Karlsruher Institut für Technologie

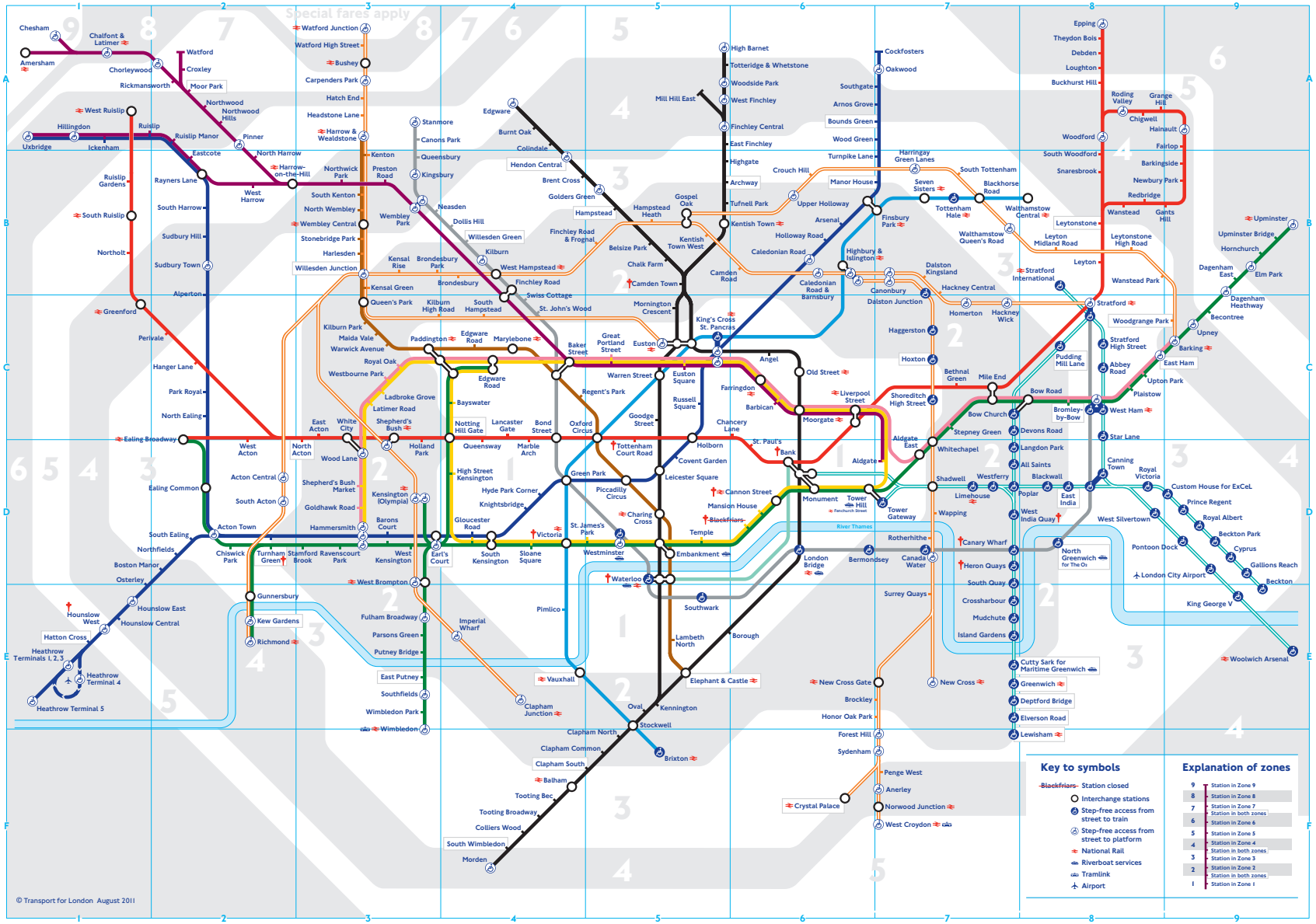
# Algorithmen für Routenplanung

2. Sitzung, Sommersemester 2014

Julian Dibbelt | 16. April 2014

Institut für Theoretische Informatik · Algorithmik · Prof. Dr. Dorothea Wagner

# Fahrplanauskunft



# Abgrenzung zu Straßennetzwerken

**Eingabe bei Straßennetzen**

- Straßenkarte bestehend aus
- Kreuzungen
- Straßensegmenten
- Verschiedene Metriken (Reisezeit, Distanz, . . . )

# Abgrenzung zu Straßennetzwerken

**Eingabe bei Straßennetzen**

- Straßenkarte bestehend aus
- Kreuzungen
- Straßensegmenten
- Verschiedene Metriken (Reisezeit, Distanz, . . . )

Was ist Eingabe bei der Fahrplanauskunft?

# Fahrpläne

**Gegeben (Fahrplan):**

- Menge $\mathcal{B}$ von Bahnhöfen (Stops, Bahnsteigen, ...),
- Menge $\mathcal{Z}$ von Zügen (Bussen, Trams, etc)
- Menge $\mathcal{C}$ von elementaren Verbindungen
- Zur Modellierung von Umstiegen:
  - Mindestumstiegszeiten am Bahnhof: *transfer* : $\mathcal{B} \to \mathbb{N}$.
  - Fußwege zwischen (nahen) Bahnhöfen: *footpath* : $\mathcal{B} \times \mathcal{B} \to \mathbb{N}$

# Fahrpläne

**Gegeben (Fahrplan):**

- Menge $\mathcal{B}$ von Bahnhöfen (Stops, Bahnsteigen, ...),
- Menge $\mathcal{Z}$ von Zügen (Bussen, Trams, etc)
- Menge $\mathcal{C}$ von elementaren Verbindungen
- Zur Modellierung von Umstiegen:
  - Mindestumstiegszeiten am Bahnhof: *transfer* : $\mathcal{B} \to \mathbb{N}$.
  - Fußwege zwischen (nahen) Bahnhöfen: *footpath* : $\mathcal{B} \times \mathcal{B} \to \mathbb{N}$

**Elementare Verbindung**: Tupel bestehend aus

- Zug $Z \in \mathcal{Z}$
- Abfahrtsbahnhof $S_{\text{dep}} \in \mathcal{B}$
- Zielbahnhof $S_{\text{arr}} \in \mathcal{B}$
- Abfahrtszeit $\tau_{\text{dep}} \in \Pi$
- Ankunftszeit $\tau_{\text{arr}} \in \Pi$

# Fahrpläne

**Gegeben (Fahrplan):**

- Menge $\mathcal{B}$ von Bahnhöfen (Stops, Bahnsteigen, ...),
- Menge $\mathcal{Z}$ von Zügen (Bussen, Trams, etc)
- Menge $\mathcal{C}$ von elementaren Verbindungen
- Zur Modellierung von Umstiegen:
  - Mindestumstiegszeiten am Bahnhof: *transfer* : $\mathcal{B} \to \mathbb{N}$.
  - Fußwege zwischen (nahen) Bahnhöfen: *footpath* : $\mathcal{B} \times \mathcal{B} \to \mathbb{N}$

**Elementare Verbindung**: Tupel bestehend aus

- Zug $Z \in \mathcal{Z}$
- Abfahrtsbahnhof $S_{\text{dep}} \in \mathcal{B}$
- Zielbahnhof $S_{\text{arr}} \in \mathcal{B}$
- Abfahrtszeit $\tau_{\text{dep}} \in \Pi$
- Ankunftszeit $\tau_{\text{arr}} \in \Pi$
- **Interpretation:** Zug $Z$ fährt von $S_{\text{dep}}$ nach $S_{\text{arr}}$ ohne Zwischenhalt von $\tau_{\text{dep}}$ bis $\tau_{\text{arr}}$ Uhr

# Fahrpläne II

**Trips**

- Fahrt *eines* Zuges *Z*
- Von Endstation zu Endstation
- Abfahrten an den Stops zu bestimmten Zeiten

# Fahrpläne II

## Trips

- Fahrt *eines* Zuges $Z$
- Von Endstation zu Endstation
- Abfahrten an den Stops zu bestimmten Zeiten

## Routen

- Partitionierung der Trips
- Zwei Trips $t_1, t_2$ gehören zur gleichen Route, gdw.
- $t_1$ und $t_2$ folgen der genau gleichen Sequenz von Stops

# Fahrpläne, Beispiel

**Beispiel für einen Fahrplan:** Menge elementarer Verbindungen

```
(IR 2269, Karlsruhe Hbf,  Pforzheim Hbf,  10:05, 10:23)
(IR 2269, Pforzheim Hbf,  Muehlacker,     10:25, 10:33)
(IR 2269, Muehlacker,     Vaihingen(Enz), 10:34, 10:40)
(IR 2269, Vaihingen(Enz), Stuttgart Hbf,  10:41, 10:57)
...
(ICE 791, Stuttgart Hbf,  Ulm Hbf,        11:12, 12:06)
(ICE 791, Ulm Hbf,        Augsburg Hbf,   12:08, 12:47)
(ICE 791, Augsburg Hbf,   Muenchen Hbf,   12:49, 13:21)
```

mit Zug-ID, Abfahrtshalt, Ankunftshalt und Anfahrts-/Ankunftszeit.

**Weiteres:** (kurze) Fußwege für Umstiege, z.B. Bahnhof zu Bahnhofsvorplatz; Mindestumstiegszeiten

# Fahrpläne, Beispiel

**Beispiel für einen Fahrplan:** Menge elementarer Verbindungen

```
(IR 2269, Karlsruhe Hbf,  Pforzheim Hbf,  10:05, 10:23)
(IR 2269, Pforzheim Hbf,  Muehlacker,     10:25, 10:33)
(IR 2269, Muehlacker,     Vaihingen(Enz), 10:34, 10:40)
(IR 2269, Vaihingen(Enz), Stuttgart Hbf,  10:41, 10:57)
...
(ICE 791, Stuttgart Hbf,  Ulm Hbf,        11:12, 12:06)
(ICE 791, Ulm Hbf,        Augsburg Hbf,   12:08, 12:47)
(ICE 791, Augsburg Hbf,   Muenchen Hbf,   12:49, 13:21)
```

mit Zug-ID, Abfahrtshalt, Ankunftshalt und Anfahrts-/Ankunftszeit.

**Weiteres:** (kurze) Fußwege für Umstiege, z.B. Bahnhof zu Bahnhofsvorplatz; Mindestumstiegszeiten

**Frage:** Wie Fahrplan modellieren?

# Modellierung von Fahrplänen

## Zwei grundlegende Ansätze

1. Modellierung als gerichteter Graph
2. Keine besondere Modellierung (benutze Fahrplan "direkt")

# Modellierung von Fahrplänen

## Zwei grundlegende Ansätze

1. Modellierung als gerichteter Graph
2. Keine besondere Modellierung (benutze Fahrplan "direkt")

Jetzt ersteres, später zweiteres.

# Modellierung von Fahrplänen

## Zwei grundlegende Ansätze

1. Modellierung als gerichteter Graph
2. Keine besondere Modellierung (benutze Fahrplan "direkt")

Jetzt ersteres, später zweiteres.

## Modellierung als Graph

- Reduziere auf (bekanntes) kürzeste-Wege-Problem
- Optimale Reiserouten entsprechen kürzesten Wegen
- Frage: Wie die Zeitabhängigkeit (Abfahrten/Ankünfte) kodieren?

# Modellierung: Zwei Ansätze

## 1. Zeitexpandiert

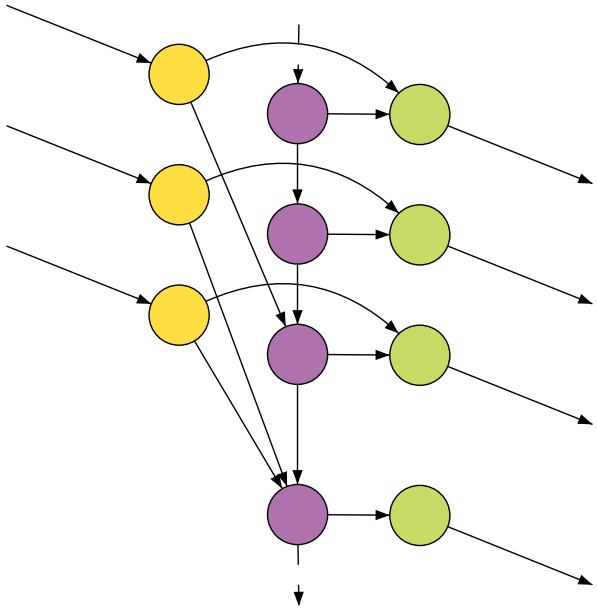
- Zeitabhängigkeiten ausrollen
- Knoten entsprechen Ereignissen im Fahrplan
- Kanten verbinden Ereignisse miteinander
  - Zugfahrt eines Zuges,
  - Umstieg zwischen Zügen,
  - Warten
- \- Großer Graph (viele Knoten und Kanten)
- \+ Einfacher Anfragealgorithmus (Dijkstra)

## 2. Zeitabhängig

- Zeitabhängigkeit an den Kanten
- Knoten entsprechen Bahnhöfen
- Kante $\Leftrightarrow$ Zug verbindet Bahnhöfe
  - Transferzeiten?
- \+ Kleiner Graph
- \- Zeitabhängige KW-Algorithmen?

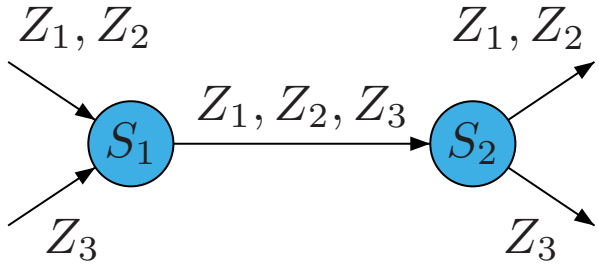
# Modellierung: Zwei Ansätze

## 1. Zeitexpandiert



- Arrival-, Transfer- und Departure-Ereignisse
- Für jeden Zug
- Kantengewicht = Zeitdiff.
  (alternativ: ungewichtet, Knotenlabel = Ereigniszeit)

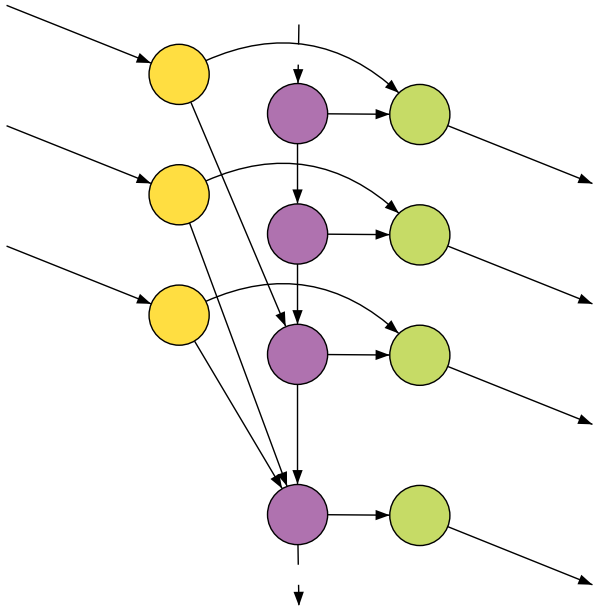
## 2. Zeitabhängig




- Pro Bahnhof: Stationsknoten
- Kanten: zeitabhängig
- Umstiege?

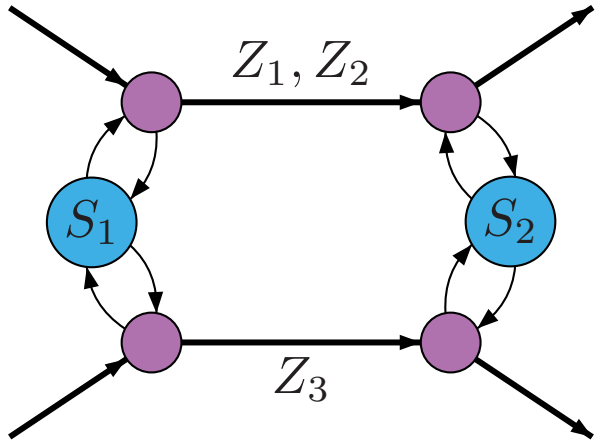
# Modellierung: Zwei Ansätze

## 1. Zeitexpandiert



- Arrival-, Transfer- und Departure-Ereignisse
- Für jeden Zug
- Kantengewicht = Zeitdiff.
  (alternativ: ungewichtet, Knotenlabel = Ereigniszeit)

## 2. Zeitabhängig




- Pro Bahnhof: Stationsknoten
- Partitioniere Züge in Routen
- Pro Route: Route-Knoten
- Routenkanten: zeitabhängig
- Stationskanten: Transferzeit

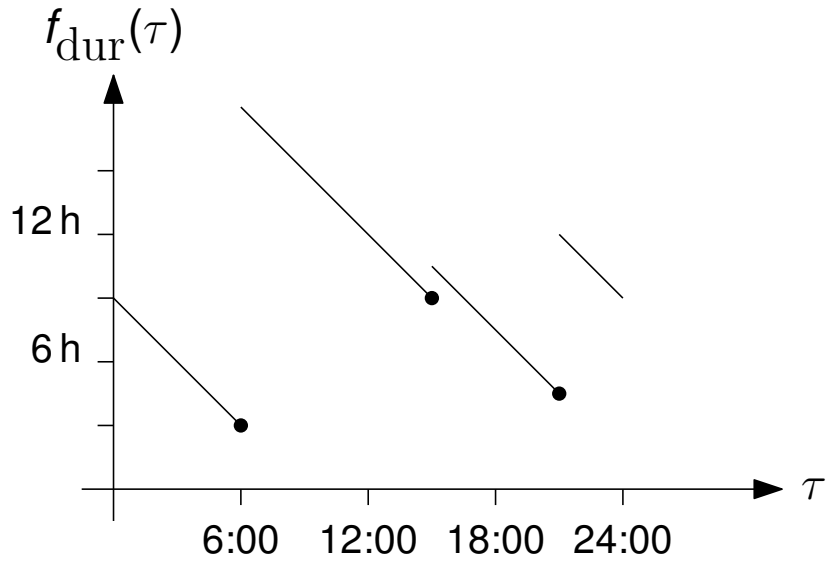
# Zeitabhängige Kanten

Elem. Verbindungen modelliert durch stückweise lineare Funktionen

Elem. Verbindungen zw. $S_i$ und $S_j$:

| id | dep.-time | travel-time |
|---|---|---|
| 1 | 06:00 | 3 h 00 min |
| 2 | 15:00 | 9 h 00 min |
| 3 | 21:00 | 4 h 30 min |
| ⋮ | ⋮ | ⋮ |

Entsprechende Funktion:




- Für jede Verbindung: Connection Point $(\tau, w)$
  $\tau \mathrel{\hat{=}}$ Abfahrtszeit, $w \mathrel{\hat{=}}$ Reisezeit (bzw. Ankunftszeit)
- Zwischen Verbindungen: Lineares Warten

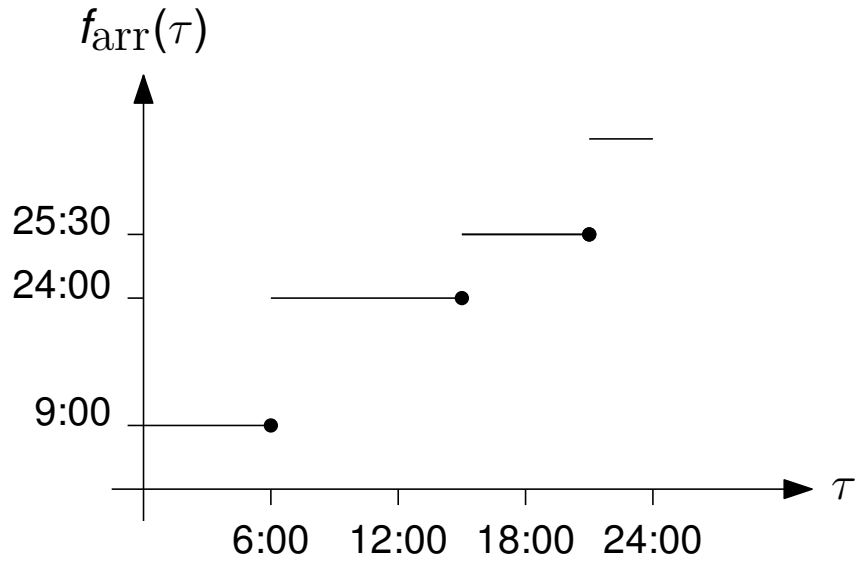
# Zeitabhängige Kanten

Elem. Verbindungen modelliert durch stückweise lineare Funktionen

Elem. Verbindungen zw. $S_i$ und $S_j$:

| id | dep.-time | travel-time |
|---|---|---|
| 1 | 06:00 | 3 h 00 min |
| 2 | 15:00 | 9 h 00 min |
| 3 | 21:00 | 4 h 30 min |
| ⋮ | ⋮ | ⋮ |

Entsprechende Funktion:




- Für jede Verbindung: Connection Point $(\tau, w)$
  $\tau \mathrel{\hat{=}}$ Abfahrtszeit, $w \mathrel{\hat{=}}$ Reisezeit (bzw. Ankunftszeit)
- Zwischen Verbindungen: Lineares Warten

# FIFO-Eigenschaft (Wdh.)

## Definition

Sei $f : \mathbb{R}_0^+ \to \mathbb{R}_0^+$ eine Reisezeit-Funktion. $f$ erfüllt die *FIFO-Eigenschaft*, wenn für jedes $\varepsilon > 0$ und alle $\tau \in \mathbb{R}_0^+$ gilt, dass

$$f(\tau) \leq \varepsilon + f(\tau + \varepsilon).$$

**Diskussion**

- Interpretation: “Warten lohnt sich nie”
- Kürzeste Wege auf Graphen mit non-FIFO Funktionen zu finden ist NP-schwer.
  (wenn Warten an Knoten nicht erlaubt ist)

  $\Rightarrow$ Sicherstellen, dass Funktionen FIFO-Eigenschaft erfüllen.

# FIFO-Eigenschaft (Wdh.)

## Definition

Sei $f : \mathbb{R}_0^+ \to \mathbb{R}_0^+$ eine Ankunftszeit-Funktion. $f$ erfüllt die *FIFO-Eigenschaft*, wenn für jedes $\varepsilon > 0$ und alle $\tau \in \mathbb{R}_0^+$ gilt, dass

$$f(\tau) \leq f(\tau + \varepsilon).$$

**Diskussion**

- Interpretation: “Warten lohnt sich nie”
- Kürzeste Wege auf Graphen mit non-FIFO Funktionen zu finden ist NP-schwer.
  (wenn Warten an Knoten nicht erlaubt ist)

  $\Rightarrow$ Sicherstellen, dass Funktionen FIFO-Eigenschaft erfüllen.

# Anfrageszenarien

**Zeit-Anfrage:**

- finde kürzesten Weg für Abfahrtszeit $\tau$
- analog zu Dijkstra?

# Anfrageszenarien

**Zeit-Anfrage:**

- finde kürzesten Weg für Abfahrtszeit $\tau$
- analog zu Dijkstra?

**Profil-Anfrage:**

- finde kürzesten Weg für alle Abfahrtszeitpunkte
- analog zu Dijkstra?

# Zeit-Anfragen

**Gegeben:** Startbahnhof $S$, Zielbahnhof $T$ und Abfahrtszeit $\tau_S$



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

# Zeit-Anfragen

**Gegeben:** Startbahnhof $S$, Zielbahnhof $T$ und Abfahrtszeit $\tau_S$

## 1. Zeitexpandiert

**Startknoten:**

- *Erstes* Transferevent in $S$ mit Zeit $\tau \geq \tau_S$.

**Zielknoten:**

- Im Vorraus unbekannt!
- Stoppkriterium: Erster gesettleter Knoten an $T$ induziert schnellste Verbindung zu $T$

## 2. Zeitabhängig

**Startknoten:**

- Bahnhofsknoten $S$

**Zielknoten:**

- Bahnhofsknoten $T$

**Anfrage:**

- Time-Dependent Dijkstra mit Zeit $\tau_S$
- Hier: Ankunftszeit im Vorraus unbekannt

# Zeit-Anfragen

**Algorithm 1**: `Time-Dijkstra`$(G = (V, E), s, \tau)$

```
1  d[s] = 0
2  Q.clear(), Q.add(s, 0)
3  while !Q.empty() do
4      u ← Q.deleteMin()
5      for all edges e = (u, v) ∈ E do
           // len(e, ·) = f^e_dep(·)
6          if d[u] + len(e, τ + d[u]) < d[v] then
7              d[v] ← d[u] + len(e, τ + d[u])
8              p[v] ← u
9              if v ∈ Q then Q.decreaseKey(v, d[v])
10             else Q.insert(v, d[v])
```

# Zeit-Anfragen

**Algorithm 2:** `Time-Dijkstra`$(G = (V, E), s, \tau)$

```
1  d[s] = τ
2  Q.clear(), Q.add(s, 0)
3  while !Q.empty() do
4      u ← Q.deleteMin()
5      for all edges e = (u, v) ∈ E do
           // len(e, ·) = f^e_arr(·)
6          if len(e, d[u]) < d[v] then
7              d[v] ← len(e, d[u])
8              p[v] ← u
9              if v ∈ Q then Q.decreaseKey(v, d[v])
10             else Q.insert(v, d[v])
```

# Diskussion Zeit-Anfragen

**Beobachtung:**

- Nur ein Unterschied zu Dijkstra
- Auswertung der Kanten

# Diskussion Zeit-Anfragen

**Beobachtung:**

- Nur ein Unterschied zu Dijkstra
- Auswertung der Kanten

**non-FIFO Netzwerke:**

- Im Kreis fahren kann sich lohnen
- NP-schwer (wenn Warten an Knoten nicht erlaubt ist)
- Transportnetzwerke sind FIFO modellierbar (notfalls Multikanten)

# Diskussion Zeit-Anfragen

**Beobachtung:**

- Nur ein Unterschied zu Dijkstra
- Auswertung der Kanten

**non-FIFO Netzwerke:**

- Im Kreis fahren kann sich lohnen
- NP-schwer (wenn Warten an Knoten nicht erlaubt ist)
- Transportnetzwerke sind FIFO modellierbar (notfalls Multikanten)

**In unserem Szenario:**

- Sicherstellen dass alle Routen FIFO sind.
- Für alle Trips $t_i, t_j$ der Route muss gelten:
- $t_i$ fährt an *jeder* Station jeweils vor $t_j$ ab (oder andersherum).

# Profil-Anfragen

**Gegeben:** Startbahnhof $S$, Zielbahnhof $T$



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

# Profil-Anfragen

**Gegeben:** Startbahnhof $S$, Zielbahnhof $T$

## 1. Zeitexpandiert

?

Geht, aber nicht Teil der VL

## 2. Zeitabhängig

**Startknoten:**

- Bahnhofsknoten $S$

**Zielknoten:**

- Bahnhofsknoten $T$

**Anfrage:**

- Label-Correcting Algorithmus von $S$

# Profil-Anfragen

**Algorithm 3:** `Profile-Search`$(G = (V, E), s)$

```
1  d_*[s] = 0
2  Q.clear(), Q.add(s, 0)
3  while !Q.empty() do
4      u ← Q.deleteMin()
5      for all edges e = (u, v) ∈ E do
6          if d_*[u] ⊕ len(e) ≱ d_*[v] then
7              d_*[v] ← min(d_*[u] ⊕ len(e), d_*[v])
8              if v ∈ Q then Q.decreaseKey(v, d[v])
9              else Q.insert(v, d[v])
```

# Diskussion Profil-Anfragen

**Beobachtungen:**

- Operationen auf Funktionen
- Knotenlabel: Funktion
- Knotenlabel nicht skalar $\Rightarrow$ keine Totalordnung der Knotenlabel
- Wonach pqueue ordnen?
- Priorität im Prinzip frei wählbar
  ($\underline{d}[u]$ ist das Minimum der Funktion $d_*[u]$)
- Knoten können mehrfach besucht werden $\Rightarrow$ label-correcting

# Operationen

**Funktion gegeben durch:**

- Menge von Interpolationspunkten
- $I^f := \{(t_1^f, w_1^f), \ldots, (t_k^f, w_k^f)\}$

**3 Operationen notwendig:**

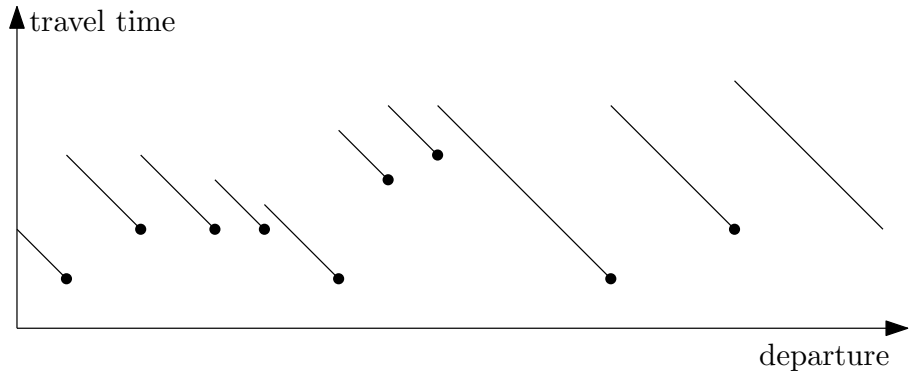
- Auswertung
- Linken $\oplus$
- Minimumsbildung

# Public Transport: Auswertung

**Evaluation von $f(\tau)$:**

- Suche Punkte mit $t_i \geq \tau$ und $t_i - \tau$ minimal
- dann Evaluation durch

$$f(\tau) = w_i + (t_i - \tau)$$

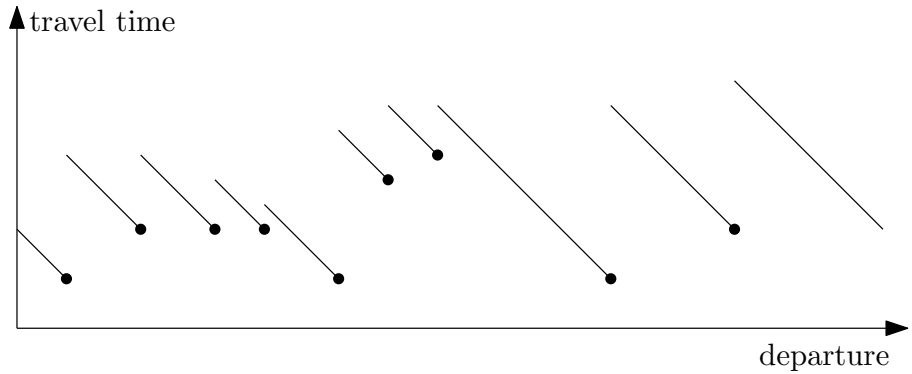


# Public Transport: Auswertung

**Evaluation von $f(\tau)$:**

- Suche Punkte mit $t_i \geq \tau$ und $t_i - \tau$ minimal
- dann Evaluation durch

$$f(\tau) = w_i + (t_i - \tau)$$




**Problem:**

- Finden von $t_i$ und $t_{i+1}$
- Theoretisch:
  - Lineare Suche: $\mathcal{O}(|I|)$
  - Binäre Suche: $\mathcal{O}(\log_2 |I|)$
- praktisch:
  - $|I| < 30$: Lineare Suche
  - Sonst: Lineare Suche mit Startpunkt $\frac{\tau}{\Pi} \cdot |I|$

# Linken

## Definition

Seien $f : \mathbb{R}_0^+ \to \mathbb{R}_0^+$ und $g : \mathbb{R}_0^+$ zwei Reisezeit-Funktionen die die FIFO-Eigenschaft erfüllen. Die Linkoperation $f \oplus g$ ist dann definiert durch

$$f \oplus g := f + g \circ (\mathrm{id} + f)$$

**Oder**

$$(f \oplus g)(\tau) := f(\tau) + g(\tau + f(\tau))$$

# Linken

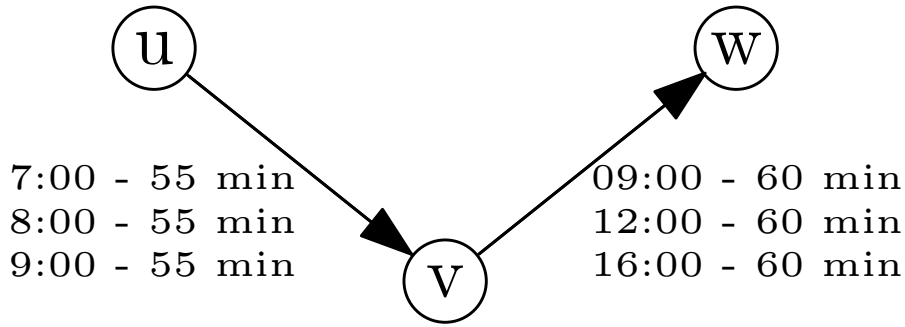
## Definition

Seien $f : \mathbb{R}_0^+ \to \mathbb{R}_0^+$ und $g : \mathbb{R}_0^+$ zwei Ankunftszeit-Funktionen die die FIFO-Eigenschaft erfüllen. Die Linkoperation $f \oplus g$ ist dann definiert durch

$$f \oplus g := g \circ f$$

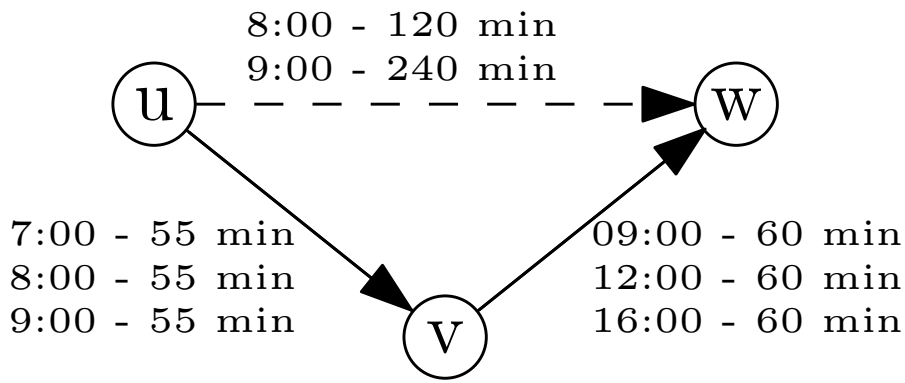
# Public Transport: Link

## Linken zweier Funktionen $f$ und $g$



# Public Transport: Link

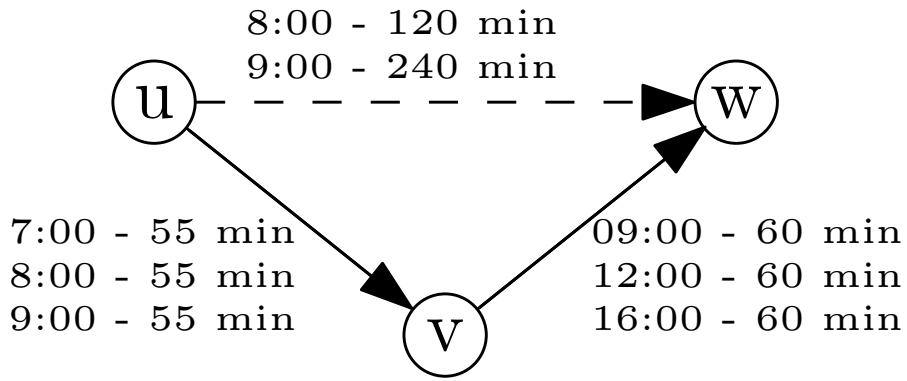
## Linken zweier Funktionen $f$ und $g$



# Public Transport: Link

**Linken zweier Funktionen $f$ und $g$**

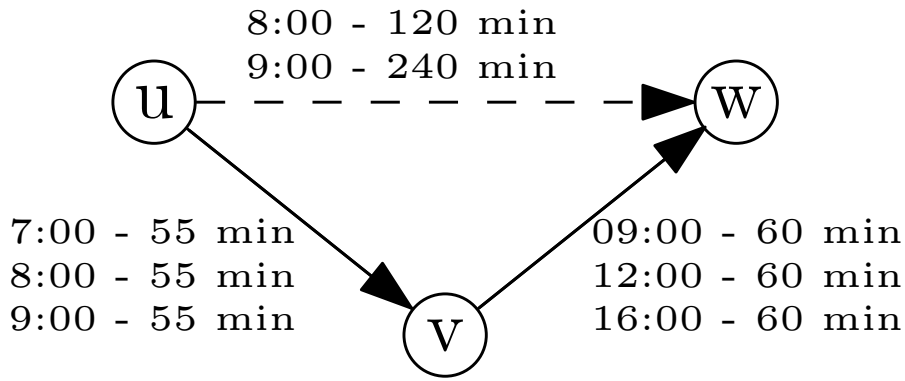
- Für jeden Punkt $(t_i^f, w_i^f)$ bestimme den Verbindungspunkt $(t_j^g, w_j^g)$ mit $t_j^g - t_i^f - w_i^f \geq 0$ minimal
- Erste Verbindung, die man auf $g$ erreichen kann



# Public Transport: Link

**Linken zweier Funktionen $f$ und $g$**

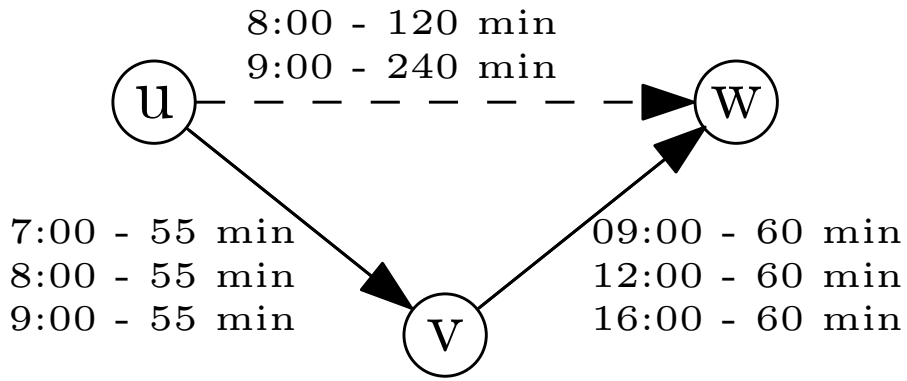
- Für jeden Punkt $(t_i^f, w_i^f)$ bestimme den Verbindungspunkt $(t_j^g, w_j^g)$ mit $t_j^g - t_i^f - w_i^f \geq 0$ minimal
- Erste Verbindung, die man auf $g$ erreichen kann
- Füge $(t_i^f, t_j^g + w_j^g - t_i^f)$ hinzu



# Public Transport: Link

**Linken zweier Funktionen $f$ und $g$**

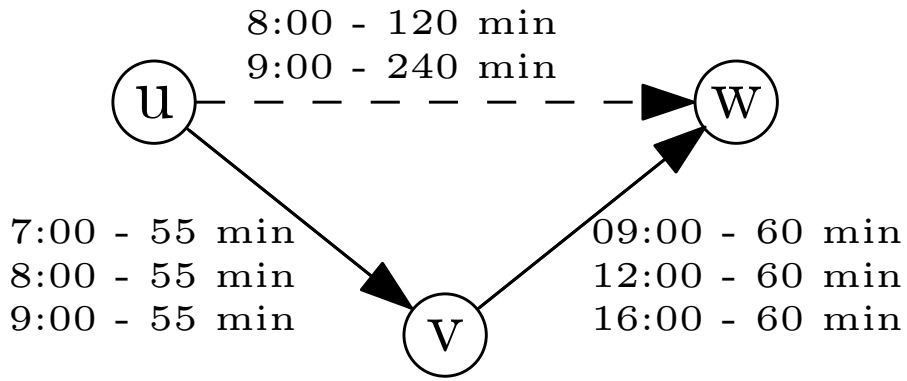
- Für jeden Punkt $(t_i^f, w_i^f)$ bestimme den Verbindungspunkt $(t_j^g, w_j^g)$ mit $t_j^g - t_i^f - w_i^f \geq 0$ minimal
- Erste Verbindung, die man auf $g$ erreichen kann
- Füge $(t_i^f, t_j^g + w_j^g - t_i^f)$ hinzu
- Wenn zwei Punkte den gleichen Verbindungspunkt haben, behalte nur den mit größerem $t_i^f$



# Public Transport: Link

**Linken zweier Funktionen $f$ und $g$**

- Für jeden Punkt $(t_i^f, w_i^f)$ bestimme den Verbindungspunkt $(t_j^g, w_j^g)$ mit $t_j^g - t_i^f - w_i^f \geq 0$ minimal
- Erste Verbindung, die man auf $g$ erreichen kann
- Füge $(t_i^f, t_j^g + w_j^g - t_i^f)$ hinzu
- Wenn zwei Punkte den gleichen Verbindungspunkt haben, behalte nur den mit größerem $t_i^f$
- Sweep-Algorithmus



# Public Transport: Diskussion Link

**Laufzeit**

- Sweep-Algorithmus
- $\mathcal{O}(|I^f| + |I^g|)$
- Zum Vergleich: Zeitunabhängig: $\mathcal{O}(1)$

# Public Transport: Diskussion Link

Karlsruher Institut für Technologie

**Laufzeit**

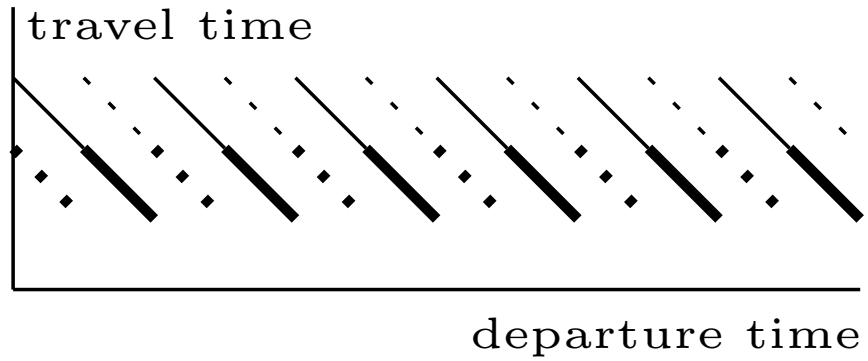
- Sweep-Algorithmus
- $\mathcal{O}(|I^f| + |I^g|)$
- Zum Vergleich: Zeitunabhängig: $\mathcal{O}(1)$

**Speicherverbrauch**

- Gelinkte Funktion hat $\min\{|I^f|, |I^g|\}$ Interpolationspunkte

# Public Transport: Merge

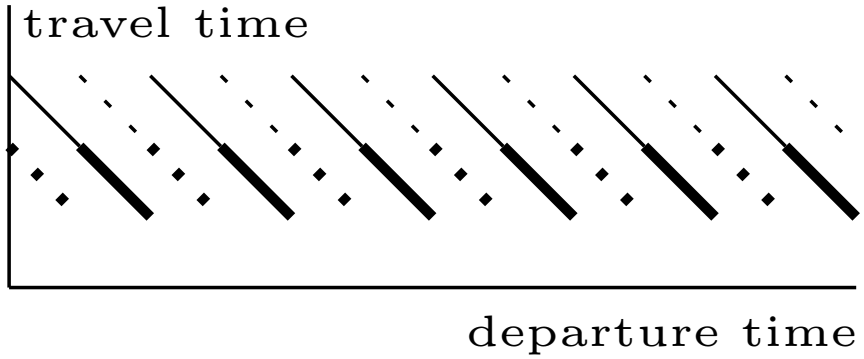
**Minimum zweier Funktionen $f$ und $g$**



# Public Transport: Merge

**Minimum zweier Funktionen $f$ und $g$**

- Für alle $(t_i^f, w_i^f)$: behalte Punkt, wenn $w_i^f < g(t_i^f)$
- Für alle $(t_j^g, w_j^g)$: behalte Punkt, wenn $w_j^g < f(t_j^g)$
- Keine Schnittepunkte möglich(!)



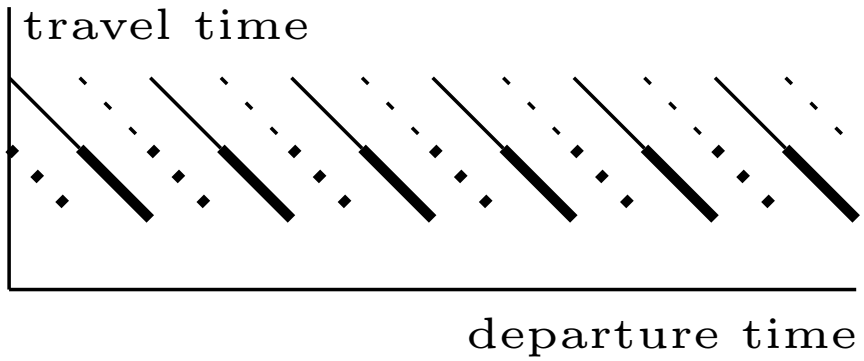
# Public Transport: Merge

**Minimum zweier Funktionen $f$ und $g$**

- Für alle $(t_i^f, w_i^f)$: behalte Punkt, wenn $w_i^f < g(t_i^f)$
- Für alle $(t_j^g, w_j^g)$: behalte Punkt, wenn $w_j^g < f(t_j^g)$
- Keine Schnittepunkte möglich(!)

**Vorgehen:**

- Linearer Sweep



# Public Transport: Diskussion Merge

**Laufzeit**

- Sweep-Algorithmus
- $\mathcal{O}(|I^f| + |I^g|)$
- Zum Vergleich: Zeitunabhängig: $\mathcal{O}(1)$

# Public Transport: Diskussion Merge

Karlsruher Institut für Technologie

**Laufzeit**

- Sweep-Algorithmus
- $\mathcal{O}(|I^f| + |I^g|)$
- Zum Vergleich: Zeitunabhängig: $\mathcal{O}(1)$

**Speicherverbrauch**

- Keine Schnittpunkte

$\Rightarrow$ Minimum-Funktion kann maximal $|I^f| + |I^g|$ Interpolationspunkte enthalten

# Profil-Anfragen

**Gegeben:**
Zeitabhängiges Netzwerk $G = (V, E)$ und Startbahnhof $S$.

# Profil-Anfragen

**Gegeben:**
Zeitabhängiges Netzwerk $G = (V, E)$ und Startbahnhof $S$.

**Problem (Profil-Anfrage)**:
Berechne die *Reisezeitfunktion* $\text{dist}_S(v, \tau)$, so dass $\text{dist}_S(v, \tau)$ die Länge des kürzesten Weges von $S$ nach $v$ in $G$ zur Abfahrtszeit $\tau$ an $S$ für alle $\tau \in \Pi$ und $v \in V$ ist.

# Profil-Anfragen

**Gegeben:**
Zeitabhängiges Netzwerk $G = (V, E)$ und Startbahnhof $S$.

**Problem (Profil-Anfrage)**:
Berechne die *Reisezeitfunktion* $\text{dist}_S(v, \tau)$, so dass $\text{dist}_S(v, \tau)$ die Länge des kürzesten Weges von $S$ nach $v$ in $G$ zur Abfahrtszeit $\tau$ an $S$ für alle $\tau \in \Pi$ und $v \in V$ ist.

**Bisheriger Ansatz:**
Erweitere Dijkstra's Algorithmus zu Label-Correcting Algorithmus

- Benutze Funktionen statt Konstanten
- Verliert Label-Setting Eigenschaft von Dijkstra
- Deutlich langsamer als Dijkstra ($\approx$ Faktor 50)

# Hauptidee

**Beobachtung:**
Jeder Reiseplan ab $S$ (irgendwohin) beginnt mit einer Verb. an $S$.

# Hauptidee

**Beobachtung:**
Jeder Reiseplan ab $S$ (irgendwohin) beginnt mit einer Verb. an $S$.

## Naiver Ansatz

Für jede ausgehende Verbindung $c_i$ an $S$:
Separate Zeitanfrage mit Abfahrtszeit $\tau_{\text{dep}}(c_i)$.

# Hauptidee

**Beobachtung:**
Jeder Reiseplan ab $S$ (irgendwohin) beginnt mit einer Verb. an $S$.

## Naiver Ansatz

Für jede ausgehende Verbindung $c_i$ an $S$:
Separate Zeitanfrage mit Abfahrtszeit $\tau_{\text{dep}}(c_i)$.

**Nachteile**

- Zu viele redundante Berechnungen
- Nicht jede Verbindung ab $S$ trägt zu $\text{dist}_S(v, \cdot)$ bei
  Langsame Züge für weite Reisen machen wenig Sinn

# Hauptidee

**Beobachtung:**
Jeder Reiseplan ab $S$ (irgendwohin) beginnt mit einer Verb. an $S$.

## Naiver Ansatz

Für jede ausgehende Verbindung $c_i$ an $S$:
Separate Zeitanfrage mit Abfahrtszeit $\tau_{\text{dep}}(c_i)$.

**Nachteile**

- Zu viele redundante Berechnungen
- Nicht jede Verbindung ab $S$ trägt zu $\text{dist}_S(v, \cdot)$ bei
  Langsame Züge für weite Reisen machen wenig Sinn

| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 6:30<br>8:30 | 7:04<br>8:30 | 9:26<br>14:28 | 10:34<br>14:28 | 11:08<br>14:28 | 12:42<br>14:28 | 13:01<br>16:46 | 13:58<br>16:46 | 16:46<br>23:30 | 18:24<br>23:30 | 19:20<br>23:30 | 21:08<br>23:30 |

# Hauptidee

**Beobachtung:**
Jeder Reiseplan ab $S$ (irgendwohin) beginnt mit einer Verb. an $S$.

## Naiver Ansatz

Für jede ausgehende Verbindung $c_i$ an $S$:
Separate Zeitanfrage mit Abfahrtszeit $\tau_{\text{dep}}(c_i)$.

**Nachteile**

- Zu viele redundante Berechnungen
- Nicht jede Verbindung ab $S$ trägt zu $\text{dist}_S(v, \cdot)$ bei
  Langsame Züge für weite Reisen machen wenig Sinn

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 6:30<br>8:30 | 7:04<br>8:30 | 9:26<br>14:28 | 10:34<br>14:28 | 11:08<br>14:28 | 12:42<br>14:28 | 13:01<br>16:46 | 13:58<br>16:46 | 16:46<br>23:30 | 18:24<br>23:30 | 19:20<br>23:30 | 21:08<br>23:30 |

# Hauptidee

**Beobachtung:**
Jeder Reiseplan ab $S$ (irgendwohin) beginnt mit einer Verb. an $S$.

## Naiver Ansatz

Für jede ausgehende Verbindung $c_i$ an $S$:
Separate Zeitanfrage mit Abfahrtszeit $\tau_{\text{dep}}(c_i)$.

**Nachteile**

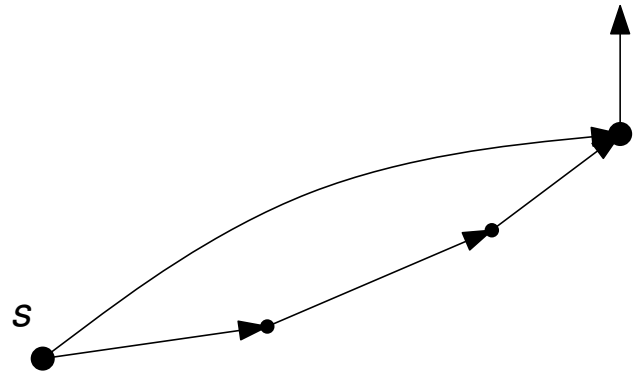
- Zu viele redundante Berechnungen
- Nicht jede Verbindung ab $S$ trägt zu $\text{dist}_S(v, \cdot)$ bei
  Langsame Züge für weite Reisen machen wenig Sinn

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 6:30<br>8:30 | 7:04<br>8:30 | 9:26<br>14:28 | 10:34<br>14:28 | 11:08<br>14:28 | 12:42<br>14:28 | 13:01<br>16:46 | 13:58<br>16:46 | 16:46<br>23:30 | 18:24<br>23:30 | 19:20<br>23:30 | 21:08<br>23:30 |

(“Connection reduction”)

# Self-Pruning

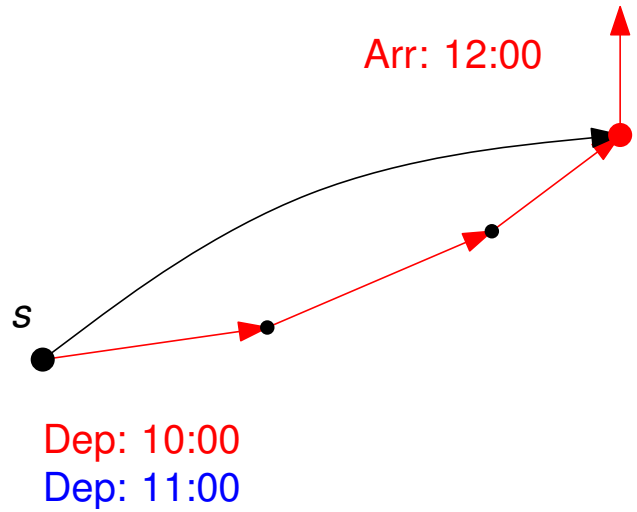
**Beobachtung:**
Verbindungen können sich dominieren.




Dep: 10:00
Dep: 11:00

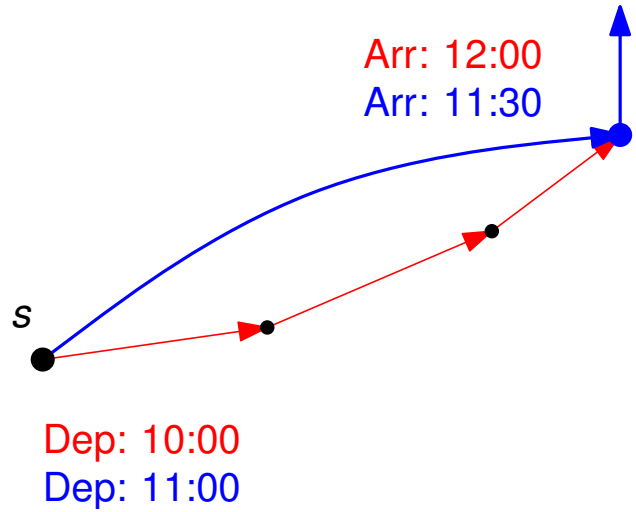
# Self-Pruning

**Beobachtung:**
Verbindungen können sich dominieren.



# Self-Pruning

**Beobachtung:**
Verbindungen können sich dominieren.

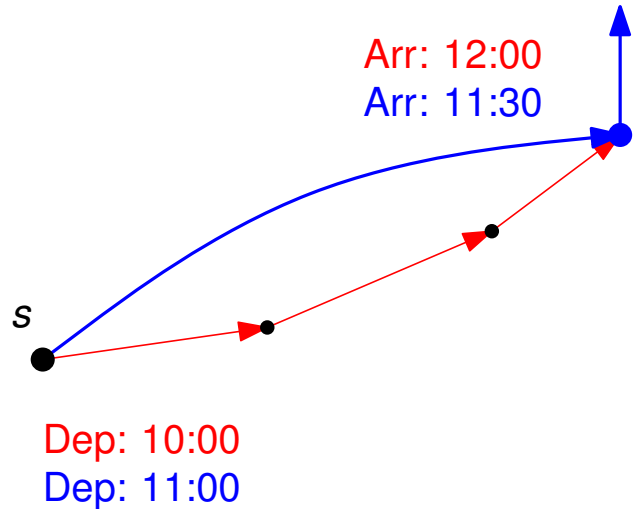


# Self-Pruning

**Beobachtung:**
Verbindungen können sich dominieren.

Einführung: **Self-Pruning** (SP):

1. Benutze eine gemeinsame Queue
2. Keys sind Ankunftszeiten
3. Sortiere Verb. $c_i$ an $S$ nach Abfahrtszeit




Beim Settlen von Knoten $v$ und Verb.-Index $i$:
Prüfe ob $v$ bereits gesettled mit Verbindung $j > i$; Dann **Prune** $i$ an $v$

# Self-Pruning II

## Integration von Self-Pruning (SP):

- Verwalte Label maxconn($v$) an jedem Knoten $v$
  Gibt maximale Verbindung an mit der $v$ gesettled wurde
- Update maxconn($v$) beim Settlen von $v$

# Self-Pruning II

## Integration von Self-Pruning (SP):

- Verwalte Label maxconn($v$) an jedem Knoten $v$
  Gibt maximale Verbindung an mit der $v$ gesettled wurde
- Update maxconn($v$) beim Settlen von $v$

Beim Settlen von Knoten $v$ und Verb.-Index $i$:
Prüfe ob $v$ bereits gesettled mit Verbindung $j > i$; Dann **Prune** $i$ an $v$

# Self-Pruning II

## Integration von Self-Pruning (SP):

- Verwalte Label maxconn($v$) an jedem Knoten $v$
  Gibt maximale Verbindung an mit der $v$ gesettled wurde
- Update maxconn($v$) beim Settlen von $v$

Beim Settlen von Knoten $v$ und Verb.-Index $i$:
Prüfe ob maxconn($v$) $> i$; Dann **Prune** $i$ an $v$

# Self-Pruning II

**Integration von Self-Pruning (SP):**

- Verwalte Label maxconn($v$) an jedem Knoten $v$
  Gibt maximale Verbindung an mit der $v$ gesettled wurde
- Update maxconn($v$) beim Settlen von $v$

Beim Settlen von Knoten $v$ und Verb.-Index $i$:
Prüfe ob maxconn($v$) $> i$; Dann **Prune** $i$ an $v$

**Wiederherstellung von Dijkstra's Label-Setting Eigenschaft pro Verbindung**

# Self-Pruning II

**Integration von Self-Pruning (SP):**

- Verwalte Label maxconn($v$) an jedem Knoten $v$
  Gibt maximale Verbindung an mit der $v$ gesettled wurde
- Update maxconn($v$) beim Settlen von $v$

Beim Settlen von Knoten $v$ und Verb.-Index $i$:
Prüfe ob maxconn($v$) $> i$; Dann **Prune** $i$ an $v$

**Wiederherstellung von Dijkstra's Label-Setting Eigenschaft pro Verbindung**

$\Rightarrow$ Self-Pruning Connection-Setting Algorithmus (SPCS)

# Parallelisierung: Idee

**Gegeben:**

Shared Memory Processing mit $p$ Cores



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

# Parallelisierung: Idee

**Gegeben:**
Shared Memory Processing mit $p$ Cores

**Idee:**
Verteile Verbindungen $c_i$ von $S$ auf verschiedene Threads

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** 6:30 | **1** 7:04 | **2** 9:26 | **3** 10:34 | **4** 11:08 | **5** 12:42 | **6** 13:01 | **7** 13:58 | **8** 16:46 | **9** 18:24 | **10** 19:20 | **11** 21:08 |
| Thread 0 | | | Thread 1 | | | Thread 2 | | | Thread 3 | | |

- Jeder Thread führt SPCS auf seiner Teilmenge der Verbindungen aus
- Ergebnisse werden im Anschluss zu $\text{dist}_S(v, \cdot)$ zusammengeführt
- Führe Connection Reduction auf gemergtem Ergebnis durch

# Stoppkriterium

**Dijkstra's Algorithmus:**
Breche die Suche ab, sobald $T$ abgearbeitet wurde.

# Stoppkriterium

**Dijkstra's Algorithmus:**
Breche die Suche ab, sobald $T$ abgearbeitet wurde.

kann adaptiert werden durch

**Parallel Self-Pruning Connection-Setting:**

- Verwalte globales Label $T_m := -\infty$

# Stoppkriterium

**Dijkstra's Algorithmus:**
Breche die Suche ab, sobald $T$ abgearbeitet wurde.

kann adaptiert werden durch

**Parallel Self-Pruning Connection-Setting:**

- Verwalte globales Label $T_m := -\infty$
- Wenn Verbindung $i$ an $T$ abgearbeitet wird, setze $T_m := \max\{T_m, i\}$

# Stoppkriterium

**Dijkstra's Algorithmus:**
Breche die Suche ab, sobald $T$ abgearbeitet wurde.

kann adaptiert werden durch

**Parallel Self-Pruning Connection-Setting:**

- Verwalte globales Label $T_m := -\infty$
- Wenn Verbindung $i$ an $T$ abgearbeitet wird, setze $T_m := \max\{T_m, i\}$
- Prune alle Verbindungen $j < T_m$ (an jedem Knoten)

# Stoppkriterium

**Dijkstra's Algorithmus:**
Breche die Suche ab, sobald $T$ abgearbeitet wurde.

kann adaptiert werden durch

**Parallel Self-Pruning Connection-Setting:**

- Verwalte globales Label $T_m := -\infty$
- Wenn Verbindung $i$ an $T$ abgearbeitet wird, setze $T_m := \max\{T_m, i\}$
- Prune alle Verbindungen $j < T_m$ (an jedem Knoten)
- Halte an, wenn Priority-Queue leer läuft

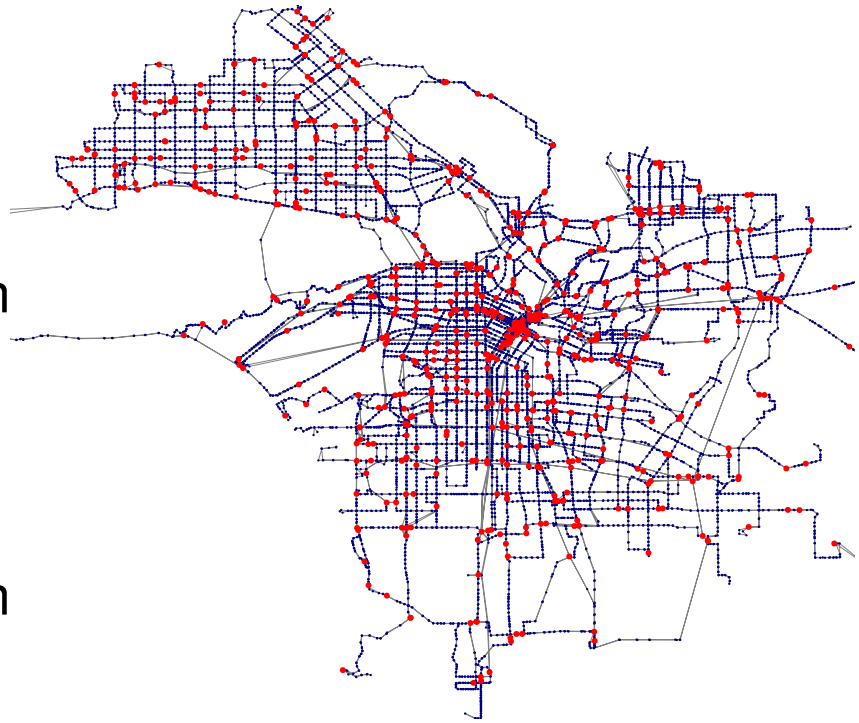
# Eingabe

Netzwerk von **Los Angeles**:

- 15 581 Stationen,
- 1 046 580 elem. Verbindungen

Zugnetz von **Europa**:

- 30 517 Stationen,
- 1 775 533 elem. Verbindungen



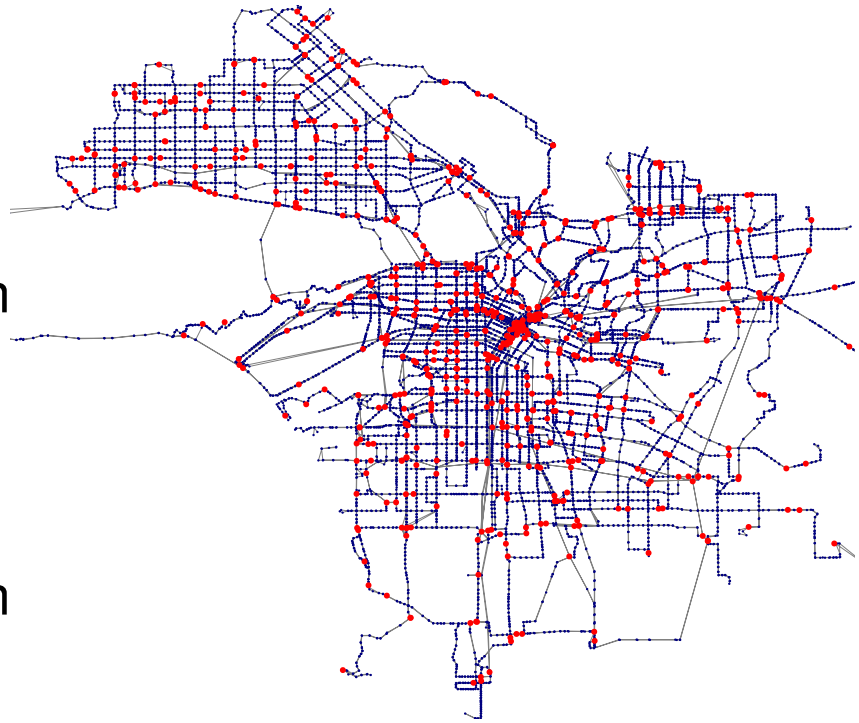
# Eingabe

Netzwerk von **Los Angeles**:

- 15 581 Stationen,
- 1 046 580 elem. Verbindungen

Zugnetz von **Europa**:

- 30 517 Stationen,
- 1 775 533 elem. Verbindungen



Auswertung durch 1 000 Anfragen wobei Start- und Zielbahnhöfe gleichverteilt zufällig gewählt.

# One-to-All Anfragen

| | $p$ | Los Angeles Settled Conns | Los Angeles Time [ms] | Los Angeles Spd Up | Los Angeles Std-Dev | Europe Settled Conns | Europe Time [ms] | Europe Spd Up | Europe Std-Dev |
|---|---|---|---|---|---|---|---|---|---|
| **PSPCS:** | 1 | 2.5 M | 1209.0 | 1.0 | — | 3.3 M | 2152.0 | 1.0 | — |
| EQUICONN | 2 | 2.5 M | 690.0 | 1.8 | 14.7 % | 3.1 M | 1054.2 | 2.0 | 16.1 % |
| | 4 | 2.5 M | 417.4 | 2.9 | 18.2 % | 3.4 M | 673.8 | 3.2 | 24.4 % |
| | 8 | 2.5 M | 267.7 | 4.5 | 20.0 % | 4.2 M | 510.9 | 4.2 | 23.8 % |
| **LC:** | 1 | 18.9 M | 1482.1 | — | — | 17.4 M | 2497.1 | — | — |

# One-to-All Anfragen

| | | Los Angeles | | | | Europe | | | |
|---|---|---|---|---|---|---|---|---|---|
| | $p$ | Settled Conns | Time [ms] | Spd Up | Std-Dev | Settled Conns | Time [ms] | Spd Up | Std-Dev |
| **PSPCS:** | 1 | 2.5 M | 1209.0 | 1.0 | — | 3.3 M | 2152.0 | 1.0 | — |
| EQUICONN | 2 | 2.5 M | 690.0 | 1.8 | 14.7 % | 3.1 M | 1054.2 | 2.0 | 16.1 % |
| | 4 | 2.5 M | 417.4 | 2.9 | 18.2 % | 3.4 M | 673.8 | 3.2 | 24.4 % |
| | 8 | 2.5 M | 267.7 | 4.5 | 20.0 % | 4.2 M | 510.9 | 4.2 | 23.8 % |
| **LC:** | 1 | 18.9 M | 1482.1 | — | — | 17.4 M | 2497.1 | — | — |

- PSPCS deutlich weniger Verbindungen als LC

# One-to-All Anfragen

| | | Los Angeles | | | | Europe | | | |
|---|---|---|---|---|---|---|---|---|---|
| | $p$ | Settled Conns | Time [ms] | Spd Up | Std-Dev | Settled Conns | Time [ms] | Spd Up | Std-Dev |
| **PSPCS:** | 1 | 2.5 M | 1209.0 | 1.0 | — | 3.3 M | 2152.0 | 1.0 | — |
| EQUICONN | 2 | 2.5 M | 690.0 | 1.8 | 14.7 % | 3.1 M | 1054.2 | 2.0 | 16.1 % |
| | 4 | 2.5 M | 417.4 | 2.9 | 18.2 % | 3.4 M | 673.8 | 3.2 | 24.4 % |
| | 8 | 2.5 M | 267.7 | 4.5 | 20.0 % | 4.2 M | 510.9 | 4.2 | 23.8 % |
| **LC:** | 1 | 18.9 M | 1482.1 | — | — | 17.4 M | 2497.1 | — | — |

- PSPCS deutlich weniger Verbindungen als LC
- PSPCS skaliert sehr gut mit zunehmender Anzahl Cores

# One-to-All Anfragen

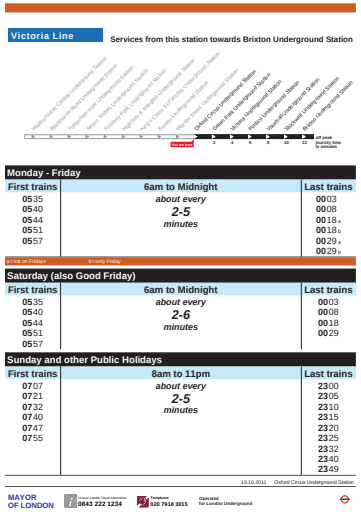
| | $p$ | Los Angeles Settled Conns | Los Angeles Time [ms] | Los Angeles Spd Up | Los Angeles Std-Dev | Europe Settled Conns | Europe Time [ms] | Europe Spd Up | Europe Std-Dev |
|---|---|---|---|---|---|---|---|---|---|
| **PSPCS:** | 1 | 2.5 M | 1209.0 | 1.0 | — | 3.3 M | 2152.0 | 1.0 | — |
| EQUICONN | 2 | 2.5 M | 690.0 | 1.8 | 14.7 % | 3.1 M | 1054.2 | 2.0 | 16.1 % |
| | 4 | 2.5 M | 417.4 | 2.9 | 18.2 % | 3.4 M | 673.8 | 3.2 | 24.4 % |
| | 8 | 2.5 M | 267.7 | 4.5 | 20.0 % | 4.2 M | 510.9 | 4.2 | 23.8 % |
| **LC:** | 1 | 18.9 M | 1482.1 | — | — | 17.4 M | 2497.1 | — | — |

- PSPCS deutlich weniger Verbindungen als LC
- PSPCS skaliert sehr gut mit zunehmender Anzahl Cores
- Einfache Partitionierung liefert bereits gute Ergebnisse

# Bis jetzt. . .

**Erinnerung:** Ein Fahrplan besteht aus

- Stops (Bahnhöfe, Bahnsteige, . . . ),
- Routen (Bus-, U-Bahn Linien, . . . ),
- Trips mit Abfahrt-/Ankunftszeiten,
- und Fußwegen zum Umsteigen.



Victoria Line — Services from this station towards Brixton Underground Station

Monday - Friday

| First trains | 6am to Midnight | Last trains |
|---|---|---|
| 0535 | about every | 0003 |
| 0540 | 2-5 | 0008 |
| 0544 | minutes | 0018 a |
| 0551 | | 0018 b |
| 0557 | | 0029 a |
| | | 0029 b |

Saturday (also Good Friday)

| First trains | 6am to Midnight | Last trains |
|---|---|---|
| 0535 | about every | 0003 |
| 0540 | 2-6 | 0008 |
| 0544 | minutes | 0018 |
| 0551 | | 0029 |
| 0557 | | |

Sunday and other Public Holidays

| First trains | 8am to 11pm | Last trains |
|---|---|---|
| 0707 | about every | 2300 |
| 0721 | 2-5 | 2305 |
| 0732 | minutes | 2310 |
| 0740 | | 2315 |
| 0747 | | 2320 |
| 0755 | | 2325 |
| | | 2332 |
| | | 2340 |
| | | 2349 |

13.10.2011 Oxford Circus Underground Station

MAYOR OF LONDON — 0843 222 1234 — 020 7918 3015 — Operated for London Underground

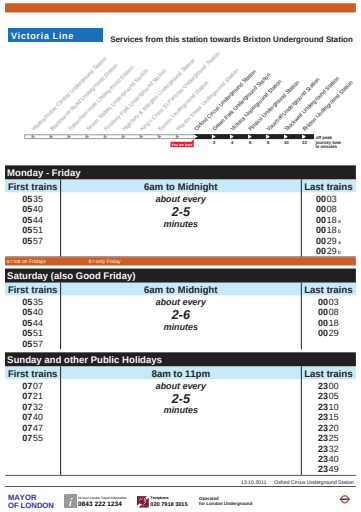


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

# Bis jetzt...

**Erinnerung:** Ein Fahrplan besteht aus

- Stops (Bahnhöfe, Bahnsteige, ...),
- Routen (Bus-, U-Bahn Linien, ...),
- Trips mit Abfahrt-/Ankunftszeiten,
- und Fußwegen zum Umsteigen.



Victoria Line — Services from this station towards Brixton Underground Station

Monday - Friday

| First trains | 6am to Midnight | Last trains |
|---|---|---|
| 0535 | about every | 0003 |
| 0540 | 2-5 | 0008 |
| 0544 | minutes | 0018 a |
| 0551 | | 0018 b |
| 0557 | | 0029 a |
| | | 0029 b |

Saturday (also Good Friday)

| First trains | 6am to Midnight | Last trains |
|---|---|---|
| 0535 | about every | 0003 |
| 0540 | 2-6 | 0008 |
| 0544 | minutes | 0018 |
| 0551 | | 0029 |
| 0557 | | |

Sunday and other Public Holidays

| First trains | 8am to 11pm | Last trains |
|---|---|---|
| 0707 | about every | 2300 |
| 0721 | 2-5 | 2305 |
| 0732 | minutes | 2310 |
| 0740 | | 2315 |
| 0747 | | 2320 |
| 0755 | | 2325 |
| | | 2332 |
| | | 2340 |
| | | 2349 |

13.10.2011 Oxford Circus Underground Station

MAYOR OF LONDON — 0843 222 1234 — 020 7918 3015 — Operated for London Underground

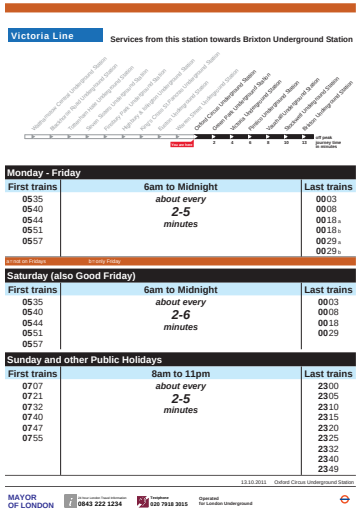
**Earliest Arrival Problem:**
Gegeben Stops $s$, $t$ und Abfahrtszeit $\tau$, berechne

- Route zu $t$ die an $s$ nicht früher als $\tau$ abfährt,
- und an $t$ frühstmöglich ankommt.

# Bis jetzt...

**Erinnerung:** Ein Fahrplan besteht aus

- Stops (Bahnhöfe, Bahnsteige, ...),
- Routen (Bus-, U-Bahn Linien, ...),
- Trips mit Abfahrt-/Ankunftszeiten,
- und Fußwegen zum Umsteigen.



Victoria Line — Services from this station towards Brixton Underground Station

Monday - Friday

| First trains | 6am to Midnight | Last trains |
|---|---|---|
| 0535 | about every | 0003 |
| 0540 | 2-5 | 0008 |
| 0544 | minutes | 0018 a |
| 0551 | | 0018 b |
| 0557 | | 0029 a |
| | | 0029 b |

Saturday (also Good Friday)

| First trains | 6am to Midnight | Last trains |
|---|---|---|
| 0535 | about every | 0003 |
| 0540 | 2-6 | 0008 |
| 0544 | minutes | 0018 |
| 0551 | | 0029 |
| 0557 | | |

Sunday and other Public Holidays

| First trains | 8am to 11pm | Last trains |
|---|---|---|
| 0707 | about every | 2300 |
| 0721 | 2-5 | 2305 |
| 0732 | minutes | 2310 |
| 0740 | | 2315 |
| 0747 | | 2320 |
| 0755 | | 2325 |
| | | 2332 |
| | | 2340 |
| | | 2349 |

13.10.2011 Oxford Circus Underground Station

MAYOR OF LONDON — 0843 222 1234 — 020 7918 3015 — Operated for London Underground

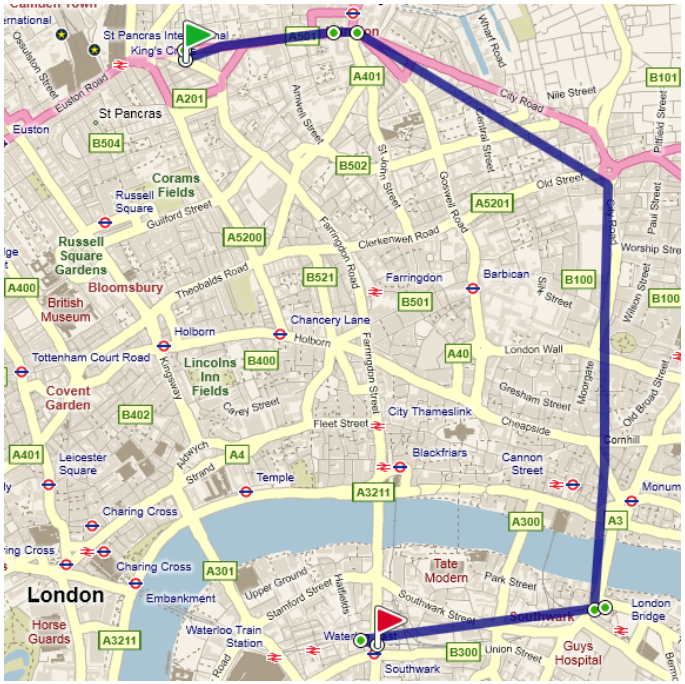
**Earliest Arrival Problem:**
Gegeben Stops $s$, $t$ und Abfahrtszeit $\tau$, berechne

- Route zu $t$ die an $s$ nicht früher als $\tau$ abfährt,
- und an $t$ frühstmöglich ankommt.

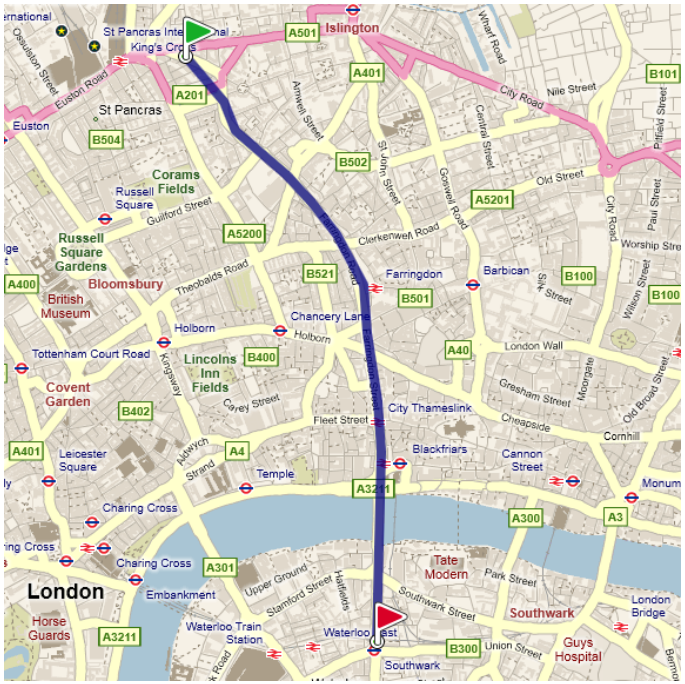
Reicht uns das?

# Einbeziehen von Umstiegen

Umstiege zu betrachten ist wichtig!



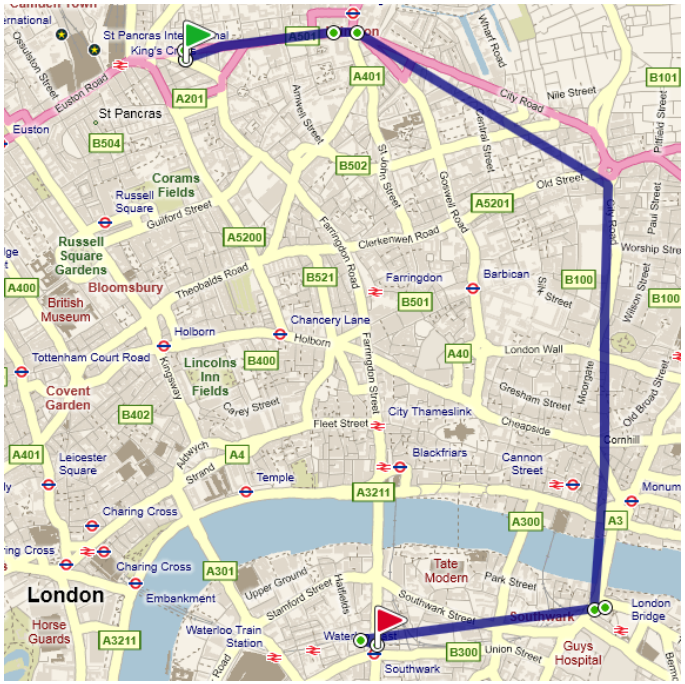

Ankunft 11:08 Uhr, 2 Umstiege



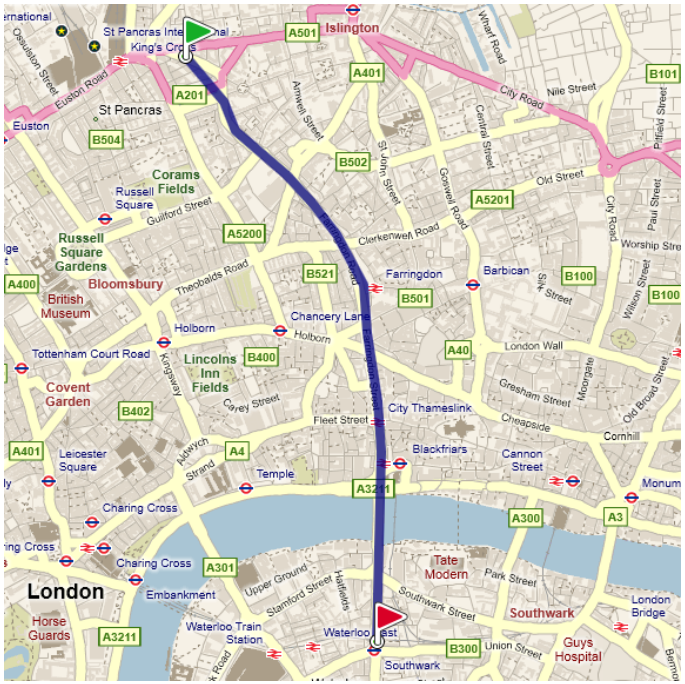

Ankunft 11:09 Uhr, 0 Umstiege

# Einbeziehen von Umstiegen

Umstiege zu betrachten ist wichtig!




Ankunft 11:08 Uhr, 2 Umstiege




Ankunft 11:09 Uhr, 0 Umstiege

**Idee:** Berechne „gute“ Routen für Ankunftszeit *und* Anzahl Umstiege.

# Pareto-Menge

## Definition (Pareto-Optimum)

Zu einer Menge $M$ von $n$-Tupeln heißt ein Tupel $m_i = (x_1, \ldots, x_n) \in M$ *Pareto-Optimum*, wenn es kein anderes $m_j \in M$ gibt, so dass $m_j$ in allen Werten besser als $m_i$ ist ($m_j$ *dominiert* $m_i$).

# Pareto-Menge

## Definition (Pareto-Optimum)

Zu einer Menge $M$ von $n$-Tupeln heißt ein Tupel $m_i = (x_1, \ldots, x_n) \in M$ *Pareto-Optimum*, wenn es kein anderes $m_j \in M$ gibt, so dass $m_j$ in allen Werten besser als $m_i$ ist ($m_j$ *dominiert* $m_i$).

Die Menge $M$ heißt *Pareto-Menge*, wenn alle $m \in M$ Pareto-optimal.

# Pareto-Menge

## Definition (Pareto-Optimum)

Zu einer Menge $M$ von $n$-Tupeln heißt ein Tupel $m_i = (x_1, \ldots, x_n) \in M$ *Pareto-Optimum*, wenn es kein anderes $m_j \in M$ gibt, so dass $m_j$ in allen Werten besser als $m_i$ ist ($m_j$ *dominiert* $m_i$).

Die Menge $M$ heißt *Pareto-Menge*, wenn alle $m \in M$ Pareto-optimal.

**Beispiel:** Betrachte Tupel aus Ankunftszeit und # Umstiege.

$M = \{(14{:}00 \text{ Uhr}, 5),\ (15{:}13 \text{ Uhr}, 3),\ (13{:}45 \text{ Uhr}, 4),\ (15{:}15 \text{ Uhr}, 0)\}$.

# Pareto-Menge

## Definition (Pareto-Optimum)

Zu einer Menge $M$ von $n$-Tupeln heißt ein Tupel $m_i = (x_1, \ldots, x_n) \in M$ *Pareto-Optimum*, wenn es kein anderes $m_j \in M$ gibt, so dass $m_j$ in allen Werten besser als $m_i$ ist ($m_j$ *dominiert* $m_i$).

Die Menge $M$ heißt *Pareto-Menge*, wenn alle $m \in M$ Pareto-optimal.

**Beispiel:** Betrachte Tupel aus Ankunftszeit und # Umstiege.

$M = \{(14{:}00 \text{ Uhr}, 5), (15{:}13 \text{ Uhr}, 3), (13{:}45 \text{ Uhr}, 4), (15{:}15 \text{ Uhr}, 0)\}$.

# Pareto-Menge

## Definition (Pareto-Optimum)

Zu einer Menge $M$ von $n$-Tupeln heißt ein Tupel $m_i = (x_1, \ldots, x_n) \in M$ *Pareto-Optimum*, wenn es kein anderes $m_j \in M$ gibt, so dass $m_j$ in allen Werten besser als $m_i$ ist ($m_j$ *dominiert* $m_i$).

Die Menge $M$ heißt *Pareto-Menge*, wenn alle $m \in M$ Pareto-optimal.

**Beispiel:** Betrachte Tupel aus Ankunftszeit und # Umstiege.

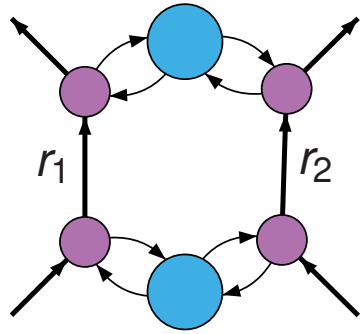
$M = \{(14{:}00 \text{ Uhr}, 5), (15{:}13 \text{ Uhr}, 3), (13{:}45 \text{ Uhr}, 4), (15{:}15 \text{ Uhr}, 0)\}$.

Wie effizient berechnen?

# Multi-Label-Correcting Ansatz

**Idee**

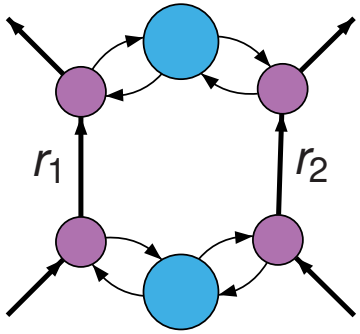
- Benutze (zeitabhängiges) Graphmodell
- Grundlage: Dijkstra's Algorithmus



# Multi-Label-Correcting Ansatz

**Idee**

- Benutze (zeitabhängiges) Graphmodell
- Grundlage: Dijkstra's Algorithmus




. . . aber. . .

- Label $\ell$ sind 2-Tupel aus Ankunftszeit und # Umstiege
- An jedem Knoten $u \in V$: Pareto-Menge $B_u$ von Labeln
- Priority Queue verwaltet Label statt Knoten
- Priorität ist Ankunftszeit
  Wieder: keine Totalordnung der Label $\Rightarrow$ Label-correcting Algo
- Dominanz von Labeln in $B_u$ on-the-fly

# Multi-Label-Correcting (MLC)

MLC($G = (V, E)$, $s$, $\tau$)

1 $B_u \leftarrow \{\}$ for each $u \in V$; $B_s \leftarrow \{(\tau, 0)\}$
2 $Q.clear()$, $Q.insert(s, (\tau, 0))$
3 **while** !$Q.empty()$ **do**
4  $u$ and $\ell = (\tau, tr) \leftarrow Q.deleteMin()$
5  **for** *all edges* $e = (u, v) \in E$ **do**
6  **if** *e is a transfer edge* **then** $tr' \leftarrow tr + 1$ **else** $tr' \leftarrow tr$
7  $\ell' \leftarrow (\tau + \text{len}(e, \tau), tr')$
8  **if** $\ell'$ *is not dominated by any* $\ell'' \in B_v$ **then**
9  $B_v.insert(\ell')$
10  Remove non-Pareto-optimal labels from $B_v$
11  $Q.insert(v, \ell')$

# Diskussion: MLC

**Diskussion:**

- Pareto-Mengen $B_u$ sind *dynamische* Datenstrukturen $\rightsquigarrow$ teuer!
- Sehr viele Queue-Operationen
- Testen der Dominanz in $\mathcal{O}(|B_u|)$ möglich
- Stoppkriterium?

# Diskussion: MLC

**Diskussion:**

- Pareto-Mengen $B_u$ sind *dynamische* Datenstrukturen $\rightsquigarrow$ teuer!
- Sehr viele Queue-Operationen
- Testen der Dominanz in $\mathcal{O}(|B_u|)$ möglich
- Stoppkriterium?

**Verbesserungen für MLC:**

- Jedes $B_u$ verwaltet bestes ungesettletes Label selbst
  $\Rightarrow$ Priority Queue auf Knoten statt Labeln
- Label-Forwarding:
  Wenn Kante keine Kosten hat, überspringe Queue
- Target-Pruning:
  An Knoten $u$, verwerfe Label $\ell'$, wenn $B_t$ dominiert $\ell'$

# Nächste Termine

Karlsruher Institut für Technologie

## Mittwoch, 23.4.2014

Montag, 28.4.2014
Mittwoch, 30.4.2014

# Literatur I



Daniel Delling, Bastian Katz, and Thomas Pajor.
Parallel computation of best connections in public transportation networks.
*ACM Journal of Experimental Algorithmics*, 17(4):4.1–4.26, July 2012.


Yann Disser, Matthias Müller–Hannemann, and Mathias Schnee.
Multi-criteria shortest paths in time-dependent train networks.
In *Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08)*, volume 5038 of *Lecture Notes in Computer Science*, pages 347–361. Springer, June 2008.


Daniel Delling, Thomas Pajor, and Dorothea Wagner.
Engineering time-expanded graphs for faster timetable information.
In *Robust and Online Large-Scale Optimization*, volume 5868 of *Lecture Notes in Computer Science*, pages 182–206. Springer, 2009.


Evangelia Pyrga, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis.
Efficient models for timetable information in public transportation systems.
*ACM Journal of Experimental Algorithmics*, 12(2.4):1–39, 2008.