Karlsruher Institut für Technologie

Algorithmen für Routenplanung

10. Vorlesung, Sommersemester 2024

Moritz Laupichler | 29. Mai 2024

Optimierungskriterien

Bisher:

Kürzester Weg

- Eine Metrik

Alternativrouten

- Eine Metrik; nicht kürzeste, dennoch sinnvolle Wege

Optimierungskriterien

Bisher:

Kürzester Weg

- Eine Metrik

Alternativrouten

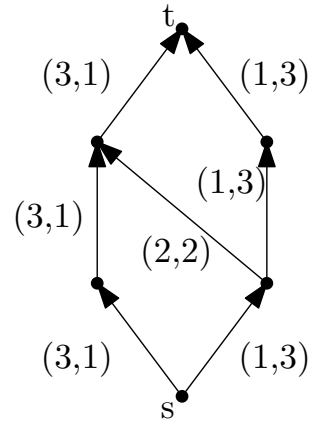
- Eine Metrik; nicht kürzeste, dennoch sinnvolle Wege

Jetzt: Mehrere Metriken

Multikriterielle Optimierung

Szenario:

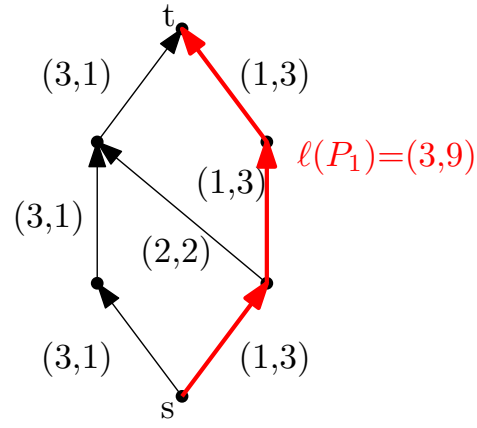
- Mehrere Gewichte an Kanten
  (z.B. Reisezeit, Kosten, Energieverbrauch)



Multikriterielle Optimierung

Szenario:

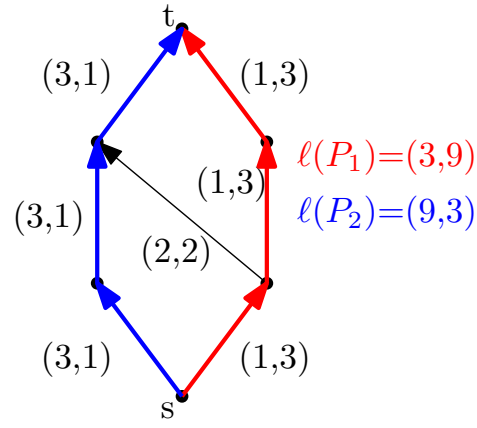
- Mehrere Gewichte an Kanten
  (z.B. Reisezeit, Kosten, Energieverbrauch)
- Berechne alle Pareto-optimalen Pfade
  - Pfad ist Pareto-optimal :⇔
    Kein anderer Pfad dominiert ihn
  - Pfad dominiert anderen :⇔
    Jedes Kriterium ist gleich oder besser
  - Grenzfall: Gleich gut in allen Kriterien
    (wird unterschiedlich behandelt)



Multikriterielle Optimierung

Szenario:

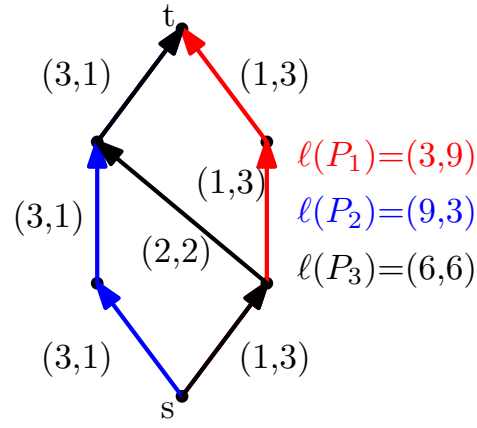
- Mehrere Gewichte an Kanten
  (z.B. Reisezeit, Kosten, Energieverbrauch)
- Berechne alle Pareto-optimalen Pfade
  - Pfad ist Pareto-optimal :⇔
    Kein anderer Pfad dominiert ihn
  - Pfad dominiert anderen :⇔
    Jedes Kriterium ist gleich oder besser
  - Grenzfall: Gleich gut in allen Kriterien
    (wird unterschiedlich behandelt)



Multikriterielle Optimierung

Szenario:

- Mehrere Gewichte an Kanten
  (z.B. Reisezeit, Kosten, Energieverbrauch)
- Berechne alle Pareto-optimalen Pfade
  - Pfad ist Pareto-optimal :⇔
    Kein anderer Pfad dominiert ihn
  - Pfad dominiert anderen :⇔
    Jedes Kriterium ist gleich oder besser
  - Grenzfall: Gleich gut in allen Kriterien
    (wird unterschiedlich behandelt)



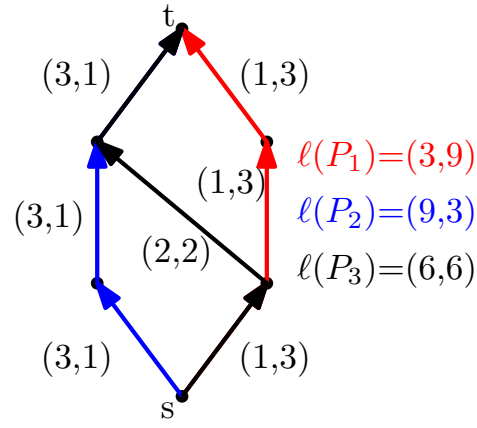
Multikriterielle Optimierung

Szenario:

- Mehrere Gewichte an Kanten
  (z.B. Reisezeit, Kosten, Energieverbrauch)
- Berechne alle Pareto-optimalen Pfade
  - Pfad ist Pareto-optimal :⇔
    Kein anderer Pfad dominiert ihn
  - Pfad dominiert anderen :⇔
    Jedes Kriterium ist gleich oder besser
  - Grenzfall: Gleich gut in allen Kriterien
    (wird unterschiedlich behandelt)

Herausforderung:

- Viele Pfade zum Ziel



Pareto-Menge

Definition (Pareto-Dominanz)

Gegeben zwei n-Tupel $m_i = (x_1, \ldots, x_n)$ und $m_j = (y_1, \ldots, y_n)$:

$$m_i \text{ dominiert } m_j \iff \forall k : x_k \leq y_k \text{ und } \exists \ell : x_\ell < y_\ell$$

(m_i ist in keinem Wert schlechter als m_j und in mindestens einem besser)

Pareto-Menge

Definition (Pareto-Dominanz)

Gegeben zwei n-Tupel $m_i = (x_1, \ldots, x_n)$ und $m_j = (y_1, \ldots, y_n)$:

$$m_i \text{ dominiert } m_j \iff \forall k : x_k \leq y_k \text{ und } \exists \ell : x_\ell < y_\ell$$

(m_i ist in keinem Wert schlechter als m_j und in mindestens einem besser)

Definition (Pareto-Optimum)

Gegeben eine Menge M von n-Tupeln: Tupel $m_i \in M$ heißt Pareto-optimal, wenn es kein anderes $m_j \in M$ gibt, das m_i dominiert.

Menge $M' \subseteq M$ heißt Pareto-Menge, wenn alle $m \in M'$ Pareto-optimal.

Pareto-Menge

Definition (Pareto-Dominanz)

Gegeben zwei n-Tupel $m_i = (x_1, \ldots, x_n)$ und $m_j = (y_1, \ldots, y_n)$:

$$m_i \text{ dominiert } m_j \iff \forall k : x_k \leq y_k \text{ und } \exists \ell : x_\ell < y_\ell$$

(m_i ist in keinem Wert schlechter als m_j und in mindestens einem besser)

Definition (Pareto-Optimum)

Gegeben eine Menge M von n-Tupeln: Tupel $m_i \in M$ heißt Pareto-optimal, wenn es kein anderes $m_j \in M$ gibt, das m_i dominiert.

Menge $M' \subseteq M$ heißt Pareto-Menge, wenn alle $m \in M'$ Pareto-optimal.

Beispiel: Öffentlicher Verkehr (Ankunftszeit und # Umstiege)

Pareto-Menge

Definition (Pareto-Dominanz)

Gegeben zwei n-Tupel $m_i = (x_1, \ldots, x_n)$ und $m_j = (y_1, \ldots, y_n)$:

$$m_i \text{ dominiert } m_j \iff \forall k : x_k \leq y_k \text{ und } \exists \ell : x_\ell < y_\ell$$

(m_i ist in keinem Wert schlechter als m_j und in mindestens einem besser)

Definition (Pareto-Optimum)

Gegeben eine Menge M von n-Tupeln: Tupel $m_i \in M$ heißt Pareto-optimal, wenn es kein anderes $m_j \in M$ gibt, das m_i dominiert.

Menge $M' \subseteq M$ heißt Pareto-Menge, wenn alle $m \in M'$ Pareto-optimal.

Beispiel: Öffentlicher Verkehr (Ankunftszeit und # Umstiege)

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

Pareto-Menge

Definition (Pareto-Dominanz)

Gegeben zwei n-Tupel $m_i = (x_1, \ldots, x_n)$ und $m_j = (y_1, \ldots, y_n)$:

$$m_i \text{ dominiert } m_j \iff \forall k : x_k \leq y_k \text{ und } \exists \ell : x_\ell < y_\ell$$

(m_i ist in keinem Wert schlechter als m_j und in mindestens einem besser)

Definition (Pareto-Optimum)

Gegeben eine Menge M von n-Tupeln: Tupel $m_i \in M$ heißt Pareto-optimal, wenn es kein anderes $m_j \in M$ gibt, das m_i dominiert.

Menge $M' \subseteq M$ heißt Pareto-Menge, wenn alle $m \in M'$ Pareto-optimal.

Beispiel: Öffentlicher Verkehr (Ankunftszeit und # Umstiege)

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

Pareto-Menge

Definition (Pareto-Dominanz)

Gegeben zwei n-Tupel $m_i = (x_1, \ldots, x_n)$ und $m_j = (y_1, \ldots, y_n)$:

$$m_i \text{ dominiert } m_j \iff \forall k : x_k \leq y_k \text{ und } \exists \ell : x_\ell < y_\ell$$

(m_i ist in keinem Wert schlechter als m_j und in mindestens einem besser)

Definition (Pareto-Optimum)

Gegeben eine Menge M von n-Tupeln: Tupel $m_i \in M$ heißt Pareto-optimal, wenn es kein anderes $m_j \in M$ gibt, das m_i dominiert.

Menge $M' \subseteq M$ heißt Pareto-Menge, wenn alle $m \in M'$ Pareto-optimal.

Beispiel: Öffentlicher Verkehr (Ankunftszeit und # Umstiege)

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

Pareto-Menge

Definition (Pareto-Dominanz)

Gegeben zwei n-Tupel $m_i = (x_1, \ldots, x_n)$ und $m_j = (y_1, \ldots, y_n)$:

$$m_i \text{ dominiert } m_j \iff \forall k : x_k \leq y_k \text{ und } \exists \ell : x_\ell < y_\ell$$

(m_i ist in keinem Wert schlechter als m_j und in mindestens einem besser)

Definition (Pareto-Optimum)

Gegeben eine Menge M von n-Tupeln: Tupel $m_i \in M$ heißt Pareto-optimal, wenn es kein anderes $m_j \in M$ gibt, das m_i dominiert.

Menge $M' \subseteq M$ heißt Pareto-Menge, wenn alle $m \in M'$ Pareto-optimal.

Beispiel: Öffentlicher Verkehr (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?

Multi-Criteria Dijkstra (MCD)

Idee:

- Benutze Graph mit Kantengewicht len: $E \to \mathbb{R}^n_{\geq 0}$
- Grundlage: Dijkstras Algorithmus

Multi-Criteria Dijkstra (MCD)

Idee:

- Benutze Graph mit Kantengewicht len: $E \to \mathbb{R}^n_{\geq 0}$
- Grundlage: Dijkstras Algorithmus

Aber:

- Label ℓ sind n-Tupel $(x_1, \ldots, x_n)$
- An jedem Knoten $v \in V$: Pareto-Menge B_v von Labels
- Jedes Label entspricht einem (Pareto-optimalen) s–u-Pfad
- Priority Queue verwaltet Label statt Knoten
- Queue-Key ist Prioritätsfunktion $k(x_1, \ldots, x_n)$
  - Meist: Linearkombination oder lexikographische Sortierung
  - Priority Queue enthält ganze Label, nicht nur den Key
- Dominanz von Labels in B_v on-the-fly

Multi-Criteria Dijkstra (MCD)

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

```
1  B_v ← ∞ for all v ∈ V
2  B_s ← 0
3  Q.clear()
4  Q.insert(s, B_s)
5  while !Q.empty() do
6      (u, ℓ = dist) ← Q.deleteMin()
7      for all edges e = (u, v) ∈ E do
8          ℓ' ← dist + len(e)
9          if not B_v ≤ ℓ' then
10             B_v ← ℓ'
11             Q.insert(v, ℓ')
```

Multi-Criteria Dijkstra (MCD)

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

```
1  B_v ← ∅ for all v ∈ V
2  B_s ← {(0, ..., 0)}
3  Q.clear()
4  Q.insert(s, B_s)
5  while !Q.empty() do
6      (u, ℓ = dist) ← Q.deleteMin()
7      for all edges e = (u, v) ∈ E do
8          ℓ' ← dist + len(e)
9          if not B_v ≤ ℓ' then
10             B_v ← ℓ'
11             Q.insert(v, ℓ')
```

Multi-Criteria Dijkstra (MCD)

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

```
1  B_v ← ∅ for all v ∈ V
2  B_s ← {(0, ..., 0)}
3  Q.clear()
4  Q.insert(s, B_s)
5  while !Q.empty() do
6      (u, ℓ = (x_1, ..., x_n)) ← Q.deleteMin()
7      for all edges e = (u, v) ∈ E do
8          ℓ' ← dist + len(e)
9          if not B_v ≤ ℓ' then
10             B_v ← ℓ'
11             Q.insert(v, ℓ')
```

Multi-Criteria Dijkstra (MCD)

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

```
1  B_v ← ∅ for all v ∈ V
2  B_s ← {(0, ..., 0)}
3  Q.clear()
4  Q.insert(s, B_s)
5  while !Q.empty() do
6      (u, ℓ = (x_1, ..., x_n)) ← Q.deleteMin()
7      for all edges e = (u, v) ∈ E do
8          ℓ' ← (x_1 + len(e)_1, ..., x_n + len(e)_n)
9          if not B_v ≤ ℓ' then
10             B_v ← ℓ'
11             Q.insert(v, ℓ')
```

Multi-Criteria Dijkstra (MCD)

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

```
1  B_v ← ∅ for all v ∈ V
2  B_s ← {(0, ..., 0)}
3  Q.clear()
4  Q.insert(s, B_s)
5  while !Q.empty() do
6      (u, ℓ = (x_1, ..., x_n)) ← Q.deleteMin()
7      for all edges e = (u, v) ∈ E do
8          ℓ' ← (x_1 + len(e)_1, ..., x_n + len(e)_n)
9          if not B_v dominates ℓ' then
10             B_v ← ℓ'
11             Q.insert(v, ℓ')
```

Multi-Criteria Dijkstra (MCD)

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

```
1  B_v ← ∅ for all v ∈ V
2  B_s ← {(0, ..., 0)}
3  Q.clear()
4  Q.insert(s, B_s)
5  while !Q.empty() do
6      (u, ℓ = (x_1, ..., x_n)) ← Q.deleteMin()
7      for all edges e = (u, v) ∈ E do
8          ℓ' ← (x_1 + len(e)_1, ..., x_n + len(e)_n)
9          if not any ℓ'' ∈ B_v dominates ℓ' then
10             B_v ← ℓ'
11             Q.insert(v, ℓ')
```

Multi-Criteria Dijkstra (MCD)

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

```
1  B_v ← ∅ for all v ∈ V
2  B_s ← {(0, ..., 0)}
3  Q.clear()
4  Q.insert(s, B_s)
5  while !Q.empty() do
6      (u, ℓ = (x_1, ..., x_n)) ← Q.deleteMin()
7      for all edges e = (u, v) ∈ E do
8          ℓ' ← (x_1 + len(e)_1, ..., x_n + len(e)_n)
9          if not any ℓ'' ∈ B_v dominates ℓ' then
10             B_v.insert(ℓ')
11             Q.insert(v, ℓ')
```

Multi-Criteria Dijkstra (MCD)

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

```
1  B_v ← ∅ for all v ∈ V
2  B_s ← {(0, ..., 0)}
3  Q.clear()
4  Q.insert(s, B_s)
5  while !Q.empty() do
6      (u, ℓ = (x_1, ..., x_n)) ← Q.deleteMin()
7      for all edges e = (u, v) ∈ E do
8          ℓ' ← (x_1 + len(e)_1, ..., x_n + len(e)_n)
9          if not any ℓ'' ∈ B_v dominates ℓ' then
10             B_v.insert(ℓ')
11             Remove non-Pareto-optimal labels from B_v and Q
12             Q.insert(v, ℓ')
```

Diskussion: MCD

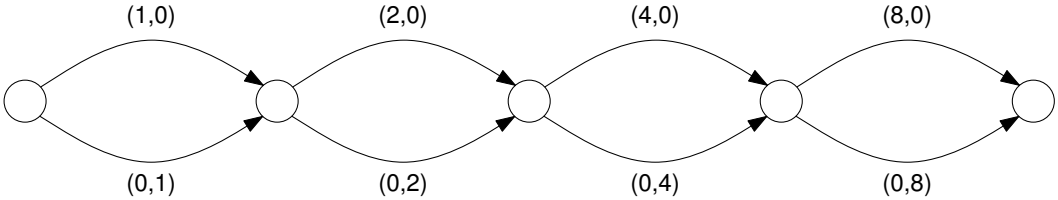
- Falls extrahiertes Label immer Pareto-optimal (bzgl. aller $\ell \in Q$):
  - MCD label-setting (einmal extrahierte Labels werden nie dominiert)
    dafür muss die Längenfunktion natürlich auch positiv sein
  - Gilt für Linearkombination und lexikographische Sortierung
- Pareto-Mengen B_u sind dynamische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

Diskussion: MCD

- 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 B_u sind dynamische Datenstrukturen $\rightsquigarrow$ teuer!
- Sehr viele Queue-Operationen
- Testen der Dominanz in $\mathcal{O}(|B_u|)$ möglich
- Stoppkriterium?
  - Pareto-Menge nicht vollständig, wenn t erreicht
  - Queue leer laufen lassen

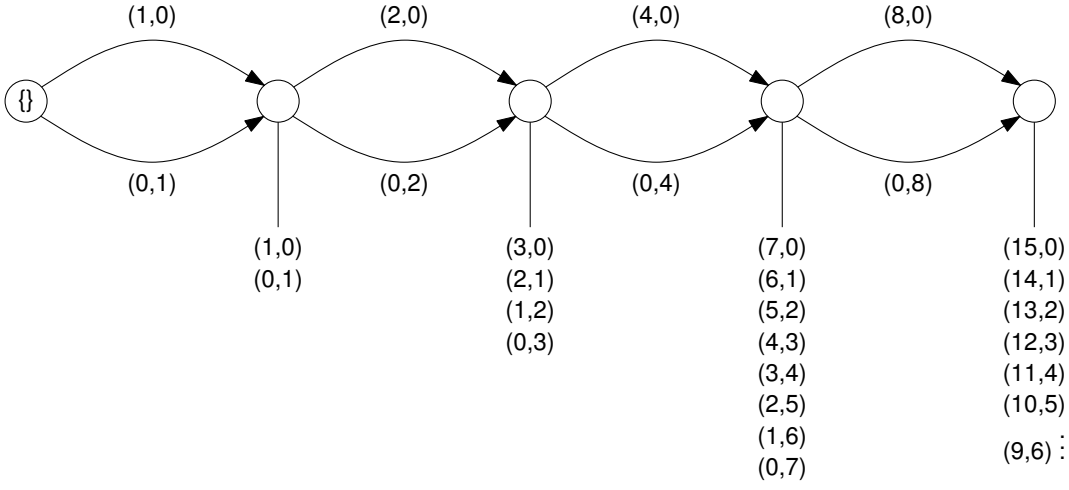
Exponentielle Laufzeit

Exponentiell wachsende Lösungsmenge bei zwei Kriterien:



Exponentielle Laufzeit

Exponentiell wachsende Lösungsmenge bei zwei Kriterien:



Verbesserungen

- Jedes B_u verwaltet seine eigene lokale Priority Queue
  $\Rightarrow$ globale Priority Queue auf Knoten statt Labels
- Hopping Reduction:
  Relaxieren der Kante zum Parent-Knoten p ist unnötig
  (kann keine Verbesserung bringen, kostet aber $\mathcal{O}(|B_p|)$ für Test)
  $\Rightarrow$ Überspringe Kante zum Parent-Knoten des aktuellen Labels
- Target-Pruning: (statt Abbruchkriterium)
  Verwerfe Label, wenn es von der tentativen Pareto-Menge B_t am Ziel t dominiert wird

Verbesserungen

- Jedes B_u verwaltet seine eigene lokale Priority Queue
  $\Rightarrow$ globale Priority Queue auf Knoten statt Labels
- Hopping Reduction:
  Relaxieren der Kante zum Parent-Knoten p ist unnötig
  (kann keine Verbesserung bringen, kostet aber $\mathcal{O}(|B_p|)$ für Test)
  $\Rightarrow$ Überspringe Kante zum Parent-Knoten des aktuellen Labels
- Target-Pruning: (statt Abbruchkriterium)
  Verwerfe Label, wenn es von der tentativen Pareto-Menge B_t am Ziel t dominiert wird

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

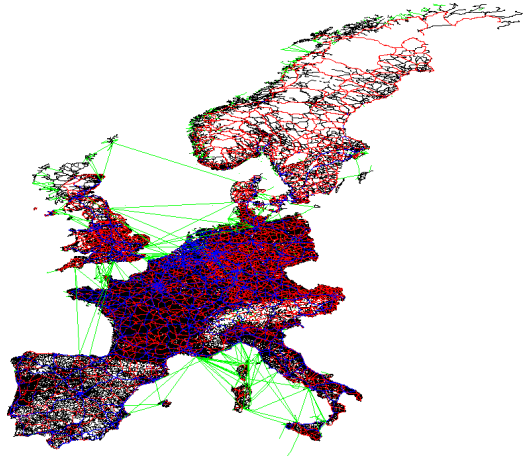
Eingabe

Straßengraphen von:

- Luxemburg
- Karlsruhe
- Europa

Metriken:

- Fahrzeit schnelles Auto
- Fahrzeit langsames Auto
- Kosten
- Distanz
- Unit-Metrik (alle Kantengewichte 1)



Ähnliche Metriken: Europa

Beobachtungen:

- Nicht viele zusätzliche Lösungen
- Anzahl Lösungen und Queue Extracts korrelieren
- Queryzeit steigt viel stärker
- Anzahl Dominanztests ist superlinear

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

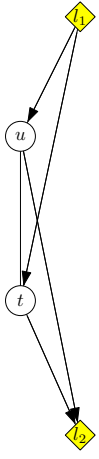
Verschiedene Metriken

	Luxemburg			Karlsruhe		
metrics	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

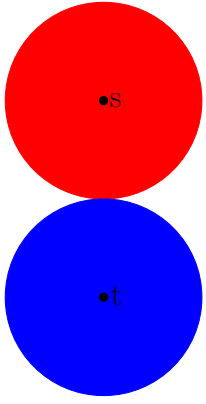
- Je nach Kriterien kann Lösungsmenge stark ansteigen

Beschleunigungstechniken

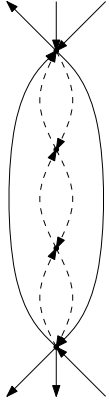
Landmarken



Bidirektionale Suche



Kontraktion



Arc-Flags

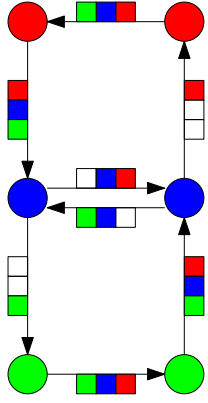
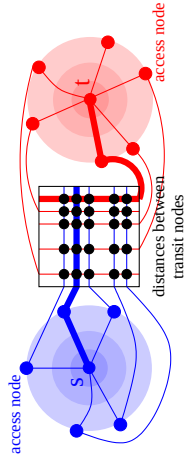


Table-Lookups



Wdh.: ALT

Vorberechnung:

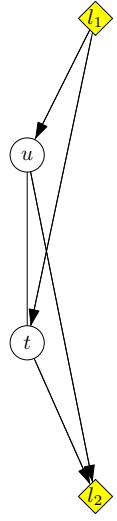
- Wähle einige Knoten (≈ 16) als Landmarken
- Berechne Abstände von und zu allen Landmarken

Anfrage:

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

$$\begin{aligned} d(u,t) &\geq d(\ell_1,t) - d(\ell_1,u) \\ d(u,t) &\geq d(u,\ell_2) - d(t,\ell_2) \end{aligned}$$

- Benutze untere Schranken als A*-Potential
- Verändert Reihenfolge der besuchten Knoten



Anpassung

Idee:

- Berechne Distanzen pro Metrik unabhängig voneinander
- Pro Knoten u und Landmarke ℓ_i:
  Distanzvektor $(d_1(u, \ell_i), \ldots, d_n(u, \ell_i))$
- Liefert Potentiale $\pi_1, \ldots, \pi_n$
- Zielrichtung: Queue-Key eines Labels ist $k(x_1 + \pi_1, \ldots, x_n + \pi_n)$.
- Potential π_i liefert untere Schranke für $d_i(u, t)$
  $\Rightarrow$ Nutze $(x_1 + \pi_1, \ldots, x_n + \pi_n)$ auch für Target-Pruning

Anpassung

Idee:

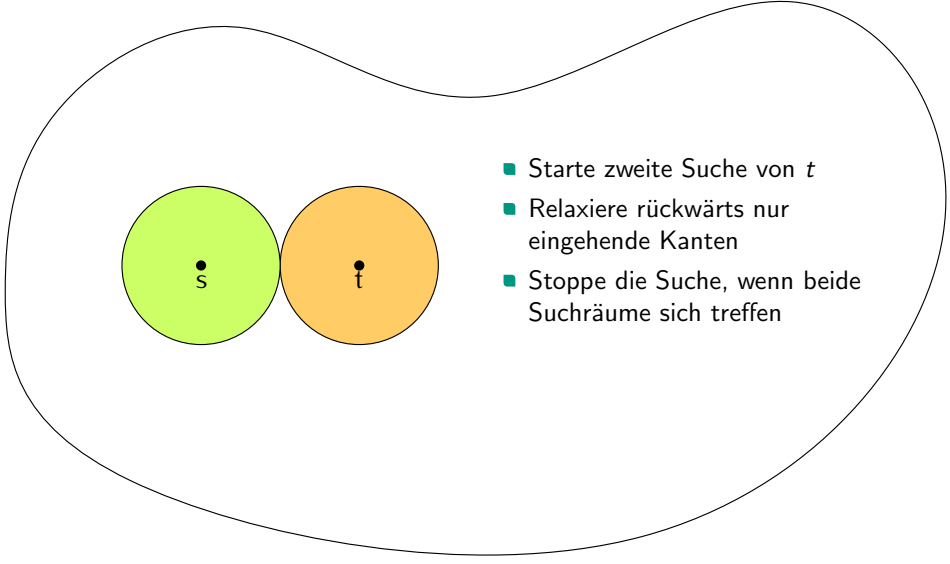
- Berechne Distanzen pro Metrik unabhängig voneinander
- Pro Knoten u und Landmarke ℓ_i:
  Distanzvektor $(d_1(u, \ell_i), \ldots, d_n(u, \ell_i))$
- Liefert Potentiale $\pi_1, \ldots, \pi_n$
- Zielrichtung: Queue-Key eines Labels ist $k(x_1 + \pi_1, \ldots, x_n + \pi_n)$.
- Potential π_i liefert untere Schranke für $d_i(u, t)$
  $\Rightarrow$ Nutze $(x_1 + \pi_1, \ldots, x_n + \pi_n)$ auch für Target-Pruning

Modifikation:

Berechne zur Queryzeit $(d_1(u, t), \ldots, d_n(u, t))$

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

Bidirektionale Suche



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

Anpassung

Idee:

- Rückwärtssuche kein Problem (analog)

Anpassung

Idee:

- Rückwärtssuche kein Problem (analog)

Offenes Problem:

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

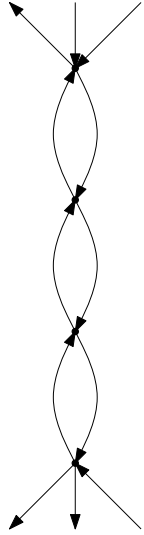
Kontraktion

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



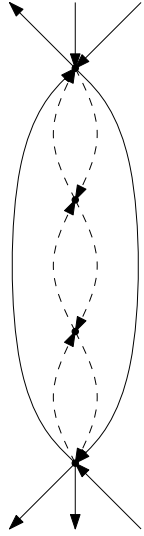
Kontraktion

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

Unikriteriell:

- 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

Multikriteriell:

Anpassung Zeugensuche

Unikriteriell:

- 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

Multikriteriell:

- Lösche Kante (u, v), wenn (u, v) kein Pareto-optimaler u-v-Pfad ist
- Lokale multikriterielle Suche
- Kann zu (Pareto-optimalen) Multikanten führen
- Problem: „Explosion“ der Anzahl der Pfade

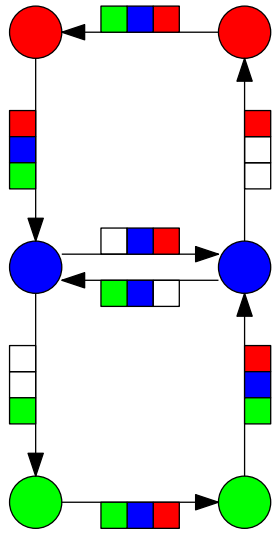
Arc-Flags

Idee:

- Partitioniere den Graph in k Zellen
- Hänge ein Label mit k Bits an jede Kante
- Gibt an, ob Kante 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...

- Unikriteriell: Kante auf kürzestem Pfad in Zielzelle liegt
- Multikriteriell: Kante auf Pareto-optimalem Pfad in Zielzelle liegt

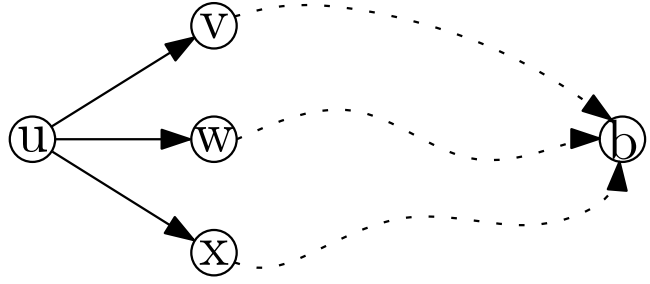
Anpassung

Setze Flagge, wenn...

- Unikriteriell: Kante auf kürzestem Pfad in Zielzelle liegt
- Multikriteriell: Kante auf Pareto-optimalem Pfad in Zielzelle liegt

Berechnung:

- Für jeden Randknoten b und jeden Knoten u:
- Berechne alle Pareto-optimalen u-b-Pfade
- Flagge alle Kanten auf diesen Pfaden



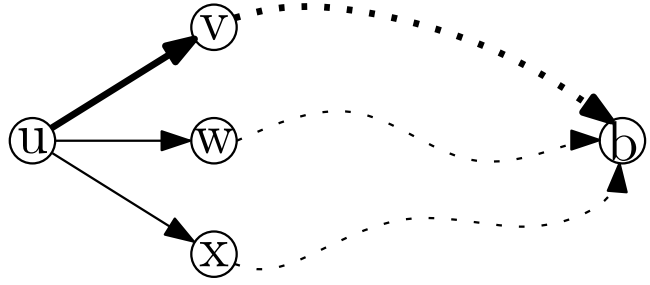
Anpassung

Setze Flagge, wenn...

- Unikriteriell: Kante auf kürzestem Pfad in Zielzelle liegt
- Multikriteriell: Kante auf Pareto-optimalem Pfad in Zielzelle liegt

Berechnung:

- Für jeden Randknoten b und jeden Knoten u:
- Berechne alle Pareto-optimalen u-b-Pfade
- Flagge alle Kanten auf diesen Pfaden



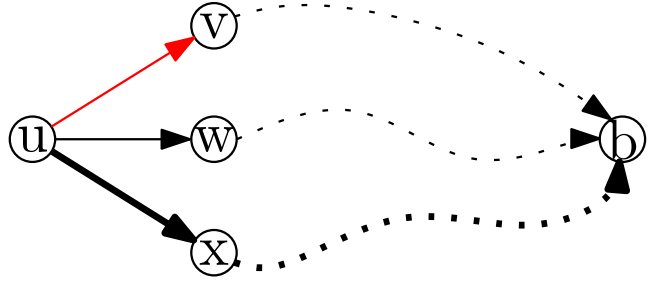
Anpassung

Setze Flagge, wenn...

- Unikriteriell: Kante auf kürzestem Pfad in Zielzelle liegt
- Multikriteriell: Kante auf Pareto-optimalem Pfad in Zielzelle liegt

Berechnung:

- Für jeden Randknoten b und jeden Knoten u:
- Berechne alle Pareto-optimalen u-b-Pfade
- Flagge alle Kanten auf diesen Pfaden



Anpassung

Setze Flagge, wenn...

- Unikriteriell: Kante auf kürzestem Pfad in Zielzelle liegt
- Multikriteriell: Kante auf Pareto-optimalem Pfad in Zielzelle liegt

Berechnung:

- Für jeden Randknoten b und jeden Knoten u:
- Berechne alle Pareto-optimalen u-b-Pfade
- Flagge alle Kanten auf diesen Pfaden

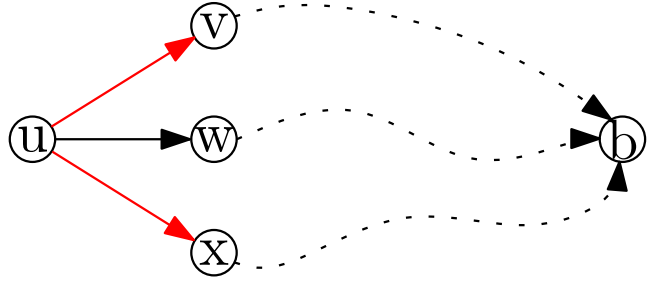
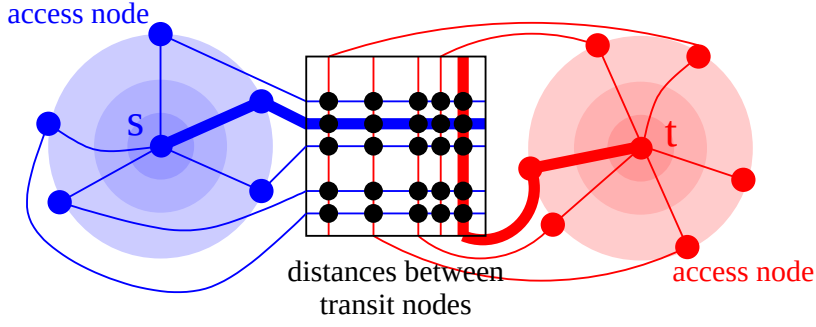


Table-Lookups

Idee:

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



Anpassung

Beobachtung:

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

Also:

- Speicherverbrauch deutlich zu groß!

Diskussion Basismodule

Basismodule:

- – Bidirektionale Suche
- \+ Landmarken/A*
- \+ Kontraktion
- \+ Arc-Flags
- – Table-Lookups

Experimente

Pareto-SHARC [DW09] (nur als Beispiel)

	Luxemburg					Karlsruhe				
	Prepro	Query				Prepro	Query			
metrics	time [h:m]	target labels	#del. mins	time [ms]	spd up	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

Zusammenfassung

- Berechnung der Pareto-Menge nicht effizient möglich
- Auch mit Beschleunigungstechniken 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...

Constrained Shortest Paths

Constrained Shortest Path (CSP)

Ziel:

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

Constrained Shortest Path (CSP)

Ziel:

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

Definition: Constrained Shortest Path Problem

Gegeben: $G = (V, E)$, Länge $\ell\colon E \to \mathbb{N}_0$, Gewicht $\omega\colon E \to \mathbb{N}_0$, 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?

Das entsprechende Optimierungsproblem lautet:

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

Constrained Shortest Path (CSP)

Theorem

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

Constrained Shortest Path (CSP)

Theorem

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

Beweis:

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

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

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

- Beweis durch Reduktion von Partition

Partition

Definition: Partition

Gegeben: Endliche Menge A sowie Größe $f : A \to \mathbb{N}_0$
Problem: Existiert ein Teilmenge $A' \subseteq A$, für die gilt:

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

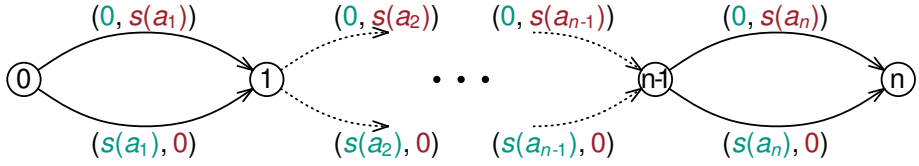
Anmerkung:

- In Karps 21 $\mathcal{NP}$-vollständigen Problemen enthalten [Kar72]
- Partition ist (schwach) $\mathcal{NP}$-vollständig
  - $\mathcal{NP}$-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:

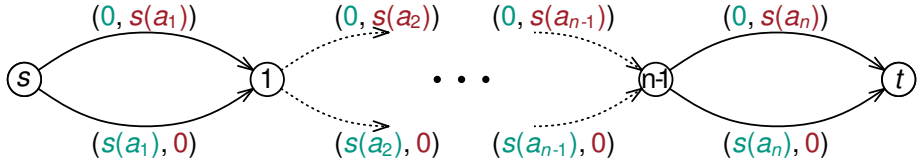
- Sei $\Pi = (A = \{a_1, \ldots, a_n\}, f)$ eine Partition-Instanz
- Konstruiere Graph $G = (V, E)$ mit $n + 1$ Knoten ($V = \{0, 1, \ldots, n\}$)
- Knoten $i - 1$ und i sind jeweils durch zwei Kanten verbunden
  - Eine Kante hat Länge 0 und Gewicht $f(a_i)$
  - Die andere Kante hat Länge $f(a_i)$ und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



Beweis: CSP ist $\mathcal{NP}$-vollständig

Reduktion von Partition:

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



Es gilt:

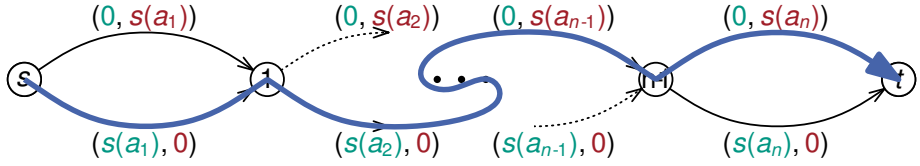
- Π ist Partition-Ja-Instanz $\Leftrightarrow$ $(G, \ell, \omega, L, W, s, t)$ ist CSP-Ja-Instanz

□

Beweis: CSP ist $\mathcal{NP}$-vollständig

Reduktion von Partition:

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



Beispiel:

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

□

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 $> W$

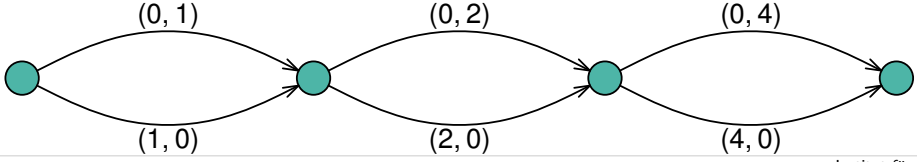
Multi-Criteria Dijkstra für CSP

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



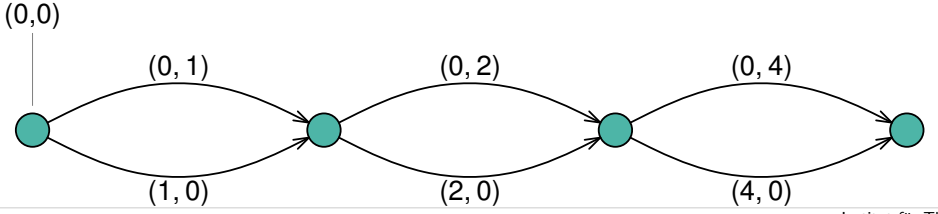
Multi-Criteria Dijkstra für CSP

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



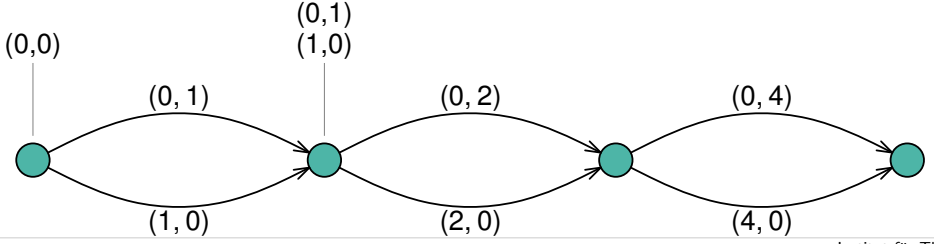
Multi-Criteria Dijkstra für CSP

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



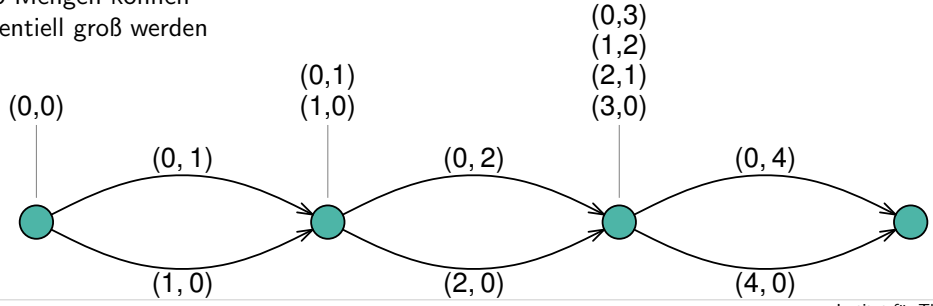
Multi-Criteria Dijkstra für CSP

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



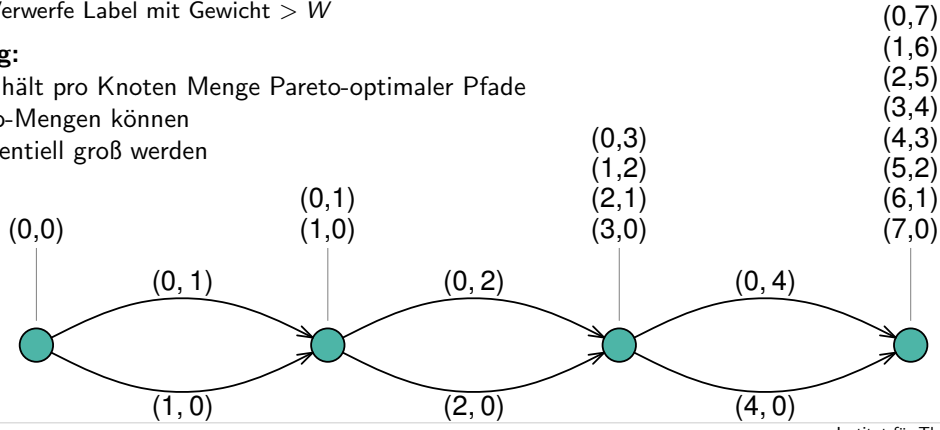
Multi-Criteria Dijkstra für CSP

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



Schnellste zulässige Route

Verbesserungen: Standard-Beschleunigungen von MCD übertragbar:

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

Schnellste zulässige Route

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
  (Annahme: Queue ist nach Länge sortiert)

Zusammenfassung

- Multikriterielle Optimierung
  - Verallgemeinerung zu Multi-Criteria Dijkstra
  - Anpassung von Beschleunigungstechniken
- Constrained Shortest Paths
  - $\mathcal{NP}$-schwer
  - Anpassung MC-Dijkstra

Literatur I

Daniel Delling and Dorothea Wagner.
Pareto paths with SHARC.
In *Proceedings of the 8th International Symposium on Experimental Algorithms (SEA'09)*, volume 5526 of *Lecture Notes in Computer Science*, pages 125–136. Springer, June 2009.

Richard M. Karp.
Reducibility among combinatorial problems.
In *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.