

Algorithmen für Routenplanung

12. Vorlesung, Sommersemester 2018

Tim Zeitz | 13. Juni 2018

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



- 1 Grundlagen
 - Datenstrukturen
 - Algorithmen
 - “Basisoperation” Kontraktion
- 2 Beschleunigungstechniken in Straßennetzen
- 3 Erweiterte Problemstellungen in Straßennetzen
- 4 Fahrplanauskunft
- 5 Multimodale Routenplanung

- ① Grundlagen
- ② Beschleunigungstechniken in Straßennetzen
 - ALT
 - Arc-Flags
 - Reach
 - MLD
 - (C)CH
 - HL
 - TNR
- ③ Erweiterte Problemstellungen in Straßennetzen
- ④ Fahrplanauskunft
- ⑤ Multimodale Routenplanung

- ① Grundlagen
- ② Beschleunigungstechniken in Straßennetzen
- ③ Erweiterte Problemstellungen in Straßennetzen
 - Alternativrouten
 - zeitabhängige Routen
 - one2all, one2many, many2many
 - POIs
 - Isochronen
 - Abbiegekosten
 - multikriterielle Routen
 - Elektromobilität
- ④ Fahrplanauskunft
- ⑤ Multimodale Routenplanung

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 → Stau
- Mittags gibt es weniger Stau
- Abends fährt jeder nach Hause → Stau in die andere Richtung
- Aber nicht Sonntags



2. Zeitabhängiges Szenario

Motivation:

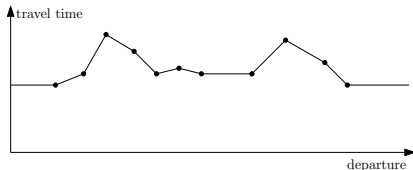
- Stau um Städte herum folgt vorhersehbaren Mustern
- Morgens geht jeder zur Arbeit → Stau
- Mittags gibt es weniger Stau
- Abends fährt jeder nach Hause → Stau in die andere Richtung
- Aber nicht Sonntags



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

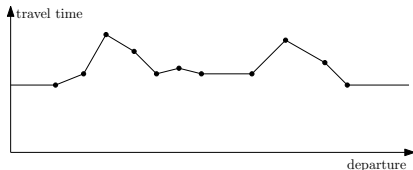
Zeitabhängige Kantengewichte

- **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?

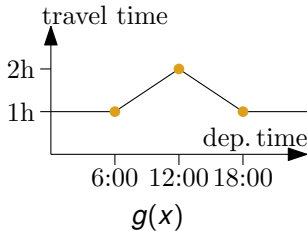
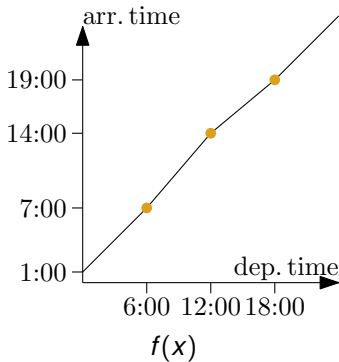
- In der akademischen Forschung nimmt man oft an, dass Interpolationspunkte pro Kante gegeben sind
- Viele Probleme mit dieser Annahme:
 - Viele Interpolationspunkte pro Graph
 - Großer Speicherverbrauch
 - Sehr problematisch
 - Erhobene Daten sind stark verrauscht und Vorhersagen ungenau
 - Interpolationspunkte 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

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

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



- 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

Definition

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

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

Diskussion

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

⇒ Sicherstellen, dass Funktionen FIFO-Eigenschaft erfüllen.

Hauptproblem:

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

Hauptproblem:

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

Vorgehen:

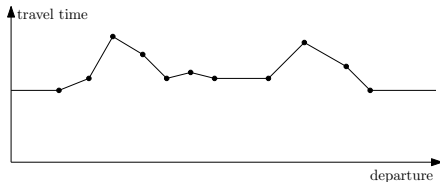
- Modellierung
- Anpassung Dijkstra
- Anpassung Beschleunigungstechniken

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



Eigenschaften “Zeitabhängigkeit”:

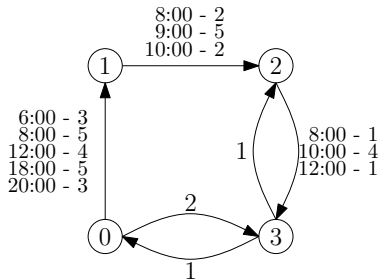
- Topologie ändert sich nicht
- Kanten gemischt zeitabhängig und konstant
- variable (!) Anzahl Interpolationspunkte pro Kante

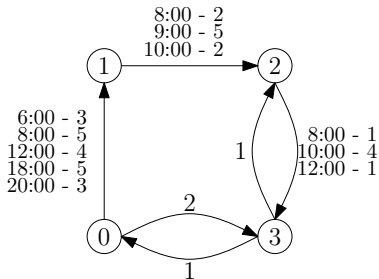
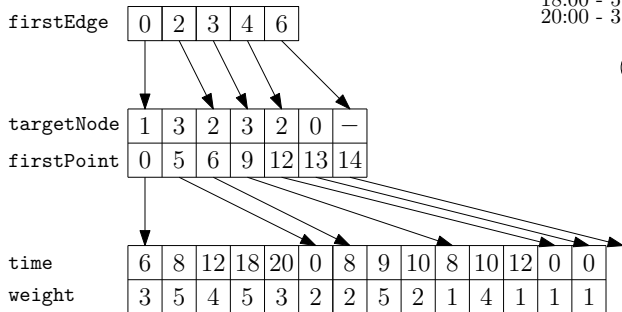
Eigenschaften “Zeitabhängigkeit”:

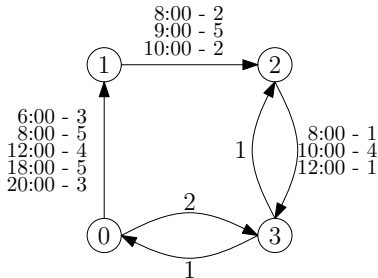
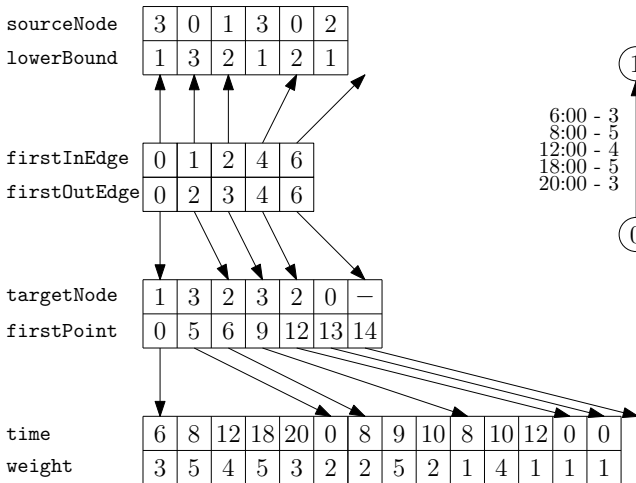
- Topologie ändert sich nicht
- Kanten gemischt zeitabhängig und konstant
- variable (!) Anzahl Interpolationspunkte pro Kante

Voraussetzung:

- FIFO gilt auf allen Kanten







Zeit-Anfrage:

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

Zeit-Anfrage:

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

Profil-Anfrage:

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

Ziel: finde kürzesten Weg für Abfahrtszeit τ

Time-Dijkstra($G = (V, E), s, \tau$)

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

Beobachtung:

- Nur ein Unterschied zu Dijkstra
- Auswertung der Kanten

Beobachtung:

- Nur ein Unterschied zu Dijkstra
- Auswertung der Kanten

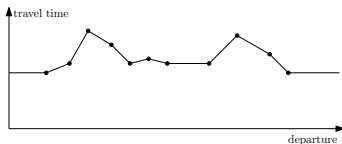
non-FIFO Netzwerke:

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

Evaluation von $f(\tau)$:

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

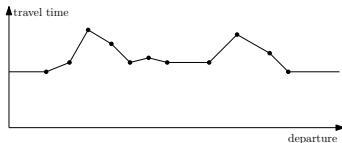
$$f(\tau) = w_i + (\tau - t_i) \cdot \frac{w_{i+1} - w_i}{t_{i+1} - t_i}$$



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}
 - 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

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    $u \leftarrow Q.deleteMin()$ 
5   for all edges  $e = (u, v) \in E$  do
6     if  $d_*[u] \oplus \text{len}(e) \not\leq d_*[v]$  then
7        $d_*[v] \leftarrow \min(d_*[u] \oplus \text{len}(e), d_*[v])$ 
8       if  $v \in Q$  then  $Q.decreaseKey(v, \underline{d}[v])$ 
9
10      else  $Q.insert(v, \underline{d}[v])$ 
```

Beobachtungen:

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

Herausforderungen:

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

Funktion gegeben durch:

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

3 Operationen notwendig:

- Auswertung
- Linken \oplus
- Minimumsbildung
- Vergleich $\not\leq$
(ist analog zu Minimumsbildung)

Definition

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

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

Oder

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

Definition

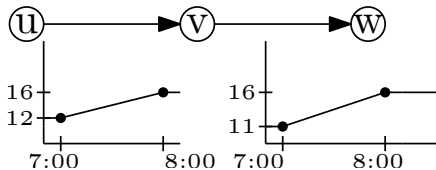
Seien $f : \mathbb{R}_0^+ \rightarrow \mathbb{R}_0^+$ und $g : \mathbb{R}_0^+ \rightarrow \mathbb{R}_0^+$ zwei Ankunftszeitfunktionen die die FIFO-Eigenschaft erfüllen. Die Linkoperation $f \oplus g$ ist dann definiert durch

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

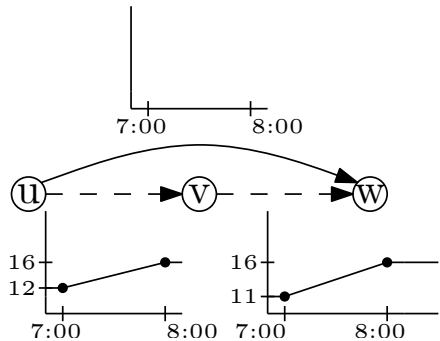
Oder

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

Linken zweier Funktionen f und g

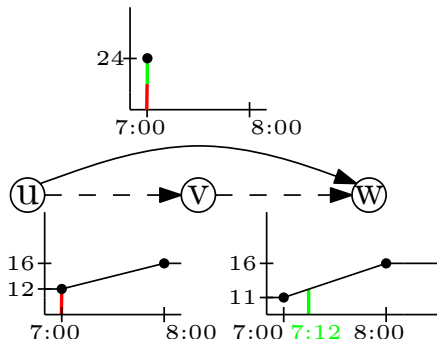


Linken zweier Funktionen f und g



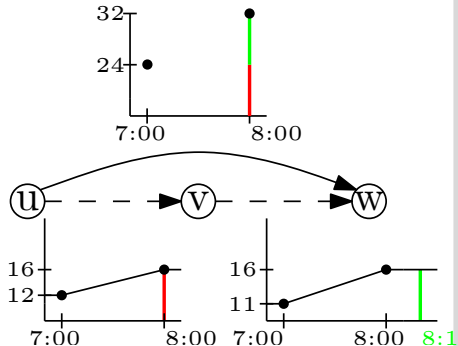
Linken zweier Funktionen f und g

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



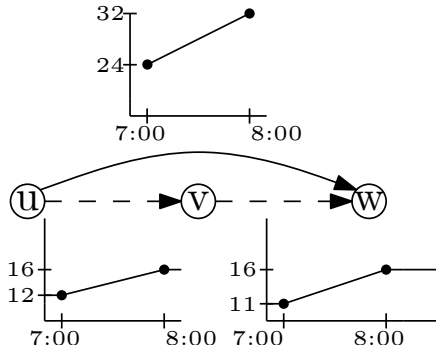
Linken zweier Funktionen f und g

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



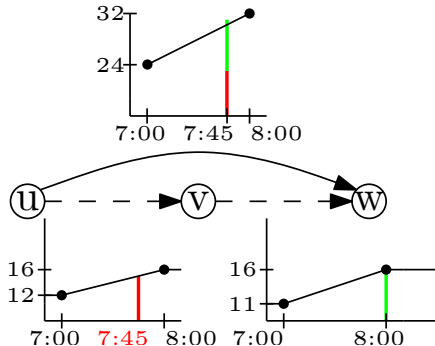
Linken zweier Funktionen f und g

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



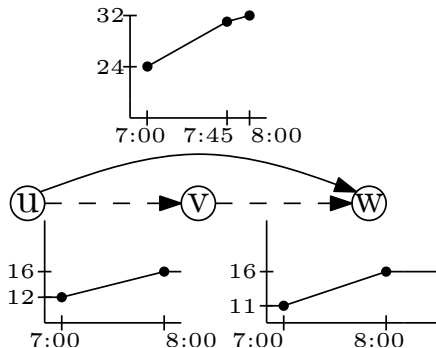
Linken zweier Funktionen f und g

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



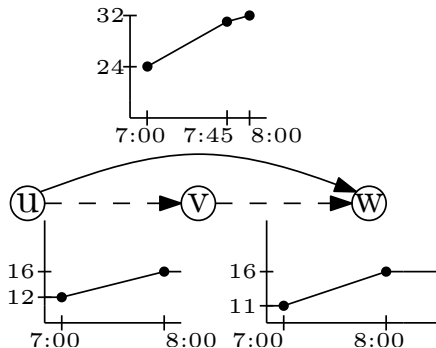
Linken zweier Funktionen f und g

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



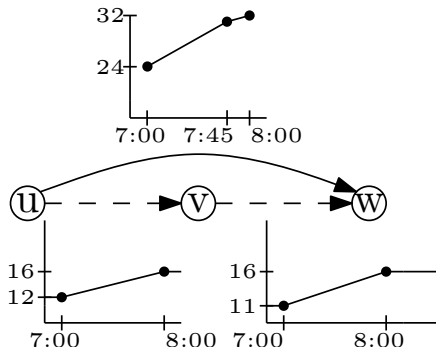
Linken zweier Funktionen f und g

- $f \oplus g$ enthält auf jeden Fall $\{(t_1^f, w_1^f + g(t_1^f + w_1^f)), \dots, (t_j^f, w_j^f + g(t_j^f + w_j^f))\}$
- 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 zweier Funktionen f und g

- $f \oplus g$ enthält auf jeden Fall $\{(t_1^f, w_1^f + g(t_1^f + w_1^f)), \dots, (t_j^f, w_j^f + g(t_j^f + w_j^f))\}$
- 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



Demo Time



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)

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

Laufzeit

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

Laufzeit

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

Speicherverbrauch

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

Laufzeit

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

Speicherverbrauch

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

Problem:

- Während Profilsuche kann ein Pfad mehreren Tausend Kanten entsprechen

Laufzeit

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

Speicherverbrauch

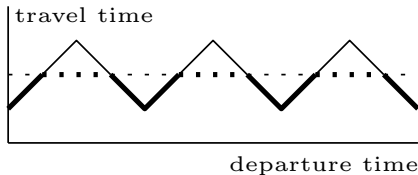
- Geknickte Funktion hat $\approx |I^f| + |I^g|$ Interpolationspunkte

Problem:

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

Minimum zweier Funktionen f und g

- Für alle (t_j^f, w_j^f) : behalte Punkt, wenn $w_j^f < g(t_j^f)$
- Für alle (t_j^g, w_j^g) : behalte Punkt, wenn $w_j^g < f(t_j^g)$
- Schnittpunkte müssen ebenfalls eingefügt werden

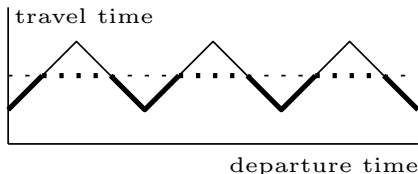


Minimum zweier Funktionen f und g

- Für alle (t_j^f, w_j^f) : behalte Punkt, wenn $w_j^f < g(t_j^f)$
- 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



Laufzeit

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

Laufzeit

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

Speicherverbrauch

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

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

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

Problem 2: Numerische Instabilität

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

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

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?

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

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
- Offen Frage: Kann man diese Divergenz zeigen?

- Netzwerk Deutschland $|V| \approx 4.7$ Mio., $|E| \approx 10.8$ Mio.
- 5 Verkehrsszenarien:
 - 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

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

⇒ inpraktikabel

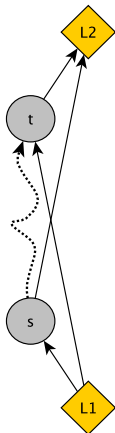
Zeitabhängige Netzwerke (Basics)

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

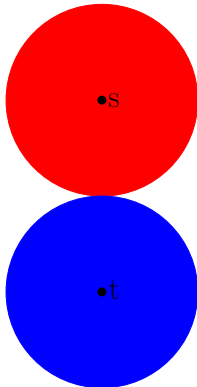
Beschleunigungstechniken



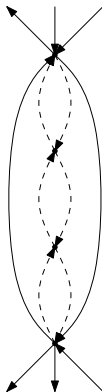
Landmarken



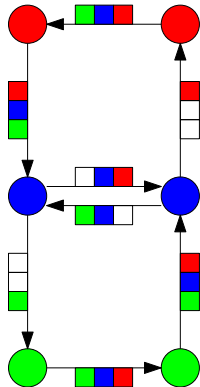
Bidirektionale Suche



Kontraktion



Arc-Flags



Vorbereitung:

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

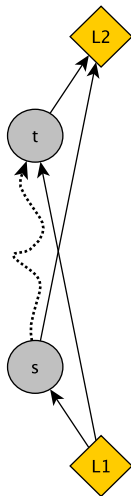
Anfrage:

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

$$d(s, t) \geq d(L_1, t) - d(L_1, s)$$

$$d(s, t) \geq d(s, L_2) - d(t, L_2)$$

- verändert **Reihenfolge** der besuchten Knoten



Beobachtung:

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

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

- durch Erhöhen der Kantengewichte wird dies nicht verletzt

Beobachtung:

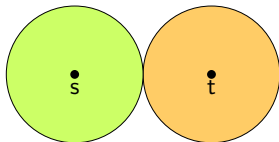
- Korrektheit von ALT basiert darauf, dass reduzierte Kosten größer oder gleich 0 sind

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

- 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



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

Zeitanfragen:

- Ankunft unbekannt \Rightarrow Rückwärtsuche?

Zeitanfragen:

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

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 $\min\text{Key}(\vec{Q}) + \min\text{Key}(\overleftarrow{Q}) \geq \bar{\mu}$
Erinnere: key von v ist der lowerbound seiner Profildfunktion

Idee:

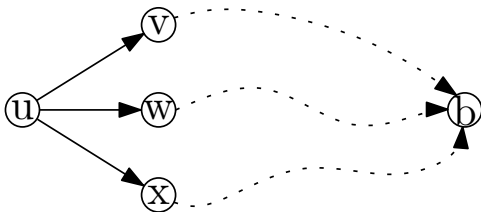
- ändere **Intuition** einer gesetzten Flagge
- Konzept **bleibt gleich**: Eine Flagge pro Kante und Region
- setze Flagge wenn Kante **mindestens ein mal** am Tag “wichtig” ist

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 $\text{len}(u, v) \oplus d_*(v, b) \neq d_*(u, b)$

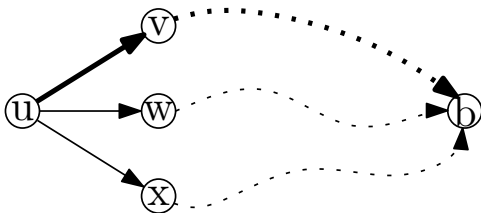


Idee:

- 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 $\text{len}(u, v) \oplus d_*(v, b) \neq d_*(u, b)$

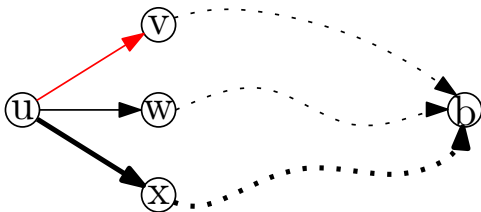


Idee:

- 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 $\text{len}(u, v) \oplus d_*(v, b) \neq d_*(u, b)$

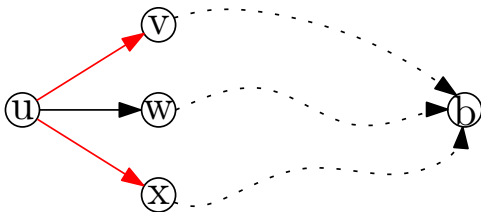


Idee:

- 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:

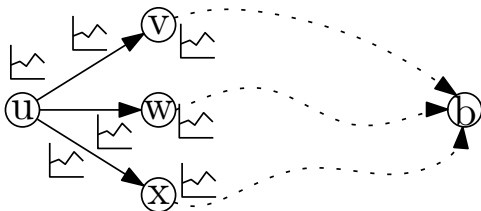
- für alle Randknoten b und alle Knoten u :
- Berechne Abstandsfunktion $d_*(u, b)$
- setze Flagge wenn gilt $\text{len}(u, v) \oplus d_*(v, b) \neq d_*(u, b)$



Beobachtung:

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

Idee:

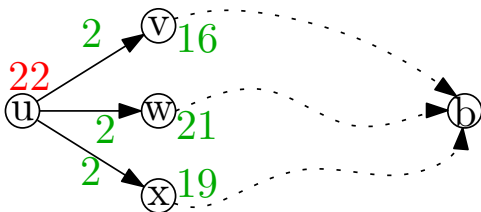


Beobachtung:

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

Idee:

- benutze über- und Unterapproximation

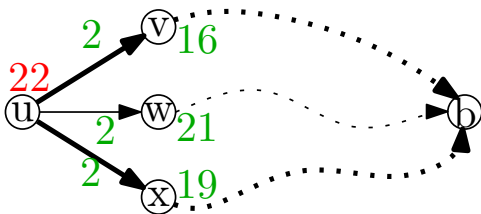


Beobachtung:

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

Idee:

- benutze über- und Unterapproximation

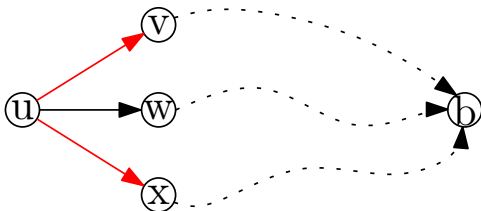


Beobachtung:

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

Idee:

- benutze **über-** und **Unterapproximation**
- ⇒ schnellere Vorberechnung, langsamere Anfragen
- ⇒ aber immer noch **korrekt**



Idee:

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

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?

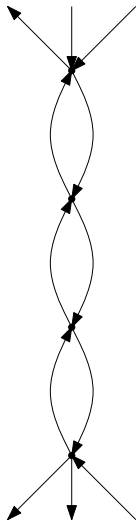
Kontraktion

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



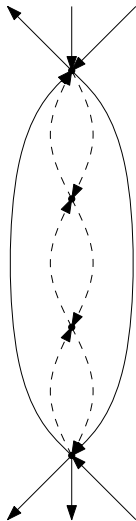
Kontraktion

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



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:

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

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

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**

Basismodule:

- 0 Bidirektionale Suche
- + Landmarken
- + Kontraktion
- + Arc-Flags

Somit sind folgende Algorithmen gute Kandidaten

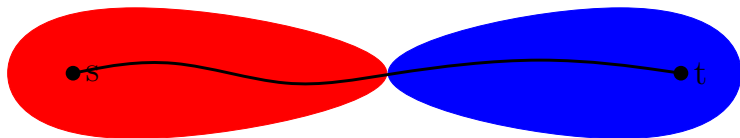
- ALT
- Core-ALT
- SHARC
- Contraction Hierarchies
- MLD (CRP)

Bidirektionaler zeitabhängiger ALT

● s

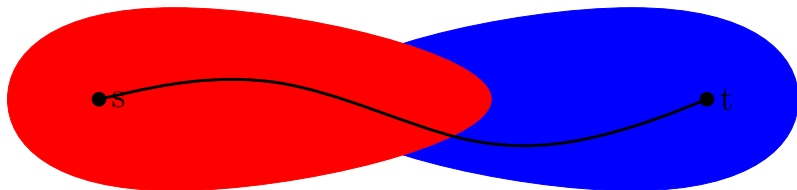
● t

Idee - Drei Phasen:



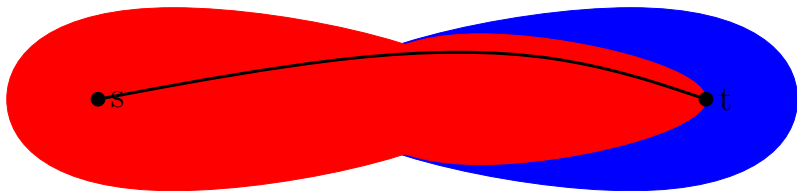
Idee - Drei Phasen:

- 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



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 $\minKey(\overleftarrow{Q}) > \mu$



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 $\minKey(\overleftarrow{Q}) > \mu$
- 3 Vorwärtssuche arbeitet weiter bis t abgearbeitet worden ist und besucht nur Knoten, die die Rückwärtssuche zuvor besucht hat

Beobachtung:

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

Approximation:

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

scen.	algorithm	K	Error			Query		
			rate	relative av.	max	#sett. nodes	#rel. edges	time [ms]
mid	uni-ALT	–	0.0%	0.000%	0.00%	200 236	239 112	147.20
	TDALT	1.00	0.0%	0.000%	0.00%	116 476	138 696	98.27
		1.15	12.4%	0.094%	14.32%	50 764	60 398	36.91
		1.50	12.5%	0.097%	27.59%	50 742	60 371	36.86
Sat	uni-ALT	–	0.0%	0.000%	0.00%	148 331	177 568	100.07
	TDALT	1.00	0.0%	0.000%	0.00%	63 717	76 001	47.41
		1.15	10.5%	0.088%	13.97%	50 042	59 607	36.00
		1.50	10.6%	0.089%	26.17%	50 036	59 600	35.63
Sun	uni-ALT	–	0.0%	0.000%	0.00%	142 631	170 670	92.79
	TDALT	1.00	0.0%	0.000%	0.00%	58 956	70 333	42.96
		1.15	10.4%	0.088%	14.28%	50 349	59 994	36.04
		1.50	10.5%	0.089%	32.08%	50 345	59 988	35.74

Idee

- begrenze Beschleunigungstechnik auf kleinen Subgraphen (Kern)

s ●

● t

Vorbereitung

- kontrahiere Graphen zu einem Kern
- Landmarken nur im Kern

Anfrage

Idee

- begrenze Beschleunigungstechnik auf kleinen Subgraphen (**Kern**)



Vorbereitung

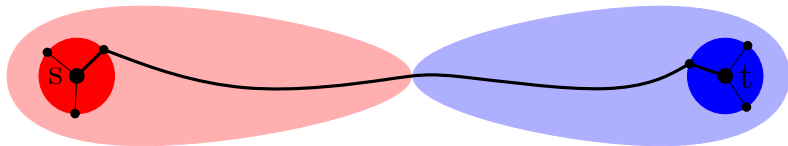
- kontrahiere Graphen zu einem Kern
- Landmarken nur im Kern

Anfrage

- Initialphase: normaler Dijkstra

Idee

- begrenze Beschleunigungstechnik auf kleinen Subgraphen (Kern)



Vorbereitung

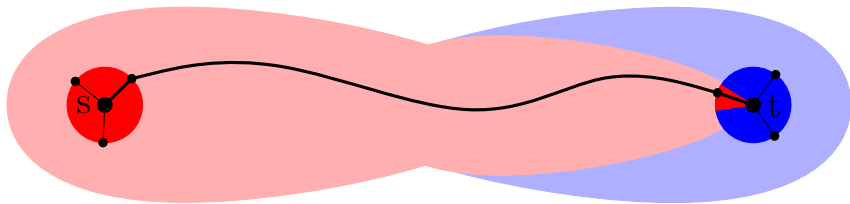
- kontrahiere Graphen zu einem Kern
- Landmarken nur im Kern

Anfrage

- Initialphase: normaler Dijkstra
- benutze Landmarken nur im Kern

Idee

- begrenze Beschleunigungstechnik auf kleinen Subgraphen (**Kern**)



Vorbereitung

- 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

Experimente (Synth. TD Europe)

CORE			PREPROCESSING				EXACT QUERY	
par.	core	nodes	time	space	increase in		#sett.	time
<i>c</i>	<i>h</i>		[m]	[B/n]	edges	points	nodes	[ms]
0.0	0	100.0%	28	256	0.0%	0.0%	2931080	2939.3
0.5	10	35.6%	15	99	9.8%	21.1%	1165840	1224.8
1.0	20	6.9%	18	41	12.6%	69.6%	233788	320.5
2.0	30	3.2%	30	45	9.9%	114.1%	108306	180.0
2.5	40	2.5%	39	50	9.1%	138.0%	84119	149.7
3.0	50	2.0%	50	56	8.7%	161.2%	70348	133.2
3.5	60	1.8%	60	61	8.5%	181.1%	60636	122.3
4.0	70	1.5%	88	74	8.5%	223.1%	52908	115.2
5.0	100	1.2%	134	89	8.6%	273.5%	45020	110.6

Experimente (TD Germany)

scenario	K	Preproc.		Error			Query		
		time [min]	space [B/n]	rate	relative av.	max	#settled nodes	#relaxed edges	time [ms]
Monday	1.00	9	50.3	0.0%	0.000%	0.00%	2984	11316	4.84
	1.15	9	50.3	8.3%	0.051%	11.00%	1588	5303	1.84
	1.50	9	50.3	8.3%	0.052%	17.25%	1587	5301	1.84
midweek	1.00	9	50.3	0.0%	0.000%	0.00%	3190	12255	5.36
	1.15	9	50.3	8.2%	0.051%	13.84%	1593	5339	1.87
	1.50	9	50.3	8.2%	0.052%	13.84%	1592	5337	1.86
Friday	1.00	8	44.9	0.0%	0.000%	0.00%	3097	12162	5.21
	1.15	8	44.9	7.8%	0.052%	11.29%	1579	5376	1.82
	1.50	8	44.9	7.8%	0.054%	21.19%	1579	5374	1.82
Saturday	1.00	6	27.8	0.0%	0.000%	0.00%	1856	7188	2.42
	1.15	6	27.8	4.4%	0.031%	11.50%	1539	5542	1.71
	1.50	6	27.8	4.4%	0.031%	24.17%	1539	5541	1.71
Sunday	1.00	5	19.1	0.0%	0.000%	0.00%	1773	6712	2.13
	1.15	5	19.1	4.0%	0.029%	12.72%	1551	5541	1.68
	1.50	5	19.1	4.1%	0.029%	17.84%	1550	5540	1.68

Vorbereitung:

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

Anpassung (grob):

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

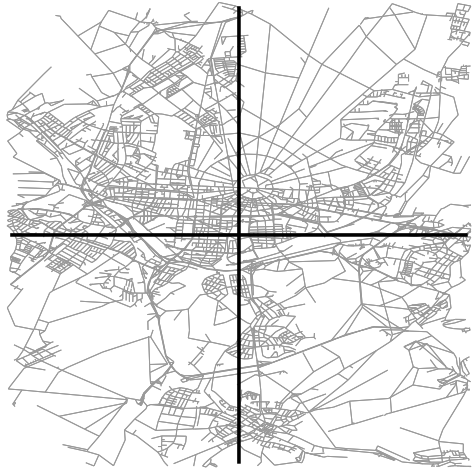


Vorbereitung:

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

Anpassung (grob):

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

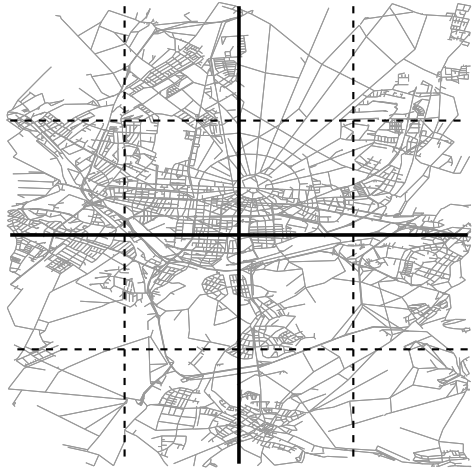


Vorbereitung:

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

Anpassung (grob):

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

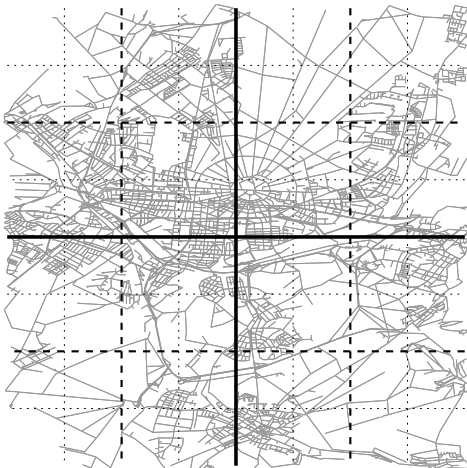


Vorbereitung:

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

Anpassung (grob):

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

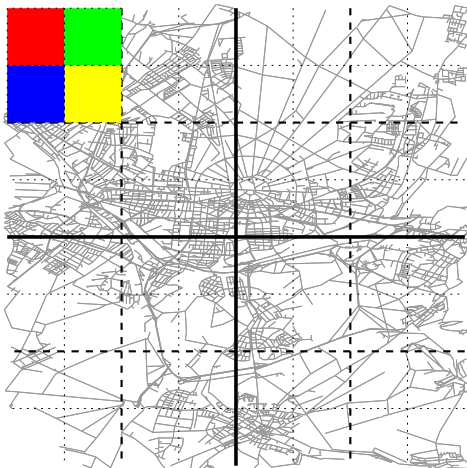


Vorbereitung:

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

Anpassung (grob):

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

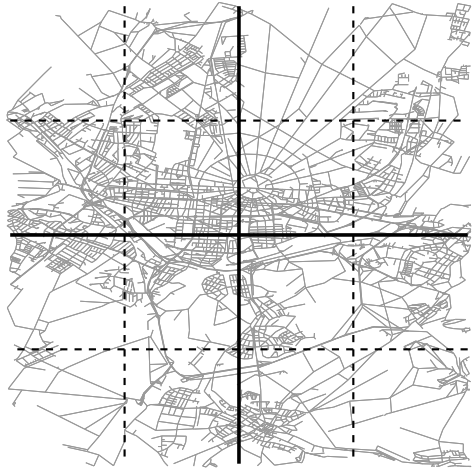


Vorbereitung:

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

Anpassung (grob):

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

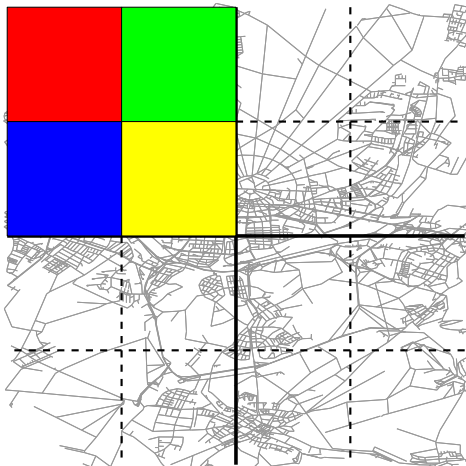


Vorbereitung:

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

Anpassung (grob):

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

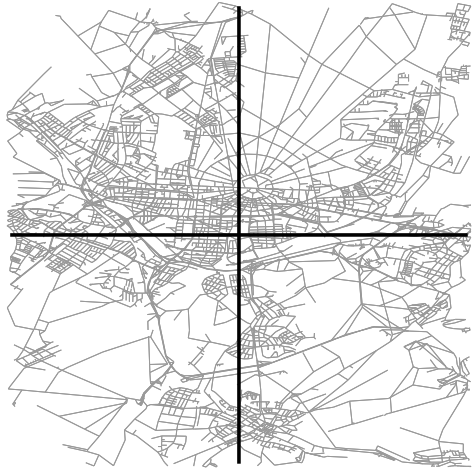


Vorbereitung:

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

Anpassung (grob):

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

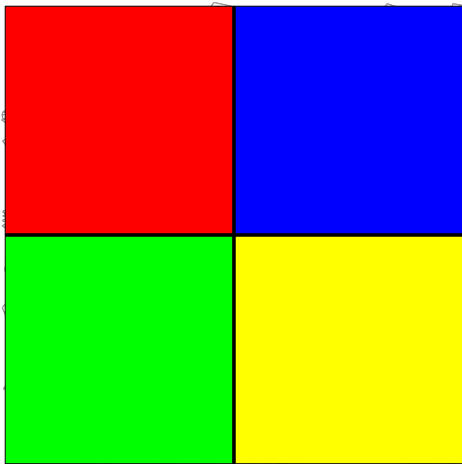


Vorbereitung:

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

Anpassung (grob):

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



scenario	algom	Preprocessing				Time-Queries	
		time [h:m]	space [B/n]	edge inc.	points inc.	time [ms]	speed up
Monday	eco	1:16	156.6	25.4%	366.8%	24.55	63
midweek	eco	1:16	154.9	25.4%	363.8%	25.06	60
Friday	eco	1:10	142.0	25.4%	358.0%	22.07	69
Saturday	eco	0:42	90.3	25.0%	283.6%	5.34	276
Sunday	eco	0:30	64.6	24.6%	215.8%	1.86	787
no traffic	static	0:06	13.5	23.9%	23.9%	0.30	4 075

scenario	algo	Prepro		Error			Time-Queries	
		time [h:m]	space [B/n]	error -rate	max rel.	max abs.[s]	time [ms]	spd up
Monday	heu	3:30	138.2	0.46%	0.54%	39.3	0.69	2 253
midweek	heu	3:26	137.2	0.82%	0.61%	48.3	0.69	2 164
Friday	heu	3:14	125.2	0.50%	0.50%	50.3	0.64	2 358
Saturday	heu	2:13	80.4	0.18%	0.23%	16.9	0.51	2 887
Sunday	heu	1:48	58.8	0.09%	0.36%	14.9	0.46	3 163
no	static	0:06	13.5	0.00%	0.00%	0.0	0.30	4 075

Beobachtung:

- Fehler sehr gering
- hoher Speicherverbrauch

traffic	var.	Time-Queries		Profile-Queries		
		#del. mins	time [ms]	#del. mins	#re- ins.	time [ms]
Monday	eco	19 136	24.55	19 768	402	51 122
	heu	810	0.69	1 071	24	1 008
midweek	eco	19 425	25.06	20 538	432	60 147
	heu	818	0.69	1 100	27	1 075
Friday	eco	17 412	22.07	19 530	346	52 780
	heu	769	0.64	1 049	21	832
Saturday	eco	5 284	5.34	5 495	44	3 330
	agg	721	0.58	865	9	134
	heu	666	0.51	798	8	98
Sunday	eco	2 142	1.86	2 294	12	536
	agg	670	0.50	781	5	57
	heu	635	0.46	738	5	45

Vorbereitung:

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

Vorbereitung:

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

Anfrage

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

Vorbereitung:

- 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 Vorwärtssuche in $(V, \uparrow E \cup \downarrow E')$ (Phase 2)

input	type of ordering	Contr.			Queries	
		ordering [h:m]	const. [h:m]	space [B/n]	time [ms]	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

Idee:

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

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**

Approximation (TD Germany), Zeitanfragen

method	ε	space		time		delMin		edges		evals		error [%]	
	[%]	[B/n]	GRO	[ms]	SPD	#	SPD	#	SPD	#	SPD	MAX	AVG
Germany midweek													
TCH	-	994	10.4	0.72	1 440	520	4 616	5 813	951	1 269	162	0.00	0.00
TCH (cor.)	0.0	994	10.4	0.74	1 401	639	3 756	7 092	780	76	2 704	0.00	0.00
	0.1	286	3.0	0.71	1 460	642	3 739	7 128	770	77	2 669	0.10	0.02
	1.0	214	2.3	0.72	1 440	654	3 670	7 262	762	84	2 446	1.01	0.27
	10.0	113	1.2	1.03	1 006	897	2 676	10 096	548	223	921	9.75	3.84
ATCH (UoD)	0.1	308	3.2	1.10	942	554	4 332	7 734	715	3 080	67	0.00	0.00
	1.0	239	2.5	1.27	816	582	4 124	8 338	664	3 347	61	0.00	0.00
	10.0	163	1.7	2.40	432	824	2 913	21 036	263	7 486	27	0.00	0.00
	∞	118	1.2	1.45	714	698	3 439	20 116	275	3 153	65	0.00	0.00

Profilsuchen

Variante 1:

- normale Profilsuche in der CH
- langsam

Variante 1:

- normale Profilsuche in der CH
- langsam

Variante 2:

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

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

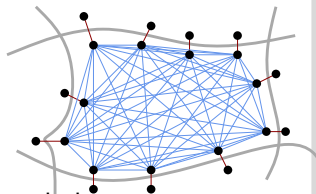
⇒ ca. 30 ms

Profilsuchen (TD Germany)

method	ε	space		time	delMin	edges	points	error [%]	
	[%]	[B/n]	GRO	[ms]	#	#	#	MAX	AVG
Germany midweek									
TCH	–	994	10.4	1 112.02	570	6 796	20 623 155	0.00	0.00
TCH (cor.)	0.0	994	10.4	88.87	646	7 170	1 437 892	0.00	0.00
	0.1	286	3.0	6.13	650	7 208	86 391	0.10	0.02
	1.0	214	2.3	2.94	662	7 348	35 769	1.03	0.27
	10.0	113	1.2	2.48	923	10 361	23 010	9.69	3.84
ATCH (CC)	0.1	308	3.2	36.22	650	29 551	576 099	0.00	0.00
	1.0	239	2.5	32.75	675	32 131	531 795	0.00	0.00
	10.0	163	1.7	105.45	889	92 740	1 731 359	0.00	0.00
	∞	118	1.2	76.58	578	59 368	1 278 095	0.00	0.00

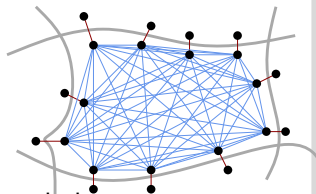
Beobachtungen

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



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 Cliques?
- Hilft geschickte Approximation?

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.

Performance (Synth. TD Europe)

	ε [%]	Custom. [s]	EA query [ms]	Max err [%]
Dijkstra	—	—	1652.6	—
TDCRP	0.1	449	4.1	0.34
	1.0	201	3.2	3.02
	10.0	77	2.5	24.27
inex.TCH	0.1	—	1.8	0.14
	1.0	—	1.4	1.46
	10.0	—	1.3	15.34
ATCH	0.1	—	1.1	0.00
	1.0	—	1.2	0.00
	10.0	—	3.6	0.00

Performance (Synth. TD Europe)

	ϵ [%]	Custom. [s]	EA query [ms]	Max err [%]
Dijkstra	—	—	1652.6	—
TDCRP	0.1	449	4.1	0.34
	1.0	201	3.2	3.02
	10.0	77	2.5	24.27
inex.TCH	0.1	—	1.8	0.14
	1.0	—	1.4	1.46
	10.0	—	1.3	15.34
ATCH	0.1	—	1.1	0.00
	1.0	—	1.2	0.00
	10.0	—	3.6	0.00

ATCH-ähnliche exakte TDCRP-Anfrage möglich:

- uni-direktional \Rightarrow weniger Phasen
- Aber: Unpacking ist teurer, Hopping bei LC-Suchen (kein DAG)
- bisher nicht implementiert

TDCRP robuster gegen Veränderungen in der Eingabe

Network	TCH		TDCRP	
	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?

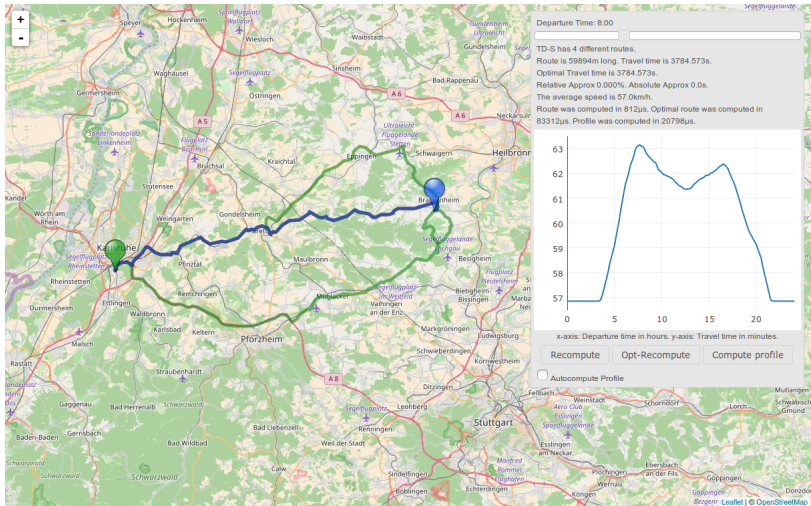
- Bis hierhin: Viele komplizierte Algorithmen
- Geht das nicht einfacher?

- 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

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 Earliest Arrival: TD Germany

	Relative Error [%]		Run T.
	avg.	max.	[ms]
TDCALT-K1.00	0	0	3.77
TDCALT-K1.15	0.051	13.84	1.31
eco SHARC	0	0	19.7
eco L-SHARC	0	0	5.0
heu SHARC	n/r	0.61	0.54
heu L-SHARC	n/r	0.61	0.30
TCH	0	0	1.12
TDCRP (0.1)	0.05	0.25	1.92
TDCRP (1.0)	0.68	2.85	1.66
Freeflow	0.140	12.4	0.12
Avgflow	0.050	6.5	0.14
TD-S+4	0.002	2.0	0.97
TD-S+9	0.001	2.0	2.09

Performance Profil: TD Germany

	Run T. [ms]
eco SHARC	47 388
heu SHARC	847
ATCH $\epsilon=2.5\%$	30
TD-S+P4	19.5
TD-S+P9	22.2

Montag, 25. Juni 2018

- Daniel Delling:
Engineering 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.