Karlsruher Institut für Technologie

Algorithmen für Routenplanung

15. Vorlesung, Sommersemester 2019

Tim Zeitz | 1. Juli 2019

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

1. Dynamische Szenarien

Szenario:

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

Hauptproblem:

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

Lösung:

- “Customizable” Techniken (MLD, CCH)
- Anpassungen auch für ALT und “klassische” CH möglich

2. Zeitabhängiges Szenario

Motivation:

- Stau um Städte herum folgt vorhersehbaren Mustern
- Morgens geht jeder zur Arbeit $\rightarrow$ Stau
- Mittags gibt es weniger Stau
- Abends fährt jeder nach Hause $\rightarrow$ Stau in die andere Richtung
- Aber nicht Sonntags

2. Zeitabhängiges Szenario

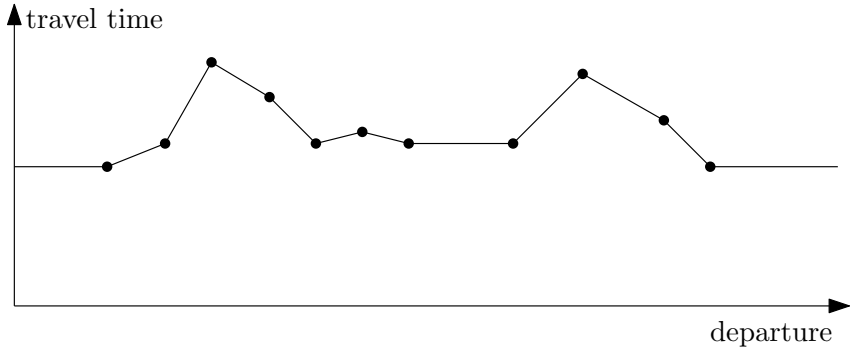
Motivation:

- Stau um Städte herum folgt vorhersehbaren Mustern
- Morgens geht jeder zur Arbeit $\rightarrow$ Stau
- Mittags gibt es weniger Stau
- Abends fährt jeder nach Hause $\rightarrow$ Stau in die andere Richtung
- Aber nicht Sonntags

- Aggregiere historische Daten um eine Vorhersage für jeden Wochentag zu machen

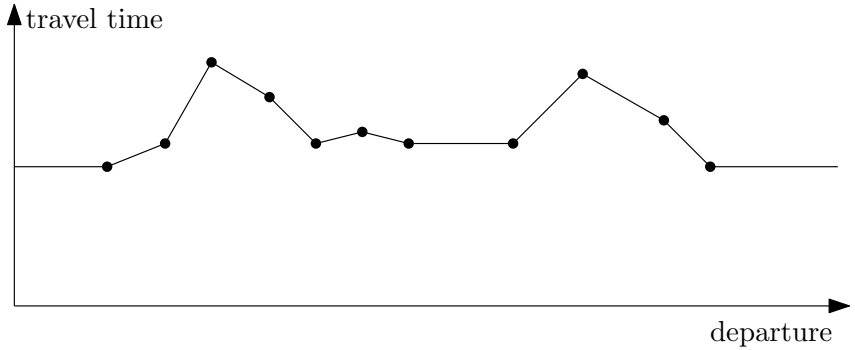
Zeitabhängige Kantengewichte

- **bisher:** An jeder Kante steht ein skalares Kantengewicht
- **neu:** An jeder Kanten steht eine Funktion
- Die Funktion bildet den Zeitpunkt an dem eine Kante betreten wird auf die Fahrzeit ab



Zeitabhängige Kantengewichte

- **bisher:** An jeder Kante steht ein skalares Kantengewicht
- **neu:** An jeder Kanten steht eine Funktion
- Die Funktion bildet den Zeitpunkt an dem eine Kante betreten wird auf die Fahrzeit ab



Problemstellung

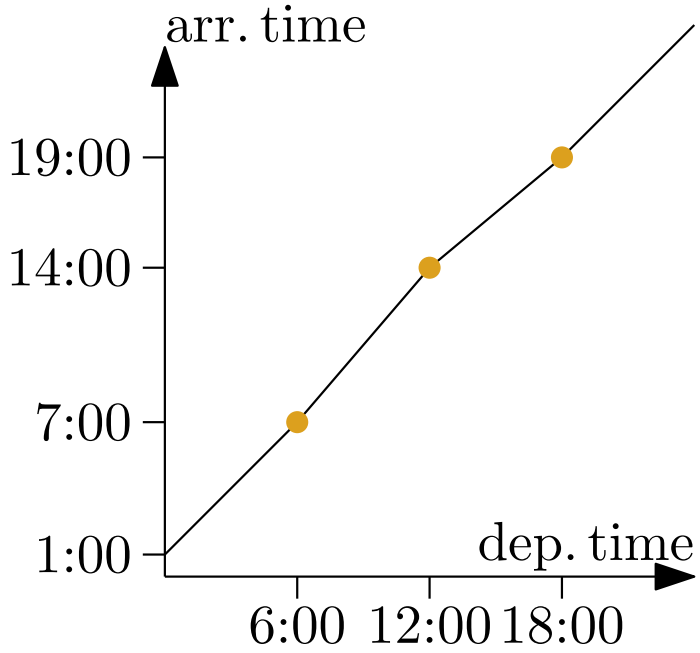
Mit zeitabhängigen Gewichten ist die Frage

- Wie komme ich von s nach t?

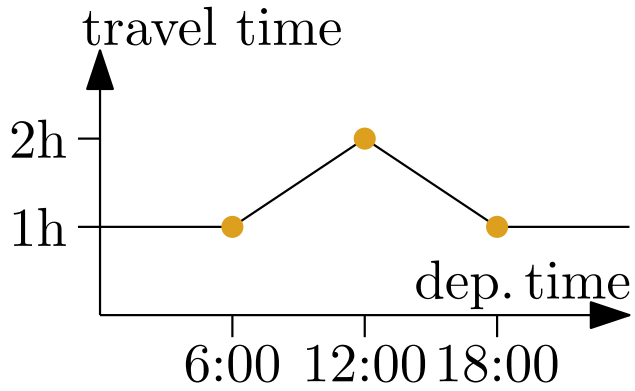
nicht mehr wohl geformt.
Wir betrachten nun das Problem der frühesten Ankunft:

- Wie komme ich von s nach t wenn ich um τ losfahre?

Zwei Sichtweisen



$f(x)$



$g(x)$

- Zwei Sichtweise auf die selbe Information
- $f(x) = g(x) + x$
- Aussagen in der einen Sichtweise lassen sich immer auf die andere übertragen
- Wir wechseln ständig zwischen beiden Sichtweisen und nehmen die, die gerade am besten passt

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

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

Diskussion

- Interpretation: “Warten/Später ankommen lohnt sich nie”
- Kürzeste Wege auf Graphen mit non-FIFO Funktionen zu finden ist (schwach) NP-schwer.
  (wenn warten an Knoten nicht erlaubt ist)
- Reduktion von Partition

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

Weitere Darstellungsformen

- In der akademischen Forschung nimmt man oft an, dass Interpolationspunkte pro Kante gegeben sind
- Viele Probleme mit dieser Annahme:
  - Viele Interpolationspunkte pro Graph
    $\rightarrow$ Großer Speicherverbrauch
    $\rightarrow$ Sehr problematisch
  - Erhobene Daten sind stark verrauscht und Vorhersagen ungenau
    $\rightarrow$ Interpolationspunkte suggerieren eine Genauigkeit die reale Daten nicht hergeben
- **Deswegen**: in der Praxis auch andere Darstellungsformen verbreitet
- In der Vorlesung bleiben wir aber bei der akademischen Sichtweise

Weitere Darstellungsformen

Option 1

- Teile den Tag in Abschnitte ein, z.b., in 24 Stunden
  - Abschnitte müssen nicht gleich groß sein
  - Unterteilung kann pro Kante variieren
- Speichere für jeden Abschnitt eine Reisegeschwindigkeit
  - Geschwindigkeiten können gerundet sein: z.B. 5 km/h Schritte
- Alle Fahrzeuge auf einer Kanten ändern spontan ihre Reisegeschwindigkeit wenn die Tageszeit über eine Abschnittsgrenzen fortschreitet
  - Aus dieser Eigenschaft folgt FIFO

Weitere Darstellungsformen

Option 2

- Speichere kleine globale Menge von Funktionskurven $\mathbb{F}$ explizit
  - Funktionen aus $\mathbb{F}$ werden meistens mittels Interpolationspunkte und linearer Interpolation dargestellt
  - Jede dieser Funktionen ist FIFO
- An jeder Kante speichert man einen fixen Satz an skalaren Attributen:
  - Funktion-ID in $\mathbb{F}$
  - Kanten-Länge
  - Diverse Strauchungs- und Streckungsfaktoren die die Funktion aus $\mathbb{F}$ transformieren

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

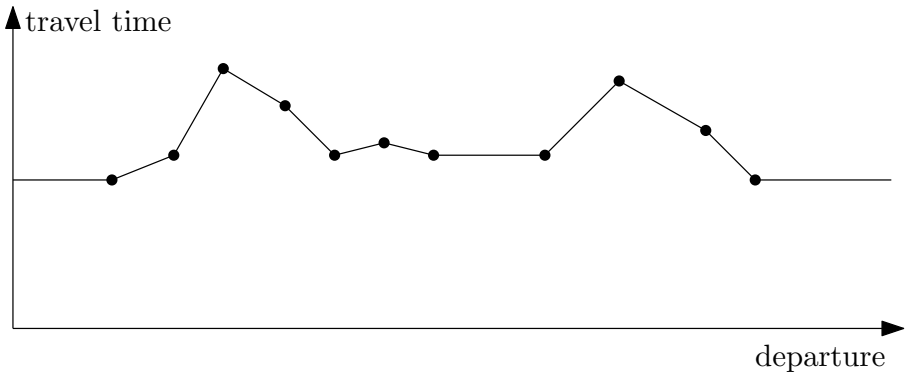
Zeitabhängige Straßennetzwerke

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



Diskussion

Eigenschaften “Zeitabhängigkeit”:

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

Diskussion

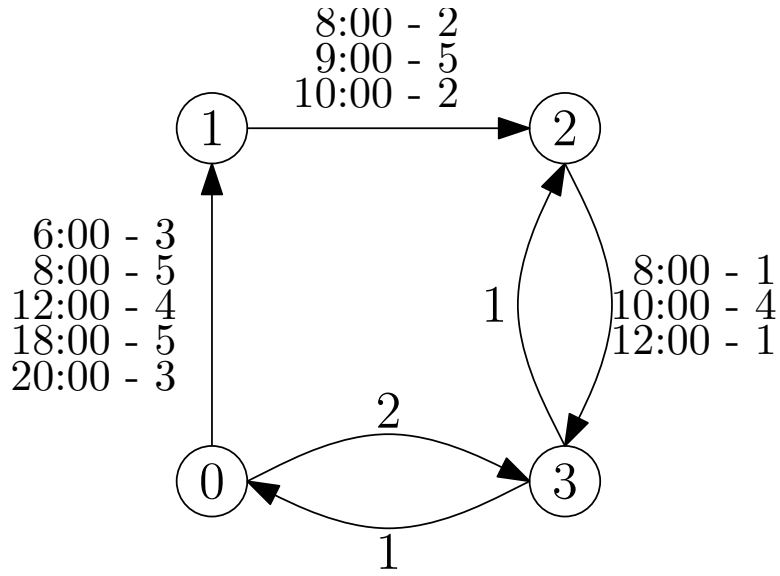
Eigenschaften “Zeitabhängigkeit”:

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

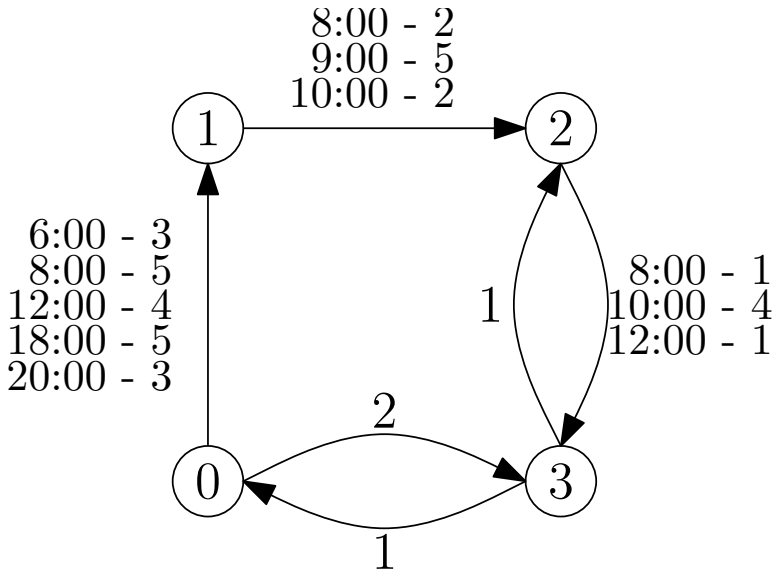
Voraussetzung:

- FIFO gilt auf allen Kanten

Datenstruktur



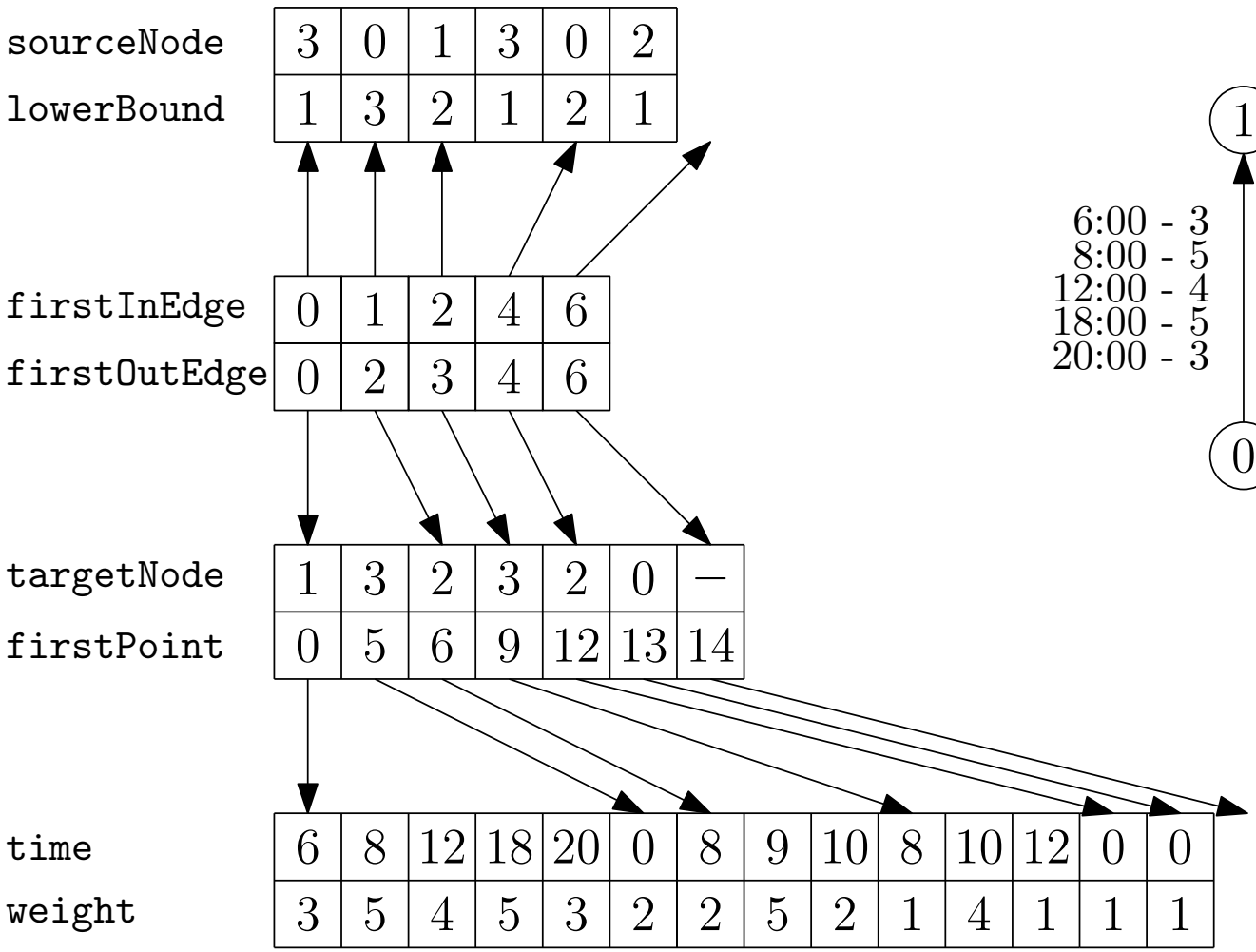
Datenstruktur

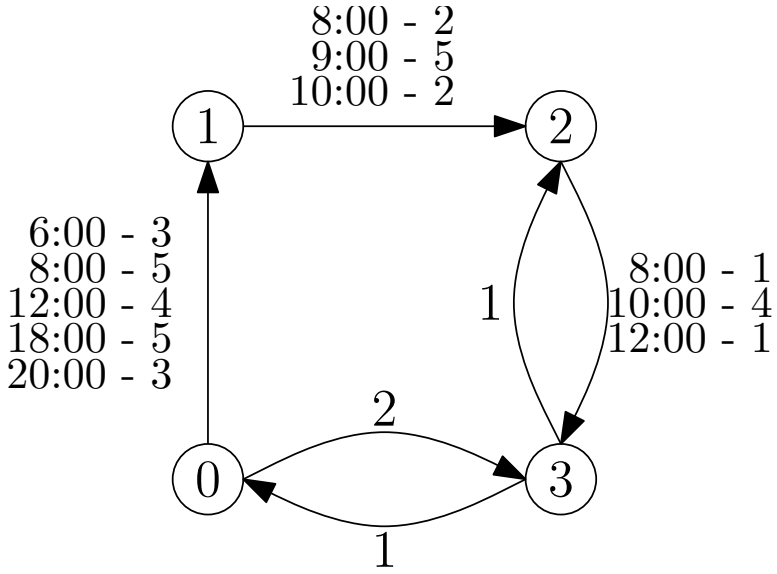


firstEdge	0	2	3	4	6		
targetNode	1	3	2	3	2	0	–
firstPoint	0	5	6	9	12	13	14

time	6	8	12	18	20	0	8	9	10	8	10	12	0	0
weight	3	5	4	5	3	2	2	5	2	1	4	1	1	1

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

Ziel: finde kürzesten Weg für Abfahrtszeit τ

`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          if d_τ[u] + len(e, τ + d_τ[u]) < d_τ[v] then
7              d_τ[v] ← d_τ[u] + len(e, τ + d_τ[u])
8              p_τ[v] ← u
9              if v ∈ Q then Q.decreaseKey(v, d_τ[v])
10
11             else Q.insert(v, d_τ[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:

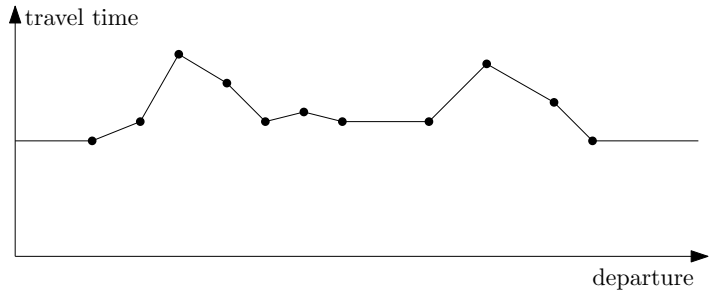
- Im Kreis fahren kann sich lohnen
- NP-schwer (wenn warten an Knoten nicht erlaubt ist)
- Transportnetzwerke sind meist FIFO modellierbar

Auswertung

Evaluation von $f(\tau)$:

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

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

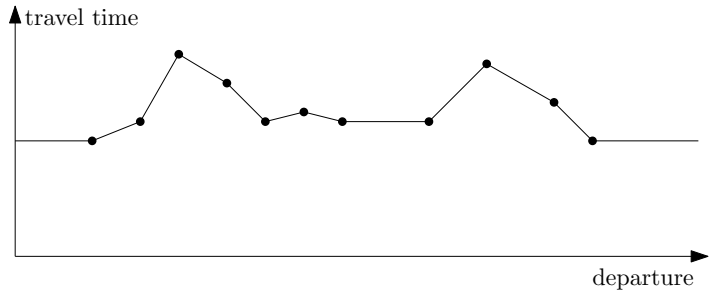


Auswertung

Evaluation von $f(\tau)$:

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

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



Problem:

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

Profil-Anfragen

Ziel: finde kürzesten Weg für alle Abfahrtszeitpunkte

`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
10             else Q.insert(v, d[v])
```

Diskussion Profil-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
- Vergleich $\not\geq$
  (ist analog zu Minimumsbildung)

Linken - Reisezeitfunktionen

Definition

Seien $f : \mathbb{R}_0^+ \to \mathbb{R}_0^+$ und $g : \mathbb{R}_0^+ \to \mathbb{R}_0^+$ zwei Reisezeitfunktionen 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 - Ankunftszeitfunktionen

Definition

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

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

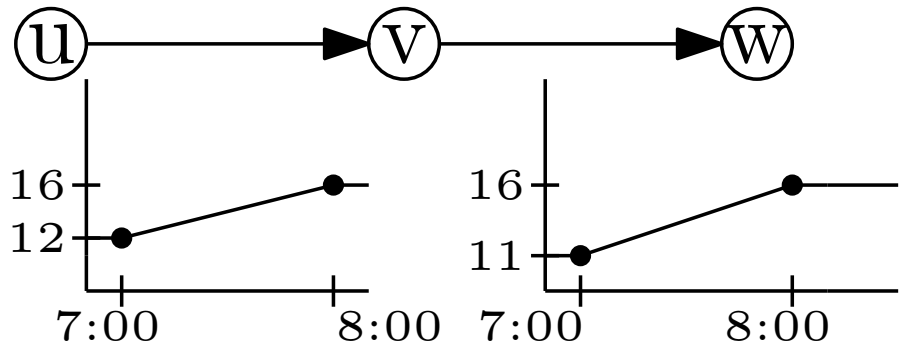
Oder

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

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

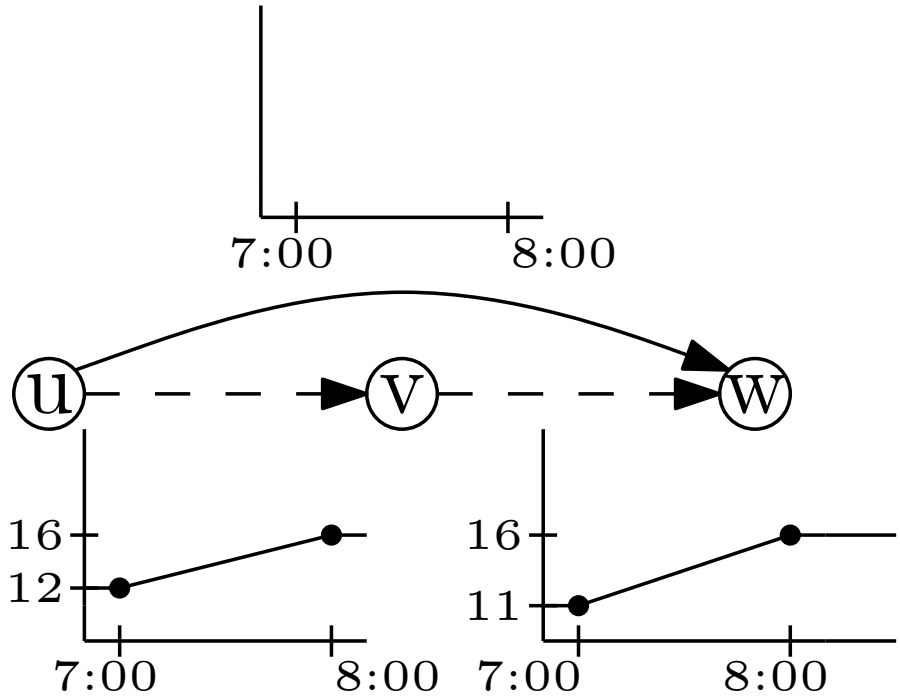
Linken

Linken zweier Funktionen f und g



Linken

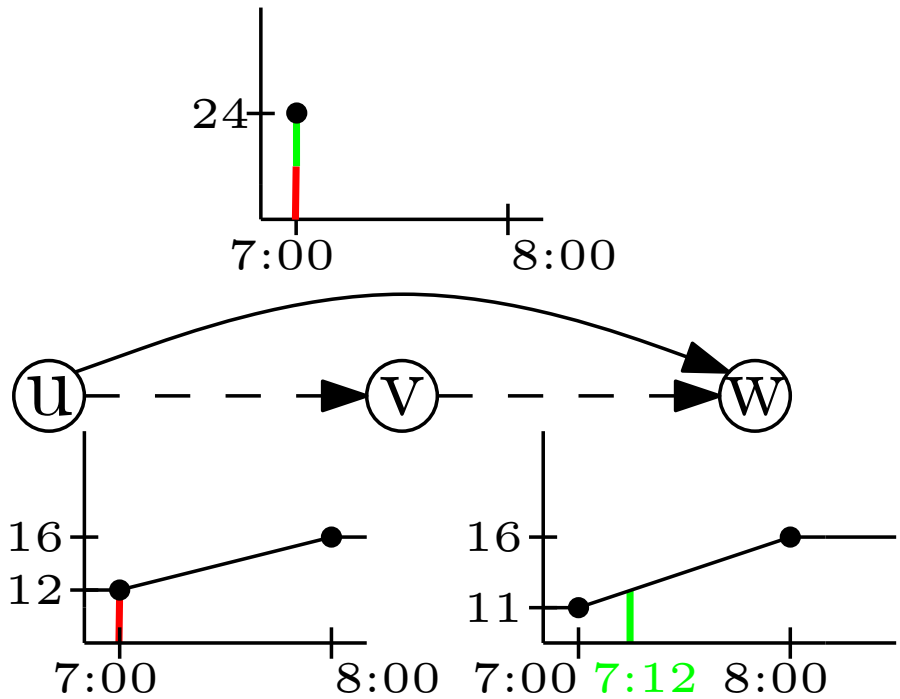
Linken zweier Funktionen f und g



Linken

Linken zweier Funktionen f und g

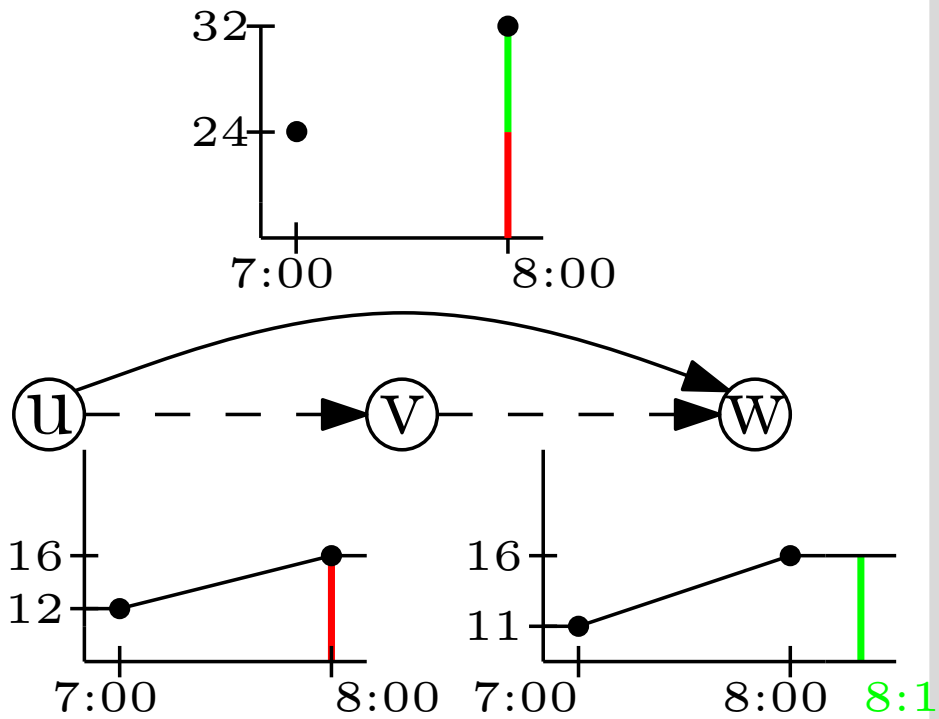
- $f \oplus g$ enthält auf jeden Fall
  $\{(t_1^f, w_1^f + g(t_1^f + w_1^f)), \ldots, (t_l^f, w_l^f + g(t_l^f + w_l^f))\}$



Linken

Linken zweier Funktionen f und g

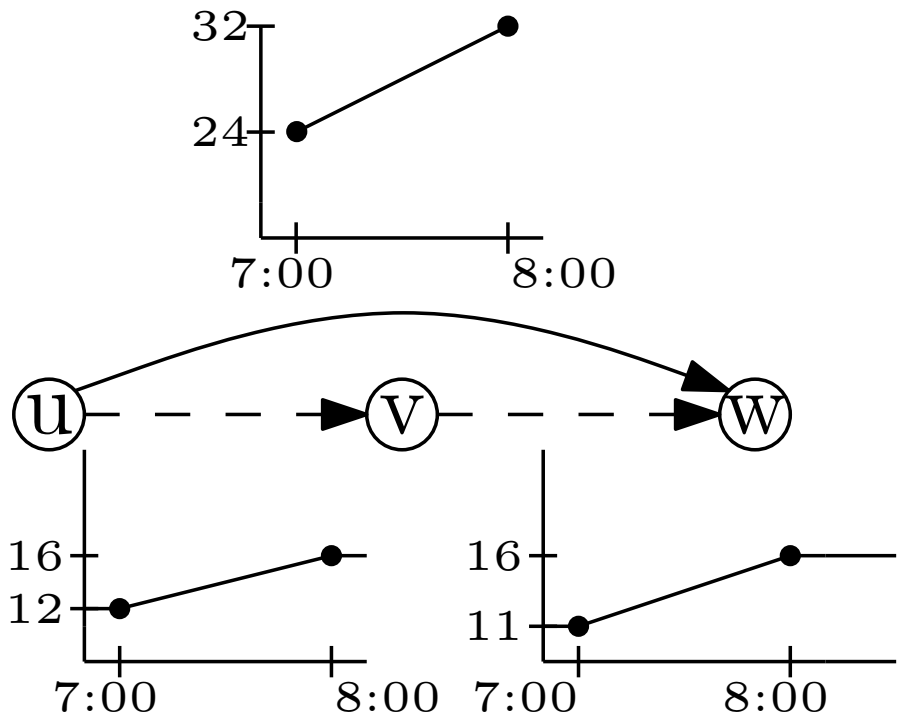
- $f \oplus g$ enthält auf jeden Fall
  $\{(t_1^f, w_1^f + g(t_1^f + w_1^f)), \ldots, (t_l^f, w_l^f + g(t_l^f + w_l^f))\}$



Linken

Linken zweier Funktionen f und g

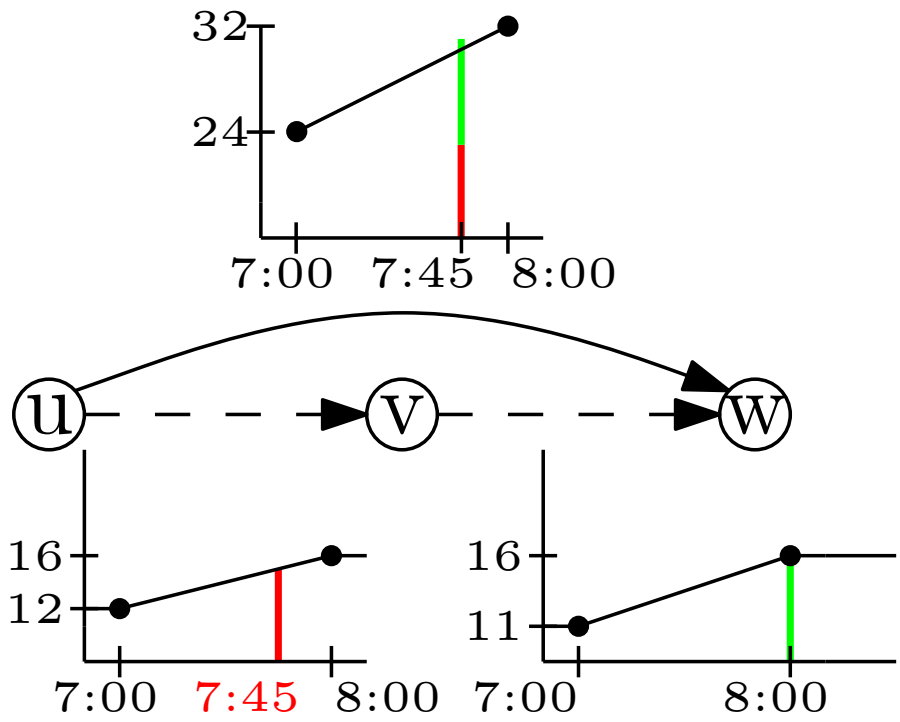
- $f \oplus g$ enthält auf jeden Fall
  $\{(t_1^f, w_1^f + g(t_1^f + w_1^f)), \ldots, (t_l^f, w_l^f + g(t_l^f + w_l^f))\}$



Linken

Linken zweier Funktionen f und g

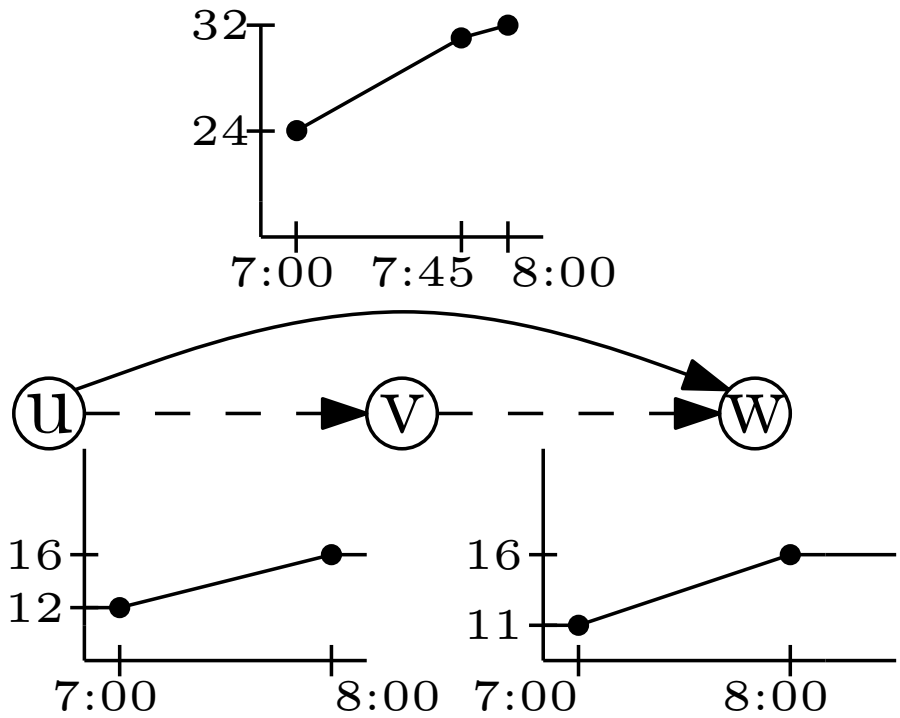
- $f \oplus g$ enthält auf jeden Fall
  $\{(t_1^f, w_1^f + g(t_1^f + w_1^f)), \ldots, (t_l^f, w_l^f + g(t_l^f + w_l^f))\}$



Linken

Linken zweier Funktionen f und g

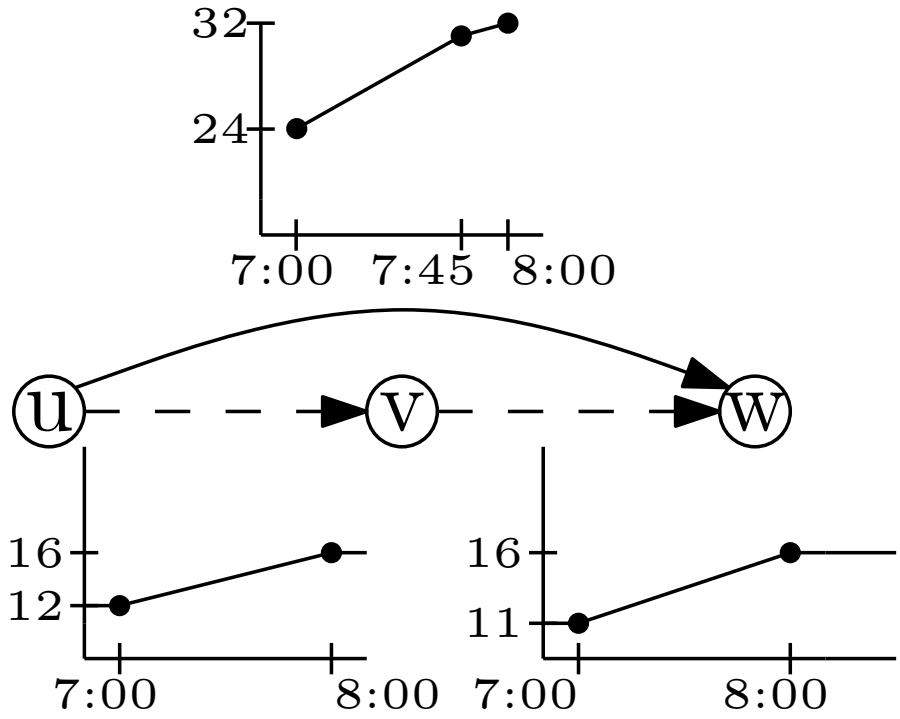
- $f \oplus g$ enthält auf jeden Fall $\{(t_1^f, w_1^f + g(t_1^f + w_1^f)), \ldots, (t_l^f, w_l^f + g(t_l^f + w_l^f))\}$
- Zusätzliche Interpolationspunkte an t_j^{-1} mit $f(t_j^{-1}) + t_j^{-1} = t_j^g$



Linken

Linken zweier Funktionen f und g

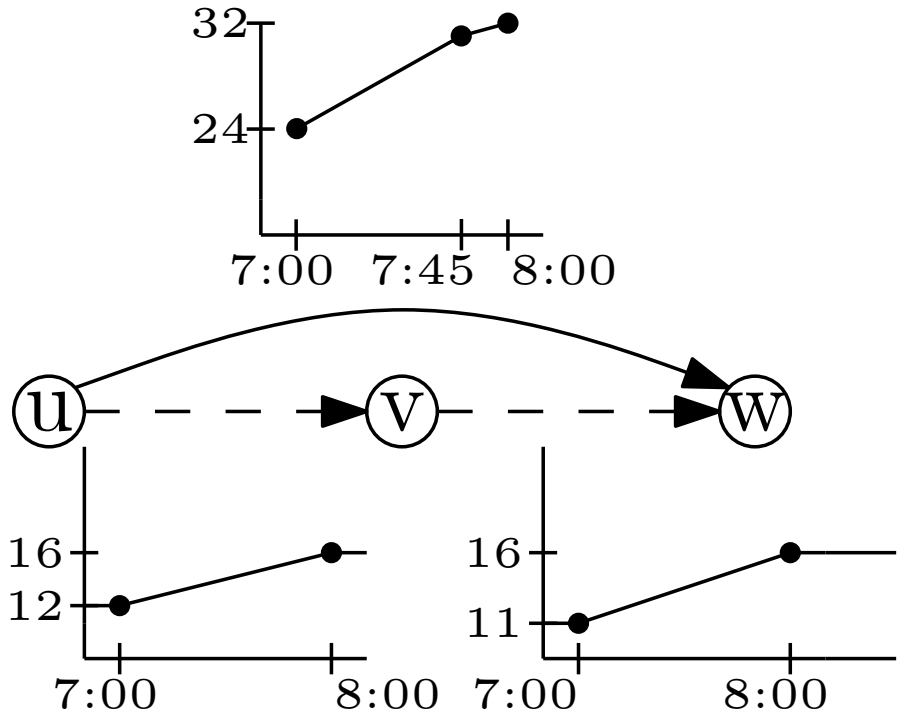
- $f \oplus g$ enthält auf jeden Fall $\{(t_1^f, w_1^f + g(t_1^f + w_1^f)), \ldots, (t_l^f, w_l^f + g(t_l^f + w_l^f))\}$
- Zusätzliche Interpolationspunkte an t_j^{-1} mit $f(t_j^{-1}) + t_j^{-1} = t_j^g$
- Füge $(t_j^{-1}, f(t_j^{-1}) + w_j^g)$ für alle Punkte von g zu $f \oplus g$



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



Funktionsinvertierung

Ankunftszeit-Funktion $f^{-}1$ an Stelle x auswerten:

- Seien (x_i, y_i) die Interpolationspunkte von f
- Die Interpolationspunkte von $f^{-}1$ sind (y_i, x_i)

Funktionsinvertierung

Ankunftszeit-Funktion $f^{-}1$ an Stelle x auswerten:

- Seien (x_i, y_i) die Interpolationspunkte von f
- Die Interpolationspunkte von $f^{-}1$ sind (y_i, x_i)
- Funktionsevaluation analog zu der von f
- Die Interpolationspunkte von f^{-1} müssen nicht explizit gespeichert werden

Diskussion Linken

Laufzeit

- Sweep Algorithmus
- $\mathcal{O}(|l^f| + |l^g|)$
- 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

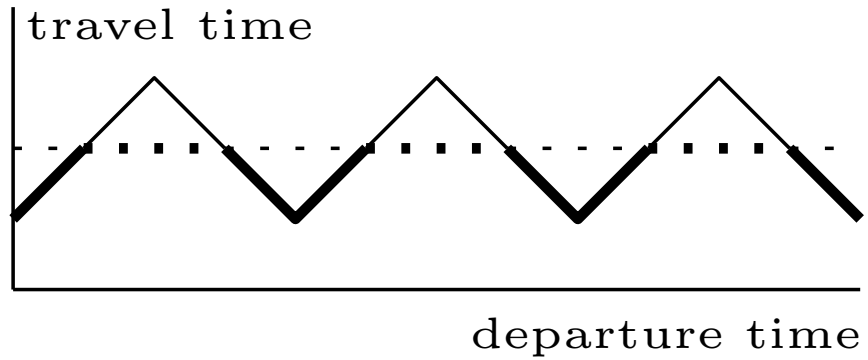
Problem:

- 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



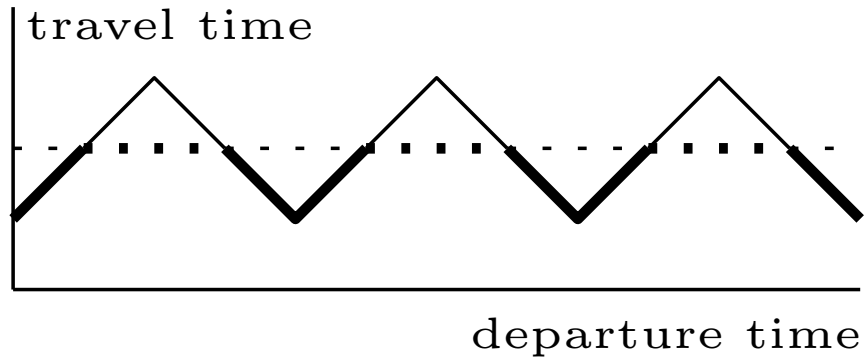
Merge von Funktionen

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

Speicherverbrauch

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

Problem:

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

Linken & Mergen: Probleme

Problem 1: Anzahl an Interpolationspunkte

- Je länger ein Shortcut je mehr Interpolationspunkte hat er
- Die Suchräume schrumpfen drastisch bezüglich der Anzahl an Knoten und Kanten
- aber viel weniger bezüglich der Anzahl an Interpolationspunkte

Linken & Mergen: Probleme

Problem 2: Numerische Instabilität

- Linken und Mergen nicht genau falls mit Gleitkommazahlen implementiert
- Fehler akkumulieren sich falls man Linken und Mergen verkettet

Linken & Mergen: Probleme

Problem 2: Numerische Instabilität

- Linken und Mergen nicht genau falls mit Gleitkommazahlen implementiert
- Fehler akkumulieren sich falls man Linken und Mergen verkettet
- Aber: Alle Operation bleiben in den rationalen Zahlen.

Linken & Mergen: Probleme

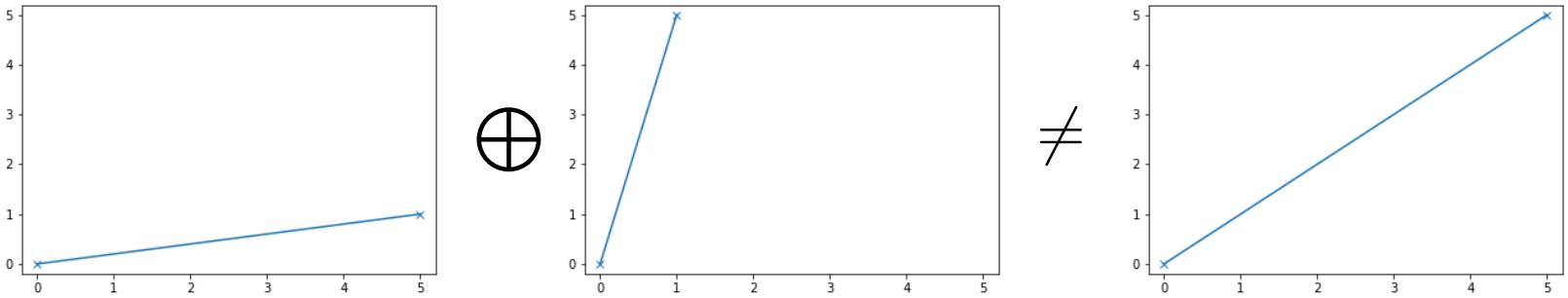
Problem 2: Numerische Instabilität

- Linken und Mergen nicht genau falls mit Gleitkommazahlen implementiert
- Fehler akkumulieren sich falls man Linken und Mergen verkettet
- Aber: Alle Operation bleiben in den rationalen Zahlen.
  - Mit Brüchen arbeiten?
  - Vorläufige Experimente zeigen, dass Teiler und Nenner unkontrolliert wachsen
  - Offene Frage: Kann man diese Divergenz zeigen?

Linken & Mergen: Probleme

Problem 2: Numerische Instabilität

- Linken und Mergen nicht genau falls mit Gleitkommazahlen implementiert
- Fehler akkumulieren sich falls man Linken und Mergen verkettet
- Aber: Alle Operation bleiben in den rationalen Zahlen.
  - Mit Brüchen arbeiten?
  - Vorläufige Experimente zeigen, dass Teiler und Nenner unkontrolliert wachsen
  - Offene Frage: Kann man diese Divergenz zeigen?
- Integer auch kaputt



Eingabe

- Netzwerk Deutschland $|V| \approx 4.7$ Mio., $|E| \approx 10.8$ Mio. von 2006
- Standard-Instanz in der Forschung
- Basierend auf realen Verkehrsdaten
- 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

Mehr Instanzen

	Nodes [K]	Arcs [K]	TD arcs [%]	Avg. $\lvert f \rvert$ per TD arc
Ger06	4 688	10 796	8	17.6
SynEur	18 010	42 189	0.1	13.2
Ger17	7 248	15 752	29	29.6
Eur17	25 758	55 504	27	27.5

Profilsuchen

Beobachtung:

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

$\Rightarrow$ inpraktikabel

Zusammenfassung

Zeitabhängige Netzwerke (Basics)

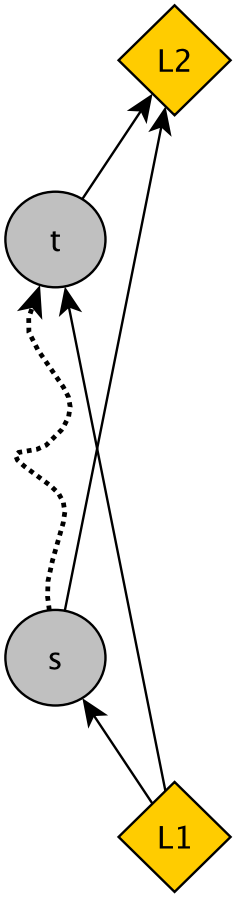
- Funktionen statt Konstanten an Kanten
- Operationen werden teurer
  - $\mathcal{O}(\log |f|)$ für Auswertung
  - $\mathcal{O}(|f^f| + |f^g|)$ für Linken und Minimum
  - Speicherverbrauch explodiert
- Zeitanfragen:
  - Normaler Dijkstra
  - Kaum langsamer (lediglich Auswertung)
- Profilanfragen
  -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

Beschleunigungstechniken

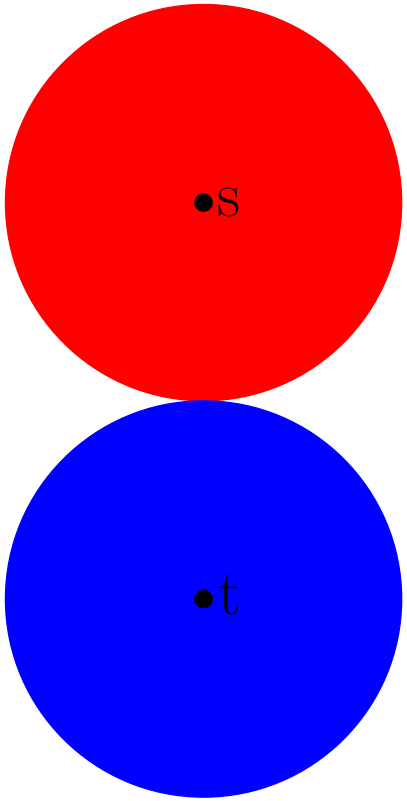
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Beschleunigungstechniken

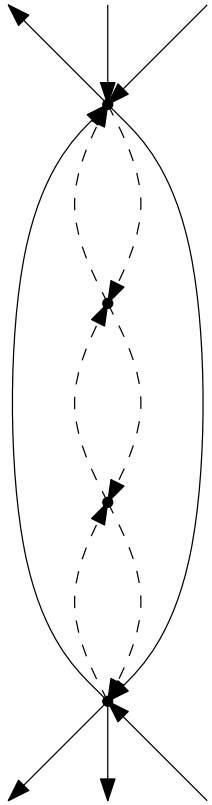
Landmarken



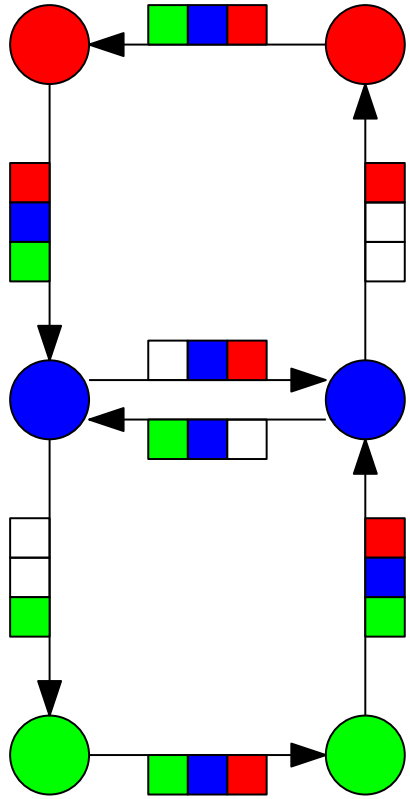
Bidirektionale Suche



Kontraktion



Arc-Flags



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Landmarken

Vorberechnung:

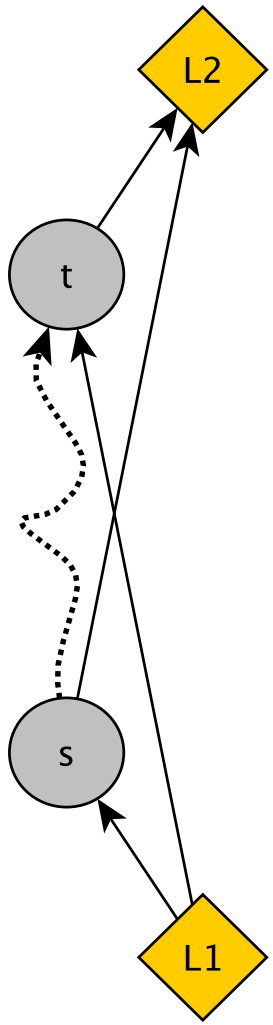
- wähle eine Hand voll (≈ 16) Knoten als Landmarken
- berechne Abstände von und zu allen Landmarken

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



Anpassung

Beobachtung:

- Korrektheit von ALT basiert darauf, dass reduzierte Kosten größer oder gleich 0 sind

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

- durch Erhöhen der Kantengewichte wird dies nicht verletzt

Anpassung

Beobachtung:

- Korrektheit von ALT basiert darauf, dass reduzierte Kosten größer oder gleich 0 sind

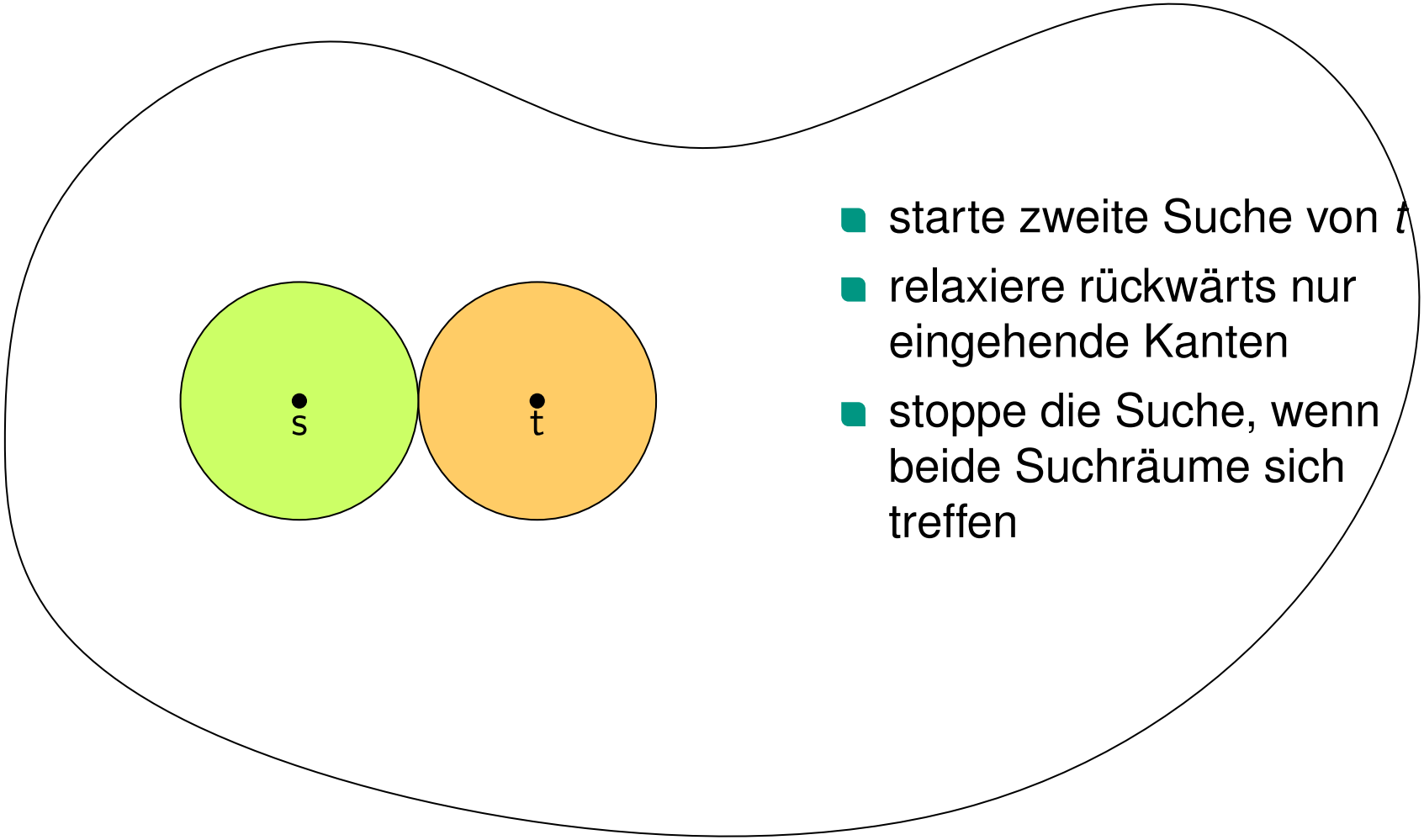
$$\text{len}_\pi(u, v) = \text{len}(u, v) - \pi(u) + \pi(v) \overset{!}{\geq} 0$$

- durch Erhöhen der Kantengewichte wird dies nicht verletzt

Somit:

- Definiere lowerbound-Graph $\underline{G} = (V, E, \underline{\text{len}})$ mit $\underline{\text{len}} := \min \text{len}$
- Vorberechnung auf lowerbound-Graph
- korrekt aber eventuell langsamere Anfragezeiten

Bidirektionale Suche



- starte zweite Suche von t
- relaxiere rückwärts nur eingehende Kanten
- stoppe die Suche, wenn beide Suchräume sich treffen

Anpassung

Zeitanfragen:

- Ankunft unbekannt $\Rightarrow$ Rückwärtsuche?

Anpassung

Zeitanfragen:

- Ankunft unbekannt $\Rightarrow$ Rückwärtsuche?
- Rückwärtssuche nur zum Einschränken der Vorwärtssuche benutzen
- je nach Beschleunigungstechnik verschieden $\rightsquigarrow$ später

Anpassung

Zeitanfragen:

- Ankunft unbekannt $\Rightarrow$ Rückwärtsuche?
- Rückwärtssuche nur zum Einschränken der Vorwärtssuche benutzen
- je nach Beschleunigungstechnik verschieden $\rightsquigarrow$ später

Profilanfragen:

- Anfrage zu allen Startzeitpunkten
- somit Rückwärtsuche kein Problem
- μ: tentative Abstandsfunktion
- breche ab, wenn $\text{minKey}(\overrightarrow{Q}) + \text{minKey}(\overleftarrow{Q}) \geq \overline{\mu}$
  Erinnere: key von v ist der lowerbound seiner Profilfunktion

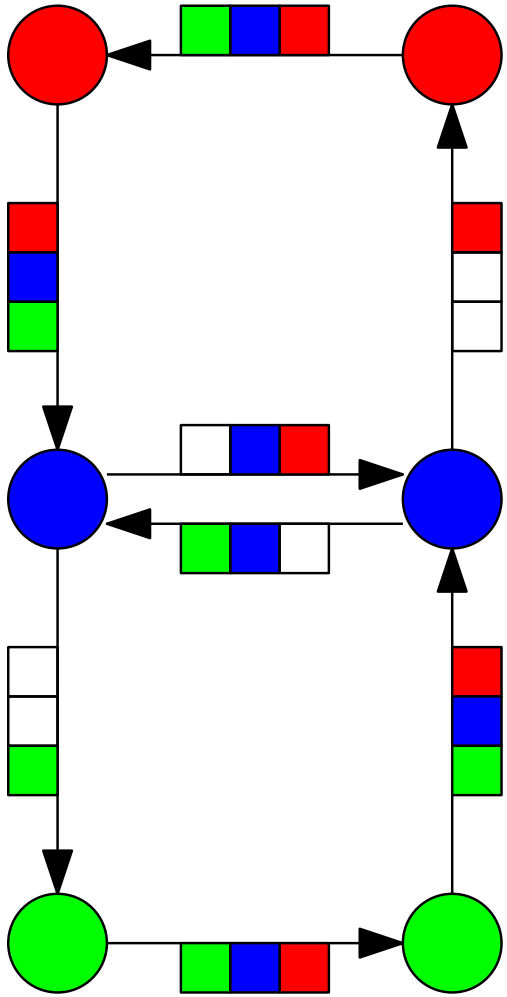
Arc-Flags

Idee:

- partitioniere den Graph in k Zellen
- hänge ein Label mit k Bits an jede Kante
- zeigt ob e wichtig für die Zielzelle ist
- modifizierter Dijkstra überspringt unwichtige Kanten

Beobachtung:

- Partition wird auf ungewichtetem Grahen durchgeführt
- Flaggen müssen allerdings angepasst werden



Anpassung

Idee:

- ändere Intuition einer gesetzten Flagge
- Konzept bleibt gleich: Eine Flagge pro Kante und Region
- setze Flagge wenn Kante mindestens ein mal am Tag “wichtig” ist

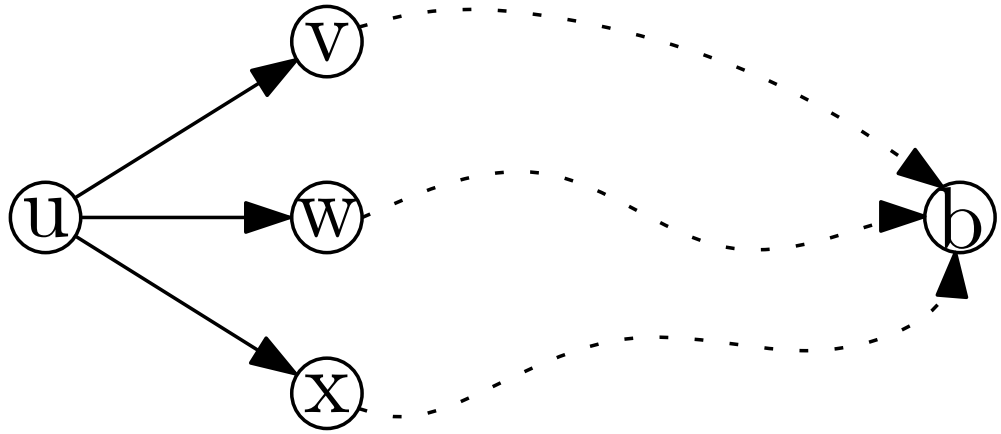
Anpassung

Idee:

- ändere Intuition einer gesetzten Flagge
- Konzept bleibt gleich: Eine Flagge pro Kante und Region
- setze Flagge wenn Kante mindestens ein mal am Tag “wichtig” ist

Anpassung:

- für alle Randknoten b und alle Knoten u:
- Berechne Abstandsfunktion $d_*(u, b)$
- setze Flagge wenn gilt $\text{len}(u, v) \oplus d_*(v, b) \not> d_*(u, b)$



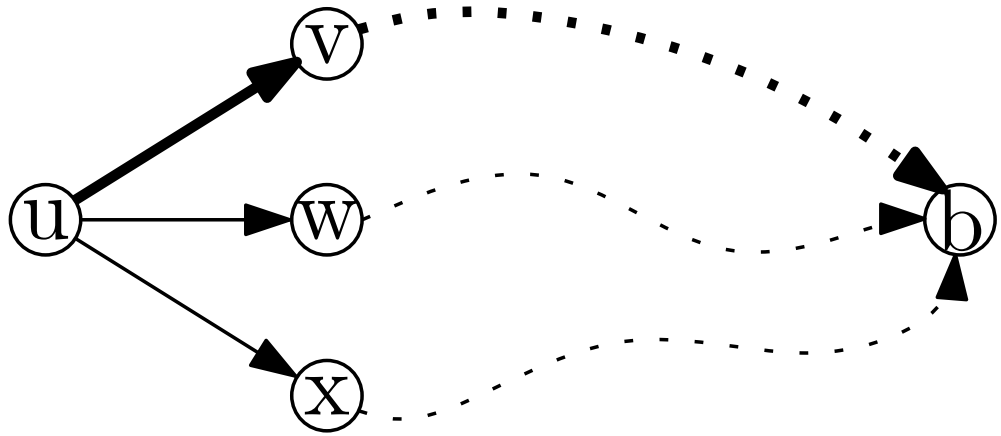
Anpassung

Idee:

- ändere Intuition einer gesetzten Flagge
- Konzept bleibt gleich: Eine Flagge pro Kante und Region
- setze Flagge wenn Kante mindestens ein mal am Tag “wichtig” ist

Anpassung:

- für alle Randknoten b und alle Knoten u:
- Berechne Abstandsfunktion $d_*(u, b)$
- setze Flagge wenn gilt $\text{len}(u, v) \oplus d_*(v, b) \not> d_*(u, b)$



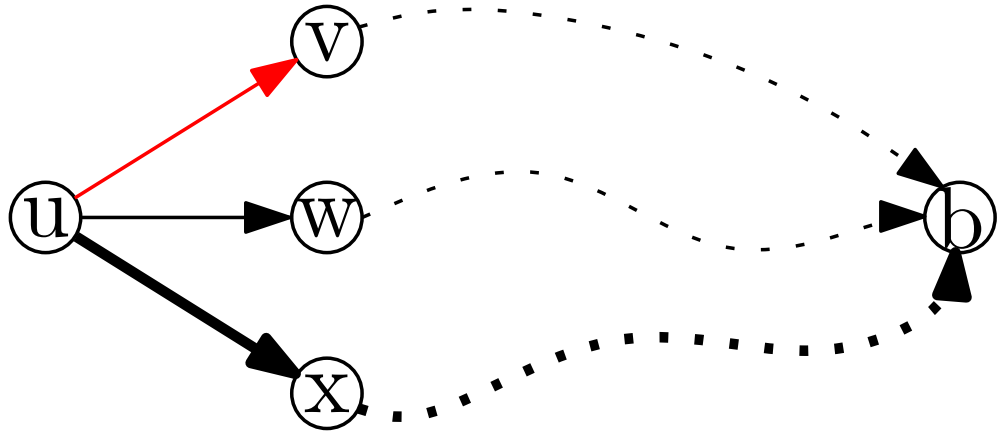
Anpassung

Idee:

- ändere Intuition einer gesetzten Flagge
- Konzept bleibt gleich: Eine Flagge pro Kante und Region
- setze Flagge wenn Kante mindestens ein mal am Tag “wichtig” ist

Anpassung:

- für alle Randknoten b und alle Knoten u:
- Berechne Abstandsfunktion $d_*(u, b)$
- setze Flagge wenn gilt $\text{len}(u, v) \oplus d_*(v, b) \not> d_*(u, b)$



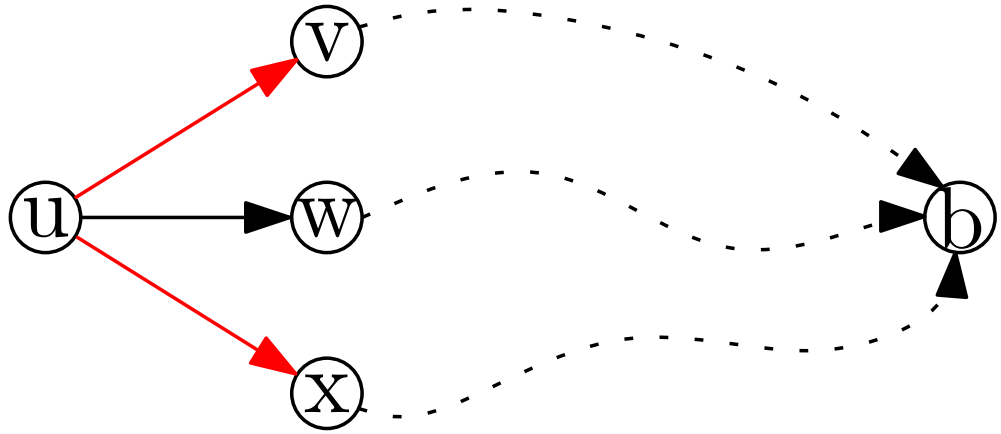
Anpassung

Idee:

- ändere Intuition einer gesetzten Flagge
- Konzept bleibt gleich: Eine Flagge pro Kante und Region
- setze Flagge wenn Kante mindestens ein mal am Tag “wichtig” ist

Anpassung:

- für alle Randknoten b und alle Knoten u:
- Berechne Abstandsfunktion $d_*(u, b)$
- setze Flagge wenn gilt $\text{len}(u, v) \oplus d_*(v, b) \not> d_*(u, b)$

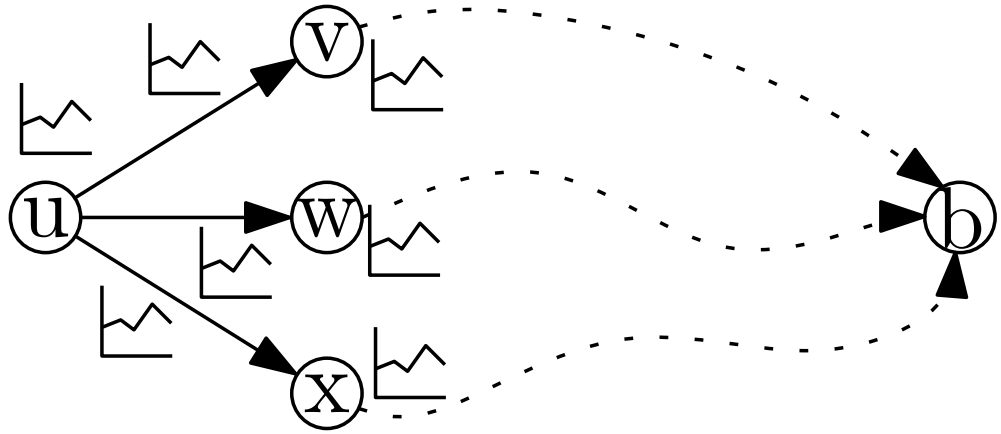


Approximation Arc-Flags

Beobachtung:

- viele Interpolationspunkte
- Berechnung der Abstandsfunktionen ist sehr zeitintensiv
- Laufzeit stark abhängig von der Komplexität der Funktionen

Idee:



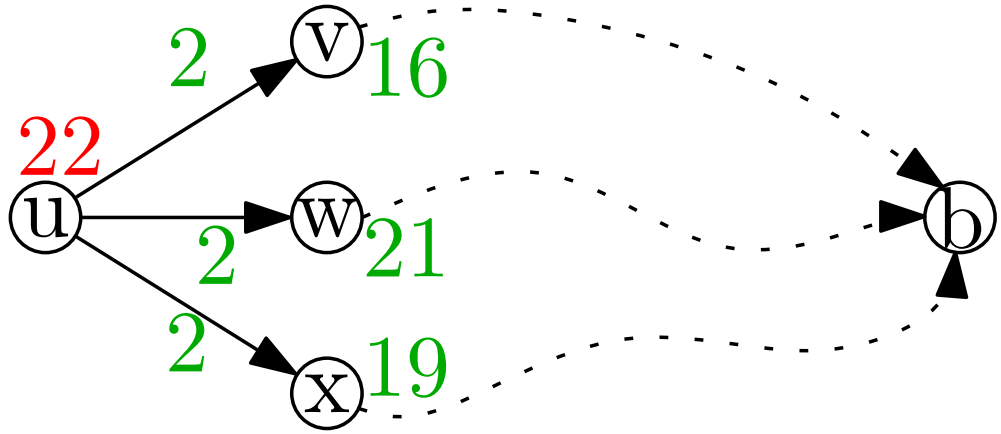
Approximation Arc-Flags

Beobachtung:

- viele Interpolationspunkte
- Berechnung der Abstandsfunktionen ist sehr zeitintensiv
- Laufzeit stark abhängig von der Komplexität der Funktionen

Idee:

- benutze über- and Unterapproximation



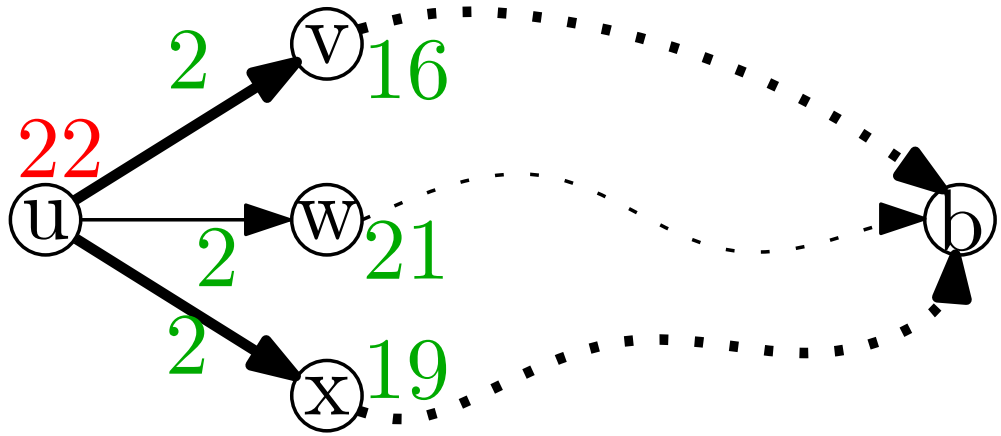
Approximation Arc-Flags

Beobachtung:

- viele Interpolationspunkte
- Berechnung der Abstandsfunktionen ist sehr zeitintensiv
- Laufzeit stark abhängig von der Komplexität der Funktionen

Idee:

- benutze über- and Unterapproximation



Approximation Arc-Flags

Beobachtung:

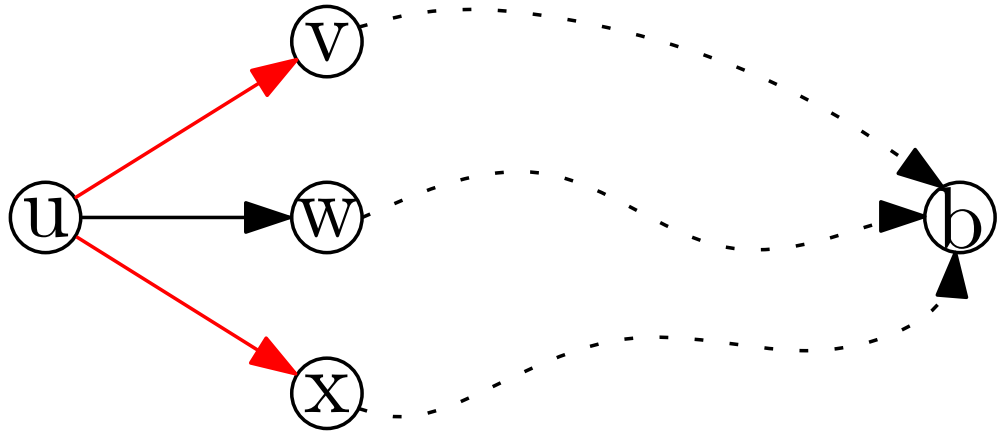
- viele Interpolationspunkte
- Berechnung der Abstandsfunktionen ist sehr zeitintensiv
- Laufzeit stark abhängig von der Komplexität der Funktionen

Idee:

- benutze über- and Unterapproximation

$\Rightarrow$ schnellere Vorberechnung, langsamere Anfragen

$\Rightarrow$ aber immer noch korrekt



Heuristische Flaggen

Idee:

- führe von jedem Randknoten K Zeitanfragen aus
- mit fester Ankunftszeit
- setze Flagge, wenn Kante auf einem dem Bäume eine Baumkante ist

Heuristische Flaggen

Idee:

- führe von jedem Randknoten K Zeitanfragen aus
- mit fester Ankunftszeit
- setze Flagge, wenn Kante auf einem dem Bäume eine Baumkante ist

Beobachtungen:

- Flaggen eventuell nicht korrekt
- ein Pfad wird aber immer gefunden
- Fehlerrate?

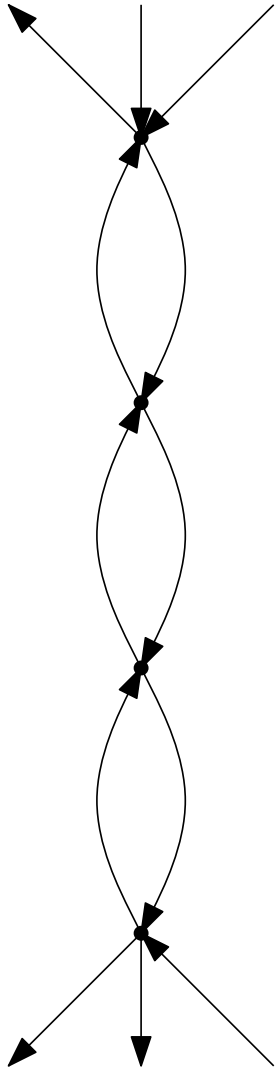
Kontraktion

Knoten-Reduktion:

- entferne Knoten
- füge neue Kanten (Shortcuts) hinzu, um die Abstände zwischen verbleibenden Knoten zu erhalten

Zeugensuche:

- behalte nur relevante Shortcuts
- lokale Suche während oder nach Knoten-Reduktion



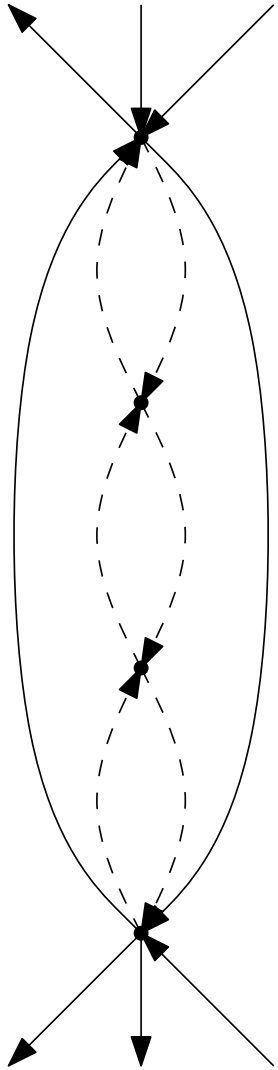
Kontraktion

Knoten-Reduktion:

- entferne Knoten
- füge neue Kanten (Shortcuts) hinzu, um die Abstände zwischen verbleibenden Knoten zu erhalten

Zeugensuche:

- behalte nur relevante Shortcuts
- lokale Suche während oder nach Knoten-Reduktion



Anpassung Zeugensuche

Zeitunabhängig:

- Kante (u, v) nicht nötig, wenn (u, v) nicht Teil des kürzesten Weges von u nach v ist, also $\text{len}(u, v) > d(u, v)$
- lokale Dijkstra-Suche von u

Zeitabhängig:

Anpassung Zeugensuche

Zeitunabhängig:

- Kante (u, v) nicht nötig, wenn (u, v) nicht Teil des kürzesten Weges von u nach v ist, also $\text{len}(u, v) > d(u, v)$
- lokale Dijkstra-Suche von u

Zeitabhängig:

- Kante (u, v) nicht nötig, wenn (u, v) nicht Teil eines kürzesten Wege von u nach v ist, also $\text{len}(u, v) > d_*(u, v)$
- lokale Profilsuche
- Problem: deutlich langsamer

Korridorsuche

Idee:

- führe zunächst zwei Dijkstra-Suchen mit $\underline{\text{len}}$ und $\overline{\text{len}}$ durch
- relaxiere dann nur solche Kanten (u, v), für die $\underline{d(s,u)} + \underline{\text{len}(u,v)} \leq \overline{d(s,v)}$ gilt
- lokale Profilsuche in diesem Korridor

Anmerkung:

- auch zur Beschleunigung von s-t Profil-Suchen

Approximation der Shortcuts

Problem:

- hoher Speicherbedarf der Shortcuts

Ideen:

- Shortcuts nur approximieren, inexakte Anfragen
- Keine Gewichte am Shortcut speichern, stattdessen on-the-fly entpacken und Pfad linken spart Speicher, kostet Laufzeit
- speichere auf Shortcuts Über- und Unterapproximation der Funktionen
  - induzieren wieder Korridor (aber genaueren als nur Min/Max!)
  - entpacke Shortcuts im Korridor, dies gibt einen Teil des Originalgraphen
  - benutze nun die nicht-approximierten Originalkanten für eine exakte Suche

Anpassung der Basismodule

Basismodule:

0 Bidirektionale Suche
+ Landmarken
+ Kontraktion
+ Arc-Flags

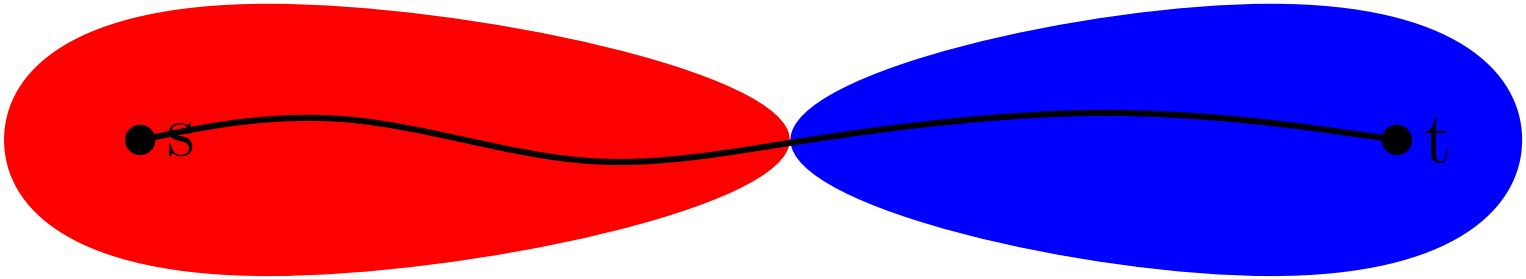
Somit sind folgende Algorithmen gute Kandidaten

- ALT
- Core-ALT
- SHARC
- Contraction Hierarchies
- MLD (CRP)

Bidirektionaler zeitabhängiger ALT

Idee - Drei Phasen:

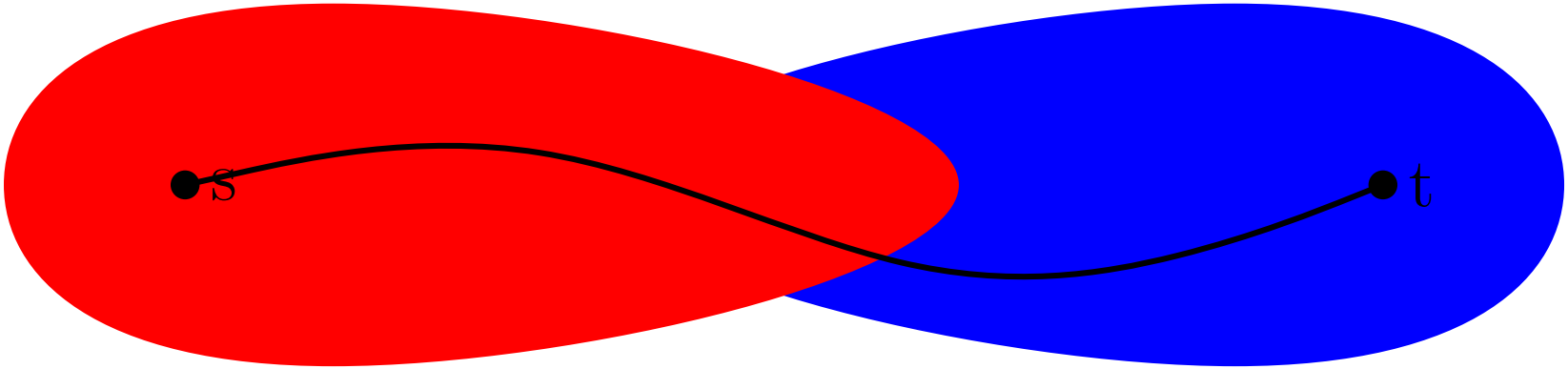
Bidirektionaler zeitabhängiger ALT



Idee - Drei Phasen:

1. Vorwärts zeitabhängig, Rückwärtssuche benutzt Minima der Funktionen. Fertig wenn Suchen sich treffen. Berechne zeitabhängige tentative Distanz μ
   - Distanz der Rückwärtssuche: untere Schranke $\Rightarrow$ nicht geeignet
   - Variante 1: Durch Auswerten des gefundenen Pfades
   - Variante 2: Rückwärtssuche schleift auch Maxima durch

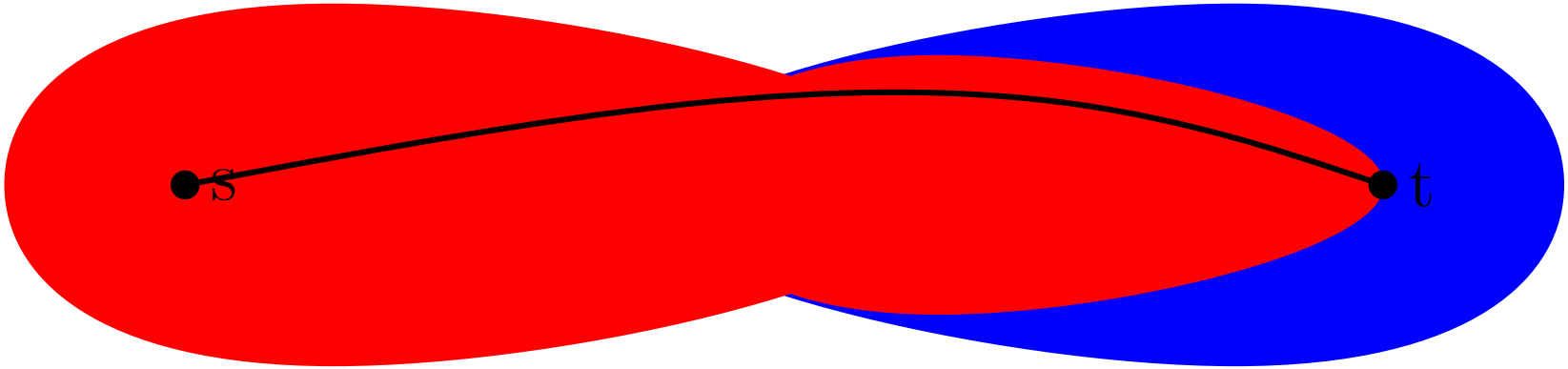
Bidirektionaler zeitabhängiger ALT



Idee - Drei Phasen:

1. Vorwärts zeitabhängig, Rückwärtssuche benutzt Minima der Funktionen. Fertig wenn Suchen sich treffen. Berechne zeitabhängige tentative Distanz μ
   - Distanz der Rückwärtssuche: untere Schranke $\Rightarrow$ nicht geeignet
   - Variante 1: Durch Auswerten des gefundenen Pfades
   - Variante 2: Rückwärtssuche schleift auch Maxima durch
2. Rückwärtssuche weiter bis $\text{minKey}(\overleftarrow{Q}) > \mu$

Bidirektionaler zeitabhängiger ALT



Idee - Drei Phasen:

1. Vorwärts zeitabhängig, Rückwärtssuche benutzt Minima der Funktionen. Fertig wenn Suchen sich treffen. Berechne zeitabhängige tentative Distanz μ
   - Distanz der Rückwärtssuche: untere Schranke $\Rightarrow$ nicht geeignet
   - Variante 1: Durch Auswerten des gefundenen Pfades
   - Variante 2: Rückwärtssuche schleift auch Maxima durch
2. Rückwärtssuche weiter bis $\text{minKey}(\overleftarrow{Q}) > \mu$
3. Vorwärtssuche arbeitet weiter bis t abgearbeitet worden ist und besucht nur Knoten, die die Rückwärtssuche zuvor besucht hat

Approximation

Beobachtung:

- Phase 2 läuft recht lange weiter, bis $\text{minKey}(\overleftarrow{Q}) > \mu$ gilt
- insbesondere dann schlecht, wenn die lower bounds stark vom echten Wert abweichen

Approximation:

- breche Phase 2 bereits ab, wenn $\text{minKey}(\overleftarrow{Q}) \cdot K > \mu$ gilt
- dann ist der berechnete Weg eine K-Approximation des kürzesten Weges

Core-ALT

Idee

- begrenze Beschleunigungstechnik auf kleinen Subgraphen (Kern)

Vorberechnung

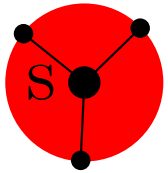
- kontrahiere Graphen zu einem Kern
- Landmarken nur im Kern

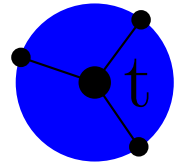
Anfrage

Core-ALT

Idee

- begrenze Beschleunigungstechnik auf kleinen Subgraphen (Kern)





Vorberechnung

- kontrahiere Graphen zu einem Kern
- Landmarken nur im Kern

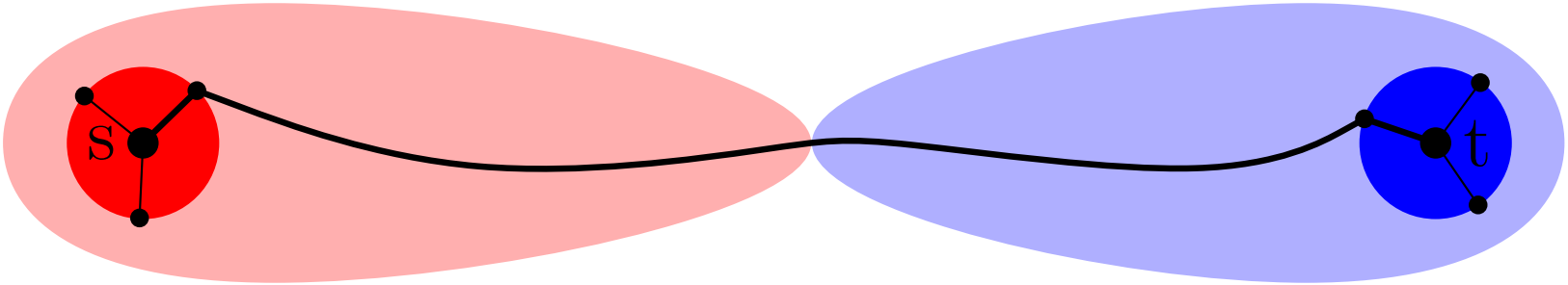
Anfrage

- Initialphase: normaler Dijkstra

Core-ALT

Idee

- begrenze Beschleunigungstechnik auf kleinen Subgraphen (Kern)



Vorberechnung

- kontrahiere Graphen zu einem Kern
- Landmarken nur im Kern

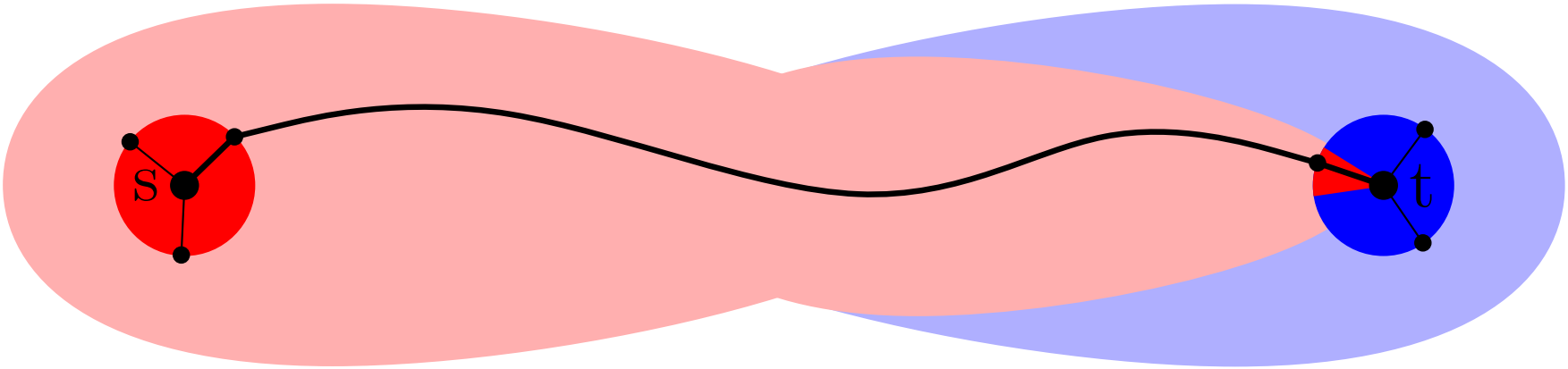
Anfrage

- Initialphase: normaler Dijkstra
- benutze Landmarken nur im Kern

Core-ALT

Idee

- begrenze Beschleunigungstechnik auf kleinen Subgraphen (Kern)



Vorberechnung

- kontrahiere Graphen zu einem Kern
- Landmarken nur im Kern

Anfrage

- Initialphase: normaler Dijkstra
- benutze Landmarken nur im Kern
- zeitabhängig:
  - Rückwärtssuche ist zeitunabhängig
  - Vorwärtssuche darf alle Knoten der Rückwärtssuche besuchen

SHARC (Kontraktion, Arc-Flags)

Vorberechnung:

- Multi-Level-Partition
- iterativer Prozess:
  - kontrahiere Subgraphen
  - berechne Flaggen
- Flaggenverfeinerung

Anpassung (grob):

- Kontraktion und Flaggen berechnung anpassen
- Verfeinerung durch (lokale) Profilsuchen

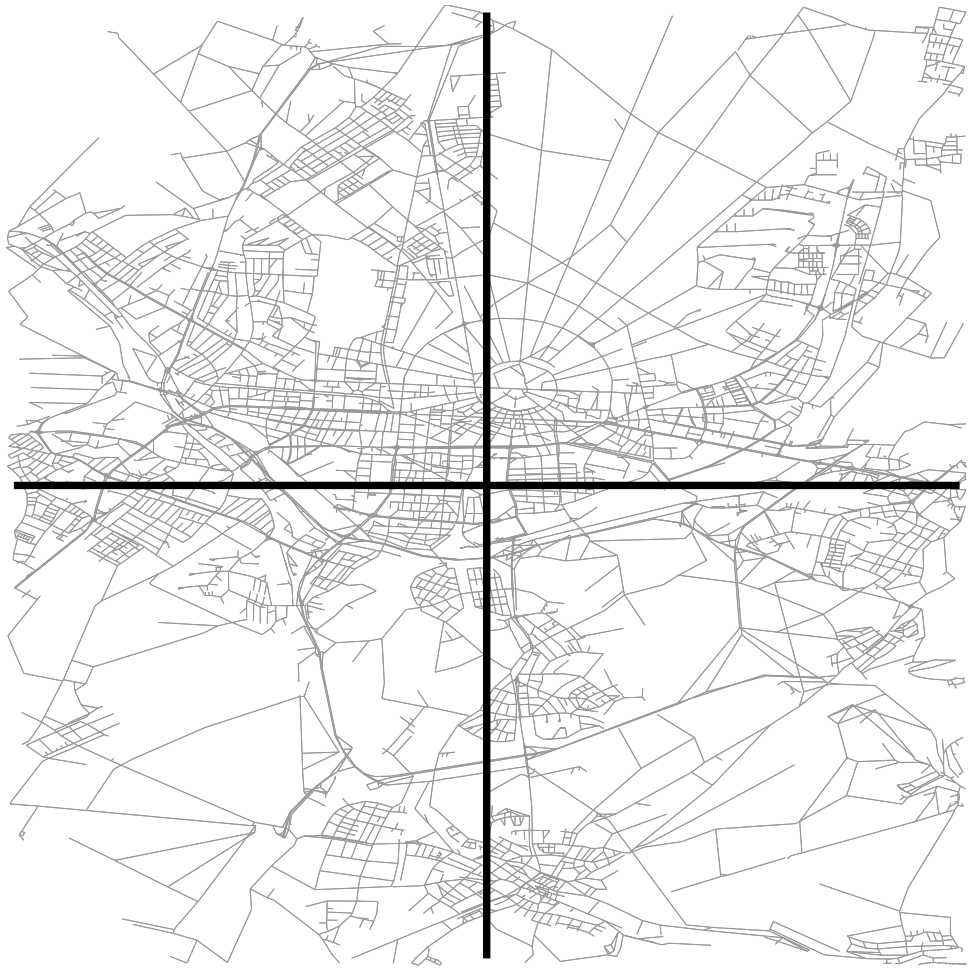
SHARC (Kontraktion, Arc-Flags)

Vorberechnung:

- Multi-Level-Partition
- iterativer Prozess:
  - kontrahiere Subgraphen
  - berechne Flaggen
- Flaggenverfeinerung

Anpassung (grob):

- Kontraktion und Flaggen berechnung anpassen
- Verfeinerung durch (lokale) Profilsuchen



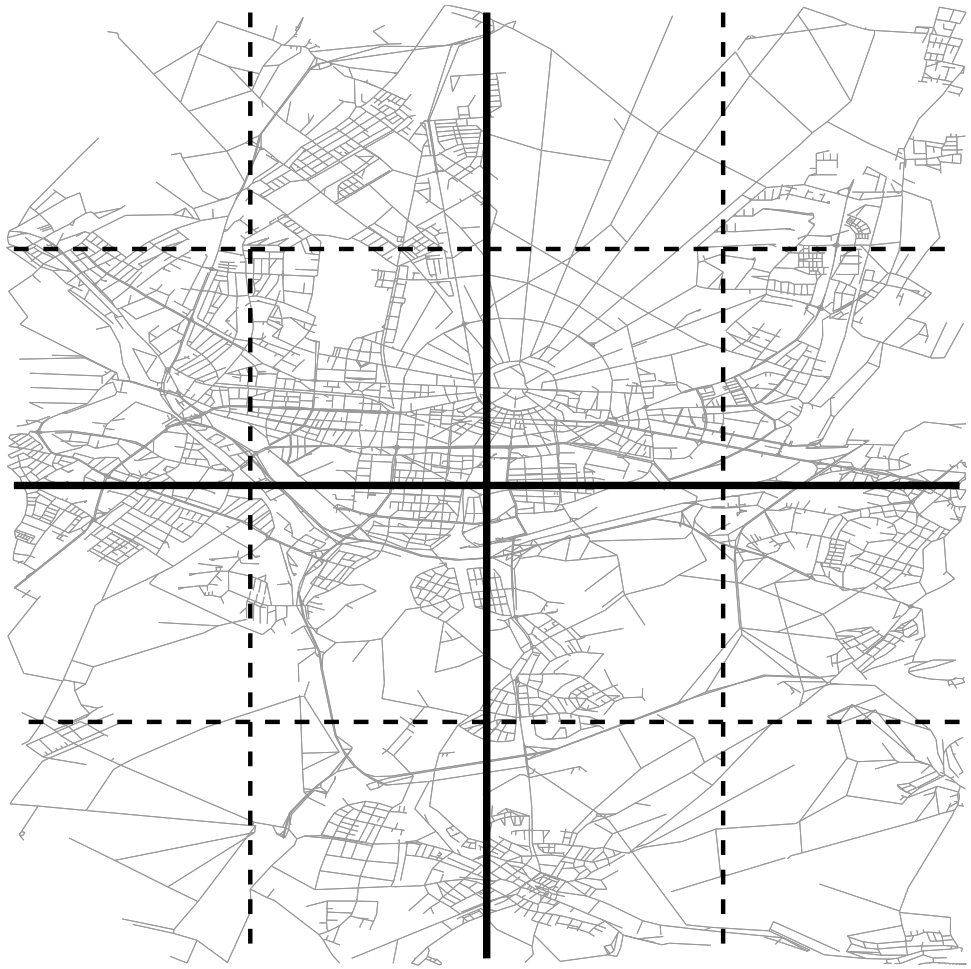
SHARC (Kontraktion, Arc-Flags)

Vorberechnung:

- Multi-Level-Partition
- iterativer Prozess:
  - kontrahiere Subgraphen
  - berechne Flaggen
- Flaggenverfeinerung

Anpassung (grob):

- Kontraktion und Flaggen berechnung anpassen
- Verfeinerung durch (lokale) Profilsuchen



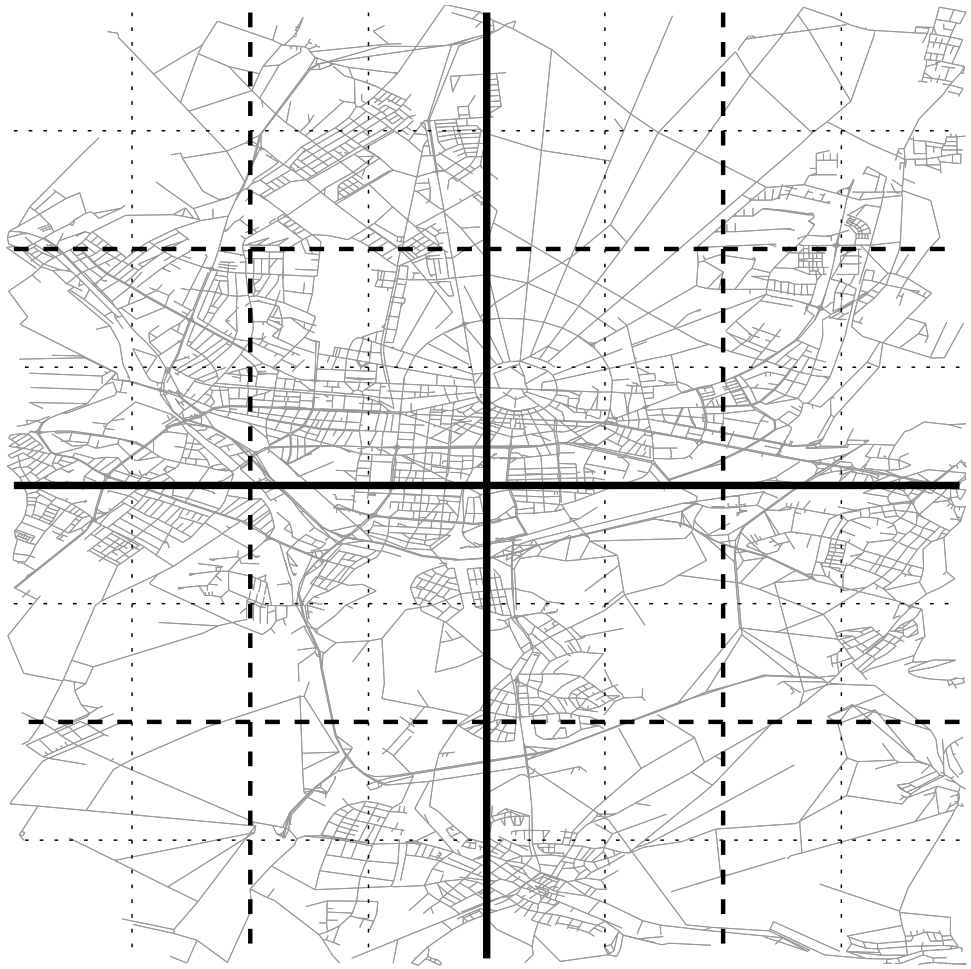
SHARC (Kontraktion, Arc-Flags)

Vorberechnung:

- Multi-Level-Partition
- iterativer Prozess:
  - kontrahiere Subgraphen
  - berechne Flaggen
- Flaggenverfeinerung

Anpassung (grob):

- Kontraktion und Flaggen berechnung anpassen
- Verfeinerung durch (lokale) Profilsuchen



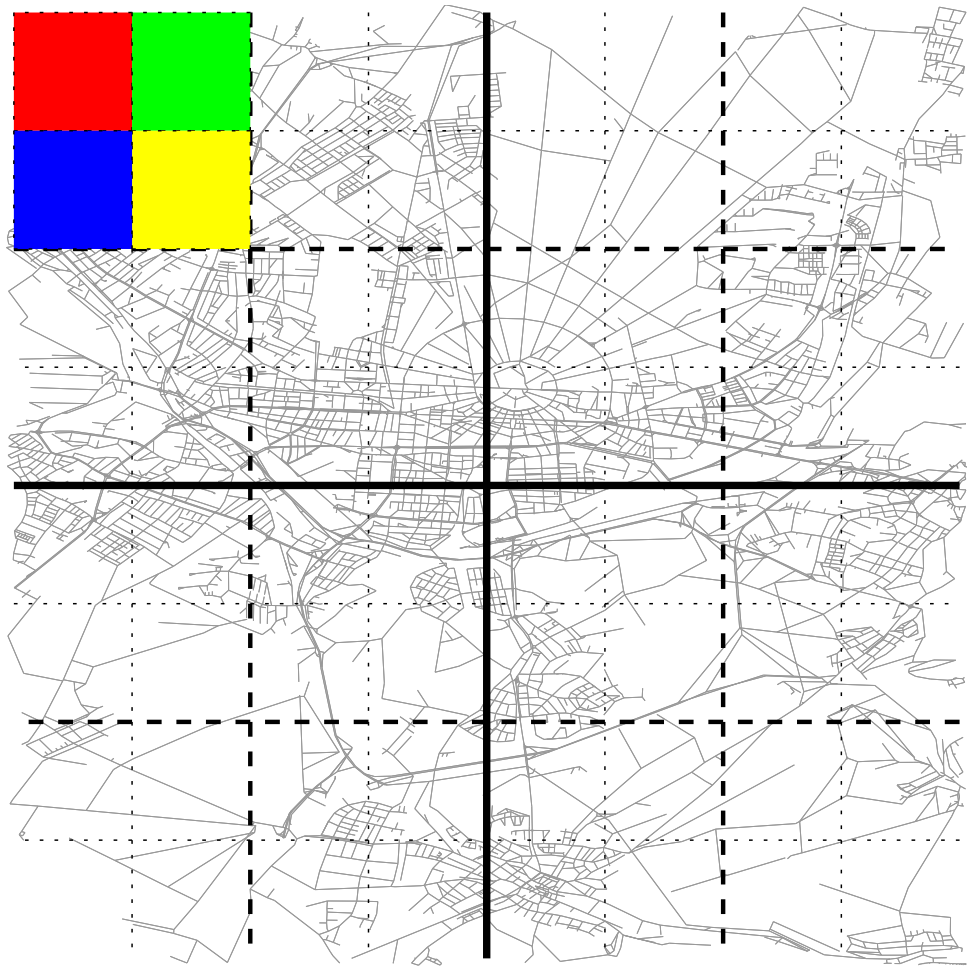
SHARC (Kontraktion, Arc-Flags)

Vorberechnung:

- Multi-Level-Partition
- iterativer Prozess:
  - kontrahiere Subgraphen
  - berechne Flaggen
- Flaggenverfeinerung

Anpassung (grob):

- Kontraktion und Flaggen berechnung anpassen
- Verfeinerung durch (lokale) Profilsuchen



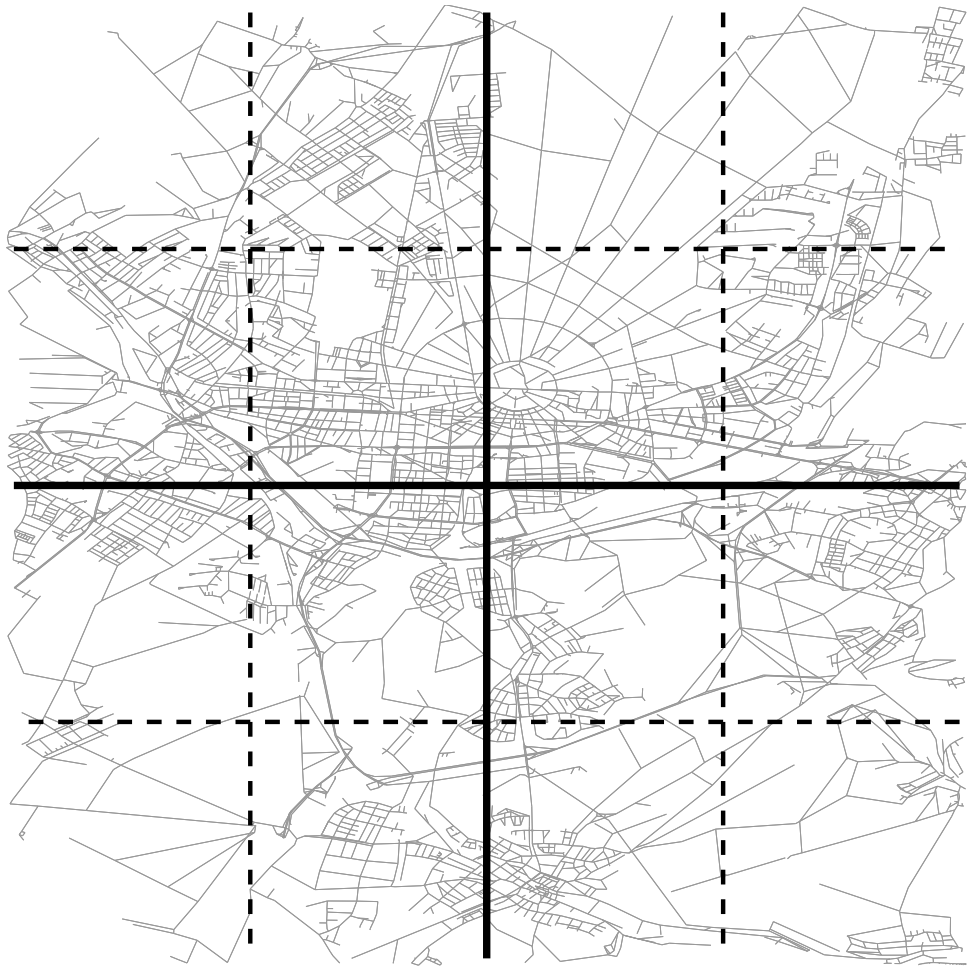
SHARC (Kontraktion, Arc-Flags)

Vorberechnung:

- Multi-Level-Partition
- iterativer Prozess:
  - kontrahiere Subgraphen
  - berechne Flaggen
- Flaggenverfeinerung

Anpassung (grob):

- Kontraktion und Flaggen berechnung anpassen
- Verfeinerung durch (lokale) Profilsuchen



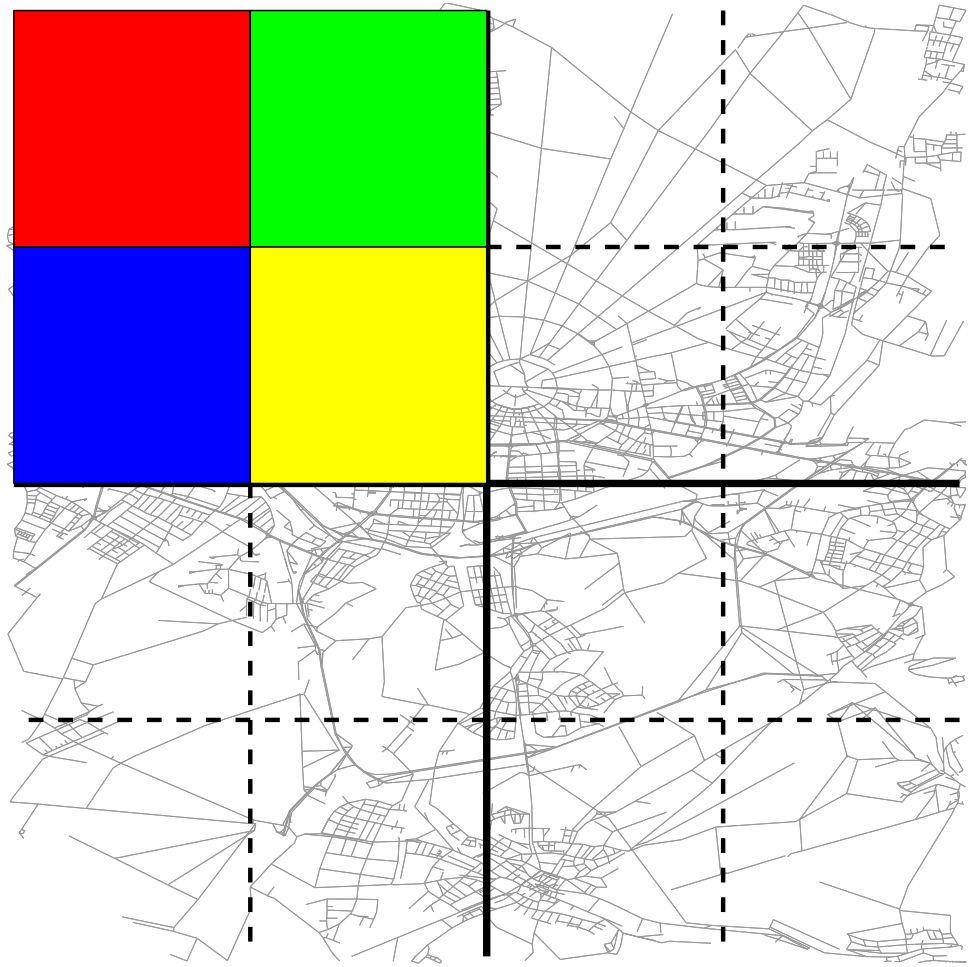
SHARC (Kontraktion, Arc-Flags)

Vorberechnung:

- Multi-Level-Partition
- iterativer Prozess:
  - kontrahiere Subgraphen
  - berechne Flaggen
- Flaggenverfeinerung

Anpassung (grob):

- Kontraktion und Flaggen berechnung anpassen
- Verfeinerung durch (lokale) Profilsuchen



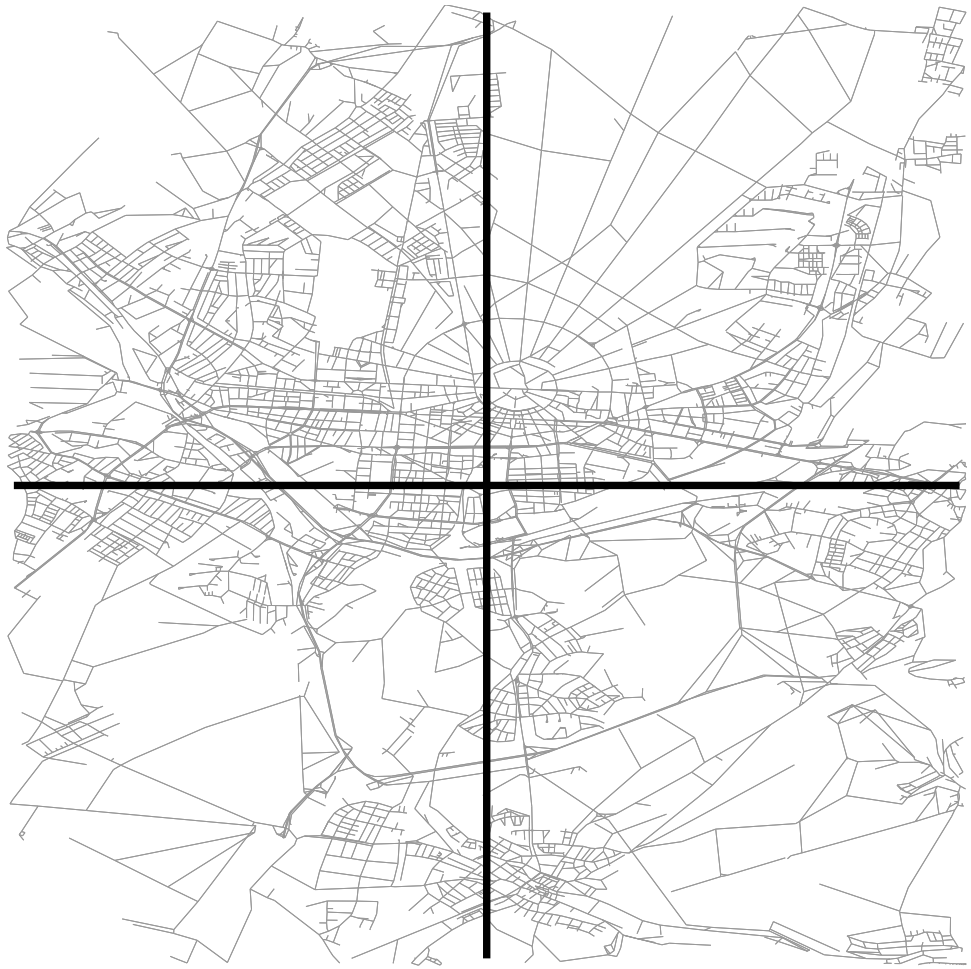
SHARC (Kontraktion, Arc-Flags)

Vorberechnung:

- Multi-Level-Partition
- iterativer Prozess:
  - kontrahiere Subgraphen
  - berechne Flaggen
- Flaggenverfeinerung

Anpassung (grob):

- Kontraktion und Flaggen berechnung anpassen
- Verfeinerung durch (lokale) Profilsuchen



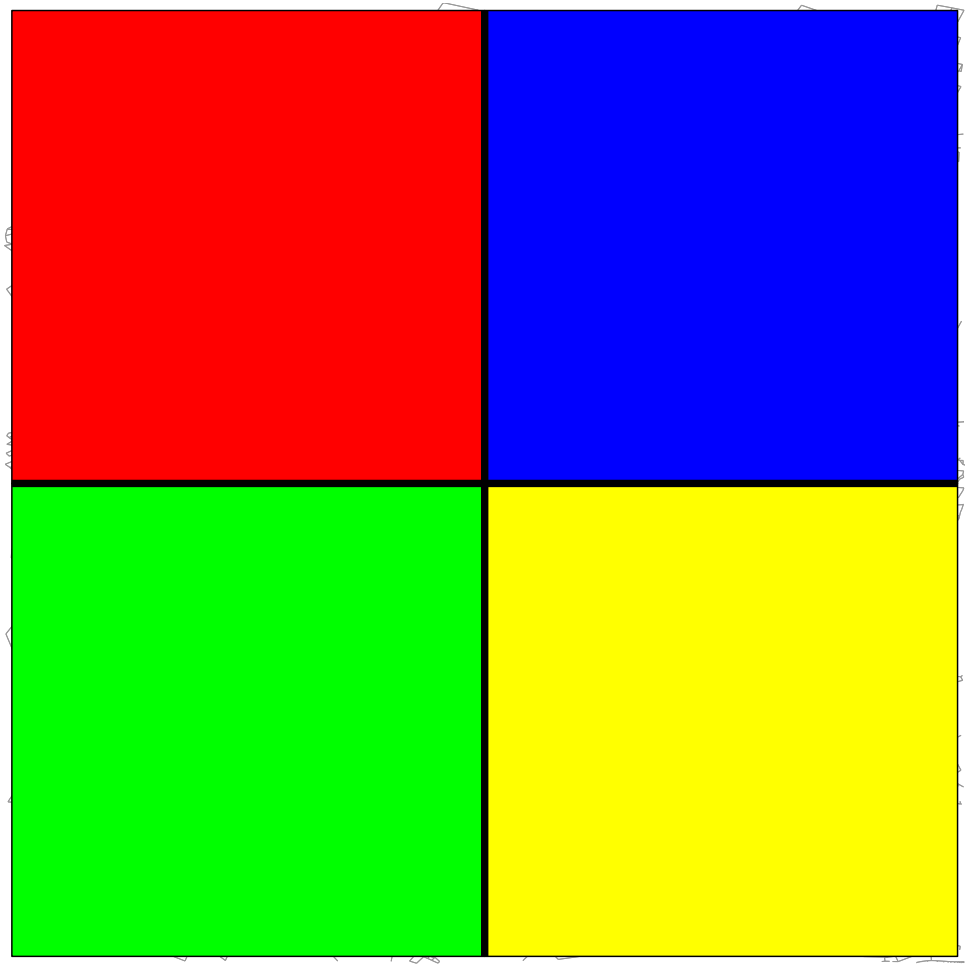
SHARC (Kontraktion, Arc-Flags)

Vorberechnung:

- Multi-Level-Partition
- iterativer Prozess:
  - kontrahiere Subgraphen
  - berechne Flaggen
- Flaggenverfeinerung

Anpassung (grob):

- Kontraktion und Flaggen berechnung anpassen
- Verfeinerung durch (lokale) Profilsuchen



Contraction Hierarchies

Vorberechnung:

- benutze gleiche Knotenordnung
- kontrahiere zeitabhängig
- erzeugt Suchgraphen $G' = (V, \uparrow E \cup \downarrow E)$

Contraction Hierarchies

Vorberechnung:

- benutze gleiche Knotenordnung
- kontrahiere zeitabhängig
- erzeugt Suchgraphen $G' = (V, \uparrow E \cup \downarrow E)$

Anfrage

- Rückwärtssuche schwierig (Ankunftszeit unbekannt)
- Kompletten Rückwärtsaufwärtssuchraum markieren?

Contraction Hierarchies

Vorberechnung:

- benutze gleiche Knotenordnung
- kontrahiere zeitabhängig
- erzeugt Suchgraphen $G' = (V, \uparrow E \cup \downarrow E)$

Anfrage

- Rückwärtssuche schwierig (Ankunftszeit unbekannt)
- Kompletten Rückwärtsaufwärtssuchraum markieren?
- Rückwärts aufwärts mittels min-max Suche (Phase 1)
  - Intervallsuche: jeder Knoten bekommt eine untere und obere Reisezeitschranke
  - markiere alle Kanten (u, v) aus $\downarrow E$ mit $\underline{d(u, v)} + \underline{d(v, t)} \leq \overline{d(u, t)}$
  - diese Menge sei $\downarrow E'$
- zeitabhängige Vorwärtsuche in $(V, \uparrow E \cup \downarrow E')$ (Phase 2)

Experimente (TD Germany)

input	type of ordering	Contr. ordering [h:m]	Contr. const. [h:m]	Contr. space [B/n]	Queries time [ms]	Queries speed up
Monday	static min	0:05	0:20	1 035	1.19	1 240
	timed	1:47	0:14	750	1.19	1 244
midweek	static min	0:05	0:20	1 029	1.22	1 212
	timed	1:48	0:14	743	1.19	1 242
Friday	static min	0:05	0:16	856	1.11	1 381
	timed	1:30	0:12	620	1.13	1 362
Saturday	static min	0:05	0:08	391	0.81	1 763
	timed	0:52	0:08	282	1.09	1 313
Sunday	static min	0:05	0:06	248	0.71	1 980
	timed	0:38	0:07	177	1.07	1 321

Approximation

Idee:

- speicher jeden Shortcut als Approximation
- reduziert Speicher um bis zu Faktor 10
- Aber: selbst die Vorwärtssuche liefert jetzt nur noch (Earliest Arrival) Approximationsintervalle!

Approximation

Idee:

- speicher jeden Shortcut als Approximation
- reduziert Speicher um bis zu Faktor 10
- Aber: selbst die Vorwärtssuche liefert jetzt nur noch (Earliest Arrival) Approximationsintervalle!

Query (viele Phasen!):

- Phase 1: Rück-auf: Min/max-Intervall, Vor-auf: Ankunfts-Intervall
- Phase 2: Vor-ab: Ankunftsintervall
- Phase 3: Rück-auf: Reisezeit-Intervall (verschärft Schranken)
- alle Knoten an denen Suchen sich treffen: Kandidaten C
- entpacke alle (approximierten) Shortcuts auf s–C–t Pfaden
- erzeugt (exakten weil Originalkanten) Subgraphen (Korridor)
- Time-dependent Dijkstra im Korridor (Phase 4)
- nicht viel langsamer (Faktor 2)
- ist exakt

Profilsuchen

Variante 1:

- normale Profilsuche in der CH
- langsam

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Profilsuchen

Variante 1:

- normale Profilsuche in der CH
- langsam

Variante 2:

- normale Profilsuche im Korridor (min/max, Approximation)
- besser, aber es geht noch besser

Profilsuchen

Variante 1:

- normale Profilsuche in der CH
- langsam

Variante 2:

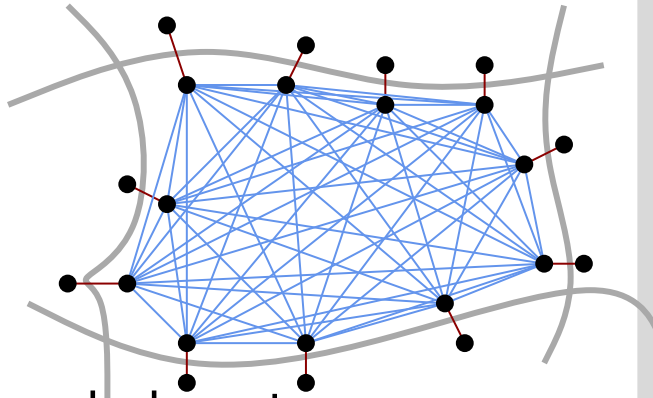
- normale Profilsuche im Korridor (min/max, Approximation)
- besser, aber es geht noch besser

Variante 3:

- Kontraktion des Korridors:
  - halte Start- und Zielknoten fest
  - Führe exakte Kontraktion durch (Linken von Kanten, keine Approximation)
  - Priorisiere Knoten mit unkomplexen inzidenten Kanten
- Balancierte Berechnung

$\Rightarrow$ ca. 30 ms

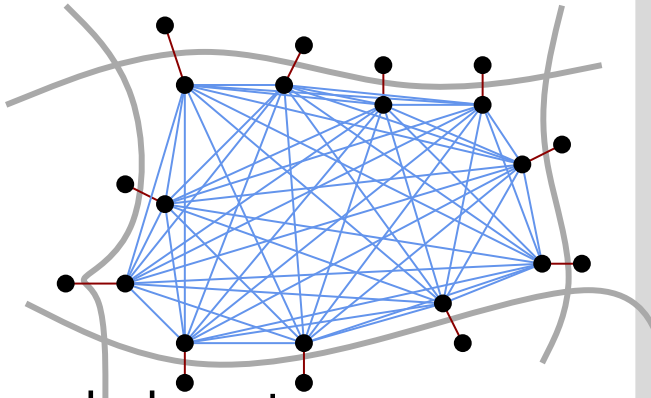
MLD (CRP)



Beobachtungen

- Partitionierung metrik-unabhängig
- Viele Shortcuts / Overlay-Kanten
- Großer Vorteil von MLD (eigentlich):
  Komplettes Speicherlayout nach Partitionierung bekannt
- Uni-direktionale Anfragen möglich (Partition steuert Suchlevel)

MLD (CRP)



Beobachtungen

- Partitionierung metrik-unabhängig
- Viele Shortcuts / Overlay-Kanten
- Großer Vorteil von MLD (eigentlich):
  Komplettes Speicherlayout nach Partitionierung bekannt
- Uni-direktionale Anfragen möglich (Partition steuert Suchlevel)

TDCRP

- Speicherlayout hängt an Komplexität der
  Overlay-Kanten-Funktionen; die ist aber vorab unbekannt
- Lohnt sich noch das Speichern der kompletten Cliquen?
- Hilft geschickte Approximation?

Approximation (TDCRP)

Approximation hilft auch hier (sehr viel)

ε [%]		Lvl 1	Lvl 2	Lvl 3	Lvl 4	Lvl 5	Lvl 6
0	breakpoints	99 M	397 M	813 M	1 356 M	—	—
	td.arc.cplx.	21	69	188	507	—	—
0.1	breakpoints	65 M	126 M	142 M	121 M	68 M	26 M
	td.arc.cplx.	14	22	33	45	50	47
1.0	breakpoints	51 M	73 M	62 M	41 M	21 M	8 M
	td.arc.cplx.	11	13	14	15	15	14
10.0	breakpoints	28 M	28 M	19 M	12 M	6 M	1 M
	td.arc.cplx.	6	5	5	5	4	2

Approximation nach jedem Level.

Robustheit (Synth. TD Europe)

TDCRP robuster gegen Veränderungen in der Eingabe

	TCH		TDCRP	
Network	Pre. [s]	Q. [ms]	Cust. [s]	Q. [ms]
Europe	1 479	1.37	109	5.75
Europe, bad traffic	7 772	5.87	208	8.01
Europe, avoid highways	8 956	19.54	127	8.29

Was fehlt?

- Bis hierhin: Viele komplizierte Algorithmen
- Geht das nicht einfacher?

Was fehlt?

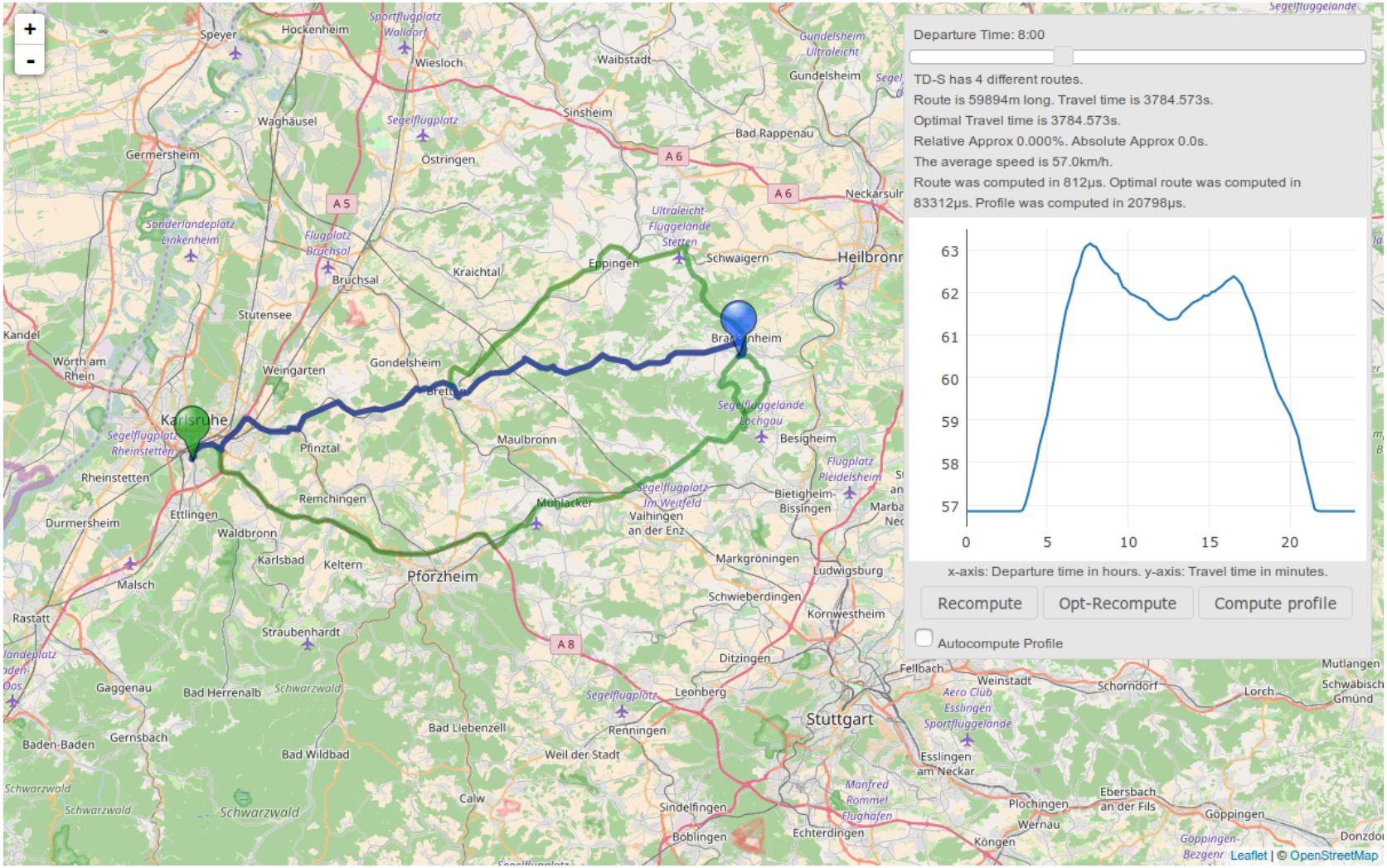
- Bis hierhin: Viele komplizierte Algorithmen
- Geht das nicht einfacher?
- Ein sinnvoller Grundalgorithmus:
  - Berechne eine CH auf dem untere-Schranken-Graph
  - Berechne zeitunabhängigen Pfad
  - Rechne zeitabhängige Fahrzeit entlang des Pfads nach
- heißt Freeflow

Time-Dependent Sampling

Etwas ausgefeilter:

- Tag in charakteristische Abschnitte aufteilen
  Rush-hour Morgens, Mittags, Rush-hour Nachmittags, Nachts, (Live)
- Zeit-unabhängiger Graph für jeden Abschnitt
  - Gewichte: Durchschnittlich travel time pro Abschnitt
- Eine (C)CH pro Abschnitt
- Zeit-unabhängiger kürzester Pfad pro Abschnitt
- Pfade zu Subgraph vereinigen
- **Earliest arrival**: TD-Dijkstra in Subgraph
- **Profil**: Alle 10 min. TD-Dijkstra's algo. in Subgraph

Time-Dependent Sampling



Performance EA Queries: Ger06

	Prepro.	Custom.	Space	Query	Rel. error [%]	
	Cores $\times$ [s]	Cores $\times$ [s]	[GB]	[ms]	Avg.	Max.
TD-Dijkstra	-	-	-	525.48	-	-
TDCALT	540	-	0.23	5.36	-	-
TDCALT-K1.15	540	-	0.23	1.87	0.050	13.840
eco L-SHARC	4 680	-	1.03	6.31	-	-
heu SHARC	12 360	-	0.64	0.69	n/r	0.610
KaTCH	16 $\times$ 170	-	4.66	0.63	-	-
TCH	8 $\times$ 378	8 $\times$ 74	4.66	0.75	-	-
ATCH (1.0)	8 $\times$ 378	8 $\times$ 74	1.12	1.24	-	-
ATCH (∞)	8 $\times$ 378	8 $\times$ 74	0.55	1.66	-	-
inex. TCH (0.1)	8 $\times$ 378	8 $\times$ 74	1.34	0.70	0.020	0.100
inex. TCH (1.0)	8 $\times$ 378	8 $\times$ 74	1.00	0.69	0.270	1.010
TD-CRP (0.1)	16 $\times$ 273	16 $\times$ 16	0.78	1.92	0.050	0.250
TD-CRP (1.0)	16 $\times$ 273	16 $\times$ 8	0.36	1.66	0.680	2.850
TD-S+9	547	-	3.61	1.67	0.001	1.523
CATCHUp	16 $\times$ 31	16 $\times$ 18	1.06	0.70	-	-

Performance Profil: Ger06

	Run T. [ms]
eco SHARC	47 388
heu SHARC	847
ATCH (2.5)	30
TD-S+P4	19.5
TD-S+P9	22.2

Performance EA Queries: Ger17

	Prepro.	Custom.	Space	Query	Rel. error [%]	
	Cores $\times$ [s]	Cores $\times$ [s]	[GB]	[ms]	Avg.	Max.
TD-Dijkstra	-	-	-	869.79	-	-
KaTCH	16 $\times$ 874	-	42.81	1.38	-	-
TD-S+9	617	-	5.28	2.28	0.001	0.963
CATCHUp	16 $\times$ 35	16 $\times$ 92	1.50	1.87	-	-

Performance EA Queries: Eur17

	Prepro.	Custom.	Space	Query	Rel. error [%]	
	Cores $\times$ [s]	Cores $\times$ [s]	[GB]	[ms]	Avg.	Max.
TD-Dijkstra	-	-	-	2 581.16	-	-
KaTCH	16 $\times$ 3 089	-	146.97	OOM	-	-
TD-S+9	3 368	-	18.84	4.03	0.002	1.159
CATCHUp	16 $\times$ 196	16 $\times$ 479	5.48	4.50	-	-

Nächster Termin

Karlsruher Institut für Technologie

Montag, 8. Juli 2019

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Literatur

- Daniel Delling:
  Enginering and Augmenting Route Planning Algorithms
  Ph.D. Thesis, Universität Karlsruhe (TH), 2009.
- Gernot Veit Batz, Robert Geisberger, Peter Sanders, Christian Vetter:
  Minimum Time-Dependent Travel Times with Contraction Hierarchies
  Journal of Experimental Algorithmics, 2013.
- Moritz Baum, Julian Dibbelt, Thomas Pajor, Dorothea Wagner:
  Dynamic Time-Dependent Route Planning in Road Networks with User Preferences
  In: *Proceedings of the 9th International Symposium on Experimental Algorithms (SEA'16)*, 2016. **Dynamic Time-Dependent Routing in Road Networks Through Sampling**
  In: *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'17)*, 2017.