Karlsruher Institut für Technologie

Algorithmen für Routenplanung

15. Vorlesung, Sommersemester 2023

Jonas Sauer | 19. Juni 2023

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
- Wir können nicht für jeden Stau die ganze Vorberechnung wiederholen

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

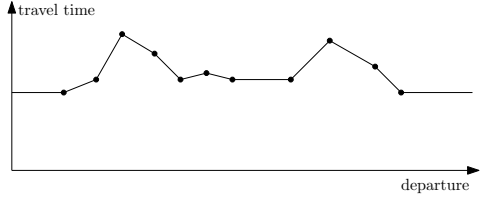
Motivation:

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

- Aggregiere historische Daten zu einer Vorhersage für jeden Wochentag

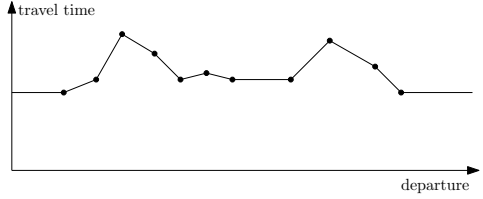
Zeitabhängige Kantengewichte

- **Bisher:** An jeder Kante steht ein skalares Kantengewicht
- **Jetzt:** An jeder Kante steht eine Funktion
- Bildet den Zeitpunkt, an dem die Kante betreten wird, auf die Fahrzeit ab



Zeitabhängige Kantengewichte

- **Bisher:** An jeder Kante steht ein skalares Kantengewicht
- **Jetzt:** An jeder Kante steht eine Funktion
- Bildet den Zeitpunkt, an dem die Kante betreten wird, auf die Fahrzeit ab



Problemstellung

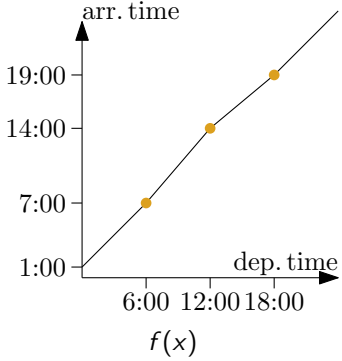
Mit zeitabhängigen Gewichten ist die Frage

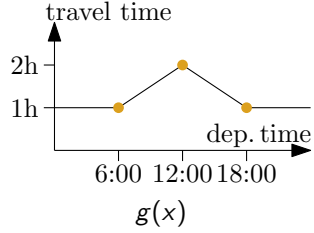
- Wie komme ich von s nach t?

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

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

Zwei Sichtweisen





- Zwei Sichtweisen auf dieselbe Information
- $f(x) = g(x) + x$
- Aussagen in einer Sichtweise lassen sich immer auf die andere übertragen
- Wir wechseln zwischen beiden Sichtweisen und nehmen immer die, die gerade besser passt

FIFO-Eigenschaft

Definition

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

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

FIFO-Eigenschaft

Definition

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

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

- Interpretation: „Später losfahren lohnt sich nie“
- Automatisch erfüllt, wenn Warten an Knoten erlaubt ist
- Gegenbeispiel: Temporäre Straßensperrungen

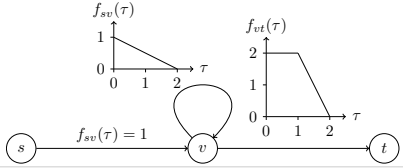
FIFO-Eigenschaft

Definition

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

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

- Interpretation: „Später losfahren lohnt sich nie“
- Automatisch erfüllt, wenn Warten an Knoten erlaubt ist
- Gegenbeispiel: Temporäre Straßensperrungen
- Kürzeste-Wege-Problem ohne FIFO-Eigenschaft ist NP-schwer
  - Schwach NP-schwer für Funktionen über $\mathbb{N}$ (Reduktion von Partition)
  - Stark NP-schwer schon für stückweise lineare Funktionen über $\mathbb{Q}$ mit allen Stützstellen in $\mathbb{N}^2$



FIFO-Eigenschaft

Definition

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

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

- Interpretation: „Später losfahren lohnt sich nie“
- Automatisch erfüllt, wenn Warten an Knoten erlaubt ist
- Gegenbeispiel: Temporäre Straßensperrungen
- Kürzeste-Wege-Problem ohne FIFO-Eigenschaft ist NP-schwer
  - Schwach NP-schwer für Funktionen über $\mathbb{N}$ (Reduktion von Partition)
  - Stark NP-schwer schon für stückweise lineare Funktionen über $\mathbb{Q}$ mit allen Stützstellen in $\mathbb{N}^2$
  - Kürzeste Wege ggf. nicht mehr wohldefiniert

FIFO-Eigenschaft

Definition

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

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

- Interpretation: „Später losfahren lohnt sich nie“
- Automatisch erfüllt, wenn Warten an Knoten erlaubt ist
- Gegenbeispiel: Temporäre Straßensperrungen
- Kürzeste-Wege-Problem ohne FIFO-Eigenschaft ist NP-schwer
  - Schwach NP-schwer für Funktionen über $\mathbb{N}$ (Reduktion von Partition)
  - Stark NP-schwer schon für stückweise lineare Funktionen über $\mathbb{Q}$ mit allen Stützstellen in $\mathbb{N}^2$
  - Kürzeste Wege ggf. nicht mehr wohldefiniert

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

Weitere Darstellungsformen

Akademische Sichtweise:

- Für jede Kante ist eine Funktion gegeben
- Darstellung als Folge von Stützpunkten
- Dazwischen lineare Interpolation

Probleme:

- 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 Intervalle ein (z.B. 24 $\times$ 1 Stunde)
  - Intervalle müssen nicht gleich groß sein
  - Unterteilung kann pro Kante variieren
- Konstante Reisegeschwindigkeit pro Intervall
  - Geschwindigkeiten können gerundet sein: z.B. 5-km/h-Schritte
- Alle Fahrzeuge auf einer Kante ändern spontan ihre Reisegeschwindigkeit, wenn die Tageszeit eine Intervallgrenze überschreitet
  - Aus dieser Eigenschaft folgt FIFO

Weitere Darstellungsformen

Option 2:

- Speichere kleine globale Menge von Funktionskurven $\mathbb{F}$ explizit
  - Funktionen aus $\mathbb{F}$ werden meistens durch lineare Interpolation zwischen einer Menge an Stützpunkten dargestellt
  - Jede dieser Funktionen ist FIFO
- An jeder Kante speichert man einen Satz an skalaren Attributen:
  - Funktion-ID in $\mathbb{F}$
  - Kantenlänge
  - Diverse Strauchungs- und Streckungsfaktoren, die die Funktion aus $\mathbb{F}$ transformieren

Herausforderung

Hauptprobleme:

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

Herausforderung

Hauptprobleme:

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

Vorgehen:

- Modellierung
- Anpassung Dijkstra
- Anpassung Beschleunigungstechniken

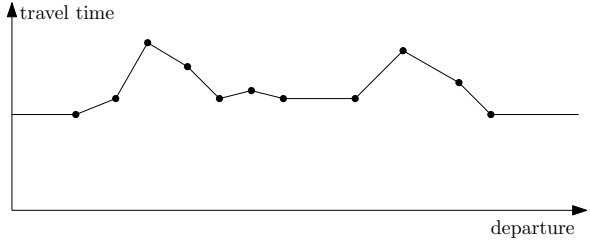
Zeitabhängige Straßennetzwerke

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, nur Reisezeit
- Kanten gemischt zeitabhängig und konstant
- Variable (!) Anzahl Interpolationspunkte pro Kante

Diskussion

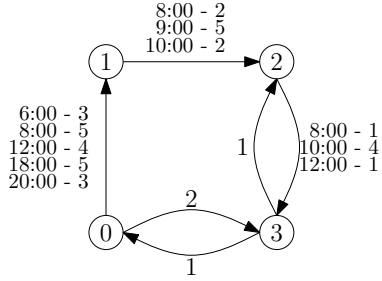
Eigenschaften „Zeitabhängigkeit“:

- Topologie ändert sich nicht, nur Reisezeit
- Kanten gemischt zeitabhängig und konstant
- Variable (!) Anzahl Interpolationspunkte pro Kante

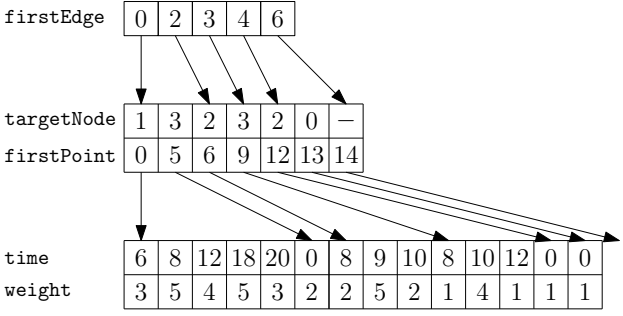
Voraussetzung:

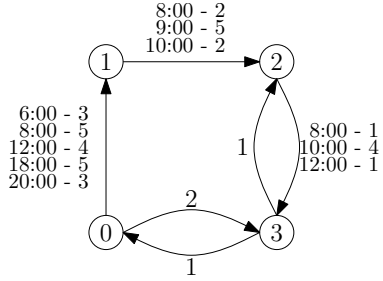
- FIFO gilt auf allen Kanten

Datenstruktur

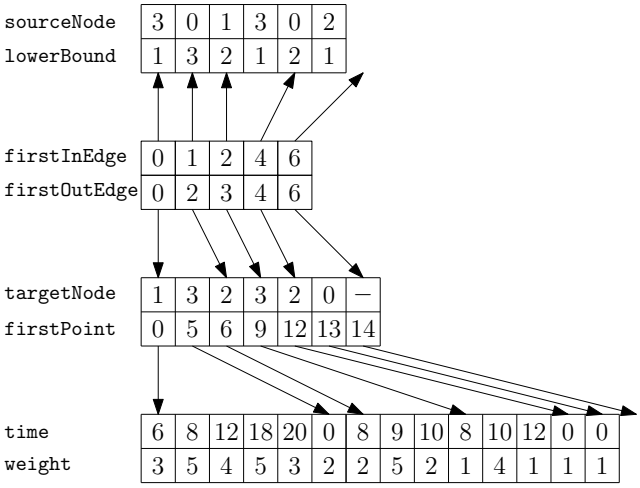


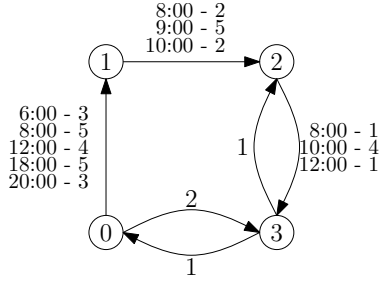
Datenstruktur





Datenstruktur





Anfrageszenarien

Zeit-Anfrage:

- Finde kürzesten Weg für Abfahrtszeit τ
- Analog zu Dijkstra?

Anfrageszenarien

Zeit-Anfrage:

- Finde kürzesten Weg für Abfahrtszeit τ
- Analog zu Dijkstra?

Profil-Anfrage:

- Finde kürzeste Reisezeit 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             else Q.insert(v, d_τ[v])
```

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

Diskussion Profil-Anfragen

Beobachtungen:

- Operationen auf Funktionen
- Knotenlabel: Funktion
- Knotenlabel nicht skalar $\Rightarrow$ keine Totalordnung der Knotenlabel
- Wonach Priority Queue ordnen?
- Priorität im Prinzip frei wählbar
  (z.B.: $\underline{d}[u] :=$ Minimum der Funktion $d_*[u]$)
- Knoten können mehrfach besucht werden $\Rightarrow$ label-correcting

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:

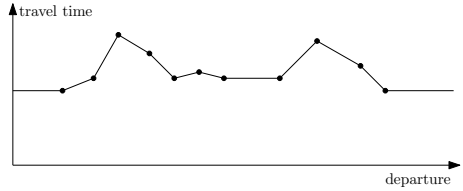
- Auswertung
- Linken $\oplus$
- Minimumsbildung
- Vergleich $\not\geq$
  (ist analog zu Minimumsbildung)

Auswertung

Auswertung von $f(\tau)$:

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

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

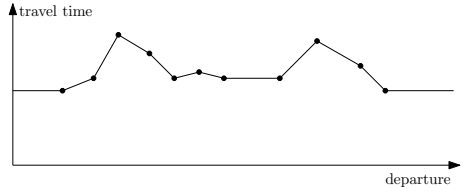


Auswertung

Auswertung 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:

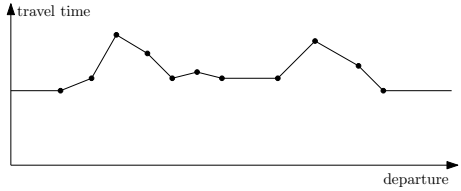
- Finden von t_i und t_{i+1}
  - Achtung: Sonderfall am Periodenrand
- Theoretisch:
  - Lineare Suche: $\mathcal{O}(|I|)$
  - Binäre Suche: $\mathcal{O}(\log_2 |I|)$

Auswertung

Auswertung 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
- Praktisch:
  - Binäre Suche: $\mathcal{O}(\log_2 |I|)$, aber schlecht für Pipelining
  - Oder lineare Suche mit Startpunkt $\frac{\tau}{\Pi} \cdot |I|$
    (wobei Π die Periodendauer ist)
    - Gut, wenn Punkte uniform verteilt
  - Oder Startpunkte für Zeitintervalle merken
  - Kommt drauf an:
    - Maschine, Cacheeffekte, Größe der Punktearrays, Verteilung der Punkte, ...
    - Benchmarken!

Linken – Reisezeit-Funktionen

Definition

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

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

Oder

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

Linken – Ankunftszeit-Funktionen

Definition

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

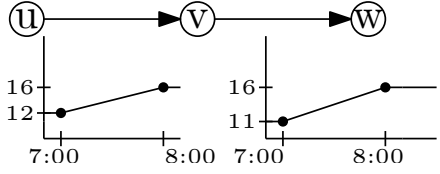
$$f \oplus g := g \circ f$$

Oder

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

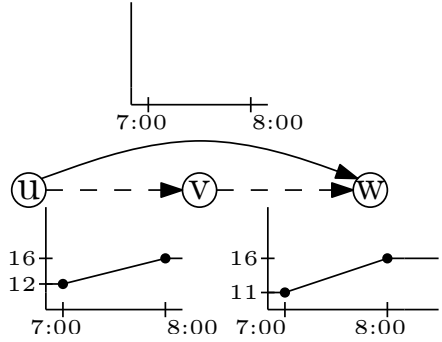
Linken – Reisezeit-Funktionen

Linken zweier Reisezeit-Funktionen f und g:



Linken – Reisezeit-Funktionen

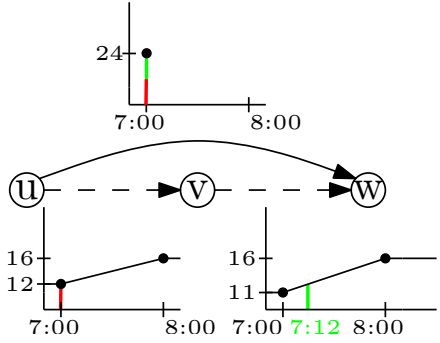
Linken zweier Reisezeit-Funktionen f und g:



Linken – Reisezeit-Funktionen

Linken zweier Reisezeit-Funktionen f und g:

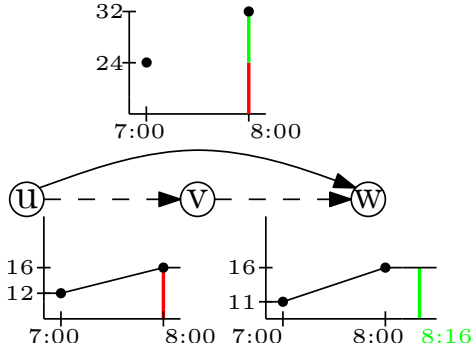
- $f \oplus g$ enthält auf jeden Fall $\{(t_1^f, w_1^f + g(t_1^f + w_1^f)), \ldots, (t_l^f, w_l^f + g(t_l^f + w_l^f))\}$



Linken – Reisezeit-Funktionen

Linken zweier Reisezeit-Funktionen f und g:

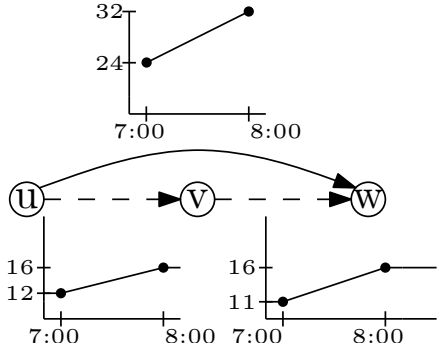
- $f \oplus g$ enthält auf jeden Fall $\{(t_1^f, w_1^f + g(t_1^f + w_1^f)), \ldots, (t_l^f, w_l^f + g(t_l^f + w_l^f))\}$



Linken – Reisezeit-Funktionen

Linken zweier Reisezeit-Funktionen f und g:

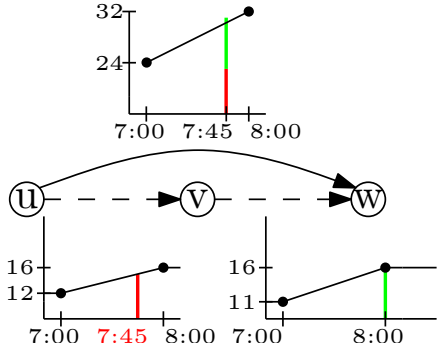
- $f \oplus g$ enthält auf jeden Fall $\{(t_1^f, w_1^f + g(t_1^f + w_1^f)), \ldots, (t_l^f, w_l^f + g(t_l^f + w_l^f))\}$



Linken – Reisezeit-Funktionen

Linken zweier Reisezeit-Funktionen f und g:

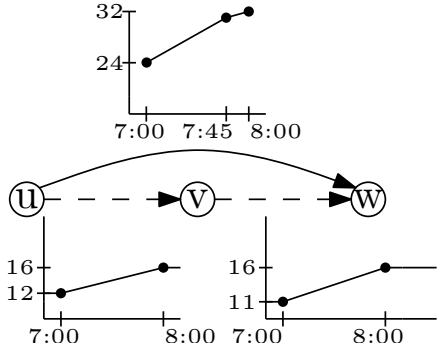
- $f \oplus g$ enthält auf jeden Fall $\{(t_1^f, w_1^f + g(t_1^f + w_1^f)), \ldots, (t_l^f, w_l^f + g(t_l^f + w_l^f))\}$



Linken – Reisezeit-Funktionen

Linken zweier Reisezeit-Funktionen f und g:

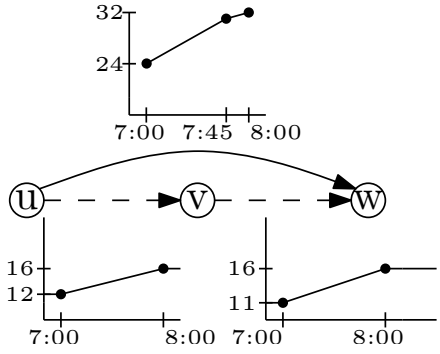
- $f \oplus g$ enthält auf jeden Fall $\{(t_1^f, w_1^f + g(t_1^f + w_1^f)), \ldots, (t_l^f, w_l^f + g(t_l^f + w_l^f))\}$
- Zusätzliche Interpolationspunkte an t_j^{-1} mit $t_j^{-1} + f(t_j^{-1}) = t_j^g$



Linken – Reisezeit-Funktionen

Linken zweier Reisezeit-Funktionen f und g:

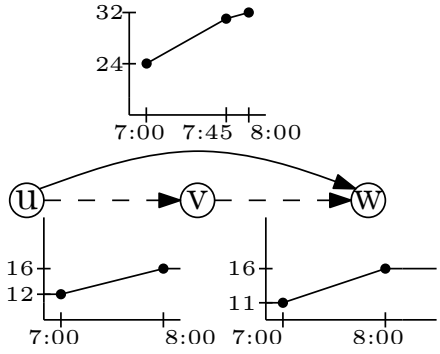
- $f \oplus g$ enthält auf jeden Fall $\{(t_1^f, w_1^f + g(t_1^f + w_1^f)), \ldots, (t_l^f, w_l^f + g(t_l^f + w_l^f))\}$
- Zusätzliche Interpolationspunkte an t_j^{-1} mit $t_j^{-1} + f(t_j^{-1}) = t_j^g$
- Füge $(t_j^{-1}, f(t_j^{-1}) + w_j^g)$ für alle Punkte von g zu $f \oplus g$ hinzu



Linken – Reisezeit-Funktionen

Linken zweier Reisezeit-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 $t_j^{-1} + f(t_j^{-1}) = t_j^g$
- Füge $(t_j^{-1}, f(t_j^{-1}) + w_j^g)$ für alle Punkte von g zu $f \oplus g$ hinzu
- Durch linearen Sweeping-Algorithmus implementierbar



Linken – Ankunftszeit-Funktionen

Linken zweier Ankunftszeit-Funktionen f und g:

- Für jeden Punkt (t_i^f, w_i^f) von f enthält $f \oplus g$ den Punkt $\left(t_i^f, g(w_i^f)\right)$
- Für jeden Punkt (t_j^g, w_j^g) von g enthält $f \oplus g$ den Punkt $\left(f^{-1}(t_j^g), w_j^g\right)$

$\Rightarrow$ viel einfacher als mit Reisezeit-Funktionen

Linken – Ankunftszeit-Funktionen

Linken zweier Ankunftszeit-Funktionen f und g:

- Für jeden Punkt (t_i^f, w_i^f) von f enthält $f \oplus g$ den Punkt $\left(t_i^f, g(w_i^f)\right)$
- Für jeden Punkt (t_j^g, w_j^g) von g enthält $f \oplus g$ den Punkt $\left(f^{-1}(t_j^g), w_j^g\right)$

$\Rightarrow$ viel einfacher als mit Reisezeit-Funktionen

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)

Linken – Ankunftszeit-Funktionen

Linken zweier Ankunftszeit-Funktionen f und g:

- Für jeden Punkt (t_i^f, w_i^f) von f enthält $f \oplus g$ den Punkt $\left(t_i^f, g(w_i^f)\right)$
- Für jeden Punkt (t_j^g, w_j^g) von g enthält $f \oplus g$ den Punkt $\left(f^{-1}(t_j^g), w_j^g\right)$

$\Rightarrow$ viel einfacher als mit Reisezeit-Funktionen

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)
- Funktionsauswertung analog zu der von f
- Die Interpolationspunkte von f^{-1} müssen nicht explizit gespeichert werden

Diskussion Linken

Laufzeit:

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

Diskussion Linken

Laufzeit:

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

Speicherverbrauch:

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

Diskussion Linken

Laufzeit:

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

Speicherverbrauch:

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

Problem:

- In Profilsuchen können Pfade aus tausenden Kanten bestehen

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

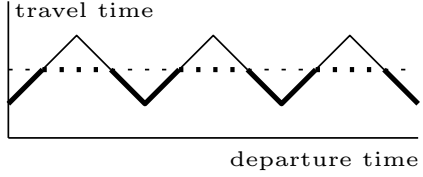
Problem:

- In Profilsuchen können Pfade aus tausenden Kanten bestehen
- Shortcuts...

Mergen – Reisezeit-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



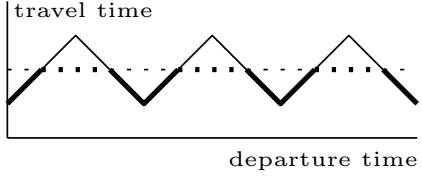
Mergen – Reisezeit-Funktionen

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 Interpolationspunkte

- Je länger ein Shortcut, desto mehr Interpolationspunkte
- Kleiner Suchraum bezüglich Anzahl Knoten und Kanten
- Aber trotzdem viele Interpolationspunkte
- Teure Vorberechnungen

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 Operationen bleiben in den rationalen Zahlen

Linken & Mergen: Probleme

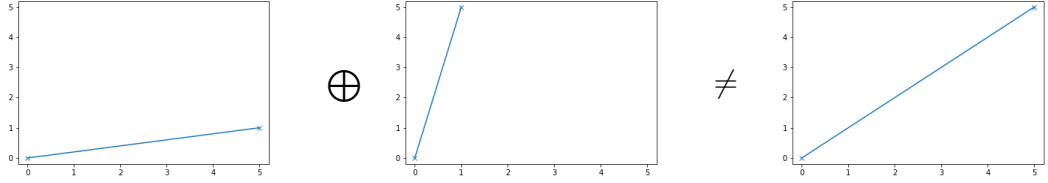
Problem 2: Numerische Instabilität

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



Linken & Mergen: Probleme

Problem 3: Implementierung :'(

Eingabe

- Graph: Deutschland-Instanz von 2006
- $|V| \approx 4,7$ Mio., $|E| \approx 10,8$ Mio.
- 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%	1 219.4	0.00%
Montag	2 377 830	6.18%	1 553.5	27.40%
DiDo	2 305 440	2.94%	1 502.9	23.25%
Freitag	2 340 360	4.50%	1 517.2	24.42%
Samstag	2 329 250	4.01%	1 470.4	20.59%
Sonntag	2 348 470	4.87%	1 464.4	20.09%

Beobachtung:

- Kaum Veränderung in Suchraum
- Anfragen etwas langsamer durch Auswertung

Mehr Instanzen

		Nodes	Arcs [$\cdot 10^6$]	Avg. $\lvert f \rvert$	Rel. Delay [%]		Size
		[$\cdot 10^6$]	(TD [%])	TD arcs	All	TD	[GB]
Ber	Tuesday	0.4	1.0 (27.4)	75.0	3.1	17.6	0.2
	Saturday	0.4	1.0 (20.2)	69.1	2.1	14.8	0.1
Ger06	midweek	4.7	10.8 (7.2)	19.5	1.7	33.1	0.3
	Saturday	4.7	10.8 (3.9)	15.8	0.8	28.5	0.2
SynEur	Low	18.0	42.2 (0.1)	13.2	0.3	125.2	0.8
	Medium	18.0	42.2 (1.0)	13.2	0.8	124.9	0.8
	High	18.0	42.2 (6.2)	13.2	4.6	124.8	1.0
Ger17	Tuesday	7.2	15.8 (29.2)	31.6	3.5	20.8	1.3
Eur17	Tuesday	25.8	55.5 (27.2)	29.5	2.7	19.0	4.2
Eur20	Tuesday	28.5	60.9 (76.3)	22.5	21.0	34.9	8.7

Profilsuchen

Beobachtung:

- Nicht durchführbar auf Europa-Instanz
  - Zu hoher Speicherbedarf ($> 32\,\text{GiB}$ RAM)
- Extrapoliert:
  - Suchraum steigt um ca. 10%
  - Suchzeiten um einen Faktor von bis zu 2 500 über Dijkstra

$\Rightarrow$ impraktikabel

Zusammenfassung

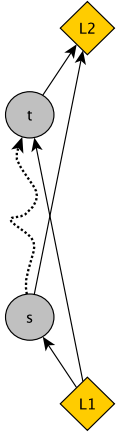
Zeitabhängige Netzwerke (Basics)

- 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

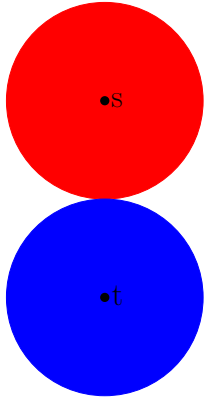
Bausteine

Beschleunigungstechniken

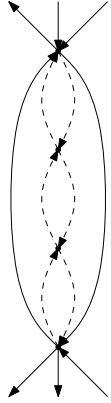
Landmarken



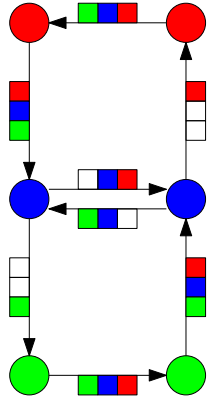
Bidirektionale Suche



Kontraktion



Arc-Flags



Wdh.: ALT

Vorberechnung:

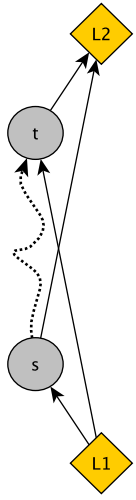
- Wähle einige Knoten (≈ 16) als Landmarken
- Berechne Abstände von und zu allen Landmarken

Anfrage:

- Benutze Landmarken und Dreiecksungleichung, um eine untere Schranke für den Abstand zum Ziel zu bestimmen:

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

- Benutze untere Schranken als A*-Potential
- 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:

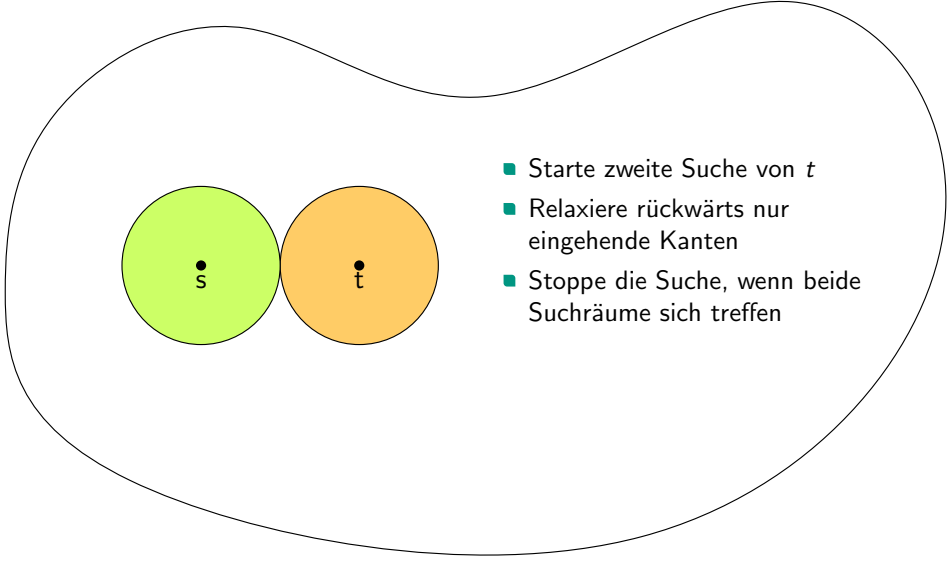
$$\text{len}_\pi(u, v) = \text{len}(u, v) - \pi(u) + \pi(v) \overset{!}{\geq} 0$$

- Durch Erhöhen der Kantengewichte wird dies nicht verletzt

Somit:

- Definiere Lower-Bound-Graph $\underline{G} = (V, E, \underline{\text{len}})$ mit $\underline{\text{len}} := \min \text{len}$
- Vorberechnung auf Lower-Bound-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 zur Einschränkung der Vorwärtssuche
- Je nach Beschleunigungstechnik verschieden $\rightsquigarrow$ später

Anpassung

Zeitanfragen:

- Ankunft unbekannt $\Rightarrow$ Rückwärtsuche?
- Rückwärtssuche nur zur Einschränkung der Vorwärtssuche
- 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 untere Schranke seiner Profilfunktion

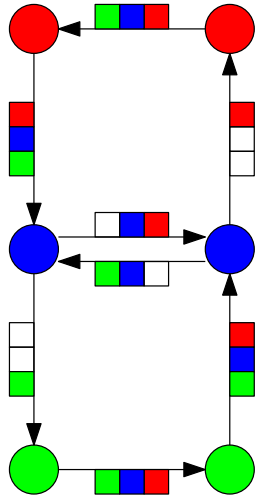
Arc-Flags

Idee:

- Partitioniere den Graph in k Zellen
- Hänge ein Label mit k Bits an jede Kante
- Gibt an, ob e für Zielzelle benötigt wird
- Modifizierter Dijkstra überspringt Kanten ohne Flagge

Beobachtung:

- Partition wird auf ungewichtetem Graphen durchgeführt $\rightarrow$ keine Änderung nötig
- Aber Flaggen müssen anders gesetzt werden



Anpassung

Setze Flagge, wenn...

- Zeitunabhängig: Kante auf kürzestem Pfad in Zielzelle liegt
- Zeitabhängig: Kante mindestens zu einem Zeitpunkt auf kürzestem Pfad in Zielzelle liegt

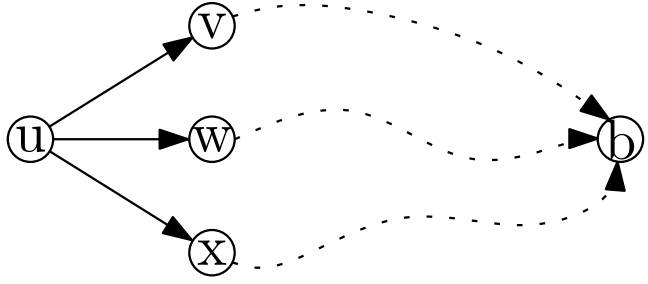
Anpassung

Setze Flagge, wenn...

- Zeitunabhängig: Kante auf kürzestem Pfad in Zielzelle liegt
- Zeitabhängig: Kante mindestens zu einem Zeitpunkt auf kürzestem Pfad in Zielzelle liegt

Anpassung:

- Für jeden Randknoten b und jeden Knoten u:
- Berechne Abstandsfunktion $d_*(u, b)$
- Setze Flagge, wenn gilt: $\text{len}(u, v) \oplus d_*(v, b) \not\succ d_*(u, b)$



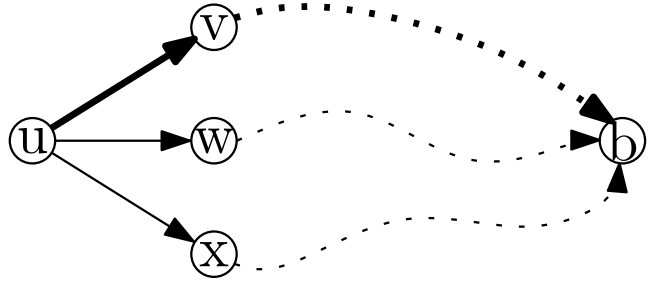
Anpassung

Setze Flagge, wenn...

- Zeitunabhängig: Kante auf kürzestem Pfad in Zielzelle liegt
- Zeitabhängig: Kante mindestens zu einem Zeitpunkt auf kürzestem Pfad in Zielzelle liegt

Anpassung:

- Für jeden Randknoten b und jeden Knoten u:
- Berechne Abstandsfunktion $d_*(u, b)$
- Setze Flagge, wenn gilt: $\text{len}(u, v) \oplus d_*(v, b) \not\succ d_*(u, b)$



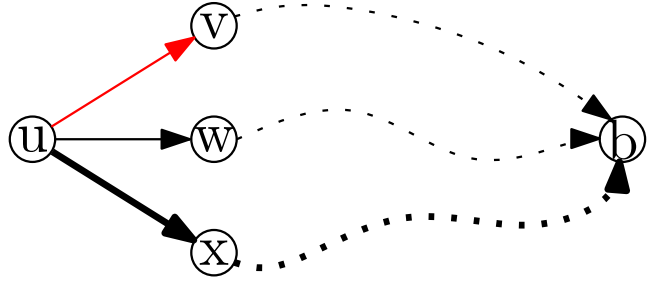
Anpassung

Setze Flagge, wenn...

- Zeitunabhängig: Kante auf kürzestem Pfad in Zielzelle liegt
- Zeitabhängig: Kante mindestens zu einem Zeitpunkt auf kürzestem Pfad in Zielzelle liegt

Anpassung:

- Für jeden Randknoten b und jeden Knoten u:
- Berechne Abstandsfunktion $d_*(u, b)$
- Setze Flagge, wenn gilt: $\text{len}(u, v) \oplus d_*(v, b) \not\succ d_*(u, b)$



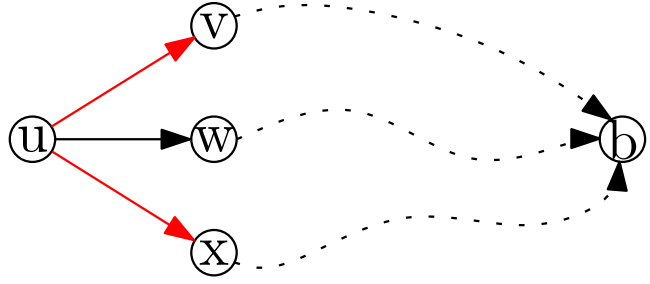
Anpassung

Setze Flagge, wenn...

- Zeitunabhängig: Kante auf kürzestem Pfad in Zielzelle liegt
- Zeitabhängig: Kante mindestens zu einem Zeitpunkt auf kürzestem Pfad in Zielzelle liegt

Anpassung:

- Für jeden Randknoten b und jeden Knoten u:
- Berechne Abstandsfunktion $d_*(u, b)$
- Setze Flagge, wenn gilt: $\text{len}(u, v) \oplus d_*(v, b) \not\succ d_*(u, b)$

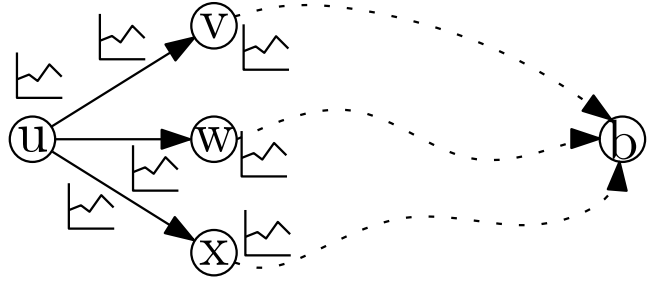


Approximation Arc-Flags

Beobachtung:

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

Idee:



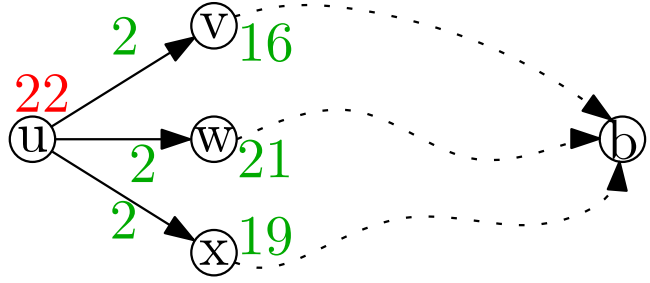
Approximation Arc-Flags

Beobachtung:

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

Idee:

- Benutze Über- and Unterapproximation



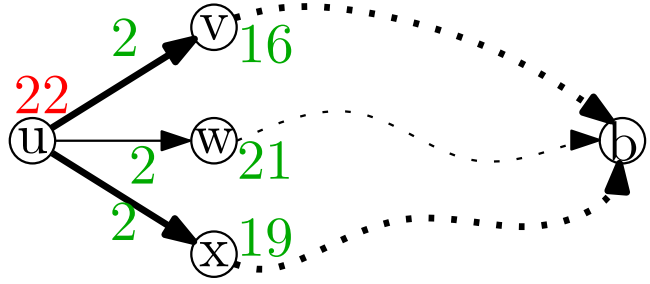
Approximation Arc-Flags

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:

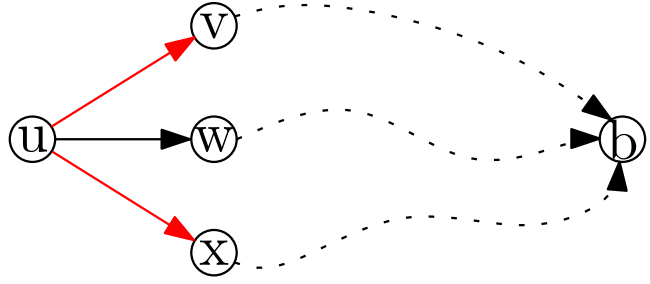
- Viele Interpolationspunkte
- Berechnung der Abstandsfunktionen ist sehr zeitintensiv
- Laufzeit stark abhängig von der Komplexität der Funktionen

Idee:

- Benutze Über- and Unterapproximation

⇒ Schnellere Vorberechnung, langsamere Anfragen

⇒ 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?

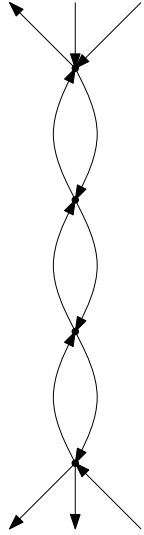
Kontraktion

Knoten-Kontraktion:

- Entferne Knoten iterativ
- Füge neue Kanten (Shortcuts) hinzu, um die Abstände zwischen verbleibenden Knoten zu erhalten

Zeugensuche:

- Behalte nur relevante Shortcuts
- Suche kürzere Pfade (Zeugen) während oder nach Kontraktion



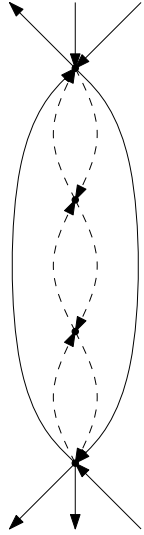
Kontraktion

Knoten-Kontraktion:

- Entferne Knoten iterativ
- Füge neue Kanten (Shortcuts) hinzu, um die Abstände zwischen verbleibenden Knoten zu erhalten

Zeugensuche:

- Behalte nur relevante Shortcuts
- Suche kürzere Pfade (Zeugen) während oder nach Kontraktion



Anpassung Zeugensuche

Zeitunabhängig:

- Lösche Kante (u, v), wenn (u, v) nicht der kürzeste u-v-Pfad ist, also $\text{len}(u, v) > d(u, v)$
- Lokale Dijkstra-Suche von u

Zeitabhängig:

Anpassung Zeugensuche

Zeitunabhängig:

- Lösche Kante (u, v), wenn (u, v) nicht der kürzeste u-v-Pfad ist, also $\text{len}(u, v) > d(u, v)$
- Lokale Dijkstra-Suche von u

Zeitabhängig:

- Lösche Kante (u, v), wenn (u, v) zu keinem Zeitpunkt der kürzeste u-v-Pfad 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
- CH
- SHARC
- MLD (CRP)

Literatur I

- 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
  In: *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.
- Ben Strasser: **Dynamic Time-Dependent Routing in Road Networks Through Sampling**
  In: *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'17)*, 2017.

Literatur II

- Ben Strasser, Dorothea Wagner, Tim Zeitz: **Space-efficient, Fast and Exact Routing in Time-dependent Road Networks**
  In: *Proceedings of the 28th Annual European Symposium on Algorithms (ESA'20)*