Karlsruher Institut für Technologie

Algorithmen für Routenplanung

12. Sitzung, Sommersemester 2013

Julian Dibbelt | 17. Juni 2013

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

Diskussion

Stoff bisher

- Problemtransformation auf Graph + Dijkstra
- Vorberechnungstechniken (point-to-point)
- Weitere Anfrageszenarien: one-to-all, many-to-many, POIs
- Turn-Costs
- Alternativrouten (fast optimal, hinreichend unterschiedlich)

Diskussion

Stoff bisher

- Problemtransformation auf Graph + Dijkstra
- Vorberechnungstechniken (point-to-point)
- Weitere Anfrageszenarien: one-to-all, many-to-many, POIs
- Turn-Costs
- Alternativrouten (fast optimal, hinreichend unterschiedlich)

Was fehlt noch für Einsatz in Produktion?

Diskussion

Stoff bisher

- Problemtransformation auf Graph + Dijkstra
- Vorberechnungstechniken (point-to-point)
- Weitere Anfrageszenarien: one-to-all, many-to-many, POIs
- Turn-Costs
- Alternativrouten (fast optimal, hinreichend unterschiedlich)

Was fehlt noch für Einsatz in Produktion?

- Berücksichtigung von (aktuellen) Staus
- Implementation auf Mobilgeräten
- Berücksichtigung von historischem Wissen über Staus
- Was ist eigentlich mit Bus+Bahn?

Dynamische Szenarien

Szenario:

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

Herausforderung

Hauptproblem:

- Kantengewichte ändern sich
- Vorberechnung basiert auf ursprünglichen Kantengewichten
- komplette Vorberechnung für jeden Stau wenig sinnvoll

Herausforderung

Hauptproblem:

- Kantengewichte ändern sich
- Vorberechnung basiert auf ursprünglichen Kantengewichten
- komplette Vorberechnung für jeden Stau wenig sinnvoll

Serverszenario:

- identifiziere den Teil der beeinträchtigten Vorberechnung
- aktualisiere diesen Teil

Herausforderung

Hauptproblem:

- Kantengewichte ändern sich
- Vorberechnung basiert auf ursprünglichen Kantengewichten
- komplette Vorberechnung für jeden Stau wenig sinnvoll

Serverszenario:

- identifiziere den Teil der beeinträchtigten Vorberechnung
- aktualisiere diesen Teil

mobiles Szenario:

- robuste Vorberechnung
- immer noch korrekt, wenn Kantengewichte sich erhöhen
- dadurch eventuell Anfragen langsamer
- oder passe Query an, so dass Vorberechnung nicht aktualisiert werden muss

Landmarken [Goldberg, Harrelson 05]

Vorberechnung:

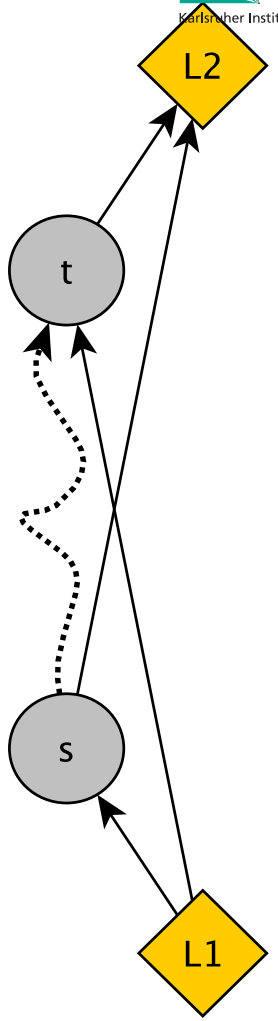
- wähle eine Hand voll (≈ 16) Knoten als Landmarken
- berechne Abstände von und zu allen Landmarken

Anfrage:

- benutze Landmarken und Dreiecksungleichung um eine untere Schranke für den Abstand zum Ziel zu bestimmen

$$\begin{aligned} d(s,t) &\geq d(L_1,t) - d(L_1,s) \\ d(s,t) &\geq d(s,L_2) - d(t,L_2) \end{aligned}$$

- verändert Reihenfolge der besuchten Knoten



Serverszenario

Beobachtung:

- Landmarkenwahl ist Heuristik, also nicht ändern
- Distanzen von und zu Landmarken sind nicht korrekt

Vorgehen:

- aktualisiere diese Distanzen
- benutze dafür dynamisierte Dijkstra-Variante, die nur betroffene Teilbäume (jeder Landmarke) neuberechnet
- dann Anfragen korrekt und (in etwa) so schnell wie bei kompletter neuer Vorberechnung

Mobiles Szenario

Beobachtung:

- Korrektheit von ALT basiert darauf, dass reduzierten Kantengewichte größer gleich 0 sind

$$\text{len}_\pi(u, v) = \text{len}(u, v) - \pi(u) + \pi(v) \geq 0$$

- durch Erhöhen der Kantengewichte wird dies nicht verletzt
- durch Staus können Reisezeiten nicht unter Initialwert fallen

Mobiles Szenario

Beobachtung:

- Korrektheit von ALT basiert darauf, dass reduzierten Kantengewichte größer gleich 0 sind

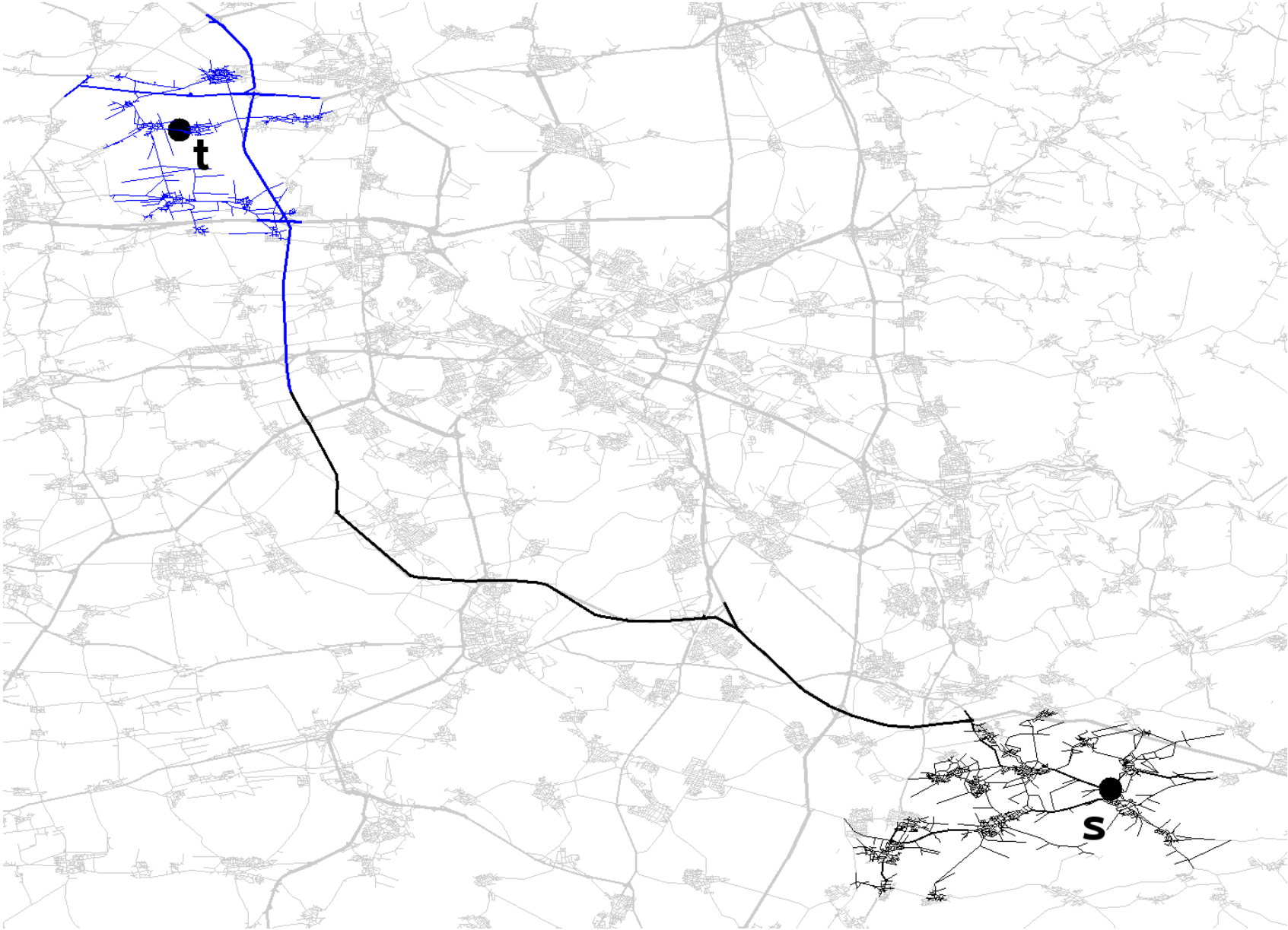
$$\text{len}_\pi(u, v) = \text{len}(u, v) - \pi(u) + \pi(v) \geq 0$$

- durch Erhöhen der Kantengewichte wird dies nicht verletzt
- durch Staus können Reisezeiten nicht unter Initialwert fallen

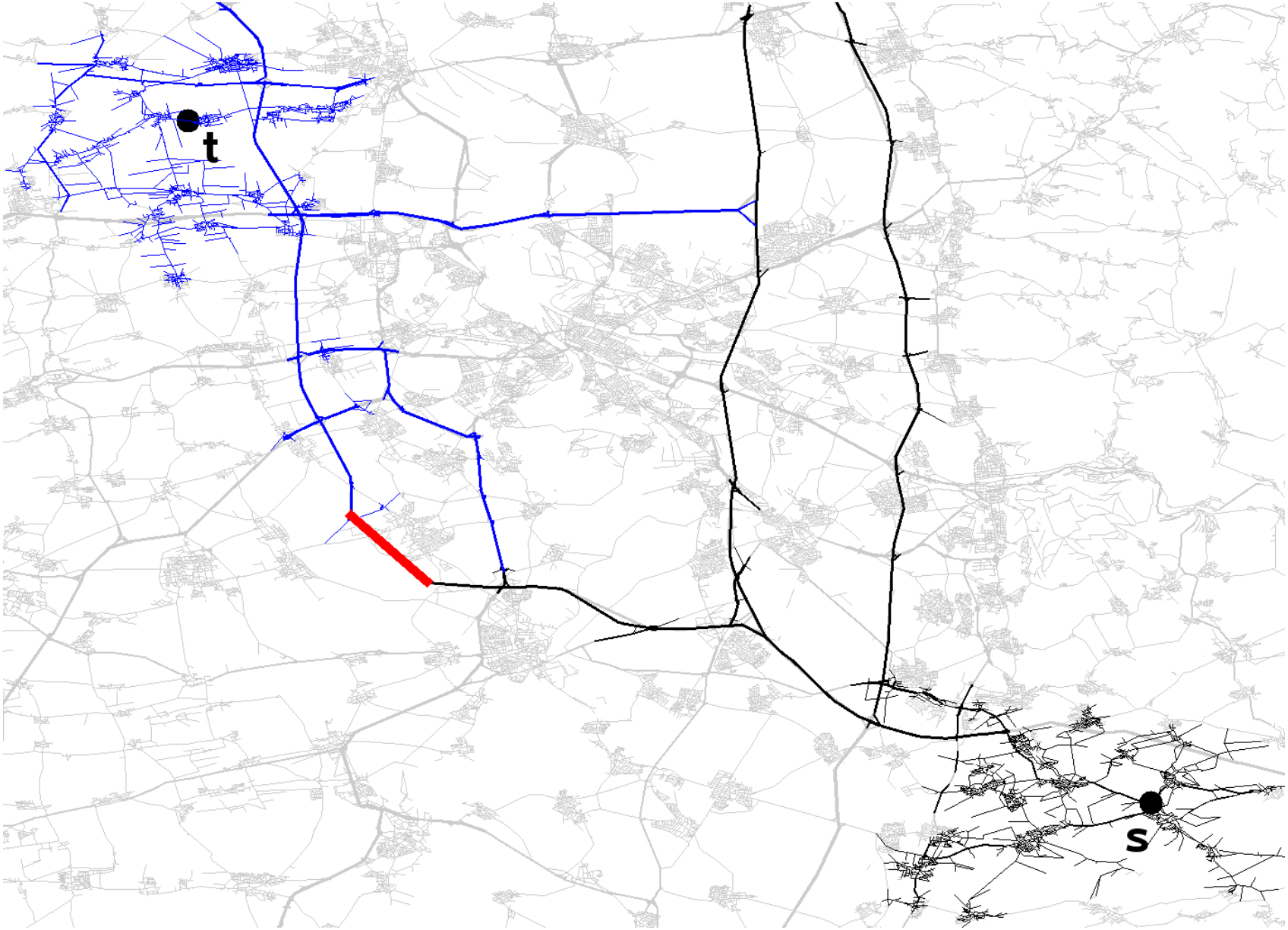
Idee:

- Vorberechnung auf staufreiem Graphen
- Suchanfragen ohne Aktualisierung
- korrekt aber eventuell langsamere Anfragezeiten

Beispiel

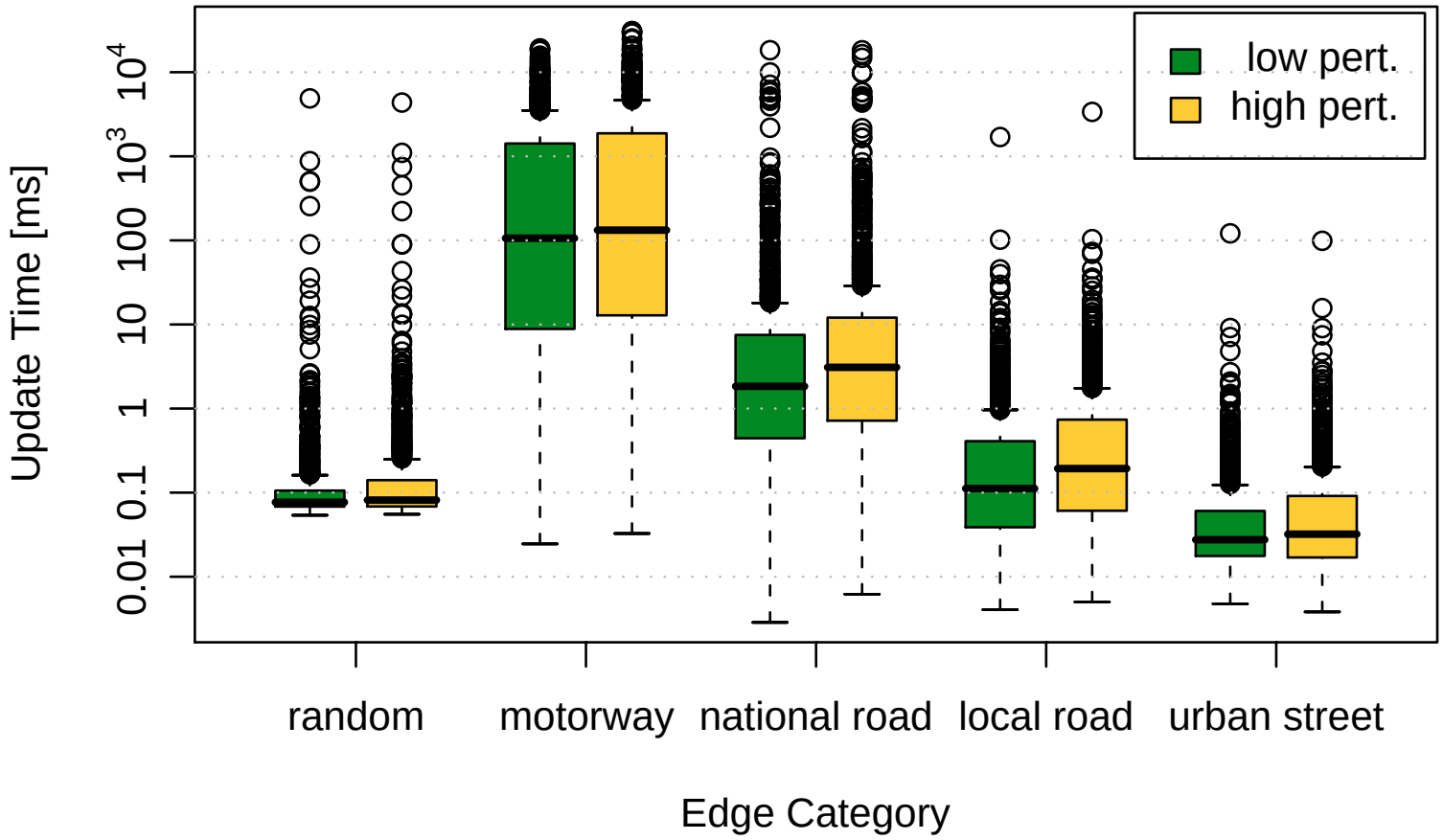


Beispiel



Aktualisierung der Vorberechnung

Aktualisieren von 32 kürzeste-Wege Bäumen für 16 Landmarken:



No-update Variante: Typ des Updates

- Zufallsanfragen nach 1 000 updates
- erhöhe Kantengewicht um x2 (x10 in Klammern)
- verschiedene Straßenkategorien

road type	affected queries	16 landmarks search space		32 landmarks search space	
no updates	0.0 %	74 699	(74 699)	40 945	(40 945)
all	7.5 %	74 700	(77 759)	41 044	(43 919)
urban	0.8 %	74 796	(74 859)	40 996	(41 120)
local	1.5 %	74 659	(74 669)	40 949	(40 995)
national	28.1 %	74 920	(75 777)	41 251	(42 279)
motorway	95.3 %	97 249	(265 472)	59 550	(224 268)

Beobachtung:

- nur Autobahnen haben Einfluß auf Suchraumgröße

No-update Variante: Anzahl Updates

- Zufallsanfragen nach Autobahn updates (x2, x10 in Klammern)
- verschiedene Anzahl Updates

updates	affected queries	16 landmarks search space		32 landmarks search space	
0	0.0 %	74 699	(74 699)	40 945	(40 945)
100	39.9 %	75 691	(91 610)	41 725	(56 349)
200	64.7 %	78 533	(107 084)	44 220	(69 906)
500	87.1 %	86 284	(165 022)	50 007	(124 712)
1 000	95.3 %	97 249	(265 472)	59 550	(224 268)
2 000	97.8 %	154 112	(572 961)	115 111	(531 801)
5 000	99.1 %	320 624	(1 286 317)	279 758	(1 247 628)
10 000	99.5 %	595 740	(2 048 455)	553 590	(1 991 297)

Beobachtung:

- $\leq$ 1000 gut verkraftbar

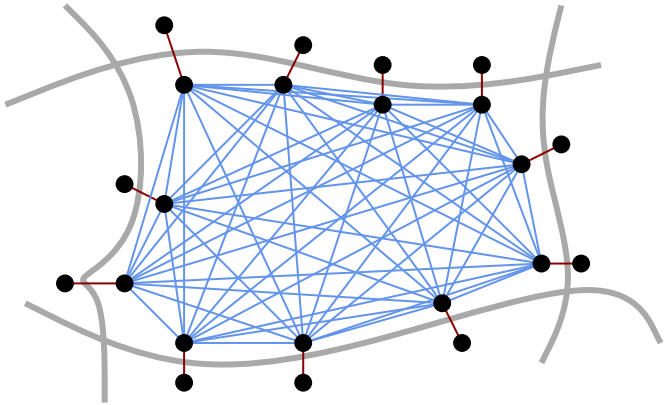
Customizable Route Planning

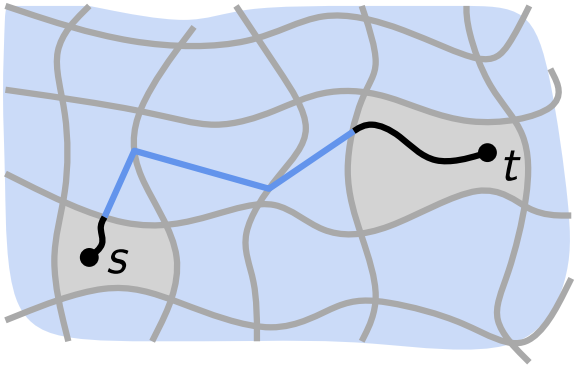
Idee:

- partitioniere Graphen
- Berechne Distanzen zwischen Randknoten *in jeder Zelle*

Overlay Graph:

- Randknoten
- Cliquen in jeder Zelle
- Schnittkanten





Suchgraph:

- Start- und Zielzelle...
- ...plus Overlaygraph.
- (bidirektionaler) Dijkstra

Optimierung: multiple Level

Update der Vorberechnung

Beobachtung:

- Shortcuts (Cliquenkanten) können invalidiert werden
- Shortcuts repräsentieren Pfade innerhalb der Zelle

Update:

- identifiziere Zelle in der die aktualisierte Kante liegt für jedes Level der Multi-Level Partition
- wiederhole (bottom-up) Preprocessing für diese Zellen
- dauert weniger als 10 ms

Mobiles Szenario

Beobachtung:

- Update invalidiert Zellen-Overlay
- erhöht meistens nur einige Overlay-Gewichte
- restliche Overlay-Kanten bleiben korrekt

Query:

- Suche muss bei invalidierten Zellen absteigen
- jeder Abstieg erhöht Suchraum um ca. 50%

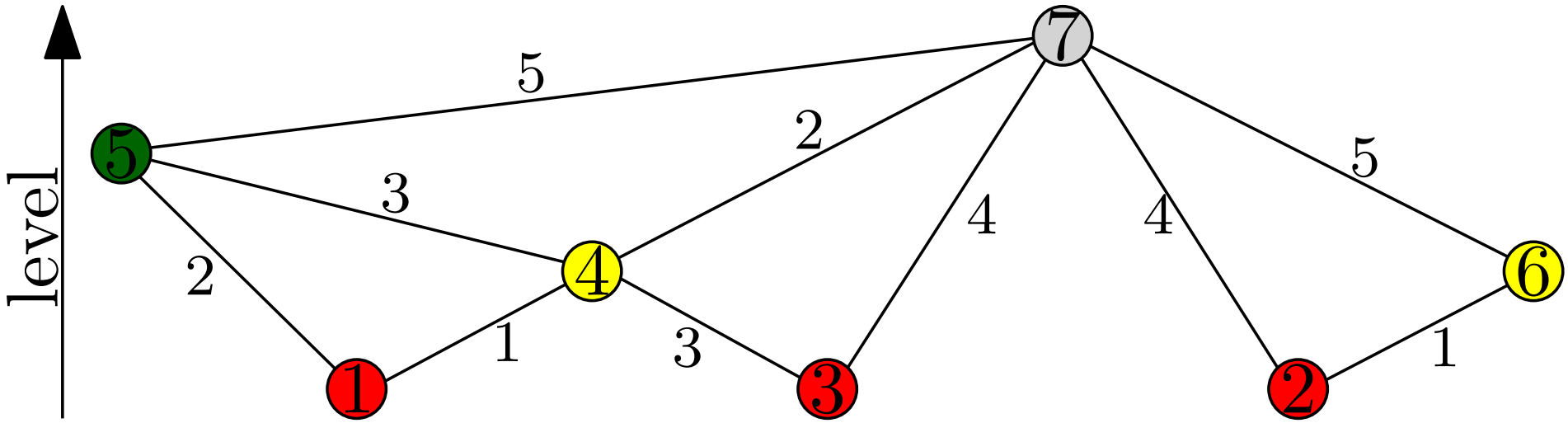
Optimierung:

- markiere invalidierte Zellen (ohne Overlay zu aktualisieren)
- führe Anfrage aus (ohne Abstieg)
- wenn der Pfad nicht die aktualiesierte Kante enhält $\Rightarrow$ fertig
- sonst Suche mit Abstieg

Contraction Hierarchies

preprocessing:

- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu
- Levelzuordnung (ca. 150 in Strassennetzwerken)



Update der Vorberechnung I

Beobachtung:

- Knotenordnung ist heuristik
- also behalte Ordnung bei
- geänderte Kante (u, v) kann zu Shortcuts beitragen
- aber auch Teil von Zeugensuchen gewesen sein
- Frage: welche Knoten müssen neu kontrahiert werden (mit Zeugensuche)

Update der Vorberechnung II

Idee:

- wiederhole Kontraktion für Knotenmenge U
- füge alle aufwärts erreichbaren Knoten von u, v zu U
- speicher für jede Kante e auf welchen Zeugensuchen sie benutzt wurde. Die zugehörigen Knoten speicher in A_e. Kann einfach während Vorberechnung gespeichert werden.
- füge alle aufwärts erreichbaren Knoten von A_e (und aller A_f für alle Shortcuts f, die e enthalten) zu U
- wiederhole Kontraktion für alle Knoten U bezüglich Originalordnung

Beobachtung:

- erhöht Speicherverbrauch von CH deutlich
- Updatezeit abhängig von Art der Kante

Mobiles Szenario

Beobachtung:

- Kontraktion von Knoten zu teuer

Mobiles Szenario

Beobachtung:

- Kontraktion von Knoten zu teuer

Idee:

- identiziere Knotenmenge U wie zuvor
- erlaube Abstieg an Kanten (u, v) mit $u, v \in U$
- iterativer Ansatz (wie bei CRP) auch hier möglich

Weitere Techniken

Arc-Flags:

- Aktualisierung aller Bäume nötig
- momentan keine vernünftige Update-Routine vorhanden

Transit-Node Routing

- schwierig
- Tabellen und Access-Nodes können sich ändern

HubLabels

- halte CH vor
- nach Update: aktualisierung alles Labels
- nicht getestet: nicht alle Labels müssen aktualisiert werden

Zusammenfassung

- Stau erhöht Kantengewicht temporär
- Anpassung Landmarken, CRP trivial
- Anpassung CH machbar
- sehr schwierig bei anderen Techniken

Anmerkungen

- Mobil: nur “nahe” Staus beachten?
- viele Staus verhersehbar

Literatur I

Dynamische Szenarien:

- Daniel Delling, Dorothea Wagner **Landmark-Based Routing in Dynamic Graphs**
  In: *Proceedings of the 6th Workshop on Experimental Algorithms (WEA'07)*, 2007
- Robert Geisberger, Peter Sanders, Dominik Schultes, Christian Vetter
  Exact Routing in Large Road Networks Using Contraction Hierarchies
  In: *Transportation Science*, 2012
- Daniel Delling, Andrew V. Goldberg, Thomas Pajor, Renato Werneck
  Customizable Route Planning
  In: *Proceedings of the 10th International Symposium on Experimental Algorithms (SEA'11)*, 2011

Diskussion

Wie historische Daten einbeziehen?

Zeitabhängiges Szenario

Szenario:

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

Herausforderung

Hauptproblem:

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

Herausforderung

Hauptproblem:

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

Vorgehen:

- Modellierung
- Anpassung Dijkstra
- Anpassung Beschleunigungstechniken

Herausforderung

Hauptproblem:

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

Vorgehen:

- Modellierung
- Anpassung Dijkstra
- Anpassung Beschleunigungstechniken

Heute:

- Modellierung und Dijkstra

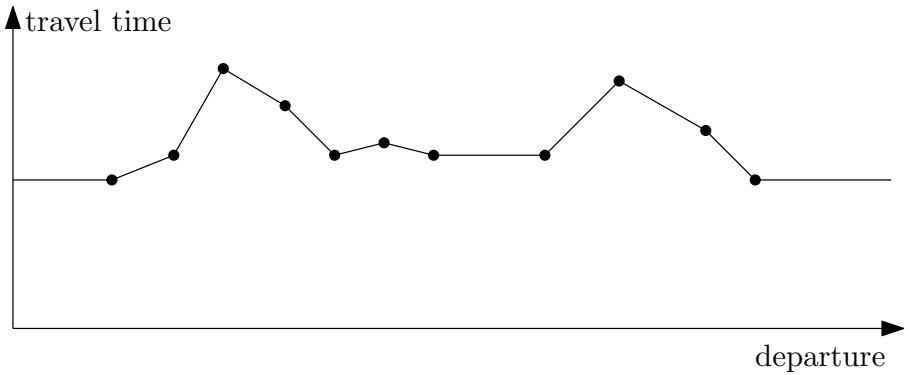
Zeitabhängige Straßennetzwerke

Eingabe:

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

Somit an jeder Kante:

- Periodische stückweise lineare Funktion
- Definiert durch Stützpunkte
- Interpoliere linear zwischen Stützpunkten



FIFO-Eigenschaft

Definition

Sei $f : \mathbb{R}_0^+ \to \mathbb{R}_0^+$ 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/Später ankommen 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.

Diskussion

Eigenschaften:

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

Diskussion

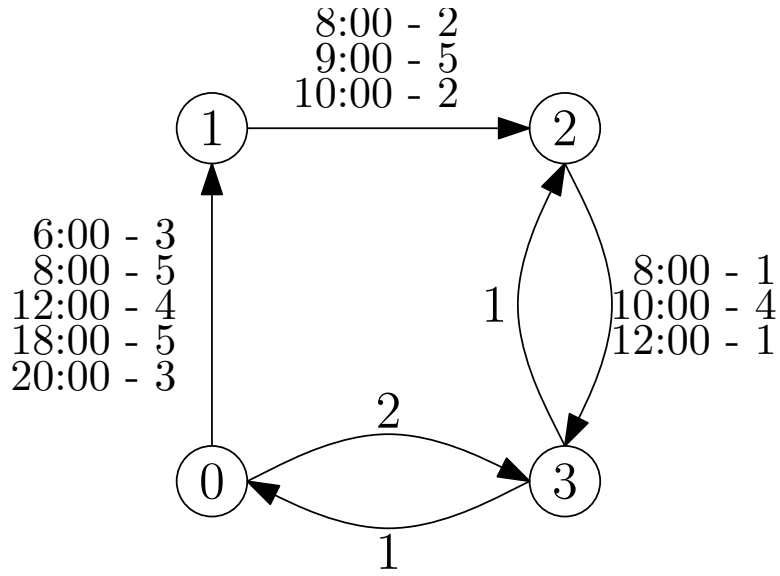
Eigenschaften:

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

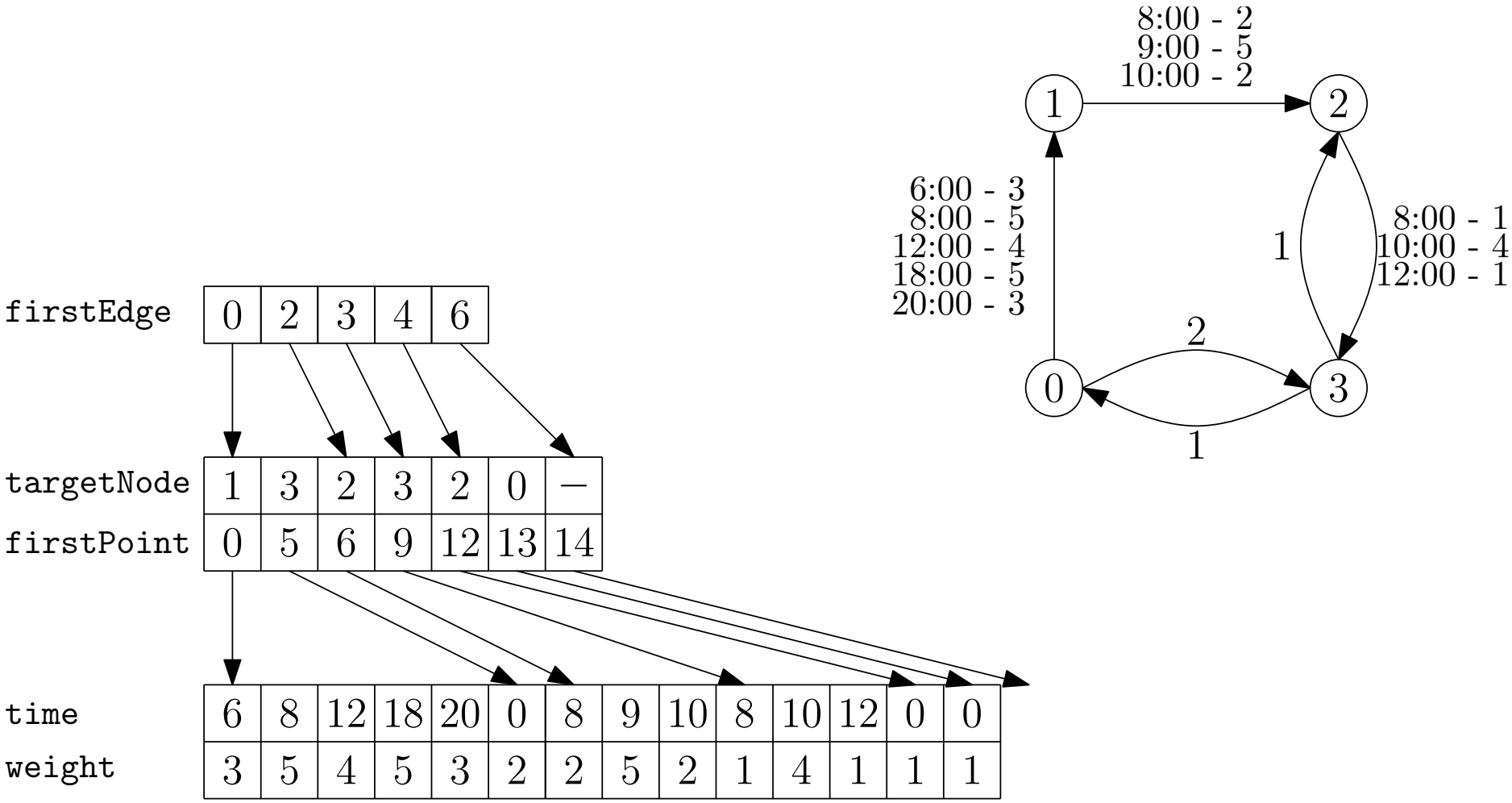
Beobachtungen:

- FIFO gilt auf allen Kanten
- später wichtig

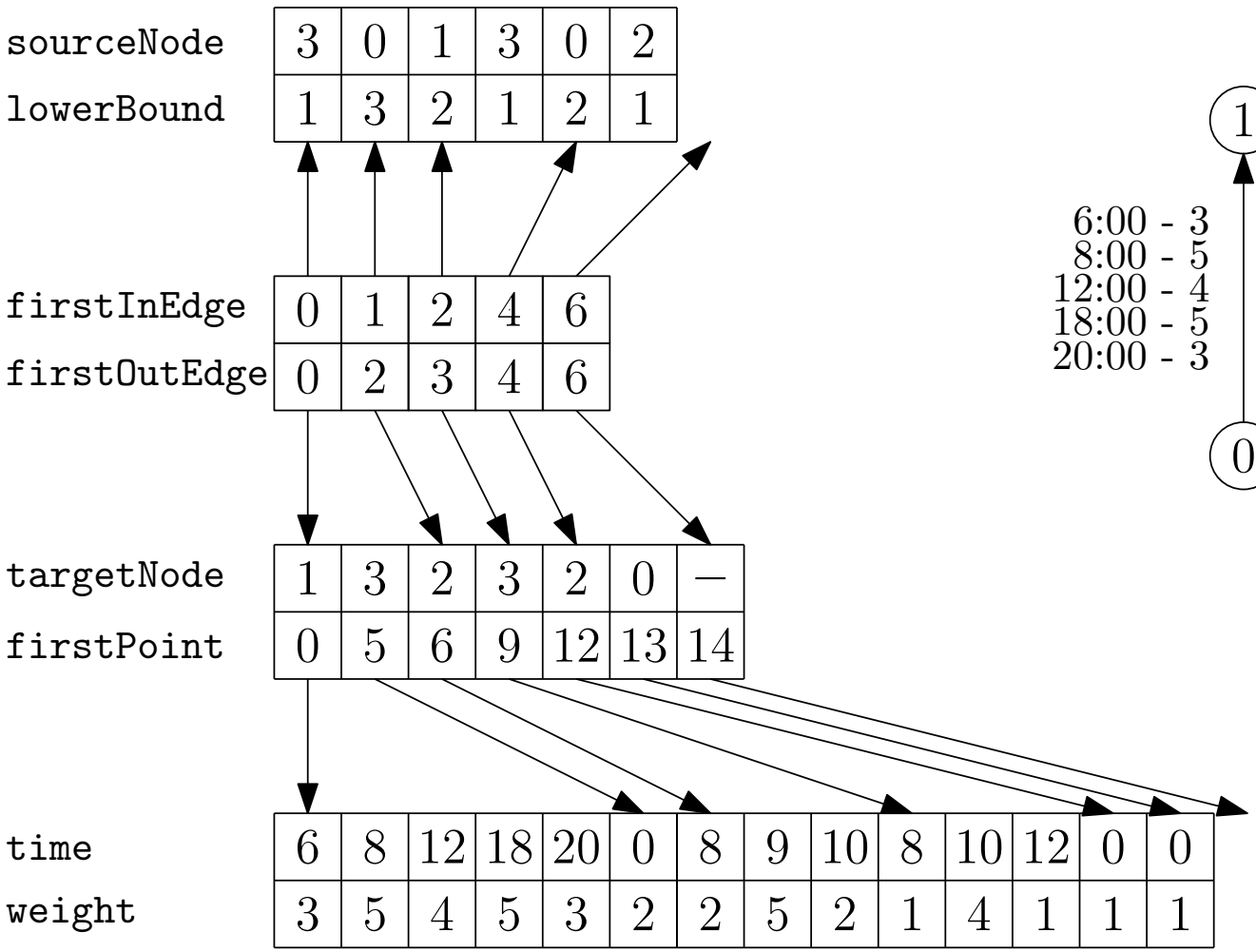
Datenstruktur

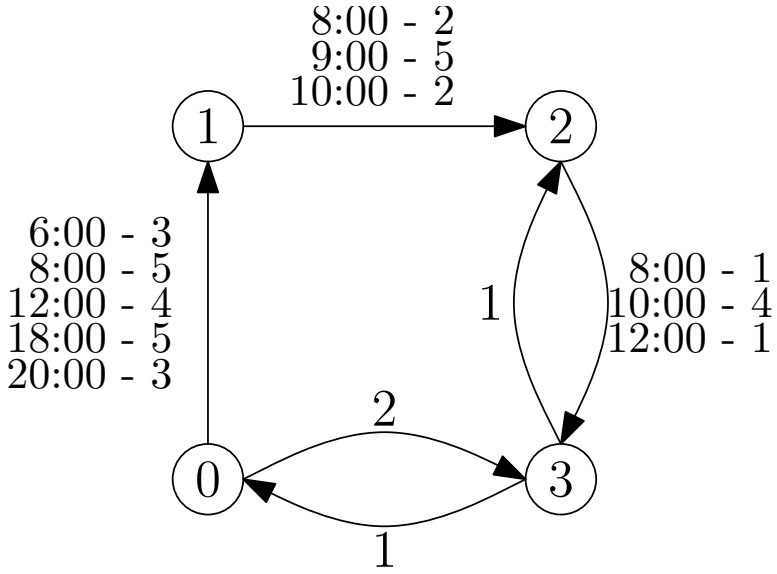


Datenstruktur



Datenstruktur





Anfrageszenarien

Zeit-Anfrage:

- finde kürzesten Weg für Abfahrtszeit τ
- analog zu Dijkstra?

Anfrageszenarien

Zeit-Anfrage:

- finde kürzesten Weg für Abfahrtszeit τ
- analog zu Dijkstra?

Profil-Anfrage:

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

Zeit-Anfragen

Karlsruher Institut für Technologie

`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 $\quad u \leftarrow Q.deleteMin()$
5 $\quad$ **for** *all edges* $e = (u, v) \in E$ **do**
6 $\quad\quad$ **if** $d_\tau[u] + \text{len}(e, \tau + d_\tau[u]) < d_\tau[v]$ **then**
7 $\quad\quad\quad d_\tau[v] \leftarrow d_\tau[u] + \text{len}(e, \tau + d_\tau[u])$
8 $\quad\quad\quad p_\tau[v] \leftarrow u$
9 $\quad\quad\quad$ **if** $v \in Q$ **then** $Q.decreaseKey(v, d_\tau[v])$
10 $\quad\quad\quad$ **else** $Q.insert(v, d_\tau[v])$

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)

Profil-Anfragen

Karlsruher Institut für Technologie

`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 Profile-Anfragen

Beobachtungen:

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

Herausforderungen:

- Wie effizient $\oplus$ berechnen (Linken)?
- Wie effizient Minimum bilden?

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:

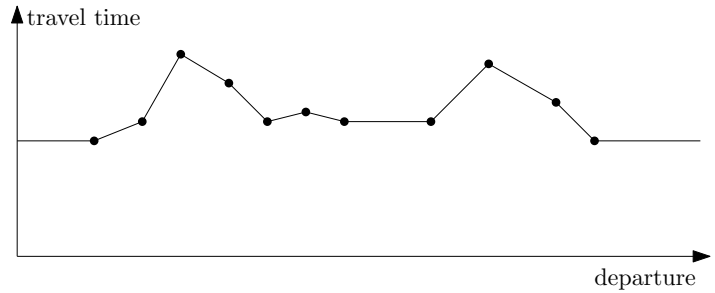
- Auswertung
- Linken $\oplus$
- Minimumsbildung

Auswertung

Evaluation von $f(\tau)$:

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

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

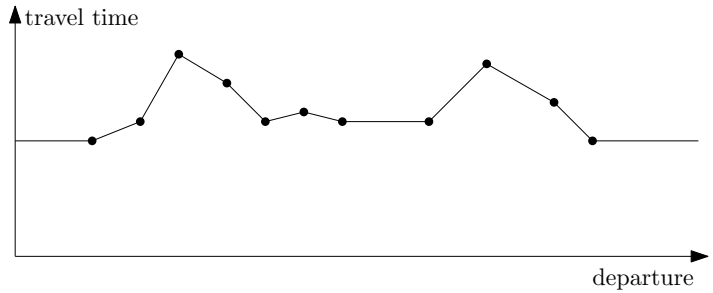


Auswertung

Evaluation von $f(\tau)$:

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

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



Problem:

- Finden von t_i und t_{i+1}
- Theoretisch:
  - Lineare Suche: $\mathcal{O}(|I|)$
  - Binäre Suche: $\mathcal{O}(\log_2 |I|)$
- Praktisch:
  - $|I| < 30 \Rightarrow$ lineare Suche
  - Sonst: Lineare Suche mit Startpunkt $\frac{\tau}{\Pi} \cdot |I|$
    wobei Π die Periodendauer ist

Linken

Definition

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

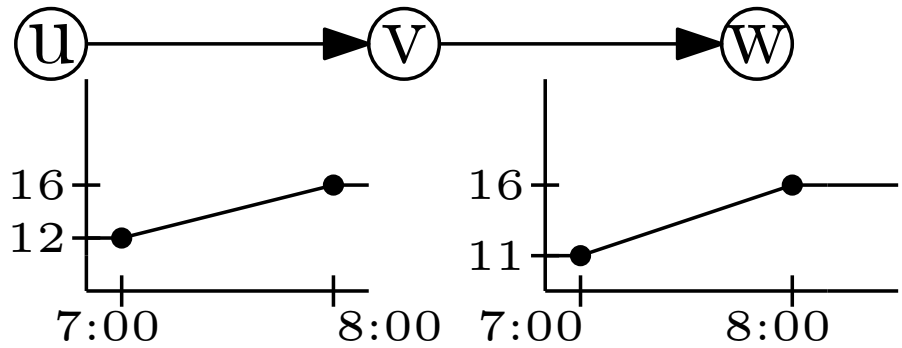
$$f \oplus g := f + g \circ (\mathrm{id} + f)$$

Oder

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

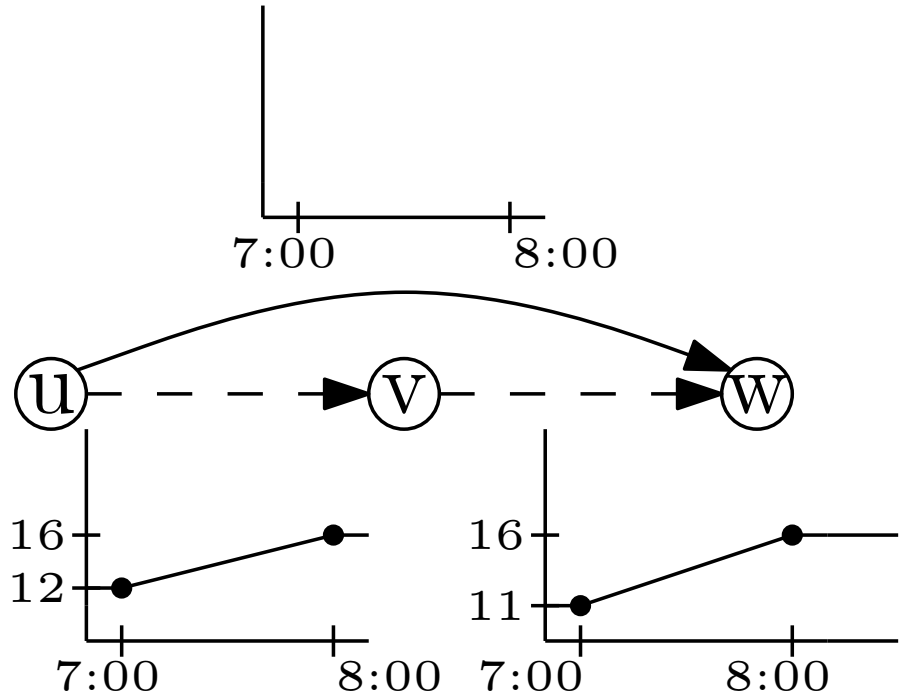
Linken

Linken zweier Funktionen f und g



Linken

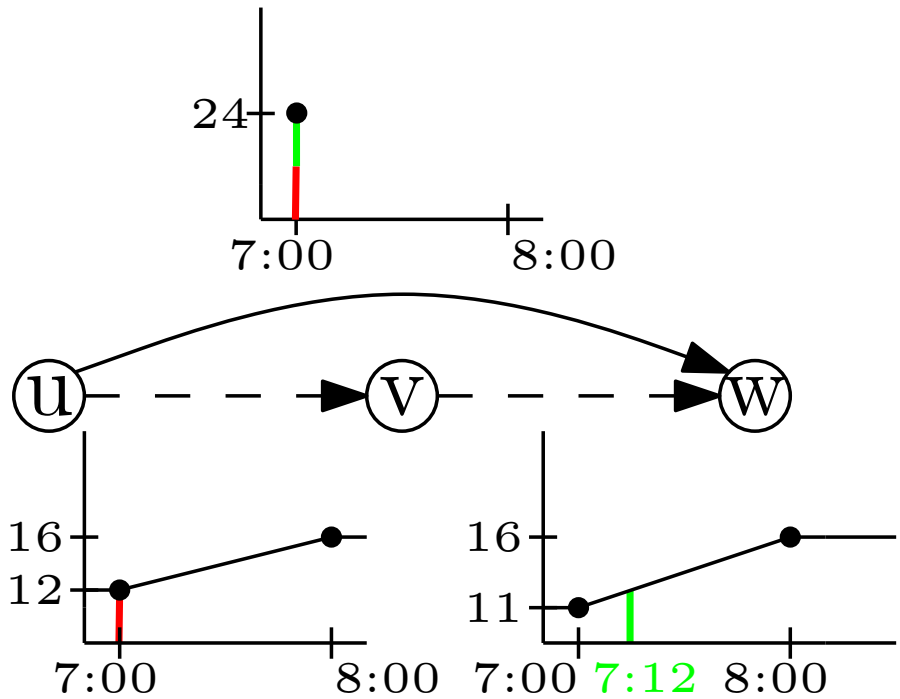
Linken zweier Funktionen f und g



Linken

Linken zweier Funktionen f und g

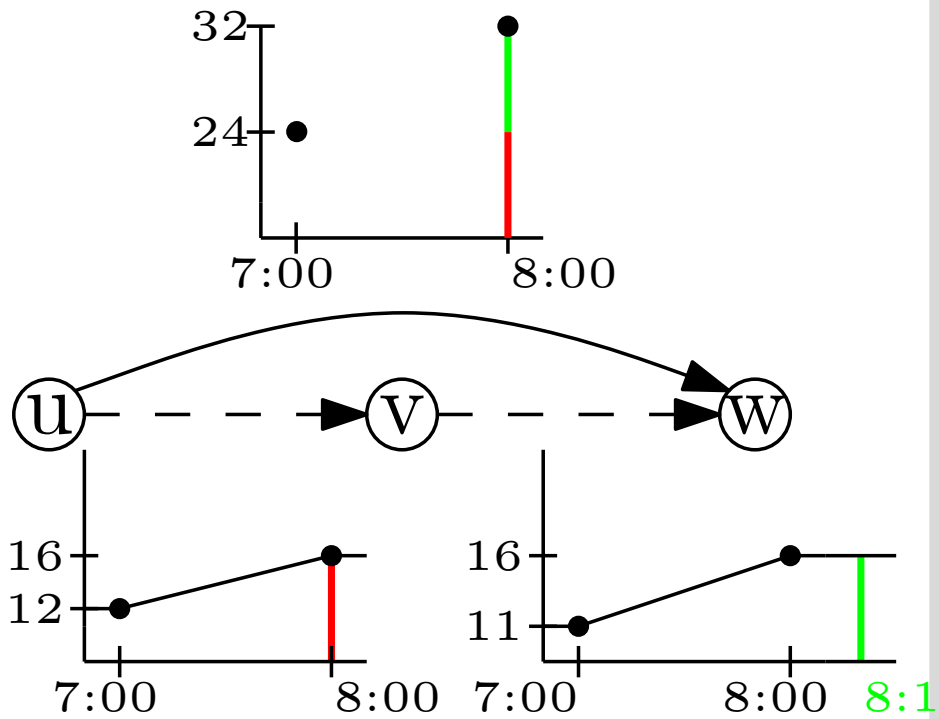
- $f \oplus g$ enthält auf jeden Fall
  $\{(t_1^f, w_1^f + g(t_1^f + w_1^f)), \ldots, (t_l^f, w_l^f + g(t_l^f + w_l^f))\}$



Linken

Linken zweier Funktionen f und g

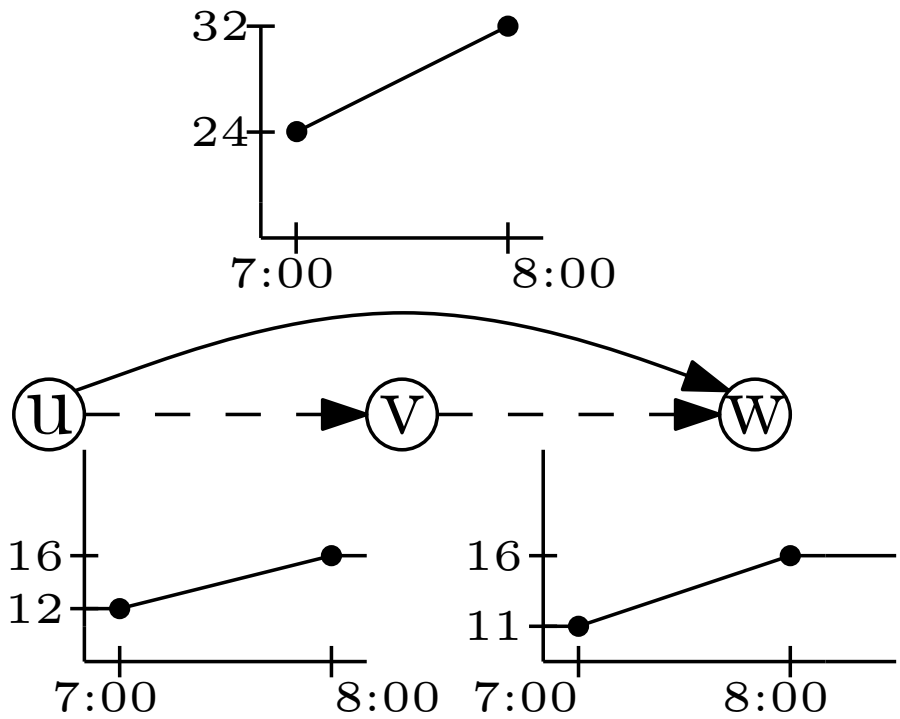
- $f \oplus g$ enthält auf jeden Fall
  $\{(t_1^f, w_1^f + g(t_1^f + w_1^f)), \ldots, (t_l^f, w_l^f + g(t_l^f + w_l^f))\}$



Linken

Linken zweier Funktionen f und g

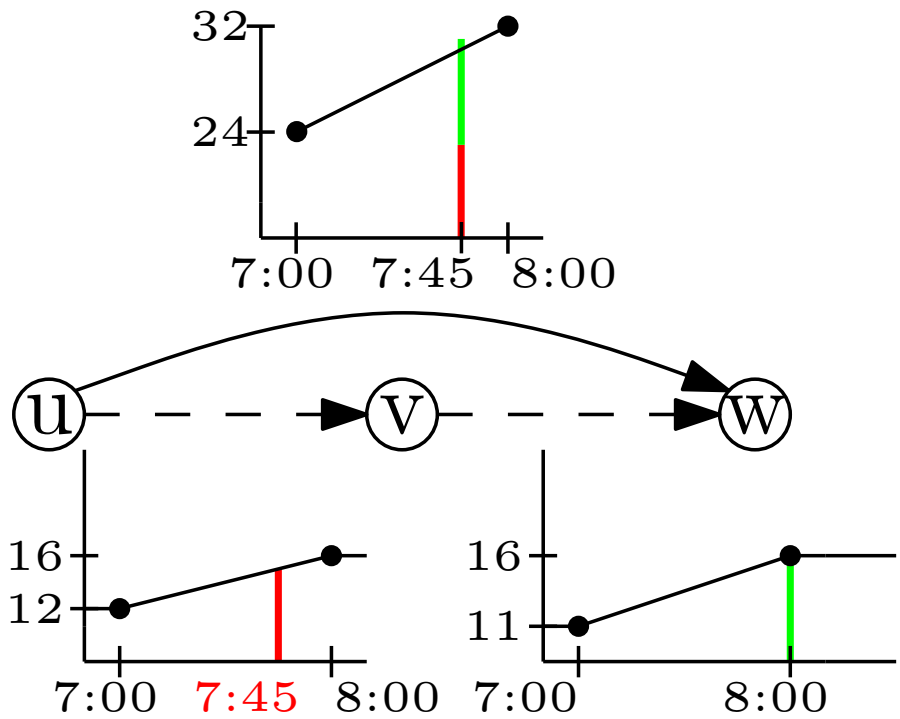
- $f \oplus g$ enthält auf jeden Fall
  $\{(t_1^f, w_1^f + g(t_1^f + w_1^f)), \ldots, (t_l^f, w_l^f + g(t_l^f + w_l^f))\}$



Linken

Linken zweier Funktionen f und g

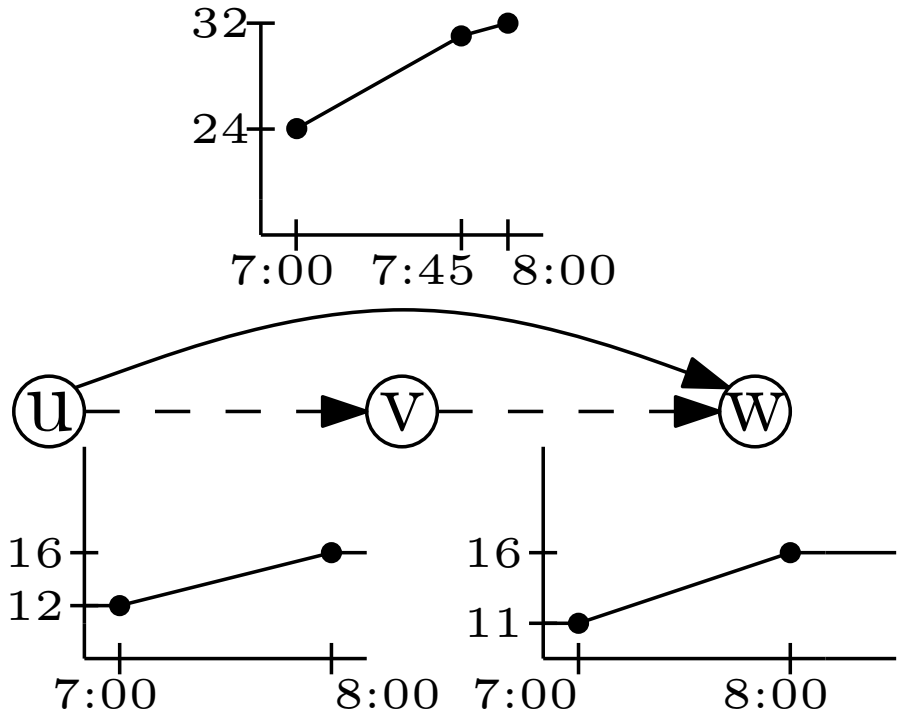
- $f \oplus g$ enthält auf jeden Fall
  $\{(t_1^f, w_1^f + g(t_1^f + w_1^f)), \ldots, (t_l^f, w_l^f + g(t_l^f + w_l^f))\}$



Linken

Linken zweier Funktionen f und g

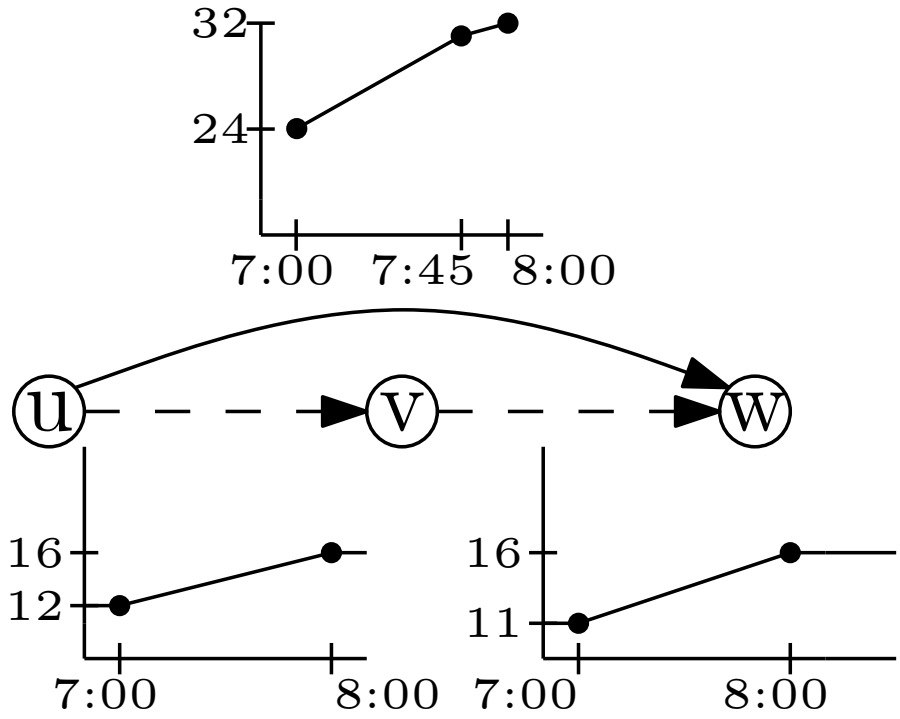
- $f \oplus g$ enthält auf jeden Fall $\{(t_1^f, w_1^f + g(t_1^f + w_1^f)), \ldots, (t_l^f, w_l^f + g(t_l^f + w_l^f))\}$
- Zusätzliche Interpolationspunkte an t_j^{-1} mit $f(t_j^{-1}) + t_j^{-1} = t_j^g$



Linken

Linken zweier Funktionen f und g

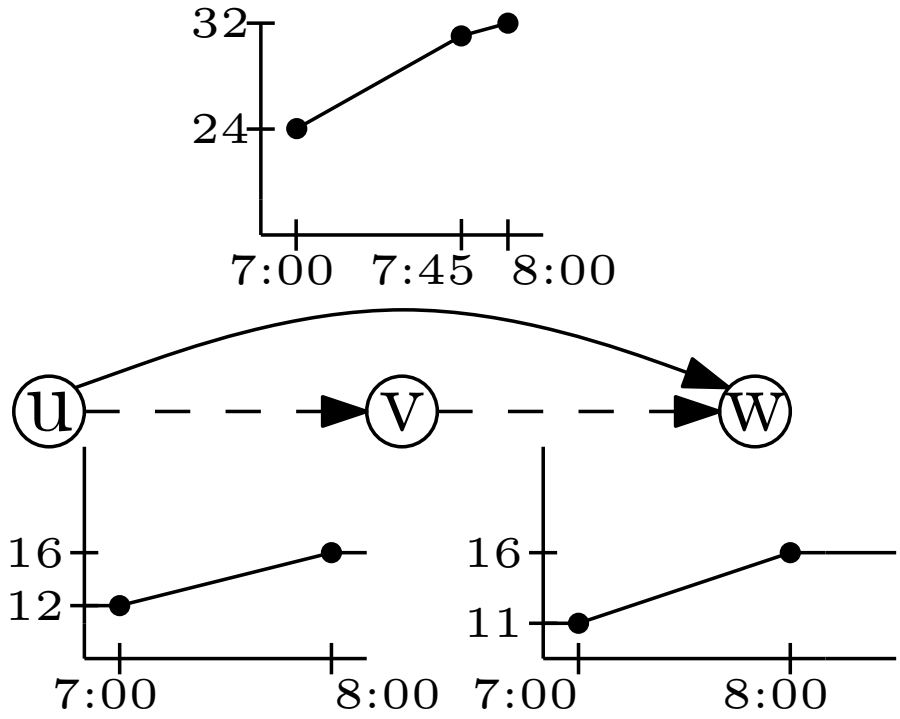
- $f \oplus g$ enthält auf jeden Fall $\{(t_1^f, w_1^f + g(t_1^f + w_1^f)), \ldots, (t_l^f, w_l^f + g(t_l^f + w_l^f))\}$
- Zusätzliche Interpolationspunkte an t_j^{-1} mit $f(t_j^{-1}) + t_j^{-1} = t_j^g$
- Füge $(t_j^{-1}, f(t_j^{-1}) + w_j^g)$ für alle Punkte von g zu $f \oplus g$



Linken

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



Diskussion Linken

Laufzeit

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

Diskussion Linken

Laufzeit

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

Speicherverbrauch

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

Diskussion Linken

Laufzeit

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

Speicherverbrauch

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

Problem:

- Während Profilsuche kann ein Pfad mehreren Tausend Kanten entsprechen

Diskussion Linken

Laufzeit

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

Speicherverbrauch

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

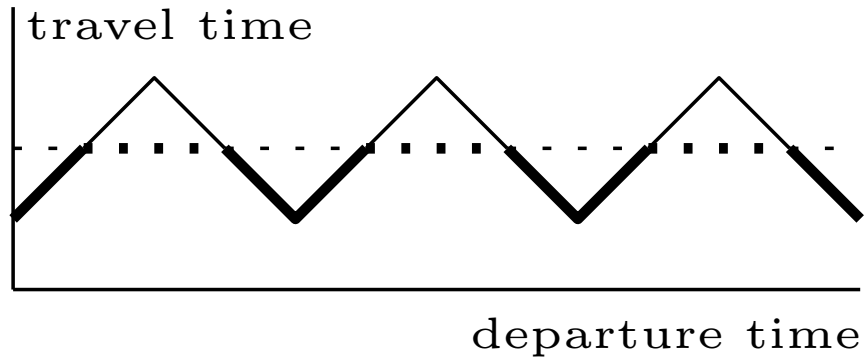
Problem:

- Während Profilsuche kann ein Pfad mehreren Tausend Kanten entsprechen
- Shortcuts...

Merge von Funktionen

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



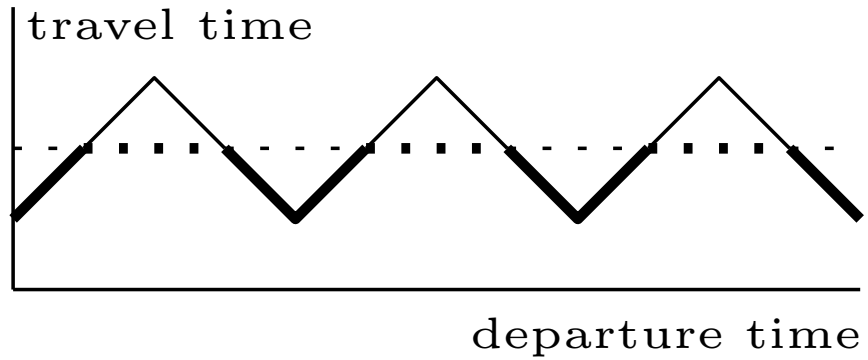
Merge von Funktionen

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

Vorgehen:

- Linearer sweep über die Stützstellen
- Evaluiere, welcher Abschnitt oben
- Checke ob Schnittpunkt existiert
- Vorsicht bei der Numerik



Diskussion Merge

Laufzeit

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

Diskussion Merge

Laufzeit

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

Speicherverbrauch

- Minimum-Funktion kann mehr als $|I^f| + |I^g|$ Interpolationspunkte enthalten

Diskussion Merge

Laufzeit

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

Speicherverbrauch

- Minimum-Funktion kann mehr als $|f| + |g|$ Interpolationspunkte enthalten

Problem:

- Während Profilsuche werden Funktionen gemergt
- Laufzeit der Profilsuchen wird durch diese Operationen (link + merge) dominiert

Eingabe

- 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: $\approx 3\%$

”Grad” der Zeitabhängigkeit

	#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

Beobachtung:

- Nicht durchführbar auf Europa-Instanz durch zu großen Speicherbedarf ($>$ 32 GiB RAM)
- Interpoliert:
  - Suchraum steigt um ca. 10%
  - Suchzeiten um einen Faktor von bis zu 2 500 über Dijkstra

$\Rightarrow$ inpraktikabel

Zusammenfassung

Zeitabhängige Netzwerke (Basics)

- Funktionen statt Konstanten an Kanten
- Operationen werden teurer
  - $\mathcal{O}(\log |f|)$ für Auswertung
  - $\mathcal{O}(|f^f| + |f^g|)$ für Linken und Minimum
  - Speicherverbrauch explodiert
- Zeitanfragen:
  - Normaler Dijkstra
  - Kaum langsamer (lediglich Auswertung)
- Profilanfragen
  - nicht zu handhaben

Literatur II

Zeitabhänige Routenplanung

- Daniel Delling:
  Engineering and Augmenting Route Planning Algorithms
  Ph.D. Thesis, Universität Karlsruhe (TH), 2009.

Nächste Termine

Mittwoch, 19.6.2013
Montag, 24.6.2013