# Algorithmen für Routenplanung – Vorlesung 9

Daniel Delling

Lehrstuhl für Algorithmik I
Institut für theoretische Informatik
Universität Karlsruhe (TH)
Forschungsuniversität · gegründet 1825



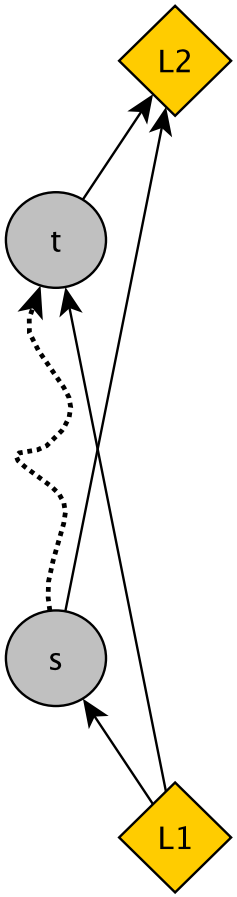
# *Letztes Mal: Dynamische Szenarien*


**Szenario:**

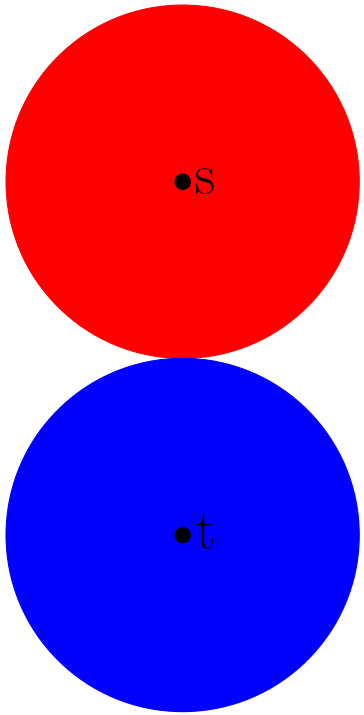
- Unfall auf einer Straße
- Reisezeit ändert sich auf dieser Straße
- berechne schnellsten Weg bezüglich der aktualisierten Reisezeiten

# Vier Bausteine

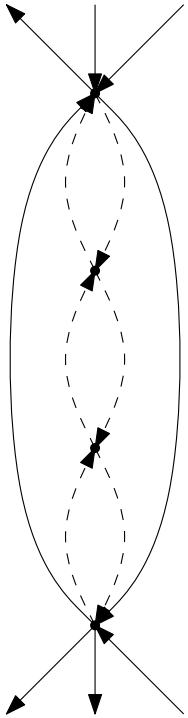
**Landmarken**




**Bidirektionale Suche**




**Kontraktion**



**Arc-Flags**

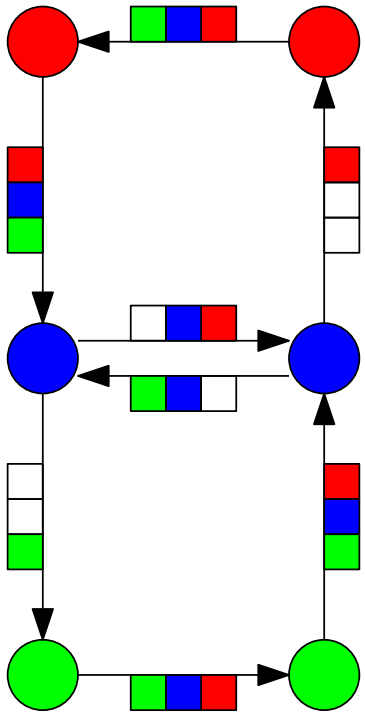
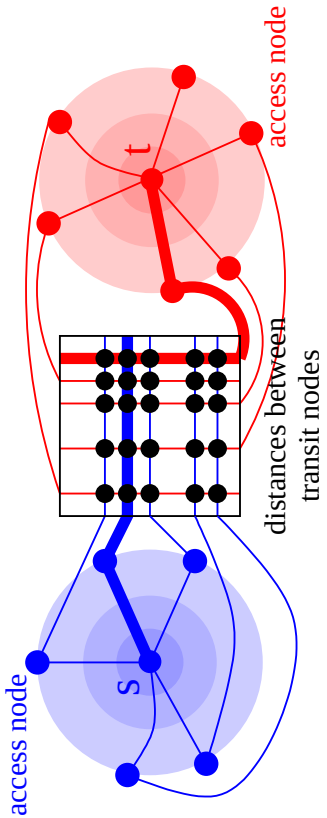


**Table-Lookups**



# Thema Heute: Zeitabhängiges Szenario


**Szenario:**

- Historische Daten für Verkehrssituation verfügbar
- Verkehrssituation vorhersagbar
- berechne schnellsten Weg bezüglich der erwarteten Verkehrssituation (zu einem gegebenen Startzeitpunkt)

**Beobachtung:**

- Kein konzeptioneller Unterschied mehr zu Public Transport
- Somit (eventuell) Techniken übertragbar

# *Herausforderung*

**Hauptproblem:**

- Kürzester Weg hängt von Abfahrtszeitpunkt ab
- Eingabegröße steigt massiv an

**Vorgehen:**

- Modellierung
- Anpassung Dijkstra
- Anpassung der Basismodule

**Heute:**

- Modellierung und Dijkstra

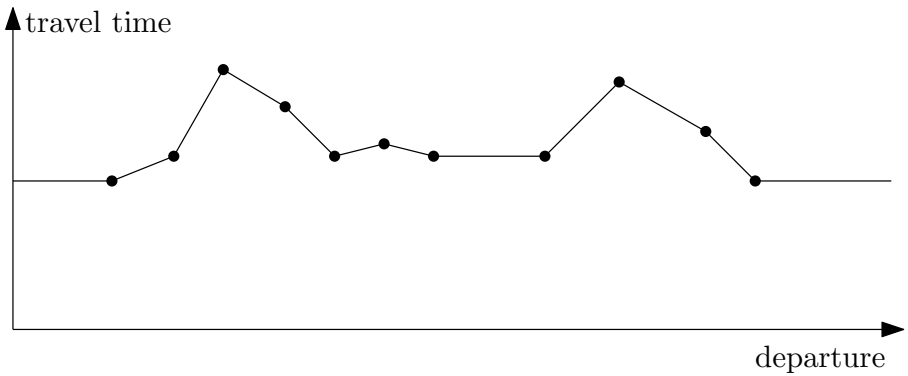
# Zeitabhängige Straßennetzwerke

**Eingabe:**

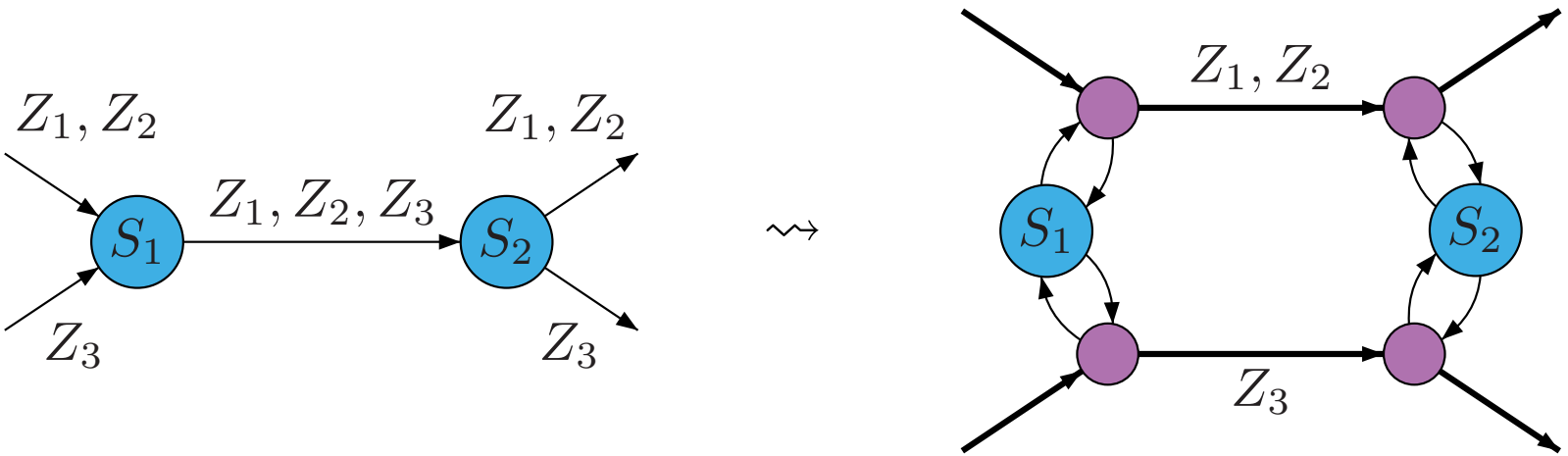
- Durchschnittliche Reisezeit zu bestimmten Zeitpunkten
- Jeden Wochentag verschieden
- Sonderfälle: Urlaubszeit

**Somit:**

- periodische stückweise lineare funktionen
- definiert durch Stützpunkte
- interpoliere linear zwischen Stützpunkten



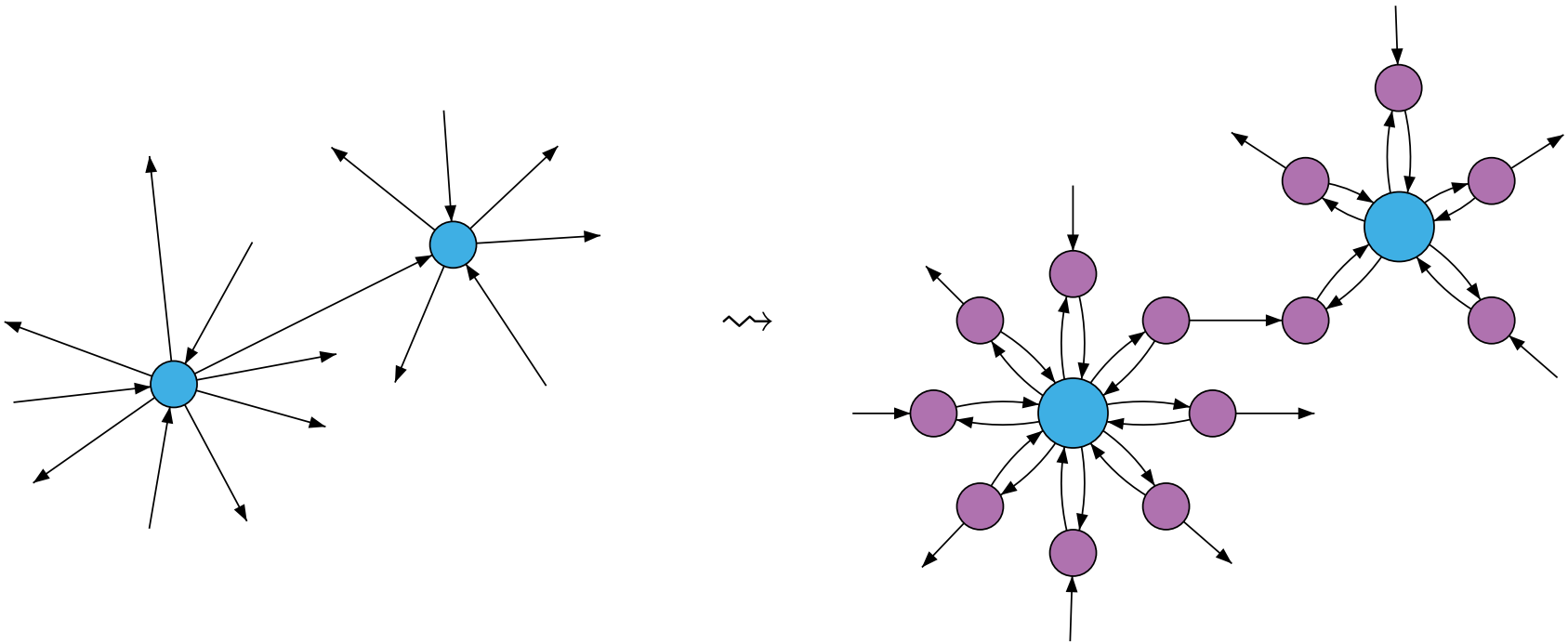
# Eisenbahn-Netzwerke




- Teile Züge $Z$ in Routen ein
- Für alle Routen, die eine Station bedienen, ein Extra-Knoten

# *Flugnetzwerke*

Nutzung des Eisenbahnansatzes nicht sinnvoll

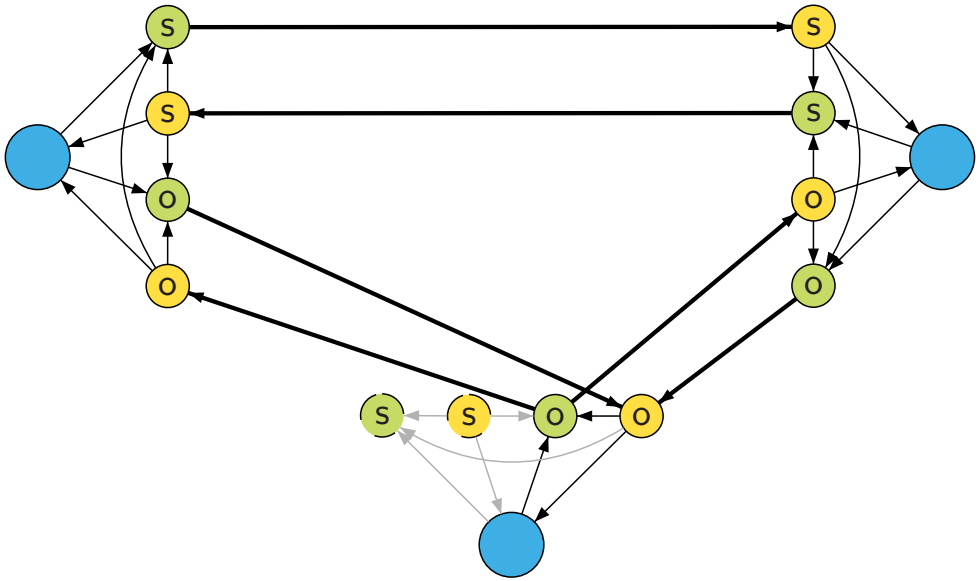


Graphen werden zu groß.

# *Flugnetzwerke*

**Modellierung:**

- Check-in,
- check-out,
- transfer,
- zwischen Allianzen.




⇒ Zwei Knoten pro Allianz (Abflug und Ankunftsknoten)

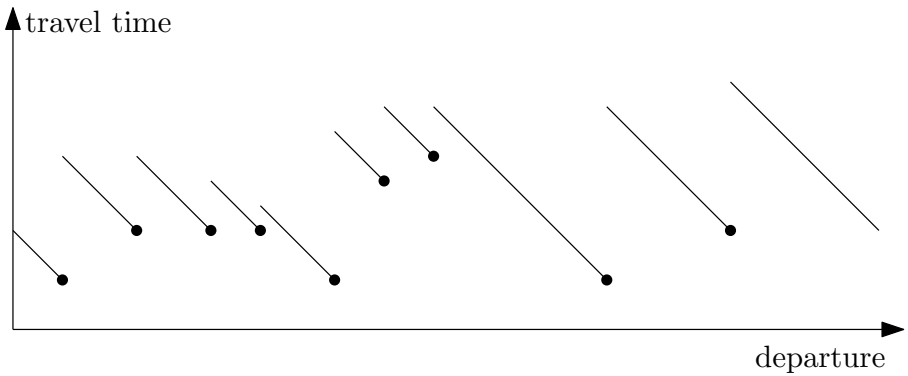
# Public Transport Funktionen

**Eingabe:**

- Reisezeit zu bestimmten Zeitpunkten
- Jeden Tag verschieden

**Somit:**

- periodische stückweise lineare funktionen
- definiert durch Stützpunkte
- Wartezeit zur nächsten Verbindung plus Reisezeit



# *Diskussion*

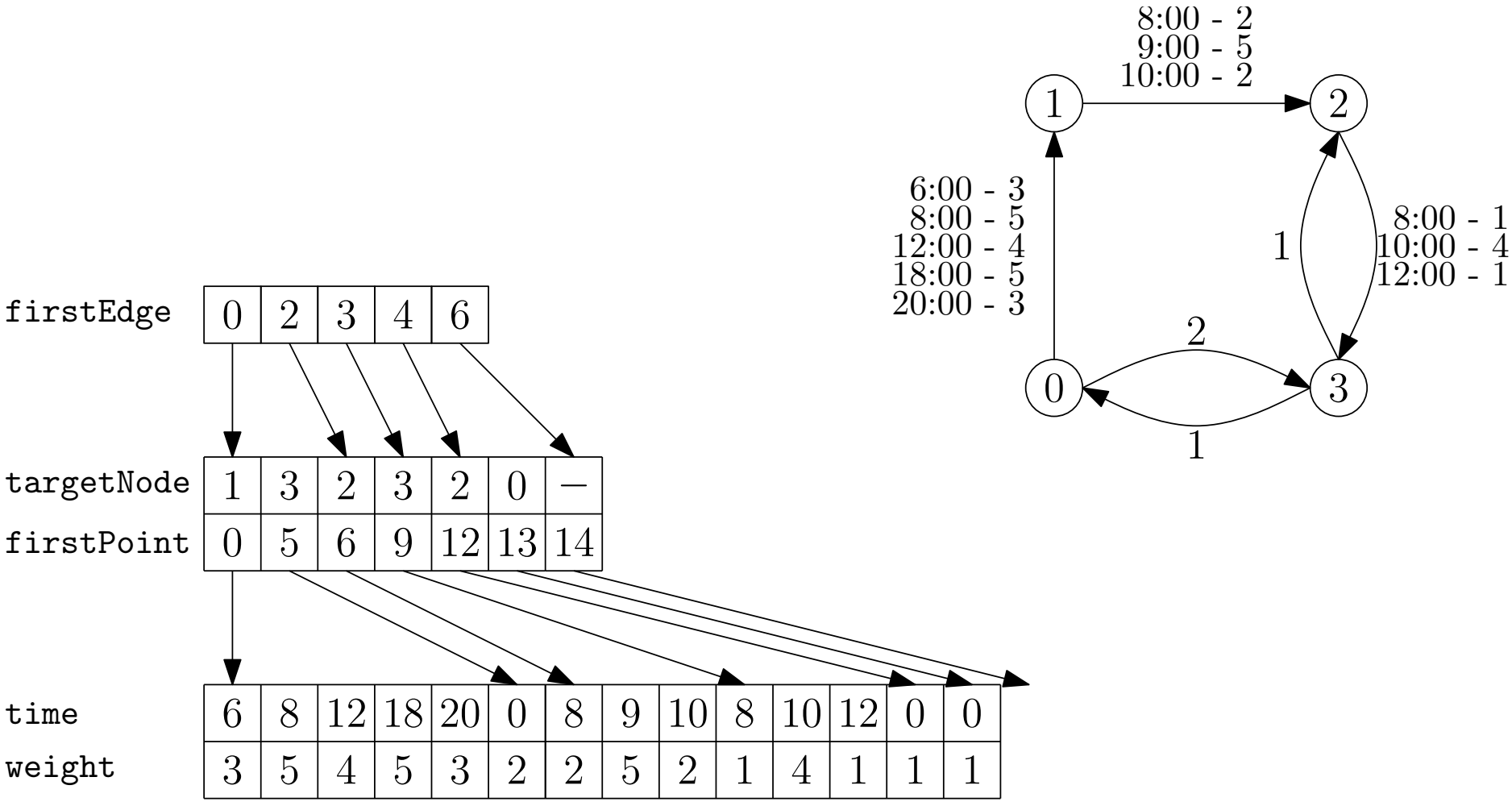
**Eigenschaften:**

- Topologie ändert sich nicht
- Kanten gemischt zeitabhängig und konstant
- variable (!) Anzahl Interpolationspunkte pro Kante

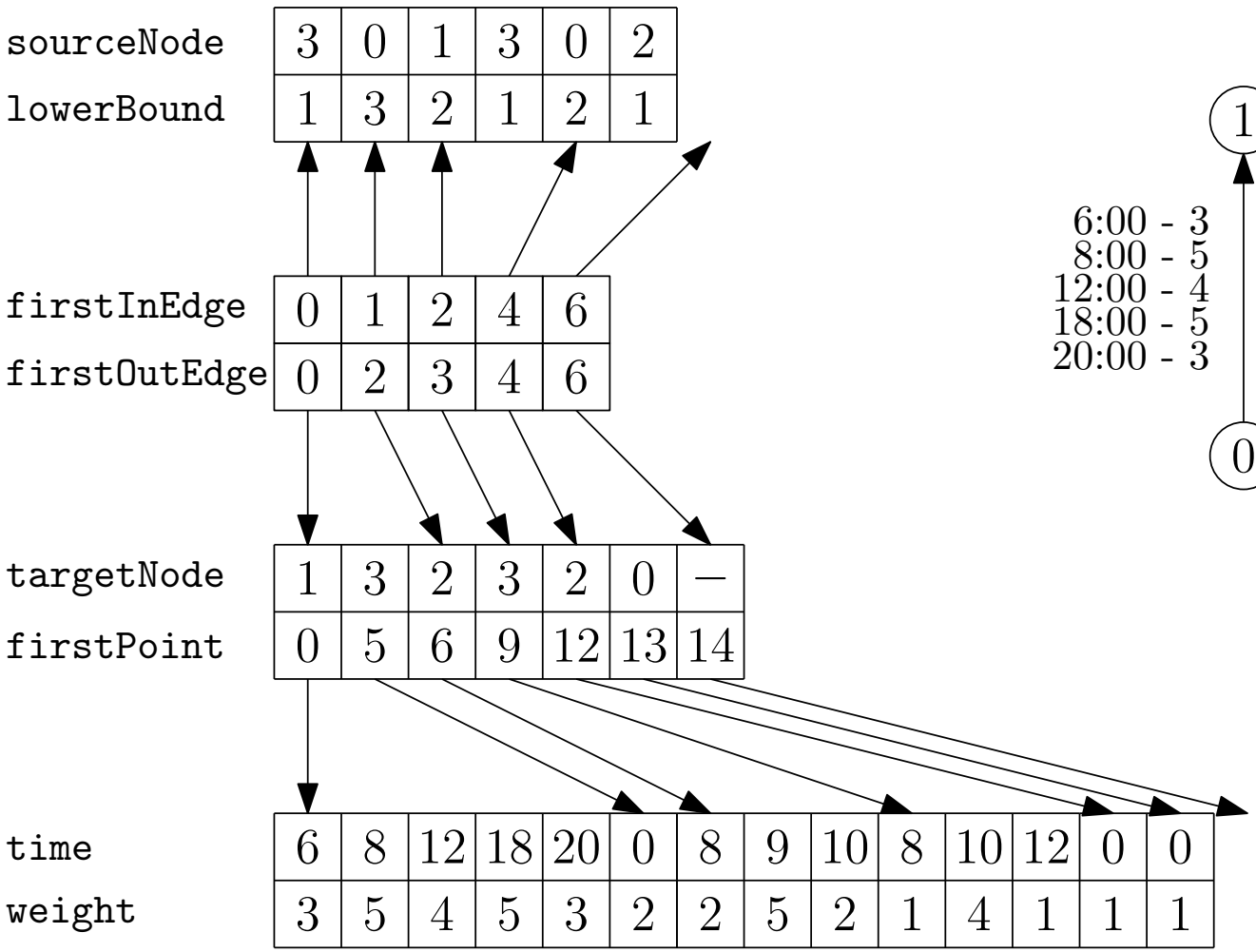
**Beobachtungen:**

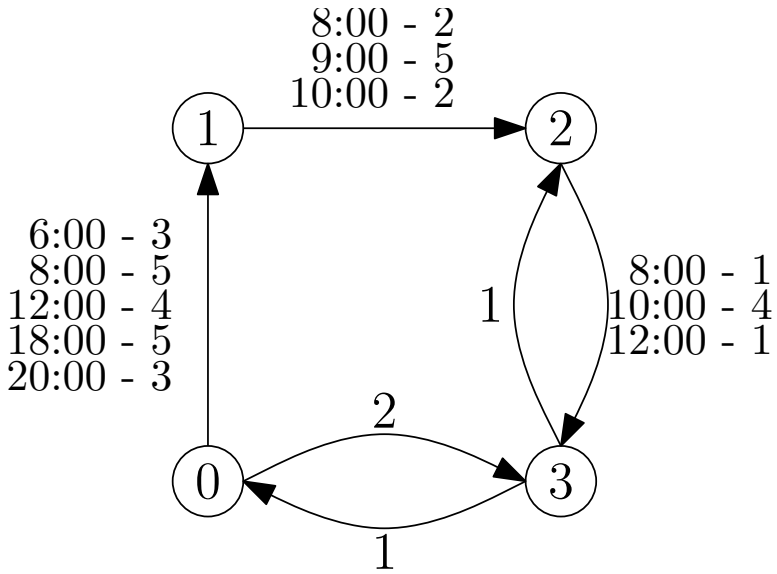
- FIFO gilt auf allen Kanten
- später wichtig

# Datenstruktur



# Datenstruktur



# *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

TIME-DIJKSTRA($G = (V, E), s, \tau$)

```
1  d_τ[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          tmp ← d_τ[u] + len(e, τ + d_τ[u])
7          if tmp < d_τ[v] then
8              d_τ[v] ← tmp
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

**non-FIFO:**

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

# Profil-Anfragen

**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          tmp ← d_*[u] ⊕ len(e)
7          if tmp ≱ d_*[v] then
8              d_*[v] ← min{tmp, d_*[v]}
9              if v ∈ Q then Q.decreaseKey(v, d[v])
10             else Q.insert(v, d[v])
```

Daniel Delling – Algorithmen für Routenplanung – Vorlesung 9
Lehrstuhl für Algorithmik I
Institut für theoretische Informatik


# *Diskussion Profile-Anfragen*

**Beobachtungen:**

- Operationen auf Funktionen
- priorität frei wählbar
- Knoten können mehrfach besucht werden $\Rightarrow$ label-correcting

**somit:**

- wie linken?
- wie Minimum bilden (mergen)?

# Operationen

**Funktion gegeben durch:**

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

**3 Operationen:**

- Auswertung
- Link
- Minimumsbildung

**Beobachtung:**

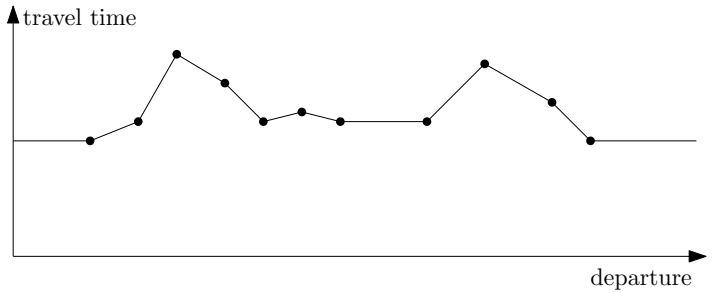
- unterschiedlich für Straße und Schiene



# Straße: Auswertung

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

- suche Punkte mit $t_i \geq \tau$ und $t_{i+1} \leq \tau$
- dann Evaluation durch




$$f(\tau) = w_i + (\tau - t_i) * \frac{w_{i+1} - w_i}{t_{i+1} - t_i}$$

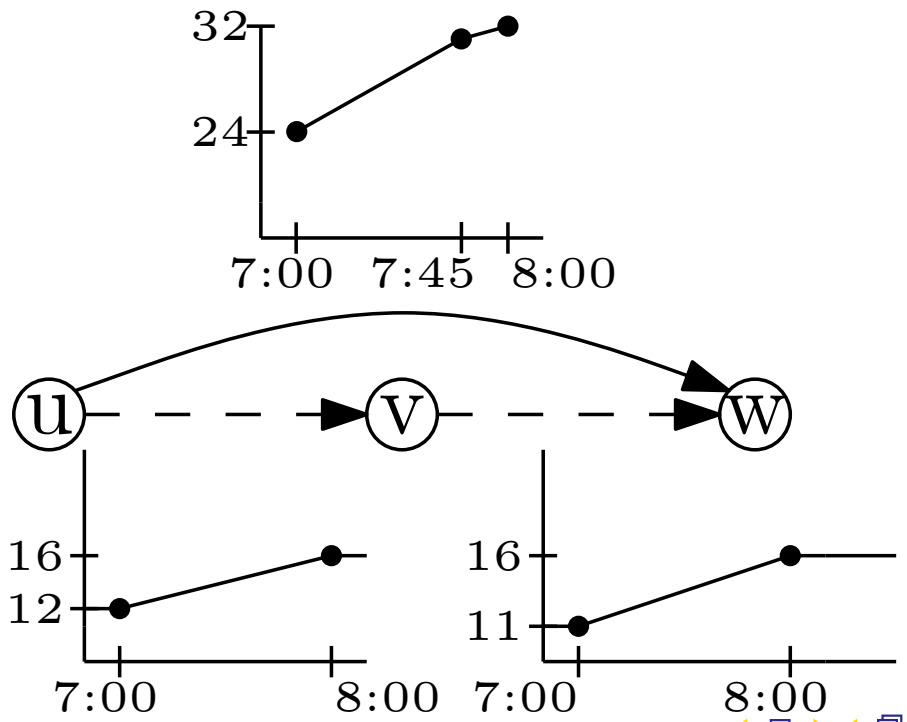
**Problem:**

- finden von $t_i$ und $t_{i+1}$
- theoretisch:
  - lineare Suche: $O(|I|)$
  - binäre Suche: $O(\log_2 |I|)$
- praktisch:
  - $|I| < 30$: lineare Suche
  - sonst: lineare Suche mit Startpunkt $\tau/\Pi * |I|$

# Straße: Link

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

- $f \oplus g$ enthält auf jeden Fall $\{(t_1^f, w_1^f + g(t_1^f + w_1^f)), \ldots, (t_l^f, w_l^f + g(t_l^f + w_l^f))\}$
- zusätzliche Interpolationspunkte an $t_j^{-1}$ mit $f(t_j^{-1}) + t_j^{-1} = t_j^g$
- füge $(t_j^{-1}, f(t_j^{-1}) + w_j^g)$ für alle Punkte von $g$ zu $f \oplus g$
- Durch linearen sweeping Algorithmus implementierbar



# Straße: Diskussion Link

**Laufzeit**

- sweep Algorithmus
- $O(|I^f| + |I^g|)$
- zeitunabhängig: $O(1)$

**Speicherverbrauch**

- gelinkte Funktion hat $\approx |I^f| + |I^g|$ Interpolationspunkte

**Problem:**

- während Profilsuche kann ein Pfad mehreren Tausend Kanten entsprechen...
- Shortcuts.....
- es kommt noch schlimmer....

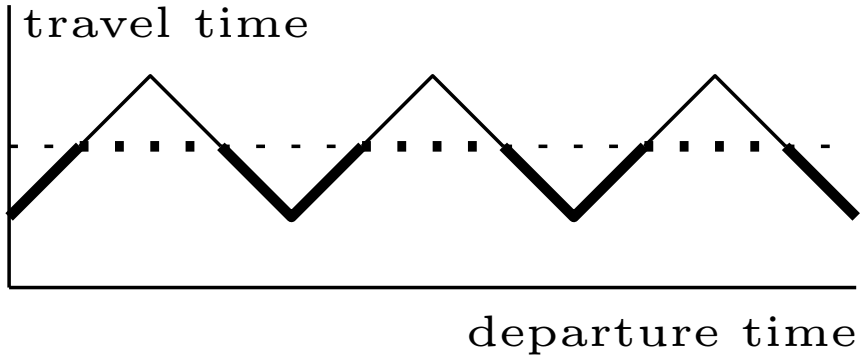
# Straße: Merge

**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 $w_j^g < f(t_j^g)$
- Schnittepunkte müssen ebenfalls eingefügt werden

**Vorgehen:**

- linearer sweep
- evaluiere, welcher Abschnitt oben
- checke ob Schnittpunkt existiert



# Straße: Diskussion Merge

**Laufzeit**

- sweep Algorithmus
- $O(|I^f| + |I^g|)$
- zeitunabhängig: $O(1)$

**Speicherverbrauch**

- minimum-Funktion kann mehr als $|I^f| + |I^g|$ Interpolationspunkte haben

**Problem:**

- während Profilsuche werden Funktionen gemergt
- Laufzeit der Profilsuchen wird durch diese Operationen dominiert

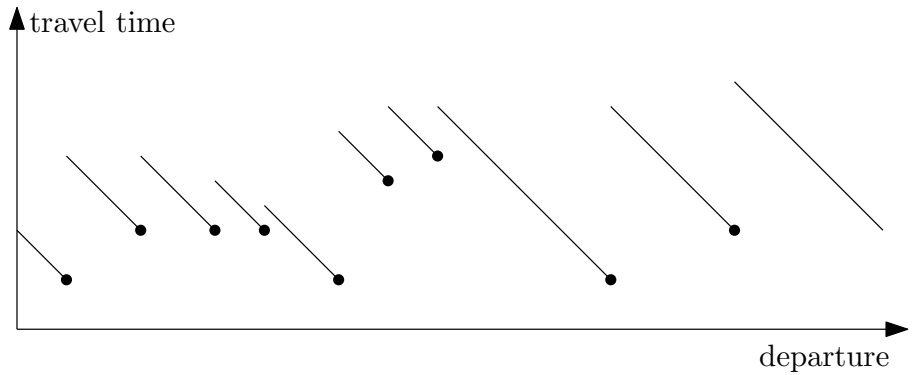


# Public Transport: Auswertung

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

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

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




**Problem:**

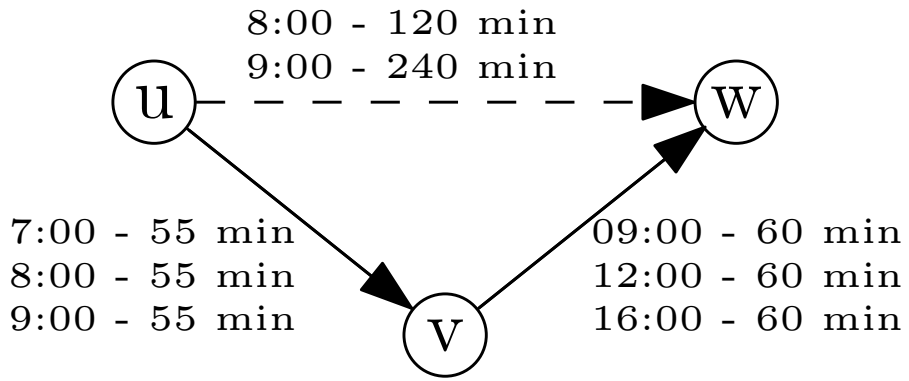
- finden von $t_i$ und $t_{i+1}$
- theoretisch:
  - lineare Suche: $O(|I|)$
  - binäre Suche: $O(\log_2 |I|)$
- praktisch:
  - $|I| < 30$: lineare Suche
  - sonst: lineare Suche mit Startpunkt $\tau/\Pi * |I|$



# Public Transport: Link

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

- für jede 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 $(t_i^f, (t_j^g - t_i^f + w_j^g))$ hinzu
- wenn zwei Punkte den gleichen Verbindungspunkt haben, behalte nur den mit größerem $t_i^f$
- wieder sweep-algorithmus



# Public Transport: Diskussion Link

**Laufzeit**

- sweep Algorithmus
- $O(|I^f| + |I^g|)$
- zeitunabhängig: $O(1)$

**Speicherverbrauch**

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

**Somit:**

- deutlich gutmütiger als Straßengraph-Funktionen



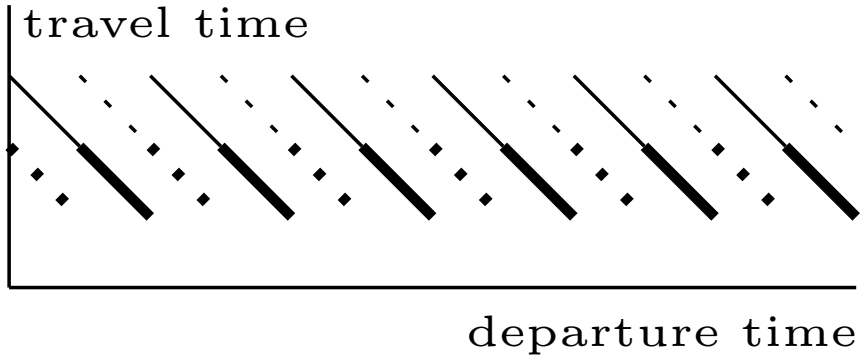
# Public Transport: Merge

**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
- $O(|I^f| + |I^g|)$
- zeitunabhängig: $O(1)$

**Speicherverbrauch**

- da keine Schnittpunkte
- minimum-Funktion kann maximal $|I^f| + |I^g|$ Interpolationspunkte haben



# Public Transport vs. Schiene

**Laufzeit Operationen**

- gleich für beide
- $O(\log |I|)$ für Auswertung
- $O(|I^f| + |I^g|)$ für linken und minimum

**Speicherverbrauch**

- Public Transport 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



# Eingabe

**Straße:**

- Netzwerk Deutschland $|V| \approx 4.7$ Mio., $|E| \approx 10.8$ Mio.
- 5 Verkehrszenarien:
  - Montag: $\approx 8\%$ Kanten zeitabhängig
  - Dienstag - Donnerstag: $\approx 8\%$
  - Freitag: $\approx 7\%$
  - Samstag: $\approx 5\%$
  - Sonntag: $\approx 3\%$

**Schiene:**

- Europa Fernverbindungen
- 30156 Stationen, 1.8 Verbindungen
- $|V| = 0.4$ Mio., $|E| = 1.4$ Mio.

# *Einfluss Grad zeitabhängigkeit auf Zeitanfragen*

| | #delete mins | slow-down | time [ms] | slow-down |
|---|---|---|---|---|
| kein | 2,239,500 | 0.00% | 1219.4 | 0.00% |
| Montag | 2,377,830 | 6.18% | 1553.5 | 27.40% |
| DiDo | 2,305,440 | 2.94% | 1502.9 | 23.25% |
| Freitag | 2,340,360 | 4.50% | 1517.2 | 24.42% |
| Samstag | 2,329,250 | 4.01% | 1470.4 | 20.59% |
| Sonntag | 2,348,470 | 4.87% | 1464.4 | 20.09% |

**Beobachtung:**

- kaum Veränderung in Suchraum
- Anfragen etwas langsamer durch Auswertung

# *Profilsuchen Straße*

**Beobachtung:**

- nicht durchführbar durch zu großen Speicherbedarf (32 GB RAM)
- interpoliert:
  - Suchraum steigt um ca. 10%
  - Suchzeiten um einen Faktor von bis zu 2500
- also inpraktikabel

# Profilsuchen Schiene

| | #delete mins | time [ms] |
|---|---|---|
| Zeit-Anfragen | 260 095 | 125 |
| Profil-Anfragen | 1 919 662 | 5 327 |

**Beobachtung:**

- delete mins steigen an (ungefähr Faktor 8)
- Query Zeit steigt an um Faktor 42
- Verlust für Operationen ist ca. 5

# Zusammenfassung Zeitabhängige Netzwerke (Basics)

- Funktionen werden an Kanten gehangen
- Operationen werden teurer
  - $O(\log |I|)$ für Auswertung
  - $O(|I^f| + |I^g|)$ für linken und minimum
  - Straßennetzwerke: Speicherverbrauch explodiert
  - Eisenbahn gutartiger
- Zeit- und Profilanfragen
- Zeitanfragen:
  - normaler Dijkstra
  - kaum langsamer (lediglich Auswertung)
- Profilanfragen
  - in PTN gut nutzbar
  - Straßennetzwerke nicht zu handhaben
  - aber eigentlich wollen wir das haben



# *Literatur*

**Basics Time-Dependency:**

- Delling 08,09

**Anmerkung:**

- wird auf der Homepage verlinkt