Karlsruher Institut für Technologie

Algorithmen für Routenplanung

11. Sitzung, Sommersemester 2014

Moritz Baum | 02. Juni 2014

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

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.

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

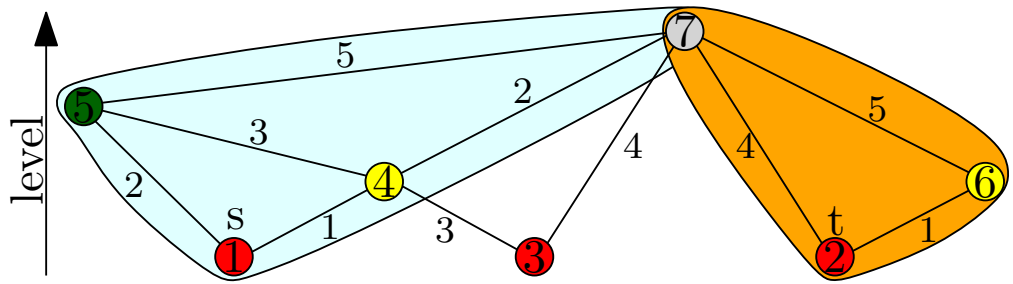
Beispiele:

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

Hierarchische Techniken

Beobachtung:

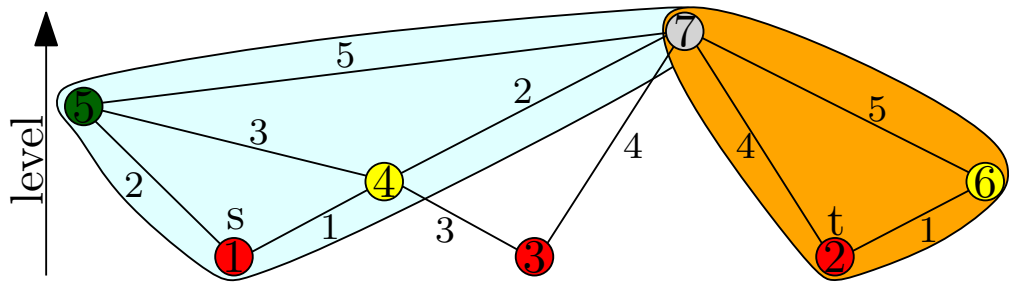
- suchen nur aufwärts
- sind nicht zielgerichtet



Hierarchische Techniken

Beobachtung:

- suchen nur aufwärts
- sind nicht zielgerichtet



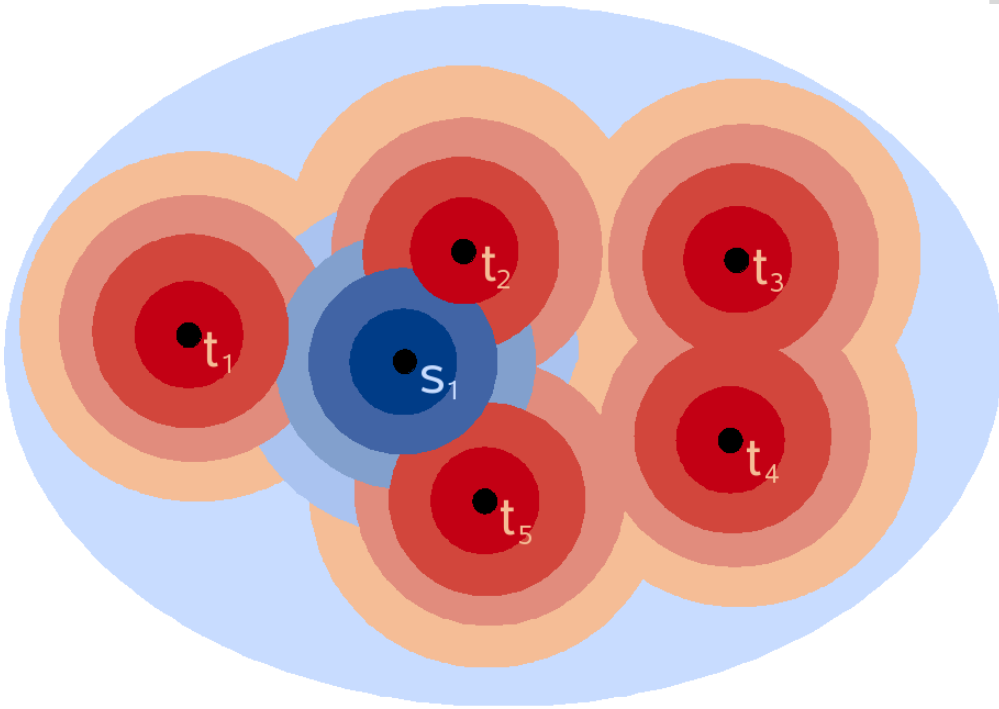
somit:

- 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:

- breche nicht ab
- für jedes erreichte u:
  - füge Element $(u, t, d(u, t))$ in einen Vektor ein

Kompression:

- sortiere Einträge nach u
- speichere für jeden Knoten u einen Bucket $\beta(u)$ mit allen $(t, d(u, t))$ ab
- mittels Adjazenzarray

während Vorwärtssuche:

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

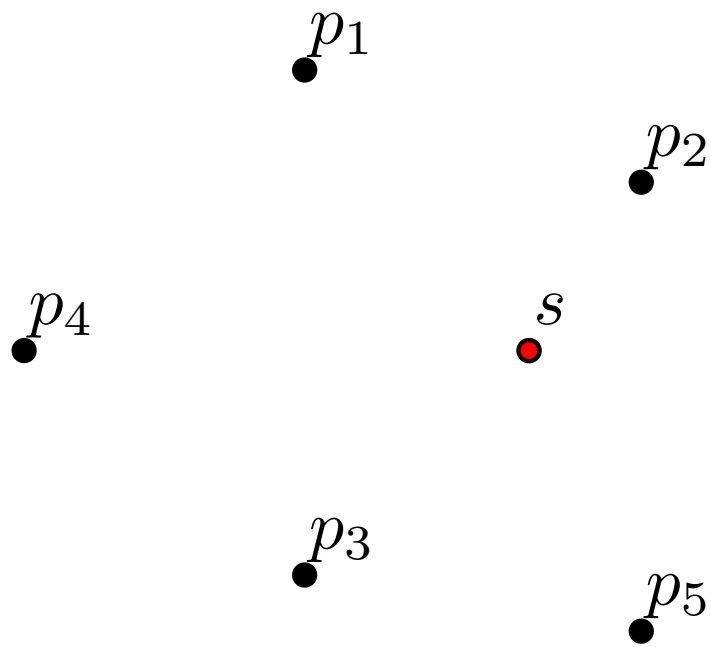
Schneiden der Suchräume

während Rückwärtssuchen:

- breche nicht ab
- für jedes erreichte u:
  - füge Element $(u, t, d(u, t))$ in einen Vektor ein

Kompression:

- sortiere Einträge nach u
- speichere für jeden Knoten u einen Bucket $\beta(u)$ mit allen $(t, d(u, t))$ ab
- mittels Adjazenzarray



während Vorwärtssuche:

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

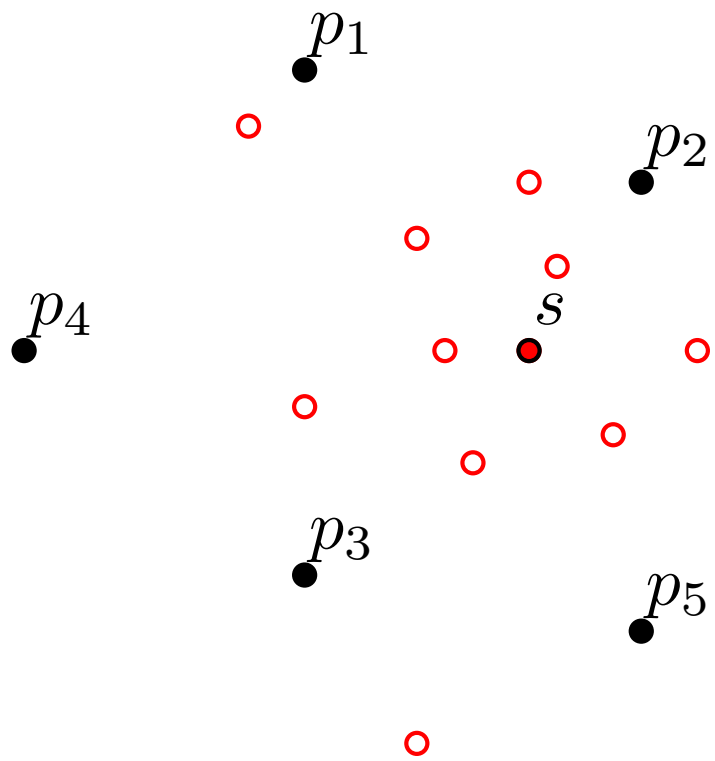
Schneiden der Suchräume

während Rückwärtssuchen:

- breche nicht ab
- für jedes erreichte u:
  - füge Element $(u, t, d(u, t))$ in einen Vektor ein

Kompression:

- sortiere Einträge nach u
- speichere für jeden Knoten u einen Bucket $\beta(u)$ mit allen $(t, d(u, t))$ ab
- mittels Adjazenzarray



während Vorwärtssuche:

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

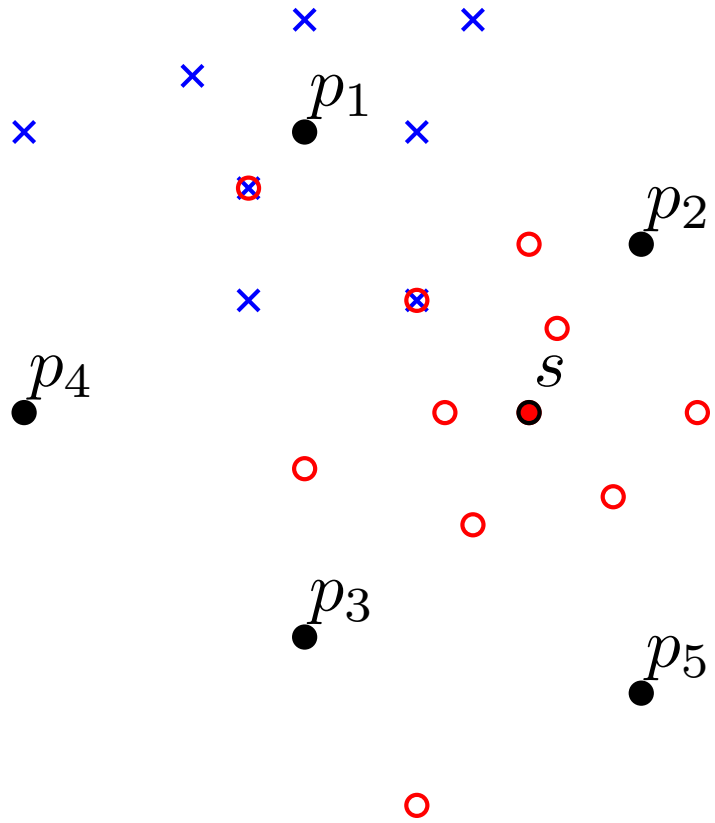
Schneiden der Suchräume

während Rückwärtssuchen:

- breche nicht ab
- für jedes erreichte u:
  - füge Element $(u, t, d(u, t))$ in einen Vektor ein

Kompression:

- sortiere Einträge nach u
- speichere für jeden Knoten u einen Bucket $\beta(u)$ mit allen $(t, d(u, t))$ ab
- mittels Adjazenzarray



während Vorwärtssuche:

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

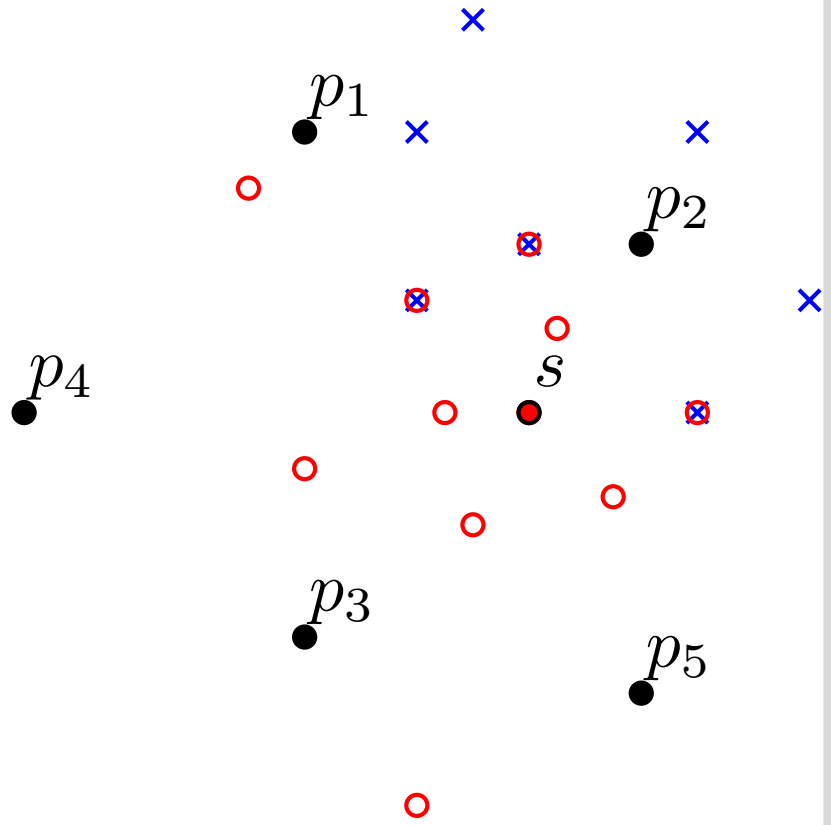
Schneiden der Suchräume

während Rückwärtssuchen:

- breche nicht ab
- für jedes erreichte u:
  - füge Element $(u, t, d(u, t))$ in einen Vektor ein

Kompression:

- sortiere Einträge nach u
- speichere für jeden Knoten u einen Bucket $\beta(u)$ mit allen $(t, d(u, t))$ ab
- mittels Adjazenzarray



während Vorwärtssuche:

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

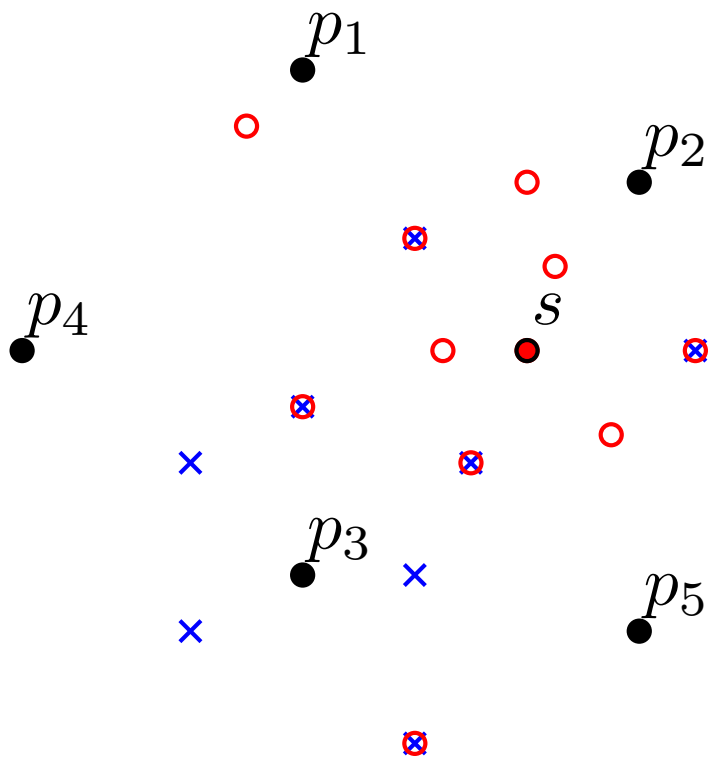
Schneiden der Suchräume

während Rückwärtssuchen:

- breche nicht ab
- für jedes erreichte u:
  - füge Element $(u, t, d(u, t))$ in einen Vektor ein

Kompression:

- sortiere Einträge nach u
- speichere für jeden Knoten u einen Bucket $\beta(u)$ mit allen $(t, d(u, t))$ ab
- mittels Adjazenzarray



während Vorwärtssuche:

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

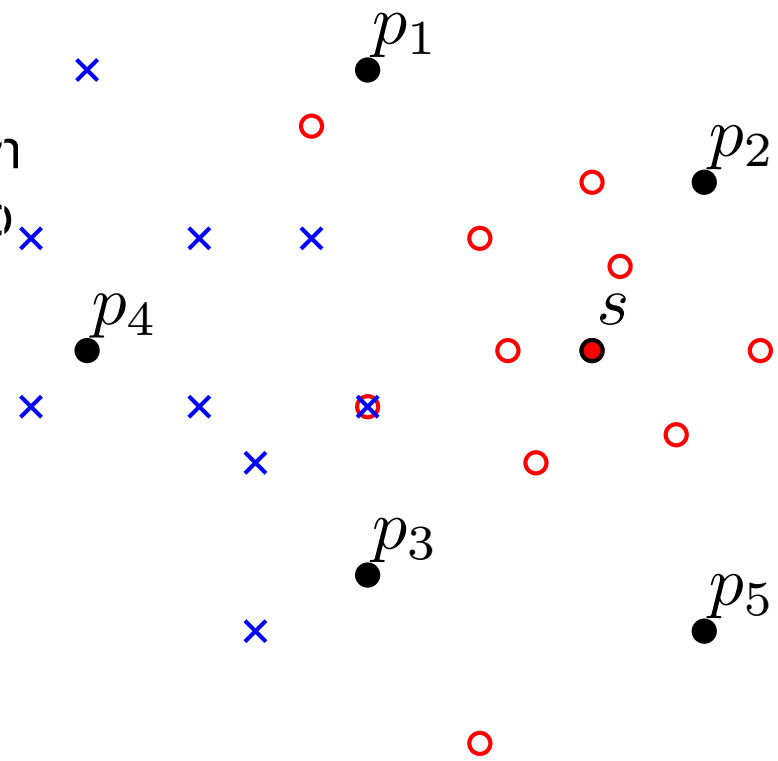
Schneiden der Suchräume

während Rückwärtssuchen:

- breche nicht ab
- für jedes erreichte u:
  - füge Element $(u, t, d(u, t))$ in einen Vektor ein

Kompression:

- sortiere Einträge nach u
- speichere für jeden Knoten u einen Bucket $\beta(u)$ mit allen $(t, d(u, t))$ ab
- mittels Adjazenzarray



während Vorwärtssuche:

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

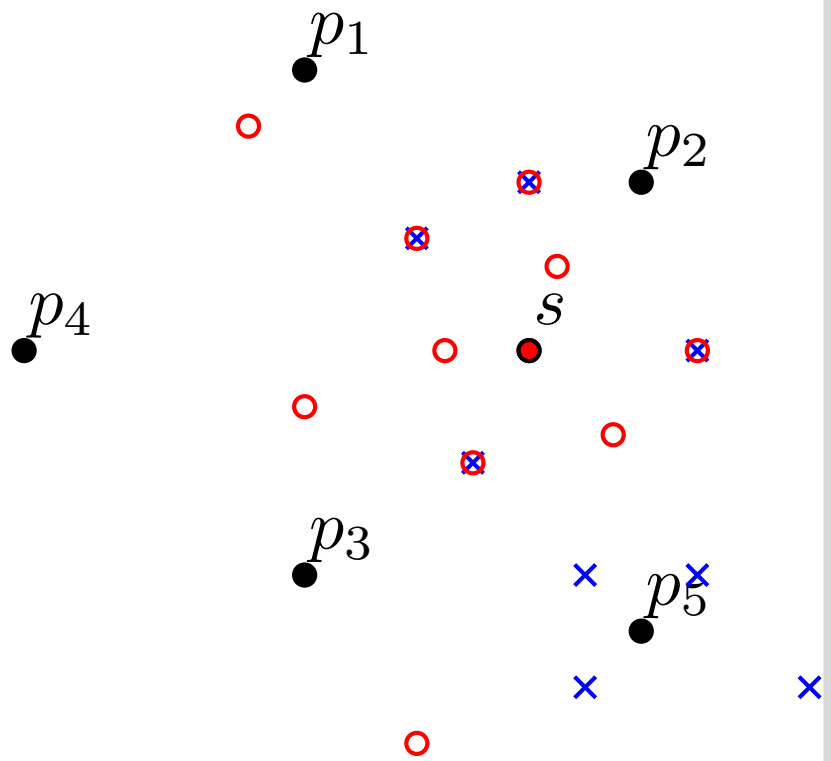
Schneiden der Suchräume

während Rückwärtssuchen:

- breche nicht ab
- für jedes erreichte u:
  - füge Element $(u, t, d(u, t))$ in einen Vektor ein

Kompression:

- sortiere Einträge nach u
- speichere für jeden Knoten u einen Bucket $\beta(u)$ mit allen $(t, d(u, t))$ ab
- mittels Adjazenzarray



während Vorwärtssuche:

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

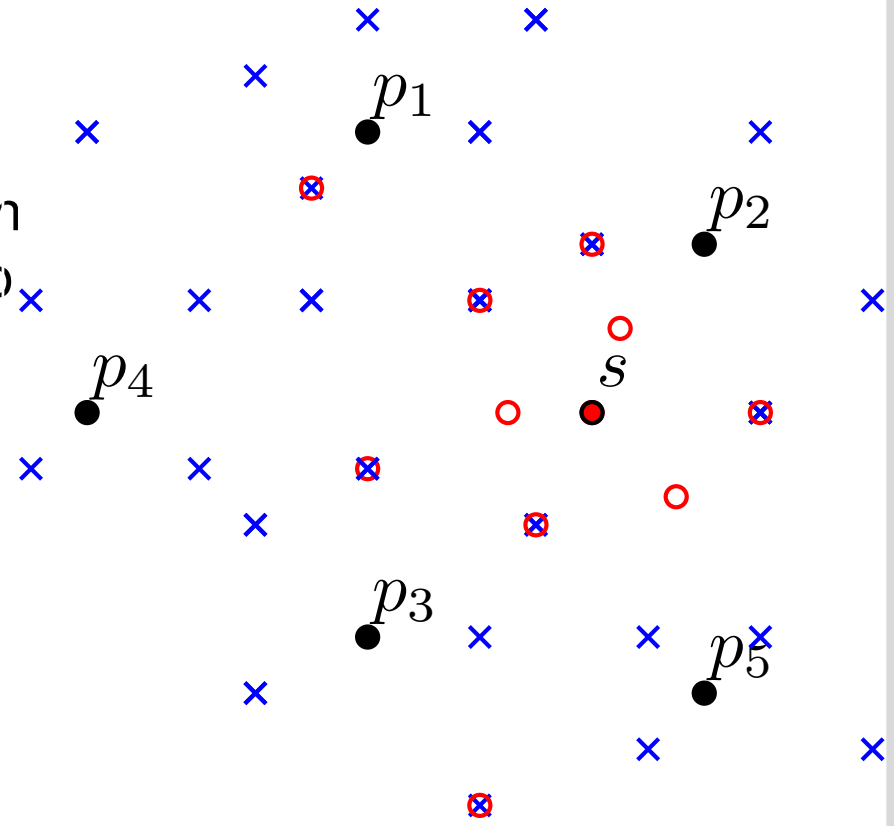
Schneiden der Suchräume

während Rückwärtssuchen:

- breche nicht ab
- für jedes erreichte u:
  - füge Element $(u, t, d(u, t))$ in einen Vektor ein

Kompression:

- sortiere Einträge nach u
- speichere für jeden Knoten u einen Bucket $\beta(u)$ mit allen $(t, d(u, t))$ ab
- mittels Adjazenzarray



während Vorwärtssuche:

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

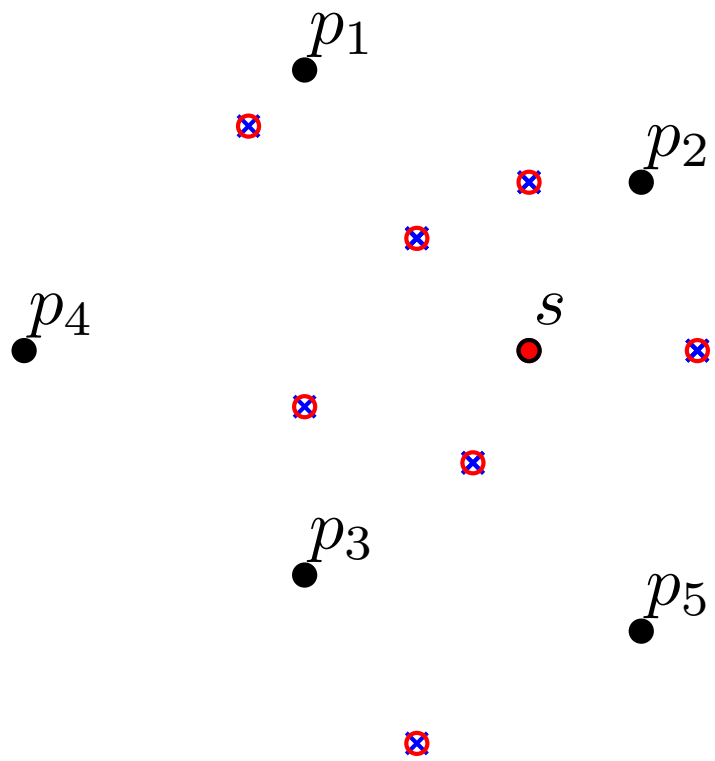
Schneiden der Suchräume

während Rückwärtssuchen:

- breche nicht ab
- für jedes erreichte u:
  - füge Element $(u, t, d(u, t))$ in einen Vektor ein

Kompression:

- sortiere Einträge nach u
- speichere für jeden Knoten u einen Bucket $\beta(u)$ mit allen $(t, d(u, t))$ ab
- mittels Adjazenzarray



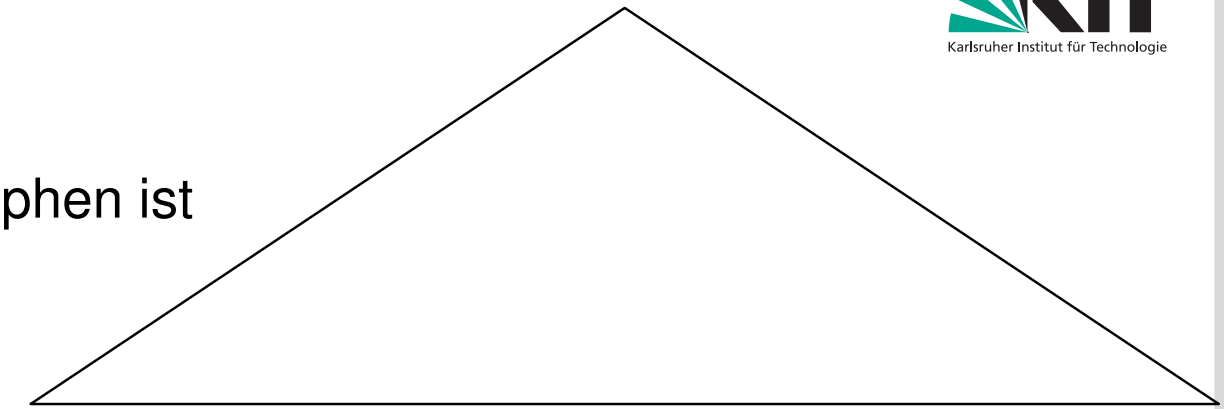
während Vorwärtssuche:

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

RPHAST

Beobachtung:

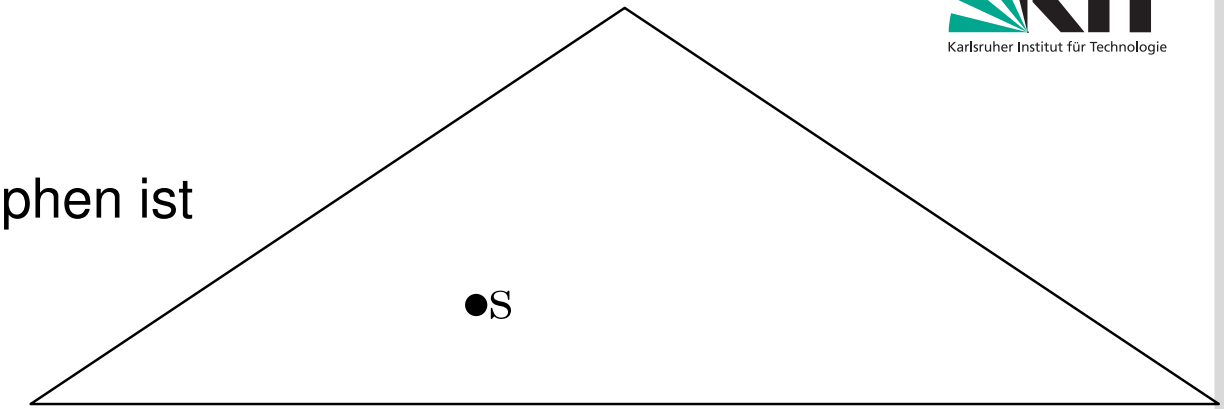
- Sweep über den Graphen ist der Flaschenhals



RPHAST

Beobachtung:

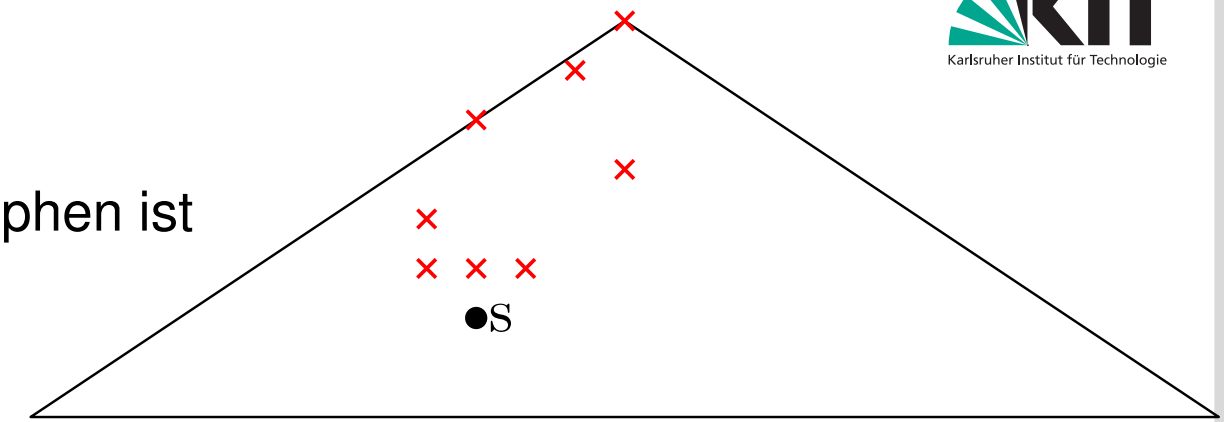
- Sweep über den Graphen ist der Flaschenhals



RPHAST

Beobachtung:

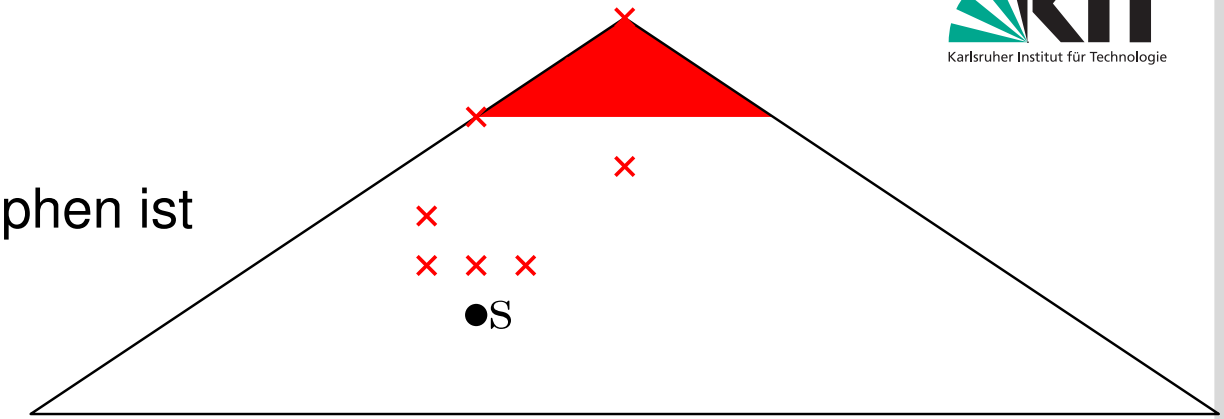
- Sweep über den Graphen ist der Flaschenhals



RPHAST

Beobachtung:

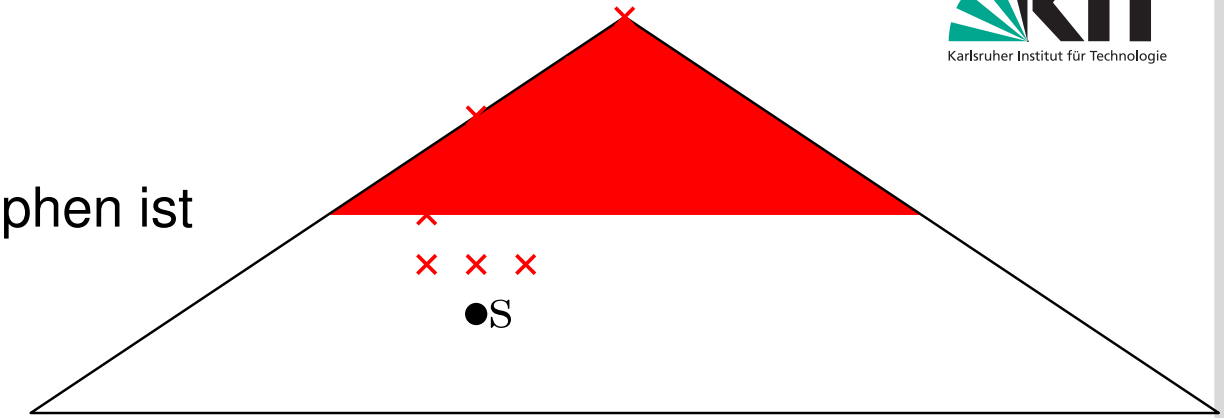
- Sweep über den Graphen ist der Flaschenhals



RPHAST

Beobachtung:

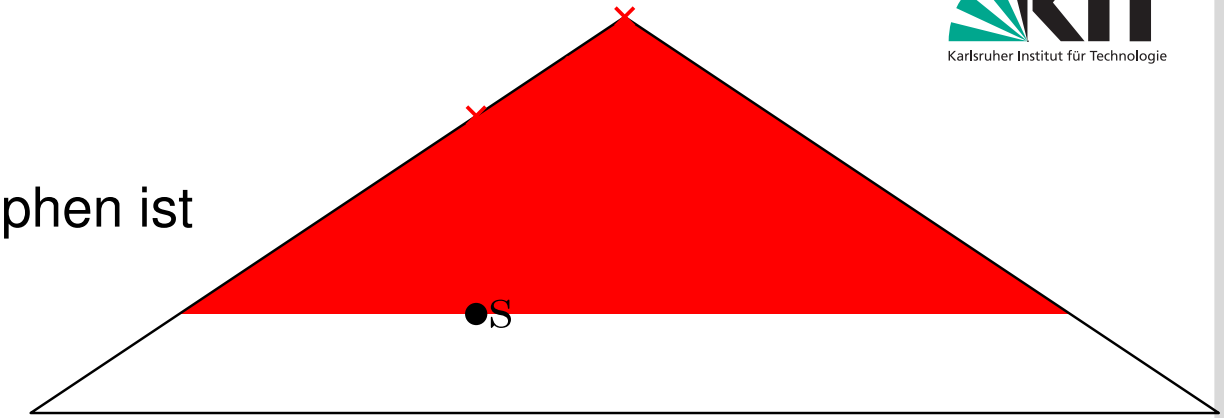
- Sweep über den Graphen ist der Flaschenhals



RPHAST

Beobachtung:

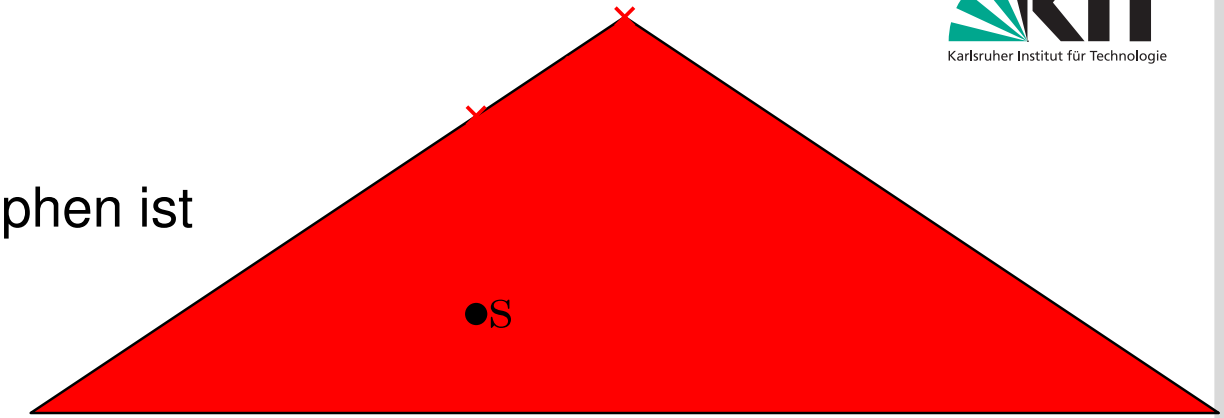
- Sweep über den Graphen ist der Flaschenhals



RPHAST

Beobachtung:

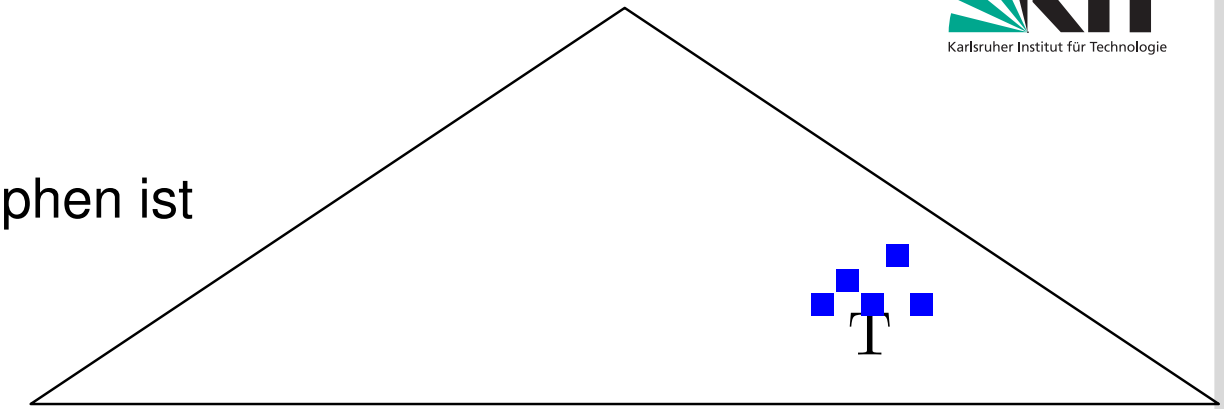
- Sweep über den Graphen ist der Flaschenhals



RPHAST

Beobachtung:

- Sweep über den Graphen ist der Flaschenhals



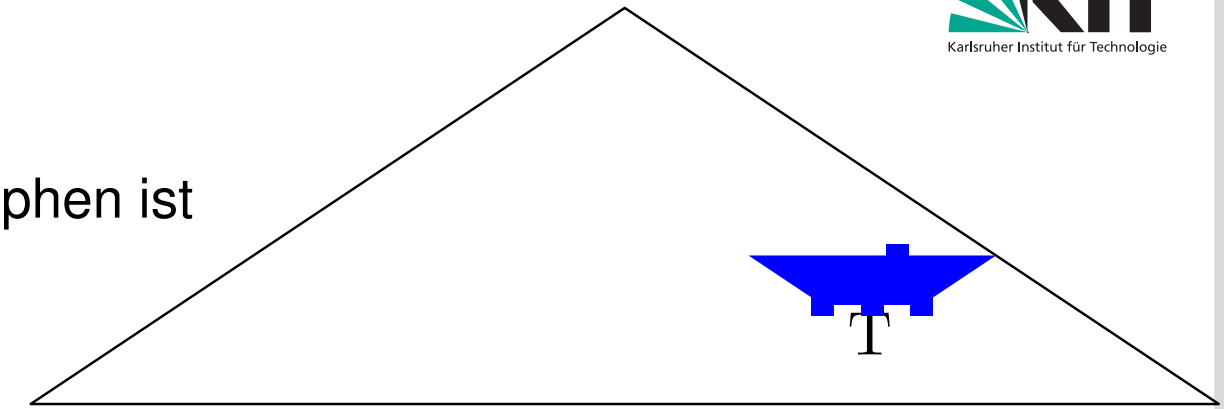
Idee:

- extrahiere relevanten Teil des Graphen (Ziel Selektion)

RPHAST

Beobachtung:

- Sweep über den Graphen ist der Flaschenhals



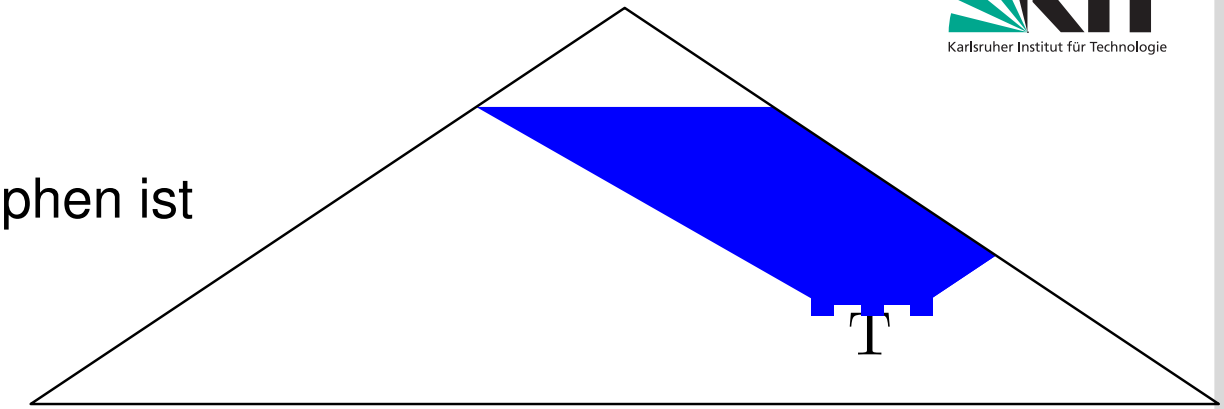
Idee:

- extrahiere relevanten Teil des Graphen (Ziel Selektion)

RPHAST

Beobachtung:

- Sweep über den Graphen ist der Flaschenhals



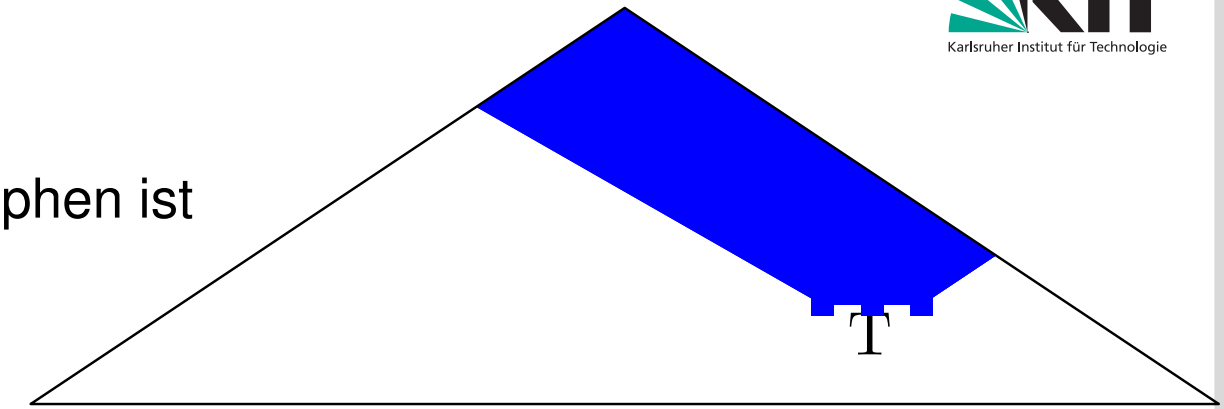
Idee:

- extrahiere relevanten Teil des Graphen (Ziel Selektion)

RPHAST

Beobachtung:

- Sweep über den Graphen ist der Flaschenhals



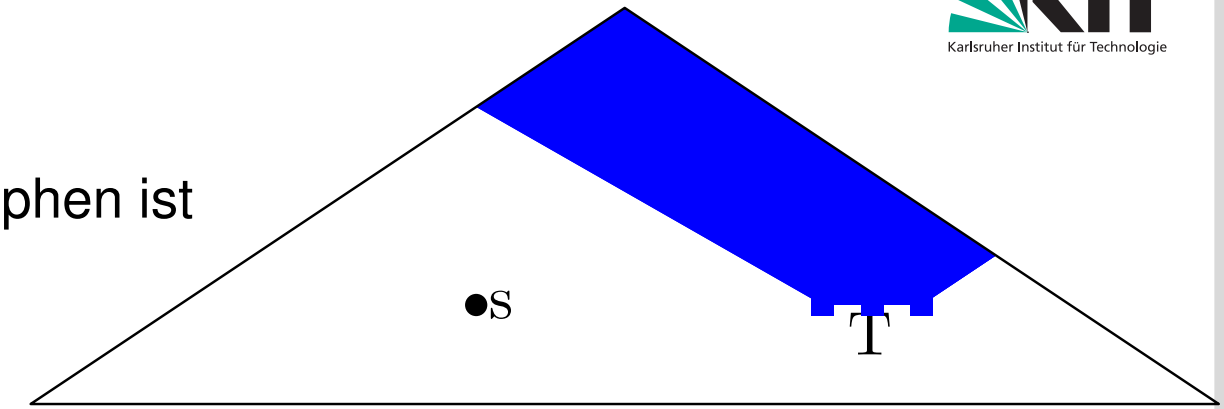
Idee:

- extrahiere relevanten Teil des Graphen (Ziel Selektion)

RPHAST

Beobachtung:

- Sweep über den Graphen ist der Flaschenhals



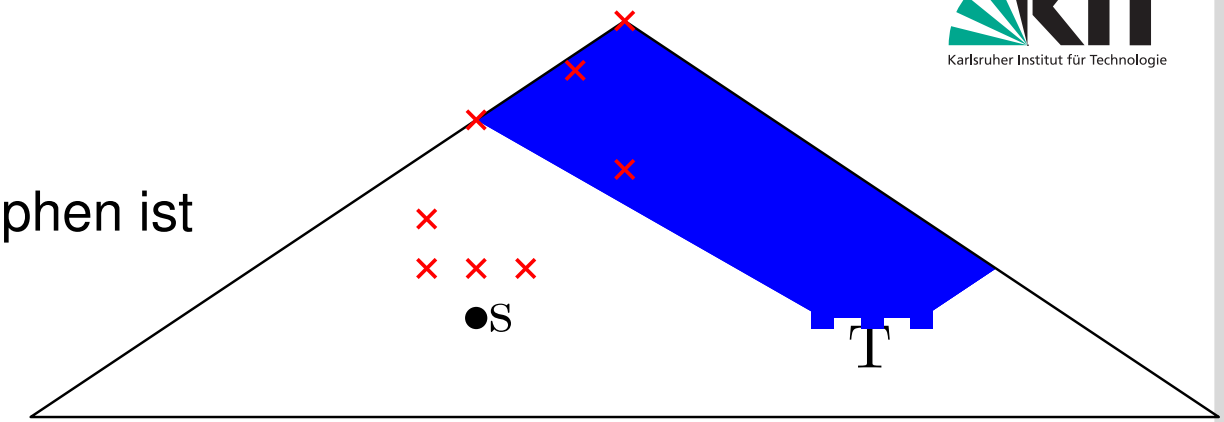
Idee:

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

RPHAST

Beobachtung:

- Sweep über den Graphen ist der Flaschenhals



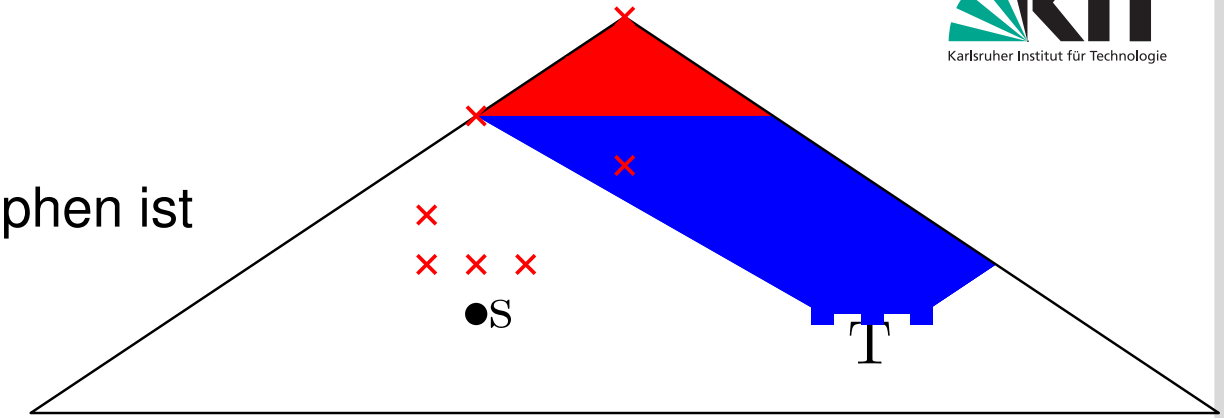
Idee:

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

RPHAST

Beobachtung:

- Sweep über den Graphen ist der Flaschenhals



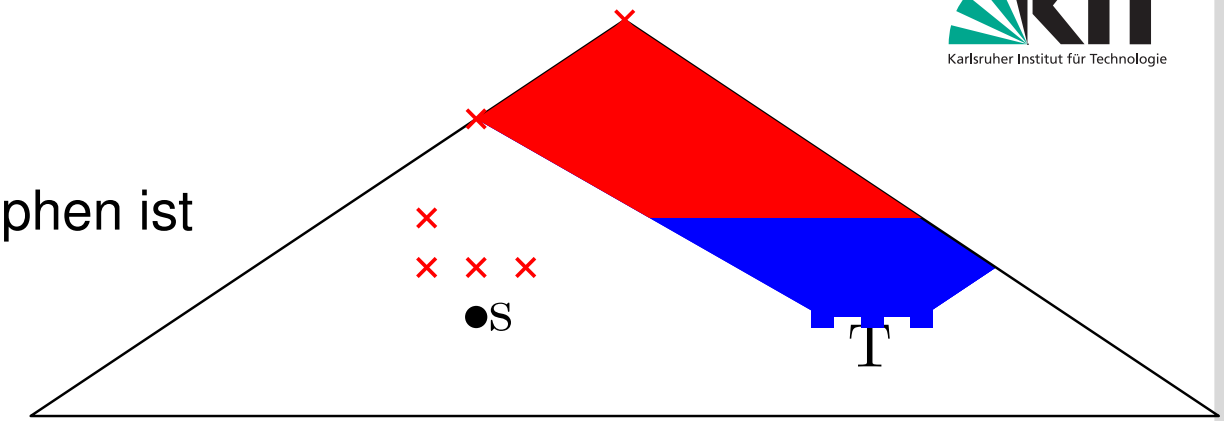
Idee:

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

RPHAST

Beobachtung:

- Sweep über den Graphen ist der Flaschenhals



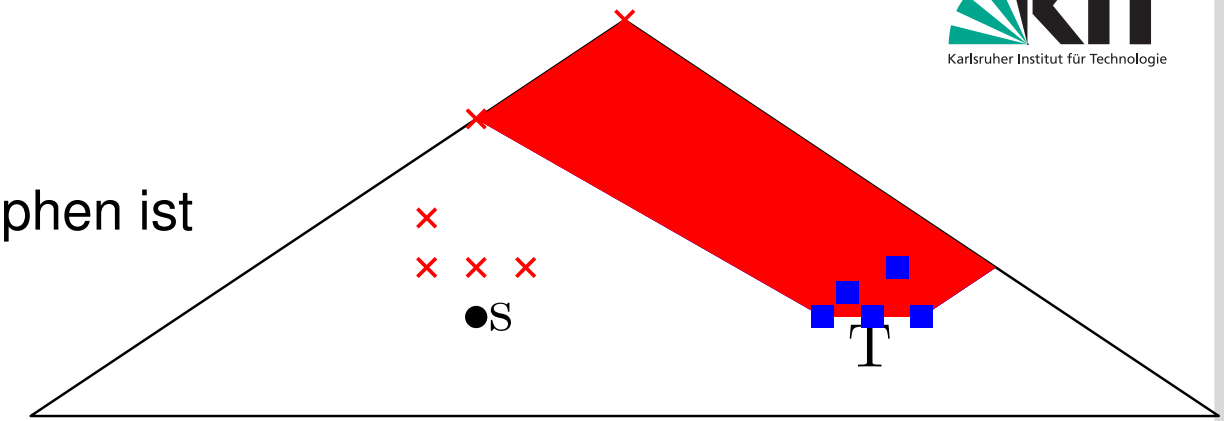
Idee:

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

RPHAST

Beobachtung:

- Sweep über den Graphen ist der Flaschenhals



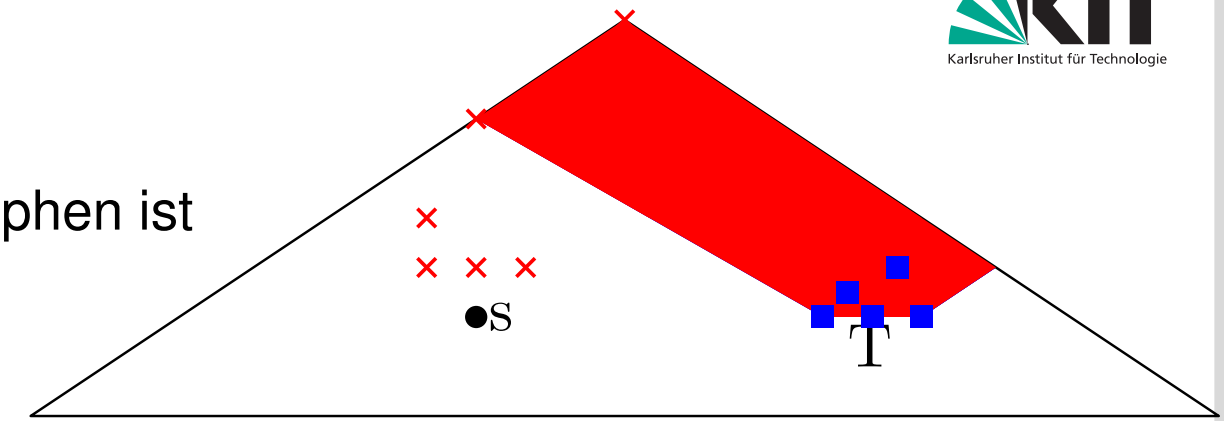
Idee:

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

RPHAST

Beobachtung:

- 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.

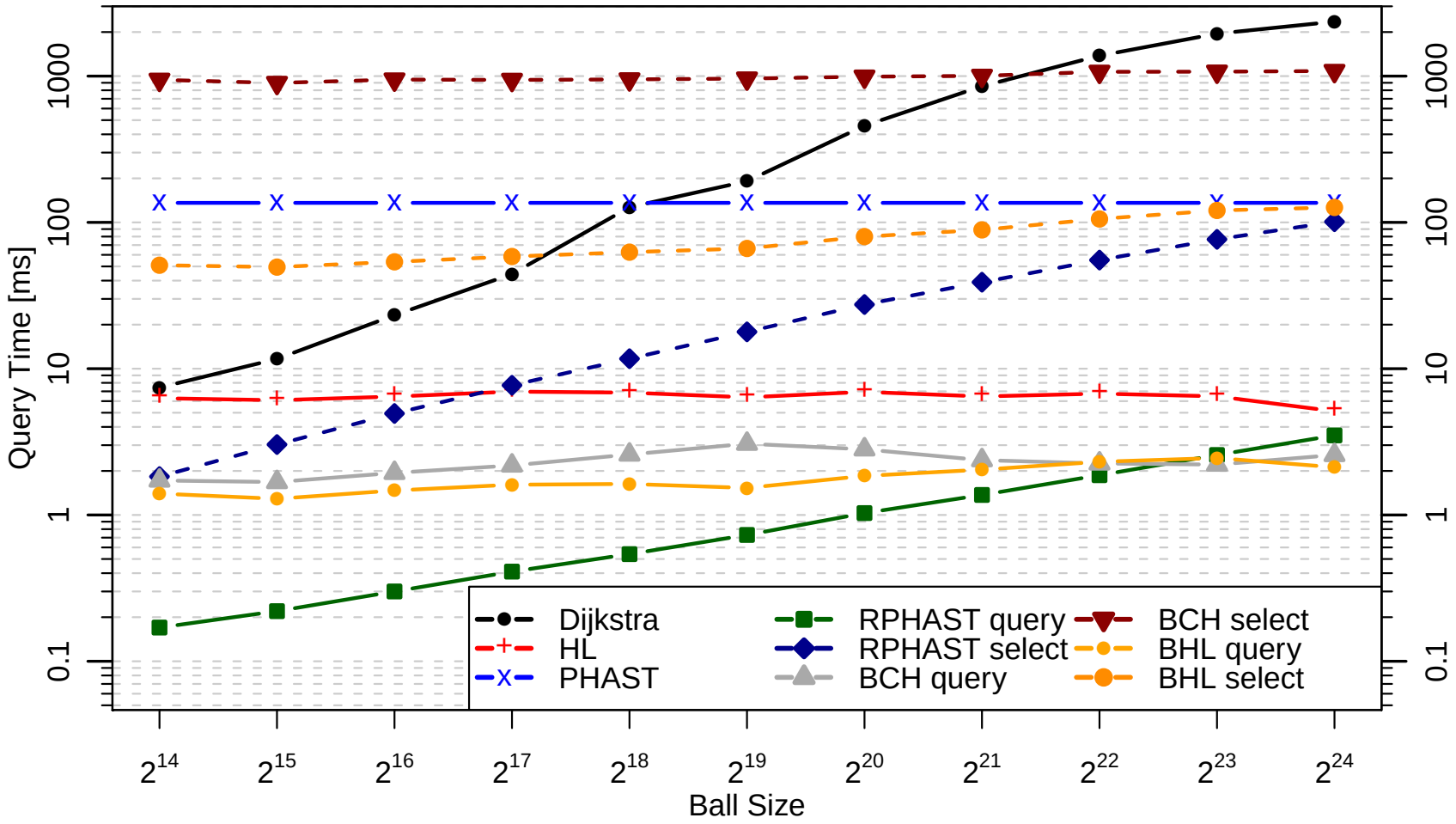
Setup:

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

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

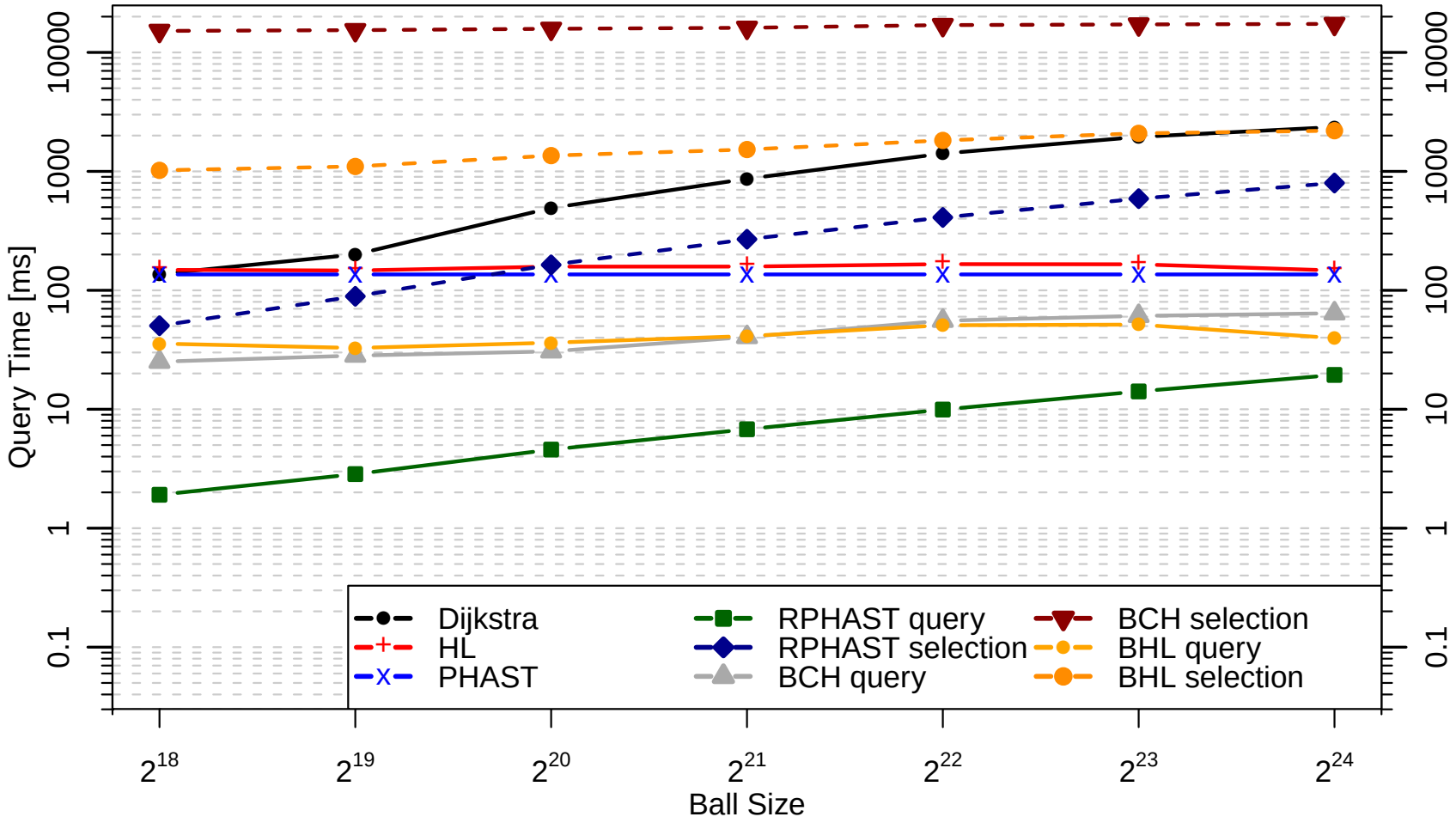
Experimente I

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



Experimente II

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



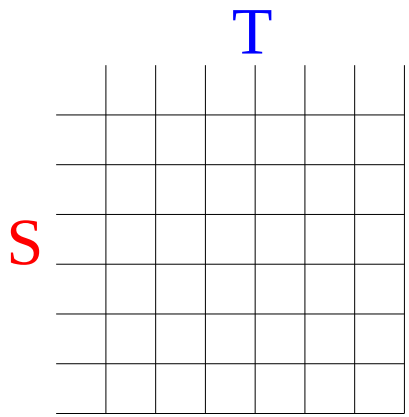
Many-to-Many Kürzeste Wege

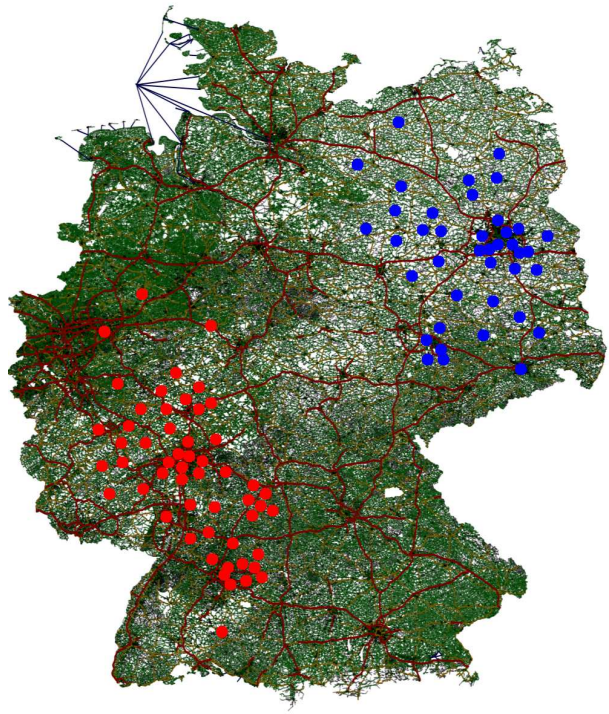
Gegeben:

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

Gesucht:

- Distanzmatrix D





Many-to-Many Kürzeste Wege

Gegeben:

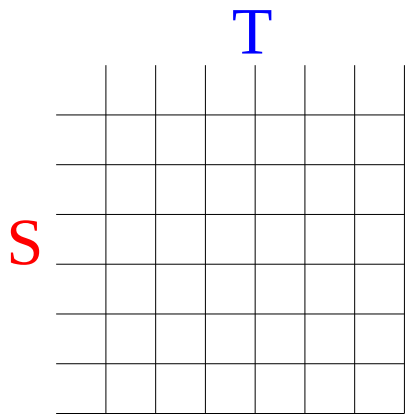
- Graph
- Knotenmengen $S, T \in V$

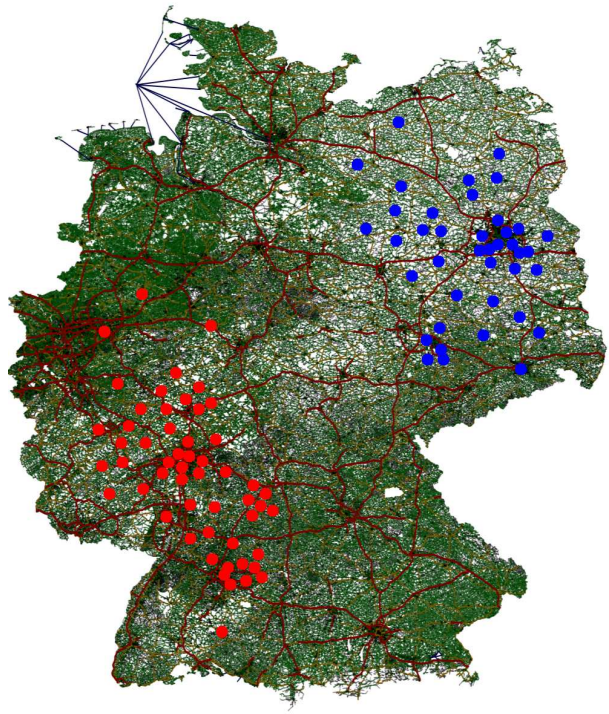
Gesucht:

- Distanzmatrix D

Anwendungen:

- vehicle routing
- traveling salesman





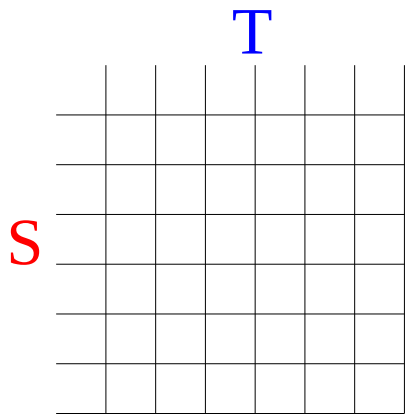
Many-to-Many Kürzeste Wege

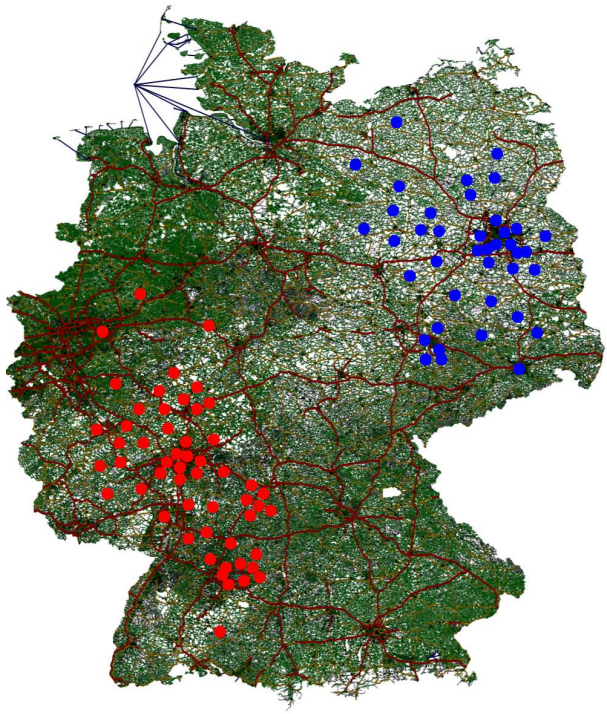
Gegeben:

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

Gesucht:

- Distanzmatrix D





Anwendungen:

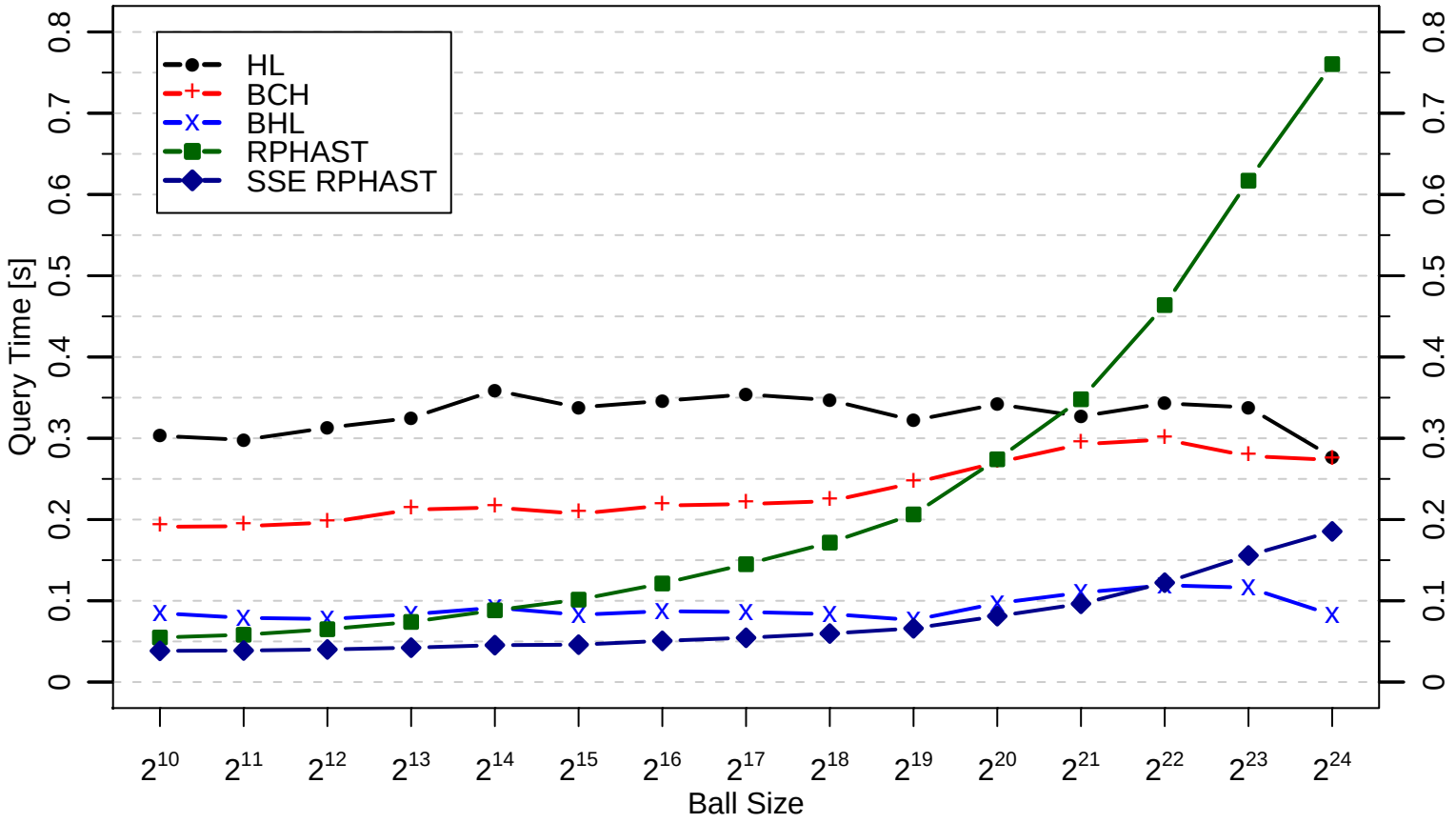
- vehicle routing
- traveling salesman

Lösung:

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

Experimente I

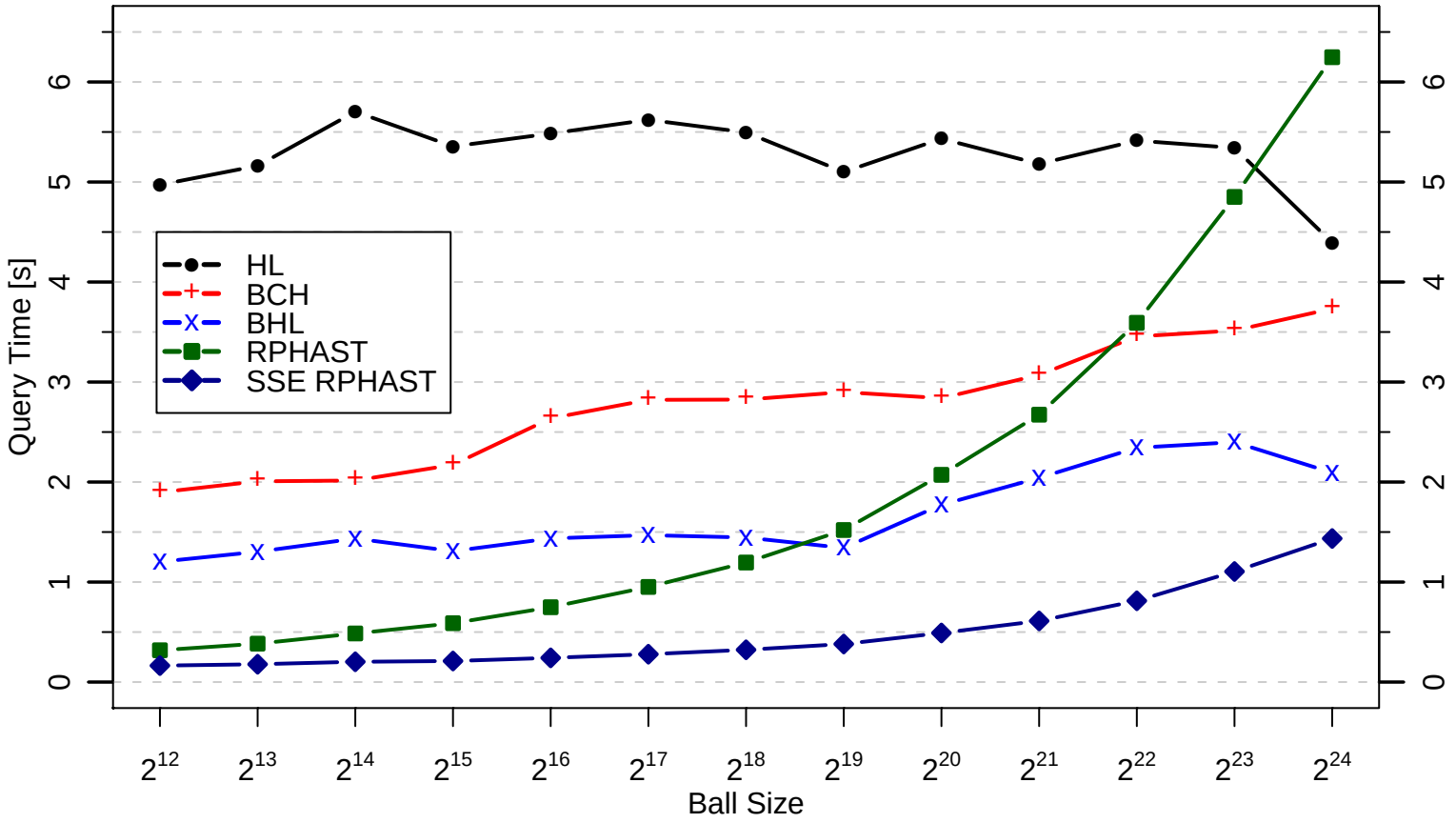
input: Westeuropa (18M Knoten), $|S| = |T| = 2^{10}$



Beobachtung: alle Techniken unter einer Sekunde

Experimente II

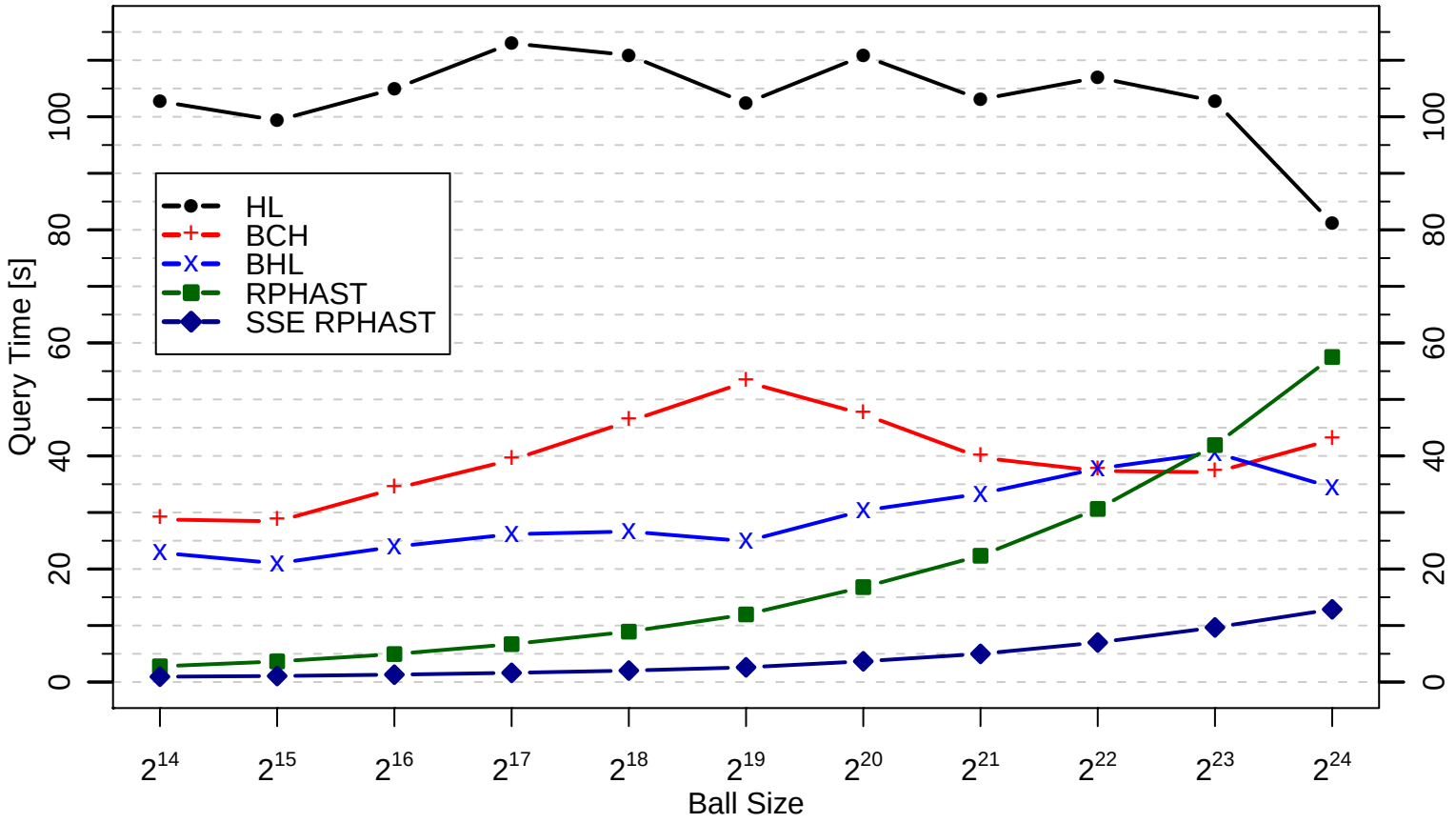
input: Westeuropa (18M Knoten), $|S| = |T| = 2^{12}$



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

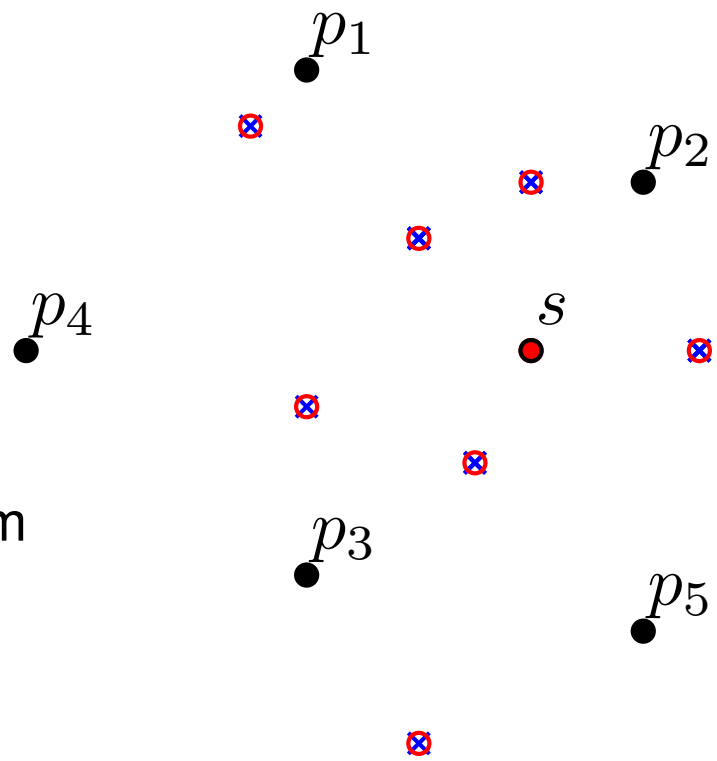
Nächster POI

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



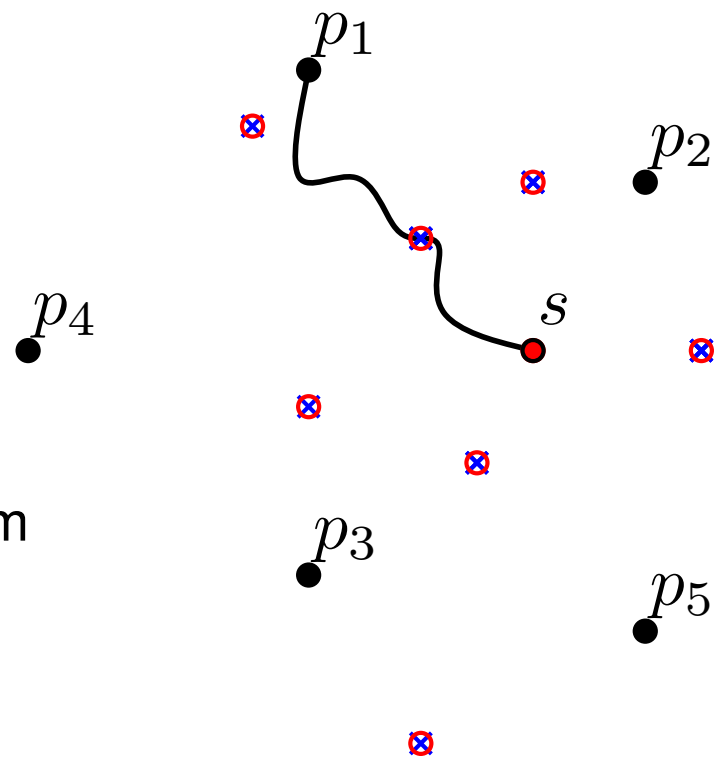
Nächster POI

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



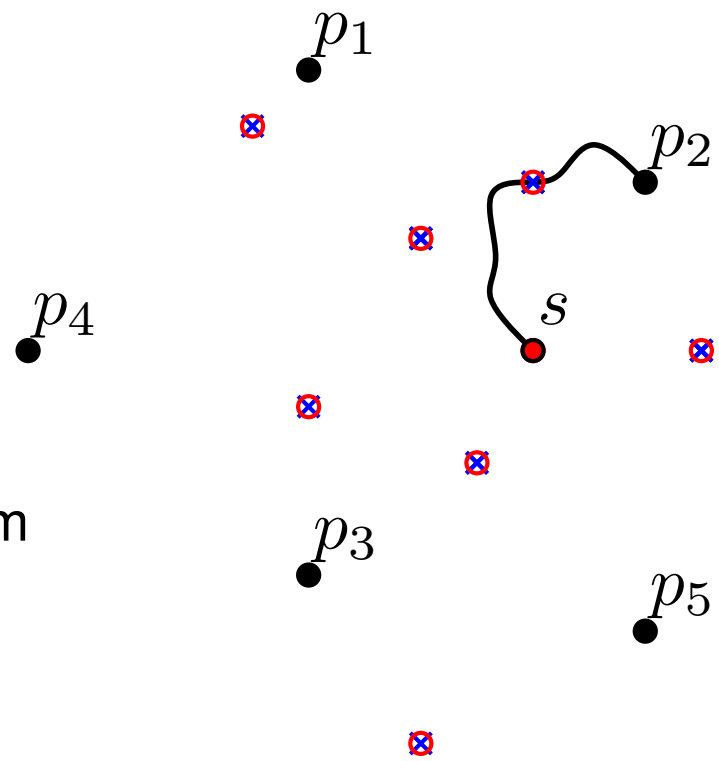
Nächster POI

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



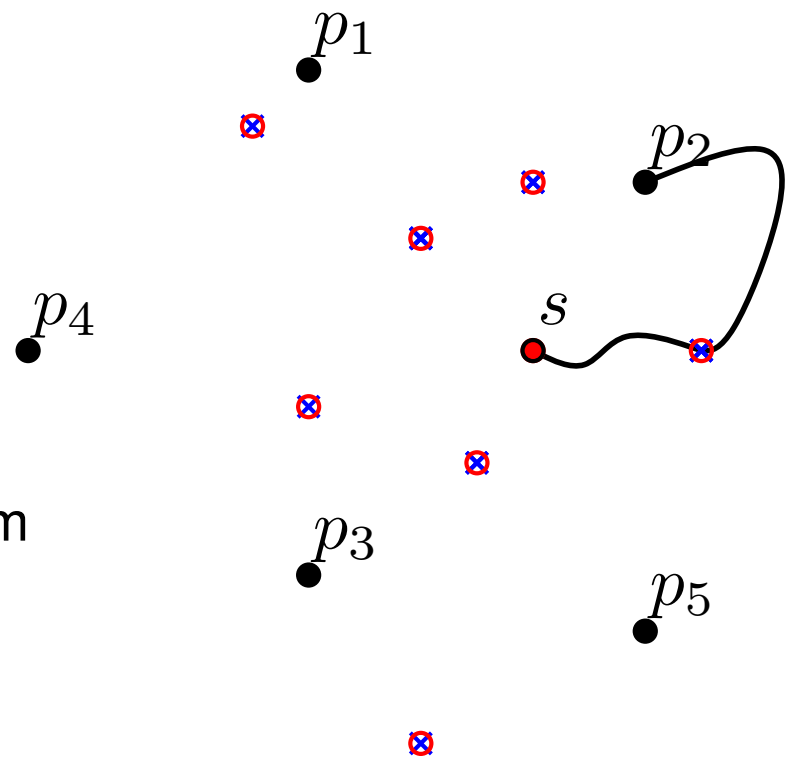
Nächster POI

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



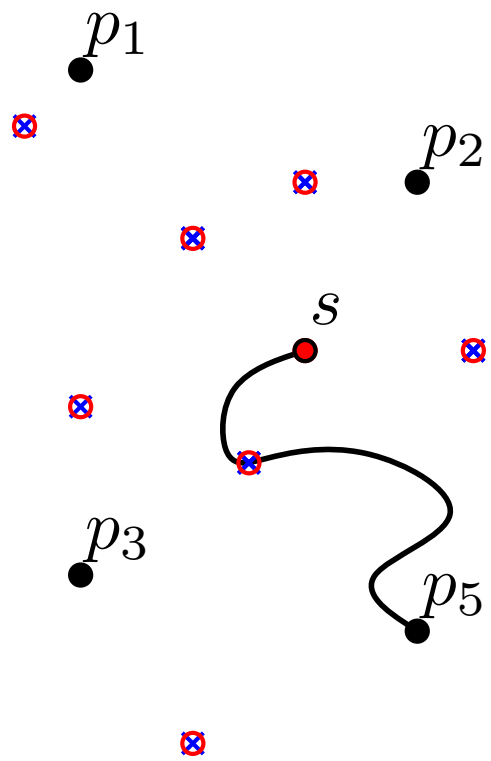
Nächster POI

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



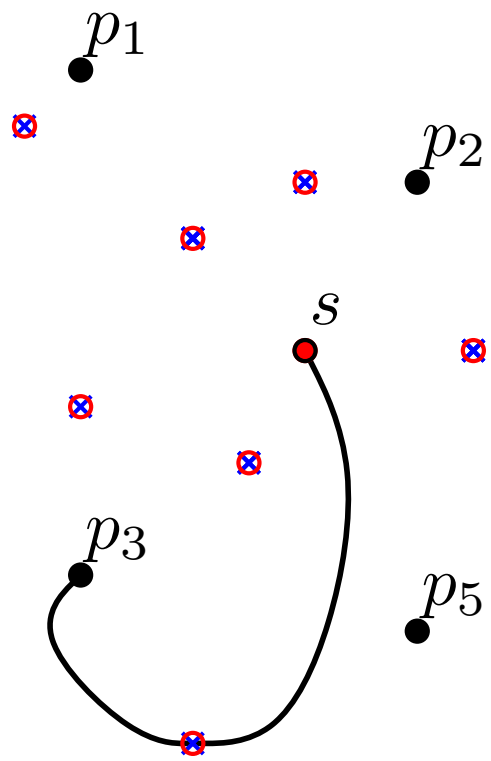
Nächster POI

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



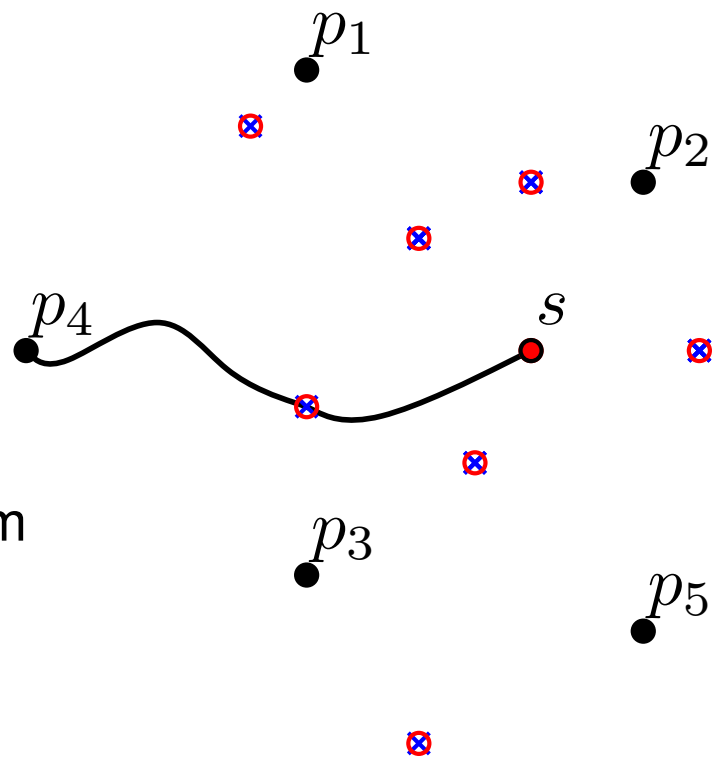
Nächster POI

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



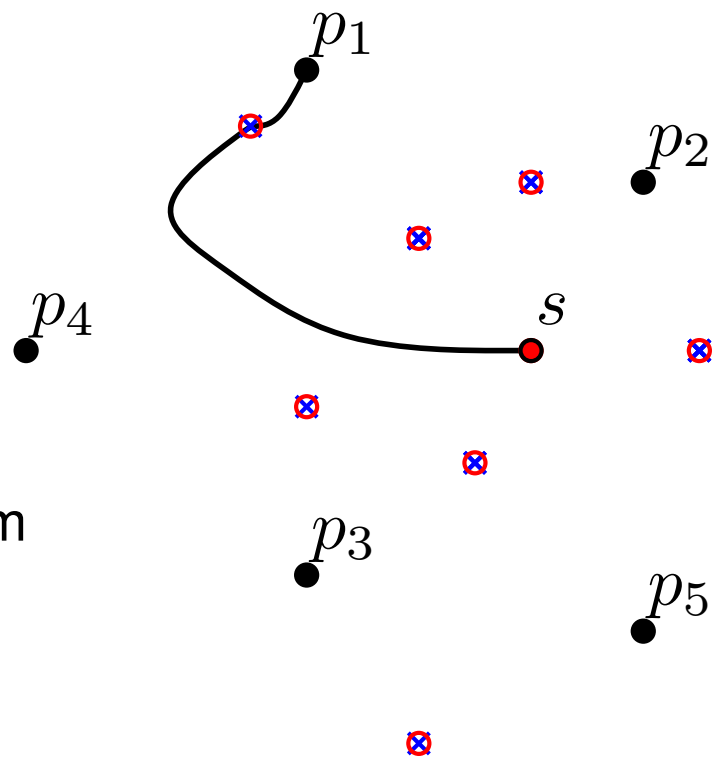
Nächster POI

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



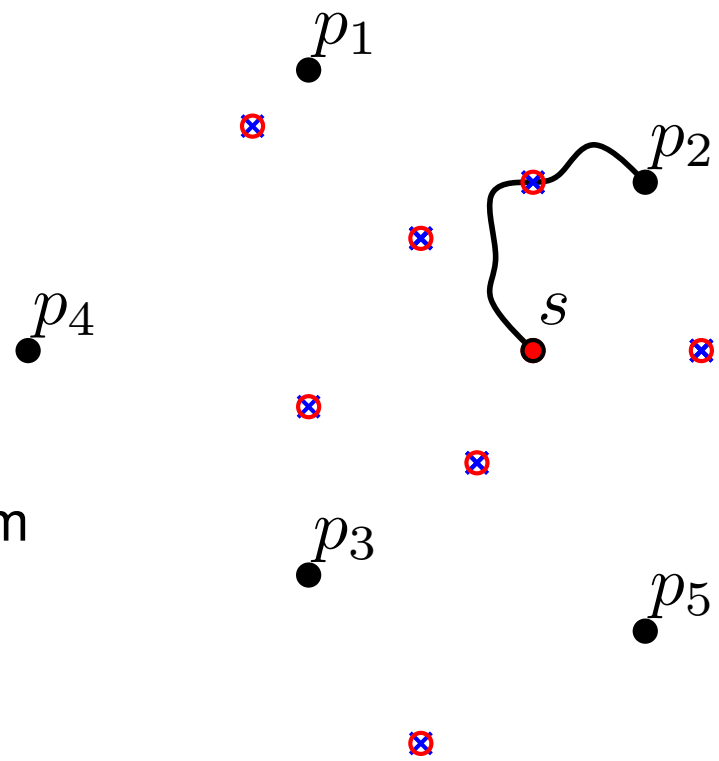
Nächster POI

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



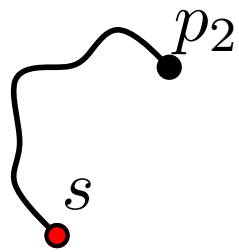
Nächster POI

Szenario:

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

Lösung:

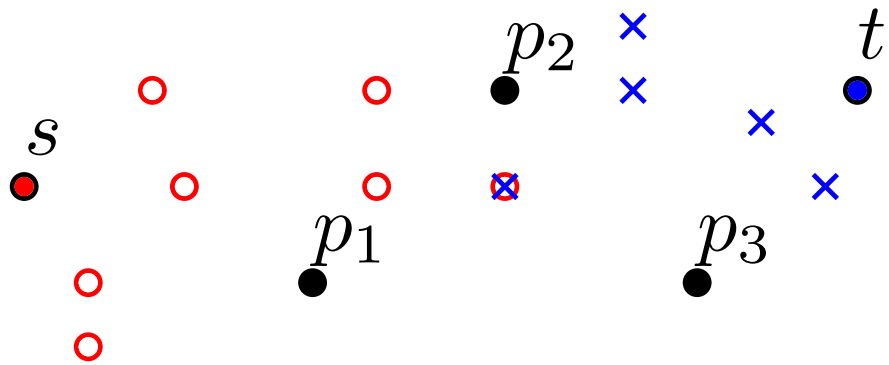
- wie one-to-many
- ordne die buckets pro Knoten auch nach aufsteigender Distanz
- in jedem Bucket müssen nur die k nächsten POIs durchsucht werden
- Laufzeit für POI Query nicht abhängig von Anzahl POIs im System
- Laufzeit: Suchraum $\cdot$ k



Bester Via Knoten

Szenario:

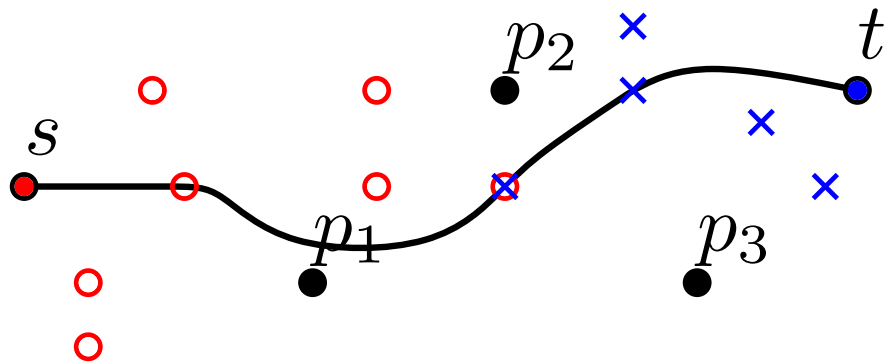
- finde k best Via Knoten POIs von einem Startknoten s zu einem Zielknoten t
- minimiere $\text{dist}(s, p) + \text{dist}(p, t)$ über alle POIs p



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



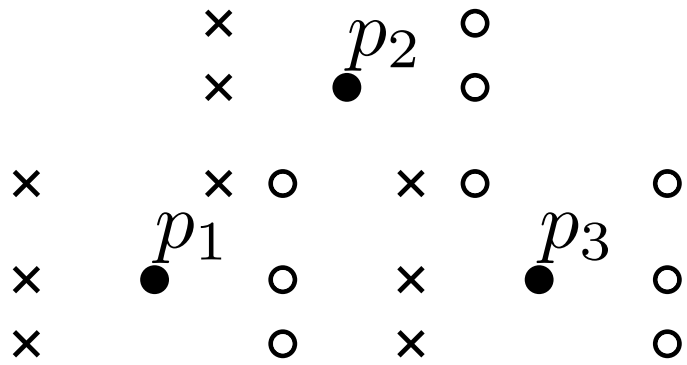
Bester Via Knoten

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$^2 \cdot k$



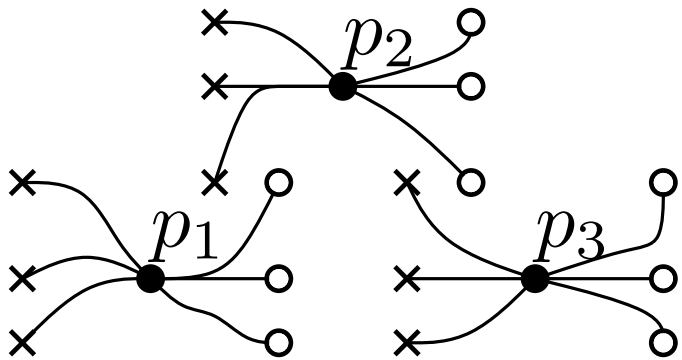
Bester Via Knoten

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$^2 \cdot k$



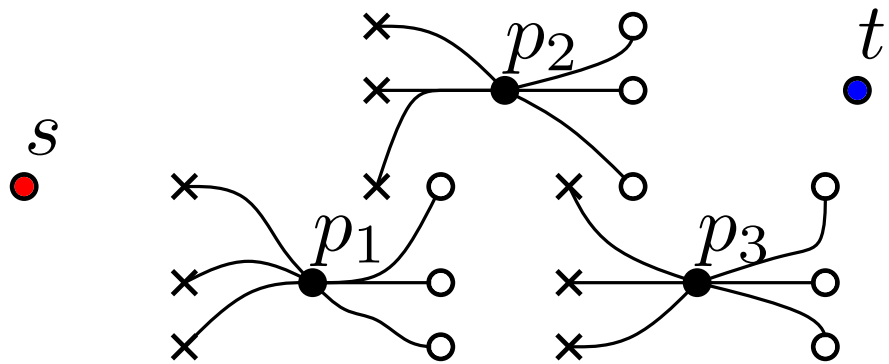
Bester Via Knoten

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$^2 \cdot k$



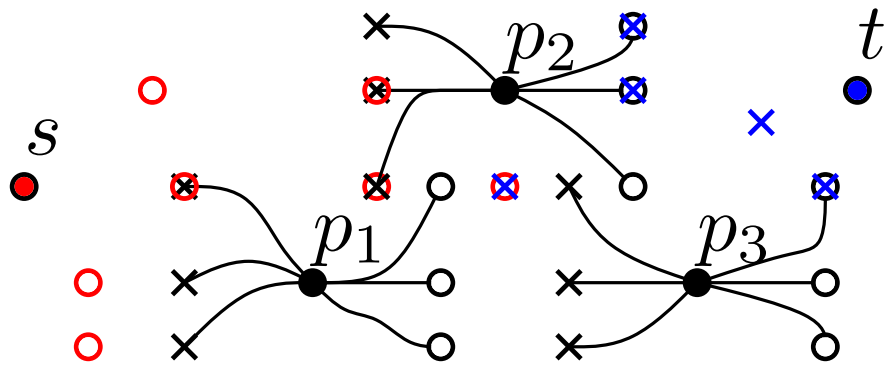
Bester Via Knoten

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$^2 \cdot k$



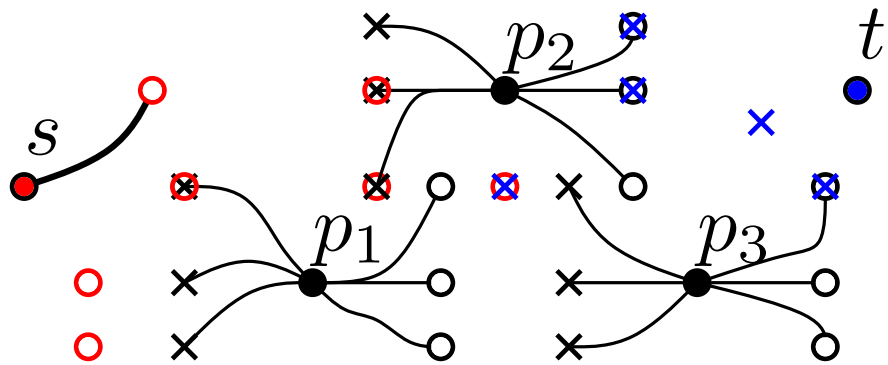
Bester Via Knoten

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$^2 \cdot k$



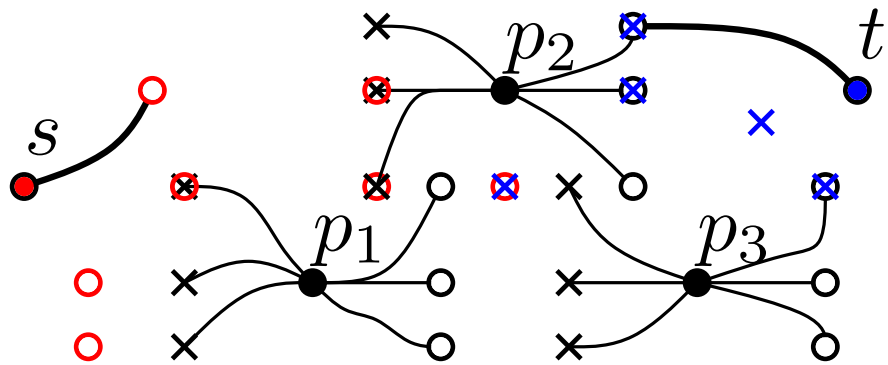
Bester Via Knoten

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$^2 \cdot k$



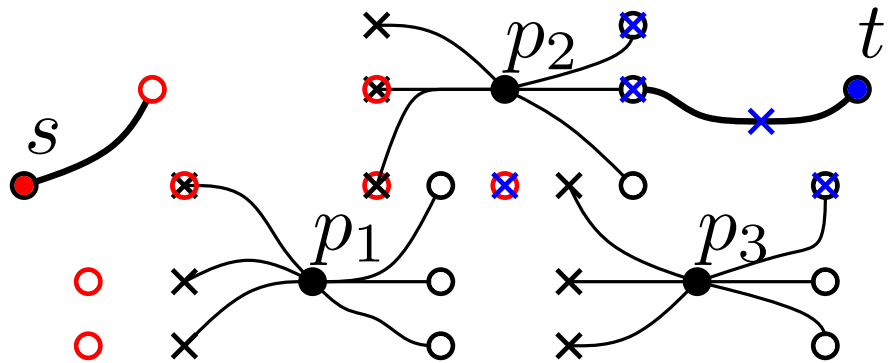
Bester Via Knoten

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$^2 \cdot k$



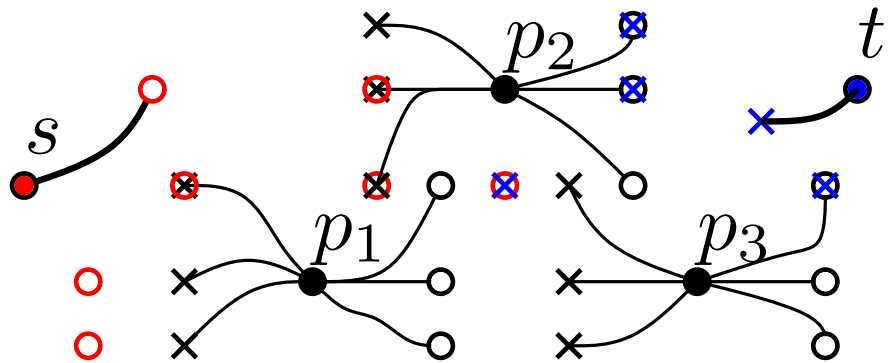
Bester Via Knoten

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$^2 \cdot k$



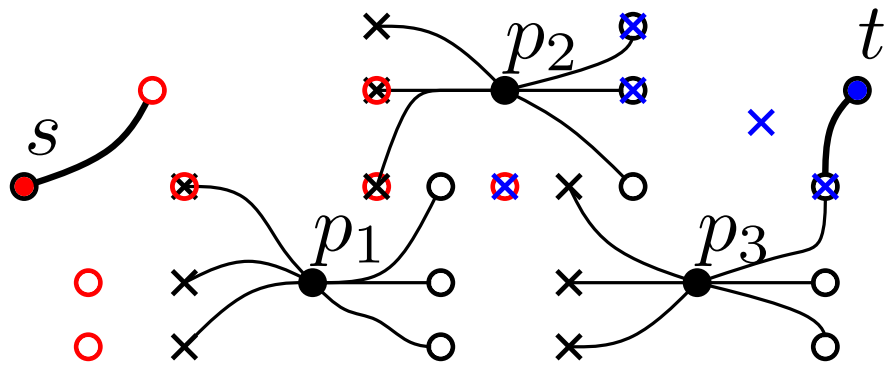
Bester Via Knoten

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$^2 \cdot k$



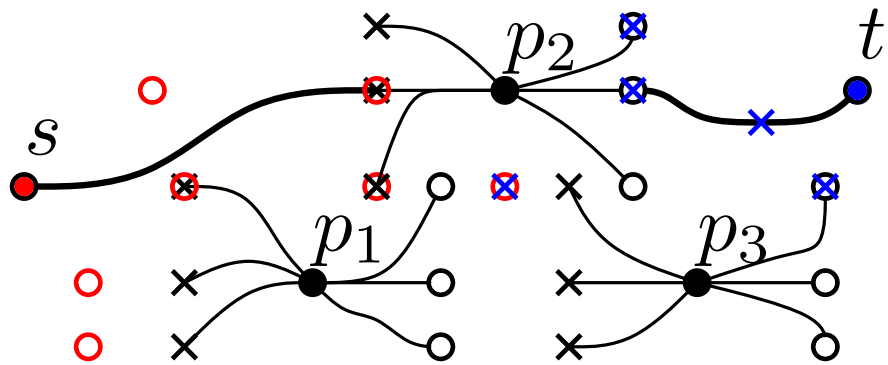
Bester Via Knoten

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
```

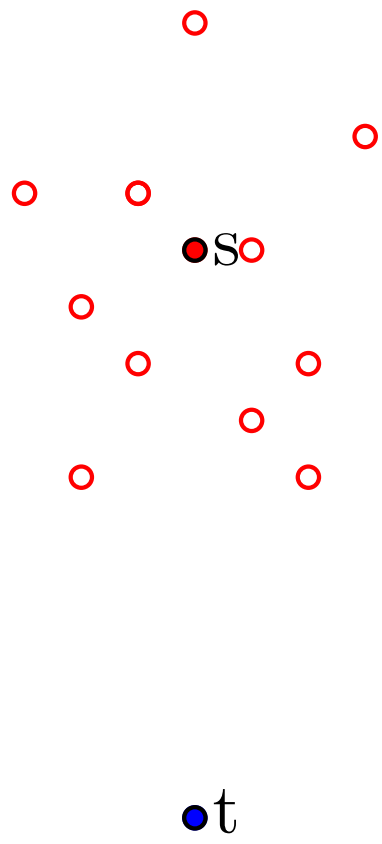
Wiederholung: HLDB SQL Query

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
```



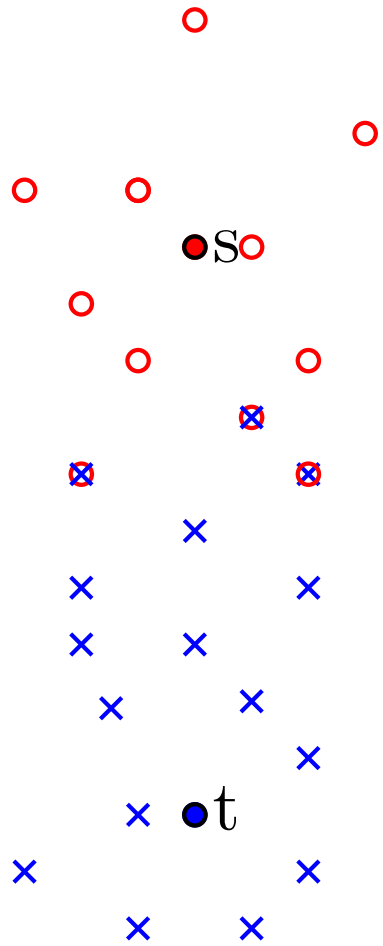
Wiederholung: HLDB SQL Query

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
```



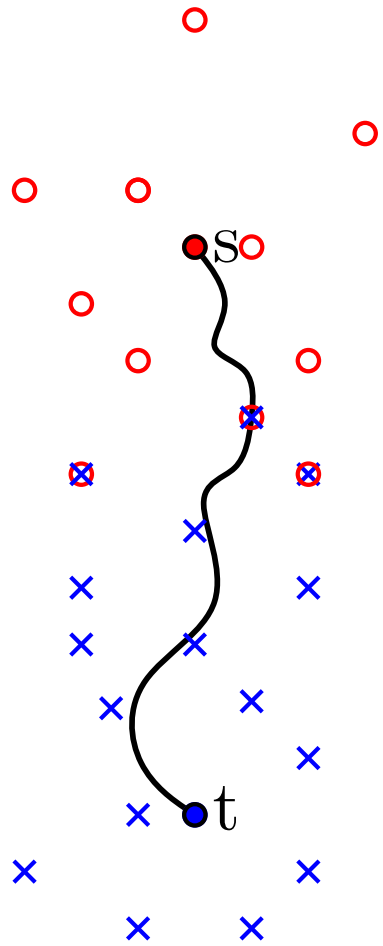
Wiederholung: HLDB SQL Query

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
```



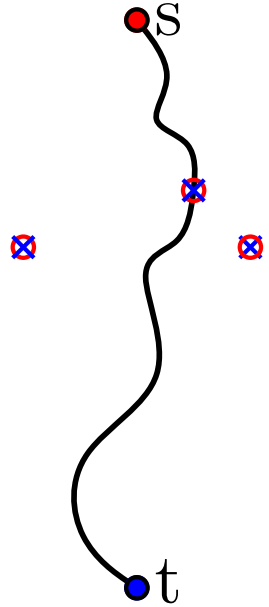
Wiederholung: HLDB SQL Query

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
```



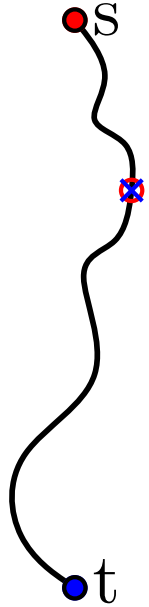
Wiederholung: HLDB SQL Query

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

Algorithm: SQL_DIST

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

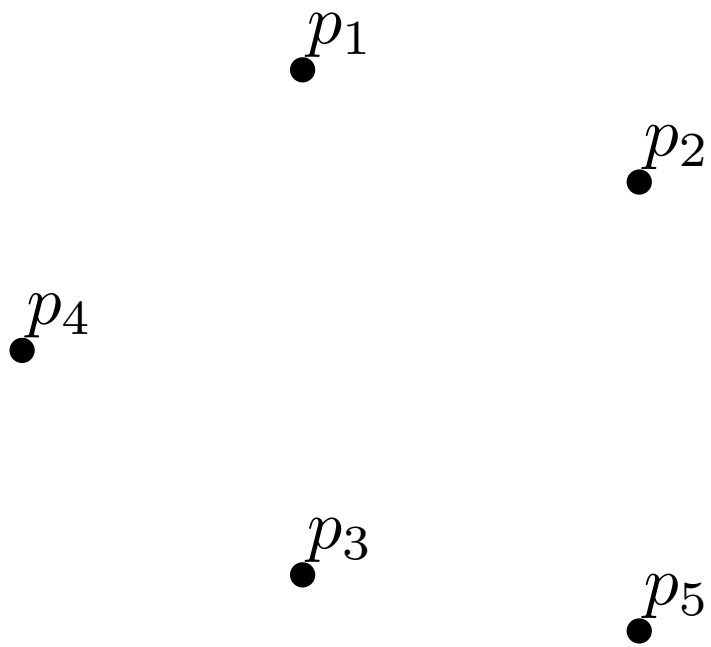
```
1 SELECT
2       MIN(forward.dist+backward.dist)
3 FROM forward,backward
4 WHERE
5       forward.node = s AND
6       backward.node = t AND
7       forward.hub = backward.hub
```



POI Anfragen in SQL

Idee:

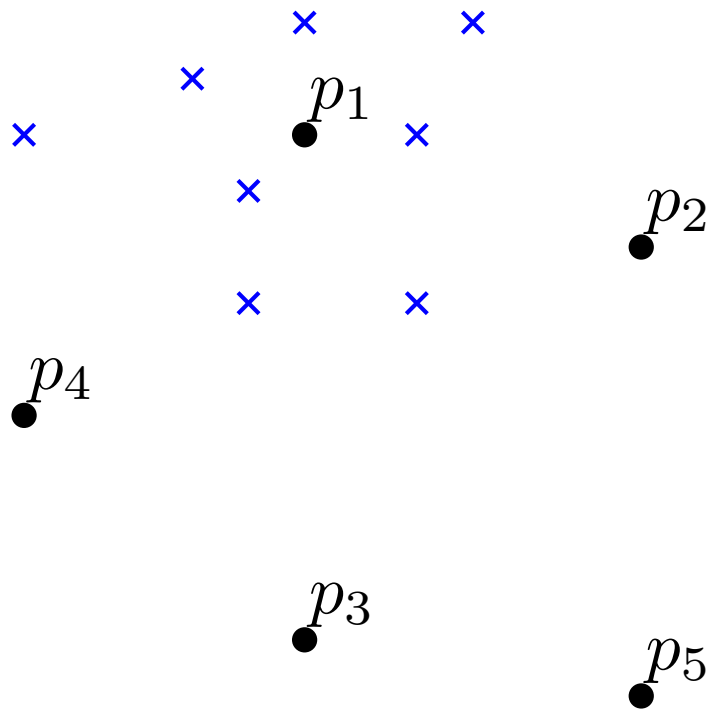
- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`



POI Anfragen in SQL

Idee:

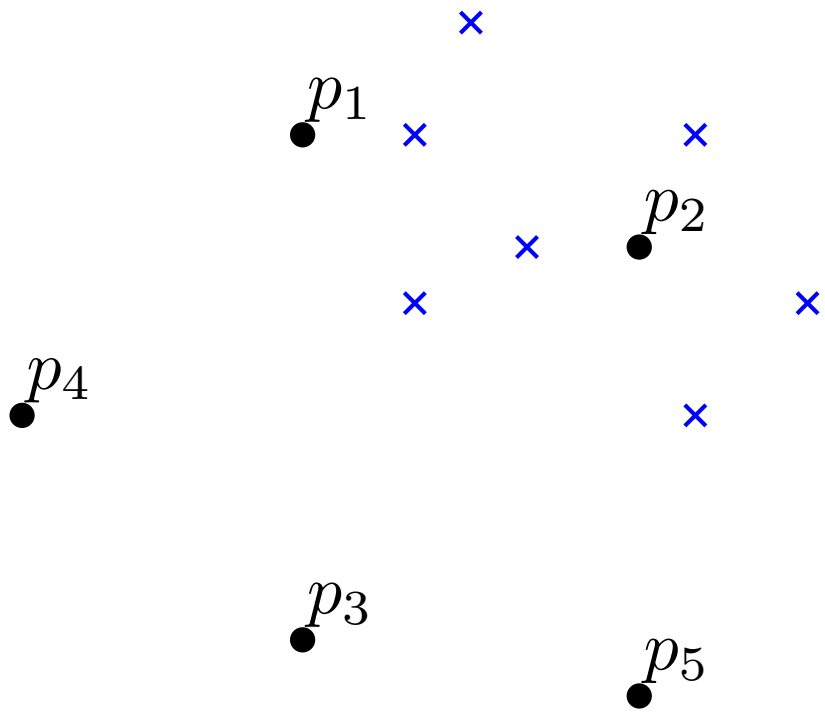
- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`



POI Anfragen in SQL

Idee:

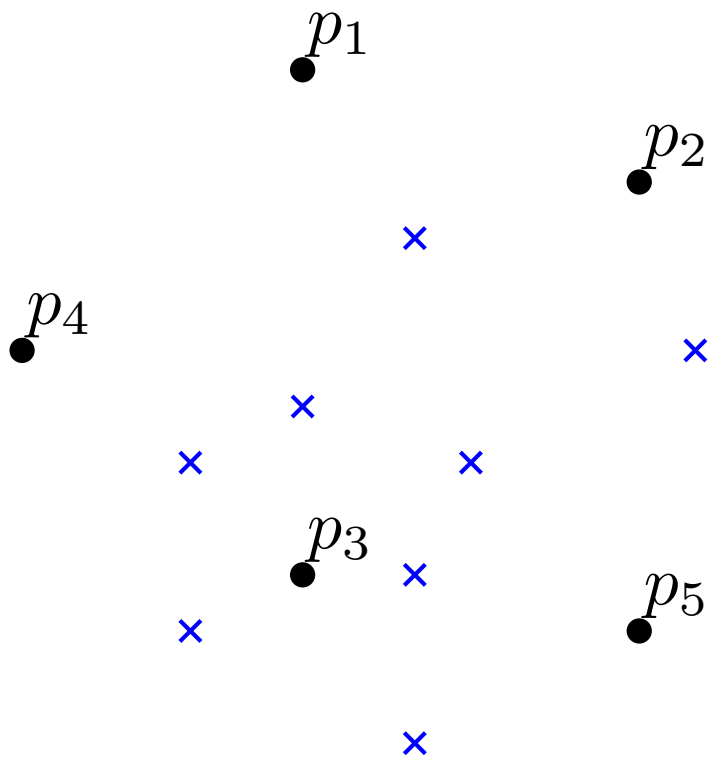
- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`



POI Anfragen in SQL

Idee:

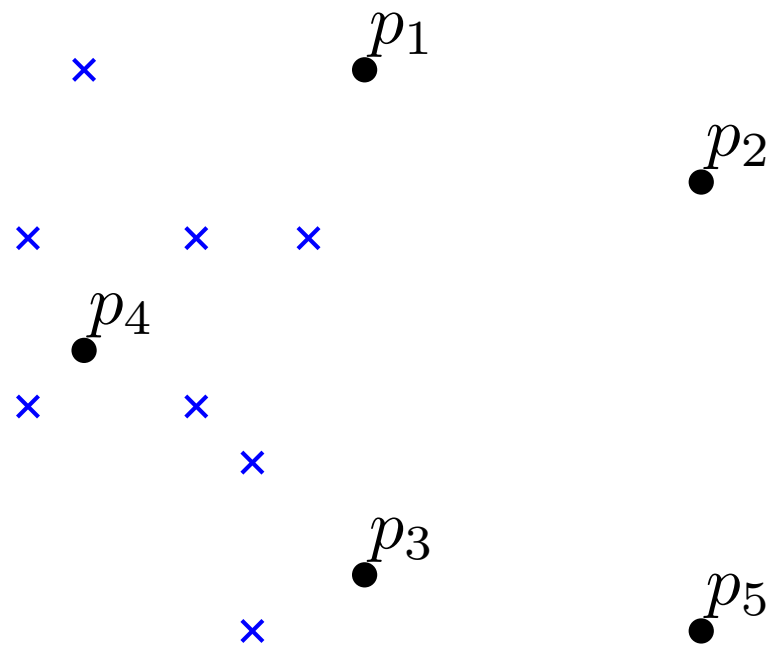
- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`



POI Anfragen in SQL

Idee:

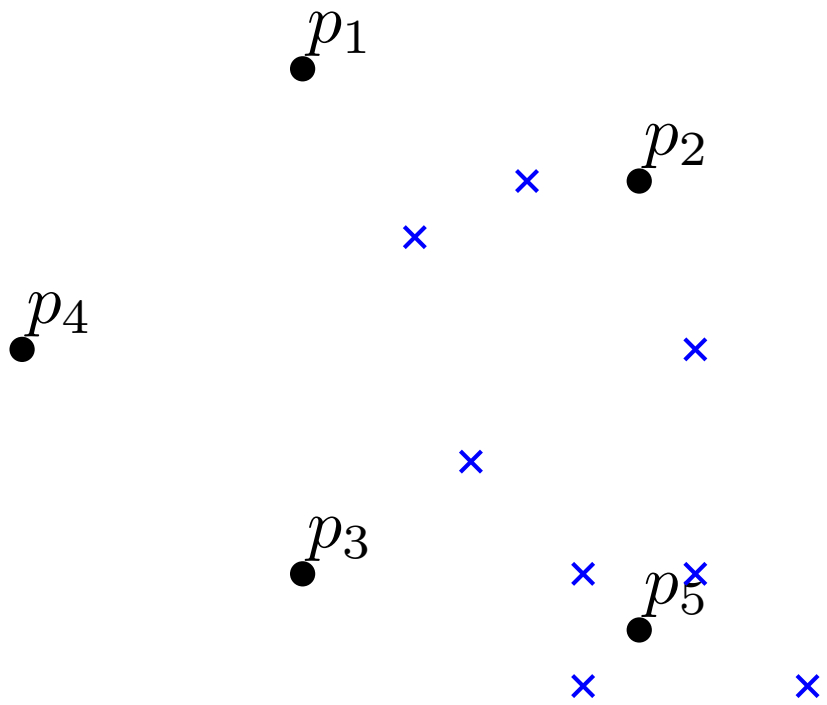
- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`



POI Anfragen in SQL

Idee:

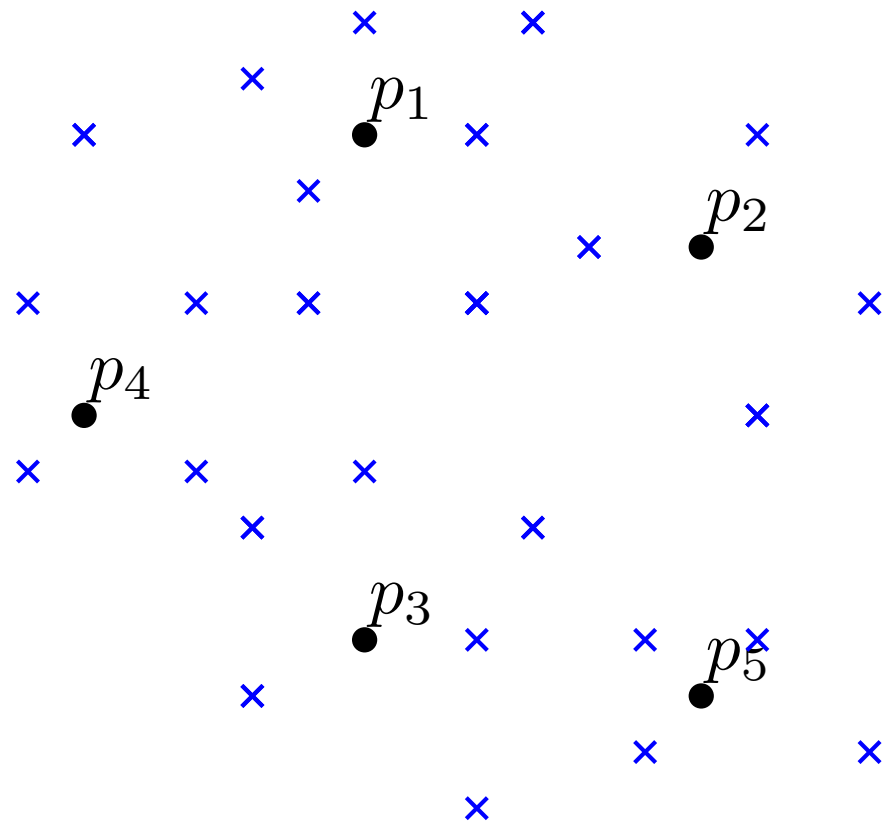
- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`



POI Anfragen in SQL

Idee:

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



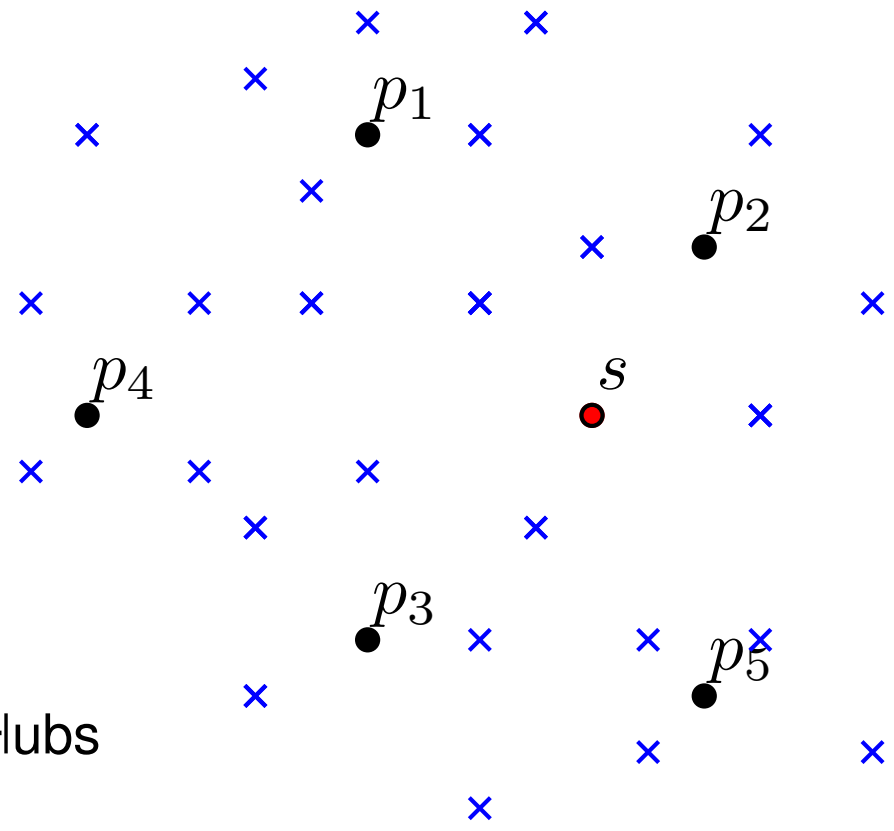
POI Anfragen in SQL

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



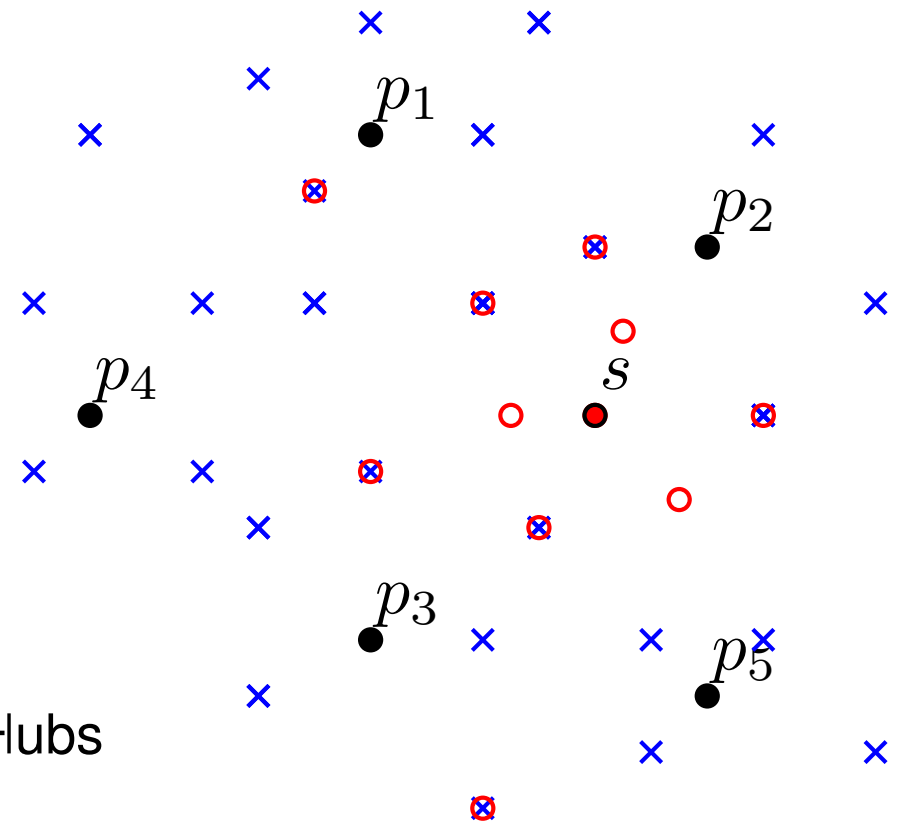
POI Anfragen in SQL

Idee:

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

Query:

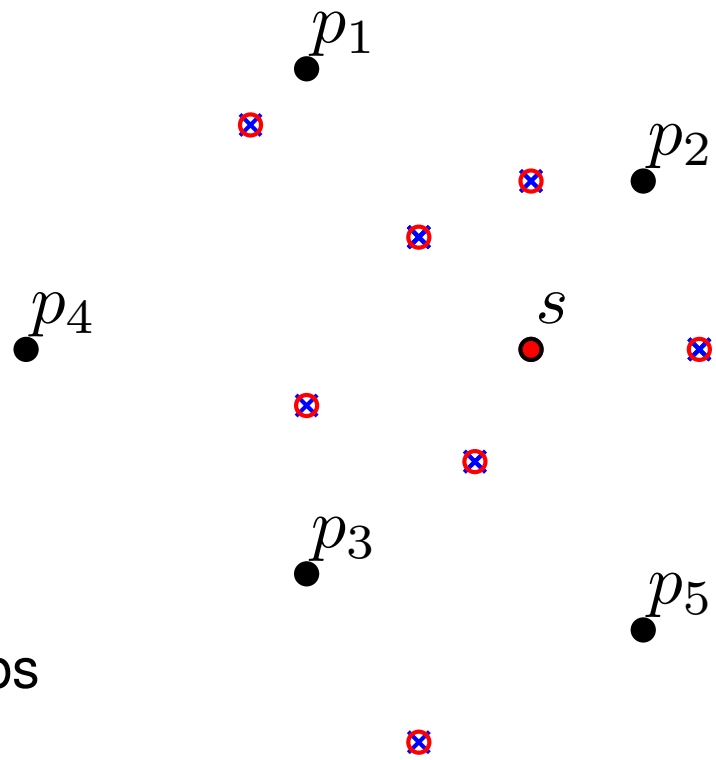
- iteriert über alle ausgehenden Hubs
- für jeden Hub werden nur die (k) nächsten POIs betrachtet
- Antwort: die k insgesamt nächsten POIs



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

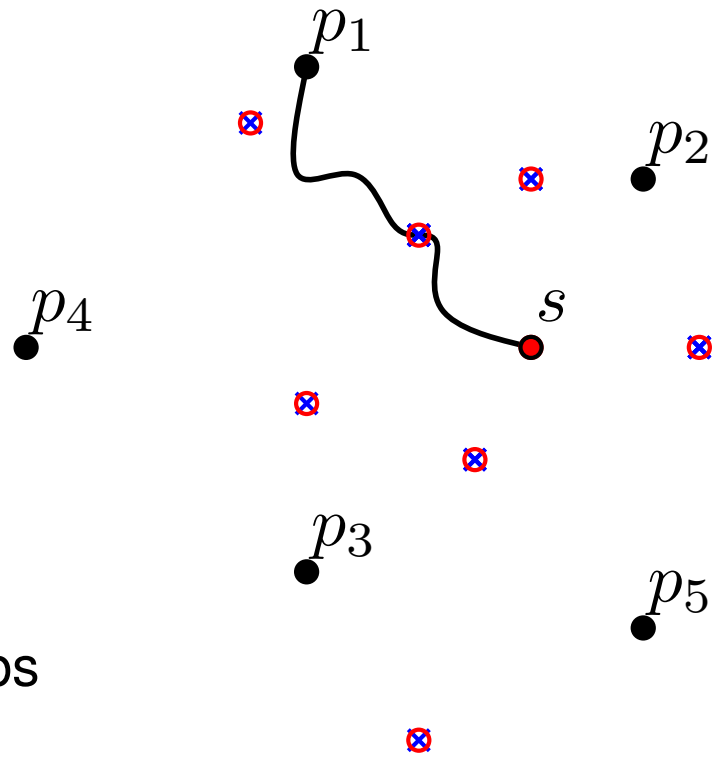
POI Anfragen in SQL

Idee:

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

Query:

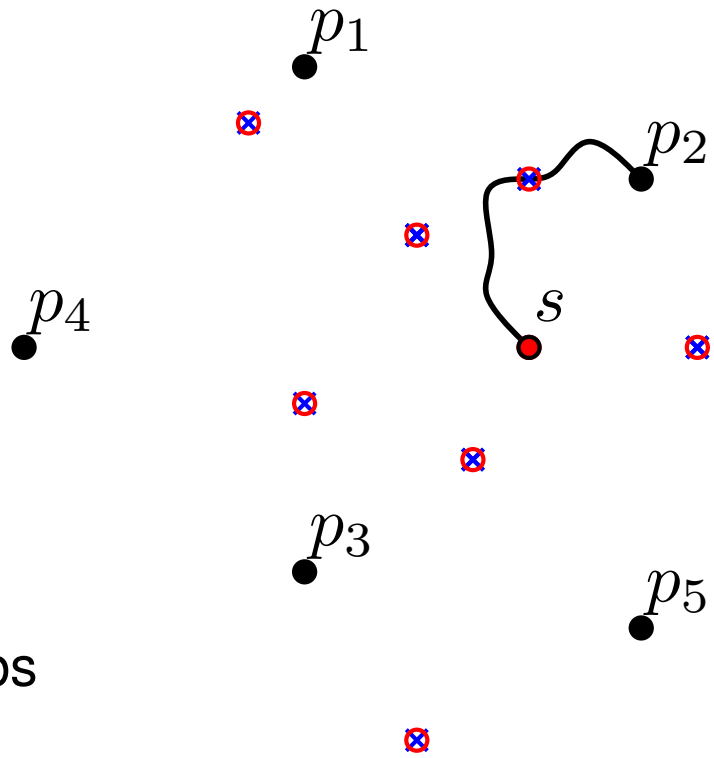
- iteriert über alle ausgehenden Hubs
- für jeden Hub werden nur die (k) nächsten POIs betrachtet
- Antwort: die k insgesamt nächsten POIs



POI Anfragen in SQL

Idee:

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



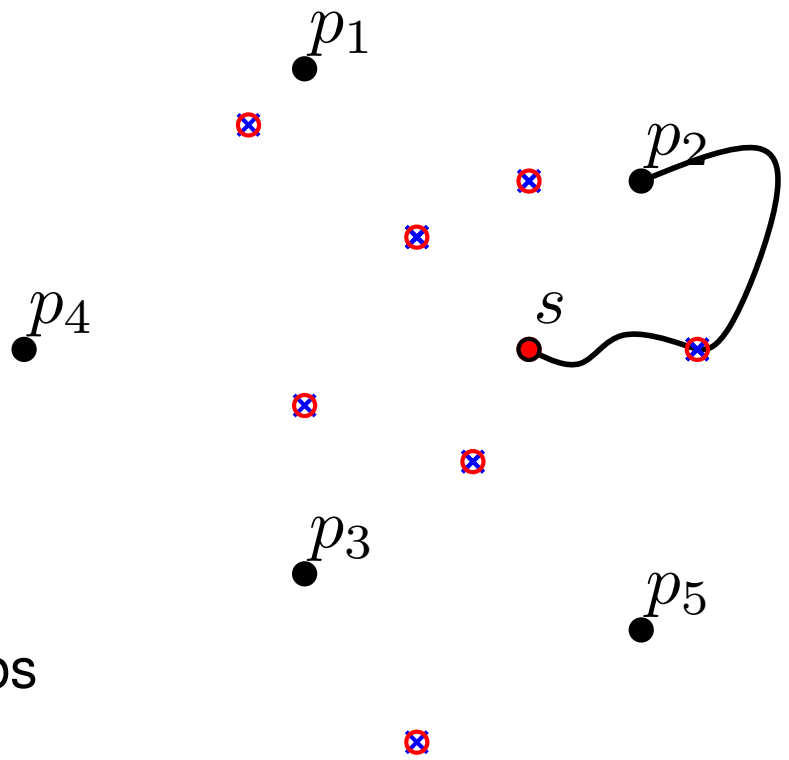
Query:

- 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`



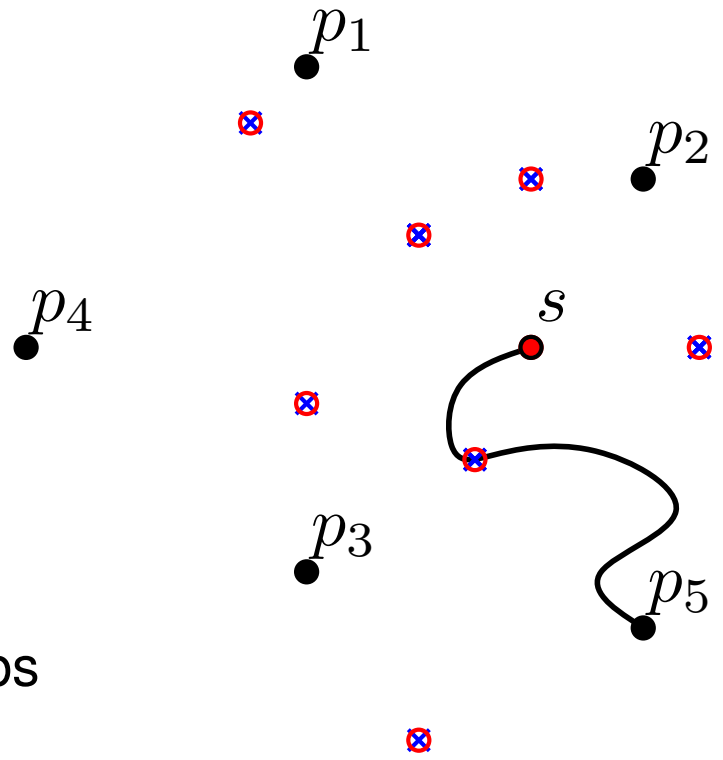
Query:

- 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

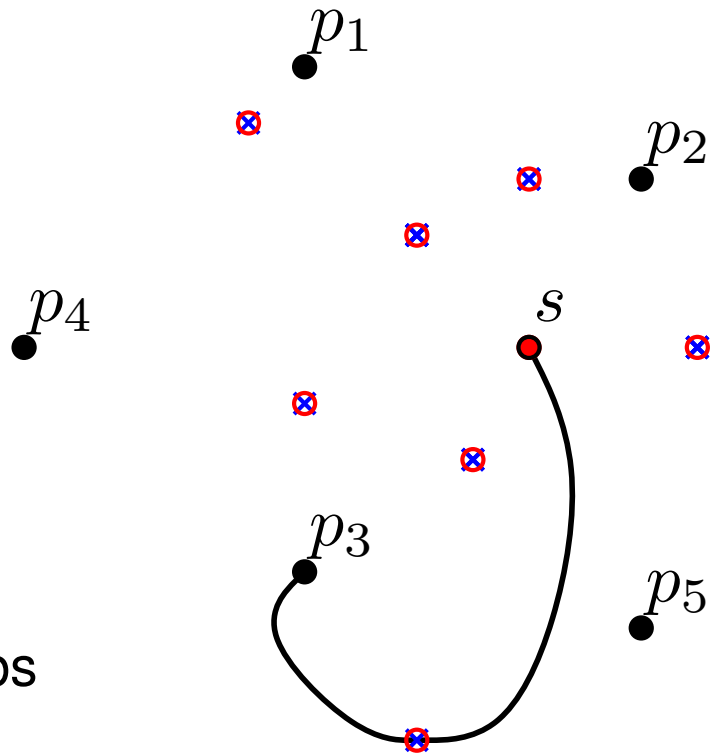
POI Anfragen in SQL

Idee:

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

Query:

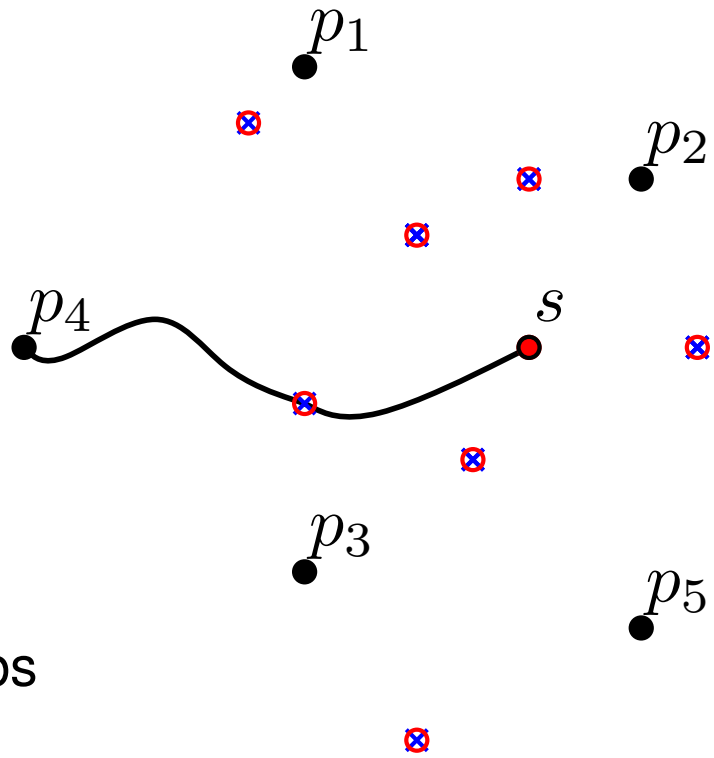
- iteriert über alle ausgehenden Hubs
- für jeden Hub werden nur die (k) nächsten POIs betrachtet
- Antwort: die k insgesamt nächsten POIs



POI Anfragen in SQL

Idee:

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



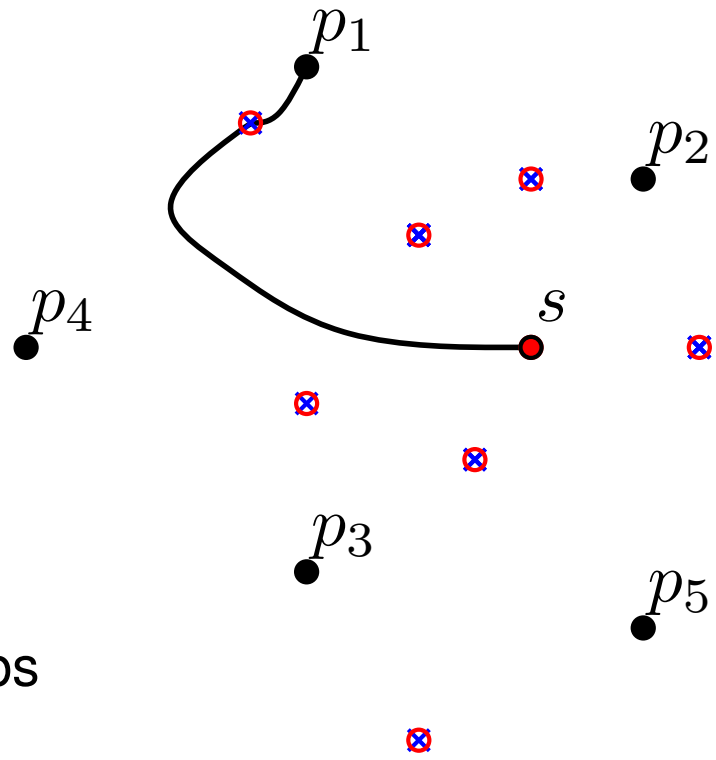
Query:

- 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`



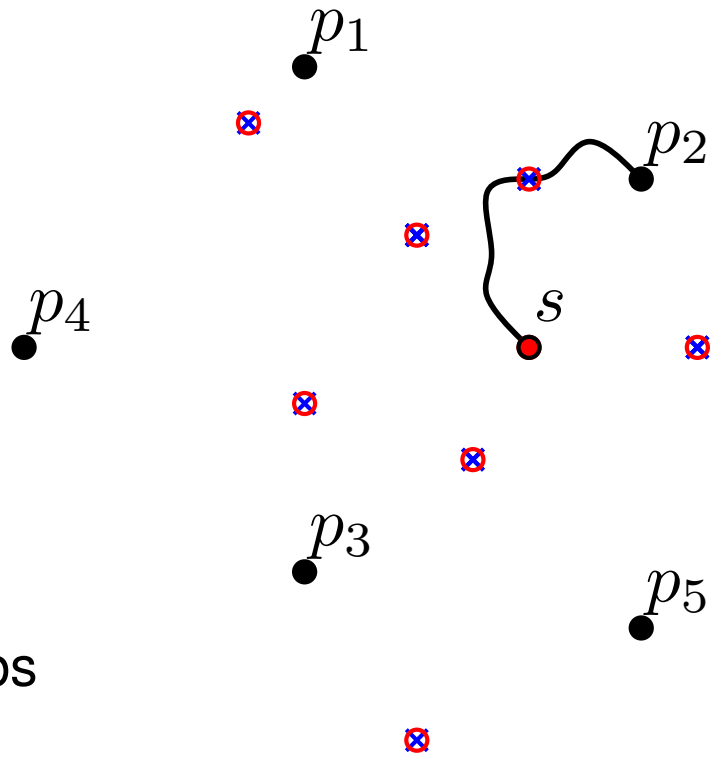
Query:

- 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`

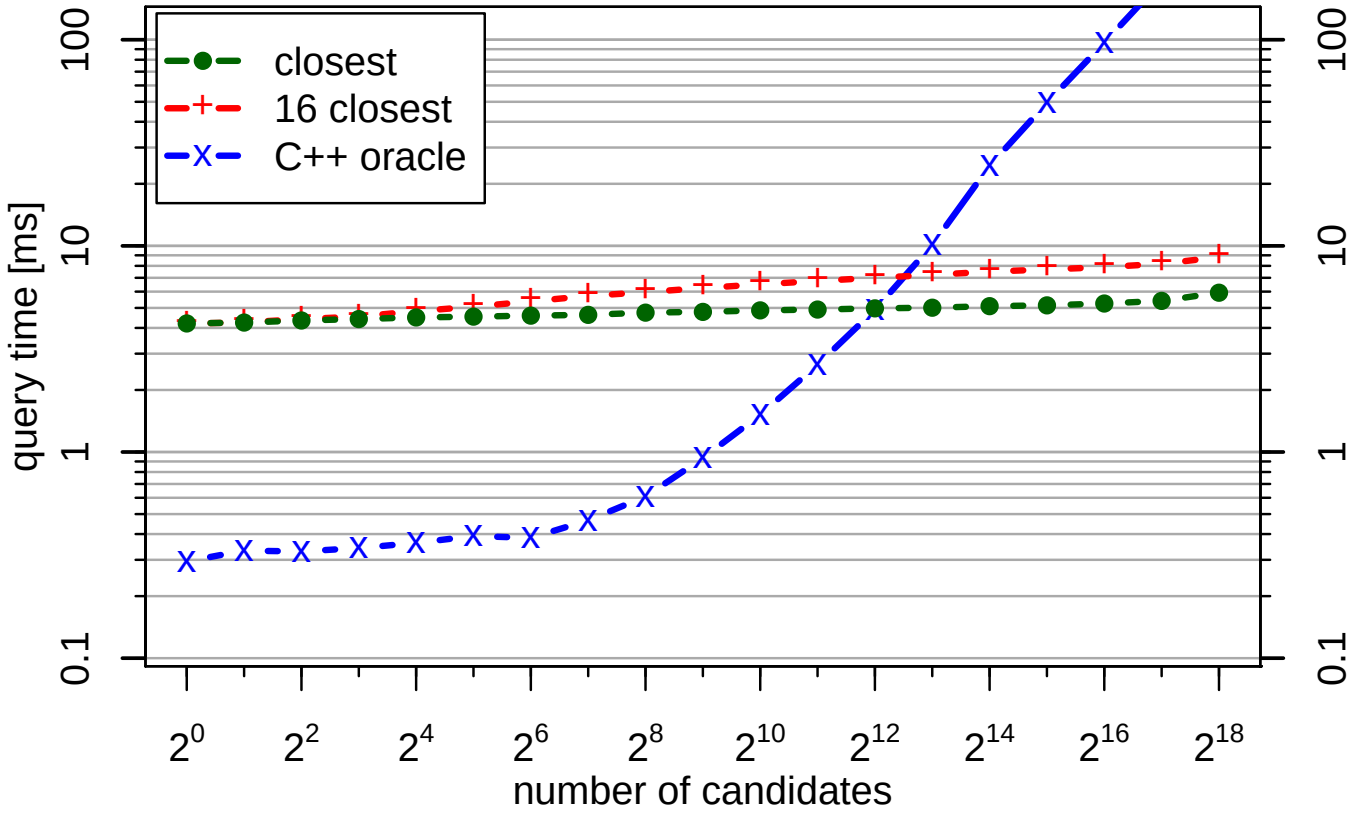


Query:

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

Ergebnisse POI

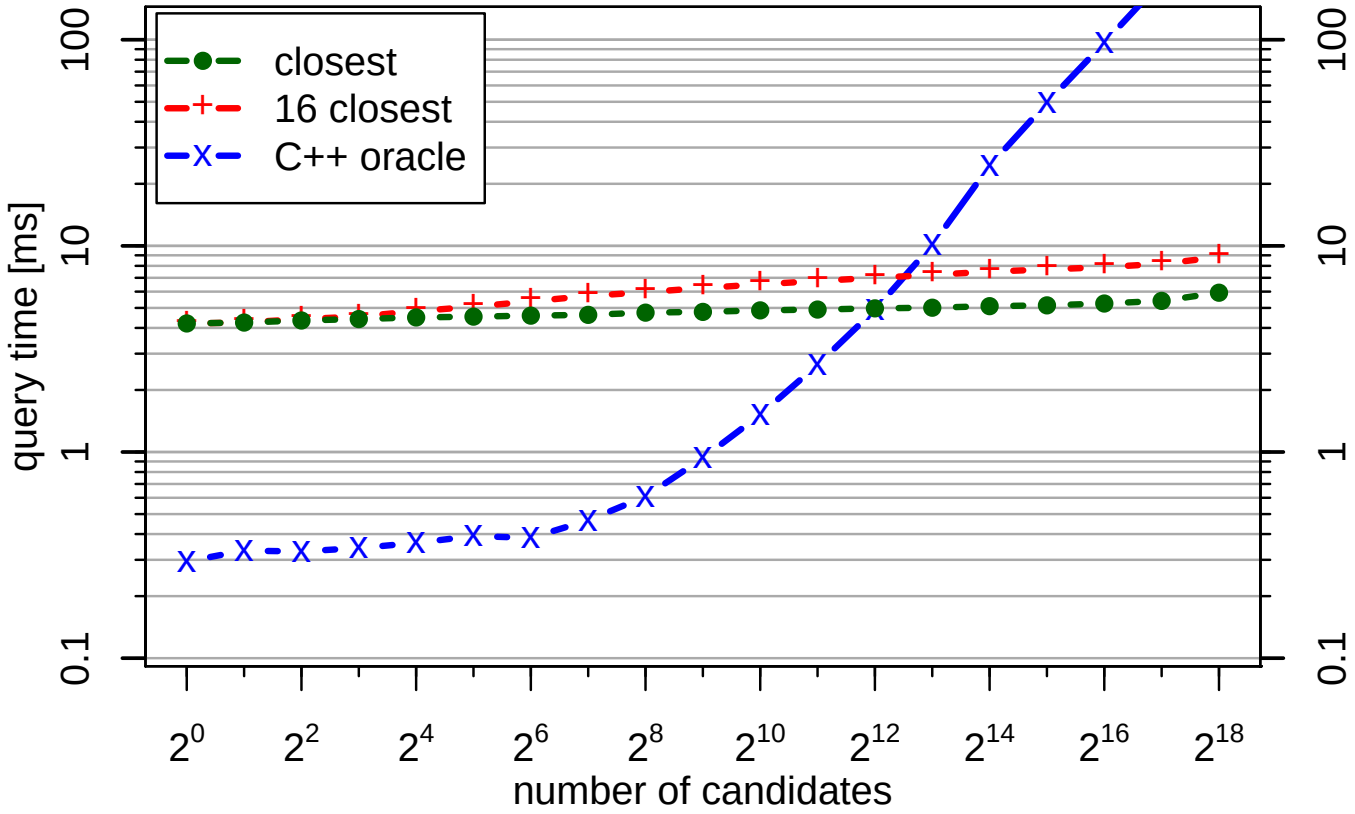
Setup: verschiedene Anzahl POIs, zufällig gewählt



- externes Punkt-zu-Punkt Orakel: skaliert schlecht

Ergebnisse POI

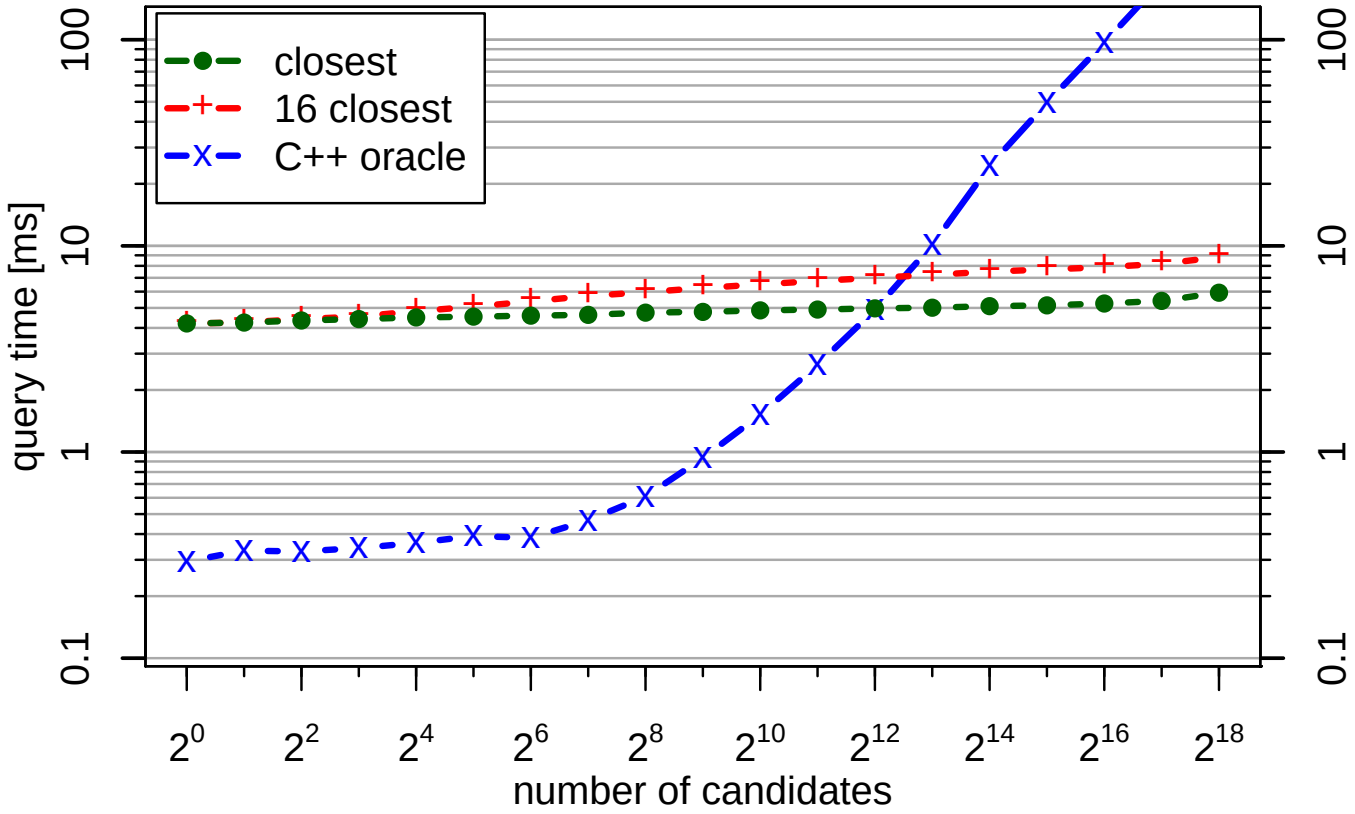
Setup: verschiedene Anzahl POIs, zufällig gewählt



- 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

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

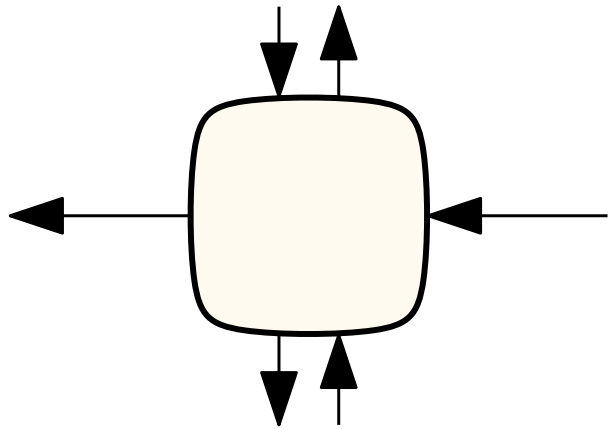
Erweiterte Szenarien

- Abbiegekosten
- Dynamische Routenplanung
- Mobiles Szenario

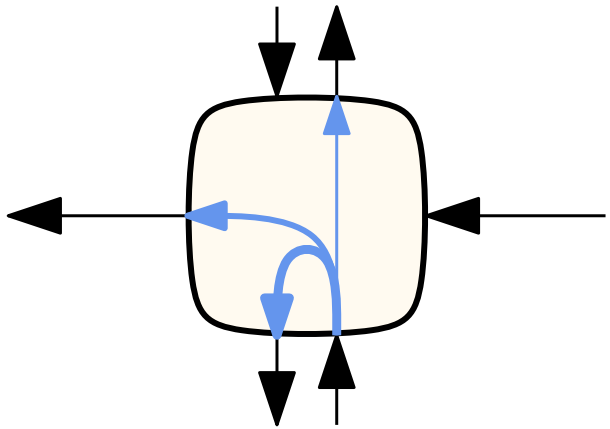
Abbiegeverbote/-kosten

bisher:

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



Abbiegeverbote/-kosten



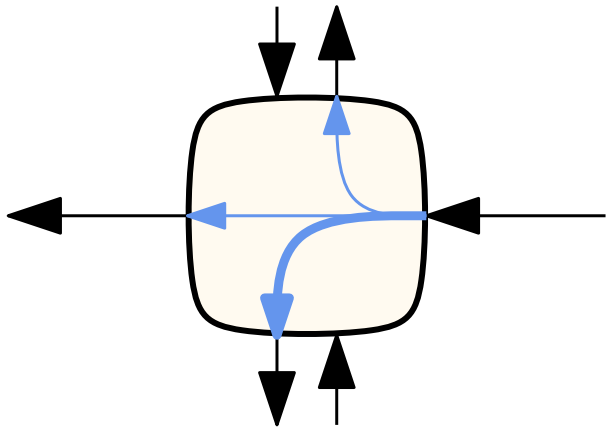
bisher:

- 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



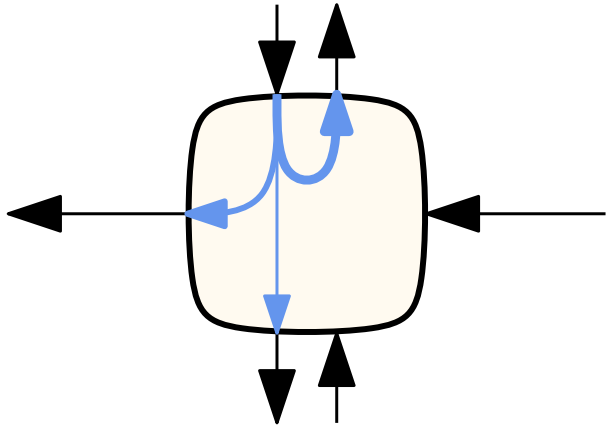
bisher:

- 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

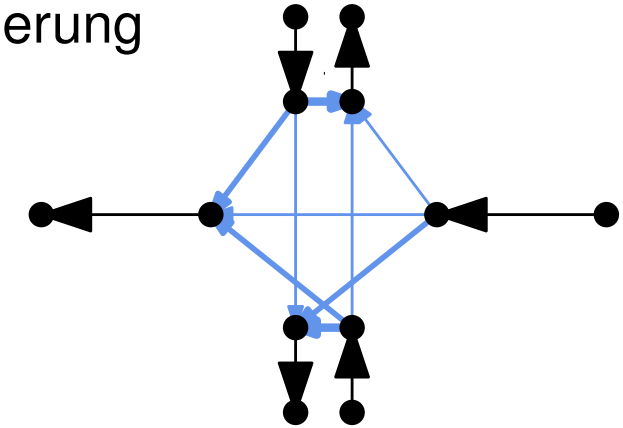
aber:

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

Modellierung

Möglichkeit I:

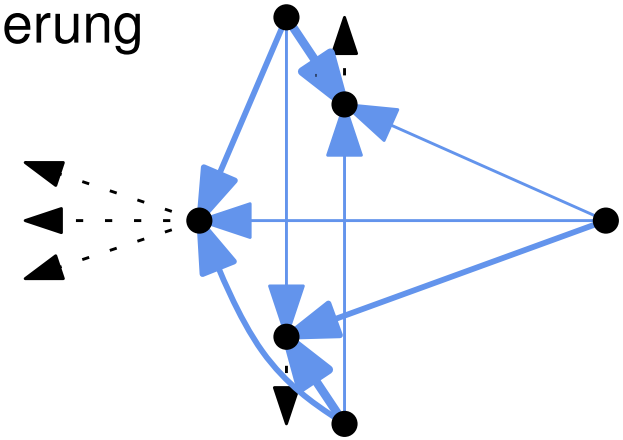
- Vergrössern des Graphen durch Ausmodellierung
- redundante Information



Modellierung

Möglichkeit I:

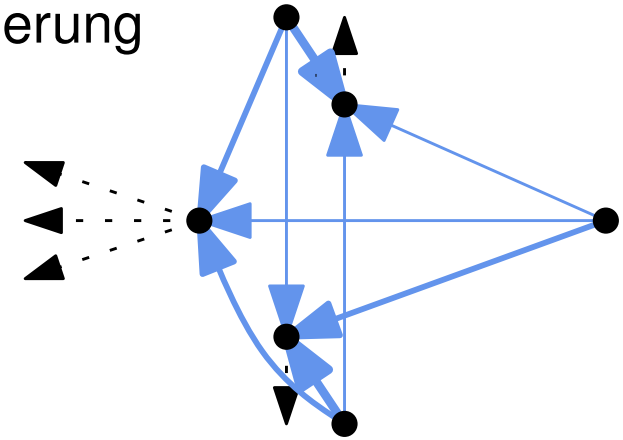
- Vergrössern des Graphen durch Ausmodellierung
- redundante Information
- entferne einen Knoten pro Strasse
- kantenbasierter Graph da
  - Strassen $\rightarrow$ Knoten
  - Turns $\rightarrow$ Kanten



Modellierung

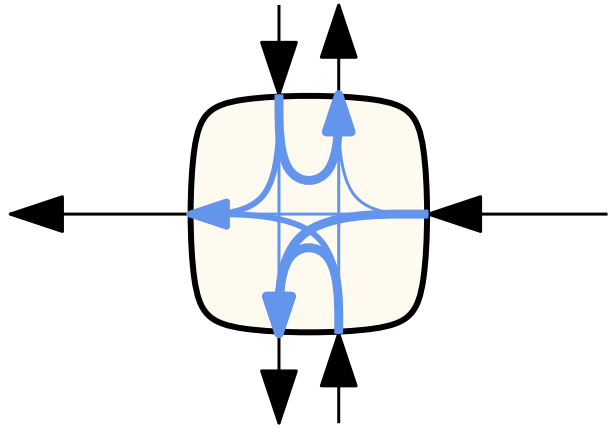
Möglichkeit I:

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



Möglichkeit II:

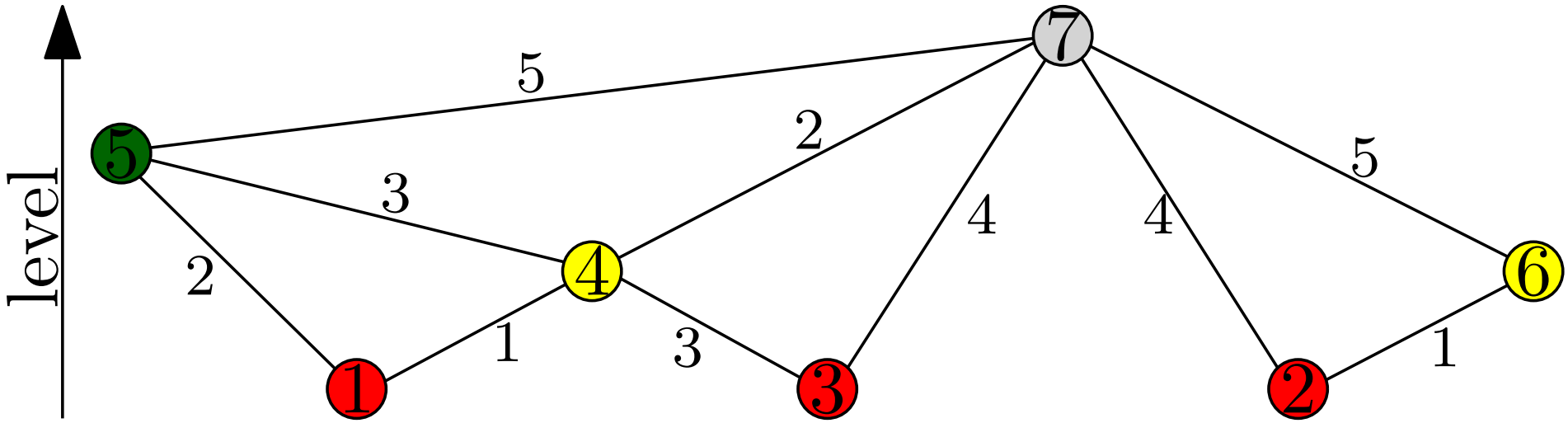
- behalte Kreuzungen als Knoten
- speicher Abbiegetabelle
  Abb. Eingangs- $\times$ Ausgangspunkte $\rightarrow$ Kosten
- Beobachtung: viele Knoten haben die gleiche Abbiegetabelle
- also speicher jede Tabelle einmal, Knoten speichern Tabellen-ID



Wdh: Contraction Hierarchies

Preprocessing:

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



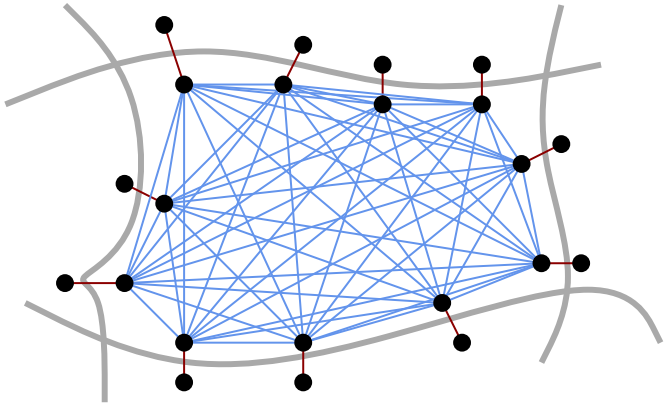
Wdh: Multi Level Dijkstra

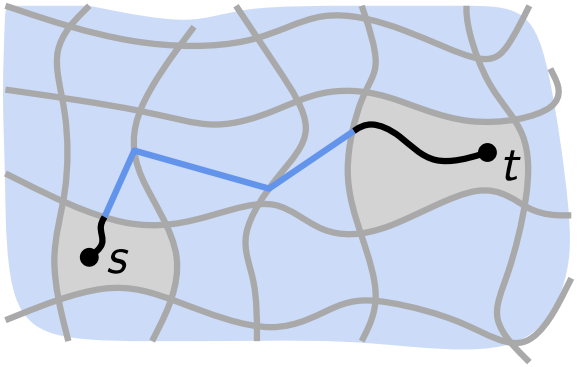
Idee:

- 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

Optimierung:

- berechne für jede Kreuzung Schranken
- reduziert Suchraum wieder um ca einen Faktor 2
  aber immer noch langsamer als ohne Turns

Kompaktes Modell

CH

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Kompaktes Modell

CH

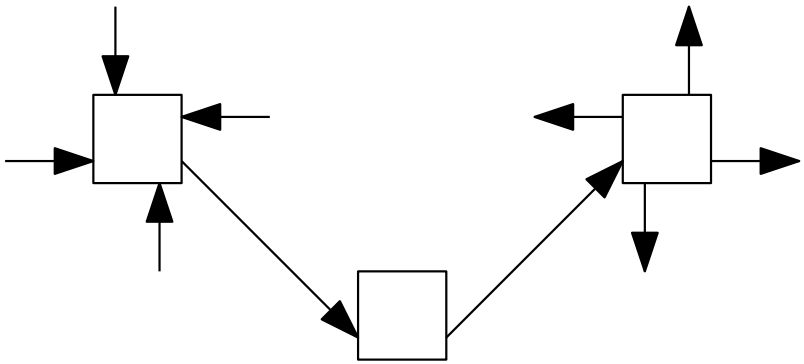
- Zeugensuche wird komplizierter

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Kompaktes Modell

CH

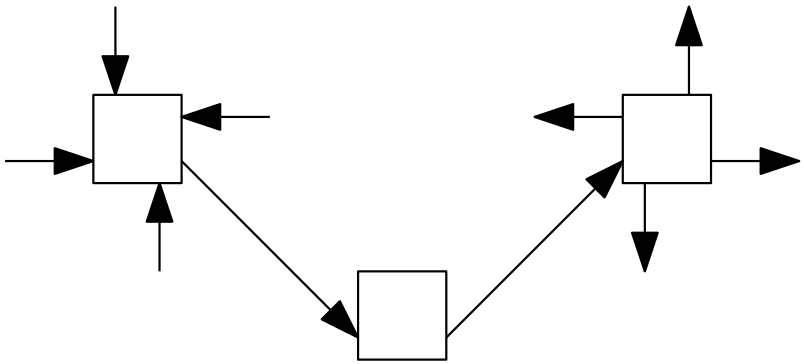
- Zeugensuche wird komplizierter
  - für jedes Paar eingehender und ausgehender Kanten muss eine Zeugensuche durchgeführt werden



Kompaktes Modell

CH

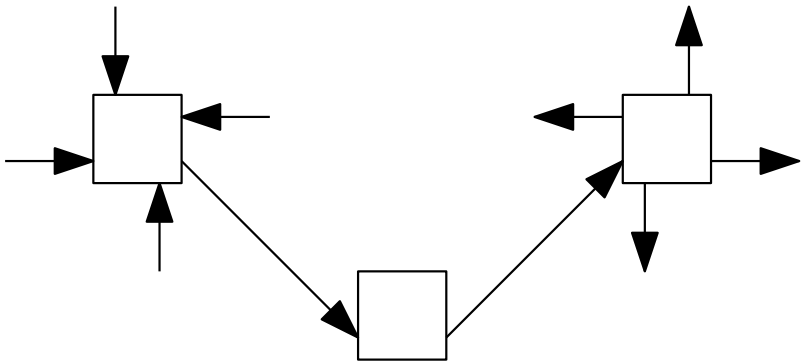
- Zeugensuche wird komplizierter
  - für jedes Paar eingehender und ausgehender Kanten muss eine Zeugensuche durchgeführt werden
  - es können Self-Loops entstehen

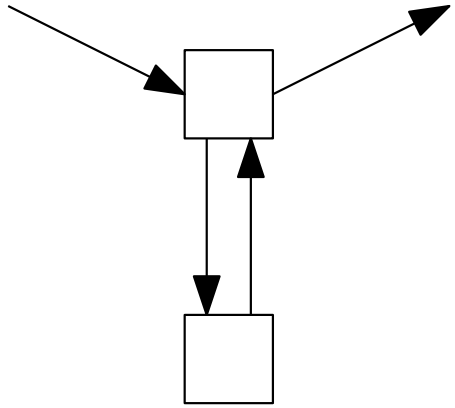


Kompaktes Modell

CH

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

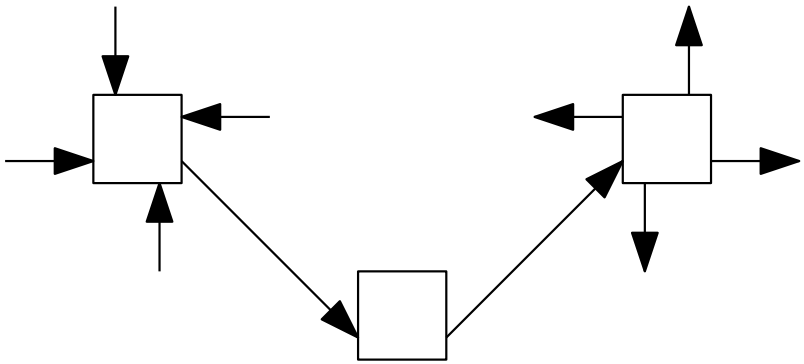


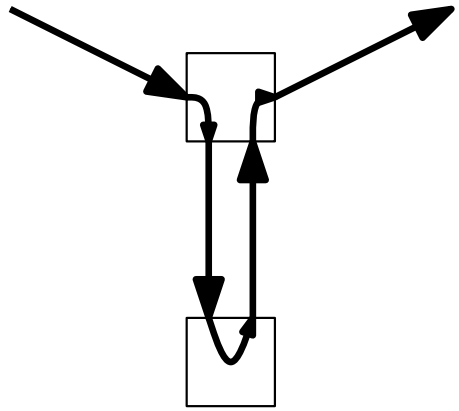


Kompaktes Modell

CH

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

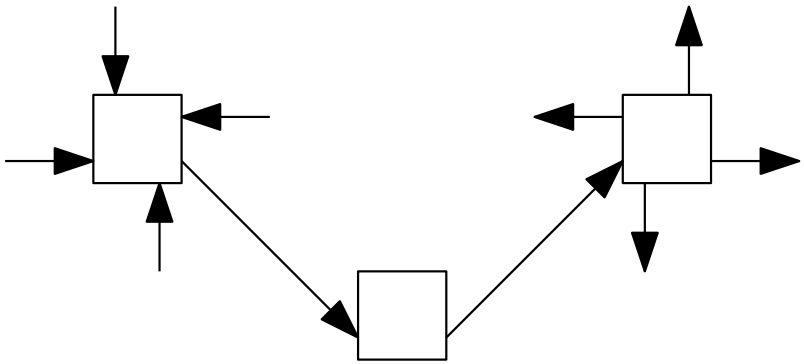


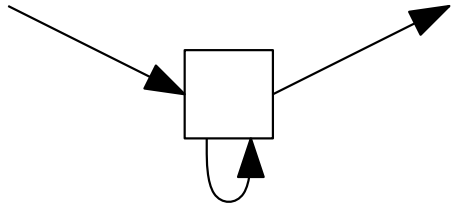


Kompaktes Modell

CH

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





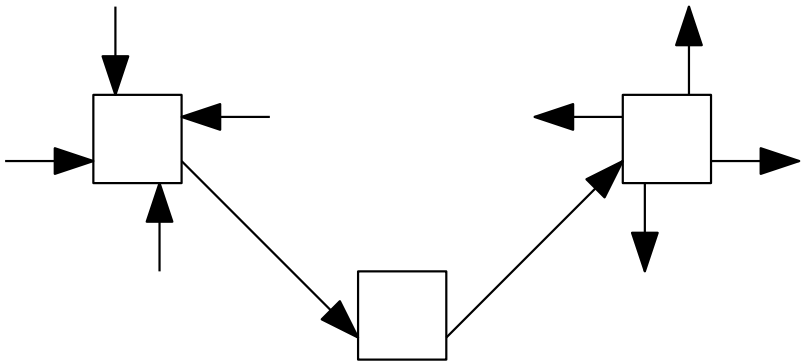
Kompaktes Modell

CH

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

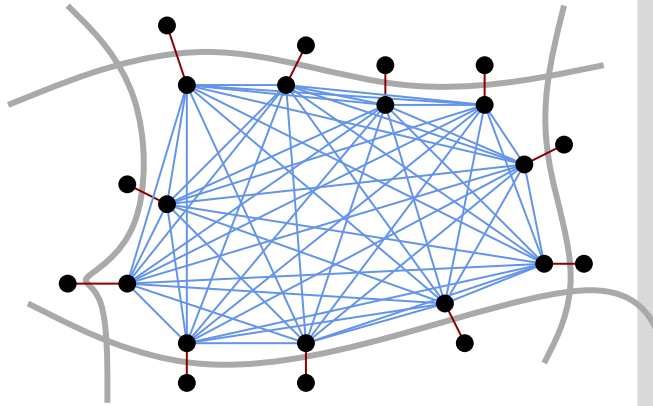
$\Rightarrow$ Anpassung schwierig

$\Rightarrow$ Zeugensuche schlägt häufig fehl



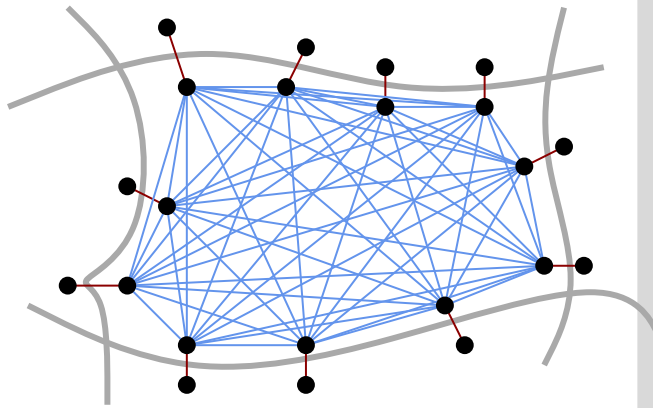
Kompaktes Modell

MLD



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Kompaktes Modell



MLD

- Schnittkanten bleiben erhalten
- Schnittkante $\rightarrow$ 2 Knoten auf Overlay
- Turns müssen nur auf unterstem 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:

- CH hat massive Probleme mit Turns
- MLD kaum (einer der Hauptgründe für Entwicklung von MLD)

Diskussion

Stoff bisher

- Problemtransformation auf Graph + Dijkstra
- Vorberechnungstechniken (point-to-point)
- Weitere Anfrageszenarien: one-to-all, many-to-many, POIs
- Turn-Costs
- Alternativrouten (fast optimal, hinreichend unterschiedlich)

Diskussion

Stoff bisher

- Problemtransformation auf Graph + Dijkstra
- Vorberechnungstechniken (point-to-point)
- Weitere Anfrageszenarien: one-to-all, many-to-many, POIs
- Turn-Costs
- Alternativrouten (fast optimal, hinreichend unterschiedlich)

Was fehlt noch für Einsatz in Produktion?

Diskussion

Stoff bisher

- Problemtransformation auf Graph + Dijkstra
- Vorberechnungstechniken (point-to-point)
- Weitere Anfrageszenarien: one-to-all, many-to-many, POIs
- Turn-Costs
- Alternativrouten (fast optimal, hinreichend unterschiedlich)

Was fehlt noch für Einsatz in Produktion?

- Berücksichtigung von (aktuellen) Staus
- Implementation auf Mobilgeräten
- Berücksichtigung von historischem Wissen über Staus
- Was ist eigentlich mit Bus+Bahn?

Dynamische Szenarien

Szenario:

- Unfall auf einer Straße
- Reisezeit ändert sich auf dieser Straße
- berechne schnellsten Weg bezüglich der aktualisierten Reisezeiten

Herausforderung

Hauptproblem:

- Kantengewichte ändern sich
- Vorberechnung basiert auf ursprünglichen Kantengewichten
- komplette Vorberechnung für jeden Stau wenig sinnvoll

Herausforderung

Hauptproblem:

- Kantengewichte ändern sich
- Vorberechnung basiert auf ursprünglichen Kantengewichten
- komplette Vorberechnung für jeden Stau wenig sinnvoll

Serverszenario:

- identifiziere den Teil der beeinträchtigten Vorberechnung
- aktualisiere diesen Teil

Herausforderung

Hauptproblem:

- Kantengewichte ändern sich
- Vorberechnung basiert auf ursprünglichen Kantengewichten
- komplette Vorberechnung für jeden Stau wenig sinnvoll

Serverszenario:

- identifiziere den Teil der beeinträchtigten Vorberechnung
- aktualisiere diesen Teil

mobiles Szenario:

- robuste Vorberechnung
- immer noch korrekt, wenn Kantengewichte sich erhöhen
- dadurch eventuell Anfragen langsamer
- oder passe Query an, so dass Vorberechnung nicht aktualisiert werden muss

Landmarken [Goldberg, Harrelson 05]

Vorberechnung:

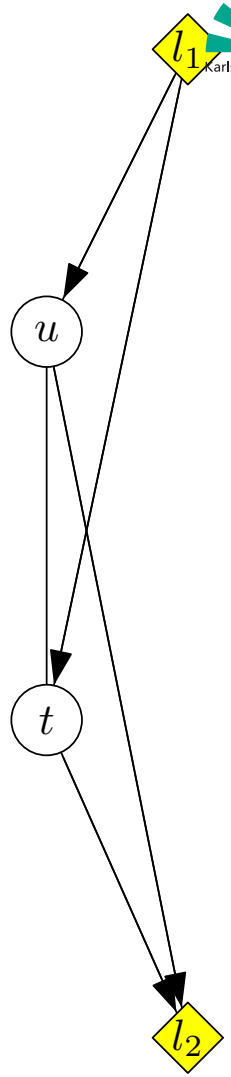
- wähle eine Hand voll (≈ 16) Knoten als Landmarken
- berechne Abstände von und zu allen Landmarken

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



Serverszenario

Beobachtung:

- Landmarkenwahl ist Heuristik, also nicht ändern
- Distanzen von und zu Landmarken sind nicht korrekt

Vorgehen:

- aktualisiere diese Distanzen
- benutze dafür dynamisierte Dijkstra-Variante, die nur betroffene Teilbäume (jeder Landmarke) neuberechnet
- dann Anfragen korrekt und (in etwa) so schnell wie bei kompletter neuer Vorberechnung

Mobiles Szenario

Beobachtung:

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

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

- durch Erhöhen der Kantengewichte wird dies nicht verletzt
- durch Staus können Reisezeiten nicht unter Initialwert fallen

Mobiles Szenario

Beobachtung:

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

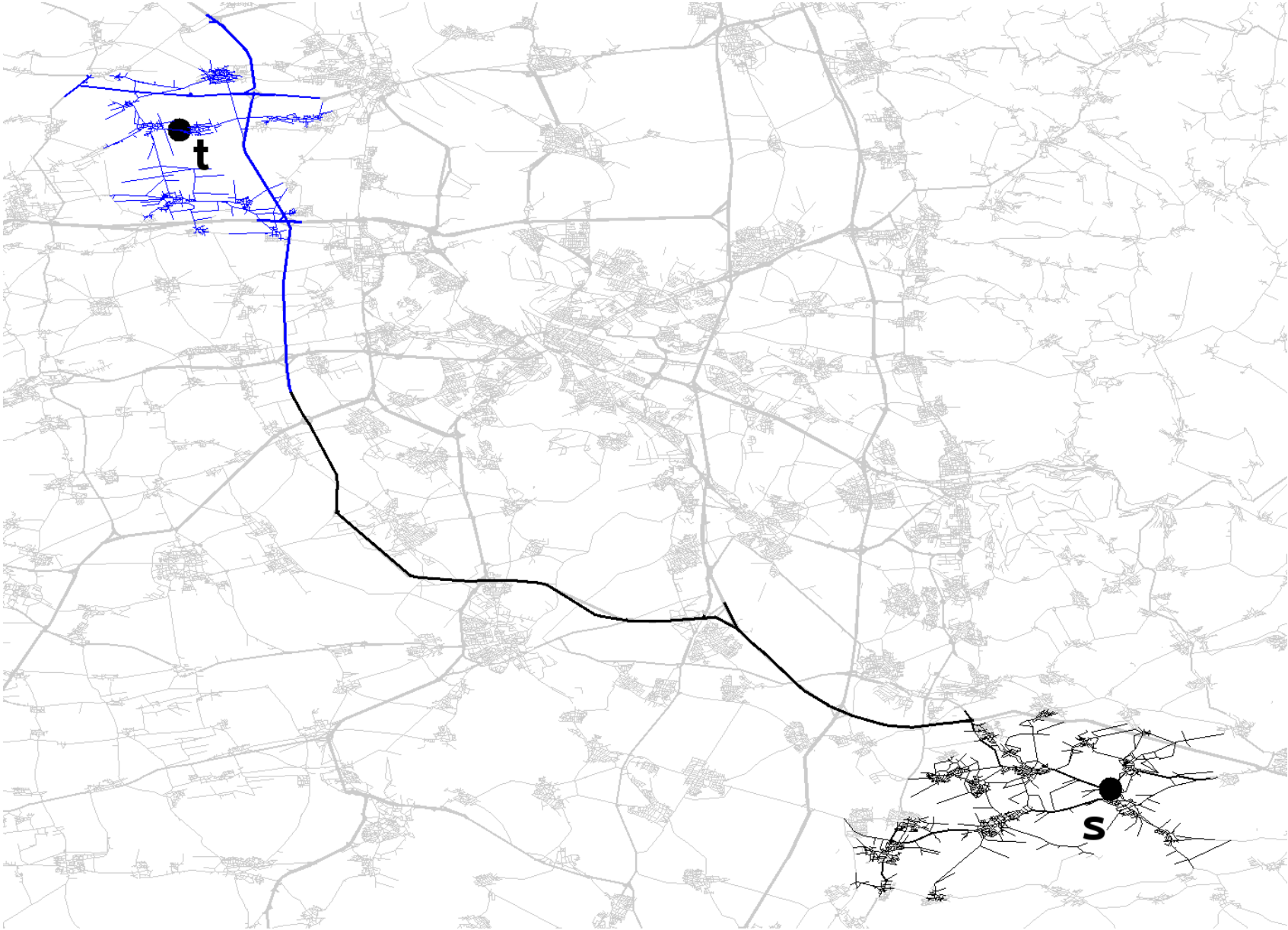
$$\text{len}_\pi(u, v) = \text{len}(u, v) - \pi(u) + \pi(v) \geq 0$$

- durch Erhöhen der Kantengewichte wird dies nicht verletzt
- durch Staus können Reisezeiten nicht unter Initialwert fallen

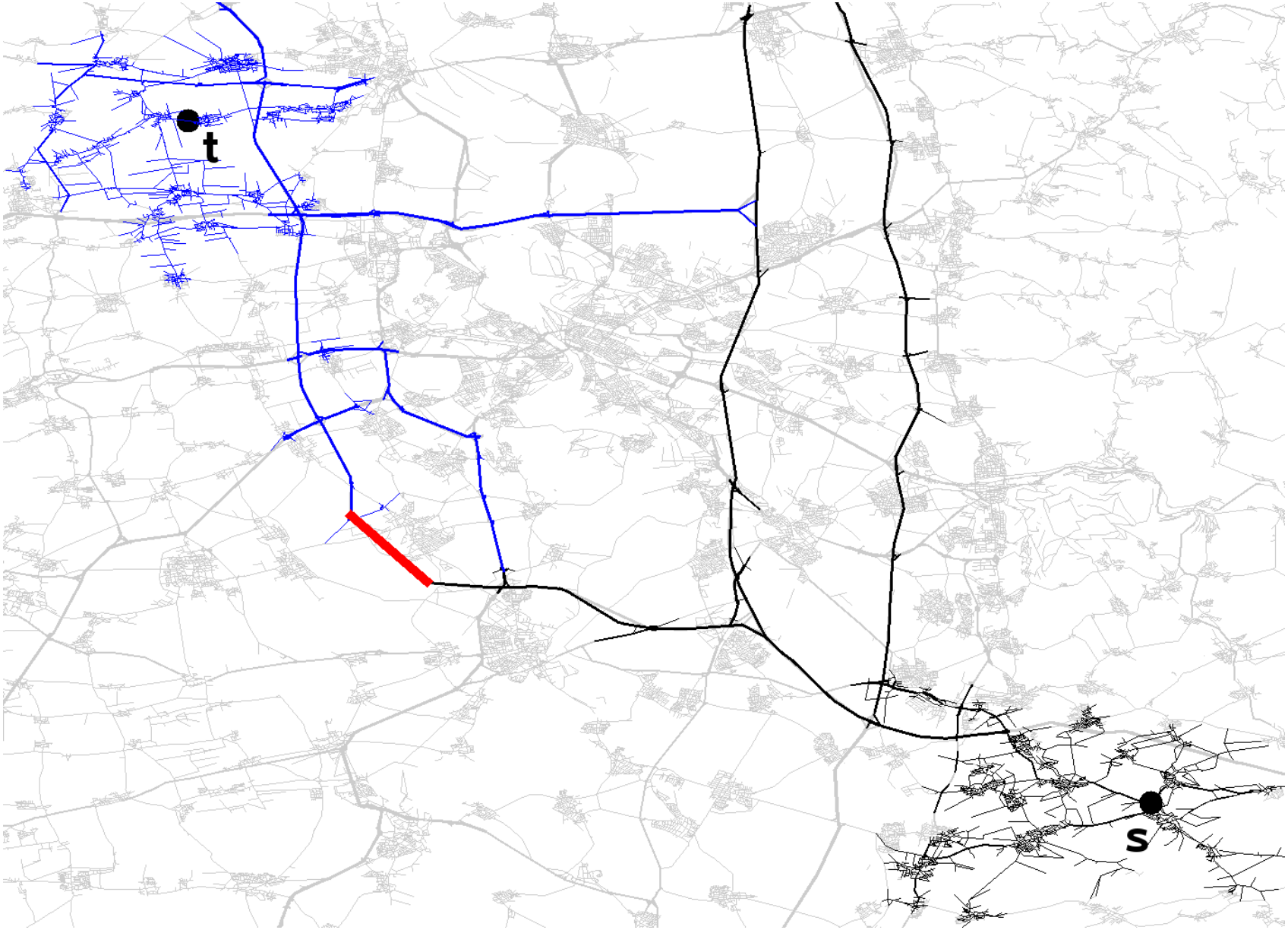
Idee:

- Vorberechnung auf staufreiem Graphen
- Suchanfragen ohne Aktualisierung
- korrekt aber eventuell langsamere Anfragezeiten

Beispiel

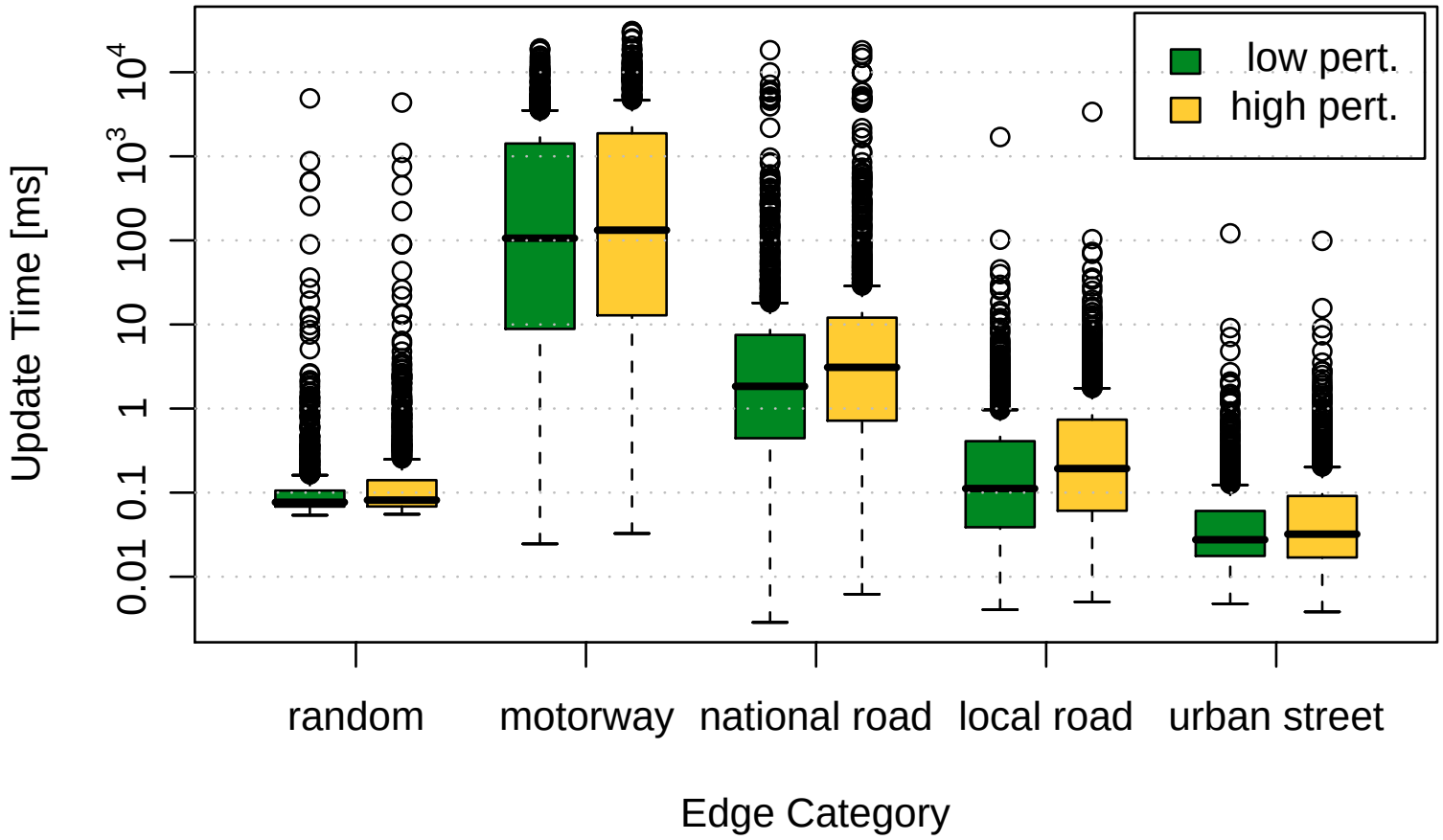


Beispiel



Aktualisierung der Vorberechnung

Aktualisieren von 32 kürzeste-Wege Bäumen für 16 Landmarken:



No-update Variante: Typ des Updates

- Zufallsanfragen nach 1 000 updates
- erhöhe Kantengewicht um x2 (x10 in Klammern)
- verschiedene Straßenkategorien

road type	affected queries	16 landmarks search space		32 landmarks search space	
no updates	0.0 %	74 699	(74 699)	40 945	(40 945)
all	7.5 %	74 700	(77 759)	41 044	(43 919)
urban	0.8 %	74 796	(74 859)	40 996	(41 120)
local	1.5 %	74 659	(74 669)	40 949	(40 995)
national	28.1 %	74 920	(75 777)	41 251	(42 279)
motorway	95.3 %	97 249	(265 472)	59 550	(224 268)

Beobachtung:

- nur Autobahnen haben Einfluß auf Suchraumgröße

No-update Variante: Anzahl Updates

- Zufallsanfragen nach Autobahn updates (x2, x10 in Klammern)
- verschiedene Anzahl Updates

updates	affected queries	16 landmarks search space		32 landmarks search space	
0	0.0 %	74 699	(74 699)	40 945	(40 945)
100	39.9 %	75 691	(91 610)	41 725	(56 349)
200	64.7 %	78 533	(107 084)	44 220	(69 906)
500	87.1 %	86 284	(165 022)	50 007	(124 712)
1 000	95.3 %	97 249	(265 472)	59 550	(224 268)
2 000	97.8 %	154 112	(572 961)	115 111	(531 801)
5 000	99.1 %	320 624	(1 286 317)	279 758	(1 247 628)
10 000	99.5 %	595 740	(2 048 455)	553 590	(1 991 297)

Beobachtung:

- $\leq$ 1000 gut verkraftbar

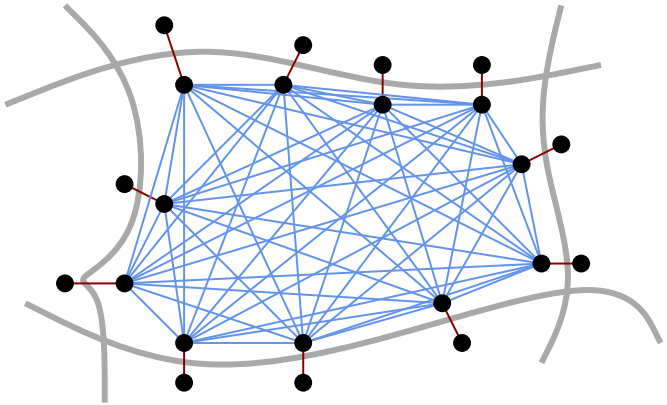
Customizable Route Planning

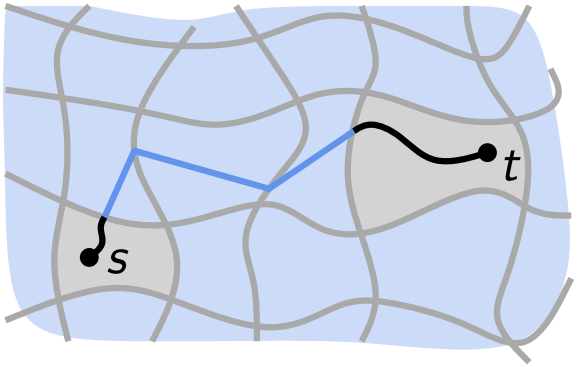
Idee:

- 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

Update der Vorberechnung

Beobachtung:

- Shortcuts (Cliquenkanten) können invalidiert werden
- Shortcuts repräsentieren Pfade innerhalb der Zelle

Update:

- identifiziere Zelle in der die aktualisierte Kante liegt für jedes Level der Multi-Level Partition
- wiederhole (bottom-up) Preprocessing für diese Zellen
- dauert weniger als 10 ms

Mobiles Szenario

Beobachtung:

- Update invalidiert Zellen-Overlay
- erhöht meistens nur einige Overlay-Gewichte
- restliche Overlay-Kanten bleiben korrekt

Query:

- Suche muss bei invalidierten Zellen absteigen
- jeder Abstieg erhöht Suchraum um ca. 50%

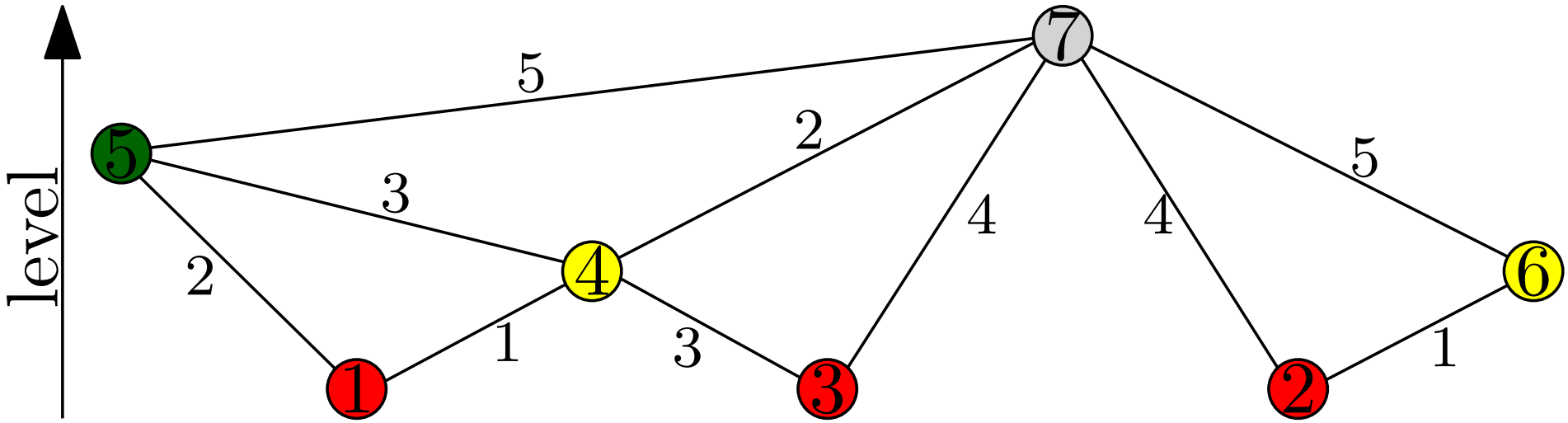
Optimierung:

- markiere invalidierte Zellen (ohne Overlay zu aktualisieren)
- führe Anfrage aus (ohne Abstieg)
- wenn der Pfad nicht die aktualiesierte Kante enhält $\Rightarrow$ fertig
- sonst Suche mit Abstieg

Contraction Hierarchies

preprocessing:

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



Update der Vorberechnung I

Beobachtung:

- Knotenordnung ist Heuristik
- also behalte Ordnung bei
- geänderte Kante (u, v) kann zu Shortcuts beitragen
- aber auch Teil von Zeugensuchen gewesen sein
- Frage: welche Knoten müssen neu kontrahiert werden (mit Zeugensuche)

Update der Vorberechnung II

Idee:

- wiederhole Kontraktion für Knotenmenge U
- füge alle aufwärts erreichbaren Knoten von u, v zu U
- speicher für jede Kante e auf welchen Zeugensuchen sie benutzt wurde. Die zugehörigen Knoten speicher in A_e. Kann einfach während Vorberechnung gespeichert werden.
- füge alle aufwärts erreichbaren Knoten von A_e (und aller A_f für alle Shortcuts f, die e enthalten) zu U
- wiederhole Kontraktion für alle Knoten U bezüglich Originalordnung

Beobachtung:

- erhöht Speicherverbrauch von CH deutlich
- Updatezeit abhängig von Art der Kante

Mobiles Szenario

Beobachtung:

- Kontraktion von Knoten zu teuer

Mobiles Szenario

Beobachtung:

- Kontraktion von Knoten zu teuer

Idee:

- identiziere Knotenmenge U wie zuvor
- erlaube Abstieg an Kanten (u, v) mit $u, v \in U$
- iterativer Ansatz (wie bei CRP) auch hier möglich

Weitere Techniken

Arc-Flags:

- Aktualisierung aller Bäume nötig
- momentan keine vernünftige Update-Routine vorhanden

Transit-Node Routing

- schwierig
- Tabellen und Access-Nodes können sich ändern

HubLabels

- halte CH vor
- nach Update: Aktualisierung aller Labels
- nicht getestet: nicht alle Labels müssen aktualisiert werden

Zusammenfassung

- Stau erhöht Kantengewicht temporär
- Anpassung Landmarken, CRP trivial
- Anpassung CH machbar
- sehr schwierig bei anderen Techniken

Anmerkungen

- Mobil: nur “nahe” Staus beachten?
- viele Staus vorhersehbar

Szenario

Routenplanung auf Mobilen Geräten

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Szenario

Routenplanung auf Mobilen Geräten

2 Varianten:

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Szenario

Routenplanung auf Mobilen Geräten

2 Varianten:

- server-basiert
  - Gerät sendet request und kriegt komplette Route übertragen
  - mit aktuellen Stauinformationen
  - Was, wenn der Fahrer sich verfährt?
  - Was passiert in Funklöchern

Szenario

Routenplanung auf Mobilen Geräten

2 Varianten:

- server-basiert
  - Gerät sendet request und kriegt komplette Route übertragen
  - mit aktuellen Stauinformationen
  - Was, wenn der Fahrer sich verfährt?
  - Was passiert in Funklöchern
- direkt auf Gerät
  - wie Kartendaten aktualisieren?
  - Staus?
  - limitierte Rechenpower

Ein Kompromiss

Idee:

- weiterhin server-basiert
- während ursprünglicher Anfrage haben wir Zugang zum Server
- aber sende mehr als die Route
- (Visualisierungsdaten auf dem Gerät)

Ein Kompromiss

Idee:

- weiterhin server-basiert
- während ursprünglicher Anfrage haben wir Zugang zum Server
- aber sende mehr als die Route
- (Visualisierungsdaten auf dem Gerät)

Herausforderung:

- Tradeoff zwischen Robustheit und Menge an Information, die wir senden müssen
- Tradeoff zwischen zwei Extrema
  - kürzesten Weg (wenig robust, klein)
  - voller kürzeste Wege Baum T (sehr robust, aber zuviele Daten)

Ein Kompromiss

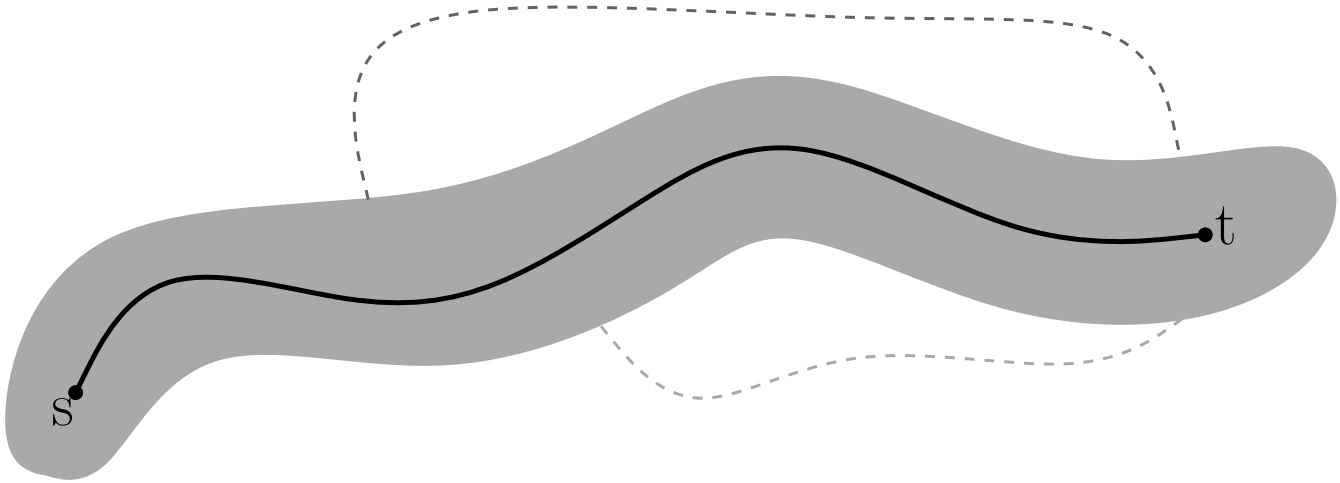
Idee:

- weiterhin server-basiert
- während ursprünglicher Anfrage haben wir Zugang zum Server
- aber sende mehr als die Route
- (Visualisierungsdaten auf dem Gerät)

Herausforderung:

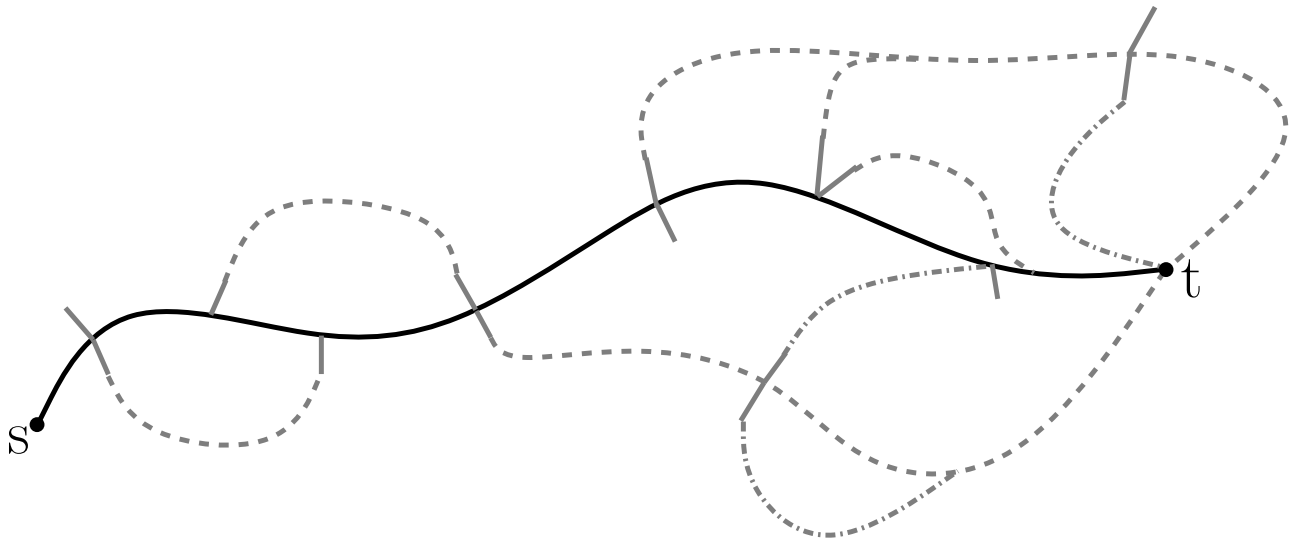
- Tradeoff zwischen Robustheit und Menge an Information, die wir senden müssen
- Tradeoff zwischen zwei Extrema
  - kürzesten Weg (wenig robust, klein)
  - voller kürzeste Wege Baum T (sehr robust, aber zuviele Daten)
- finde “sinnvollen” Teil von T?

Perimeter-Korridor



- wähle maximale Abweichzeit d
- berechne alle Knoten N, die maximal d weit weg vom kürzesten Weg sind
- Korridor: alle Knoten, die auf den kürzesten Wegen von N nach t liegen

Abbiege-Korridor



- wähle maximale Abweichen k
- starte von kürzestem Weg und füge ihn zu Korridor C hinzu
- wiederhole k mal
  - bestimme Nachbarknoten D zu C
  - füge alle kürzesten Wege von D zu t zu C hinzu

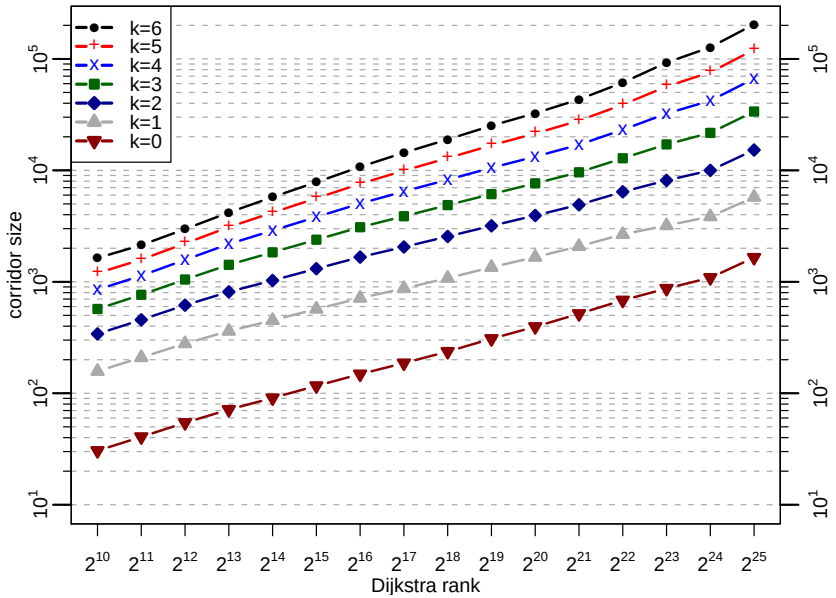
Evaluation

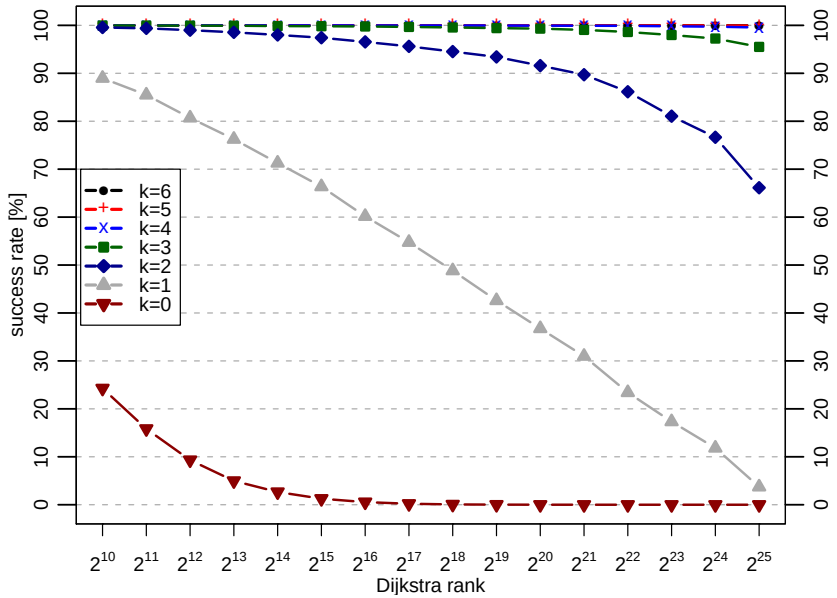
Setup:

- Europa
- Zufallsfahrer DD(x): macht an jeder Kreuzung eine falsche Abbiegung mit Wahrscheinlichkeit x

turn corridor			perimeter corridor		
	size	success [%]	τ	size	success [%]
k	$\lvert V \rvert$	DD(5%)	[s]	$\lvert V \rvert$	DD(5%)
0	1 351	0.0	0	1 351	0.0
1	4 835	10.3	10	6 223	7.0
2	12 204	73.2	20	7 689	8.7
3	25 892	96.8	30	9 570	10.6
4	50 271	99.7	50	14 284	15.0
5	88 742	100.0	100	31 547	28.2
6	148 370	100.0	300	209 513	79.6

Ergebnisse

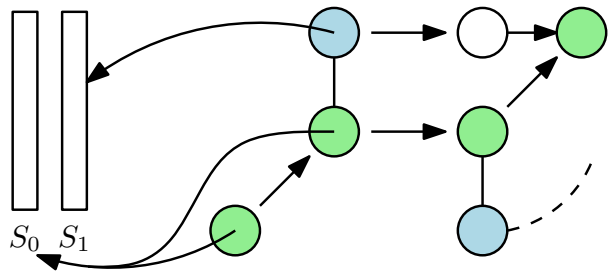




Beobachtungen

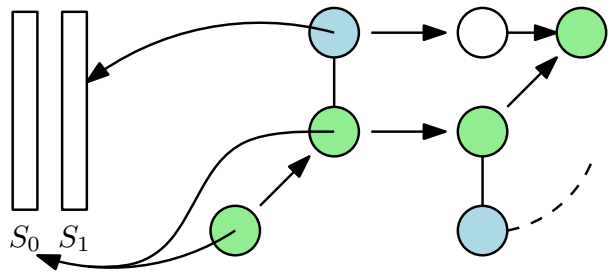
- Korridorgrösse verdoppelt sich nach jeder Iteration
- nahe 100% Erfolgsquote für $k = 3$

Generelles Vorgehen



- wir bauen k-turn Korridor C auf
- verwalte zwei Stacks (S_0, S_1)
- S_0 mit s initialisiert, S_1 leer
- führe $k + 1$ Iterationen durch
  - bearbeite alle Knoten in S_0:
    - nimm u vom Stack und füge u zu C hinzu
    - scanne alle ausgehenden Kanten (u, v) mit $v \neq C$
    - wenn v auf dem kürzesten u–t Weg, füge v zu S_0
    - sonst zu S_1
  - $S_0 = S_1$, $S_1 = \emptyset$

Generelles Vorgehen



- wir bauen k-turn Korridor C auf
- verwalte zwei Stacks (S_0, S_1)
- S_0 mit s initialisiert, S_1 leer
- führe $k + 1$ Iterationen durch
  - bearbeite alle Knoten in S_0:
    - nimm u vom Stack und füge u zu C hinzu
    - scanne alle ausgehenden Kanten (u, v) mit $v \neq C$
    - wenn v auf dem kürzesten u–t Weg, füge v zu S_0
    - sonst zu S_1
  - $S_0 = S_1$, $S_1 = \emptyset$

Beobachtungen:

- am Ende der Iteration i ist C der $i - 1$-turn Korridor
- und S_1 enthält die Nachbarknoten zu C

Berechnunsansätze

- benutze Dijkstra zum Bestimmen aller Distanzen

Berechnunsansätze

- benutze Dijkstra zum Bestimmen aller Distanzen
- PHAST

Berechnunsansätze

- benutze Dijkstra zum Bestimmen aller Distanzen
- PHAST
- benutze Punkt-zu-Punkt Anfragen um die kürzesten Wege von S_0 zu t am Anfang jeder Iteration zu bestimmen

Berechnunsansätze

- benutze Dijkstra zum Bestimmen aller Distanzen
- PHAST
- benutze Punkt-zu-Punkt Anfragen um die kürzesten Wege von S_0 zu t am Anfang jeder Iteration zu bestimmen
- benutze One-To-Many Techniken zur Beschleunigung
  - Buckets
  - RPHAST Variante (TCC)

Ergebnisse

Setup:

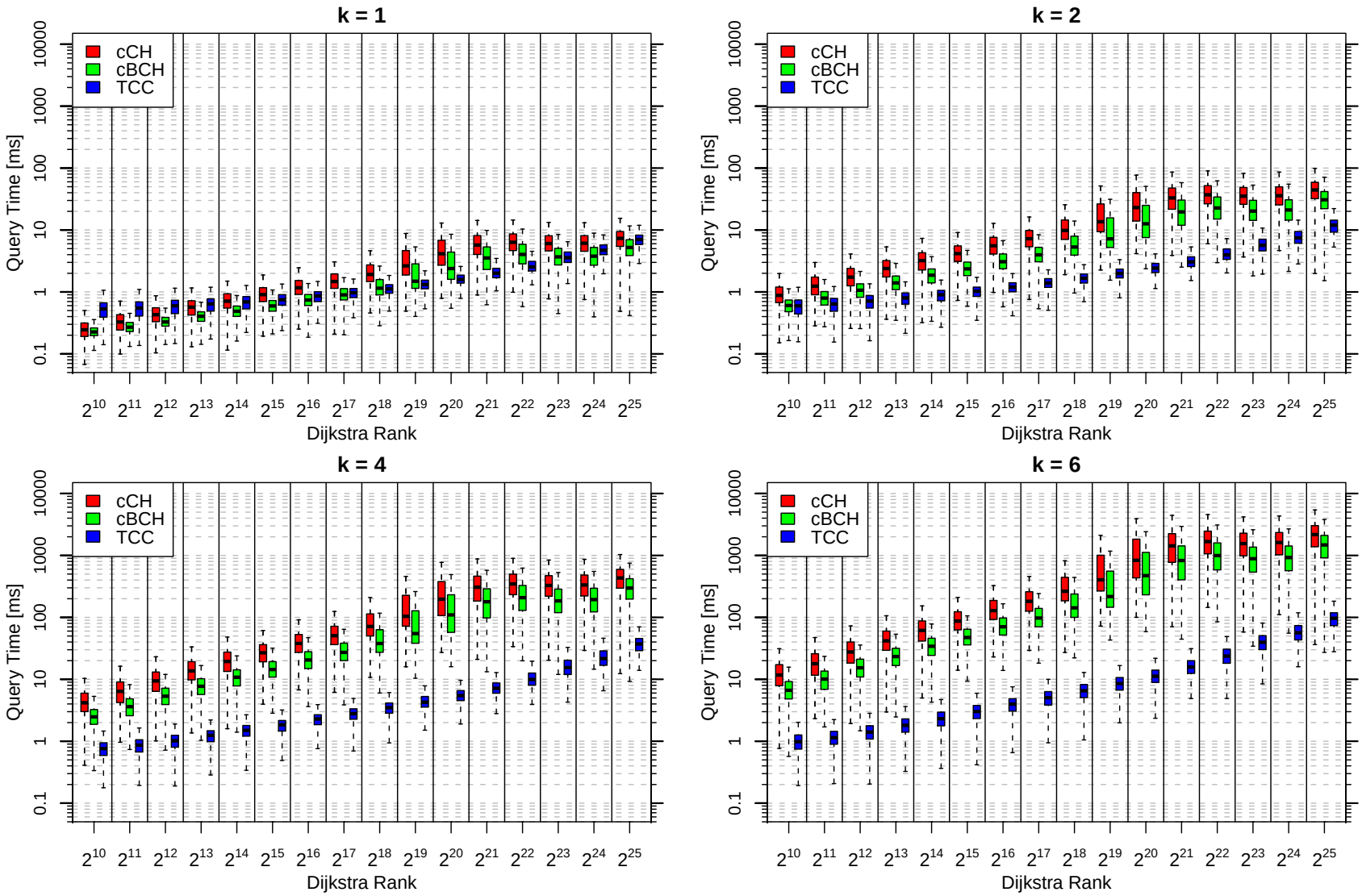
- Europa inkl. Turn Costs
- sequentielle Berechnung

k	cCH	cBCH	cPHAST	TCC
0	0.33	0.34	968.42	0.73
1	7.35	5.26	969.32	5.67
2	44.45	30.96	970.36	9.73
3	156.95	100.81	973.36	16.26
4	382.51	263.00	974.14	27.34
5	795.26	545.97	977.30	42.54
6	1643.34	1132.72	982.83	68.66

Beobachtung:

- TCC macht Berechnung schnell genug

Ergebnisse II



Zusammenfassung

- Korridor erlauben robuste mobile Routenplanung
- flexibler Tradeoff zwischen Robustheit und Datenmenge
- können effizient berechnet werden

Nächste Termine

Karlsruher Institut für Technologie

Mittwoch, 11.6.2014

(Auf englisch)

Mittwoch, 18.6.2014
(Auf englisch)

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