Karlsruher Institut für Technologie

Algorithmen für Routenplanung

6. Vorlesung, Sommersemester 2015

Ben Strasser | 11. Mai 2014

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

Kürzeste Wege in Straßennetzwerken

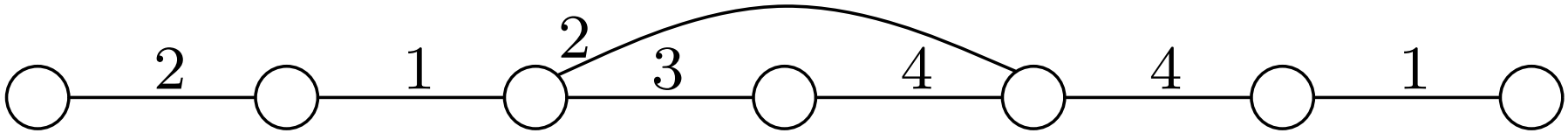
Beschleunigungstechniken (Fortsetzung)

- Transit-Node Routing
- Hub-Labels

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Wiederholung: CH

preprocessing:

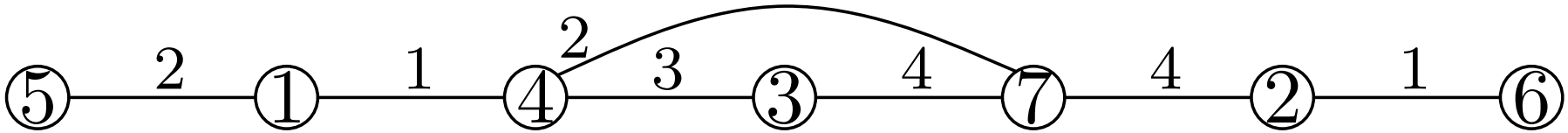


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Wiederholung: CH

preprocessing:

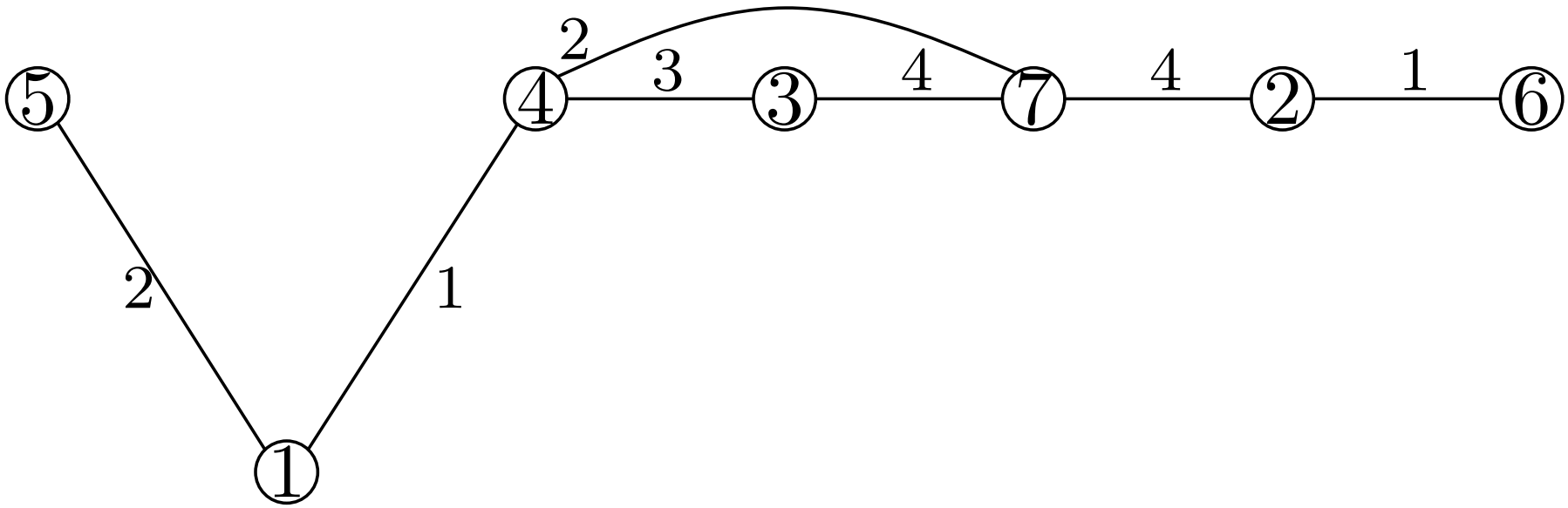
- ordne Knoten nach Wichtigkeit



Wiederholung: CH

preprocessing:

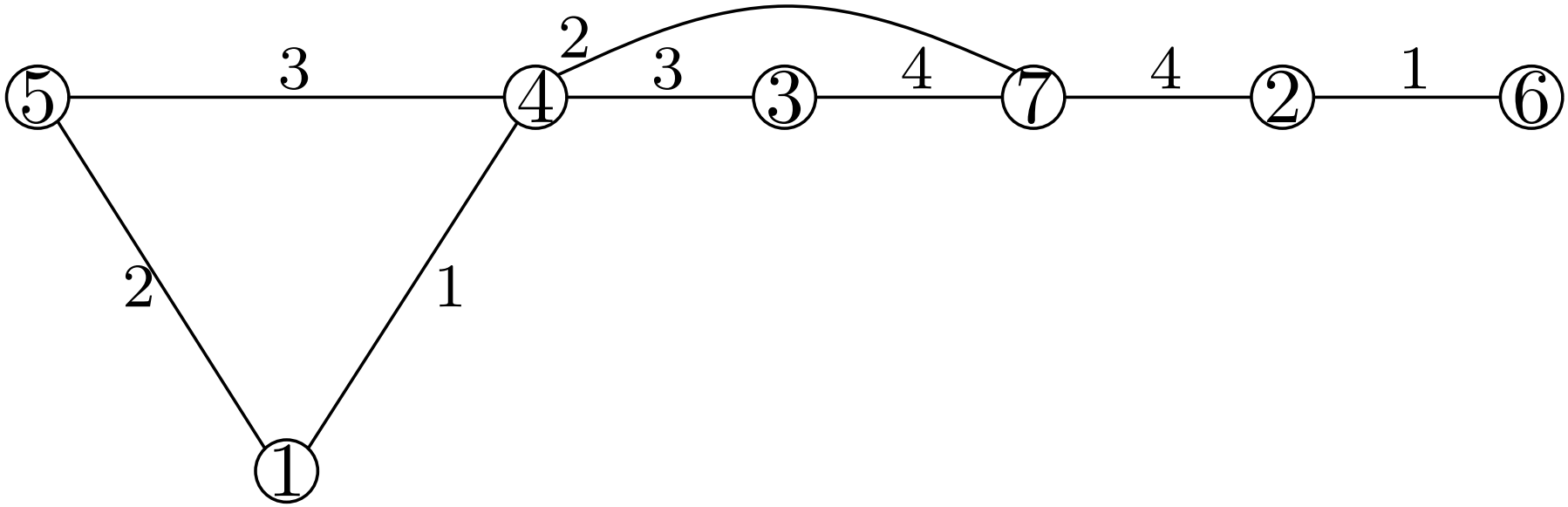
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu



Wiederholung: CH

preprocessing:

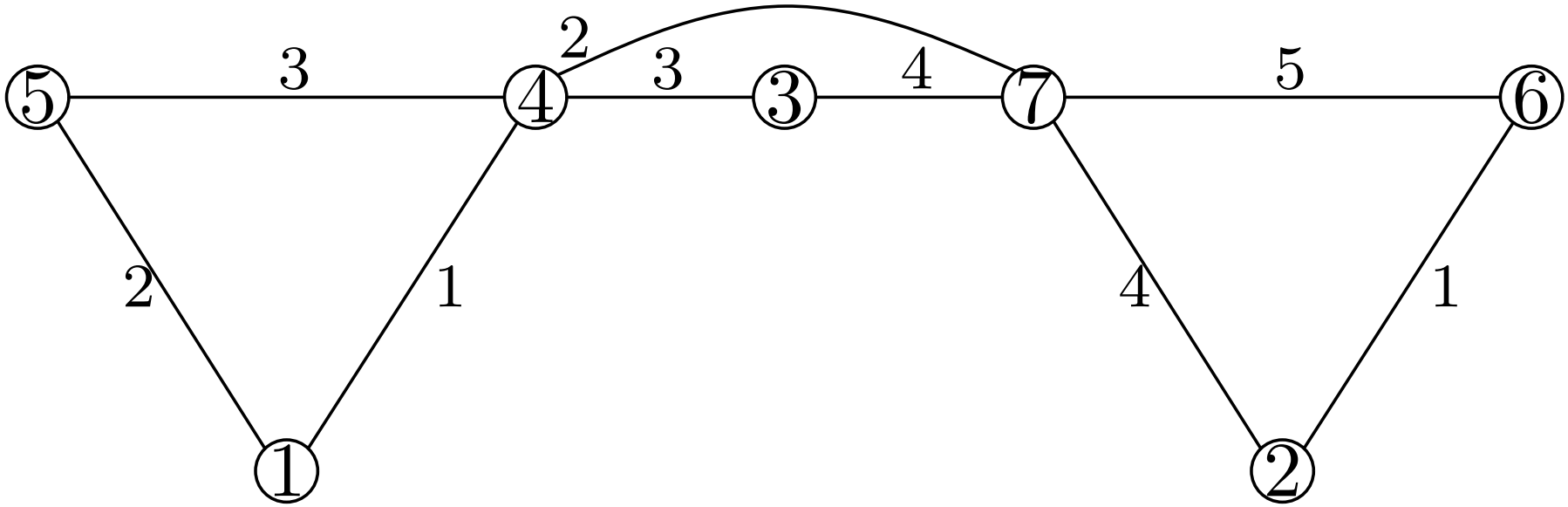
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu



Wiederholung: CH

preprocessing:

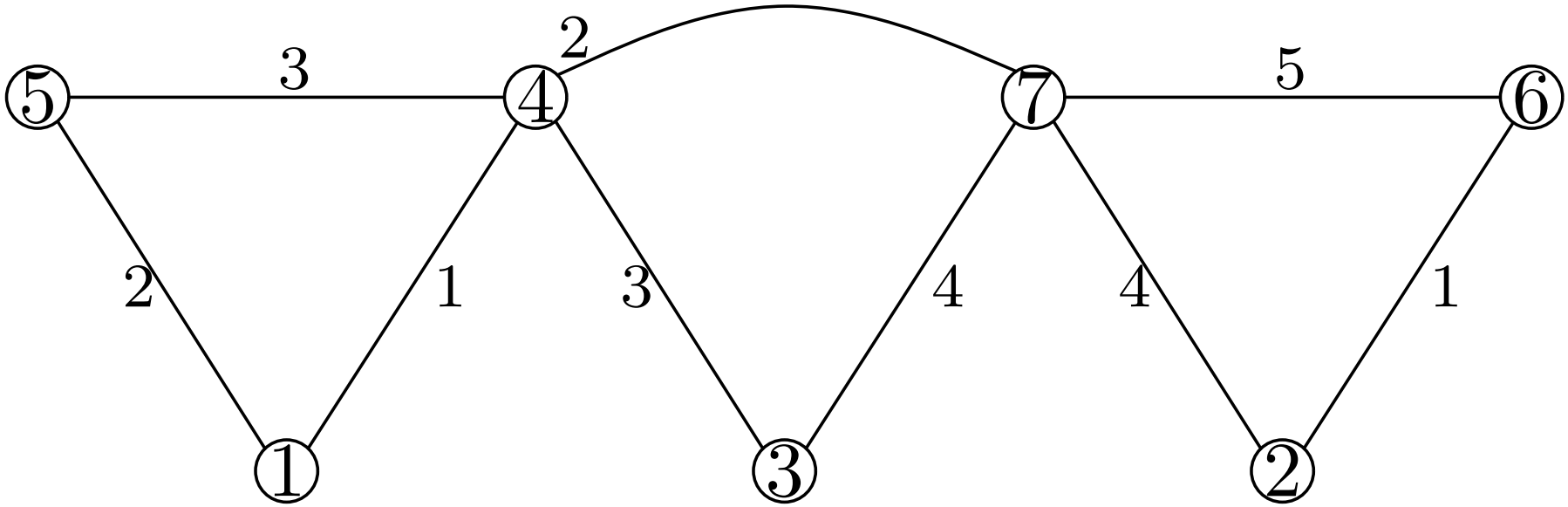
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu



Wiederholung: CH

preprocessing:

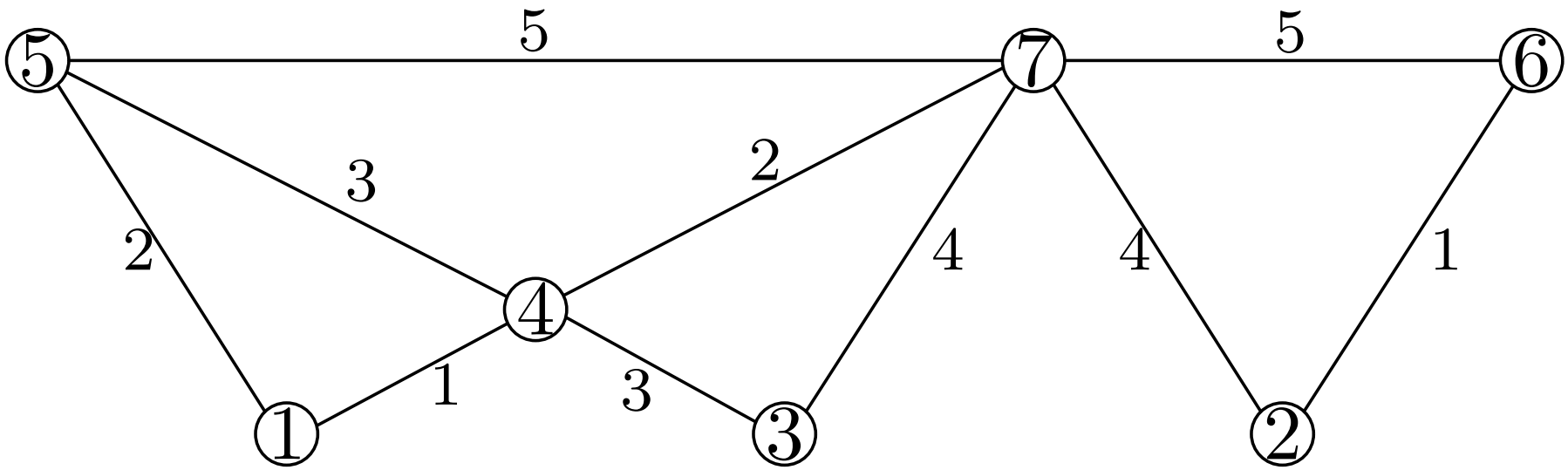
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu



Wiederholung: CH

preprocessing:

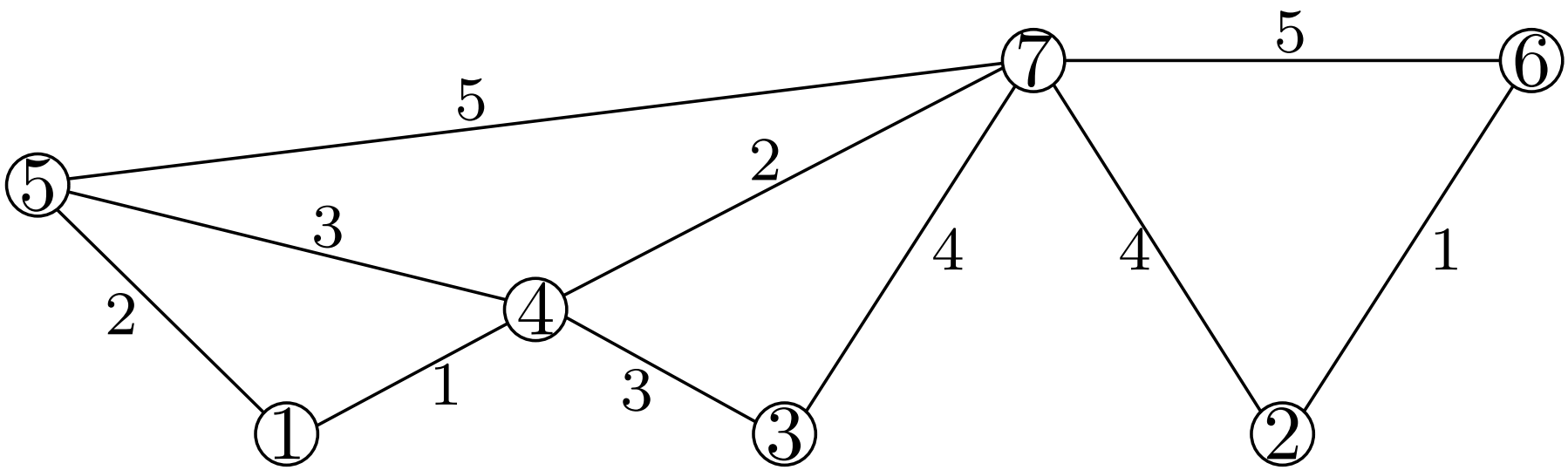
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu



Wiederholung: CH

preprocessing:

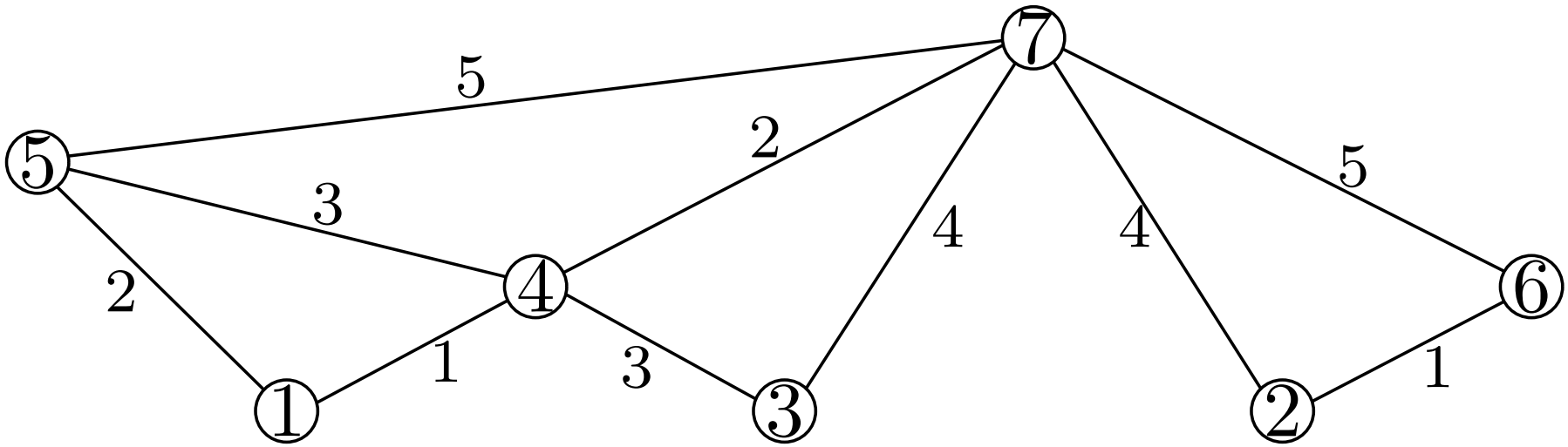
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu



Wiederholung: CH

preprocessing:

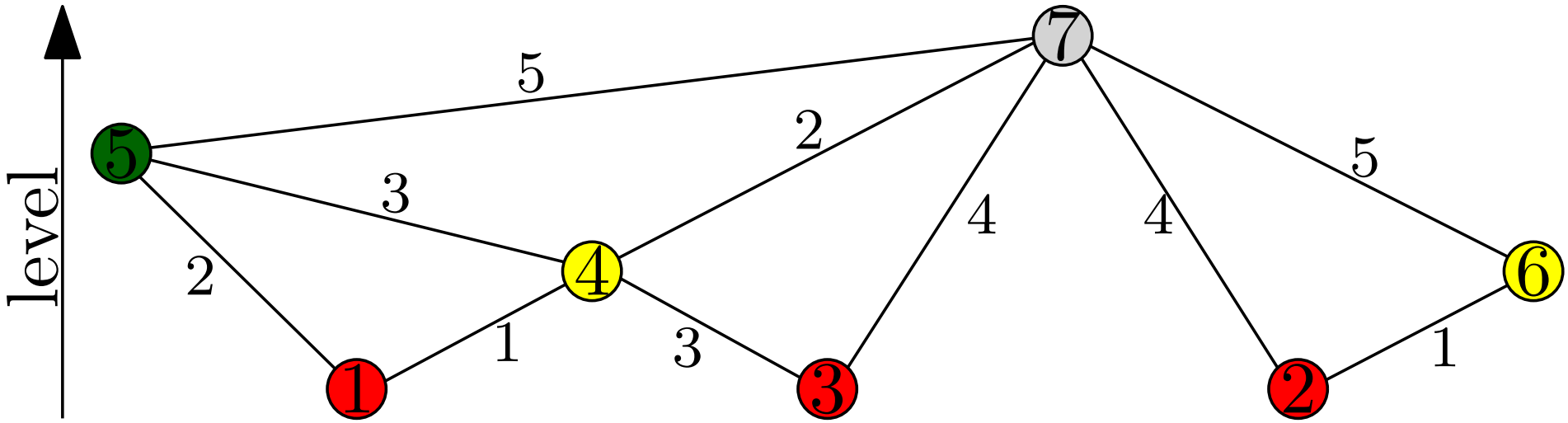
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu



Wiederholung: CH

preprocessing:

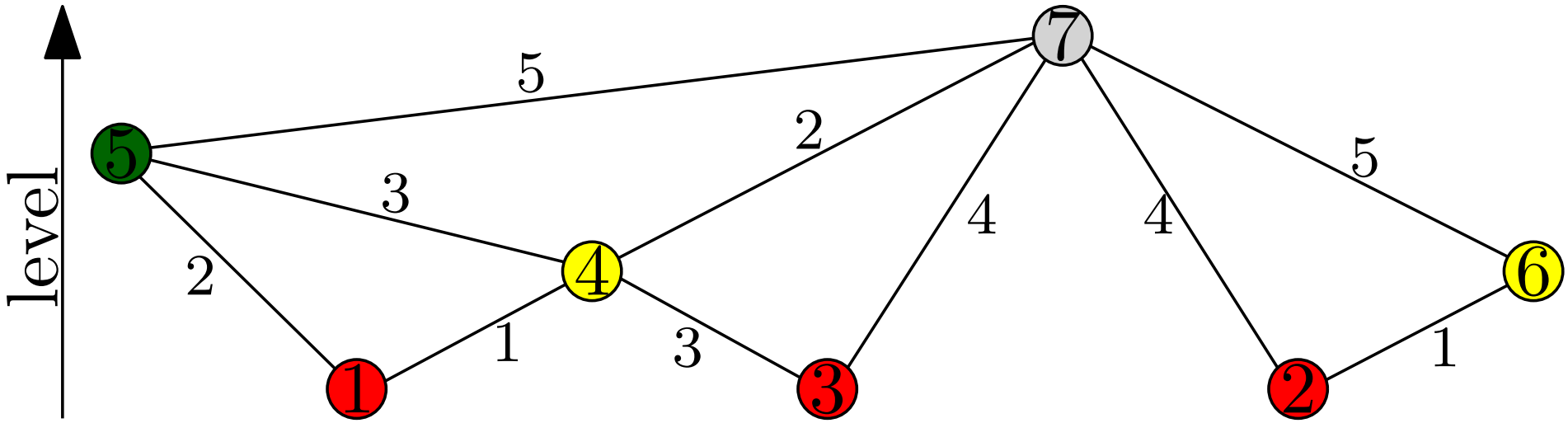
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu
- Levelzuordnung



Wiederholung: CH

Punkt-zu-Punkt Anfragen

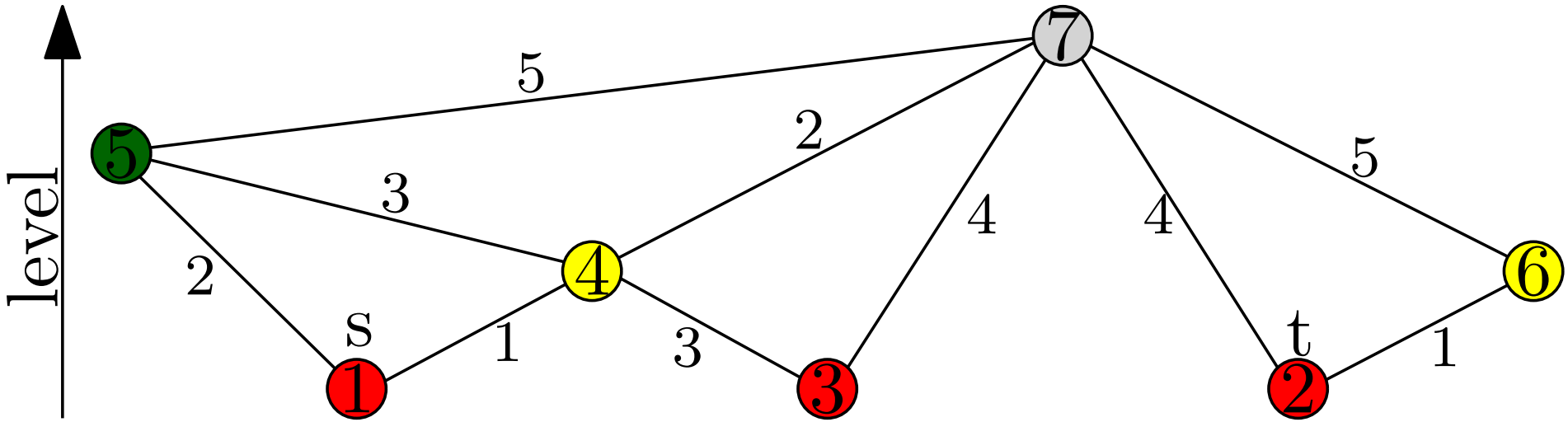
- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten



Wiederholung: CH

Punkt-zu-Punkt Anfragen

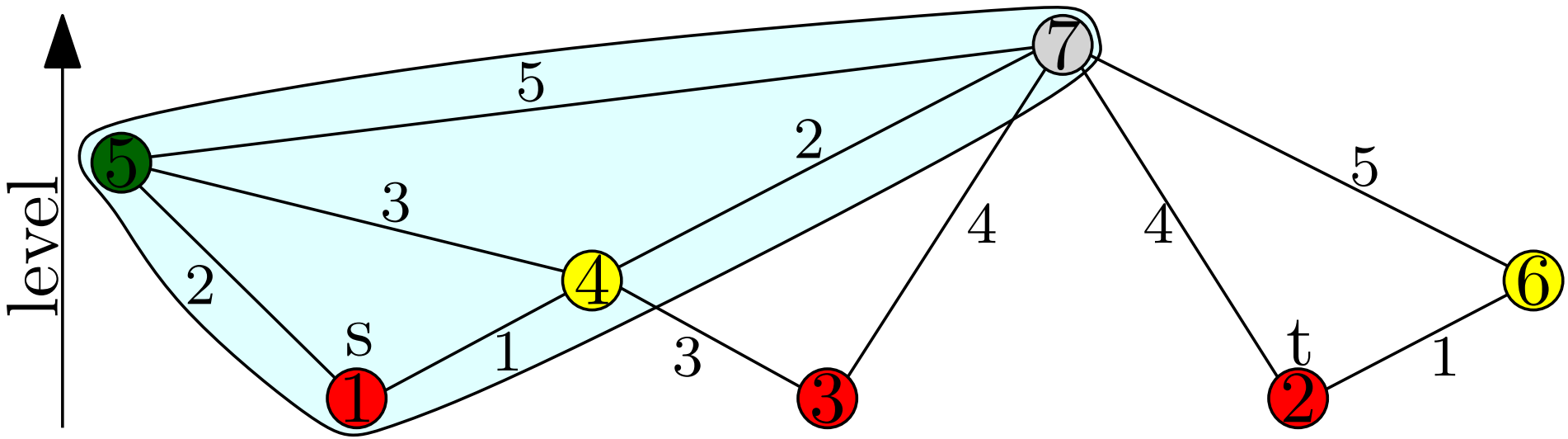
- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten



Wiederholung: CH

Punkt-zu-Punkt Anfragen

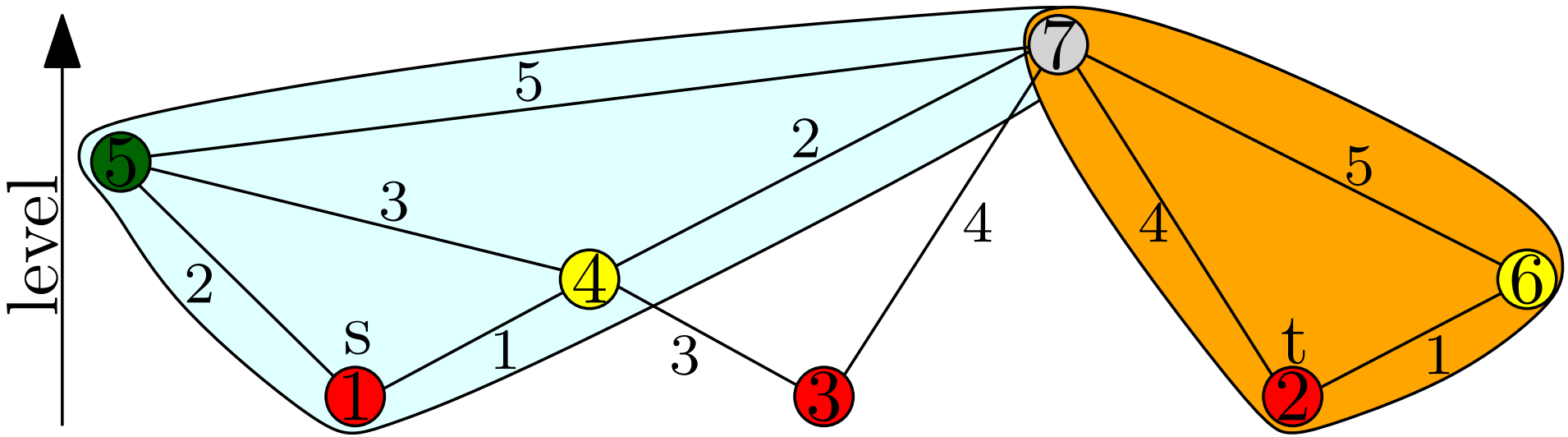
- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten



Wiederholung: CH

Punkt-zu-Punkt Anfragen

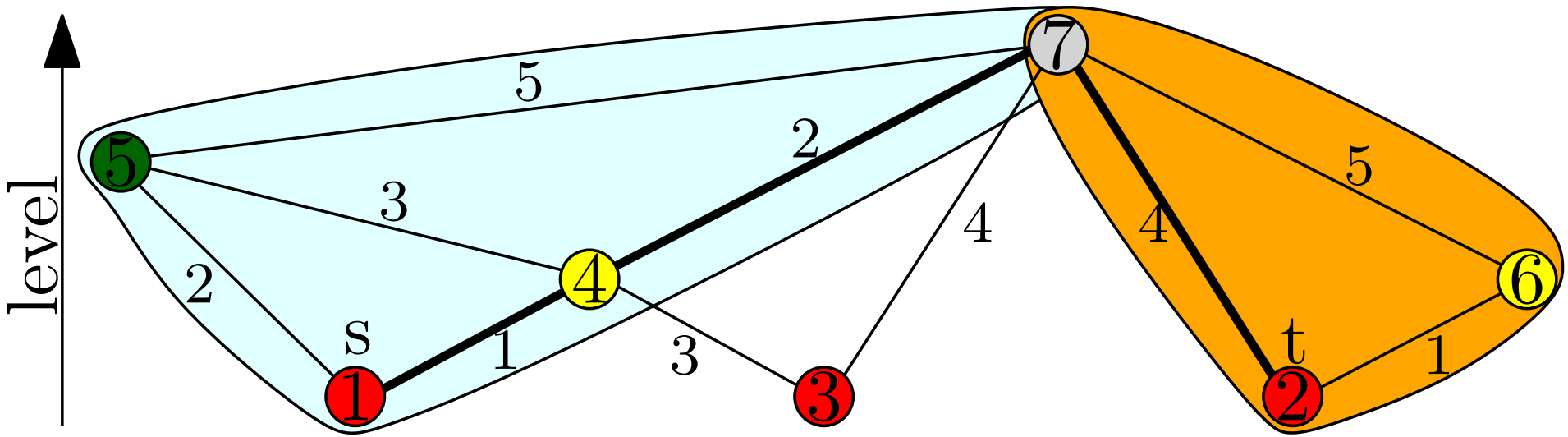
- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten



Wiederholung: CH

Punkt-zu-Punkt Anfragen

- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten



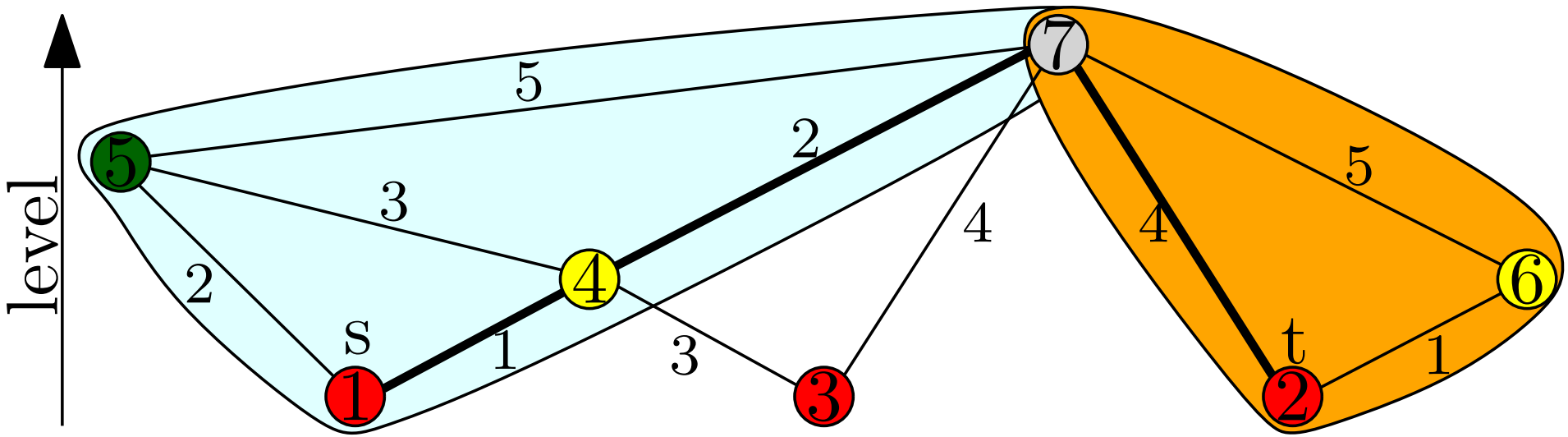
Wiederholung: CH

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

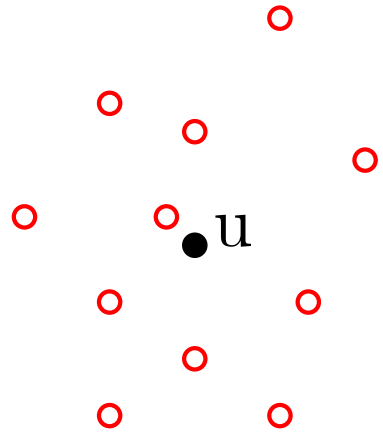
Vorberechung:

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

Hublabels

Vorberechung:

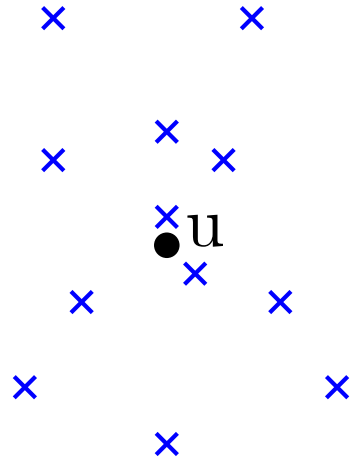
- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
  - dist(u, v) für jeden Hub $v \in L_f(u)$



Hublabels

Vorberechung:

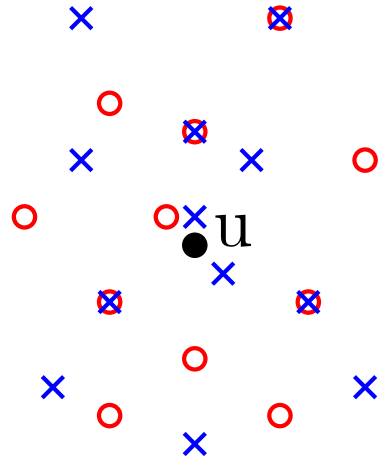
- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
  - dist(u, v) für jeden Hub $v \in L_f(u)$
  - dist(v, u) für jeden Hub $v \in L_b(u)$



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

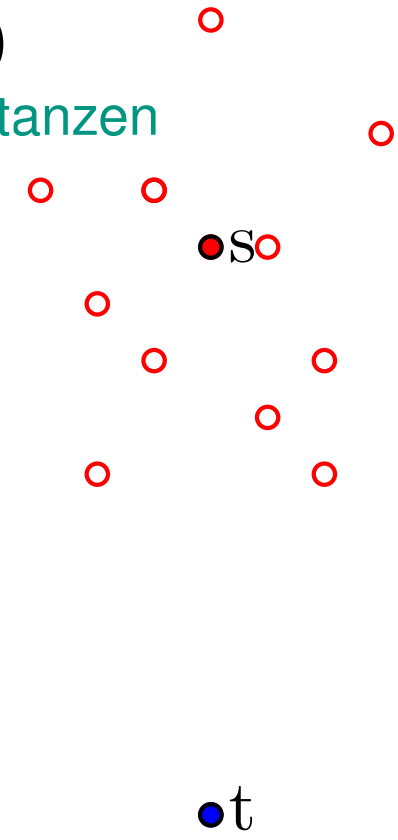
Vorberechung:

- 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:

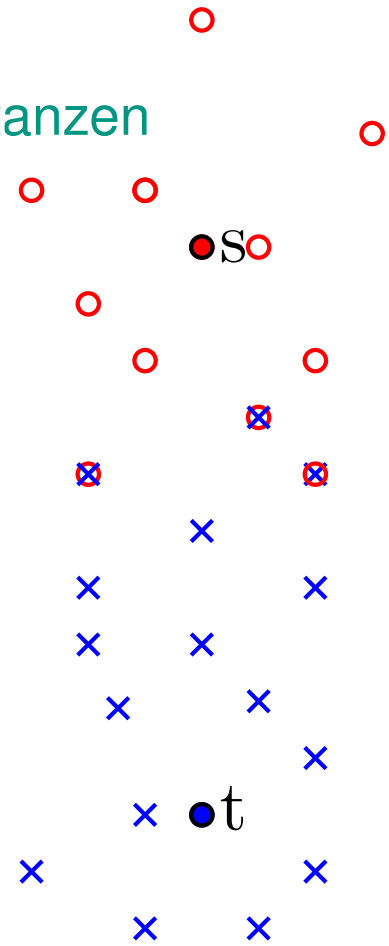
- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
  - dist(u, v) für jeden Hub $v \in L_f(u)$
  - dist(v, u) für jeden Hub $v \in L_b(u)$
- die Label müssen die cover property einhalten:
  $\forall s, t$, $L_f(s) \cap L_b(t)$ überdeckt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



Hublabels

Vorberechung:

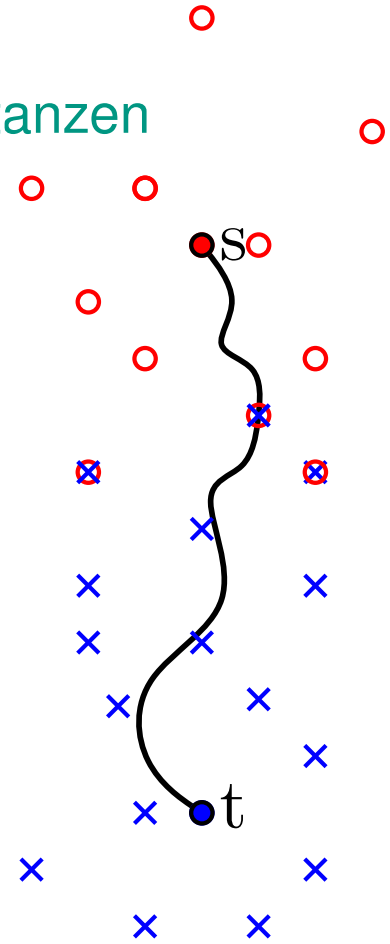
- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
  - dist(u, v) für jeden Hub $v \in L_f(u)$
  - dist(v, u) für jeden Hub $v \in L_b(u)$
- die Label müssen die cover property einhalten:
  $\forall s, t$, $L_f(s) \cap L_b(t)$ überdeckt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



Hublabels

Vorberechung:

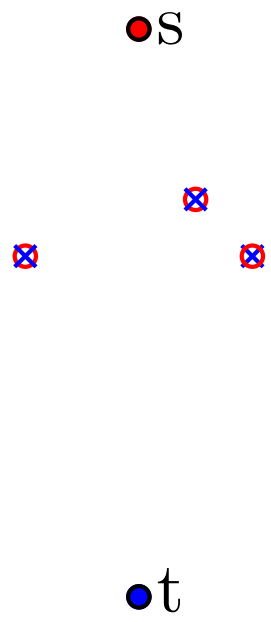
- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
  - dist(u, v) für jeden Hub $v \in L_f(u)$
  - dist(v, u) für jeden Hub $v \in L_b(u)$
- die Label müssen die cover property einhalten:
  $\forall s, t$, $L_f(s) \cap L_b(t)$ überdeckt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



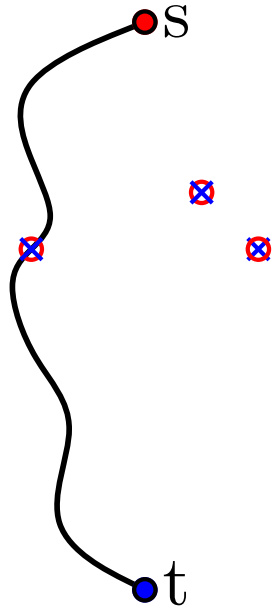
s–t Anfrage:

- 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
  - $\text{dist}(u, v)$ für jeden Hub $v \in L_f(u)$
  - $\text{dist}(v, u)$ 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



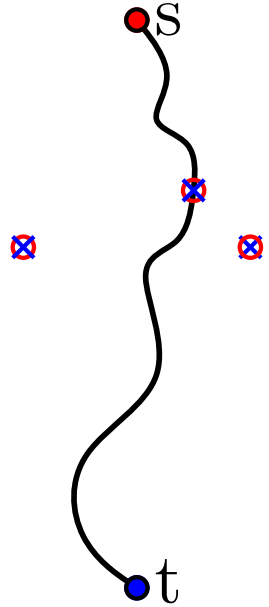
s–t Anfrage:

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



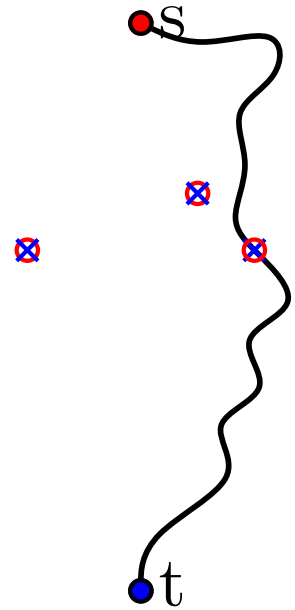
s–t Anfrage:

- 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
  - $\text{dist}(u, v)$ für jeden Hub $v \in L_f(u)$
  - $\text{dist}(v, u)$ 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



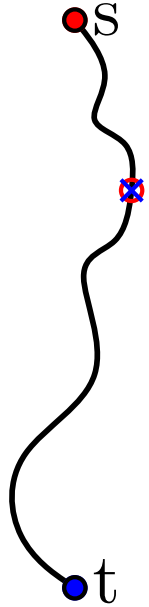
s–t Anfrage:

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



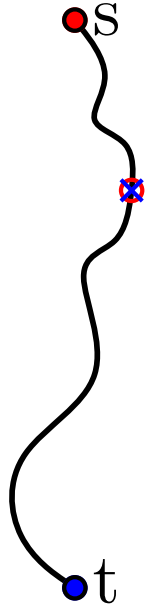
s–t Anfrage:

- 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
  - $\text{dist}(u, v)$ für jeden Hub $v \in L_f(u)$
  - $\text{dist}(v, u)$ 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 $\text{dist}(s, v) + \text{dist}(v, t)$ minimiert

Beobachtungen:

- Laufzeit hängt von Labelgrösse ab
- wie effizient berechnen?

Hublabels

Speichern der Labels:

- als Menge von Hub,Distanz Paaren

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

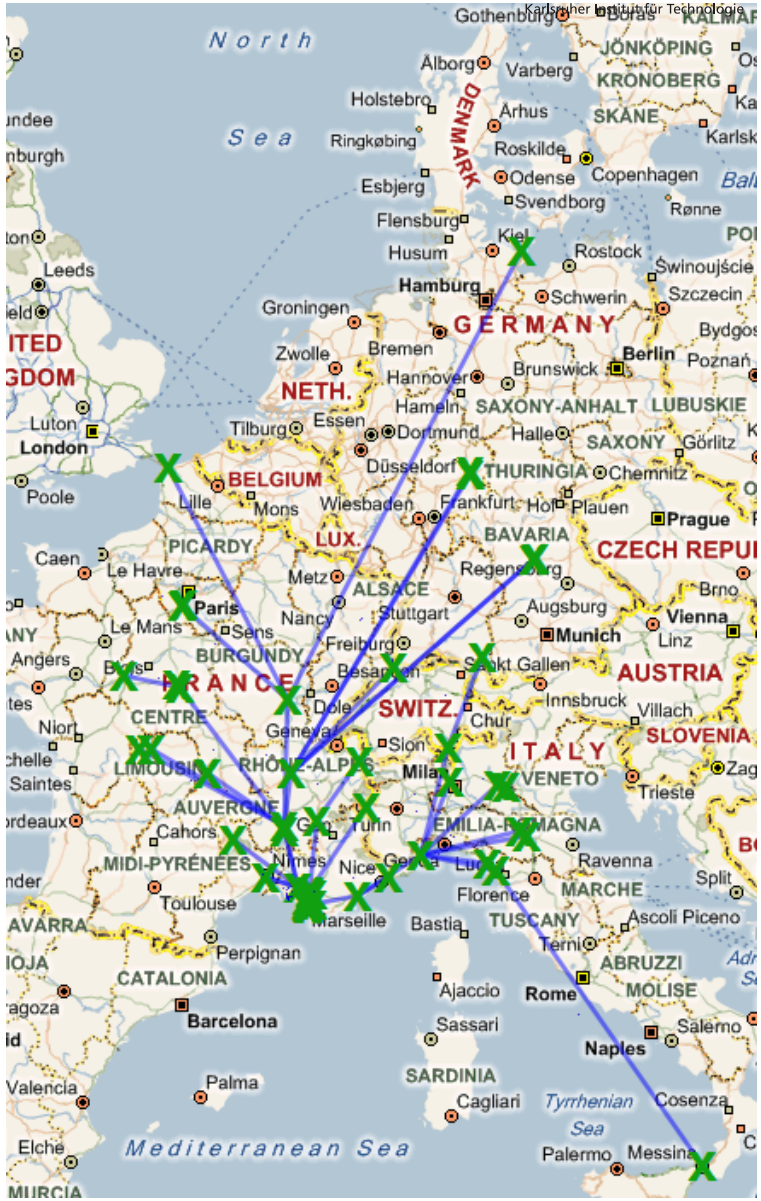
Hublabels

Speichern der Labels:

- als Menge von Hub,Distanz Paaren

$L_f(s)$

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

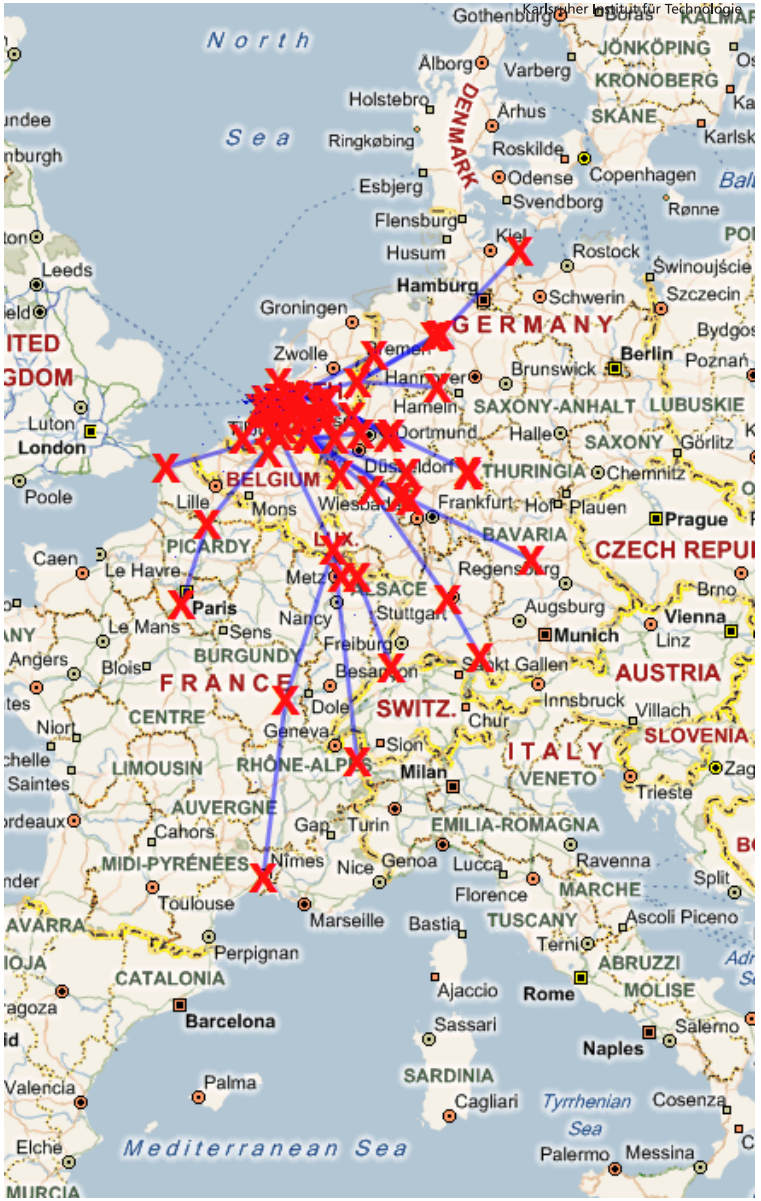


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Hublabels

Speichern der Labels:

- als Menge von Hub,Distanz Paaren



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

$L_b(t)$ | 2,0 | 6,1 | 7,4 | 8,1 | 9,3 |

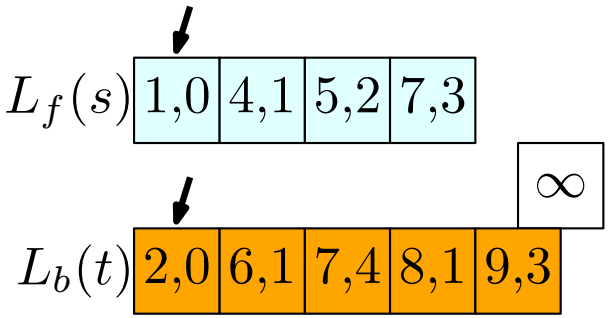
Hublabels

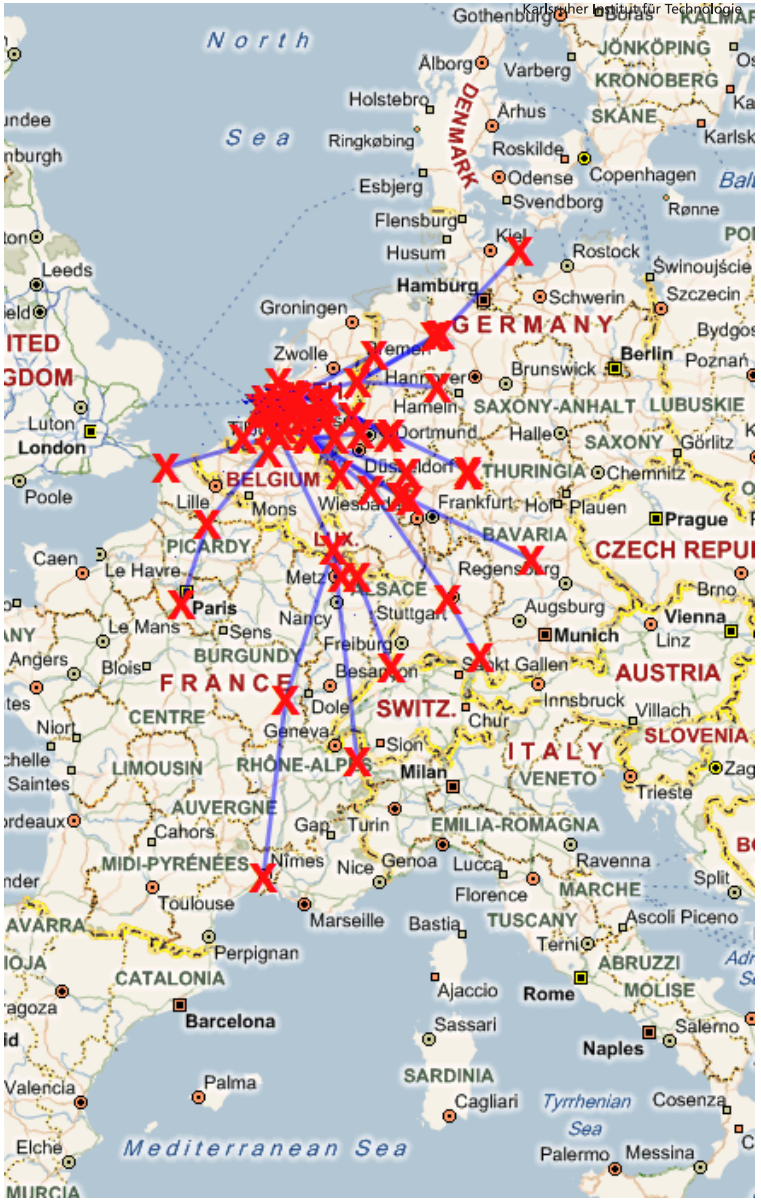
Speichern der Labels:

- als Menge von Hub,Distanz Paaren

Anfrage:

- scannen von zwei Arrays
- nur einige Speicherzugriffe nötig





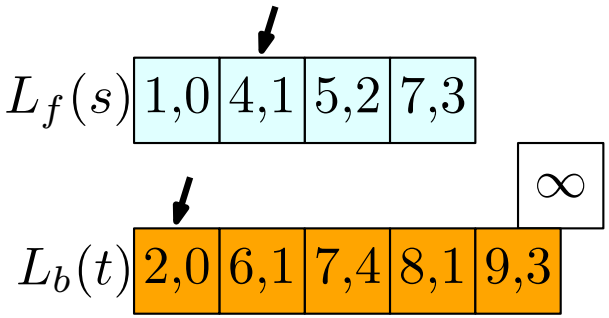
Hublabels

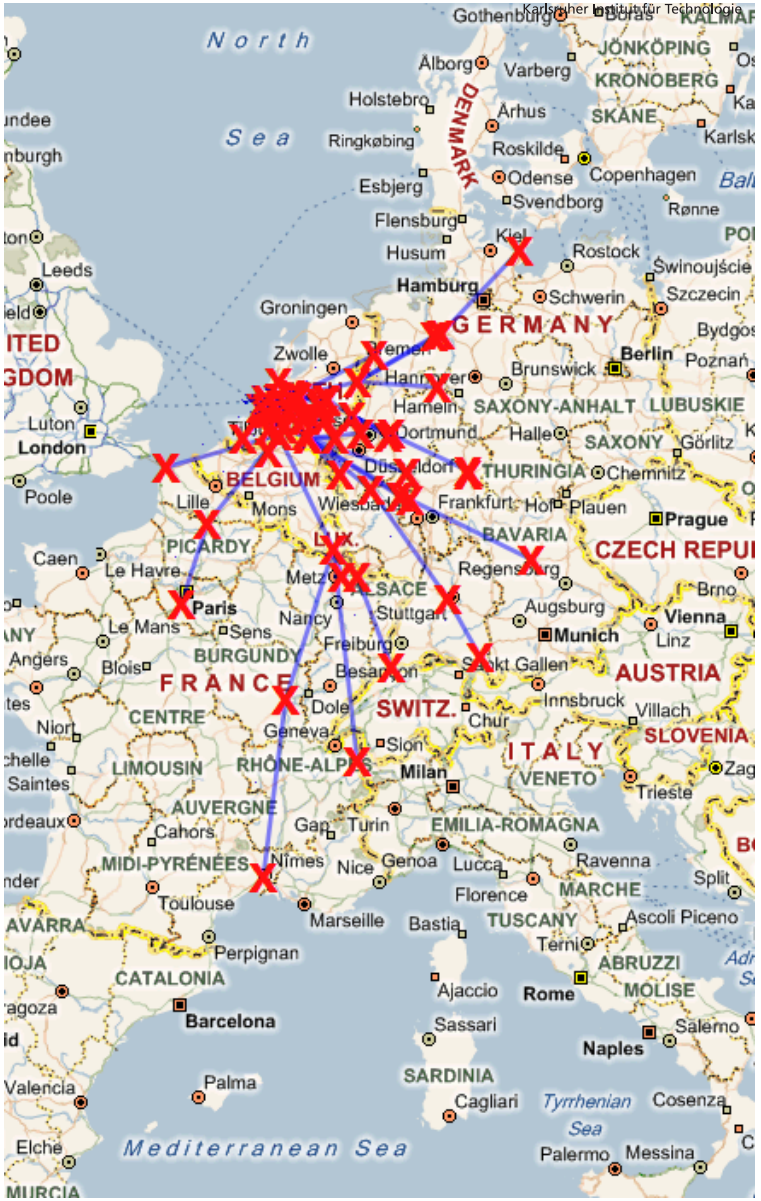
Speichern der Labels:

- als Menge von Hub,Distanz Paaren

Anfrage:

- scannen von zwei Arrays
- nur einige Speicherzugriffe nötig





Institut für Theoretische Informatik
Lehrstuhl Algorithmik

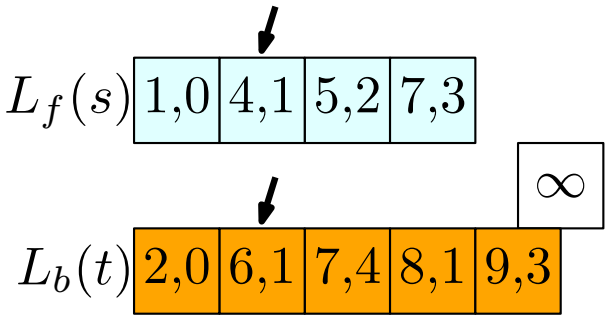
Hublabels

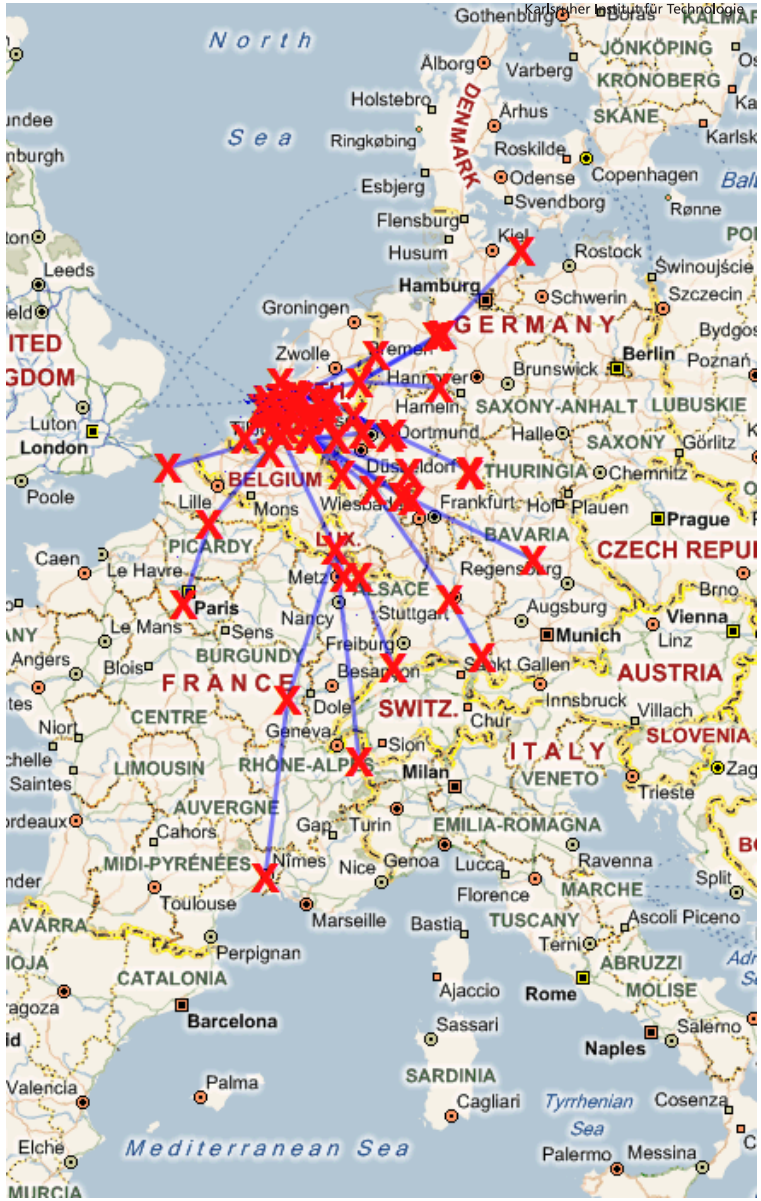
Speichern der Labels:

- als Menge von Hub,Distanz Paaren

Anfrage:

- scannen von zwei Arrays
- nur einige Speicherzugriffe nötig





Institut für Theoretische Informatik
Lehrstuhl Algorithmik

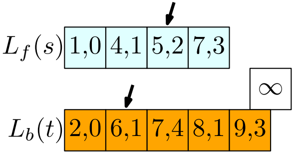
Hublabels

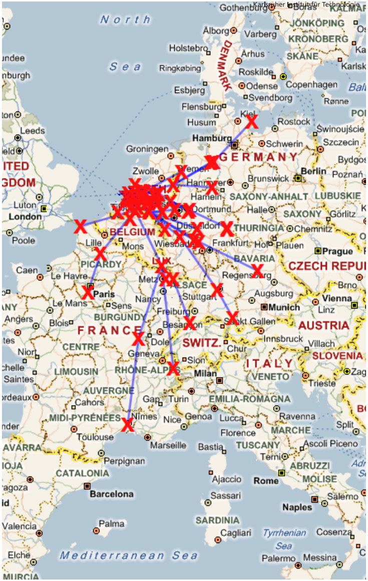
Speichern der Labels:

- als Menge von Hub,Distanz Paaren

Anfrage:

- scannen von zwei Arrays
- nur einige Speicherzugriffe nötig





Institut für Theoretische Informatik
Lehrstuhl Algorithmik

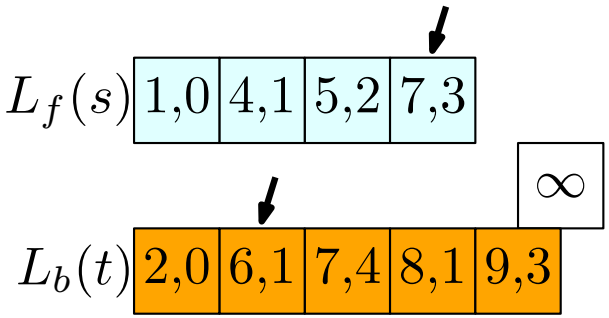
Hublabels

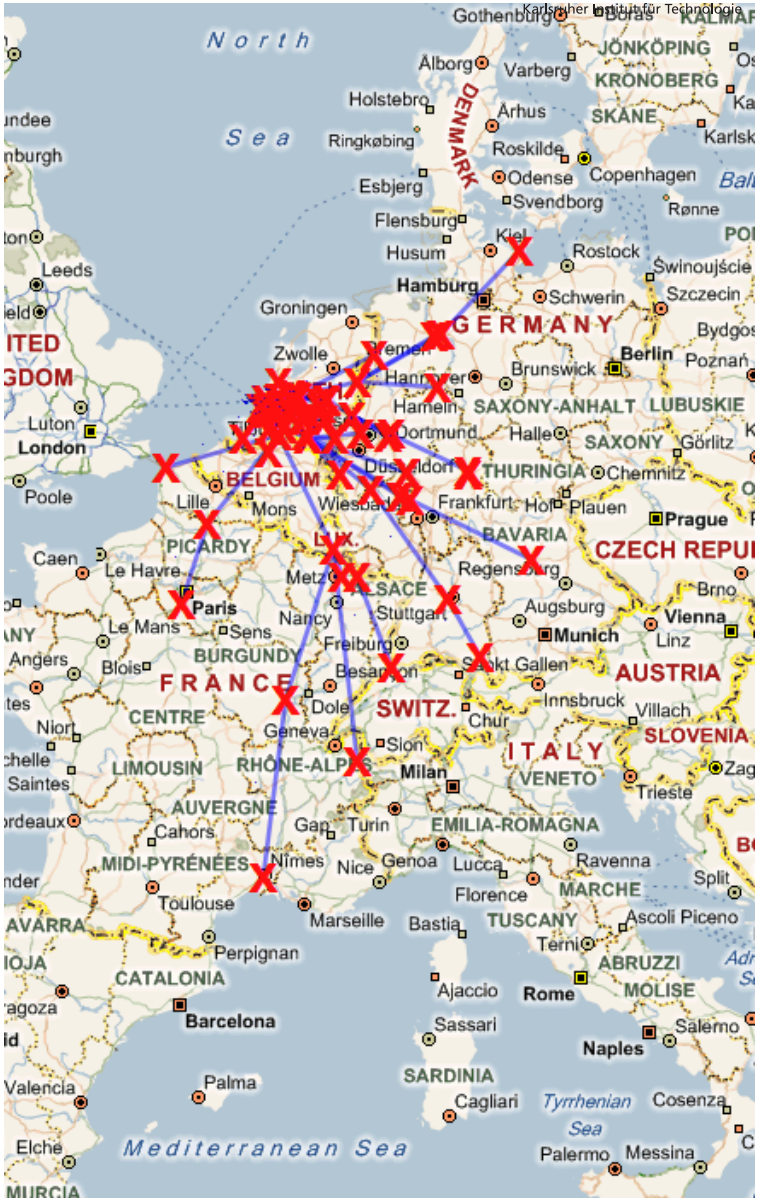
Speichern der Labels:

- als Menge von Hub,Distanz Paaren

Anfrage:

- scannen von zwei Arrays
- nur einige Speicherzugriffe nötig





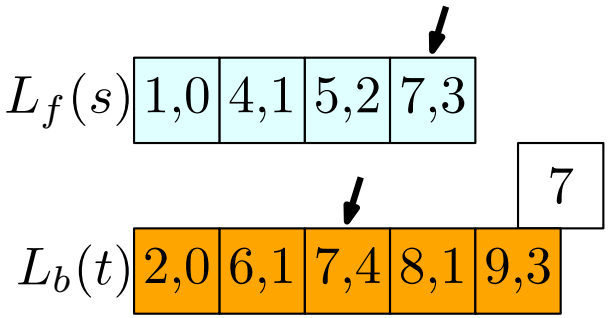
Hublabels

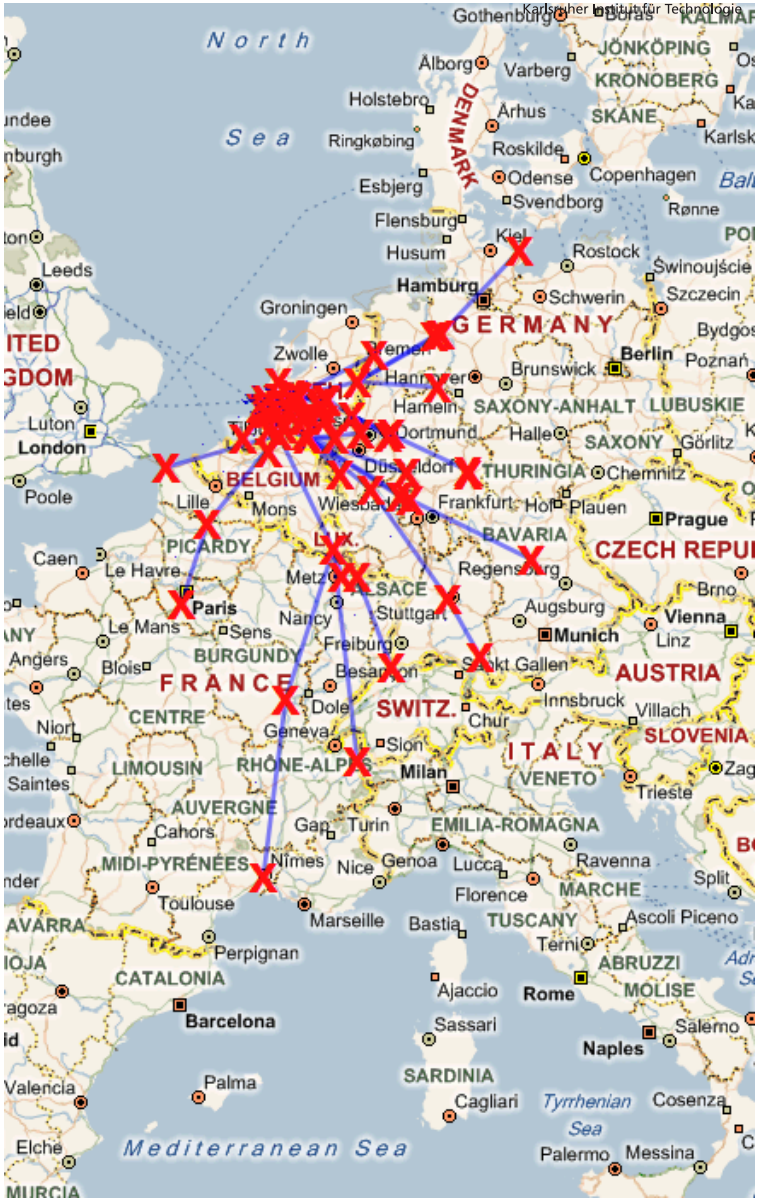
Speichern der Labels:

- als Menge von Hub,Distanz Paaren

Anfrage:

- scannen von zwei Arrays
- nur einige Speicherzugriffe nötig





Institut für Theoretische Informatik
Lehrstuhl Algorithmik

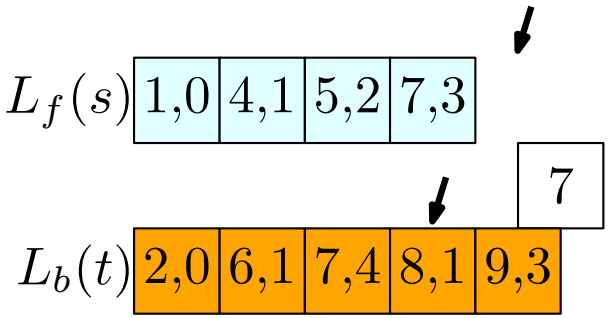
Hublabels

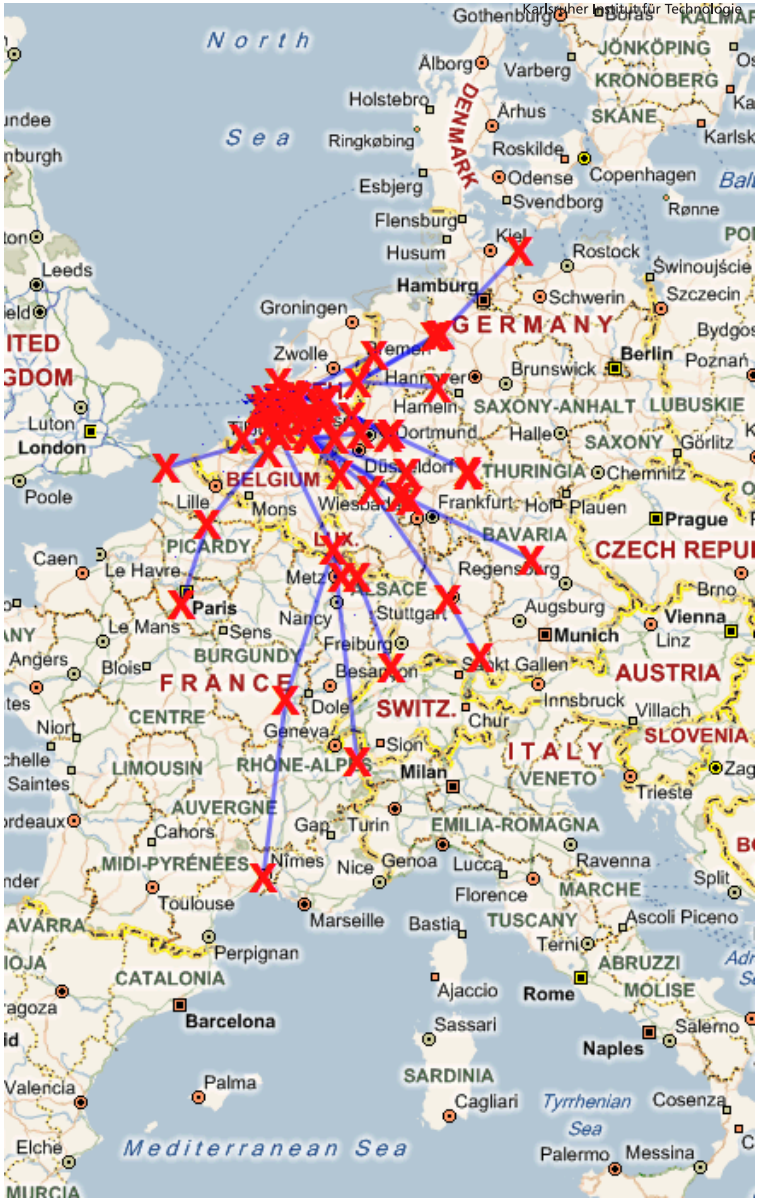
Speichern der Labels:

- als Menge von Hub,Distanz Paaren

Anfrage:

- scannen von zwei Arrays
- nur einige Speicherzugriffe nötig





Hublabels

Speichern der Labels:

- als Menge von Hub,Distanz Paaren

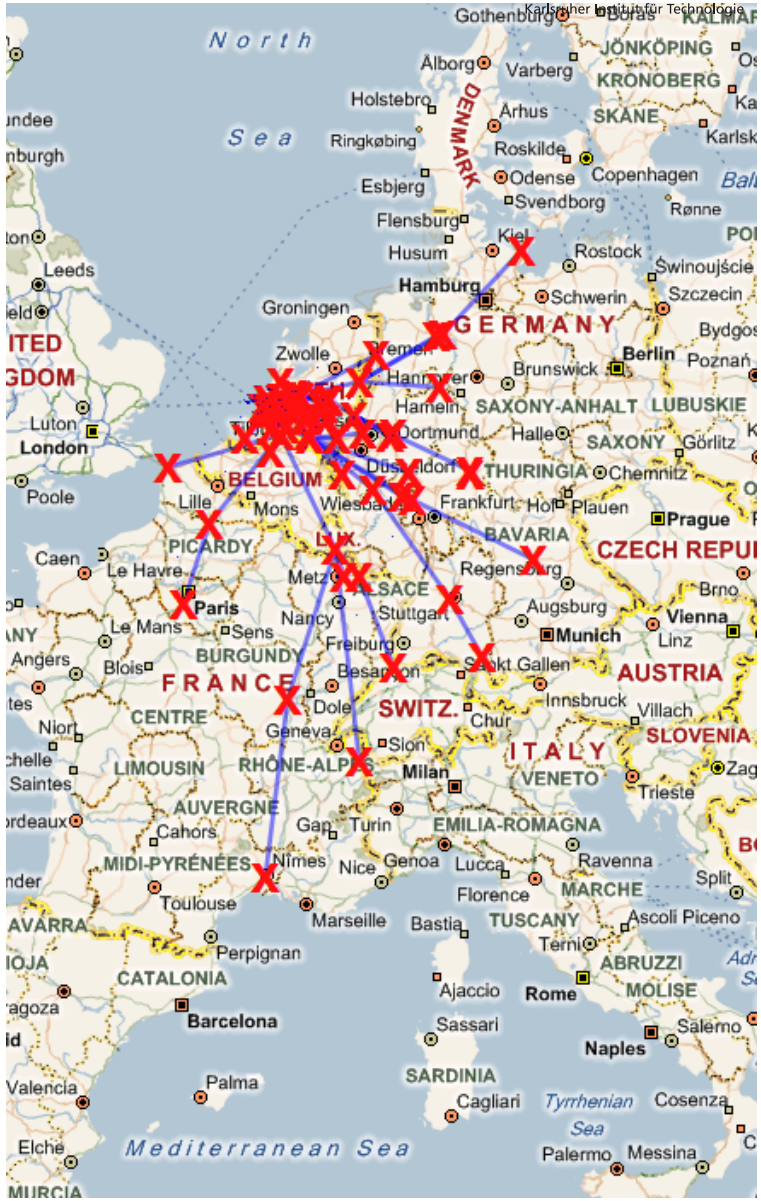
Anfrage:

- 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 |



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, sie sind 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, sie sind nicht kanonisch
- $\rightarrow$ Überflüssige Knoten filtern

- Im folgenden betrachten wir nur noch hierarchisch Hublabels
- Für Beweise nehmen wir ferner an, dass kürzeste Wege eindeutig sind und Graphen ungerichtet sind

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 genau dann wenn $h \neq m(v,h)$.

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 genau dann wenn $h \neq m(v,h)$.

Wenn $h = m(v,h)$ dann dürfen wir h nicht löschen.

- Der gemeinsame Hub von h und v muss höher sein als h und v
- Der höchste Knoten auf dem kürzesten vh-Pfad ist h
- $\Longrightarrow$ Nur h selber kann der gemeinsame Hub sein
- $\Longrightarrow$ h darf nicht gelöscht werden

Labels Kanonisch Machen

Satz

Wir können einen Hub h aus dem Label $L(v)$ von v löschen genau dann wenn $h \neq m(v, h)$.

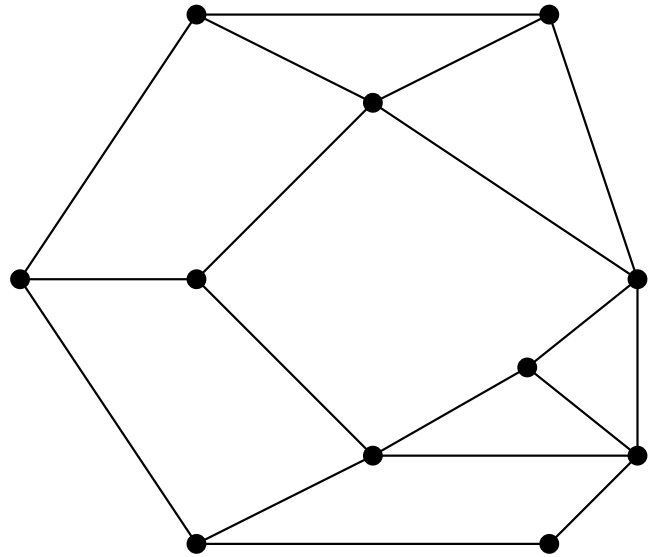
Wenn $h \neq m(v, h)$ dann dürfen wir h löschen.

- Label von v is nur relevant für Pfade die bei v beginnen (oder enden) und durch h gehen
- Sei t der Endpunkt eines solchen Pfads
- Auf all diesen Pfaden gibt es einen höheren Knoten, nämlich $h' = m(v, h)$, da der vh-Pfad Teilpfad ist
- Damit der vh'-Pfad und die $h't$-Pfade gefunden werden muss h' im Label von v und allen t sein
- Wir können h löschen, da für jede Anfrage wo h gemeinsamer Hub ist, ist auch h' gemeinsamer Hub

Vorberechnung

Idee:

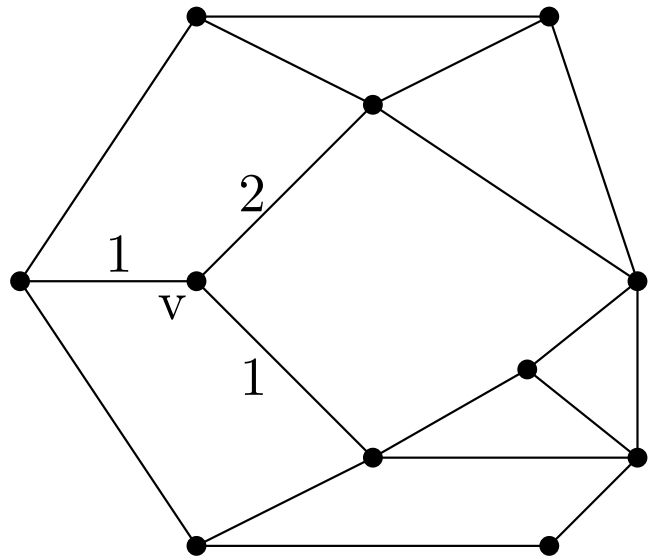
- benutze Knotenordnung



Vorberechnung

Idee:

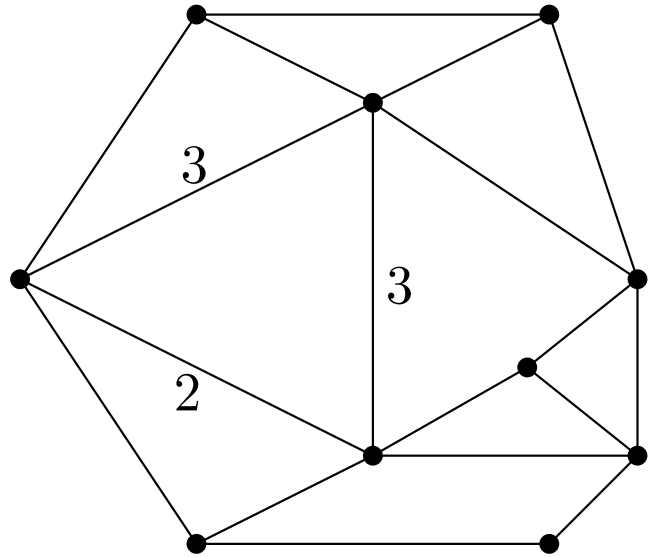
- benutze Knotenordnung
- kontrahiere Knoten v



Vorberechnung

Idee:

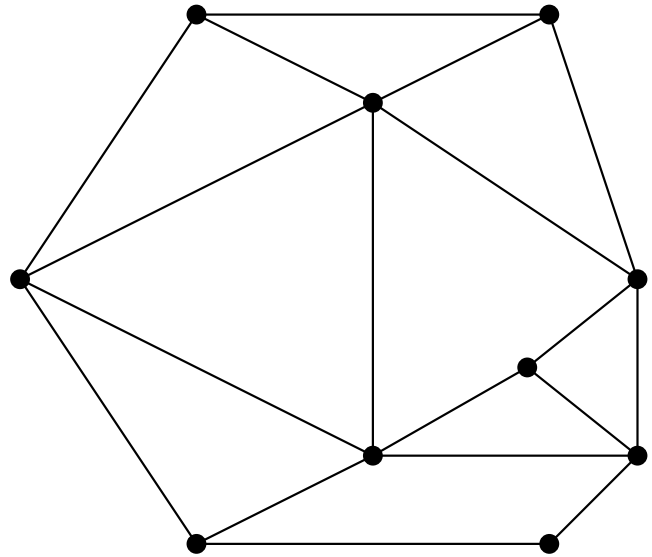
- benutze Knotenordnung
- kontrahiere Knoten v



Vorberechnung

Idee:

- benutze Knotenordnung
- kontrahiere Knoten v

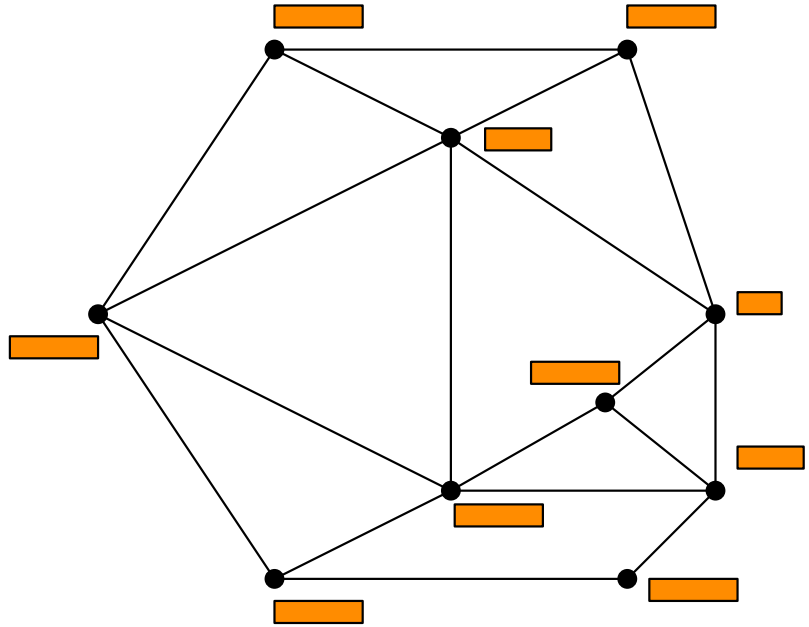


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Vorberechnung

Idee:

- benutze Knotenordnung
- kontrahiere Knoten v
- berechne Labels rekursiv

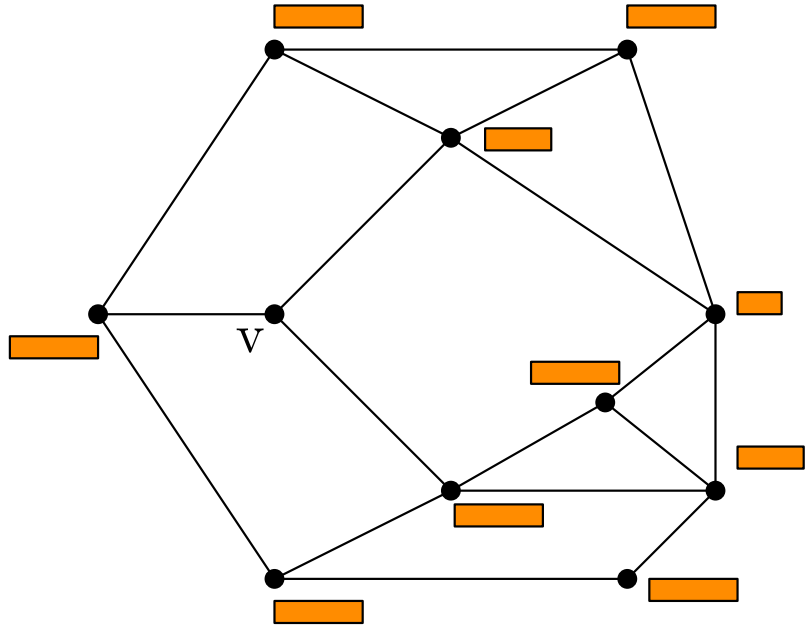


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Vorberechnung

Idee:

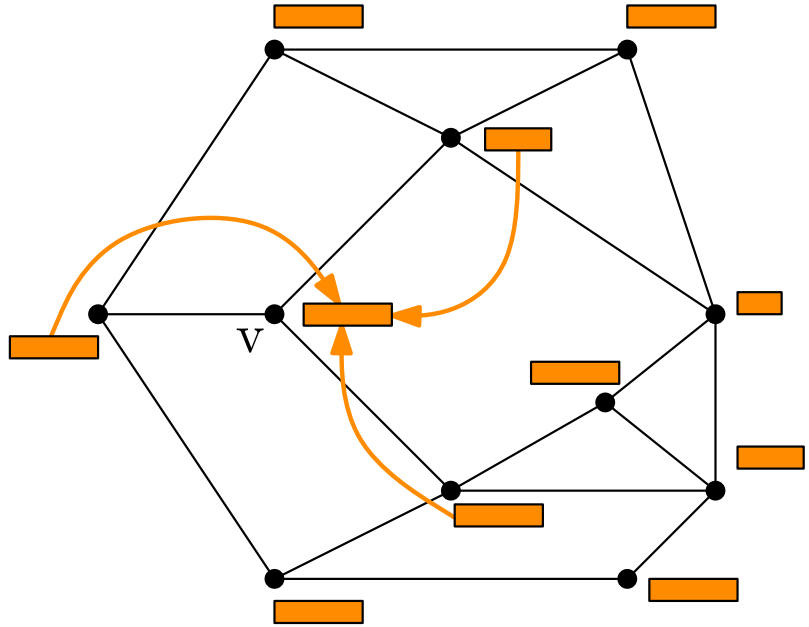
- benutze Knotenordnung
- kontrahiere Knoten v
- berechne Labels rekursiv



Vorberechnung

Idee:

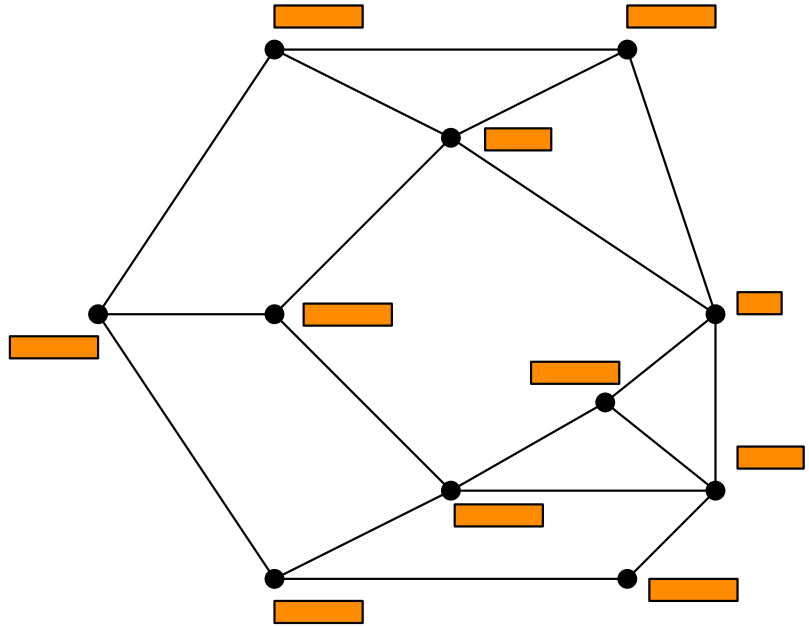
- benutze Knotenordnung
- kontrahiere Knoten v
- berechne Labels rekursiv
- merge Labels der Aufwärts-Nachbarn von v
- dünne Label aus



Vorberechnung

Idee:

- benutze Knotenordnung
- kontrahiere Knoten v
- berechne Labels rekursiv
- 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

Mergen von Labels

Generell:

- $L_f(v) = \bigcup_{(v,u) \in G^+} (L_f(u) + (v,u))$
- wenn ein Hub mehrfach im resultierendem Label, behalte nur den mit minimaler Distanz

Mergen von Labels

Generell:

- $L_f(v) = \bigcup_{(v,u)\in G^+}(L_f(u) + (v,u))$
- wenn ein Hub mehrfach im resultierendem Label, behalte nur den mit minimaler Distanz

Ausdünnen:

- manche Knoten im Label sind nicht notwendig
- **Ziel**: Für jeden st-Pfad reicht es wenn der Zwischenknoten h mit dem höchsten Rank in $L_f(h)$ und $L_b(h)$ liegt
- Es sei h ein Knoten in $L_f(s)$. Es sei h' der höchste Knoten have dem kürzesten sh-Pfad.
- Wenn $h \neq h'$ dann ist h nie der höchste Knoten auf einem kürzesten st-Pfad durch h
  $\Rightarrow$ Wir können h rauslöschen.

Pruned Labeling:[DGPW14, AIY13] Alternative Labelkonstruktion

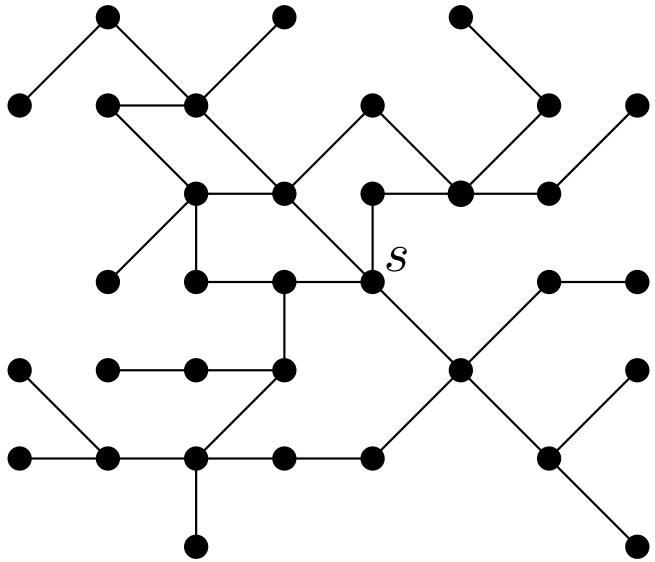
Idee:

- Verteile Hubs auf Labels absteigend nach Knotenordnung
- h ist Hub von $v \iff$ es auf dem hv-Pfad keinen höheren Knoten gibt
- Starte Dijkstra von h und besuche alle v in denen h liegt, breche alle Pfade über höhere Knoten führen

Pruned Labeling:[DGPW14, AIY13] Alternative Labelkonstruktion

Idee:

- Verteile Hubs auf Labels absteigend nach Knotenordnung
- h ist Hub von $v \iff$ es auf dem hv-Pfad keinen höheren Knoten gibt
- Starte Dijkstra von h und besuche alle v in denen h liegt, breche alle Pfade über höhere Knoten führen

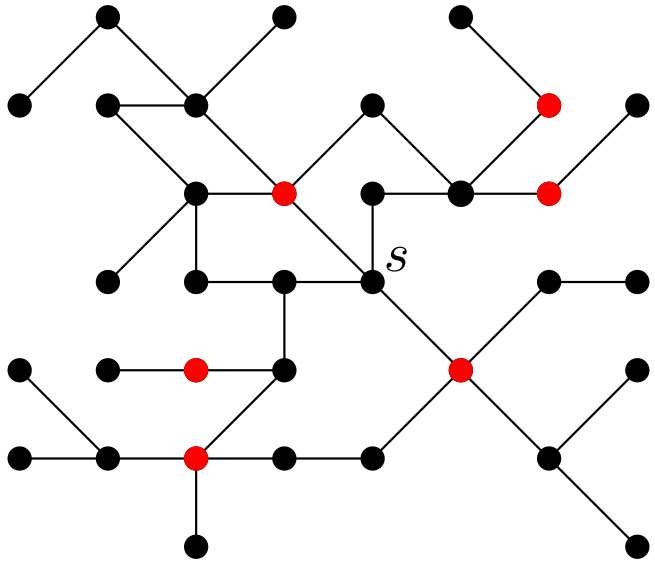


Ziel: s in alle Labels verteilen

Pruned Labeling:[DGPW14, AIY13] Alternative Labelkonstruktion

Idee:

- Verteile Hubs auf Labels absteigend nach Knotenordnung
- h ist Hub von $v \iff$ es auf dem hv-Pfad keinen höheren Knoten gibt
- Starte Dijkstra von h und besuche alle v in denen h liegt, breche alle Pfade über höhere Knoten führen

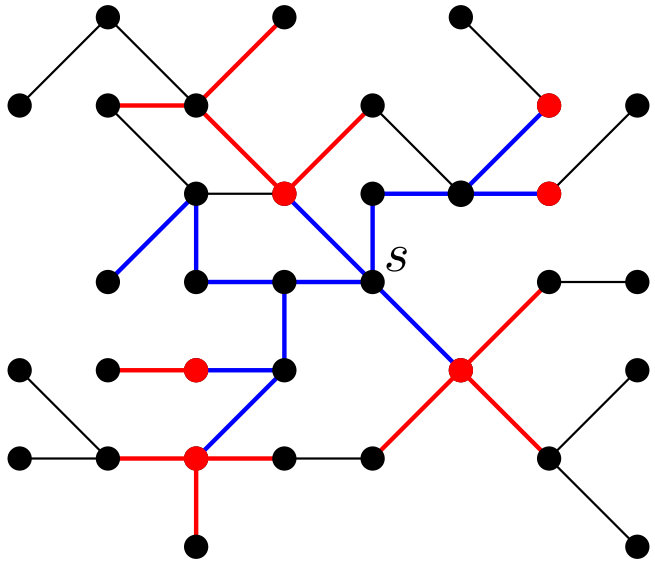


Rote Knoten haben höheren Rank

Pruned Labeling:[DGPW14, AIY13] Alternative Labelkonstruktion

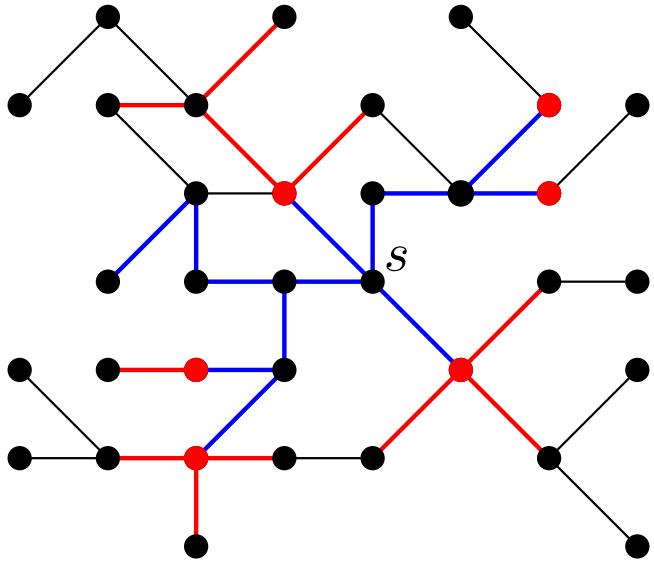
Idee:

- Verteile Hubs auf Labels absteigend nach Knotenordnung
- h ist Hub von $v \iff$ es auf dem hv-Pfad keinen höheren Knoten gibt
- Starte Dijkstra von h und besuche alle v in denen h liegt, breche alle Pfade über höhere Knoten führen



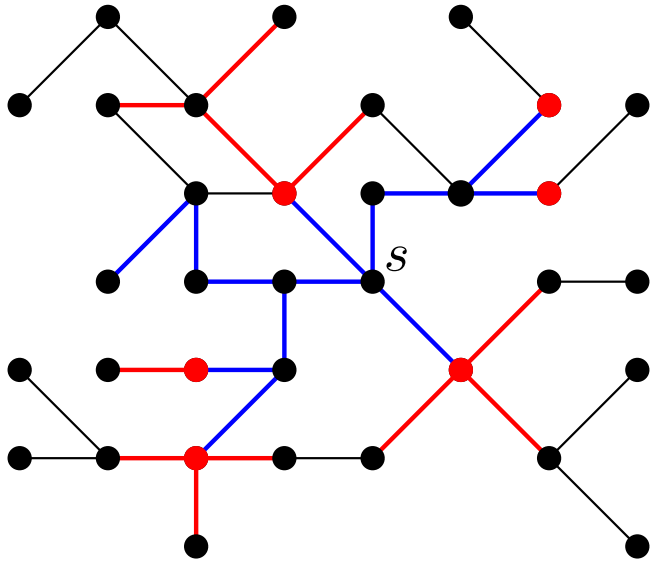
s kommt in über blaue Pfade erreichbaren Label

Pruned Labeling:[DGPW14, AIY13]



- Knoten x ist rot $\iff$ $\exists$ xs-Pfad durch einen roten Knoten
- Bereits aufgebauten partiellen Labels reichen um dies zu testen
- $\rightarrow$ HL-Query auf partiellen Labels kann als Dijkstra-Pruning Regel benutzt werden
- $\rightarrow$ Nur noch der blaue Teilbaum wird besucht
- Pruning macht den Algorithmus leicht schneller

Pruned Labeling:[DGPW14, AIY13]



- Knoten x ist rot $\iff \exists$ xs-Pfad durch einen roten Knoten
- Bereits aufgebauten partiellen Labels reichen um dies zu testen
- $\rightarrow$ HL-Query auf partiellen Labels kann als Dijkstra-Pruning Regel benutzt werden
- $\rightarrow$ Nur noch der blaue Teilbaum wird besucht
- Pruning macht den Algorithmus leicht schneller
- Pruned Labeling funktioniert besser auf Graphen wo der Core Knoten mit hohen Grad enthält, als der Grundalgorithmus.

Partitionen Ausnutzen

Idee:

- permutiere hub IDs so, dass Hubs mit hohem Rank kleine IDs haben

$\Rightarrow$ sind somit am Anfang des Labels

- partitioniere die Eingabe (wieder mal...)
- für jedes (q, r)-Paar Regionen
  - betrachte alle Pfade von Zelle q nach Zelle r
  - bestimme den verwendeten Hub mit minimalen Rank
  - speichere dessen ID in einer Tabelle (Orakel)

Partitionen Ausnutzen

Idee:

- permutiere hub IDs so, dass Hubs mit hohem Rank kleine IDs haben

$\Rightarrow$ sind somit am Anfang des Labels

- partitioniere die Eingabe (wieder mal...)
- für jedes (q, r)-Paar Regionen
  - betrachte alle Pfade von Zelle q nach Zelle r
  - bestimme den verwendeten Hub mit minimalen Rank
  - speichere dessen ID in einer Tabelle (Orakel)

Partitionen Ausnutzen

Idee:

- permutiere hub IDs so, dass Hubs mit hohem Rank kleine IDs haben

$\Rightarrow$ sind somit am Anfang des Labels

- partitioniere die Eingabe (wieder mal...)
- für jedes (q, r)-Paar Regionen
  - betrachte alle Pfade von Zelle q nach Zelle r
  - bestimme den verwendeten Hub mit minimalen Rank
  - speichere dessen ID in einer Tabelle (Orakel)

Anfrage:

- bestimme minimale Hub-ID mit Orakel-Tabelle
- stoppe Iteration der Labels nach dieser ID

$\Rightarrow$ beschleunigt globale Anfragen

$L(s)$	2	3	6	35	37	102	155	172

$Or(s,t)$

$L(t)$	2	6	8	43	45	85		10

Partitionen Ausnutzen

Idee:

- permutiere hub IDs so, dass Hubs mit hohem Rank kleine IDs haben

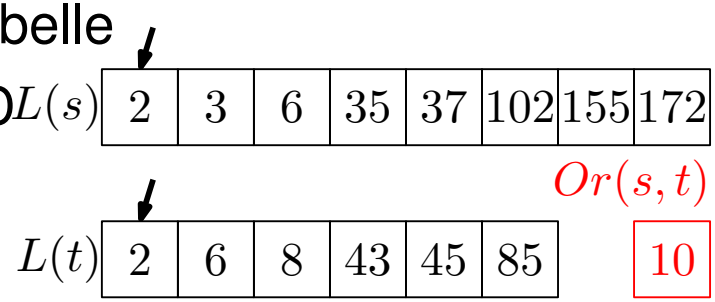
$\Rightarrow$ sind somit am Anfang des Labels

- partitioniere die Eingabe (wieder mal...)
- für jedes (q, r)-Paar Regionen
  - betrachte alle Pfade von Zelle q nach Zelle r
  - bestimme den verwendeten Hub mit minimalen Rank
  - speichere dessen ID in einer Tabelle (Orakel)

Anfrage:

- bestimme minimale Hub-ID mit Orakel-Tabelle
- stoppe Iteration der Labels nach dieser ID

$\Rightarrow$ beschleunigt globale Anfragen



Partitionen Ausnutzen

Idee:

- permutiere hub IDs so, dass Hubs mit hohem Rank kleine IDs haben

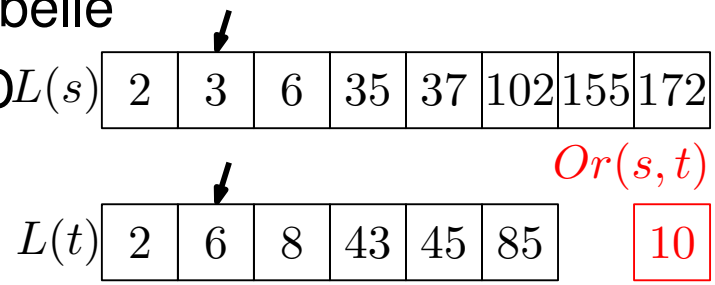
$\Rightarrow$ sind somit am Anfang des Labels

- partitioniere die Eingabe (wieder mal...)
- für jedes (q, r)-Paar Regionen
  - betrachte alle Pfade von Zelle q nach Zelle r
  - bestimme den verwendeten Hub mit minimalen Rank
  - speichere dessen ID in einer Tabelle (Orakel)

Anfrage:

- bestimme minimale Hub-ID mit Orakel-Tabelle
- stoppe Iteration der Labels nach dieser ID

$\Rightarrow$ beschleunigt globale Anfragen



Partitionen Ausnutzen

Idee:

- permutiere hub IDs so, dass Hubs mit hohem Rank kleine IDs haben

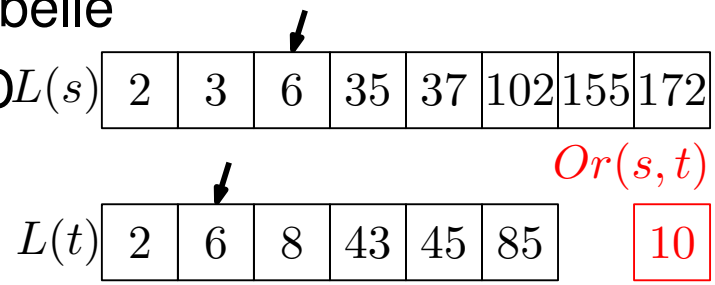
$\Rightarrow$ sind somit am Anfang des Labels

- partitioniere die Eingabe (wieder mal...)
- für jedes (q, r)-Paar Regionen
  - betrachte alle Pfade von Zelle q nach Zelle r
  - bestimme den verwendeten Hub mit minimalen Rank
  - speichere dessen ID in einer Tabelle (Orakel)

Anfrage:

- bestimme minimale Hub-ID mit Orakel-Tabelle
- stoppe Iteration der Labels nach dieser ID

$\Rightarrow$ beschleunigt globale Anfragen



Partitionen Ausnutzen

Idee:

- permutiere hub IDs so, dass Hubs mit hohem Rank kleine IDs haben

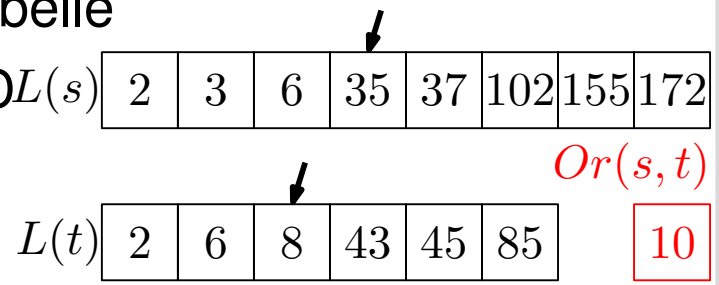
$\Rightarrow$ sind somit am Anfang des Labels

- partitioniere die Eingabe (wieder mal...)
- für jedes (q, r)-Paar Regionen
  - betrachte alle Pfade von Zelle q nach Zelle r
  - bestimme den verwendeten Hub mit minimalen Rank
  - speichere dessen ID in einer Tabelle (Orakel)

Anfrage:

- bestimme minimale Hub-ID mit Orakel-Tabelle
- stoppe Iteration der Labels nach dieser ID

$\Rightarrow$ beschleunigt globale Anfragen



Label Greedy - Ordnung [ADGW12]

Sei U_v die Menge der nicht überdeckten Pfaden auf denen v liegt.

Zur Erinnerung:

- Knoten werden bei Path-Greedy nach Anzahl überdeckter Pfade geordnet
- d.h. wähle v, so dass $|U_v|$ minimal ist

Neue Idee:

- Sei S_v Startknoten eines Pfads in U_v, T_v analog
- Wähle Knoten v, so dass $|S_v| + |T_v|$ minimal ist
- Noch besser: Wähle v, so dass $|U_v|/(|S_v| + |T_v|)$ maximal ist

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

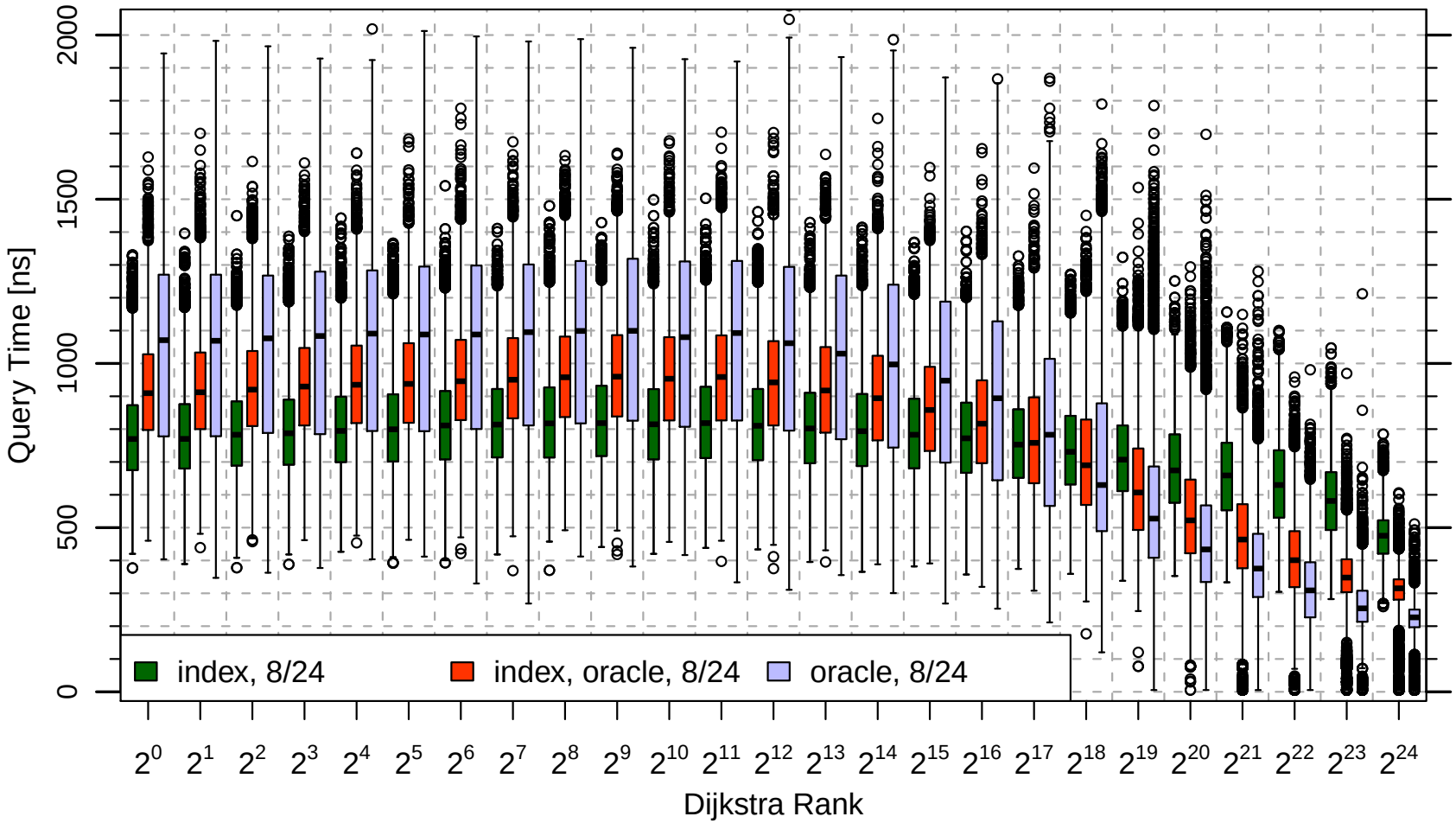
Ergebnisse

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

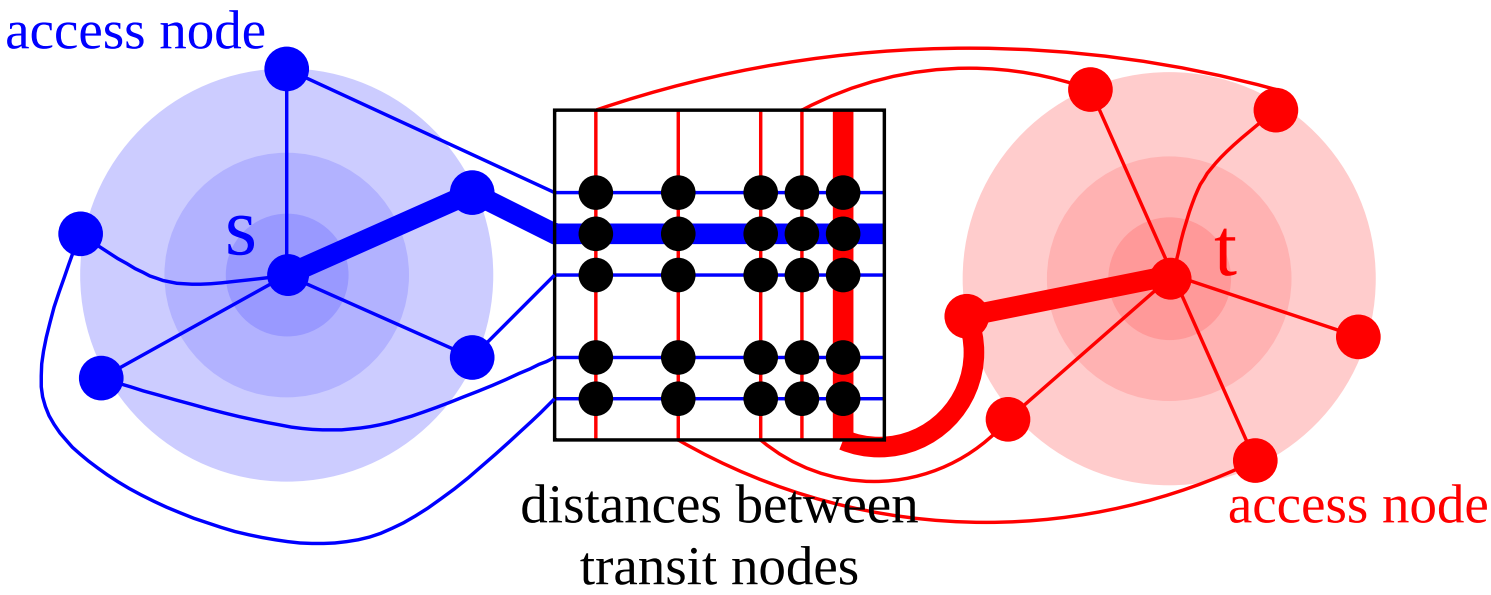
Lokale Queries



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
- wird zu einem späterem Zeitpunkt noch einmal wichtig

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

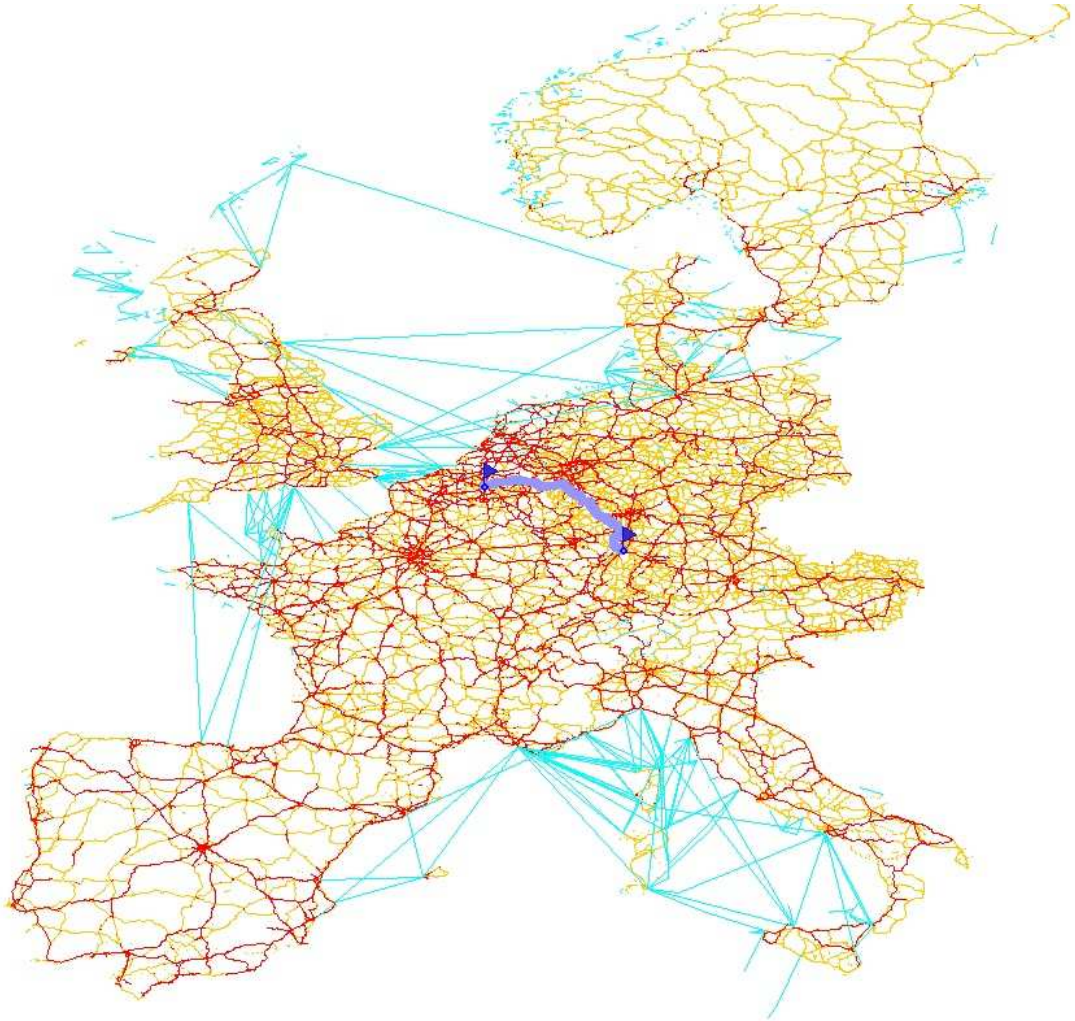
Transit-Node Routing

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

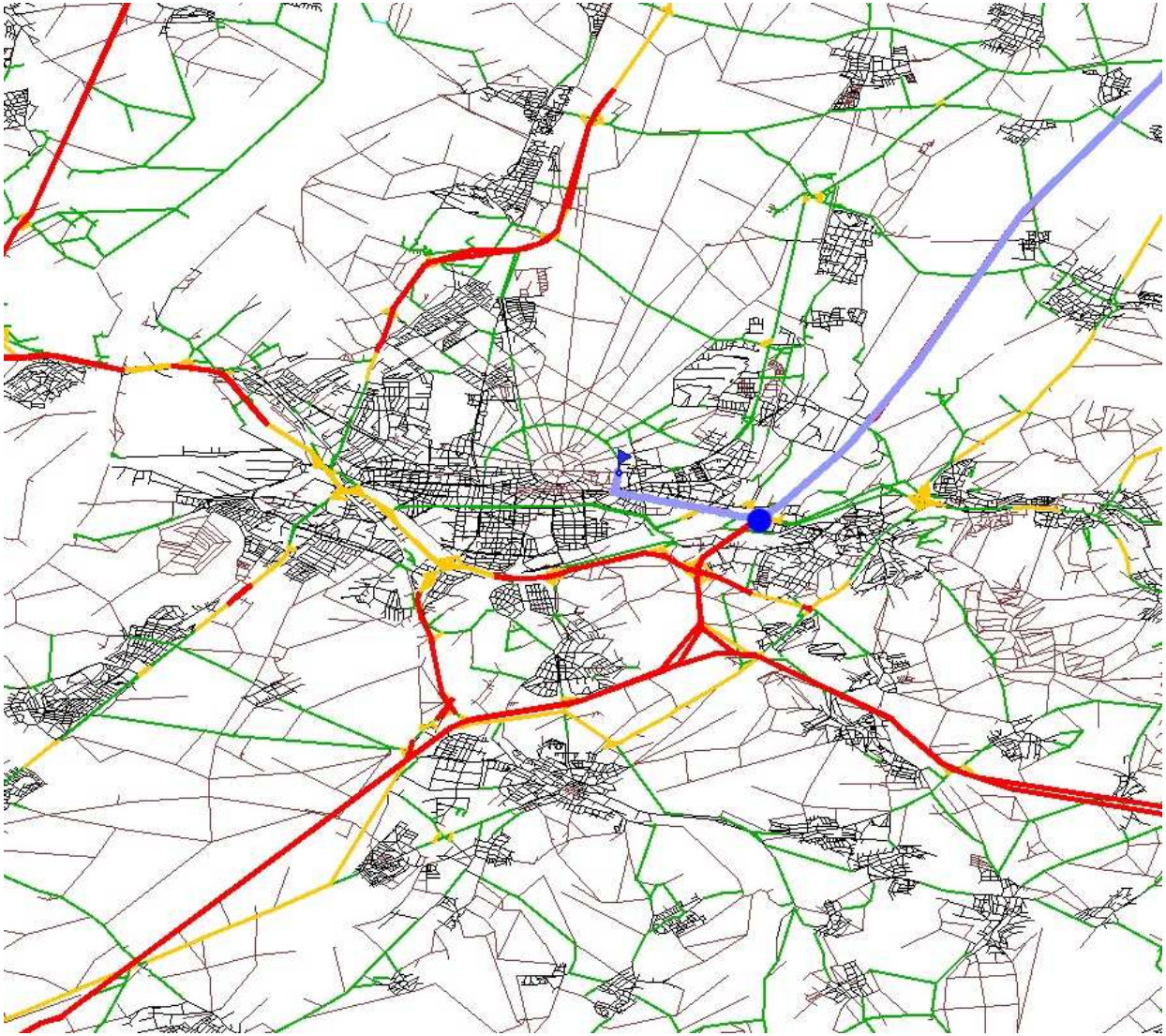


Beobachtung:

- 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

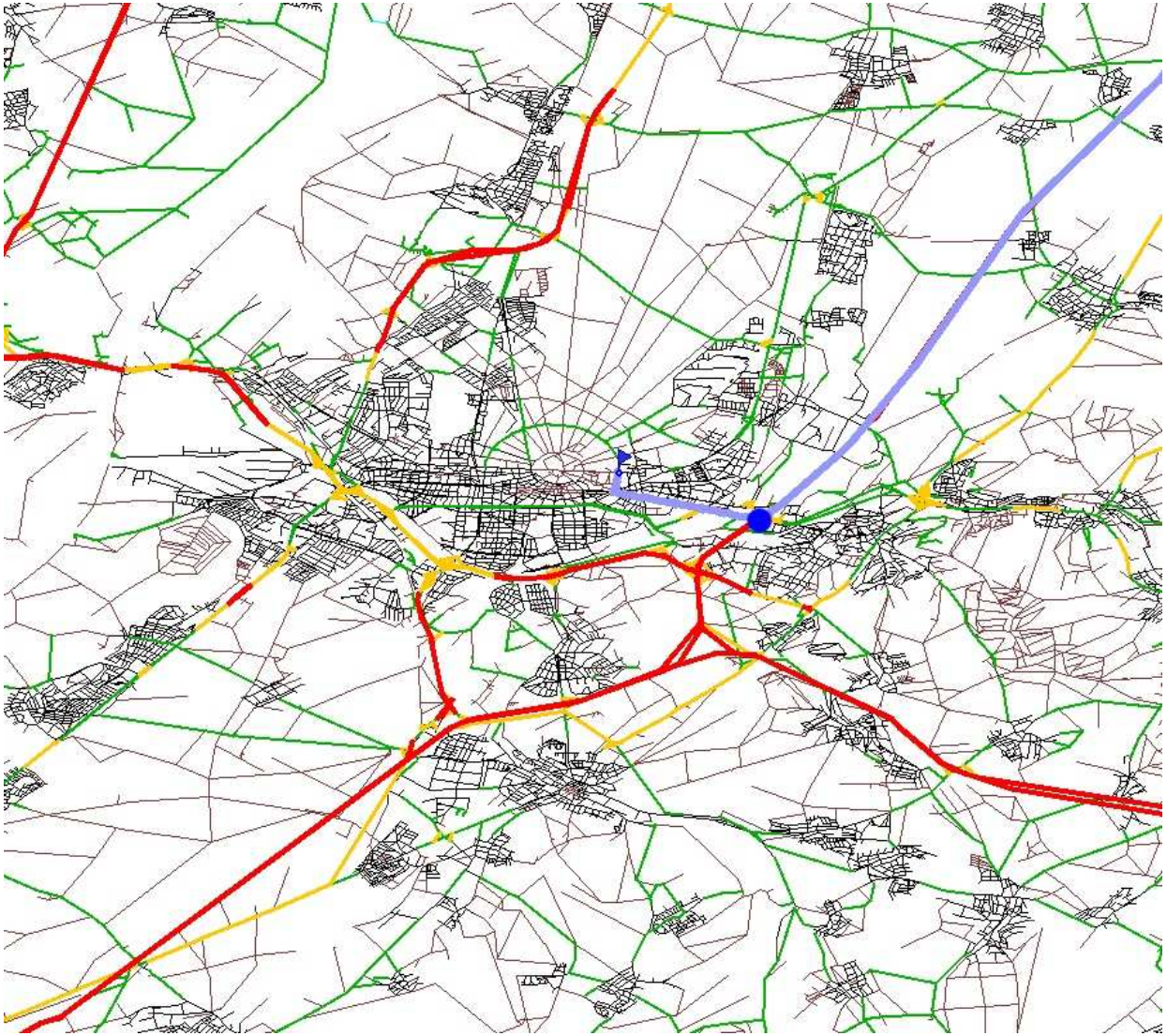


Beobachtung:

- 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

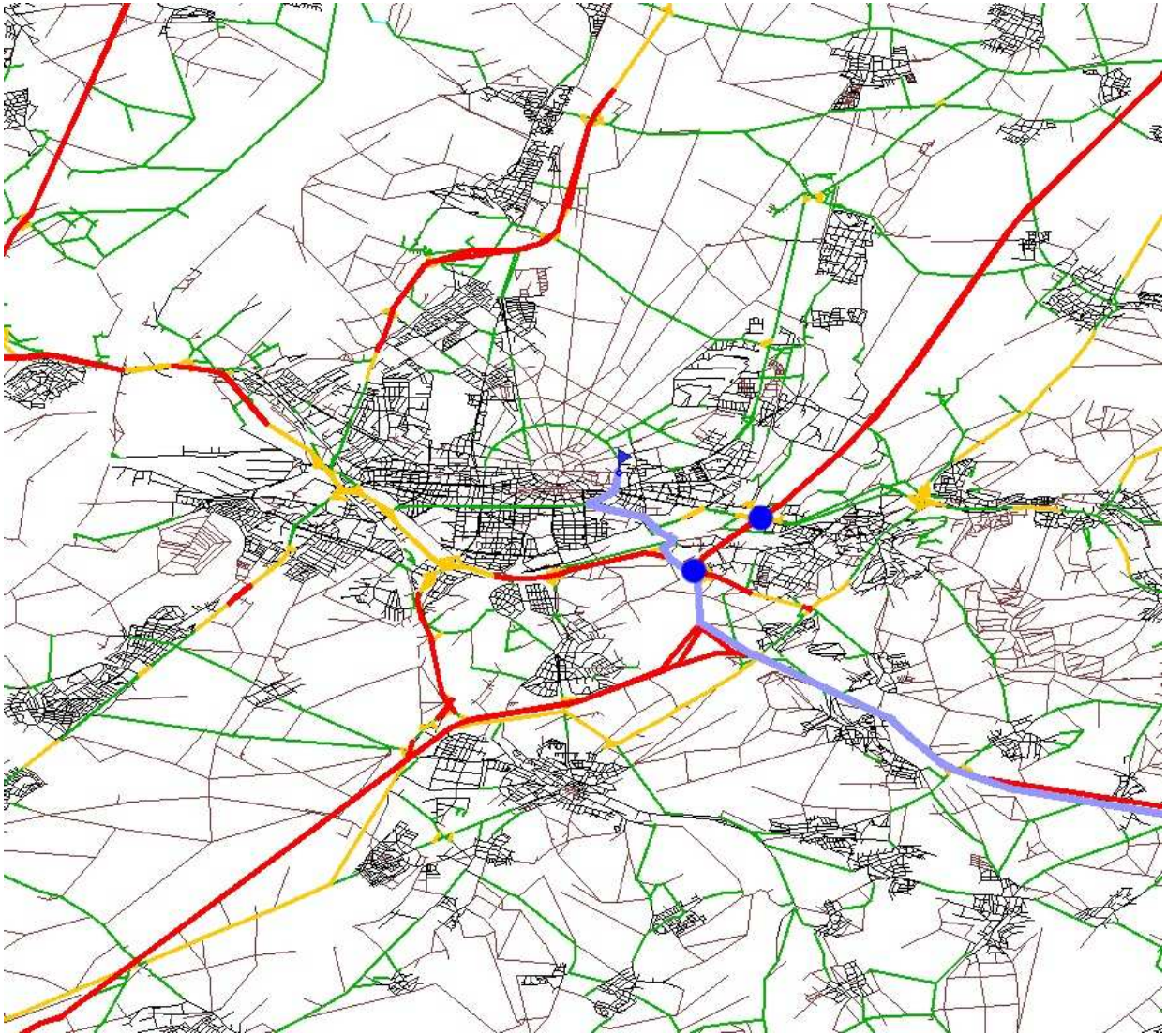


Beobachtung:

- 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

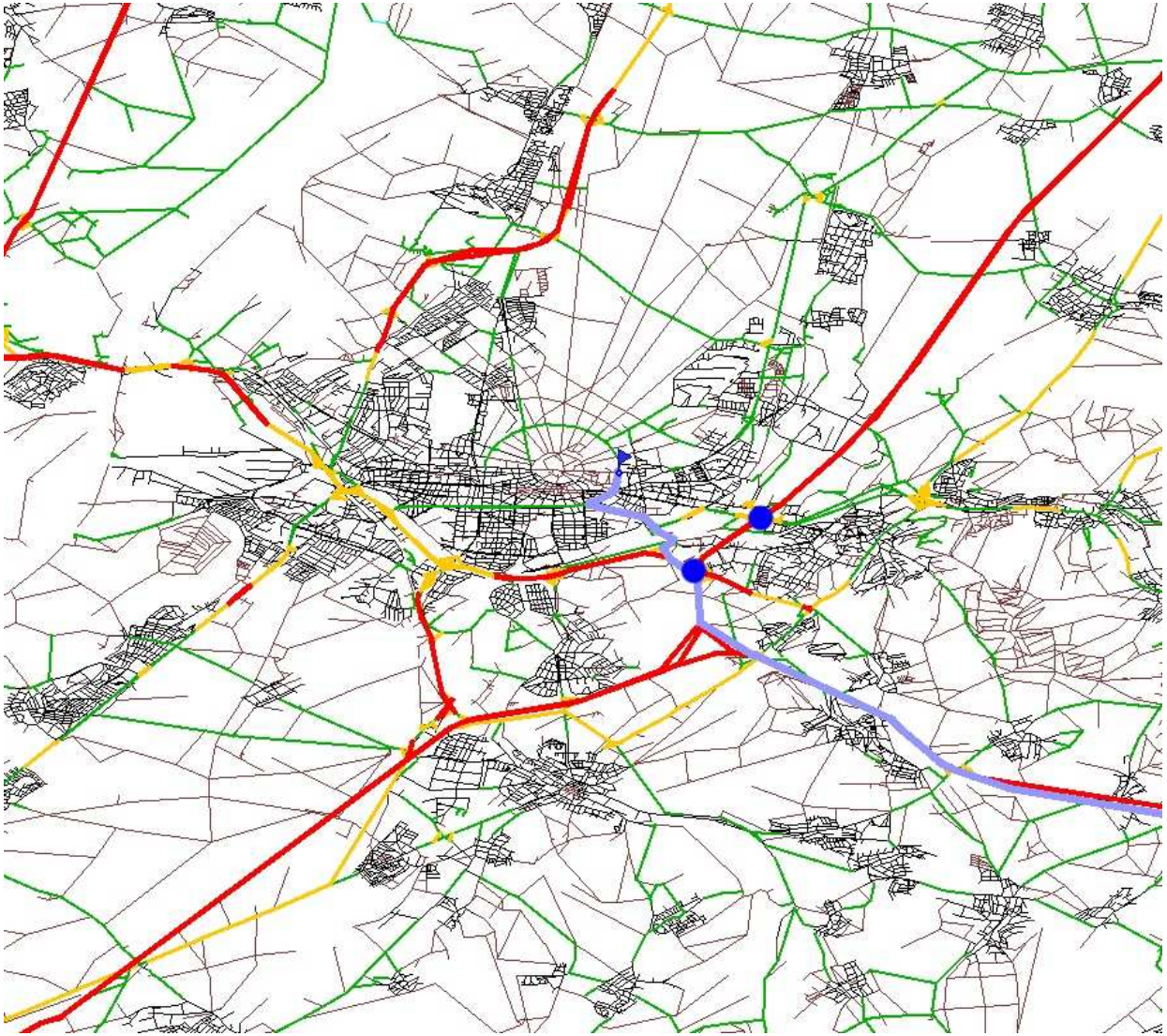


Beobachtung:

- 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

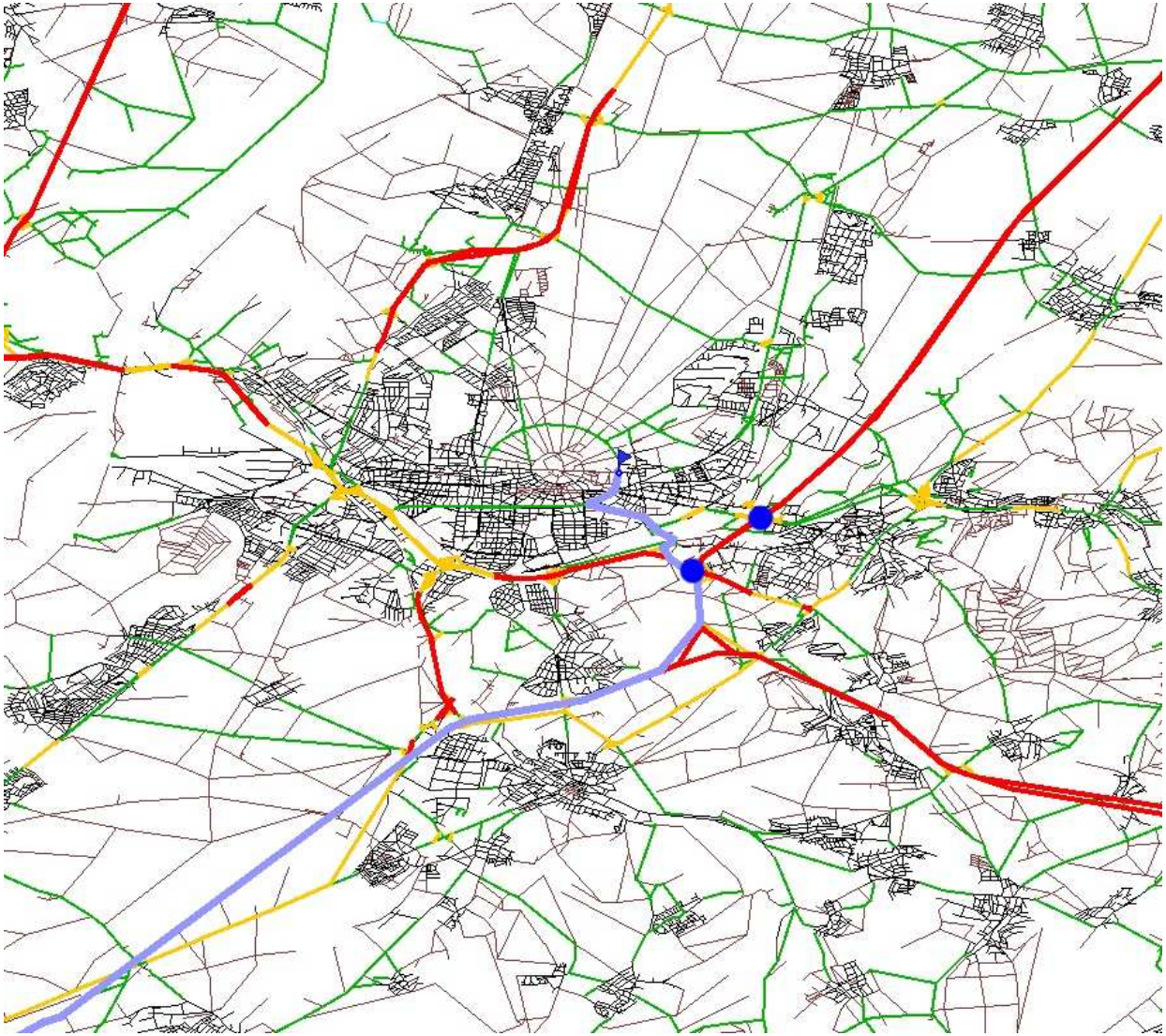


Beobachtung:

- 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

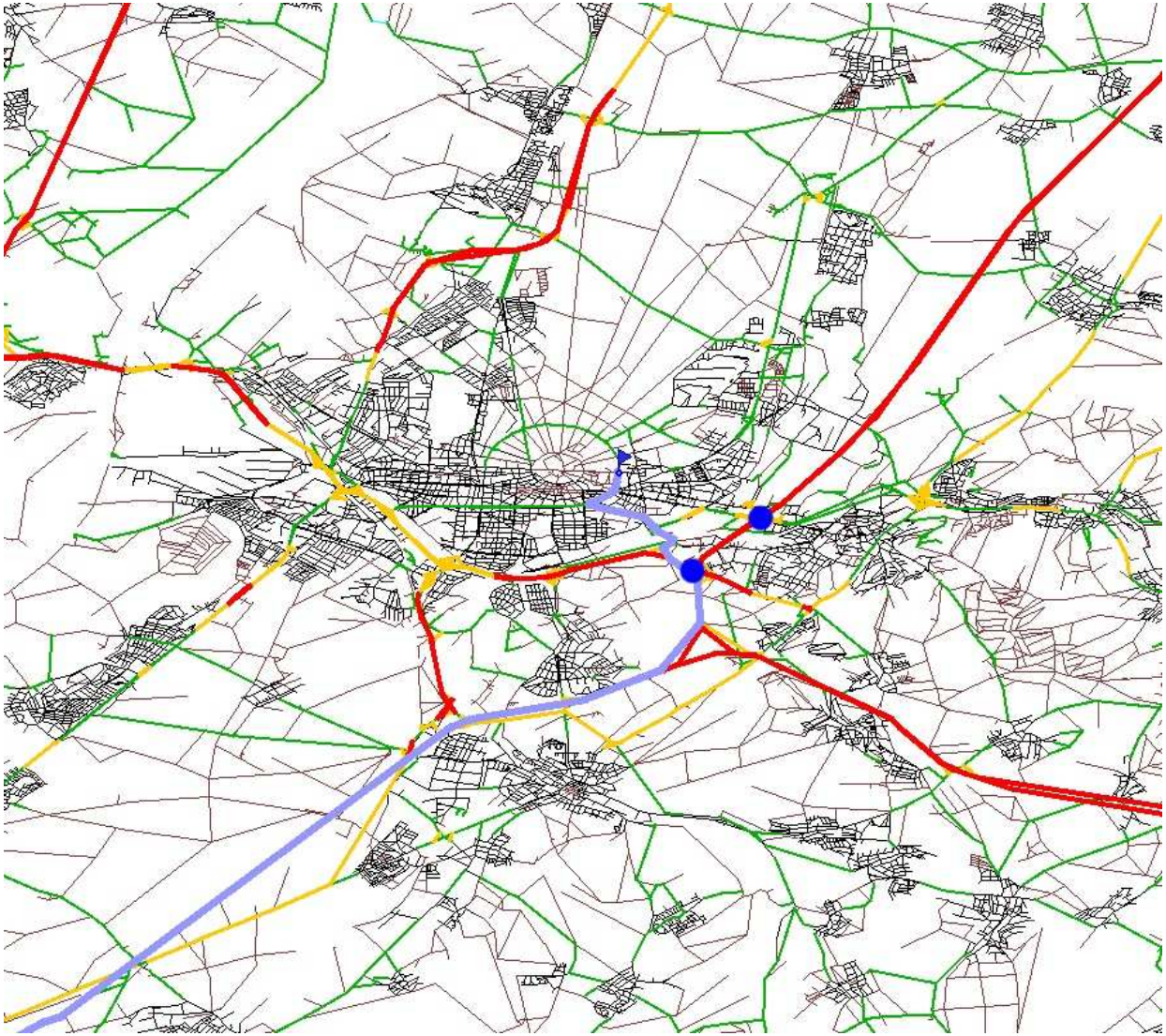


Beobachtung:

- 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

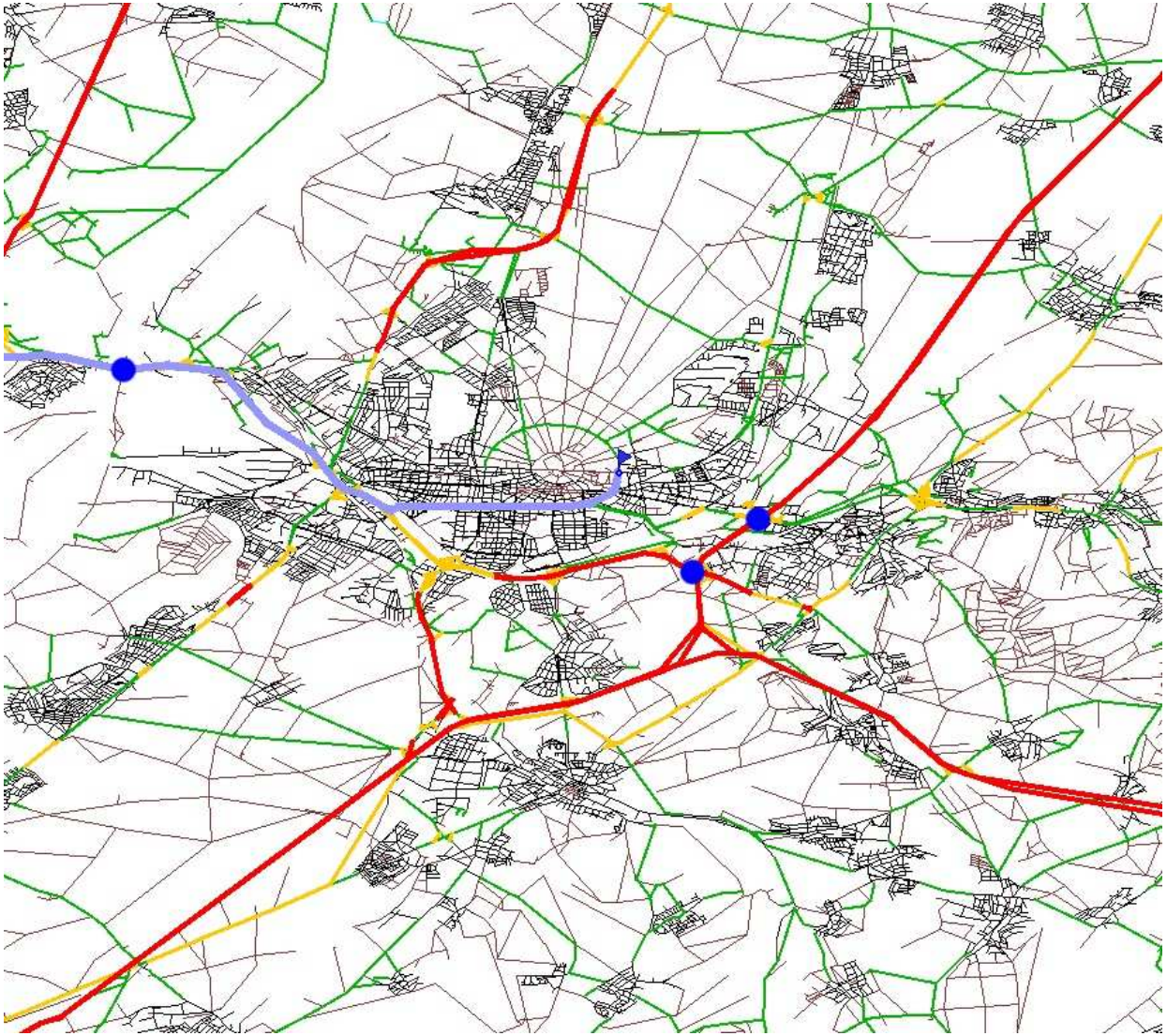


Beobachtung:

- 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

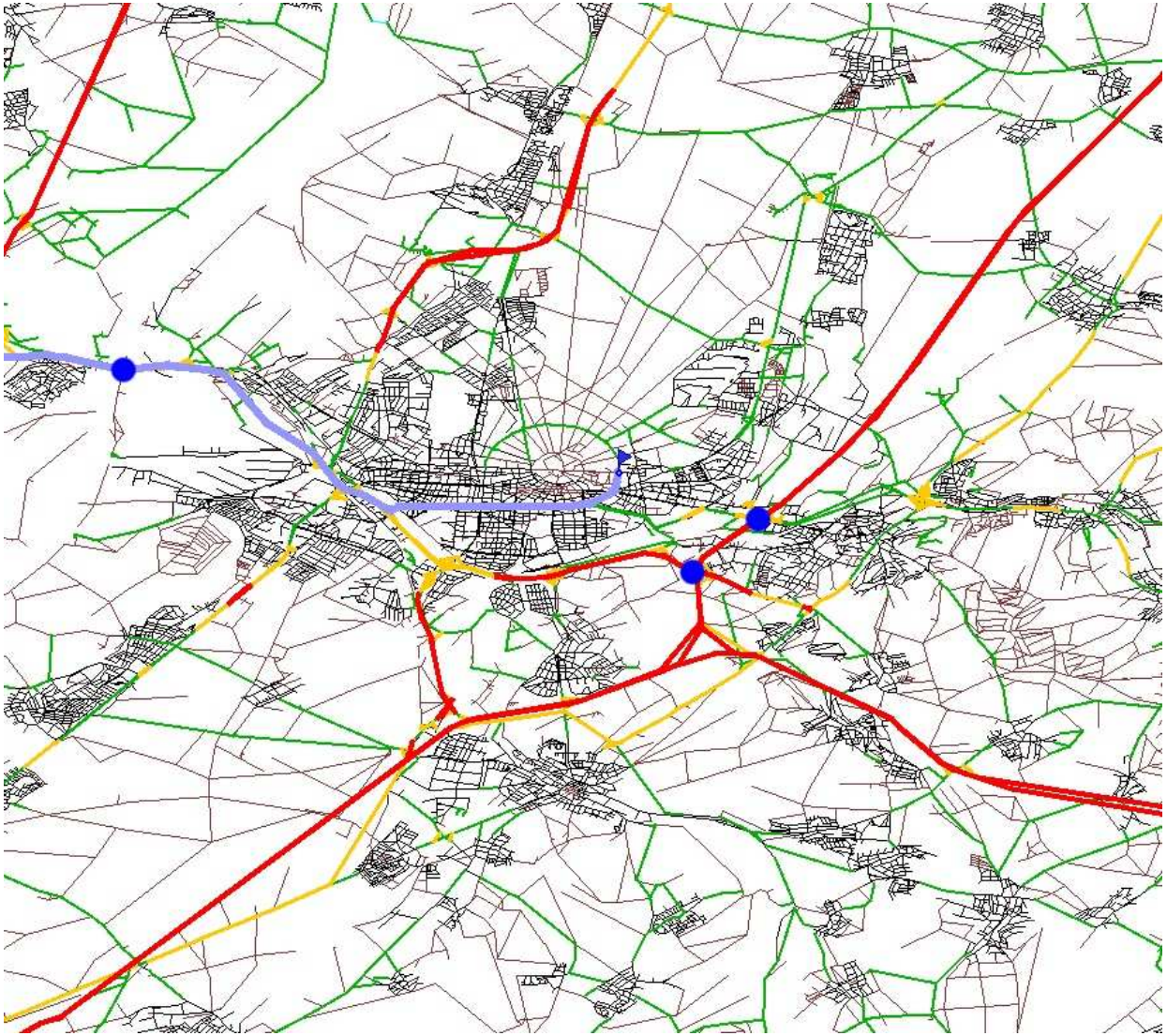


Beobachtung:

- 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

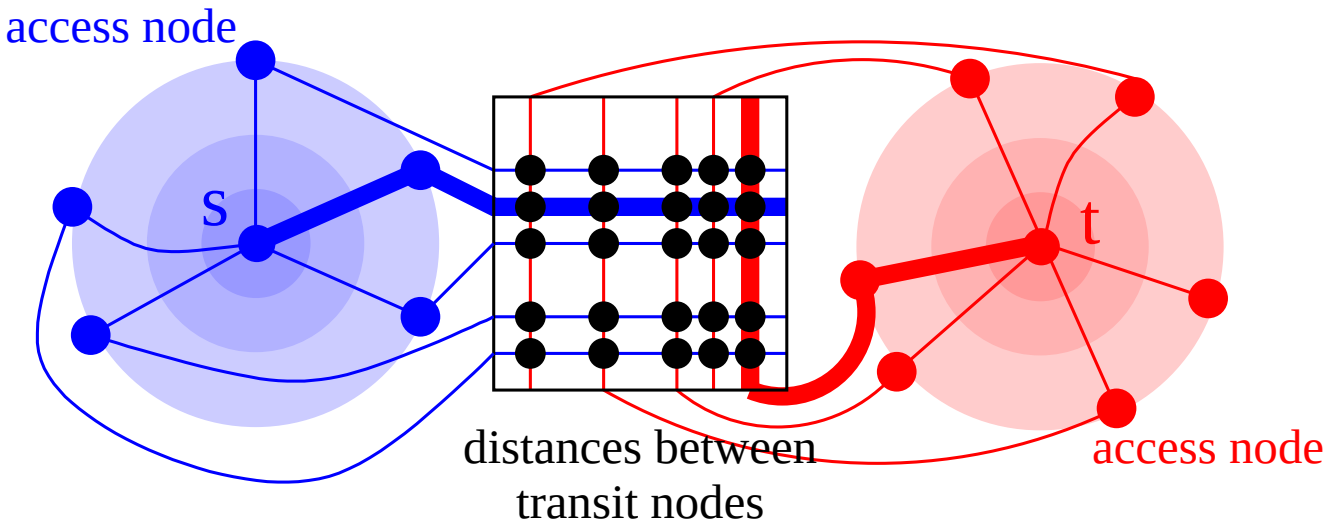
Transit-Node Routing

Idee:

- reduziere Anfragen auf Table-Lookups
- identifiziere “wichtige” Knoten
- vollständige Distanztabellen zwischen diesen Knoten

Probleme:

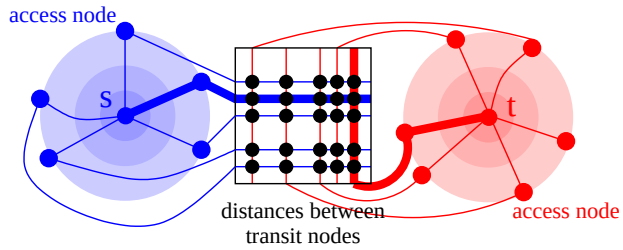
- Speicherverbrauch
- nahe Anfragen



Generelles TNR-Framework

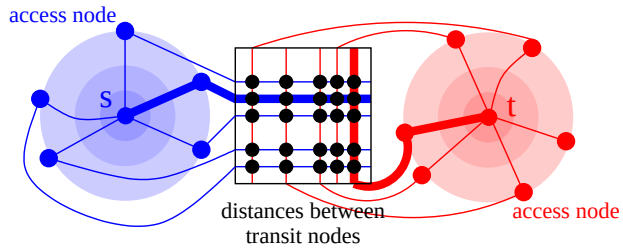
- Wähle transit nodes: $T \subseteq V$
- Bestimme für jeden Knoten v eine Menge von Vorwärts $\overrightarrow{A}(v)$ und Rückwärts $\overleftarrow{A}(v)$ access nodes
- Vorberechnete Distanzen: D_T und d_A

Generelles TNR-Framework



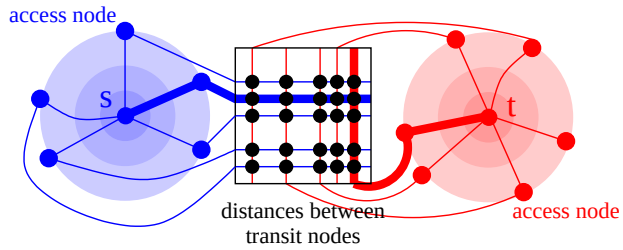
- Wähle transit nodes: $T \subseteq V$
- Bestimme für jeden Knoten v eine Menge von Vorwärts $\overrightarrow{A}(v)$ und Rückwärts $\overleftarrow{A}(v)$ access nodes
- Vorberechnete Distanzen: D_T und d_A

Generelles TNR-Framework



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

Generelles TNR-Framework



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

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-Nodes bestimmen?
- Wie Access-Nodes und deren Distanz bestimmen?
- Wie Distanztabelle zwischen Transit-Nodes berechnen?
- Welcher Locality-Filter?
- Wie lokale Anfragen berechnen?

Zutaten TNR-Verfahren

Also:

- Wie Transit-Nodes bestimmen?
- Wie Access-Nodes und deren Distanz bestimmen?
- Wie Distanztabelle zwischen Transit-Nodes berechnen?
- Welcher Locality-Filter?
- Wie lokale Anfragen berechnen?

Ideen?

Zutaten TNR-Verfahren

Also:

- Wie Transit-Nodes bestimmen?
- Wie Access-Nodes und deren Distanz bestimmen?
- Wie Distanztabelle zwischen Transit-Nodes berechnen?
- Welcher Locality-Filter?
- 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

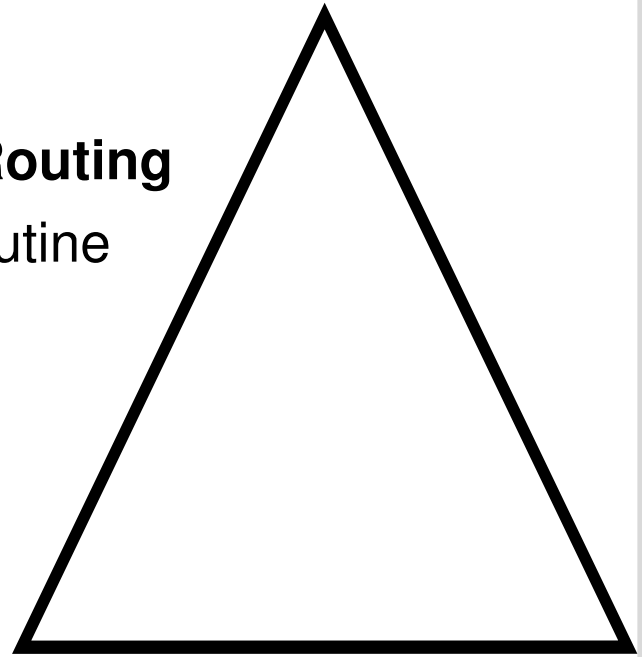
Contraction Hierarchies based Transit Node Routing

- Use CH for preprocessing and as fallback routine
- Take top k nodes as Transit nodes...

CH-TNR

Contraction Hierarchies based Transit Node Routing

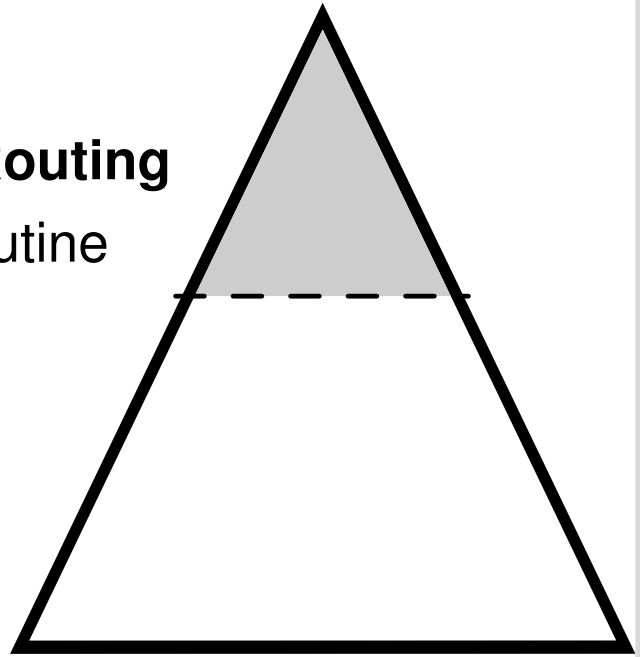
- Use CH for preprocessing and as fallback routine
- Take top k nodes as Transit nodes...



CH-TNR

Contraction Hierarchies based Transit Node Routing

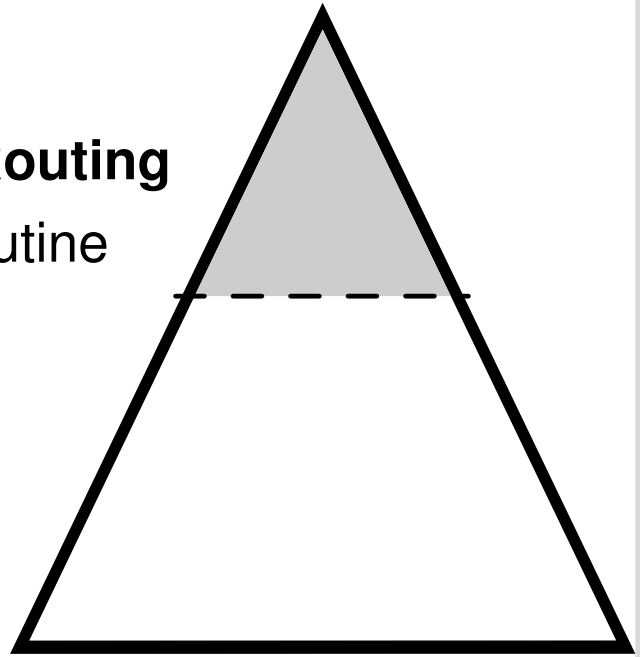
- Use CH for preprocessing and as fallback routine
- Take top k nodes as Transit nodes...



CH-TNR

Contraction Hierarchies based Transit Node Routing

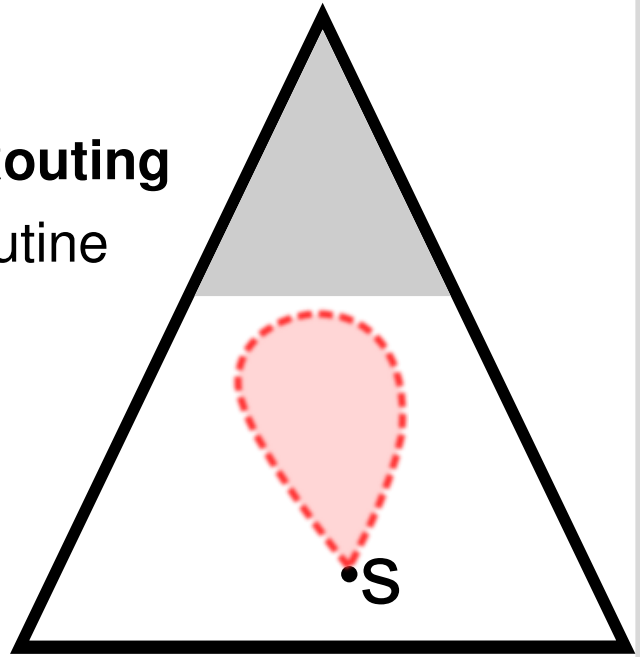
- Use CH for preprocessing and as fallback routine
- Take top k nodes as Transit nodes...
- ...and use them to compute Access nodes



CH-TNR

Contraction Hierarchies based Transit Node Routing

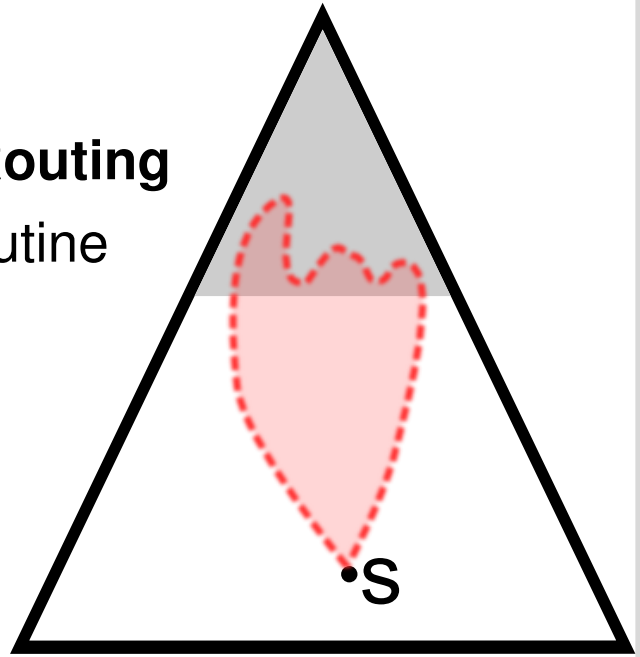
- Use CH for preprocessing and as fallback routine
- Take top k nodes as Transit nodes...
- ...and use them to compute Access nodes



CH-TNR

Contraction Hierarchies based Transit Node Routing

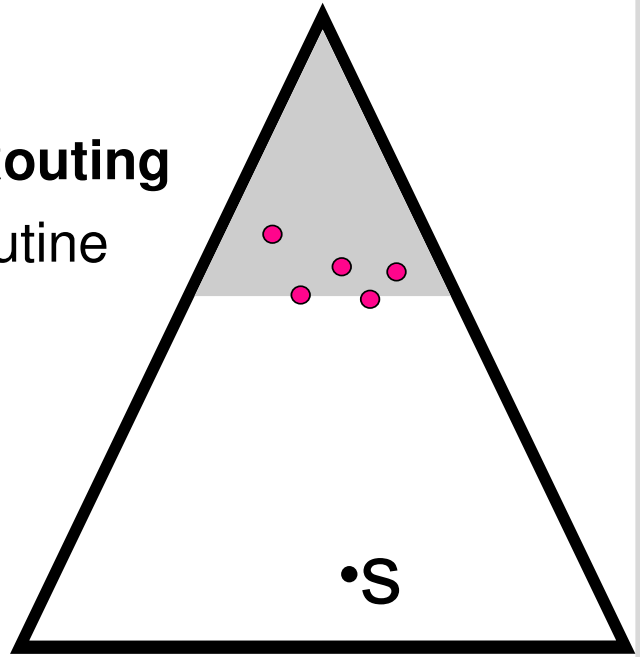
- Use CH for preprocessing and as fallback routine
- Take top k nodes as Transit nodes...
- ...and use them to compute Access nodes



CH-TNR

Contraction Hierarchies based Transit Node Routing

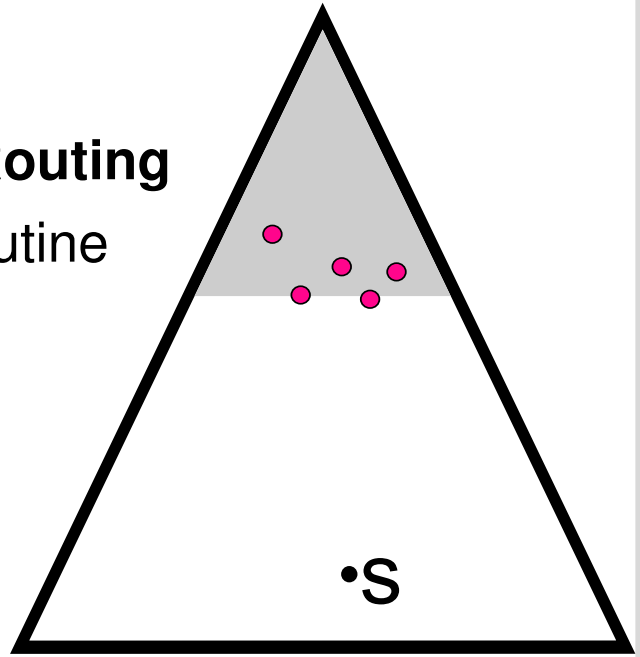
- Use CH for preprocessing and as fallback routine
- Take top k nodes as Transit nodes...
- ...and use them to compute Access nodes



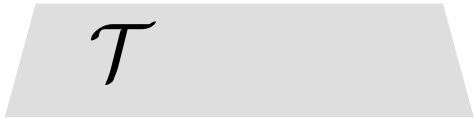
CH-TNR

Contraction Hierarchies based Transit Node Routing

- Use CH for preprocessing and as fallback routine
- Take top k nodes as Transit nodes...
- ...and use them to compute Access nodes
- Locality filter!?

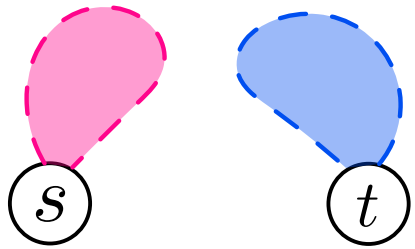


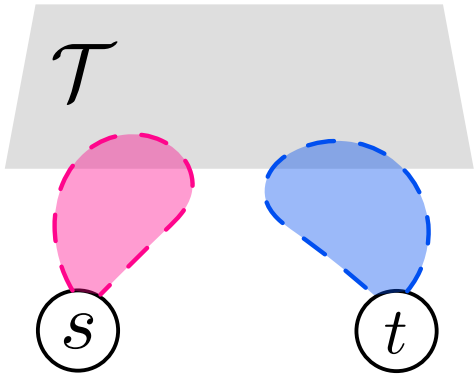
CH-based TNR



Properties of a local query

- Consider meeting point m of a $s - t$–search

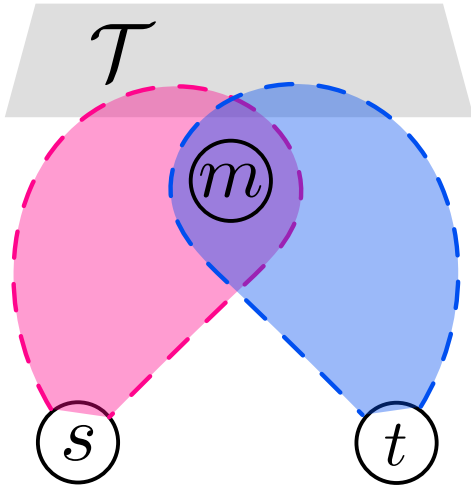


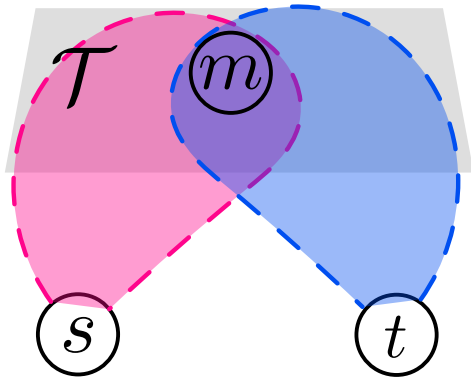


CH-based TNR

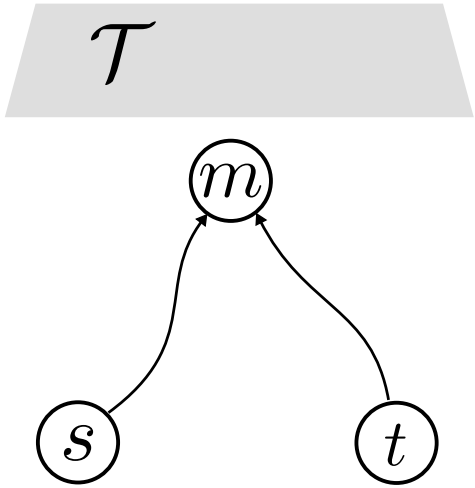
Properties of a local query

- Consider meeting point m of a $s-t$–search



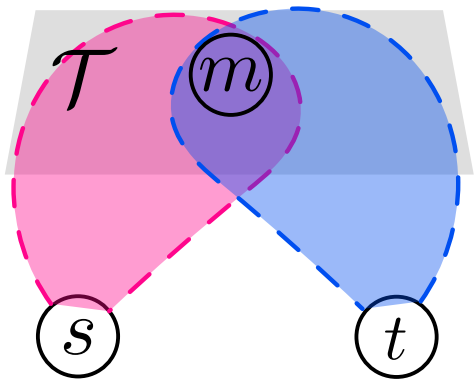


CH-based TNR



Properties of a local query

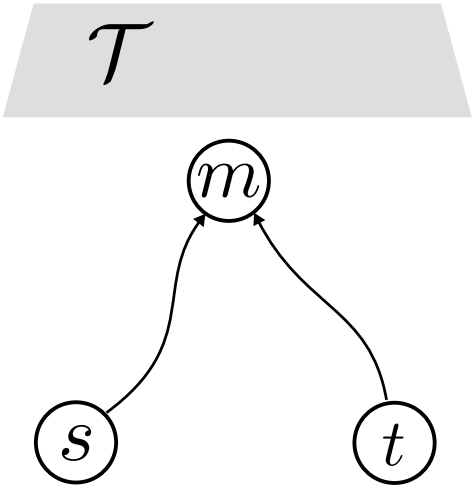
- Consider meeting point m of a $s-t$–search

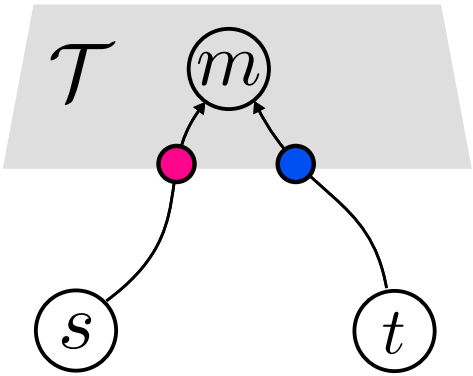


CH-based TNR

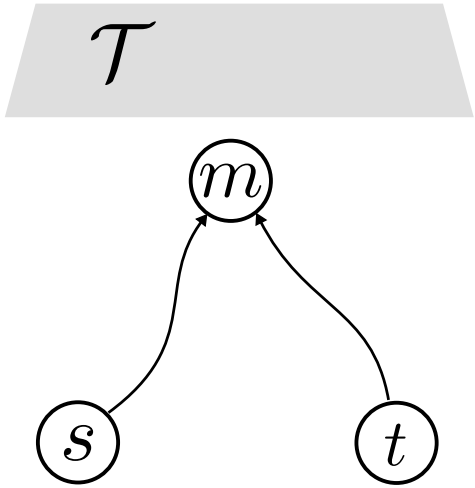
Properties of a local query

- Consider meeting point m of a $s-t$–search
- $m \notin \mathcal{T} \iff$ local query





CH-based TNR

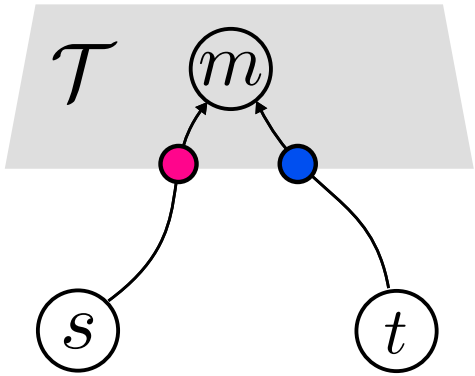


Properties of a local query

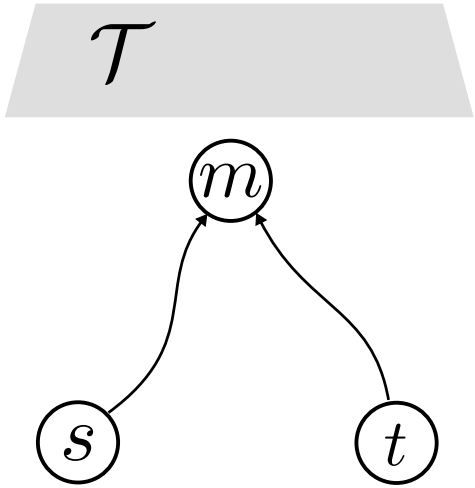
- Consider meeting point m of a $s-t$–search
- $m \notin \mathcal{T} \iff$ local query

Search space based locality filter

- Store search spaces below Transit Nodes $S : V \to V \setminus \mathcal{T}$
- Computation for free during preprocessing of Access Nodes
- During Query: $\mathcal{L} = (S(s) \cap S(t) \neq \emptyset)$
- Space requirements!



CH-based TNR

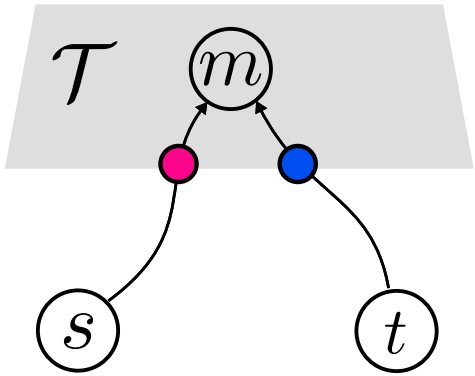


Properties of a local query

- Consider meeting point m of a $s-t$–search
- $m \notin \mathcal{T} \iff$ local query

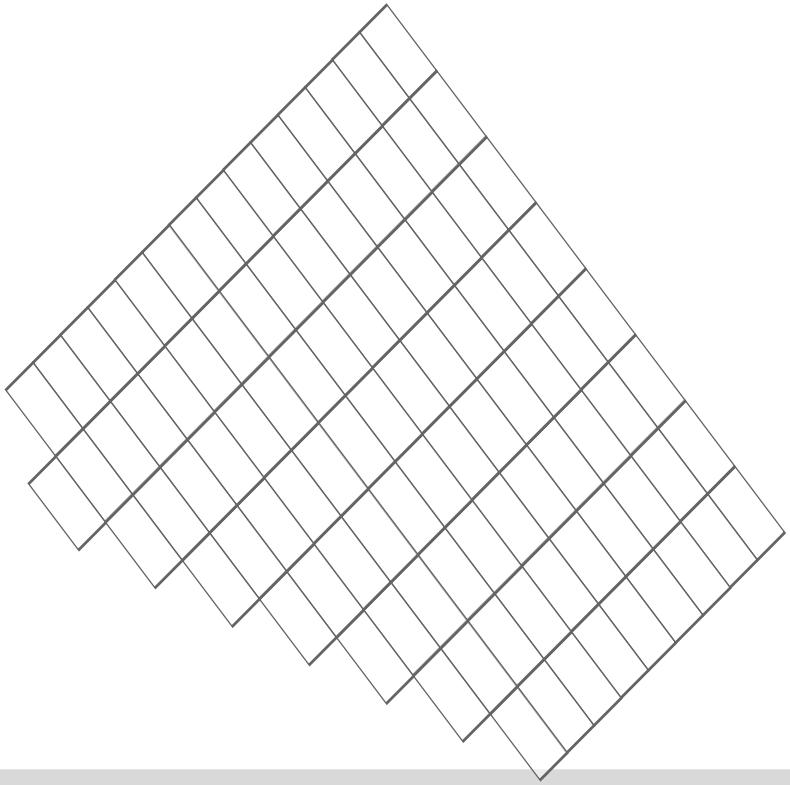
Search space based locality filter

- Store search spaces below Transit Nodes $S : V \to V \setminus \mathcal{T}$
- Computation for free during preprocessing of Access Nodes
- During Query: $\mathcal{L} = (S(s) \cap S(t) \neq \emptyset)$
- Space requirements!
- False positives are allowed



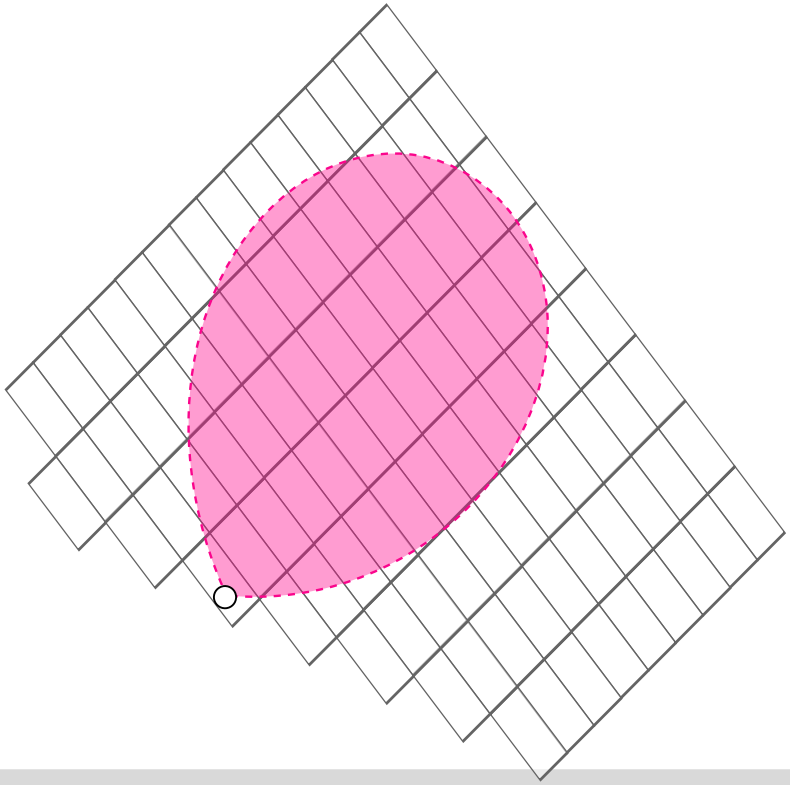
Locality Filter

- Partition the graph into regions
- Over-Approximation of the search spaces using touched regions.



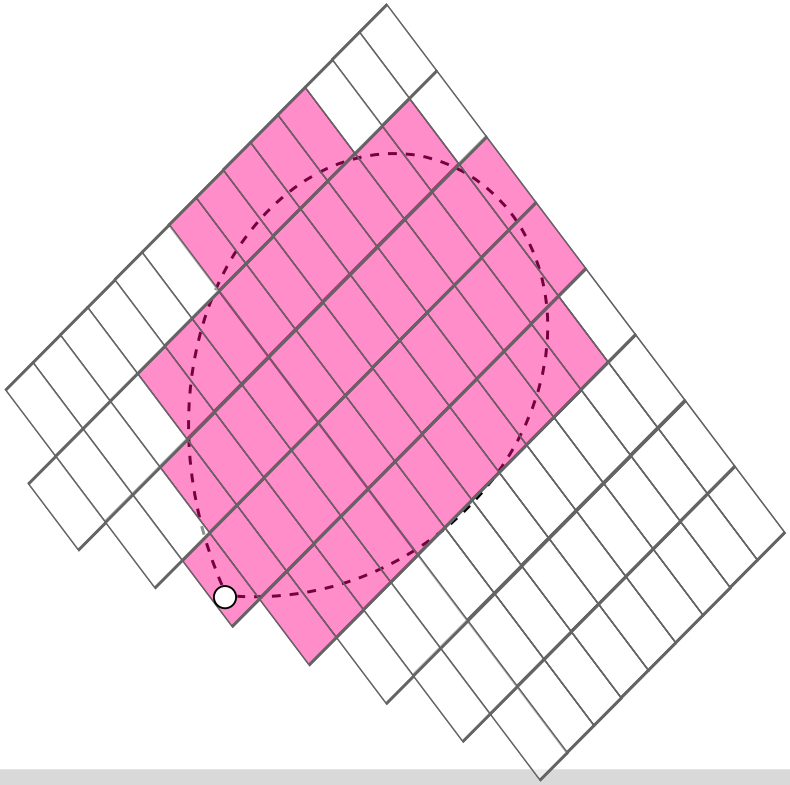
Locality Filter

- Partition the graph into regions
- Over-Approximation of the search spaces using touched regions.



Locality Filter

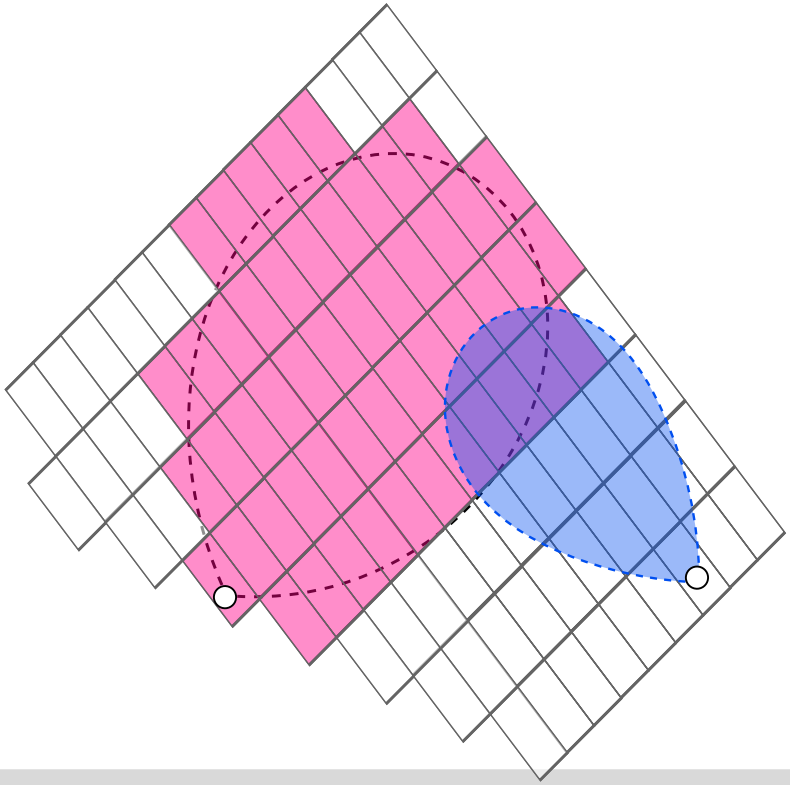
- Partition the graph into regions
- Over-Approximation of the search spaces using touched regions.



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

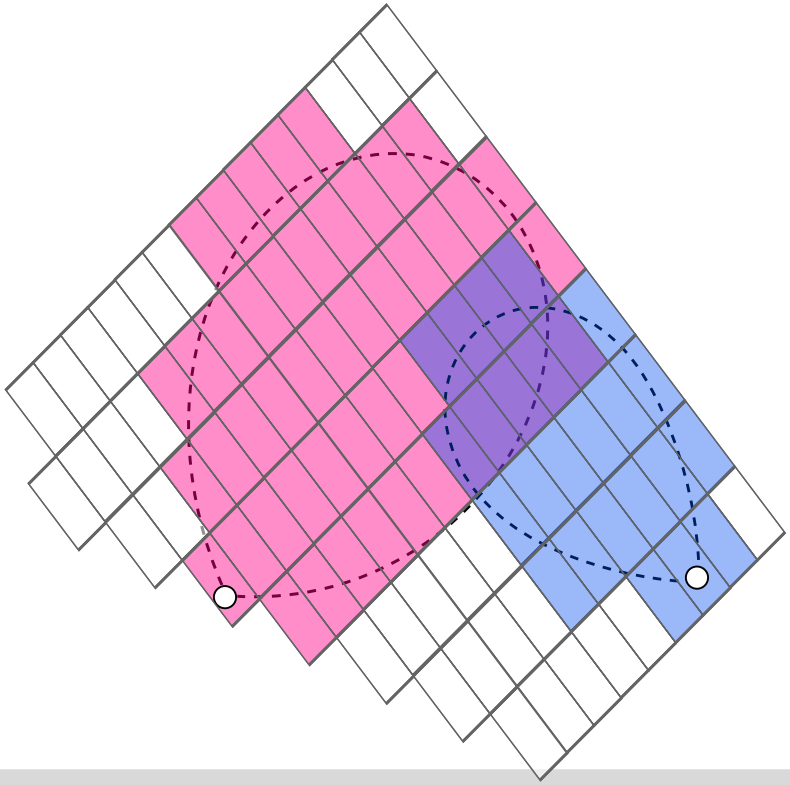
Locality Filter

- Partition the graph into regions
- Over-Approximation of the search spaces using touched regions.
- Correctness: $m \in S(s) \cap S(t) \Longrightarrow R(m) \in S'(s) \cap S'(t)$



Locality Filter

- Partition the graph into regions
- Over-Approximation of the search spaces using touched regions.
- Correctness: $m \in S(s) \cap S(t) \Longrightarrow R(m) \in S'(s) \cap S'(t)$

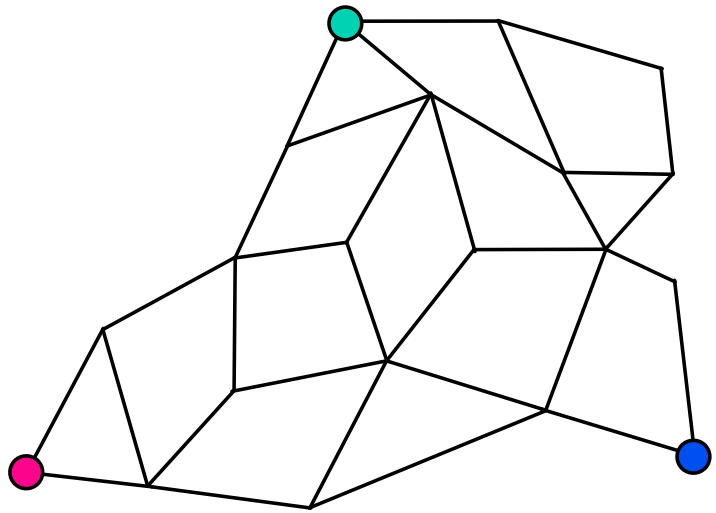


Locality Filter

- Larger regions $\rightarrow$ better compression
- More compact regions $\rightarrow$ less false positives
- Region boundaries should be close to search space boundaries

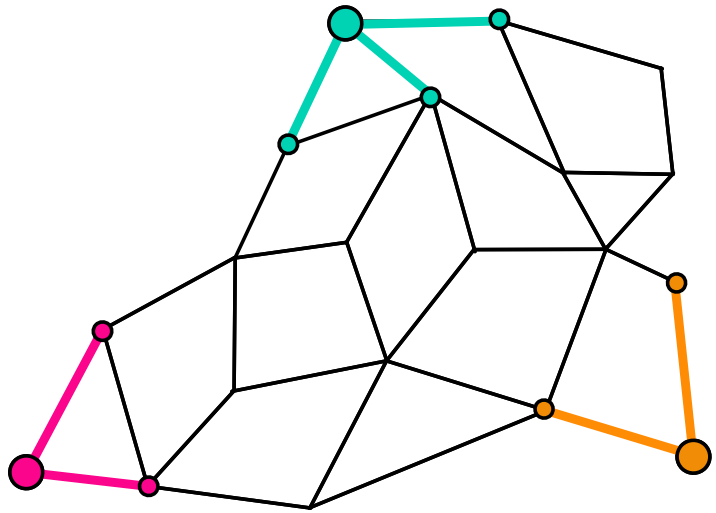
Locality Filter

- Larger regions $\rightarrow$ better compression
- More compact regions $\rightarrow$ less false positives
- Region boundaries should be close to search space boundaries
- Use graph Voronoi regions around transit nodes
- Represent region with its center node
- $\mathrm{Vor}(v) := \{u \in V : \forall w \in \mathcal{T} \setminus \{v\} : \mu(u,v) \leq \mu(u,w)\}$ for $v \in \mathcal{T}$
- Efficient computation with multi-source Dijkstra



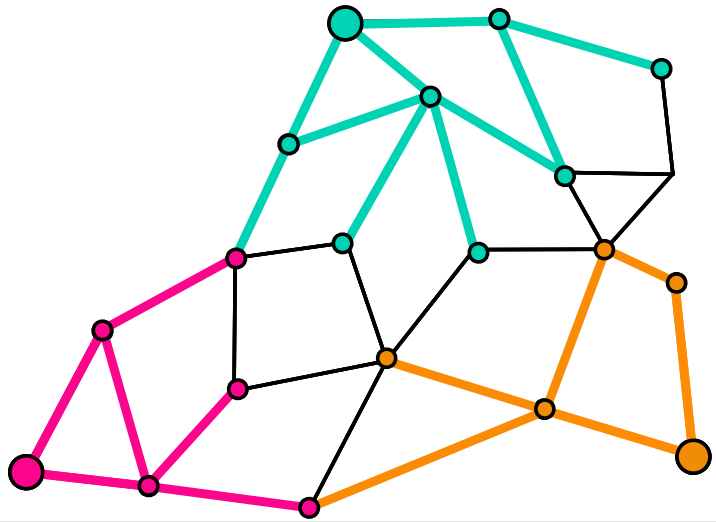
Locality Filter

- Larger regions $\rightarrow$ better compression
- More compact regions $\rightarrow$ less false positives
- Region boundaries should be close to search space boundaries
- Use graph Voronoi regions around transit nodes
- Represent region with its center node
- $\mathrm{Vor}(v) := \{u \in V : \forall w \in \mathcal{T} \setminus \{v\} : \mu(u,v) \leq \mu(u,w)\}$ for $v \in \mathcal{T}$
- Efficient computation with multi-source Dijkstra



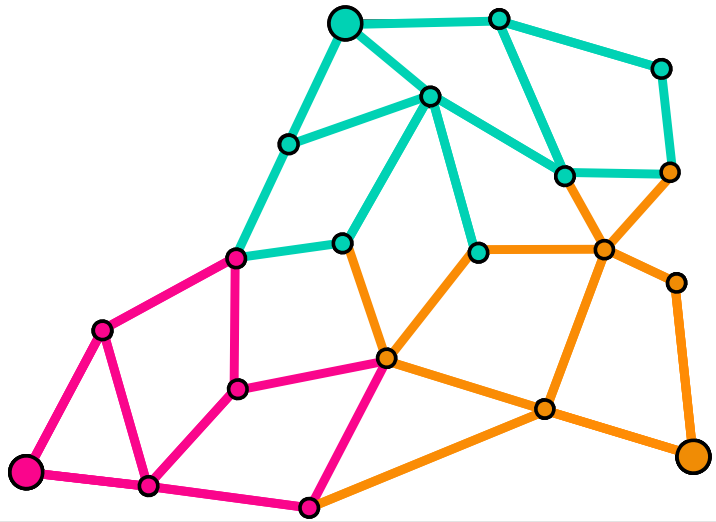
Locality Filter

- Larger regions $\rightarrow$ better compression
- More compact regions $\rightarrow$ less false positives
- Region boundaries should be close to search space boundaries
- Use graph Voronoi regions around transit nodes
- Represent region with its center node
- $\mathrm{Vor}(v) := \{u \in V : \forall w \in \mathcal{T} \setminus \{v\} : \mu(u,v) \leq \mu(u,w)\}$ for $v \in \mathcal{T}$
- Efficient computation with multi-source Dijkstra

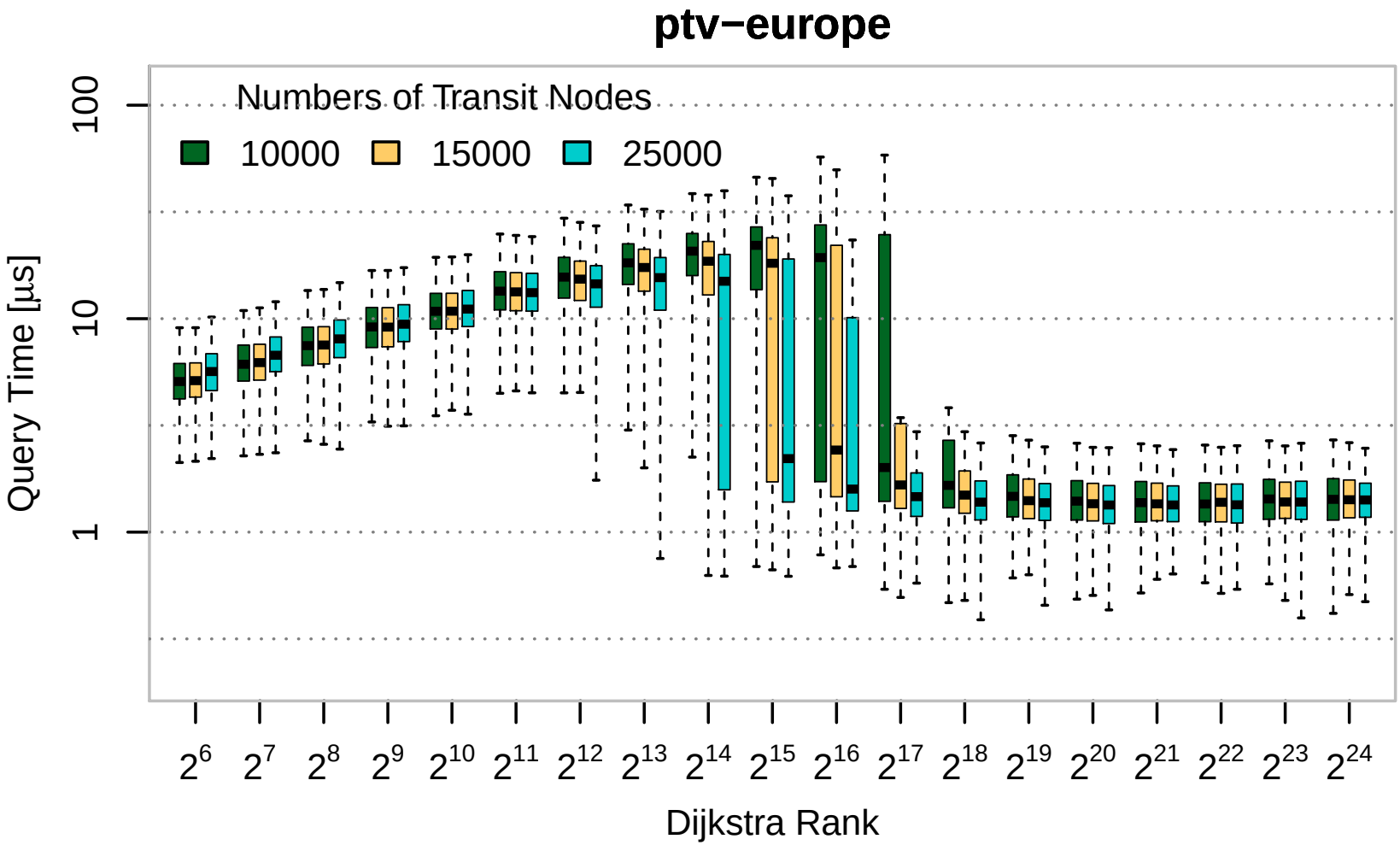


Locality Filter

- Larger regions $\rightarrow$ better compression
- More compact regions $\rightarrow$ less false positives
- Region boundaries should be close to search space boundaries
- Use graph Voronoi regions around transit nodes
- Represent region with its center node
- $\mathrm{Vor}(v) := \{u \in V : \forall w \in \mathcal{T} \setminus \{v\} : \mu(u,v) \leq \mu(u,w)\}$ for $v \in \mathcal{T}$
- Efficient computation with multi-source Dijkstra



Dijkstra Rank



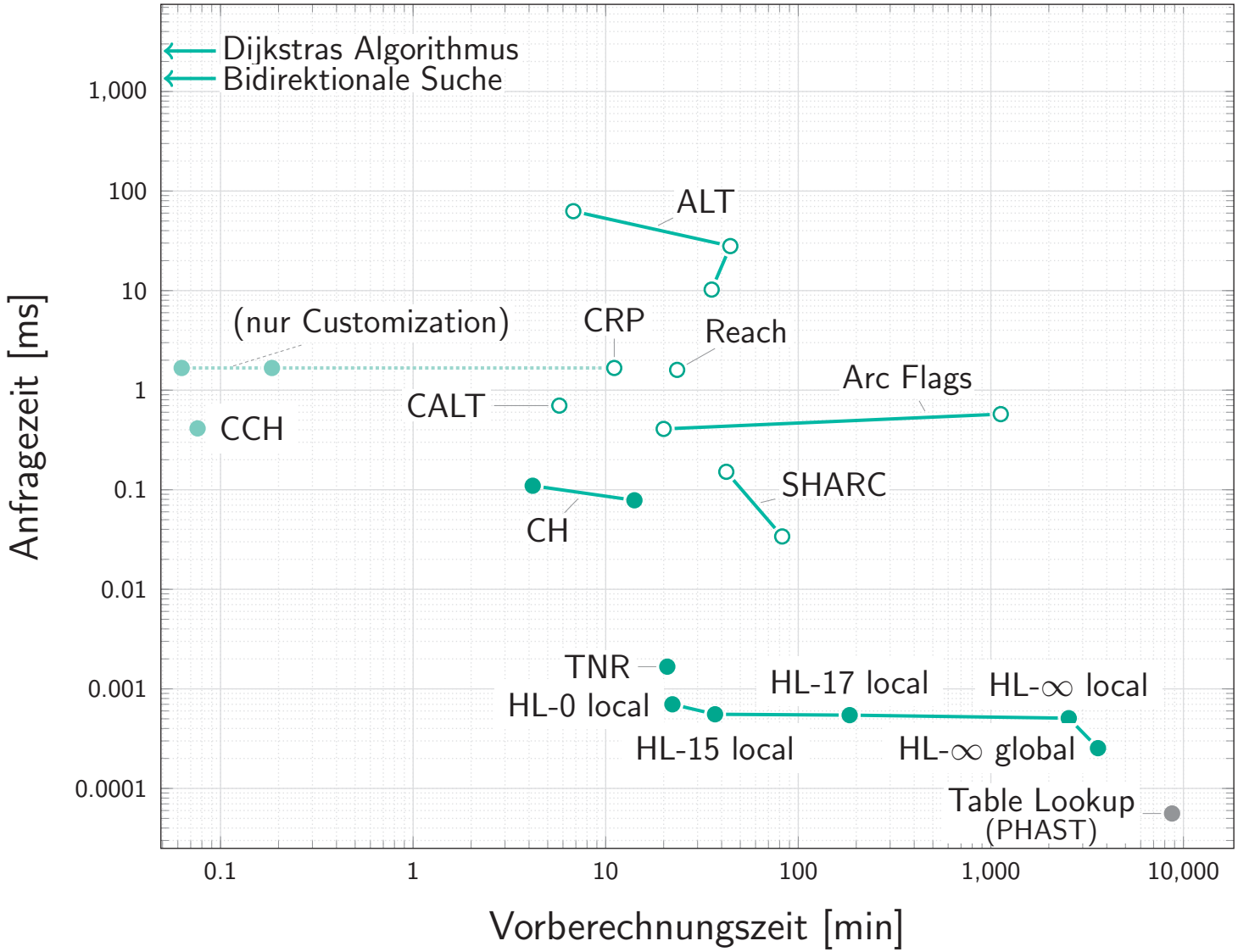
Frage: Welche durchschnittliche Laufzeit ergibt sich?

Zusammenfassung

Transit-Node Routing

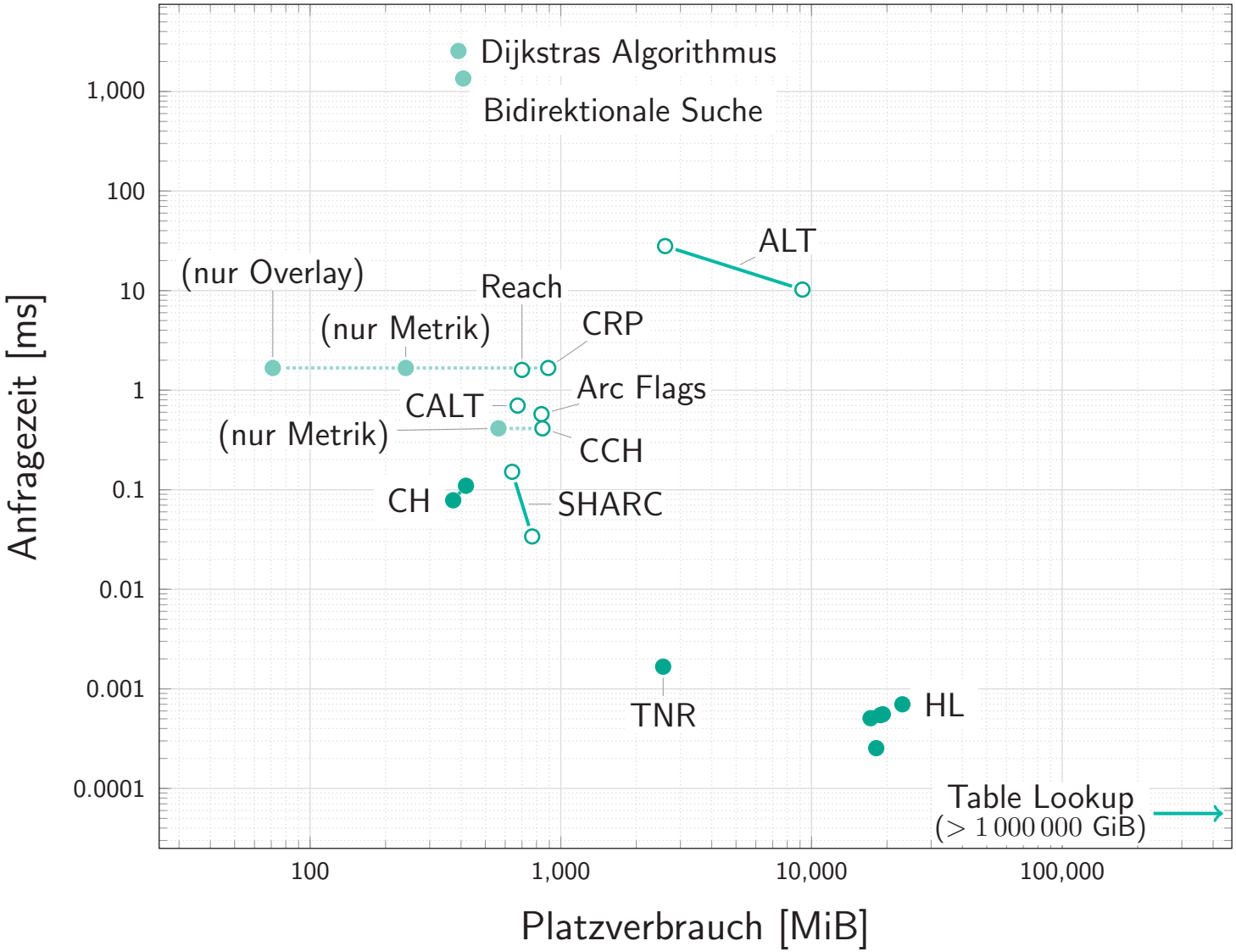
- ersetzt Suche (fast) komplett durch Table-Lookups
- 4 Zutaten:
  - Transit-Nodes
  - Distanztabelle
  - Access-Nodes
  - Locality-Filter

Übersicht bisherige Techniken

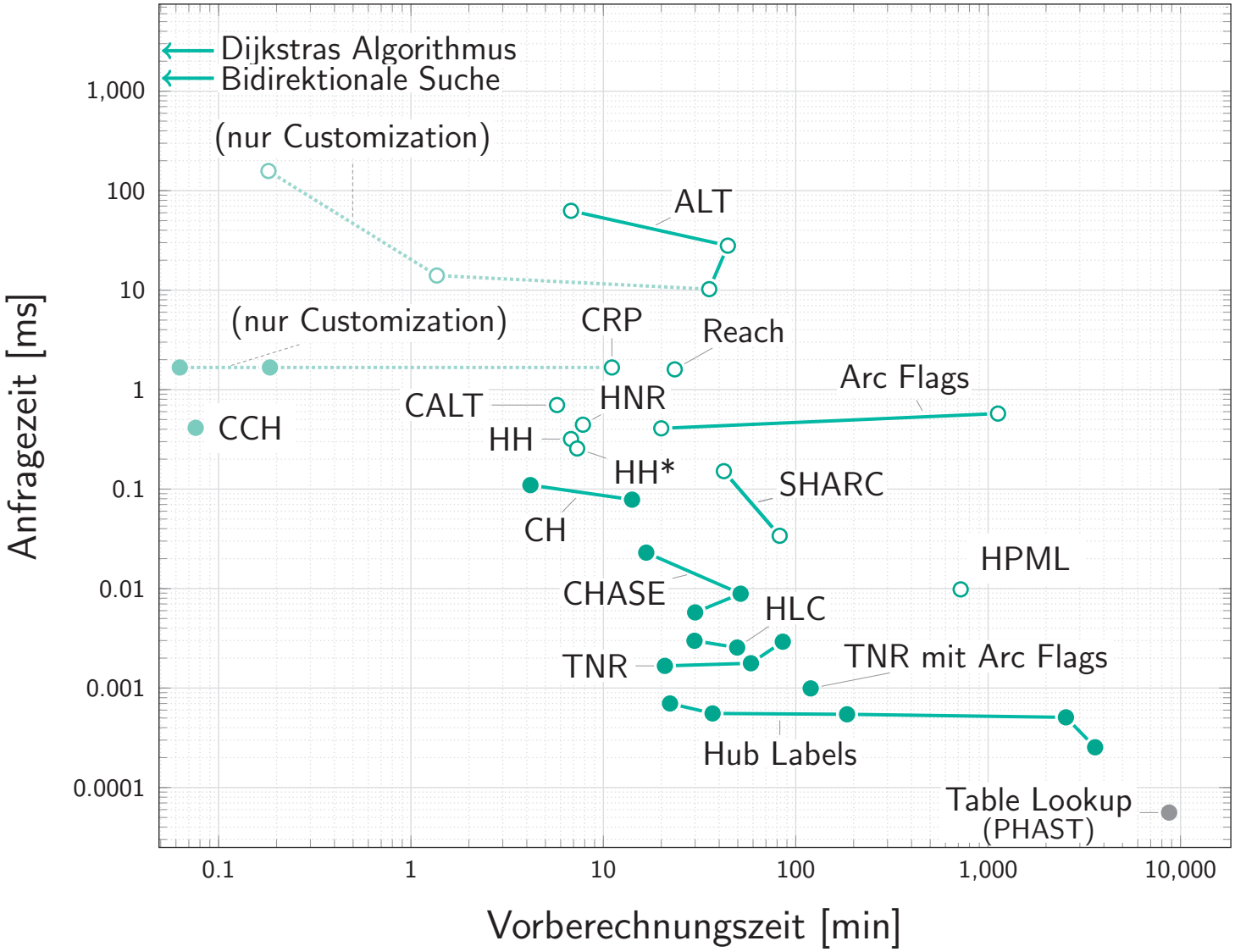


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Übersicht bisherige Techniken

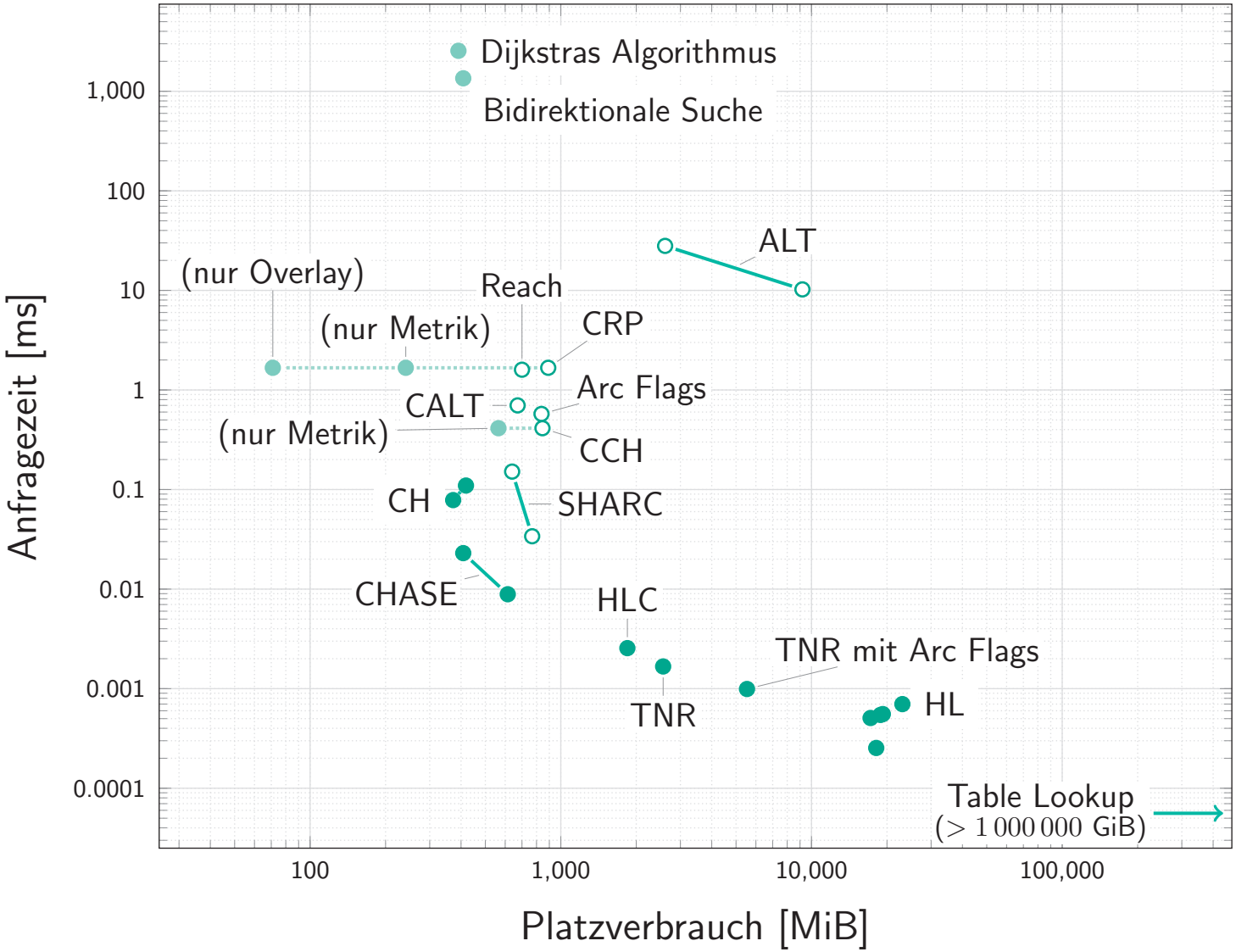


“Komplett”übersicht One-to-One



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

“Komplett”übersicht One-to-One



Literatur I

Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck.

Hierarchical hub labelings for shortest paths.

In *Proceedings of the 20th Annual European Symposium on Algorithms (ESA'12)*, volume 7501 of *Lecture Notes in Computer Science*, pages 24–35. Springer, 2012.

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.

Literatur II

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.

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.

Literatur III

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.

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.