Algorithmen für Routenplanung

9. Sitzung, Sommersemester 2014

Ben Strasser | 16. Mai 2014

INSTITUT FÜR THEORETISCHE INFORMATIK · ALGORITHMIK · PROF. DR. DOROTHEA WAGNER

Kürzeste Wege in Straßennetzwerken

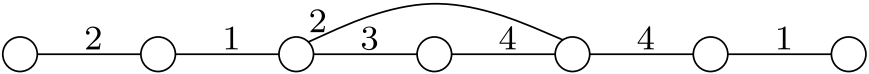
Beschleunigungstechniken (Fortsetzung)

- Transit-Node Routing
- Hub-Labels

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Wiederholung: CH

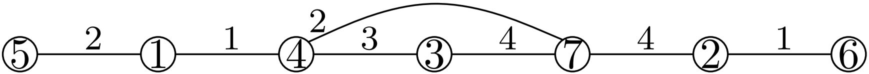
preprocessing:



Wiederholung: CH

preprocessing:

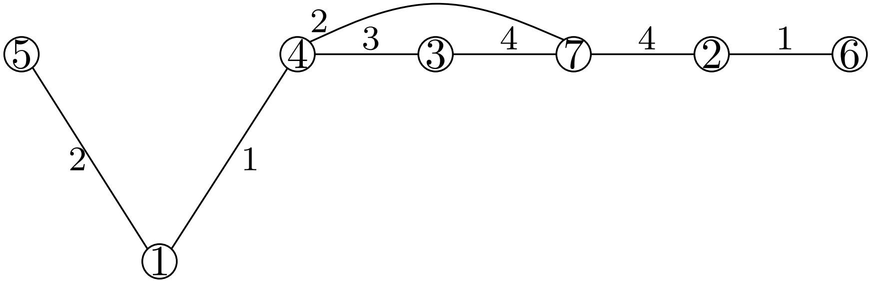
- ordne Knoten nach Wichtigkeit



Wiederholung: CH

preprocessing:

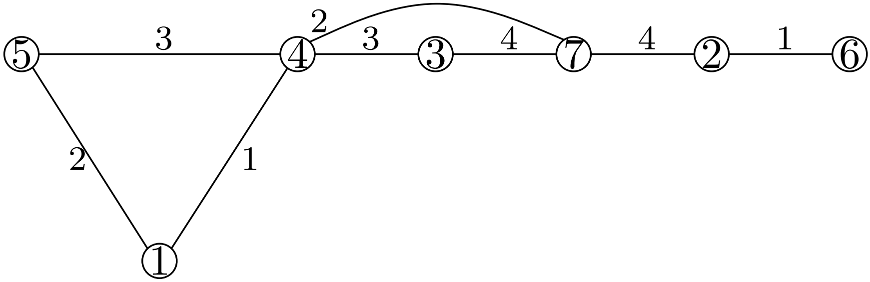
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu



Wiederholung: CH

preprocessing:

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



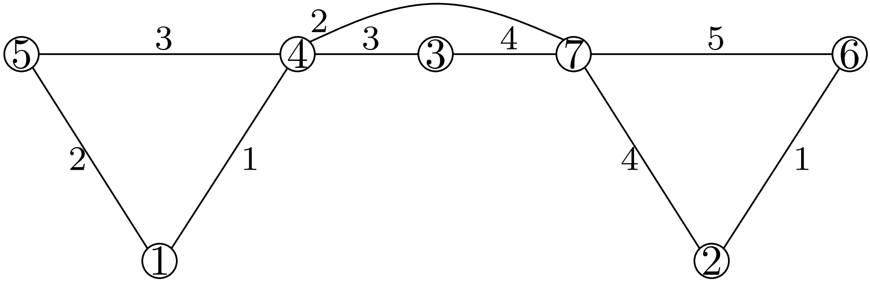
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Wiederholung: CH

Karlsruher Institut für Technologie

preprocessing:

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

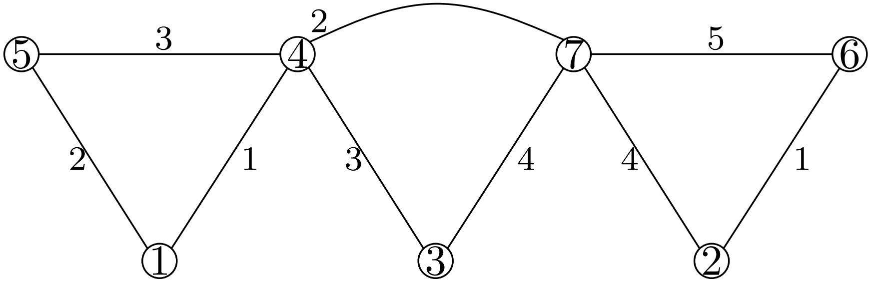


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Wiederholung: CH

preprocessing:

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

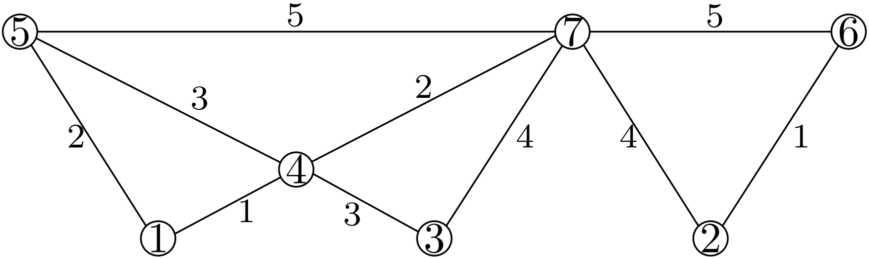


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Wiederholung: CH

preprocessing:

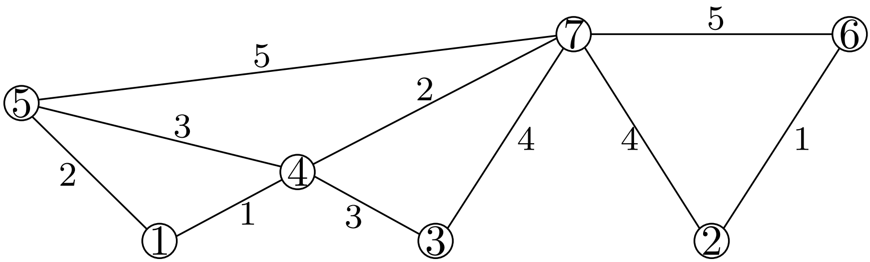
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu



Wiederholung: CH

preprocessing:

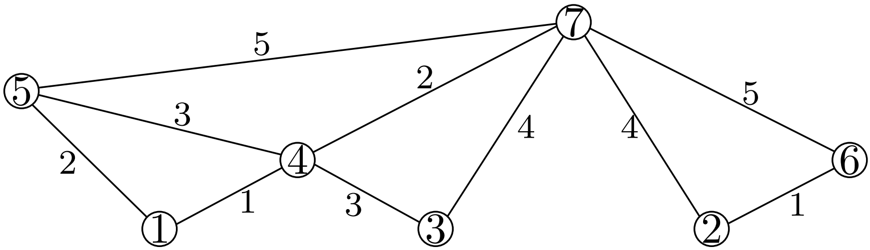
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu



Wiederholung: CH

preprocessing:

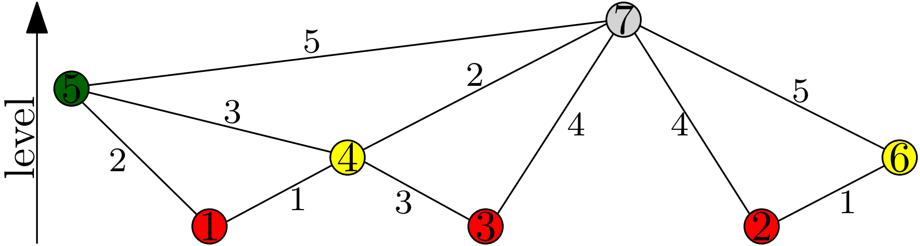
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu



Wiederholung: CH

preprocessing:

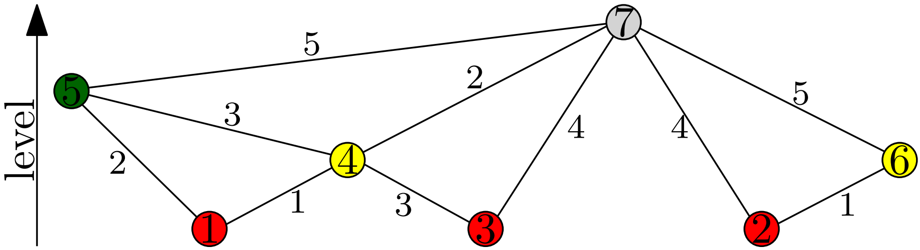
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu
- Levelzuordnung



Wiederholung: CH

Punkt-zu-Punkt Anfragen

- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten



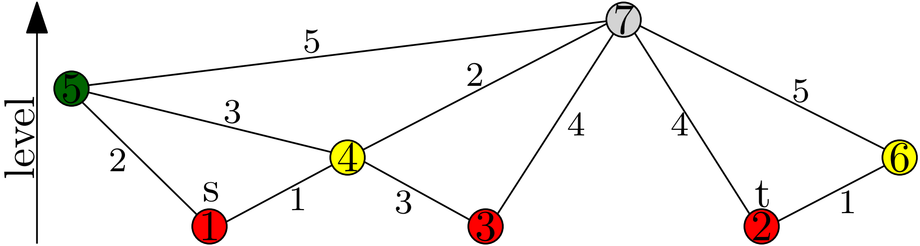
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Wiederholung: CH

Karlsruher Institut für Technologie

Punkt-zu-Punkt Anfragen

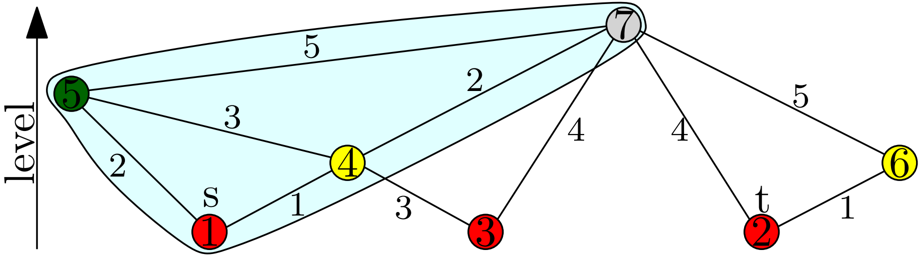
- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten



Wiederholung: CH

Punkt-zu-Punkt Anfragen

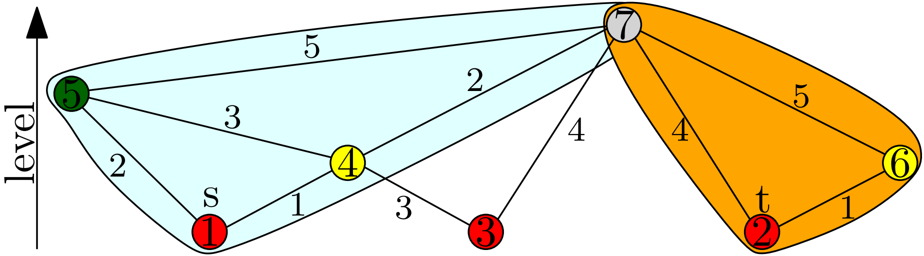
- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten



Wiederholung: CH

Punkt-zu-Punkt Anfragen

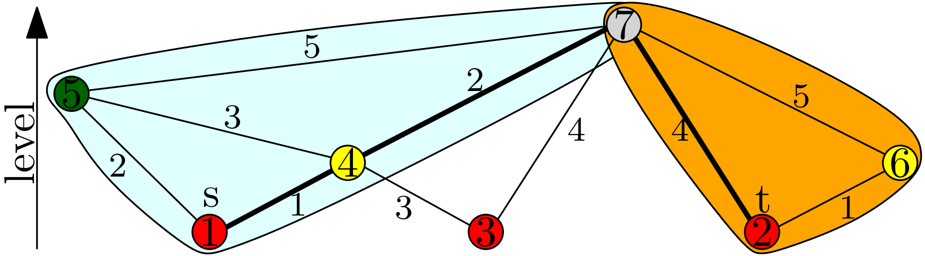
- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten



Wiederholung: CH

Punkt-zu-Punkt Anfragen

- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

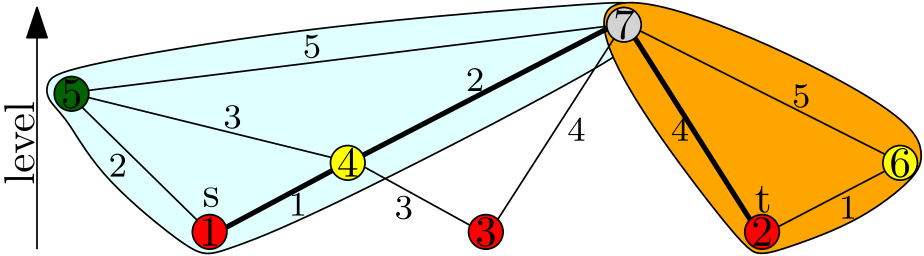
Wiederholung: CH

Punkt-zu-Punkt Anfragen

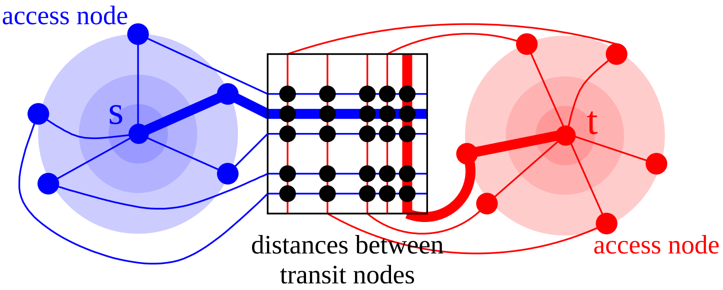
- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten

Korrektheit:

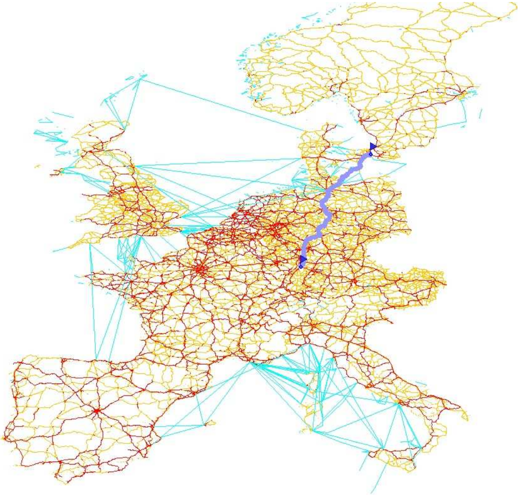
- es gibt einen wichtigsten Knoten auf dem Pfad
- dieser wird von Vorwärts- und Rückwärtssuche gescannt



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

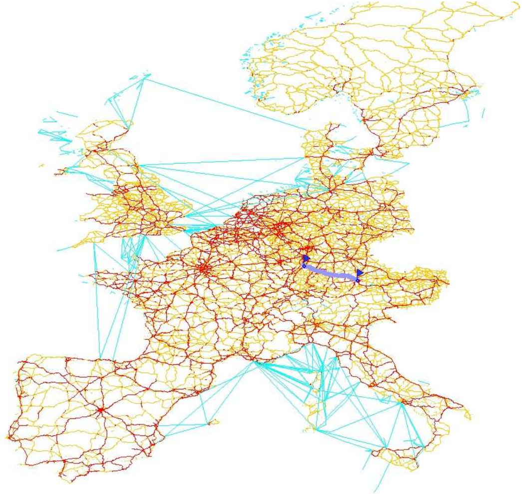
KIT
Karlsruher Institut für Technologie

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

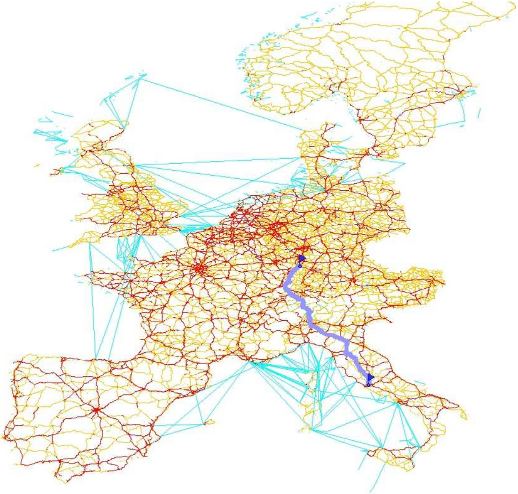


Beobachtung:

- 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

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

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

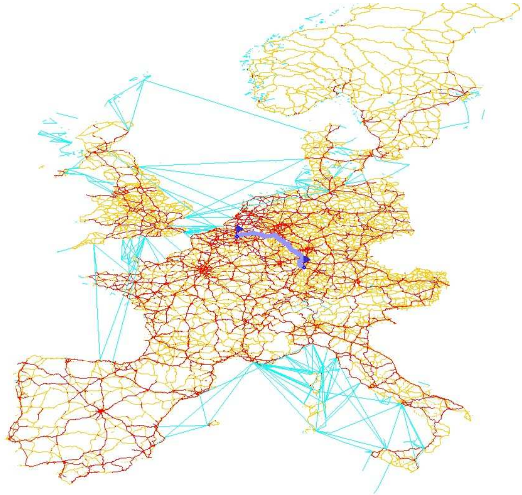
Transit-Node Routing

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



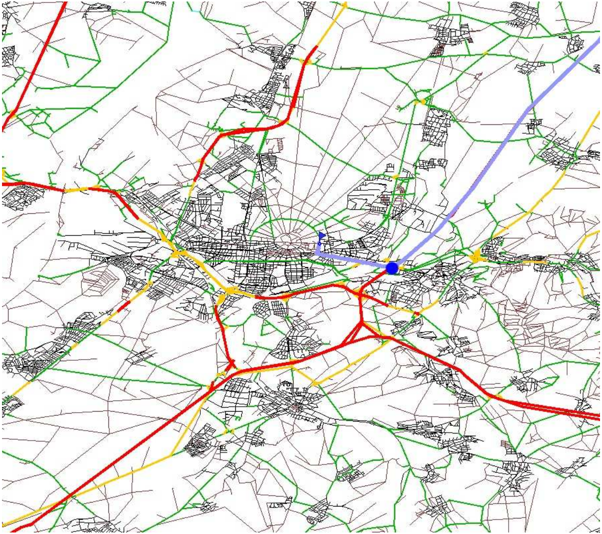
Beobachtung:

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

Karlsruhe nach. . .
Brüssel

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Transit-Node Routing

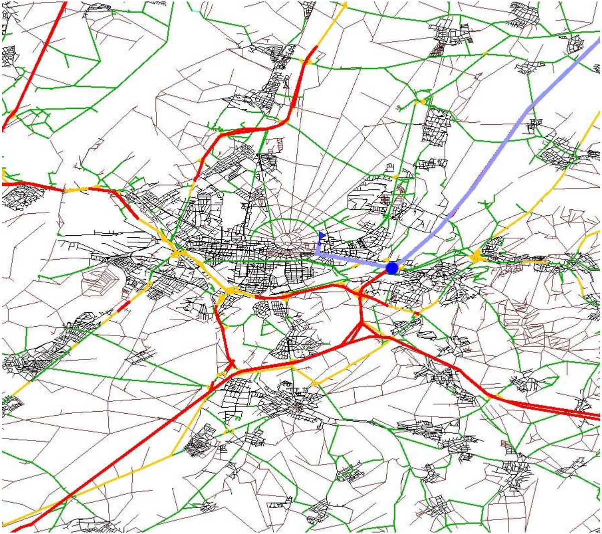


Beobachtung:

- 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

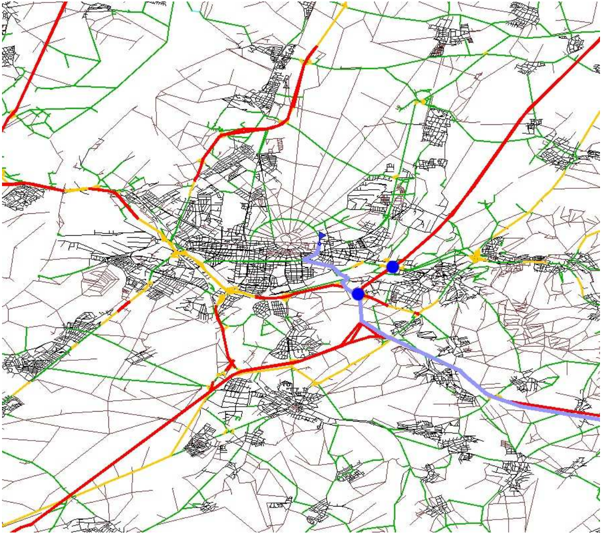


Beobachtung:

- 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

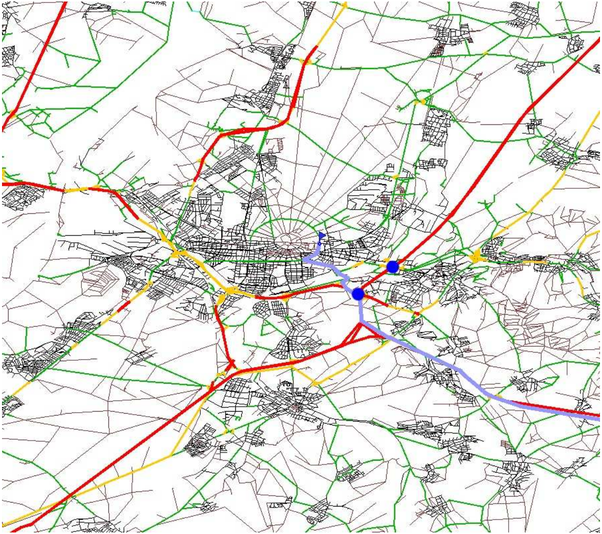


Beobachtung:

- 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

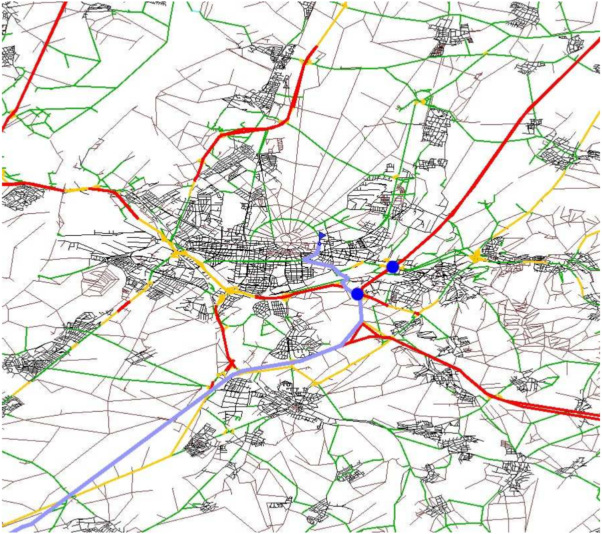


Beobachtung:

- 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

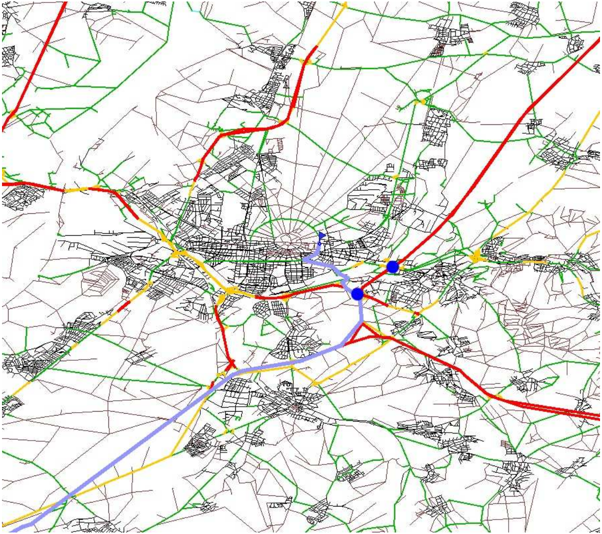


Beobachtung:

- 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



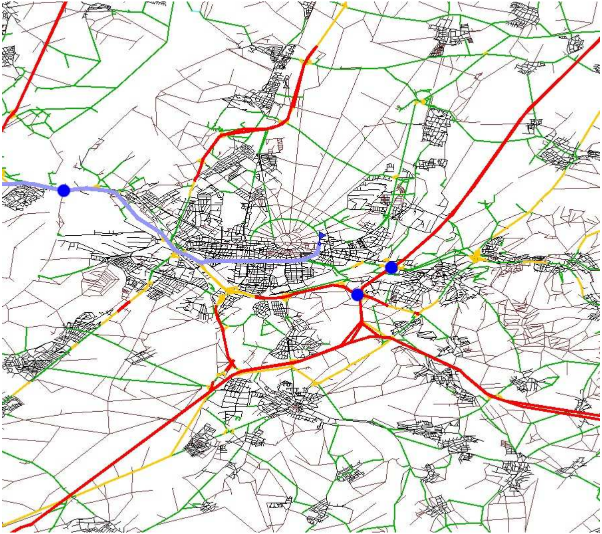
Beobachtung:

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

Karlsruhe nach. . .
Paris

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Transit-Node Routing

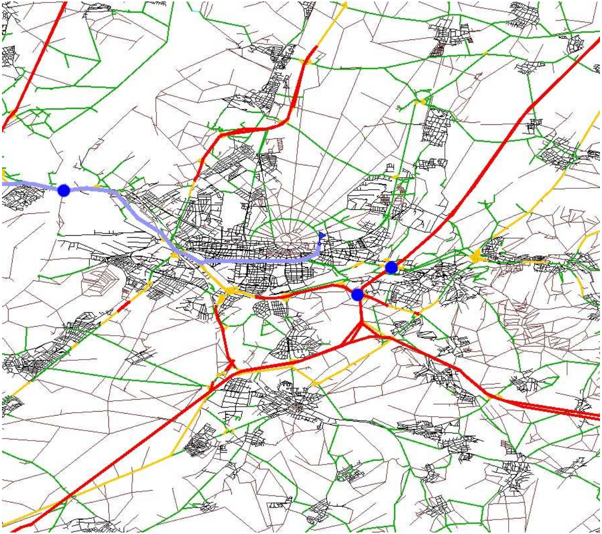


Beobachtung:

- 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

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

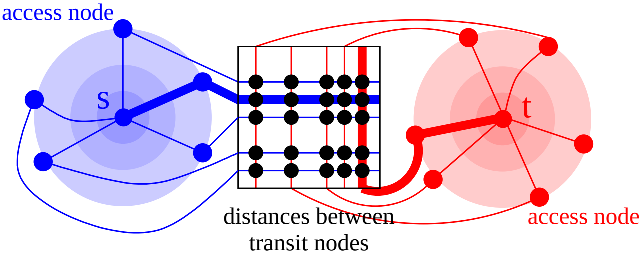
Transit-Node Routing

Idee:

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

Probleme:

- Speicherverbrauch
- nahe Anfragen



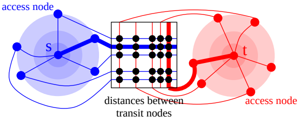
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

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

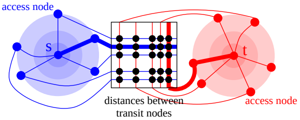
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

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

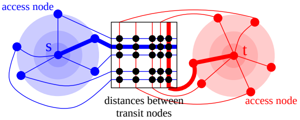
Generelles TNR-Framework



- 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
- $\text{dist}(s,t) \stackrel{?}{=} \min_{u \in \overrightarrow{A}(s), v \in \overleftarrow{A}(t)} \{d_A(s,u) + D_T(u,v) + d_A(v,t)\}$

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

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
- $\mathrm{dist}(s,t) \stackrel{?}{=} \min_{u \in \overrightarrow{A}(s), v \in \overleftarrow{A}(t)} \{d_A(s,u) + D_T(u,v) + d_A(v,t)\}$

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]

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

CH-TNR

Contraction Hierarchies based Transit Node Routing

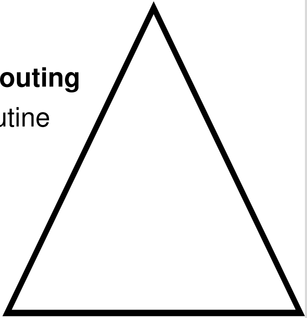
- Use CH for preprocessing and as fallback routine
- Take top k nodes as Transit nodes. . .

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

CH-TNR

Contraction Hierarchies based Transit Node Routing

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

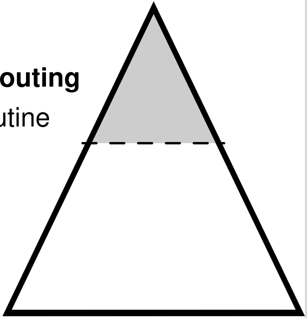


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

CH-TNR

Contraction Hierarchies based Transit Node Routing

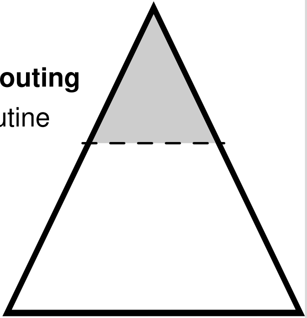
- Use CH for preprocessing and as fallback routine
- Take top k nodes as Transit nodes...



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

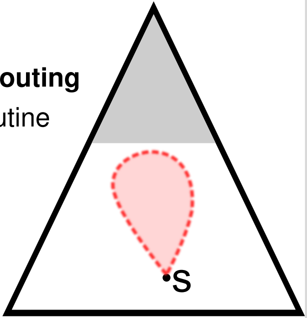


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

CH-TNR

Contraction Hierarchies based Transit Node Routing

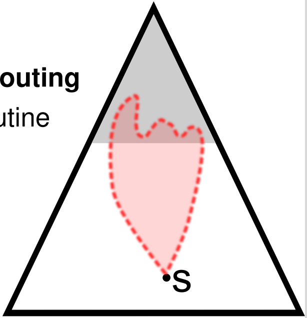
- Use CH for preprocessing and as fallback routine
- Take top k nodes as Transit nodes…
- …and use them to compute Access nodes



CH-TNR

Contraction Hierarchies based Transit Node Routing

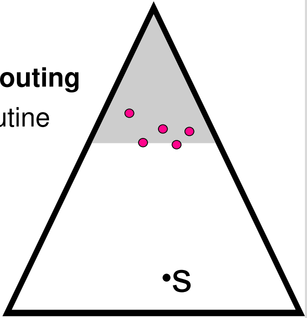
- Use CH for preprocessing and as fallback routine
- Take top k nodes as Transit nodes. . .
- . . . and use them to compute Access nodes



CH-TNR

Contraction Hierarchies based Transit Node Routing

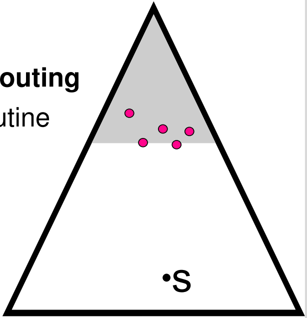
- Use CH for preprocessing and as fallback routine
- Take top *k* nodes as Transit nodes...
- ...and use them to compute Access nodes



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!?

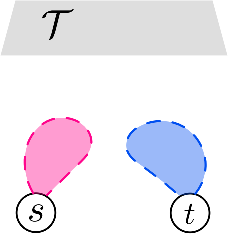


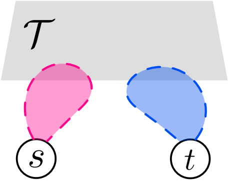
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

CH-based TNR

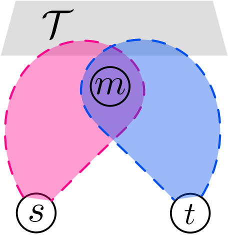
Properties of a local query

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



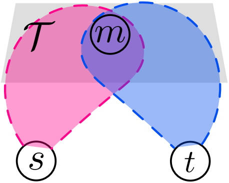


CH-based TNR



Properties of a local query

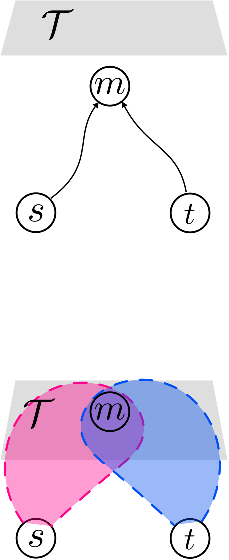
- Consider meeting point m of a $s - t$–search



CH-based TNR

Properties of a local query

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

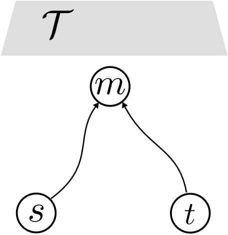


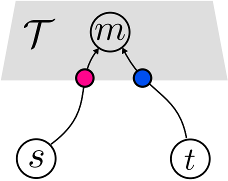
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

CH-based TNR

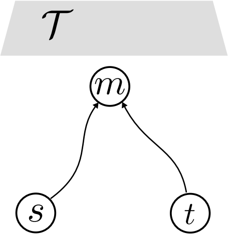
Properties of a local query

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





CH-based TNR

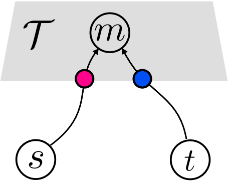


Properties of a local query

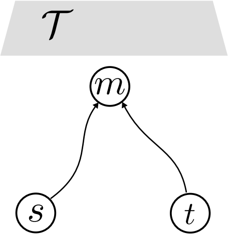
- Consider meeting point m of a $s-t$–search
- $m \notin \mathcal{T} \iff$ local query

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

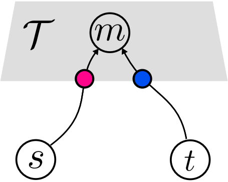


Properties of a local query

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

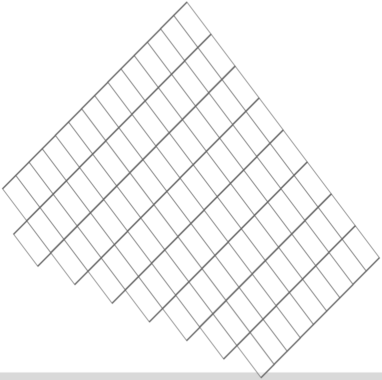
Search space based locality filter

- 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

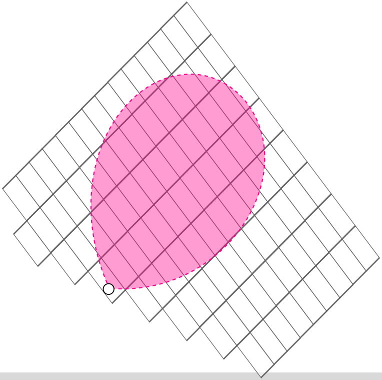
- Partition the graph into regions
- Over-Approximation of the search spaces using touched regions.



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Locality Filter

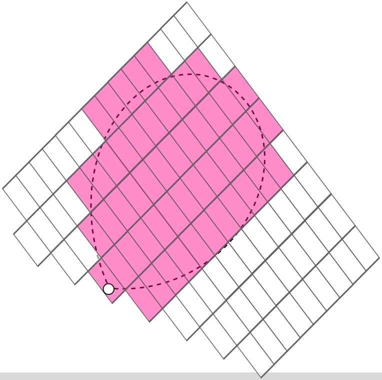
- Partition the graph into regions
- Over-Approximation of the search spaces using touched regions.



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Locality Filter

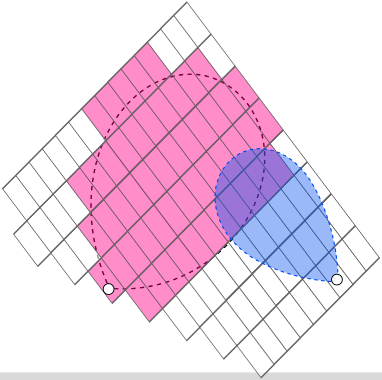
- Partition the graph into regions
- Over-Approximation of the search spaces using touched regions.



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Locality Filter

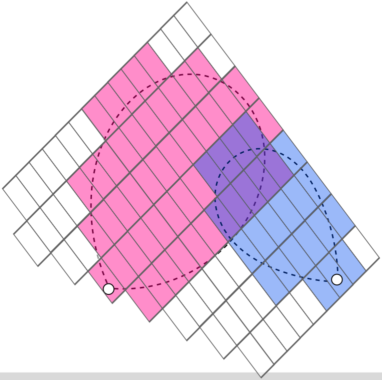
- Partition the graph into regions
- Over-Approximation of the search spaces using touched regions.
- Correctness: $m \in S(s) \cap S(t) \Longrightarrow R(m) \in S'(s) \cap S'(t)$



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Locality Filter

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



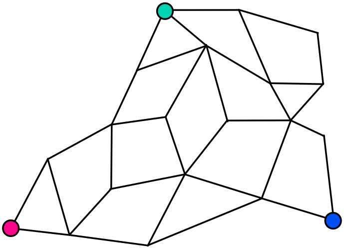
Locality Filter

- 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 [M88]

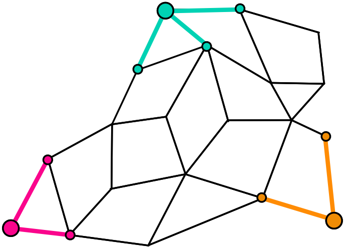


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Locality Filter

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

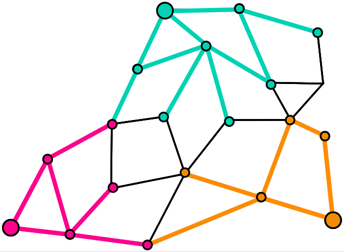
- Use graph Voronoi regions around transit nodes
- Represent region with its center node
- $\mathrm{Vor}(v) := \{u \in V : \forall w \in \mathcal{T} \setminus \{v\} : \mu(u,v) \leq \mu(u,w)\}$ for $v \in \mathcal{T}$
- Efficient computation with multi-source Dijkstra [M88]



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 [M88]

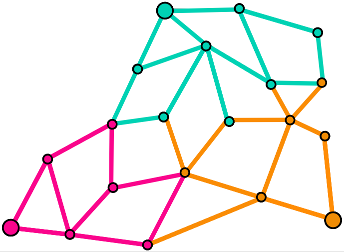


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

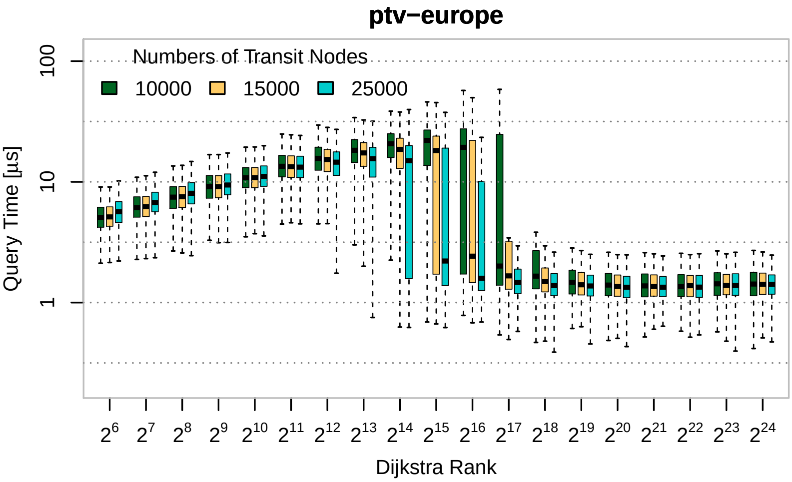
Locality Filter

- 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 [M88]



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

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Labeling

Vorberechung:

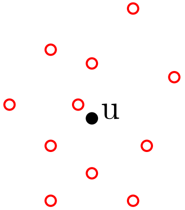
- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Labeling

Vorberechung:

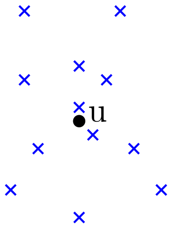
- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
  - $\text{dist}(u, v)$ für jeden Hub $v \in L_f(u)$



Labeling

Vorberechung:

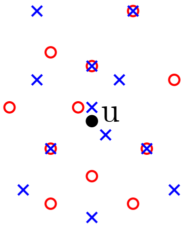
- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
  - $\mathrm{dist}(u, v)$ für jeden Hub $v \in L_f(u)$
  - $\mathrm{dist}(v, u)$ für jeden Hub $v \in L_b(u)$



Labeling

Vorberechung:

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



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Labeling

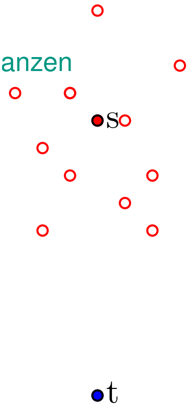
Vorberechung:

- 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

Labeling

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

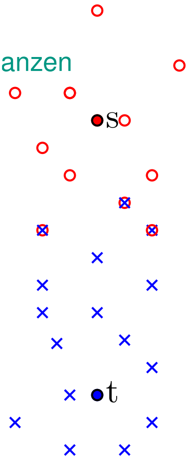


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Labeling

Vorberechung:

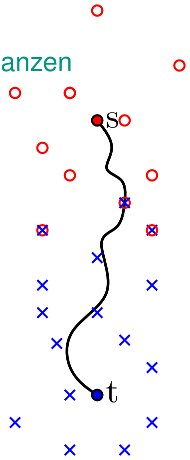
- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
  - dist(u, v) für jeden Hub $v \in L_f(u)$
  - dist(v, u) für jeden Hub $v \in L_b(u)$
- die Label müssen die cover property einhalten:
  $\forall s, t,\ L_f(s) \cap L_b(t)$ überdeckt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



Labeling

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

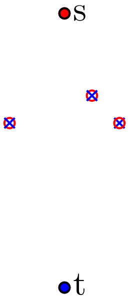


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Labeling

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:**

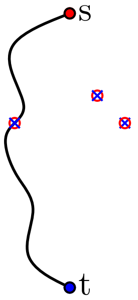
- finde Knoten $v \in L_f(s) \cap L_b(t)$ ...

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Labeling

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



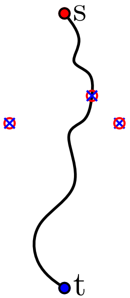
s–t Anfrage:

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

Labeling

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:

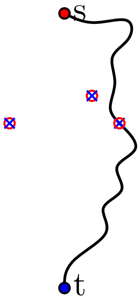
- finde Knoten $v \in L_f(s) \cap L_b(t) \ldots$
- $\ldots$ der dist$(s, v)+$ dist(v, t) minimiert

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Labeling

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:

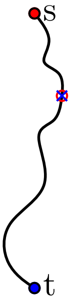
- finde Knoten $v \in L_f(s) \cap L_b(t) \ldots$
- $\ldots$ der $\text{dist}(s, v) + \text{dist}(v, t)$ minimiert

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Labeling

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:

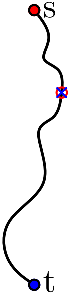
- finde Knoten $v \in L_f(s) \cap L_b(t) \ldots$
- $\ldots$ der $\text{dist}(s, v) + \text{dist}(v, t)$ minimiert

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Labeling

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

Beobachtungen:

- Laufzeit hängt von Labelgrösse ab
- wie effizient berechnen?

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Hub Labels

Speichern der Labels:

- als Menge von Hub,Distanz Paaren

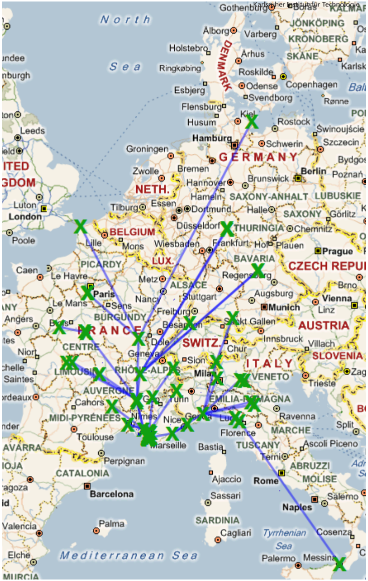
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Hub Labels

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

Hub Labels

Speichern der Labels:

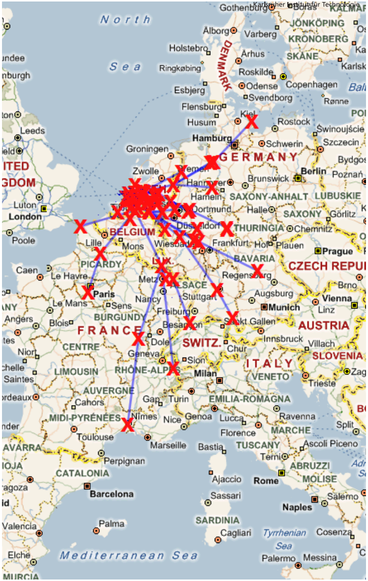
- als Menge von Hub,Distanz Paaren

$L_f(s)$

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

$L_b(t)$

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



Hub Labels

Speichern der Labels:

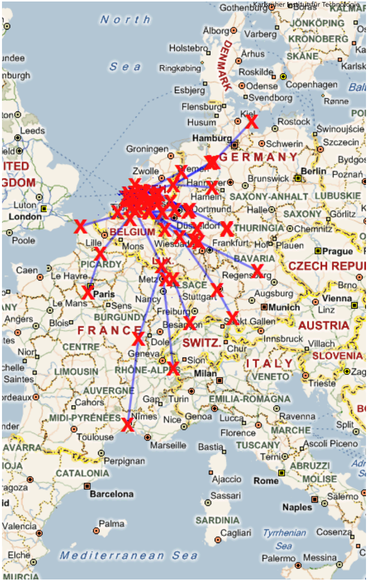
- als Menge von Hub,Distanz Paaren

Anfrage:

- scannen von zwei Arrays
- nur einige Speicherzugriffe nötig

$L_f(s)$: | 1,0 | 4,1 | 5,2 | 7,3 | ∞

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



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Hub Labels

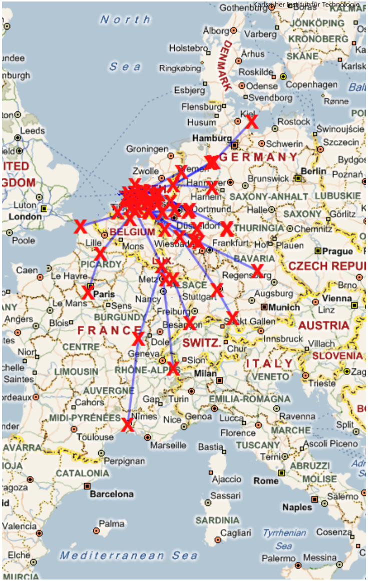
Speichern der Labels:

- als Menge von Hub,Distanz Paaren

Anfrage:

- scannen von zwei Arrays
- nur einige Speicherzugriffe nötig

$L_f(s)$	1,0	4,1	5,2	7,3	
					∞
$L_b(t)$	2,0	6,1	7,4	8,1	9,3



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Hub Labels

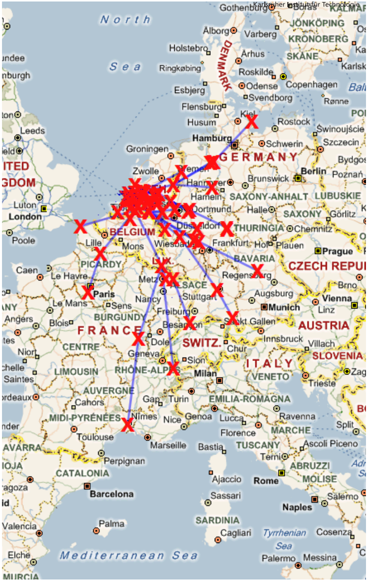
Speichern der Labels:

- als Menge von Hub,Distanz Paaren

Anfrage:

- scannen von zwei Arrays
- nur einige Speicherzugriffe nötig

$L_f(s)$	1,0	4,1	5,2	7,3	∞
$L_b(t)$	2,0	6,1	7,4	8,1	9,3



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

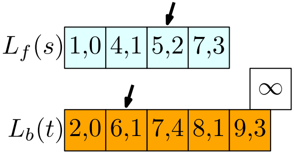
Hub Labels

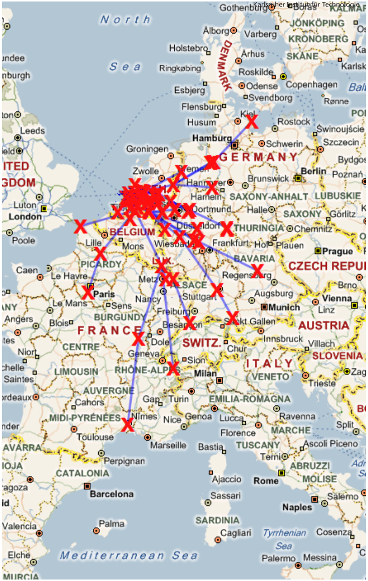
Speichern der Labels:

- als Menge von Hub,Distanz Paaren

Anfrage:

- scannen von zwei Arrays
- nur einige Speicherzugriffe nötig





Hub Labels

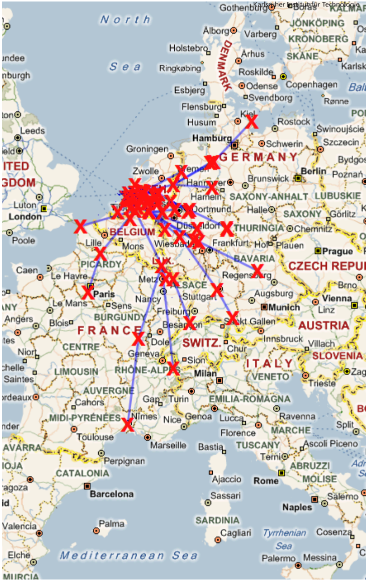
Speichern der Labels:

- als Menge von Hub,Distanz Paaren

Anfrage:

- scannen von zwei Arrays
- nur einige Speicherzugriffe nötig

$L_f(s)$	1,0	4,1	5,2	7,3	∞
$L_b(t)$	2,0	6,1	7,4	8,1	9,3



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Hub Labels

Speichern der Labels:

- als Menge von Hub,Distanz Paaren

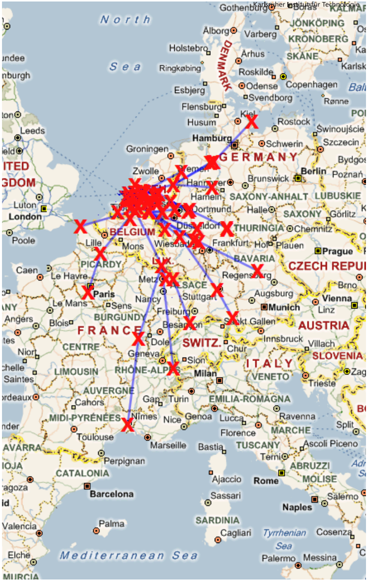
Anfrage:

- scannen von zwei Arrays
- nur einige Speicherzugriffe nötig

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

7

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



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Hub Labels

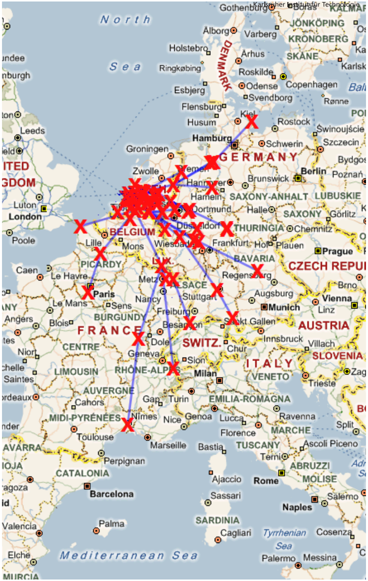
Speichern der Labels:

- als Menge von Hub,Distanz Paaren

Anfrage:

- scannen von zwei Arrays
- nur einige Speicherzugriffe nötig

$L_f(s)$	1,0	4,1	5,2	7,3	
					7
$L_b(t)$	2,0	6,1	7,4	8,1	9,3



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Hub Labels

Speichern der Labels:

- als Menge von Hub,Distanz Paaren

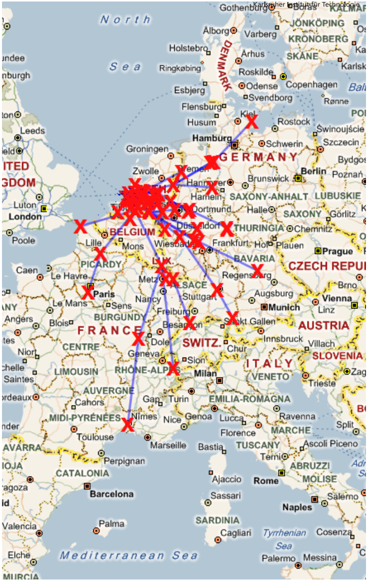
Anfrage:

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

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

$\boxed{7}$

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

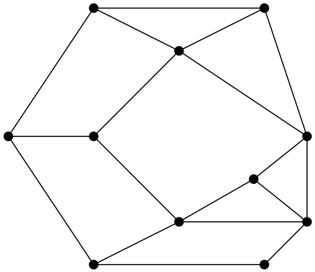


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Vorberechnung

Idee:

- benutze Knotenordnung

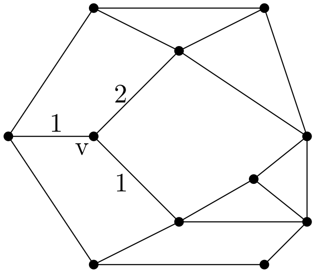


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Vorberechnung

Idee:

- benutze Knotenordnung
- kontrahiere Knoten v

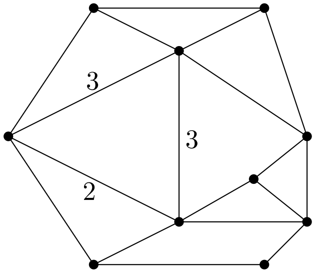


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Vorberechnung

Idee:

- benutze Knotenordnung
- kontrahiere Knoten *v*

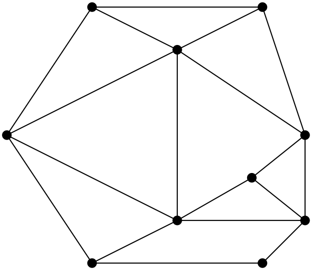


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Vorberechnung

Idee:

- benutze Knotenordnung
- kontrahiere Knoten *v*

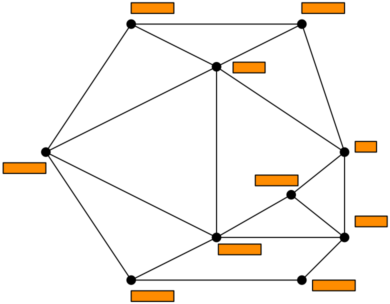


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Vorberechnung

Idee:

- benutze Knotenordnung
- kontrahiere Knoten v
- berechne Labels rekursiv

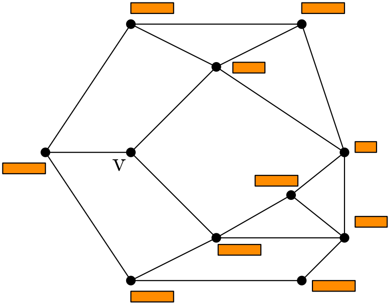


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Vorberechnung

Idee:

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

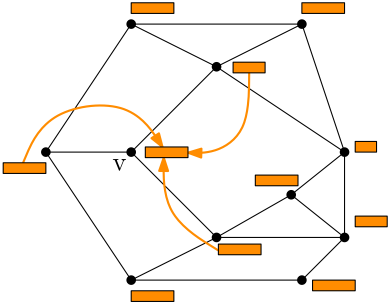


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Vorberechnung

Idee:

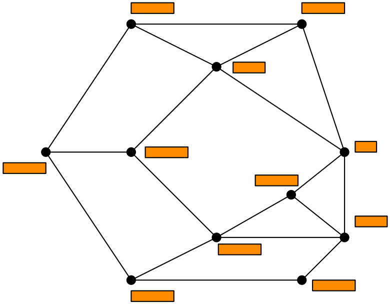
- benutze Knotenordnung
- kontrahiere Knoten *v*
- berechne Labels rekursiv
- merge Labels der Aufwärts-Nachbarn von *v*
- dünne Label aus



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*

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

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

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

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.

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Weitere Beschleunigung

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)

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Weitere Beschleunigung

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)

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Weitere Beschleunigung

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

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Weitere Beschleunigung

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		$\boxed{10}$

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Weitere Beschleunigung

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

$Or(s,t)$ $\boxed{10}$

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

Weitere Beschleunigung

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	$\boxed{10}$

Weitere Beschleunigung

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

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

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

HL-x mit 2^x top-down ordnung, ∞ mit Hybrid Ordnung

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

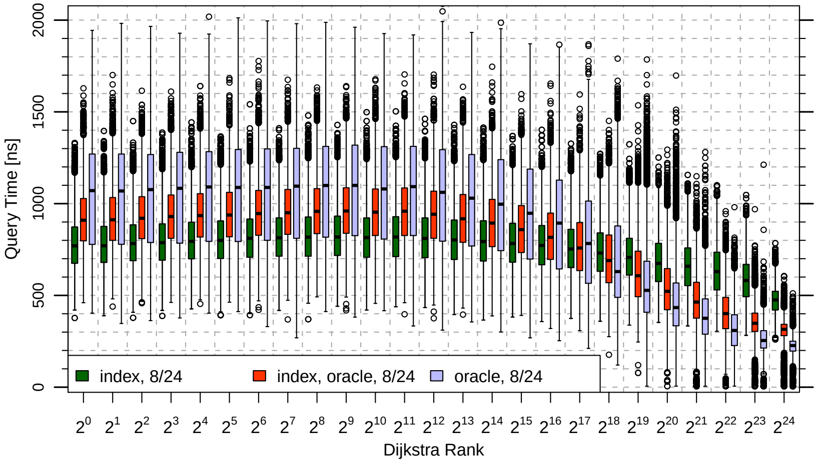
Ergebnisse

method	preprocessing time [h:m]	preprocessing space [GB]	query 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

HL-x mit 2^x top-down ordnung, ∞ mit Hybrid Ordnung

- HL ist Faktor 100 schneller als CH (Speedup 10 Mio)
- hoher Speicherverbrauch (durch Kompression reduzierbar)
- nur 5 mal langsamer als ein Speicherzugriff

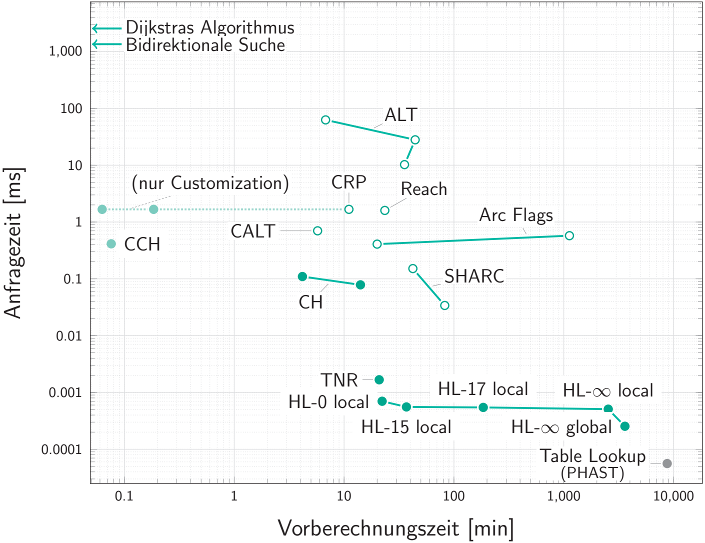
Lokale Queries



Zusammenfassung

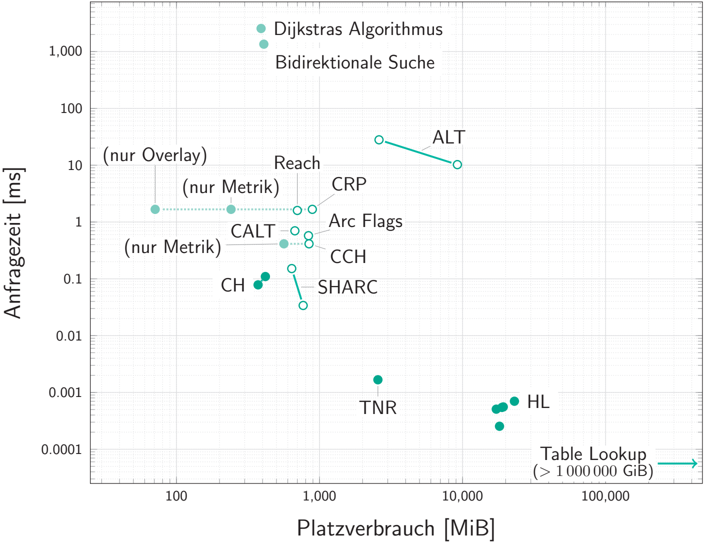
- Knotenordnung definiert Labeling
- Beschleunigung gegenüber CH von Faktor mehr als 100
- durch bessere Lokalität
- nur 5 mal langsamer als ein Speicherzugriff
- schnellster Algorithmus momentan
- beschleunigt lokale und globale Anfragen
- aber Speicherverbrauch sehr hoch
- wird zu einem späterem Zeitpunkt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

Übersicht bisherige Techniken

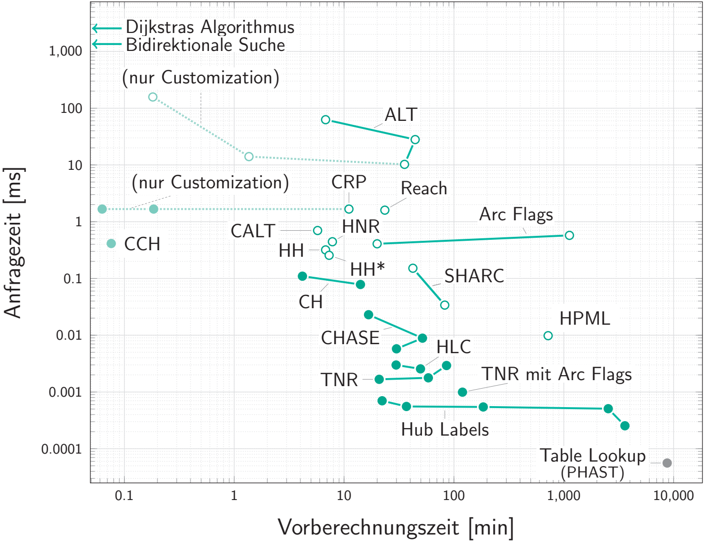


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

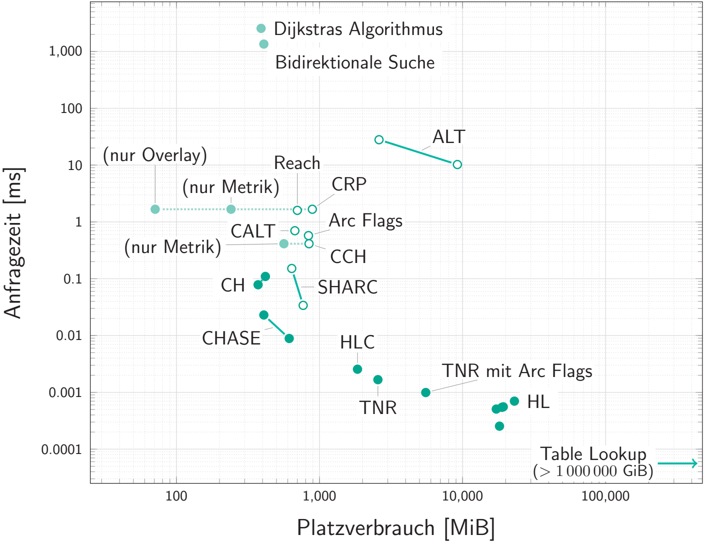
Übersicht bisherige Techniken



“Komplett”übersicht One-to-One



“Komplett”übersicht One-to-One



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Nächste Termine

Montag, 19.5.2014

Mittwoch, 21.5.2014

Literatur I

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.

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.