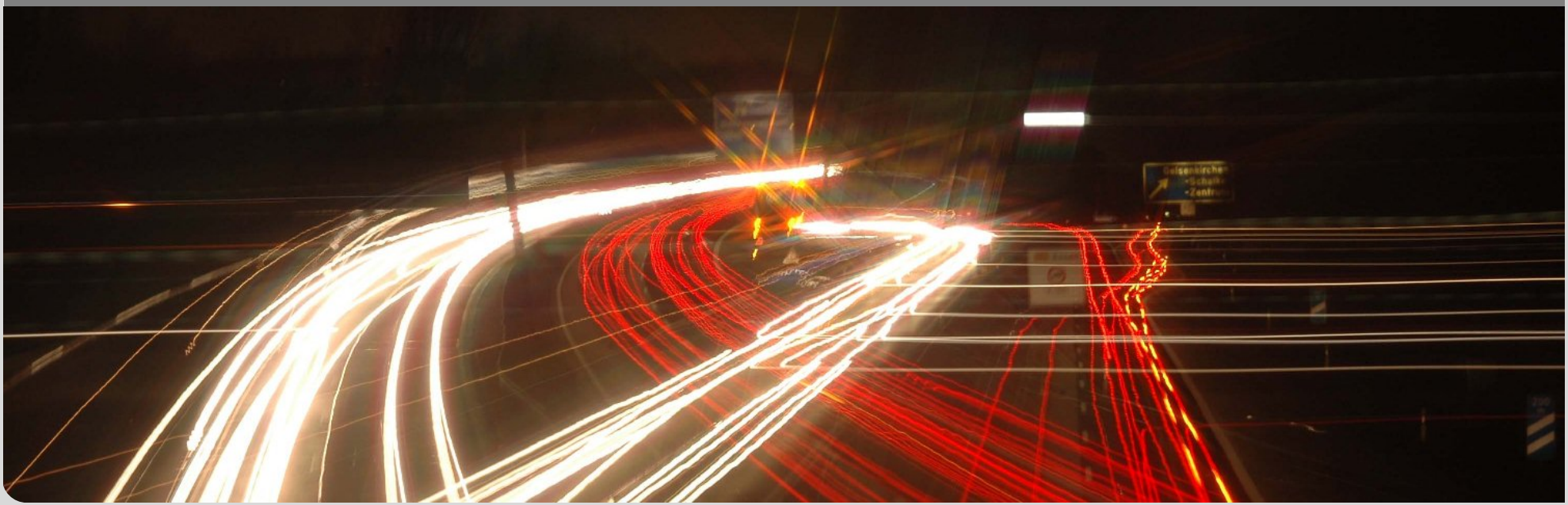
Karlsruher Institut für Technologie

Algorithmen für Routenplanung

11. Vorlesung, Sommersemester 2016

Ben Strasser | 30. Mai 2016

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



Stau

Stauarten

- Wir betrachten zwei Arten von Stau:
  - Unvorhersehbare Staus: z.B. ein Autounfall
  - Vorhersehbare Staus: z.B. Pendler
- Beide Arten fundamental verschieden
- Unvorhersehbare Staus $\rightarrow$ 3-Phasen Technik
- Vorhersehbare Staus $\rightarrow$ zeitabhängigen Kantengewichten

- **Heute:** Zeitabhängige Kantengewichten

Historische Staudaten

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

Historische Staudaten

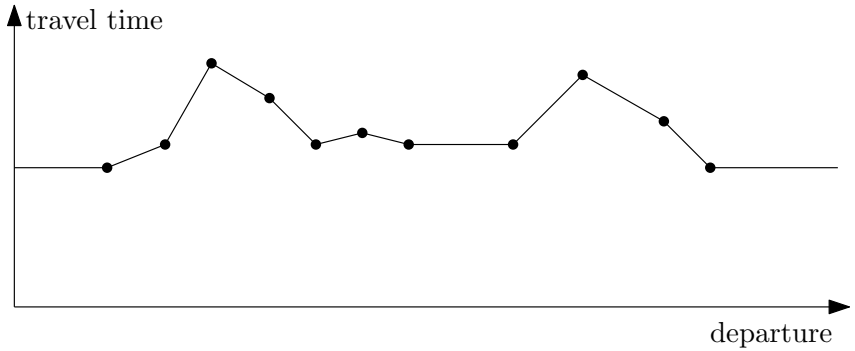
Motivation:

- 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

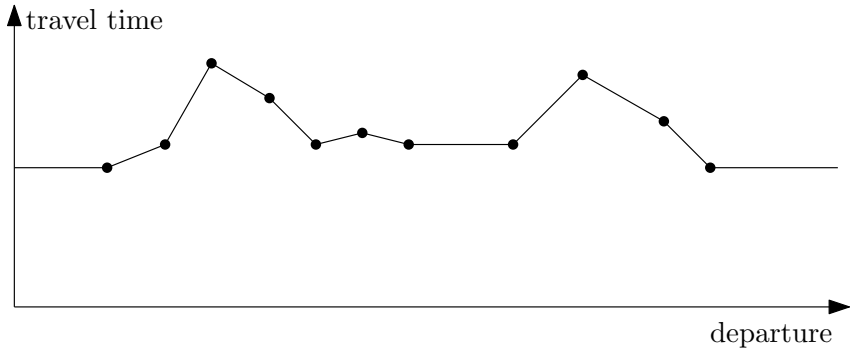
Zeitabhängige Kantengewichte

- **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?

Zeitunabhängige Grundversion von Dijkstras Algorithmus

DIJKSTRA($G = (V, E)$, s)

```
1  forall the nodes v ∈ V do
2      d[v] = ∞, p[v] = NULL
3  d[s] = 0
4  Q.clear(), Q.insert(s, 0) while !Q.empty() do
5      u ← Q.deleteMin() forall the edges e = (u, v) ∈ E do
6          if d[u] + len(e) < d[v] then
7              d[v] ← d[u] + len(e)
8              p[v] ← u
9              if v ∈ Q then Q.decreaseKey(v, d[v])
10             else Q.insert(v, d[v])
```

Zeitabhängige Erweiterung von Dijkstras Algorithmus

DIJKSTRA($G = (V, E)$, s, τ)

```
1  forall the nodes v ∈ V do
2      d[v] = ∞, p[v] = NULL
3  d[s] = 0
4  Q.clear(), Q.insert(s, τ) while !Q.empty() do
5      u ← Q.deleteMin() forall the edges e = (u, v) ∈ E do
6          if d[u] + len(e, d[u]) < d[v] then
7              d[v] ← d[u] + len(e, d[u])
8              p[v] ← u
9              if v ∈ Q then Q.decreaseKey(v, d[v])
10             else Q.insert(v, d[v])
```

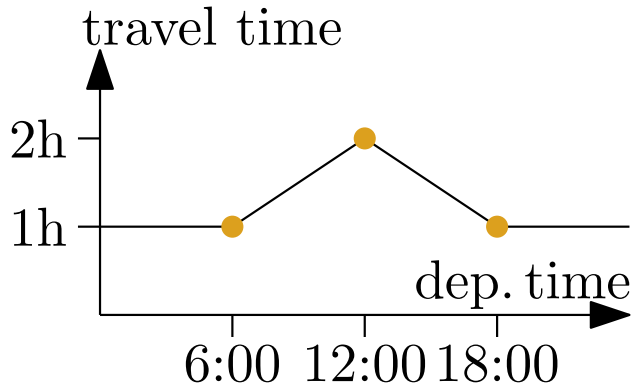

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Dijkstras Algorithmus

- Der Algorithmus von Dijkstra kann leicht angepasst werden
- Bei schneller Kanten-Funktions-Evaluation: Nicht viel langsamer als Grundvariante
- Beschleunigungstechniken?
- Wie Kanten-Funktionen darstellen?

Reisezeit-Funktionen

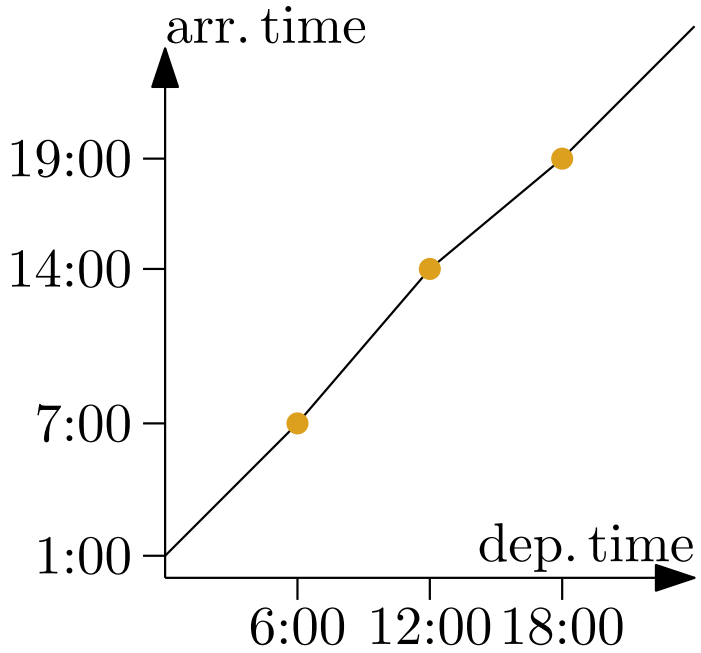
Abfahrtszeit	Reisezeit
6:00	1h
12:00	2h
18:00	1h



- Speichere Interpolationspunkte
- Interpoliere linear

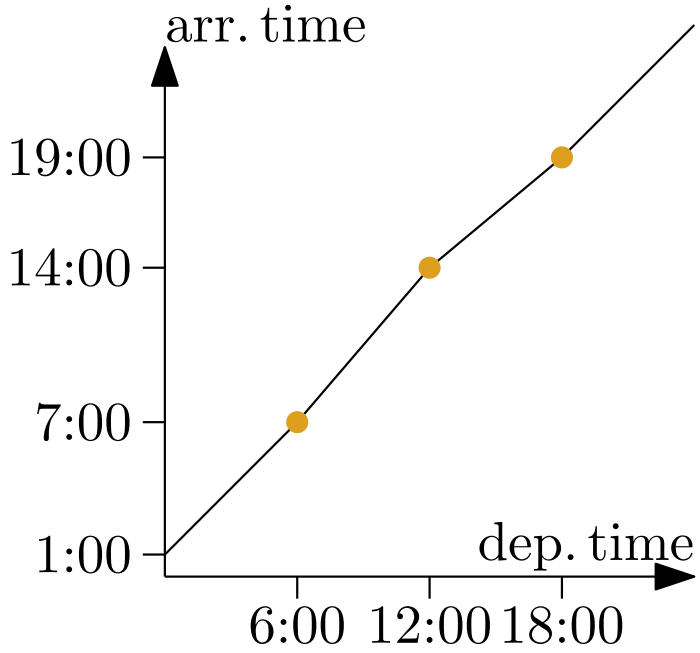
Ankunftszeit-Funktion

Abfahrtszeit	Ankunftszeit
6:00	7:00
12:00	14:00
18:00	19:00

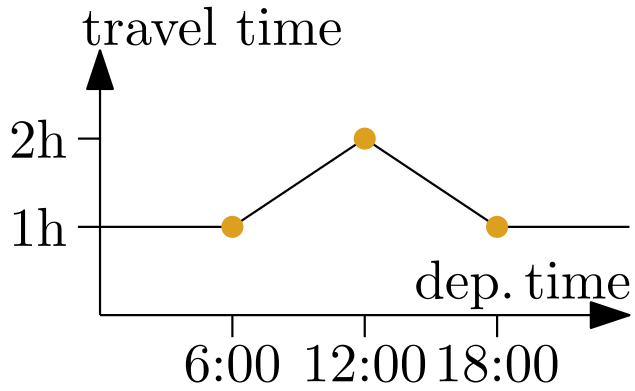


- Speichere Interpolationspunkte
- Interpoliere linear

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

Eigenschaften

FIFO: First-In-First-Out

- Wenn man später los fährt, kommt man später an
- Eigenschaft von den meisten realistischen Straßennetzwerken
- Formal gilt für jede Ankunftszeit-Funktion f:

$$f(x) < f(y) \text{ für } x < y$$

- Aus FIFO folgt, dass f eine Bijektion ist und damit, dass f^{-1} existiert
- Das Problem der frühesten Ankunft ist NP-schwer ohne FIFO

Eigenschaften

FIFO: First-In-First-Out

- Wenn man später los fährt, kommt man später an
- Eigenschaft von den meisten realistischen Straßennetzwerken
- Formal gilt für jede Ankunftszeit-Funktion f:

$$f(x) < f(y) \text{ für } x < y$$

- Aus FIFO folgt, dass f eine Bijektion ist und damit, dass f^{-1} existiert
- Das Problem der frühesten Ankunft ist NP-schwer ohne FIFO

Periodizität

- **Vereinfachung**: Nach einer Periode Π (24h in den Beispielen) wiederholen sich die Muster
- Formal gilt für jede Reisezeitfunktion g:

$$g(x) = g(x + k\Pi) \text{ für alle Ganzzahlen } k$$

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

Funktionsevaluation

Ankunftszeit-Funktion f an Stelle x auswerten:

- Finde Interpolationspunkt (x_l, y_l) vor x und Interpolationspunkt (x_r, y_r) nach x
- Berechne

$$f(x) = y_l + (y_r - y_l) \cdot \frac{x_r - x}{x_r - x_l}$$

- Kann mit einer binären Suche gemacht werden
- Achtung: Sonderfall am Periodenrand

Funktionsevaluation

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

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

Profil-Anfragen

Profil-Problem

- **Eingabe**: Knoten s und t
- **Ausgabe**: Eine Funktion f die eine Abfahrtszeit τ an s auf die entsprechende Ankunftszeit $f(\tau)$ an t abbildet

Profil-Anfragen

Profil-Problem

- **Eingabe**: Knoten s und t
- **Ausgabe**: Eine Funktion f die eine Abfahrtszeit τ an s auf die entsprechende Ankunftszeit $f(\tau)$ an t abbildet

- Nicht sinnvoll berechenbar auf großen Graphen ohne Beschleunigungstechnik
- Für kleinere Graphen kann man eine Erweiterung von Dijkstras Algorithmus einsetzen
  Mehr dazu später

Obere und Untere Schranken

- Viele Beschleunigungstechniken nutzen obere und untere Schranken

- Sei f die Reisezeitfunktion einer Kante e
- Die obere Schranke von e ist $\overline{\text{len}}(e) = \max_x f(x)$
- Die untere Schranke von e ist $\underline{\text{len}}(e) = \min_x f(x)$
- **Intuition**:
  - $\overline{\text{len}}$ = schlimmer geht es nicht
  - $\underline{\text{len}}$ = kein Stau
- Wir erhalten zwei zeitunabhängige Graphen: Der Obere-Schranken-Graph $\overline{G}$ und der Untere-Schranken-Graph $\underline{G}$

Beschleunigungstechniken

ALT [NDLS12]

- Für die Korrektheit müssen die Potentiale gültige untere Schranken zu den Landmarks sein
- $\rightarrow$ Berechne Potentiale auf $\underline{G}$

Beschleunigungstechniken

ALT [NDLS12]

- Für die Korrektheit müssen die Potentiale gültige untere Schranken zu den Landmarks sein
- $\rightarrow$ Berechne Potentiale auf $\underline{G}$
- Diese Potentiale sind nur gut wenn die Reisezeiten nicht zu sehr variieren
- Je höher die Variation, je eher verhält sich ALT wie Dijkstras Algorithmus

Beschleunigungstechniken

Arc-Flags [Del11]

- **Idee**:Setze eine Flagge falls eine Kante zu irgendeinem Zeitpunkt auf einem kürzesten Pfad liegt

Beschleunigungstechniken

Arc-Flags [Del11]

- **Idee**:Setze eine Flagge falls eine Kante zu irgendeinem Zeitpunkt auf einem kürzesten Pfad liegt

- Exakte Vorberechnung benötigt Profilsuchen
- Nicht praktikabel

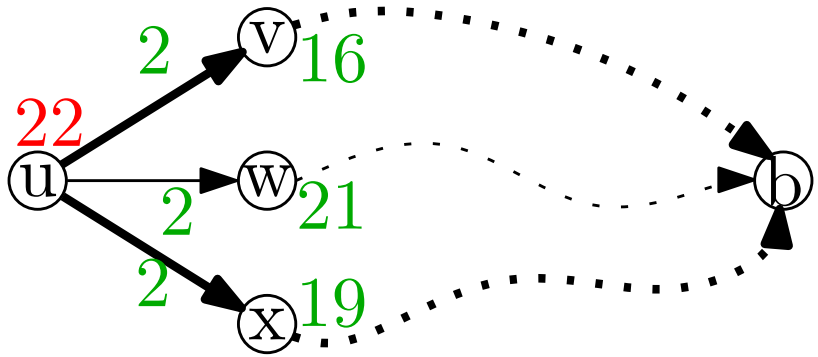
Beschleunigungstechniken

Arc-Flags [Del11]

- Nutze Schranken
- Sei b ein Randknoten
- Berechne $\underline{\text{dist}}(x, b)$ und $\overline{\text{dist}}(x, b)$ für alle x mit Dijkstra
- Markiere eine Kante (x, y) wenn

$$\overline{\text{dist}}(x, b) >= \underline{\text{len}}(x, y) + \underline{\text{dist}}(y, b)$$

- Markiert mehr Kanten als nötig
- Aber korrekt



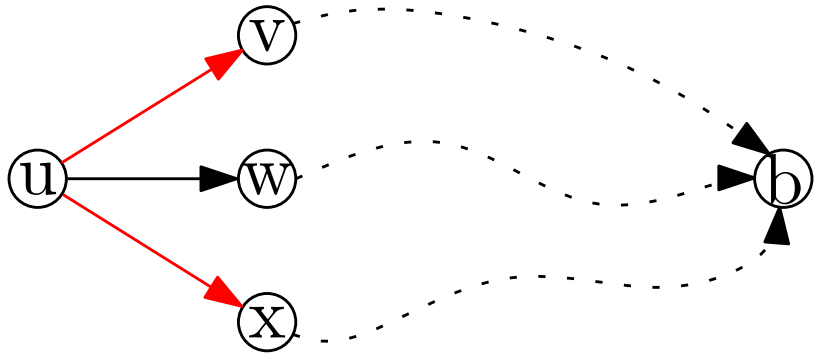
Beschleunigungstechniken

Arc-Flags [Del11]

- Nutze Schranken
- Sei b ein Randknoten
- Berechne $\underline{\text{dist}}(x, b)$ und $\overline{\text{dist}}(x, b)$ für alle x mit Dijkstra
- Markiere eine Kante (x, y) wenn

$$\overline{\text{dist}}(x, b) >= \underline{\text{len}}(x, y) + \underline{\text{dist}}(y, b)$$

- Markiert mehr Kanten als nötig
- Aber korrekt



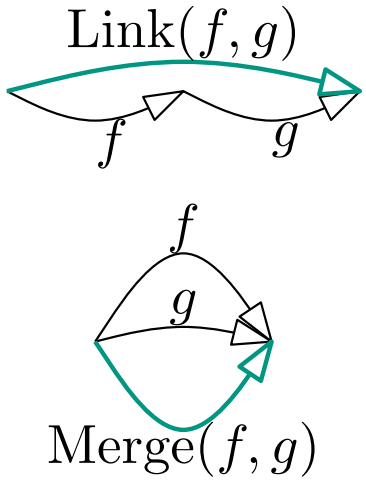
Zeitabhängige Shortcuts

- Shortcuts sind ein zentraler Baustein vieler Techniken
- Um Shortcuts einsetzen zu können brauchen wir zwei Operationen auf den Gewichtungfunktionen

Zeitabhängige Shortcuts

- Shortcuts sind ein zentraler Baustein vieler Techniken
- Um Shortcuts einsetzen zu können brauchen wir zwei Operationen auf den Gewichtungfunktionen

- Linking: Zwei Kanten hintereinander hängen
- Merging: Zwei parallele Kanten zusammenführen



Linking : Berechne $h(x) = g(f(x))$

Function f	
Dep.Time	Arr.Time
6:00	7:00
12:00	14:00
18:00	19:00

Function g	
Dep.Time	Arr.Time
8:00	10:00
10:00	11:00
15:00	17:00

Linking : Berechne $h(x) = g(f(x))$

Karlsruher Institut für Technologie

Function f	
Dep.Time	Arr.Time
6:00	7:00
12:00	14:00
18:00	19:00

Function g	
Dep.Time	Arr.Time
8:00	10:00
10:00	11:00
15:00	17:00

Function h		
Dep.Time	Mid.Time	Arr.Time
6:00	7:00	?
?	8:00	10:00
?	10:00	11:00
12:00	14:00	?
?	15:00	17:00
18:00	19:00	?

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Linking : Berechne $h(x) = g(f(x))$

Karlsruher Institut für Technologie

Function f	
Dep.Time	Arr.Time
6:00	7:00
12:00	14:00
18:00	19:00

Function g	
Dep.Time	Arr.Time
8:00	10:00
10:00	11:00
15:00	17:00

Function h		
Dep.Time	Mid.Time	Arr.Time
6:00	7:00	g(7:00)
f^{-1}(8:00)	8:00	10:00
f^{-1}(10:00)	10:00	11:00
12:00	14:00	g(14:00)
f^{-1}(15:00)	15:00	17:00
18:00	19:00	g(19:00)

Linking : Berechne $h(x) = g(f(x))$

Karlsruher Institut für Technologie

Function f	
Dep.Time	Arr.Time
6:00	7:00
12:00	14:00
18:00	19:00

Function g	
Dep.Time	Arr.Time
8:00	10:00
10:00	11:00
15:00	17:00

Function h		
Dep.Time	Mid.Time	Arr.Time
6:00	7:00	9:00
≈6:51	8:00	10:00
≈8:34	10:00	11:00
12:00	14:00	15:48
13:12	15:00	17:00
18:00	19:00	21:00

Linking : Berechne $h(x) = g(f(x))$

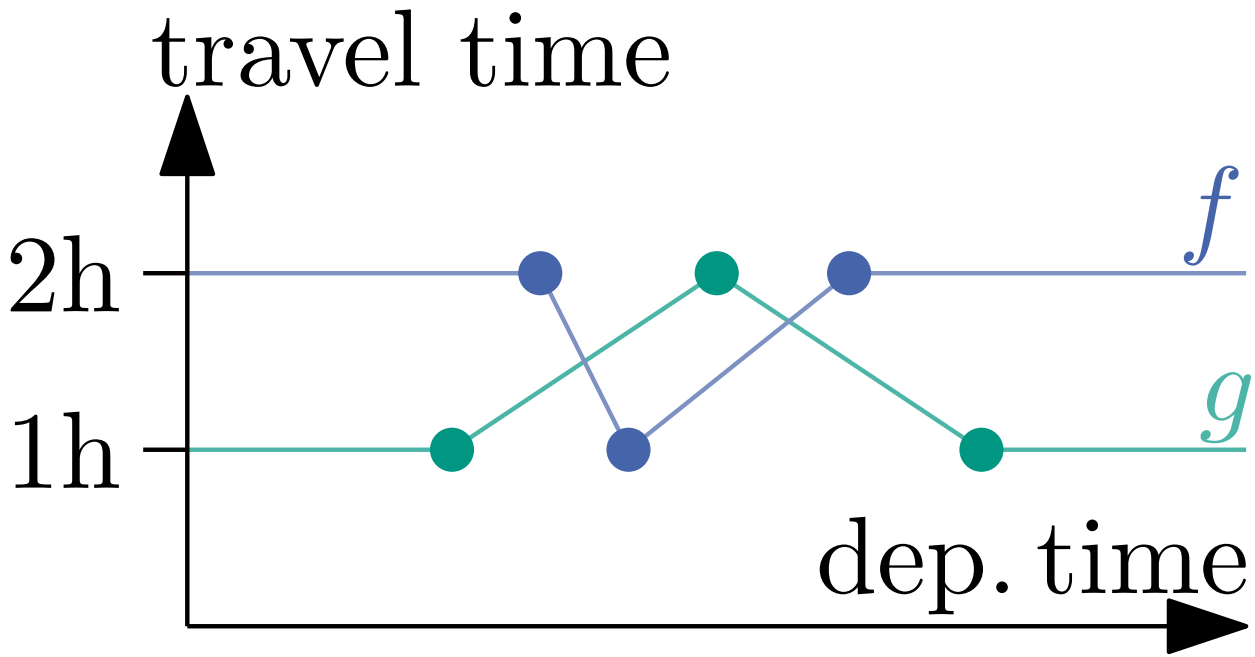
Karlsruher Institut für Technologie

Function f	
Dep.Time	Arr.Time
6:00	7:00
12:00	14:00
18:00	19:00

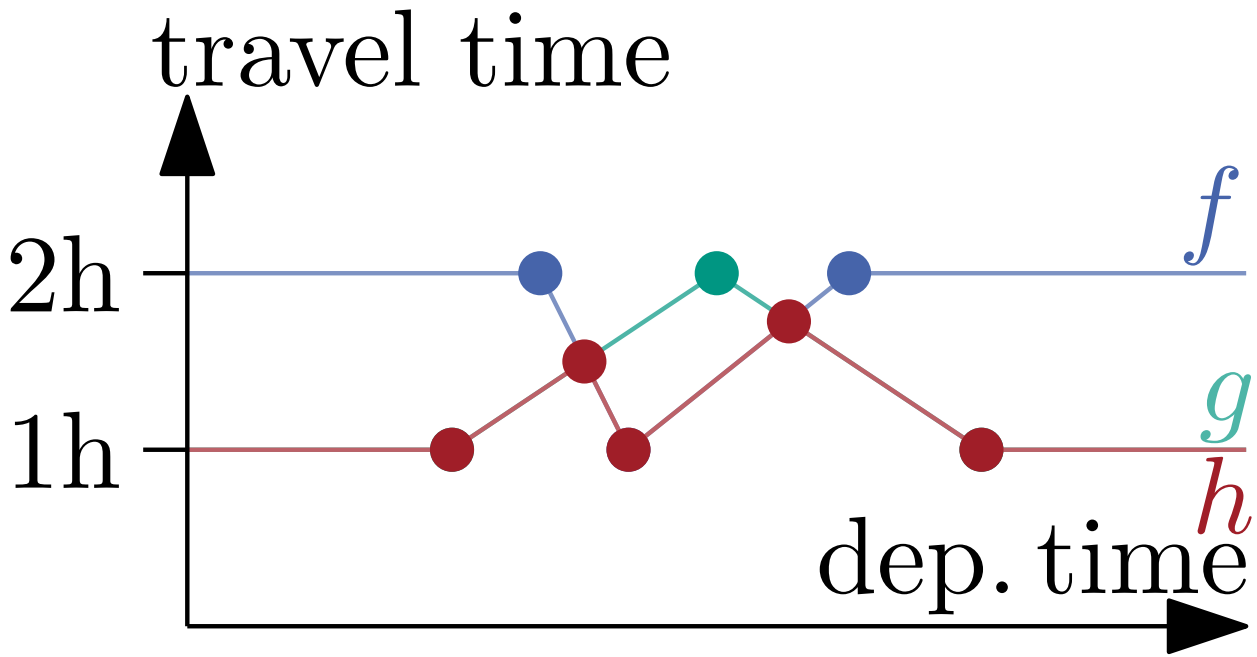
Function g	
Dep.Time	Arr.Time
8:00	10:00
10:00	11:00
15:00	17:00

Function h	
Dep.Time	Arr.Time
6:00	9:00
≈6:51	10:00
≈8:34	11:00
12:00	15:48
13:12	17:00
18:00	21:00

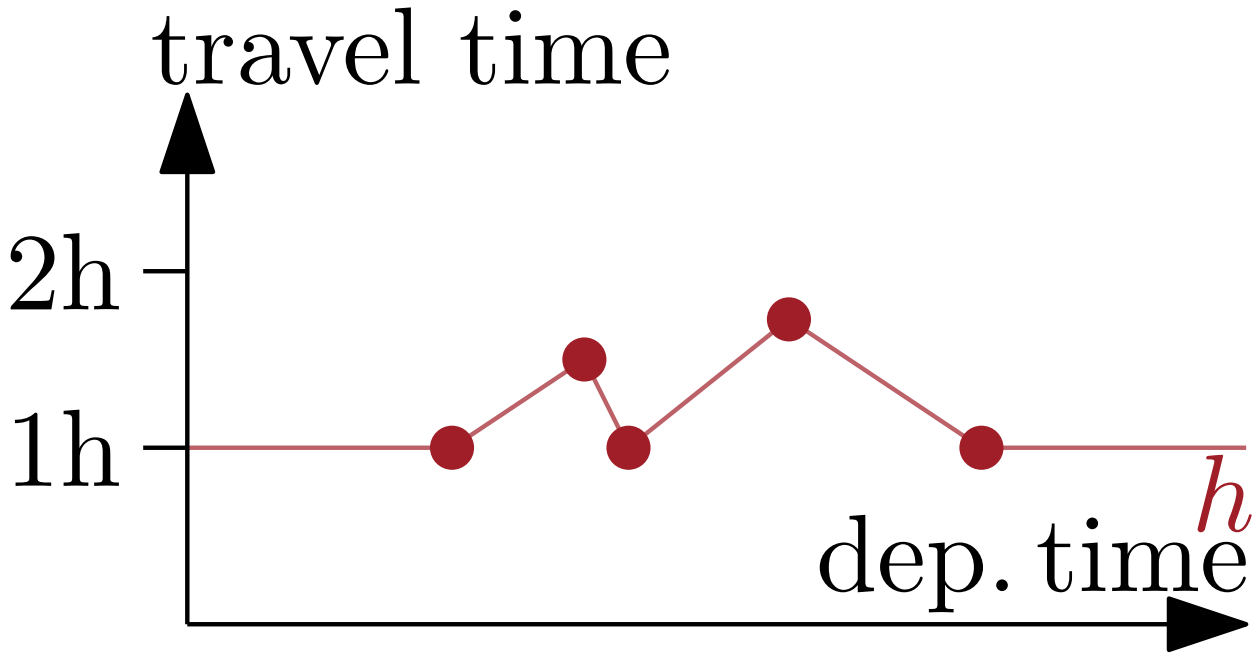
Merging : $h(x) = \min\{f(x), g(x)\}$



Merging : $h(x) = \min\{f(x), g(x)\}$



Merging : $h(x) = \min\{f(x), g(x)\}$



Komplexität

$|f| = \#$ Interpolationspunkte von f

Linking:

- Platz: $|h| \leq |f| + |g|$
- Laufzeit: $O(|f| + |g|)$
  - Koordinierte Iteration über beide Funktionen

Merging:

- Platz: $|h| \leq 2 \cdot (|f| + |g|)$
  (Beweis auf nächster Folie)
- Laufzeit: $O(|f| + |g|)$
  - Koordinierte Iteration über beide Funktionen

Komplexität

Zu Zeigen:

Beim Mergen: $|h| \leq 2 \cdot (|f| + |g|)$

- Betrachte X die Menge aller x-Koordinaten in denen f oder g einen Interpolationspunkt hat
- Aus Konstruktion folgt $|X| \leq |f| + |g|$
- Betrachte Abschnitt zwischen zwei aufeinanderfolgenden Elementen aus X
- Es gibt höchstens $|X|$ Abschnitte
- Auf jedem Abschnitt ist sowohl f als auf g linear
- Es gibt pro Abschnitt höchstens ein Schnittpunkt
- In h gibt es pro Abschnitt deswegen höchstens ein Interpolationspunkt
- An allen Stellen in X kann h höchstens ein Interpolationspunkt haben
- $2|X| \leq 2(|f| + |g|)$ ist deswegen eine obere Schranke

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 a
- 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

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 a
- 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?

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 a
- 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 waschen
- Offen Frage: Kann man diese Divergenz zeigen?

Zeitabhängige CH

- Wir können nun zeitabhängige Shortcuts bauen
- Können wir damit eine zeitabhängige CH bauen?

Zeitabhängige CH

- Wir können nun zeitabhängige Shortcuts bauen
- Können wir damit eine zeitabhängige CH bauen?
- Wie machen wir die Rückwärtssuche ohne die Ankunftszeit zu kennen?

Zeitabhängige CH

- Wir können nun zeitabhängige Shortcuts bauen
- Können wir damit eine zeitabhängige CH bauen?
- Wie machen wir die Rückwärtssuche ohne die Ankunftszeit zu kennen?
- Wir brauchen eine Profilanfrage

Erweiterung von Dijkstras Algorithmus für Profilanfragen

- 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

Erweiterung von Dijkstras Algorithmus für Profilanfragen

Ziel: finde kürzesten Weg für alle Abfahrtszeitpunkte

`Profile-Search`$(G = (V, E), s)$

```
1  d_*[s] = Funktion die alles auf 0 abbildet
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          x ← Merge(Link(d_*[u], len(e)), d_*[v])
7          if d_*[v] ≠ x then
8              d_*[v] ← x
9              if v ∈ Q then Q.decreaseKey(v, d[v])
10
11             else Q.insert(v, d[v])
```

Zeitabhängige CH

- Haben wir nun alles?
- Anfrage langsam wegen Profilanfrage
- Großer Speicherverbrauch, da lange Shortcuts viele Interpolationspunkte brauchen

Zeitabhängige CH

- Haben wir nun alles?
- Anfrage langsam wegen Profilanfrage
- Großer Speicherverbrauch, da lange Shortcuts viele Interpolationspunkte brauchen

- Nutze obere und untere Schranken um nur relevante Teile des Profils zu berechnen
- Speichere approximierte Gewichtsfunktionen
  - Nur approximierte Ankunftszeit

Zeitabhängige CH

- Haben wir nun alles?
- Anfrage langsam wegen Profilanfrage
- Großer Speicherverbrauch, da lange Shortcuts viele Interpolationspunkte brauchen

- Nutze obere und untere Schranken um nur relevante Teile des Profils zu berechnen
- Speichere approximierte Gewichtsfunktionen
  - Nur approximierte Ankunftszeit

- Nutze approximierte um relevanten Teil des Eingabegraphs zu extrahieren
- Führe die Erweiterungen von Dijkstras auf Teilgraph aus

Zeitabhängige CH

- Haben wir nun alles?
- Anfrage langsam wegen Profilanfrage
- Großer Speicherverbrauch, da lange Shortcuts viele Interpolationspunkte brauchen

- Nutze obere und untere Schranken um nur relevante Teile des Profils zu berechnen
- Speichere approximierte Gewichtsfunktionen
  - Nur approximierte Ankunftszeit

- Nutze approximierte um relevanten Teil des Eingabegraphs zu extrahieren
- Führe die Erweiterungen von Dijkstras auf Teilgraph aus

- Mit CH können auch Profilanfragen beantwortet werden [BGSV13]

Zeitabhängiges MLO

- Auf die gleiche Weise wie bei der CH kann man MLO/MLD/CRP zeitabhängig machen
- [BDPW16]
- Customizable?

Zeitabhängiges MLO

- Auf die gleiche Weise wie bei der CH kann man MLO/MLD/CRP zeitabhängig machen
- [BDPW16]
- Customizable?
- Nicht wirklich: Gewichte anpassen braucht mehr als 1min. (Aber viel schneller als CH.)

Zeitabhängiges MLO

- Auf die gleiche Weise wie bei der CH kann man MLO/MLD/CRP zeitabhängig machen
- [BDPW16]
- Customizable?
- Nicht wirklich: Gewichte anpassen braucht mehr als 1min. (Aber viel schneller als CH.)

Probleme

- Bei CH: Knotenorndung wird so gewählt, dass wenige Interpolationspunkte entstehen
- Bei MLO: Partitionierung ist unabhängig von Gewichten
- $\rightarrow$ Bei MLO gibt es noch mehr Interpolationspunkte pro Shortcut
- $\rightarrow$ Approximation

Approximation

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 (Imai-Iri) nach jedem Level.

Was fehlt?

- Bis hierhin: Viele mehr oder weniger komplexe Algorithmen
- Was fehlt?

Was fehlt?

- Bis hierhin: Viele mehr oder weniger komplexe Algorithmen
- Was fehlt?
- Ein sinnvoller Grundalgorithmus:
  - Berechne eine CH auf dem untere-Schranken-Graph
  - Berechne zeitunabhängigen Pfad
  - Rechne zeitabhängige Fahrzeit entlang des Pfads nach
- heißt Freeflow

Instanzen

- Es ist sehr schwer an zeitabhängige Daten zu kommen

Instanzen

- Es ist sehr schwer an zeitabhängige Daten zu kommen

Wir betrachten zwei Instanzen:

- Deutschland
  - Basiert auf etwa 10 Jahre alten Realwelt-Daten von Deutschland
- Europa
  - Basiert auf zeitunabhängigem Europa Straßengraphen
  - Zeitabhängigkeit künstlich generiert

Experimente

		Relative Error [%]		Run T.
		avg.	max.	[ms]
TDCALT-K1.00	[DN12]	0	0	5.36
TDCALT-K1.15	[DN12]	0.051	13.84†	1.87
eco L-SHARC	[Del11]	0	0	6.31
heu L-SHARC	[Del11]	n/r	0.61	0.38
TD-CH	[BGSV13]	0	0	0.75
TDCRP (1.0)	[BDPW16]	0.68	2.85	1.66
Freeflow		0.03	2.52	0.24
FLAT-SR_{2000}	[KMP+16]	n/r	1.444†	1.28

Instanz: Deutschland

Experimente

		Relative Error [%]		Run T.
		avg.	max.	[ms]
TDALT-K1.20	[NDLS12]	0.652	18.19	180.9
TDCALT-K1.00	[DN12]	0	0	121.4
TDCALT-K1.25	[DN12]	0.549	15.52†	5.4
eco L-SHARC	[Del11]	0	0	38.29
heu L-SHARC	[Del11]	n/r	1.60	2.13
TD-CH	[BGSV13]	0	0	2.11
TDCRP (1.0)	[BDPW16]	0.54	3.21	5.75
Freeflow		0.50	19.32	0.29

Instanz: Europa

Literatur I

Moritz Baum, Julian Dibbelt, Thomas Pajor, and Dorothea Wagner.
Dynamic time-dependent route planning in road networks with user preferences.
In *Proceedings of the 15th International Symposium on Experimental Algorithms (SEA'16)*, Lecture Notes in Computer Science. Springer, 2016.

Gernot Veit Batz, Robert Geisberger, Peter Sanders, and Christian Vetter.
Minimum time-dependent travel times with contraction hierarchies.
ACM Journal of Experimental Algorithmics, 18(1.4):1–43, April 2013.

Daniel Delling.
Time-dependent SHARC-routing.
Algorithmica, 60(1):60–94, May 2011.

Daniel Delling and Giacomo Nannicini.
Core routing on dynamic time-dependent road networks.
Informs Journal on Computing, 24(2):187–201, 2012.

Literatur II

Spyros Kontogiannis, George Michalopoulos, Georgia Papastavrou, Andreas Paraskevopoulos, Dorothea Wagner, and Christos Zaroliagis.

Engineering oracles for time-dependent road networks.

In *Proceedings of the 18th Meeting on Algorithm Engineering and Experiments (ALENEX'16)*. SIAM, 2016.

Giacomo Nannicini, Daniel Delling, Leo Liberti, and Dominik Schultes.

Bidirectional A* search on time-dependent road networks.

Networks, 59:240–251, 2012.

Best Paper Award.