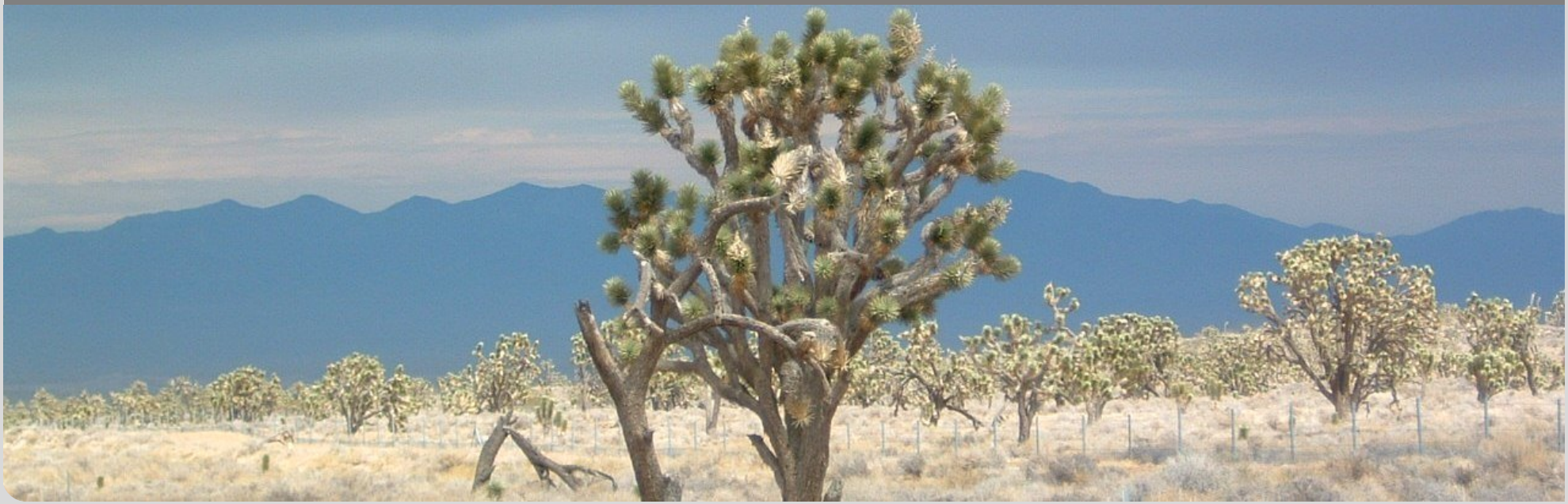
Karlsruher Institut für Technologie

Algorithmen für Routenplanung

10. Vorlesung, Sommersemester 2019

Tobias Zündorf | 8. Juni 2020

Institut für Theoretische Informatik · Algorithmik



Kürzeste Wege in Straßennetzwerken

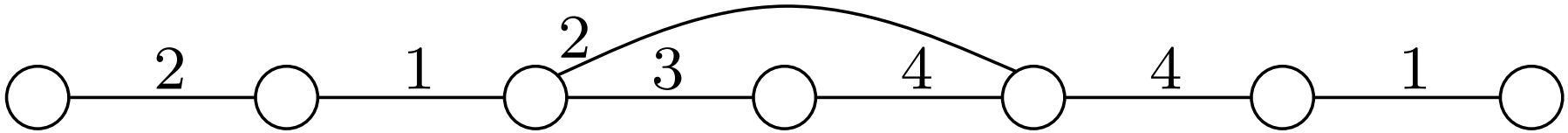
Beschleunigungstechniken (Fortsetzung)

- Hub Labeling (HL)
- Transit-Node Routing (TNR)

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Wiederholung: CH

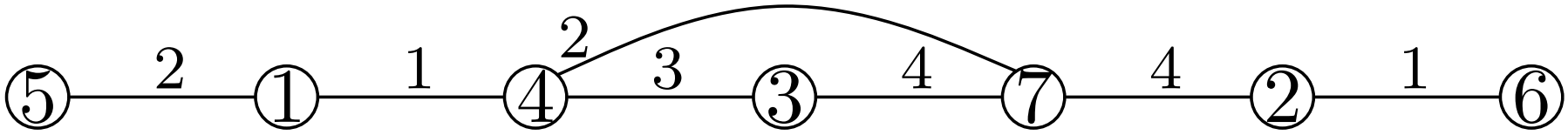
Preprocessing:



Wiederholung: CH

Preprocessing:

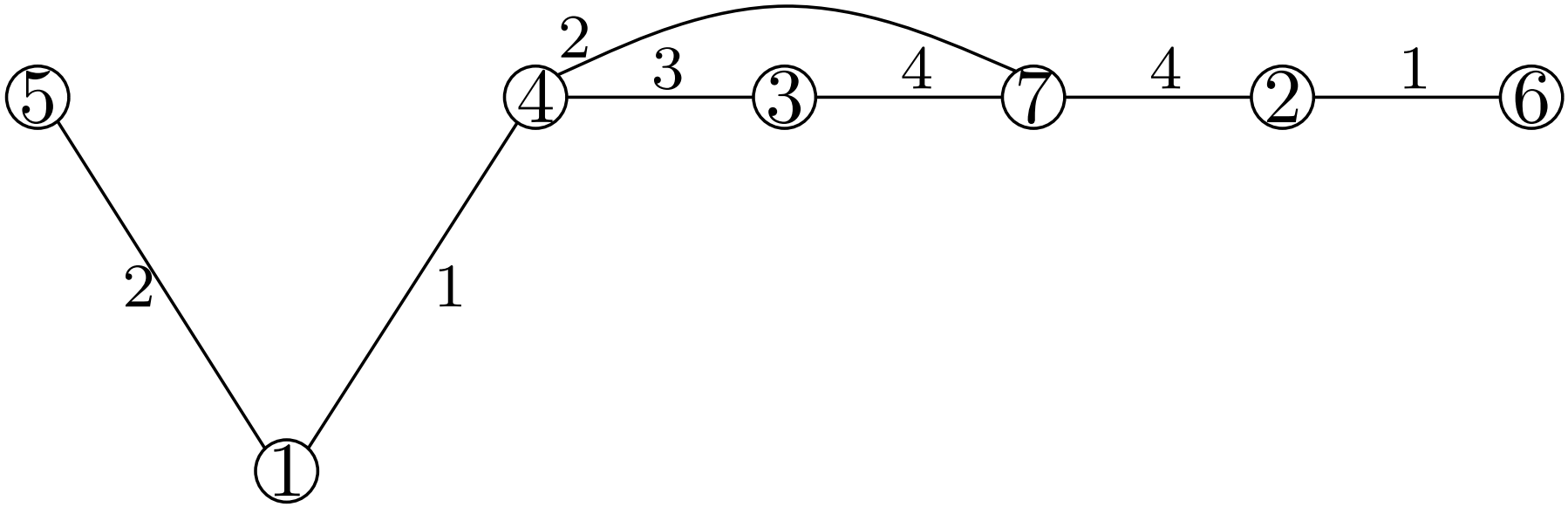
- ordne Knoten nach Wichtigkeit



Wiederholung: CH

Preprocessing:

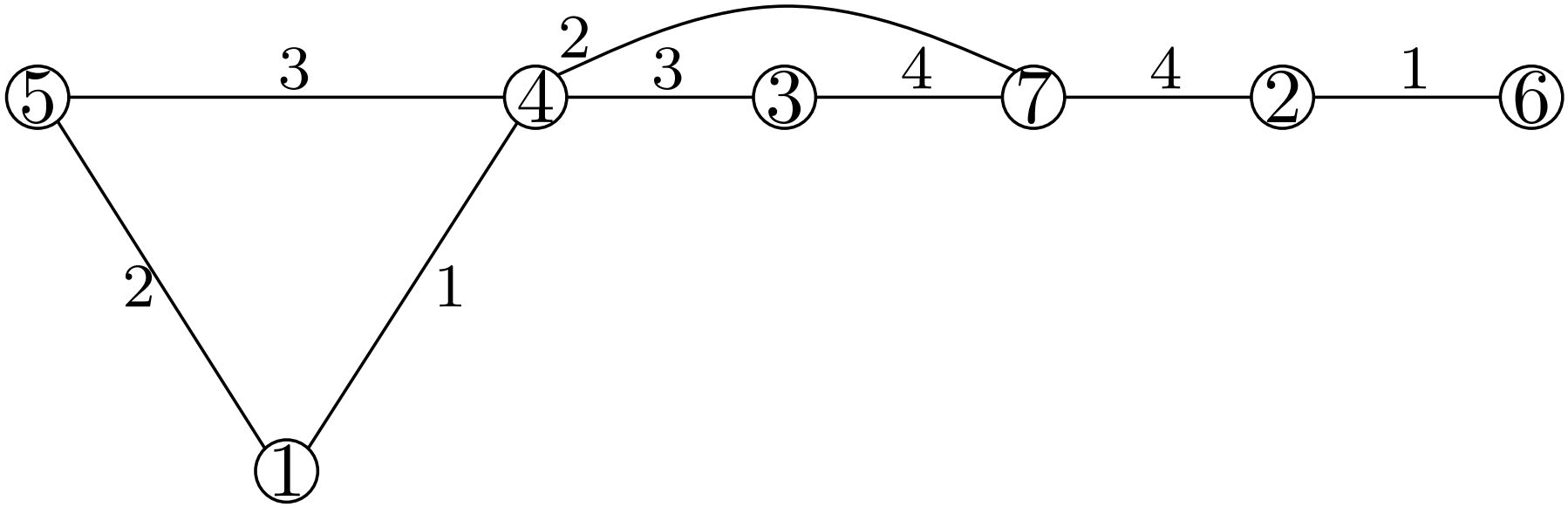
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu



Wiederholung: CH

Preprocessing:

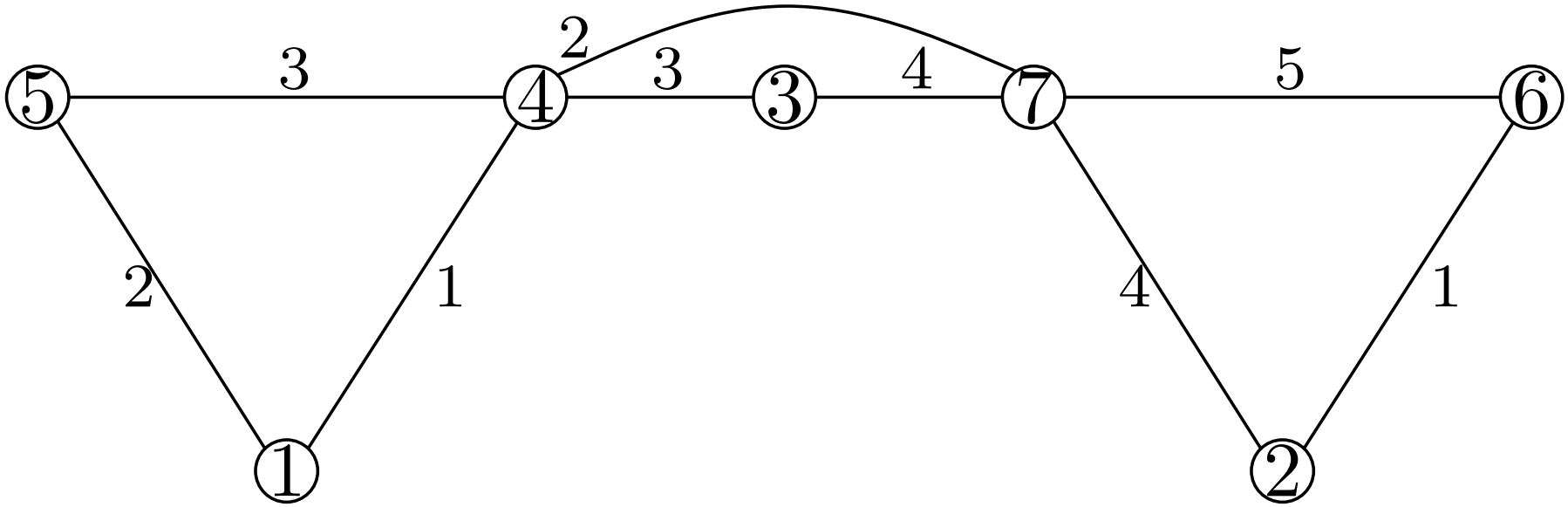
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu



Wiederholung: CH

Preprocessing:

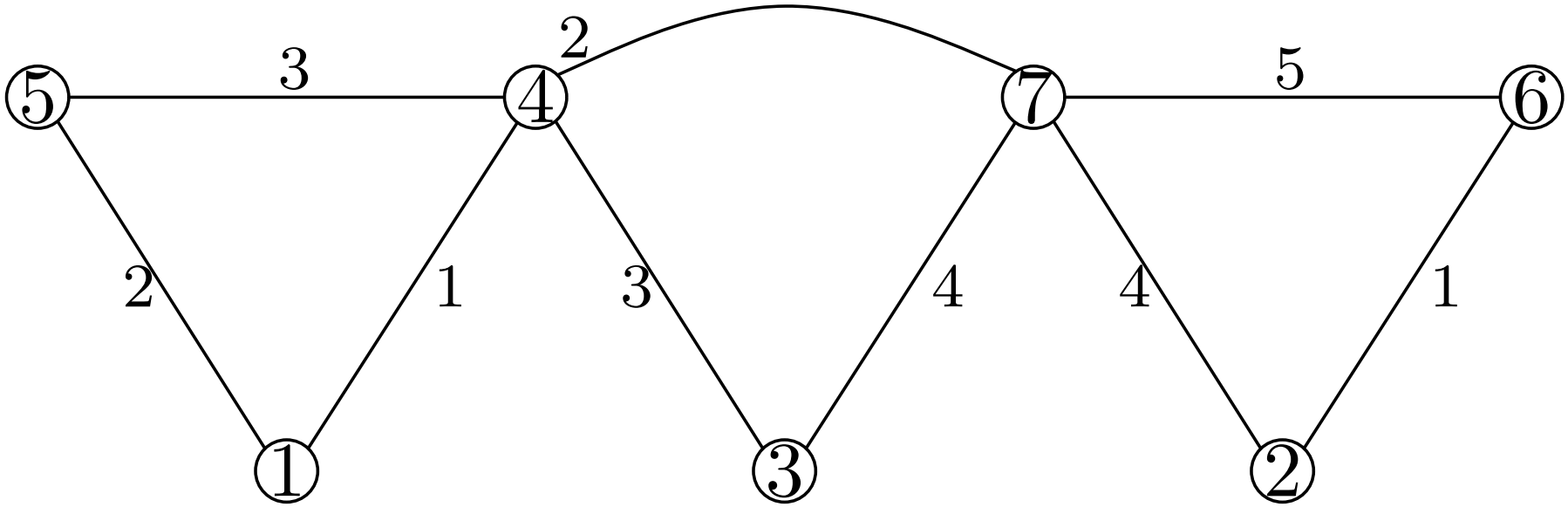
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu



Wiederholung: CH

Preprocessing:

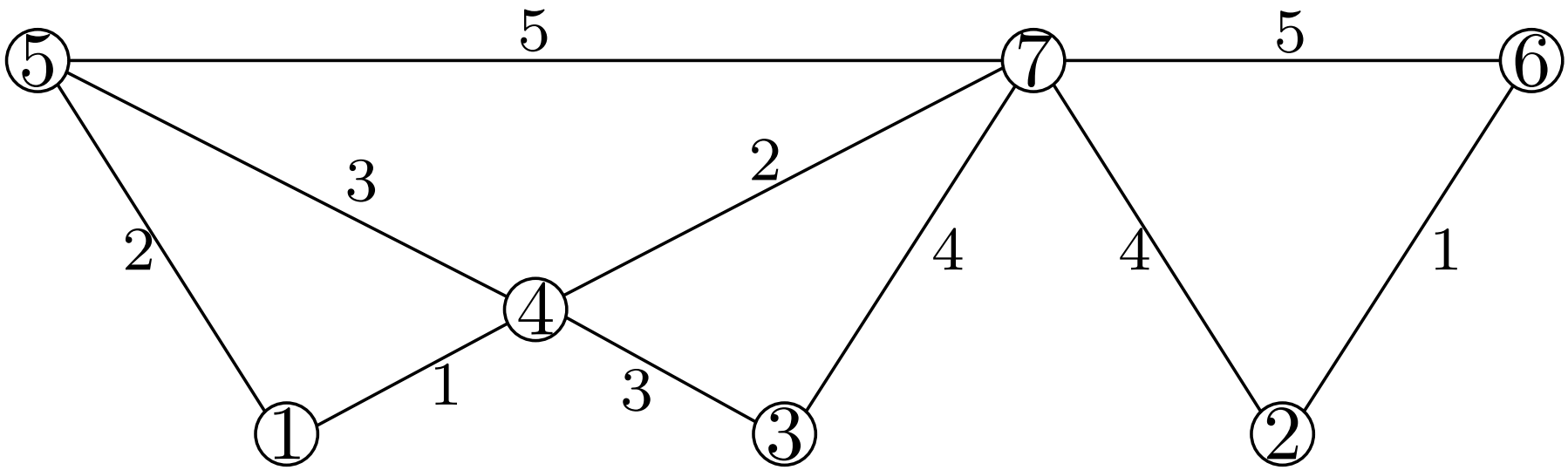
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu



Wiederholung: CH

Preprocessing:

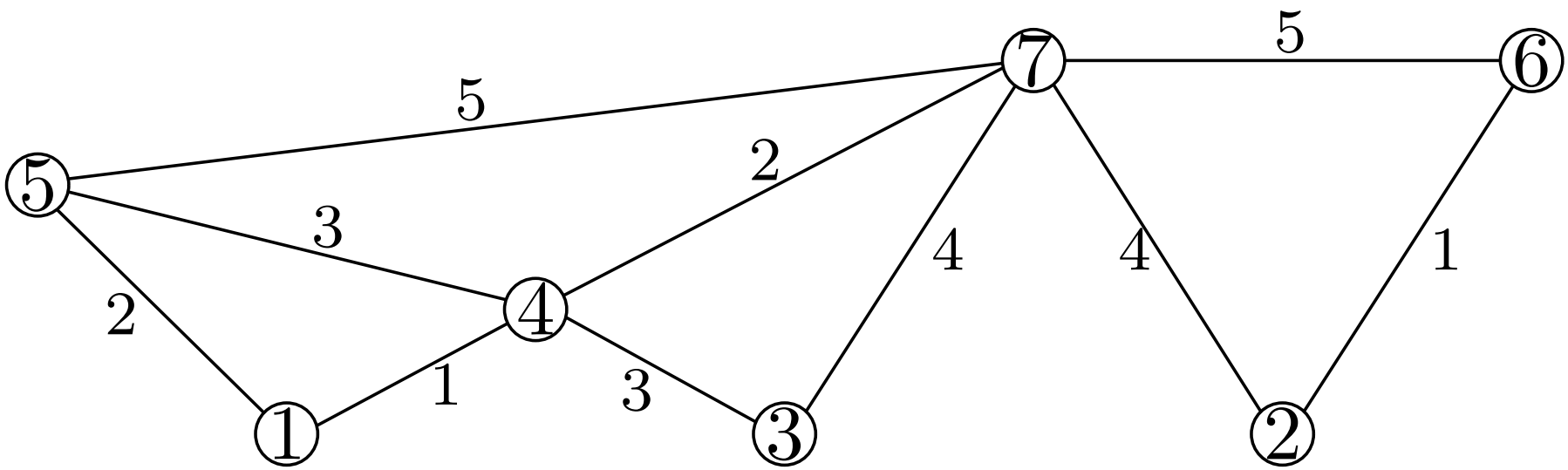
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu



Wiederholung: CH

Preprocessing:

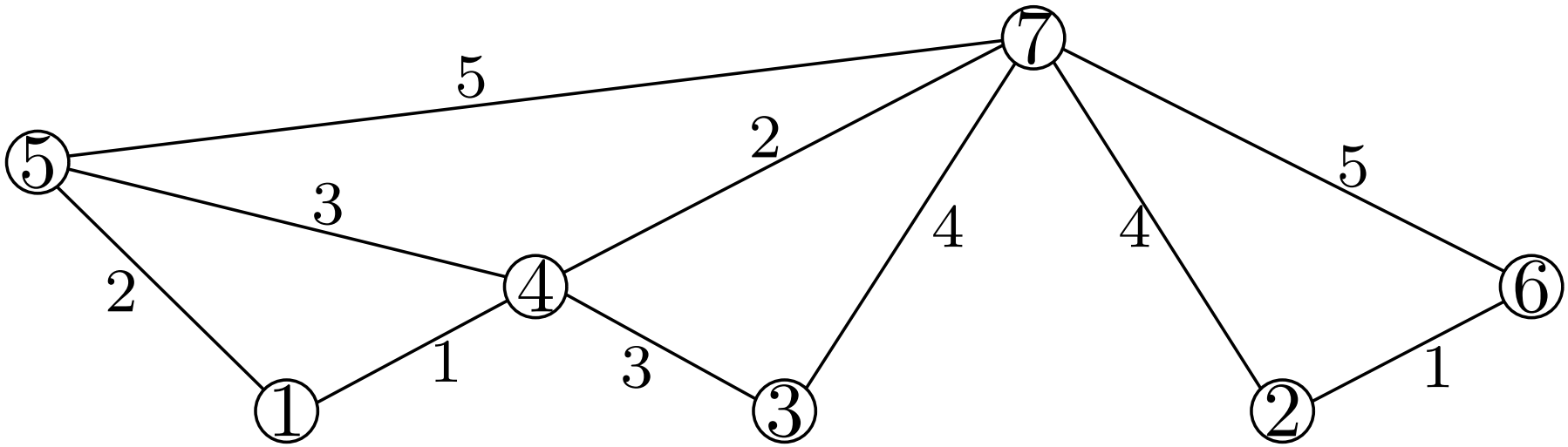
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu



Wiederholung: CH

Preprocessing:

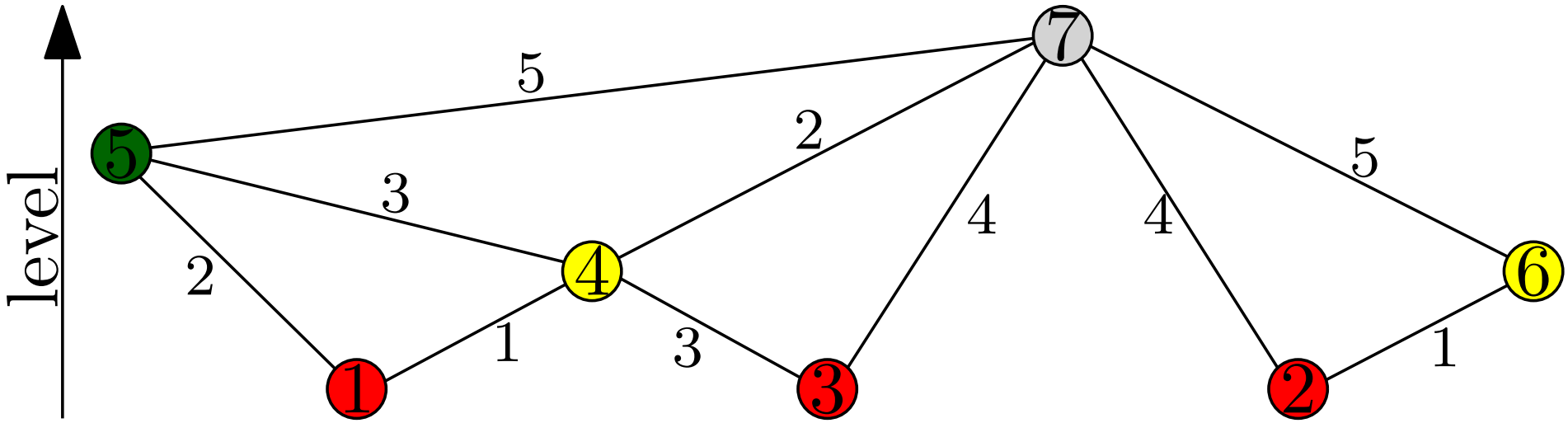
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu



Wiederholung: CH

Preprocessing:

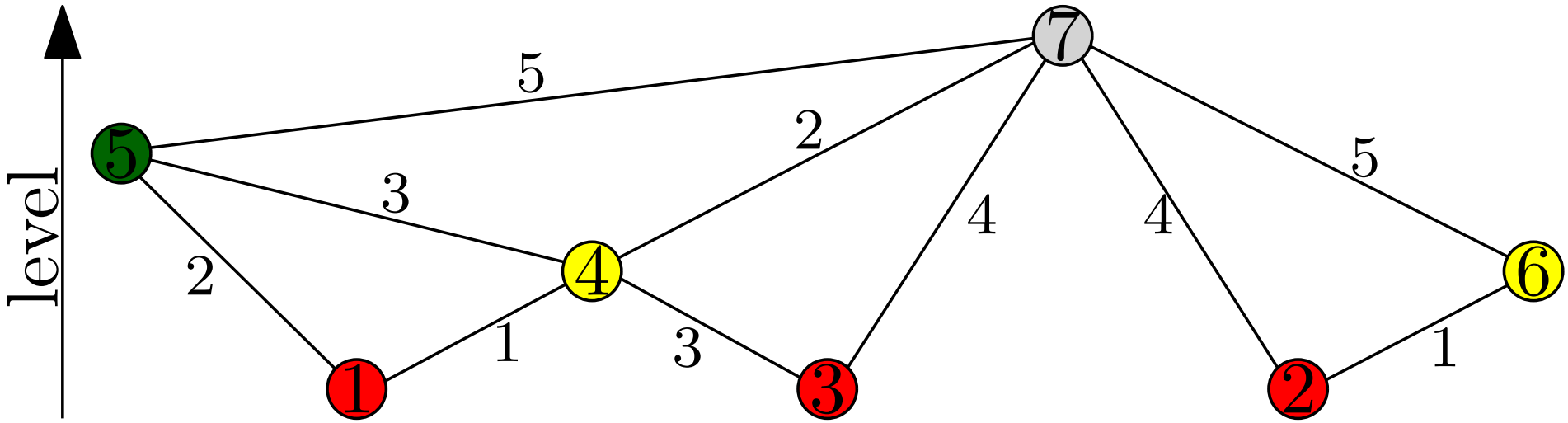
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu
- Levelzuordnung



Wiederholung: CH

Punkt-zu-Punkt Anfragen:

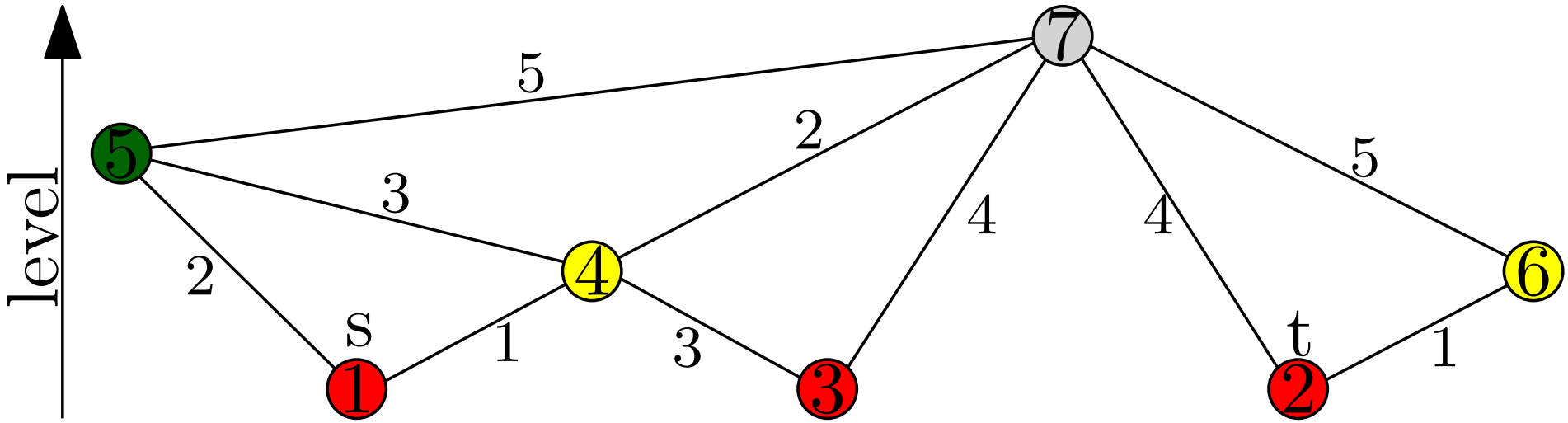
- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten



Wiederholung: CH

Punkt-zu-Punkt Anfragen:

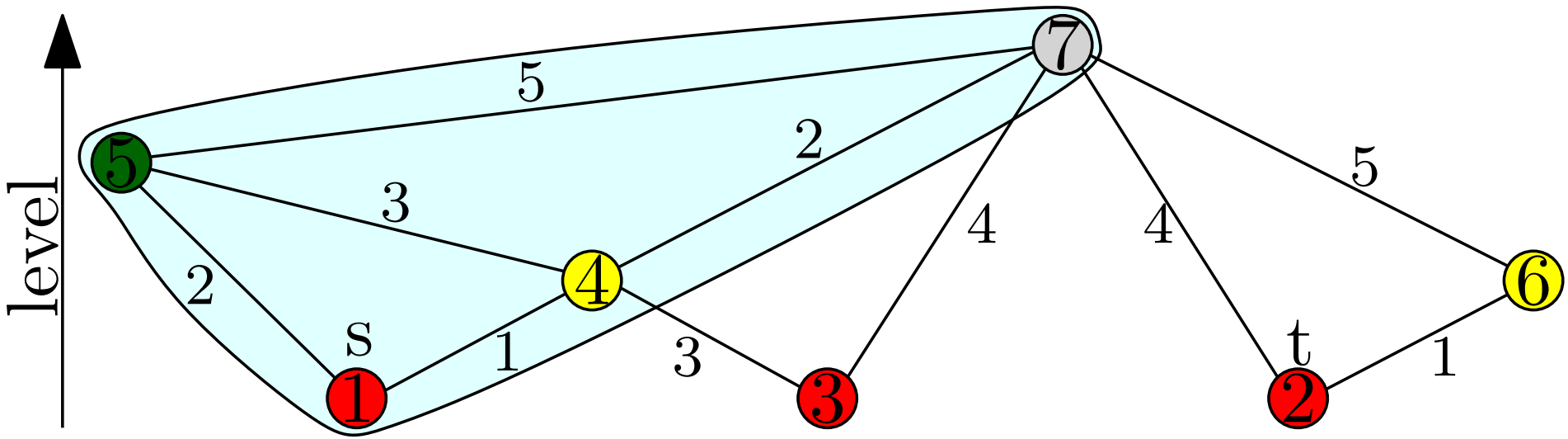
- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten



Wiederholung: CH

Punkt-zu-Punkt Anfragen:

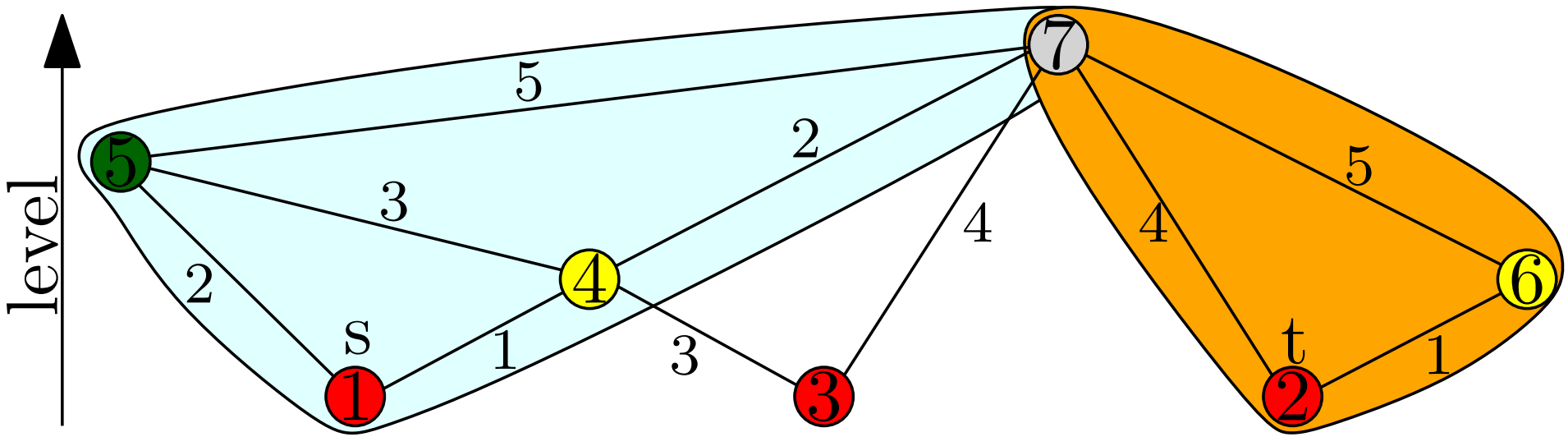
- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten



Wiederholung: CH

Punkt-zu-Punkt Anfragen:

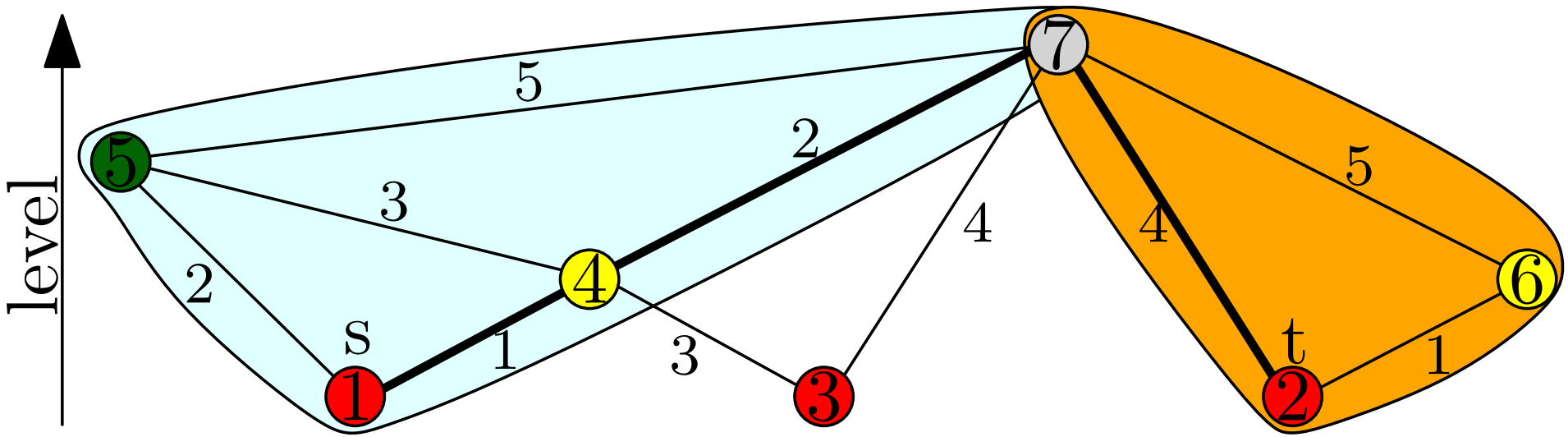
- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten



Wiederholung: CH

Punkt-zu-Punkt Anfragen:

- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten



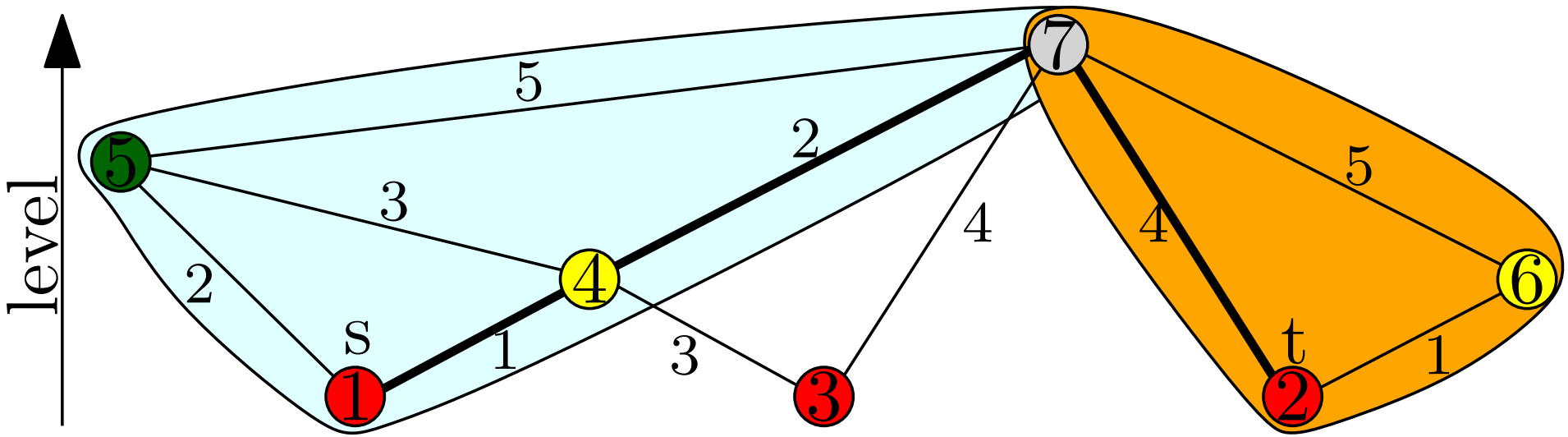
Wiederholung: CH

Punkt-zu-Punkt Anfragen:

- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten

Korrektheit:

- es gibt einen wichtigsten Knoten auf dem Pfad
- dieser wird von Vorwärts- und Rückwärtssuche gescannt



HubLabels

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Hublabels

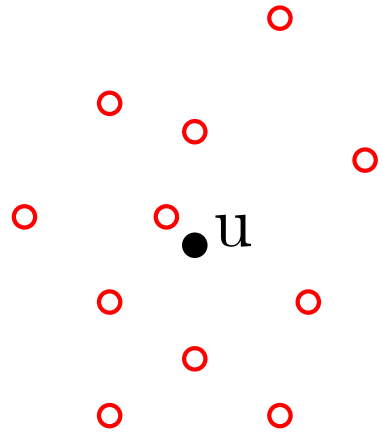
Vorberechung:

- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$

Hublabels

Vorberechung:

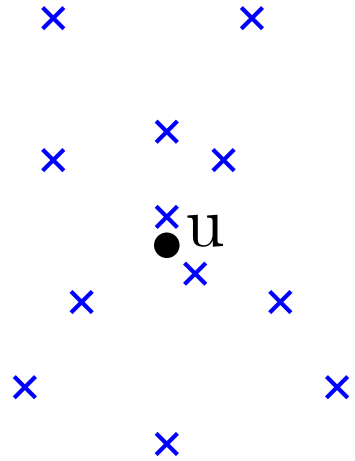
- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
  - dist(u, v) für jeden Hub $v \in L_f(u)$



Hublabels

Vorberechung:

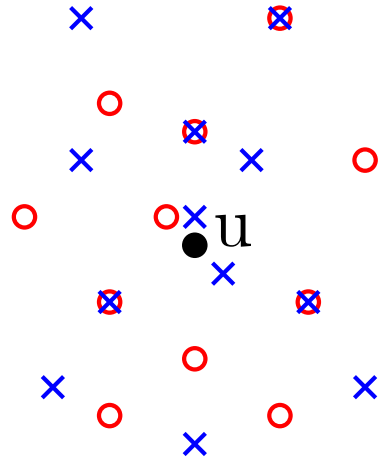
- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
  - dist(u, v) für jeden Hub $v \in L_f(u)$
  - dist(v, u) für jeden Hub $v \in L_b(u)$



Hublabels

Vorberechung:

- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
  - dist(u, v) für jeden Hub $v \in L_f(u)$
  - dist(v, u) für jeden Hub $v \in L_b(u)$



Hublabels

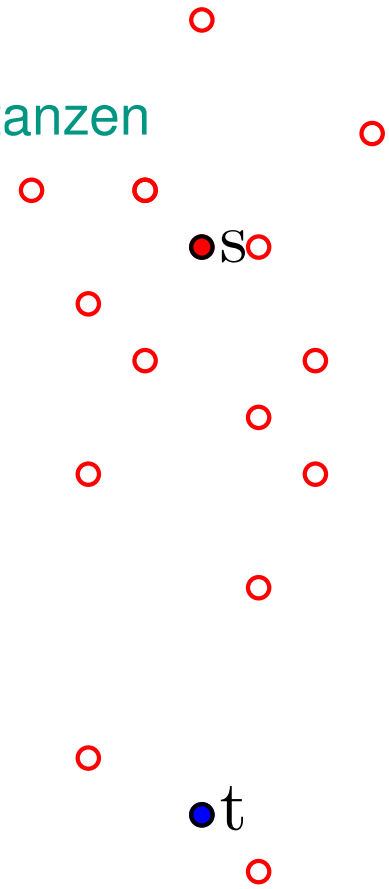
Vorberechung:

- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
  - dist(u, v) für jeden Hub $v \in L_f(u)$
  - dist(v, u) für jeden Hub $v \in L_b(u)$
- die Label müssen die cover property einhalten:
  $\forall s, t$, $L_f(s) \cap L_b(t)$ überdeckt den kürzesten s–t Pfad

Hublabels

Vorberechung:

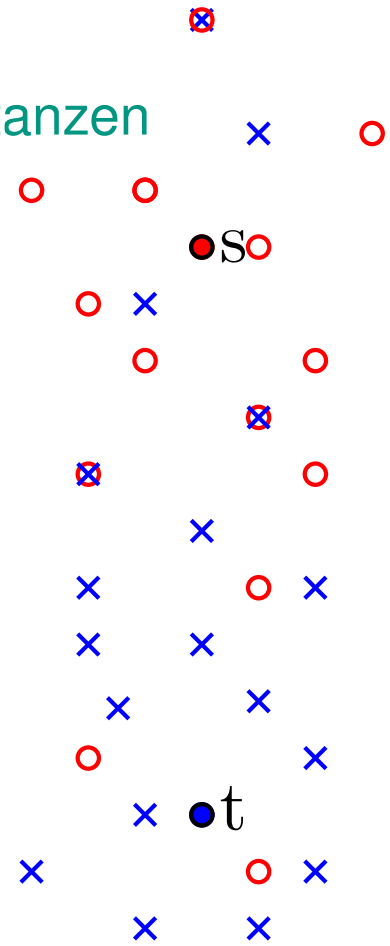
- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
  - dist(u, v) für jeden Hub $v \in L_f(u)$
  - dist(v, u) für jeden Hub $v \in L_b(u)$
- die Label müssen die cover property einhalten:
  $\forall s, t$, $L_f(s) \cap L_b(t)$ überdeckt den kürzesten s–t Pfad



Hublabels

Vorberechung:

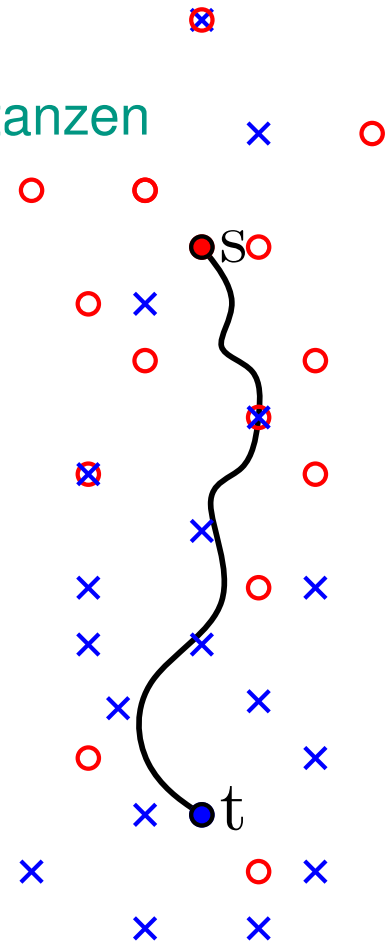
- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
  - dist(u, v) für jeden Hub $v \in L_f(u)$
  - dist(v, u) für jeden Hub $v \in L_b(u)$
- die Label müssen die cover property einhalten:
  $\forall s, t$, $L_f(s) \cap L_b(t)$ überdeckt den kürzesten s–t Pfad



Hublabels

Vorberechung:

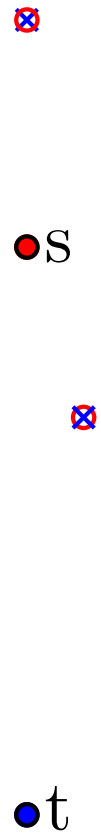
- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
  - dist(u, v) für jeden Hub $v \in L_f(u)$
  - dist(v, u) für jeden Hub $v \in L_b(u)$
- die Label müssen die cover property einhalten:
  $\forall s, t$, $L_f(s) \cap L_b(t)$ überdeckt den kürzesten s–t Pfad



Hublabels

Vorberechung:

- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
  - dist(u, v) für jeden Hub $v \in L_f(u)$
  - dist(v, u) für jeden Hub $v \in L_b(u)$
- die Label müssen die cover property einhalten:
  $\forall s, t$, $L_f(s) \cap L_b(t)$ überdeckt den kürzesten s–t Pfad



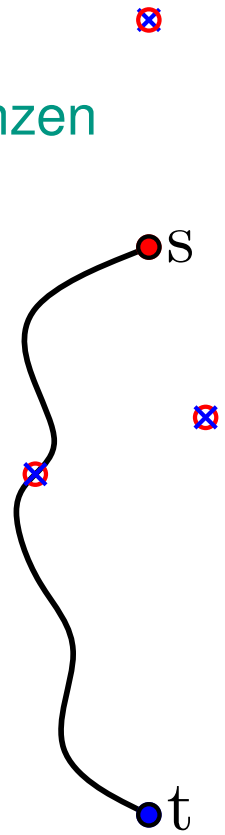
s–t Anfrage:

- finde Knoten $v \in L_f(s) \cap L_b(t)$ ...

Hublabels

Vorberechung:

- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
  - dist(u, v) für jeden Hub $v \in L_f(u)$
  - dist(v, u) für jeden Hub $v \in L_b(u)$
- die Label müssen die cover property einhalten:
  $\forall s, t$, $L_f(s) \cap L_b(t)$ überdeckt den kürzesten s–t Pfad



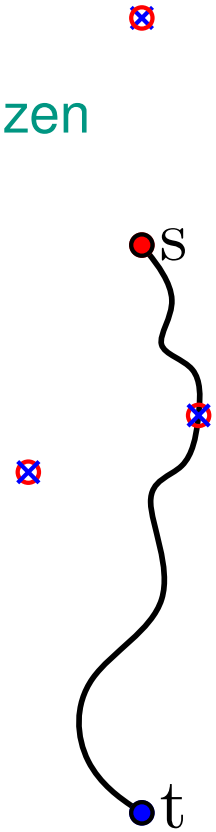
s–t Anfrage:

- finde Knoten $v \in L_f(s) \cap L_b(t)$ ...
- ... der dist(s, v)+ dist(v, t) minimiert

Hublabels

Vorberechung:

- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
  - dist(u, v) für jeden Hub $v \in L_f(u)$
  - dist(v, u) für jeden Hub $v \in L_b(u)$
- die Label müssen die cover property einhalten:
  $\forall s, t$, $L_f(s) \cap L_b(t)$ überdeckt den kürzesten s–t Pfad



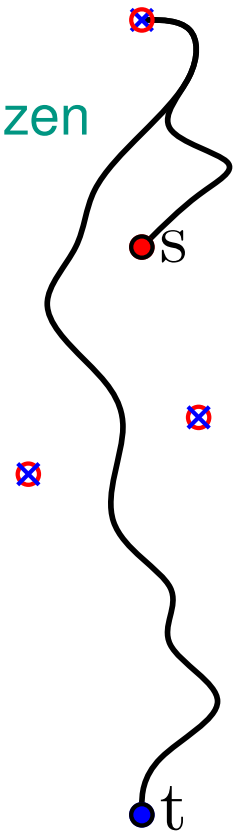
s–t Anfrage:

- finde Knoten $v \in L_f(s) \cap L_b(t)$ ...
- ... der dist(s, v)+ dist(v, t) minimiert

Hublabels

Vorberechung:

- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
  - dist(u, v) für jeden Hub $v \in L_f(u)$
  - dist(v, u) für jeden Hub $v \in L_b(u)$
- die Label müssen die cover property einhalten:
  $\forall s, t$, $L_f(s) \cap L_b(t)$ überdeckt den kürzesten s–t Pfad



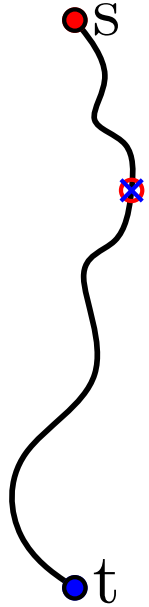
s–t Anfrage:

- finde Knoten $v \in L_f(s) \cap L_b(t)$ ...
- ... der dist(s, v)+ dist(v, t) minimiert

Hublabels

Vorberechung:

- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
  - dist(u, v) für jeden Hub $v \in L_f(u)$
  - dist(v, u) für jeden Hub $v \in L_b(u)$
- die Label müssen die cover property einhalten:
  $\forall s, t$, $L_f(s) \cap L_b(t)$ überdeckt den kürzesten s–t Pfad



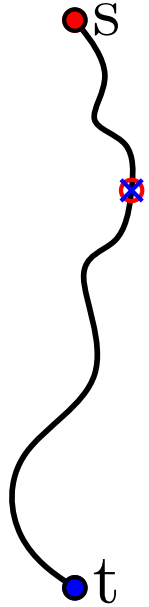
s–t Anfrage:

- finde Knoten $v \in L_f(s) \cap L_b(t)$ ...
- ... der dist(s, v)+ dist(v, t) minimiert

Hublabels

Vorberechung:

- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
  - dist(u, v) für jeden Hub $v \in L_f(u)$
  - dist(v, u) für jeden Hub $v \in L_b(u)$
- die Label müssen die cover property einhalten:
  $\forall s, t$, $L_f(s) \cap L_b(t)$ überdeckt den kürzesten s–t Pfad



s–t Anfrage:

- finde Knoten $v \in L_f(s) \cap L_b(t)$ ...
- ... der dist(s, v)+ dist(v, t) minimiert

Beobachtungen:

- Laufzeit hängt von Labelgröße ab
- wie effizient berechnen?

Hublabels

Speichern der Labels:

- als Menge von Hub, Distanz Paaren
- sortiert nach Hub-Knoten-ID

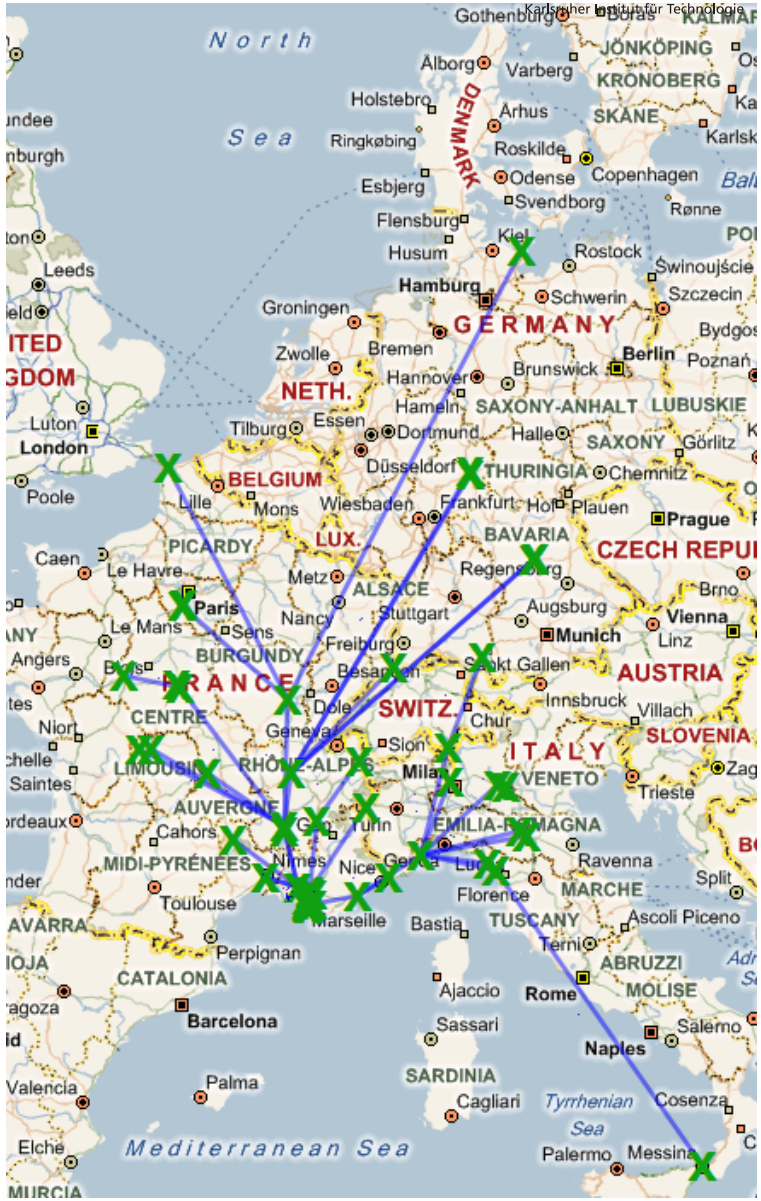
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Hublabels

Speichern der Labels:

- als Menge von | Hub, Distanz | Paaren
- sortiert nach Hub-Knoten-ID

$L_f(s)$ | 1,0 | 4,1 | 5,2 | 7,3 |



Hublabels

Speichern der Labels:

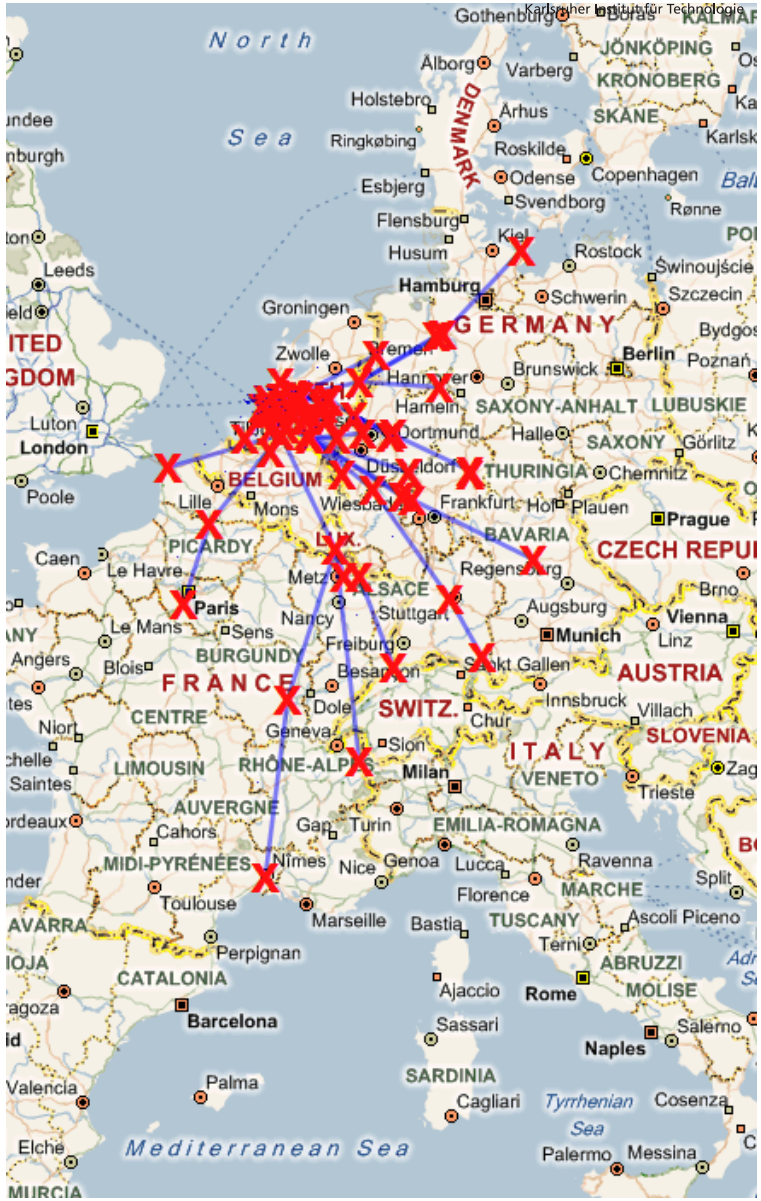
- als Menge von | Hub, Distanz | Paaren
- sortiert nach Hub-Knoten-ID

$L_f(s)$

1,0	4,1	5,2	7,3

$L_b(t)$

2,0	6,1	7,4	8,1	9,3



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

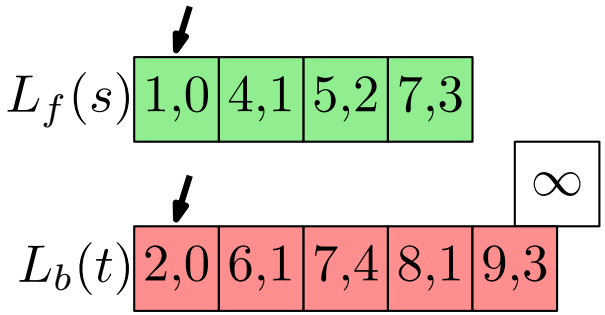
Hublabels

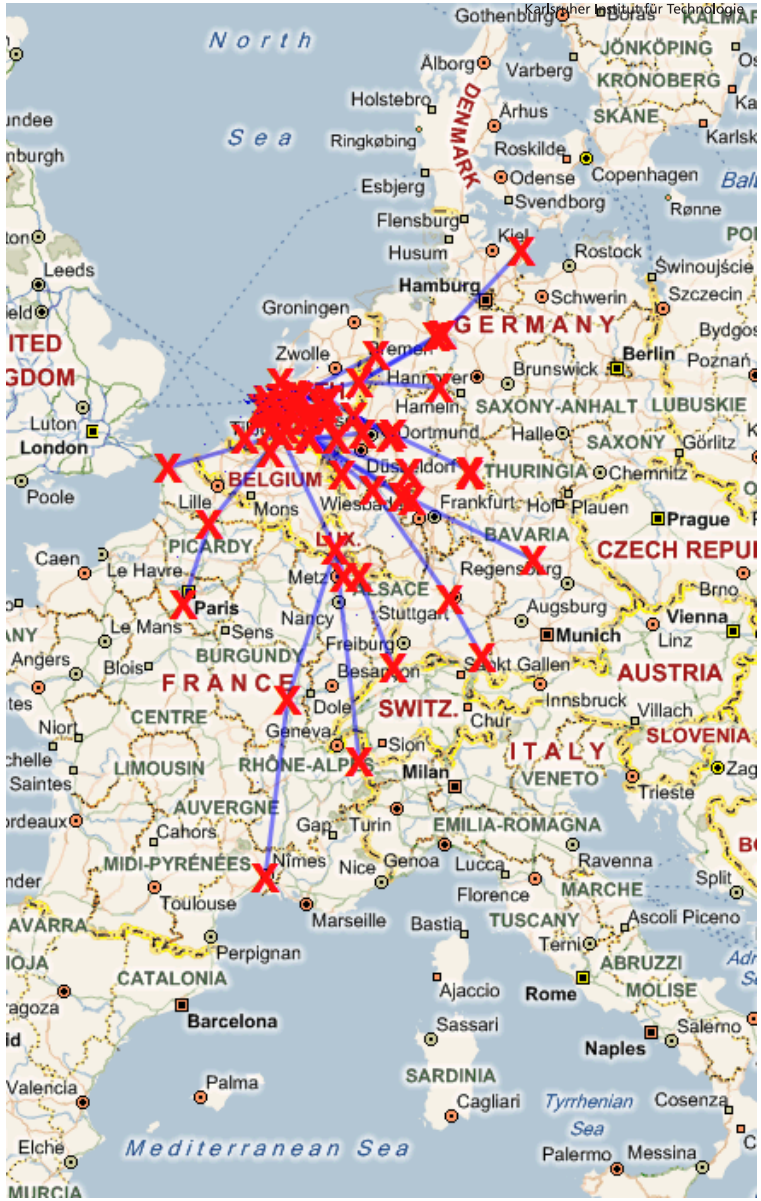
Speichern der Labels:

- als Menge von | Hub, Distanz | Paaren
- sortiert nach Hub-Knoten-ID

Anfrage:

- scannen von zwei Arrays
- nur einige Speicherzugriffe nötig





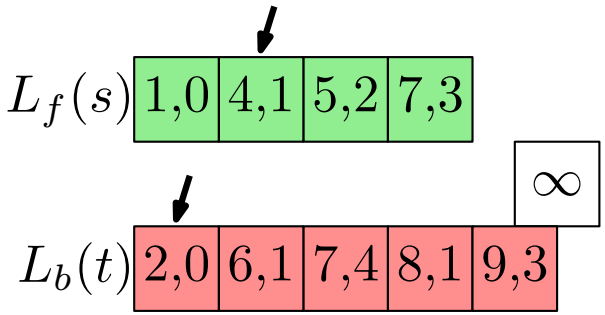
Hublabels

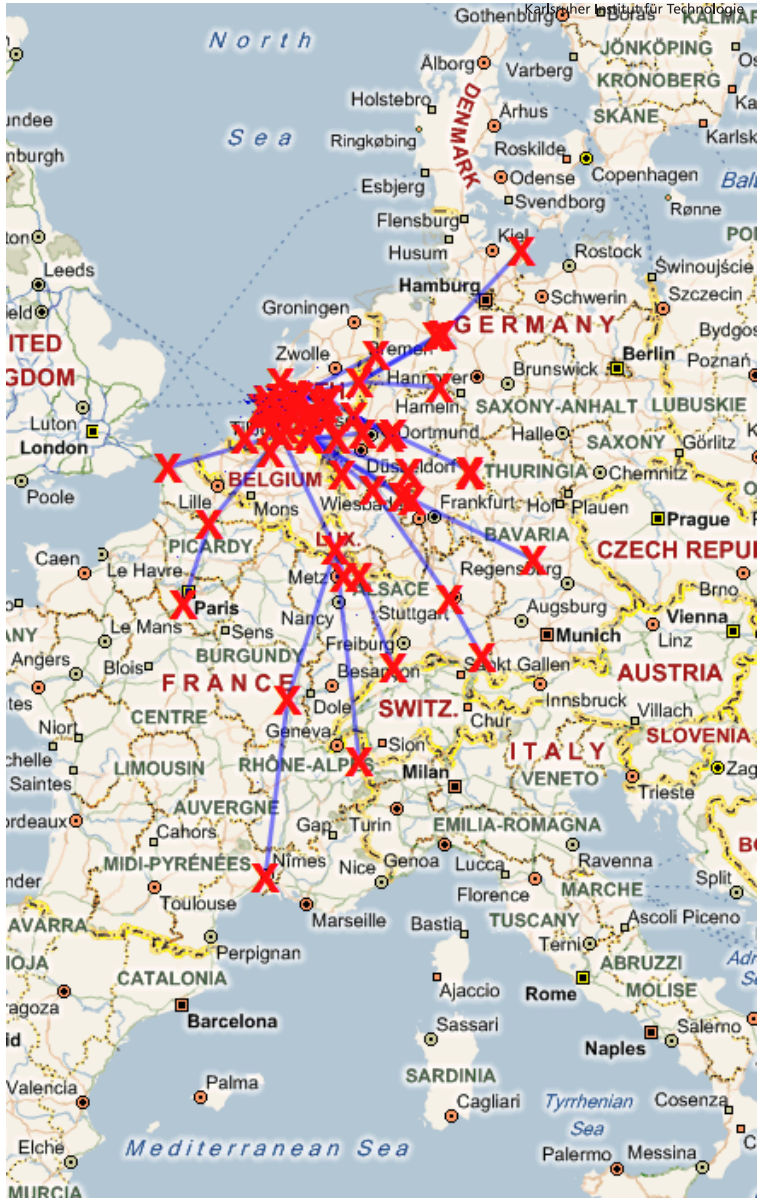
Speichern der Labels:

- als Menge von | Hub, Distanz | Paaren
- sortiert nach Hub-Knoten-ID

Anfrage:

- scannen von zwei Arrays
- nur einige Speicherzugriffe nötig





Institut für Theoretische Informatik
Lehrstuhl Algorithmik

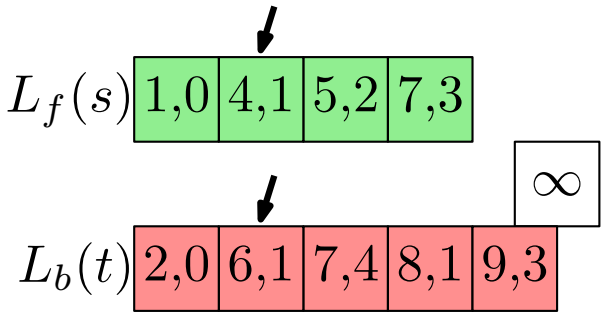
Hublabels

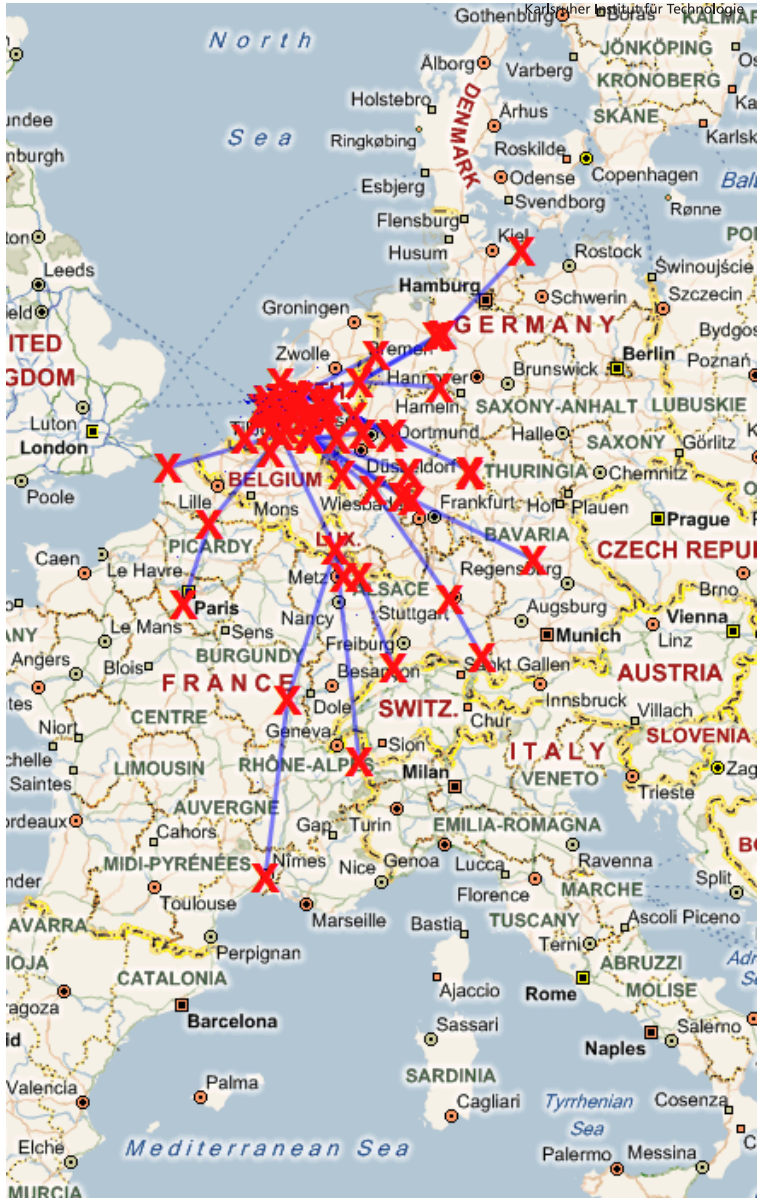
Speichern der Labels:

- als Menge von | Hub, Distanz | Paaren
- sortiert nach Hub-Knoten-ID

Anfrage:

- scannen von zwei Arrays
- nur einige Speicherzugriffe nötig





Institut für Theoretische Informatik
Lehrstuhl Algorithmik

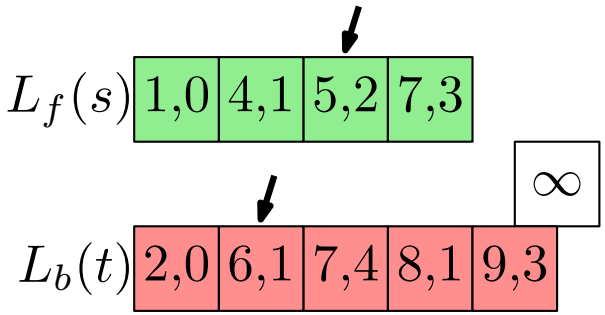
Hublabels

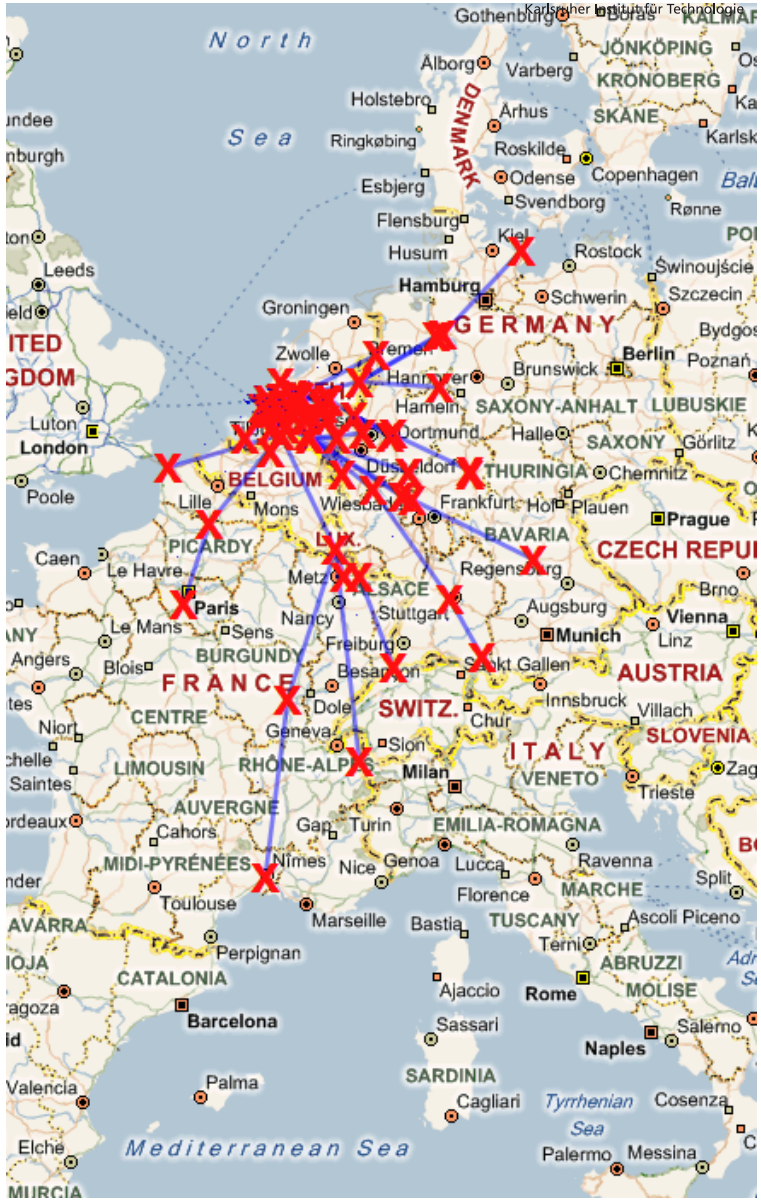
Speichern der Labels:

- als Menge von | Hub, Distanz | Paaren
- sortiert nach Hub-Knoten-ID

Anfrage:

- scannen von zwei Arrays
- nur einige Speicherzugriffe nötig





Institut für Theoretische Informatik
Lehrstuhl Algorithmik

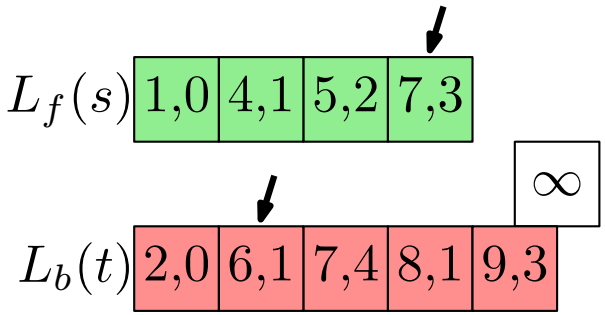
Hublabels

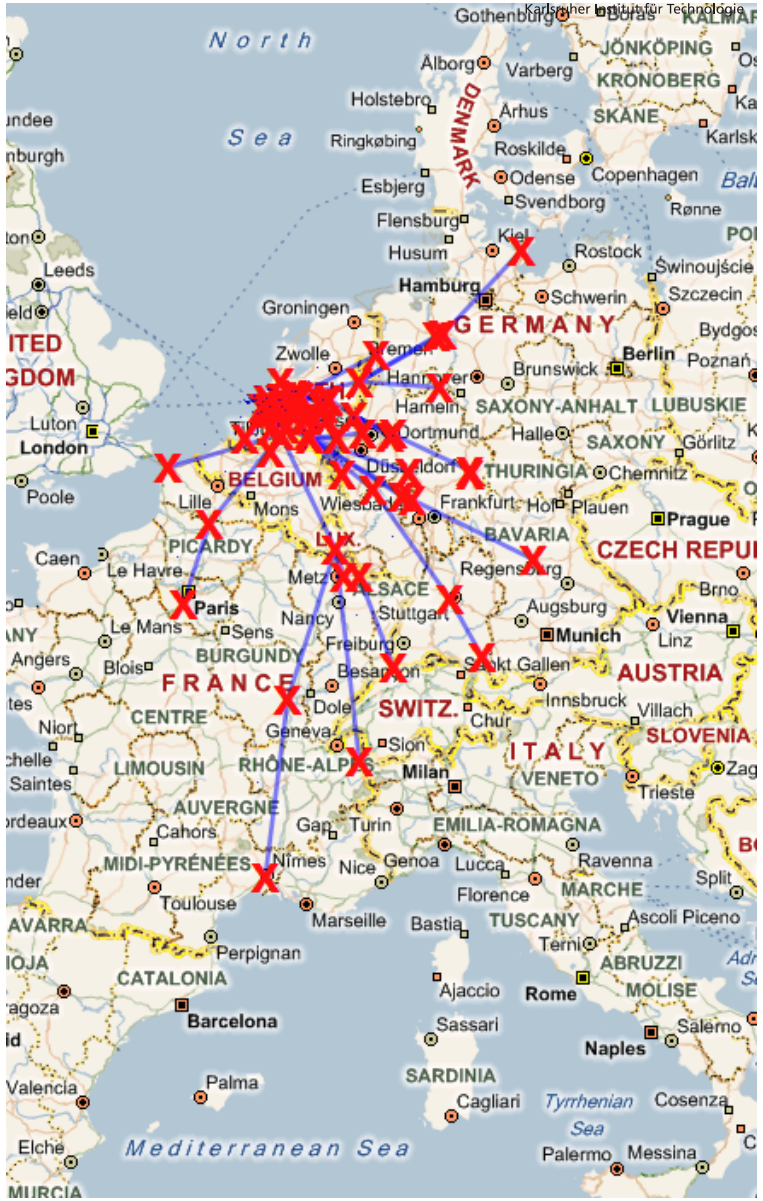
Speichern der Labels:

- als Menge von | Hub, Distanz | Paaren
- sortiert nach Hub-Knoten-ID

Anfrage:

- scannen von zwei Arrays
- nur einige Speicherzugriffe nötig





Institut für Theoretische Informatik
Lehrstuhl Algorithmik

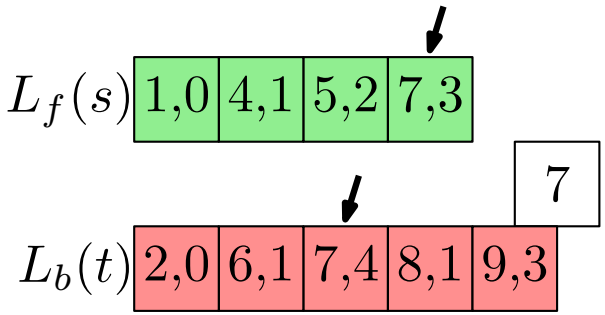
Hublabels

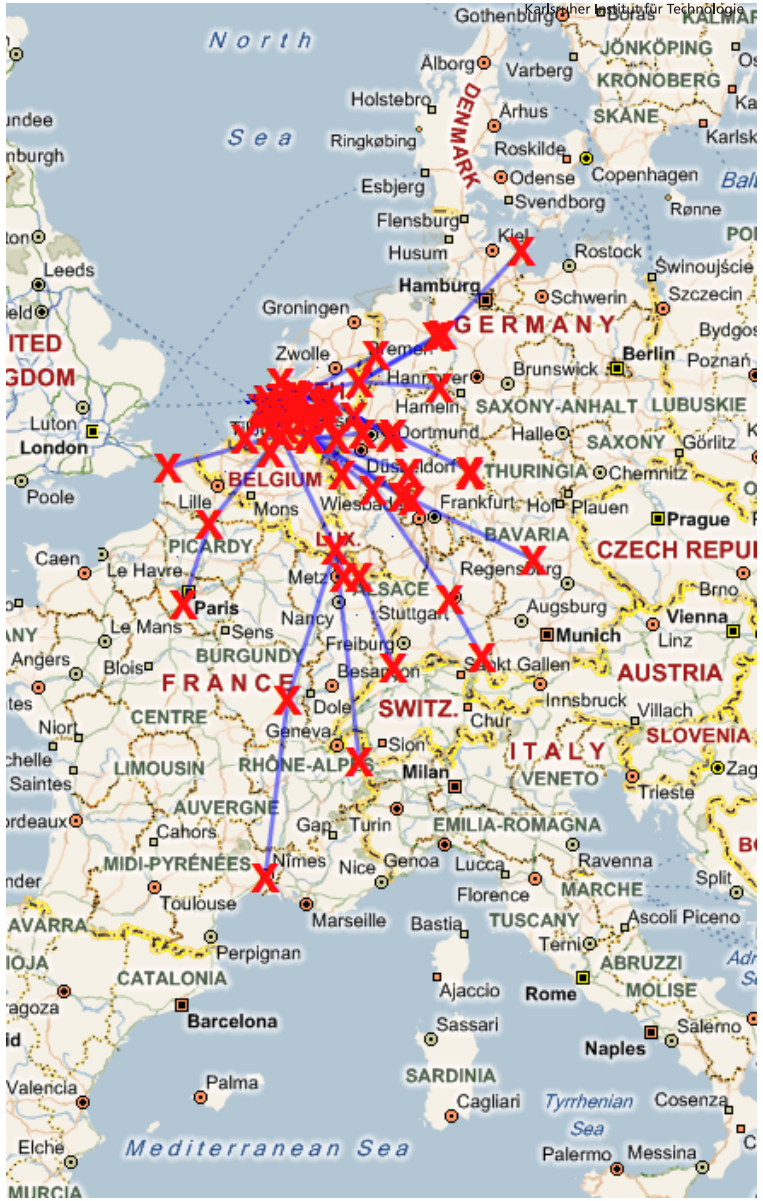
Speichern der Labels:

- als Menge von Hub, Distanz Paaren
- sortiert nach Hub-Knoten-ID

Anfrage:

- scannen von zwei Arrays
- nur einige Speicherzugriffe nötig





Institut für Theoretische Informatik
Lehrstuhl Algorithmik

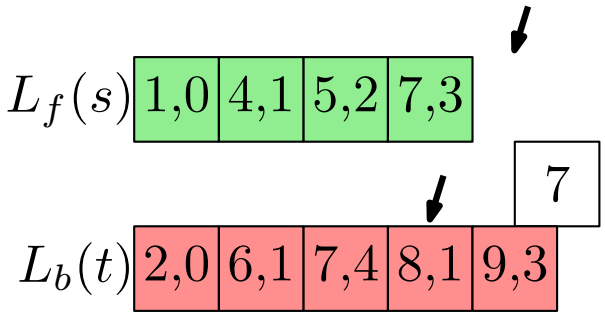
Hublabels

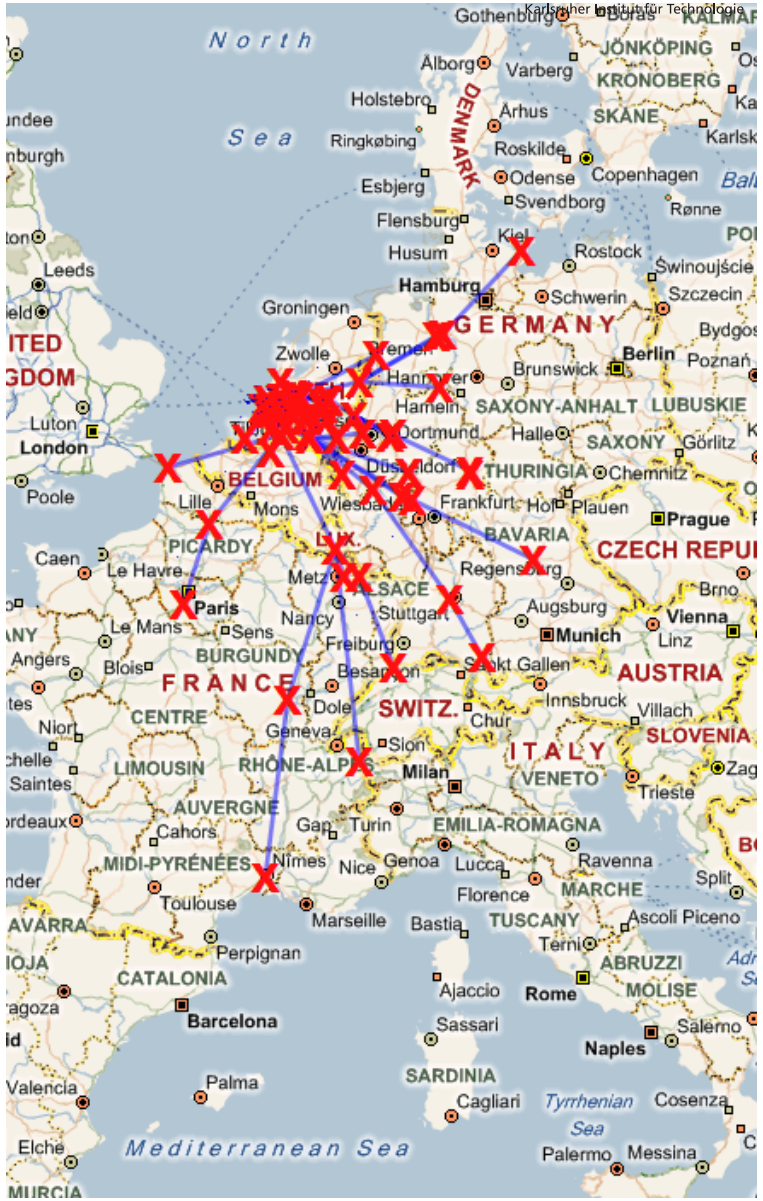
Speichern der Labels:

- als Menge von Hub, Distanz Paaren
- sortiert nach Hub-Knoten-ID

Anfrage:

- scannen von zwei Arrays
- nur einige Speicherzugriffe nötig





Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Hublabels

Speichern der Labels:

- als Menge von | Hub, Distanz | Paaren
- sortiert nach Hub-Knoten-ID

Anfrage:

- scannen von zwei Arrays
- nur einige Speicherzugriffe nötig
- sehr hohe Lokalität

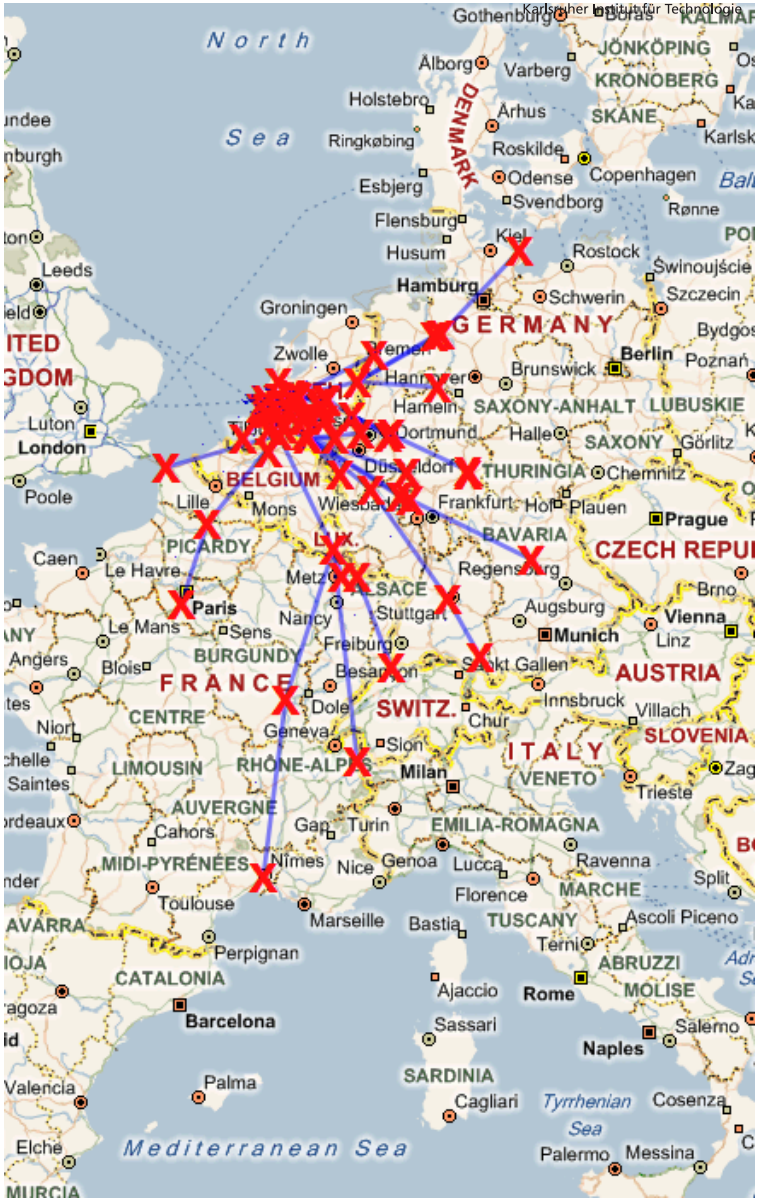
$L_f(s)$

1,0	4,1	5,2	7,3

7

$L_b(t)$

2,0	6,1	7,4	8,1	9,3



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Hublabels

Komplexität:

- Maximale Labellänge soll klein sein
- Optimale Hublabels zu berechnen ist NP-schwer [BGK+15]
- Es gibt $O(\log n)$-Approximation [GPPR04]
  - Ursprüngliche Laufzeit in $O(n^5)$
  - Wurde auf $O(n^3 \log n)$ verbessert [DGSW14]

Hierarchische Hublabels

Hierarchische Hublabels

- Jedes Labeling definiert eine Relation $\preceq$ auf den Labels wie folgt:

$$v \preceq u \iff u \in L_f(v) \cup L_b(v)$$

- Ein Labeling ist hierarchisch wenn $\preceq$ eine partielle Ordnung ist.
- Optimale Hierarchische Hublabels zu berechnen ist NP-schwer [BGK+15]

Kanonische Hublabels

- Ein kanonische Labeling bezüglich einer Knotenordnung O ist
  - hierarchisch
  - $\preceq$ muss mit O konsistent sein
  - aus keinem Label kann man ein Hub löschen
- Das kanonische Labeling ist eindeutig für eine feste Ordnung O

Verbindung zu CHs

- $\preceq$ ordnet die Knoten nach “Wichtigkeit” wie bei der CH
- CH Suchräume sind gültige hierarchisch Labels.
- aber sie sind größer als nötig (siehe stall-on-demand)
- und in der Regel sind sie nicht kanonisch
- $\rightarrow$ Überflüssige Knoten filtern

Verbindung zu CHs

- $\preceq$ ordnet die Knoten nach “Wichtigkeit” wie bei der CH
- CH Suchräume sind gültige hierarchisch Labels.
- aber sie sind größer als nötig (siehe stall-on-demand)
- und in der Regel sind sie nicht kanonisch
- $\rightarrow$ Überflüssige Knoten filtern

- Im Folgenden betrachten wir nur noch hierarchische Hublabels
- Für Beweise nehmen wir ferner an:
  - Kürzeste Wege sind eindeutig
  - Graphen sind ungerichtet

  $(L(v) := L_f(v) = L_b(v))$

Labels kanonisch machen

- Sei $m(s,t)$ der Knoten mit höchsten Rank auf dem kürzesten st-Pfad
- $m(s,t)$ ist der gemeinsame Hub von s und t über den der kürzeste Pfad geht.

Labels kanonisch machen

- Sei $m(s,t)$ der Knoten mit höchsten Rank auf dem kürzesten st-Pfad
- $m(s,t)$ ist der gemeinsame Hub von s und t über den der kürzeste Pfad geht.

Satz

Wir können einen Hub h aus dem Label $L(v)$ von v löschen

$$\Updownarrow$$

$$h \neq m(v,h)$$

Labels kanonisch machen

- Sei $m(s,t)$ der Knoten mit höchsten Rank auf dem kürzesten st-Pfad
- $m(s,t)$ ist der gemeinsame Hub von s und t über den der kürzeste Pfad geht.

Satz

Wir können einen Hub h aus dem Label $L(v)$ von v löschen

$$\Updownarrow$$

$$h \neq m(v,h)$$

- Zwei Richtungen:
- Wenn $h = m(v,h)$, dann dürfen wir h nicht aus $L(v)$ löschen.
- Wenn $h \neq m(v,h)$, dann dürfen wir h aus $L(v)$ löschen.

Labels kanonisch machen

Übersicht:

- Erste Richtung: Wenn $h = m(v, h)$, dann dürfen wir h nicht aus $L(v)$ löschen.
- Wir müssen zeigen, dass es eine Anfrage gibt, die nach der Herausnahme von h aus dem Label von v inkorrekt wird.
- Wir zeigen, dass wenn wir h löschen, die vh-Anfrage dann falsch beantwortet wird

Labels kanonisch machen

Übersicht:

- Erste Richtung: Wenn $h = m(v, h)$, dann dürfen wir h nicht aus $L(v)$ löschen.
- Wir müssen zeigen, dass es eine Anfrage gibt, die nach der Herausnahme von h aus dem Label von v inkorrekt wird.
- Wir zeigen, dass wenn wir h löschen, die vh-Anfrage dann falsch beantwortet wird

Beweis:

- Der gemeinsame Hub von h und v darf nicht niedriger sein als h und v
  (folgt direkt aus der Definition von kanonischem Labeling)
- Der höchste Knoten auf dem kürzesten vh-Pfad ist h
  (Voraussetzung)
- Da ferner jeder Knoten eine eindeutige Position hat können sich $L(v)$ und $L(h)$ nur in h schneiden
- $\Longrightarrow$ h darf nicht gelöscht werden

Labels kanonisch machen

Übersicht:

- Zweite Richtung: Wenn $h \neq m(v, h)$, dann dürfen wir h aus $L(v)$ löschen
- Wir müssen zeigen, dass alle Anfragen nach der Herausnahme von h aus dem Label von v noch korrekt sind

Labels kanonisch machen

Übersicht:

- Zweite Richtung: Wenn $h \neq m(v, h)$, dann dürfen wir h aus $L(v)$ löschen
- Wir müssen zeigen, dass alle Anfragen nach der Herausnahme von h aus dem Label von v noch korrekt sind

Beweis:

- $L(v)$ wird nur bei vt- oder sv-Anfragen angeschaut, nur diese können also inkorrekt werden
  $\rightarrow$ Betrachte ohne Beschränkung der Allgemeinheit vt-Anfragen
- Eine vt-Anfrage kann nur inkorrekt werden, wenn h auf dem kürzesten vt-Pfad liegt
- Es reicht also zu zeigen, dass alle vt-Anfragen, die durch h gehen, korrekt sind
- **Ziel:** Wir zeigen, dass diese vt-Anfragen sich nicht nur in h, sondern auch in $m(v, h)$ oder in $m(h, t)$ treffen

Labels kanonisch machen

Übersicht:

- Zweite Richtung: Wenn $h \neq m(v,h)$, dann dürfen wir h aus $L(v)$ löschen
- Wir müssen zeigen, dass alle Anfragen nach der Herausnahme von h aus dem Label von v noch korrekt sind

Beweis:

- **Ziel:** Wir zeigen, dass diese vt-Anfragen sich nicht nur in h sondern auch in $m(v,h)$ oder in $m(h,t)$ treffen
- Fall 1:
  - $m(v,h)$ höher als in $m(h,t)$
  - $m(v,h)$ höchster Knoten auf vt-Pfad
  - Nach Argument von letzter Folie: $m(v,h) \in L(v)$ und $m(v,h) \in L(t)$
  - vt-Anfrage trifft sich nicht nur in h, sondern auch in $m(v,h)$
  - Da nach Voraussetzung $h \neq m(v,h)$, können wir h löschen

Labels kanonisch machen

Übersicht:

- Zweite Richtung: Wenn $h \neq m(v, h)$, dann dürfen wir h aus $L(v)$ löschen
- Wir müssen zeigen, dass alle Anfragen nach der Herausnahme von h aus dem Label von v noch korrekt sind

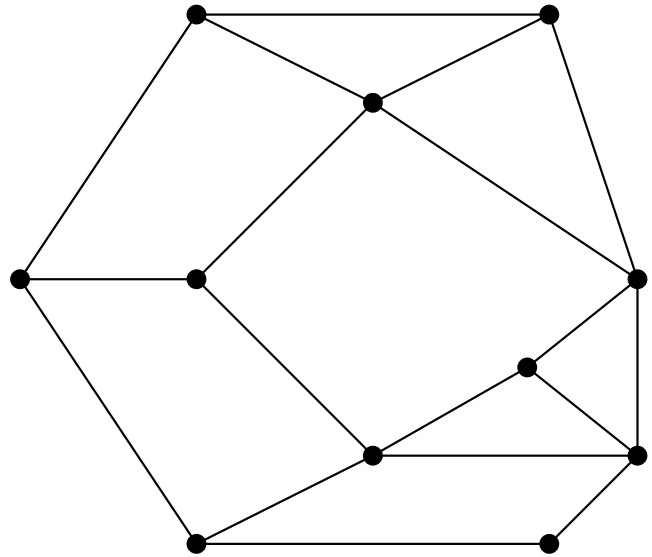
Beweis:

- **Ziel:** Wir zeigen, dass diese vt-Anfragen sich nicht nur in h sondern auch in $m(v, h)$ oder in $m(h, t)$ treffen
- Fall 2:
  - $m(h, t)$ höher als in $m(v, h)$
  - $m(h, t)$ höchster Knoten auf vt-Pfad
  - Nach Argument von letzter Folie: $m(h, t) \in L(v)$ und $m(h, t) \in L(v)$
  - vt-Anfrage trifft sich nicht nur in h, sondern auch in $m(h, t)$
  - Wenn $h = m(h, t)$, dann wäre h der höchste Knoten auf dem vt-Pfad. Das kann aber nicht sein, da $m(v, h)$ höher ist.
  - Da $h \neq m(h, t)$, können wir h löschen

Vorberechnung

Idee:

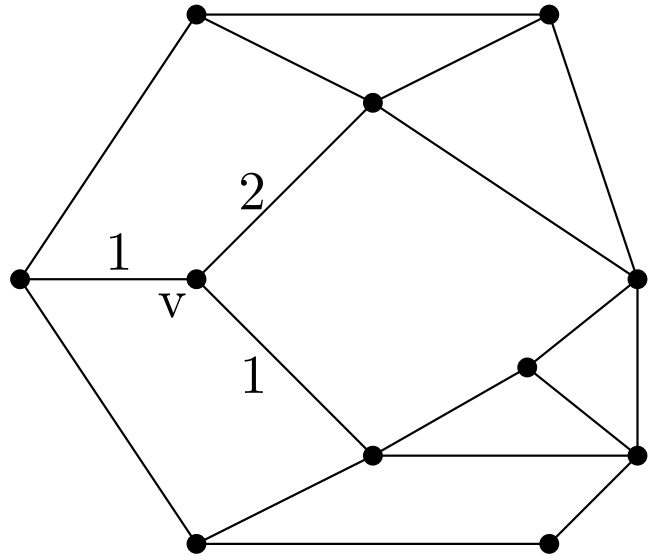
- benutze Knotenordnung



Vorberechnung

Idee:

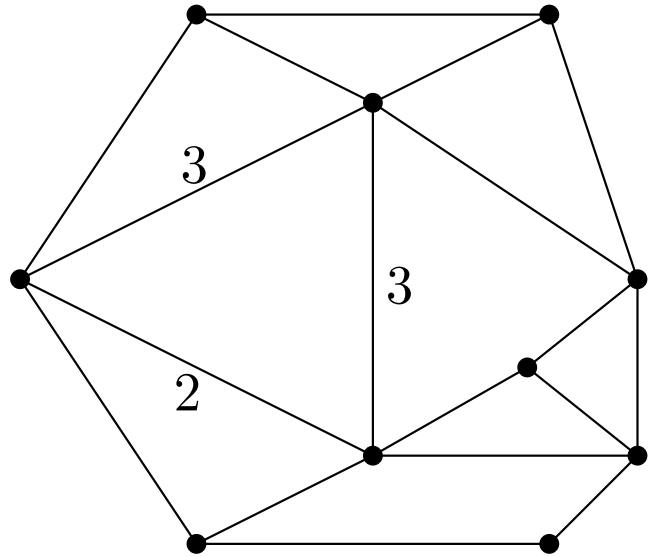
- benutze Knotenordnung
- kontrahiere Knoten v



Vorberechnung

Idee:

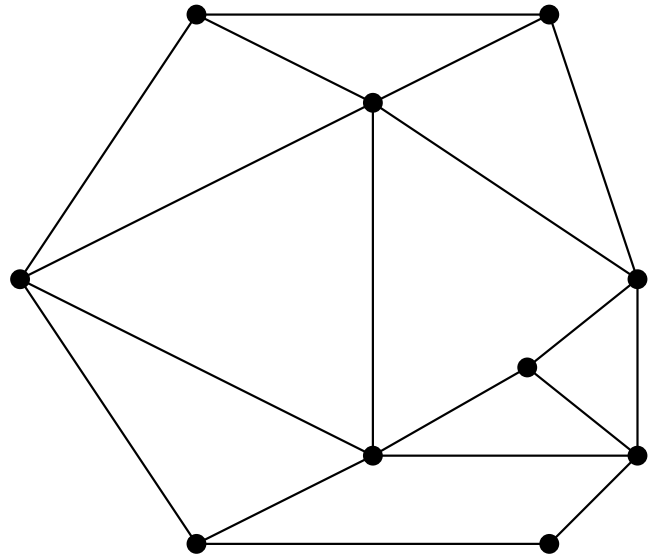
- benutze Knotenordnung
- kontrahiere Knoten v



Vorberechnung

Idee:

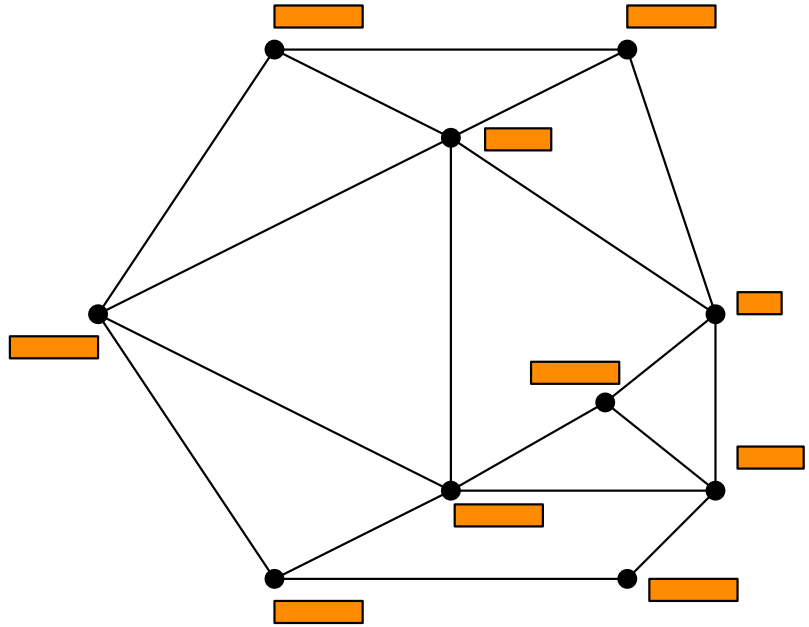
- benutze Knotenordnung
- kontrahiere Knoten v



Vorberechnung

Idee:

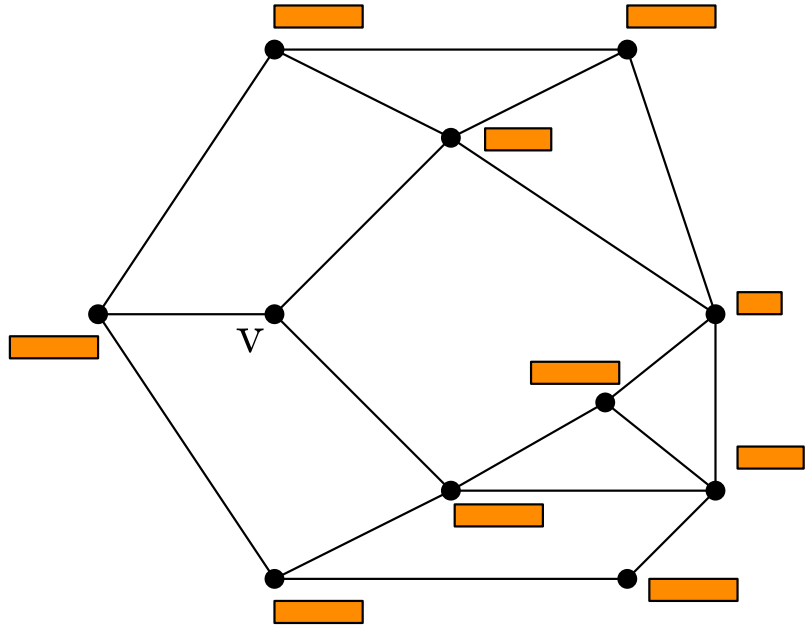
- benutze Knotenordnung
- kontrahiere Knoten v
- berechne Labels rekursiv



Vorberechnung

Idee:

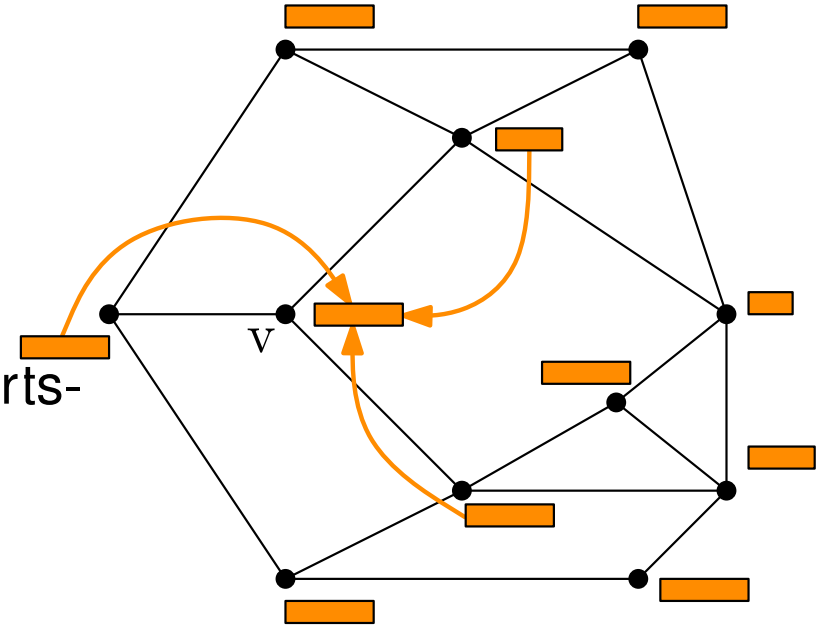
- benutze Knotenordnung
- kontrahiere Knoten v
- berechne Labels rekursiv



Vorberechnung

Idee:

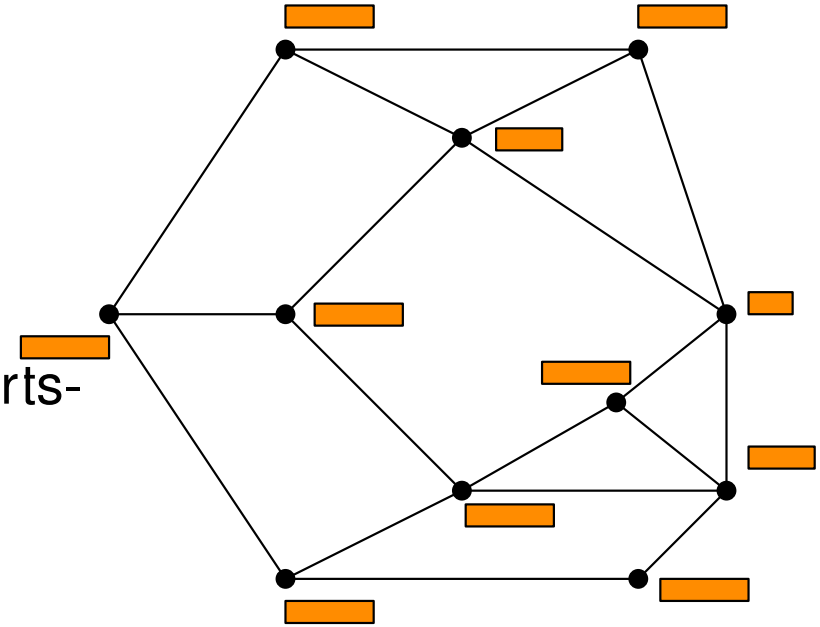
- benutze Knotenordnung
- kontrahiere Knoten v
- berechne Labels rekursiv
- vereinige (merge) Labels der Aufwärts-Nachbarn von v
- dünne Label aus



Vorberechnung

Idee:

- benutze Knotenordnung
- kontrahiere Knoten v
- berechne Labels rekursiv
- vereinige (merge) Labels der Aufwärts-Nachbarn von v
- dünne Label aus



Korrektheit:

- analog zu Korrektheit von CH
- Argumentation über den wichtigsten Knoten auf dem Pfad
- dieser ist im Vorwärtslabel von s und im Rückwärtslabel von t

Vereinigen von Labels

Generell:

- $L_f(v)$ ist die Vereinigung der Label der Aufwärtsnachbar von v im augmentierten Graph.
- Die Distanzen zu jedem Hub in $L_f(v)$ werden um die Länge der Kante zum Nachbarknoten erhöht.
- $L_f(v)$ enthält zusätzlich v als Hub mit Distanz 0.
- So konstruiertes Label ist korrekt, aber nicht kleinstmöglich

Vereinigen von Labels

Generell:

- $L_f(v)$ ist die Vereinigung der Label der Aufwärtsnachbar von v im augmentierten Graph.
- Die Distanzen zu jedem Hub in $L_f(v)$ werden um die Länge der Kante zum Nachbarknoten erhöht.
- $L_f(v)$ enthält zusätzlich v als Hub mit Distanz 0.
- So konstruiertes Label ist korrekt, aber nicht kleinstmöglich

Ausdünnen:

- manche Knoten im Label sind nicht notwendig
- **Ziel**: Filtere Hubs h für die $h \neq m(v, h)$
- Label von h ist final, da h höher als v
- Label von v ist korrekt (aber noch nicht minimal)
- Wir können eine HL-Anfrage durchführen, um $m(v, h)$ zu bestimmen
- Lösche h, wenn $h \neq m(v, h)$

Pruned Labeling:[DGPW14, AIY13]
Alternative Labelkonstruktion

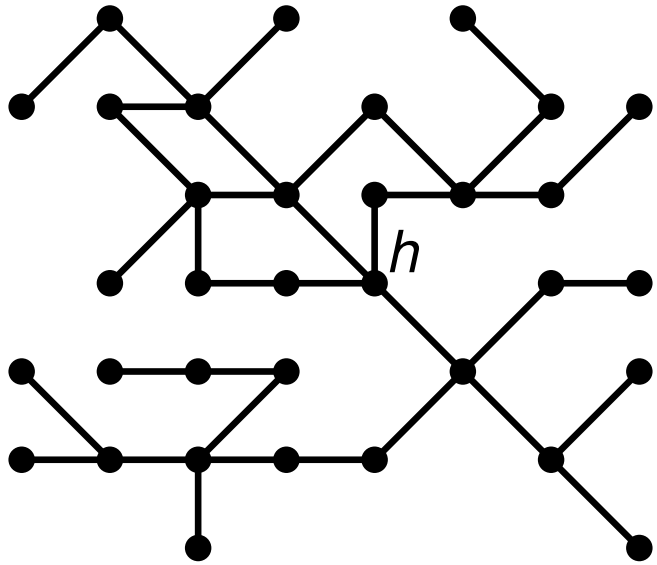
Idee:

- Verteile Hubs auf Labels
- h ist Hub von v
  $\iff$ $h = m(h, v)$
  $\iff$ es gibt auf dem hv-Pfad keinen höheren Knoten als h
- Starte Dijkstra von h und besuche alle v, in deren Label h liegt

Pruned Labeling:[DGPW14, AIY13]
Alternative Labelkonstruktion

Idee:

- Verteile Hubs auf Labels
- h ist Hub von v
  $\Longleftrightarrow$ $h = m(h, v)$
  $\Longleftrightarrow$ es gibt auf dem hv-Pfad keinen höheren Knoten als h
- Starte Dijkstra von h und besuche alle v, in deren Label h liegt

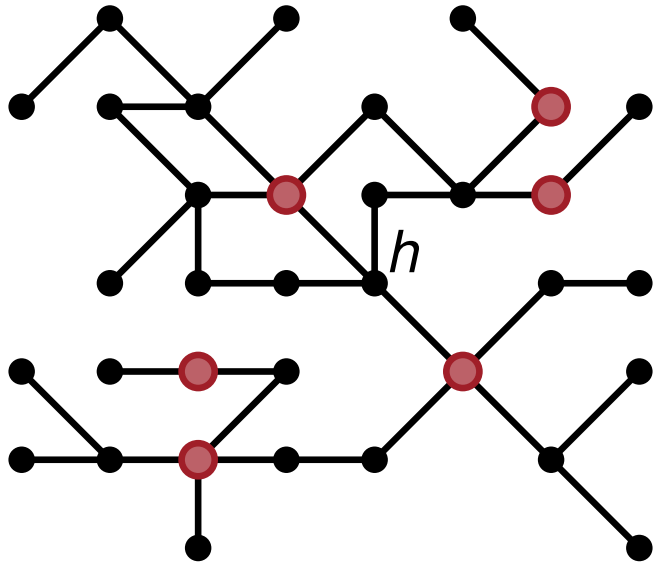


Ziel: h in alle Labels verteilen

Pruned Labeling:[DGPW14, AIY13]
Alternative Labelkonstruktion

Idee:

- Verteile Hubs auf Labels
- h ist Hub von v
  $\Longleftrightarrow$ $h = m(h, v)$
  $\Longleftrightarrow$ es gibt auf dem hv-Pfad keinen höheren Knoten als h
- Starte Dijkstra von h und besuche alle v, in deren Label h liegt

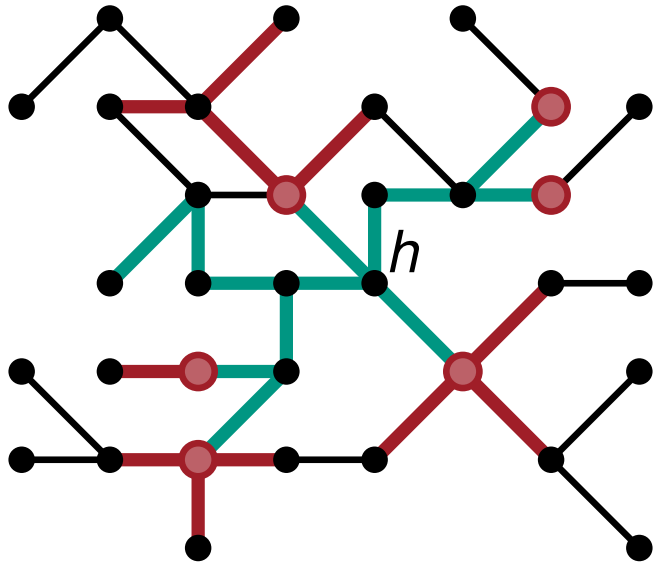


Rote Knoten sind höher als h

Pruned Labeling:[DGPW14, AIY13]
Alternative Labelkonstruktion

Idee:

- Verteile Hubs auf Labels
- h ist Hub von v
  $\Longleftrightarrow$ $h = m(h, v)$
  $\Longleftrightarrow$ es gibt auf dem hv-Pfad keinen höheren Knoten als h
- Starte Dijkstra von h und besuche alle v, in deren Label h liegt



h kommt in Label von Knoten, die über grüne Pfade erreichbar sind

Pruned Labeling:[DGPW14, AIY13]

- Dijkstras Algorithmus sucht ganzen Graph ab
- $\rightarrow$ muss vorzeitig abgebrochen werden.

Option 1:

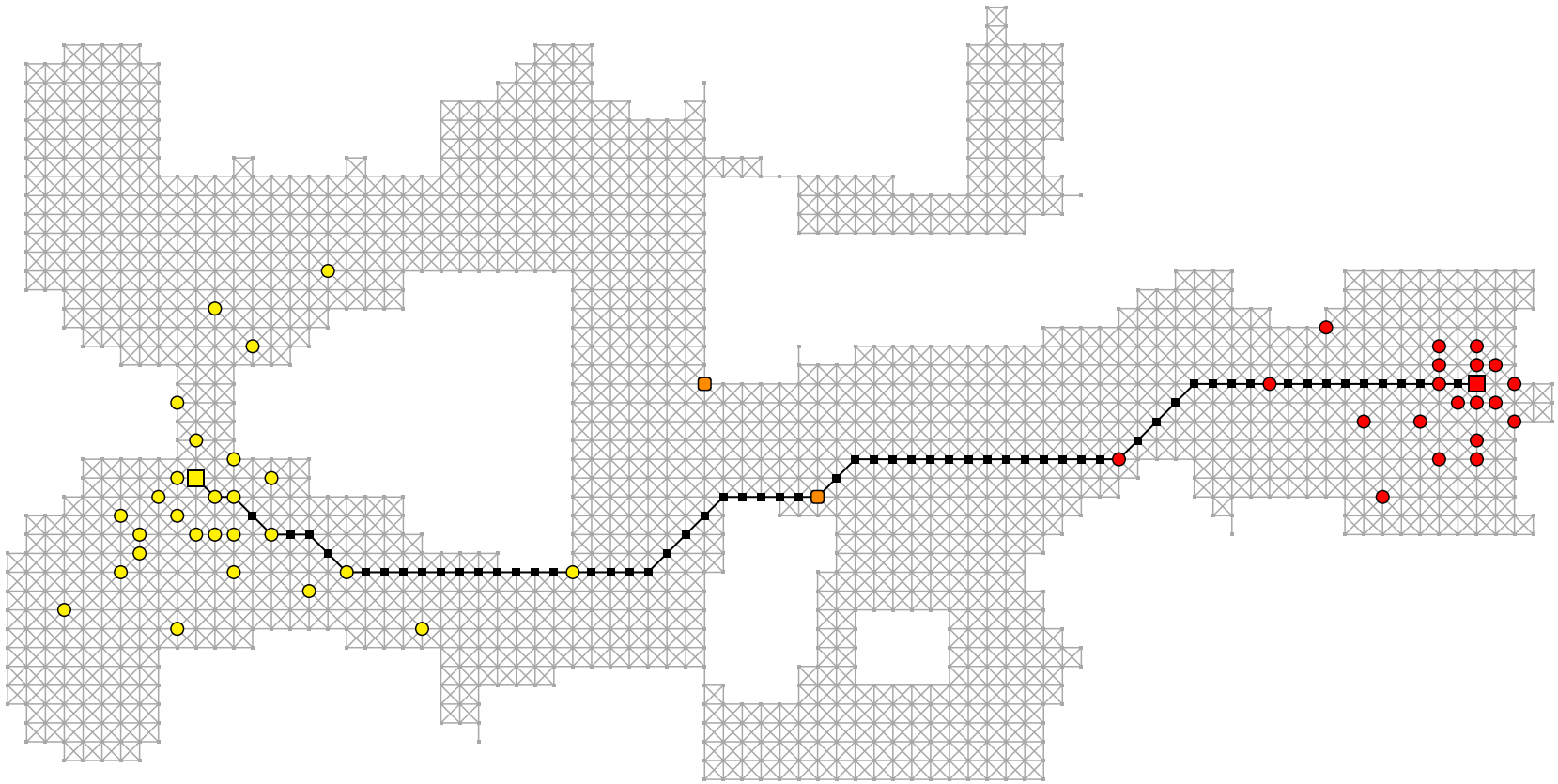
- **Beobachtung**: Wenn alle Knoten in der Queue über rote Pfade gehen, dann werden nie wieder Knoten aufgenommen die über grüne Pfade gehen
- **Idee**: Speichere welche Knoten über grüne Pfade erreichbar sind. Wenn keine grünen mehr in der Queue sind, dann kann die Suche abgebrochen werden.

Pruned Labeling:[DGPW14, AIY13]

Option 2:

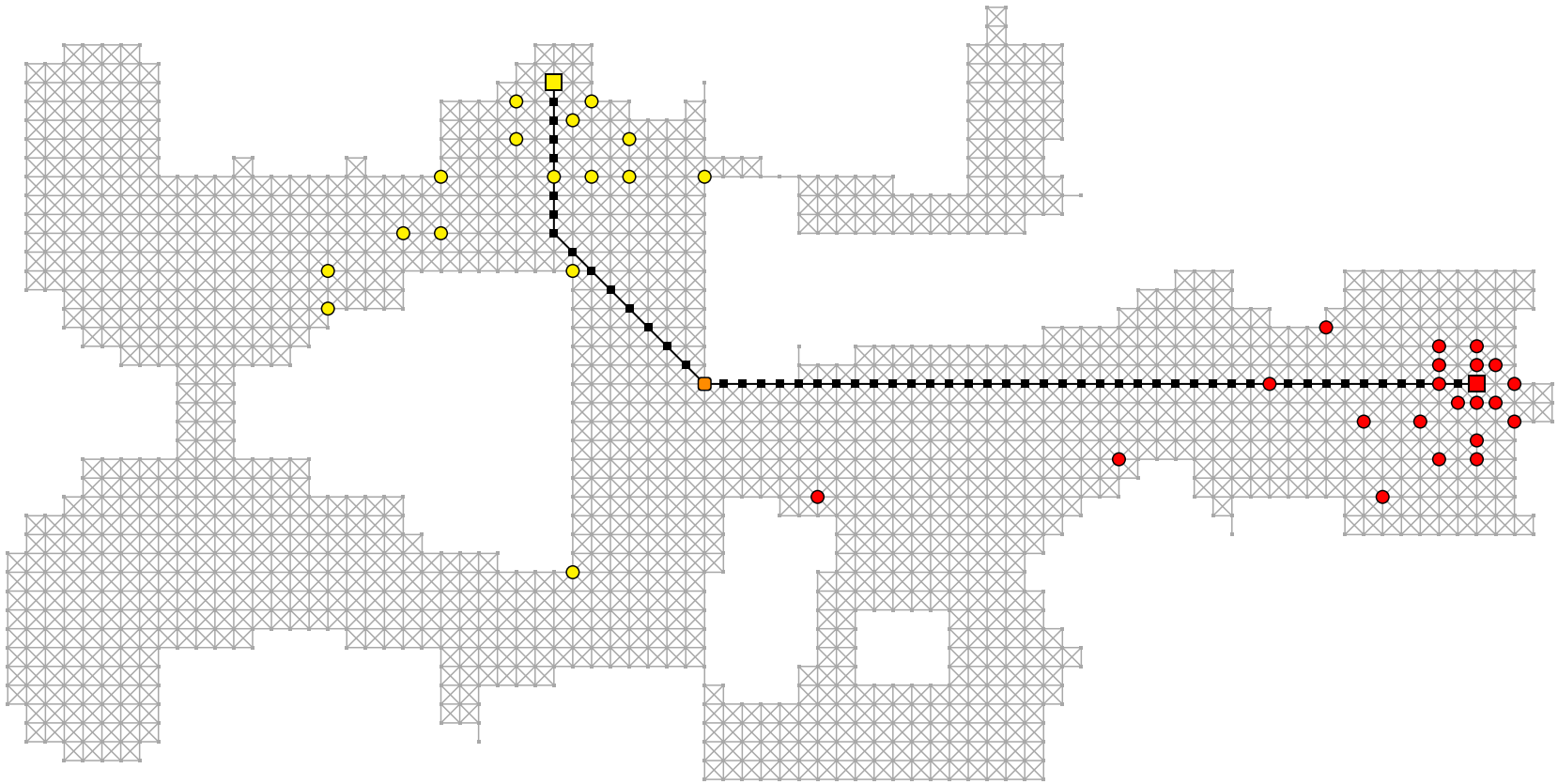
- Verteile hohe Knoten zuerst
- **Effekt**: $m(h, v)$ wird vor h verteilt (da höher) oder $m(h, v)$ ist gleich h
- Wir können deswegen $m(h, v)$ per HL-Anfrage auf den bereits aufgebauten partiellen Labels berechnen
  - Wenn die Anfrage einen höchsten gemeinsamen Knoten findet, dann ist das $m(h, v)$ und $m(h, v) \neq h$
  - Wenn die Anfrage keinen gemeinsamen Knoten findet, dann ist $m(h, v) = h$
- Damit kann eine Pruning-Regel für Dijkstras Algorithmus gebaut werden
- Nachdem ein Knoten v aus der Queue genommen wird berechnet man $m(h, v)$
  - Wenn $m(h, v) = h$, dann fügt man h in das Label von v und relaxiert die ausgehenden Kanten von v
  - Wenn $m(h, v) \neq h$, dann fügt man h nicht in das Label von v und pruned die Suche an v.

Beispiel: Grid Graph



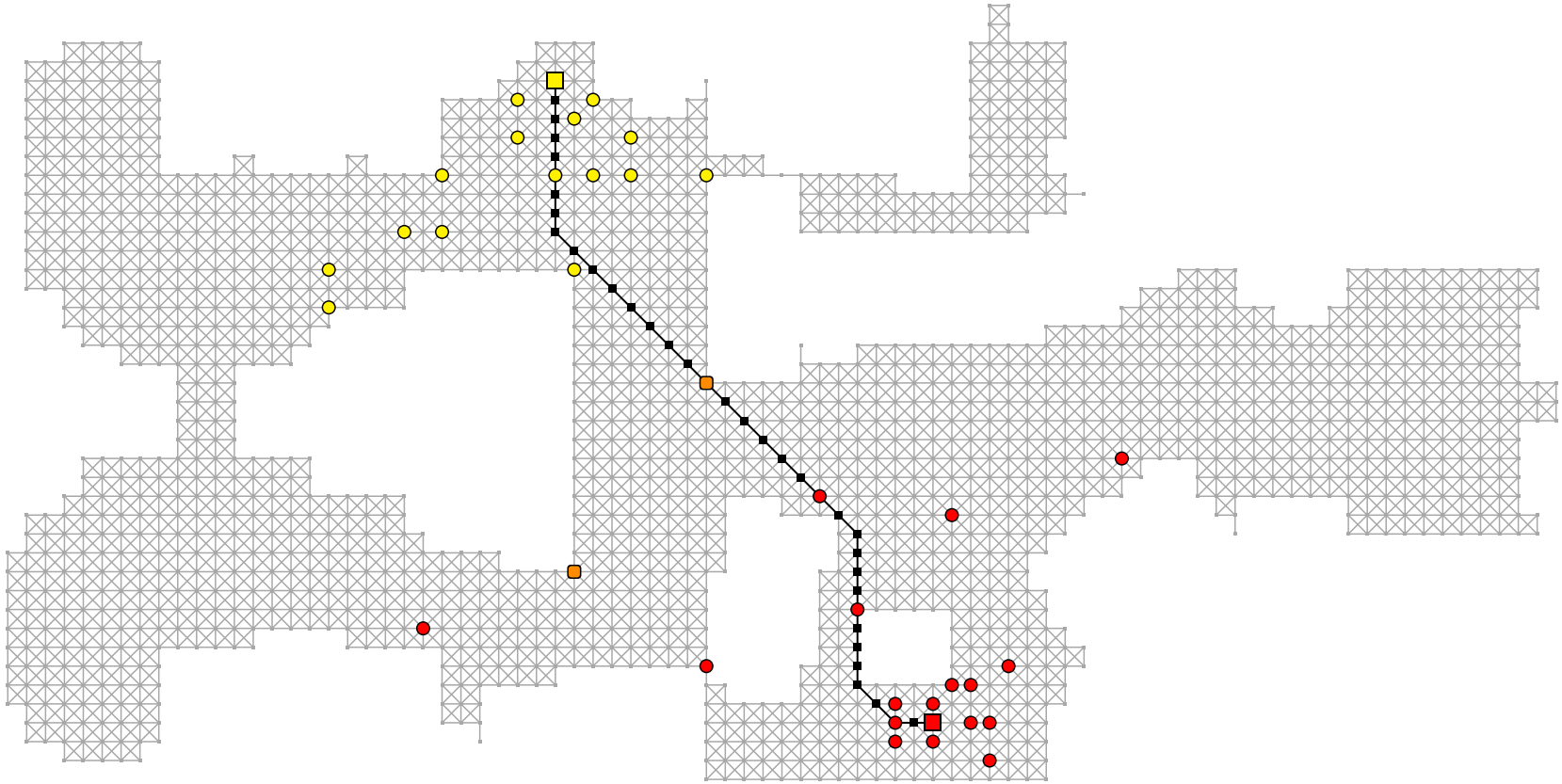
Autoren von [DGPW14] haben diese Bilder erstellt

Beispiel: Grid Graph



Autoren von [DGPW14] haben diese Bilder erstellt

Beispiel: Grid Graph



Autoren von [DGPW14] haben diese Bilder erstellt

Ergebnisse

	Preprocessing		Query
Method	time [h:m]	space [GB]	time [μs]
MLD-3	$<$ 0:01	0.4	912
CH	0:02	0.4	96.3
HL-0	0:03	22.5	0.700
HL-15	0:05	18.8	0.556
HL-17	0:25	18.4	0.545
HL-∞	5:43	16.8	0.508
HL-∞ + Oracle	6:12	17.7	0.254
Table Lookup	???	1 208 358.7	0.056

- Vorberechnung mit 12 Cores parallelisiert
- Table Lookup nimmt an, dass Speicher nicht im Cache liegt
- HL-x: Use top-down order for top 2^x vertices

Ergebnisse

	Preprocessing		Query
Method	time [h:m]	space [GB]	time [μs]
MLD-3	$<$ 0:01	0.4	912
CH	0:02	0.4	96.3
HL-0	0:03	22.5	0.700
HL-15	0:05	18.8	0.556
HL-17	0:25	18.4	0.545
HL-∞	5:43	16.8	0.508
HL-∞ + Oracle	6:12	17.7	0.254
Table Lookup	???	1 208 358.7	0.056

- Vorberechnung mit 12 Cores parallelisiert
- Table Lookup nimmt an, dass Speicher nicht im Cache liegt
- HL-x: Use top-down order for top 2^x vertices

- HL ist Faktor 100 schneller als CH (Speedup 10 Mio)
- hoher Speicherverbrauch (durch Kompression reduzierbar)

Zusammenfassung

- Knotenordnung definiert Labeling
- Beschleunigung gegenüber CH von Faktor mehr als 100
- durch bessere Lokalität
- nur 5 mal langsamer als ein Speicherzugriff
- schnellster Algorithmus momentan
- beschleunigt lokale und globale Anfragen
- aber Speicherverbrauch sehr hoch

HLDB

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Aufgabenstellung

Beobachtung:

- Queries sind schnell genug
- Visualisierung und Netzwerklatenz der Flaschenhals
- schwieriger und hoch optimierter Code

Aufgabenstellung

Beobachtung:

- Queries sind schnell genug
- Visualisierung und Netzwerklatenz der Flaschenhals
- schwieriger und hoch optimierter Code

Können wir Geschwindigkeit gegen einfachere Bedienbarkeit eintauschen?

Aufgabenstellung

Beobachtung:

- Queries sind schnell genug
- Visualisierung und Netzwerklatenz der Flaschenhals
- schwieriger und hoch optimierter Code

Können wir Geschwindigkeit gegen einfachere Bedienbarkeit eintauschen?

Idee:

- Implementiere Routenplanung direkt in SQL
- auch die Erweiterungen

Location Services in SQL

Vorteile:

- einfach zu nutzen
- Daten meist eh schon in SQL
- skalieren einfach (bestehende Datenbanksysteme, Cloud SQL)
- auch für Nicht-Routing-Experten zu nutzen
- External Memory Implementation “umsonst”

Location Services in SQL

Vorteile:

- einfach zu nutzen
- Daten meist eh schon in SQL
- skalieren einfach (bestehende Datenbanksysteme, Cloud SQL)
- auch für Nicht-Routing-Experten zu nutzen
- External Memory Implementation “umsonst”

Nachteile:

- SQL viel langsamer als optimierter C++ Code
- keine aufwändigen Datenstrukturen (Graph, Priority Queue)
- Dijkstra-basierte Techniken sind keine Option

Ideensammlung

Welcher Ansatz?

Ideensammlung

Welcher Ansatz?

- keine Priority Queue?

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Ideensammlung

Welcher Ansatz?

- keine Priority Queue?
- keine Graphdatenstruktur?

Ideensammlung

Welcher Ansatz?

- keine Priority Queue?
- keine Graphdatenstruktur?

Idee: Hub Labeling

Speichern der Label

Idee:

- berechne Label mit C++ (wie bei HubLabels)
- aber speicher die Label direkt in der Datenbank
- ein Vorwärtslabel von Knoten v mit k Hubs:
  - erzeugt k Triples $(v, u, d(v, u))$ in Tabelle `forward`
- Rückwärtslabel genauso in `backward`
- ungefähr 1.35 Milliarden Zeilen pro Tabelle (ca. 19 GB pro Richtung)

$L_f(1)$ | 1,0 | 4,1 | 5,2 | 7,3 |

$L_b(2)$ | 2,0 | 6,1 | 7,4 |

`forward`

node	hub	dist
1	1	0
1	4	1
1	5	2
1	7	3
2	2	0
⋮	⋮	⋮

`backward`

node	hub	dist
1	1	0
1	4	4
2	2	0
2	6	1
2	7	4
⋮	⋮	⋮

Speichern der Label

Idee:

- berechne Label mit C++ (wie bei HubLabels)
- aber speicher die Label direkt in der Datenbank
- ein Vorwärtslabel von Knoten v mit k Hubs:
  - erzeugt k Triples $(v, u, d(v, u))$ in Tabelle `forward`
- Rückwärtslabel genauso in `backward`
- ungefähr 1.35 Milliarden Zeilen pro Tabelle (ca. 19 GB pro Richtung)
- indiziere nach `node` (primary) und `hub` (secondary)

$L_f(1)$ | 1,0 | 4,1 | 5,2 | 7,3 |

$L_b(2)$ | 2,0 | 6,1 | 7,4 |

`forward`

node	hub	dist
1	1	0
1	4	1
1	5	2
1	7	3
2	2	0
⋮	⋮	⋮

`backward`

node	hub	dist
1	1	0
1	4	4
2	2	0
2	6	1
2	7	4
⋮	⋮	⋮

SQL Query

Algorithm 1: SQL_DIST

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

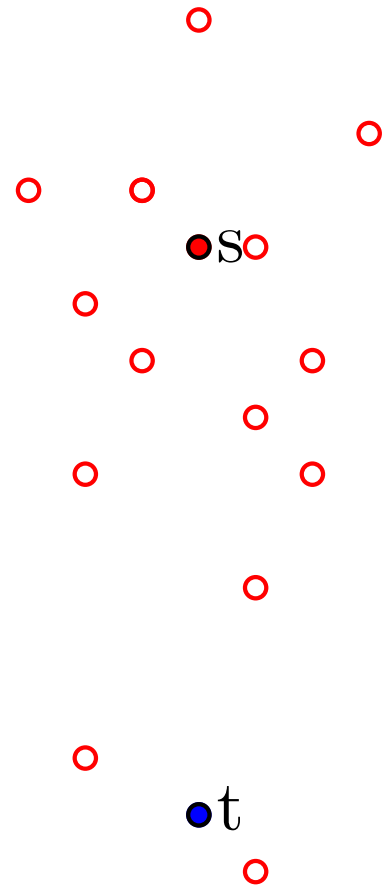
```
SELECT
    MIN(forward.dist+backward.dist)
FROM forward,backward
WHERE
    forward.node = s AND
    backward.node = t AND
    forward.hub = backward.hub
```

SQL Query

Algorithm 1: SQL_DIST

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

```
SELECT
    MIN(forward.dist+backward.dist)
FROM forward,backward
WHERE
    forward.node = s AND
    backward.node = t AND
    forward.hub = backward.hub
```

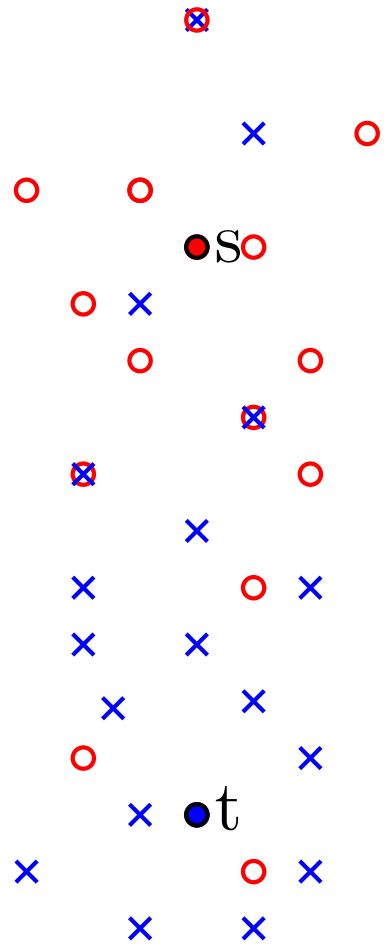


SQL Query

Algorithm 1: SQL_DIST

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

```
SELECT
    MIN(forward.dist+backward.dist)
FROM forward,backward
WHERE
    forward.node = s AND
    backward.node = t AND
    forward.hub = backward.hub
```

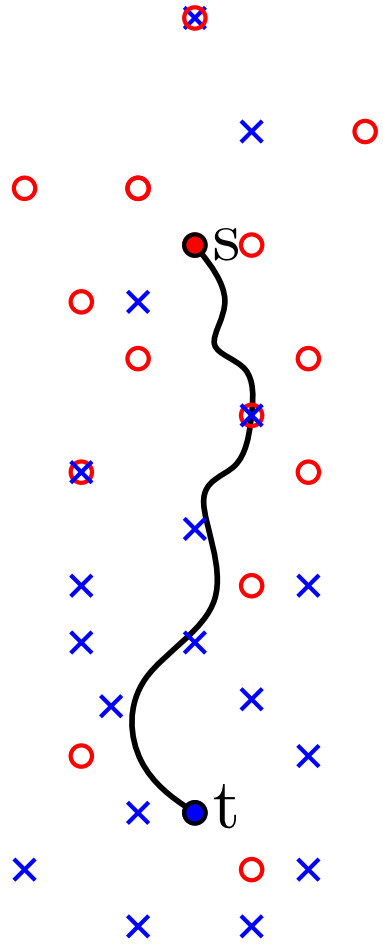


SQL Query

Algorithm 1: SQL_DIST

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

```
SELECT
    MIN(forward.dist+backward.dist)
FROM forward,backward
WHERE
    forward.node = s AND
    backward.node = t AND
    forward.hub = backward.hub
```

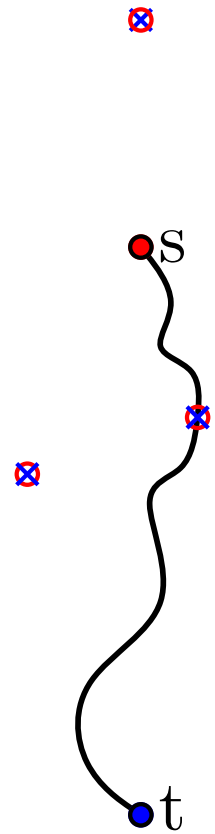


SQL Query

Algorithm 1: SQL_DIST

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

```
SELECT
    MIN(forward.dist+backward.dist)
FROM forward,backward
WHERE
    forward.node = s AND
    backward.node = t AND
    forward.hub = backward.hub
```

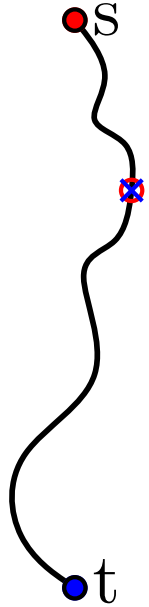


SQL Query

Algorithm 1: SQL_DIST

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

```
SELECT
    MIN(forward.dist+backward.dist)
FROM forward,backward
WHERE
    forward.node = s AND
    backward.node = t AND
    forward.hub = backward.hub
```

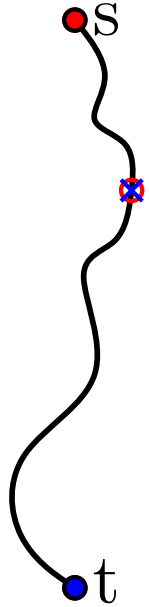


SQL Query

Algorithm 1: SQL_DIST

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

```
SELECT
    MIN(forward.dist+backward.dist)
FROM forward,backward
WHERE
    forward.node = s AND
    backward.node = t AND
    forward.hub = backward.hub
```



Bemerkung:

- berechnet nur die Distanz

Pfadentpackung

Idee:

- 2 Phasen
- speicher jeden Shortcut aus G^+ explizit (als Sequenz von Kanten IDs) in Tabelle `shortcuts`
- ca. 5 GB in Tabelle

Phase 1:

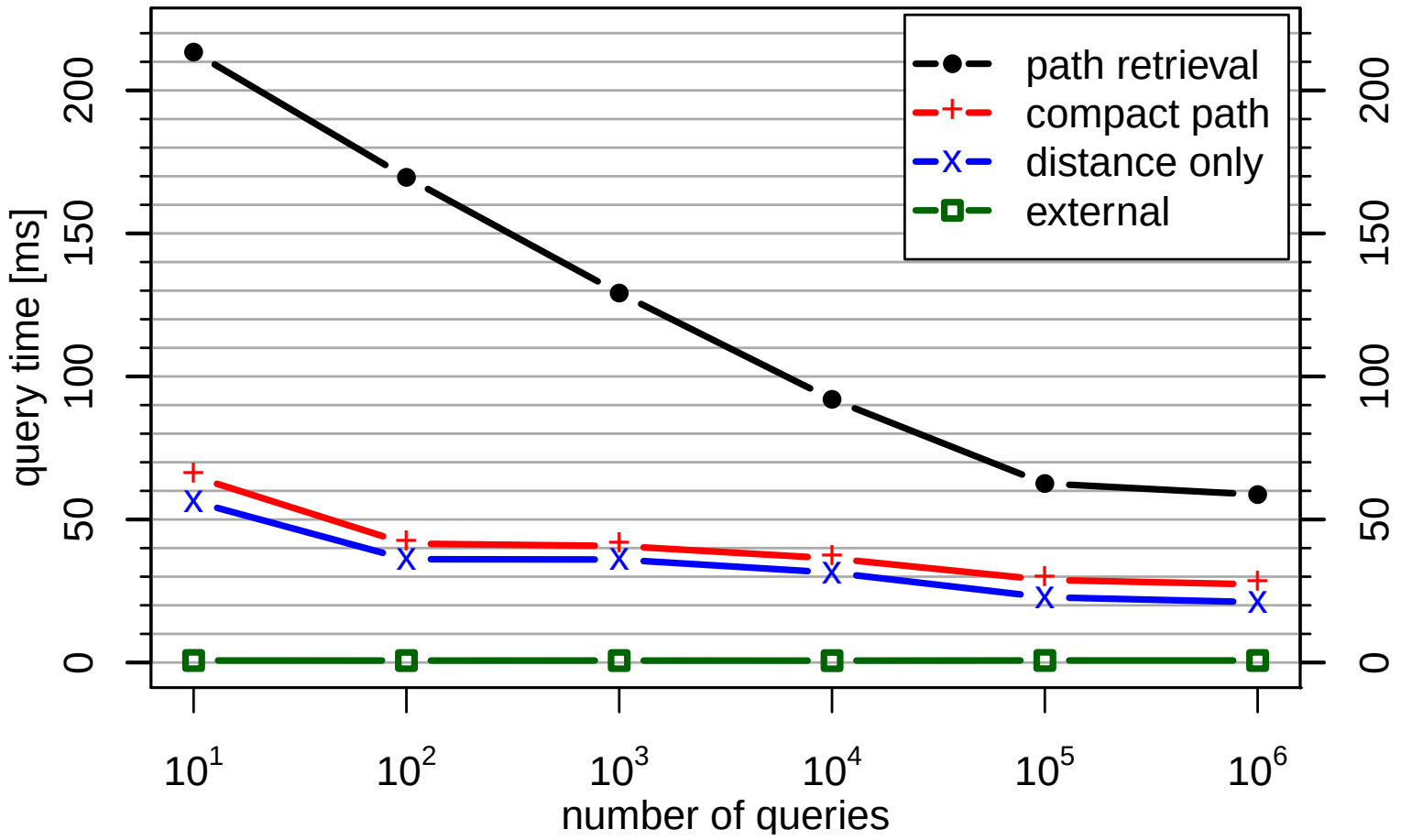
- erzeuge Pfad in G^+ durch Hubs auf dem Pfad
- erweitere Tabellen `forward` und `backward` um 2 Spalten: Parent und Shortcut
- erhöht Speicherverbrauch der Tabelle von 19 auf 32 GB

Phase 2:

- erzeuge Pfad in G durch matchen von G^+ mit `shortcuts`

Ergebnisse

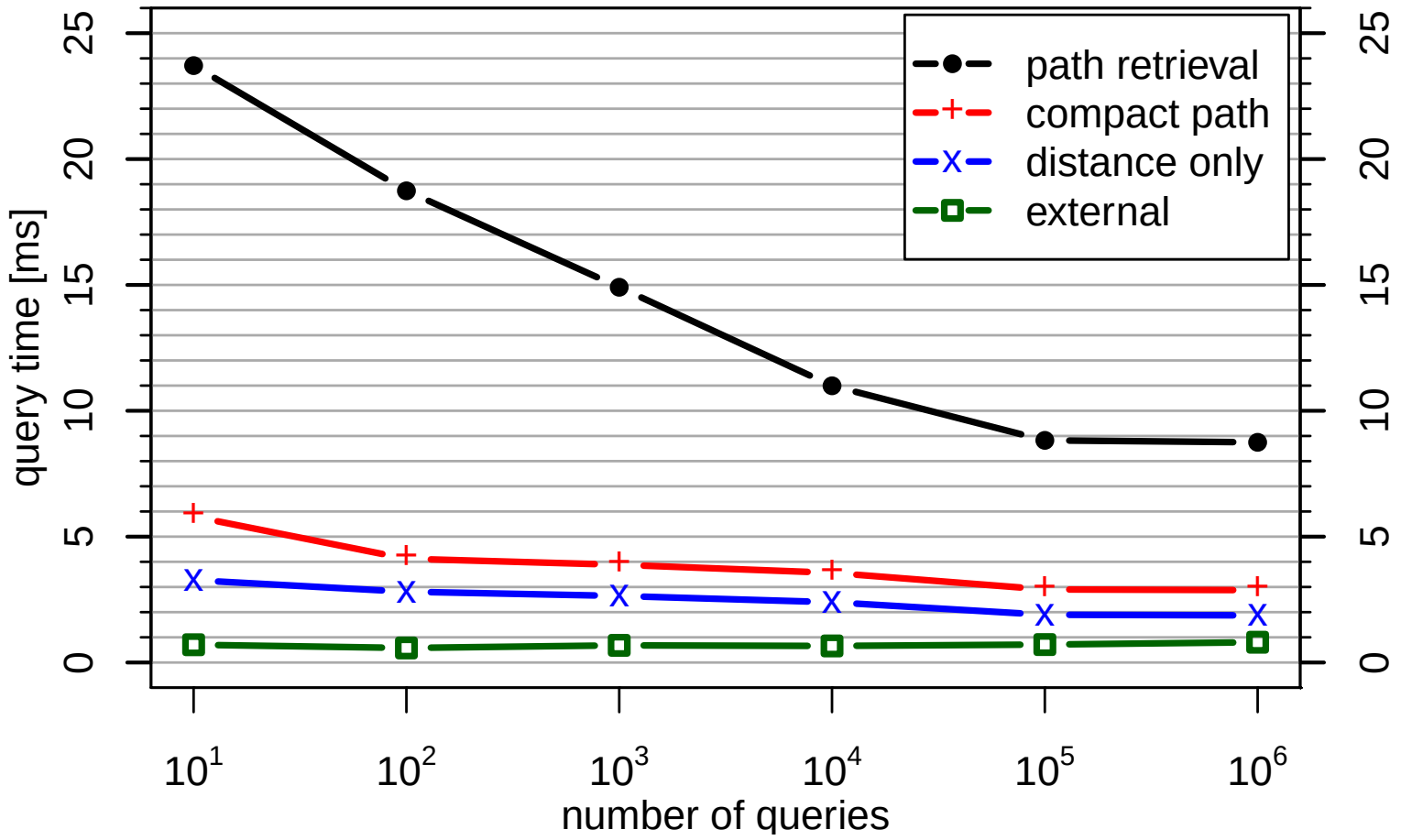
Setup: MS SQL Server 2008 R2 mit Daten auf HDD, kalter Cache



Beobachtung: Nicht schnell genug

Ergebnisse (SSD)

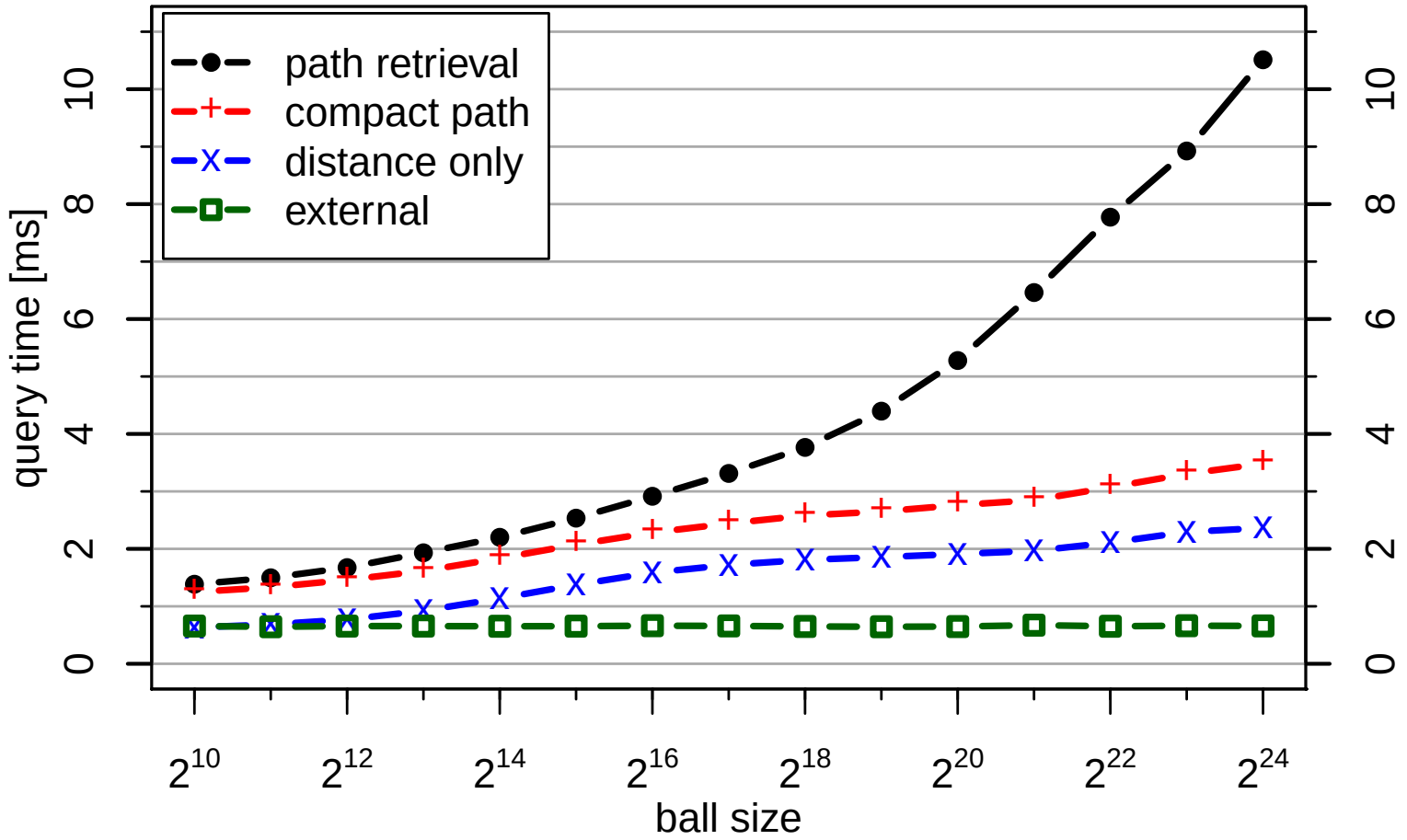
Setup: MS SQL Server 2008 R2 mit Daten auf SSD, kalter Cache



Beobachtung: SSD macht Queries schnell genug

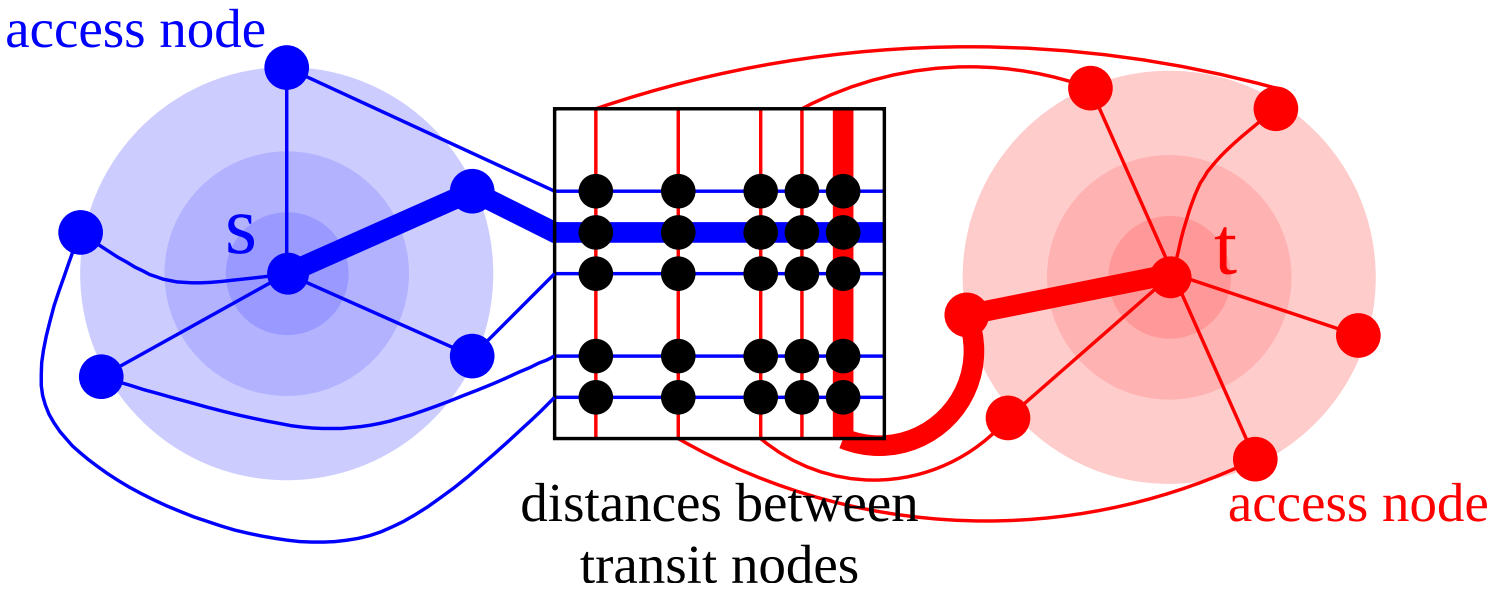
Lokale Anfrage

Setup: Anfragen mit verschiedenem Rank, 10000 Anfragen, kalter Cache



Beobachtung: praxisrelevante Anfragen sehr schnell

Transit-Node Routing



Transit-Node Routing

Beobachtung:

- wenn man weit weg fährt, fährt man immer an bestimmten Punkten vorbei
- hier: von Karlsruhe aus, an drei relevanten Stellen

Karlsruhe nach. . .
Kopenhagen

Transit-Node Routing

Beobachtung:

- wenn man weit weg fährt, fährt man immer an bestimmten Punkten vorbei
- hier: von Karlsruhe aus, an drei relevanten Stellen

Karlsruhe nach. . .
Berlin

Transit-Node Routing

Beobachtung:

- wenn man weit weg fährt, fährt man immer an bestimmten Punkten vorbei
- hier: von Karlsruhe aus, an drei relevanten Stellen

Karlsruhe nach. . .
Wien

Transit-Node Routing

Beobachtung:

- wenn man weit weg fährt, fährt man immer an bestimmten Punkten vorbei
- hier: von Karlsruhe aus, an drei relevanten Stellen

Karlsruhe nach. . .
München

Transit-Node Routing

Beobachtung:

- wenn man weit weg fährt, fährt man immer an bestimmten Punkten vorbei
- hier: von Karlsruhe aus, an drei relevanten Stellen

Karlsruhe nach. . .
Rom

Transit-Node Routing

Beobachtung:

- wenn man weit weg fährt, fährt man immer an bestimmten Punkten vorbei
- hier: von Karlsruhe aus, an drei relevanten Stellen

Karlsruhe nach. . .
Paris

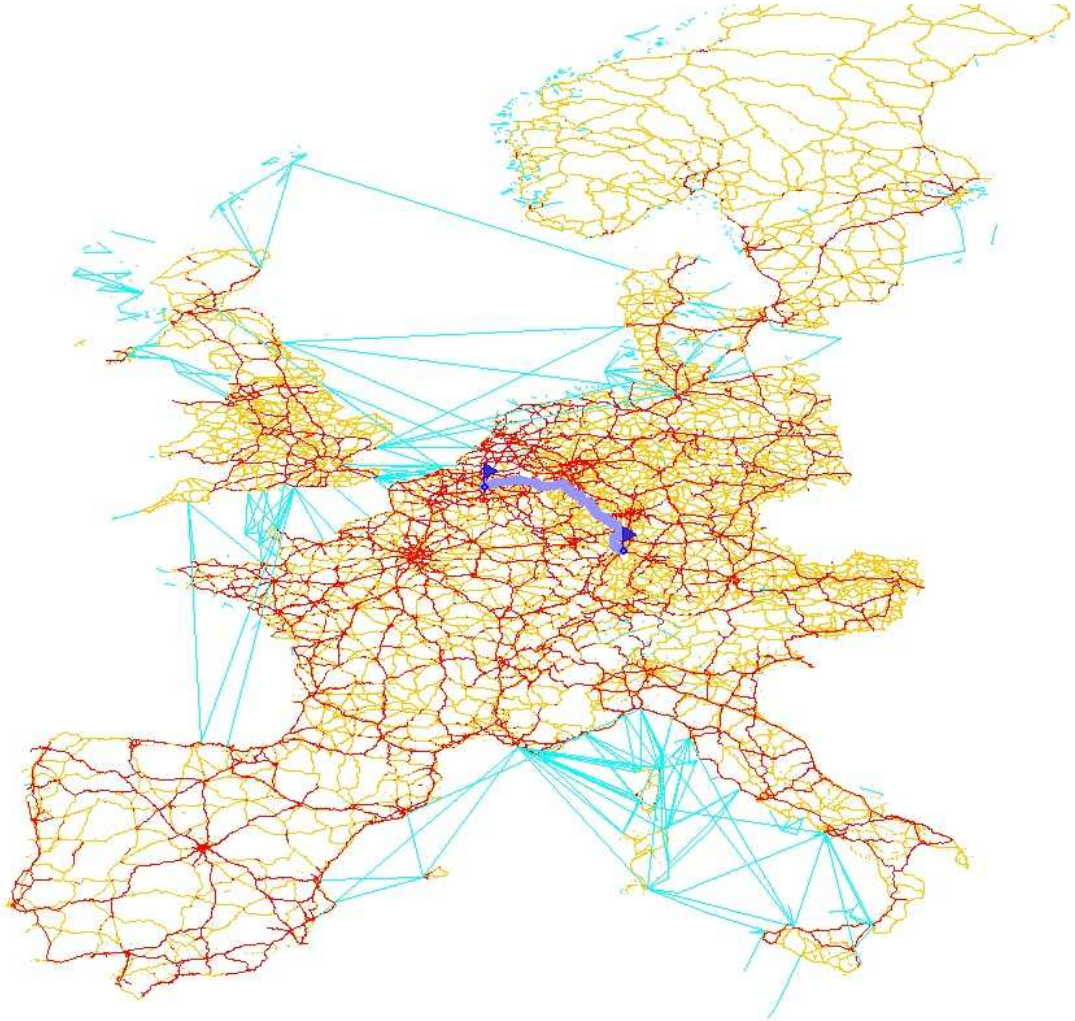
Transit-Node Routing

Beobachtung:

- wenn man weit weg fährt, fährt man immer an bestimmten Punkten vorbei
- hier: von Karlsruhe aus, an drei relevanten Stellen

Karlsruhe nach. . .
London

Transit-Node Routing

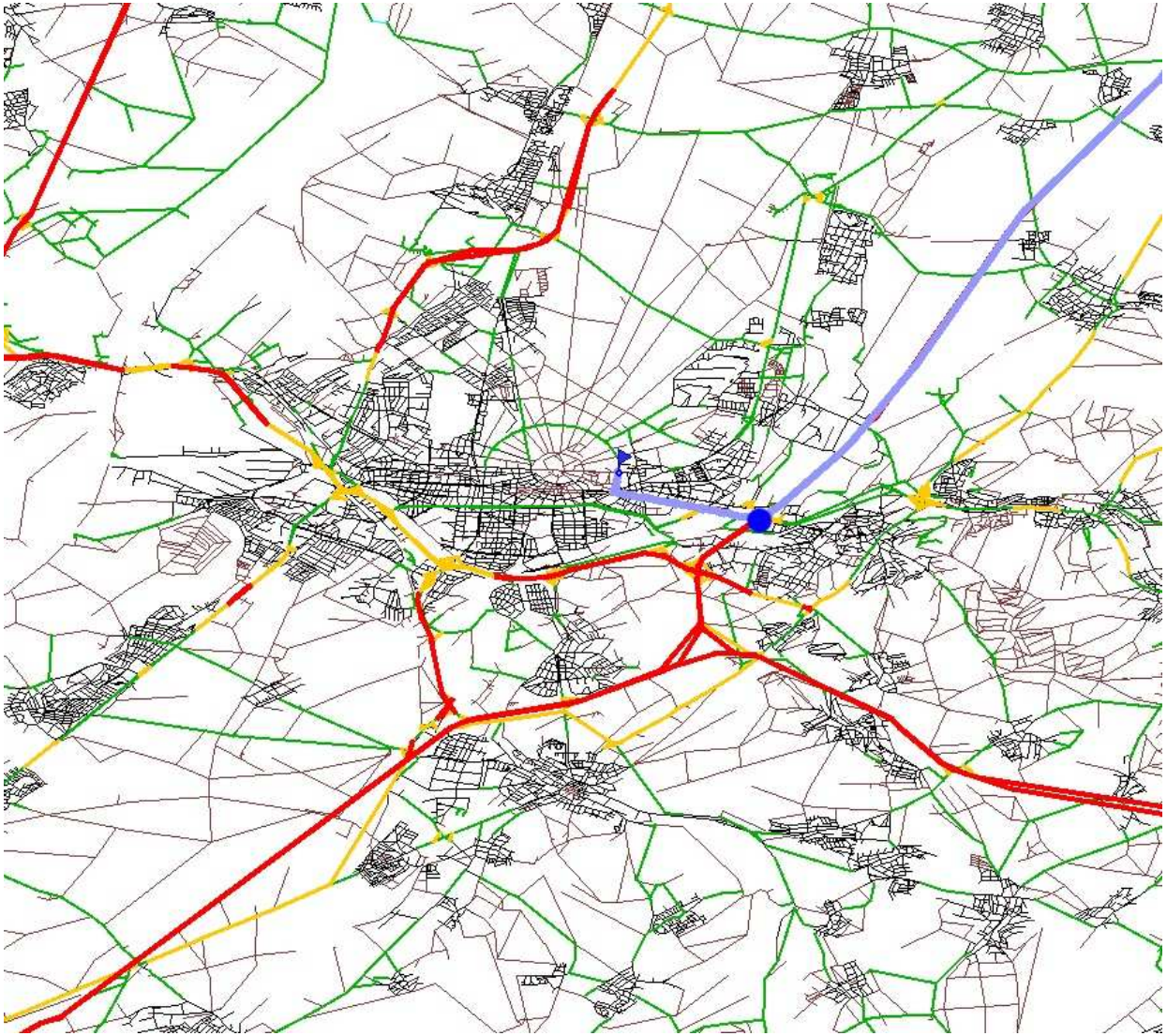


Beobachtung:

- wenn man weit weg fährt, fährt man immer an bestimmten Punkten vorbei
- hier: von Karlsruhe aus, an drei relevanten Stellen

Karlsruhe nach. . .
Brüssel

Transit-Node Routing

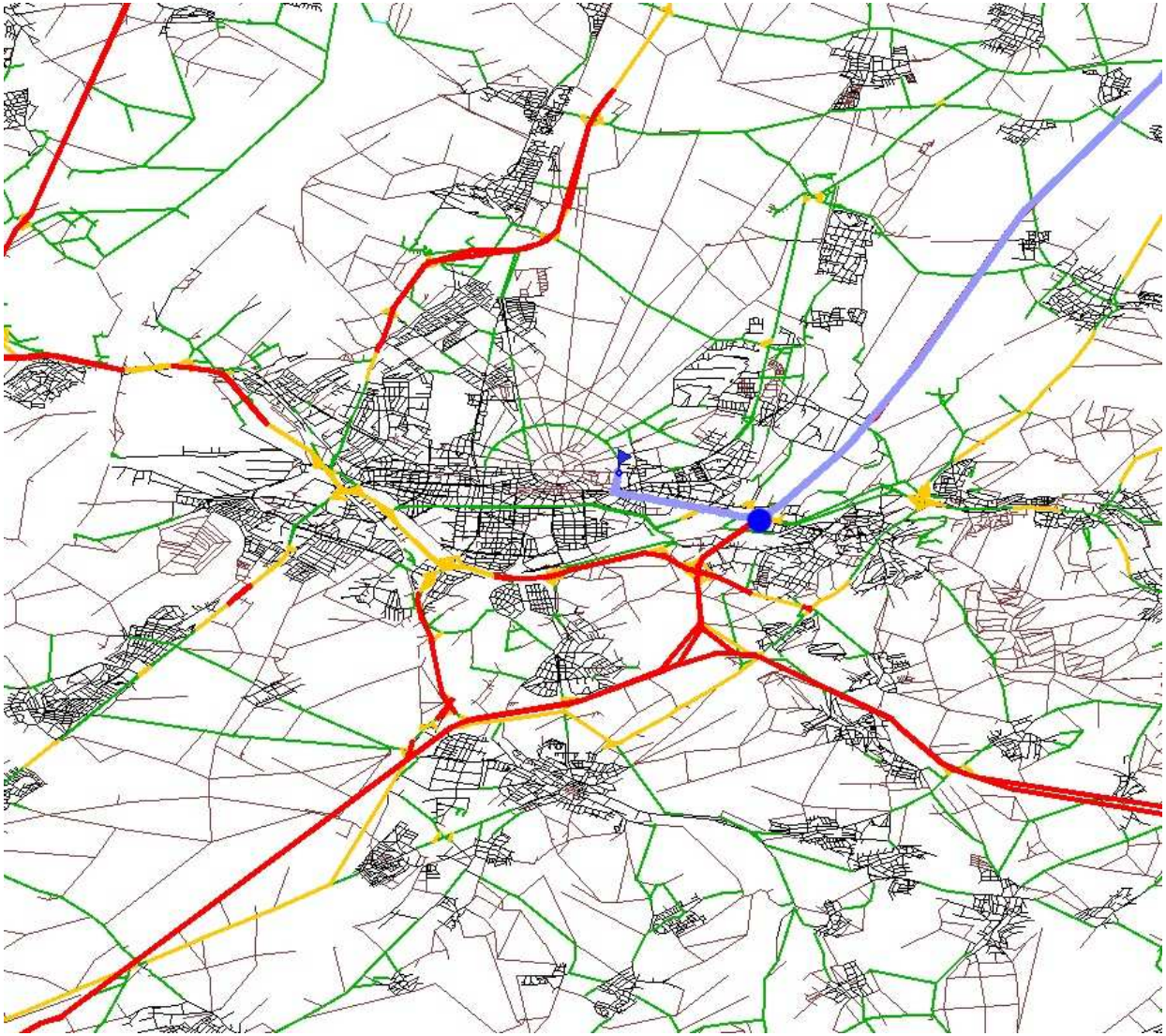


Beobachtung:

- wenn man weit weg fährt, fährt man immer an bestimmten Punkten vorbei
- hier: von Karlsruhe aus, an drei relevanten Stellen

Karlsruhe nach. . .
Kopenhagen

Transit-Node Routing

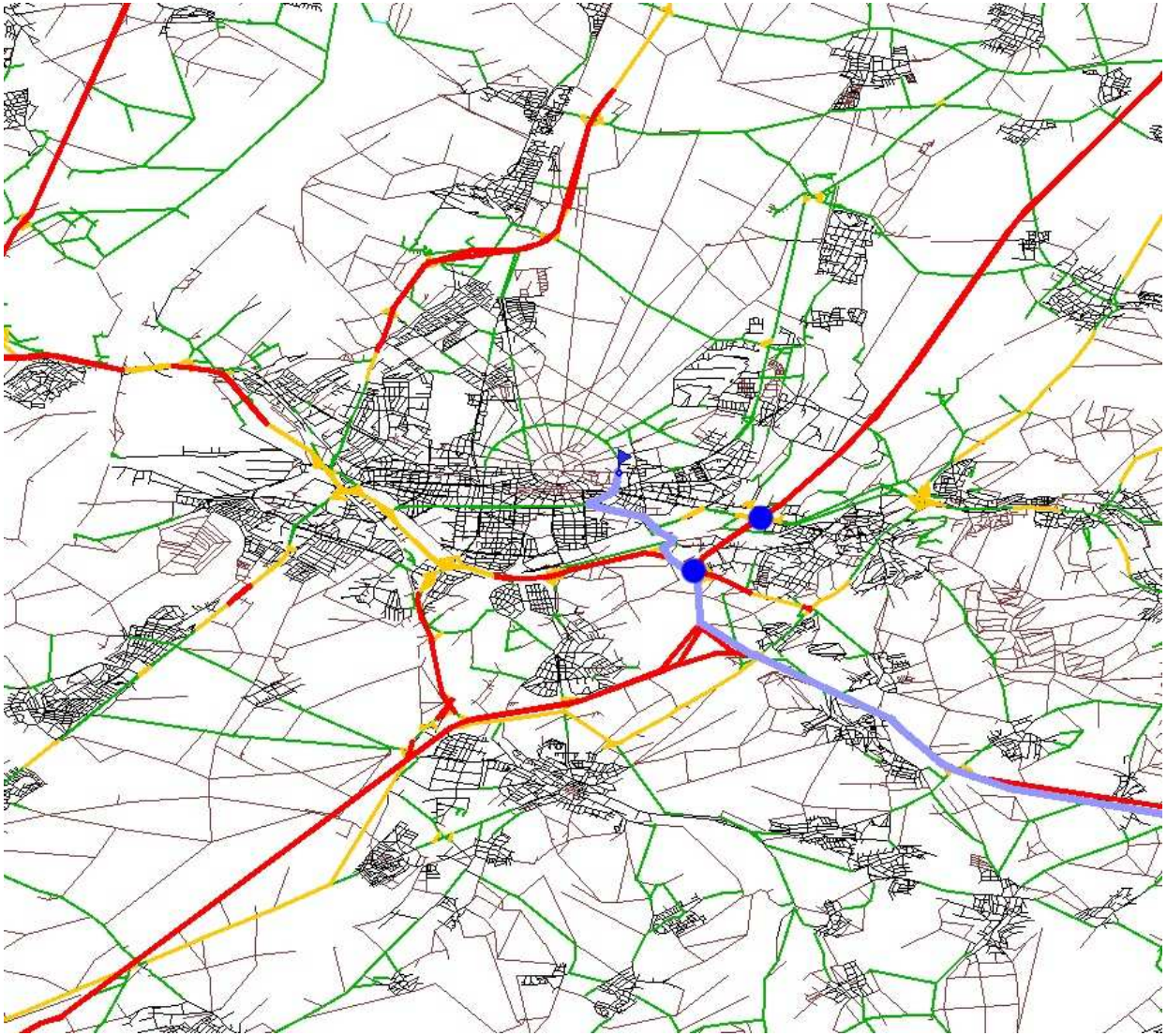


Beobachtung:

- wenn man weit weg fährt, fährt man immer an bestimmten Punkten vorbei
- hier: von Karlsruhe aus, an drei relevanten Stellen

Karlsruhe nach. . .
Berlin

Transit-Node Routing

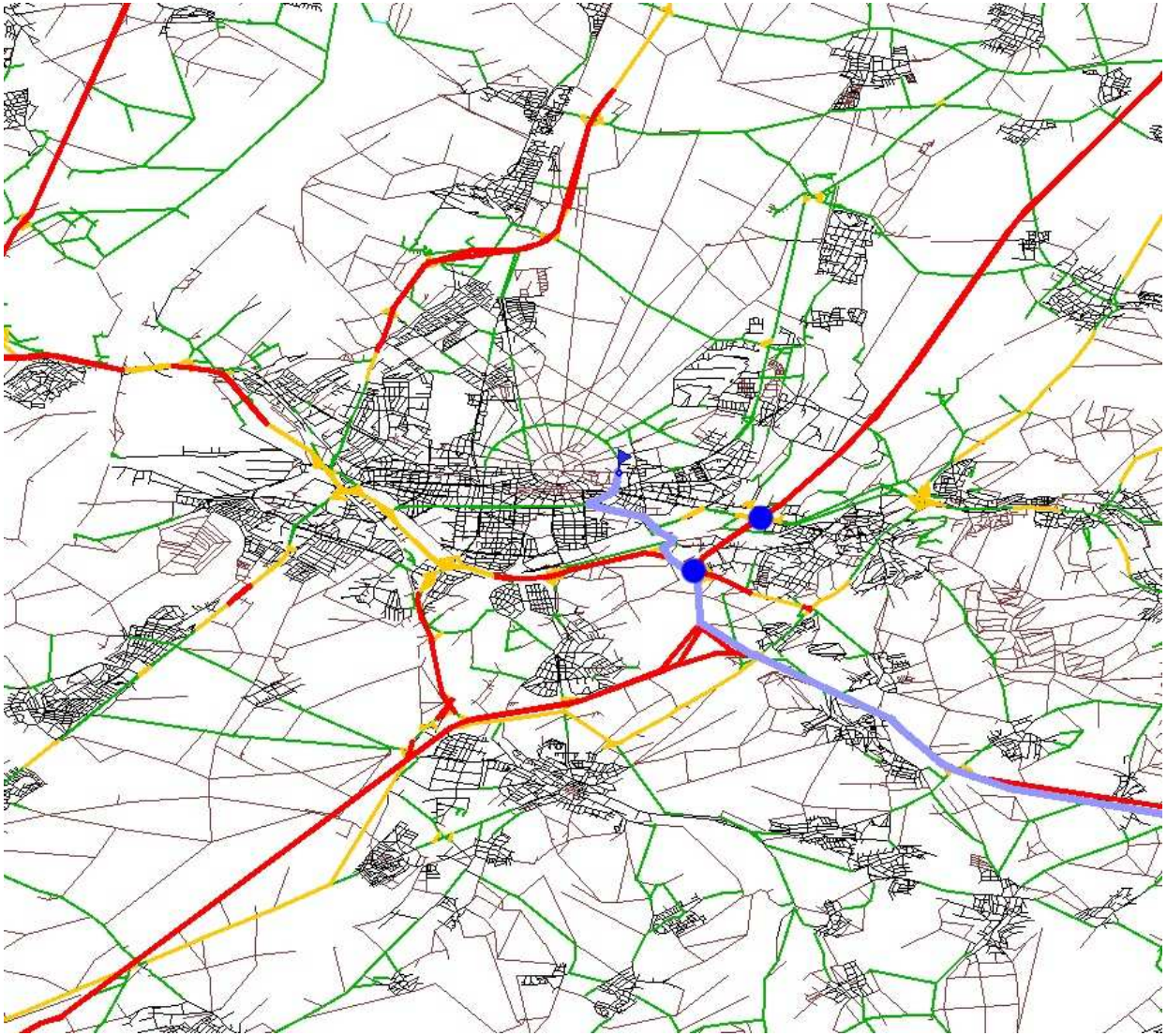


Beobachtung:

- wenn man weit weg fährt, fährt man immer an bestimmten Punkten vorbei
- hier: von Karlsruhe aus, an drei relevanten Stellen

Karlsruhe nach. . .
Wien

Transit-Node Routing

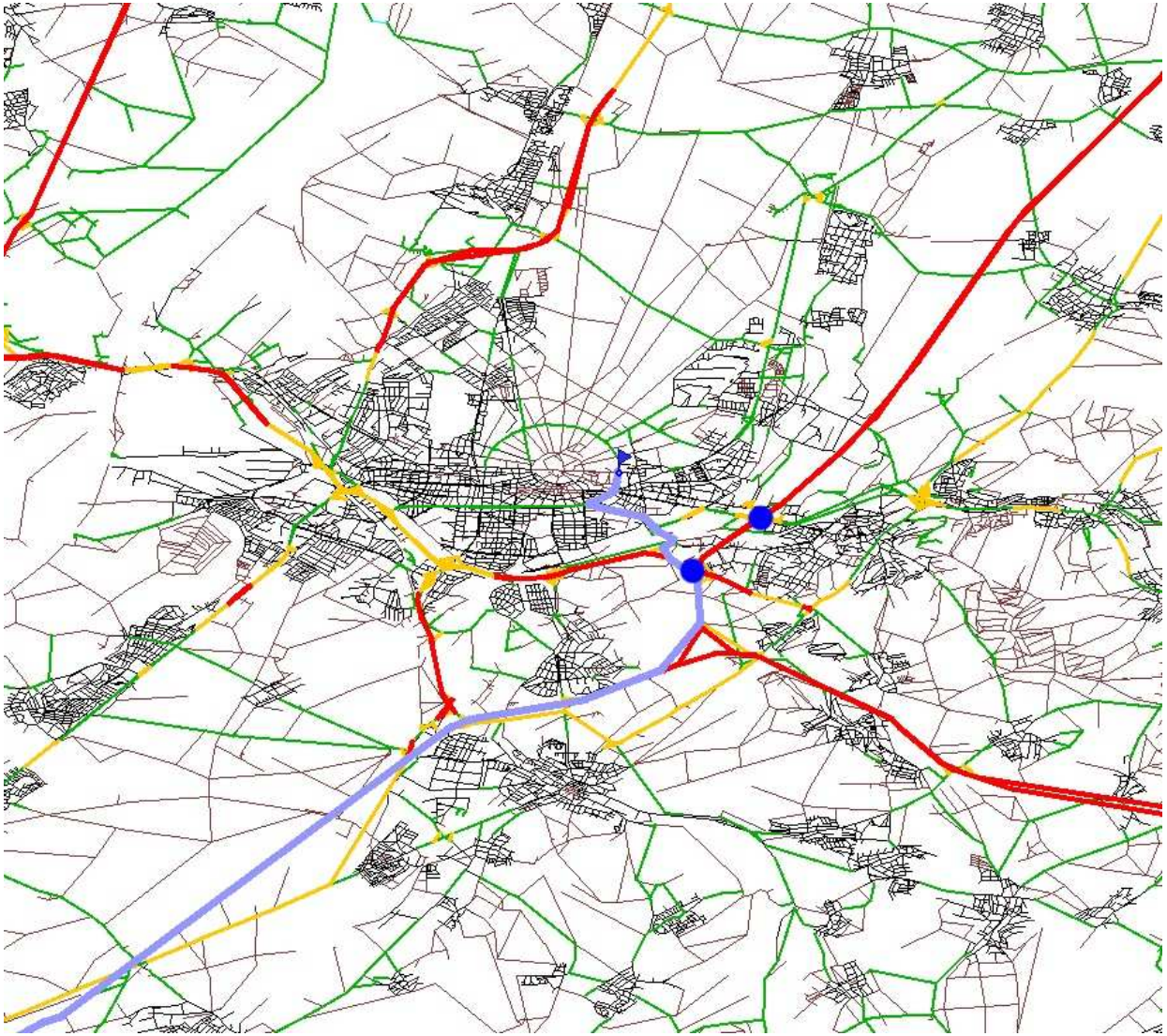


Beobachtung:

- wenn man weit weg fährt, fährt man immer an bestimmten Punkten vorbei
- hier: von Karlsruhe aus, an drei relevanten Stellen

Karlsruhe nach. . .
München

Transit-Node Routing

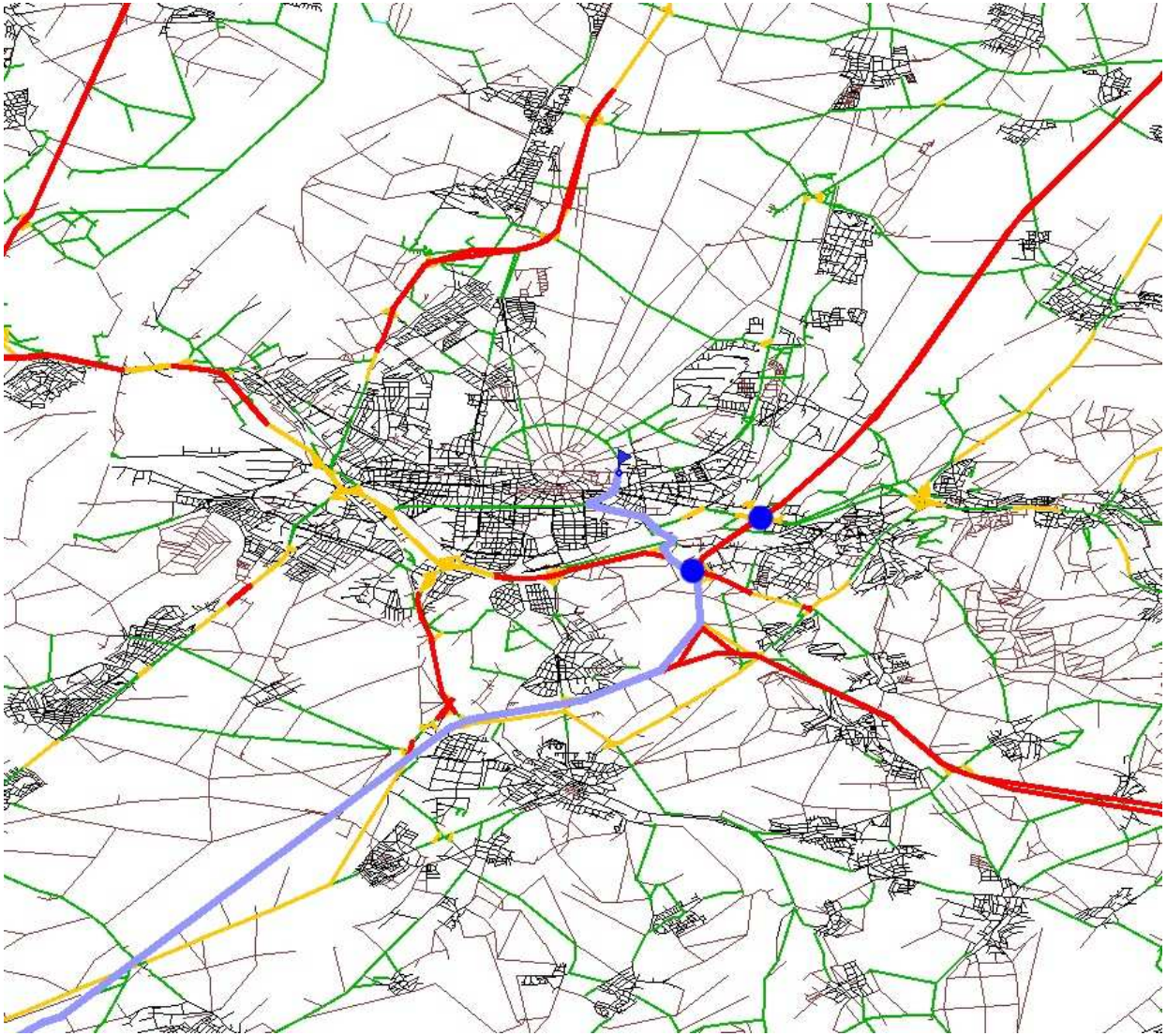


Beobachtung:

- wenn man weit weg fährt, fährt man immer an bestimmten Punkten vorbei
- hier: von Karlsruhe aus, an drei relevanten Stellen

Karlsruhe nach. . .
Rom

Transit-Node Routing

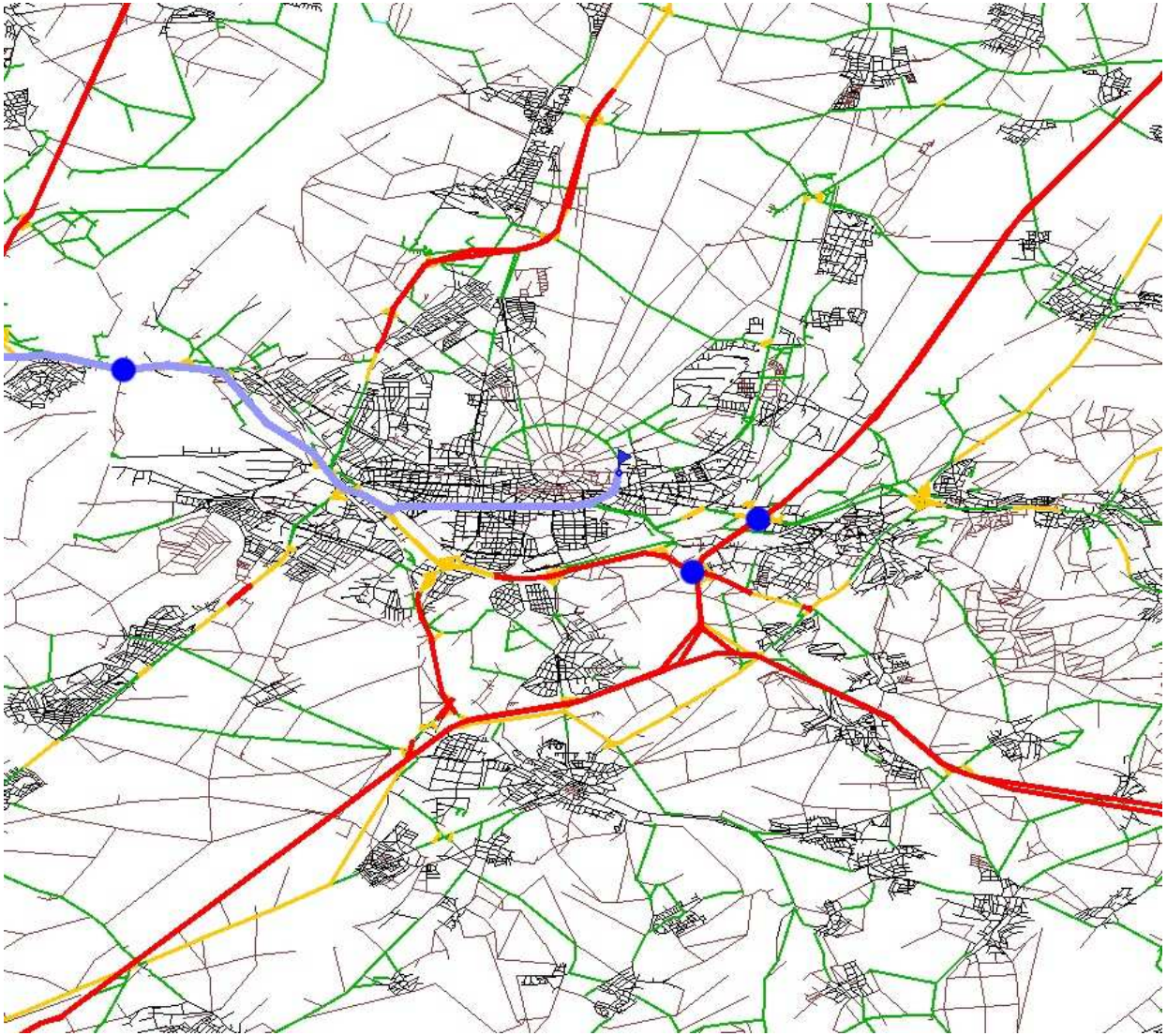


Beobachtung:

- wenn man weit weg fährt, fährt man immer an bestimmten Punkten vorbei
- hier: von Karlsruhe aus, an drei relevanten Stellen

Karlsruhe nach. . .
Paris

Transit-Node Routing

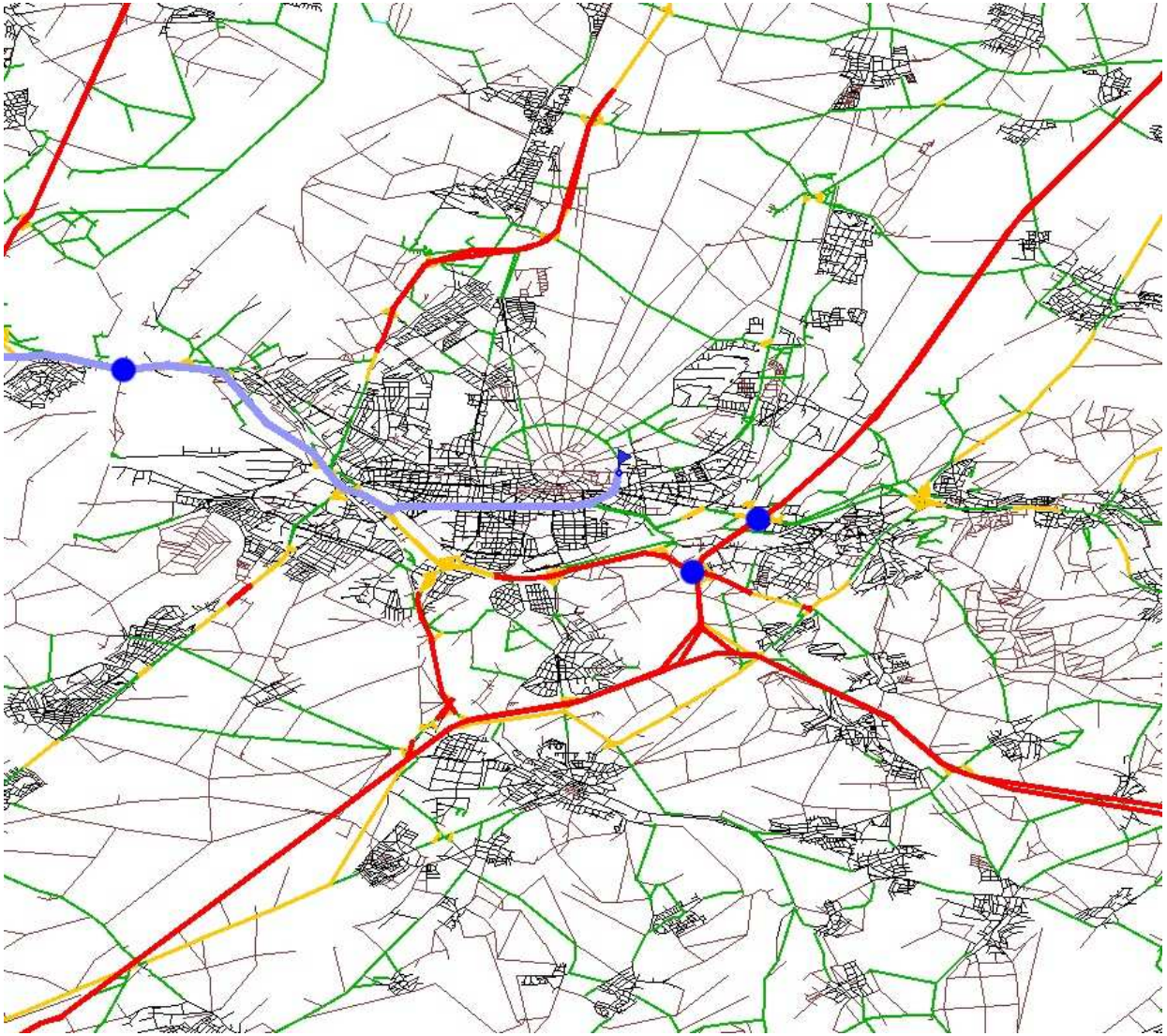


Beobachtung:

- wenn man weit weg fährt, fährt man immer an bestimmten Punkten vorbei
- hier: von Karlsruhe aus, an drei relevanten Stellen

Karlsruhe nach. . .
London

Transit-Node Routing



Beobachtung:

- wenn man weit weg fährt, fährt man immer an bestimmten Punkten vorbei
- hier: von Karlsruhe aus, an drei relevanten Stellen

Karlsruhe nach. . .
Brüssel

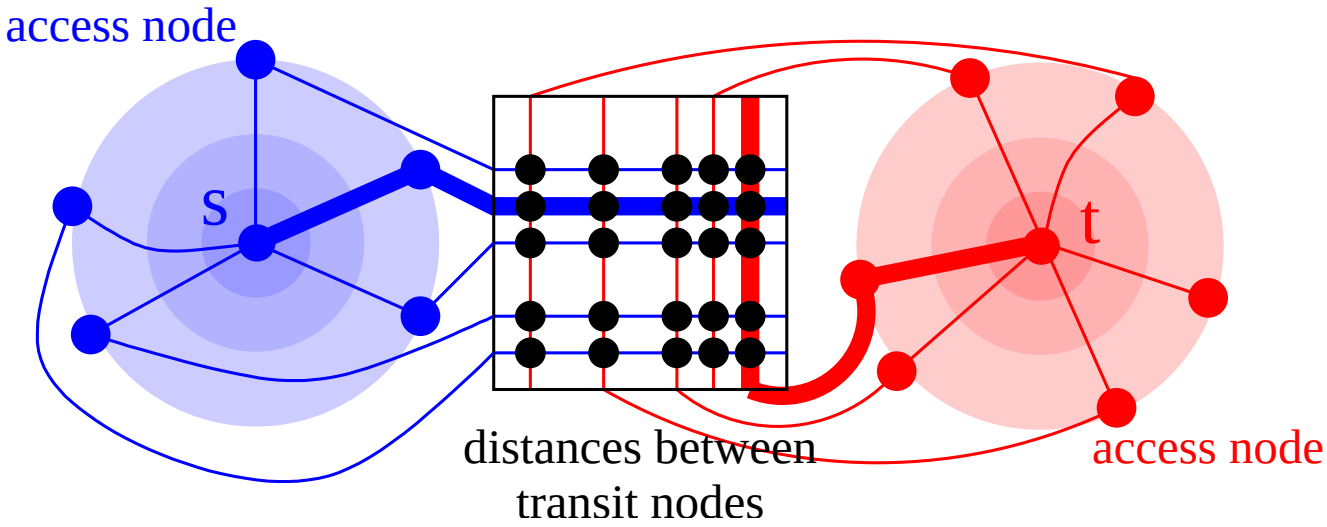
Transit-Node Routing

Idee:

- reduziere Anfragen auf Zugriffe in eine quadratische Tabellen
- identifiziere “wichtige” Knoten
- vollständige Distanztabellen zwischen diesen Knoten

Probleme:

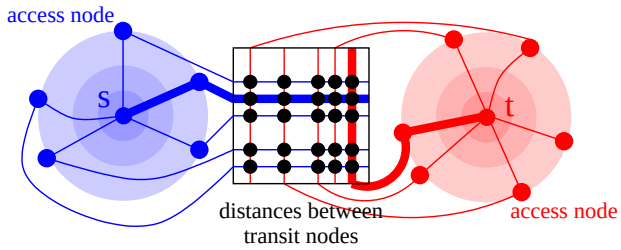
- Speicherverbrauch
- nahe Anfragen



Generelles TNR-Framework

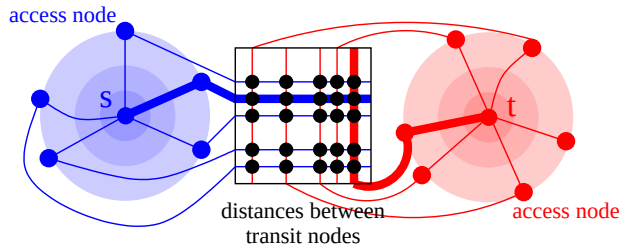
- Wähle Transit-Knoten: $\mathcal{T} \subseteq V$
- Bestimme für jeden Knoten v eine Menge von Vorwärts $\overrightarrow{A}(v)$ und Rückwärts $\overleftarrow{A}(v)$ Access-Knoten
- Vorberechnete Distanzen: $D_{\mathcal{T}}$ und d_A

Generelles TNR-Framework



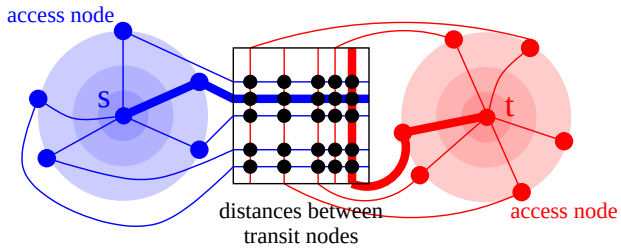
- Wähle Transit-Knoten: $\mathcal{T} \subseteq V$
- Bestimme für jeden Knoten v eine Menge von Vorwärts $\overrightarrow{A}(v)$ und Rückwärts $\overleftarrow{A}(v)$ Access-Knoten
- Vorberechnete Distanzen: $D_{\mathcal{T}}$ und d_A

Generelles TNR-Framework



- Wähle Transit-Knoten: $\mathcal{T} \subseteq V$
- Bestimme für jeden Knoten v eine Menge von Vorwärts $\overrightarrow{A}(v)$ und Rückwärts $\overleftarrow{A}(v)$ Access-Knoten
- Vorberechnete Distanzen: $D_{\mathcal{T}}$ und d_A
- $\text{dist}(s,t) \stackrel{?}{=} \min_{u \in \overrightarrow{A}(s), v \in \overleftarrow{A}(t)} \{d_A(s,u) + D_{\mathcal{T}}(u,v) + d_A(v,t)\}$

Generelles TNR-Framework



- Wähle Transit-Knoten: $\mathcal{T} \subseteq V$
- Bestimme für jeden Knoten v eine Menge von Vorwärts $\overrightarrow{A}(v)$ und Rückwärts $\overleftarrow{A}(v)$ Access-Knoten
- Vorberechnete Distanzen: $D_{\mathcal{T}}$ und d_A
- $\text{dist}(s,t) \stackrel{?}{=} \min_{u \in \overrightarrow{A}(s), v \in \overleftarrow{A}(t)} \{d_A(s,u) + D_{\mathcal{T}}(u,v) + d_A(v,t)\}$

Berechnete Distanz nur für hinreichend weite Anfragen korrekt

- Locality filter: $L : V \times V \to \{\text{true}, \text{false}\}$
- true $\longrightarrow$ Fallback-Routine für lokale Anfragen
- Einseitige Fehler erlaubt

Zutaten TNR-Verfahren

Also:

- Wie Transit-Knoten bestimmen?
- Wie Access-Knoten und deren Distanz bestimmen?
- Wie Distanztabelle zwischen Transit-Knoten berechnen?
- Welcher Lokalitätsfilter?
- Wie lokale Anfragen berechnen?

Zutaten TNR-Verfahren

Also:

- Wie Transit-Knoten bestimmen?
- Wie Access-Knoten und deren Distanz bestimmen?
- Wie Distanztabelle zwischen Transit-Knoten berechnen?
- Welcher Lokalitätsfilter?
- Wie lokale Anfragen berechnen?

Ideen?

Zutaten TNR-Verfahren

Also:

- Wie Transit-Knoten bestimmen?
- Wie Access-Knoten und deren Distanz bestimmen?
- Wie Distanztabelle zwischen Transit-Knoten berechnen?
- Welcher Lokalitätsfilter?
- Wie lokale Anfragen berechnen?

Ideen? Verschiedene Ansätze: Grid-based TNR [BFM06],

Hierarchie-basiertes TNR mit geometrischem Lokalitätsfilter [BFM+07, GSSV12], CH-TNR [ALS13]

CH-TNR

Transit Node Routing aufbauend auf CH

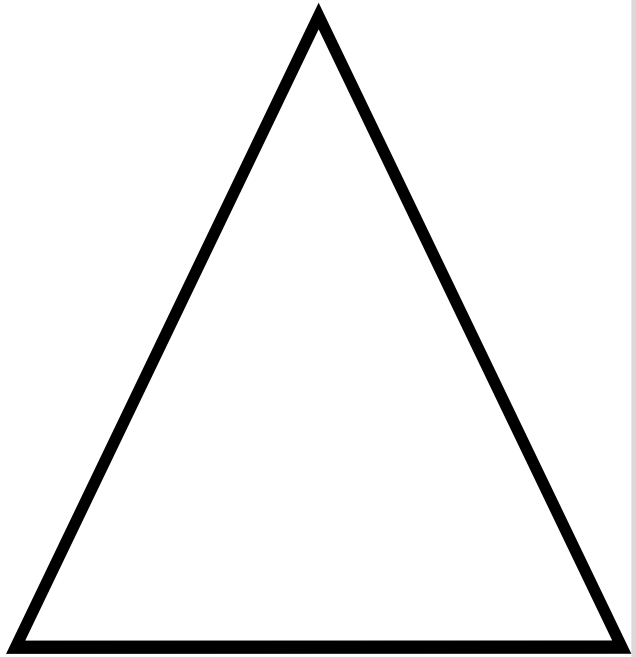
- CH für Vorberechnung und lokale Anfrage
- Top-k Knoten sind Transit-Knoten...

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

CH-TNR

Transit Node Routing aufbauend auf CH

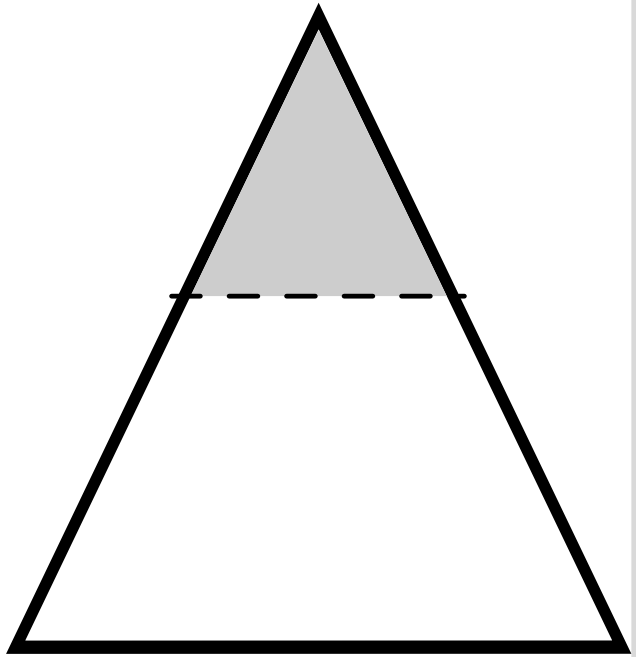
- CH für Vorberechnung und lokale Anfrage
- Top-k Knoten sind Transit-Knoten...



CH-TNR

Transit Node Routing aufbauend auf CH

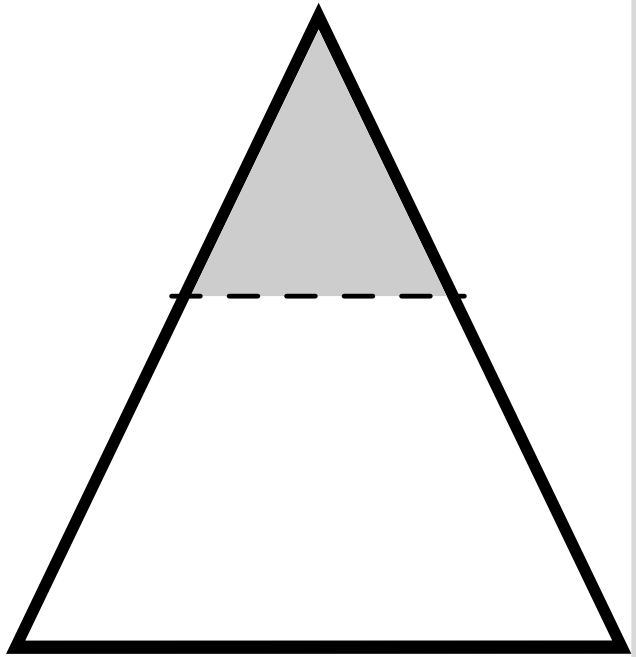
- CH für Vorberechnung und lokale Anfrage
- Top-k Knoten sind Transit-Knoten...



CH-TNR

Transit Node Routing aufbauend auf CH

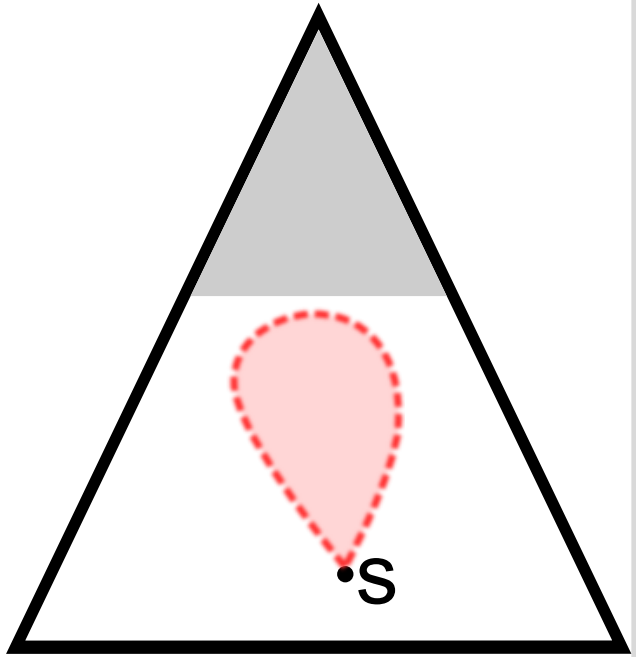
- CH für Vorberechnung und lokale Anfrage
- Top-k Knoten sind Transit-Knoten...
- ...und damit die Access-Nodes berechnen



CH-TNR

Transit Node Routing aufbauend auf CH

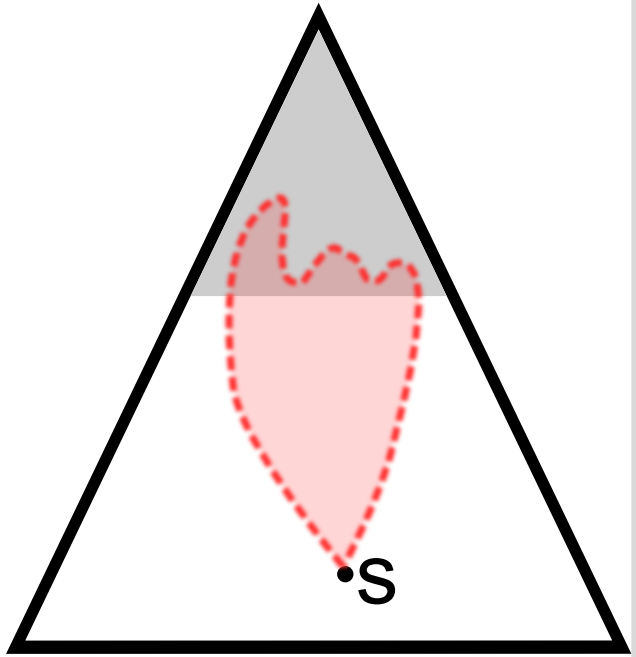
- CH für Vorberechnung und lokale Anfrage
- Top-k Knoten sind Transit-Knoten...
- ...und damit die Access-Nodes berechnen



CH-TNR

Transit Node Routing aufbauend auf CH

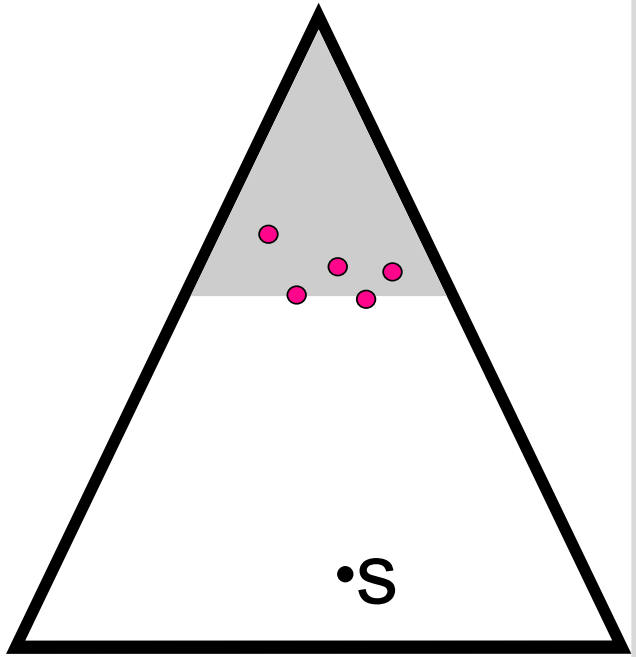
- CH für Vorberechnung und lokale Anfrage
- Top-k Knoten sind Transit-Knoten...
- ...und damit die Access-Nodes berechnen



CH-TNR

Transit Node Routing aufbauend auf CH

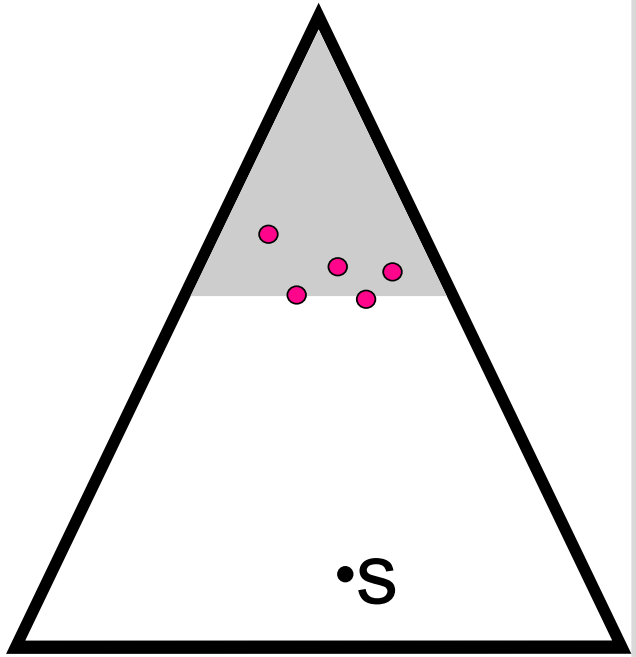
- CH für Vorberechnung und lokale Anfrage
- Top-k Knoten sind Transit-Knoten...
- ...und damit die Access-Nodes berechnen



CH-TNR

Transit Node Routing aufbauend auf CH

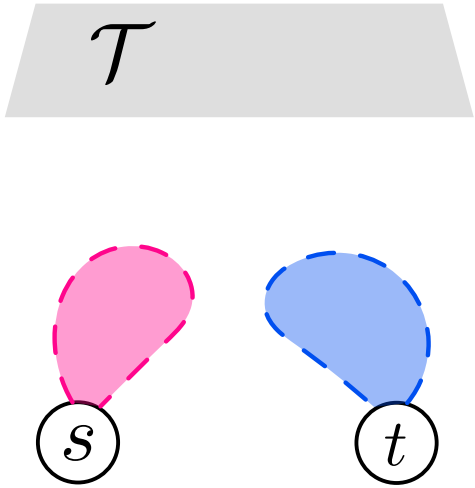
- CH für Vorberechnung und lokale Anfrage
- Top-k Knoten sind Transit-Knoten...
- ...und damit die Access-Nodes berechnen
- Lokalitätsfilter!?

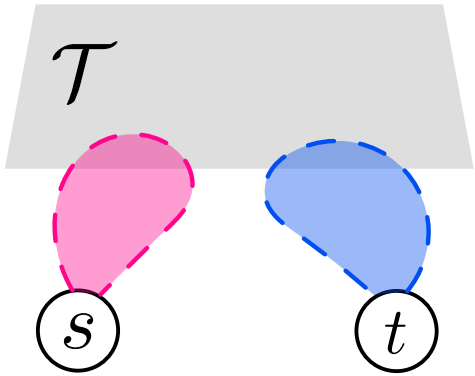


CH-basiertes TNR

Eigenschaften einer lokalen Anfrage

- Betrachte den höchsten Knoten *m* auf einem kürzesten CH hoch-runter *st*-Pfad

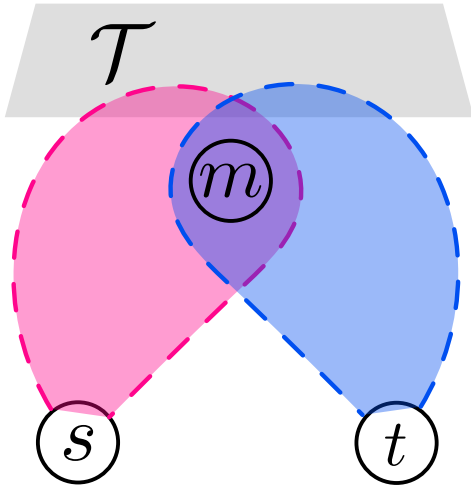


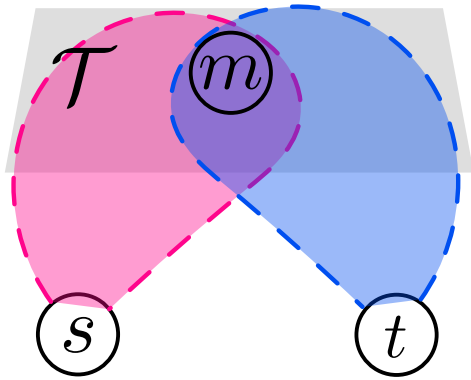


CH-basiertes TNR

Eigenschaften einer lokalen Anfrage

- Betrachte den höchsten Knoten *m* auf einem kürzesten CH hoch-runter *st*-Pfad

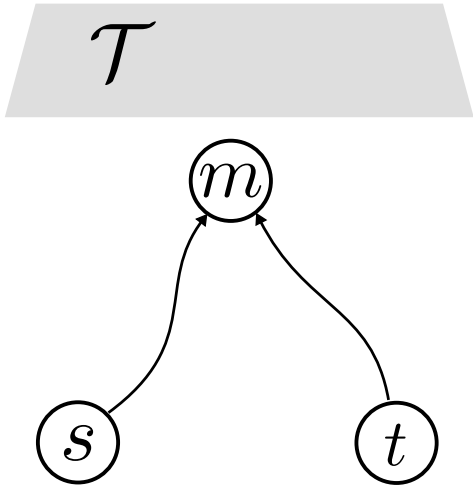


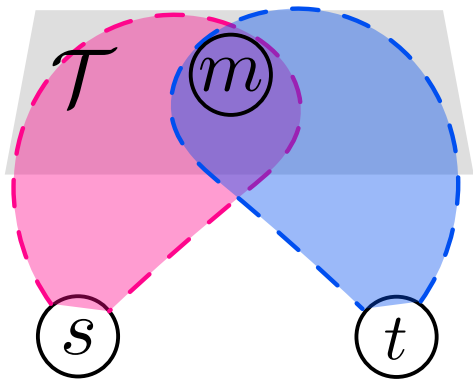


CH-basiertes TNR

Eigenschaften einer lokalen Anfrage

- Betrachte den höchsten Knoten *m* auf einem kürzesten CH hoch-runter *st*-Pfad

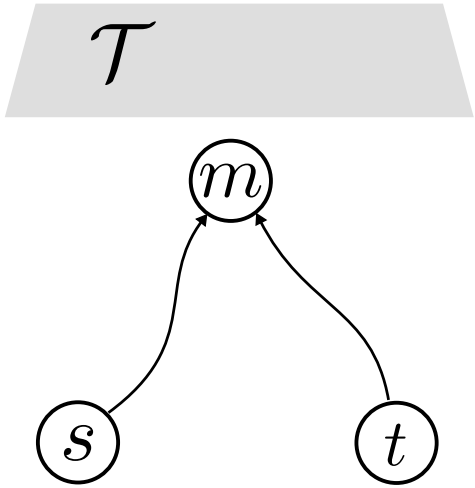


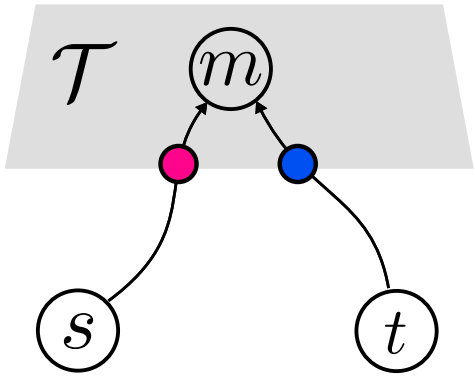


CH-basiertes TNR

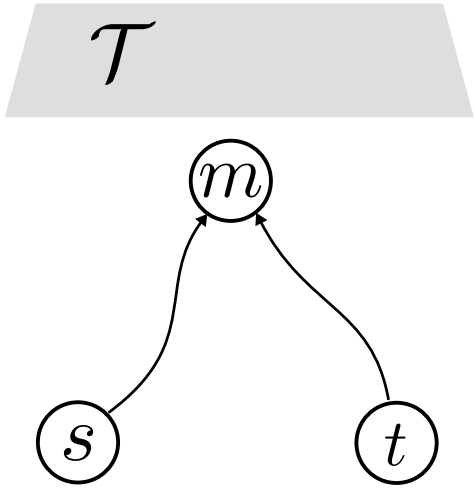
Eigenschaften einer lokalen Anfrage

- Betrachte den höchsten Knoten m auf einem kürzesten CH hoch-runter st-Pfad
- $m \notin \mathcal{T} \iff$ lokale Anfrage





CH-basiertes TNR

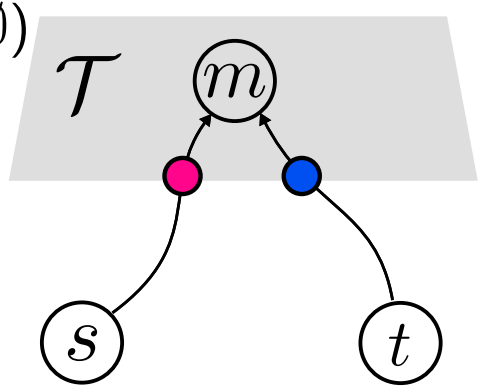


Eigenschaften einer lokalen Anfrage

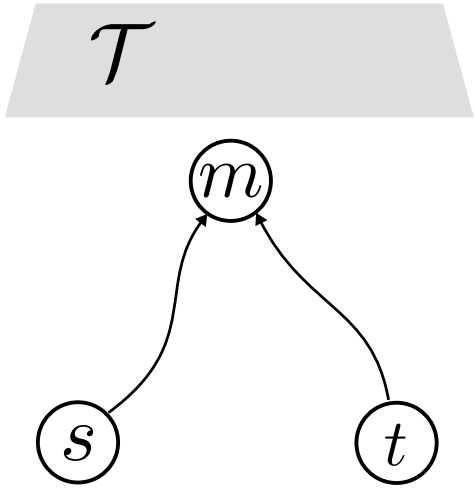
- Betrachte den höchsten Knoten m auf einem kürzesten CH hoch-runter st-Pfad
- $m \notin \mathcal{T} \iff$ lokale Anfrage

Suchraum-basierter Lokalitätsfilter

- Speichere Suchraum unterhalb der Transit-Knoten $S : V \to V \setminus \mathcal{T}$ explizit
- Fällt bei der Access-Knoten-Berechnung als Beiprodukt ab
- Während der Anfrage: $\mathcal{L} = (S(s) \cap S(t) \neq \emptyset)$
- Braucht viel Speicher!



CH-basiertes TNR

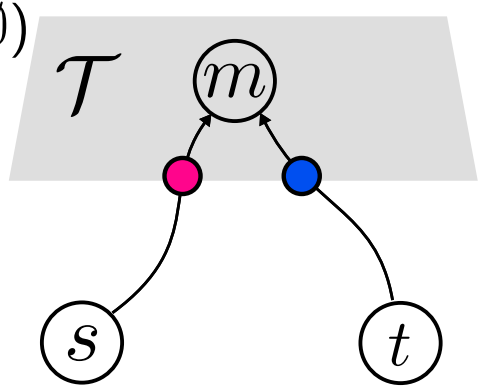


Eigenschaften einer lokalen Anfrage

- Betrachte den höchsten Knoten m auf einem kürzesten CH hoch-runter st-Pfad
- $m \notin \mathcal{T} \iff$ lokale Anfrage

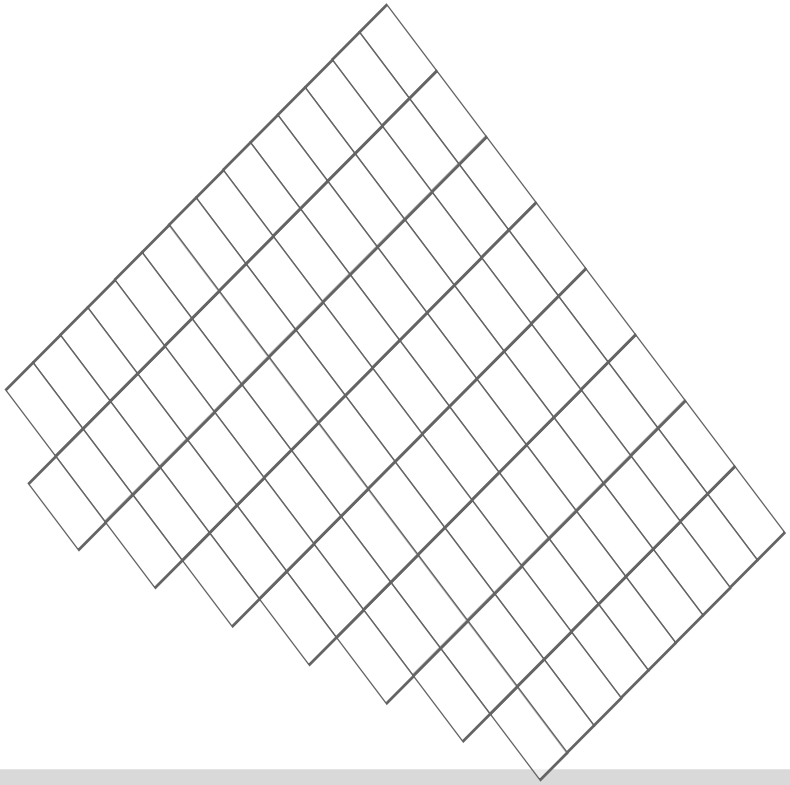
Suchraum-basierter Lokalitätsfilter

- Speichere Suchraum unterhalb der Transit-Knoten $S : V \to V \setminus \mathcal{T}$ explizit
- Fällt bei der Access-Knoten-Berechnung als Beiprodukt ab
- Während der Anfrage: $\mathcal{L} = (S(s) \cap S(t) \neq \emptyset)$
- Braucht viel Speicher!
- Einseitiger Fehler erlaubt



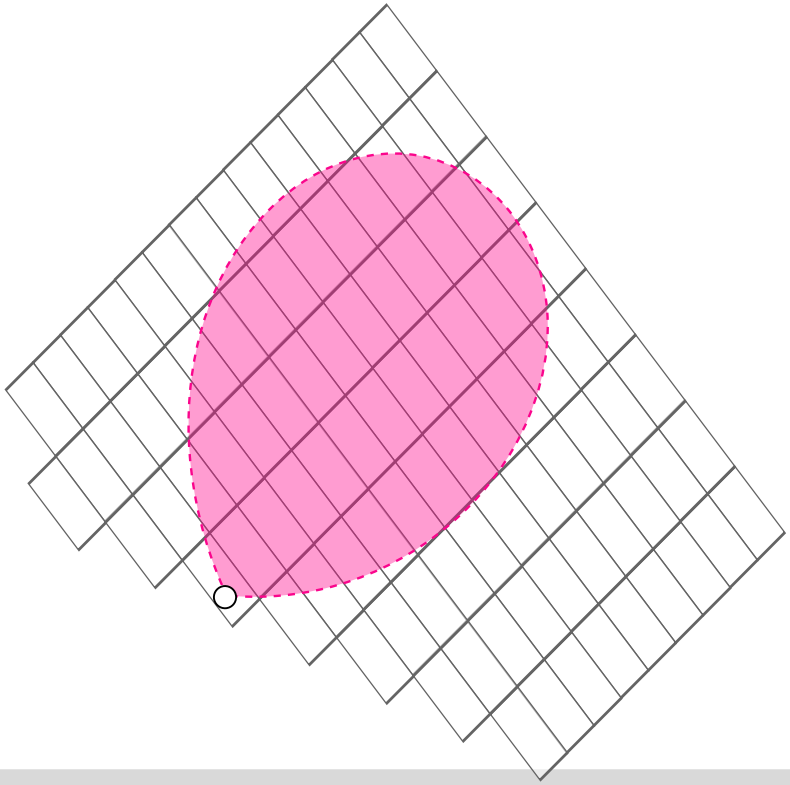
Lokalitätsfilter

- Partitioniere Graphen in Regionen
- Überapproximation des Suchraum mittels berührter Regionen.
- Wenn x im Suchraum $S(s)$ ist, dann ist die Region $R(x)$ im approximierten Suchraum $S'(s)$



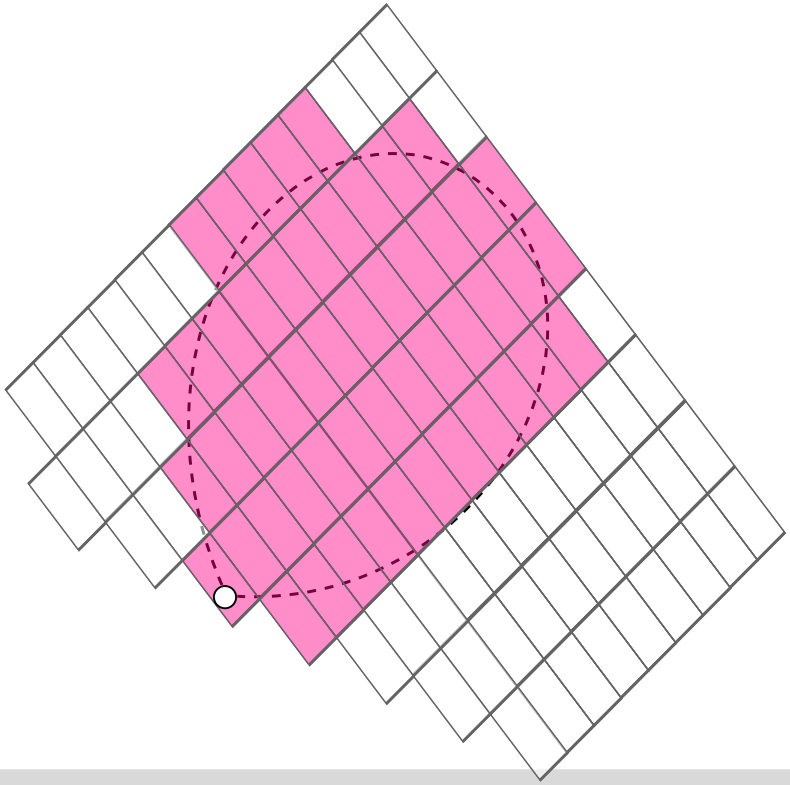
Lokalitätsfilter

- Partitioniere Graphen in Regionen
- Überapproximation des Suchraum mittels berührter Regionen.
- Wenn x im Suchraum $S(s)$ ist, dann ist die Region $R(x)$ im approximierten Suchraum $S'(s)$



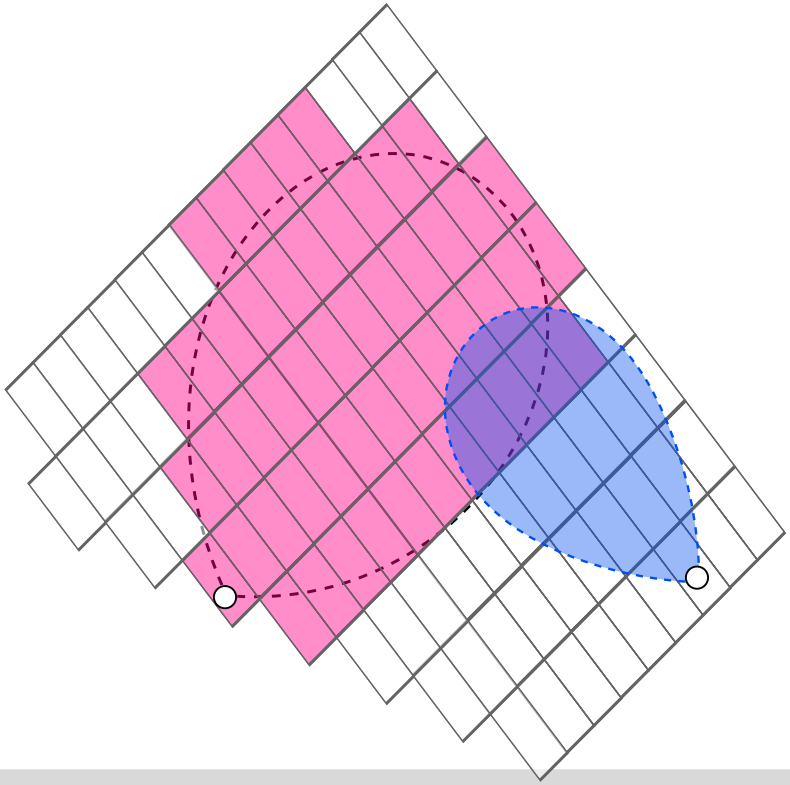
Lokalitätsfilter

- Partitioniere Graphen in Regionen
- Überapproximation des Suchraum mittels berührter Regionen.
- Wenn x im Suchraum $S(s)$ ist, dann ist die Region $R(x)$ im approximierten Suchraum $S'(s)$



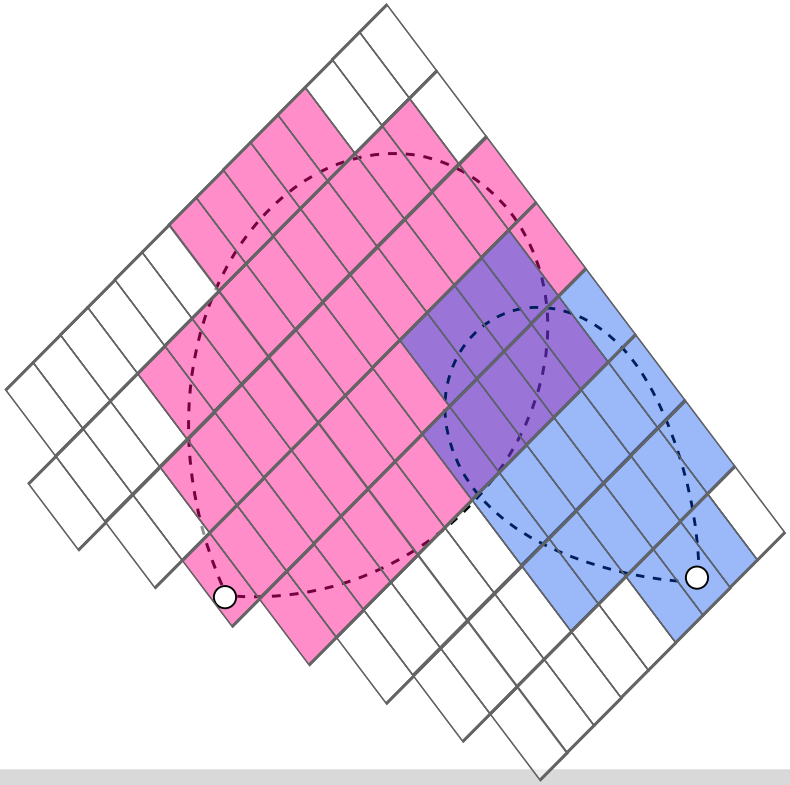
Lokalitätsfilter

- Partitioniere Graphen in Regionen
- Überapproximation des Suchraum mittels berührter Regionen.
- Wenn x im Suchraum $S(s)$ ist, dann ist die Region $R(x)$ im approximierten Suchraum $S'(s)$
- $m \in S(s) \cap S(t) \Longrightarrow R(m) \in S'(s) \cap S'(t)$



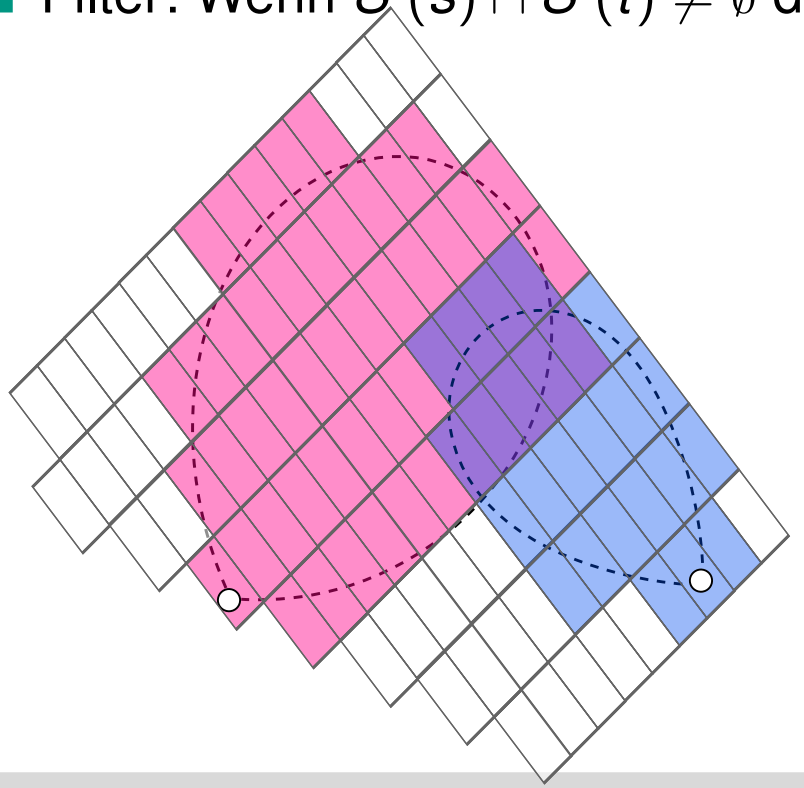
Lokalitätsfilter

- Partitioniere Graphen in Regionen
- Überapproximation des Suchraum mittels berührter Regionen.
- Wenn x im Suchraum $S(s)$ ist, dann ist die Region $R(x)$ im approximierten Suchraum $S'(s)$
- $m \in S(s) \cap S(t) \Longrightarrow R(m) \in S'(s) \cap S'(t)$

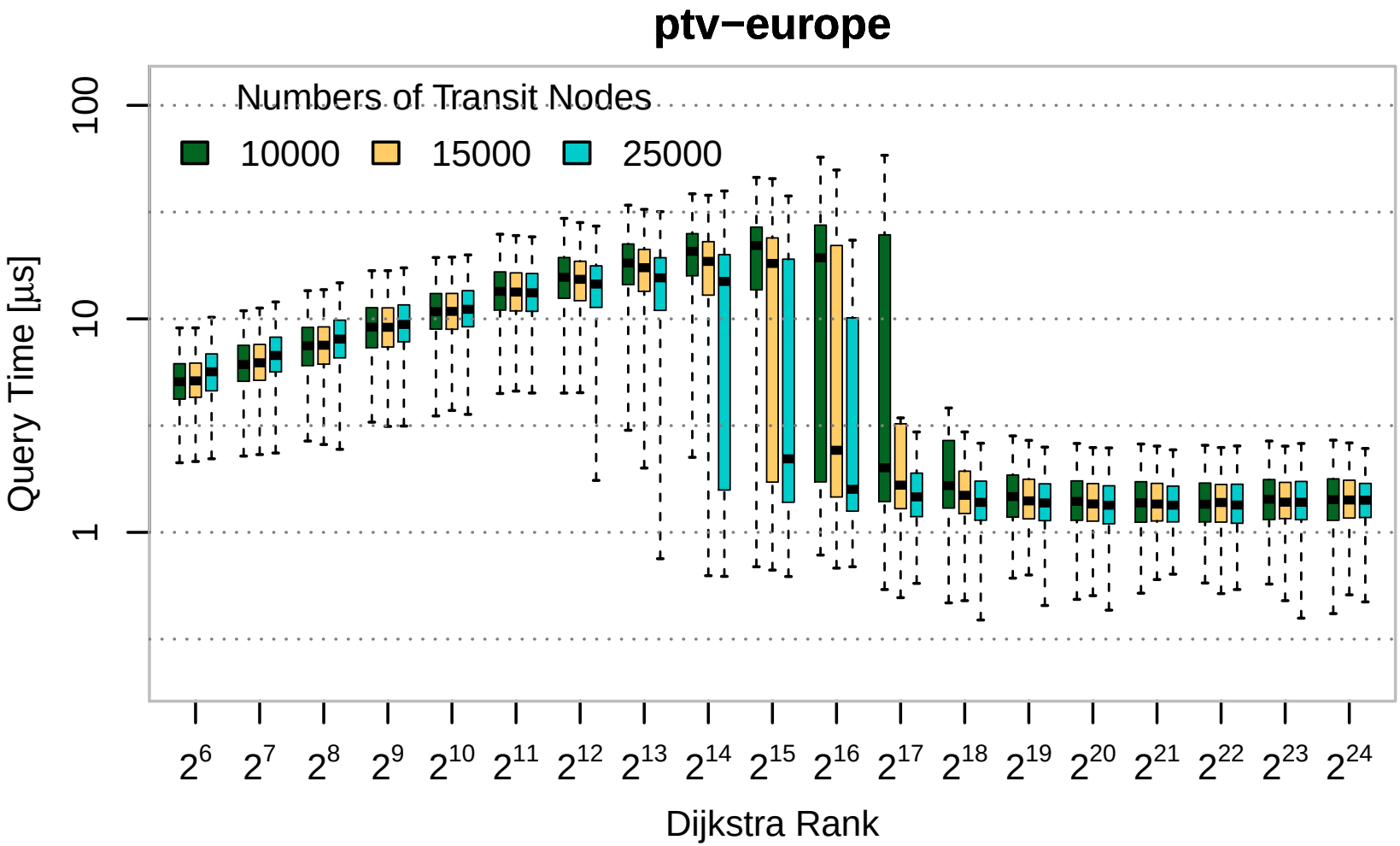


Lokalitätsfilter

- Partitioniere Graphen in Regionen
- Überapproximation des Suchraum mittels berührter Regionen.
- Wenn x im Suchraum $S(s)$ ist, dann ist die Region $R(x)$ im approximierten Suchraum $S'(s)$
- $m \in S(s) \cap S(t) \Longrightarrow R(m) \in S'(s) \cap S'(t)$
- Filter: Wenn $S'(s) \cap S'(t) \neq \emptyset$ dann lokal



Dijkstra Rank



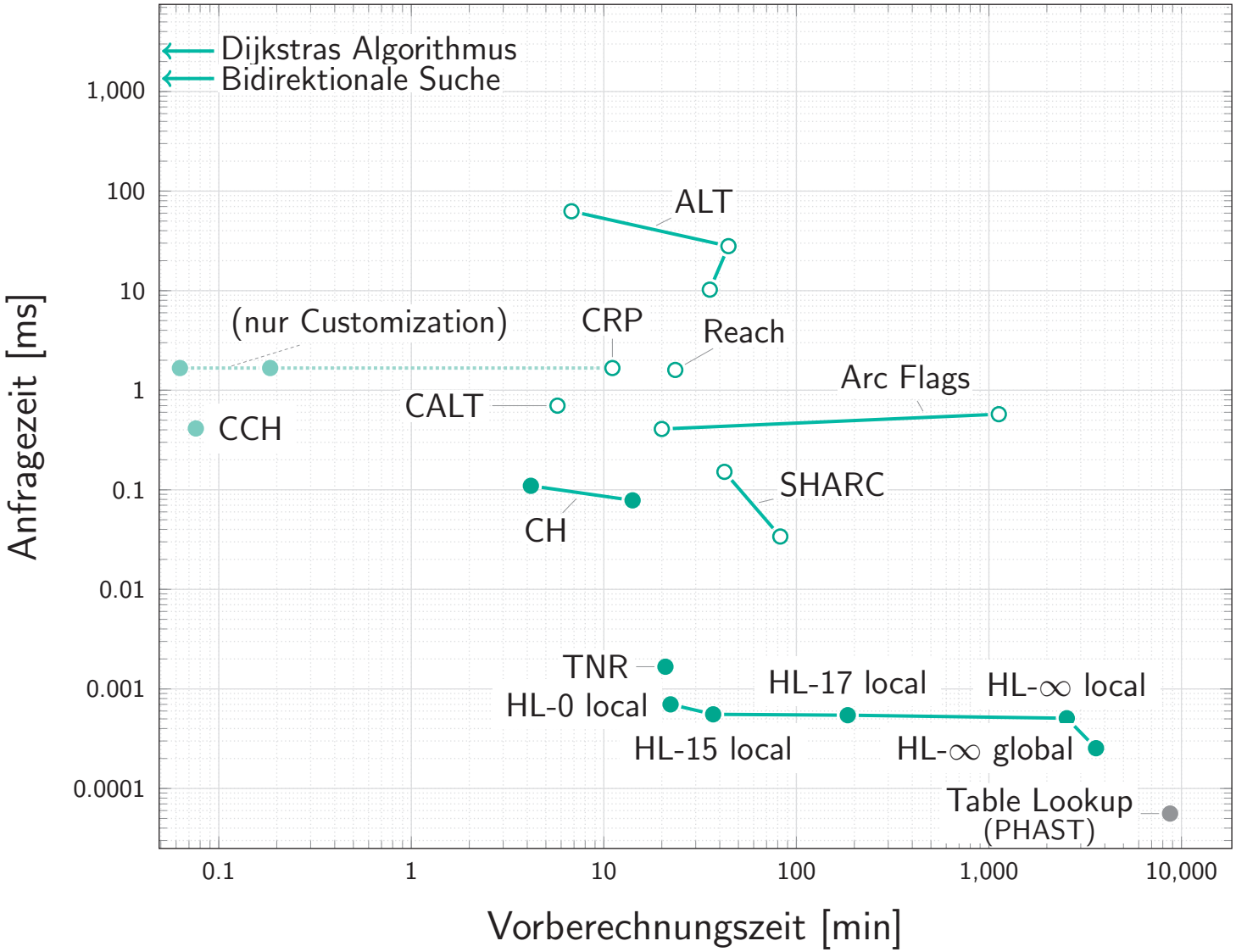
Frage: Welche durchschnittliche Laufzeit ergibt sich?

Zusammenfassung

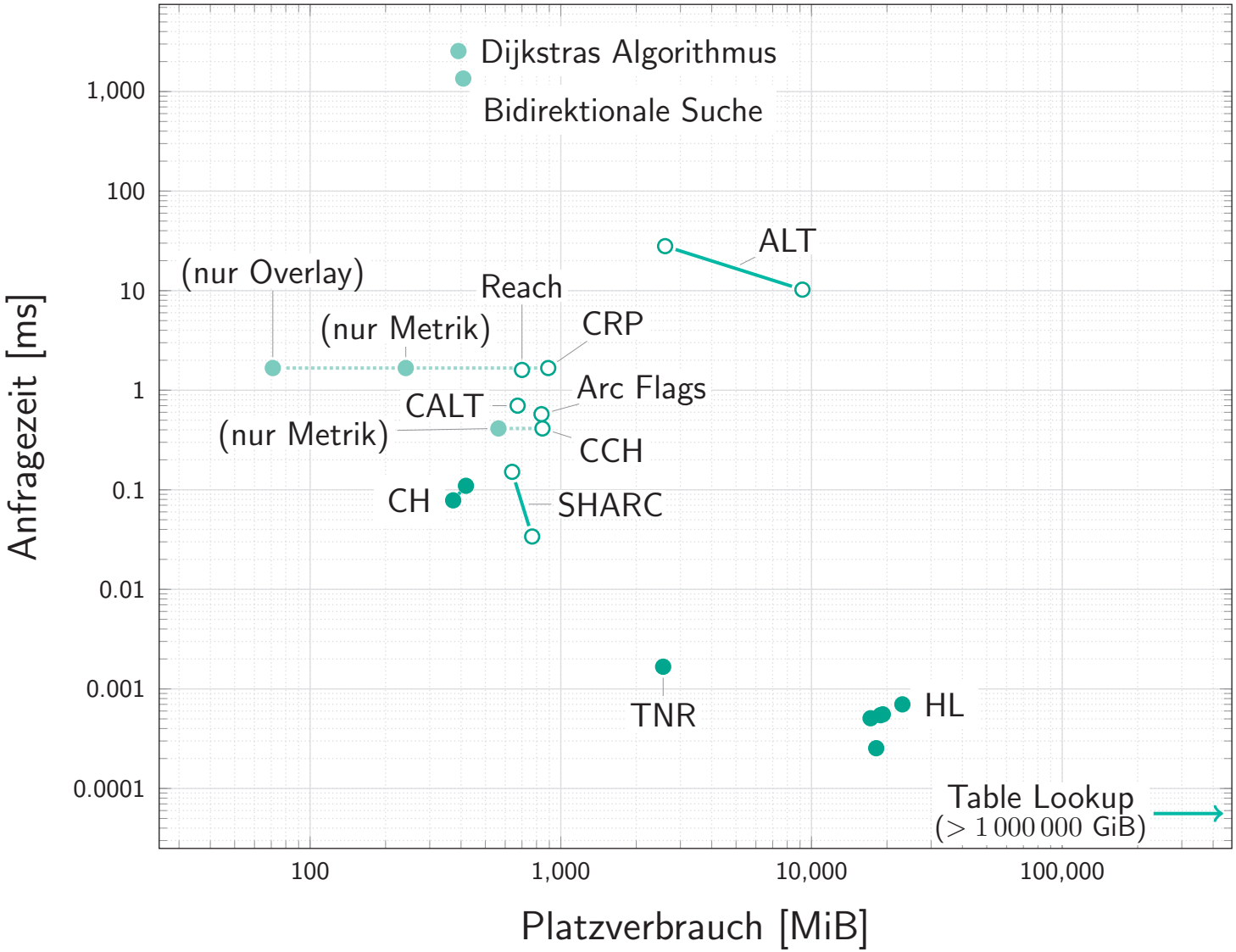
Transit-Node Routing

- ersetzt Suche (fast) komplett durch Table-Lookups
- 4 Zutaten:
  - Transit-Nodes
  - Distanztabelle
  - Access-Nodes
  - Locality-Filter

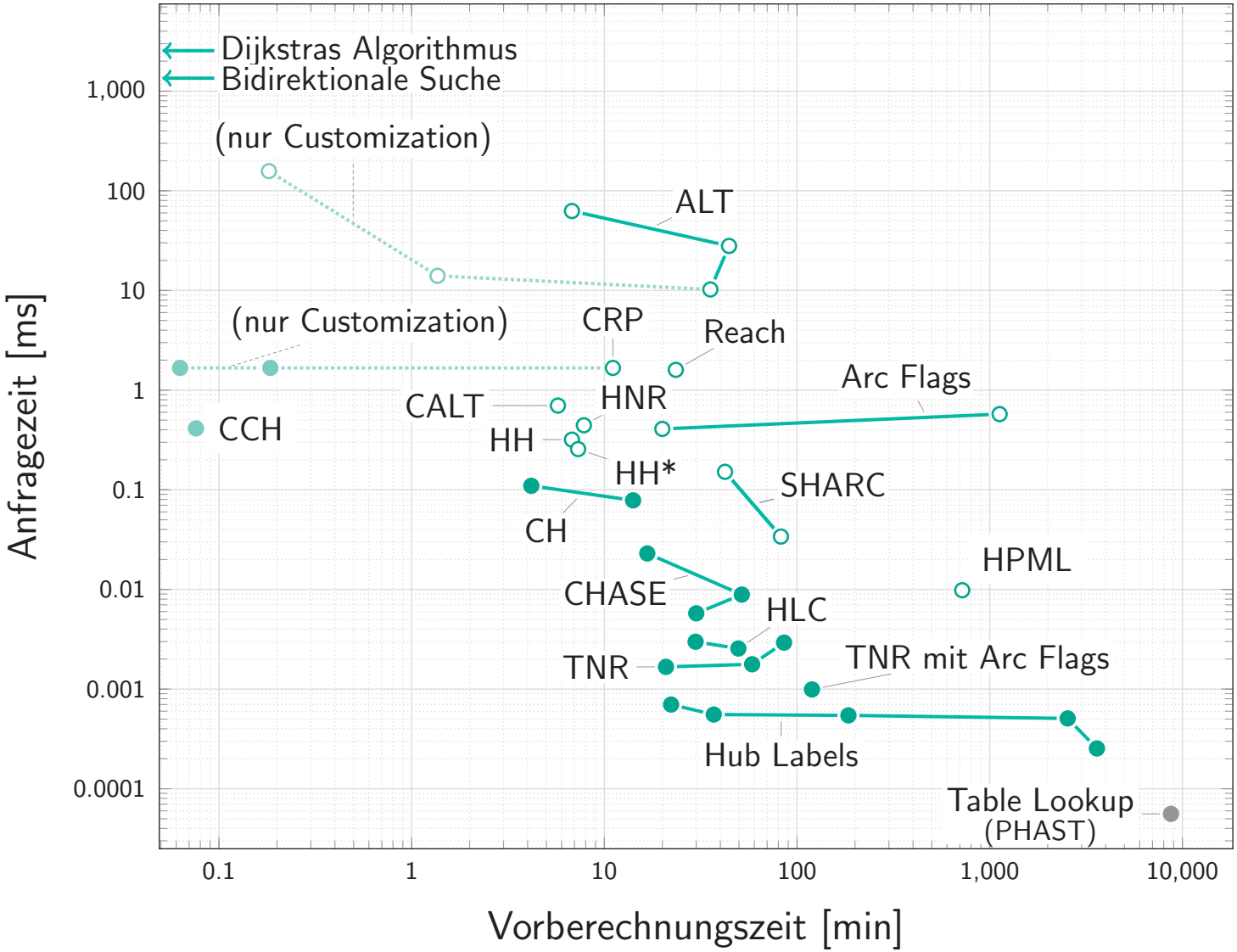
Übersicht bisherige Techniken



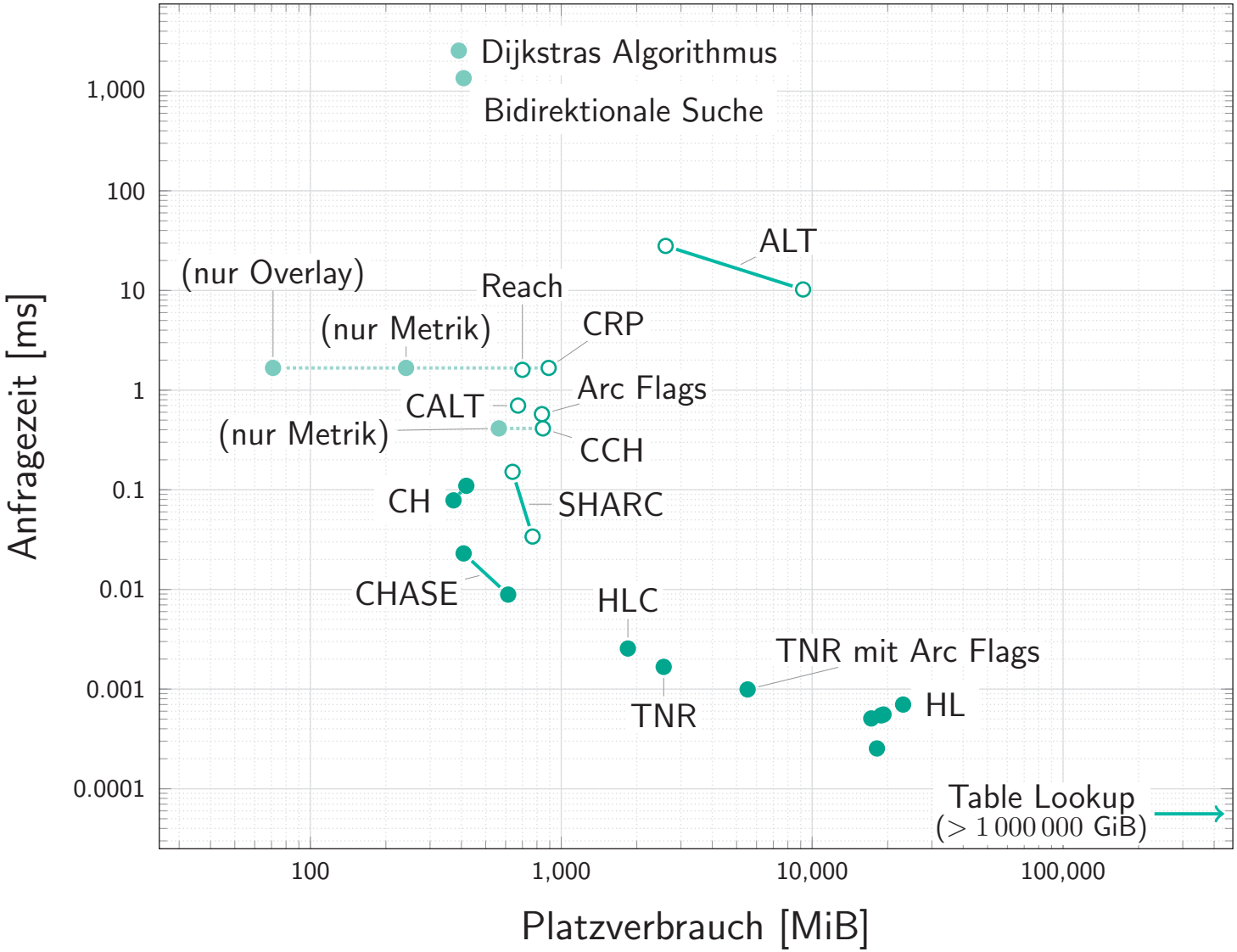
Übersicht bisherige Techniken



“Komplett”übersicht One-to-One



“Komplett”übersicht One-to-One



Literatur I

Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida.

Fast exact shortest-path distance queries on large networks by pruned landmark labeling.

In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD'13)*, pages 349–360. ACM Press, 2013.

Julian Arz, Dennis Luxen, and Peter Sanders.

Transit node routing reconsidered.

In *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13)*, volume 7933 of *Lecture Notes in Computer Science*, pages 55–66. Springer, 2013.

Holger Bast, Stefan Funke, and Domagoj Matijevic.

Transit - ultrafast shortest-path queries with linear-time preprocessing.

In *The Shortest Path Problem: Ninth DIMACS Implementation Challenge -*, November 2006.

Literatur II

Holger Bast, Stefan Funke, Domagoj Matijevic, Peter Sanders, and Dominik Schultes.

In transit to constant shortest-path queries in road networks.

In *Proceedings of the 9th Workshop on Algorithm Engineering and Experiments (ALENEX'07)*, pages 46–59. SIAM, 2007.

Maxim Babenko, Andrew V. Goldberg, Haim Kaplan, Ruslan Savchenko, and Mathias Weller.

On the complexity of hub labeling.

Technical report, ArXiv, 2015.

Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck.

Robust distance queries on massive networks.

In *Proceedings of the 22nd Annual European Symposium on Algorithms (ESA'14)*, volume 8737 of *Lecture Notes in Computer Science*, pages 321–333. Springer, September 2014.

Literatur III

Daniel Delling, Andrew V. Goldberg, Ruslan Savchenko, and Renato F. Werneck.
Hub labels: Theory and practice.
In *Proceedings of the 13th International Symposium on Experimental Algorithms (SEA'14)*, volume 8504 of *Lecture Notes in Computer Science*, pages 259–270. Springer, 2014.

Cyril Gavoille, David Peleg, Stéphane Pérennes, and Ran Raz.
Distance labeling in graphs.
Journal of Algorithms, 53:85–112, 2004.

Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter.
Exact routing in large road networks using contraction hierarchies.
Transportation Science, 46(3):388–404, August 2012.