Karlsruher Institut für Technologie

# Algorithmen für Routenplanung

14. Vorlesung, Sommersemester 2018

Tobias Zündorf | 25. Juni 2018

Institut für Theoretische Informatik · Algorithmik · Prof. Dr. Dorothea Wagner

# Diskussion

## Stoff bisher

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

# Diskussion

**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?**

# Diskussion

## 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

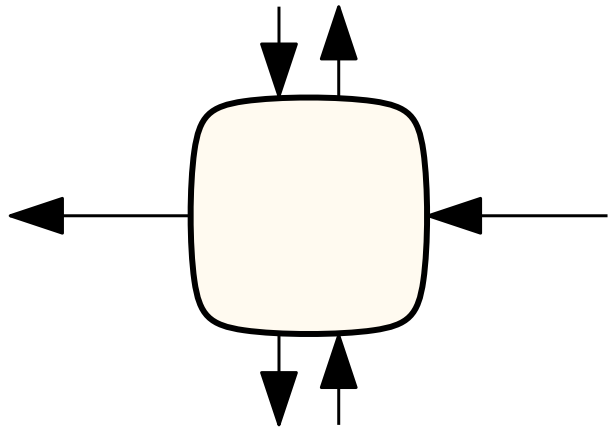


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

# Abbiegeverbote/-kosten

**Bisher:**

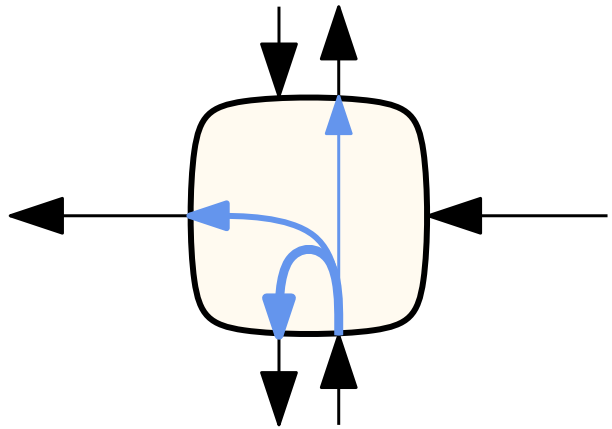
- Kreuzungen $\rightarrow$ Knoten
- Straßen $\rightarrow$ Kanten



# Abbiegeverbote/-kosten

**Bisher:**

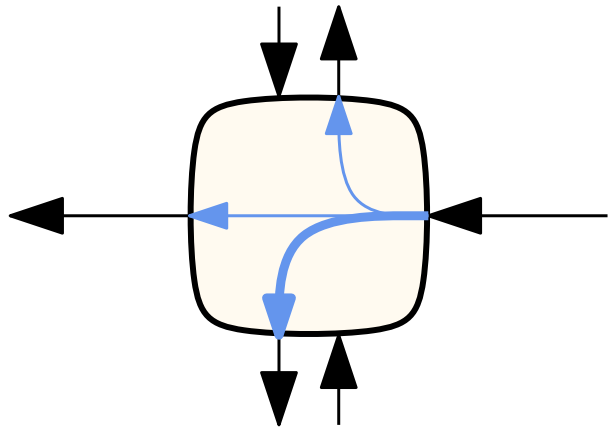
- Kreuzungen $\rightarrow$ Knoten
- Straßen $\rightarrow$ Kanten



**Aber:**

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

# Abbiegeverbote/-kosten



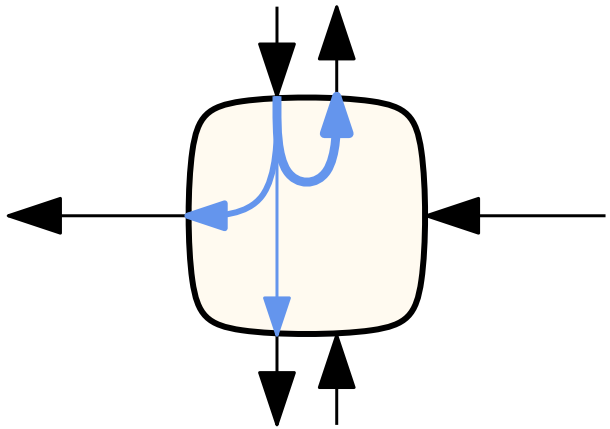
**Bisher:**

- Kreuzungen $\rightarrow$ Knoten
- Straßen $\rightarrow$ Kanten

**Aber:**

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

# Abbiegeverbote/-kosten



**Bisher:**

- Kreuzungen $\rightarrow$ Knoten
- Straßen $\rightarrow$ Kanten

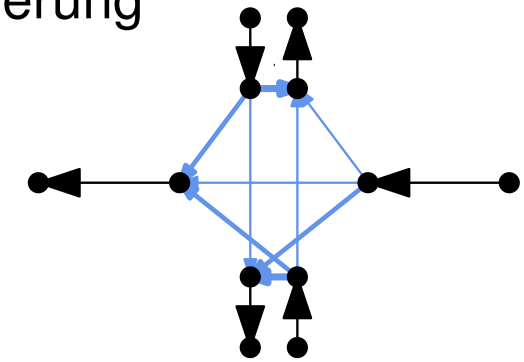
**Aber:**

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

# Modellierung

## Möglichkeit I: Expandierter Graph

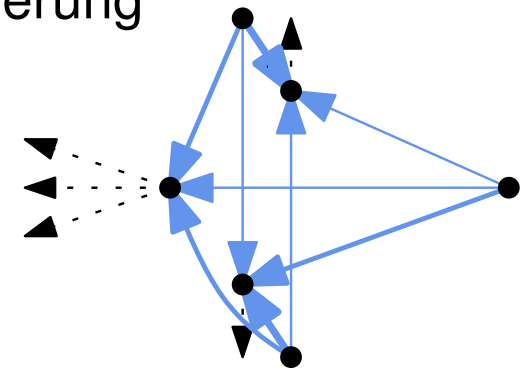
- Vergrössern des Graphen durch Ausmodellierung
- Kantenbasierter Graph da
  - Straßen $\rightarrow$ Knoten
  - Turns $\rightarrow$ Kanten
- Redundante Information



# Modellierung

## Möglichkeit I: Expandierter Graph

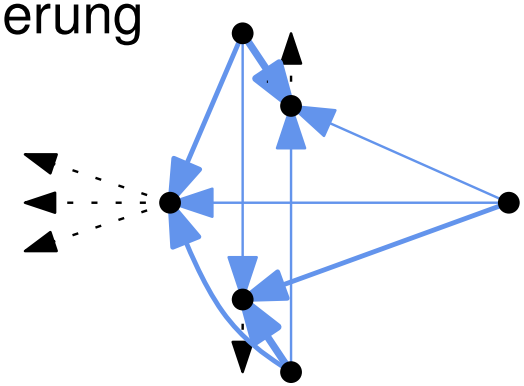
- Vergrössern des Graphen durch Ausmodellierung
- Kantenbasierter Graph da
  - Straßen $\rightarrow$ Knoten
  - Turns $\rightarrow$ Kanten
- Redundante Information
- Entferne einen Knoten pro Straße



# Modellierung

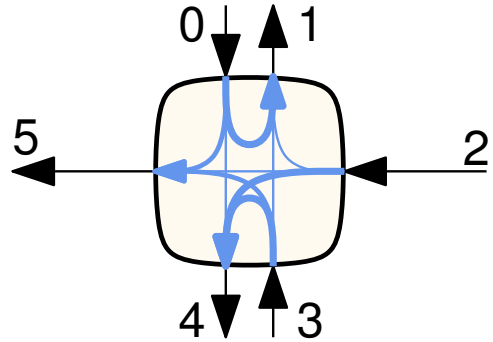
## Möglichkeit I: Expandierter Graph

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



## Möglichkeit II: Kompaktes Modell

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

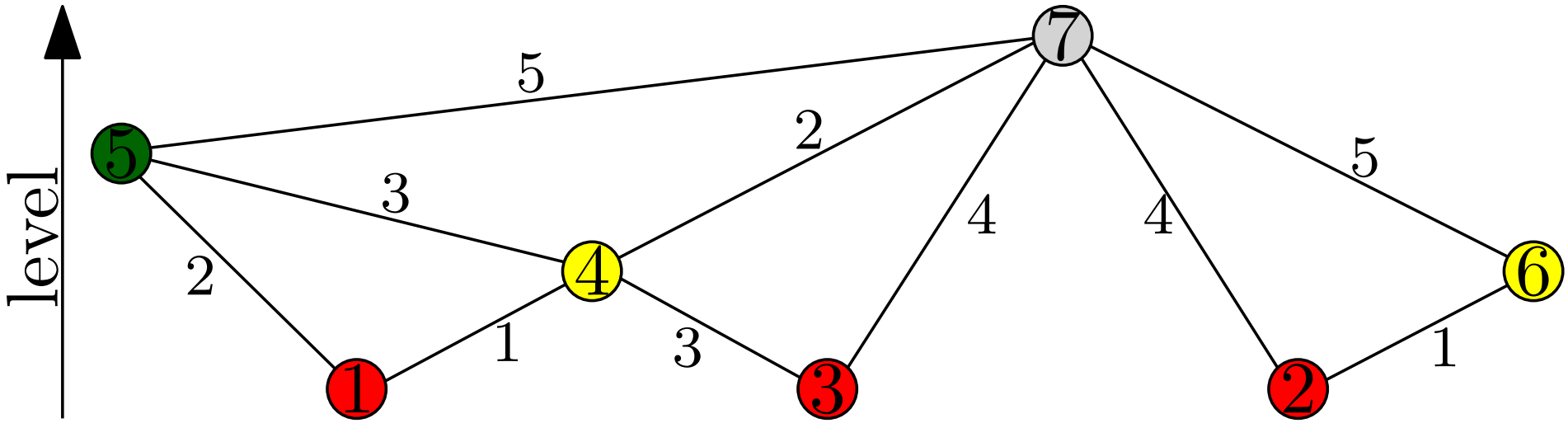



| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | · | · | · | · | · | · |
| 1 | · | · | · | | | |
| 2 | | | | | | |
| 3 | | | | | | |
| 4 | | | | | | |
| 5 | | | | | | |

# Wdh: Contraction Hierarchies

**Preprocessing:**

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



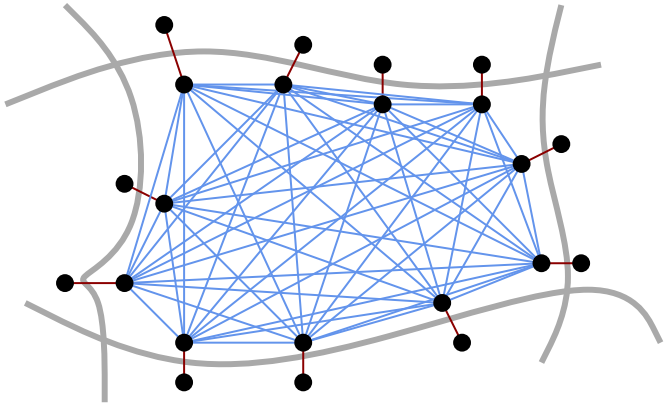
# Wdh: Multi Level Dijkstra

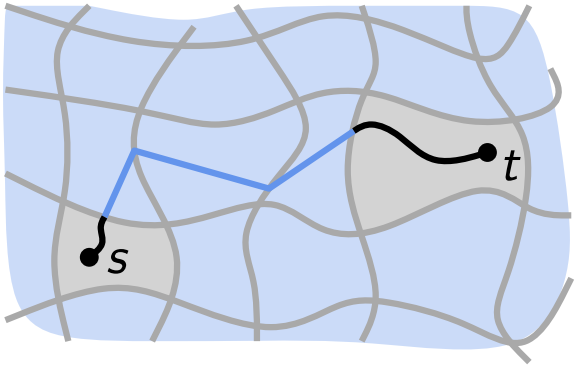
**Idee:**

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

**Overlay Graph**:

- Randknoten
- Cliquen in jeder Zelle
- Schnittkanten






**Suchgraph:**

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

**Optimierung:** multiple Level

# Expandierter Graph

**Dijkstra:**

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

# Expandierter Graph

**Dijkstra:**

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

**CH**

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

# Expandierter Graph

**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?)

# Kompaktes Modell

**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

- Zeugensuche wird komplizierter

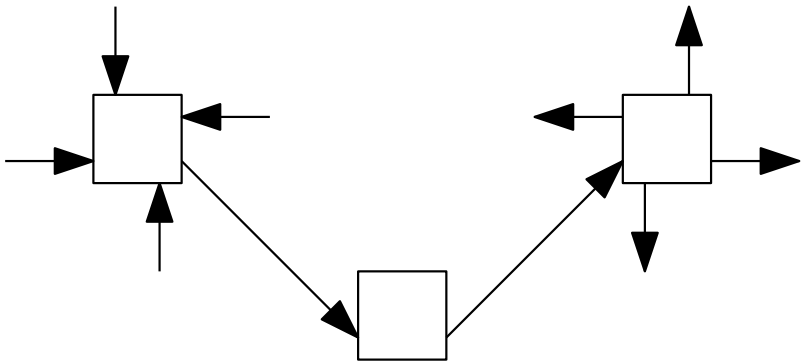


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

# Kompaktes Modell

## CH

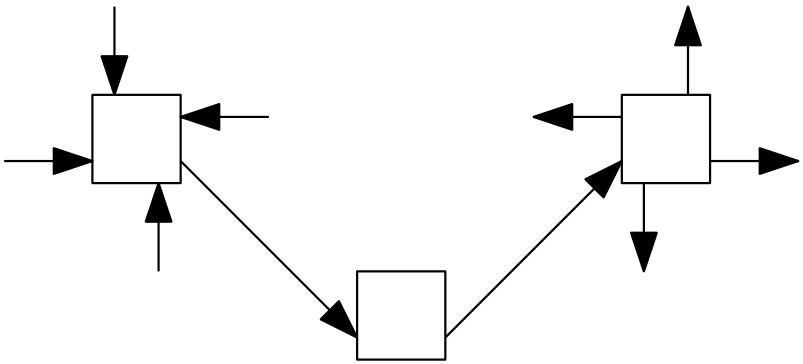
- Zeugensuche wird komplizierter
  - Eine Zeugensuche pro Paar eingehender und ausgehender Kanten



# Kompaktes Modell

## CH

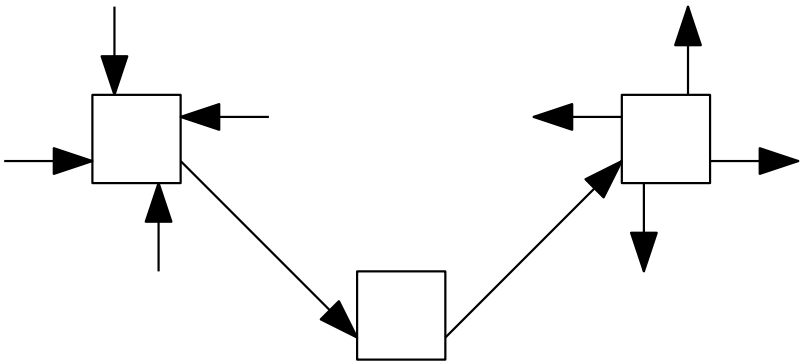
- Zeugensuche wird komplizierter
  - Eine Zeugensuche pro Paar eingehender und ausgehender Kanten
  - Es können Self-Loops entstehen

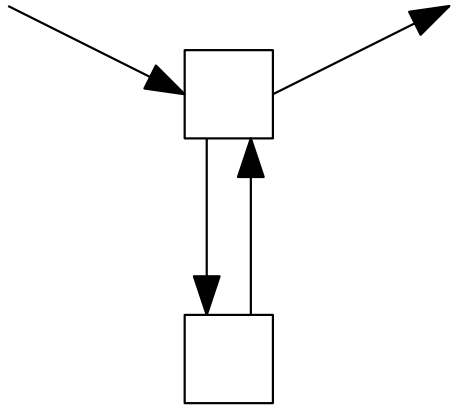


# Kompaktes Modell

## CH

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

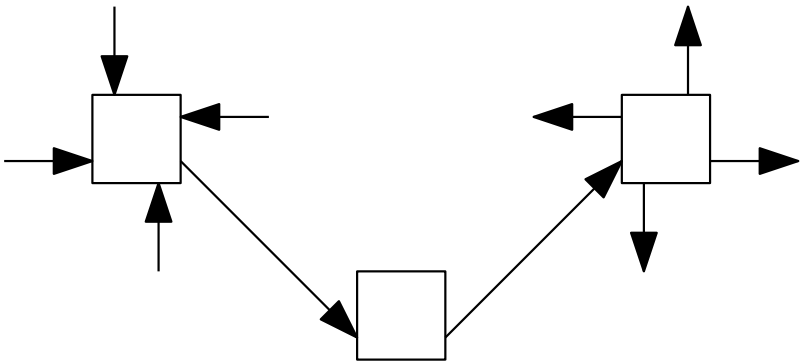


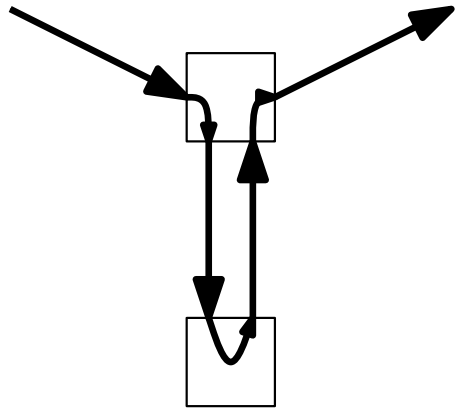


# Kompaktes Modell

## CH

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

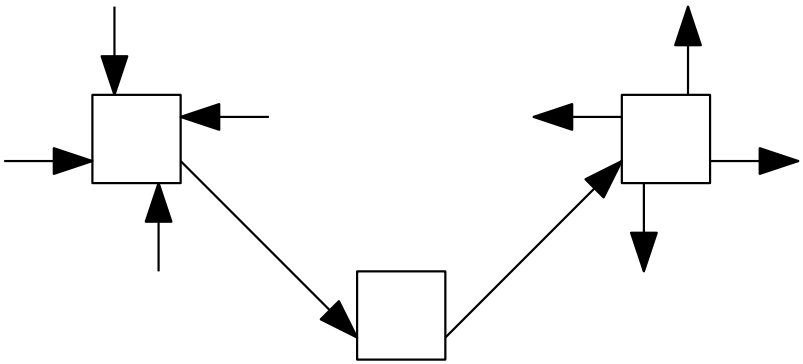


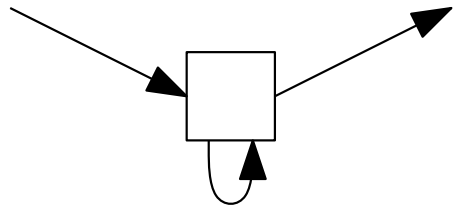


# Kompaktes Modell

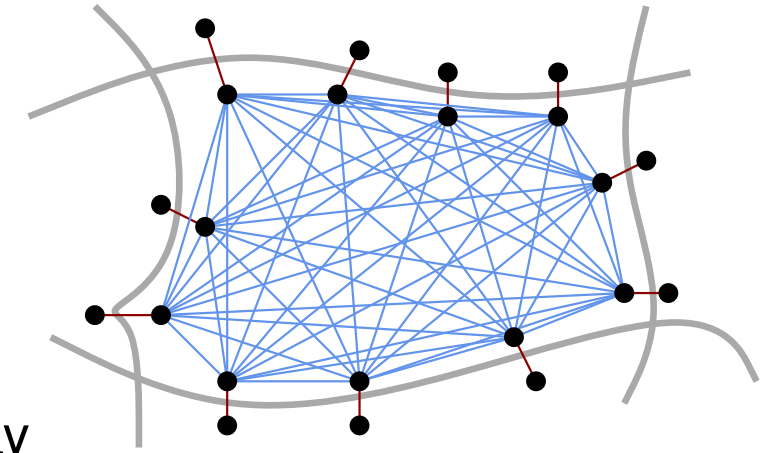
## CH

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





# Kompaktes Modell



**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

# Ergebnisse

| U-Turn cost | Algorithm | Customization time [s] | Customization [MB] | Queries #scans | Queries time [ms] |
|---|---|---|---|---|---|
| 1 s | MLD $[2^8:2^{12}:2^{16}:2^{20}]$ | 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 |
| 100 s | MLD $[2^8:2^{12}:2^{16}:2^{20}]$ | 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



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

# Optimierungskriterien

**Bisher:**

Kürzester Weg

- Eine Metrik

Alternativrouten

- Eine Metrik; fast kürzeste, dennoch sinnvolle Wege

Zeitabhängigkeit / Dynamische Szenarien

- Eine Metrik; Kantengewichte ändern sich über die Zeit

# Optimierungskriterien

**Bisher:**

Kürzester Weg

- Eine Metrik

Alternativrouten

- Eine Metrik; fast kürzeste, dennoch sinnvolle Wege

Zeitabhängigkeit / Dynamische Szenarien

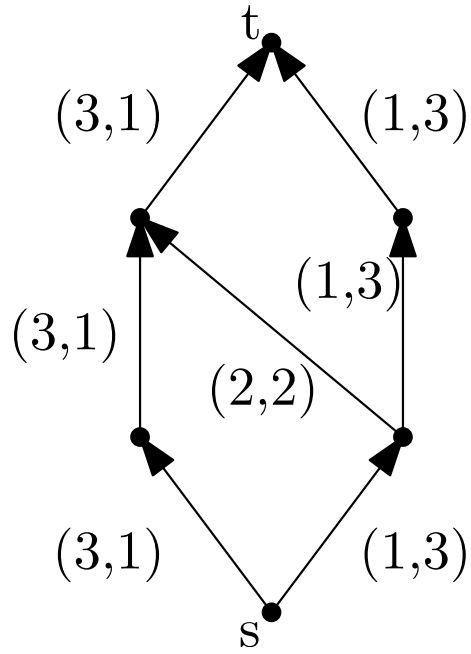
- Eine Metrik; Kantengewichte ändern sich über die Zeit

**Jetzt:** Mehrere Metriken

# Multikriterielle Optimierung

## Idee:

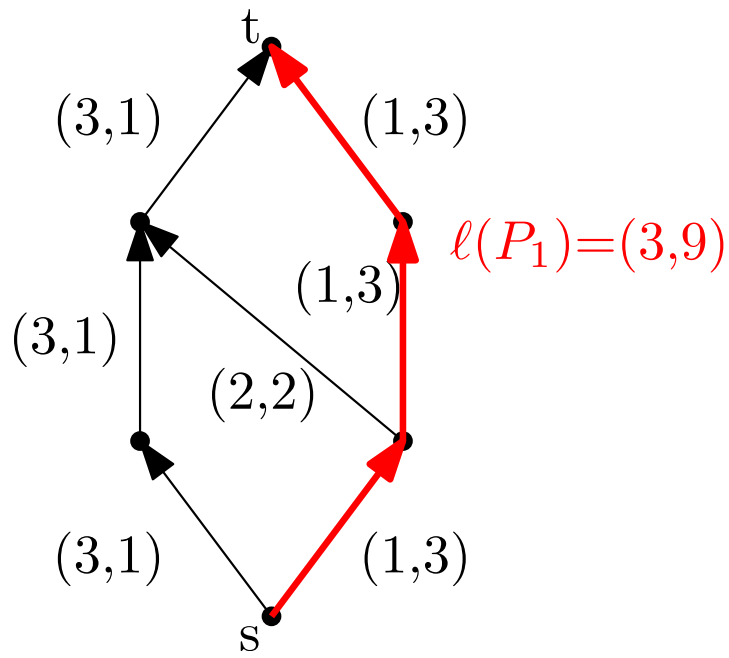
- Mehrere Gewichte an Kanten
  (Reisezeiten, Kosten, Energieverbrauch)



# Multikriterielle Optimierung

## Idee:

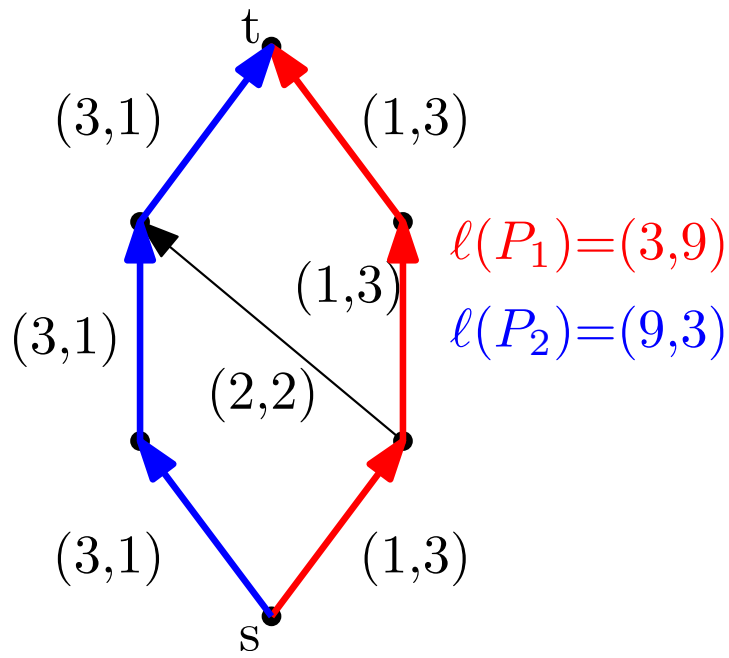
- Mehrere Gewichte an Kanten (Reisezeiten, Kosten, Energieverbrauch)
- Berechne alle Pareto-optimalen Routen
  - Route ist Pareto-optimal :⇔ Keine andere Routen dominiert sie
  - Route dominiert andere :⇔ Jedes Kriterium ist gleich oder besser
  - Grenzfall: Gleich gut in allen Kriterien (wird unterschiedlich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



# Multikriterielle Optimierung

## Idee:

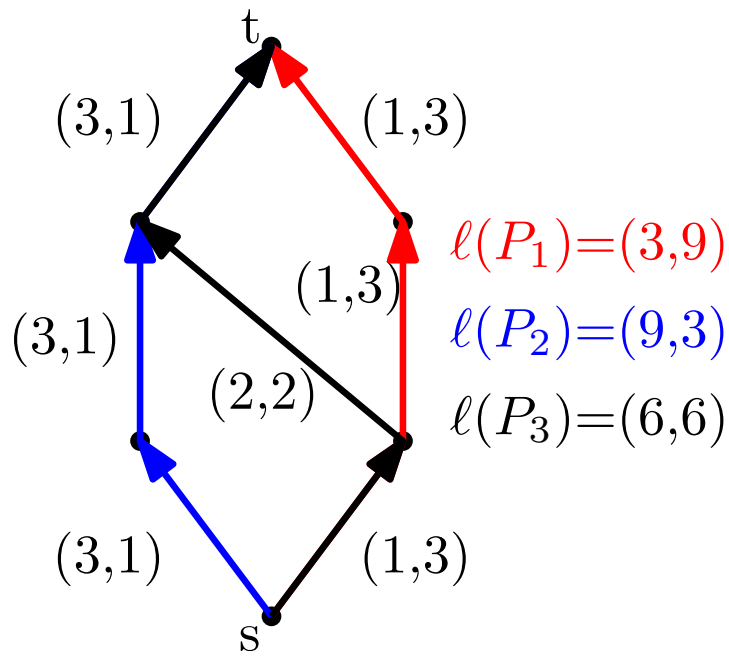
- Mehrere Gewichte an Kanten (Reisezeiten, Kosten, Energieverbrauch)
- Berechne alle Pareto-optimalen Routen
  - Route ist Pareto-optimal :⇔ Keine andere Routen dominiert sie
  - Route dominiert andere :⇔ Jedes Kriterium ist gleich oder besser
  - Grenzfall: Gleich gut in allen Kriterien (wird unterschiedlich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



# Multikriterielle Optimierung

## Idee:

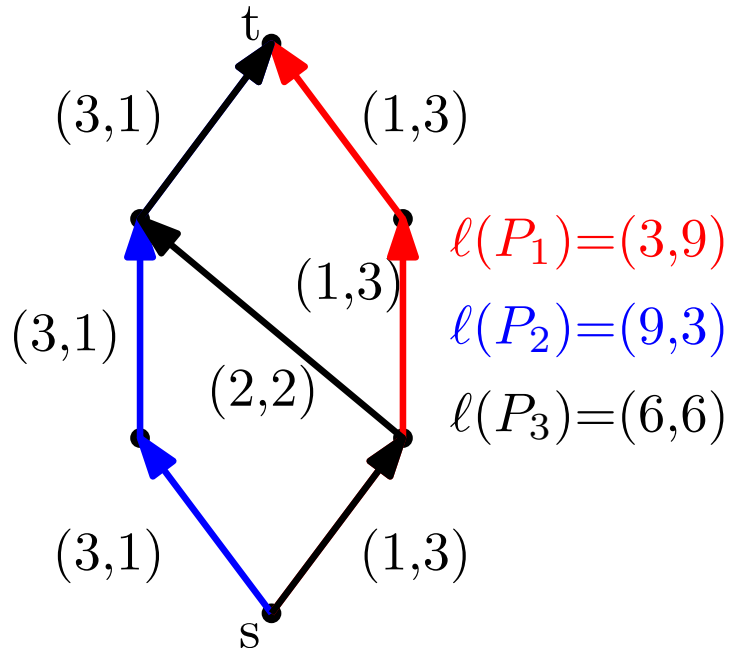
- Mehrere Gewichte an Kanten
  (Reisezeiten, Kosten, Energieverbrauch)
- Berechne alle Pareto-optimalen Routen
  - Route ist Pareto-optimal :⇔
    Keine andere Routen dominiert sie
  - Route dominiert andere :⇔
    Jedes Kriterium ist gleich oder besser
  - Grenzfall: Gleich gut in allen Kriterien
 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



# Multikriterielle Optimierung

## Idee:

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




## Herausforderung:

- Viele Routen zum Ziel

# Pareto-Menge

## Definition (Pareto-Dominanz)

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

# Pareto-Menge

## Definition (Pareto-Dominanz)

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

# Pareto-Menge

## Definition (Pareto-Dominanz)

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

# Pareto-Menge

## Definition (Pareto-Dominanz)

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

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

# Pareto-Menge

## Definition (Pareto-Dominanz)

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

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

# Pareto-Menge

## Definition (Pareto-Dominanz)

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

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

# Pareto-Menge

## Definition (Pareto-Dominanz)

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

# Multi-Criteria Dijkstra (MCD)

## Idee

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

# Multi-Criteria Dijkstra (MCD)

**Idee**

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

...aber...

- Label $\ell$ sind $n$-Tupel $(x_1, \ldots, x_n)$
- 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 $k(x_1, \ldots, x_n)$
  - Meist: Linearkombination oder lexikographische Sortierung
- Dominanz von Labeln in $L_u$ on-the-fly

# Multi-Criteria Dijkstra (MCD)

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

```
1  L_u ← ∞ for all u ∈ V
2  L_s ← 0
3  Q.clear()
4  Q.insert(s, 0)
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 L_v ≤ ℓ' then
10             L_v ← ℓ'
11             Q.insert(v, ℓ')
```

# Multi-Criteria Dijkstra (MCD)

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

```
1  L_u ← ∅ for all u ∈ V
2  L_s ← {(0, ..., 0)}
3  Q.clear()
4  Q.insert(s, k(0, ..., 0))
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 L_v ≤ ℓ' then
10             L_v ← ℓ'
11             Q.insert(v, ℓ')
```

# Multi-Criteria Dijkstra (MCD)

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

```
1  L_u ← ∅ for all u ∈ V
2  L_s ← {(0, ..., 0)}
3  Q.clear()
4  Q.insert(s, k(0, ..., 0))
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 L_v ≤ ℓ' then
10             L_v ← ℓ'
11             Q.insert(v, ℓ')
```

# Multi-Criteria Dijkstra (MCD)

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

```
1  L_u ← ∅ for all u ∈ V
2  L_s ← {(0, ..., 0)}
3  Q.clear()
4  Q.insert(s, k(0, ..., 0))
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 L_v ≤ ℓ' then
10             L_v ← ℓ'
11             Q.insert(v, ℓ')
```

# Multi-Criteria Dijkstra (MCD)

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

```
1  L_u ← ∅ for all u ∈ V
2  L_s ← {(0, ..., 0)}
3  Q.clear()
4  Q.insert(s, k(0, ..., 0))
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 L_v dominates ℓ' then
10             L_v ← ℓ'
11             Q.insert(v, ℓ')
```

# Multi-Criteria Dijkstra (MCD)

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

```
1  L_u ← ∅ for all u ∈ V
2  L_s ← {(0, ..., 0)}
3  Q.clear()
4  Q.insert(s, k(0, ..., 0))
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'' ∈ L_v dominates ℓ' then
10             L_v ← ℓ'
11             Q.insert(v, ℓ')
```

# Multi-Criteria Dijkstra (MCD)

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

```
1  L_u ← ∅ for all u ∈ V
2  L_s ← {(0, ..., 0)}
3  Q.clear()
4  Q.insert(s, k(0, ..., 0))
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'' ∈ L_v dominates ℓ' then
10             L_v.insert(ℓ')
11             Q.insert(v, ℓ')
```

# Multi-Criteria Dijkstra (MCD)

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

```
1  L_u ← ∅ for all u ∈ V
2  L_s ← {(0, ..., 0)}
3  Q.clear()
4  Q.insert(s, k(0, ..., 0))
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'' ∈ L_v dominates ℓ' then
10             L_v.insert(ℓ')
11             Remove non-Pareto-optimal labels from L_v
12             Q.insert(v, ℓ')
```

# Multi-Criteria Dijkstra (MCD)

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

```
1  L_u ← ∅ for all u ∈ V
2  L_s ← {(0, ..., 0)}
3  Q.clear()
4  Q.insert(s, k(0, ..., 0))
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'' ∈ L_v dominates ℓ' then
10             L_v.insert(ℓ')
11             Remove non-Pareto-optimal labels from L_v
12             Q.insert(v, k(ℓ'))
```

# Diskussion: MCD

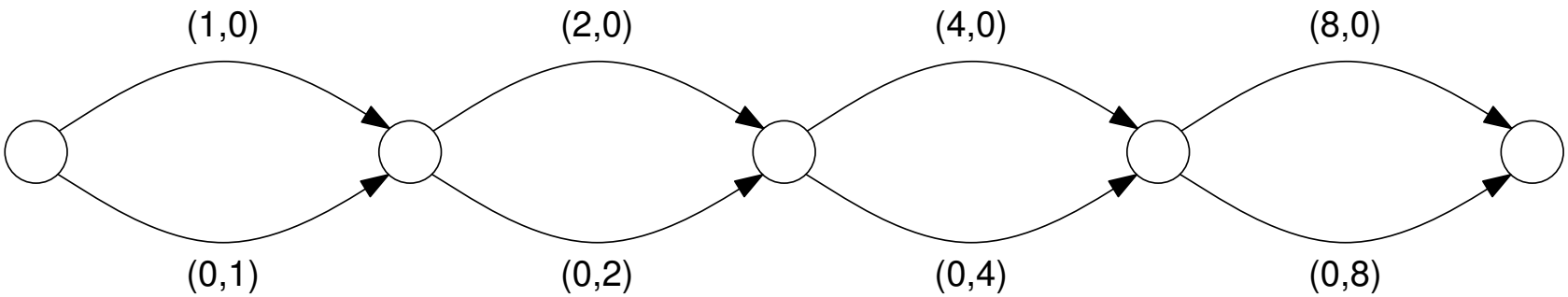
- Falls extrahiertes Label immer Pareto-optimal (bzgl. aller $\ell \in Q$):
  - MCD label-setting (einmal extrahierte Labels werden nie dominiert)
    dafür muss die Längenfunktion natürlich auch positiv sein
  - Gilt für Linearkombination und lexikographische Sortierung
- Pareto-Mengen $L_u$ sind dynamische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

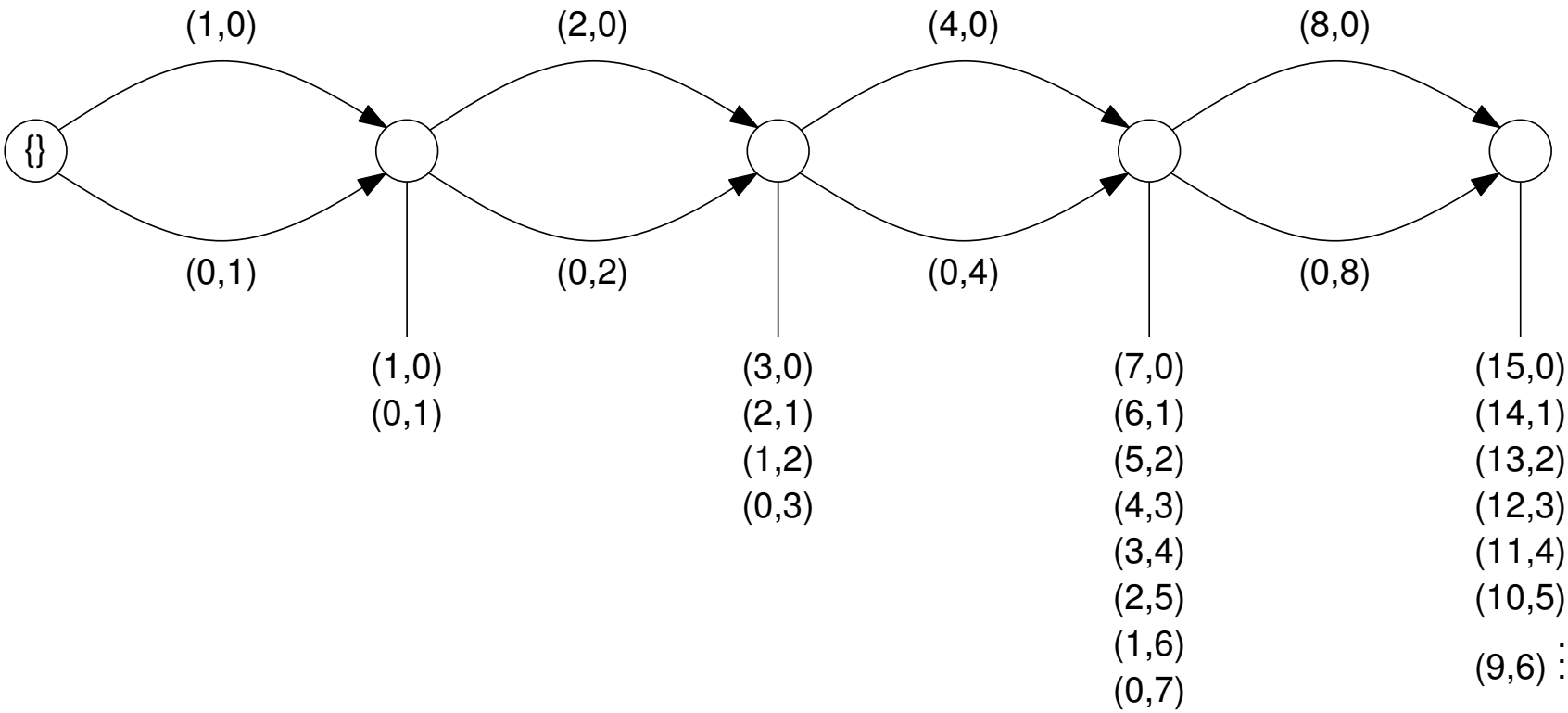
# Exponentielle Laufzeit

Exponentiell wachsende Lösungsmenge bei zwei Kriterien:



# Exponentielle Laufzeit

Exponentiell wachsende Lösungsmenge bei zwei Kriterien:



# Verbesserungen

- Jedes $L_u$ verwaltet bestes ungesettletes Label selbst
  $\Rightarrow$ 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)
  $\Rightarrow$ Überspringe Kante zum Parent-Knoten von $\ell_u$
- **Target-Pruning:**
  Abbruchkriterium funktioniert nicht (sonst nur eine Lösung)
  $\Rightarrow$ An Knoten $u$, verwerfe Label $\ell_u$, wenn es bereits von der tentativen Pareto-Menge $L_t$ am Ziel $t$ dominiert wird

# Verbesserungen

- Jedes $L_u$ verwaltet bestes ungesettletes Label selbst
  $\Rightarrow$ 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)
  $\Rightarrow$ Überspringe Kante zum Parent-Knoten von $\ell_u$
- **Target-Pruning:**
  Abbruchkriterium funktioniert nicht (sonst nur eine Lösung)
  $\Rightarrow$ An Knoten $u$, verwerfe Label $\ell_u$, 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)

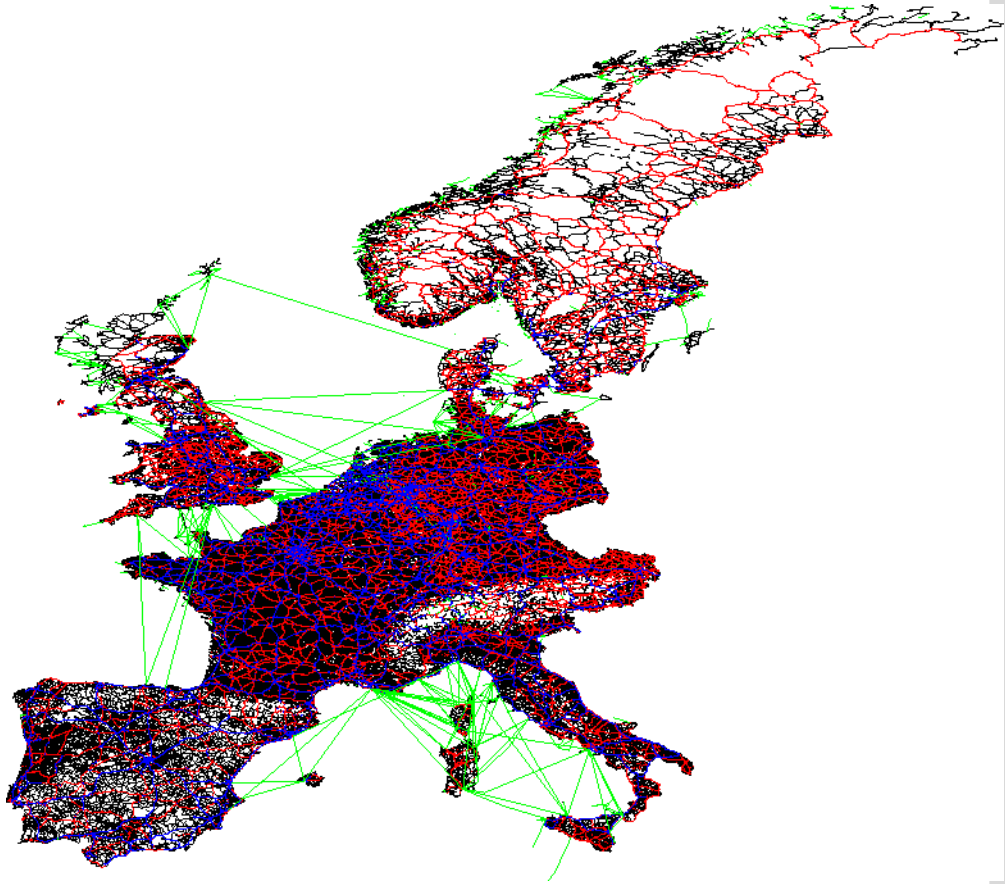
# Eingabe

**Straßengraphen von:**

- Luxemburg
- Karlsruhe
- Europa

**Metriken:**

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



# Ähnliche Metriken: Europa

**Beobachtungen:**

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

| 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 |

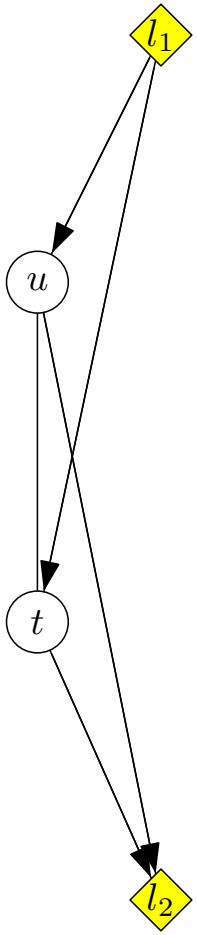
# Verschiedene Metriken

| | 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 |

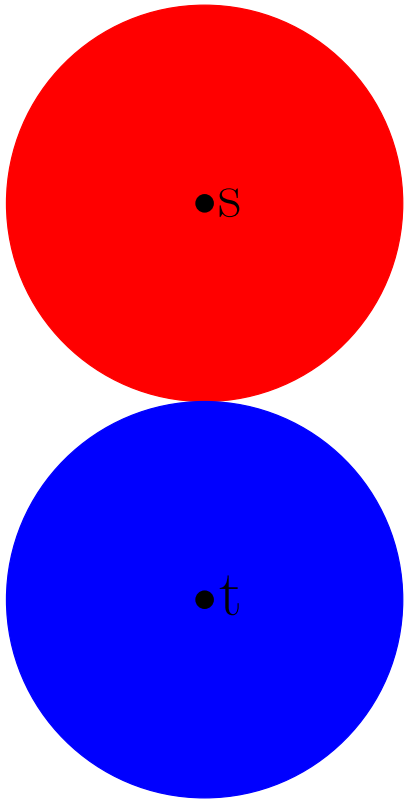
- Je nach Kriterien kann Lösungsmenge stark ansteigen

# Beschleunigungstechniken

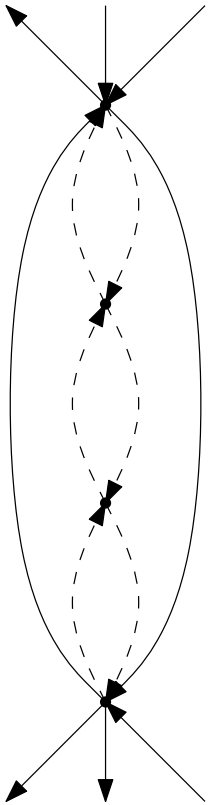
## Landmarken



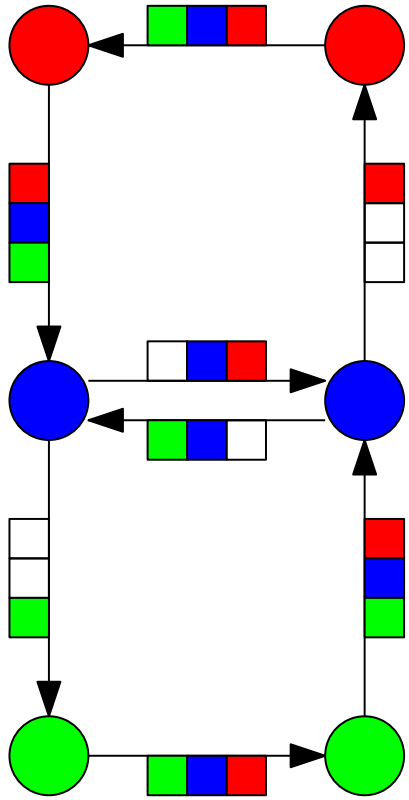

## Bidirektionale Suche



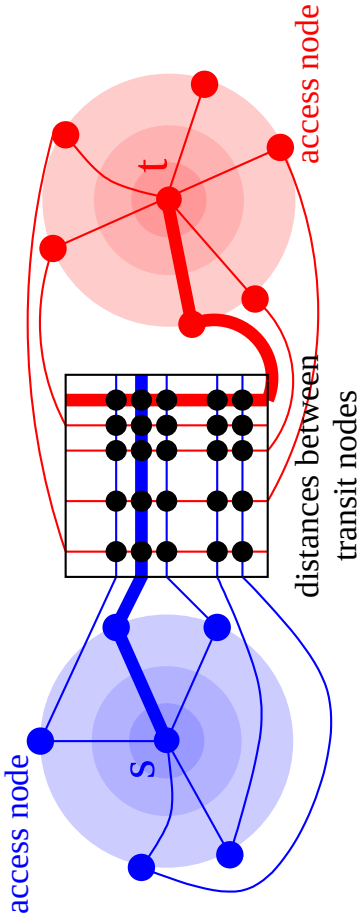

## Kontraktion



## Arc-Flags



## Table-Lookups



# Wdh.: Landmarken

**Vorberechnung:**

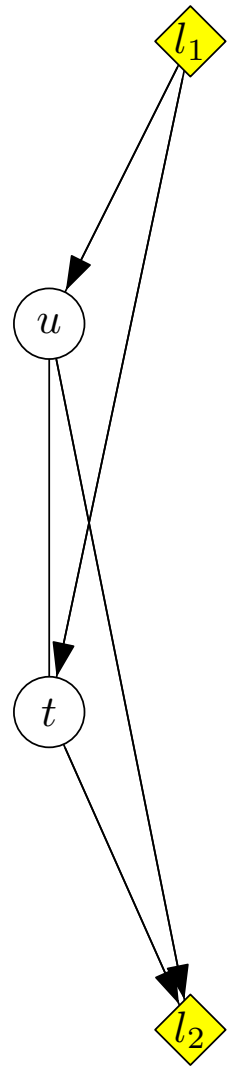
- Wähle einige Knoten ($\approx 16$) als Landmarken
- Berechne Abstände von und zu allen Landmarken

**Anfrage:**

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

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

- Verändert Reihenfolge der besuchten Knoten



# Anpassung

**Idee:**

- Berechne Distanzen pro Metrik unabhängig von einander
- Für jeden Knoten $u$:
  Distanzvektor $(d_1(u, L_i)_1, \ldots, d_n(u, L_i)_n)$ pro Landmarke $L_i$
- Liefert Potentiale $\pi_1, \ldots, \pi_n$
- Zielrichtung: Priorität eines Labels ist $(x_1 + \pi_1, \ldots, x_n + \pi_n)$.
- Potential $\pi_i$ 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:**

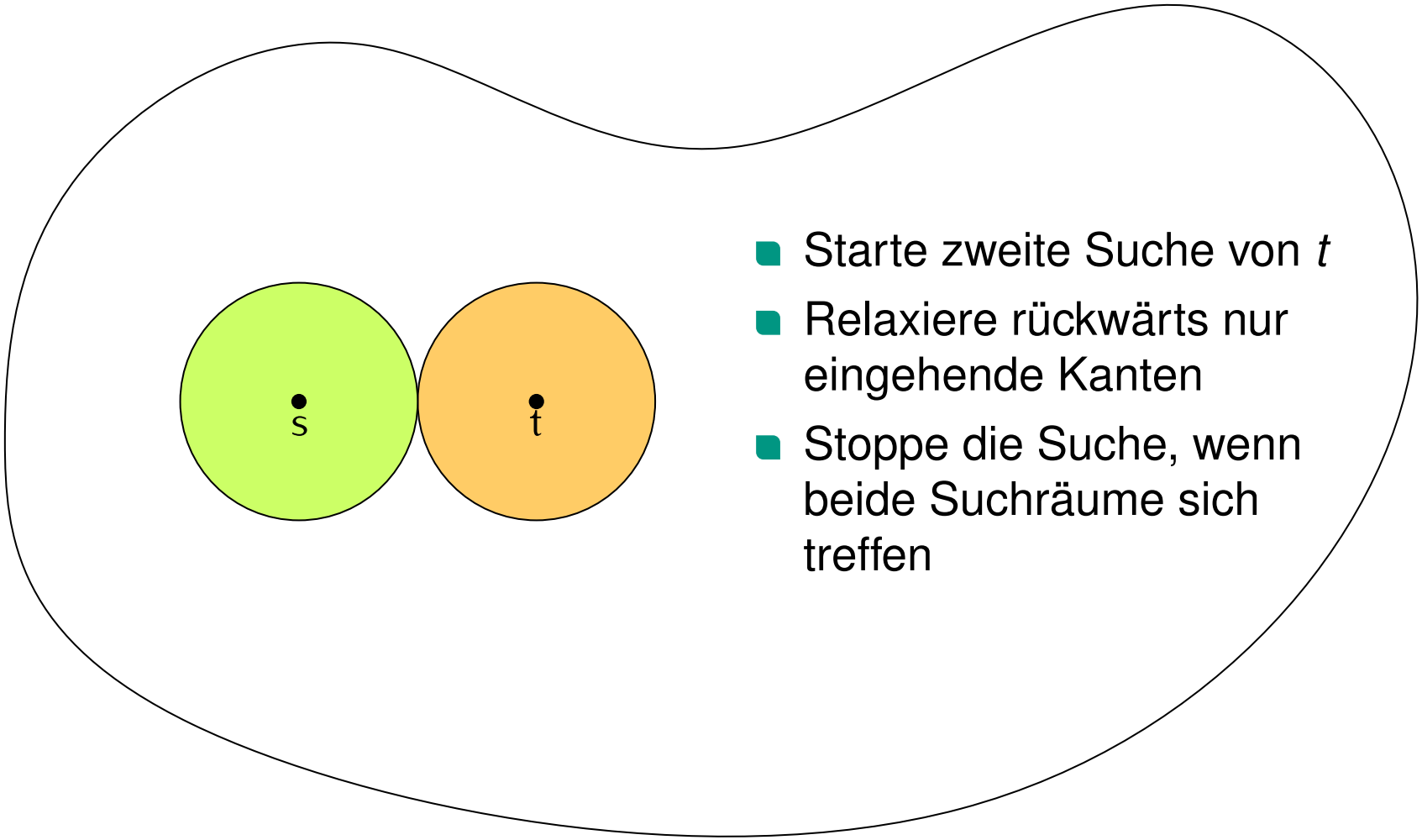
- Berechne Distanzen pro Metrik unabhängig von einander
- Für jeden Knoten $u$:
  Distanzvektor $(d_1(u, L_i)_1, \ldots, d_n(u, L_i)_n)$ pro Landmarke $L_i$
- Liefert Potentiale $\pi_1, \ldots, \pi_n$
- Zielrichtung: Priorität eines Labels ist $(x_1 + \pi_1, \ldots, x_n + \pi_n)$.
- Potential $\pi_i$ 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 $(d_1(u, t)_1, \ldots, d_n(u, t)_n)$

- 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)



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

# Anpassung

**Idee:**

- Rückwärtssuche kein Problem (analog)

**Offenes Problem:**

- Abbruchkriterium?
- Analog Target-Pruning:
  - Dominanztest mit tentativer Pareto-Menge
  - Pareto-Menge ist teuer zu verwalten
- Dominanztests dominieren Laufzeit, nicht Suchraumgröße
- Lohnt nicht recht

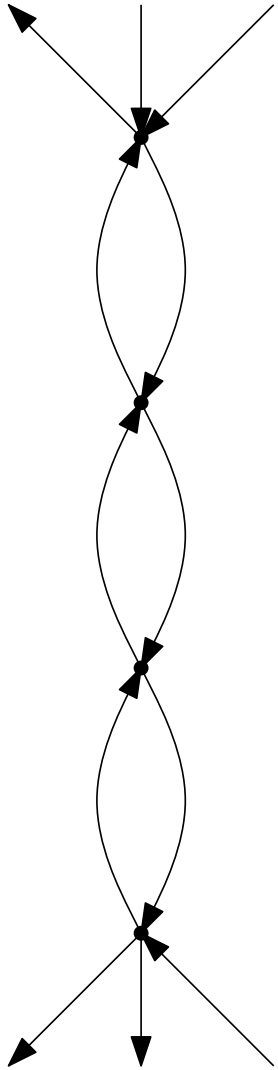
# Kontraktion

**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
- Zeugensuche während oder nach Knoten-reduktion



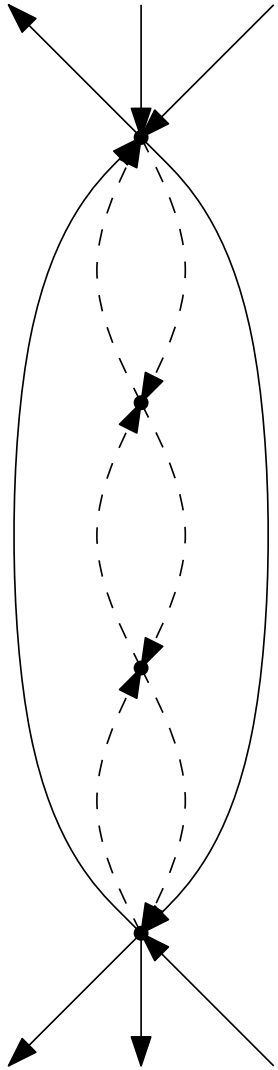
# Kontraktion

## 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
- Zeugensuche während oder nach Knoten-reduktion



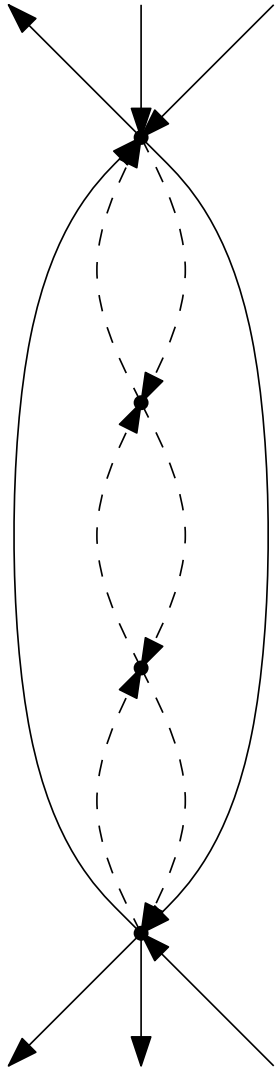
# Anpassung Knoten-Reduktion

**Beobachtung:**

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

**Somit:**

- Anpassung ohne Probleme



# Anpassung Kanten-Reduktion

**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:**

# Anpassung Kanten-Reduktion

**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

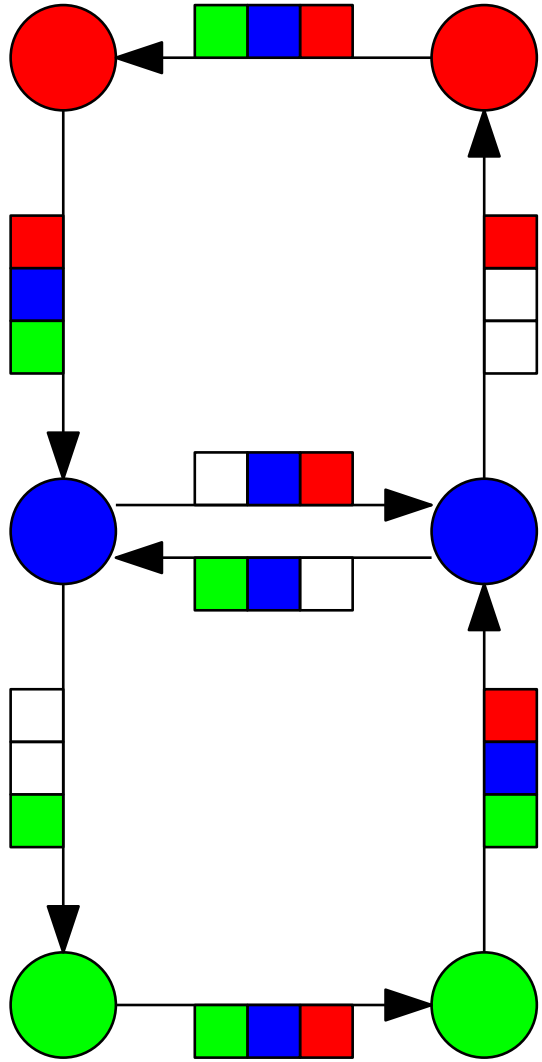
# Arc-Flags

**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 Grahen durchgeführt
- Flaggen müssen allerdings aktualisiert werden



# Anpassung

**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

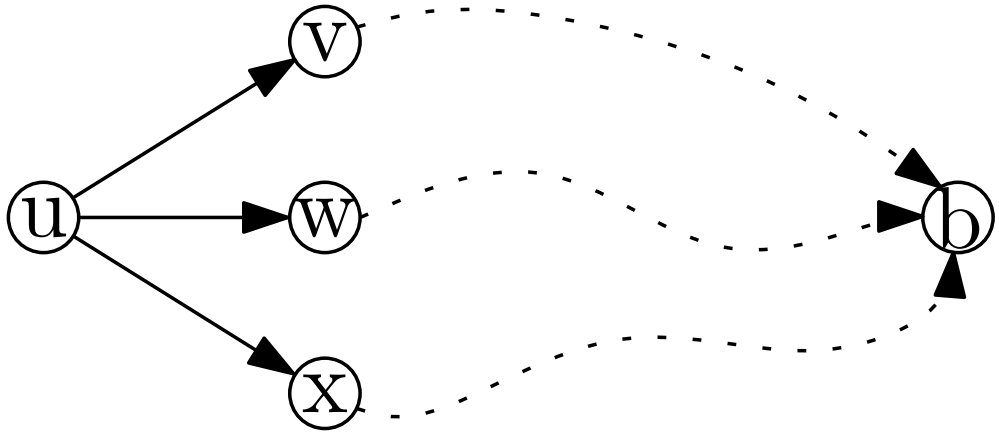
# Anpassung

**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



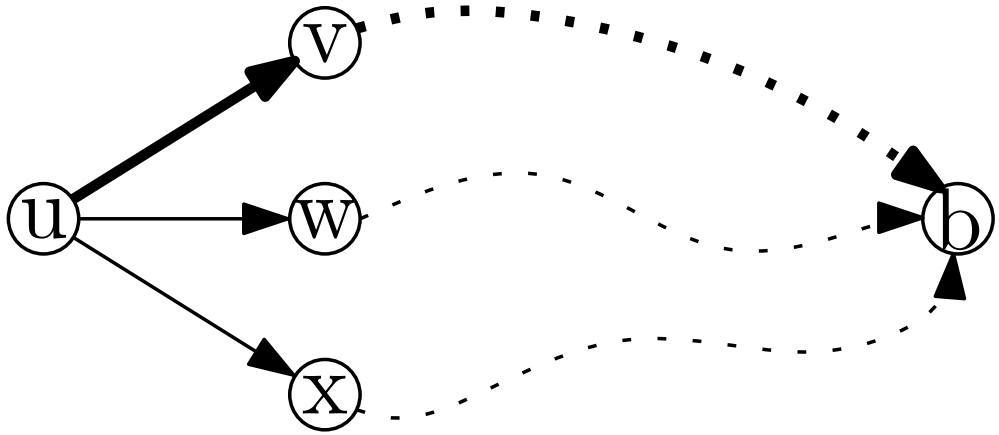
# Anpassung

**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



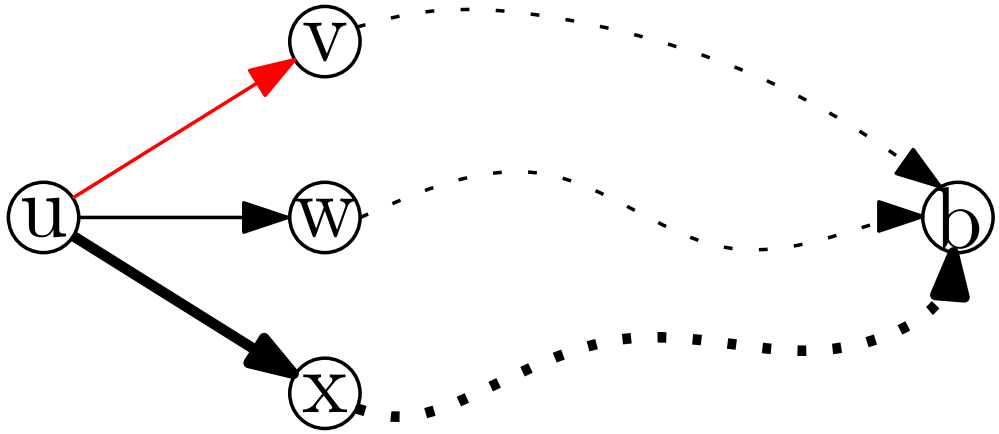
# Anpassung

**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



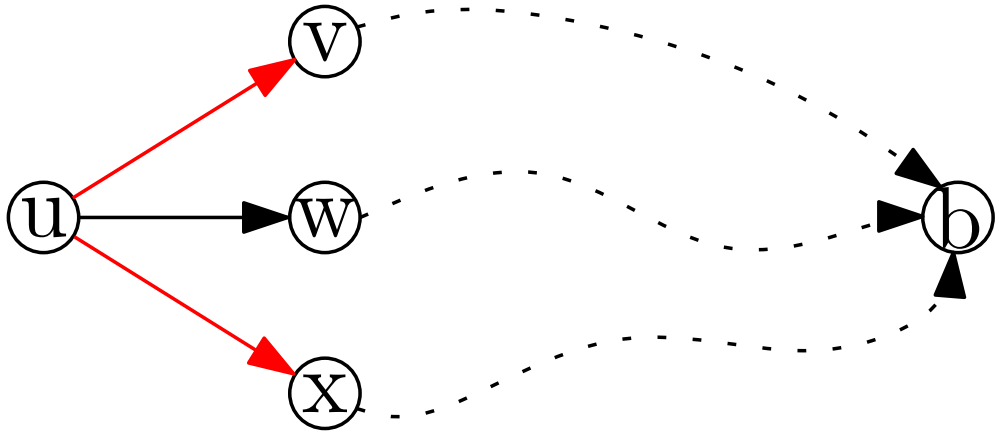
# Anpassung

**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:**

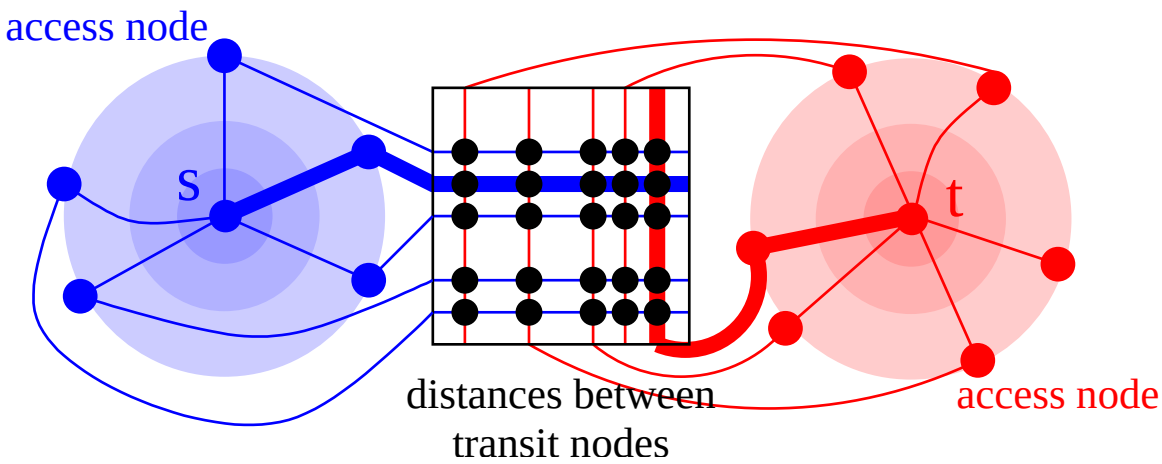
- Für alle Randknoten $b$ und alle Knoten $u$:
- Berechne Pareto-Abstände $D(u, b)$
- Setze Flagge wenn gilt $(u, v)$ zugehörige Kante eines Pareto-Pfades



# Table-Lookups

**Idee:**

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



# Anpassung

**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!

# Diskussion Basismodule

**Basismodule:**

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

# Experimente

## Pareto-SHARC (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...
- Nur konvexe Hülle (Parametric Shortest Path Problem)

# Constrained Shortest Paths



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

# 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?

**Anmerkung:** 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 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

# PARTITION

## Definition: PARTITION

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

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

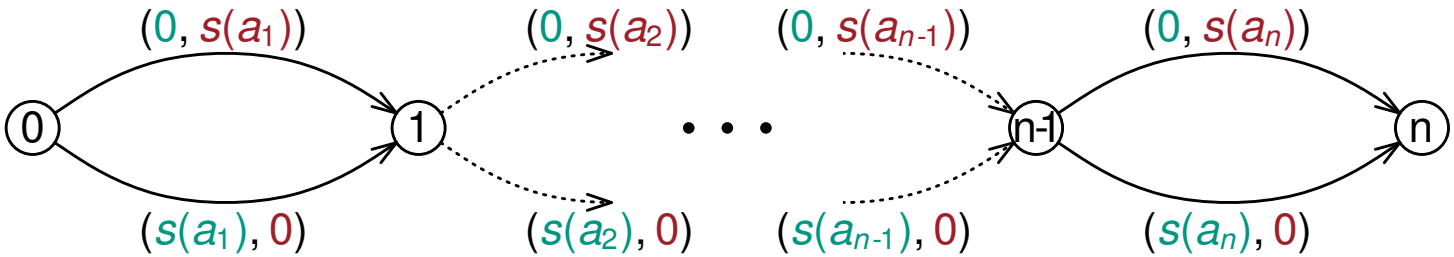
**Anmerkung:**

- 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:

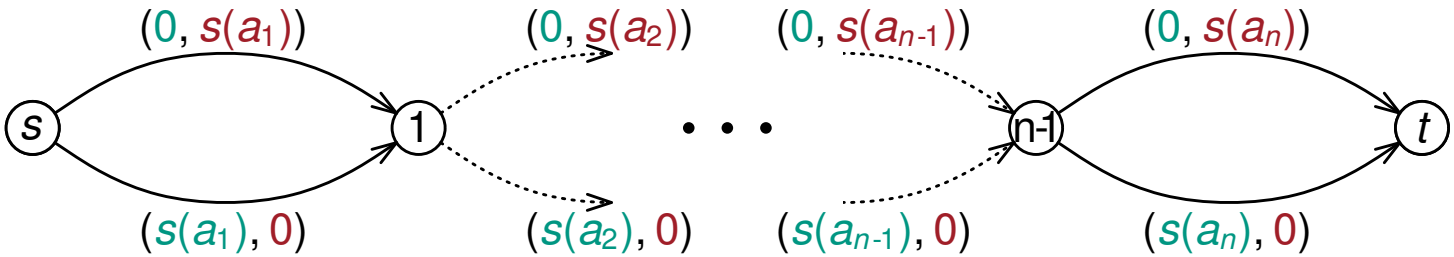
- Sei $\Pi = (A = \{a_1, \ldots, a_n\}, s)$ eine PARTITION-Instanz
- Konstruiere Graph $G = (V, E)$ mit $n + 1$ Knoten ($V = \{0, 1, \ldots, n\}$)
- Knoten $i$-1 und $i$ sind jeweils durch zwei Kanten verbunden
  - Eine Kante hat Länge 0 und Gewicht $s(a_i)$
  - Die andere Kante hat Länge $s(a_i)$ und 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 $\Pi = (A = \{a_1, \ldots, a_n\}, s)$ 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 $s(a_i)$
  - 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:**

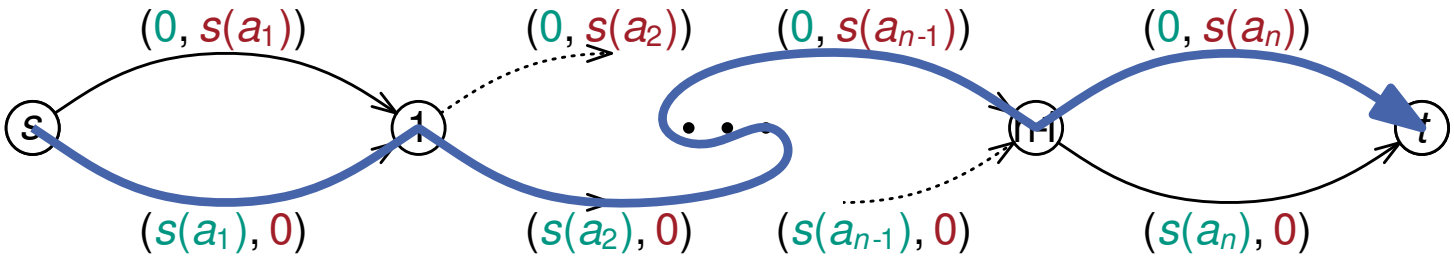
- $\Pi$ ist PARTITION Ja-Inst. $\Leftrightarrow$ $(G, \ell, \omega, L, W, s, t)$ ist CSP Ja-Inst.

□

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

## Reduktion von PARTITION:

- Sei $\Pi = (A = \{a_1, \ldots, a_n\}, s)$ 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 $s(a_i)$
  - 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: $a_1 \notin A', a_2 \notin A', \ldots, a_{n-1} \in A', a_n \in A'$

□

# 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 $> W$

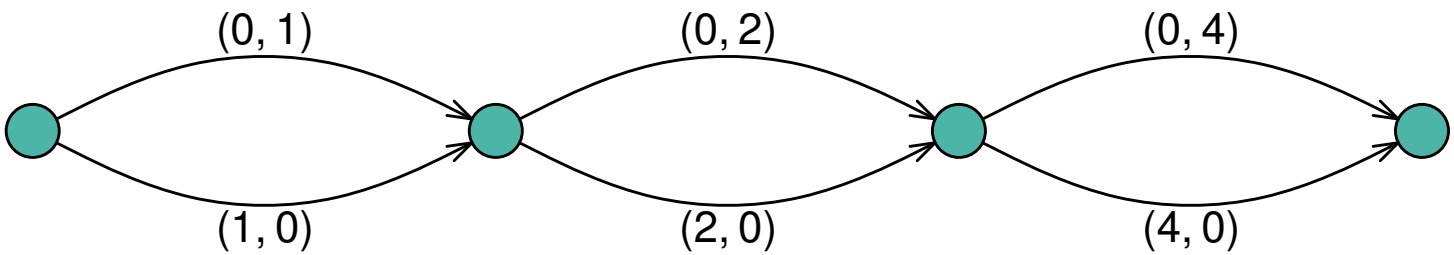
# Multi-Criteria Dijkstra für CSP

**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



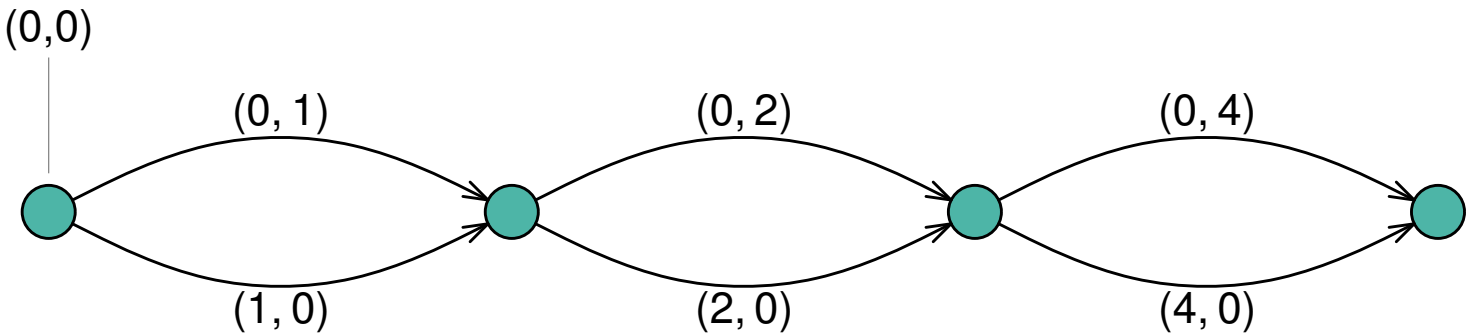
# Multi-Criteria Dijkstra für CSP

**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



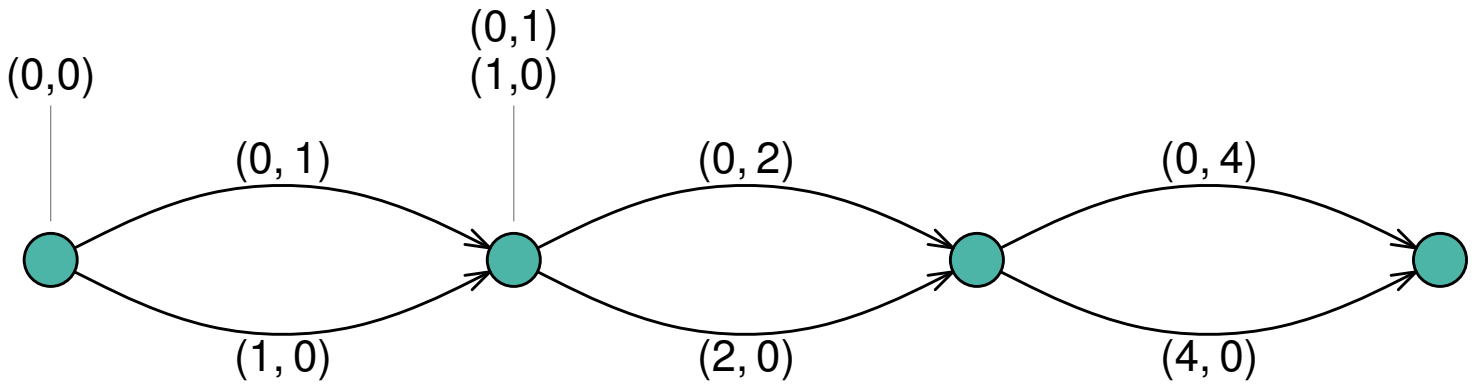
# Multi-Criteria Dijkstra für CSP

**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



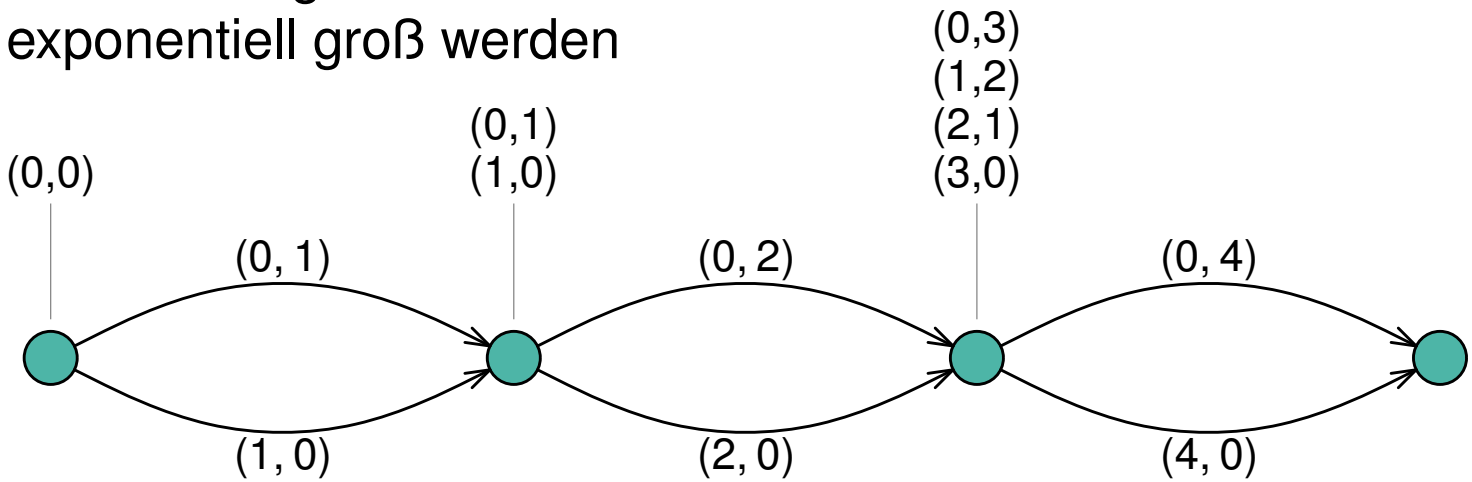
# Multi-Criteria Dijkstra für CSP

**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



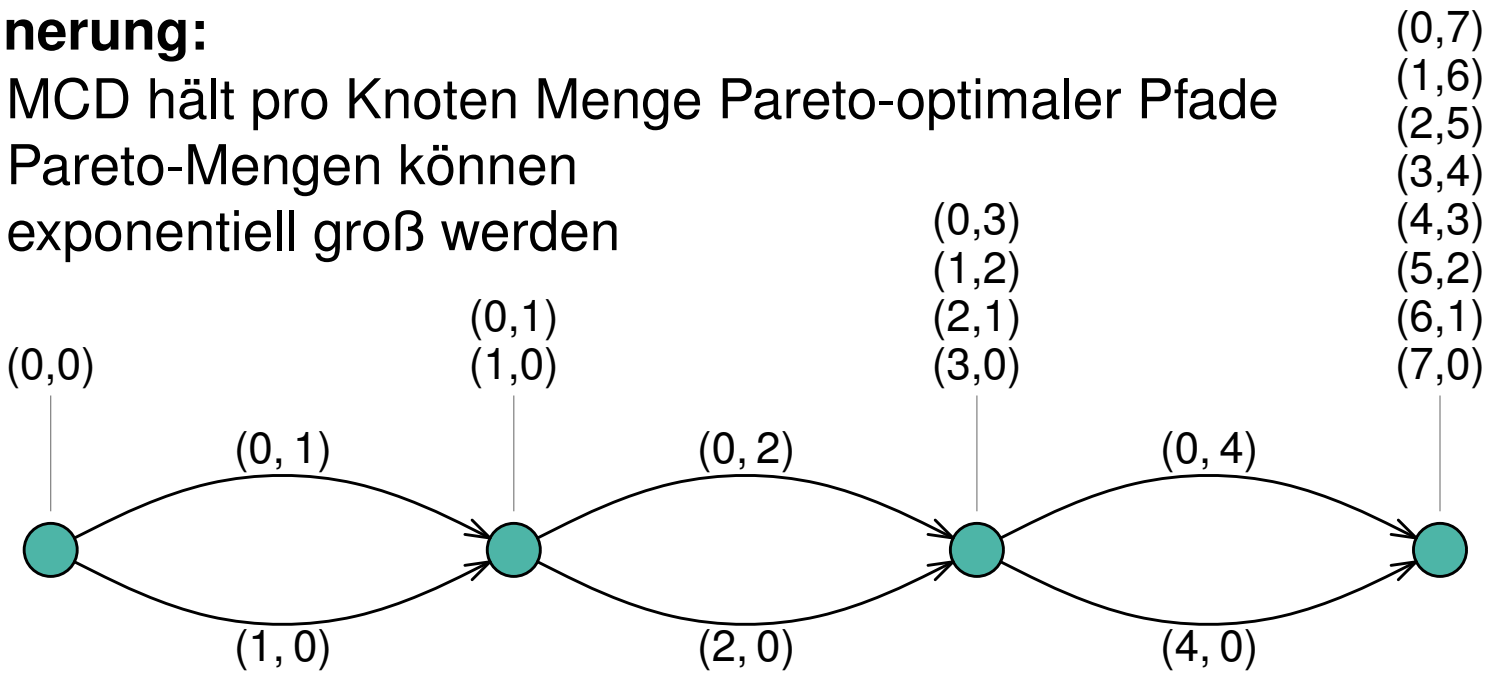
# Multi-Criteria Dijkstra für CSP

**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
  (Queue ist nach Länge sortiert)

# Zusammenfassung

- 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

# Ende

## 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.