



Algorithmen für Routenplanung

10. Vorlesung, Sommersemester 2022

Jonas Sauer | 25. Mai 2022



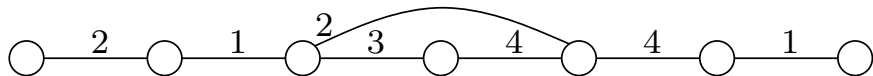
Kürzeste Wege in Straßennetzwerken

Beschleunigungstechniken (Fortsetzung)

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

Wiederholung: CH

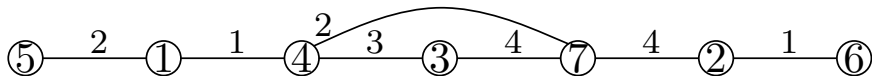
Vorbereitung:



Wiederholung: CH

Vorbereitung:

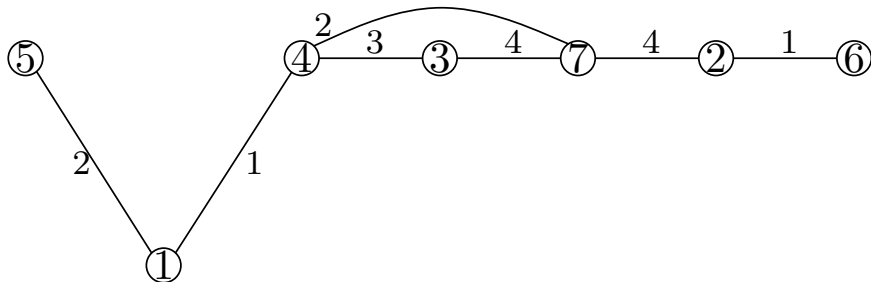
- Ordne Knoten nach Wichtigkeit



Wiederholung: CH

Vorbereitung:

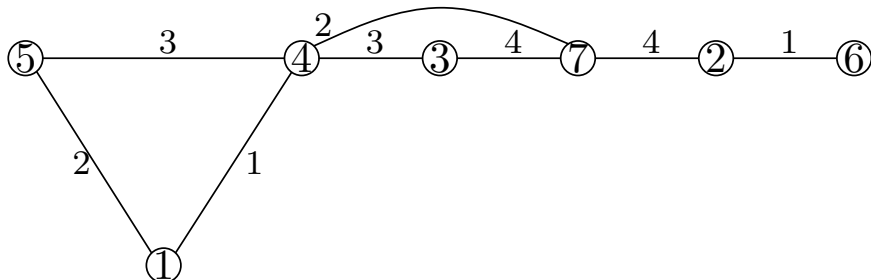
- Ordne Knoten nach Wichtigkeit
- Kontrahiere Knoten in dieser Reihenfolge
- Füge Shortcuts hinzu



Wiederholung: CH

Vorbereitung:

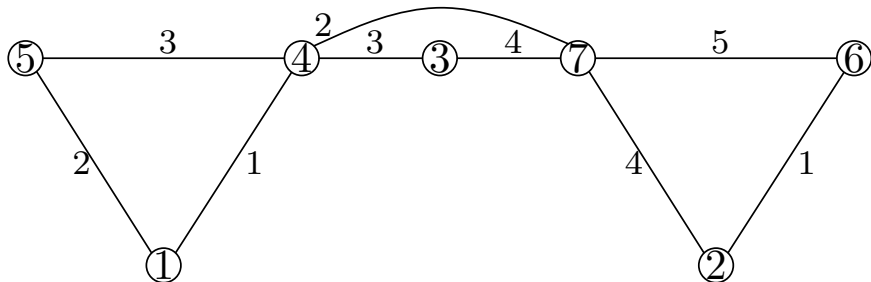
- Ordne Knoten nach Wichtigkeit
- Kontrahiere Knoten in dieser Reihenfolge
- Füge Shortcuts hinzu



Wiederholung: CH

Vorbereitung:

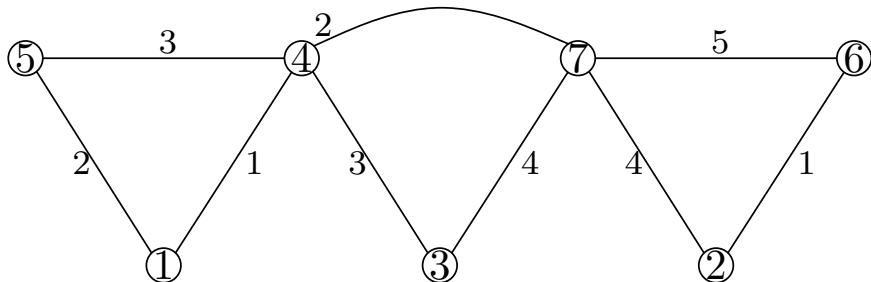
- Ordne Knoten nach Wichtigkeit
- Kontrahiere Knoten in dieser Reihenfolge
- Füge Shortcuts hinzu



Wiederholung: CH

Vorbereitung:

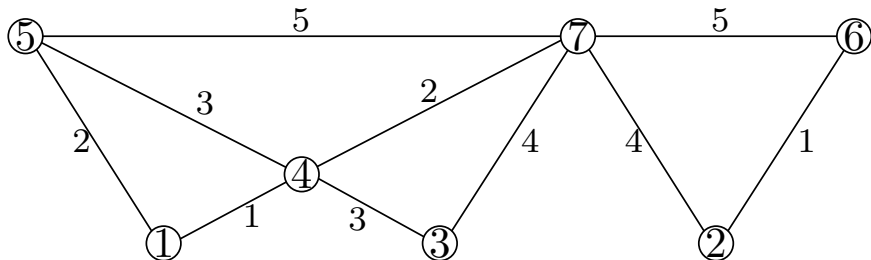
- Ordne Knoten nach Wichtigkeit
- Kontrahiere Knoten in dieser Reihenfolge
- Füge Shortcuts hinzu



Wiederholung: CH

Vorbereitung:

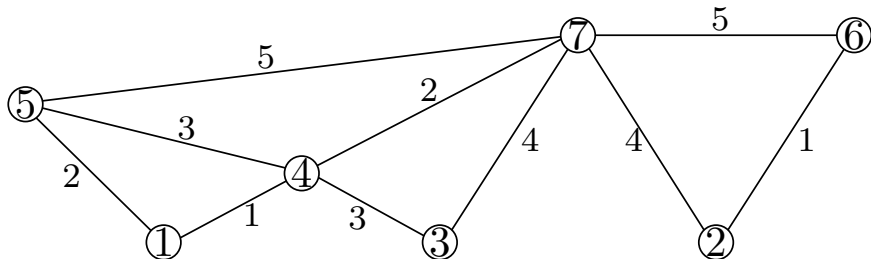
- Ordne Knoten nach Wichtigkeit
- Kontrahiere Knoten in dieser Reihenfolge
- Füge Shortcuts hinzu



Wiederholung: CH

Vorbereitung:

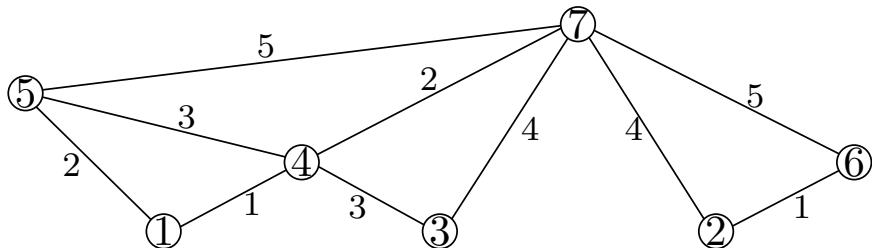
- Ordne Knoten nach Wichtigkeit
- Kontrahiere Knoten in dieser Reihenfolge
- Füge Shortcuts hinzu



Wiederholung: CH

Vorbereitung:

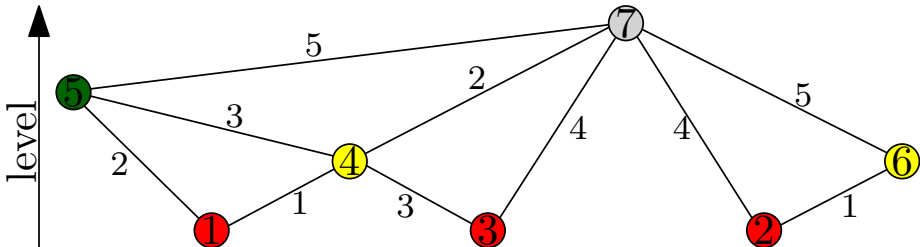
- Ordne Knoten nach Wichtigkeit
- Kontrahiere Knoten in dieser Reihenfolge
- Füge Shortcuts hinzu



Wiederholung: CH

Vorbereitung:

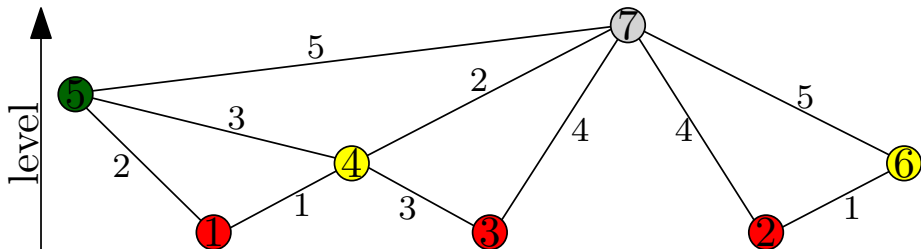
- Ordne Knoten nach Wichtigkeit
- Kontrahiere Knoten in dieser Reihenfolge
- Füge Shortcuts hinzu
- Weise den Knoten Levels zu



Wiederholung: CH

Punkt-zu-Punkt-Anfragen:

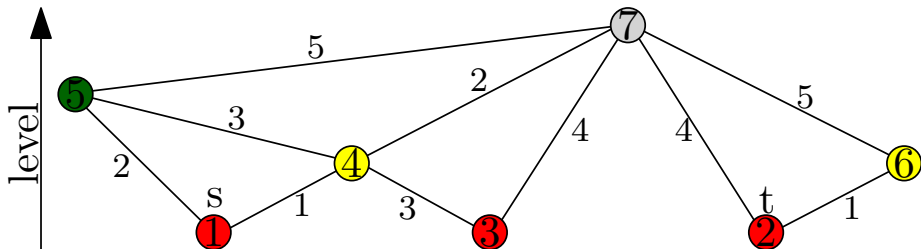
- Modifizierter **bidirektionaler** Dijkstra
- Folge nur Kanten zu wichtigeren Knoten



Wiederholung: CH

Punkt-zu-Punkt-Anfragen:

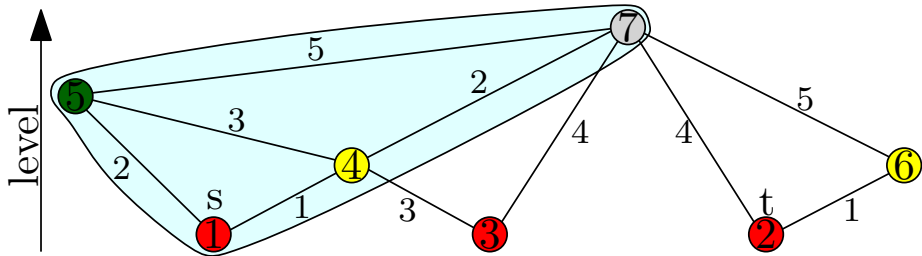
- Modifizierter bidirektionaler Dijkstra
- Folge nur Kanten zu wichtigeren Knoten



Wiederholung: CH

Punkt-zu-Punkt-Anfragen:

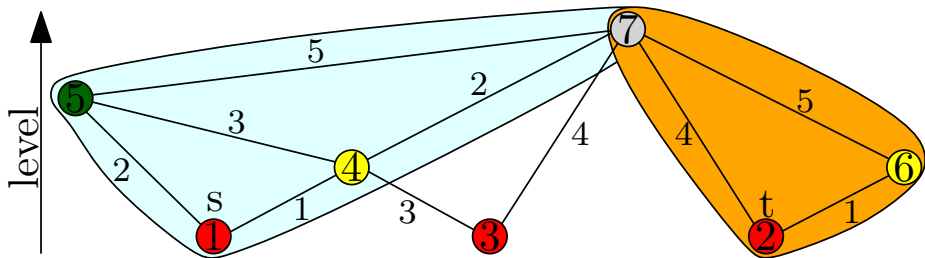
- Modifizierter **bidirektionaler** Dijkstra
- Folge nur Kanten zu wichtigeren Knoten



Wiederholung: CH

Punkt-zu-Punkt-Anfragen:

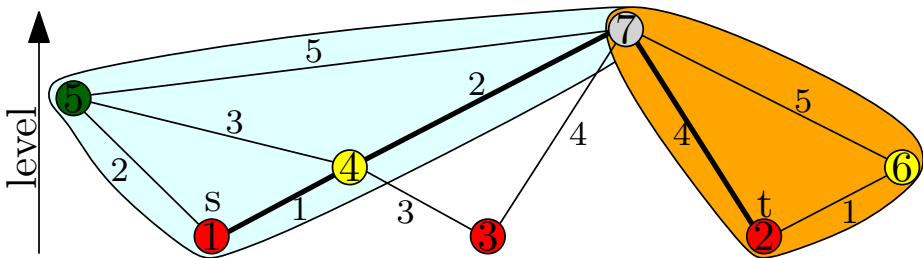
- Modifizierter **bidirektionaler** Dijkstra
- Folge nur Kanten zu wichtigeren Knoten



Wiederholung: CH

Punkt-zu-Punkt-Anfragen:

- Modifizierter **bidirektionaler** Dijkstra
- Folge nur Kanten zu wichtigeren Knoten



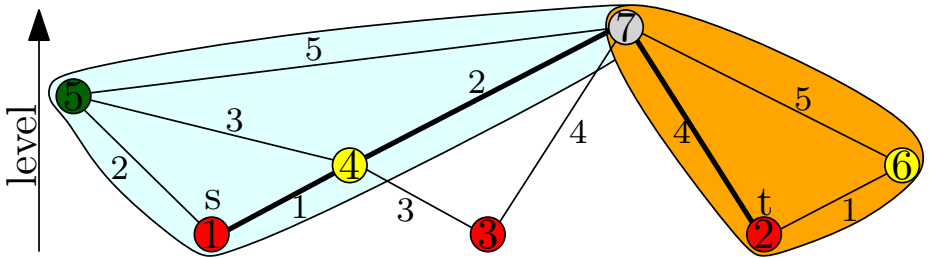
Wiederholung: CH

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



Hub Labeling

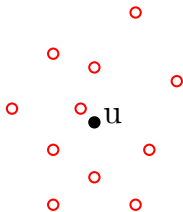
Vorbereitung:

- Für jeden Knoten u : Berechne zwei Labels $L_f(u)$, $L_b(u)$

• u

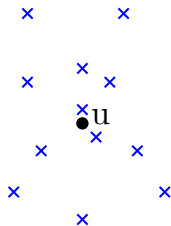
Vorbereitung:

- Für jeden Knoten u : Berechne zwei Labels $L_f(u)$, $L_b(u)$
- Ein Label ist eine Menge von Knoten (Hubs) und Distanzen
 - $\text{dist}(u, v)$ für jeden Hub $v \in L_f(u)$



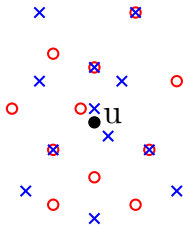
Vorbereitung:

- Für jeden Knoten u : Berechne zwei Labels $L_f(u)$, $L_b(u)$
- 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 $v \in L_b(u)$



Vorbereitung:

- Für jeden Knoten u : Berechne zwei Labels $L_f(u)$, $L_b(u)$
- 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 $v \in L_b(u)$



Vorbereitung:

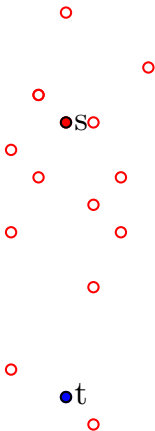
- Für jeden Knoten u : Berechne zwei Labels $L_f(u)$, $L_b(u)$
- Ein Label ist eine Menge von Knoten (Hubs) und Distanzen
 - $\text{dist}(u, v)$ für jeden Hub $v \in L_f(u)$
 - $\text{dist}(v, u)$ für jeden Hub $v \in L_b(u)$
- Die Labels müssen die **cover property** erfüllen:
 $\forall s, t \in V: L_f(s) \cap L_b(t)$ enthält ≥ 1 Knoten
auf dem kürzesten s - t -Pfad

●s

●t

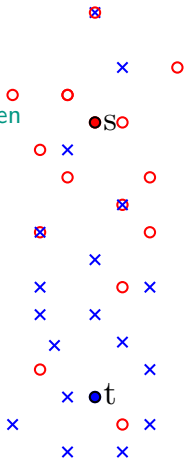
Vorbereitung:

- Für jeden Knoten u : Berechne zwei Labels $L_f(u)$, $L_b(u)$
- 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 Labels müssen die **cover property** erfüllen:
 $\forall s, t \in V: L_f(s) \cap L_b(t)$ enthält ≥ 1 Knoten
auf dem kürzesten s - t -Pfad



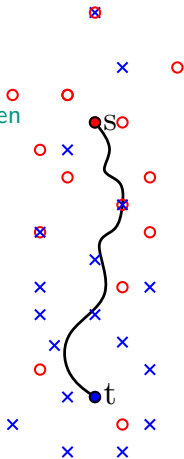
Vorbereitung:

- Für jeden Knoten u : Berechne zwei Labels $L_f(u)$, $L_b(u)$
- 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 Labels müssen die **cover property** erfüllen:
 $\forall s, t \in V: L_f(s) \cap L_b(t)$ enthält ≥ 1 Knoten
auf dem kürzesten s - t -Pfad



Vorbereitung:

- Für jeden Knoten u : Berechne zwei Labels $L_f(u)$, $L_b(u)$
- 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 Labels müssen die **cover property** erfüllen:
 $\forall s, t \in V: L_f(s) \cap L_b(t)$ enthält ≥ 1 Knoten
auf dem kürzesten s - t -Pfad



Vorbereitung:

- Für jeden Knoten u : Berechne zwei Labels $L_f(u)$, $L_b(u)$
- 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 Labels müssen die **cover property** erfüllen:
 $\forall s, t \in V: L_f(s) \cap L_b(t)$ enthält ≥ 1 Knoten
auf dem kürzesten s - t -Pfad

s - t -Anfrage:

- Finde Knoten $v \in L_f(s) \cap L_b(t) \dots$

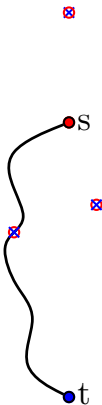


Vorbereitung:

- Für jeden Knoten u : Berechne zwei Labels $L_f(u)$, $L_b(u)$
- 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 Labels müssen die **cover property** erfüllen:
 $\forall s, t \in V: L_f(s) \cap L_b(t)$ enthält ≥ 1 Knoten
auf dem kürzesten s - t -Pfad

s - t -Anfrage:

- Finde Knoten $v \in L_f(s) \cap L_b(t) \dots$
- \dots der $\text{dist}(s, v) + \text{dist}(v, t)$ **minimiert**

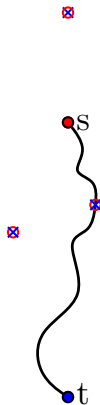


Vorbereitung:

- Für jeden Knoten u : Berechne zwei Labels $L_f(u)$, $L_b(u)$
- 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 Labels müssen die **cover property** erfüllen:
 $\forall s, t \in V: L_f(s) \cap L_b(t)$ enthält ≥ 1 Knoten
auf dem kürzesten s - t -Pfad

s - t -Anfrage:

- Finde Knoten $v \in L_f(s) \cap L_b(t) \dots$
- \dots der $\text{dist}(s, v) + \text{dist}(v, t)$ **minimiert**

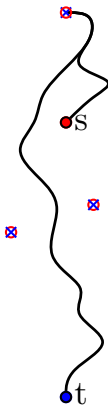


Vorbereitung:

- Für jeden Knoten u : Berechne zwei Labels $L_f(u)$, $L_b(u)$
- 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 Labels müssen die **cover property** erfüllen:
 $\forall s, t \in V: L_f(s) \cap L_b(t)$ enthält ≥ 1 Knoten
auf dem kürzesten s - t -Pfad

s - t -Anfrage:

- Finde Knoten $v \in L_f(s) \cap L_b(t) \dots$
- \dots der $\text{dist}(s, v) + \text{dist}(v, t)$ **minimiert**



Vorbereitung:

- Für jeden Knoten u : Berechne zwei Labels $L_f(u)$, $L_b(u)$
- 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 Labels müssen die **cover property** erfüllen:
 $\forall s, t \in V: L_f(s) \cap L_b(t)$ enthält ≥ 1 Knoten
auf dem kürzesten s - t -Pfad

s - t -Anfrage:

- Finde Knoten $v \in L_f(s) \cap L_b(t) \dots$
- \dots der $\text{dist}(s, v) + \text{dist}(v, t)$ **minimiert**



Vorbereitung:

- Für jeden Knoten u : Berechne zwei Labels $L_f(u)$, $L_b(u)$
- 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 Labels müssen die **cover property** erfüllen:
 $\forall s, t \in V: L_f(s) \cap L_b(t)$ enthält ≥ 1 Knoten
auf dem kürzesten s - t -Pfad

s - t -Anfrage:

- Finde Knoten $v \in L_f(s) \cap L_b(t) \dots$
- \dots der $\text{dist}(s, v) + \text{dist}(v, t)$ **minimiert**



Beobachtungen:

- Laufzeit hängt von Labelgröße ab
- Wie effizient berechnen?

Hub Labeling

Speichern der Labels:

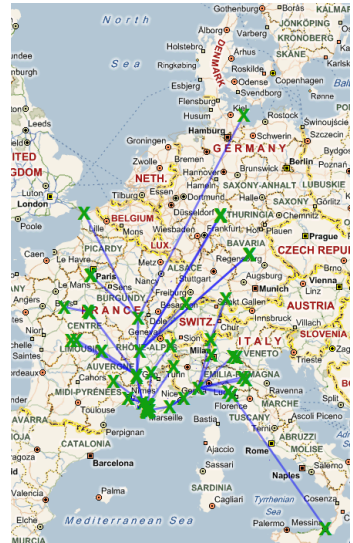
- Als Menge von (Hub, Distanz)-Paaren
- Sortiert nach Hub-Knoten-ID

Hub Labeling

Speichern der Labels:

- Als Menge von (Hub, Distanz)-Paaren
- Sortiert nach Hub-Knoten-ID

$$L_f(s) \begin{bmatrix} 1,0 & 4,1 & 5,2 & 7,3 \end{bmatrix}$$



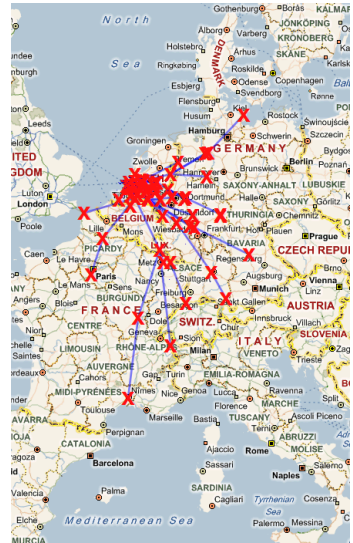
Hub Labeling

Speichern der Labels:

- Als Menge von (Hub, Distanz)-Paaren
- Sortiert nach Hub-Knoten-ID

$$L_f(s) \quad \boxed{1,0} \quad \boxed{4,1} \quad \boxed{5,2} \quad \boxed{7,3}$$

$$L_b(t) \quad \boxed{2,0} \quad \boxed{6,1} \quad \boxed{7,4} \quad \boxed{8,1} \quad \boxed{9,3}$$



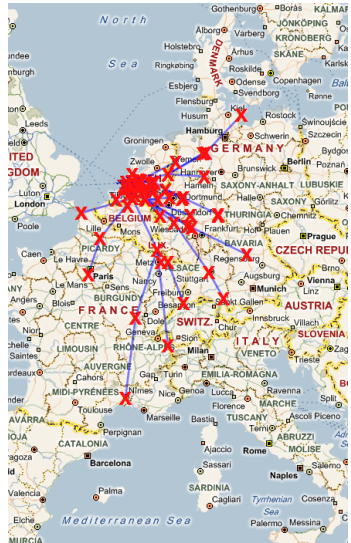
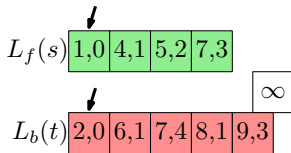
Hub Labeling

Speichern der Labels:

- Als Menge von (Hub, Distanz)-Paaren
- Sortiert nach Hub-Knoten-ID

Anfrage:

- Simultanes Scannen von zwei Arrays
- Nur einige Speicherzugriffe nötig



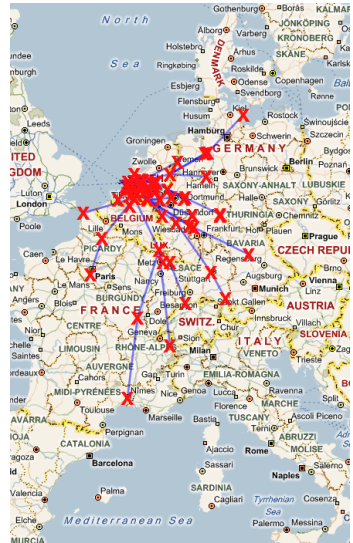
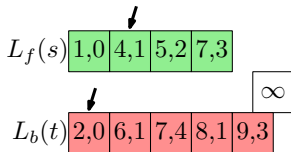
Hub Labeling

Speichern der Labels:

- Als Menge von (Hub, Distanz)-Paaren
- Sortiert nach Hub-Knoten-ID

Anfrage:

- Simultanes Scannen von zwei Arrays
- Nur einige Speicherzugriffe nötig



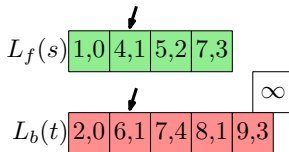
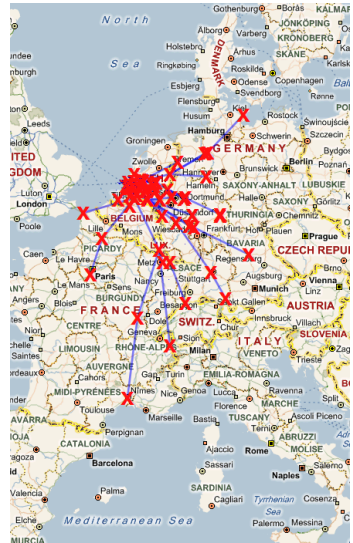
Hub Labeling

Speichern der Labels:

- Als Menge von (Hub, Distanz)-Paaren
- Sortiert nach Hub-Knoten-ID

Anfrage:

- Simultanes Scannen von zwei Arrays
- Nur einige Speicherzugriffe nötig



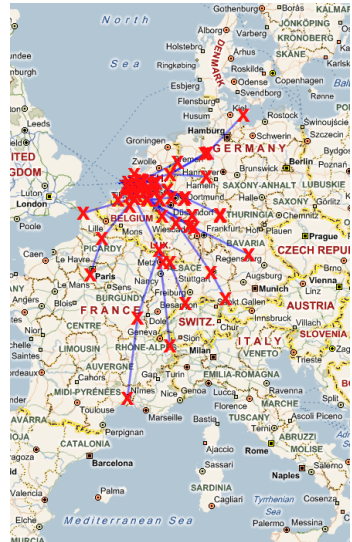
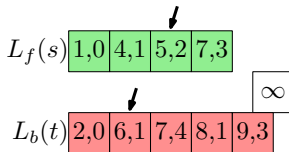
Hub Labeling

Speichern der Labels:

- Als Menge von (Hub, Distanz)-Paaren
- Sortiert nach Hub-Knoten-ID

Anfrage:

- Simultanes Scannen von zwei Arrays
- Nur einige Speicherzugriffe nötig



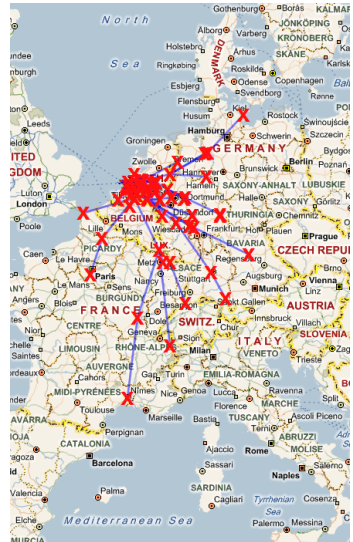
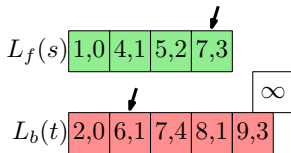
Hub Labeling

Speichern der Labels:

- Als Menge von (Hub, Distanz)-Paaren
- Sortiert nach Hub-Knoten-ID

Anfrage:

- Simultanes Scannen von zwei Arrays
- Nur einige Speicherzugriffe nötig



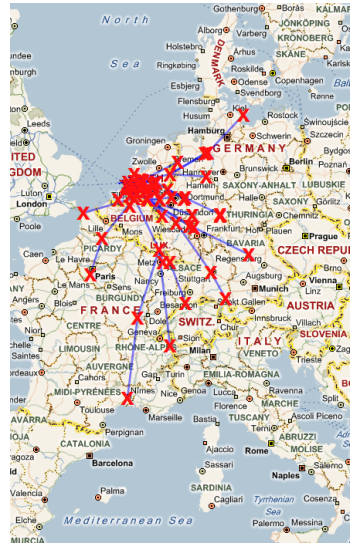
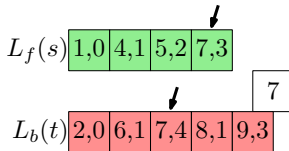
Hub Labeling

Speichern der Labels:

- Als Menge von (Hub, Distanz)-Paaren
- Sortiert nach Hub-Knoten-ID

Anfrage:

- Simultanes Scannen von zwei Arrays
- Nur einige Speicherzugriffe nötig



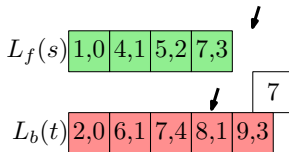
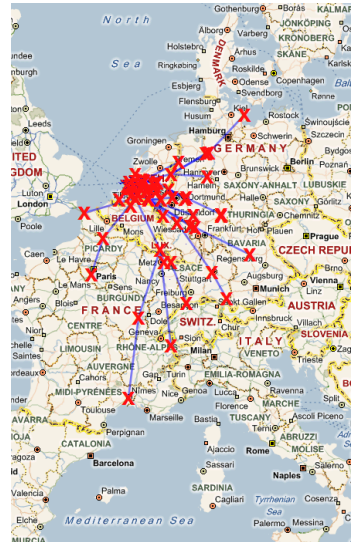
Hub Labeling

Speichern der Labels:

- Als Menge von (Hub, Distanz)-Paaren
- Sortiert nach Hub-Knoten-ID

Anfrage:

- Simultanes Scannen von zwei Arrays
- Nur einige Speicherzugriffe nötig



Hub Labeling

Speichern der Labels:

- Als Menge von (Hub, Distanz)-Paaren
- Sortiert nach Hub-Knoten-ID

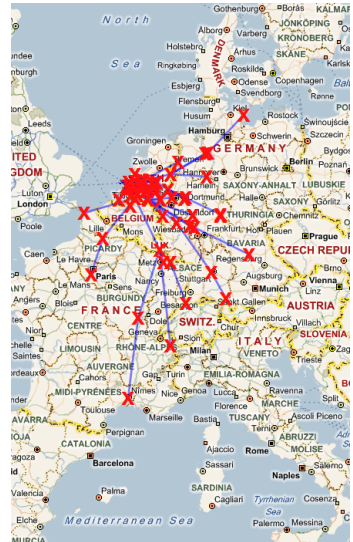
Anfrage:

- Simultanes Scannen von zwei Arrays
- Nur einige Speicherzugriffe nötig
- Sehr hohe Lokalität

$$L_f(s) \begin{array}{|c|c|c|c|} \hline 1,0 & 4,1 & 5,2 & 7,3 \\ \hline \end{array}$$

7

$$L_b(t) \begin{array}{|c|c|c|c|c|} \hline 2,0 & 6,1 & 7,4 & 8,1 & 9,3 \\ \hline \end{array}$$



Komplexität:

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

Hierarchische Hub-Labels:

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

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

- Ein Labeling ist **hierarchisch**, wenn \preceq eine partielle Ordnung ist.
- Optimale hierarchische Hub-Labels zu berechnen ist NP-schwer [BGK⁺15]

Kanonische Hub-Labels:

- Ein Labeling ist **kanonisch** bezüglich einer Knotenordnung O , wenn
 - das Labeling hierarchisch ist
 - \preceq mit O konsistent ist
 - man aus keinem Label einen Hub löschen kann
- Das kanonische Labeling ist eindeutig für eine feste Ordnung O

- \preceq ordnet die Knoten nach „Wichtigkeit“ wie bei CH
 - CH-Suchräume sind gültige hierarchische Hub-Labels
 - \preceq ist konsistent mit Kontraktionsordnung
 - Aber sie sind größer als nötig (siehe Stall-on-Demand)
 - Also nicht kanonisch
- Überflüssige Knoten filtern

- \preceq ordnet die Knoten nach „Wichtigkeit“ wie bei CH
 - CH-Suchräume sind gültige hierarchische Hub-Labels
 - \preceq ist konsistent mit Kontraktionsordnung
 - Aber sie sind größer als nötig (siehe Stall-on-Demand)
 - Also nicht kanonisch
- Überflüssige Knoten filtern
- Im Folgenden betrachten wir nur noch hierarchische Hub-Labels
 - Für Beweise nehmen wir ferner an:
 - Kürzeste Wege sind eindeutig
 - Graphen sind ungerichtet
- $$L(v) := L_f(v) = L_b(v)$$

- Sei $m(s, t)$ der Knoten mit höchstem Rank auf dem kürzesten s - t -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



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

- Sei $m(s, t)$ der Knoten mit höchsten Rank auf dem kürzesten s - t -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



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

- 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.

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:** Wenn wir h löschen, wird die v - h -Anfrage falsch beantwortet.

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:** Wenn wir h löschen, wird die v - h -Anfrage falsch beantwortet.

Beweis:

- Ein gemeinsamer Hub von h und v kann nicht niedriger sein als h oder v (folgt direkt aus der Definition von kanonischem Labeling)
- Der höchste Knoten auf dem kürzesten v - h -Pfad ist h (Voraussetzung)
- Also können sich $L(v)$ und $L(h)$ nur in h schneiden
 $\implies h$ darf nicht gelöscht werden

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

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

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:

- **Wir zeigen:** Diese v - t -Anfragen treffen sich nicht nur in h , sondern auch in $m(v, h)$ oder in $m(h, t)$
- Fall 1:
 - $m(v, h)$ höher als $m(h, t)$
 - $m(v, h)$ also auch höchster Knoten auf v - t -Pfad
 - Nach Argument von letzter Folie: $m(v, h) \in L(v)$ und $m(v, h) \in L(t)$
 - v - t -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

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

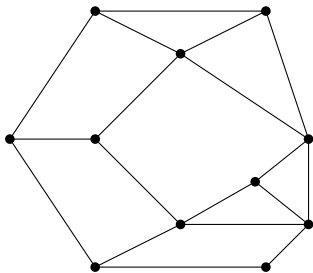
Beweis:

- **Wir zeigen:** Diese v - t -Anfragen treffen sich nicht nur in h , sondern auch in $m(v, h)$ oder in $m(h, t)$
- Fall 2:
 - $m(h, t)$ höher als in $m(v, h)$
 - $m(h, t)$ also auch höchster Knoten auf v - t -Pfad
 - Nach Argument von letzter Folie: $m(h, t) \in L(v)$ und $m(h, t) \in L(t)$
 - v - t -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 v - t -Pfad. Das kann aber nicht sein, da $m(v, h)$ höher ist.
 - Da $h \neq m(h, t)$, können wir h löschen

Vorbereitung

Idee:

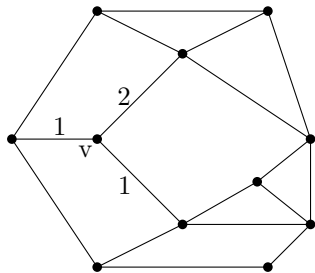
- Benutze Knotenordnung



Vorbereitung

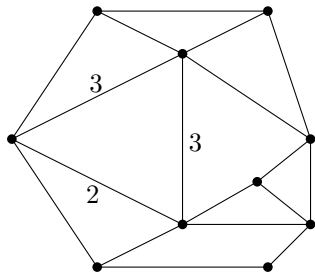
Idee:

- Benutze Knotenordnung
- Kontrahiere Knoten v



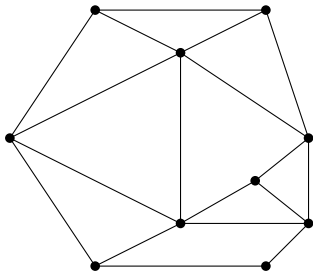
Idee:

- Benutze Knotenordnung
- Kontrahiere Knoten v



Idee:

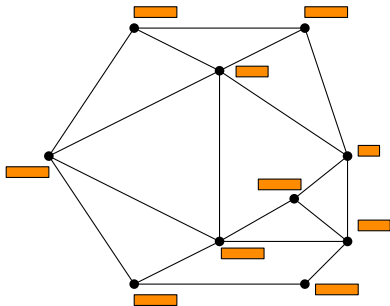
- Benutze Knotenordnung
- Kontrahiere Knoten v



Vorberechnung

Idee:

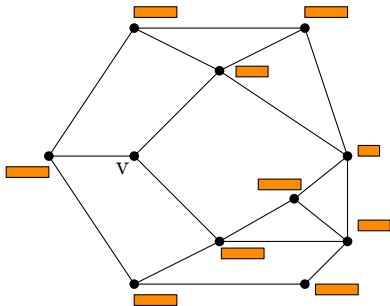
- Benutze Knotenordnung
- Kontrahiere Knoten v
- Berechne Labels rekursiv



Vorberechnung

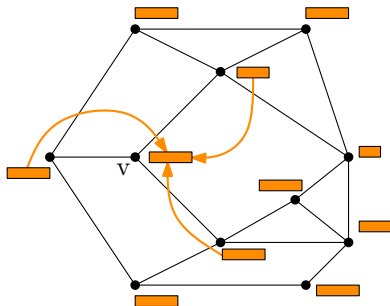
Idee:

- Benutze Knotenordnung
- Kontrahiere Knoten v
- Berechne Labels rekursiv



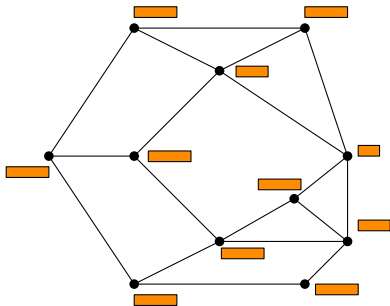
Idee:

- Benutze Knotenordnung
- Kontrahiere Knoten v
- Berechne Labels rekursiv
- Vereinige (merge) Labels der Aufwärtsnachbarn von v
- Dünne Label aus



Idee:

- Benutze Knotenordnung
- Kontrahiere Knoten v
- Berechne Labels rekursiv
- Vereinige (merge) Labels der Aufwärtsnachbarn von v
- Dünne Label aus



Korrektheit:

- Analog zur Korrektheit von CH
- Argumentation über den wichtigsten Knoten auf dem Pfad
- Dieser ist im Vorwärtslabel von s und im Rückwärtslabel von t

Generell:

- $L_f(v)$ ist die Vereinigung der Labels der Aufwärtsnachbarn 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

Generell:

- $L_f(v)$ ist die Vereinigung der Labels der Aufwärtsnachbarn 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:

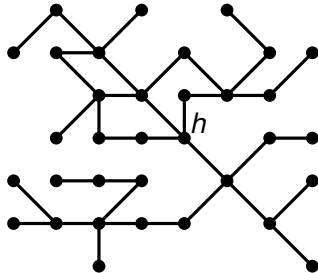
- Manche Knoten im Label sind nicht notwendig
- **Ziel:** Entferne Hubs h , für die $h \neq m(v, h)$
- Label von h ist final, da h höher als v
- Label von v ist korrekt (aber noch nicht minimal)
- Wir können eine HL-Anfrage durchführen, um $m(v, h)$ zu bestimmen
- Lösche h , wenn $h \neq m(v, h)$

Alternative Labelkonstruktion:

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

Alternative Labelkonstruktion:

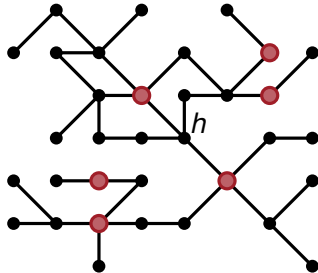
- Verteile Hubs auf Labels
- h ist Hub von v
 - $\iff h = m(h, v)$
 - \iff es gibt auf dem h - v -Pfad keinen höheren Knoten als h
- Starte Dijkstra von h und besuche alle v , in deren Label h liegt



Ziel: h in alle Labels verteilen

Alternative Labelkonstruktion:

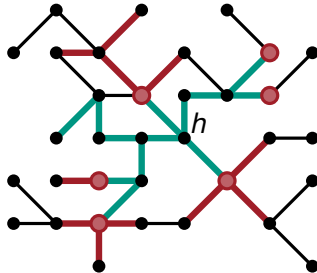
- Verteile Hubs auf Labels
- h ist Hub von v
 - $\iff h = m(h, v)$
 - \iff es gibt auf dem h - v -Pfad keinen höheren Knoten als h
- Starte Dijkstra von h und besuche alle v , in deren Label h liegt



Rote Knoten sind höher als h

Alternative Labelkonstruktion:

- Verteile Hubs auf Labels
- h ist Hub von v
 - $\iff h = m(h, v)$
 - \iff es gibt auf dem h - v -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

- Dijkstras Algorithmus sucht ganzen Graph ab
- → muss vorzeitig abgebrochen werden

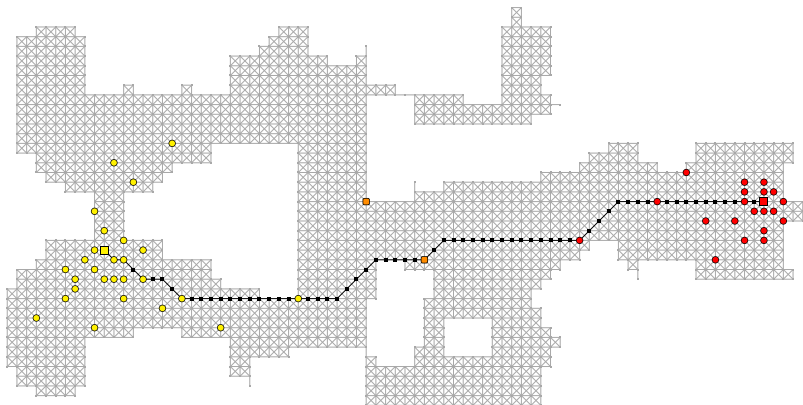
Option 1:

- **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.

Option 2:

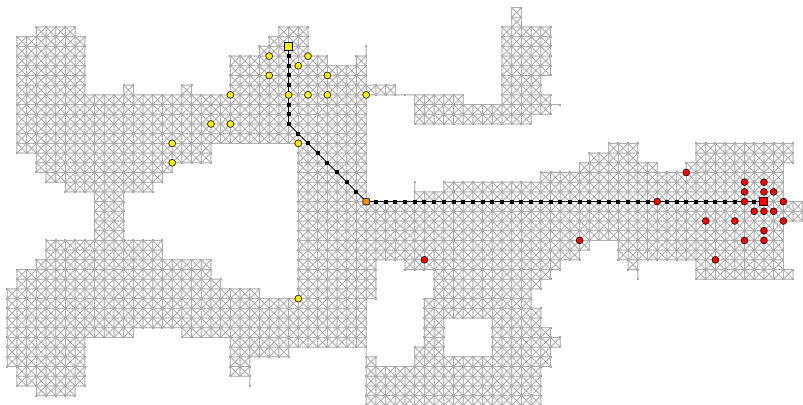
- Verteile hohe Knoten zuerst
- **Effekt:** $m(h, v)$ wird vor h verteilt (da höher) oder $m(h, v) = h$
- Wir können deswegen $m(h, v)$ per HL-Anfrage auf den bereits aufgebauten partiellen Labels berechnen
 - Wenn die Anfrage einen höchsten gemeinsamen Knoten findet, dann ist das $m(h, v)$ und $m(h, v) \neq h$
 - Wenn die Anfrage keinen gemeinsamen Knoten findet, dann ist $m(h, v) = h$
- Baue damit eine Pruning-Regel für Dijkstras Algorithmus
- Nachdem ein Knoten v aus der Queue genommen wird, berechne $m(h, v)$
 - $m(h, v) = h \rightarrow$ Füge h in das Label von v ein und relaxiere ausgehende Kanten von v
 - $m(h, v) \neq h \rightarrow$ Füge h nicht in das Label von v ein und prune die Suche an v

Beispiel: Grid Graph



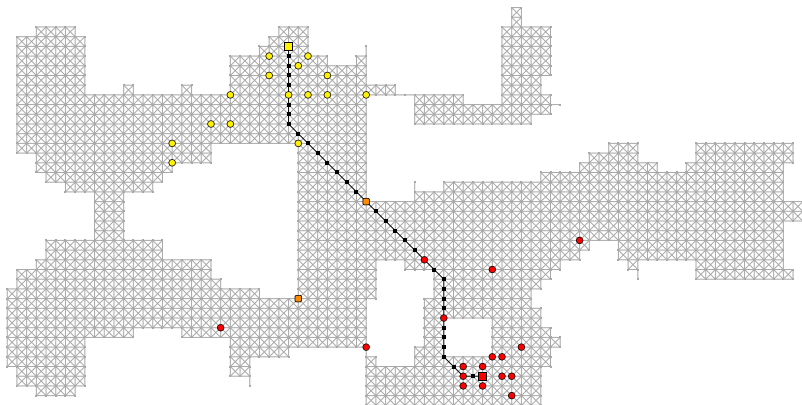
Autoren von [\[DGPW14\]](#) haben diese Bilder erstellt

Beispiel: Grid Graph



Autoren von [\[DGPW14\]](#) haben diese Bilder erstellt

Beispiel: Grid Graph



Autoren von [\[DGPW14\]](#) haben diese Bilder erstellt

Method	Preprocessing		Query
	time [h:mm]	space [GB]	time [μ s]
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 Speicherstelle nicht im Cache liegt
- HL-x: Benutze Top-Down-Ordnung für höchste 2^x Knoten

Method	Preprocessing		Query
	time [h:mm]	space [GB]	time [μ s]
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 Speicherstelle nicht im Cache liegt
- HL-x: Benutze Top-Down-Ordnung für höchste 2^x Knoten
- HL ist Faktor 100 schneller als CH (Speedup 10 Mio.)
- Hoher Speicherverbrauch (durch Kompression reduzierbar)

- 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

Beobachtung:

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

Beobachtung:

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

Können wir Geschwindigkeit gegen einfachere Bedienbarkeit eintauschen?

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

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-Implementierung „umsonst“

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-Implementierung „umsonst“

Nachteile:

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

Speichern der Label

- Berechne Labels in C++ (wie bei Hub Labeling)
- Aber speichere die Labels **direkt in der Datenbank**
- Ein Vorwärtslabel von Knoten v mit k Hubs:
 - erzeugt k **Tripel** $(v, u, d(v, u))$ in Tabelle forward
- Rückwärtslabel genauso in backward
- Ca. 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			backward		
node	hub	dist	node	hub	dist
1	1	0	1	1	0
1	4	1	1	4	4
1	5	2	2	2	0
1	7	3	2	6	1
2	2	0	2	7	4
⋮	⋮	⋮	⋮	⋮	⋮

Speichern der Label

- Berechne Labels in C++ (wie bei Hub Labeling)
- Aber speichere die Labels **direkt in der Datenbank**
- Ein Vorwärtslabel von Knoten v mit k Hubs:
 - erzeugt k **Tripel** $(v, u, d(v, u))$ in Tabelle forward
- Rückwärtslabel genauso in backward
- Ca. 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			backward		
node	hub	dist	node	hub	dist
1	1	0	1	1	0
1	4	1	1	4	4
1	5	2	2	2	0
1	7	3	2	6	1
2	2	0	2	7	4
⋮	⋮	⋮	⋮	⋮	⋮

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
```

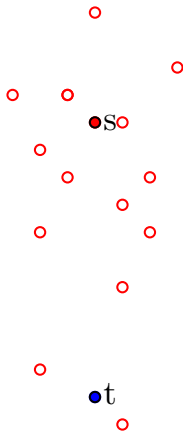
•s

•t

Algorithm 1: sql_dist

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

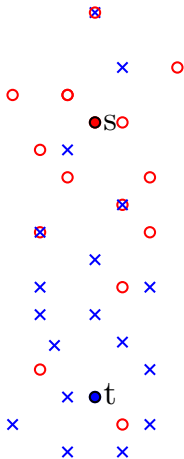
```
SELECT
    MIN(forward.dist+backward.dist)
FROM forward,backward
WHERE
    forward.node = s AND
    backward.node = t AND
    forward.hub = backward.hub
```



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
```

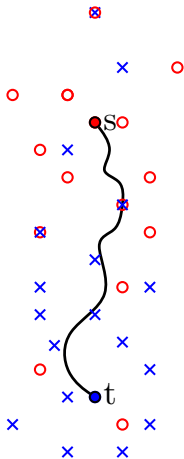


SQL-Query

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
```



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
```



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
```



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



Idee:

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

Phase 1:

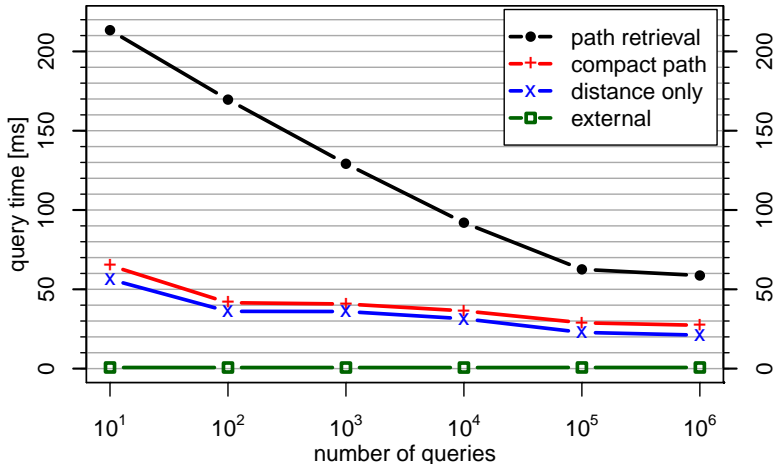
- Erzeuge Pfad in G^+ durch Hubs auf dem Pfad
- Erweitere Tabellen `forward` und `backward` um 2 Spalten: `Parent` und `Shortcut`
- Erhöht Speicherverbrauch der Tabelle von 19 auf 32 GB

Phase 2:

- Erzeuge Pfad in G durch Matchen von G^+ mit `shortcuts`

Ergebnisse

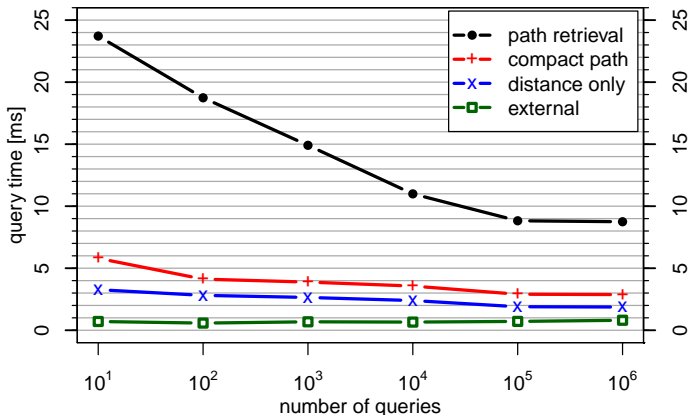
Setup: MS SQL Server 2008 R2 mit Daten auf HDD, kalter Cache



Beobachtung: Nicht schnell genug

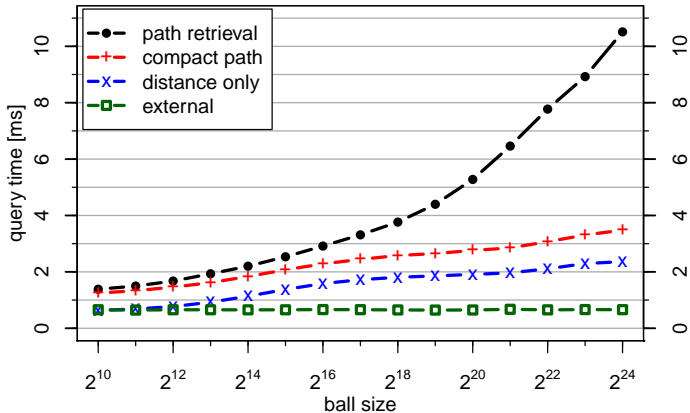
Ergebnisse (SSD)

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



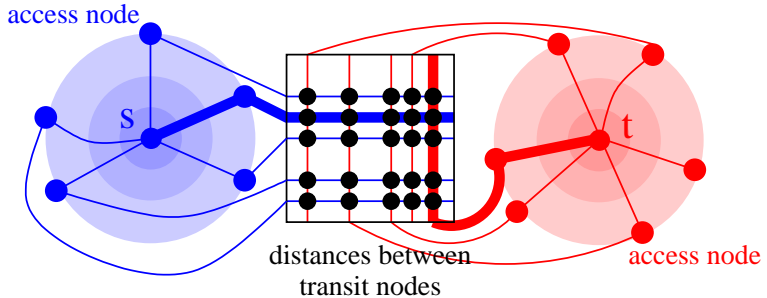
Beobachtung: SSD macht Queries schnell genug

Setup: Anfragen mit verschiedenem Rank, 10 000 Anfragen, kalter Cache

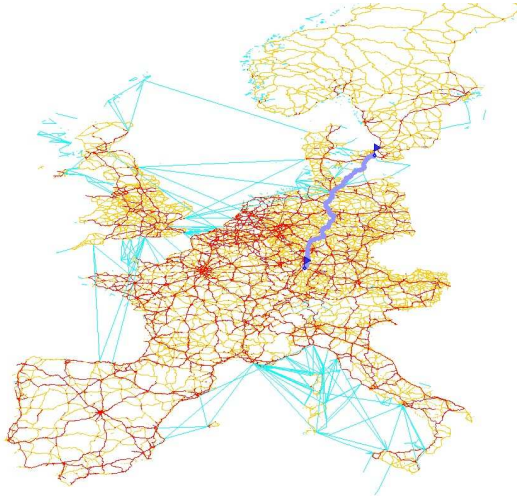


Beobachtung: praxisrelevante Anfragen sehr schnell

Transit Node Routing



Transit Node Routing

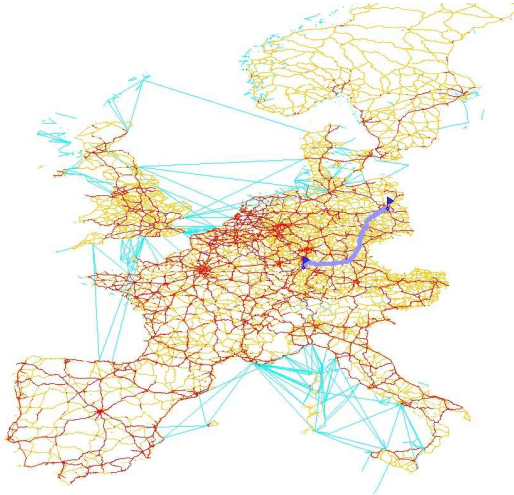


Beobachtung:

- 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

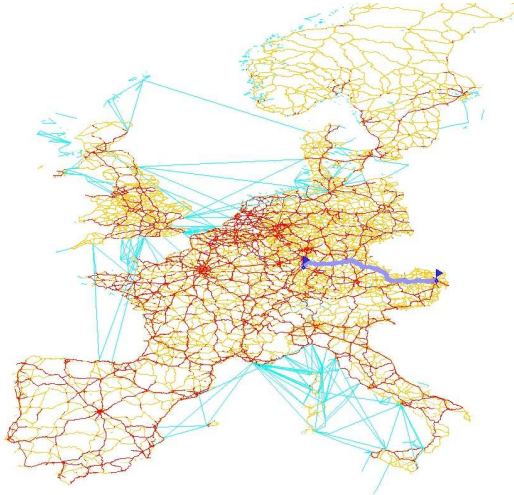


Beobachtung:

- 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

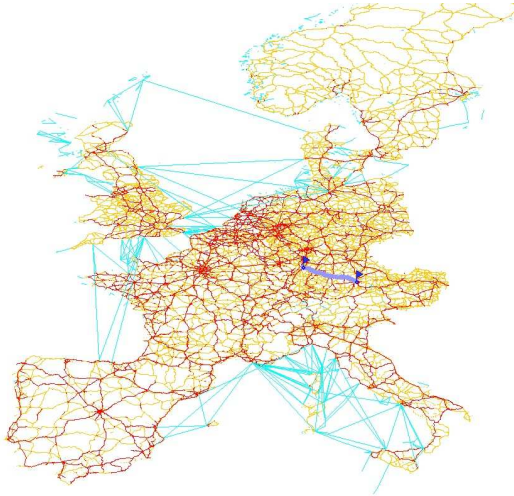


Beobachtung:

- 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

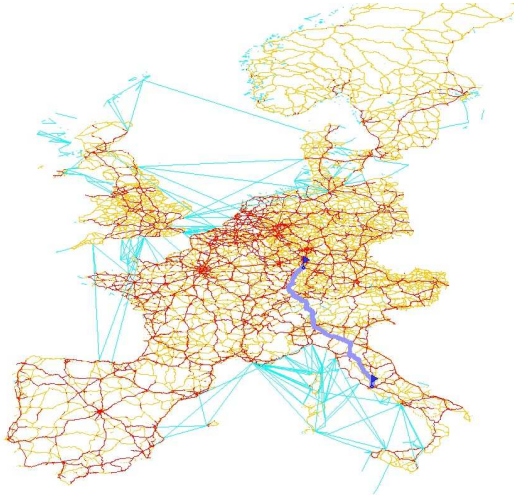


Beobachtung:

- 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

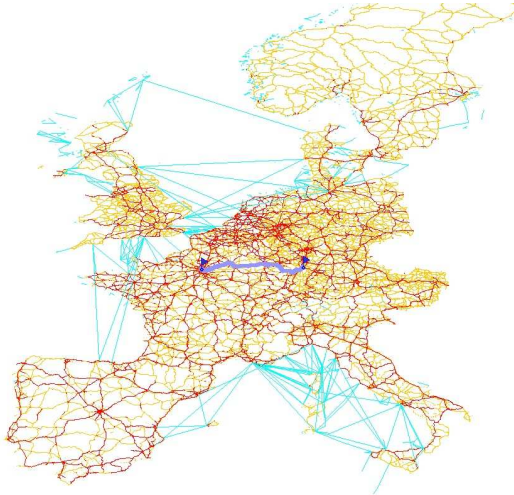


Beobachtung:

- 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

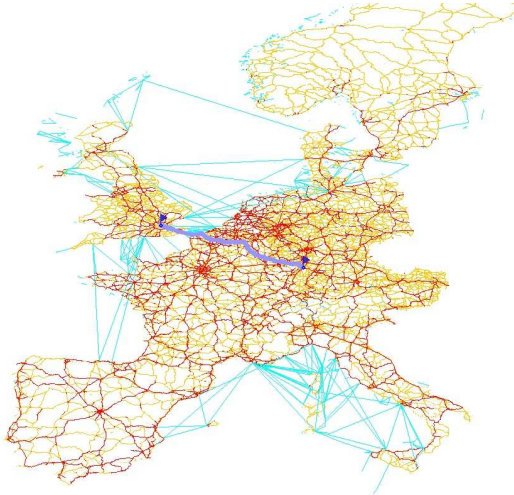


Beobachtung:

- 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

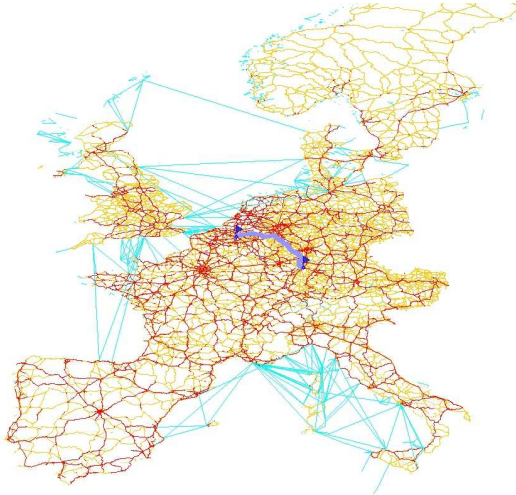


Beobachtung:

- 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

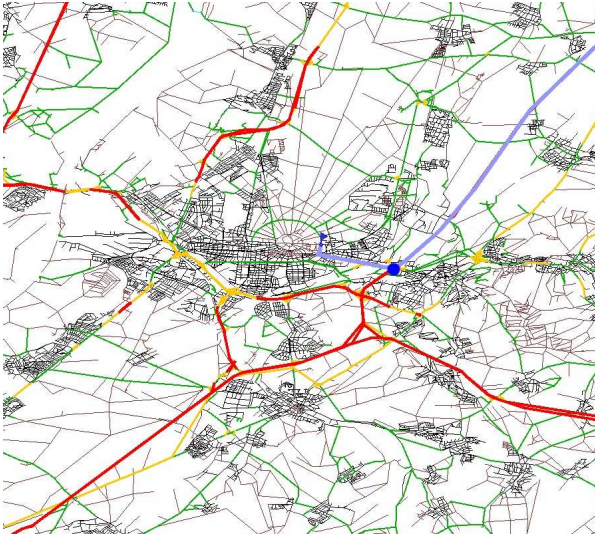


Beobachtung:

- 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

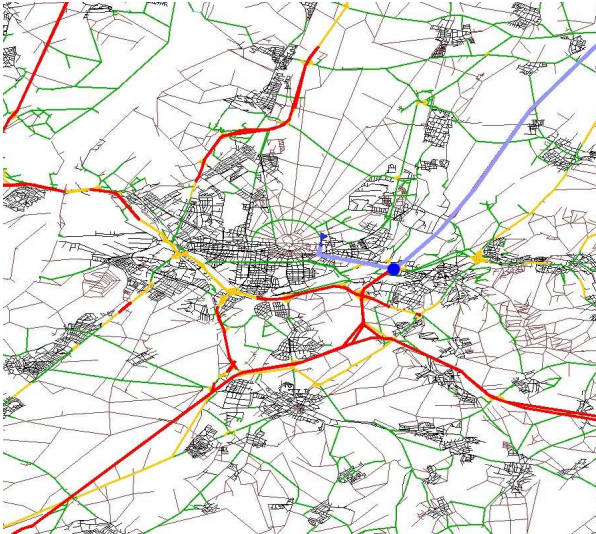


Beobachtung:

- 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

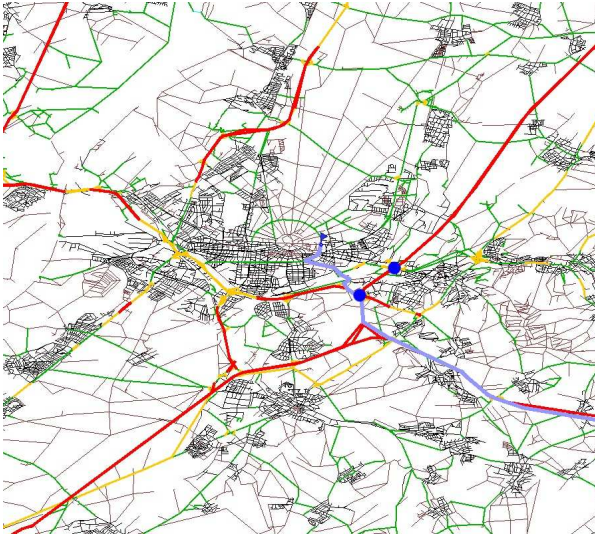


Beobachtung:

- 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

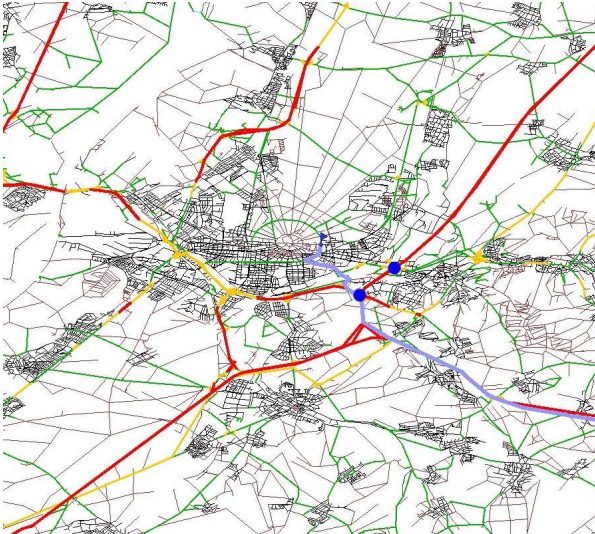


Beobachtung:

- 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

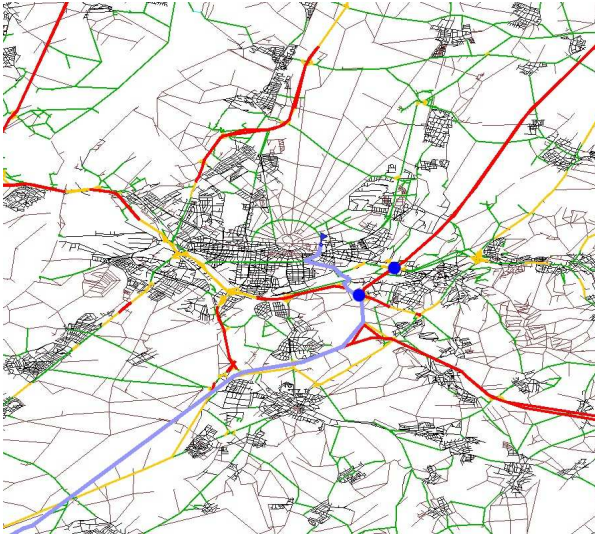


Beobachtung:

- 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

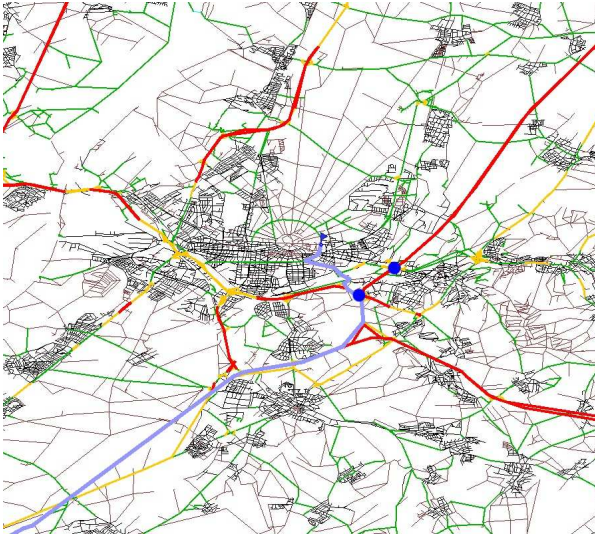


Beobachtung:

- 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

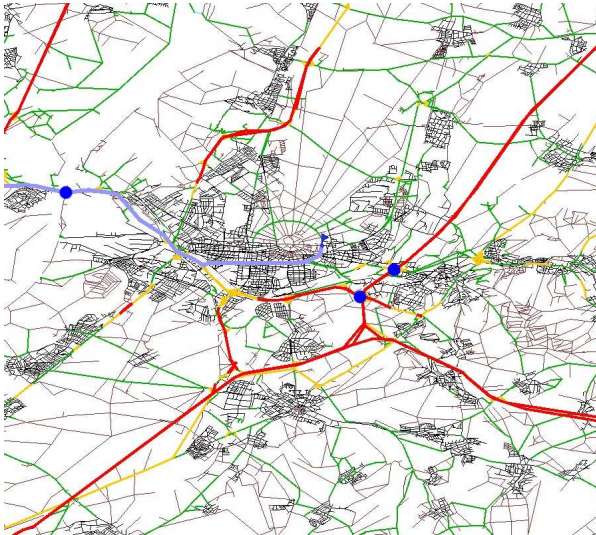


Beobachtung:

- 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

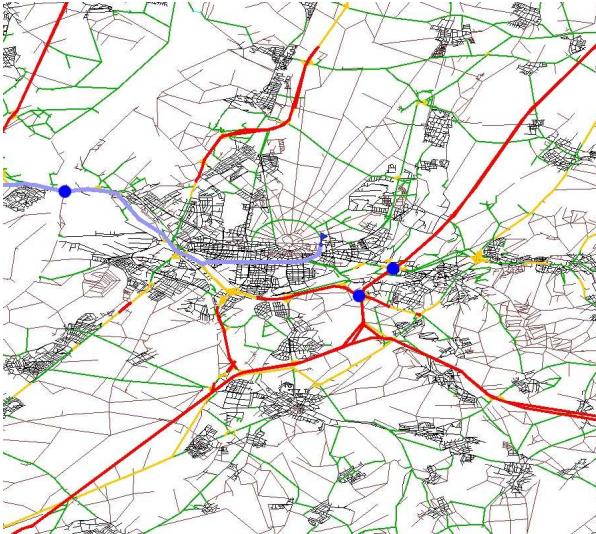


Beobachtung:

- 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

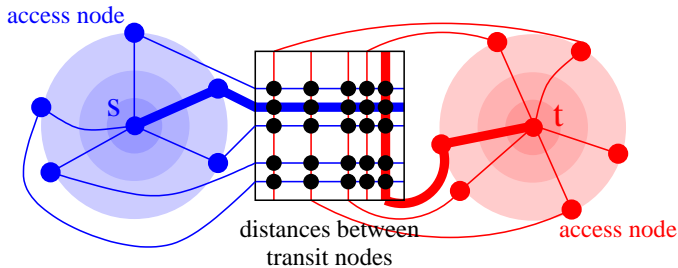
Transit Node Routing

Idee:

- Reduziere Anfragen auf Zugriffe in einer quadratischen Tabelle
- Identifiziere „wichtige“ Knoten
- Vollständige Distanztabelle zwischen diesen Knoten

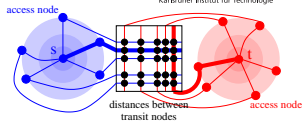
Probleme:

- Speicherverbrauch
- Nahe Anfragen



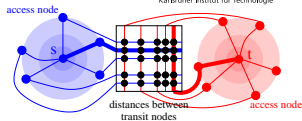
- Wähle **Transit-Knoten**: $\mathcal{T} \subseteq V$
- Bestimme **Access-Knoten** für jeden Knoten v :
 - Vorwärts-Access-Knoten $\vec{A}(v) \subseteq \mathcal{T}$
 - Rückwärts-Access-Knoten $\overleftarrow{A}(v) \subseteq \mathcal{T}$
- Vorberechnete Distanzen: $D_{\mathcal{T}}$ und d_A

Generelles TNR-Framework



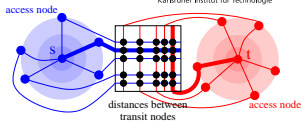
- Wähle **Transit-Knoten**: $\mathcal{T} \subseteq V$
- Bestimme **Access-Knoten** für jeden Knoten v :
 - Vorwärts-Access-Knoten $\vec{A}(v) \subseteq \mathcal{T}$
 - Rückwärts-Access-Knoten $\overleftarrow{A}(v) \subseteq \mathcal{T}$
- Vorberechnete Distanzen: $D_{\mathcal{T}}$ und d_A

Generelles TNR-Framework



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

Generelles TNR-Framework



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

Berechnete Distanz nur für hinreichend weite Anfragen korrekt!

- **Locality filter**: $L : V \times V \rightarrow \{\text{true}, \text{false}\}$
- $\text{true} \rightarrow$ **Fallback-Routine** für lokale Anfragen
- Einseitiger Fehler erlaubt

Also:

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

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?

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:

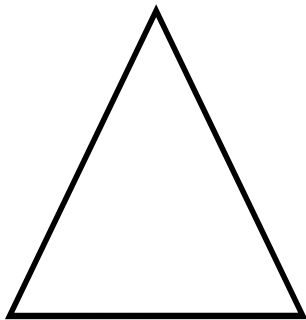
- Grid-based TNR [BFM06]
- Hierarchie-basiertes TNR mit geometrischem Lokalitätsfilter [BFM⁺07, GSSV12]
- CH-TNR [ALS13]

Transit Node Routing aufbauend auf CH:

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

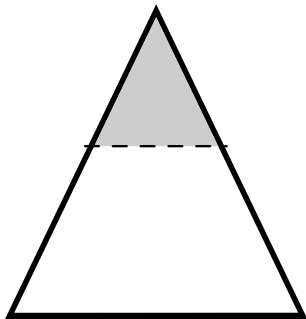
Transit Node Routing aufbauend auf CH:

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



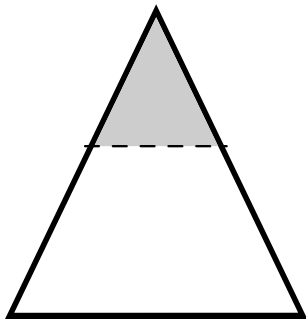
Transit Node Routing aufbauend auf CH:

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



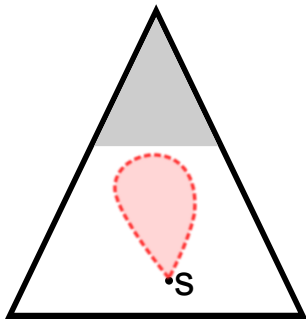
Transit Node Routing aufbauend auf CH:

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



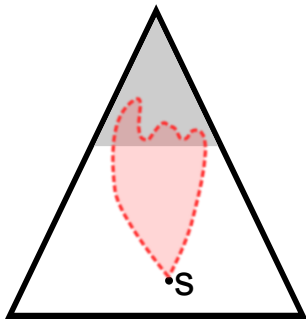
Transit Node Routing aufbauend auf CH:

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



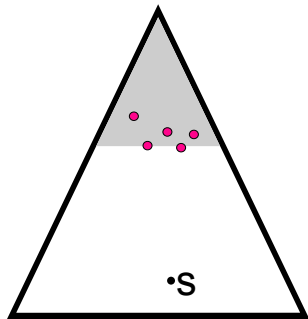
Transit Node Routing aufbauend auf CH:

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



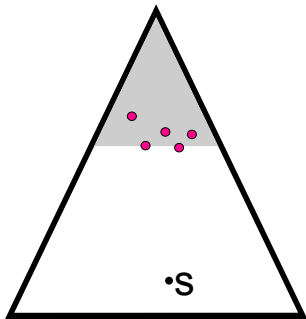
Transit Node Routing aufbauend auf CH:

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



Transit Node Routing aufbauend auf CH:

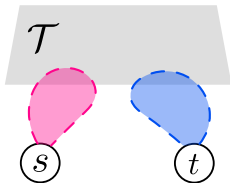
- CH für Vorberechnung und lokale Anfrage
- Top- k -Knoten sind Transit-Knoten. . .
- . . . und damit die Access-Knoten berechnen
- Lokalitatsfilter?





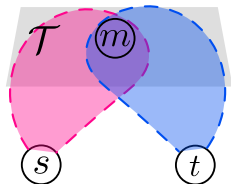
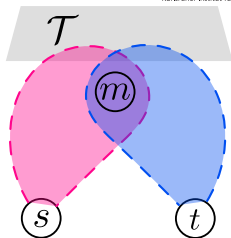
Eigenschaften einer lokalen Anfrage:

- Betrachte den höchsten Knoten m auf einem kürzesten Hoch-Runter- $s-t$ -Pfad



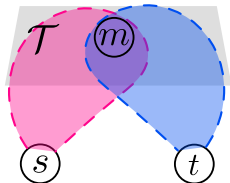
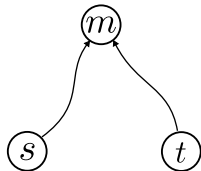
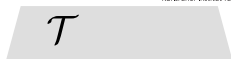
Eigenschaften einer lokalen Anfrage:

- Betrachte den höchsten Knoten m auf einem kürzesten Hoch-Runter- s - t -Pfad



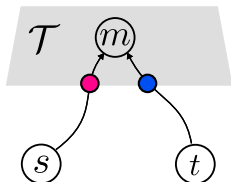
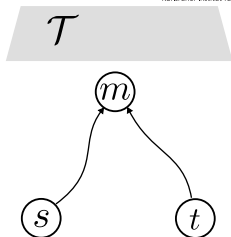
Eigenschaften einer lokalen Anfrage:

- Betrachte den höchsten Knoten m auf einem kürzesten Hoch-Runter- s - t -Pfad



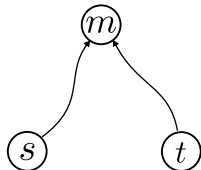
Eigenschaften einer lokalen Anfrage:

- Betrachte den höchsten Knoten m auf einem kürzesten Hoch-Runter- s - t -Pfad
- $m \notin \mathcal{T} \iff$ lokale Anfrage



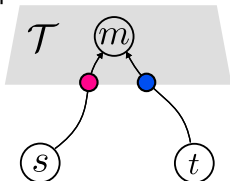
Eigenschaften einer lokalen Anfrage:

- Betrachte den höchsten Knoten m auf einem kürzesten Hoch-Runter- s - t -Pfad
- $m \notin \mathcal{T} \iff$ lokale Anfrage



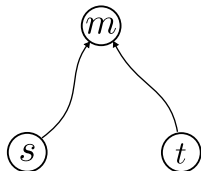
Suchraum-basierter Lokalisierungsfilter:

- Speichere Suchraum **unterhalb** der Transit-Knoten $S : V \rightarrow V \setminus \mathcal{T}$ 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!**



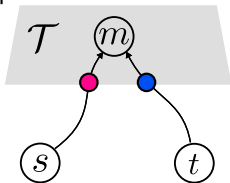
Eigenschaften einer lokalen Anfrage:

- Betrachte den höchsten Knoten m auf einem kürzesten Hoch-Runter- s - t -Pfad
- $m \notin \mathcal{T} \iff$ lokale Anfrage



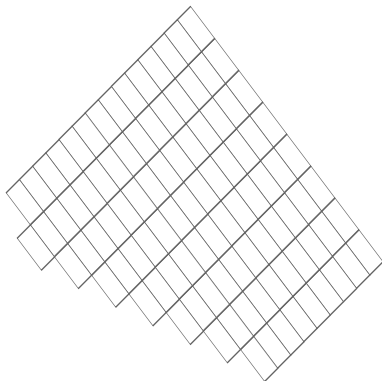
Suchraum-basierter Lokalisierungsfilter:

- Speichere Suchraum **unterhalb** der Transit-Knoten $S : V \rightarrow V \setminus \mathcal{T}$ 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



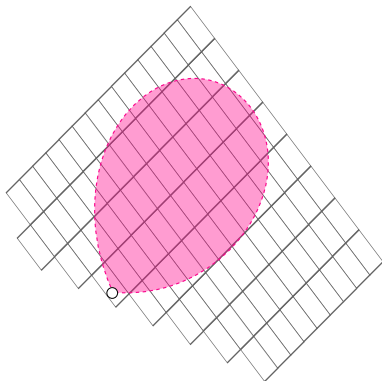
Lokalitätsfilter

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



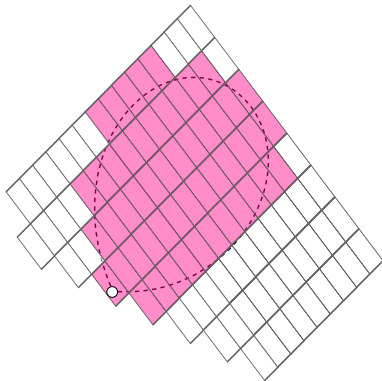
Lokalitätsfilter

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



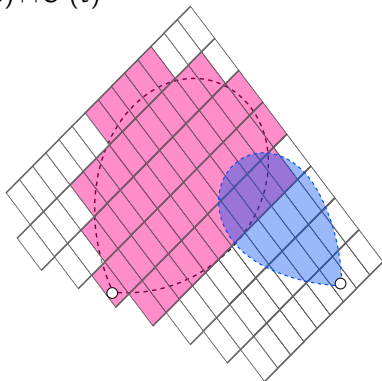
Lokalitätsfilter

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



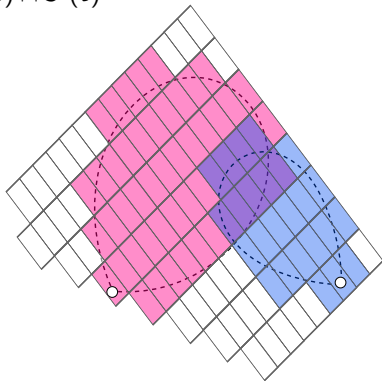
Lokalitätsfilter

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



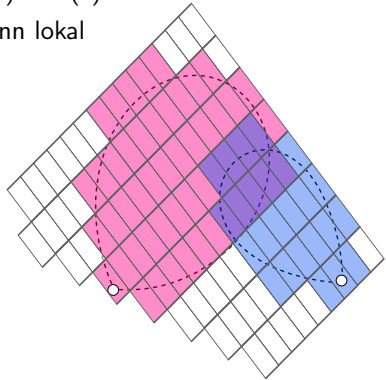
Lokalitätsfilter

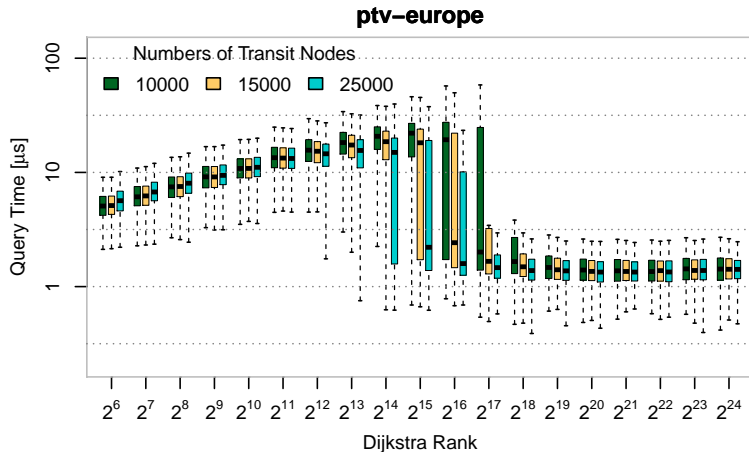
- Partitioniere Graphen in Regionen
- Überapproximation des Suchraums mittels **berührter** Regionen
- Wenn x im Suchraum $S(s)$ ist, dann ist die Region $R(x)$ im approximierten Suchraum $S'(s)$
- $m \in S(s) \cap S(t) \implies R(m) \in S'(s) \cap S'(t)$



Lokalitätsfilter

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



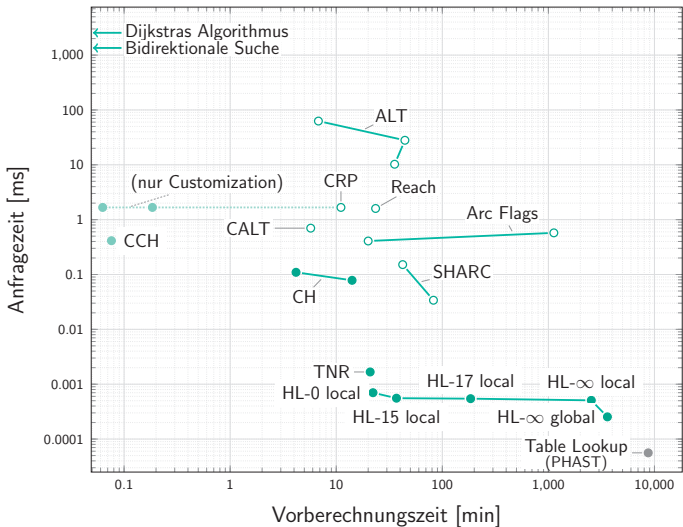


Frage: Welche durchschnittliche Laufzeit ergibt sich?

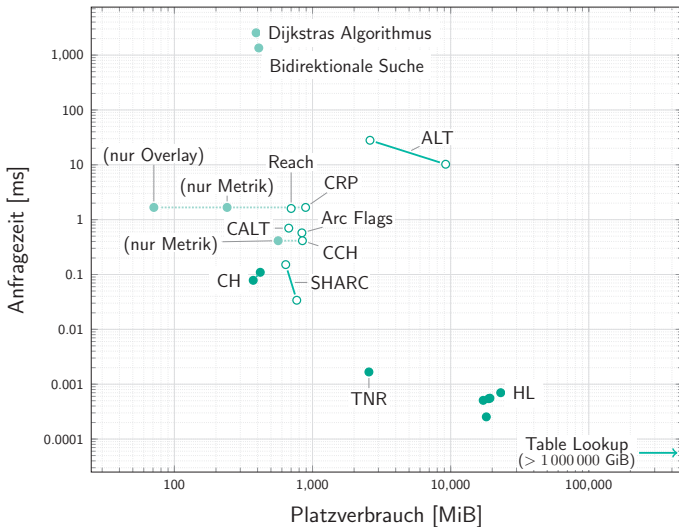
Transit Node Routing

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

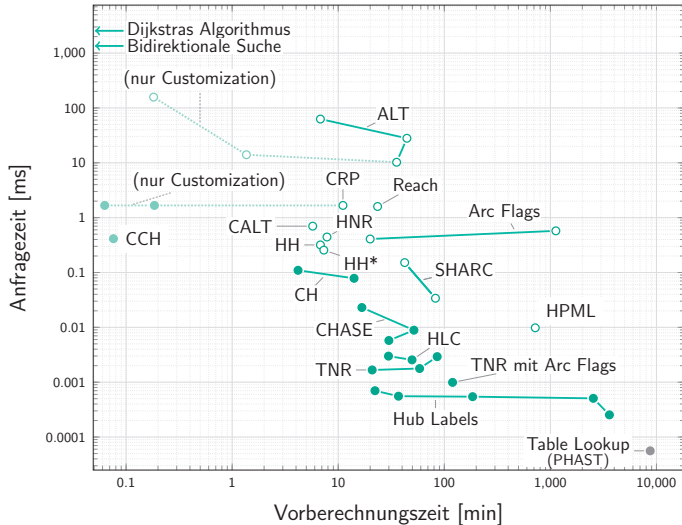
Übersicht bisherige Techniken



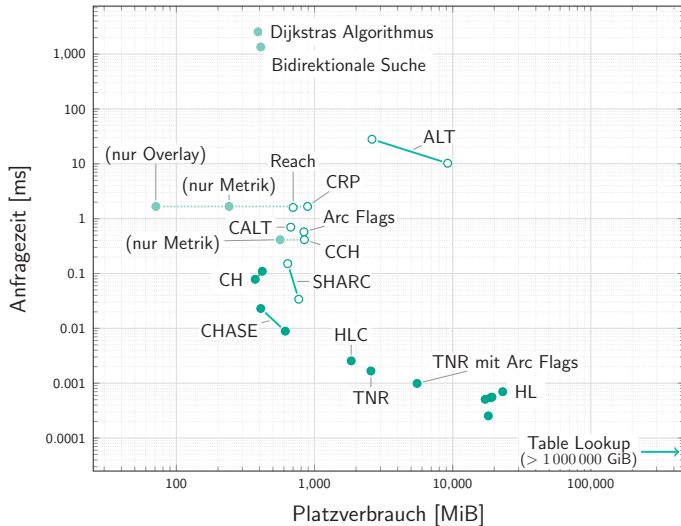
Übersicht bisherige Techniken







„Komplett“übersicht One-to-One





„Komplett“übersicht One-to-One





-  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.


-  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.

-  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.