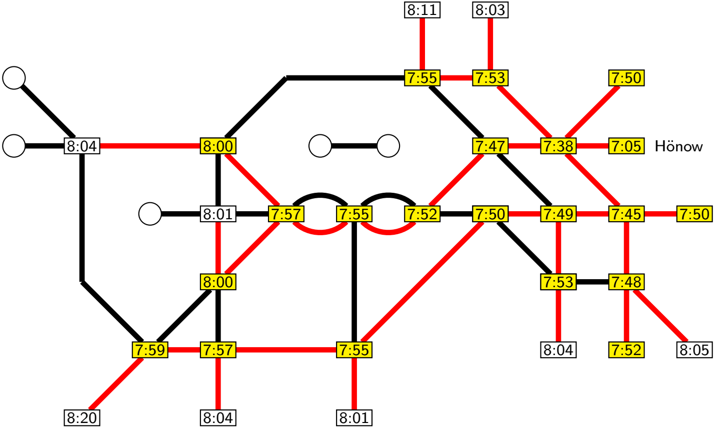
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

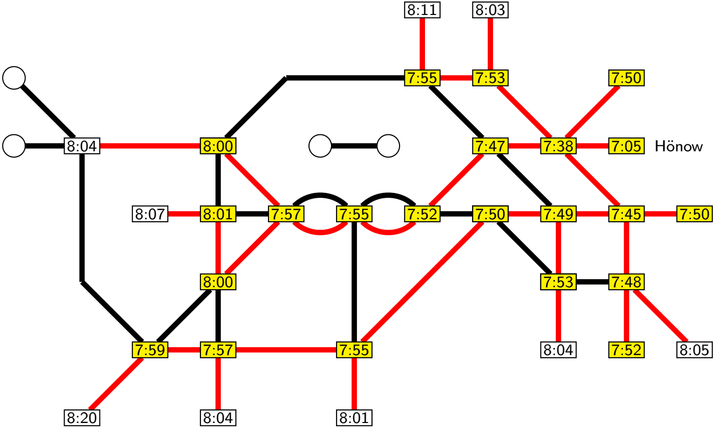
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

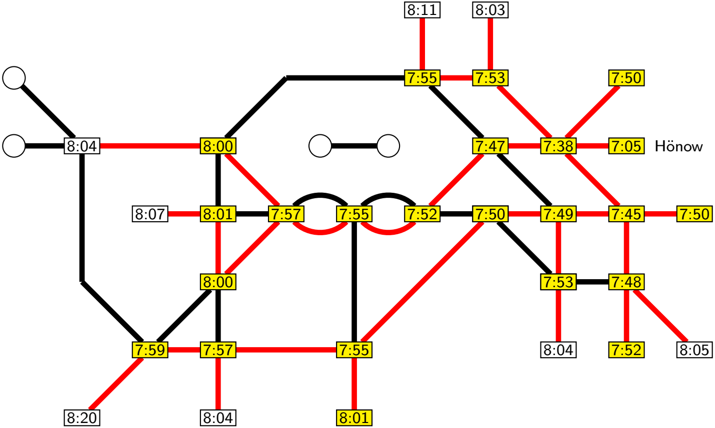
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

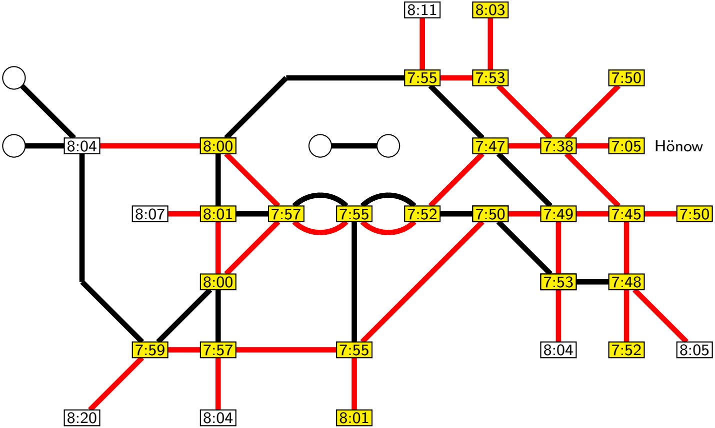
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

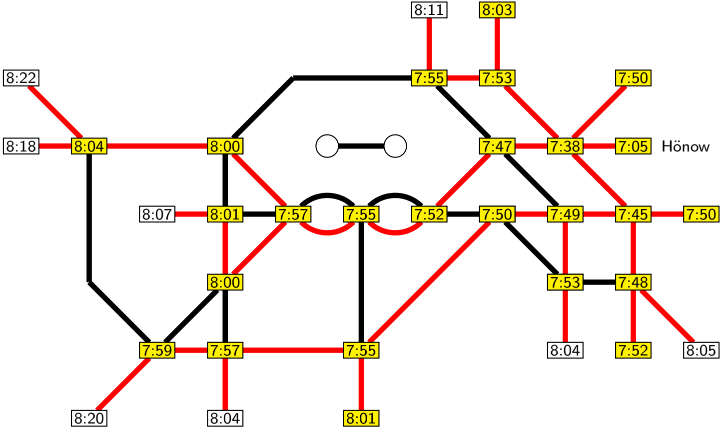
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

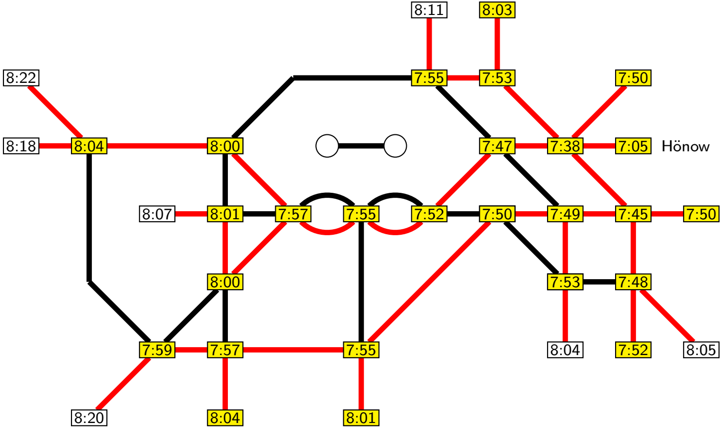
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

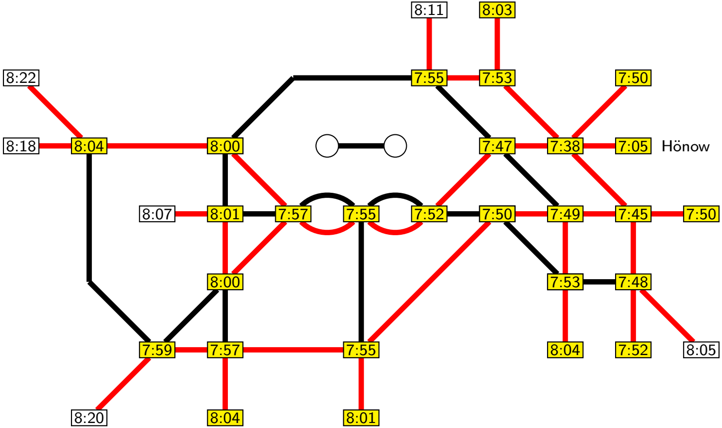
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

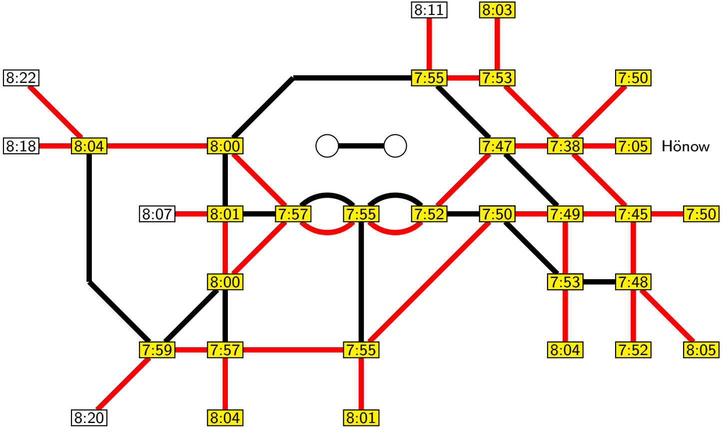
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

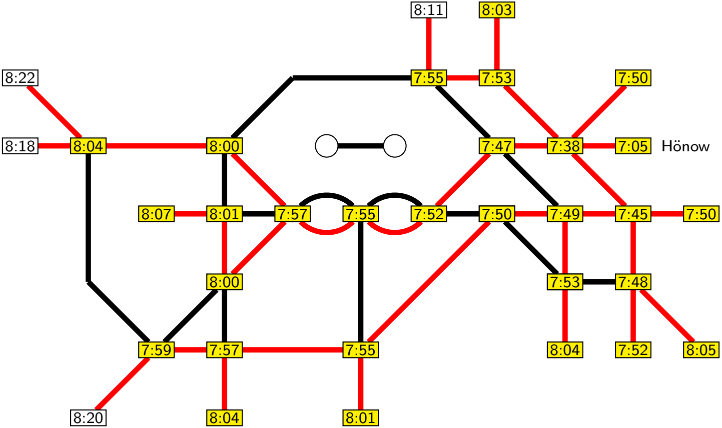
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

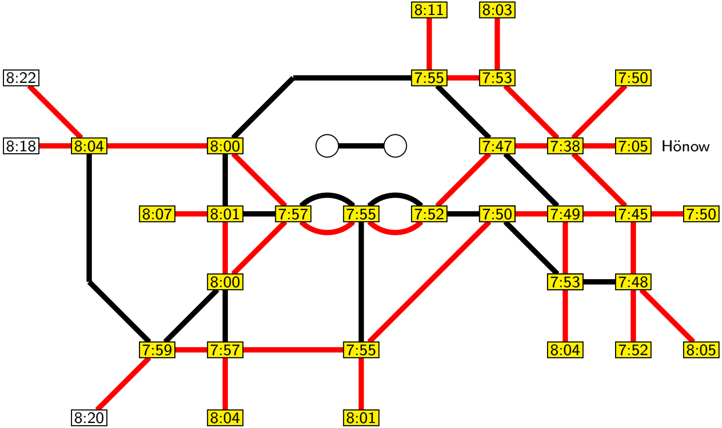
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

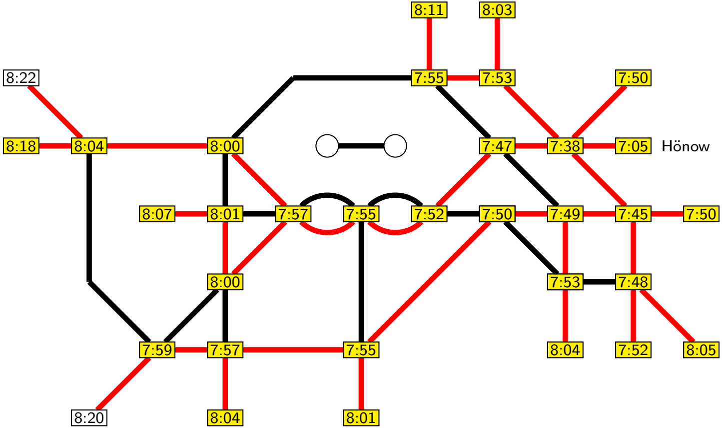
Query: Hönow @ 07:03 → all stations, without minimum change times



Time-Dependent Dijkstra

Example (shortest path tree)

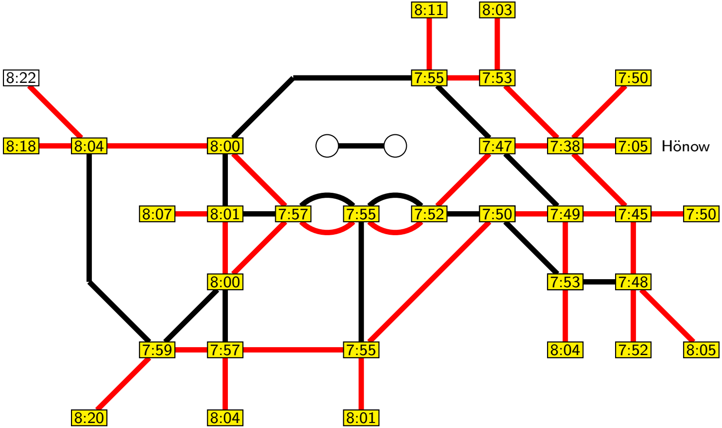
Query: Hönow @ 07:03 → all stations, without minimum change times



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 $\rightarrow$ all stations, without minimum change times