

Algorithmen für Routenplanung

13. Vorlesung, Sommersemester 2017

Tobias Zündorf | 26. Juni 2017

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



Stoff bisher

- Problemtransformation auf Graph + Dijkstra
- Vorberechnungstechniken (point-to-point, one-to-all)
- Alternativrouten (fast optimal, hinreichend unterschiedlich)
- Berücksichtigung von historischem Wissen über Staus

Stoff bisher

- Problemtransformation auf Graph + Dijkstra
- Vorberechnungstechniken (point-to-point, one-to-all)
- Alternativrouten (fast optimal, hinreichend unterschiedlich)
- Berücksichtigung von historischem Wissen über Staus

Was fehlt noch für Einsatz in Produktion?

Stoff bisher

- Problemtransformation auf Graph + Dijkstra
- Vorberechnungstechniken (point-to-point, one-to-all)
- Alternativrouten (fast optimal, hinreichend unterschiedlich)
- Berücksichtigung von historischem Wissen über Staus

Was fehlt noch für Einsatz in Produktion?

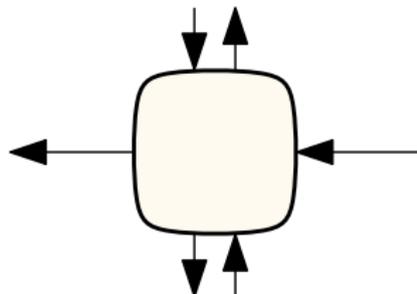
- Turn-Costs
- Andere Optimierungskriterien?
- Was ist eigentlich mit Bus & Bahn?

Abbiegekosten



Bisher:

- Kreuzungen → Knoten
- Straßen → Kanten

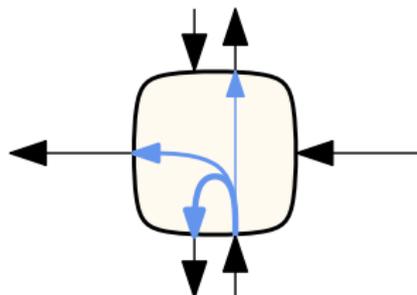


Bisher:

- Kreuzungen → Knoten
- Straßen → Kanten

Aber:

- Abbiegen manchmal verboten
- Linksabbiegen teurer als rechts
- Kosten U-Turns hoch
- wurde als einfaches Modellierungsdetail abgetan

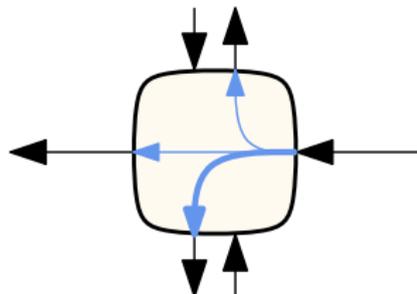


Bisher:

- Kreuzungen → Knoten
- Straßen → Kanten

Aber:

- Abbiegen manchmal verboten
- Linksabbiegen teurer als rechts
- Kosten U-Turns hoch
- wurde als einfaches Modellierungsdetail abgetan

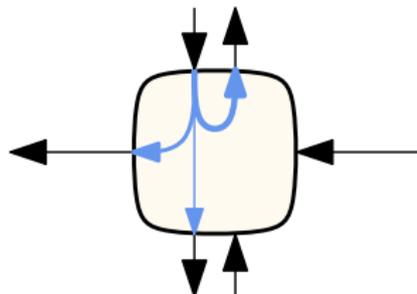


Bisher:

- Kreuzungen → Knoten
- Straßen → Kanten

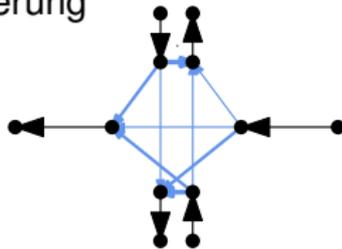
Aber:

- Abbiegen manchmal verboten
- Linksabbiegen teurer als rechts
- Kosten U-Turns hoch
- wurde als einfaches Modellierungsdetail abgetan



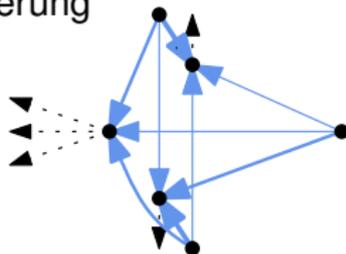
Möglichkeit I:

- Vergrößern des Graphen durch Ausmodellierung
- Kantenbasierter Graph da
 - Straßen → Knoten
 - Turns → Kanten
- Redundante Information



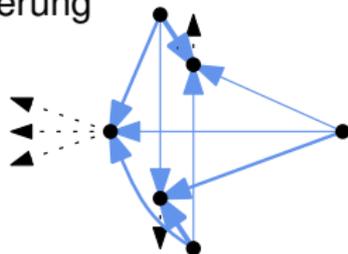
Möglichkeit I:

- Vergrößern des Graphen durch Ausmodellierung
- Kantenbasierter Graph da
 - Straßen → Knoten
 - Turns → Kanten
- Redundante Information
- Entferne einen Knoten pro Straße



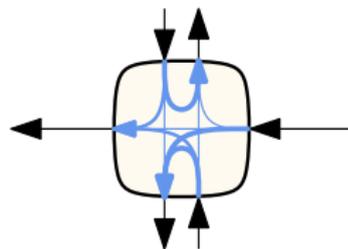
Möglichkeit I:

- Vergrößern des Graphen durch Ausmodellierung
- Kantenbasierter Graph da
 - Straßen \rightarrow Knoten
 - Turns \rightarrow Kanten
- Redundante Information
- Entferne einen Knoten pro Straße



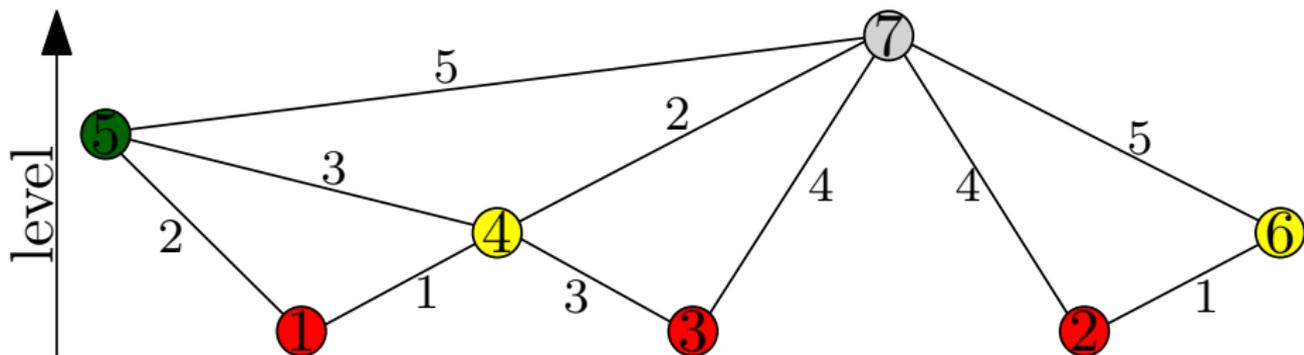
Möglichkeit II:

- Behalte Kreuzungen als Knoten
- Speichere Abbiegetabelle
Abb. Eingangs- \times Ausgangspunkte \rightarrow Kosten
- Beobachtung: Viele Knoten mit identischer Abbiegetabelle
- Speicher jede Tabelle einmal, Knoten speichern Tabellen-ID



Preprocessing:

- Ordne Knoten nach Wichtigkeit
- Bearbeite in der Reihenfolge
- Füge Shortcuts hinzu
- Levelzuordnung (ca. 150 in Straßennetzwerken)

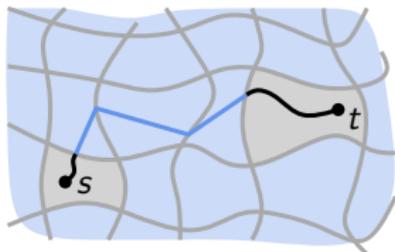
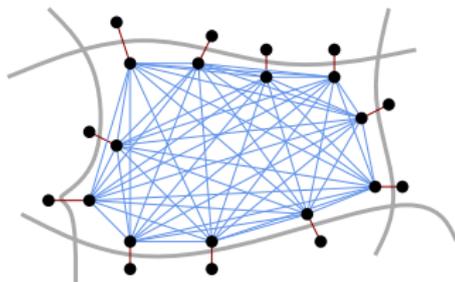


Idee:

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

Overlay Graph:

- Randknoten
- Cliques in jeder Zelle
- Schnittkanten



Suchgraph:

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

Optimierung: multiple Level

Dijkstra:

- Funktioniert ohne Anpassung
- Mehr Knoten zu scannen
- Faktor 3-4 langsamer

Dijkstra:

- Funktioniert ohne Anpassung
- Mehr Knoten zu scannen
- Faktor 3-4 langsamer

CH

- Funktioniert ohne Anpassung
- Aber: grössere Anzahl Knoten/Kanten erhöht Vorberechungszeit

Dijkstra:

- Funktioniert ohne Anpassung
- Mehr Knoten zu scannen
- Faktor 3-4 langsamer

CH

- Funktioniert ohne Anpassung
- Aber: grössere Anzahl Knoten/Kanten erhöht Vorberechungszeit

MLD

- Anzahl Schnittkanten erhöht sich
- Schnittkanten = Schnittknoten
- (Eventuell Wechsel zu Knotenseparatoren sinnvoll?)

Dijkstra:

- Turns müssen in den Suchalgorithmus integriert werden
- Kreuzungen können mehrfach gescannt werden
label-correcting bzgl. Kreuzung, label-setting bzgl. Eingangs-/Ausgangspunkte
- jede **Kante** wird höchstens einmal gescannt
- Suchraum gleich zu kantenbasiertem Modell
simuliert Dijkstra auf kantenbasiertem Graphen
- Vorteil: weniger Speicher für den Graphen

Kompaktes Modell

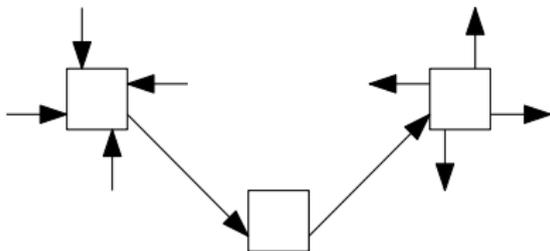
CH

CH

- Zeugensuche wird komplizierter

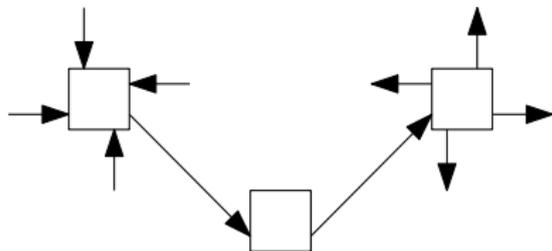
CH

- Zeugensuche wird komplizierter
 - Eine Zeugensuche pro Paar eingehender und ausgehender Kanten



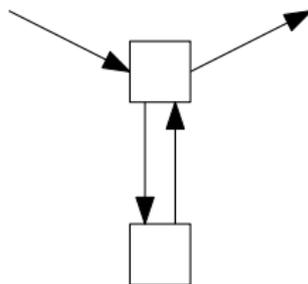
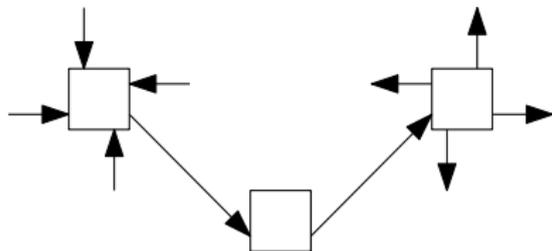
CH

- Zeugensuche wird komplizierter
 - Eine Zeugensuche pro Paar eingehender und ausgehender Kanten
 - Es können Self-Loops entstehen



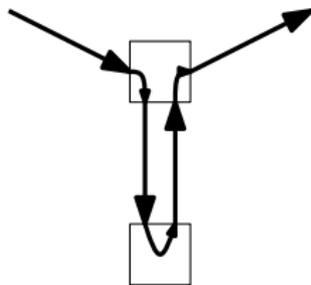
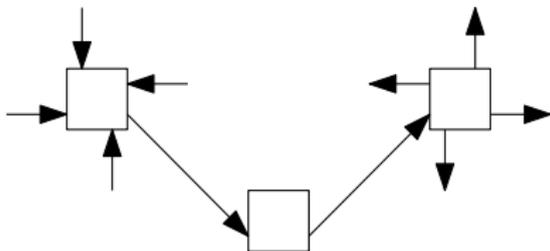
CH

- Zeugensuche wird komplizierter
 - Eine Zeugensuche pro Paar eingehender und ausgehender Kanten
 - Es können Self-Loops entstehen



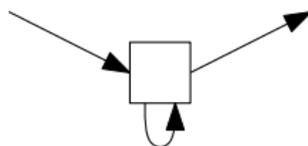
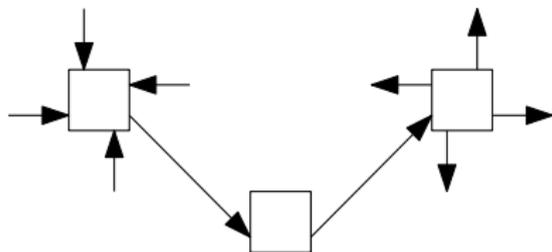
CH

- Zeugensuche wird komplizierter
 - Eine Zeugensuche pro Paar eingehender und ausgehender Kanten
 - Es können Self-Loops entstehen

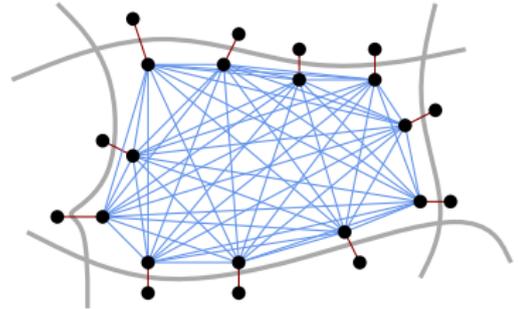


CH

- Zeugensuche wird komplizierter
 - Eine Zeugensuche pro Paar eingehender und ausgehender Kanten
 - Es können Self-Loops entstehen

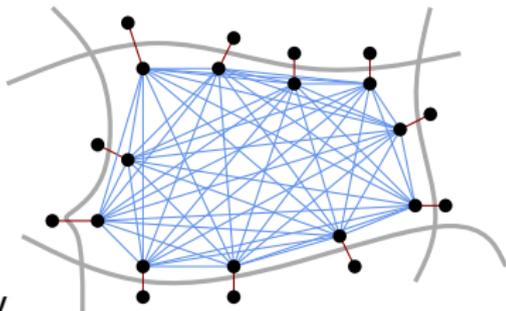


MLD



MLD

- Schnittkanten bleiben erhalten
 - Schnittkante \rightarrow 2 Knoten auf Overlay
 - Turns müssen nur auf unterstem Level beachtet werden
 - Auf Overlaygraphen: normaler Dijkstra
- \Rightarrow einfache Anpassung, aber zusätzliche Fallunterscheidung in der Query



Algorithm		Customization		Queries	
		time [s]	[MB]	#scans	time [ms]
1s	MLD-4 [2^8 : 2^{12} : 2^{16} : 2^{20}]	5.8	61.7	3556	1.18
	CH expanded	3407.4	880.6	550	0.18
	CH compact	849.0	132.5	905	0.19
100s	MLD-4 [2^8 : 2^{12} : 2^{16} : 2^{20}]	7.5	61.7	3813	1.28
	CH expanded	5799.2	931.1	597	0.21
	CH compact	23774.8	304.0	5585	2.11

Beobachtung:

- (Metrikabhängige) CH Ordnung problematisch bei Turns
- Besser: Ordnung an Turns anpassen (CH expanded vs compact)
- MLD robust

Multikriterielle Optimierung



Bisher:

Kürzester Weg

- Eine Metrik

Alternativrouten

- Eine Metrik; fast kürzeste, dennoch sinnvolle Wege

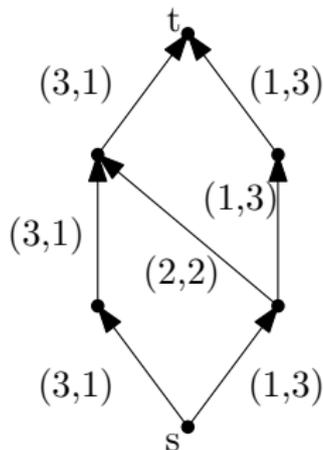
Zeitabhängigkeit / dynamische Szenarien

- Eine Metrik; Kantengewichte ändern sich über die Zeit

Jetzt: Mehrere Metriken

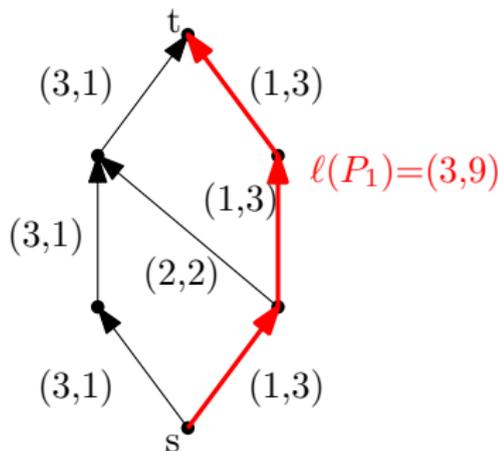
Idee:

- Mehrere Gewichte an Kanten
(Reisezeiten, Kosten, Energieverbrauch)



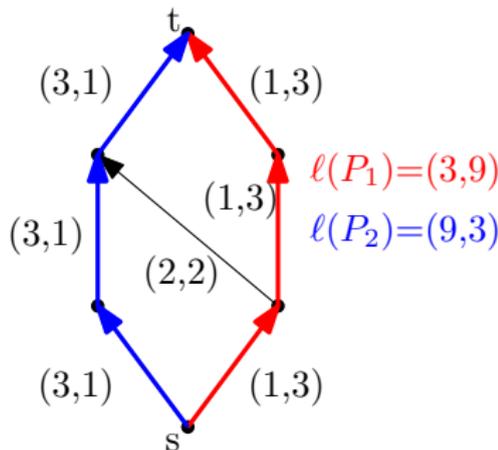
Idee:

- Mehrere Gewichte an Kanten
(Reisezeiten, Kosten, Energieverbrauch)
- Berechne alle **Pareto-optimalen** Routen
 - Route ist Pareto-optimal : \Leftrightarrow
Keine andere Routen dominiert sie
 - Route dominiert andere : \Leftrightarrow
Jedes Kriterium ist gleich oder besser
 - Grenzfall: Gleich gut in allen Kriterien
(wird unterschiedlich behandelt)



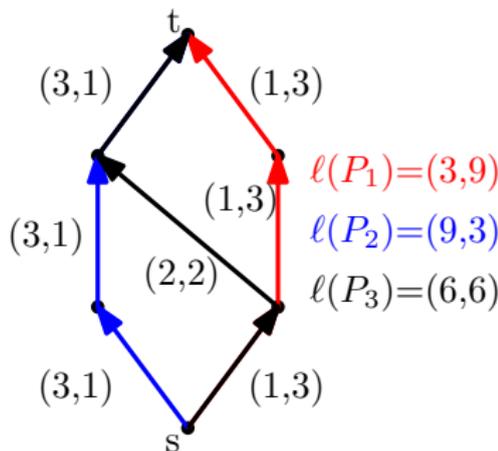
Idee:

- Mehrere Gewichte an Kanten
(Reisezeiten, Kosten, Energieverbrauch)
- Berechne alle **Pareto-optimalen** Routen
 - Route ist Pareto-optimal $:\Leftrightarrow$
Keine andere Routen dominiert sie
 - Route dominiert andere $:\Leftrightarrow$
Jedes Kriterium ist gleich oder besser
 - Grenzfall: Gleich gut in allen Kriterien
(wird unterschiedlich behandelt)



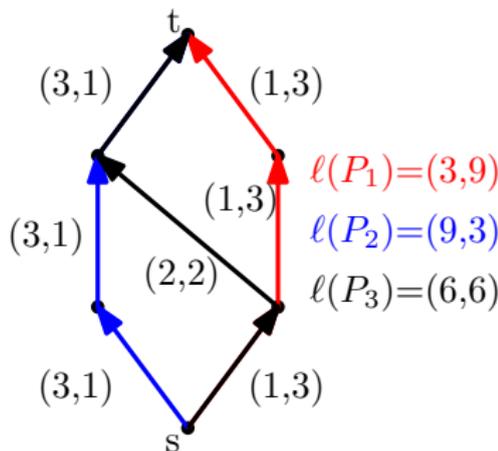
Idee:

- Mehrere Gewichte an Kanten
(Reisezeiten, Kosten, Energieverbrauch)
- Berechne alle **Pareto-optimalen** Routen
 - Route ist Pareto-optimal $:\Leftrightarrow$
Keine andere Routen dominiert sie
 - Route dominiert andere $:\Leftrightarrow$
Jedes Kriterium ist gleich oder besser
 - Grenzfall: Gleich gut in allen Kriterien
(wird unterschiedlich behandelt)



Idee:

- Mehrere Gewichte an Kanten (Reisezeiten, Kosten, Energieverbrauch)
- Berechne alle **Pareto-optimalen** Routen
 - Route ist Pareto-optimal : \Leftrightarrow Keine andere Routen dominiert sie
 - Route dominiert andere : \Leftrightarrow Jedes Kriterium ist gleich oder besser
 - Grenzfall: Gleich gut in allen Kriterien (wird unterschiedlich behandelt)



Herausforderung:

- **Viele** Routen zum Ziel

Definition (Pareto-Dominanz)

Geg. zwei n -Tupel $m_i = (x_1, \dots, x_n)$, $m_j = (y_1, \dots, y_n)$ gilt:
 m_j *domininiert* m_i gdw. m_j in allen Werten besser und in mindestens einem echt besser ist, d. h. $\forall k : y_k \leq x_k$ und $\exists l : y_l < x_l$.

Definition (Pareto-Optimum)

Zu einer Menge M von Tupeln ist ein Tupel $m_i \in M$ *Pareto-Optimum*, wenn es kein anderes $m_j \in M$ gibt, so dass m_i von m_j dominiert wird.

Die Menge M heißt *Pareto-Menge*, wenn alle $m \in M$ Pareto-optimal.

Definition (Pareto-Dominanz)

Geg. zwei n -Tupel $m_i = (x_1, \dots, x_n)$, $m_j = (y_1, \dots, y_n)$ gilt:
 m_j *dominiert* m_i gdw. m_j in allen Werten besser und in mindestens einem echt besser ist, d. h. $\forall k : y_k \leq x_k$ und $\exists l : y_l < x_l$.

Definition (Pareto-Optimum)

Zu einer Menge M von Tupeln ist ein Tupel $m_i \in M$ *Pareto-Optimum*, wenn es kein anderes $m_j \in M$ gibt, so dass m_i von m_j dominiert wird.

Die Menge M heißt *Pareto-Menge*, wenn alle $m \in M$ Pareto-optimal.

Beispiel: Public Transit (Ankunftszeit und # Umstiege)

Definition (Pareto-Dominanz)

Geg. zwei n -Tupel $m_i = (x_1, \dots, x_n)$, $m_j = (y_1, \dots, y_n)$ gilt:
 m_j *dominiert* m_i gdw. m_j in allen Werten besser und in mindestens einem echt besser ist, d. h. $\forall k : y_k \leq x_k$ und $\exists l : y_l < x_l$.

Definition (Pareto-Optimum)

Zu einer Menge M von Tupeln ist ein Tupel $m_i \in M$ *Pareto-Optimum*, wenn es kein anderes $m_j \in M$ gibt, so dass m_i von m_j dominiert wird.

Die Menge M heißt *Pareto-Menge*, wenn alle $m \in M$ Pareto-optimal.

Beispiel: Public Transit (Ankunftszeit und # Umstiege)

$M = \{(14:00 \text{ Uhr}, 5), (15:13 \text{ Uhr}, 3), (13:45 \text{ Uhr}, 4), (15:15 \text{ Uhr}, 0)\}$.

Definition (Pareto-Dominanz)

Geg. zwei n -Tupel $m_i = (x_1, \dots, x_n)$, $m_j = (y_1, \dots, y_n)$ gilt:
 m_j *dominiert* m_i gdw. m_j in allen Werten besser und in mindestens einem echt besser ist, d. h. $\forall k : y_k \leq x_k$ und $\exists l : y_l < x_l$.

Definition (Pareto-Optimum)

Zu einer Menge M von Tupeln ist ein Tupel $m_i \in M$ *Pareto-Optimum*, wenn es kein anderes $m_j \in M$ gibt, so dass m_i von m_j dominiert wird.

Die Menge M heißt *Pareto-Menge*, wenn alle $m \in M$ Pareto-optimal.

Beispiel: Public Transit (Ankunftszeit und # Umstiege)

$M = \{(14:00 \text{ Uhr}, 5), (15:13 \text{ Uhr}, 3), (13:45 \text{ Uhr}, 4), (15:15 \text{ Uhr}, 0)\}$.

Definition (Pareto-Dominanz)

Geg. zwei n -Tupel $m_i = (x_1, \dots, x_n)$, $m_j = (y_1, \dots, y_n)$ gilt:
 m_j *dominiert* m_i gdw. m_j in allen Werten besser und in mindestens einem echt besser ist, d. h. $\forall k : y_k \leq x_k$ und $\exists l : y_l < x_l$.

Definition (Pareto-Optimum)

Zu einer Menge M von Tupeln ist ein Tupel $m_i \in M$ *Pareto-Optimum*, wenn es kein anderes $m_j \in M$ gibt, so dass m_i von m_j dominiert wird.

Die Menge M heißt *Pareto-Menge*, wenn alle $m \in M$ Pareto-optimal.

Beispiel: Public Transit (Ankunftszeit und # Umstiege)

$M = \{(14:00 \text{ Uhr}, 5), (15:13 \text{ Uhr}, 3), (13:45 \text{ Uhr}, 4), (15:15 \text{ Uhr}, 0)\}$.

Wie effizient berechnen?

Idee

- Benutze Graph mit Kantengewicht $len: E \rightarrow \mathbb{R}_{\geq 0}^n$
- Grundlage: Dijkstra's Algorithmus

Idee

- Benutze Graph mit Kantengewicht $len: E \rightarrow \mathbb{R}_{\geq 0}^n$
- Grundlage: Dijkstra's Algorithmus

... aber ...

- Label ℓ sind n -Tupel (x_1, \dots, x_n)
- An jedem Knoten $u \in V$: Pareto-Menge L_u von Labeln
- Jedes Label entspricht einem (Pareto-optimalen) s - u -Pfad
- Priority Queue verwaltet Label statt Knoten
- Prioritätsfunktion $k(x_1, \dots, x_n)$
 - Meist: Linearkombination oder lexikographische Sortierung
- Dominanz von Labeln in L_u on-the-fly

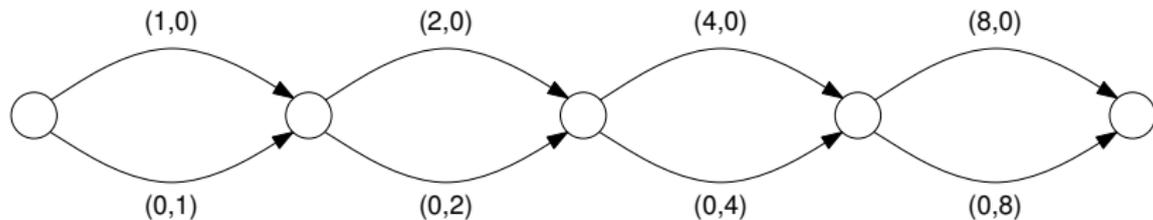
Multi-Criteria Dijkstra (MCD)

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

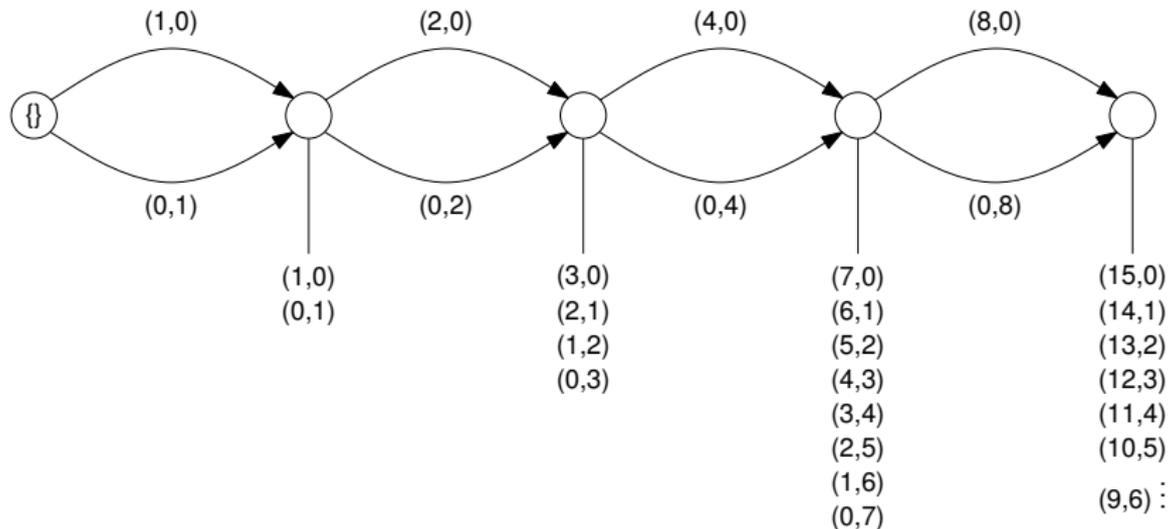
- 1 $L_u \leftarrow \emptyset$ for each $u \in V$; $L_s \leftarrow \{(0, \dots, 0)\}$
 - 2 $Q.clear()$; $Q.insert(s, k(0, \dots, 0))$
 - 3 **while** $!Q.empty()$ **do**
 - 4 u and $\ell = (x_1, \dots, x_n) \leftarrow Q.deleteMin()$
 - 5 **for all edges** $e = (u, v) \in E$ **do**
 - 6 $\ell' \leftarrow (x_1 + \text{len}(e)_1, \dots, x_n + \text{len}(e)_n)$
 - 7 **if** ℓ' is not dominated by any $\ell'' \in L_v$ **then**
 - 8 $L_v.insert(\ell')$
 - 9 Remove non-Pareto-optimal labels from L_v
 - 10 $Q.insert(v, k(\ell'))$
-

- Falls extrahiertes Label immer Pareto-optimal (bzgl. aller $\ell \in Q$):
 - MCD **label-setting** (einmal extrahierte Labels werden nie dominiert)
dafür muss die Längenfunktion natürlich auch positiv sein
 - Gilt für Linearkombination und lexikographische Sortierung
- Pareto-Mengen L_u sind **dynamische** Datenstrukturen \rightsquigarrow teuer!
- Sehr viele Queue-Operationen
- Testen der Dominanz in $\mathcal{O}(|L_u|)$ möglich
- Stoppkriterium?

Exponentiell wachsende Lösungsmenge bei zwei Kriterien:



Exponentiell wachsende Lösungsmenge bei zwei Kriterien:



- Jedes L_u verwaltet bestes ungesetztes Label selbst
⇒ Priority Queue auf Knoten statt Labeln
- **Hopping Reduction:**
Relaxierung der Kante zum Parent-Knoten p unnötig teuer (kann keine Verbesserung bringen, kostet aber $\mathcal{O}(|L_p|)$ für Test)
⇒ Überspringe Kante zum Parent-Knoten von ℓ_u
- **Target-Pruning:**
Abbruchkriterium funktioniert nicht (sonst nur eine Lösung)
⇒ An Knoten u , verwerfe Label ℓ_u , wenn es bereits von der tentativen Pareto-Menge L_t am Ziel t dominiert wird

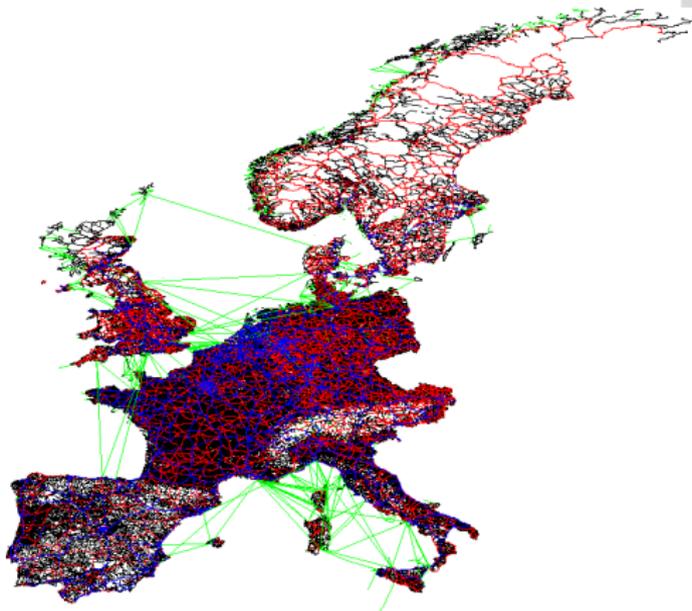
Worst-Case Laufzeit immer noch exponentiell
(aber je nach Instanz schon signifikante Beschleunigung)

Straßengraphen von:

- Luxemburg
- Karlsruhe
- Europa

Metriken:

- Fahrzeiten schnelles Auto
- Fahrzeiten langsames Auto
- Kosten
- Distanzen
- Unit Metrik



metrics	target labels	#del. mins	time [ms]
fast car (fc)	1.0	442 124	156.44
slow car (sc)	1.0	452 635	151.68
fast truck (ft)	1.0	433 834	139.51
slow truck (st)	1.0	440 273	136.85
fc + st	2.2	1 039 110	843.48
fc + ft	2.0	947 042	698.21
fc + sc	1.2	604 750	369.31
sc + lt	1.9	876 998	577.05
sc + ft	1.7	784 459	474.77
ft + st	1.3	632 052	348.43
fc + sc +st	2.3	1 078 190	956.14
fc + sc + ft	2.0	940 815	751.16
sc + ft +st	1.9	880 236	640.47
fc + sc + ft + st	2.5	1 084 780	1016.39

Beobachtungen:

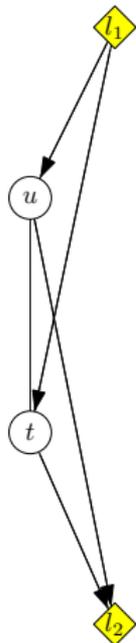
- Nicht viele zusätzliche Lösungen
- Anzahl Lösungen und Queue Extracts korrelieren
- Queryzeit steigt viel stärker
- Dominanztests sind nicht-linear

Verschiedene Metriken

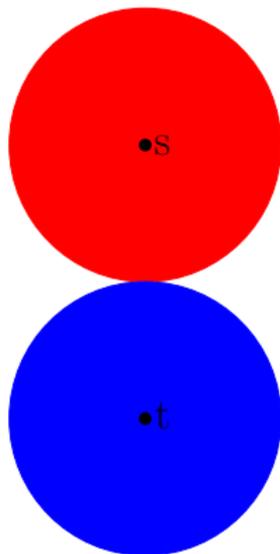
metrics	Luxemburg			Karlsruhe		
	target labels	#del. mins	time [ms]	target labels	#del. mins	time [ms]
fast car (fc)	1.0	15 469	2.89	1.0	39 001	8.2
slow truck (st)	1.0	15 384	2.80	1.0	38 117	7.1
costs	1.0	15 303	2.65	1.0	38 117	6.8
distances	1.0	15 299	2.49	1.0	39 356	7.3
unit	1.0	15 777	2.54	1.0	39 001	8.2
fc + st	2.0	30 026	8.70	1.9	77 778	28.7
fc + costs	29.6	402 232	1704.28	52.7	1 882 930	14909.5
fc + dist.	49.9	429 250	1585.23	99.4	2 475 650	30893.2
fc + unit	25.7	281 894	573.51	27.0	1 030 490	3209.9
costs + dist.	29.6	305 891	581.71	67.2	1 661 600	10815.1

Je nach Kriterien kann Lösungsmenge stark ansteigen

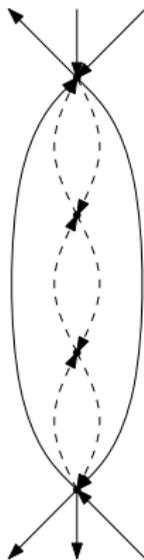
Landmarken



Bidirektionale Suche



Kontraktion



Arc-Flags

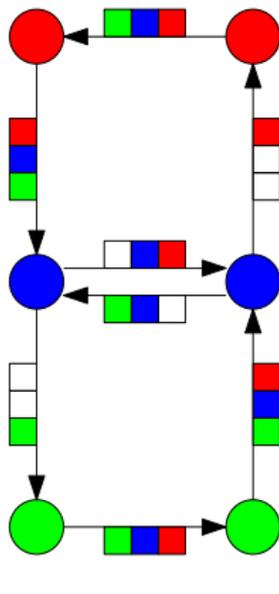
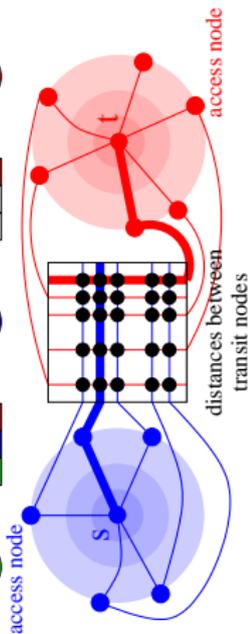


Table-Lookups



Vorbereitung:

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

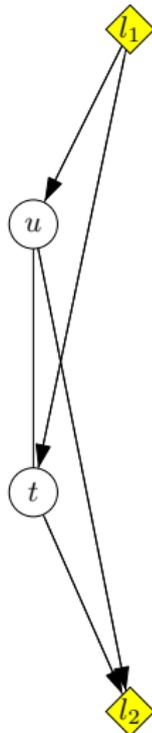
Anfrage:

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

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

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

- Verändert **Reihenfolge** der besuchten Knoten



Idee:

- Benutze einzelne Metriken zum Berechnen der Distanzen
- Für jeden Knoten u :
Distanzvektor $(d_1(u, L_i)_1, \dots, d_n(u, L_i)_n)$ pro Landmarke L_i
- Liefert Potentiale π_1, \dots, π_n
- Zielrichtung: Priorität eines Labels ist $(x_1 + \pi_1, \dots, x_n + \pi_n)$.
- Potential π_j liefert untere Schranke für $d_j(u, t)$
⇒ Nutze $(x_1 + \pi_1, \dots, x_n + \pi_n)$ auch für **Target Pruning**

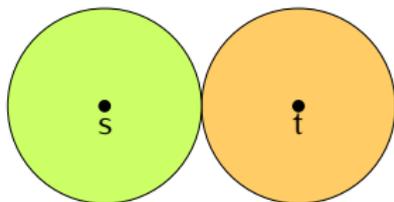
Idee:

- Benutze einzelne Metriken zum Berechnen der Distanzen
- Für jeden Knoten u :
Distanzvektor $(d_1(u, L_i)_1, \dots, d_n(u, L_i)_n)$ pro Landmarke L_i
- Liefert Potentiale π_1, \dots, π_n
- Zielrichtung: Priorität eines Labels ist $(x_1 + \pi_1, \dots, x_n + \pi_n)$.
- Potential π_i liefert untere Schranke für $d_i(u, t)$
 \Rightarrow Nutze $(x_1 + \pi_1, \dots, x_n + \pi_n)$ auch für **Target Pruning**

Modifikation:

Berechne zur Queryzeit $(d_1(u, t)_1, \dots, d_n(u, t)_n)$

- Nutze t als einzige “perfekte” Landmarke
- Kosten von n Dijkstras (meist) unerheblich für Gesamtlaufzeit
- Keine Vorberechnung



- Starte zweite Suche von t
- Relaxiere rückwärts nur eingehende Kanten
- Stoppe die Suche, wenn beide Suchräume sich treffen

Idee:

- Rückwärtssuche kein Problem (analog)

Idee:

- Rückwärtssuche kein Problem (analog)

Offenes Problem:

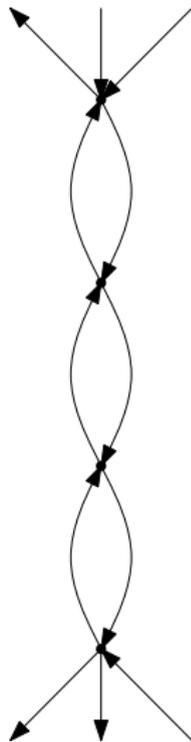
- Abbruchkriterium?
- Analog Target-Pruning: Dominanztest mit tentativer Pareto-Menge; teuer zu verwalten: Kombination der Lösungen aus Vorwärts- und Rückwärtssuche an Mittelknoten
- Lohnt nicht recht

Knoten-Reduktion:

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

Kanten-Reduktion:

- Behalte nur relevante Shortcuts
- Lokale Suche während oder nach Knoten-reduktion

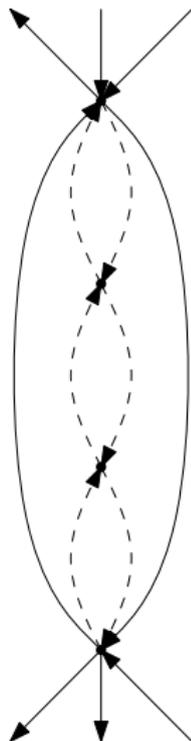


Knoten-Reduktion:

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

Kanten-Reduktion:

- Behalte nur relevante Shortcuts
- Lokale Suche während oder nach Knoten-reduktion

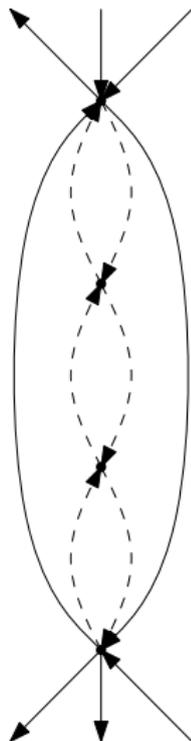


Beobachtung:

- Verfahren unabhängig von Metrik
- Shortcut muss dem (entfernten) Pfad entsprechen

Somit:

- Anpassung ohne Probleme



Unikriteriell:

- Lösche Kante (u, v) , 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

Multikriteriell:

Unikriteriell:

- Lösche Kante (u, v) , 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

Multikriteriell:

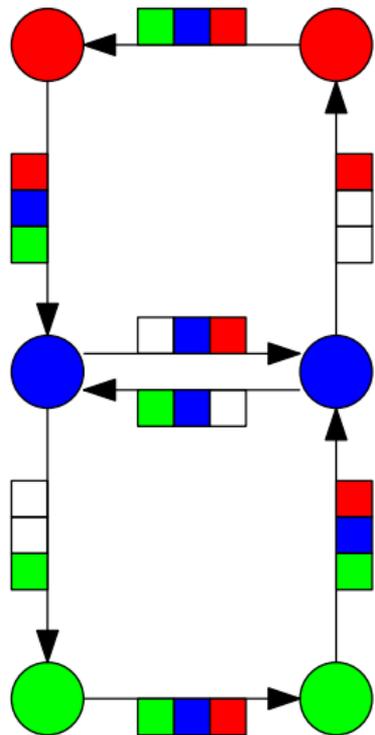
- Lösche Kante (u, v) , wenn (u, v) nicht Teil eines Pareto-Weges von u nach v ist
- Lokale multi-kriterielle Suche
- Kann zu (Pareto-optimalen) Multikanten führen
- Problem: "Explosion" der Anzahl der Routen

Idee:

- Partitioniere den Graph in k Zellen
- Hänge ein Label mit k Bits an jede Kante
- Zeigt ob e wichtig für die Zielzelle ist
- Modifizierter Dijkstra überspringt unwichtige Kanten

Beobachtung:

- Partition wird auf ungewichtetem Graphen durchgeführt
- Flaggen müssen allerdings aktualisiert werden



Idee:

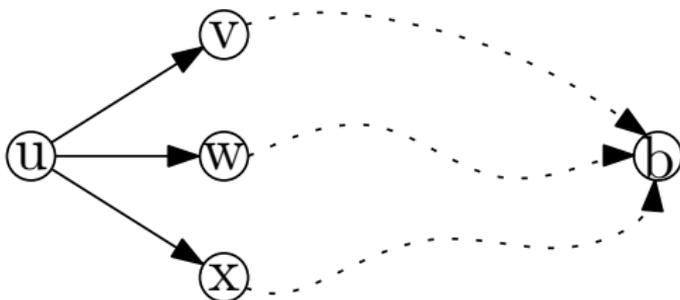
- Ändere **Intuition** einer gesetzten Flagge
- Konzept **bleibt gleich**: Eine Flagge pro Kante und Region
- Setze Flagge
 - **Multikriteriell**: wenn Kante für einen Pareto-Pfad “wichtig” ist

Idee:

- Ändere **Intuition** einer gesetzten Flagge
- Konzept **bleibt gleich**: Eine Flagge pro Kante und Region
- Setze Flagge
 - **Multikriteriell**: wenn Kante für einen Pareto-Pfad "wichtig" ist

Anpassung:

- Für alle Randknoten b und alle Knoten u :
- Berechne Pareto-Abstände $D(u, b)$
- Setze Flagge wenn gilt (u, v) zugehörige Kante eines Pareto-Pfades

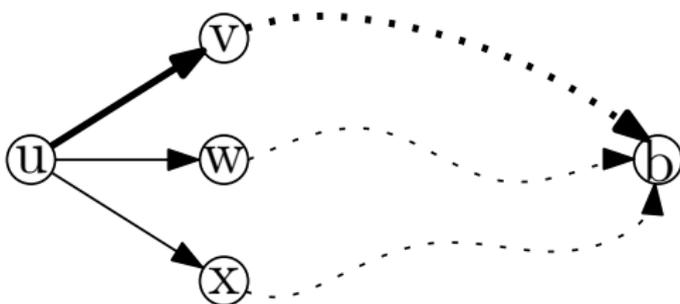


Idee:

- Ändere **Intuition** einer gesetzten Flagge
- Konzept **bleibt gleich**: Eine Flagge pro Kante und Region
- Setze Flagge
 - **Multikriteriell**: wenn Kante für einen Pareto-Pfad "wichtig" ist

Anpassung:

- Für alle Randknoten b und alle Knoten u :
- Berechne Pareto-Abstände $D(u, b)$
- Setze Flagge wenn gilt (u, v) zugehörige Kante eines Pareto-Pfades

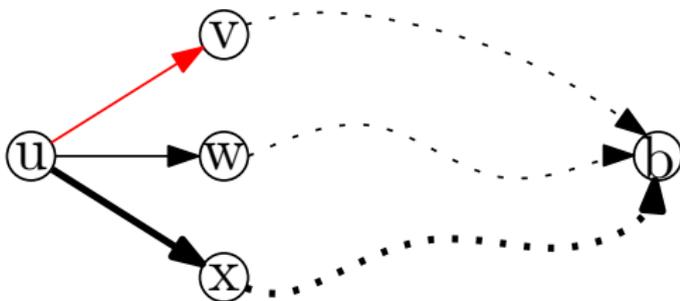


Idee:

- Ändere **Intuition** einer gesetzten Flagge
- Konzept **bleibt gleich**: Eine Flagge pro Kante und Region
- Setze Flagge
 - **Multikriteriell**: wenn Kante für einen Pareto-Pfad "wichtig" ist

Anpassung:

- Für alle Randknoten b und alle Knoten u :
- Berechne Pareto-Abstände $D(u, b)$
- Setze Flagge wenn gilt (u, v) zugehörige Kante eines Pareto-Pfades

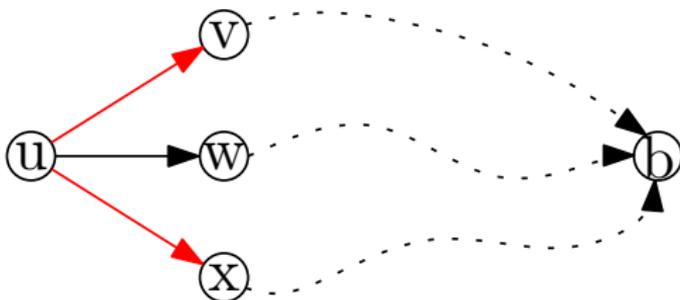


Idee:

- Ändere **Intuition** einer gesetzten Flagge
- Konzept **bleibt gleich**: Eine Flagge pro Kante und Region
- Setze Flagge
 - **Multikriteriell**: wenn Kante für einen Pareto-Pfad "wichtig" ist

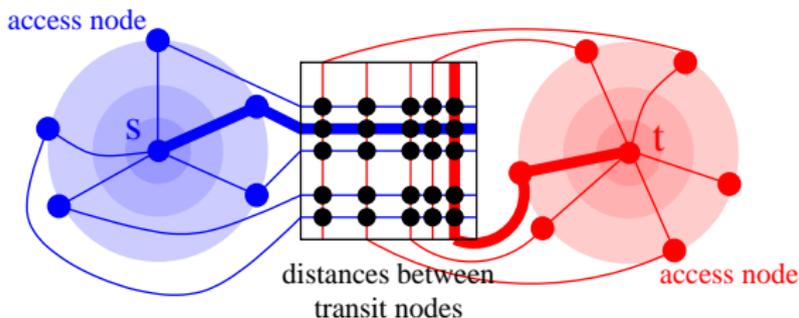
Anpassung:

- Für alle Randknoten b und alle Knoten u :
- Berechne Pareto-Abstände $D(u, b)$
- Setze Flagge wenn gilt (u, v) zugehörige Kante eines Pareto-Pfades



Idee:

- Speichere Distanztabellen
- Nur für “wichtige” Teile des Graphen
- Suchen laufen nur bis zur Tabelle
- Harmoniert gut mit hierarchischen Techniken



Beobachtung:

- Distanz-Tabelle muss Pareto-Abstände abspeichern
- Massiver Anstieg der Größe der Tabellen
- Pfadstruktur nicht mehr so gutmütig
- Deutlich mehr Access-Nodes?

Also:

- Speicherverbrauch deutlich zu groß?

Basismodule:

- ? Bidirektionale Suche
- + Landmarken / A*
- + Kontraktion
- + arc-flags
- ? Table Look-ups

Pareto-SHARC (nur als Beispiel)

metrics	Luxemburg					Karlsruhe				
	PREPRO time [h:m]	target labels	#del. mins	time [ms]	spd up	PREPRO time [h:m]	target labels	#del. mins	time [ms]	spd up
fast car (fc)	< 0:01	1.0	138	0.03	114	< 0:01	1.0	206	0.04	188
slow truck (st)	< 0:01	1.0	142	0.03	111	< 0:01	1.0	212	0.04	178
costs	< 0:01	1.0	151	0.03	96	< 0:01	1.0	244	0.05	129
distances	< 0:01	1.0	158	0.03	87	< 0:01	1.0	261	0.06	119
unit	< 0:01	1.0	149	0.03	96	< 0:01	1.0	238	0.05	147
fc + st	0:01	2.0	285	0.09	100	0:01	1.9	797	0.26	108
fc + costs	0:04	29.6	4 149	6.49	263	1:30	52.7	15 912	80.88	184
fc + dist.	0:14	49.9	8 348	20.21	78	3:58	99.4	31 279	202.15	153
fc + unit	0:06	25.7	4 923	5.13	112	0:17	27.0	11 319	16.04	200
costs + dist.	0:02	29.6	3 947	4.87	119	1:11	67.2	19 775	67.75	160

- Berechnung der (exponentiell großen) Pareto-Menge nicht effizient möglich
- Auch mit Beschleunigungstechniken daher exponentielle Laufzeit
- Laufzeit in der Praxis stark abhängig von
 - Anzahl der Kriterien
 - Korrelation der Metriken
- Praktikable Laufzeit somit oft nur mit Heuristiken möglich
 - Relaxierung der Dominanz
 - Ausdünnen von Pareto-Mengen während der Query
 - Mehr dazu später...
- Nur konvexe Hülle (\Rightarrow linear Kombination, Parametric Shortest Path Problem)

Constrained Shortest Paths



Ziel:

- Finde *kürzeste* Route die bestimmtes *Gewicht* nicht überschreitet
- Zwei Metriken auf den Kanten: *Länge* und *Gewicht*
- Optimierte die Länge und beschränke das Gewicht

Ziel:

- Finde *kürzeste* Route die bestimmtes *Gewicht* nicht überschreitet
- Zwei Metriken auf den Kanten: *Länge* und *Gewicht*
- Optimierte die Länge und beschränke das Gewicht

Definition: Constrained Shortest Path Problem

Gegeben: $G = (V, E)$, Länge $\ell: E \rightarrow \mathbb{N}_0$, Gewicht $\omega: E \rightarrow \mathbb{N}_0$, Start und Ziel $s, t \in V$ sowie Schranken $L, W \in \mathbb{N}_0$

Problem: Existiert ein einfacher Pfad P von s nach t in G , für den $\ell(P) \leq L$ und $\omega(P) \leq W$ gelten?

Anmerkung: Das entsprechende Optimierungsproblem lautet:

- Finde einen s - t -Pfad P mit minimalem $\ell(P)$ und $\omega(P) \leq W$

Theorem

Constrained Shortest Path Problem ist (schwach) \mathcal{NP} -vollständig

Theorem

Constrained Shortest Path Problem ist (schwach) \mathcal{NP} -vollständig

Beweis:

1: CSP Problem $\in \mathcal{NP}$

- Für ein Pfad P kann in polynomieller Zeit geprüft werden, ob:
 - P benutzt nur Kanten aus G
 - P hat passende Länge: $\ell(P) = L$
 - P hat passendes Gewicht: $\omega(P) = W$

2: CSP Problem ist \mathcal{NP} -schwer

- Beweis durch Reduktion von PARTITION

Definition: PARTITION

Gegeben: Endliche Menge A sowie Größe $s: A \rightarrow \mathbb{N}_0$

Problem: Existiert ein Teilmenge $A' \subseteq A$ für die gilt:

$$\sum_{a \in A'} s(a) = \sum_{a \in A \setminus A'} s(a)$$

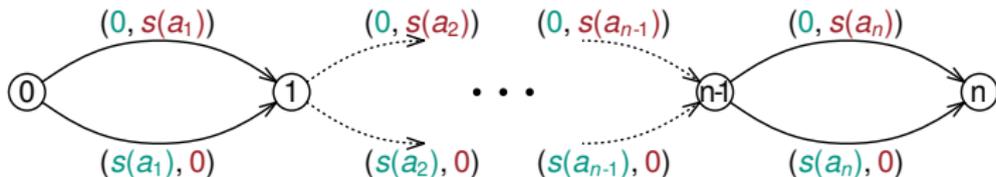
Anmerkung:

- In Karps 21 NP-vollständigen Problemen enthalten
- PARTITION ist (schwach) \mathcal{NP} -vollständig [Karp '71]
 - \mathcal{NP} -schwere Beweis benötigt exponentiell große Zahlen
 - In pseudopolynomieller Zeit lösbar (Dynamische Programmierung)
 - Pseudopolynomielle Laufzeit: Abhängig von Eingabegröße + Werten

Beweis: CSP ist \mathcal{NP} -vollständig

Reduktion von PARTITION:

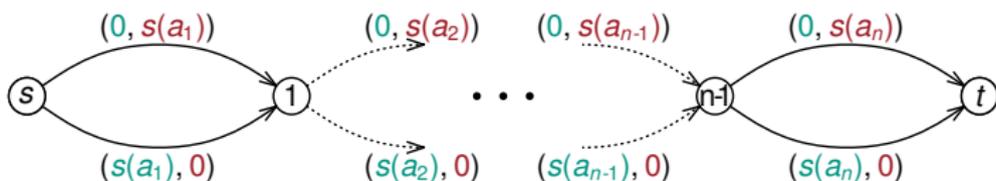
- Sei $\Pi = (A = \{a_1, \dots, a_n\}, s)$ eine PARTITION-Instanz
- Konstruiere Graph $G = (V, E)$ mit $n + 1$ Knoten ($V = \{0, 1, \dots, n\}$)
- Knoten $i-1$ und i sind jeweils durch zwei Kanten verbunden
 - Eine Kante hat Länge 0 und Gewicht $s(a_i)$
 - Die andere Kante hat Länge $s(a_i)$ und Gewicht 0
- Setze $s = 0$, $t = n$ und $L = W = \frac{1}{2} \sum_{a \in A} s(a)$



Beweis: CSP ist \mathcal{NP} -vollständig

Reduktion von PARTITION:

- Sei $\Pi = (A = \{a_1, \dots, a_n\}, s)$ eine PARTITION-Instanz
- Konstruiere Graph $G = (V, E)$ mit $n + 1$ Knoten ($V = \{0, 1, \dots, n\}$)
- Knoten $i-1$ und i sind jeweils durch zwei Kanten verbunden
 - Eine Kante hat Länge 0 und Gewicht $s(a_i)$
 - Die andere Kante hat Länge $s(a_i)$ und Gewicht 0
- Setze $s = 0$, $t = n$ und $L = W = \frac{1}{2} \sum_{a \in A} s(a)$



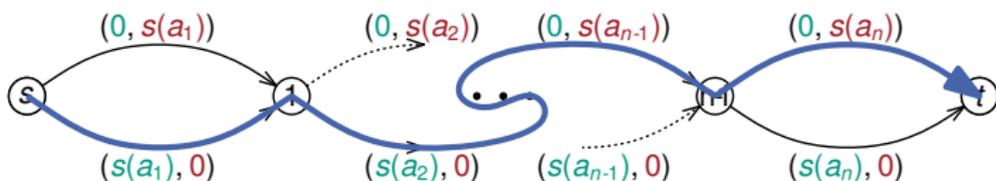
Es gilt:

- Π ist PARTITION Ja-Inst. $\Leftrightarrow (G, \ell, \omega, L, W, s, t)$ ist CSP Ja-Inst. □

Beweis: CSP ist \mathcal{NP} -vollständig

Reduktion von PARTITION:

- Sei $\Pi = (A = \{a_1, \dots, a_n\}, s)$ eine PARTITION-Instanz
- Konstruiere Graph $G = (V, E)$ mit $n + 1$ Knoten ($V = \{0, 1, \dots, n\}$)
- Knoten $i-1$ und i sind jeweils durch zwei Kanten verbunden
 - Eine Kante hat Länge 0 und Gewicht $s(a_i)$
 - Die andere Kante hat Länge $s(a_i)$ und Gewicht 0
- Setze $s = 0$, $t = n$ und $L = W = \frac{1}{2} \sum_{a \in A} s(a)$



Beispiel:

- s - t -Pfad P entspricht: $a_1 \notin A', a_2 \notin A', \dots, a_{n-1} \in A', a_n \in A'$



Multi-Criteria Dijkstra für CSP

CSP kann mit Multi-Criteria Dijkstra (MCD) gelöst werden

- Bikriterieller Ansatz mit Metriken: Länge & Gewicht
- Nutze Constraints zum prunen
 - Verwerfe Label mit Gewicht $> W$

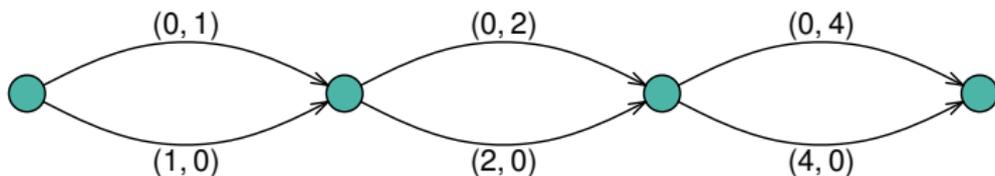
Multi-Criteria Dijkstra für CSP

CSP kann mit Multi-Criteria Dijkstra (MCD) gelöst werden

- Bikriterieller Ansatz mit Metriken: Länge & Gewicht
- Nutze Constraints zum prunen
 - Verwerfe Label mit Gewicht $> W$

Erinnerung:

- MCD hält pro Knoten Menge Pareto-optimaler Pfade
- Pareto-Mengen können exponentiell groß werden



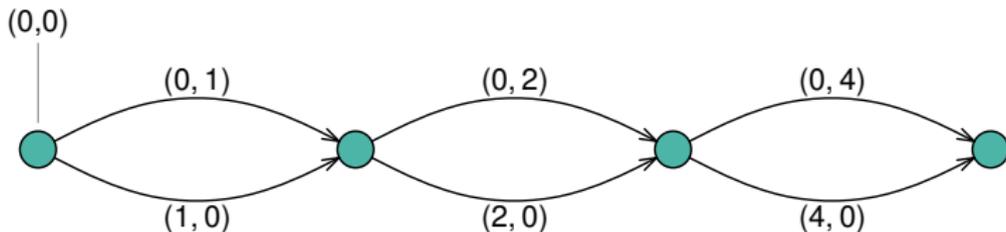
Multi-Criteria Dijkstra für CSP

CSP kann mit Multi-Criteria Dijkstra (MCD) gelöst werden

- Bikriterieller Ansatz mit Metriken: Länge & Gewicht
- Nutze Constraints zum prunen
 - Verwerfe Label mit Gewicht $> W$

Erinnerung:

- MCD hält pro Knoten Menge Pareto-optimaler Pfade
- Pareto-Mengen können exponentiell groß werden

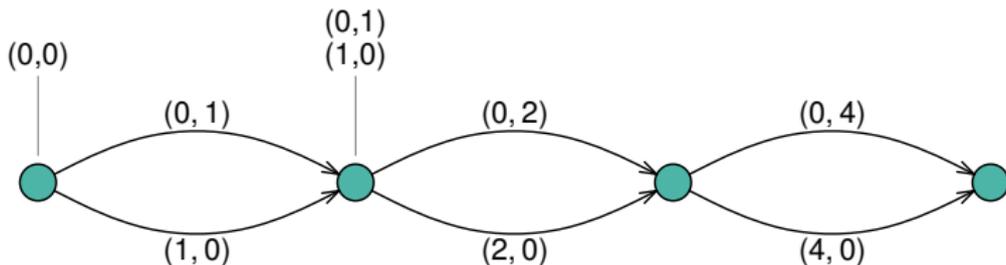


CSP kann mit Multi-Criteria Dijkstra (MCD) gelöst werden

- Bikriterieller Ansatz mit Metriken: Länge & Gewicht
- Nutze Constraints zum prunen
 - Verwerfe Label mit Gewicht $> W$

Erinnerung:

- MCD hält pro Knoten Menge Pareto-optimaler Pfade
- Pareto-Mengen können exponentiell groß werden



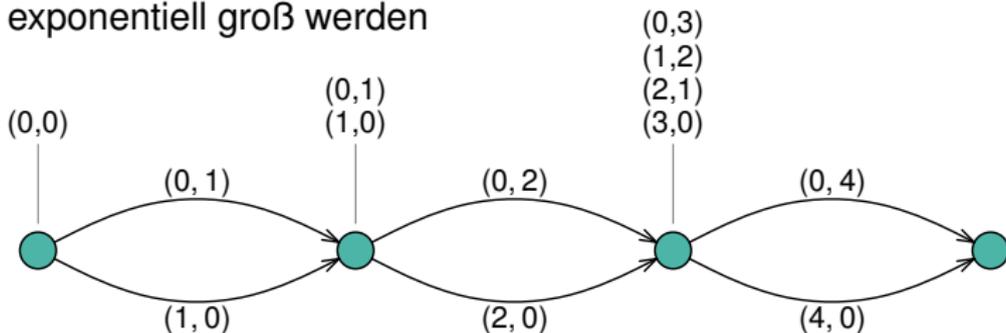
Multi-Criteria Dijkstra für CSP

CSP kann mit Multi-Criteria Dijkstra (MCD) gelöst werden

- Bikriterieller Ansatz mit Metriken: Länge & Gewicht
- Nutze Constraints zum prunen
 - Verwerfe Label mit Gewicht $> W$

Erinnerung:

- MCD hält pro Knoten Menge Pareto-optimaler Pfade
- Pareto-Mengen können exponentiell groß werden



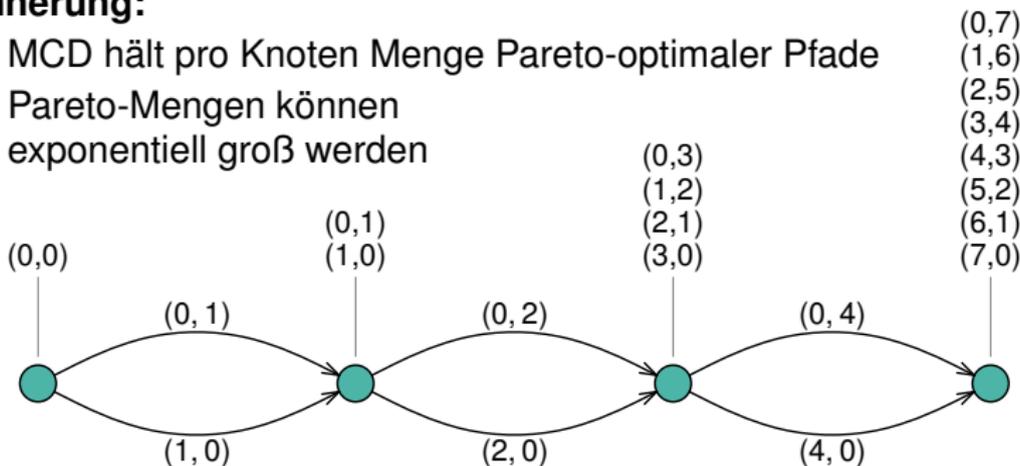
Multi-Criteria Dijkstra für CSP

CSP kann mit Multi-Criteria Dijkstra (MCD) gelöst werden

- Bikriterieller Ansatz mit Metriken: Länge & Gewicht
- Nutze Constraints zum prunen
 - Verwerfe Label mit Gewicht $> W$

Erinnerung:

- MCD hält pro Knoten Menge Pareto-optimaler Pfade
- Pareto-Mengen können exponentiell groß werden



Verbesserungen: Standard Beschleunigungen von MCD übertragbar:

- Hopping Reduction
- Nur ein Label pro Knoten in Queue
- Target Pruning

Verbesserungen: Standard Beschleunigungen von MCD übertragbar:

- Hopping Reduction
- Nur ein Label pro Knoten in Queue
- Target Pruning

Beobachtung: Wir brauchen nicht alle Pareto-Optima an t :

- Sind nur an kürzester zulässiger Route interessiert
- Stoppe sobald erstes Label an t aus Queue genommen
(Queue ist nach Länge sortiert)

- Dynamische Szenarien
 - Berücksichtigung von aktuellen Staus
 - Anpassung von Beschleunigungstechniken
- Turn-Costs
 - Expandiertes Modell vs Turn-Tabellen
 - Anpassung von CRP/CH
- Multikriterielle Optimierung
 - Verallgemeinerung zu Multi Criteria Dijkstra
 - Anpassung von Beschleunigungstechniken
- Constrained Shortest Paths
 - \mathcal{NP} -schwer
 - Anpassung MC-Dijkstra

Literatur:

- Robert Geisberger, Christian Vetter
Efficient Routing in Road Networks with Turn Costs
In: *Proceedings of the 10th International Symposium on Experimental Algorithms (SEA'11)*, pages 100-111, 2011.
- Daniel Delling and Andrew V. Goldberg and Thomas Pajor and Renato F. Werneck
Customizable Route Planning in Road Networks
Transportation Science, 2015.
- Daniel Delling, Dorothea Wagner
Pareto Paths with SHARC
In: *Proceedings of the 8th International Symposium on Experimental Algorithms (SEA'09)*, pages 125-136, 2009.