Karlsruher Institut für Technologie

Algorithmen für Routenplanung

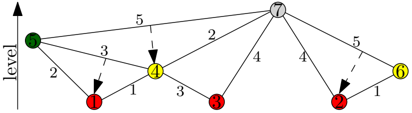
8. Sitzung, Sommersemester 2012

Thomas Pajor | 21. Mai 2012

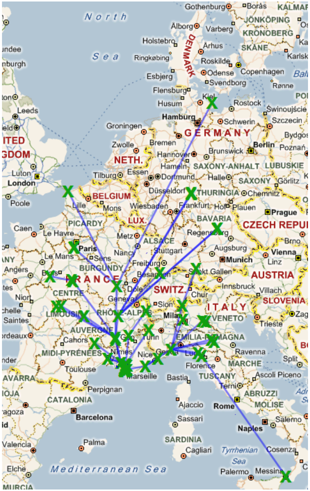
INSTITUT FÜR THEORETISCHE INFORMATIK · ALGORITHMIK · PROF. DR. DOROTHEA WAGNER

Letztes Mal

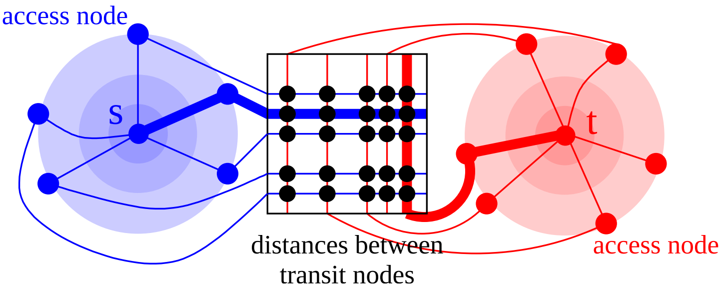
Contraction Hierarchies



Hub-Labeling



Transit-Node Routing



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

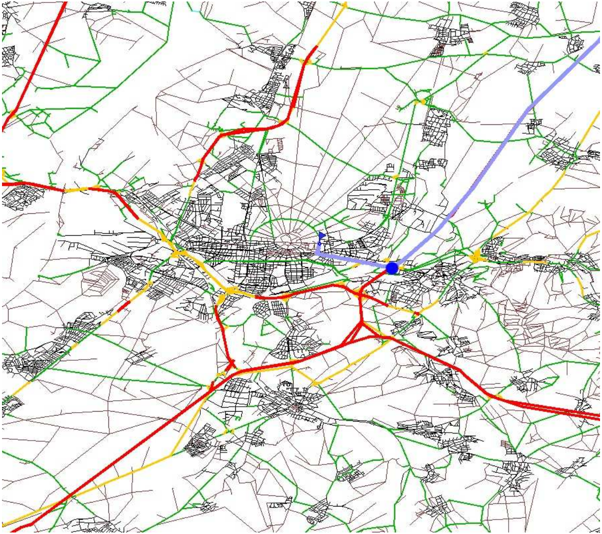
Transit-Node Routing

Beobachtung:

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

Karlsruhe nach…
Brüssel

Transit-Node Routing

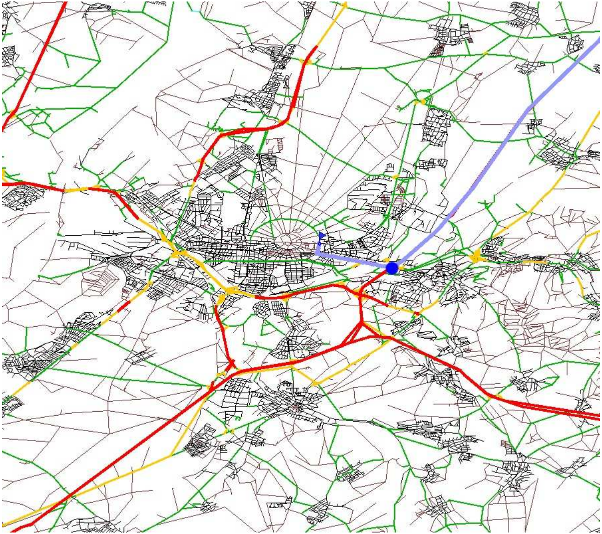


Beobachtung:

- 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

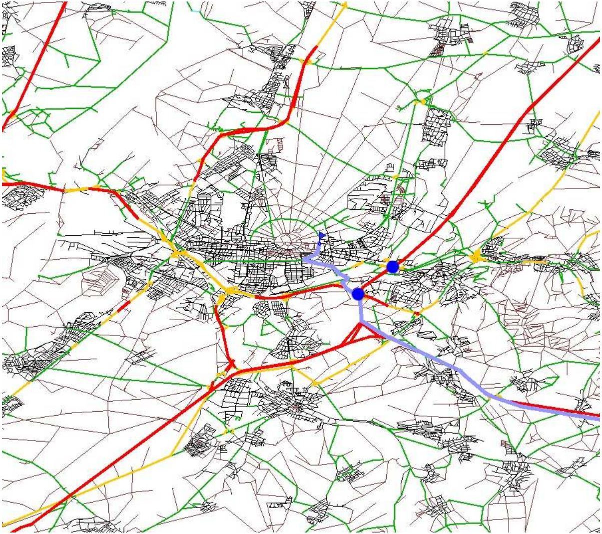


Beobachtung:

- 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

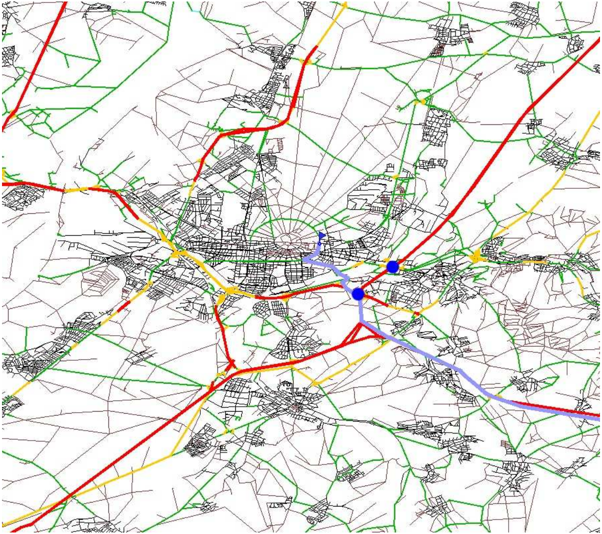


Beobachtung:

- 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

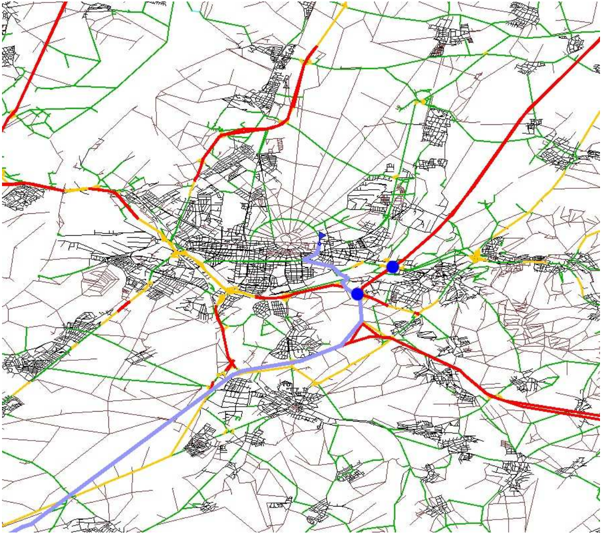


Beobachtung:

- 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

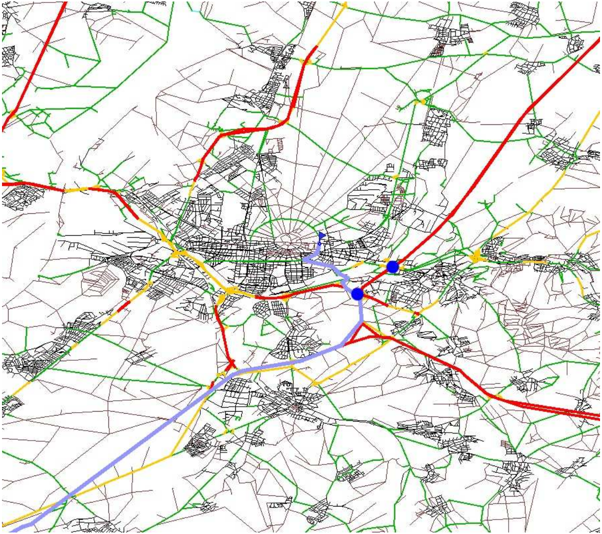


Beobachtung:

- 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

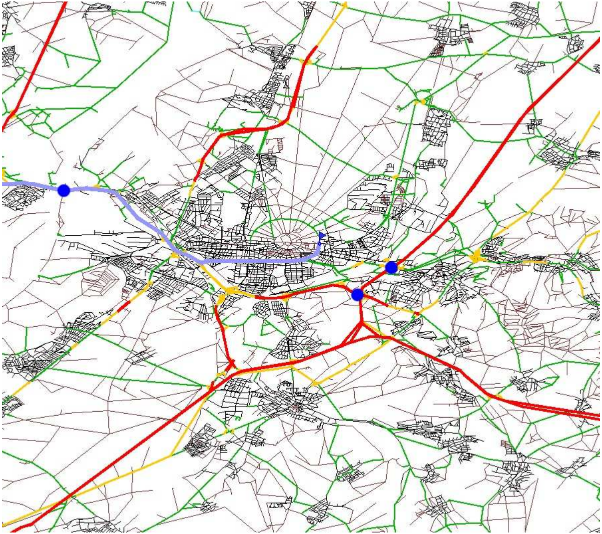


Beobachtung:

- 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



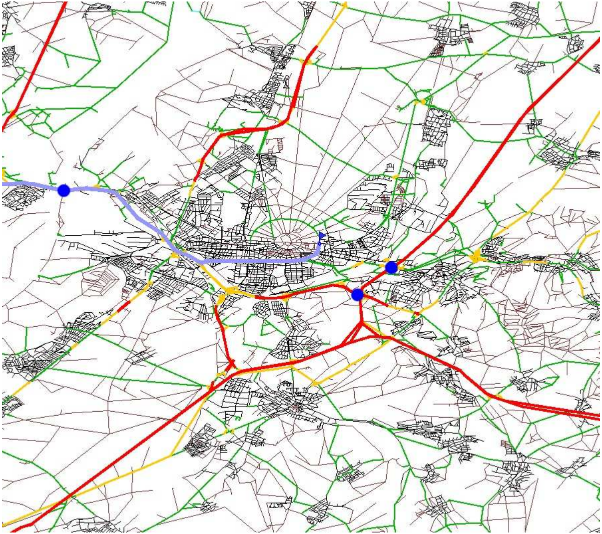
Beobachtung:

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

Karlsruhe nach...
London

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. . .
Brüssel

Transit-Node Routing

Ziel:

- reduziere Anfragen auf Table-Lookups

Transit-Node Routing

Ziel:

- reduziere Anfragen auf Table-Lookups

Idee:

- identifiziere “wichtige” Knoten
- vollständige Distanztabellen zwischen diesen Knoten

Transit-Node Routing

Ziel:

- reduziere Anfragen auf Table-Lookups

Idee:

- identifiziere “wichtige” Knoten
- vollständige Distanztabellen zwischen diesen Knoten

Probleme:

- Speicherverbrauch
- nahe Anfragen?

Transit-Node Routing

Gegeben:

- $L + 1$ Mengen von *transit nodes* $V =: T_0 \supseteq T_1 \supseteq \cdots \supseteq T_L$

Transit-Node Routing

Gegeben:

- $L + 1$ Mengen von *transit nodes* $V =: T_0 \supseteq T_1 \supseteq \cdots \supseteq T_L$

Betrachte für jeden Level ℓ mit $0 \leq \ell \leq L$:

- *access mapping* $\overrightarrow{A}_\ell : V \to 2^{T_\ell}$ mapt Knoten auf *access nodes*

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Transit-Node Routing

Gegeben:

- $L+1$ Mengen von *transit nodes* $V =: T_0 \supseteq T_1 \supseteq \cdots \supseteq T_L$

Betrachte für jeden Level ℓ mit $0 \leq \ell \leq L$:

- *access mapping* $\overrightarrow{A}_\ell : V \to 2^{T_\ell}$ mapt Knoten auf *access nodes*
- *locality filter* $L_\ell : V \times V \to \{\texttt{true}, \texttt{false}\}$ gibt an, ob $d(s,t)$ *nicht* mit Transit-Knoten der Level $\geq \ell$ berechnet werden kann

Transit-Node Routing

Gegeben:

- $L + 1$ Mengen von *transit nodes* $V =: T_0 \supseteq T_1 \supseteq \cdots \supseteq T_L$

Betrachte für jeden Level ℓ mit $0 \leq \ell \leq L$:

- *access mapping* $\overrightarrow{A}_\ell : V \to 2^{T_\ell}$ mapt Knoten auf *access nodes*
- *locality filter* $L_\ell : V \times V \to \{\texttt{true}, \texttt{false}\}$ gibt an, ob $d(s,t)$ *nicht* mit Transit-Knoten der Level $\geq \ell$ berechnet werden kann
- *Distanztabelle* $D_\ell : T_\ell \times T_\ell \to \mathbb{R}_0^+ \cup \{\infty\}$ Abstände zwischen Transit-Knoten auf Level ℓ bis auf solche, die mit höheren Leveln berechnet werden können

Transit-Node Routing

Gegeben:

- $L+1$ Mengen von *transit nodes* $V =: T_0 \supseteq T_1 \supseteq \cdots \supseteq T_L$

Betrachte für jeden Level ℓ mit $0 \leq \ell \leq L$:

- *access mapping* $\overrightarrow{A}_\ell : V \to 2^{T_\ell}$ mapt Knoten auf *access nodes*
- *locality filter* $L_\ell : V \times V \to \{\texttt{true}, \texttt{false}\}$ gibt an, ob $d(s,t)$ *nicht* mit Transit-Knoten der Level $\geq \ell$ berechnet werden kann
- *Distanztabelle* $D_\ell : T_\ell \times T_\ell \to \mathbb{R}_0^+ \cup \{\infty\}$ Abstände zwischen Transit-Knoten auf Level ℓ bis auf solche, die mit höheren Leveln berechnet werden können
- $d_\ell : V \times V \to \mathbb{R}_0^+ \cup \{\infty\}$ gibt Abstand zwischen zwei Knoten auf einem Level an, also Abstand zu access nodes und zwischen Transit-Knoten auf Level ℓ

Transit-Node Routing

Gegeben:

- $L + 1$ Mengen von *transit nodes* $V =: T_0 \supseteq T_1 \supseteq \cdots \supseteq T_L$

Betrachte für jeden Level ℓ mit $0 \leq \ell \leq L$:

- *access mapping* $\overrightarrow{A}_\ell : V \to 2^{T_\ell}$ mapt Knoten auf *access nodes*
- *locality filter* $L_\ell : V \times V \to \{\texttt{true}, \texttt{false}\}$ gibt an, ob $d(s,t)$ *nicht* mit Transit-Knoten der Level $\geq \ell$ berechnet werden kann
- *Distanztabelle* $D_\ell : T_\ell \times T_\ell \to \mathbb{R}_0^+ \cup \{\infty\}$ Abstände zwischen Transit-Knoten auf Level ℓ bis auf solche, die mit höheren Leveln berechnet werden können
- $d_\ell : V \times V \to \mathbb{R}_0^+ \cup \{\infty\}$ gibt Abstand zwischen zwei Knoten auf einem Level an, also Abstand zu access nodes und zwischen Transit-Knoten auf Level ℓ
- $d_{\geq \ell} : V \times V \to \mathbb{R}_0^+ \cup \{\infty\}$ gibt Abstand an, der mit Hilfe aller Level $\geq \ell$ berechnet werden kann

Eigenschaften

Bemerkungen:

- In der Distanztabelle werden nur relevante Informationen abgelegt

$$D_\ell(s,t) := \begin{cases} d_\ell(s,t) & \text{wenn} \quad d_\ell(s,t) < d_{\geq \ell+1}(s,t) \\ \infty & \text{sonst} \end{cases}$$

- d_ℓ gibt die Distanz in dem entsprechenden Level an

$$d_\ell(s,t) := \min_{\substack{u \in \overrightarrow{A}(s) \\ v \in \overleftarrow{A}(t)}} \{d(s,u) + D_\ell(u,v) + d(v,t)\}$$

- $d_{\geq \ell}$ gibt die Distanz aller Level $\geq \ell$ an

$$d_{\geq \ell}(s,t) := \min_{k \geq \ell} d_k(s,t)$$

TNR Anfragen

TNR-Query(s,t)

```
d' = ∞
for ℓ = L down to 0 do
    d' := min{d', d_ℓ(s, t)}
    if ¬L_ℓ(s, t) then break
return d'
```

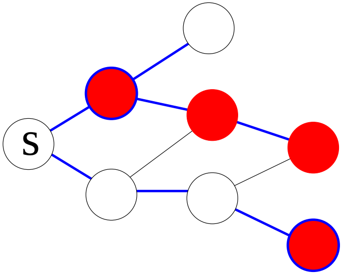
Bemerkungen:

- speichere in D_ℓ nur nicht-∞-Einträge (Hash-Tabelle)
- benutze CH (oder bel. andere Technik) für Level 0 Anfragen

Access-Nodes berechnen

Vorgehen:

- lokale Suche, bis Transit-Knoten des Levels alle Zweige abdecken



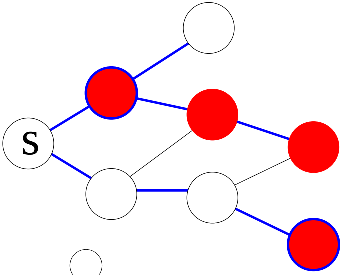
Access-Nodes berechnen

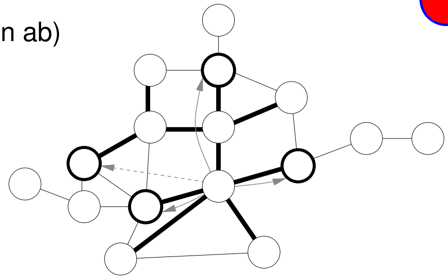
Vorgehen:

- lokale Suche, bis Transit-Knoten des Levels alle Zweige abdecken

Beschleunigung:

- *prune* an Transit-Knoten (breche Zweig bei Transit-Knoten ab)
- dünne dann später mit Hilfe der Distanz-Tabelle wieder aus





Institut für Theoretische Informatik
Lehrstuhl Algorithmik

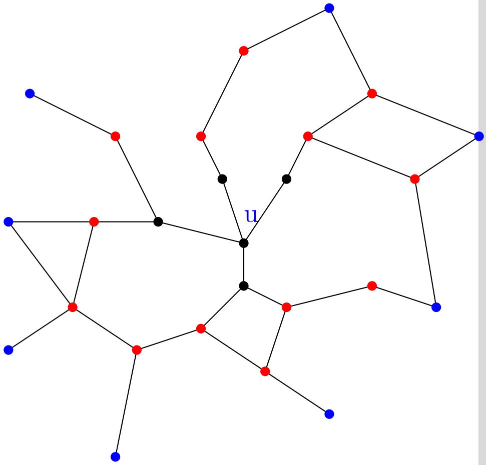
Runterpropagieren von ANs:

Idee:

- Access-Nodes können propagiert werden

$$\overrightarrow{A}_{\ell}(u) := \bigcup_{v \in \overrightarrow{A}_{\ell-1}(u)} \overrightarrow{A}_{\ell}(v)$$

- kann auf mehrere Level verallgemeinert werden
- können mit Distanztabellen wieder ausgedünnt werden



Distanztabellen

Vorgehen:

- benutze Many-to-Many Ansatz (später mehr Details dazu!)

Problem:

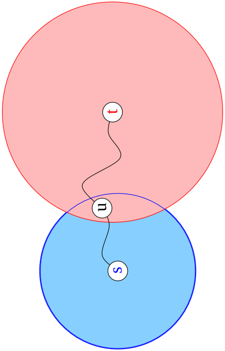
- wir speichern $d_\ell(u, v)$ in $D_\ell(u, v)$ nur, wenn $d_\ell(u, v) < d_{\ell+1}(u, v)$ gilt
- Anzahl Transit Knoten auf niederigem Level hoch $\rightsquigarrow$ ineffizient

Lösungen:

- gehe top-down vor, dann ist $d_{\ell+1}(u, v)$ verfügbar
- breche Suchen an hochleveligen Transit-Knoten ab

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Locality Filter



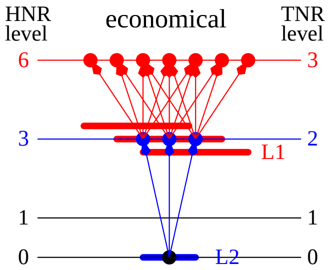
Vorgehen:

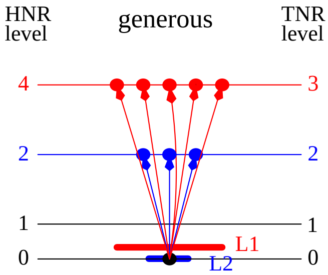
- speicher für jeden Knoten den (euklidischen) Abstand zum am weitesten entfernten lokalen Knoten ab, also $\overrightarrow{K}(u)$ und $\overleftarrow{K}(u)$
- für jeden Level, also $\overrightarrow{K}_\ell(u)$, $\overleftarrow{K}_\ell(u)$
- geht während Vorberechnung
- locality filter schlägt zu, wenn

$$||s - t|| < \overrightarrow{K}_k(s) + \overleftarrow{K}_k(t) \quad \forall k \leq \ell$$

- dadurch manchmal falscher Alarm

2 Varianten





economical:

- speicher keine transit knoten für level 1, benutze CH
- oberster locality filter auf level der transit Knoten

generous:

- mehr transit nodes
- locality filter auf unterstem Level
- höherer Platzverbrauch, schneller

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Pfadentpackung

- analog zu Contraction Hierarchies
- interpretiere jeden Eintrag als Shortcut
- speichere für jeden Eintrag den Pfad, den er repräsentiert
- rekursiv

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Vergleich zu CH

Gemeinsamkeiten:

- Level-Einteilung
- Pfadentpackung ähnlich
- TNR basiert auf CH

Unterschiede:

- weniger level
- Suche durch Table-Lookups ersetzt
- Locality Filter nötig
- Shortcuts $\mapsto$ Tabellen-Einträgen

Experimente

Eingaben:

- Straßennetzwerke
  - Europa: 18 Mio. Knoten, 42 Mio. Kanten
  - USA: 22 Mio. Knoten, 56 Mio. Kanten

Evaluation:

- Vorberechnung in Minuten und zusätzliche Bytes pro Knoten
- durchschnittlicher Suchraum (#abgearbeitete Knoten) und Suchzeiten (in *ms*) von 1 000 000 Zufallsanfragen

TNR: Vorberechnung

variant	level 3 $\lvert T_3\rvert$	$\lvert A_3\rvert$	level 2 $\lvert T_2\rvert$	$\lvert D_2\rvert$	$\lvert A_2\rvert$	overhead [B/node]	time [h]
			Europe				
eco	9 355	11.4	151 450	0.15%	5.3	99	0:25
gen	9 458	11.3	293 209	0.14%	4.4	226	1:15
			USA (Tiger)				
eco	6 449	6.8	218 153	0.20%	5.2	121	0:38
gen	10 261	6.1	449 945	0.08%	4.5	257	1:25

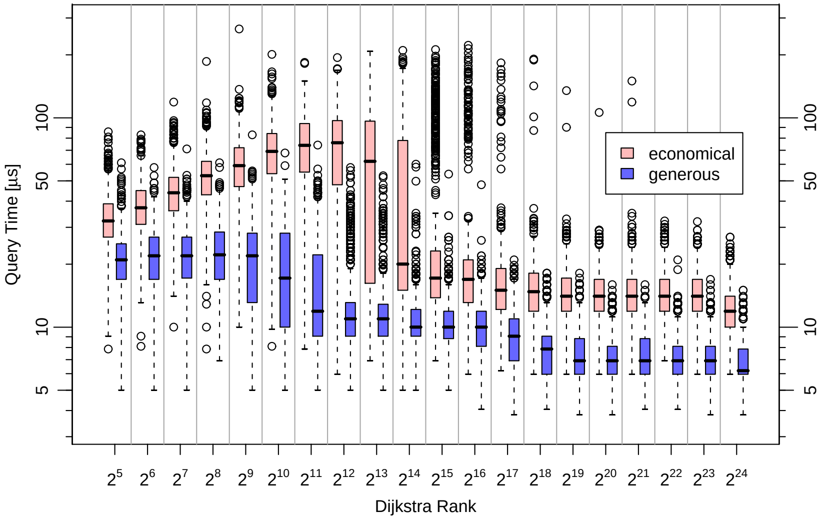
TNR: Anfragen

metric	variant	level 3 [%] wrong	level 3 [%] cont'd	level 2 [%] (level 1 [%]) wrong	level 2 [%] (level 1 [%]) cont'd	query time
Europe						
time	eco	0.57	3.36	0.0051	0.1364	11.0μs
time	gen	0.25	1.55	0.0016 (0.00019)	0.0180 (0.0180)	4.3μs
USA (Tiger)						
time	eco	0.37	2.44	0.0045	0.1130	9.5μs
time	gen	0.10	0.87	0.0010 (0.00009)	0.0124 (0.0124)	3.3μs

Übersicht: bisherige Techniken

	Vorberechnung		Anfrage		
	Zeit [h:m]	Platz [byte/n]	Such raum	Zeit [ms]	Beschl.
Dijkstra	0:00	0	9 114 385	5 591.6	1
ALT-16	1:25	128	74 669	53.6	104
Arc-Flags (128)	17:08	10	2 764	0.8	6 988
MLD-3	$<$ 0:01	1.7	6 074	0.91	6 143
CH	0:02	-3.0	284	0.09	62 128
eco TNR	0:25	120	N/A	0.011	ca. 500 000
gen TNR	1:15	247	N/A	0.0043	ca. 1 300 000
HL	6:12	967.1	N/A	0.0002	ca. 22 000 000

Dijkstra Rank



Zusammenfassung

Transit-Node Routing

- ersetzt Suche (fast) komplett durch Table-Lookups
- 3 Zutaten:
  - Distanztabelle
  - Access-Nodes
  - Locality-Filter
- Suchzeit von unter 5 μs

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Ende

Karlsruher Institut für Technologie

Literatur (Transit-Node Routing):

- Peter Sanders and Dominik Schultes:
  Robust, Almost Constant Time Shortest-Path Queries in Road Networks
  In: *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, 2009
- Dominik Schultes:
  Route Planning in Road Networks
  Ph.D. Thesis, Universität Karlsruhe (TH), 2009.

Nächste Termine

Mittwoch, 30.5.2012 (Thomas)

Montag, 4.6.2012 (Daniel)

Mittwoch, 6.6.2012 (Daniel)

Institut für Theoretische Informatik
Lehrstuhl Algorithmik