

# Algorithmen für Routenplanung

16. Sitzung, Sommersemester 2012

Thomas Pajor | 25. Juni 2012

INSTITUT FÜR THEORETISCHE INFORMATIK · ALGORITHMIK · PROF. DR. DOROTHEA WAGNER



# Fahrplanauskunft



## Eingabe bei Straßennetzen

- Straßenkarte bestehend aus
- Kreuzungen
- Straßensegmenten
- Verschiedene Metriken (Reisezeit, Distanz, ...)
- Evtl. historische Reisezeitprofile
- Abbiegekosten, ...

## Eingabe bei Straßennetzen

- Straßenkarte bestehend aus
- Kreuzungen
- Straßensegmenten
- Verschiedene Metriken (Reisezeit, Distanz, ...)
- Evtl. historische Reisezeitprofile
- Abbiegekosten, ...

Was ist Eingabe bei der Fahrplanauskunft?

## Gegeben (Fahrplan):

- Menge  $\mathcal{B}$  von Bahnhöfen (Stops, Bahnsteigen, ...),
- Menge  $\mathcal{Z}$  von Zügen (Bussen, Trams, etc)
- Menge  $\mathcal{C}$  von elementaren Verbindungen
- Mindestumstiegszeiten transfer :  $\mathcal{B} \rightarrow \mathbb{N}$ .

## Gegeben (Fahrplan):

- Menge  $\mathcal{B}$  von Bahnhöfen (Stops, Bahnsteigen, ...),
- Menge  $\mathcal{Z}$  von Zügen (Bussen, Trams, etc)
- Menge  $\mathcal{C}$  von elementaren Verbindungen
- Mindestumstiegszeiten transfer :  $\mathcal{B} \rightarrow \mathbb{N}$ .

## 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  $\tau_{\text{arr}} \in \Pi$

## Gegeben (Fahrplan):

- Menge  $\mathcal{B}$  von Bahnhöfen (Stops, Bahnsteigen, ...),
- Menge  $\mathcal{Z}$  von Zügen (Bussen, Trams, etc)
- Menge  $\mathcal{C}$  von elementaren Verbindungen
- Mindestumstiegszeiten transfer :  $\mathcal{B} \rightarrow \mathbb{N}$ .

## 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.

## Trips

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

## 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

## Beispiel für einen Fahrplan

...

|                           |                 |        |        |
|---------------------------|-----------------|--------|--------|
| (IR 2269, Karlsruhe Hbf,  | Pforzheim Hbf,  | 10:05, | 10:23) |
| (IR 2269, Pforzheim Hbf,  | Mühlacker,      | 10:25, | 10:33) |
| (IR 2269, Mühlacker,      | 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,  | München Hbf,  | 12:49, | 13:21) |

...

## Beispiel für einen Fahrplan

...

(IR 2269, Karlsruhe Hbf, Pforzheim Hbf, 10:05, 10:23)

(IR 2269, Pforzheim Hbf, Mühlacker, 10:25, 10:33)

(IR 2269, Mühlacker, 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, München Hbf, 12:49, 13:21)

...

**Frage:** Wie Fahrplan modellieren?

## Ausgabe bei der Fahrplanauskunft

- Sequenz von (Teil-)Trips aus dem Fahrplan
- Beginnend bei Startstop
- Ended bei Zielstop
- (Möglicherweise mit Fußwegen dazwischen)

## Ausgabe bei der Fahrplanauskunft

- Sequenz von (Teil-)Trips aus dem Fahrplan
- Beginnend bei Startstop
- Ended bei Zielstop
- (Möglicherweise mit Fußwegen dazwischen)

## Reiseroute ist konsistent wenn

- Erster Stop von Trip  $t_{i+1}$  entspricht letztem Stop von Trip  $t_i$
- Abfahrt von  $t_{i+1}$  ist *nach* Ankunft von  $t_i$

## Zwei grundlegende Ansätze

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

## Zwei grundlegende Ansätze

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

Jetzt ersteres, später zweiteres.

## 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
- Bekannte Techniken (evtl. leicht) übertragbar

## Modellierung:

- Für jeden Bahnhof  $S \in \mathcal{B}$ : Ein Knoten
- Kante  $(S_i, S_j)$  gdw. ex. elementare Verbindung von  $S_i$  nach  $S_j$
- Kantengewichte: Lower-Bound der Reisezeit zw. Bahnhöfen



## Modellierung:

- Für jeden Bahnhof  $S \in \mathcal{B}$ : Ein Knoten
- Kante  $(S_i, S_j)$  gdw. ex. elementare Verbindung von  $S_i$  nach  $S_j$
- Kantengewichte: Lower-Bound der Reisezeit zw. Bahnhöfen



## Nachteile:

- Unrealistische Anfragen (keine Zeitabhängigkeit)
- Mit Zeitabhängigkeit: Unrealistische Umstiege

## 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)

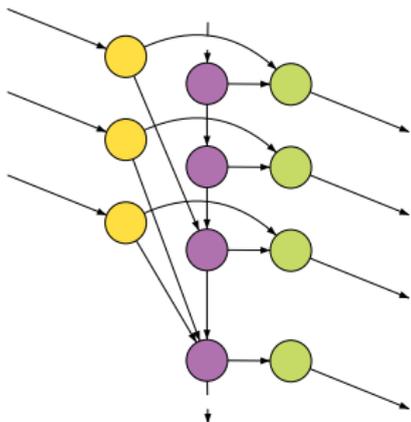
## 1. Zeitexpandiert

- 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

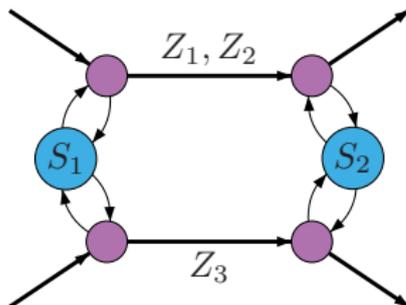
- Zeitabhängigkeit an den Kanten
- Knoten entsprechen Bahnhöfen
- Kante  $\Leftrightarrow$  Zug verbindet Bahnhöfe
  - Transferzeiten?
- + Kleiner Graph
- Zeitabhängige Routenplanung

## 1. Zeitexpandiert



- Arrival-, Transfer- und Departure-Ereignisse
- Für jeden Zug
- Kantengewicht = Zeitdifferenz

## 2. Zeitabhängig



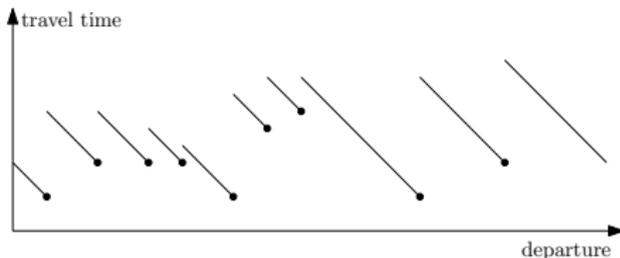
- Partitioniere Züge in Routen
- Pro Bahnhof: Stationsknoten
- Pro Route: Route-Knoten
- Routenkanten: Mehrere Züge
- Stationskanten: Transferzeit

## Eingabe:

- Reisezeit zu bestimmten Zeitpunkten (Fahrzeiten der Züge)
- Jeden Tag verschieden

## Somit:

- Periodische stückweise lineare Funktionen
- Definiert durch Stützpunkte
- Wartezeit zur nächsten Verbindung + Reisezeit



## Definition

Sei  $f : \mathbb{R}_0^+ \rightarrow \mathbb{R}_0^+$  eine 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)
- ⇒ Sicherstellen, dass Funktionen FIFO-Eigenschaft erfüllen.

## Funktion gegeben durch:

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

## 3 Operationen notwendig:

- Auswertung
- Linken  $\oplus$
- Minimumsbildung

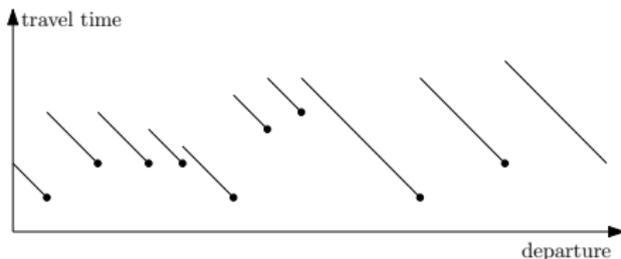
## Beobachtung:

- Unterschiedlich für Straße und Schiene!

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

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

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



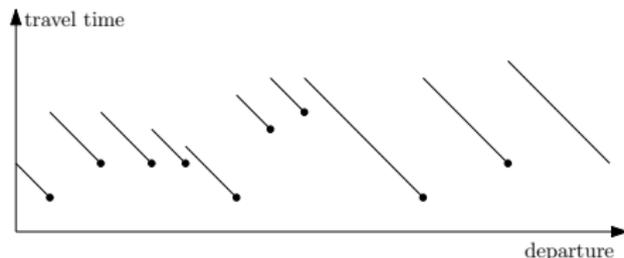
## Evaluation von $f(\tau)$ :

- 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  $\frac{\tau}{\bar{t}} \cdot |I|$



## Definition

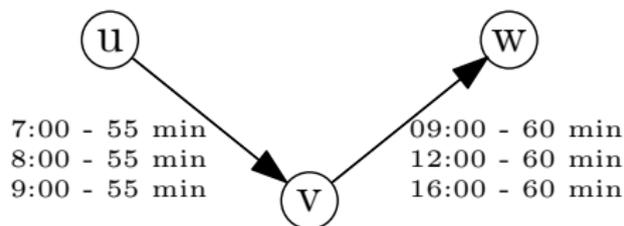
Seien  $f : \mathbb{R}_0^+ \rightarrow \mathbb{R}_0^+$  und  $g : \mathbb{R}_0^+ \rightarrow \mathbb{R}_0^+$  zwei Funktionen die die FIFO-Eigenschaft erfüllen. Die Linkoperation  $f \oplus g$  ist dann definiert durch

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

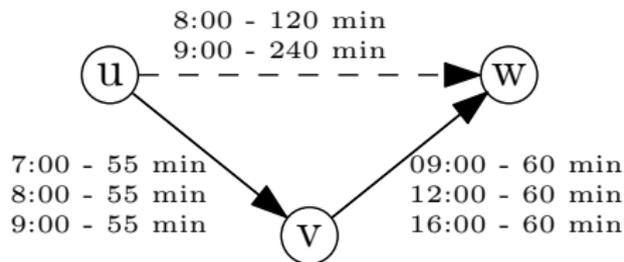
**Oder**

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

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

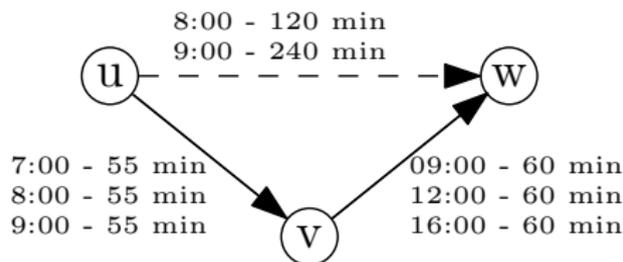


## Linken zweier Funktionen $f$ und $g$



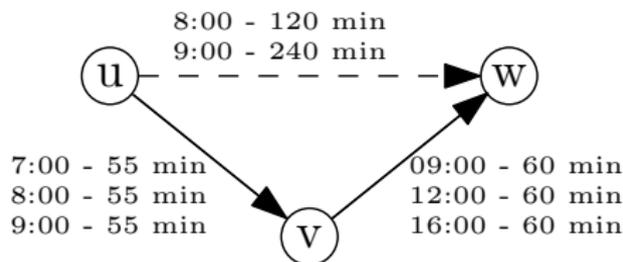
## Linken zweier Funktionen $f$ und $g$

- 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



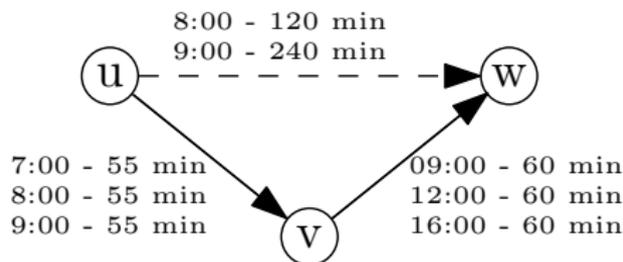
## Linken zweier Funktionen $f$ und $g$

- 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



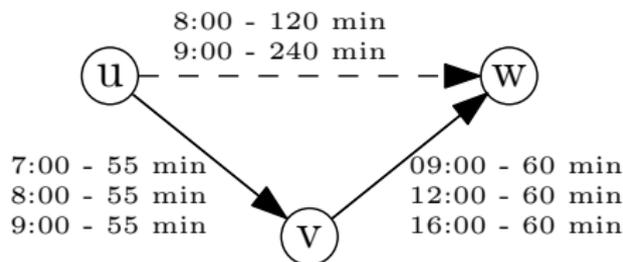
## Linken zweier Funktionen $f$ und $g$

- 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  $t_i^f$



## 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  $t_i^f$
- Wieder Sweep-Algorithmus



## Laufzeit

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

## Laufzeit

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

## Speicherverbrauch

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

## Laufzeit

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

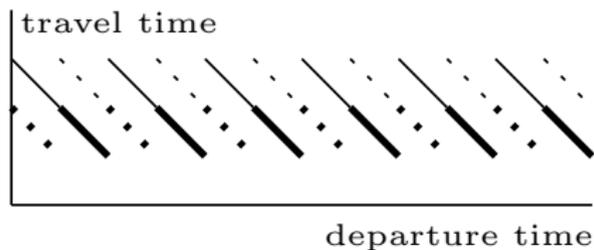
## Speicherverbrauch

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

## Somit:

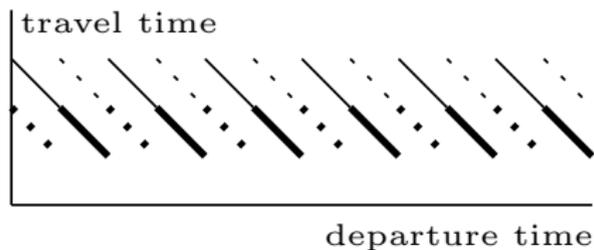
- Deutlich gutmütiger als Straßengraph-Funktionen

## Minimum zweier Funktionen $f$ und $g$



## 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 Schnittpunkte möglich(!)

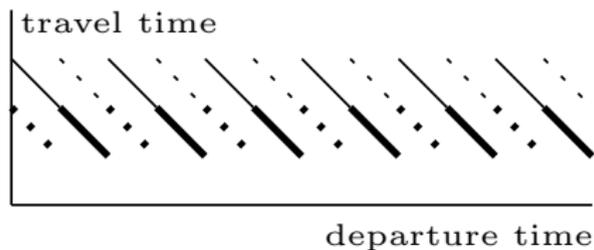


## Minimum zweier Funktionen $f$ und $g$

- 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 Schnittpunkte möglich(!)

## Vorgehen:

- Linearer Sweep



## Laufzeit

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

## Laufzeit

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

## Speicherverbrauch

- Keine Schnittpunkte
- ⇒ Minimum-Funktion kann maximal  $|I^f| + |I^g|$  Interpolationspunkte enthalten

# Schiene vs. Straße

## Laufzeit Operationen

- gleich für beide
- $\mathcal{O}(\log |I|)$  für Auswertung
- $\mathcal{O}(|I^f| + |I^g|)$  für Linken und Minimum

# Schiene vs. Straße

## Laufzeit Operationen

- gleich für beide
- $\mathcal{O}(\log |I|)$  für Auswertung
- $\mathcal{O}(|I^f| + |I^g|)$  für Linken und Minimum

## Speicherverbrauch

- Public Transport deutlich geringer
- Link:

$$|I^{f \oplus g}| \leq \min\{|I^f|, |I^g|\} \quad \text{vs.} \quad |I^{f \oplus g}| \approx |I^f| + |I^g|$$

- Merge:

$$|I^{\min\{f,g\}}| \leq |I^f| + |I^g| \quad \text{vs. eventuell} \quad |I^{\min\{f,g\}}| > (|I^f| + |I^g|)$$

# Schiene vs. Straße

## Laufzeit Operationen

- gleich für beide
- $\mathcal{O}(\log |I|)$  für Auswertung
- $\mathcal{O}(|I^f| + |I^g|)$  für Linken und Minimum

## Speicherverbrauch

- Public Transport deutlich geringer
- Link:

$$|I^{f \oplus g}| \leq \min\{|I^f|, |I^g|\} \quad \text{vs.} \quad |I^{f \oplus g}| \approx |I^f| + |I^g|$$

- Merge:

$$|I^{\min\{f,g\}}| \leq |I^f| + |I^g| \quad \text{vs. eventuell} \quad |I^{\min\{f,g\}}| > (|I^f| + |I^g|)$$

## Profilsuchen

- Somit in Public Transport Netzen wahrscheinlich schneller

# Zeit-Anfragen

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

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

## 1. Zeitexpandiert

### Startknoten:

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

### Zielknoten:

- Im Voraus unbekannt!
- Stoppkriterium: Erster gesetzter 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 Voraus unbekannt

---

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

---

```
1  $d_\tau[s] = 0$ 
2  $Q.clear(), Q.add(s, 0)$ 
3 while ! $Q.empty()$  do
4    $u \leftarrow Q.deleteMin()$ 
5   for all edges  $e = (u, v) \in E$  do
6     if  $d_\tau[u] + \text{len}(e, \tau + d_\tau[u]) < d_\tau[v]$  then
7        $d_\tau[v] \leftarrow d_\tau[u] + \text{len}(e, \tau + d_\tau[u])$ 
8        $p_\tau[v] \leftarrow u$ 
9       if  $v \in Q$  then  $Q.decreaseKey(v, d_\tau[v])$ 
10      else  $Q.insert(v, d_\tau[v])$ 
```

---

## Beobachtung:

- Nur ein Unterschied zu Dijkstra
- Auswertung der Kanten

## 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)

## 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).

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

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

## 1. Zeitexpandiert

?

## 2. Zeitabhängig

### Startknoten:

- Bahnhofsknoten  $S$

### Zielknoten:

- Bahnhofsknoten  $T$

### Anfrage:

- Label-Correcting Algorithmus von  $S$

---

**Algorithm 2:** Profile-Search( $G = (V, E), s$ )

---

```
1  $d_*[s] = 0$ 
2  $Q.clear(), Q.add(s, 0)$ 
3 while ! $Q.empty()$  do
4    $u \leftarrow Q.deleteMin()$ 
5   for all edges  $e = (u, v) \in E$  do
6     if  $d_*[u] \oplus \text{len}(e) \not\leq d_*[v]$  then
7        $d_*[v] \leftarrow \min(d_*[u] \oplus \text{len}(e), d_*[v])$ 
8       if  $v \in Q$  then  $Q.decreaseKey(v, \underline{d}[v])$ 
9       else  $Q.insert(v, \underline{d}[v])$ 
```

---

## Beobachtungen:

- Operationen auf Funktionen
- Priorität im Prinzip frei wählbar  
( $d[u]$  ist das Minimum der Funktion  $d_*[u]$ )
- Knoten können mehrfach besucht werden  $\Rightarrow$  label-correcting



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.

## Montag, 2.7.2012

Montag, 9.7.2012

Montag, 16.7.2012 (?)