Karlsruher Institut für Technologie

Algorithmen für Routenplanung

10. Sitzung, Sommersemester 2015

julian Dibbelt | 27. Mai 2015

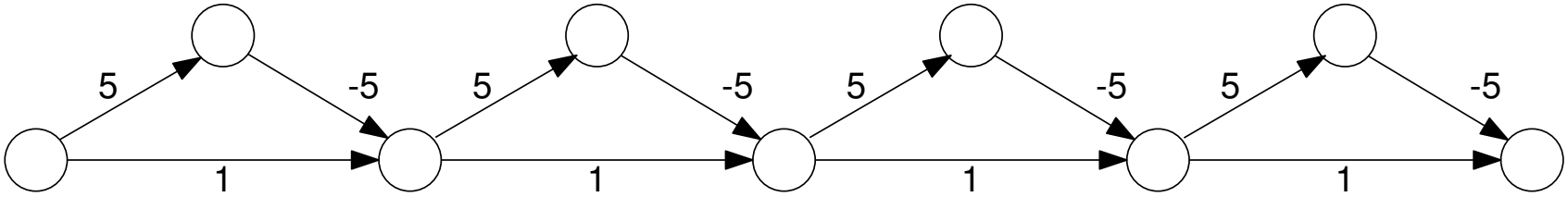
Institut für Theoretische Informatik · Algorithmik · Prof. Dr. Dorothea Wagner

Nachtrag

Einführungsvorlesung: Dijkstra's Algorithmus ist label-setting (bereits abgearbeitete Distanzen werden nicht mehr verbessert; wichtig für die Laufzeitanalyse; wichtig für ALT)

Gibt es Szenarien in denen Dijkstra's Algorithmus label-correcting ist?

Einfaches Beispiel (negative Kantengewichte):

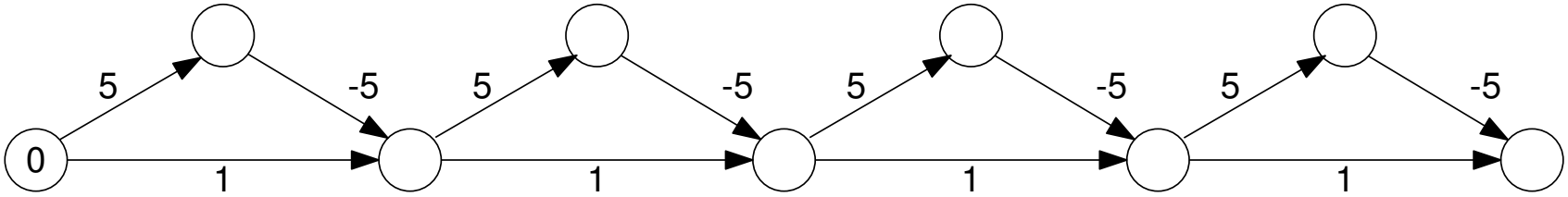


Nachtrag

Einführungsvorlesung: Dijkstra's Algorithmus ist label-setting (bereits abgearbeitete Distanzen werden nicht mehr verbessert; wichtig für die Laufzeitanalyse; wichtig für ALT)

Gibt es Szenarien in denen Dijkstra's Algorithmus label-correcting ist?

Einfaches Beispiel (negative Kantengewichte):

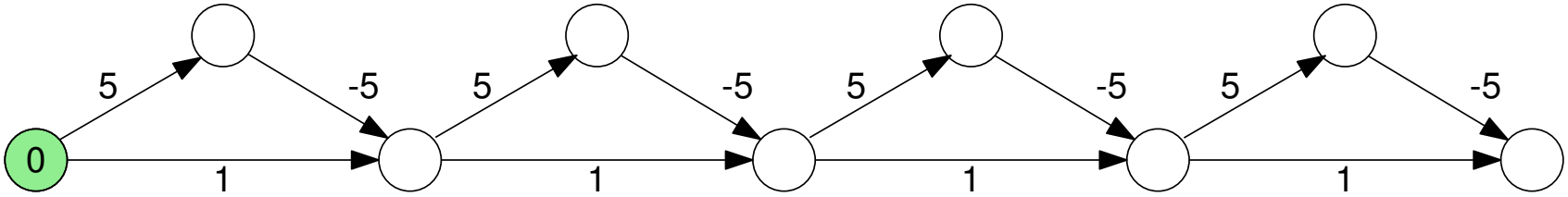


Nachtrag

Einführungsvorlesung: Dijkstra's Algorithmus ist label-setting (bereits abgearbeitete Distanzen werden nicht mehr verbessert; wichtig für die Laufzeitanalyse; wichtig für ALT)

Gibt es Szenarien in denen Dijkstra's Algorithmus label-correcting ist?

Einfaches Beispiel (negative Kantengewichte):

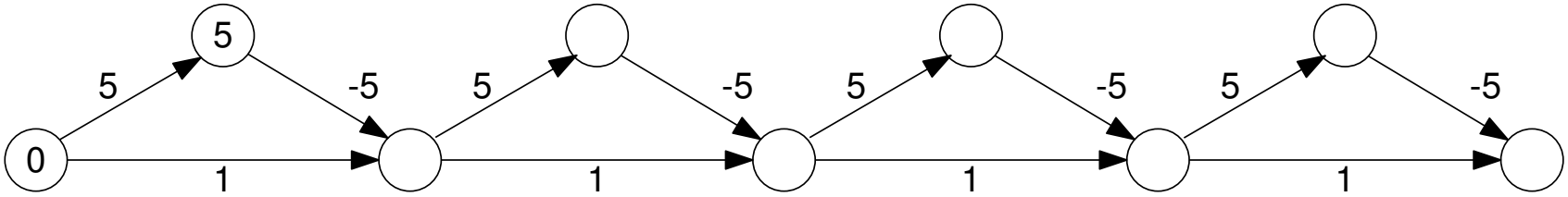


Nachtrag

Einführungsvorlesung: Dijkstra's Algorithmus ist label-setting (bereits abgearbeitete Distanzen werden nicht mehr verbessert; wichtig für die Laufzeitanalyse; wichtig für ALT)

Gibt es Szenarien in denen Dijkstra's Algorithmus label-correcting ist?

Einfaches Beispiel (negative Kantengewichte):

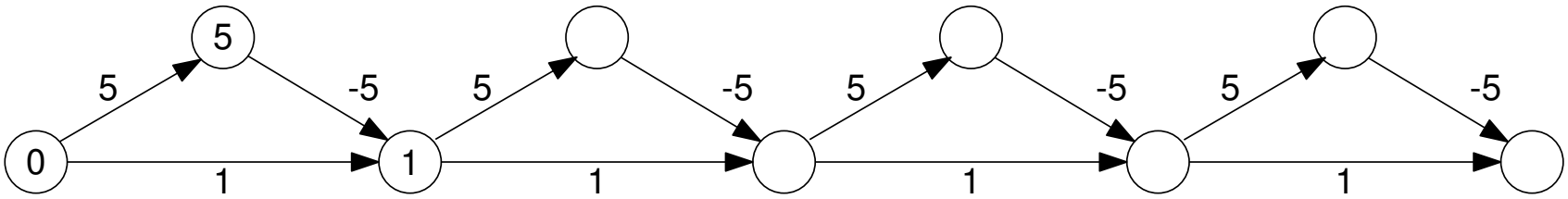


Nachtrag

Einführungsvorlesung: Dijkstra's Algorithmus ist label-setting (bereits abgearbeitete Distanzen werden nicht mehr verbessert; wichtig für die Laufzeitanalyse; wichtig für ALT)

Gibt es Szenarien in denen Dijkstra's Algorithmus label-correcting ist?

Einfaches Beispiel (negative Kantengewichte):

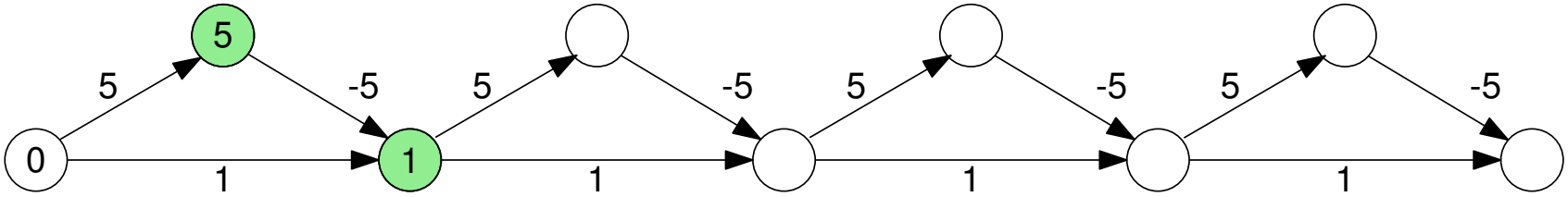


Nachtrag

Einführungsvorlesung: Dijkstra's Algorithmus ist label-setting (bereits abgearbeitete Distanzen werden nicht mehr verbessert; wichtig für die Laufzeitanalyse; wichtig für ALT)

Gibt es Szenarien in denen Dijkstra's Algorithmus label-correcting ist?

Einfaches Beispiel (negative Kantengewichte):

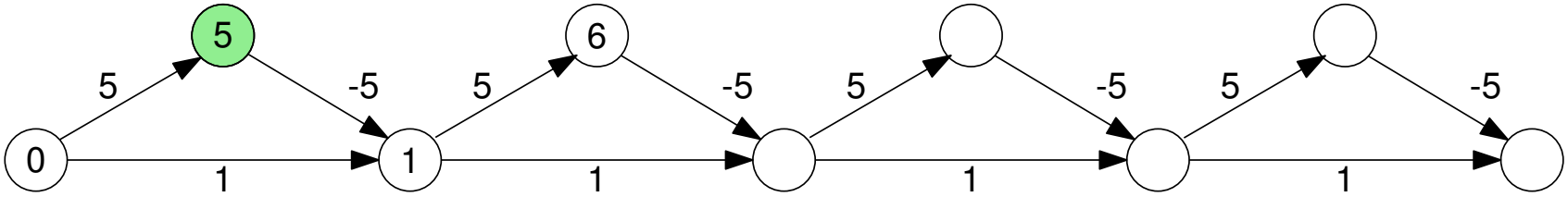


Nachtrag

Einführungsvorlesung: Dijkstra's Algorithmus ist label-setting (bereits abgearbeitete Distanzen werden nicht mehr verbessert; wichtig für die Laufzeitanalyse; wichtig für ALT)

Gibt es Szenarien in denen Dijkstra's Algorithmus label-correcting ist?

Einfaches Beispiel (negative Kantengewichte):

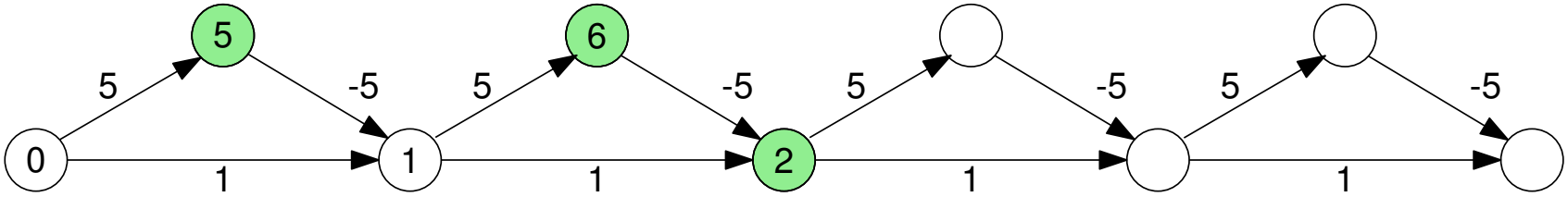


Nachtrag

Einführungsvorlesung: Dijkstra's Algorithmus ist label-setting (bereits abgearbeitete Distanzen werden nicht mehr verbessert; wichtig für die Laufzeitanalyse; wichtig für ALT)

Gibt es Szenarien in denen Dijkstra's Algorithmus label-correcting ist?

Einfaches Beispiel (negative Kantengewichte):

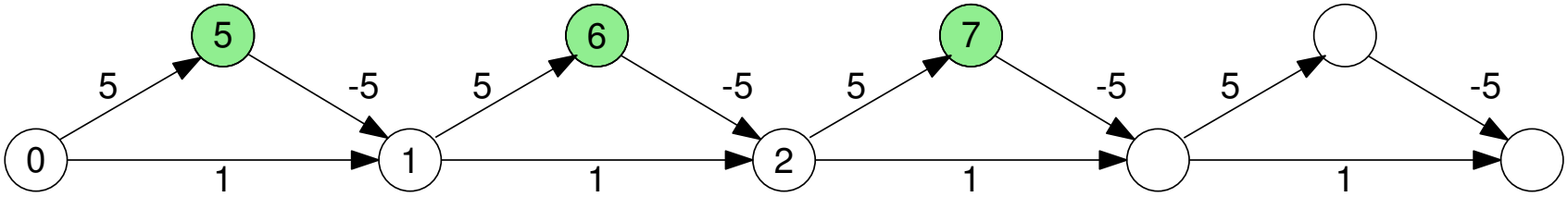


Nachtrag

Einführungsvorlesung: Dijkstra's Algorithmus ist label-setting (bereits abgearbeitete Distanzen werden nicht mehr verbessert; wichtig für die Laufzeitanalyse; wichtig für ALT)

Gibt es Szenarien in denen Dijkstra's Algorithmus label-correcting ist?

Einfaches Beispiel (negative Kantengewichte):

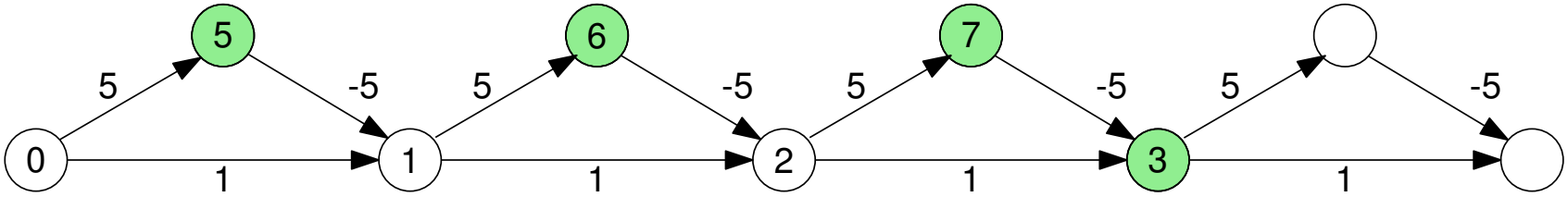


Nachtrag

Einführungsvorlesung: Dijkstra's Algorithmus ist label-setting (bereits abgearbeitete Distanzen werden nicht mehr verbessert; wichtig für die Laufzeitanalyse; wichtig für ALT)

Gibt es Szenarien in denen Dijkstra's Algorithmus label-correcting ist?

Einfaches Beispiel (negative Kantengewichte):

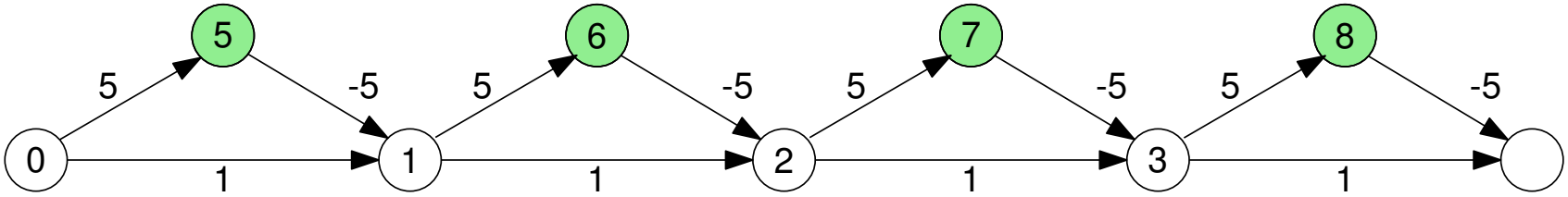


Nachtrag

Einführungsvorlesung: Dijkstra's Algorithmus ist label-setting (bereits abgearbeitete Distanzen werden nicht mehr verbessert; wichtig für die Laufzeitanalyse; wichtig für ALT)

Gibt es Szenarien in denen Dijkstra's Algorithmus label-correcting ist?

Einfaches Beispiel (negative Kantengewichte):



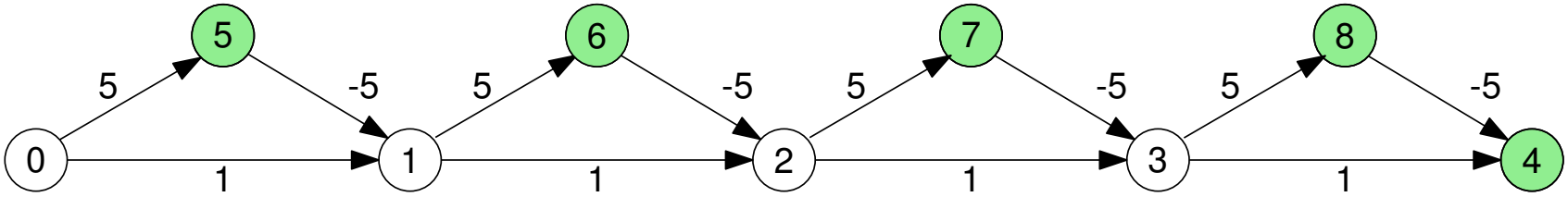
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Nachtrag

Einführungsvorlesung: Dijkstra's Algorithmus ist label-setting (bereits abgearbeitete Distanzen werden nicht mehr verbessert; wichtig für die Laufzeitanalyse; wichtig für ALT)

Gibt es Szenarien in denen Dijkstra's Algorithmus label-correcting ist?

Einfaches Beispiel (negative Kantengewichte):



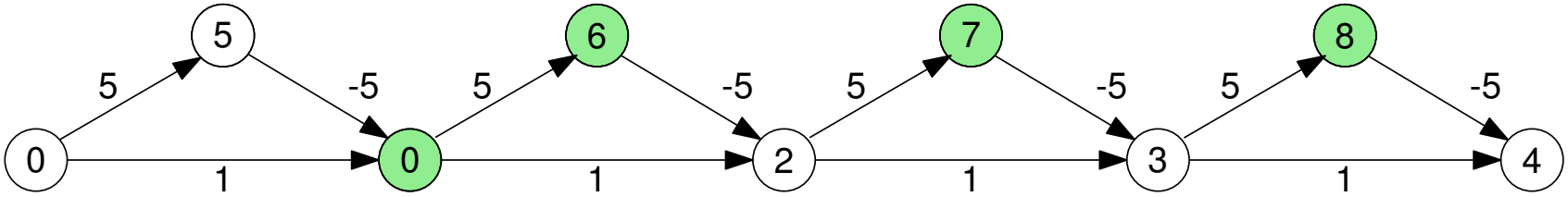
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Nachtrag

Einführungsvorlesung: Dijkstra's Algorithmus ist label-setting (bereits abgearbeitete Distanzen werden nicht mehr verbessert; wichtig für die Laufzeitanalyse; wichtig für ALT)

Gibt es Szenarien in denen Dijkstra's Algorithmus label-correcting ist?

Einfaches Beispiel (negative Kantengewichte):

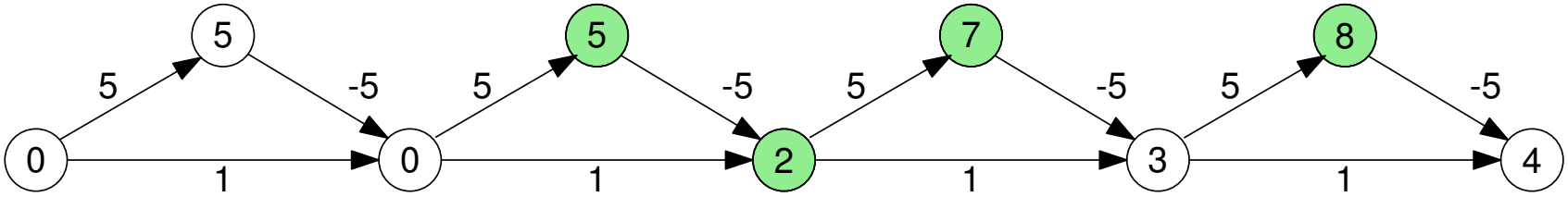


Nachtrag

Einführungsvorlesung: Dijkstra's Algorithmus ist label-setting (bereits abgearbeitete Distanzen werden nicht mehr verbessert; wichtig für die Laufzeitanalyse; wichtig für ALT)

Gibt es Szenarien in denen Dijkstra's Algorithmus label-correcting ist?

Einfaches Beispiel (negative Kantengewichte):

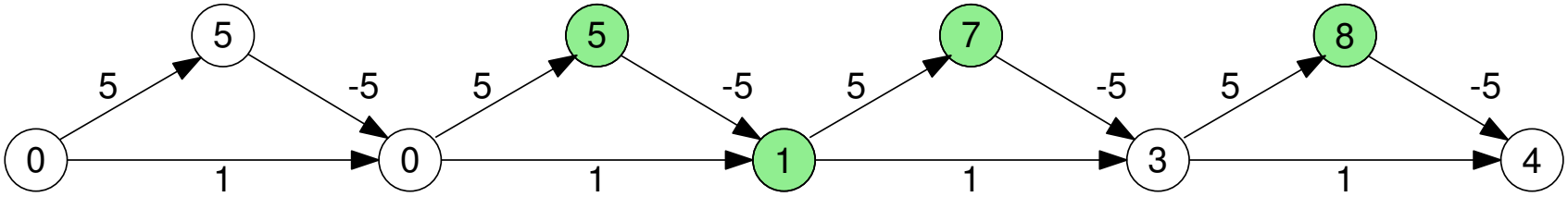


Nachtrag

Einführungsvorlesung: Dijkstra's Algorithmus ist label-setting (bereits abgearbeitete Distanzen werden nicht mehr verbessert; wichtig für die Laufzeitanalyse; wichtig für ALT)

Gibt es Szenarien in denen Dijkstra's Algorithmus label-correcting ist?

Einfaches Beispiel (negative Kantengewichte):

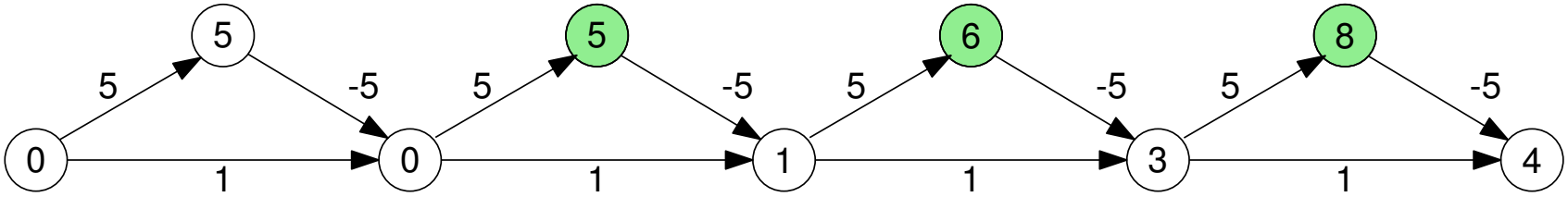


Nachtrag

Einführungsvorlesung: Dijkstra's Algorithmus ist label-setting (bereits abgearbeitete Distanzen werden nicht mehr verbessert; wichtig für die Laufzeitanalyse; wichtig für ALT)

Gibt es Szenarien in denen Dijkstra's Algorithmus label-correcting ist?

Einfaches Beispiel (negative Kantengewichte):



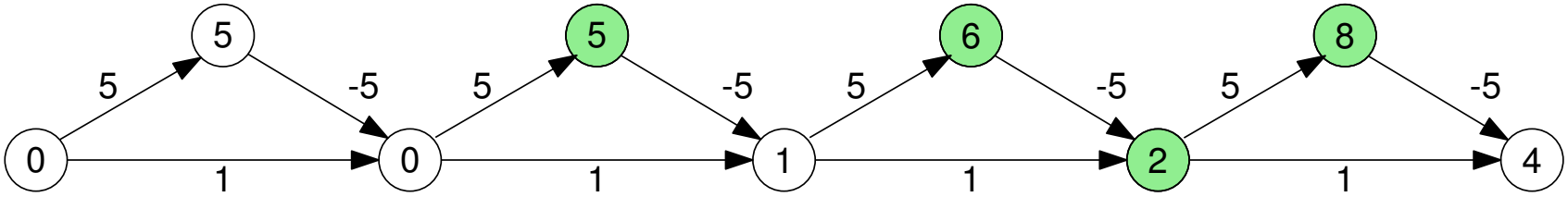
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Nachtrag

Einführungsvorlesung: Dijkstra's Algorithmus ist label-setting (bereits abgearbeitete Distanzen werden nicht mehr verbessert; wichtig für die Laufzeitanalyse; wichtig für ALT)

Gibt es Szenarien in denen Dijkstra's Algorithmus label-correcting ist?

Einfaches Beispiel (negative Kantengewichte):



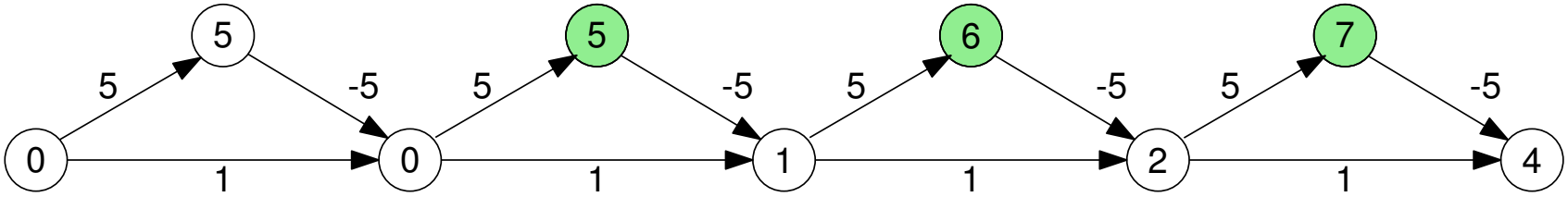
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Nachtrag

Einführungsvorlesung: Dijkstra's Algorithmus ist label-setting (bereits abgearbeitete Distanzen werden nicht mehr verbessert; wichtig für die Laufzeitanalyse; wichtig für ALT)

Gibt es Szenarien in denen Dijkstra's Algorithmus label-correcting ist?

Einfaches Beispiel (negative Kantengewichte):

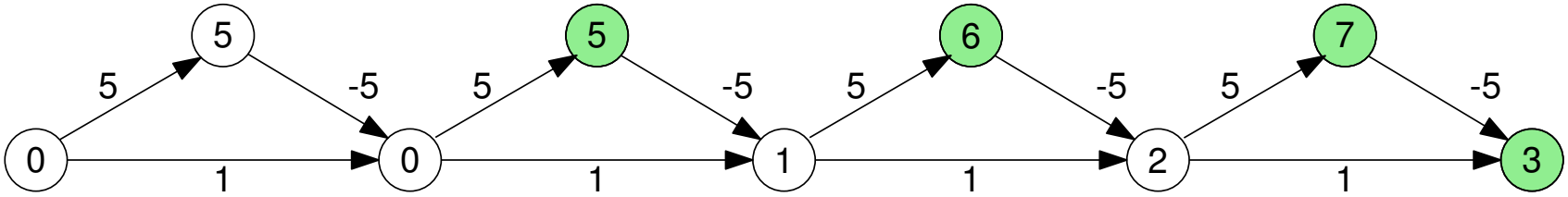


Nachtrag

Einführungsvorlesung: Dijkstra's Algorithmus ist label-setting (bereits abgearbeitete Distanzen werden nicht mehr verbessert; wichtig für die Laufzeitanalyse; wichtig für ALT)

Gibt es Szenarien in denen Dijkstra's Algorithmus label-correcting ist?

Einfaches Beispiel (negative Kantengewichte):



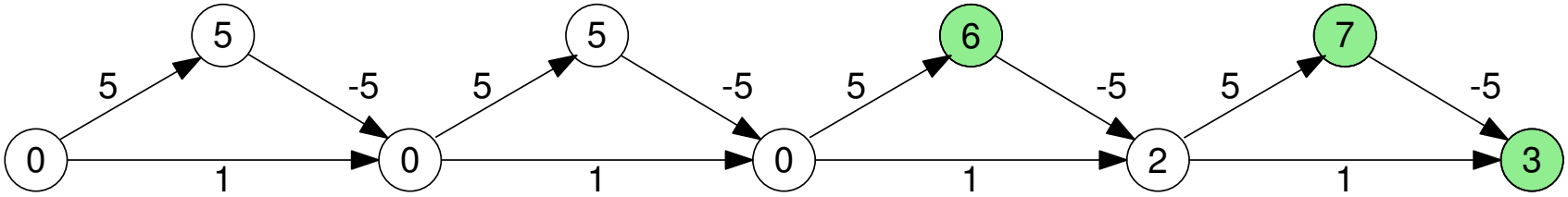
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Nachtrag

Einführungsvorlesung: Dijkstra's Algorithmus ist label-setting (bereits abgearbeitete Distanzen werden nicht mehr verbessert; wichtig für die Laufzeitanalyse; wichtig für ALT)

Gibt es Szenarien in denen Dijkstra's Algorithmus label-correcting ist?

Einfaches Beispiel (negative Kantengewichte):

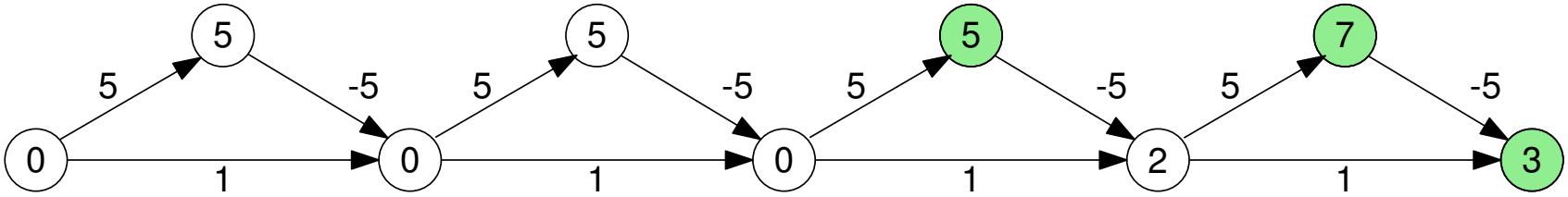


Nachtrag

Einführungsvorlesung: Dijkstra's Algorithmus ist label-setting (bereits abgearbeitete Distanzen werden nicht mehr verbessert; wichtig für die Laufzeitanalyse; wichtig für ALT)

Gibt es Szenarien in denen Dijkstra's Algorithmus label-correcting ist?

Einfaches Beispiel (negative Kantengewichte):

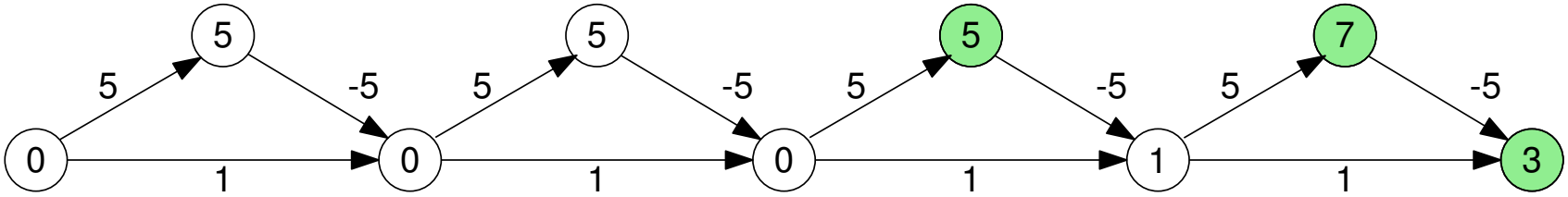


Nachtrag

Einführungsvorlesung: Dijkstra's Algorithmus ist label-setting (bereits abgearbeitete Distanzen werden nicht mehr verbessert; wichtig für die Laufzeitanalyse; wichtig für ALT)

Gibt es Szenarien in denen Dijkstra's Algorithmus label-correcting ist?

Einfaches Beispiel (negative Kantengewichte):



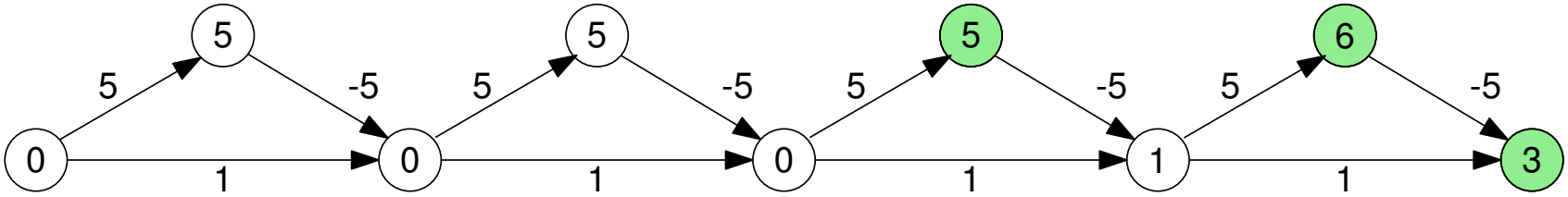
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Nachtrag

Einführungsvorlesung: Dijkstra's Algorithmus ist label-setting (bereits abgearbeitete Distanzen werden nicht mehr verbessert; wichtig für die Laufzeitanalyse; wichtig für ALT)

Gibt es Szenarien in denen Dijkstra's Algorithmus label-correcting ist?

Einfaches Beispiel (negative Kantengewichte):

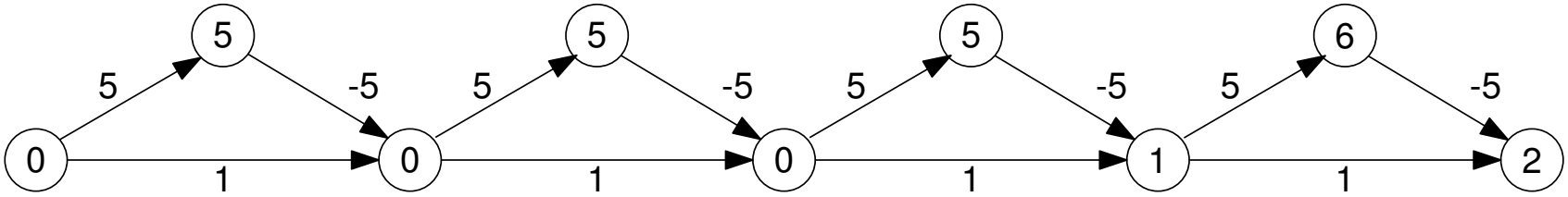


Nachtrag

Einführungsvorlesung: Dijkstra's Algorithmus ist label-setting (bereits abgearbeitete Distanzen werden nicht mehr verbessert; wichtig für die Laufzeitanalyse; wichtig für ALT)

Gibt es Szenarien in denen Dijkstra's Algorithmus label-correcting ist?

Einfaches Beispiel (negative Kantengewichte):

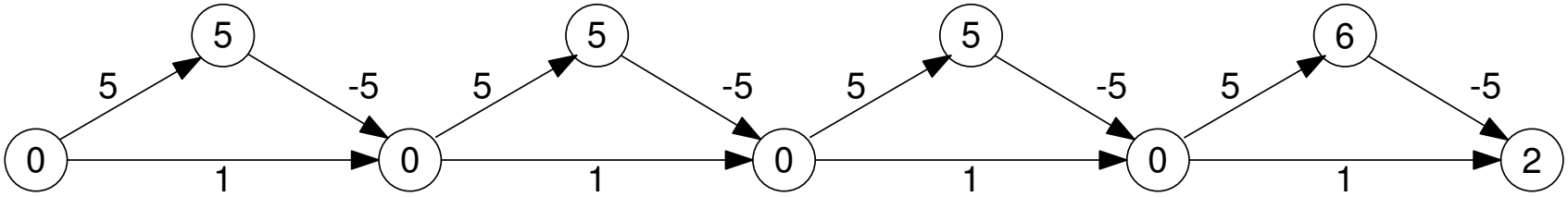


Nachtrag

Einführungsvorlesung: Dijkstra's Algorithmus ist label-setting (bereits abgearbeitete Distanzen werden nicht mehr verbessert; wichtig für die Laufzeitanalyse; wichtig für ALT)

Gibt es Szenarien in denen Dijkstra's Algorithmus label-correcting ist?

Einfaches Beispiel (negative Kantengewichte):

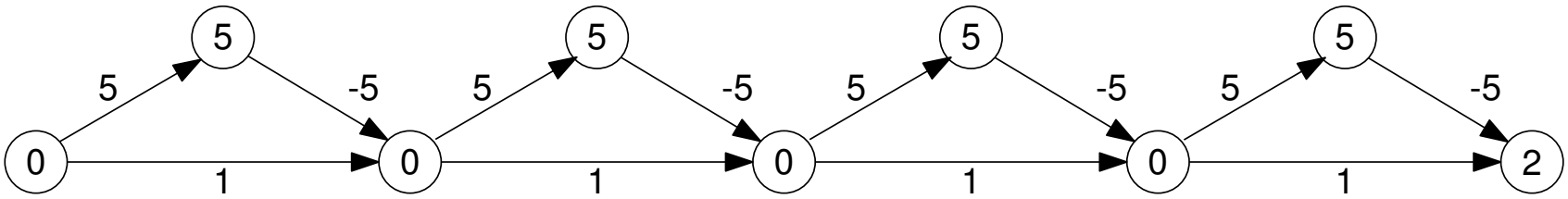


Nachtrag

Einführungsvorlesung: Dijkstra's Algorithmus ist label-setting (bereits abgearbeitete Distanzen werden nicht mehr verbessert; wichtig für die Laufzeitanalyse; wichtig für ALT)

Gibt es Szenarien in denen Dijkstra's Algorithmus label-correcting ist?

Einfaches Beispiel (negative Kantengewichte):



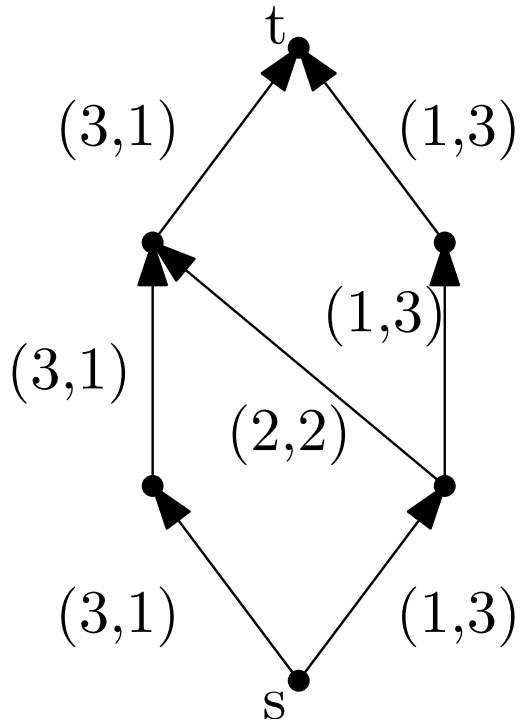
Optimierungskriterien

- Kürzester Weg (eine Metrik)
- Alternative (eine Metrik; fast kürzeste, dennoch sinnvolle Wege)
- jetzt: mehrere Metriken

Multikriterielle Optimierung

Idee:

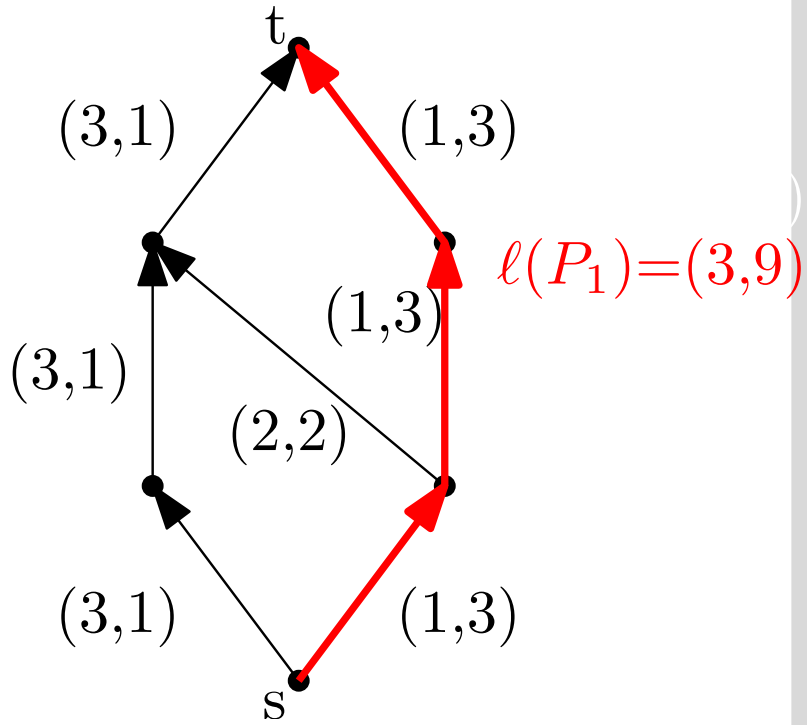
- Mehrere Gewichte an Kanten (z. B. Reisezeiten, Kosten)



Multikriterielle Optimierung

Idee:

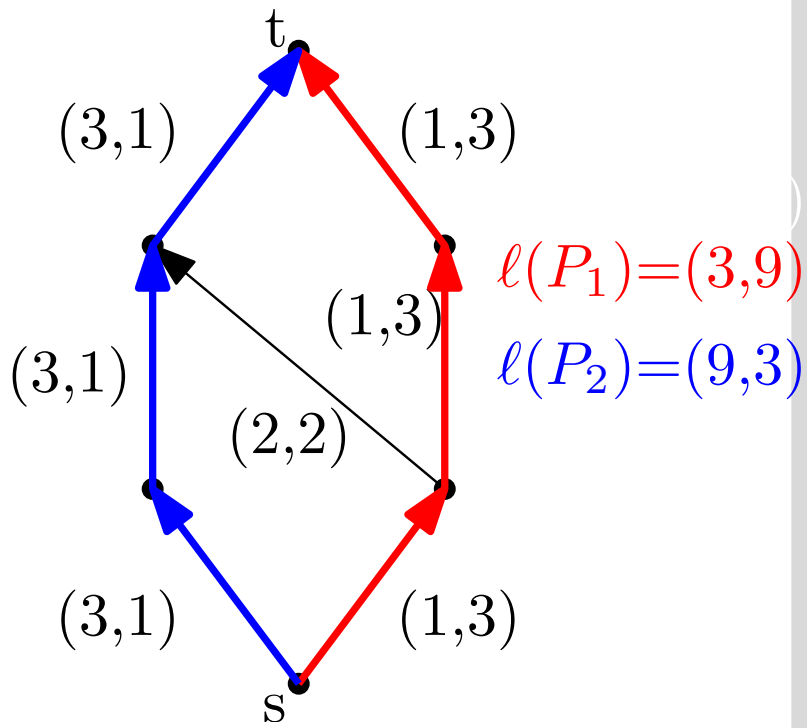
- Mehrere Gewichte an Kanten (z. B. Reisezeiten, Kosten)
- Berechne alle Pareto-optimalen Routen
  - Route ist Pareto-optimal, wenn nicht von anderen Routen dominiert
  - Route dominiert andere, wenn sie sie hinsichtlich jedes Kriteriums ersetzen kann
  - Grenzfall: Gleich gut in allen Kriterien (wird unterschiedlich behandelt)



Multikriterielle Optimierung

Idee:

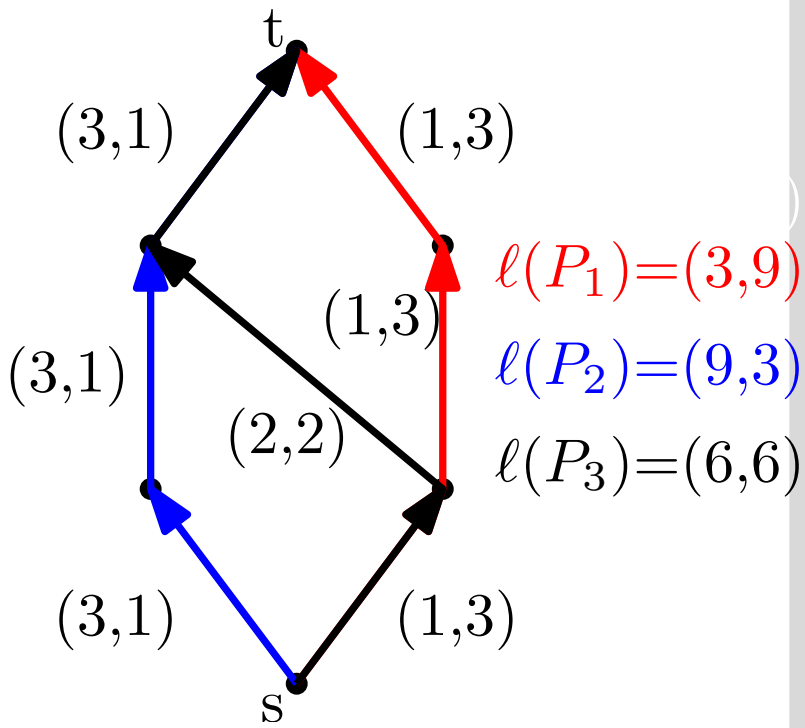
- Mehrere Gewichte an Kanten (z. B. Reisezeiten, Kosten)
- Berechne alle Pareto-optimalen Routen
  - Route ist Pareto-optimal, wenn nicht von anderen Routen dominiert
  - Route dominiert andere, wenn sie sie hinsichtlich jedes Kriteriums ersetzen kann
  - Grenzfall: Gleich gut in allen Kriterien (wird unterschiedlich behandelt)



Multikriterielle Optimierung

Idee:

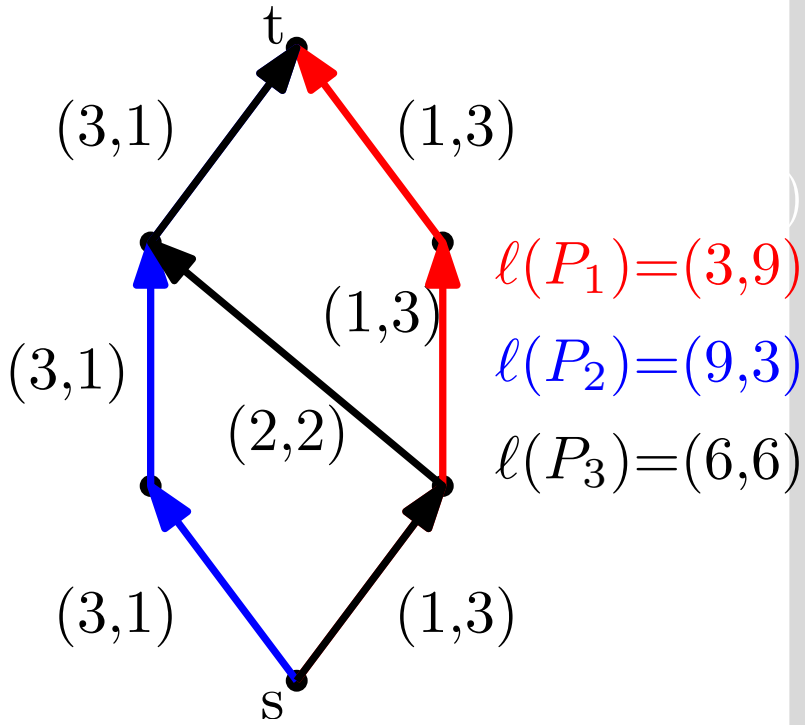
- Mehrere Gewichte an Kanten (z. B. Reisezeiten, Kosten)
- Berechne alle Pareto-optimalen Routen
  - Route ist Pareto-optimal, wenn nicht von anderen Routen dominiert
  - Route dominiert andere, wenn sie sie hinsichtlich jedes Kriteriums ersetzen kann
  - Grenzfall: Gleich gut in allen Kriterien (wird unterschiedlich behandelt)



Multikriterielle Optimierung

Idee:

- Mehrere Gewichte an Kanten (z. B. Reisezeiten, Kosten)
- Berechne alle Pareto-optimalen Routen
  - Route ist Pareto-optimal, wenn nicht von anderen Routen dominiert
  - Route dominiert andere, wenn sie sie hinsichtlich jedes Kriteriums ersetzen kann
  - 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 $\exists l : y_l < x_l$.

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)

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

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 ℓ 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)

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

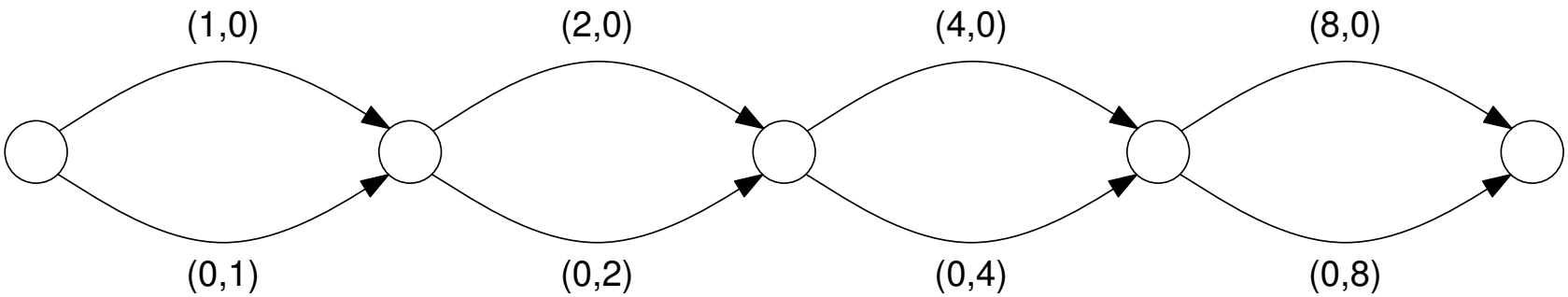
```
1  L_u ← ∅ for each u ∈ V; L_s ← {(0, ..., 0)}
2  Q.clear(); Q.insert(s, k(0, ..., 0))
3  while !Q.empty() do
4      u and ℓ = (x_1, ..., x_n) ← Q.deleteMin()
5      for all edges e = (u, v) ∈ E do
6          ℓ' ← (x_1 + len(e)_1, ..., x_n + len(e)_n)
7          if ℓ' is not dominated by any ℓ''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

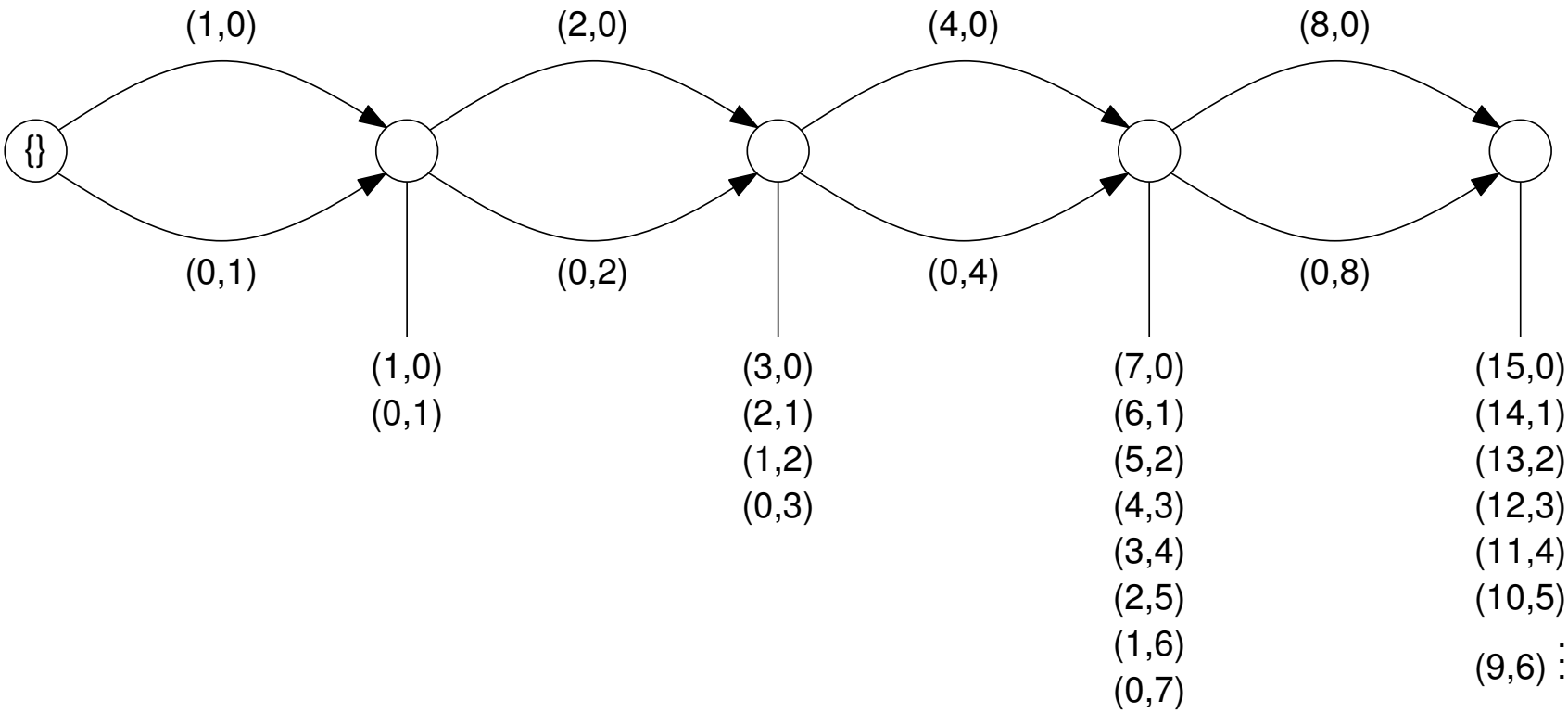
Exponentielle Laufzeit

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 ℓ_u
  (Variante: zum Parent-Knoten von u, wenn eindeutig)
- **Target-Pruning:**
  Abbruchkriterium funktioniert nicht (sonst nur eine Lösung)
  $\Rightarrow$ An Knoten u, verwerfe Label ℓ_u, wenn es bereits von der tentative Pareto-Menge L_t am Ziel t dominiert wird

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

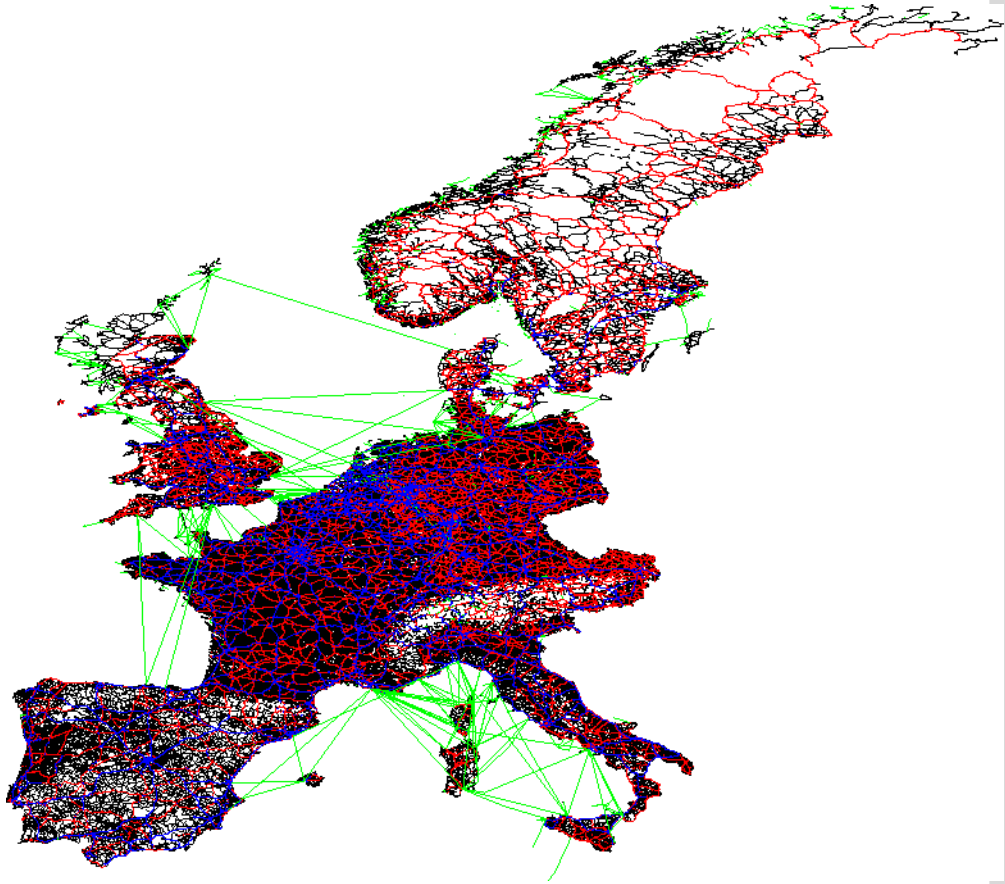
Eingabe

Straßengraphen von:

- Luxemburg
- Karlsruhe
- Niederlande
- Europa

Metriken:

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



Ähnliche Metriken: Europa

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

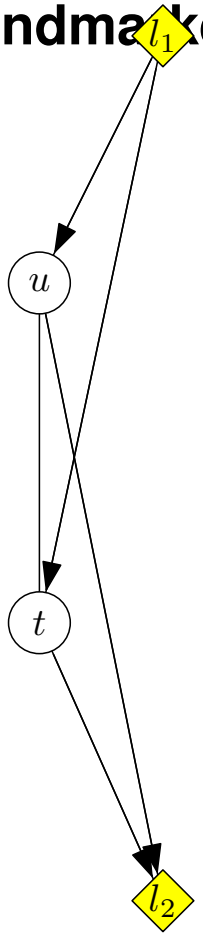
Verschiedene Metriken

	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

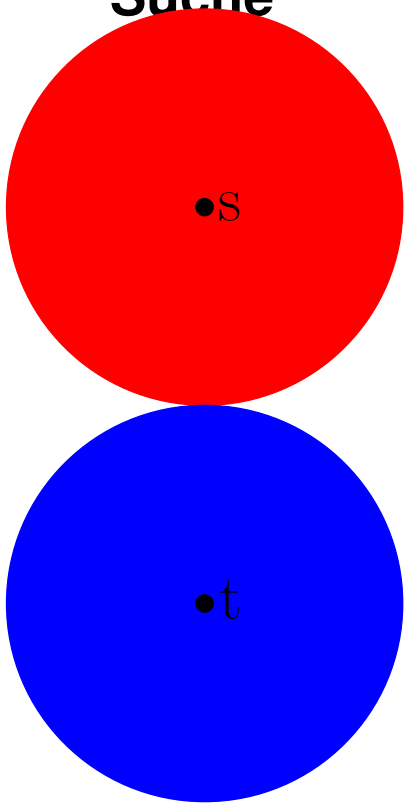
Je nach Kriterien kann Lösungsmenge stark ansteigen

Beschleunigungstechniken

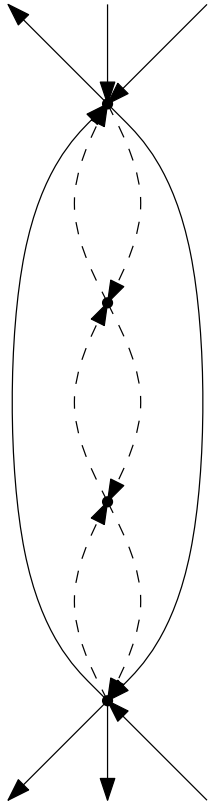
Landmarken



Bidirektionale Suche



Kontraktion



Arc-Flags

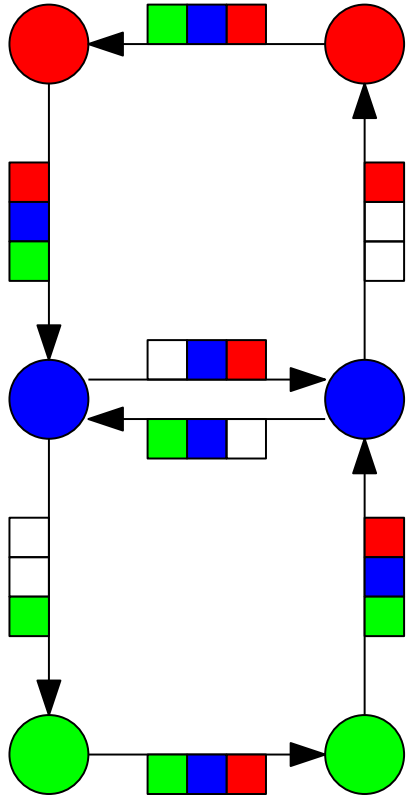
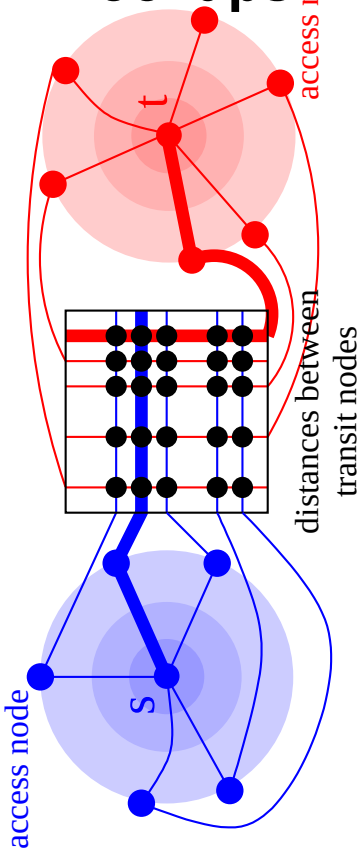


Table-Lookups



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Landmarken

Vorberechnung:

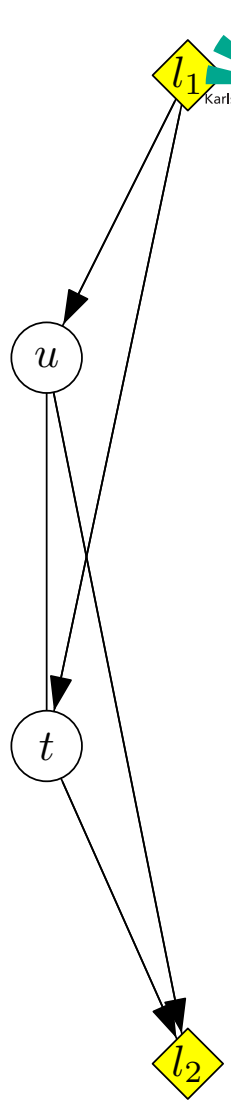
- Wähle eine Hand voll (≈ 16) Knoten als Landmarken
- Berechne Abstände von und zu allen Landmarken

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

Beobachtung:

- Korrektheit von ALT basiert darauf, dass reduzierten Kantengewichte größer gleich 0 sind

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

Anpassung

Beobachtung:

- Korrektheit von ALT basiert darauf, dass reduzierten Kantengewichte größer gleich 0 sind

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

Idee:

- Benutze einzelne Metriken zum Berechnen der Distanzen
- 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 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

Beobachtung:

- Korrektheit von ALT basiert darauf, dass reduzierten Kantengewichte größer gleich 0 sind

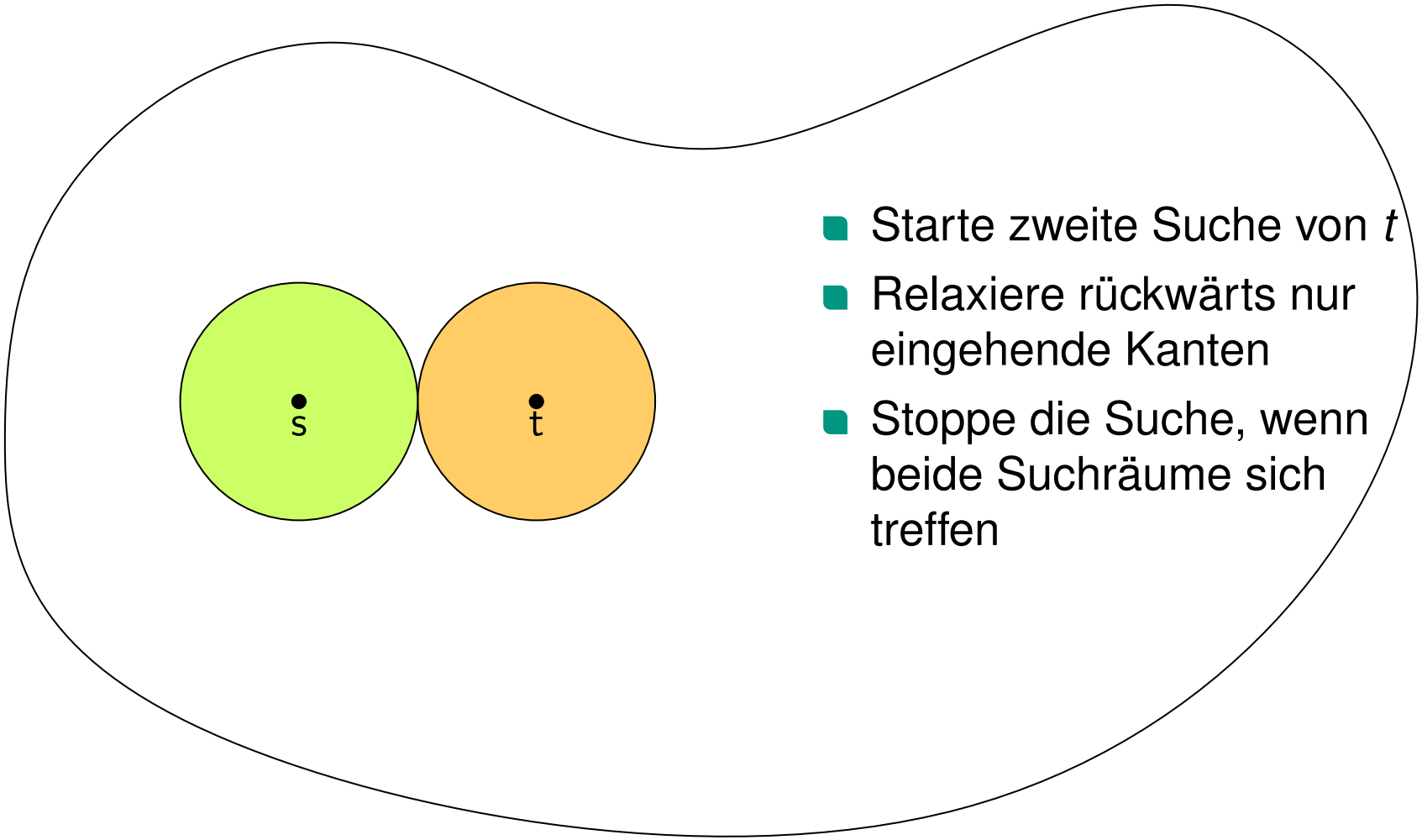
$$\text{len}_\pi(u,v) = \text{len}(u,v) - \pi(u) + \pi(v) \geq 0$$

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; teuer zu verwalten: Kombination der Lösungen aus Vorwärts- und Rückwärtssuche an Mittelknoten
- Lohnt nicht recht

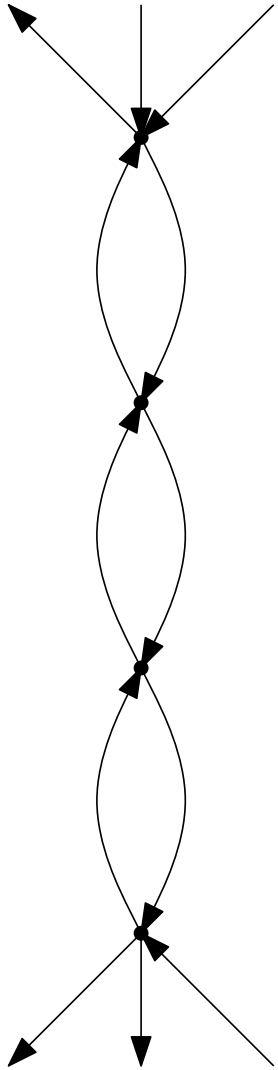
Kontraktion

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
- Lokale Suche während oder nach Knoten-reduktion



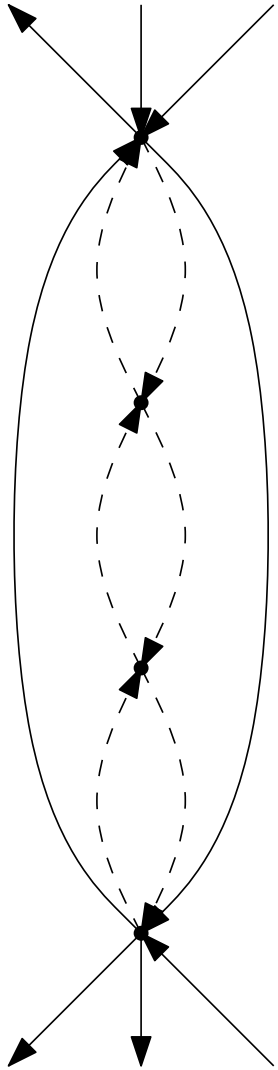
Kontraktion

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
- Lokale Suche während oder nach Knoten-reduktion



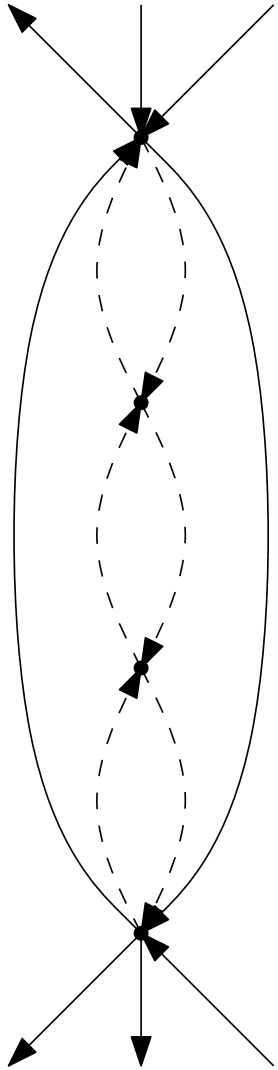
Anpassung Knoten-Reduktion

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

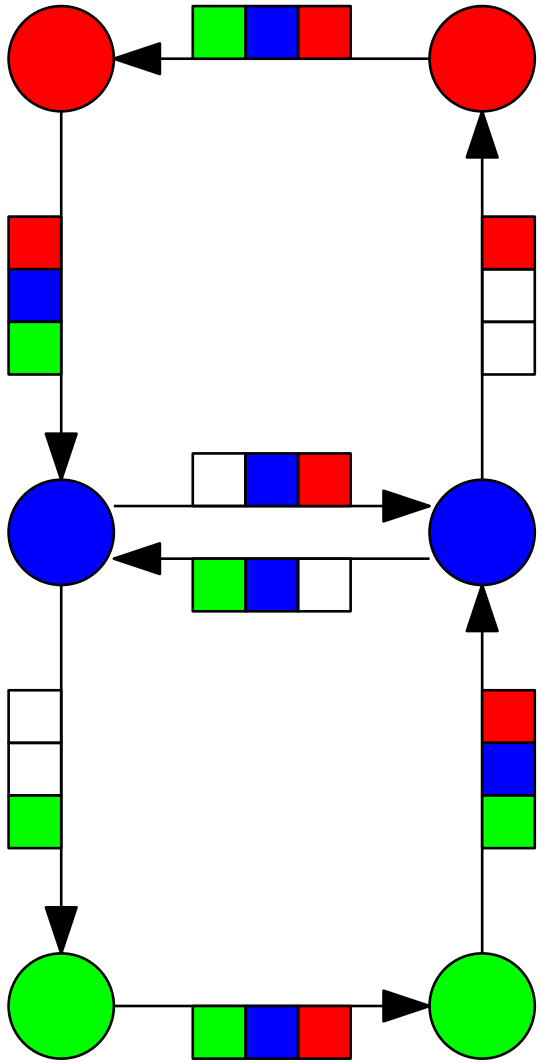
Arc-Flags

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

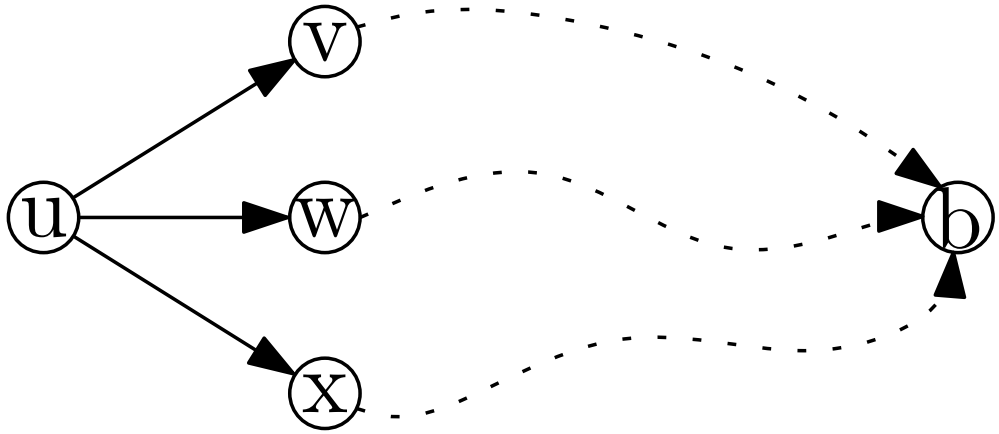
Anpassung

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



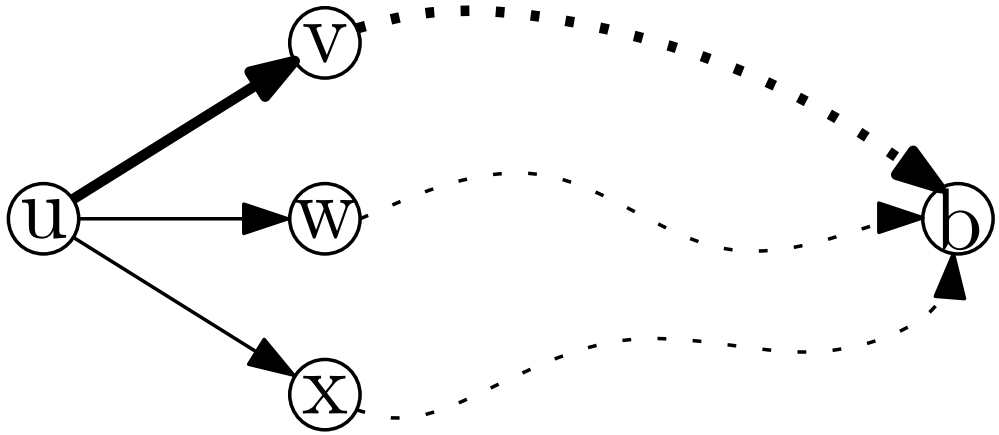
Anpassung

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



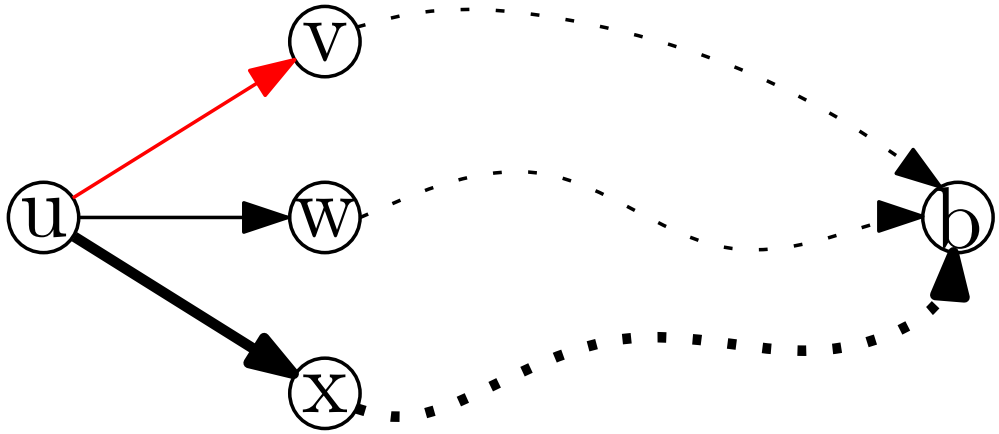
Anpassung

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

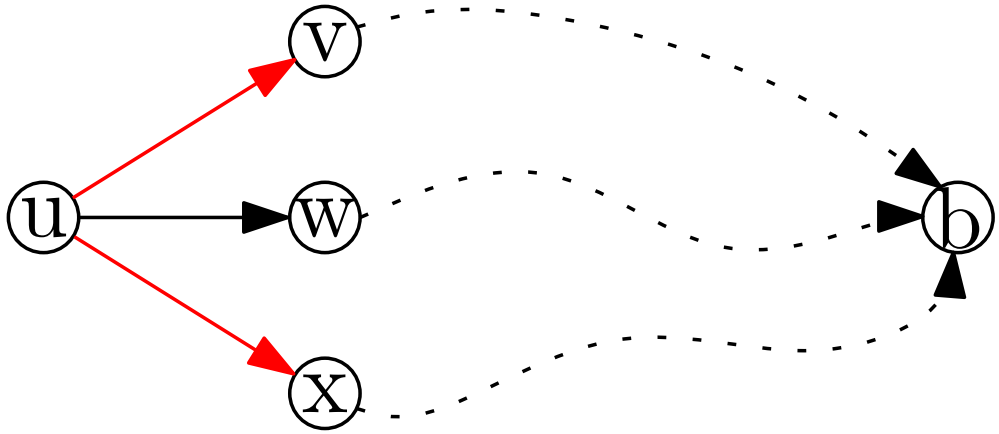
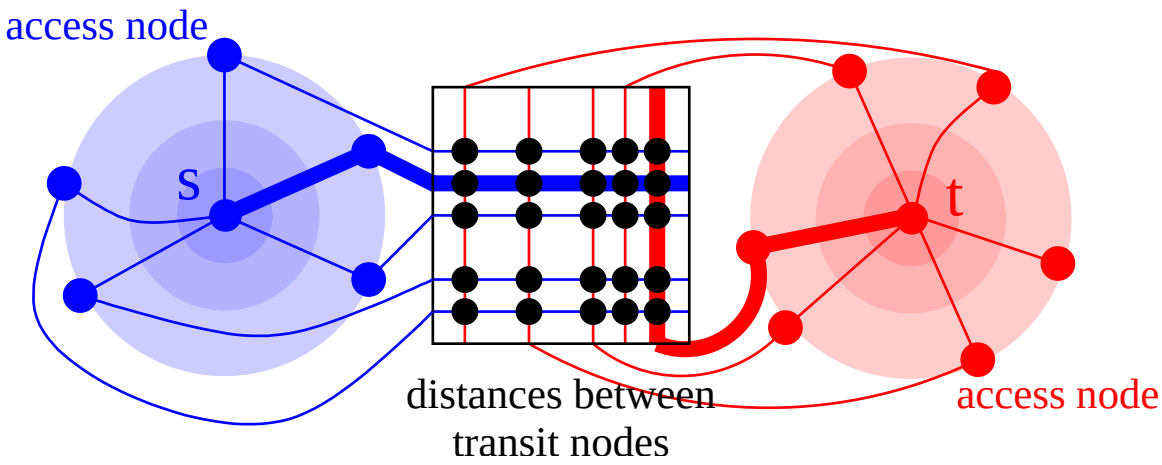


Table-Lookups

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 (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)

Anfrageszenarien

Punkt-zu-Punkt

- zwei Punkte $\rightarrow$ kürzester Weg
- wird für Routenplanung benutzt
- Beschleunigungstechniken
- HubLabels 10Mx schneller

Anfrageszenarien

Punkt-zu-Punkt

- zwei Punkte $\rightarrow$ kürzester Weg
- wird für Routenplanung benutzt
- Beschleunigungstechniken
- HubLabels 10Mx schneller

One-to-All

- ein Knoten $\rightarrow$ Distanzen zu allen Knoten
- wird für Vorberechnung benutzt
- PHAST 500x schneller (auf GPU)
- nutzt Hardware aus

Anfrageszenarien

Punkt-zu-Punkt

- zwei Punkte $\rightarrow$ kürzester Weg
- wird für Routenplanung benutzt
- Beschleunigungstechniken
- HubLabels 10Mx schneller

One-to-Many

- ein (variierender Knoten) und eine (feste) Menge $\rightarrow$ Distanz zu allen Knoten in der Menge
- wichtig für POI

One-to-All

- ein Knoten $\rightarrow$ Distanzen zu allen Knoten
- wird für Vorberechnung benutzt
- PHAST 500x schneller (auf GPU)
- nutzt Hardware aus

Anfrageszenarien

Punkt-zu-Punkt

- zwei Punkte $\rightarrow$ kürzester Weg
- wird für Routenplanung benutzt
- Beschleunigungstechniken
- HubLabels 10Mx schneller

One-to-All

- ein Knoten $\rightarrow$ Distanzen zu allen Knoten
- wird für Vorberechnung benutzt
- PHAST 500x schneller (auf GPU)
- nutzt Hardware aus

One-to-Many

- ein (variierender Knoten) und eine (feste) Menge $\rightarrow$ Distanz zu allen Knoten in der Menge
- wichtig für POI

Many-to-Many

- zwei Mengen $\rightarrow$ Distanztabelle
- wichtig für Vehicle Routing

Das One-to-Many Problem

Problem Definition:

- Eingabe: eine Knoten s und eine Menge T
- Ausgabe: Distanz von s zu allen $t \in T$
- Annahme: wir fixieren T und variieren s

Das One-to-Many Problem

Problem Definition:

- Eingabe: eine Knoten s und eine Menge T
- Ausgabe: Distanz von s zu allen $t \in T$
- Annahme: wir fixieren T und variieren s

offensichtliche Lösungen:

- Dijkstras Algorithmus (mit Stoppkriterium)
  - $\Rightarrow$ Performance stark abhängig von $|T|$ und Verteilung von T

Das One-to-Many Problem

Problem Definition:

- Eingabe: eine Knoten s und eine Menge T
- Ausgabe: Distanz von s zu allen $t \in T$
- Annahme: wir fixieren T und variieren s

offensichtliche Lösungen:

- Dijkstras Algorithmus (mit Stoppkriterium)
  $\Rightarrow$ Performance stark abhängig von $|T|$ und Verteilung von T
- $|T|$ p2p Anfragen (z.B. HL)
  $\Rightarrow$ Performance stark abhängig von $|T|$

Das One-to-Many Problem

Problem Definition:

- Eingabe: eine Knoten s und eine Menge T
- Ausgabe: Distanz von s zu allen $t \in T$
- Annahme: wir fixieren T und variieren s

offensichtliche Lösungen:

- Dijkstras Algorithmus (mit Stoppkriterium)
  $\Rightarrow$ Performance stark abhängig von $|T|$ und Verteilung von T
- $|T|$ p2p Anfragen (z.B. HL)
  $\Rightarrow$ Performance stark abhängig von $|T|$
- benutze PHAST (kein Stoppkriterium!)
  $\Rightarrow$ Overkill (vor allem für kleine T)

Erste Ideen

Vorschläge?

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Ungerichtete Bidir. Techniken

Definition:

- $\overrightarrow{\sigma}(s,t)$: Suchraum der Vorwärtssuche von s nach t
- $\overleftarrow{\sigma}(s,t)$ analog
- eine bidirektionale Suche ist Ziel-unabhängig, gdw.

$$\forall (s,t_1,t_2) \in V^3 : \overrightarrow{\sigma}(s,t_1) = \overrightarrow{\sigma}(s,t_2) \quad \text{und}$$
$$\forall (s_1,s_2,t) \in V^3 : \overleftarrow{\sigma}(s_1,t) = \overleftarrow{\sigma}(s_2,t)$$

Ungerichtete Bidir. Techniken

Definition:

- $\overrightarrow{\sigma}(s,t)$: Suchraum der Vorwärtssuche von s nach t
- $\overleftarrow{\sigma}(s,t)$ analog
- eine bidirektionale Suche ist Ziel-unabhängig, gdw.

$$\forall (s,t_1,t_2) \in V^3 : \overrightarrow{\sigma}(s,t_1) = \overrightarrow{\sigma}(s,t_2) \quad \text{und}$$
$$\forall (s_1,s_2,t) \in V^3 : \overleftarrow{\sigma}(s_1,t) = \overleftarrow{\sigma}(s_2,t)$$

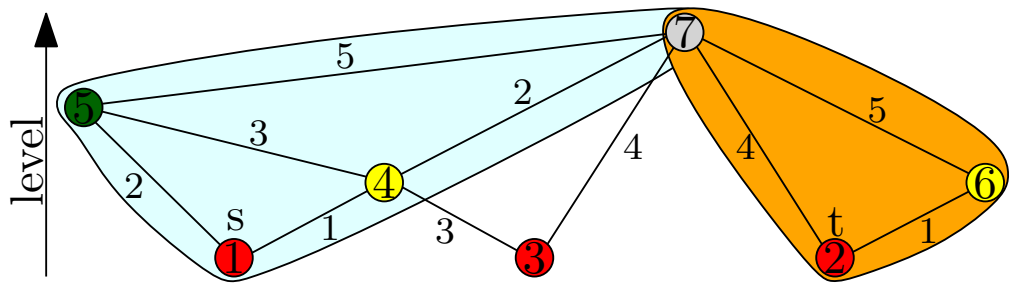
Beispiele:

- Bidirektionaler Dijkstra
- ohne Stoppkriterium, lass laufen bis Queues leer sind

Hierarchische Techniken

Beobachtung:

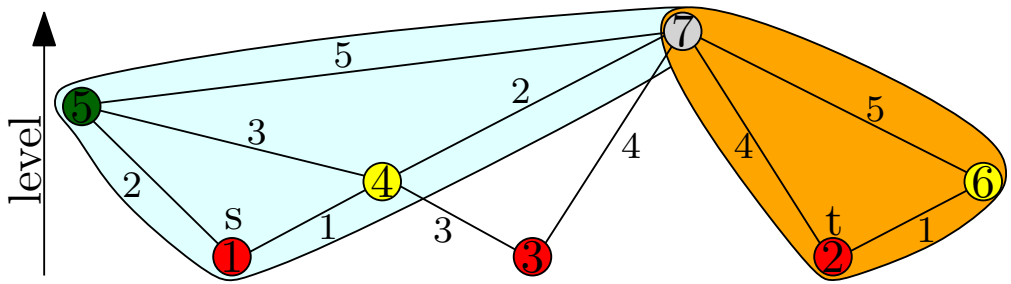
- suchen nur aufwärts
- sind nicht zielgerichtet



Hierarchische Techniken

Beobachtung:

- suchen nur aufwärts
- sind nicht zielgerichtet



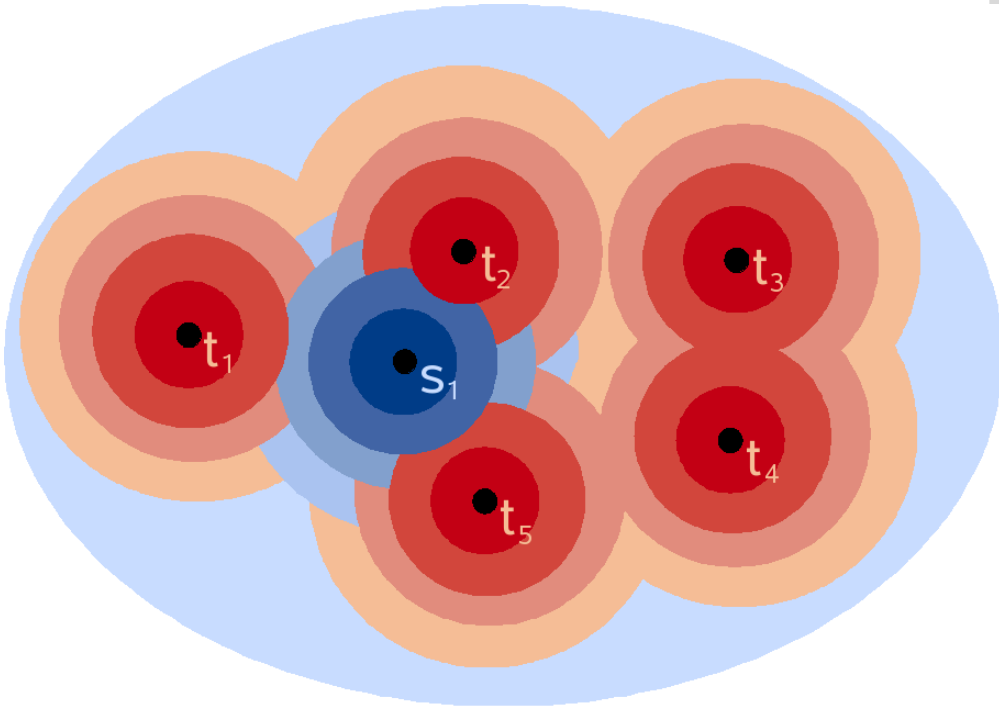
somit:

- Bidirektionaler Dijkstra
- Reach
- Contraction Hierarchies
- ohne Stoppkriterium, lass laufen bis Queues leer sind
- HL

Vorgehen

Idee:

- führe $|T|$ Rückwärtssuchen aus
- speicher für jedes besuchte u Abstände zu allen $t \in T$
- verwalte temporäres Distanzarray D_T
- führe Vorwärtssuche aus
- aktualisiere Einträge in D_T



Problem:

- Verwalten der Suchräume?

Schneiden der Suchräume

während Rückwärtssuchen: (Target selection phase)

- je $t \in T$:
- starte Suche im Rückwärts-DAG, breche nicht ab
- für jedes erreichte u:
- speichere einen Bucket $\beta(u)$ mit $(t, d(u,t))$
  Alternative Sichtweise: Füge gewichtete Abwärtskanten zu erreichbaren Zielknoten ein

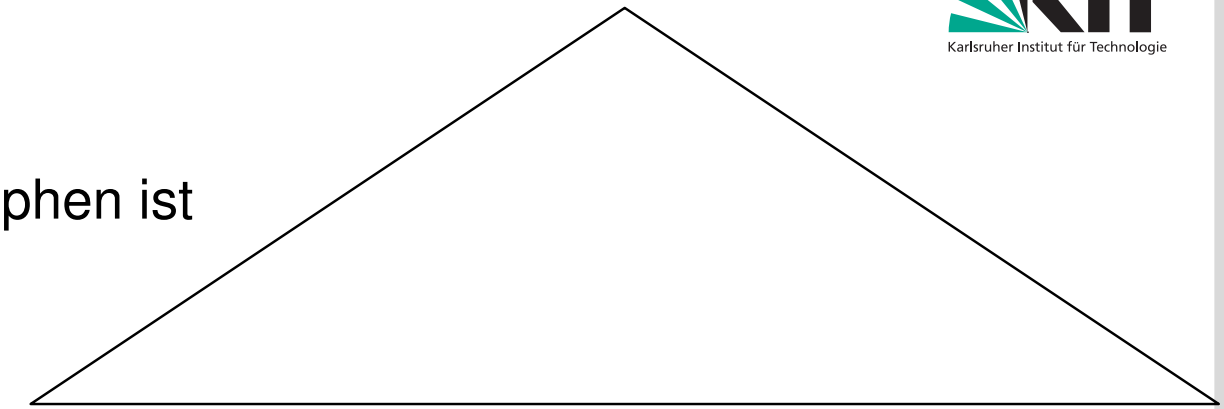
während Vorwärtssuche: (Query phase)

- breche nicht ab
- für jedes erreichte u:
  - scanne Bucket $\beta(u)$
  - aktualisiere Distanzarray D_T

RPHAST

Beobachtung PHAST:

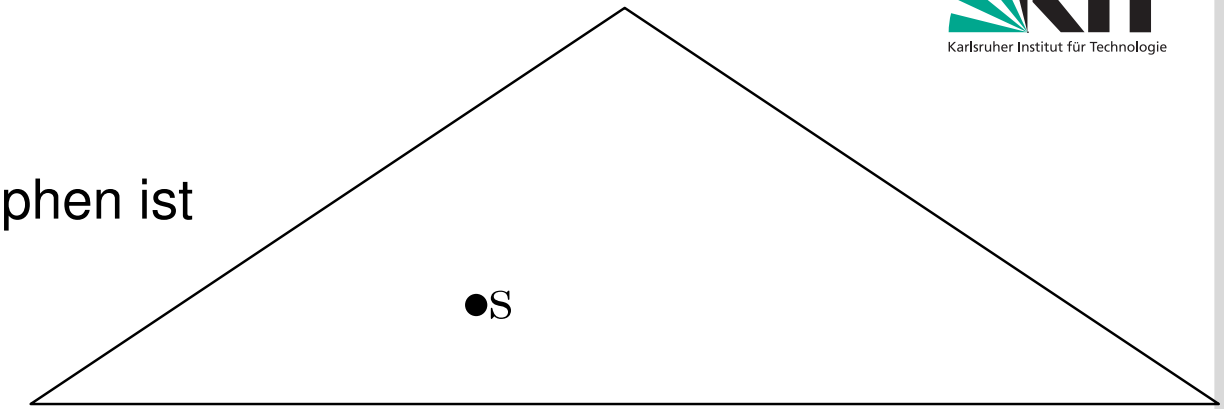
- Sweep über den Graphen ist der Flaschenhals



RPHAST

Beobachtung PHAST:

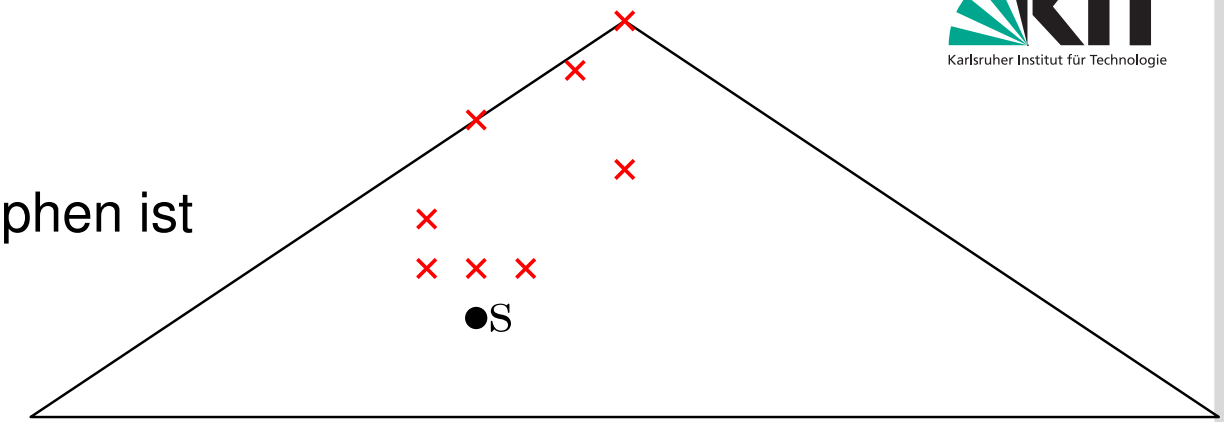
- Sweep über den Graphen ist der Flaschenhals



RPHAST

Beobachtung PHAST:

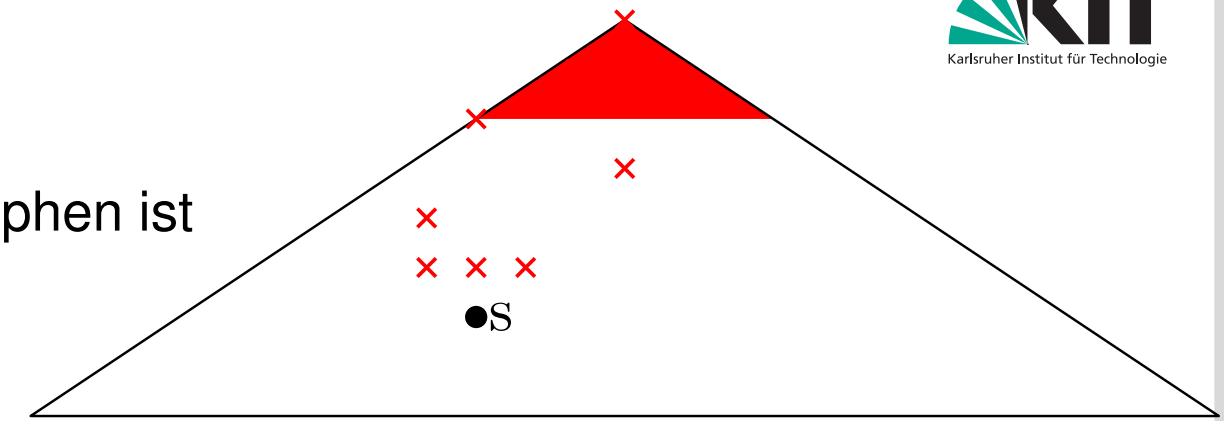
- Sweep über den Graphen ist der Flaschenhals



RPHAST

Beobachtung PHAST:

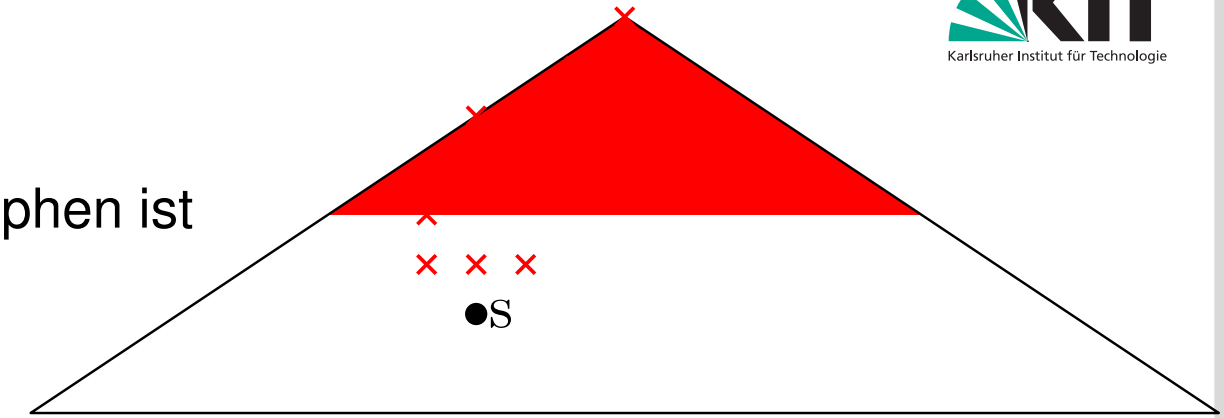
- Sweep über den Graphen ist der Flaschenhals



RPHAST

Beobachtung PHAST:

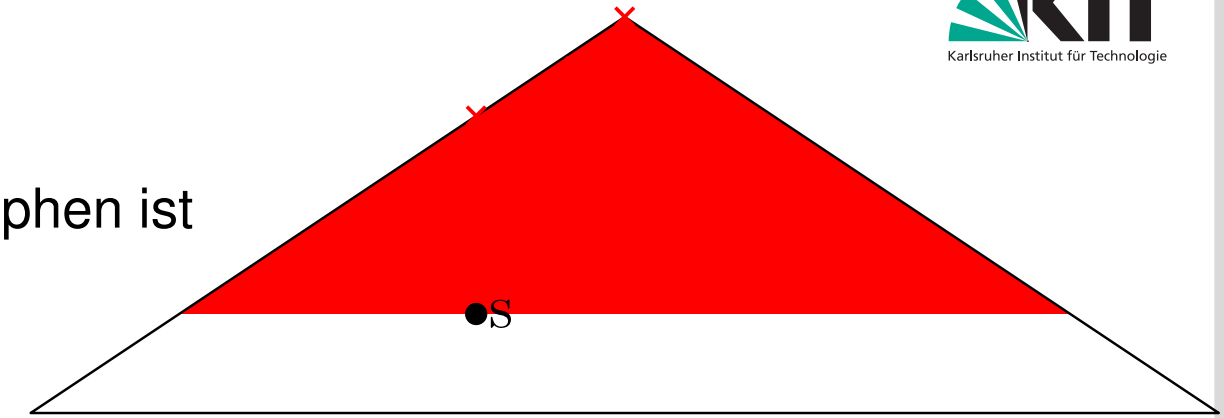
- Sweep über den Graphen ist der Flaschenhals



RPHAST

Beobachtung PHAST:

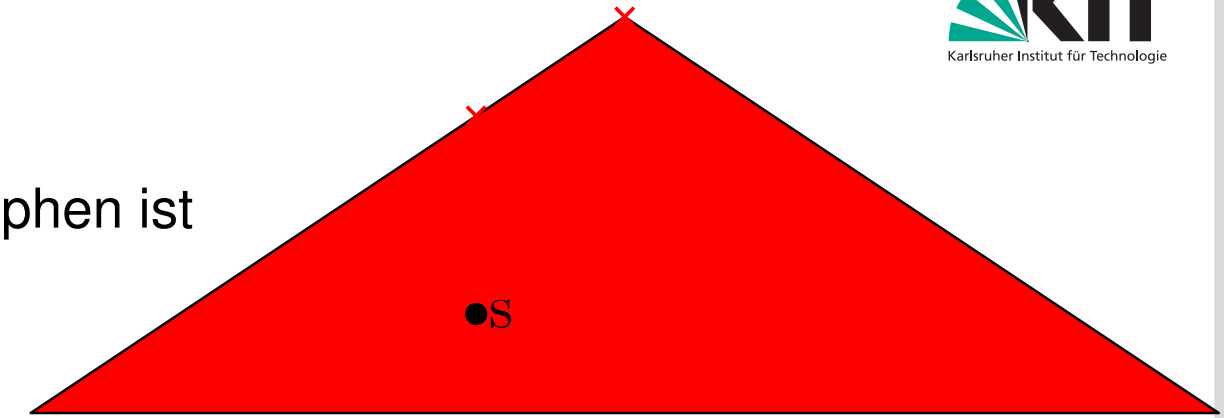
- Sweep über den Graphen ist der Flaschenhals



RPHAST

Beobachtung PHAST:

- Sweep über den Graphen ist der Flaschenhals

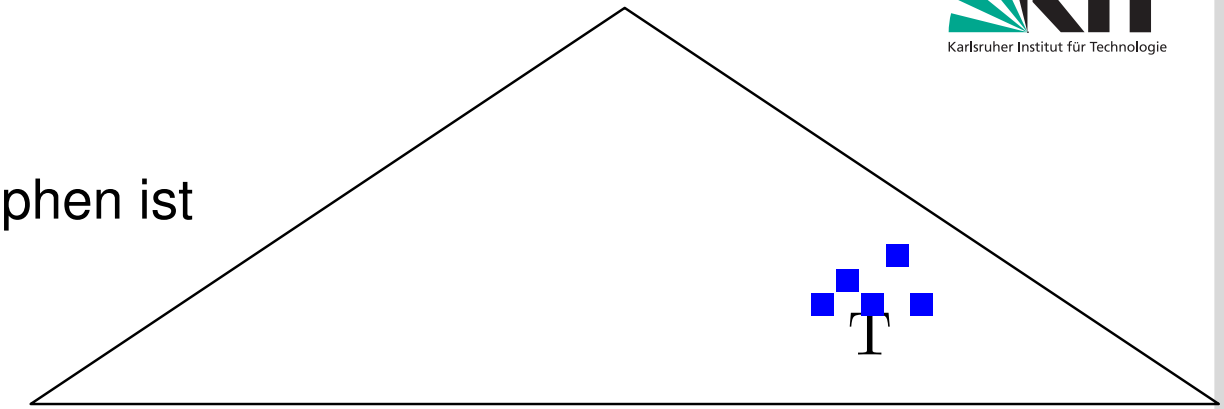


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

RPHAST

Beobachtung PHAST:

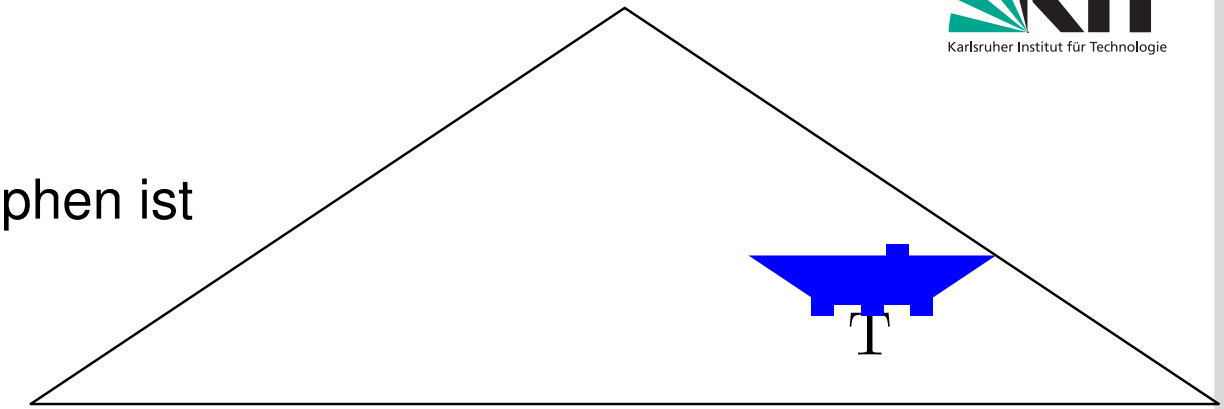
- Sweep über den Graphen ist der Flaschenhals



Idee:

- extrahiere relevanten Teil des Graphen (Ziel Selektion)

RPHAST



Beobachtung PHAST:

- Sweep über den Graphen ist der Flaschenhals

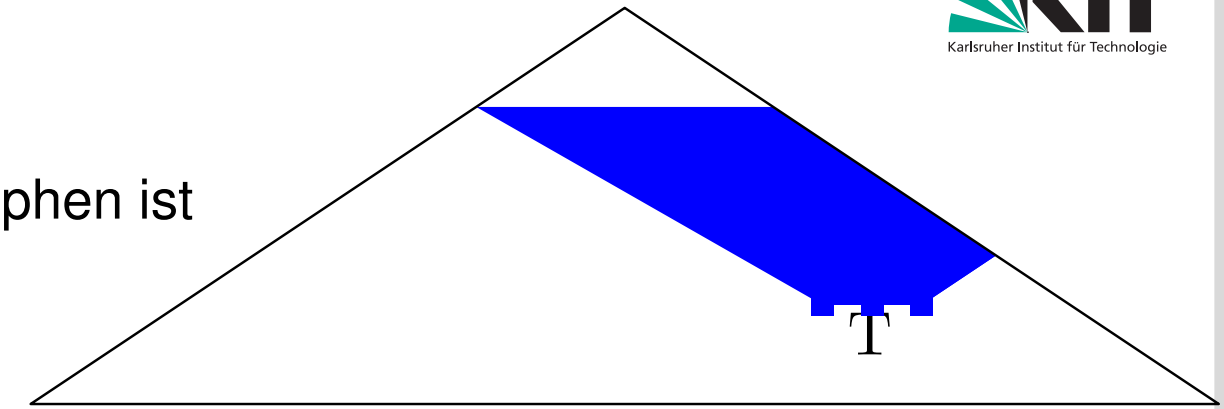
Idee:

- extrahiere relevanten Teil des Graphen (Ziel Selektion)

RPHAST

Beobachtung PHAST:

- Sweep über den Graphen ist der Flaschenhals



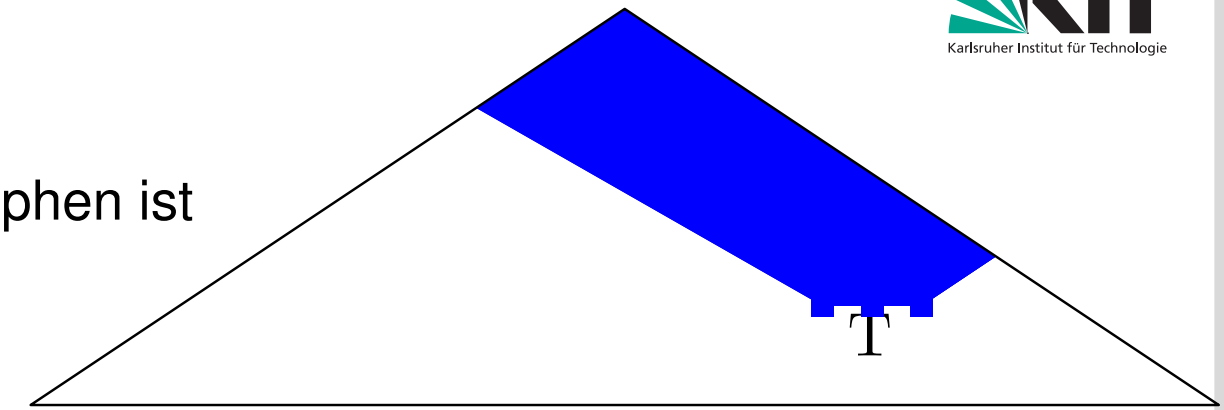
Idee:

- extrahiere relevanten Teil des Graphen (Ziel Selektion)

RPHAST

Beobachtung PHAST:

- Sweep über den Graphen ist der Flaschenhals



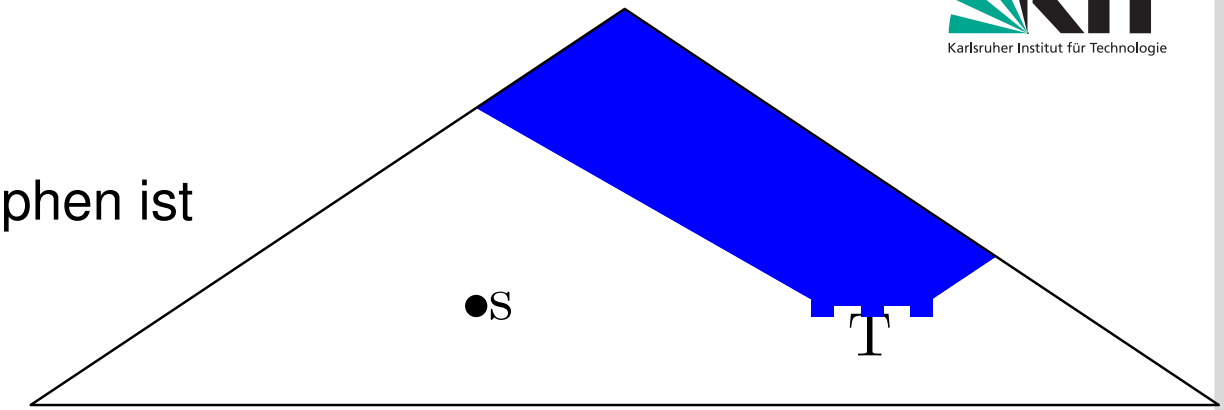
Idee:

- extrahiere relevanten Teil des Graphen (Ziel Selektion)

RPHAST

Beobachtung PHAST:

- Sweep über den Graphen ist der Flaschenhals



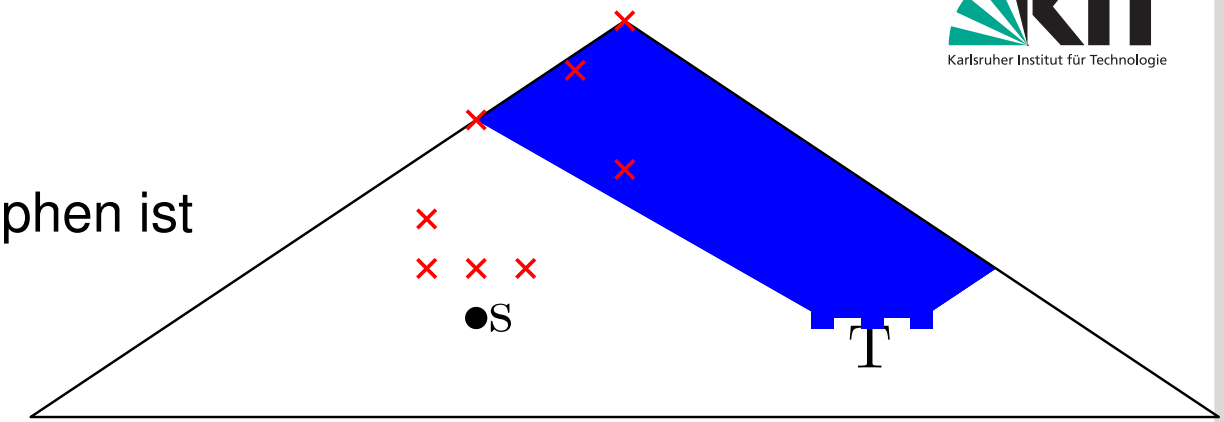
Idee:

- extrahiere relevanten Teil des Graphen (Ziel Selektion)
- Aufwärtssuche im vollen Graphen

RPHAST

Beobachtung PHAST:

- Sweep über den Graphen ist der Flaschenhals



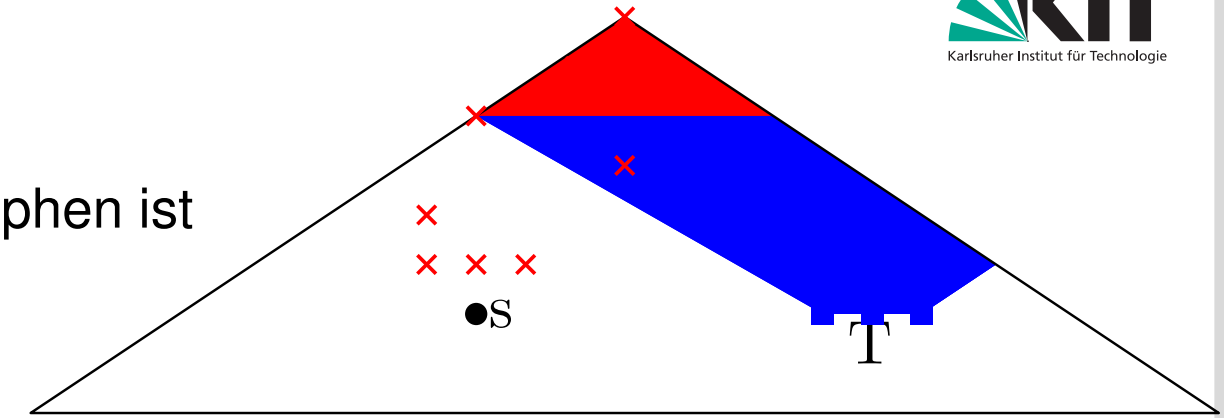
Idee:

- extrahiere relevanten Teil des Graphen (Ziel Selektion)
- Aufwärtssuche im vollen Graphen

RPHAST

Beobachtung PHAST:

- Sweep über den Graphen ist der Flaschenhals



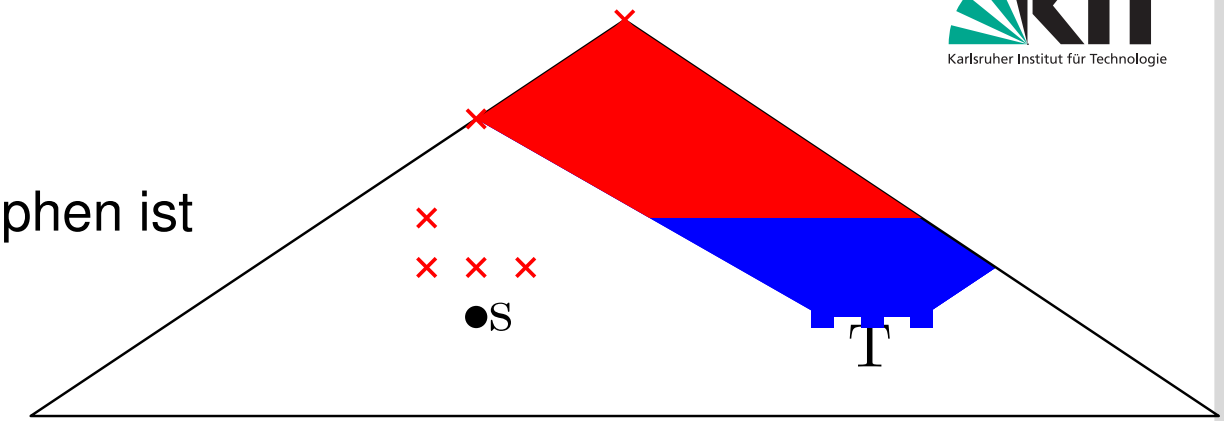
Idee:

- extrahiere relevanten Teil des Graphen (Ziel Selektion)
- Aufwärtssuche im vollen Graphen
- Sweep auf extrahiertem Graphen

RPHAST

Beobachtung PHAST:

- Sweep über den Graphen ist der Flaschenhals



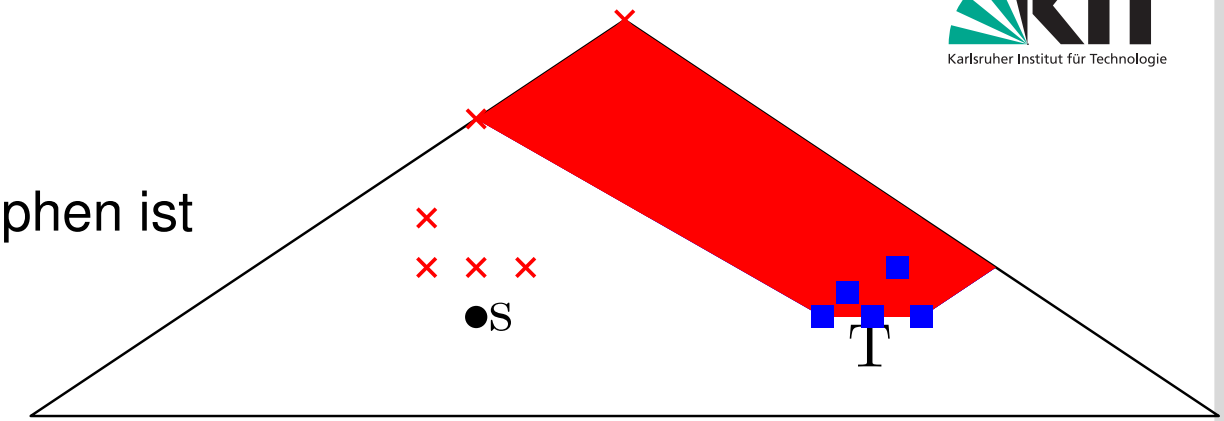
Idee:

- extrahiere relevanten Teil des Graphen (Ziel Selektion)
- Aufwärtssuche im vollen Graphen
- Sweep auf extrahiertem Graphen

RPHAST

Beobachtung PHAST:

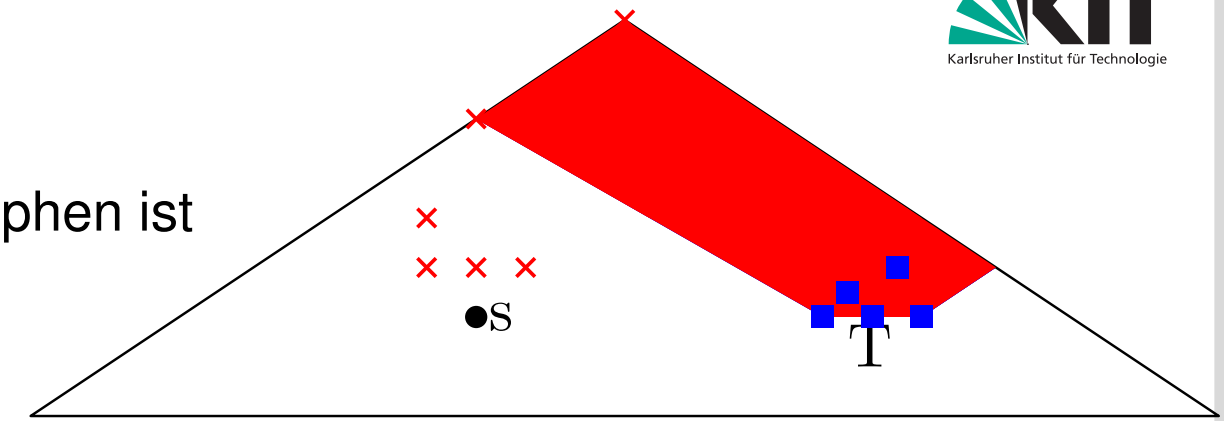
- Sweep über den Graphen ist der Flaschenhals



Idee:

- extrahiere relevanten Teil des Graphen (Ziel Selektion)
- Aufwärtssuche im vollen Graphen
- Sweep auf extrahiertem Graphen

RPHAST



Beobachtung PHAST:

- Sweep über den Graphen ist der Flaschenhals

Idee:

- extrahiere relevanten Teil des Graphen (Ziel Selektion)
- Aufwärtssuche im vollen Graphen
- Sweep auf extrahiertem Graphen

$\Rightarrow$

- Startknoten kann im ganzem Graphen liegen
- Grösse des extrahierten Graphen hängt von Verteilung und Anzahl T ab
- kann wie PHAST parallelisiert werden
- GPU implementation möglich

Experimente

Problem:
Je nach Szenario liegen die Ziele in einer kleinen Region oder sind über weite Teile des Graphen verteilt.

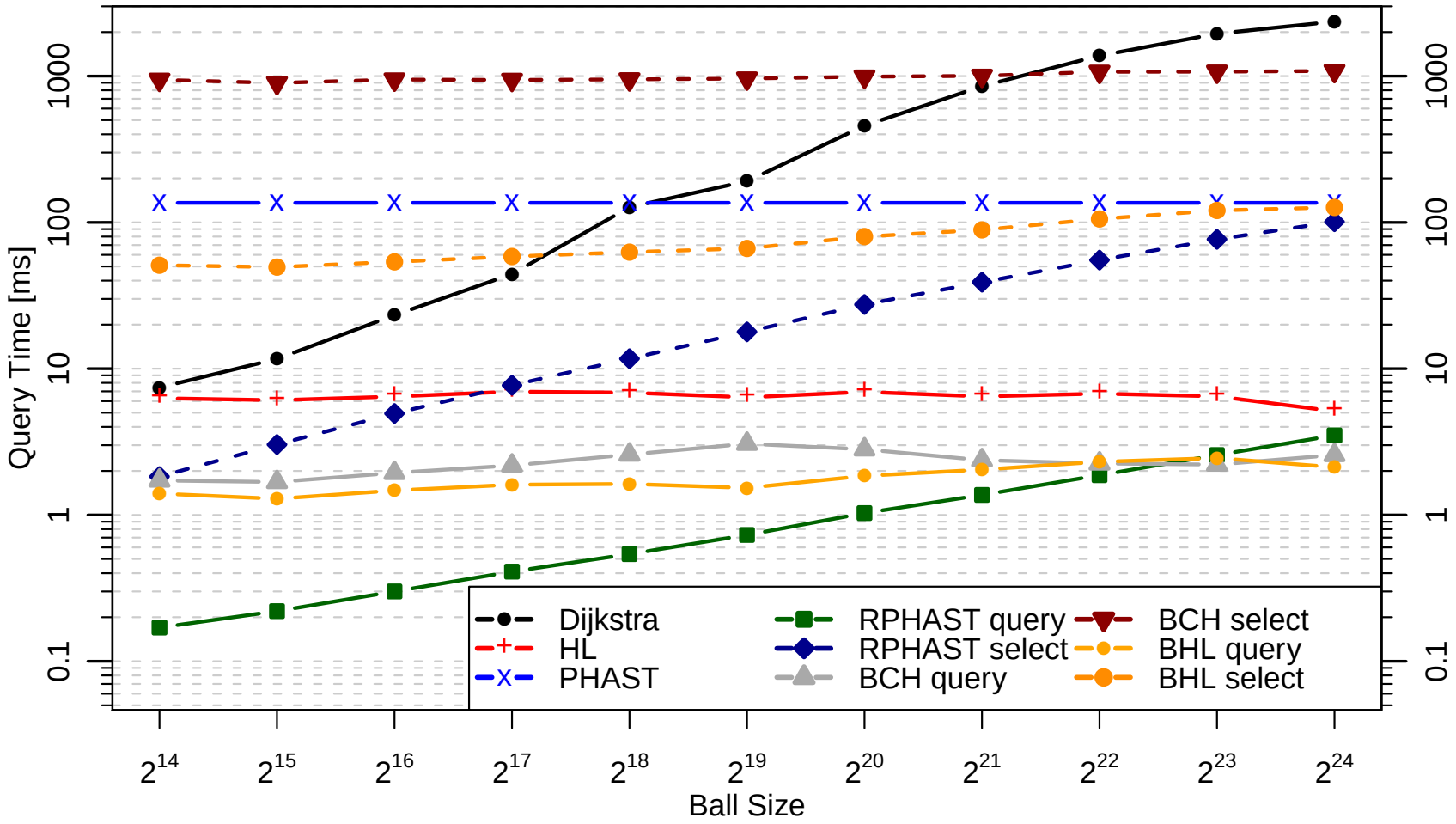
Setup:

- starte Dijkstra von zufälligem Knoten c
- brich nach B besuchten Knoten ab (Ballsize)
- wähle zufällige Zielknotenmenge $T \subseteq B$

Vergleiche Performance von Bucket CH (BCH), Bucket HL (BHL), RPHAST

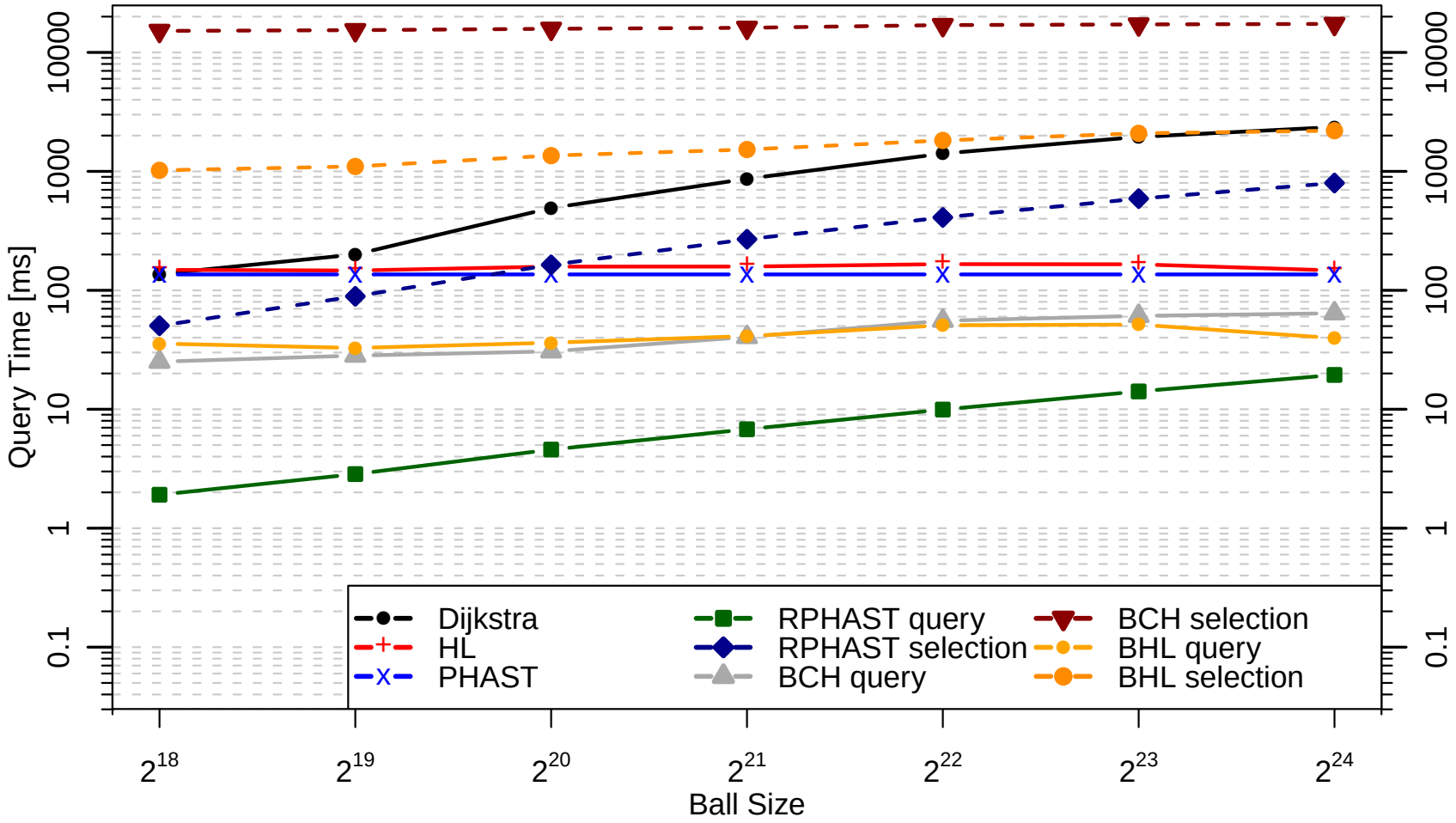
Experimente I

input: Westeuropa (18M Knoten), $|T| = 2^{14}$



Experimente II

input: Westeuropa (18M Knoten), $|T| = 2^{18}$



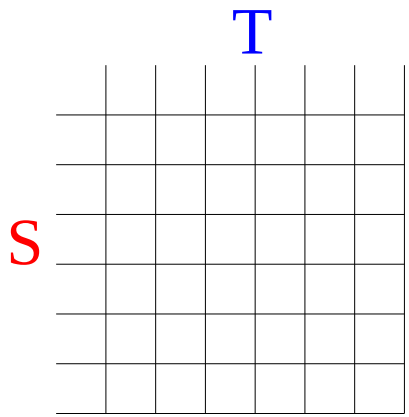
Many-to-Many Kürzeste Wege

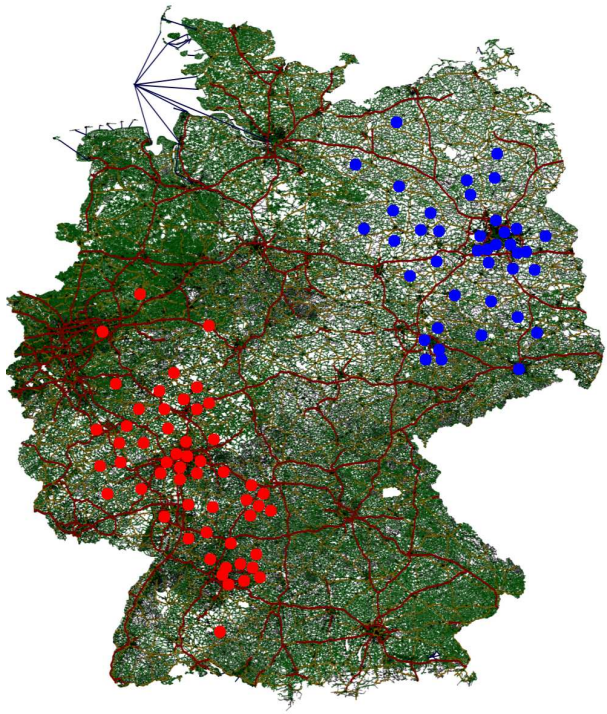
Gegeben:

- Graph
- Knotenmengen $S, T \in V$

Gesucht:

- Distanzmatrix D





Many-to-Many Kürzeste Wege

Gegeben:

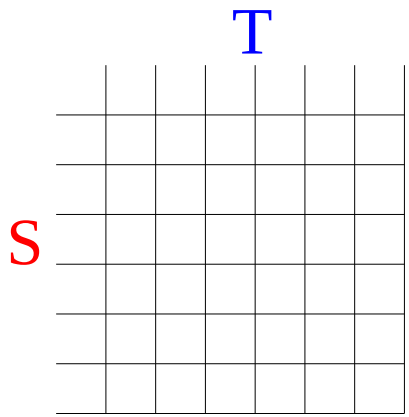
- Graph
- Knotenmengen $S, T \in V$

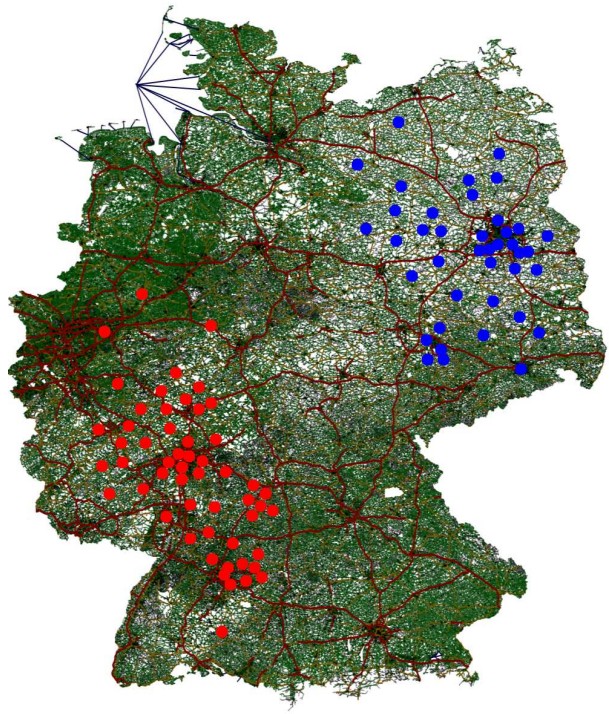
Gesucht:

- Distanzmatrix D

Anwendungen:

- vehicle routing
- traveling salesman





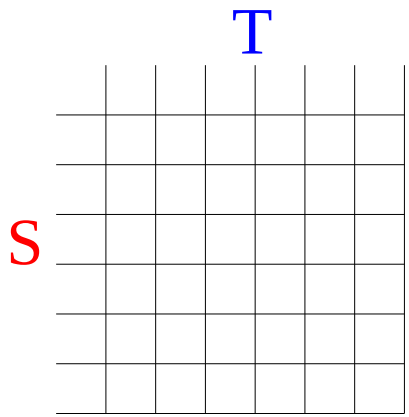
Many-to-Many Kürzeste Wege

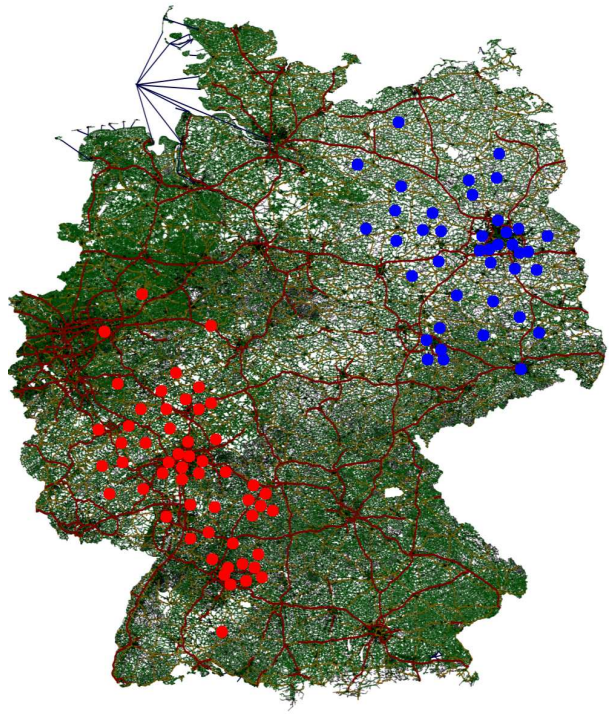
Gegeben:

- Graph
- Knotenmengen $S, T \in V$

Gesucht:

- Distanzmatrix D





Anwendungen:

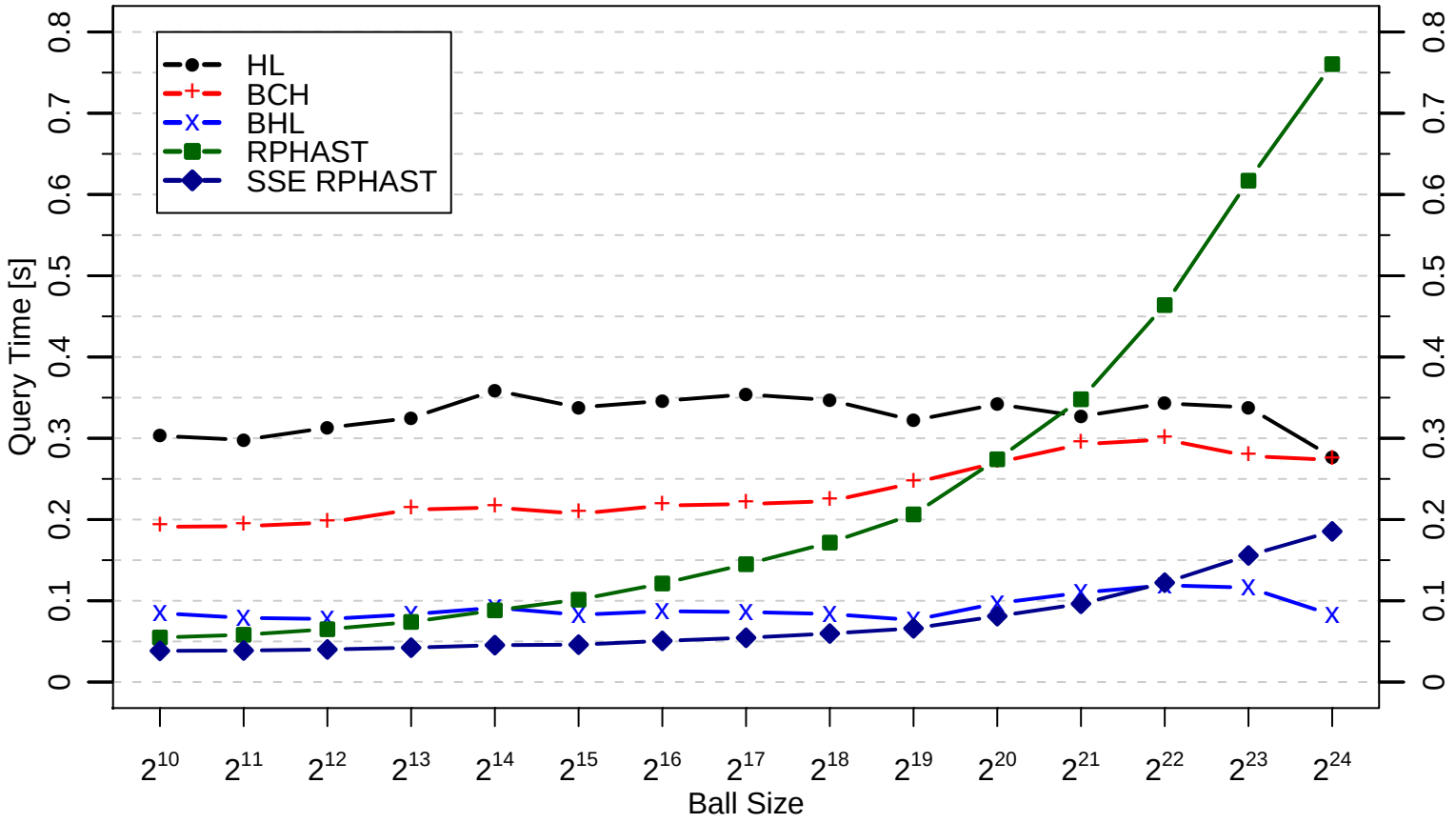
- vehicle routing
- traveling salesman

Lösung:

- $|S|$ one-to-many Anfragen
- speicher Distanzen in der Tabelle
- RPHAST kann multiples Setup (SSE) nutzen

Experimente I

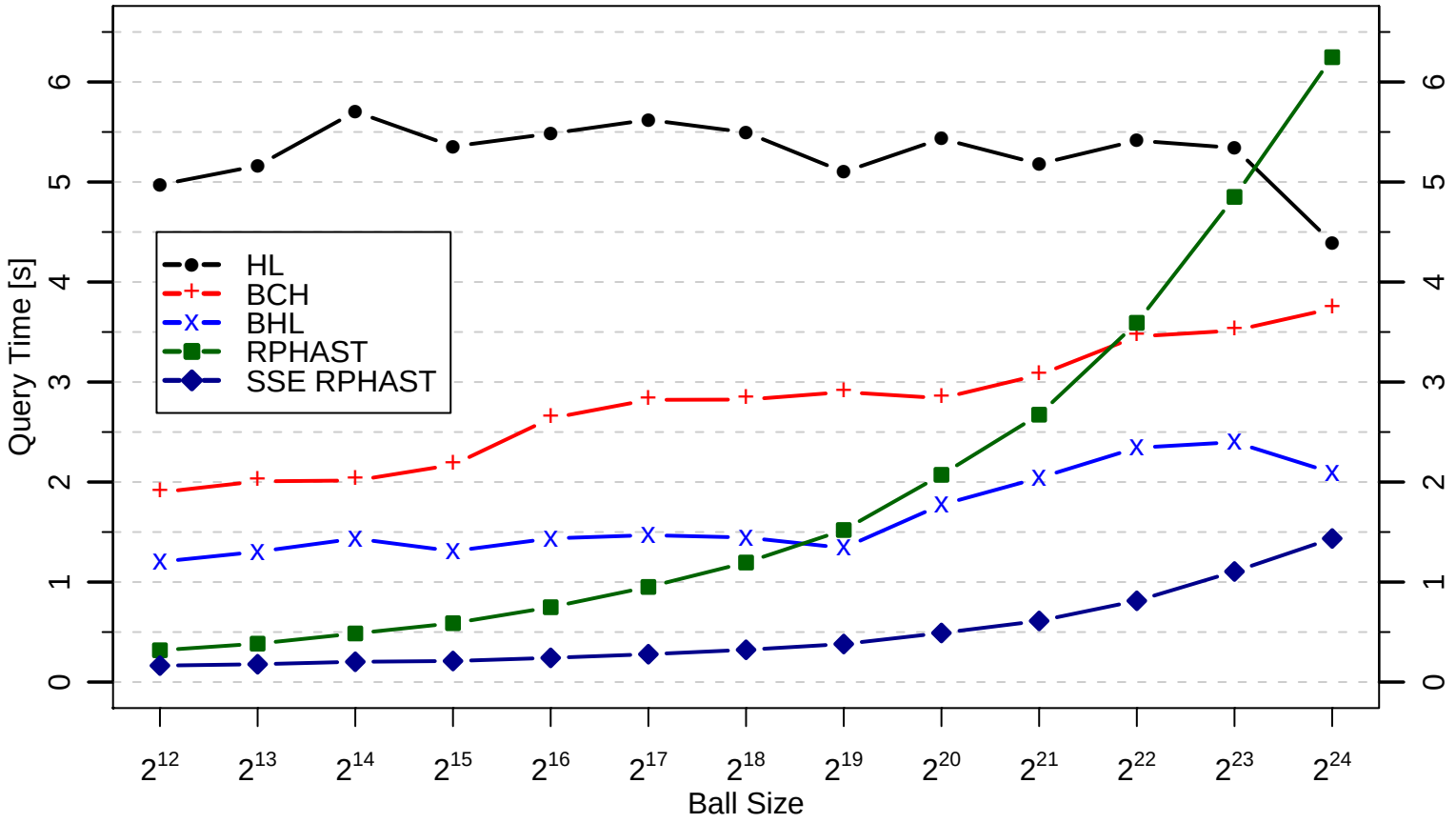
input: Westeuropa (18M Knoten), $|S| = |T| = 2^{10}$



Beobachtung: alle Techniken unter einer Sekunde

Experimente II

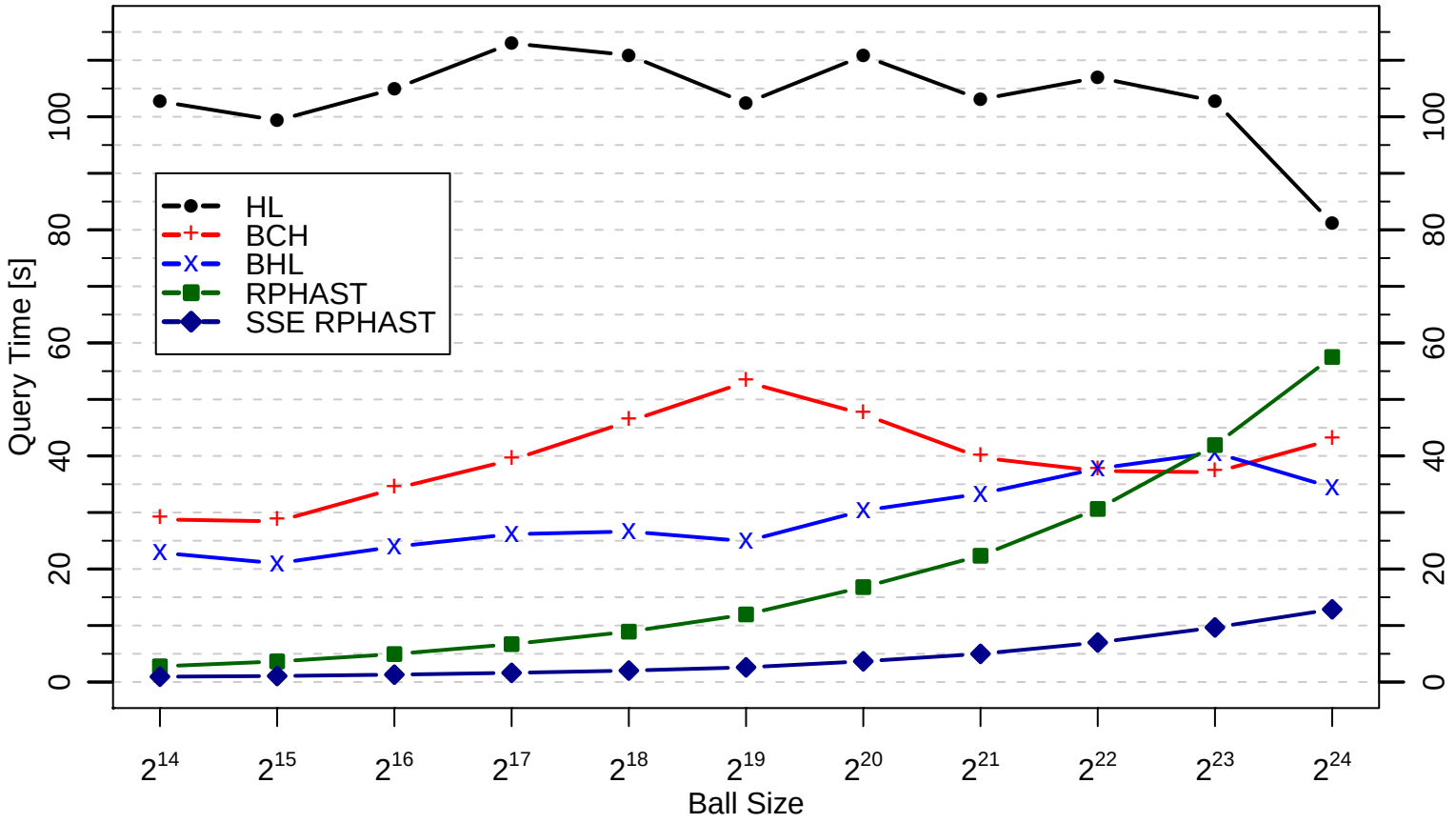
input: Westeuropa (18M Knoten), $|S| = |T| = 2^{12}$



Beobachtung: SSE PHAST am schnellsten

Experimente III

input: Westeuropa (18M Knoten), $|S| = |T| = 2^{14}$



Beobachtung: SSE PHAST am schnellsten

Nächster POI

Szenario:

- Zielknoten sind POIs (z.B. Paketshops)
- finde k nächste POIs von einem Startknoten s

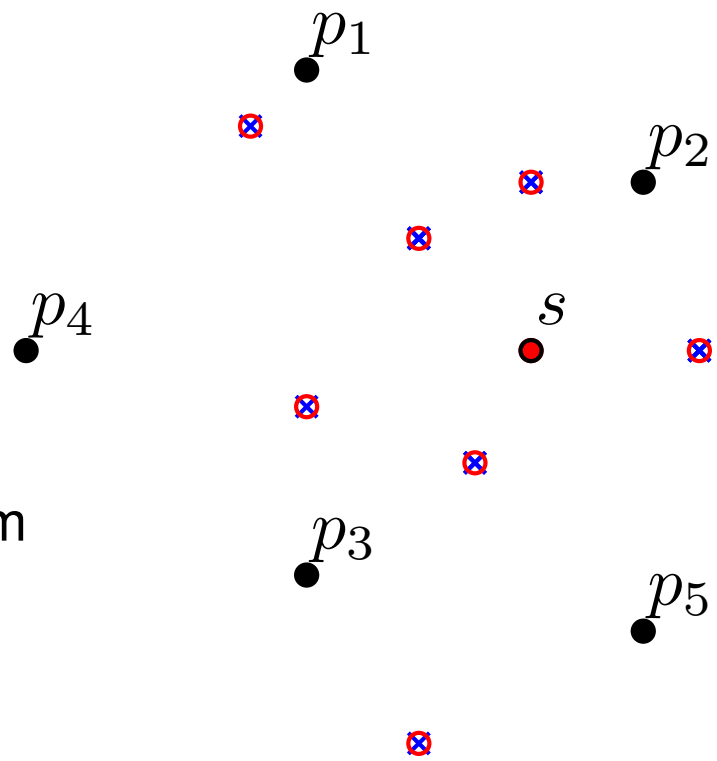
Nächster POI

Szenario:

- Zielknoten sind POIs (z.B. Paketshops)
- finde k nächste POIs von einem Startknoten s

Lösung:

- wie one-to-many
- ordne die buckets pro Knoten auch nach aufsteigender Distanz
- in jedem Bucket müssen nur die k nächsten POIs durchsucht werden
- Laufzeit für POI Query nicht abhängig von Anzahl POIs im System
- Laufzeit: Suchraum $\cdot$ k



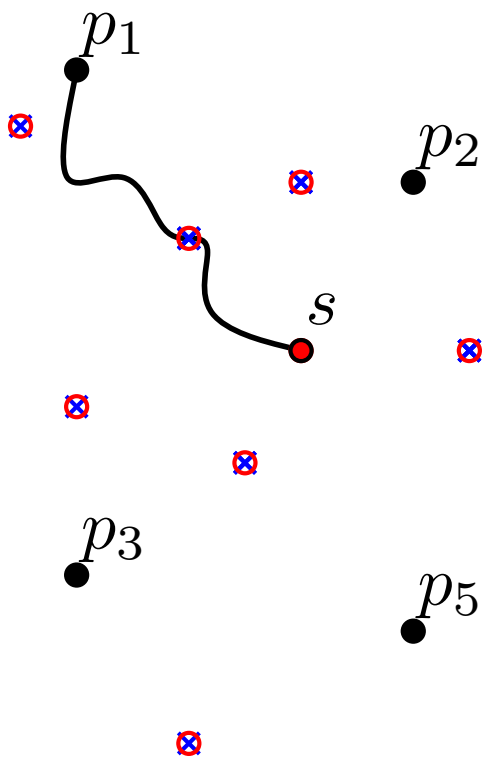
Nächster POI

Szenario:

- Zielknoten sind POIs (z.B. Paketshops)
- finde k nächste POIs von einem Startknoten s

Lösung:

- wie one-to-many
- ordne die buckets pro Knoten auch nach aufsteigender Distanz
- in jedem Bucket müssen nur die k nächsten POIs durchsucht werden
- Laufzeit für POI Query nicht abhängig von Anzahl POIs im System
- Laufzeit: Suchraum $\cdot$ k



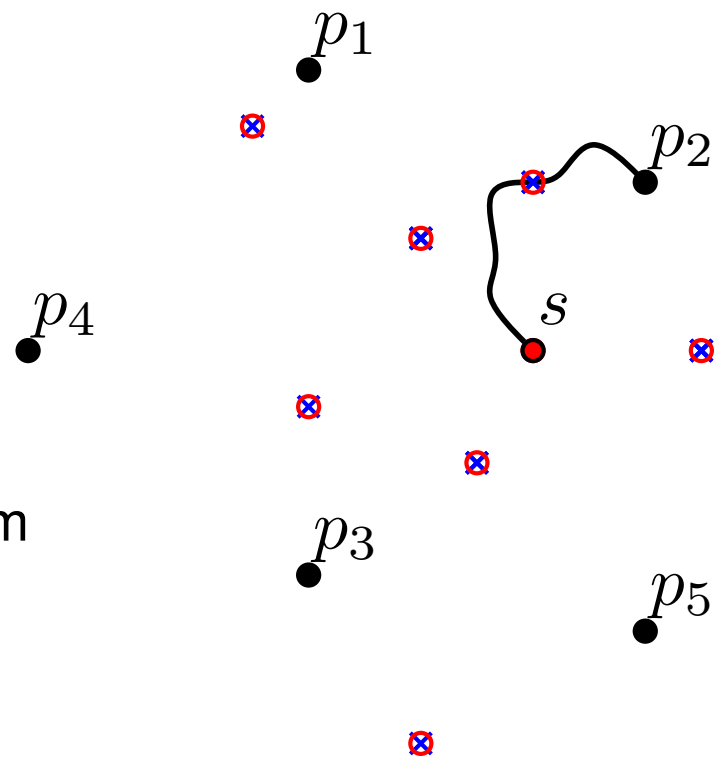
Nächster POI

Szenario:

- Zielknoten sind POIs (z.B. Paketshops)
- finde k nächste POIs von einem Startknoten s

Lösung:

- wie one-to-many
- ordne die buckets pro Knoten auch nach aufsteigender Distanz
- in jedem Bucket müssen nur die k nächsten POIs durchsucht werden
- Laufzeit für POI Query nicht abhängig von Anzahl POIs im System
- Laufzeit: Suchraum $\cdot$ k



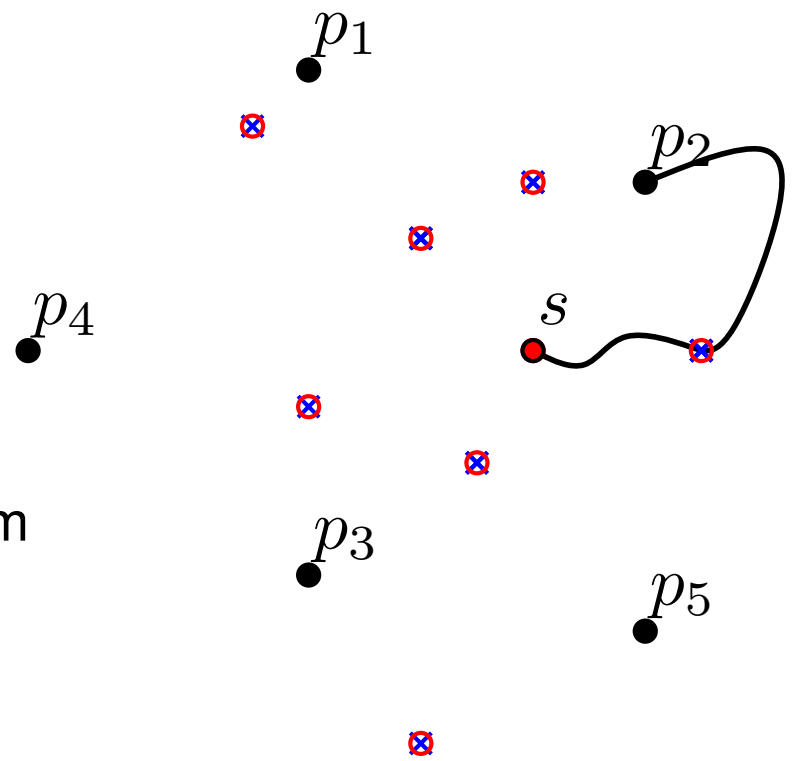
Nächster POI

Szenario:

- Zielknoten sind POIs (z.B. Paketshops)
- finde k nächste POIs von einem Startknoten s

Lösung:

- wie one-to-many
- ordne die buckets pro Knoten auch nach aufsteigender Distanz
- in jedem Bucket müssen nur die k nächsten POIs durchsucht werden
- Laufzeit für POI Query nicht abhängig von Anzahl POIs im System
- Laufzeit: Suchraum $\cdot$ k



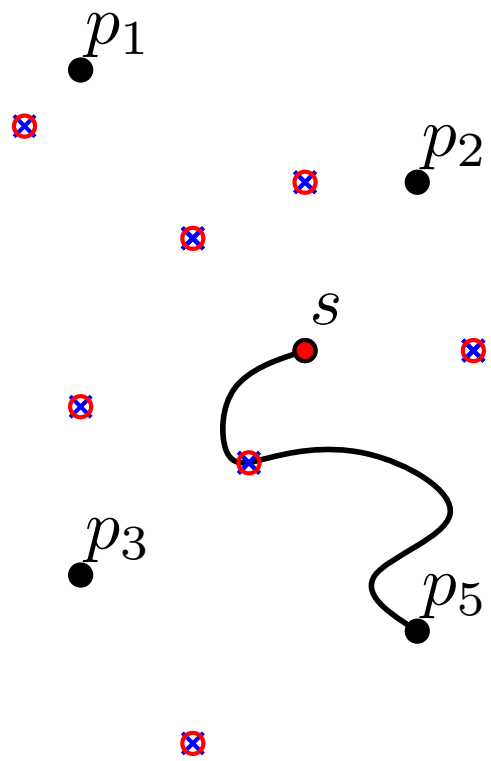
Nächster POI

Szenario:

- Zielknoten sind POIs (z.B. Paketshops)
- finde k nächste POIs von einem Startknoten s

Lösung:

- wie one-to-many
- ordne die buckets pro Knoten auch nach aufsteigender Distanz
- in jedem Bucket müssen nur die k nächsten POIs durchsucht werden
- Laufzeit für POI Query nicht abhängig von Anzahl POIs im System
- Laufzeit: Suchraum $\cdot$ k



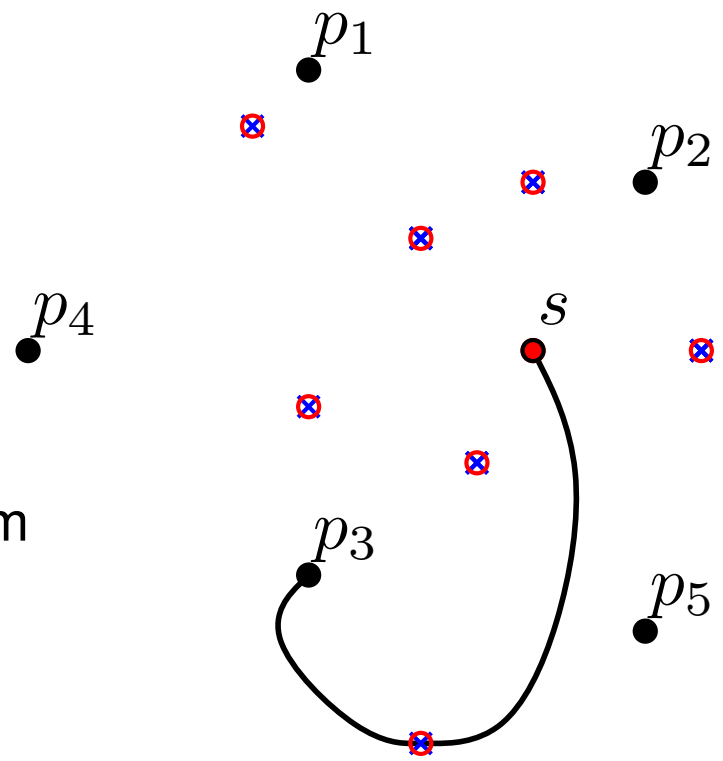
Nächster POI

Szenario:

- Zielknoten sind POIs (z.B. Paketshops)
- finde k nächste POIs von einem Startknoten s

Lösung:

- wie one-to-many
- ordne die buckets pro Knoten auch nach aufsteigender Distanz
- in jedem Bucket müssen nur die k nächsten POIs durchsucht werden
- Laufzeit für POI Query nicht abhängig von Anzahl POIs im System
- Laufzeit: Suchraum $\cdot$ k



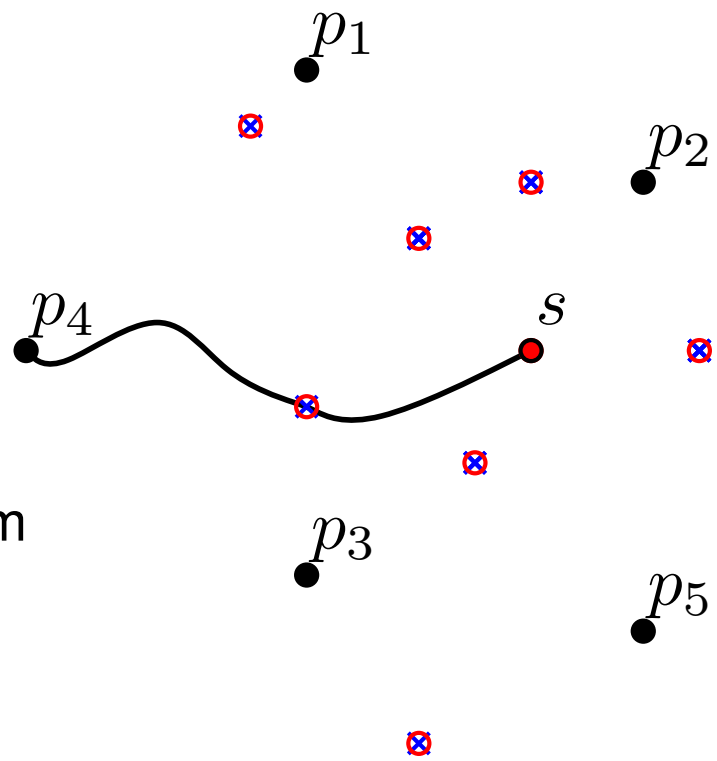
Nächster POI

Szenario:

- Zielknoten sind POIs (z.B. Paketshops)
- finde k nächste POIs von einem Startknoten s

Lösung:

- wie one-to-many
- ordne die buckets pro Knoten auch nach aufsteigender Distanz
- in jedem Bucket müssen nur die k nächsten POIs durchsucht werden
- Laufzeit für POI Query nicht abhängig von Anzahl POIs im System
- Laufzeit: Suchraum $\cdot$ k



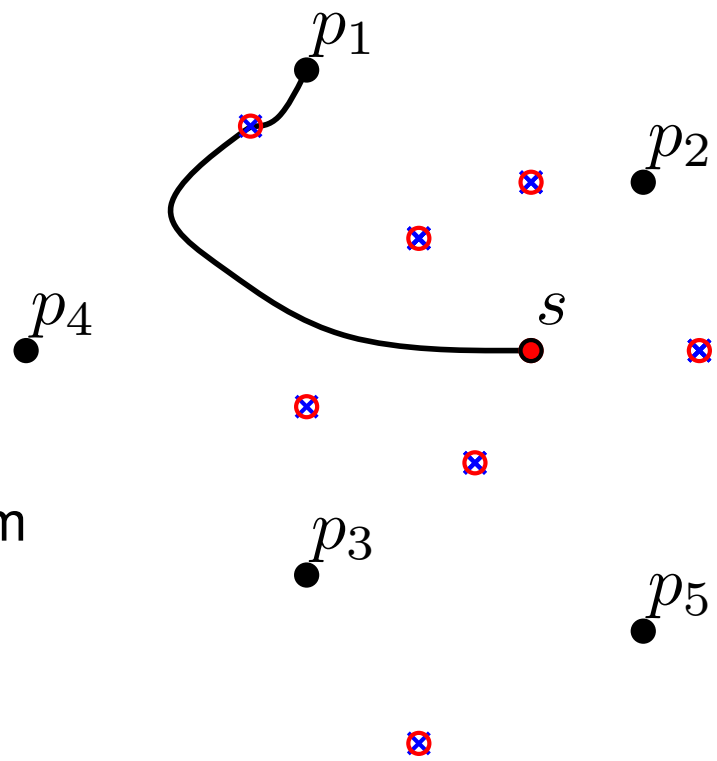
Nächster POI

Szenario:

- Zielknoten sind POIs (z.B. Paketshops)
- finde k nächste POIs von einem Startknoten s

Lösung:

- wie one-to-many
- ordne die buckets pro Knoten auch nach aufsteigender Distanz
- in jedem Bucket müssen nur die k nächsten POIs durchsucht werden
- Laufzeit für POI Query nicht abhängig von Anzahl POIs im System
- Laufzeit: Suchraum $\cdot$ k



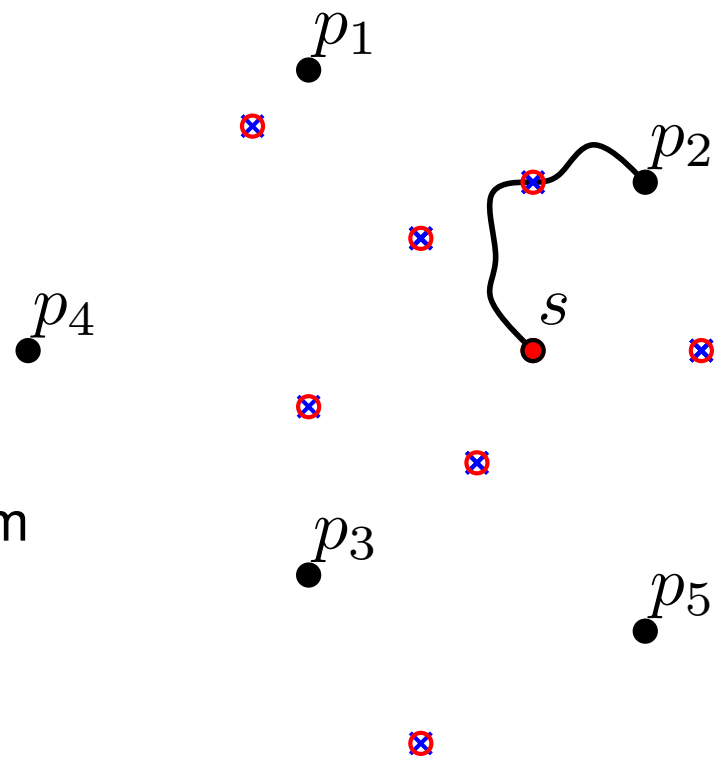
Nächster POI

Szenario:

- Zielknoten sind POIs (z.B. Paketshops)
- finde k nächste POIs von einem Startknoten s

Lösung:

- wie one-to-many
- ordne die buckets pro Knoten auch nach aufsteigender Distanz
- in jedem Bucket müssen nur die k nächsten POIs durchsucht werden
- Laufzeit für POI Query nicht abhängig von Anzahl POIs im System
- Laufzeit: Suchraum $\cdot$ k



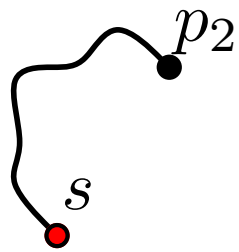
Nächster POI

Szenario:

- Zielknoten sind POIs (z.B. Paketshops)
- finde k nächste POIs von einem Startknoten s

Lösung:

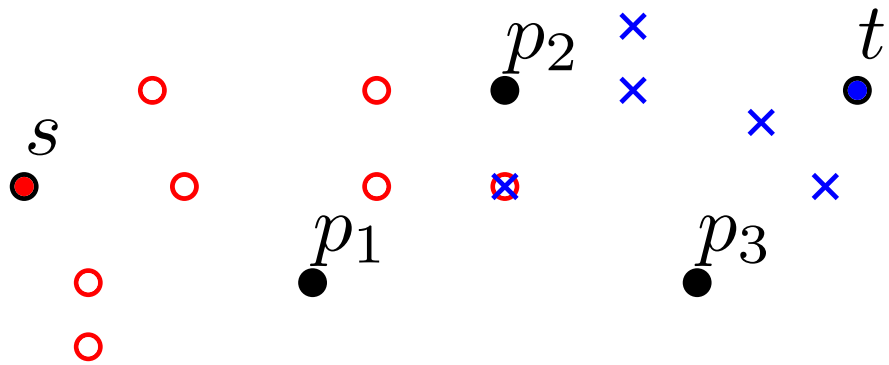
- wie one-to-many
- ordne die buckets pro Knoten auch nach aufsteigender Distanz
- in jedem Bucket müssen nur die k nächsten POIs durchsucht werden
- Laufzeit für POI Query nicht abhängig von Anzahl POIs im System
- Laufzeit: Suchraum $\cdot$ k



Bester Via Knoten

Szenario:

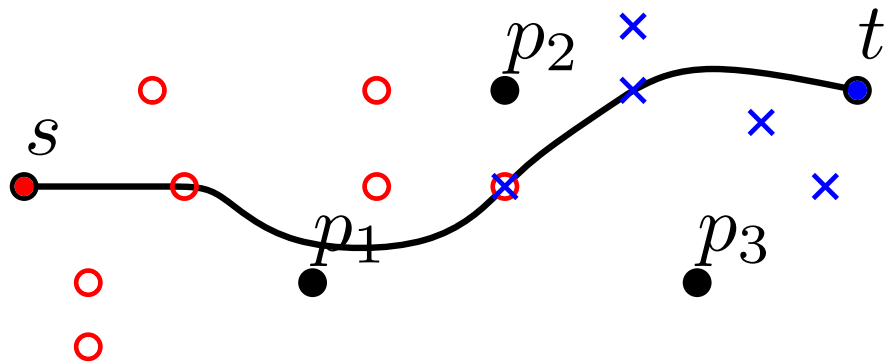
- finde k best Via Knoten POIs von einem Startknoten s zu einem Zielknoten t
- minimiere $\text{dist}(s, p) + \text{dist}(p, t)$ über alle POIs p



Bester Via Knoten

Szenario:

- finde k best Via Knoten POIs von einem Startknoten s zu einem Zielknoten t
- minimiere $\text{dist}(s, p) + \text{dist}(p, t)$ über alle POIs p



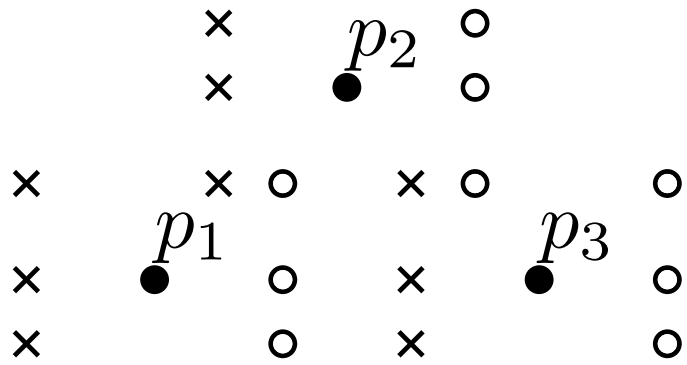
Bester Via Knoten

Szenario:

- finde k best Via Knoten POIs von einem Startknoten s zu einem Zielknoten t
- minimiere $\text{dist}(s, p) + \text{dist}(p, t)$ über alle POIs p

Lösung:

- Vorwärts- und Rückwartssuche von jedem POI
- speicher Kreuzprodukt der beiden Suchräume mit Distanz durch den POI
- Such von s und t: evaluiere jedes Paar
- Laufzeit: Suchraum$^2 \cdot k$



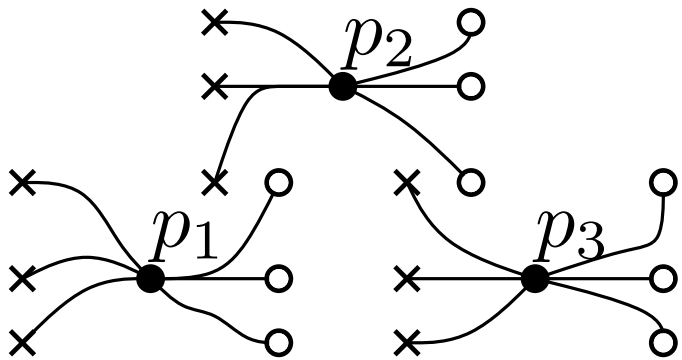
Bester Via Knoten

Szenario:

- finde k best Via Knoten POIs von einem Startknoten s zu einem Zielknoten t
- minimiere $\text{dist}(s, p) + \text{dist}(p, t)$ über alle POIs p

Lösung:

- Vorwärts- und Rückwartssuche von jedem POI
- speicher Kreuzprodukt der beiden Suchräume mit Distanz durch den POI
- Such von s und t: evaluiere jedes Paar
- Laufzeit: Suchraum$^2 \cdot k$



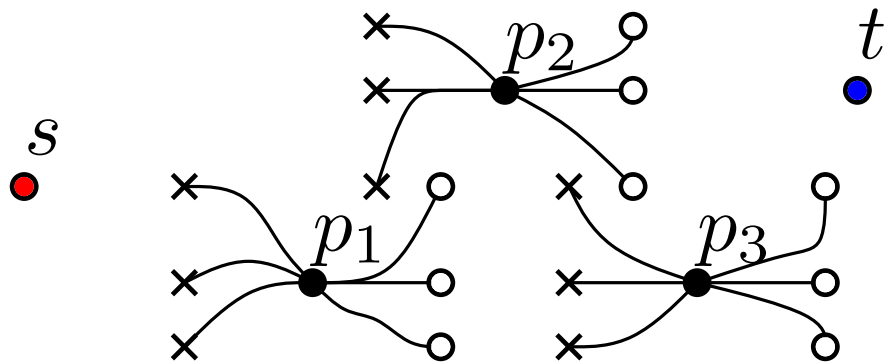
Bester Via Knoten

Szenario:

- finde k best Via Knoten POIs von einem Startknoten s zu einem Zielknoten t
- minimiere $\text{dist}(s, p) + \text{dist}(p, t)$ über alle POIs p

Lösung:

- Vorwärts- und Rückwartssuche von jedem POI
- speicher Kreuzprodukt der beiden Suchräume mit Distanz durch den POI
- Such von s und t: evaluiere jedes Paar
- Laufzeit: Suchraum$^2 \cdot k$



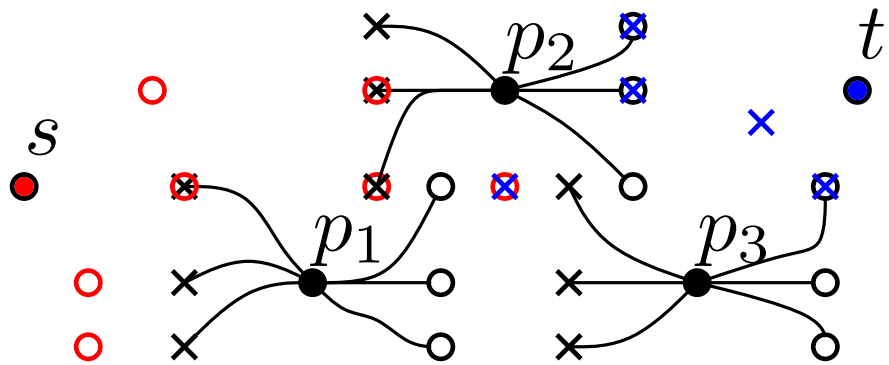
Bester Via Knoten

Szenario:

- finde k best Via Knoten POIs von einem Startknoten s zu einem Zielknoten t
- minimiere $\text{dist}(s, p) + \text{dist}(p, t)$ über alle POIs p

Lösung:

- Vorwärts- und Rückwartssuche von jedem POI
- speicher Kreuzprodukt der beiden Suchräume mit Distanz durch den POI
- Such von s und t: evaluiere jedes Paar
- Laufzeit: Suchraum$^2 \cdot k$



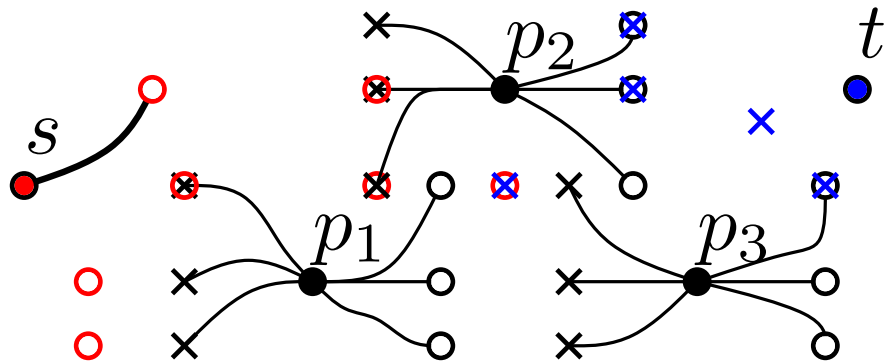
Bester Via Knoten

Szenario:

- finde k best Via Knoten POIs von einem Startknoten s zu einem Zielknoten t
- minimiere $\text{dist}(s, p) + \text{dist}(p, t)$ über alle POIs p

Lösung:

- Vorwärts- und Rückwartssuche von jedem POI
- speicher Kreuzprodukt der beiden Suchräume mit Distanz durch den POI
- Such von s und t: evaluiere jedes Paar
- Laufzeit: Suchraum$^2 \cdot k$



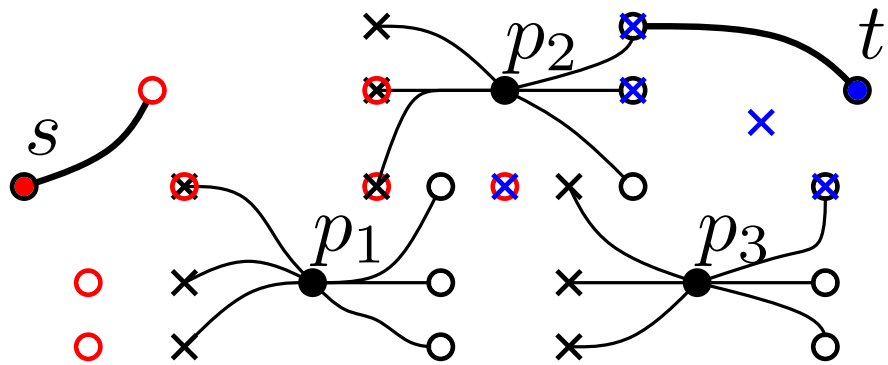
Bester Via Knoten

Szenario:

- finde k best Via Knoten POIs von einem Startknoten s zu einem Zielknoten t
- minimiere $\text{dist}(s, p) + \text{dist}(p, t)$ über alle POIs p

Lösung:

- Vorwärts- und Rückwartssuche von jedem POI
- speicher Kreuzprodukt der beiden Suchräume mit Distanz durch den POI
- Such von s und t: evaluiere jedes Paar
- Laufzeit: Suchraum$^2 \cdot k$



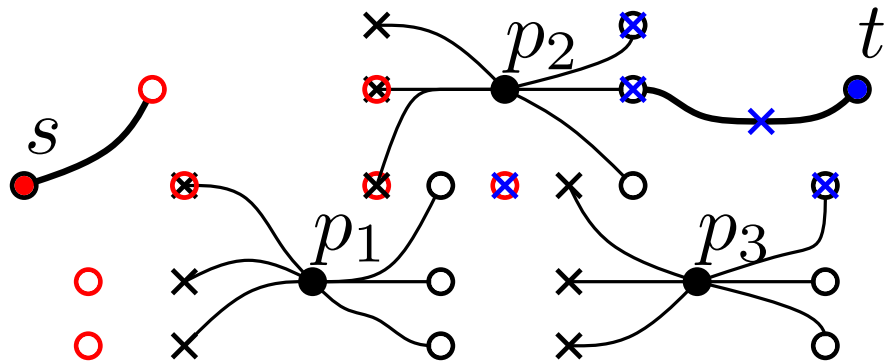
Bester Via Knoten

Szenario:

- finde k best Via Knoten POIs von einem Startknoten s zu einem Zielknoten t
- minimiere $\text{dist}(s, p) + \text{dist}(p, t)$ über alle POIs p

Lösung:

- Vorwärts- und Rückwartssuche von jedem POI
- speicher Kreuzprodukt der beiden Suchräume mit Distanz durch den POI
- Such von s und t: evaluiere jedes Paar
- Laufzeit: Suchraum$^2 \cdot k$



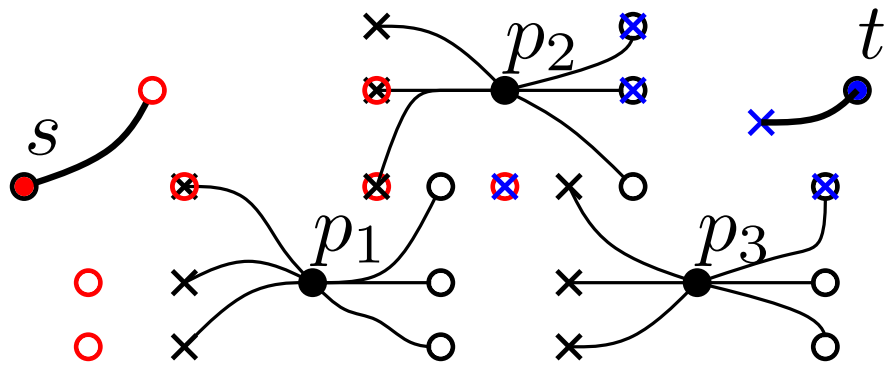
Bester Via Knoten

Szenario:

- finde k best Via Knoten POIs von einem Startknoten s zu einem Zielknoten t
- minimiere $\text{dist}(s, p) + \text{dist}(p, t)$ über alle POIs p

Lösung:

- Vorwärts- und Rückwartssuche von jedem POI
- speicher Kreuzprodukt der beiden Suchräume mit Distanz durch den POI
- Such von s und t: evaluiere jedes Paar
- Laufzeit: Suchraum$^2 \cdot k$



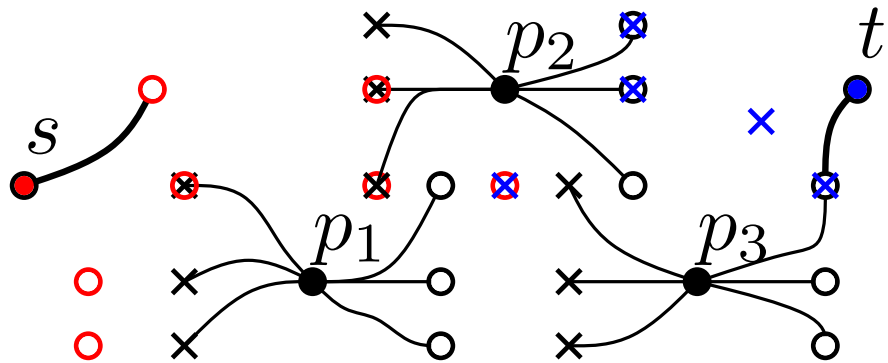
Bester Via Knoten

Szenario:

- finde k best Via Knoten POIs von einem Startknoten s zu einem Zielknoten t
- minimiere $\text{dist}(s, p) + \text{dist}(p, t)$ über alle POIs p

Lösung:

- Vorwärts- und Rückwartssuche von jedem POI
- speicher Kreuzprodukt der beiden Suchräume mit Distanz durch den POI
- Such von s und t: evaluiere jedes Paar
- Laufzeit: Suchraum$^2 \cdot k$



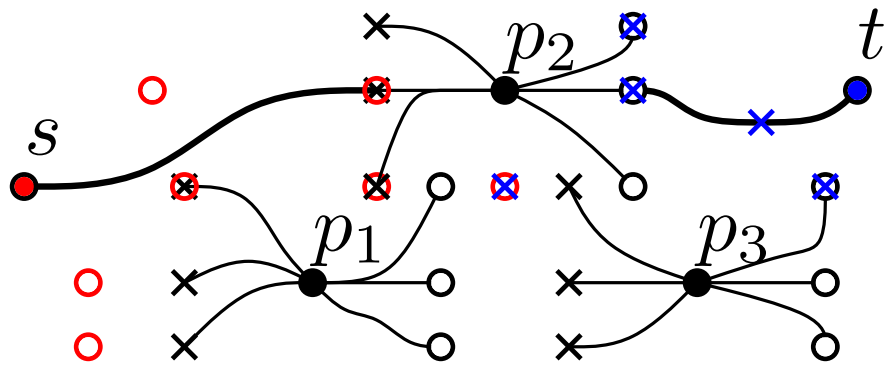
Bester Via Knoten

Szenario:

- finde k best Via Knoten POIs von einem Startknoten s zu einem Zielknoten t
- minimiere $\text{dist}(s, p) + \text{dist}(p, t)$ über alle POIs p

Lösung:

- Vorwärts- und Rückwartssuche von jedem POI
- speicher Kreuzprodukt der beiden Suchräume mit Distanz durch den POI
- Such von s und t: evaluiere jedes Paar
- Laufzeit: Suchraum$^2 \cdot k$



Wiederholung: HLDB SQL Query

Tabellen `forward` und `backward`
mit Spalten `node`, `hub`, `dist`

Algorithm: SQL_DIST

Input: source $s \in V$, target $t \in V$

```
1 SELECT
2       MIN(forward.dist+backward.dist)
3 FROM forward,backward
4 WHERE
5       forward.node = s AND
6       backward.node = t AND
7       forward.hub = backward.hub
```

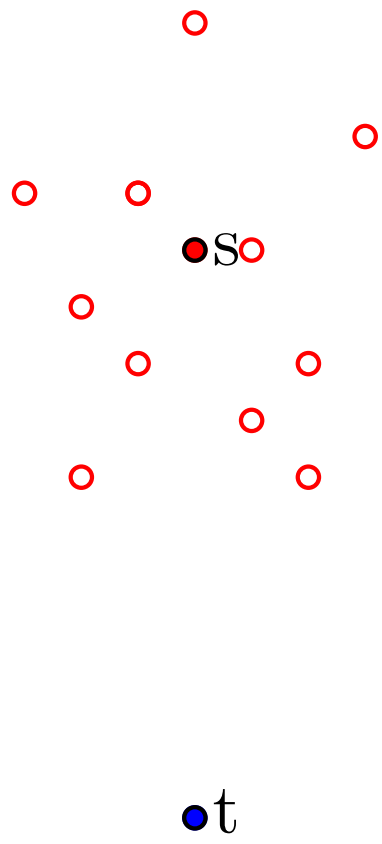
Wiederholung: HLDB SQL Query

Tabellen `forward` und `backward`
mit Spalten `node`, `hub`, `dist`

Algorithm: SQL_DIST

Input: source $s \in V$, target $t \in V$

```
1 SELECT
2       MIN(forward.dist+backward.dist)
3 FROM forward,backward
4 WHERE
5       forward.node = s AND
6       backward.node = t AND
7       forward.hub = backward.hub
```



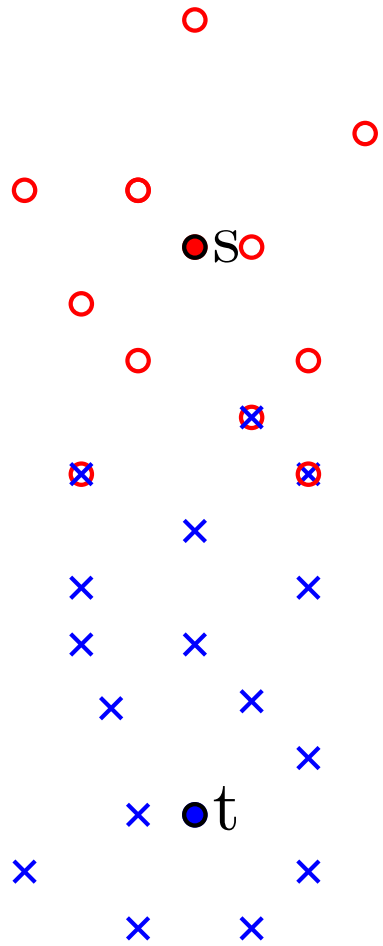
Wiederholung: HLDB SQL Query

Tabellen `forward` und `backward`
mit Spalten `node`, `hub`, `dist`

Algorithm: SQL_DIST

Input: source $s \in V$, target $t \in V$

```
1 SELECT
2       MIN(forward.dist+backward.dist)
3 FROM forward,backward
4 WHERE
5       forward.node = s AND
6       backward.node = t AND
7       forward.hub = backward.hub
```



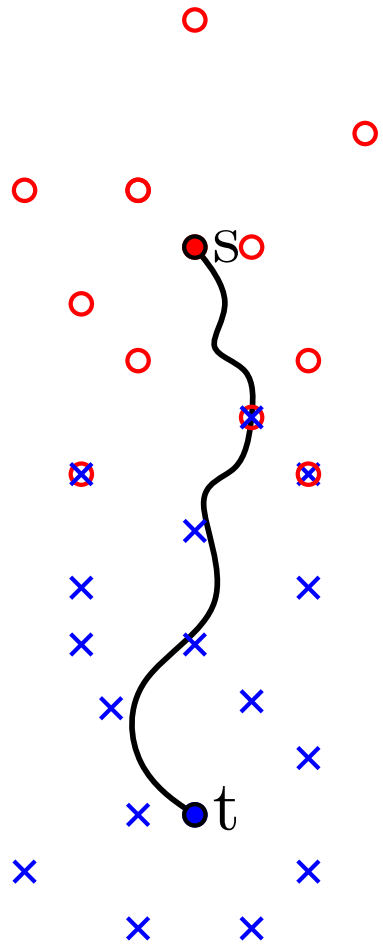
Wiederholung: HLDB SQL Query

Tabellen `forward` und `backward`
mit Spalten `node`, `hub`, `dist`

Algorithm: SQL_DIST

Input: source $s \in V$, target $t \in V$

```
1 SELECT
2       MIN(forward.dist+backward.dist)
3 FROM forward,backward
4 WHERE
5       forward.node = s AND
6       backward.node = t AND
7       forward.hub = backward.hub
```



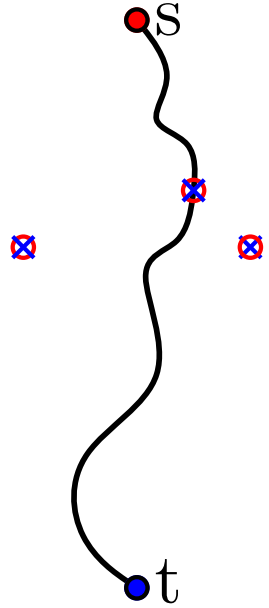
Wiederholung: HLDB SQL Query

Tabellen `forward` und `backward`
mit Spalten `node`, `hub`, `dist`

Algorithm: SQL_DIST

Input: source $s \in V$, target $t \in V$

```
1 SELECT
2       MIN(forward.dist+backward.dist)
3 FROM forward,backward
4 WHERE
5       forward.node = s AND
6       backward.node = t AND
7       forward.hub = backward.hub
```



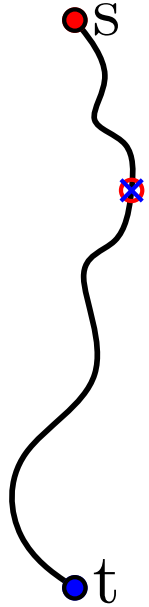
Wiederholung: HLDB SQL Query

Tabellen `forward` und `backward`
mit Spalten `node`, `hub`, `dist`

Algorithm: SQL_DIST

Input: source $s \in V$, target $t \in V$

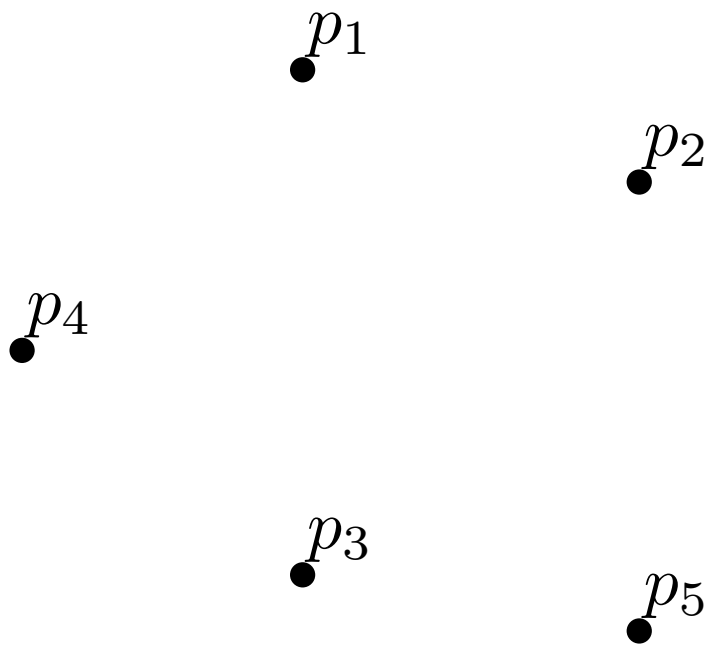
```
1 SELECT
2       MIN(forward.dist+backward.dist)
3 FROM forward,backward
4 WHERE
5       forward.node = s AND
6       backward.node = t AND
7       forward.hub = backward.hub
```



POI Anfragen in SQL

Idee:

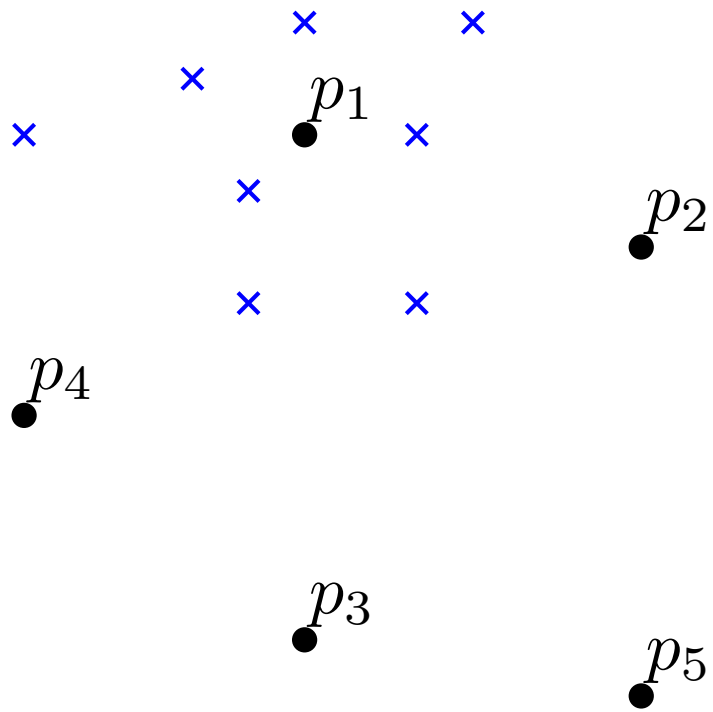
- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`



POI Anfragen in SQL

Idee:

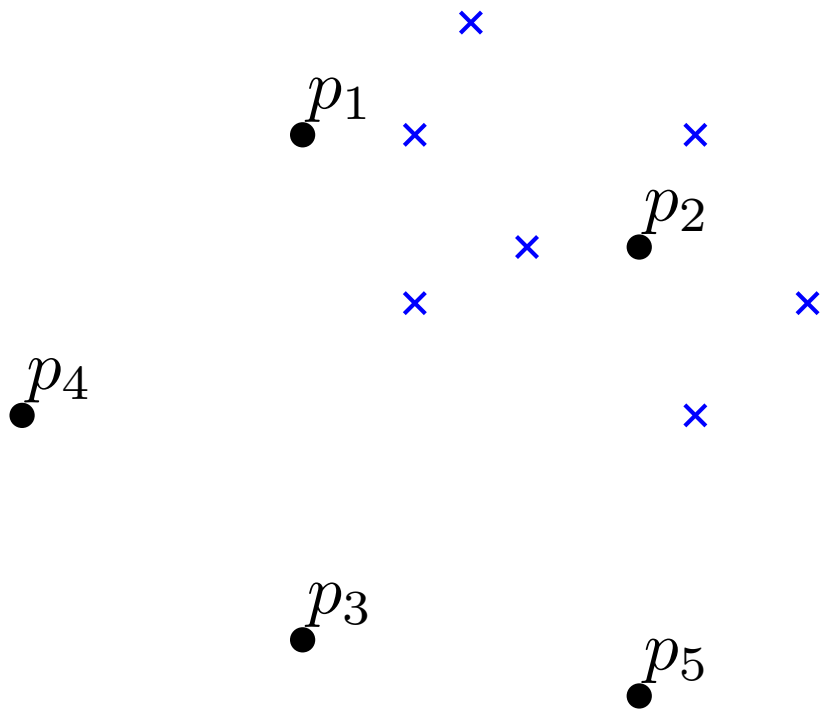
- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`



POI Anfragen in SQL

Idee:

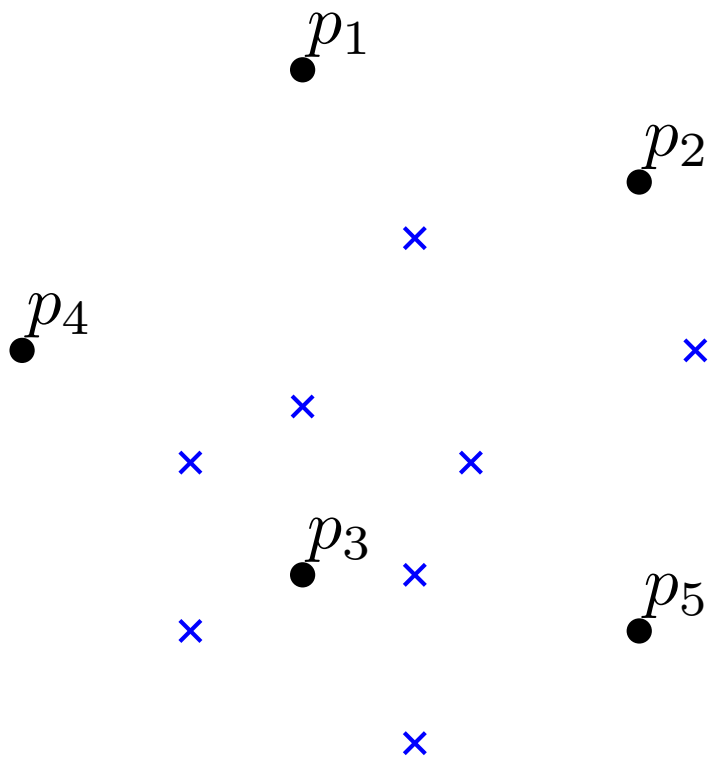
- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`



POI Anfragen in SQL

Idee:

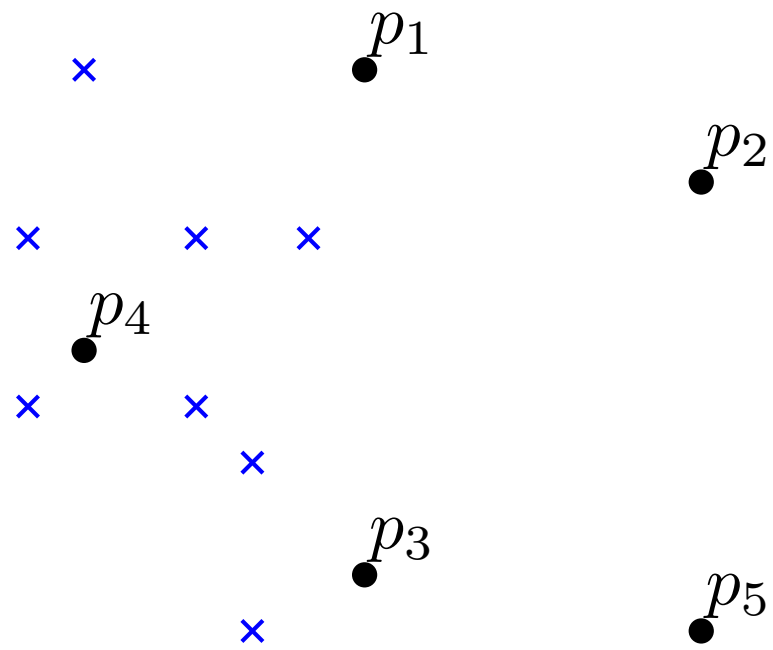
- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`



POI Anfragen in SQL

Idee:

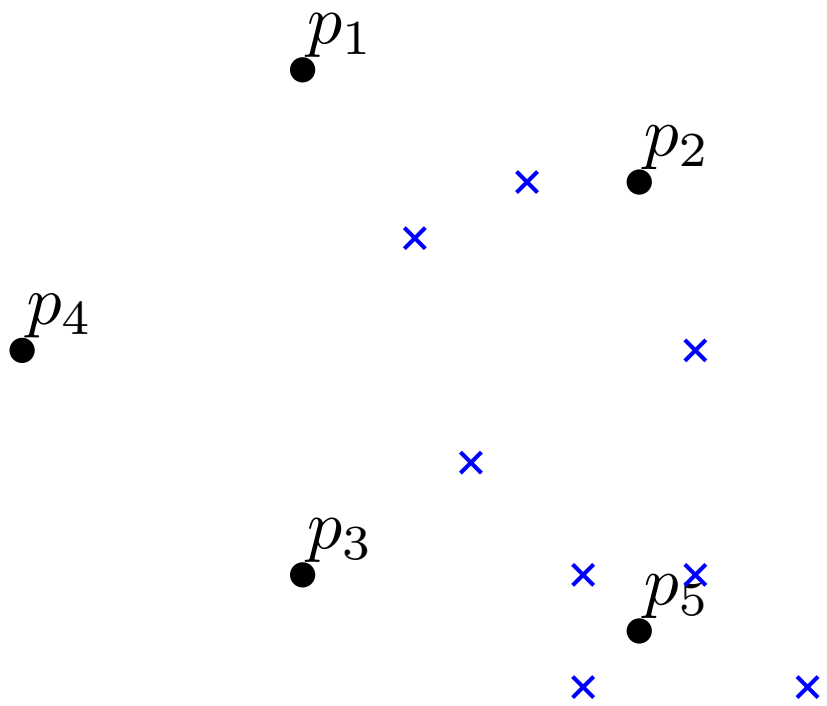
- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`



POI Anfragen in SQL

Idee:

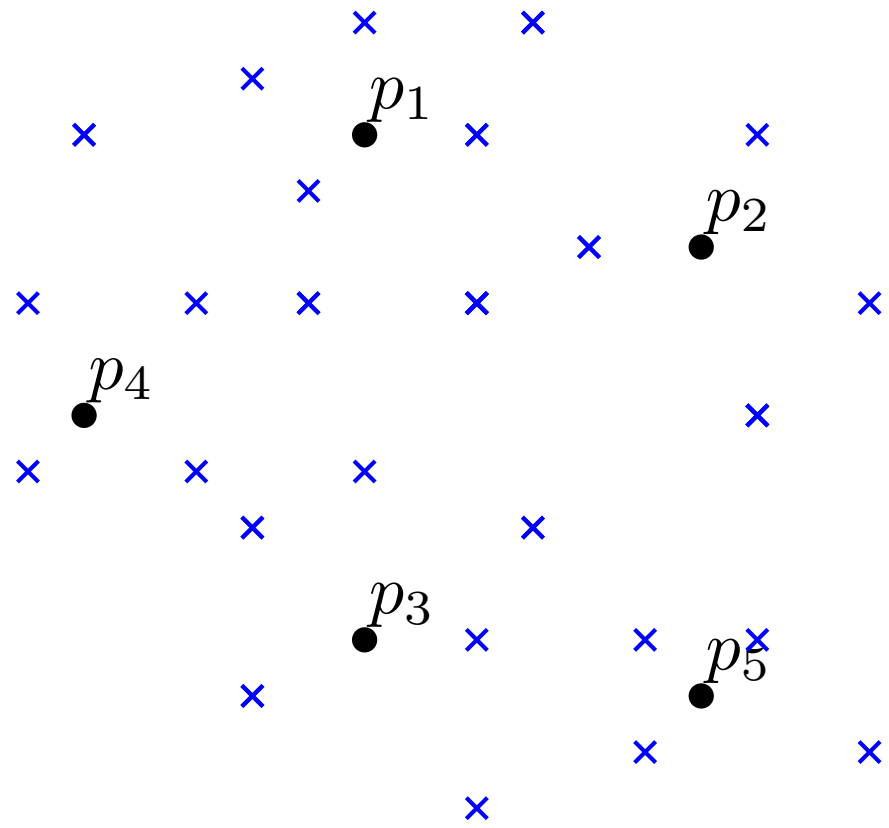
- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`



POI Anfragen in SQL

Idee:

- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`



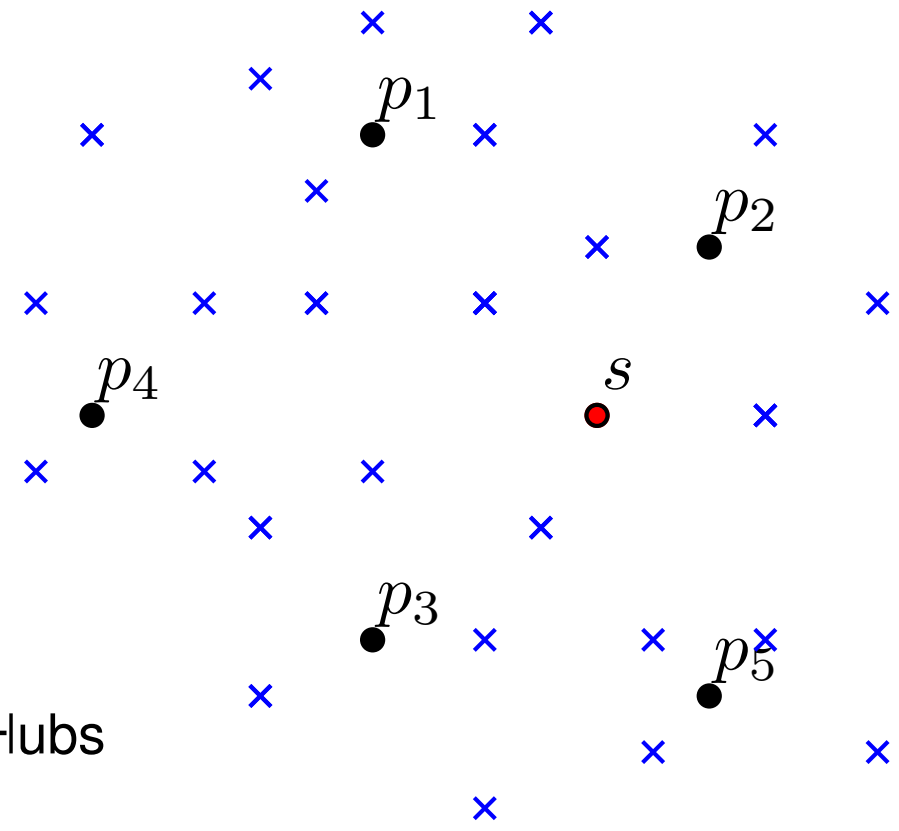
POI Anfragen in SQL

Idee:

- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`

Query:

- iteriert über alle ausgehenden Hubs
- für jeden Hub werden nur die (k) nächsten POIs betrachtet
- Antwort: die k insgesamt nächsten POIs



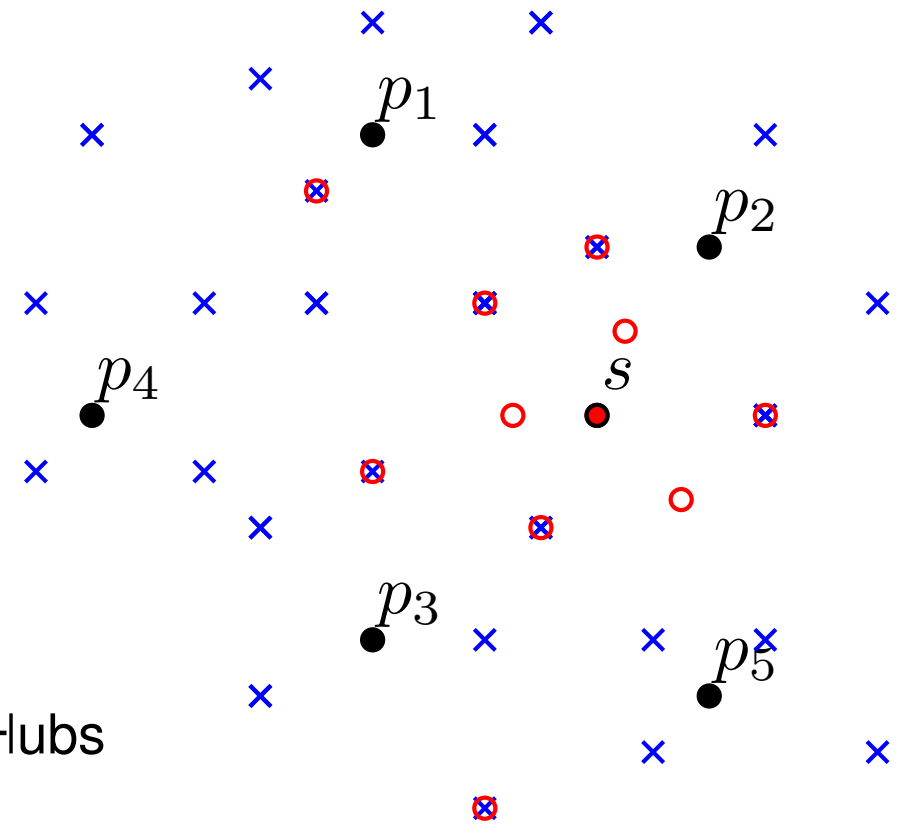
POI Anfragen in SQL

Idee:

- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`

Query:

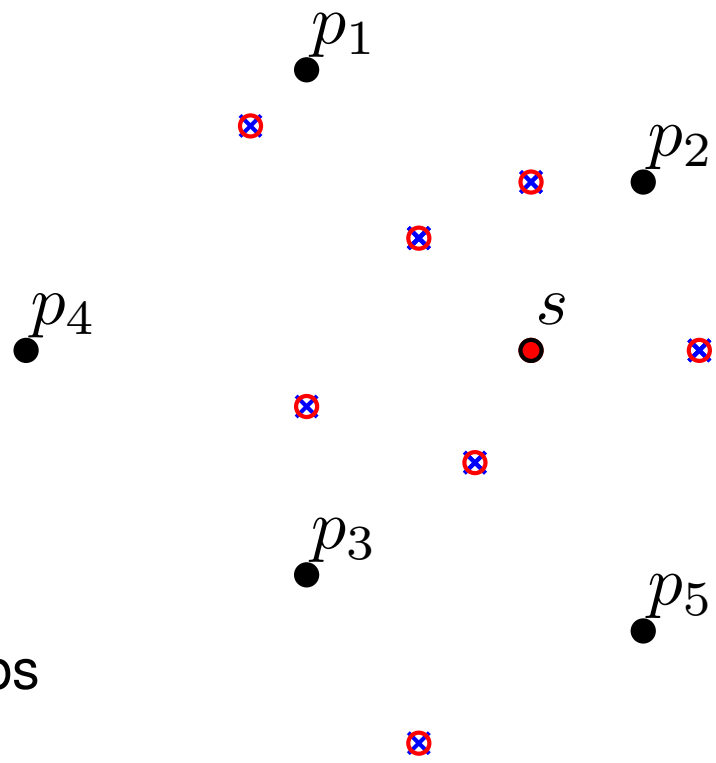
- iteriert über alle ausgehenden Hubs
- für jeden Hub werden nur die (k) nächsten POIs betrachtet
- Antwort: die k insgesamt nächsten POIs



POI Anfragen in SQL

Idee:

- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`



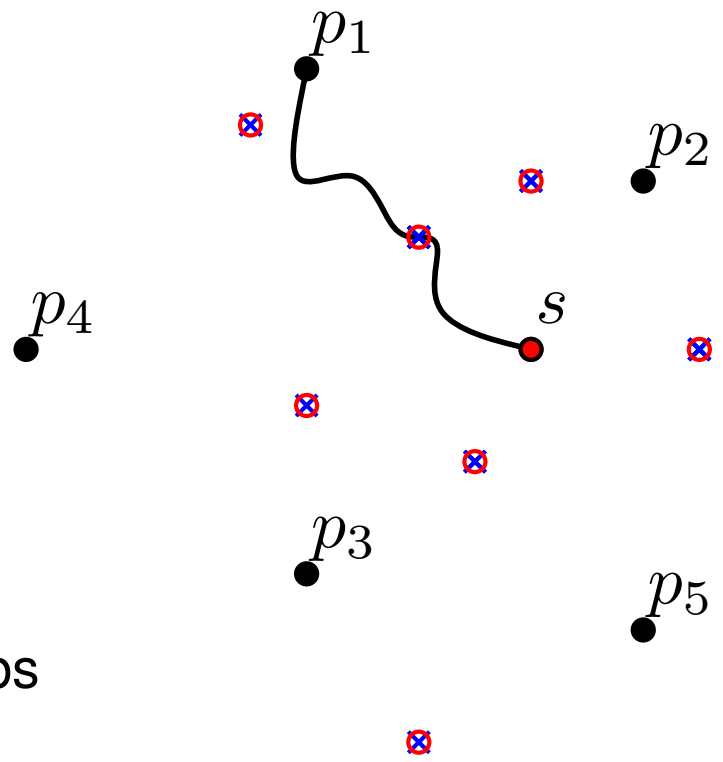
Query:

- iteriert über alle ausgehenden Hubs
- für jeden Hub werden nur die (k) nächsten POIs betrachtet
- Antwort: die k insgesamt nächsten POIs

POI Anfragen in SQL

Idee:

- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`



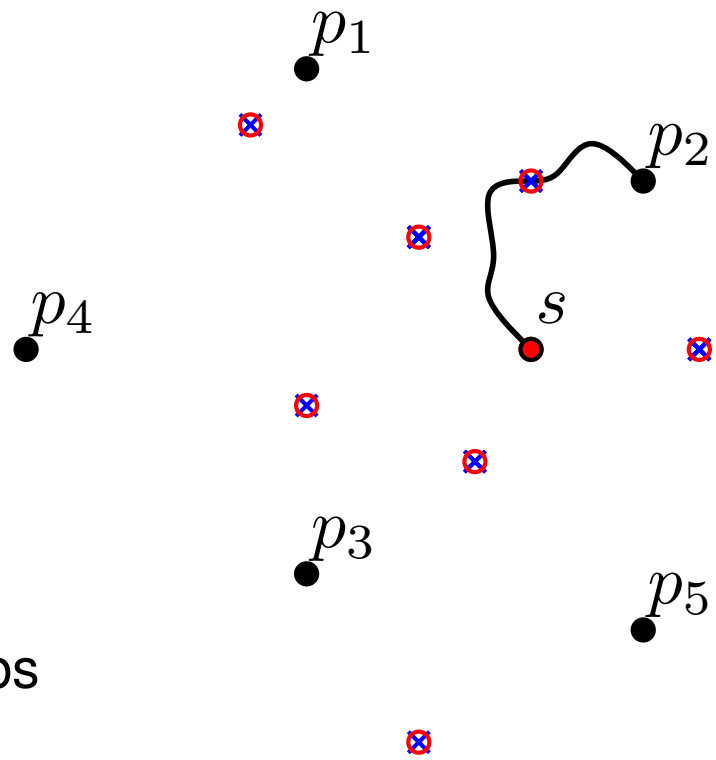
Query:

- iteriert über alle ausgehenden Hubs
- für jeden Hub werden nur die (k) nächsten POIs betrachtet
- Antwort: die k insgesamt nächsten POIs

POI Anfragen in SQL

Idee:

- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`



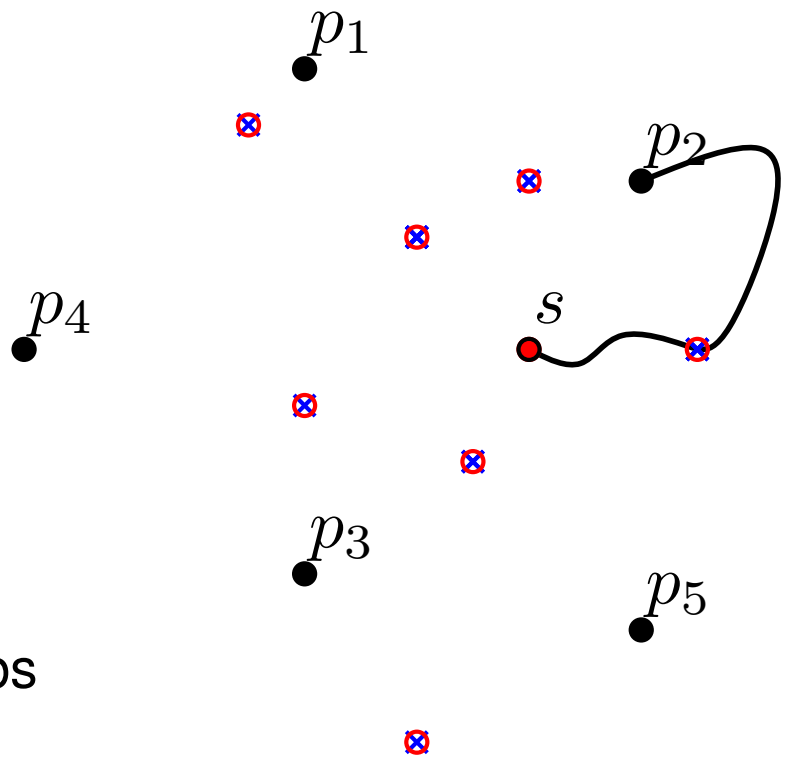
Query:

- iteriert über alle ausgehenden Hubs
- für jeden Hub werden nur die (k) nächsten POIs betrachtet
- Antwort: die k insgesamt nächsten POIs

POI Anfragen in SQL

Idee:

- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`



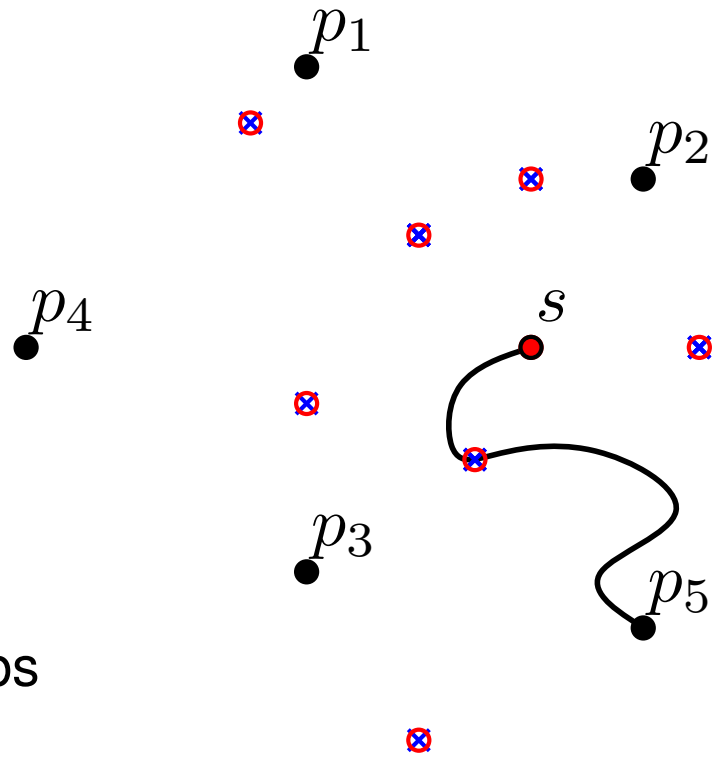
Query:

- iteriert über alle ausgehenden Hubs
- für jeden Hub werden nur die (k) nächsten POIs betrachtet
- Antwort: die k insgesamt nächsten POIs

POI Anfragen in SQL

Idee:

- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`



Query:

- iteriert über alle ausgehenden Hubs
- für jeden Hub werden nur die (k) nächsten POIs betrachtet
- Antwort: die k insgesamt nächsten POIs

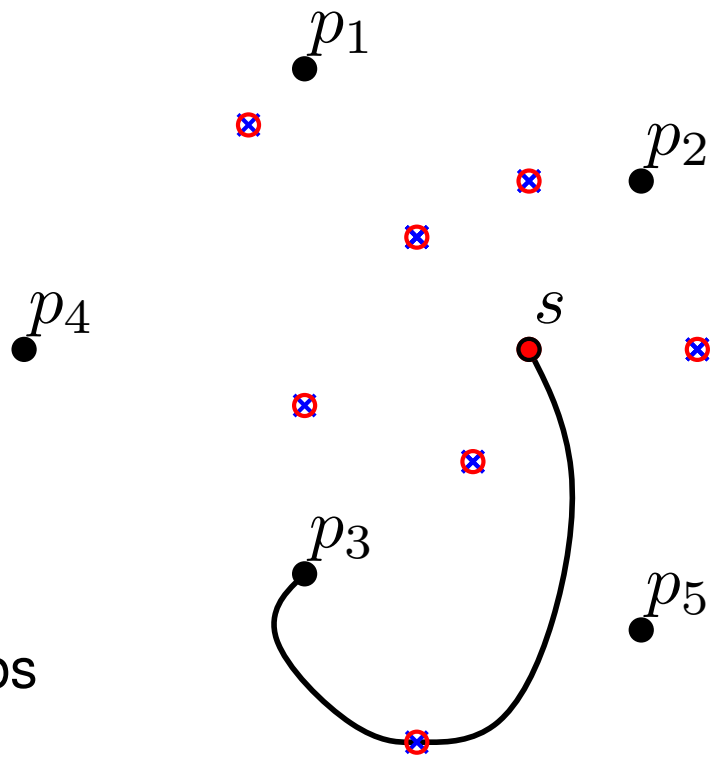
POI Anfragen in SQL

Idee:

- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`

Query:

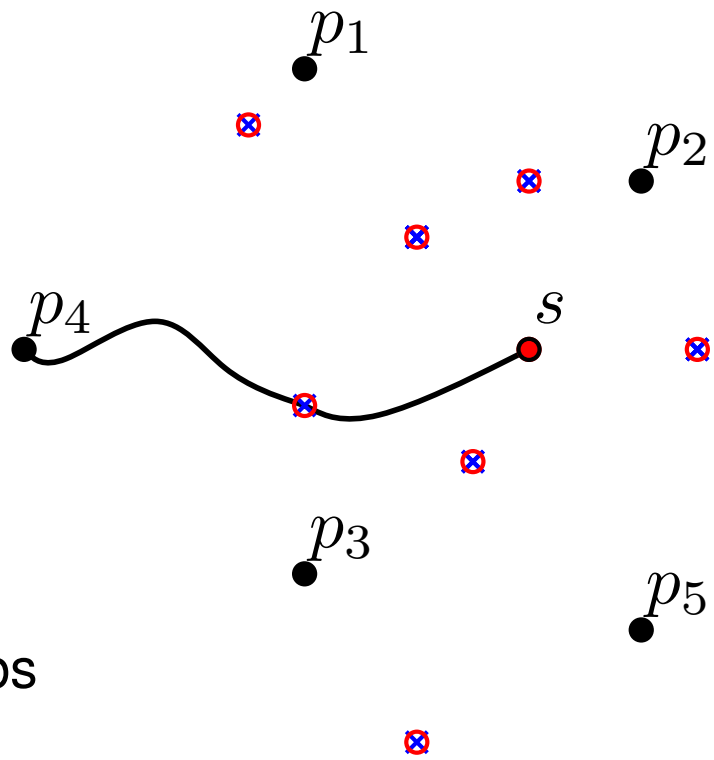
- iteriert über alle ausgehenden Hubs
- für jeden Hub werden nur die (k) nächsten POIs betrachtet
- Antwort: die k insgesamt nächsten POIs



POI Anfragen in SQL

Idee:

- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`



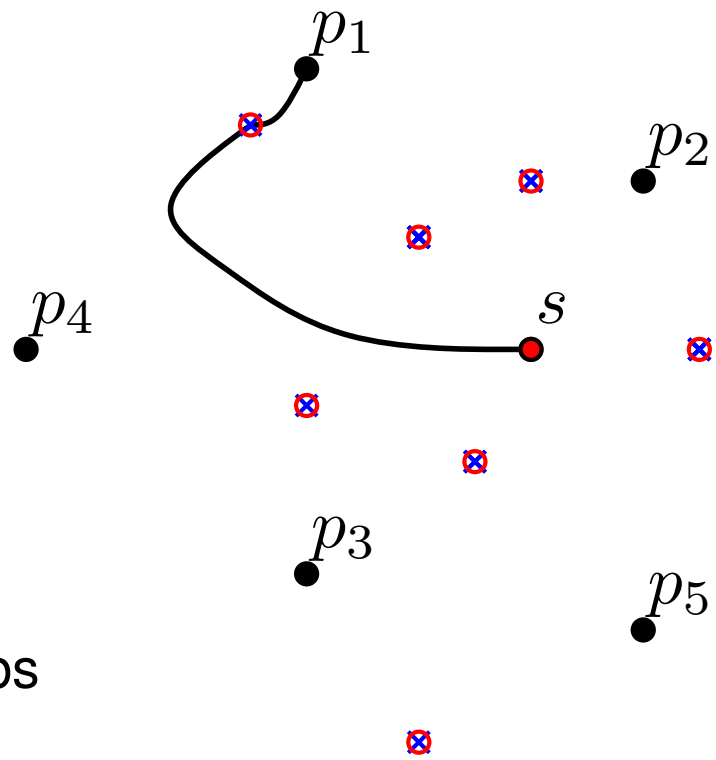
Query:

- iteriert über alle ausgehenden Hubs
- für jeden Hub werden nur die (k) nächsten POIs betrachtet
- Antwort: die k insgesamt nächsten POIs

POI Anfragen in SQL

Idee:

- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`



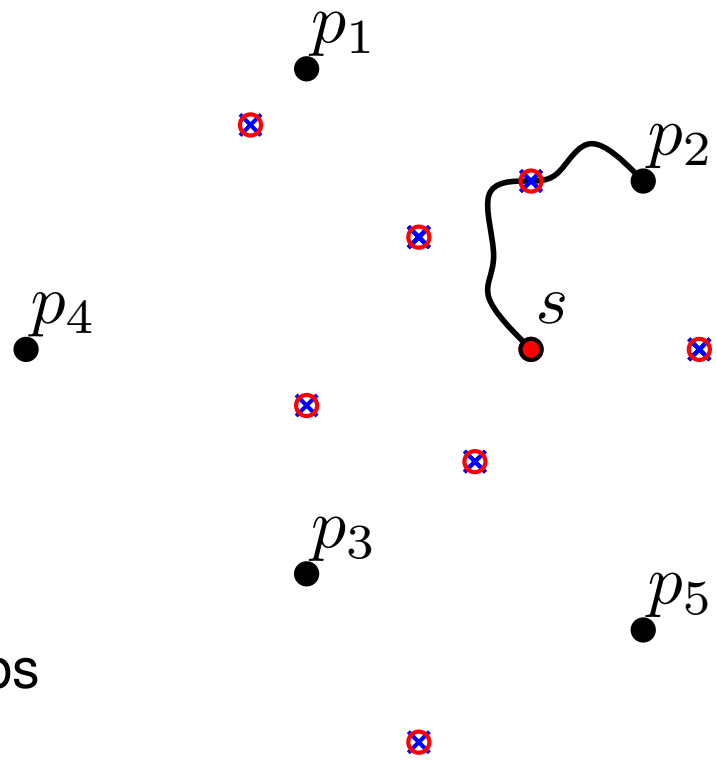
Query:

- iteriert über alle ausgehenden Hubs
- für jeden Hub werden nur die (k) nächsten POIs betrachtet
- Antwort: die k insgesamt nächsten POIs

POI Anfragen in SQL

Idee:

- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`

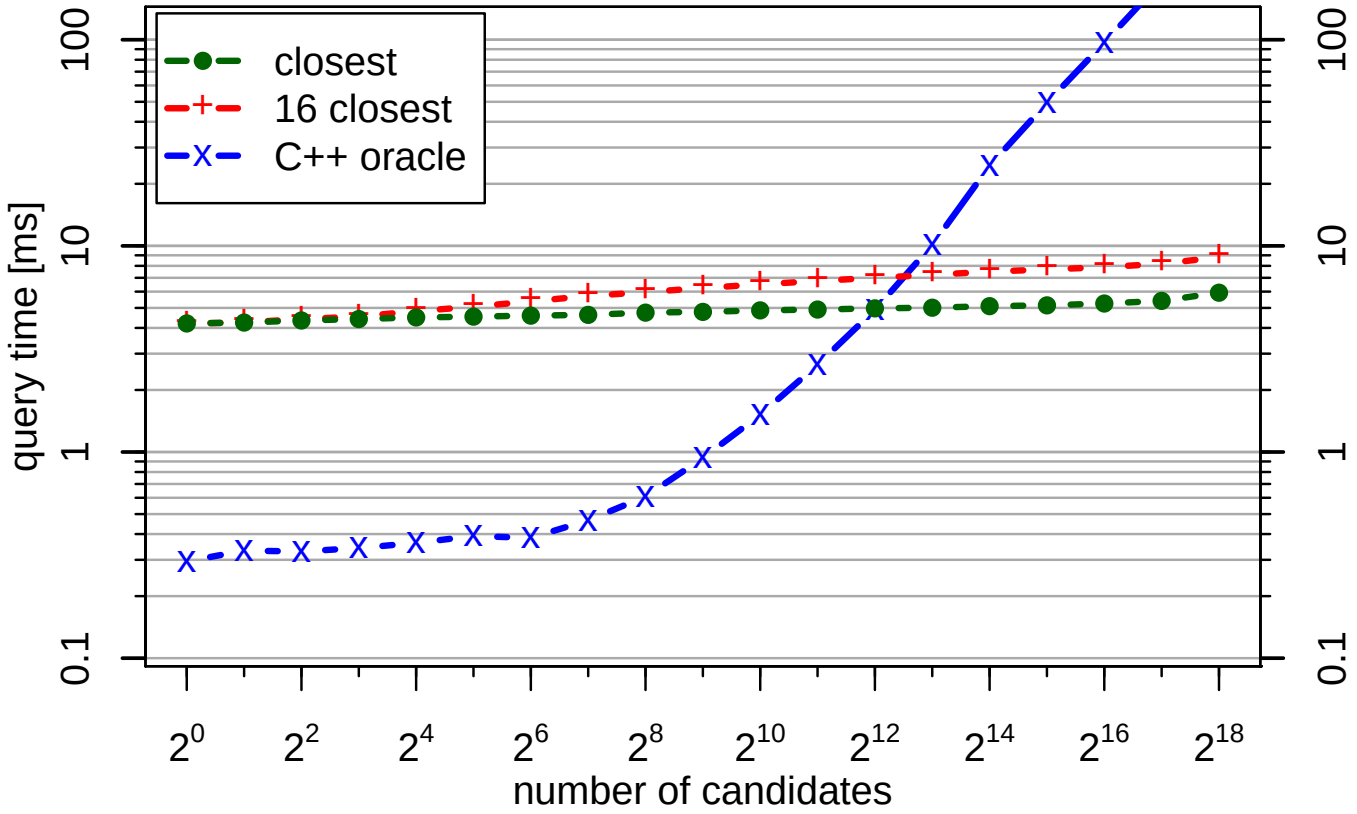


Query:

- iteriert über alle ausgehenden Hubs
- für jeden Hub werden nur die (k) nächsten POIs betrachtet
- Antwort: die k insgesamt nächsten POIs

Ergebnisse POI

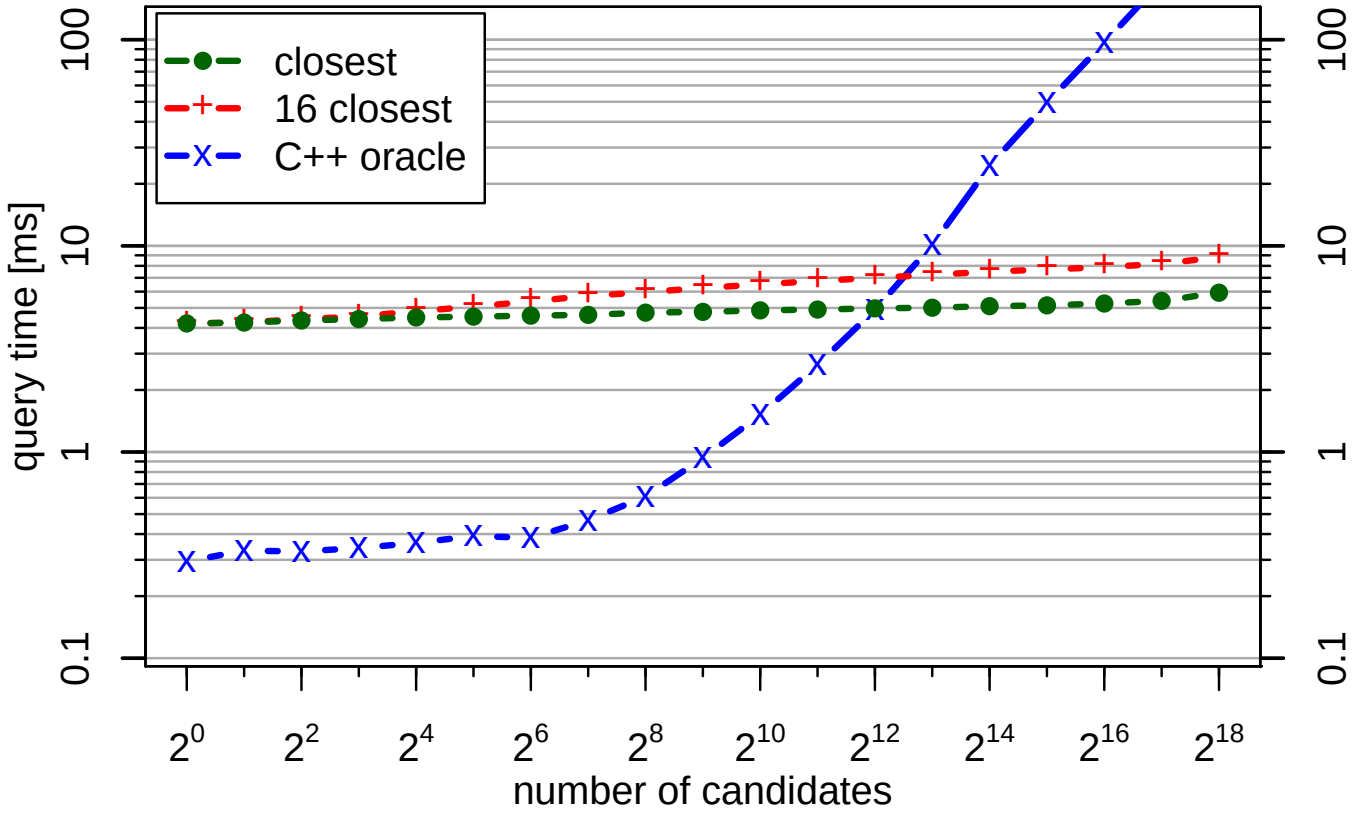
Setup: verschiedene Anzahl POIs, zufällig gewählt



- externes Punkt-zu-Punkt Orakel: skaliert schlecht

Ergebnisse POI

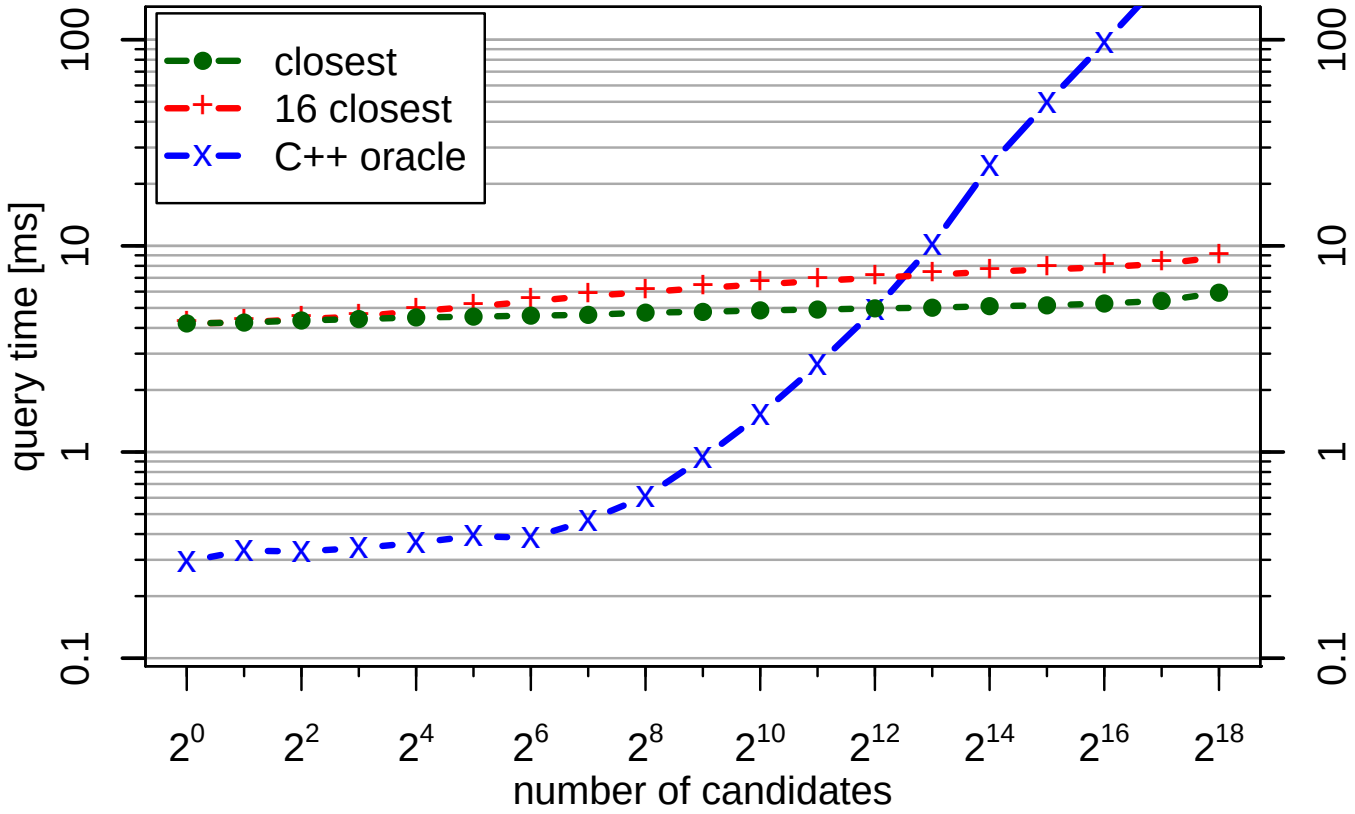
Setup: verschiedene Anzahl POIs, zufällig gewählt



- externes Punkt-zu-Punkt Orakel: skaliert schlecht
- SQL Anfragen unabhängig von Anzahl POIs

Ergebnisse POI

Setup: verschiedene Anzahl POIs, zufällig gewählt



- externes Punkt-zu-Punkt Orakel: skaliert schlecht
- SQL Anfragen unabhängig von Anzahl POIs
- weitere Constraints einfach (“jetzt geöffnet”)

Zusammenfassung

- one-to-many shortest paths
- many-to-many
- POI Anfragen
- bester Via Knoten Anfragen
- Location Services in SQL

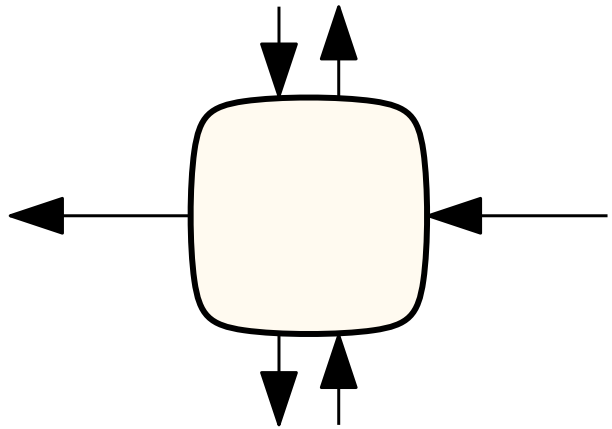
Erweiterte Szenarien

- Abbiegekosten
- Dynamische Routenplanung
- Mobiles Szenario

Abbiegeverbote/-kosten

bisher:

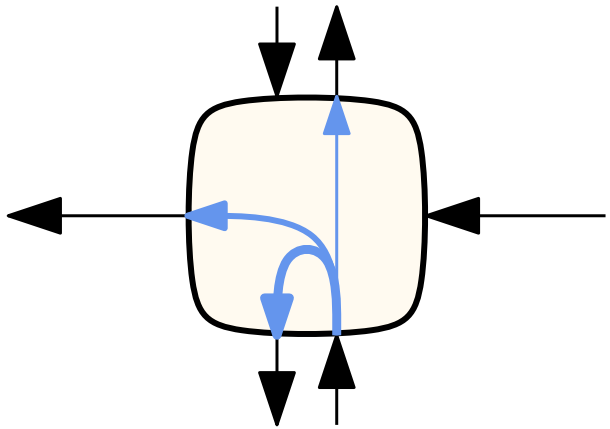
- Kreuzungen $\rightarrow$ Knoten
- Strassen $\rightarrow$ Kanten



Abbiegeverbote/-kosten

bisher:

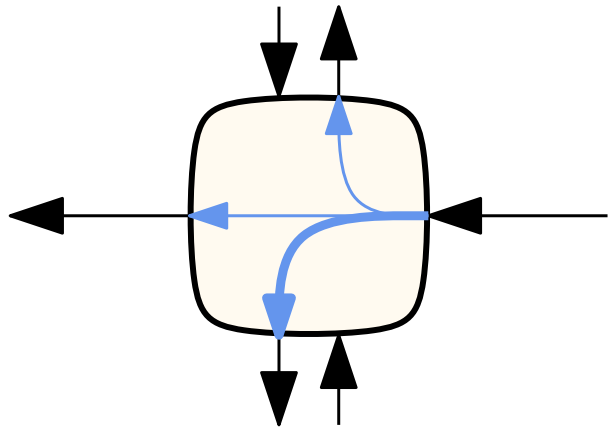
- Kreuzungen $\rightarrow$ Knoten
- Strassen $\rightarrow$ Kanten



aber:

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

Abbiegeverbote/-kosten



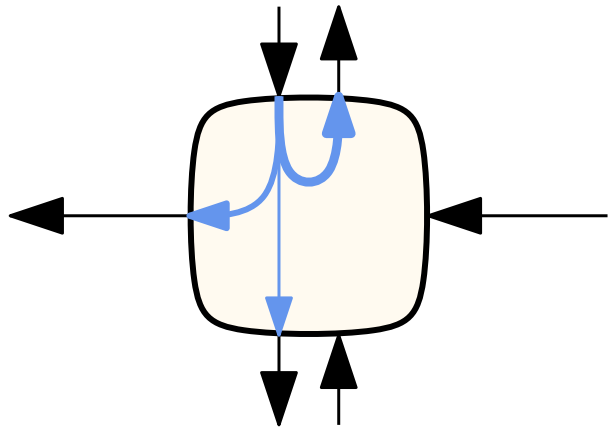
bisher:

- 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

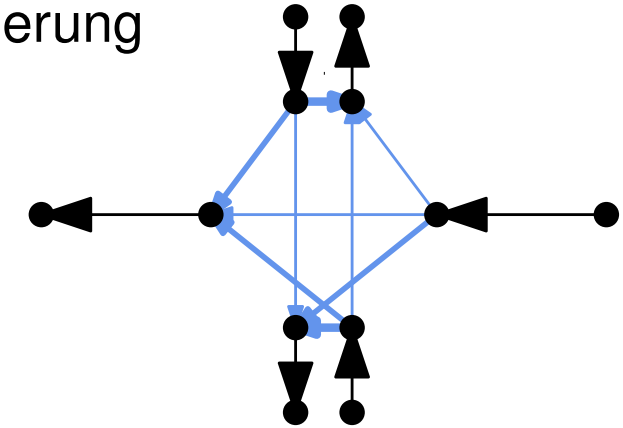
aber:

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

Modellierung

Möglichkeit I:

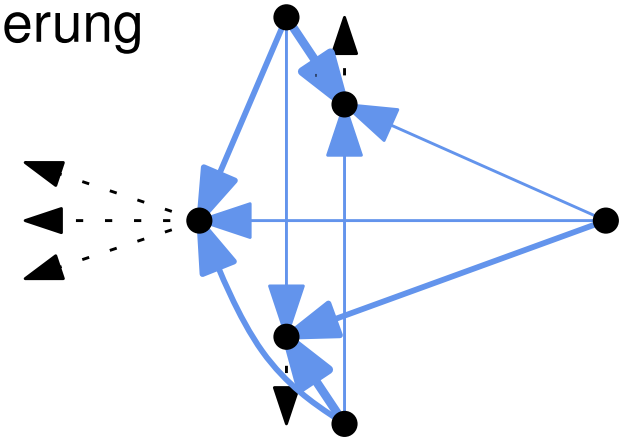
- Vergrössern des Graphen durch Ausmodellierung
- kantenbasierter Graph da
  - Strassen $\rightarrow$ Knoten
  - Turns $\rightarrow$ Kanten
- redundante Information



Modellierung

Möglichkeit I:

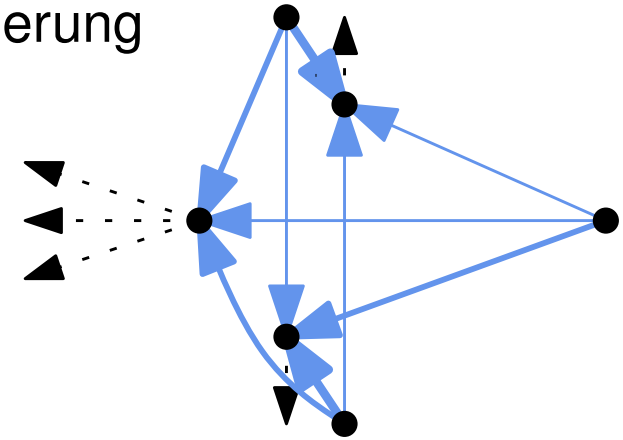
- Vergrössern des Graphen durch Ausmodellierung
- kantenbasierter Graph da
  - Strassen $\rightarrow$ Knoten
  - Turns $\rightarrow$ Kanten
- redundante Information
- entferne einen Knoten pro Strasse



Modellierung

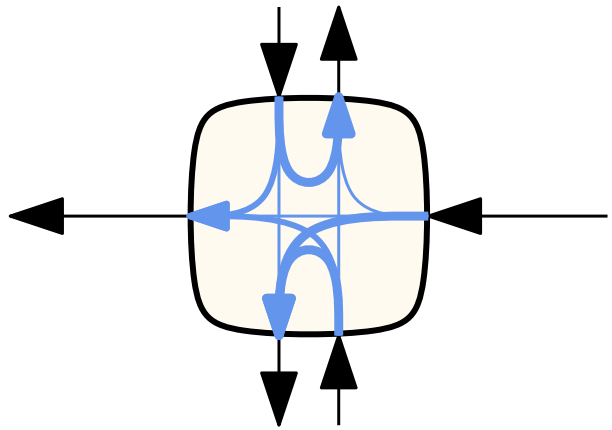
Möglichkeit I:

- Vergrössern des Graphen durch Ausmodellierung
- kantenbasierter Graph da
  - Strassen $\rightarrow$ Knoten
  - Turns $\rightarrow$ Kanten
- redundante Information
- entferne einen Knoten pro Strasse



Möglichkeit II:

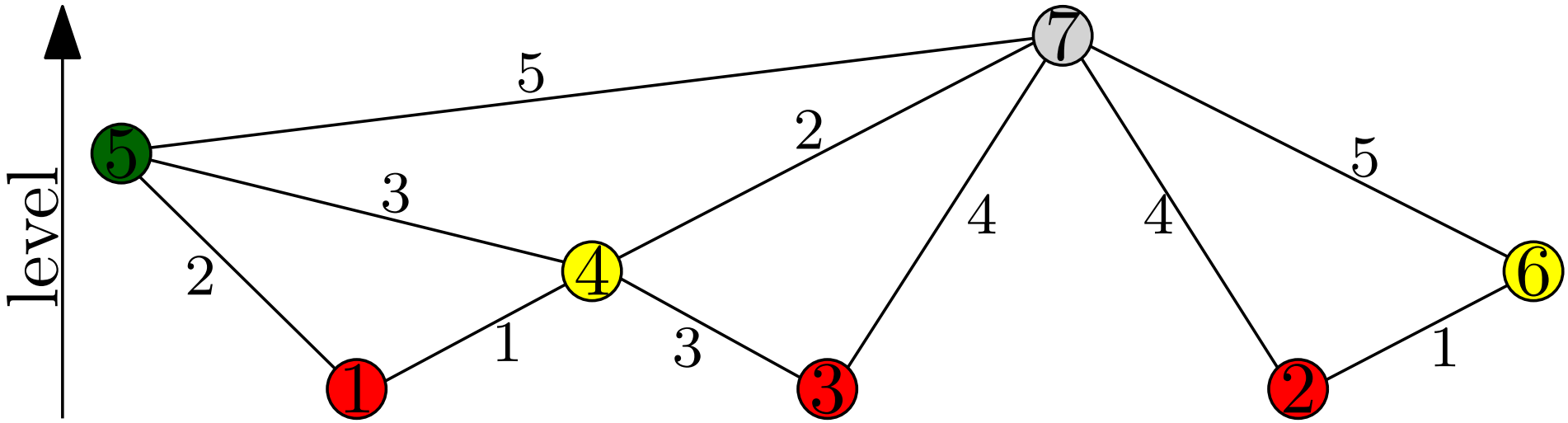
- behalte Kreuzungen als Knoten
- speicher Abbiegetabelle
  Abb. Eingangs- $\times$ Ausgangspunkte $\rightarrow$ Kosten
- Beobachtung: viele Knoten haben die gleiche Abbiegetabelle
- also speicher jede Tabelle einmal, Knoten speichern Tabellen-ID



Wdh: Contraction Hierarchies

Preprocessing:

- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu
- Levelzuordnung (ca. 150 in Strassennetzwerken)



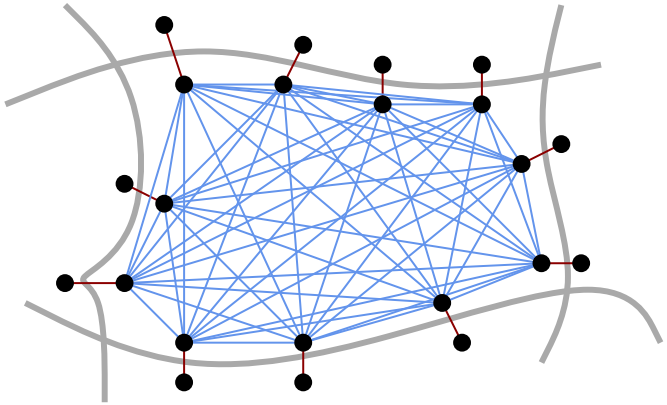
Wdh: Multi Level Dijkstra

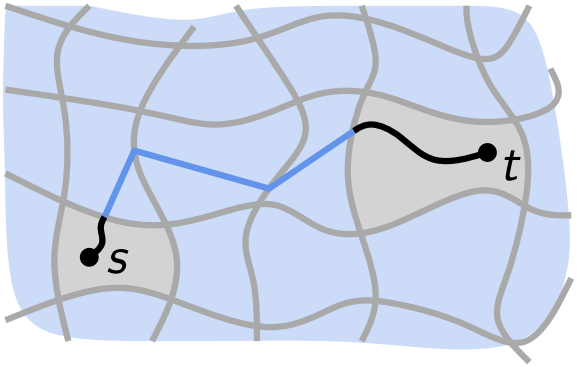
Idee:

- 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

Kantenbasierter Graph

Dijkstra:

- funktioniert ohne Anpassung
- mehr Knoten zu scannen
- Faktor 3-4 langsamer

Kantenbasierter Graph

Dijkstra:

- funktioniert ohne Anpassung
- mehr Knoten zu scannen
- Faktor 3-4 langsamer

CH

- funktioniert ohne Anpassung
- aber grössere Anzahl Knoten/Kanten erhöht Vorberechungszeit

Kantenbasierter 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

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Kompaktes Modell

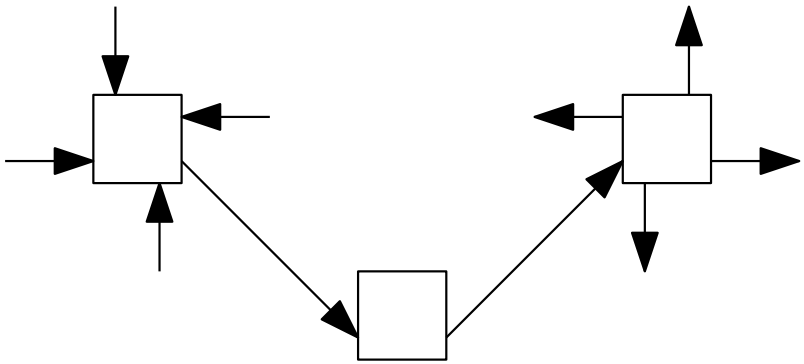
CH

- Zeugensuche wird komplizierter

Kompaktes Modell

CH

- Zeugensuche wird komplizierter
  - für jedes Paar eingehender und ausgehender Kanten muss eine Zeugensuche durchgeführt werden

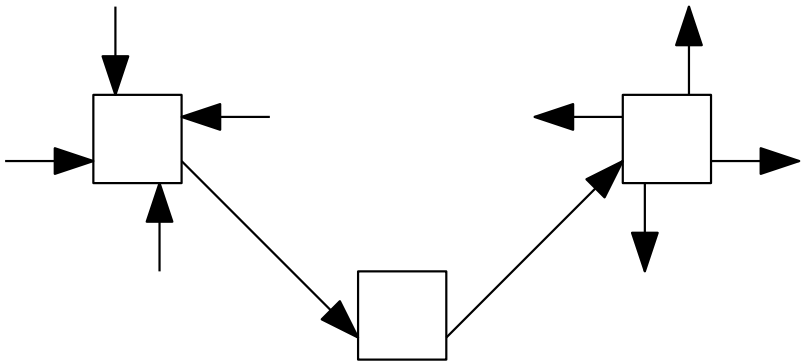


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Kompaktes Modell

CH

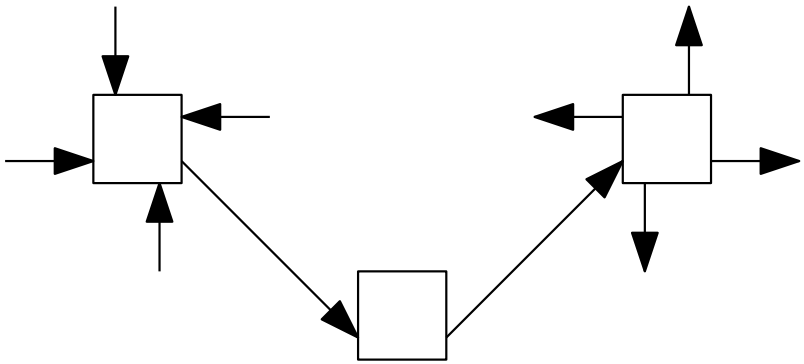
- Zeugensuche wird komplizierter
  - für jedes Paar eingehender und ausgehender Kanten muss eine Zeugensuche durchgeführt werden
  - es können Self-Loops entstehen

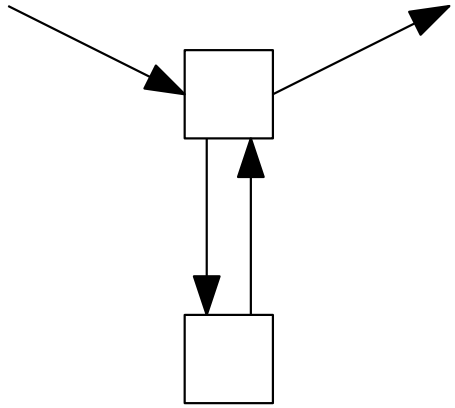


Kompaktes Modell

CH

- Zeugensuche wird komplizierter
  - für jedes Paar eingehender und ausgehender Kanten muss eine Zeugensuche durchgeführt werden
  - es können Self-Loops entstehen

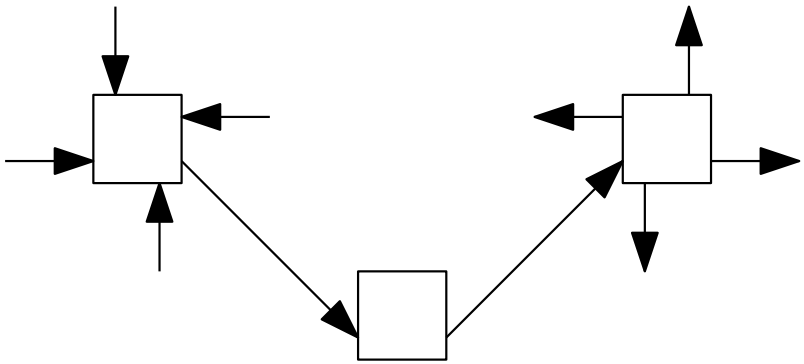


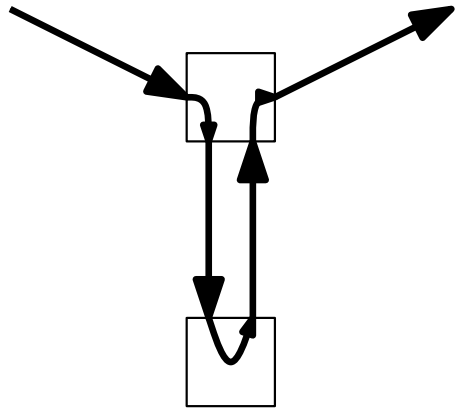


Kompaktes Modell

CH

- Zeugensuche wird komplizierter
  - für jedes Paar eingehender und ausgehender Kanten muss eine Zeugensuche durchgeführt werden
  - es können Self-Loops entstehen

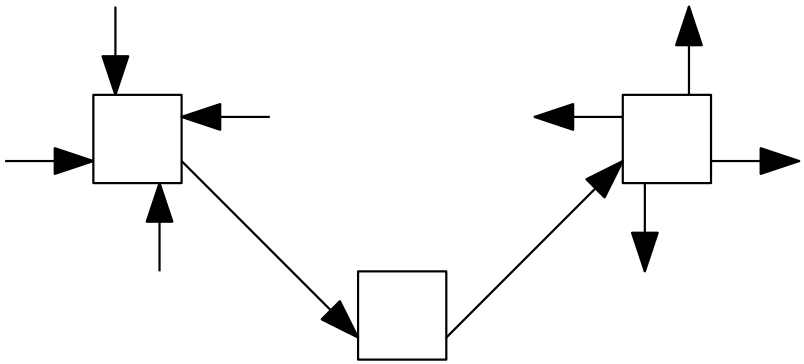


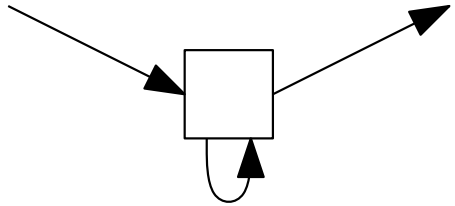


Kompaktes Modell

CH

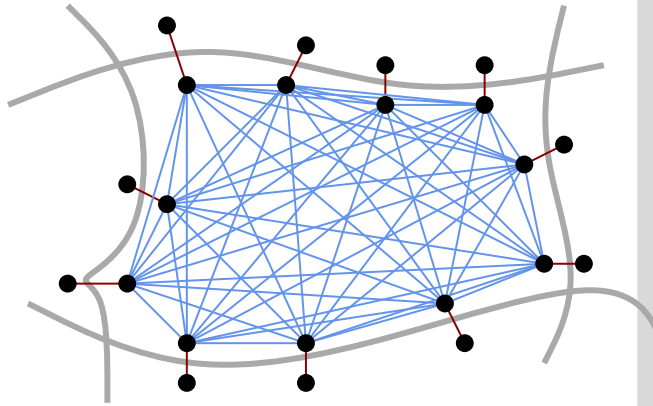
- Zeugensuche wird komplizierter
  - für jedes Paar eingehender und ausgehender Kanten muss eine Zeugensuche durchgeführt werden
  - es können Self-Loops entstehen



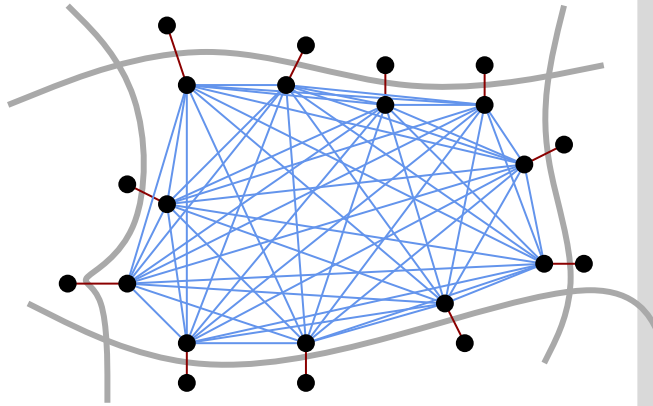


Kompaktes Modell

MLD



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

	Algorithm	Customization time [s]	Customization [MB]	Queries #scans	Queries time [ms]
1 s	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
100 s	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 Prioritätsfunktionen problematisch bei Turns
- Feingranulare Priorität auf Turnebene wichtig (vgl CH expanded vs compact)
- MLD robust

Ende

Literatur:

- Sebastian Knopp, Peter Sanders, Dominik Schultes, Frank Schulz, and Dorothea Wagner:
  Computing Many-to-Many Shortest Paths Using Highway Hierarchies
  In: *Proceedings of the 9th Workshop on Algorithm Engineering and Experiments (ALENEX'07)*, pages 36-45, 2007.
- Daniel Delling, Andrew V. Goldberg, Renato F. Werneck
  Faster Batched Shortest Paths in Road Networks
  In: *Proceedings of the 11th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'11)*, pages 52-63, 2011
- Ittai Abraham, Daniel Delling, Andrew V. Goldberg, Renato F. Werneck
  HLDB: Location-Based Services in Databases
  bald verfügbar