Karlsruher Institut für Technologie

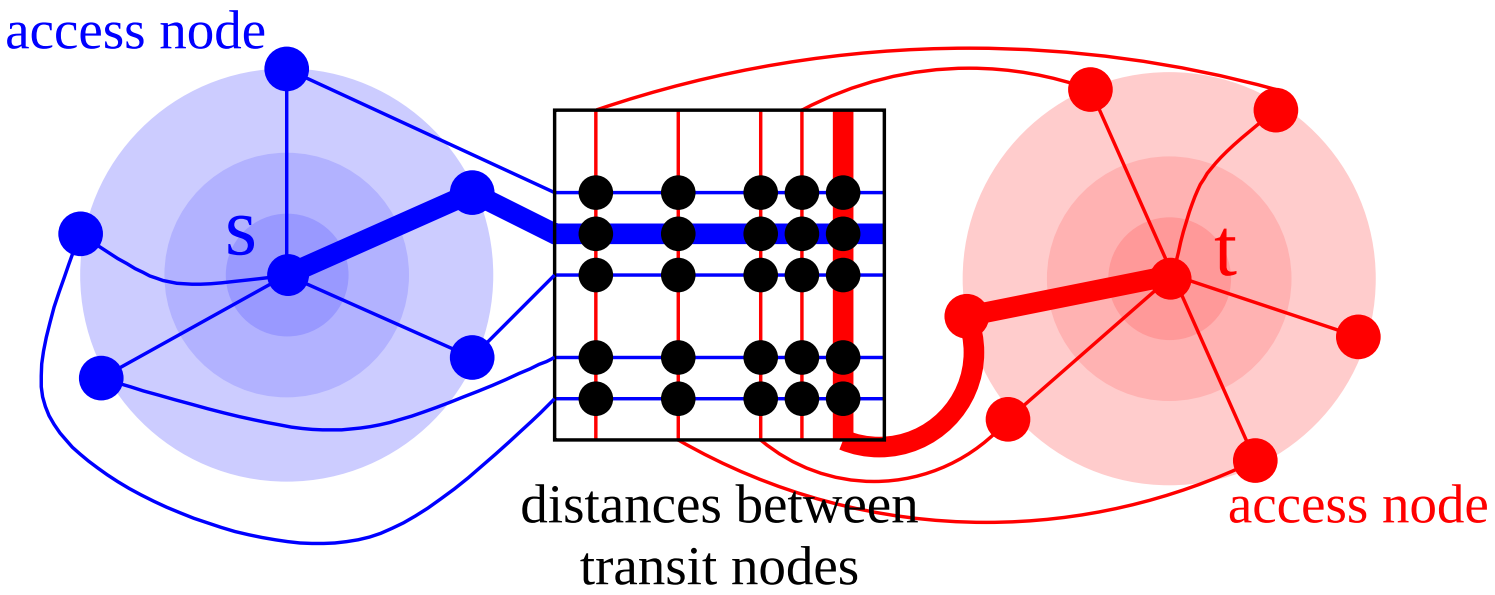
Algorithmen für Routenplanung

5. Termin, Sommersemester 2010

Reinhard Bauer | 5. Mai 2011

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

Transit-Node Routing



Transit-Node Routing

Beobachtung:

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

Karlsruhe nach. . .
Kopenhagen

Transit-Node Routing

Beobachtung:

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

Karlsruhe nach. . .
Berlin

Transit-Node Routing

Beobachtung:

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

Karlsruhe nach. . .
Wien

Transit-Node Routing

Beobachtung:

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

Karlsruhe nach. . .
München

Transit-Node Routing

Beobachtung:

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

Karlsruhe nach. . .
Rom

Transit-Node Routing

Beobachtung:

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

Karlsruhe nach. . .
Paris

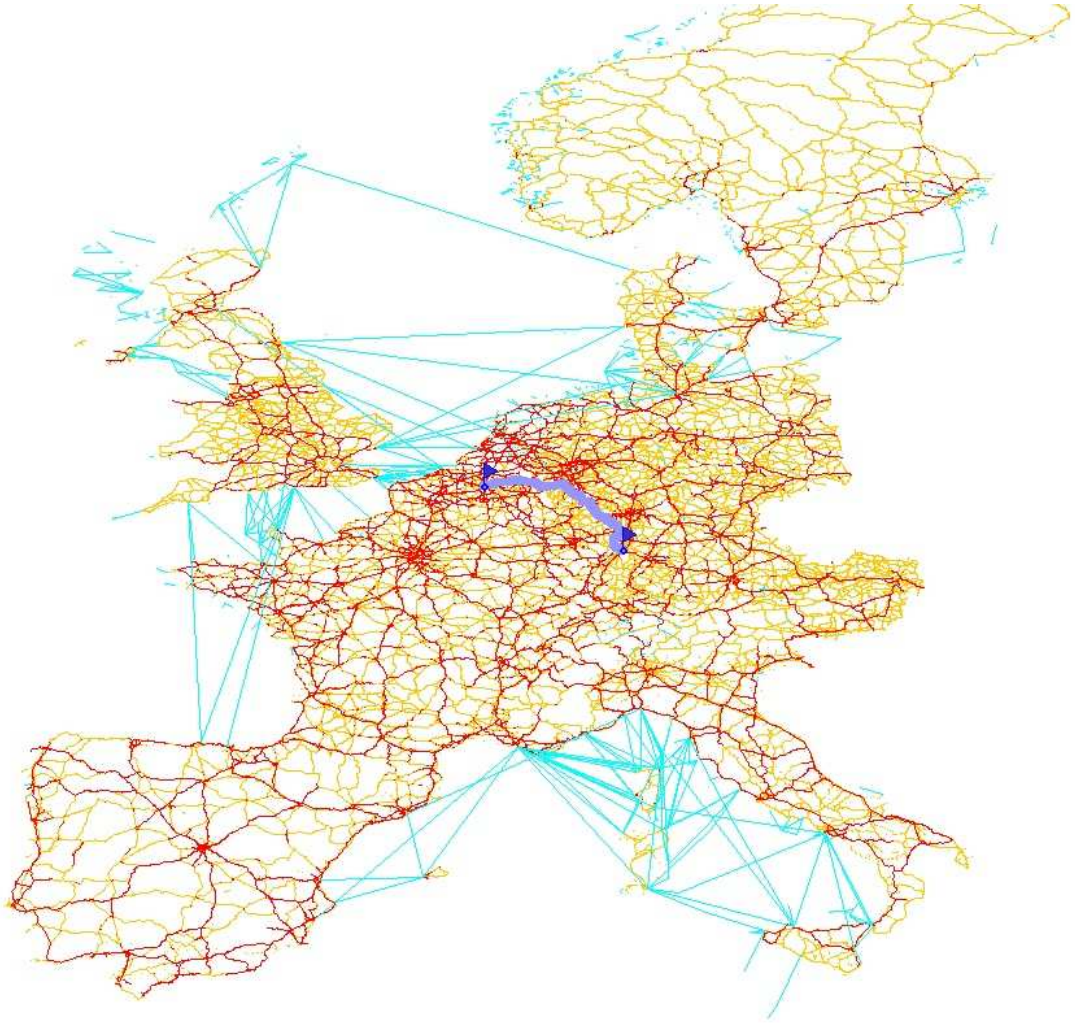
Transit-Node Routing

Beobachtung:

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

Karlsruhe nach. . .
London

Transit-Node Routing

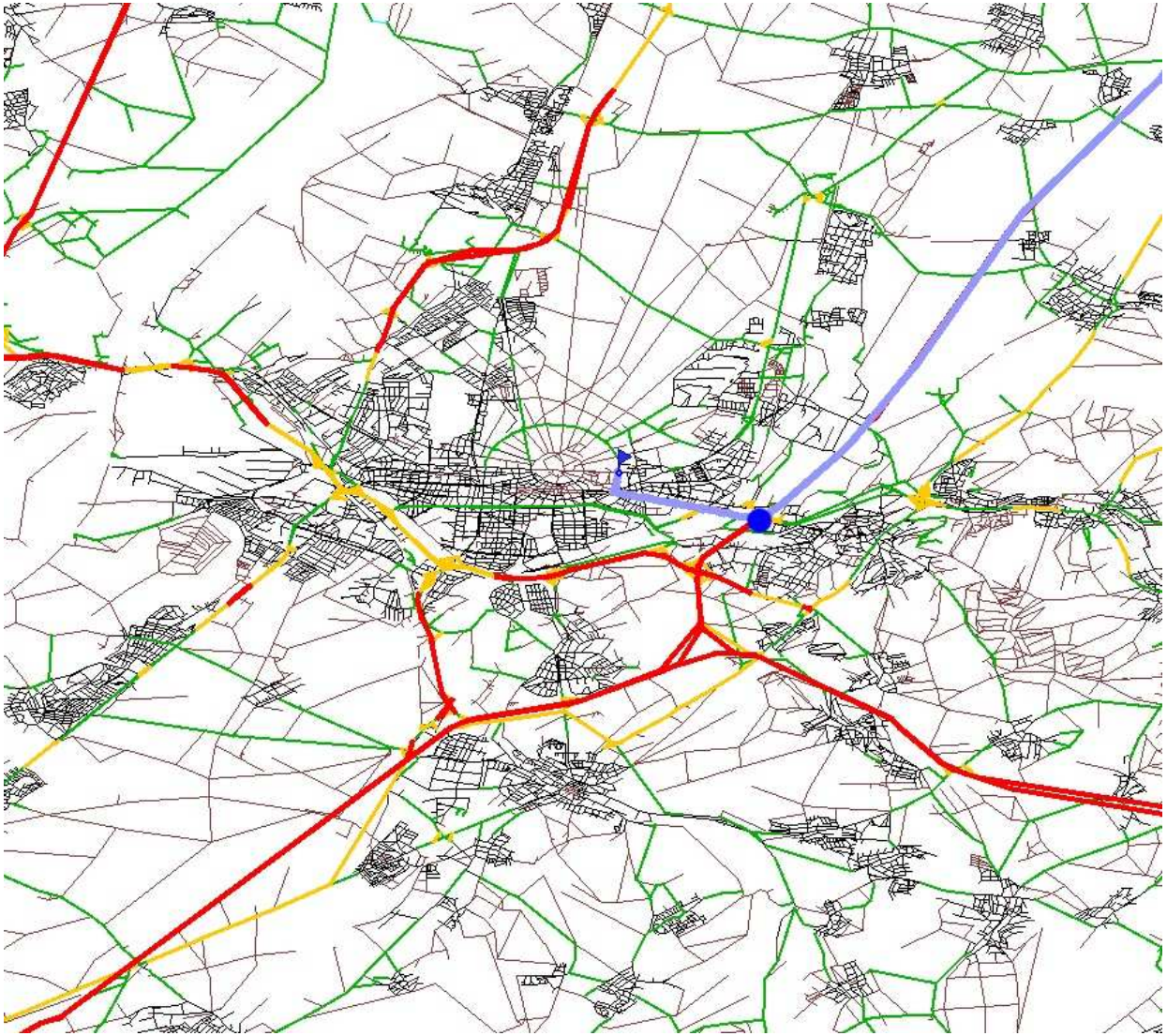


Beobachtung:

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

Karlsruhe nach. . .
Brüssel

Transit-Node Routing

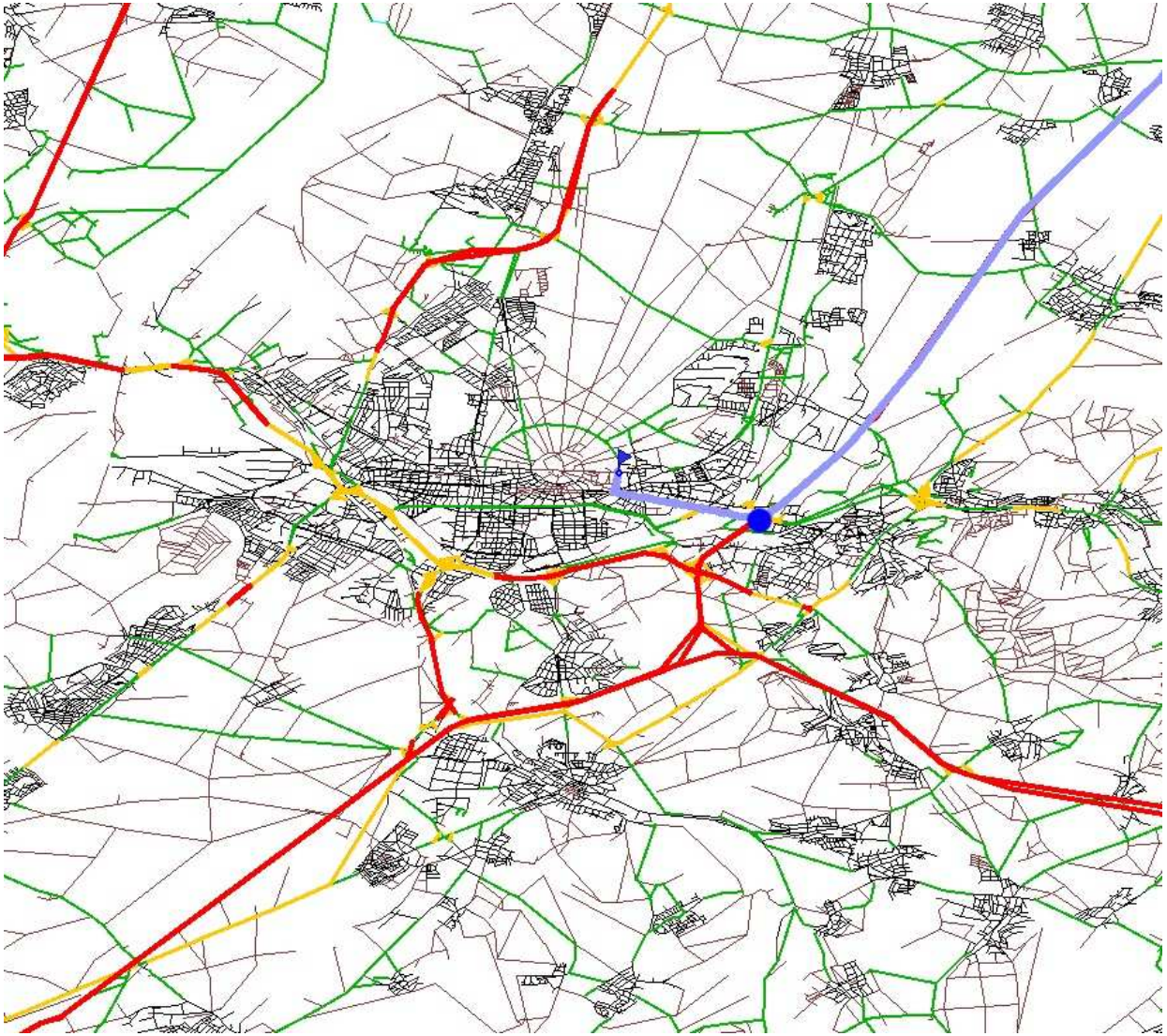


Beobachtung:

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

Karlsruhe nach. . .
Kopenhagen

Transit-Node Routing

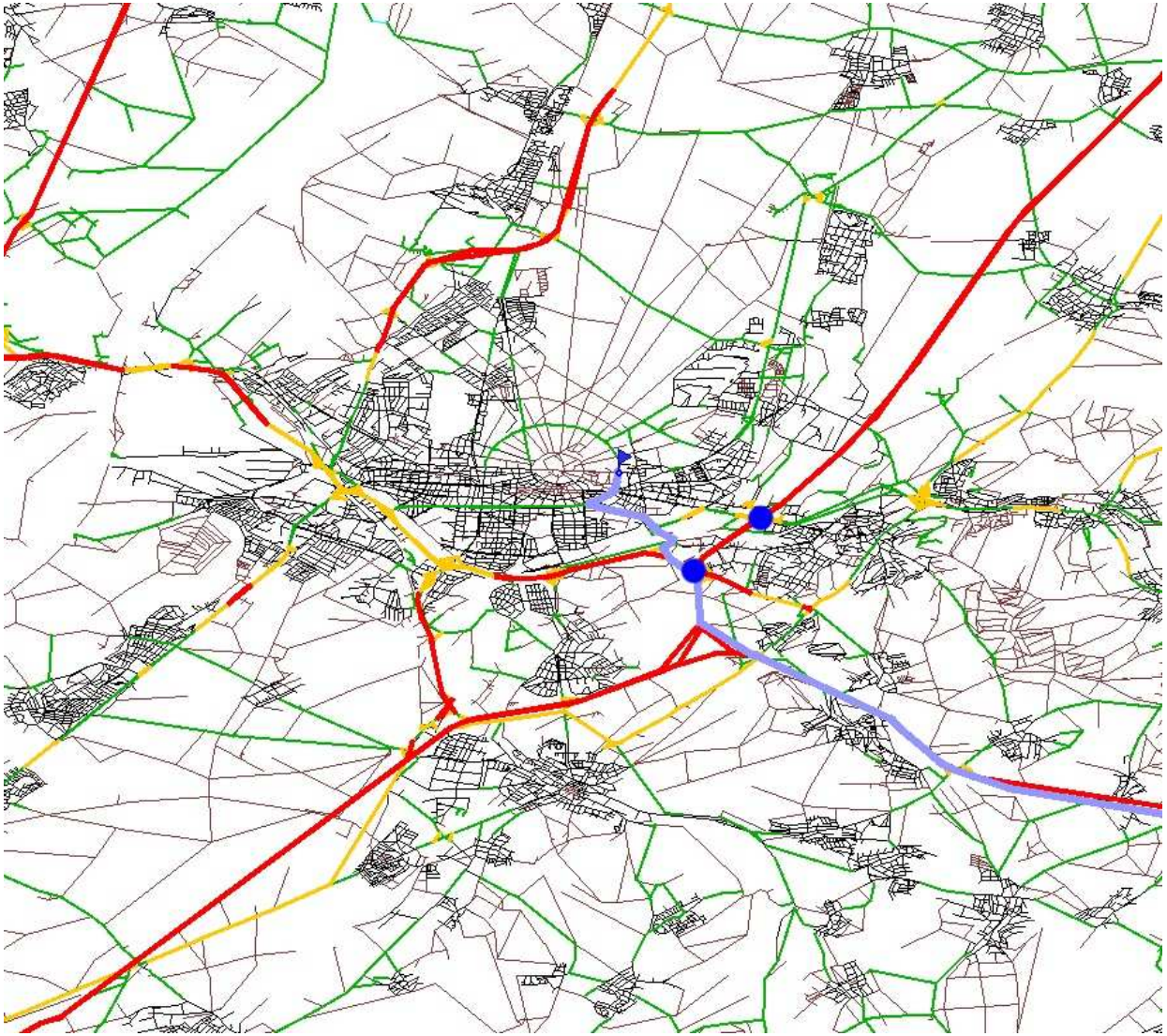


Beobachtung:

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

Karlsruhe nach. . .
Berlin

Transit-Node Routing

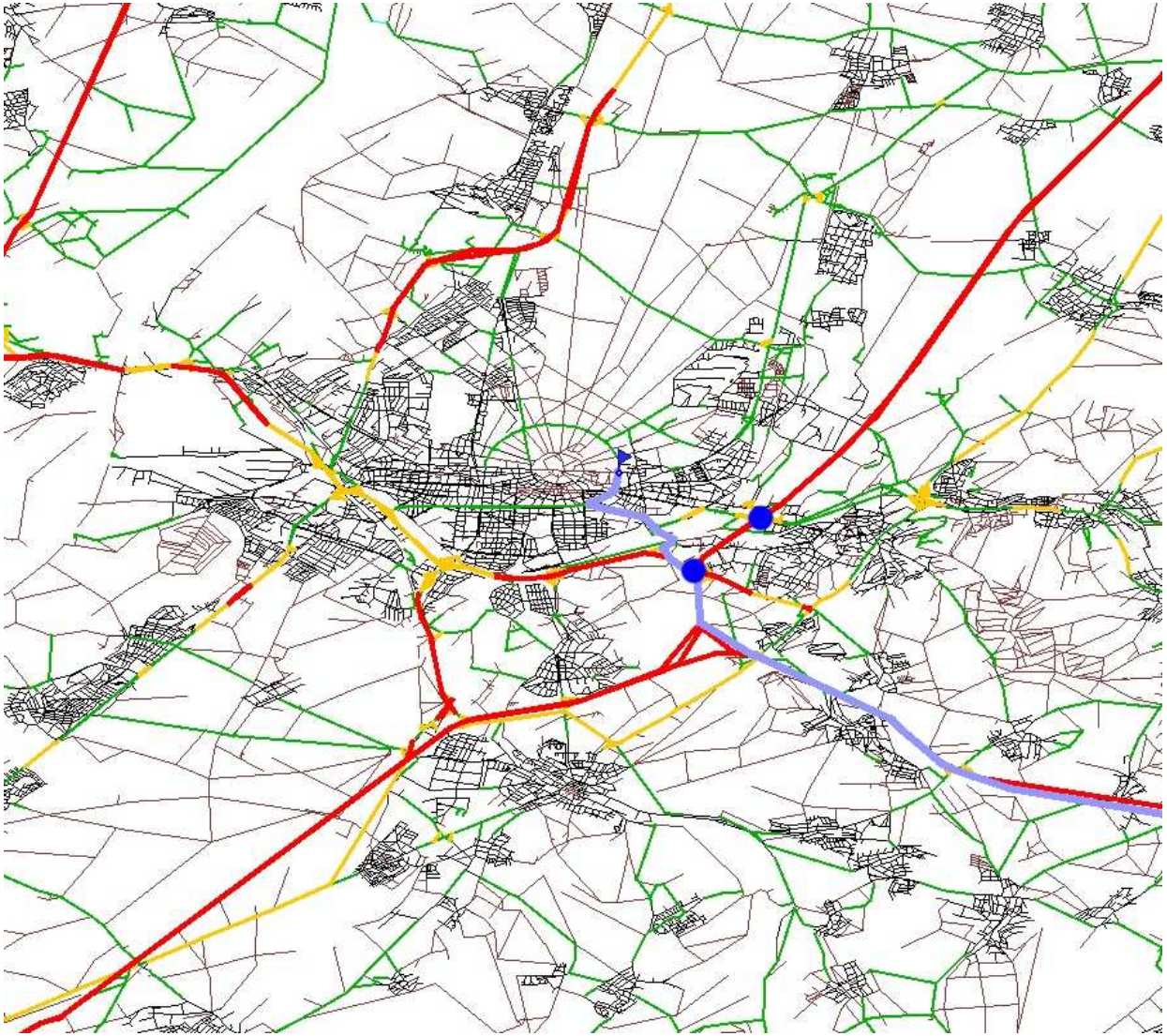


Beobachtung:

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

Karlsruhe nach. . .
Wien

Transit-Node Routing

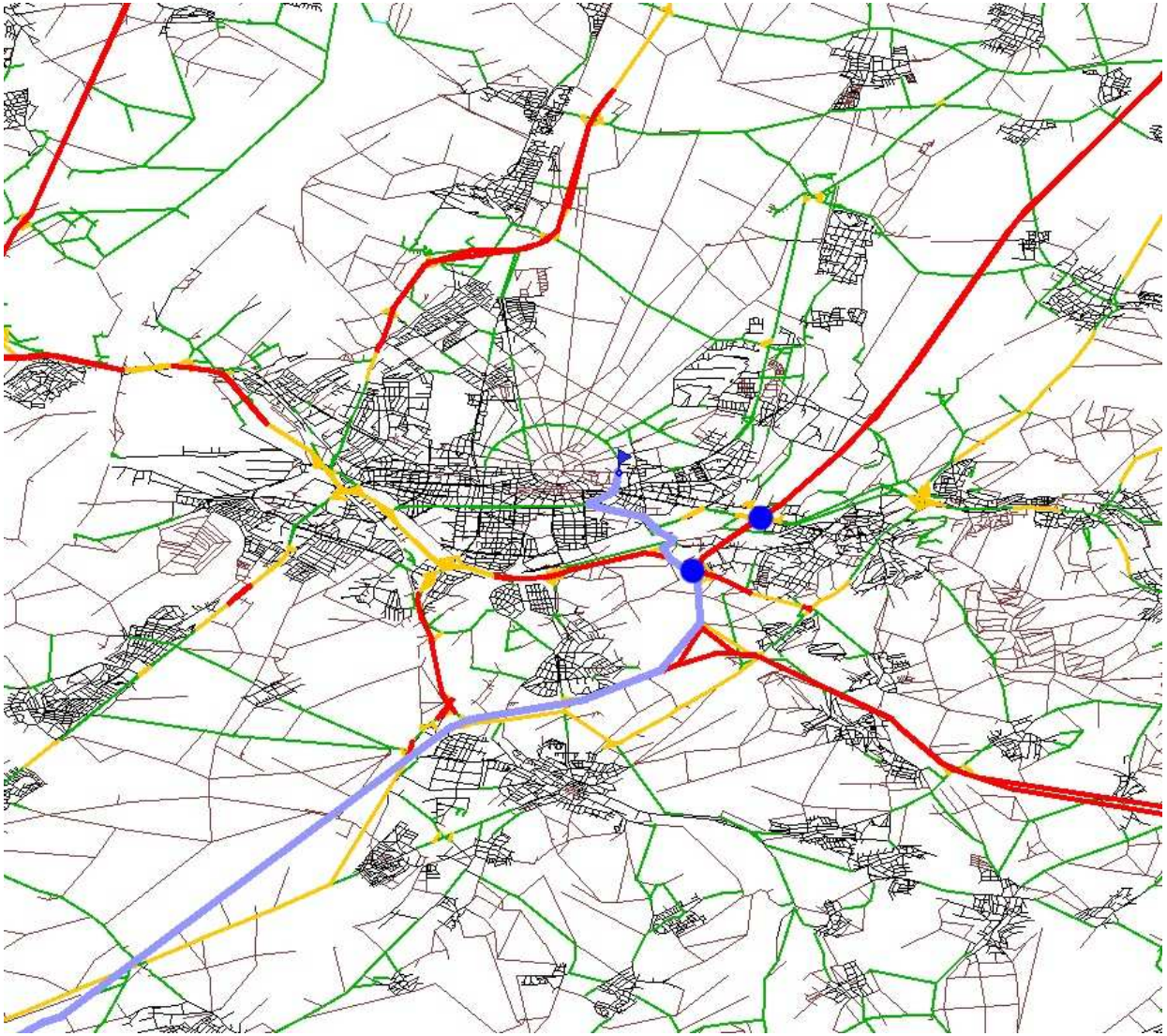


Beobachtung:

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

Karlsruhe nach. . .
München

Transit-Node Routing

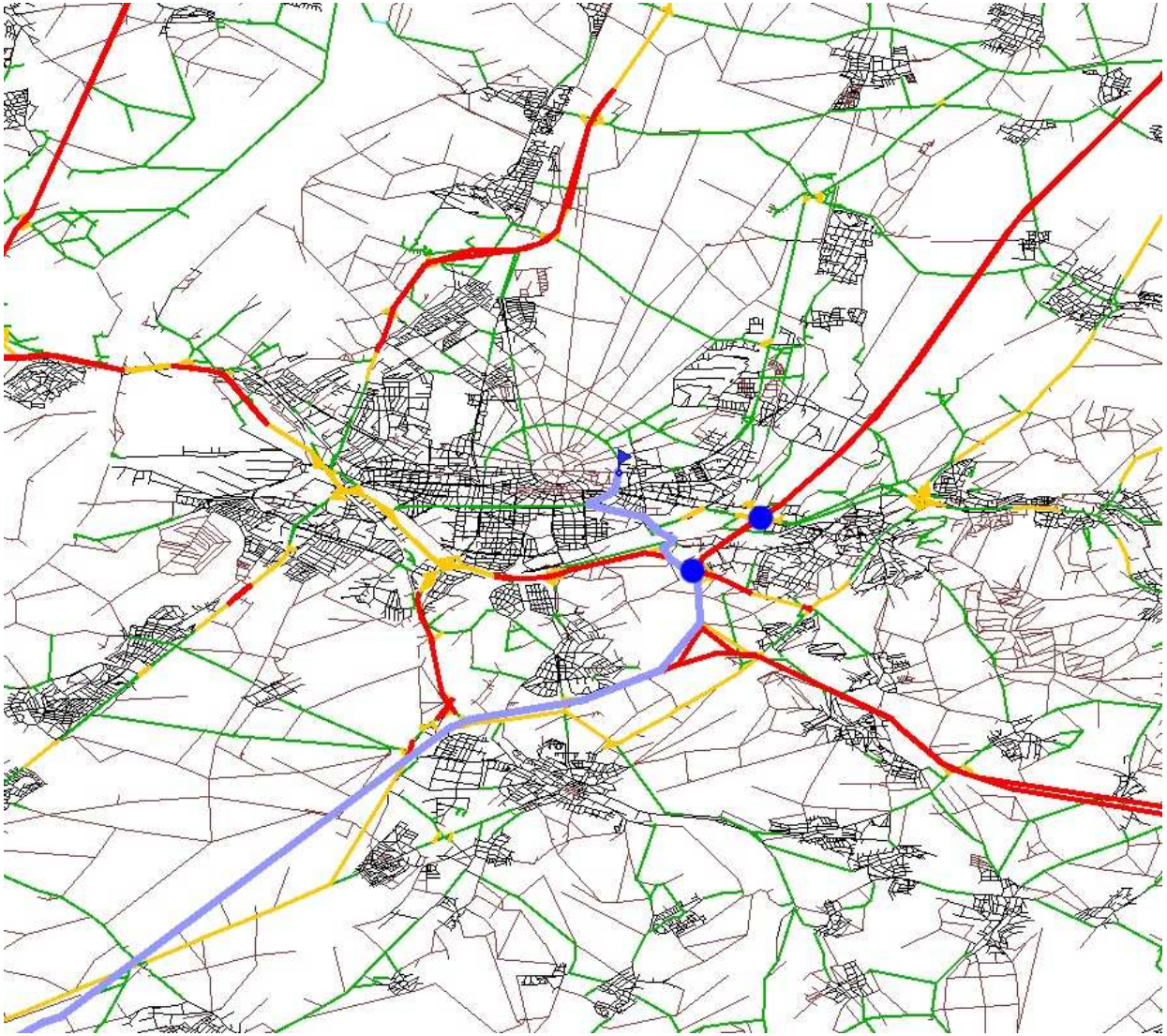


Beobachtung:

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

Karlsruhe nach. . .
Rom

Transit-Node Routing

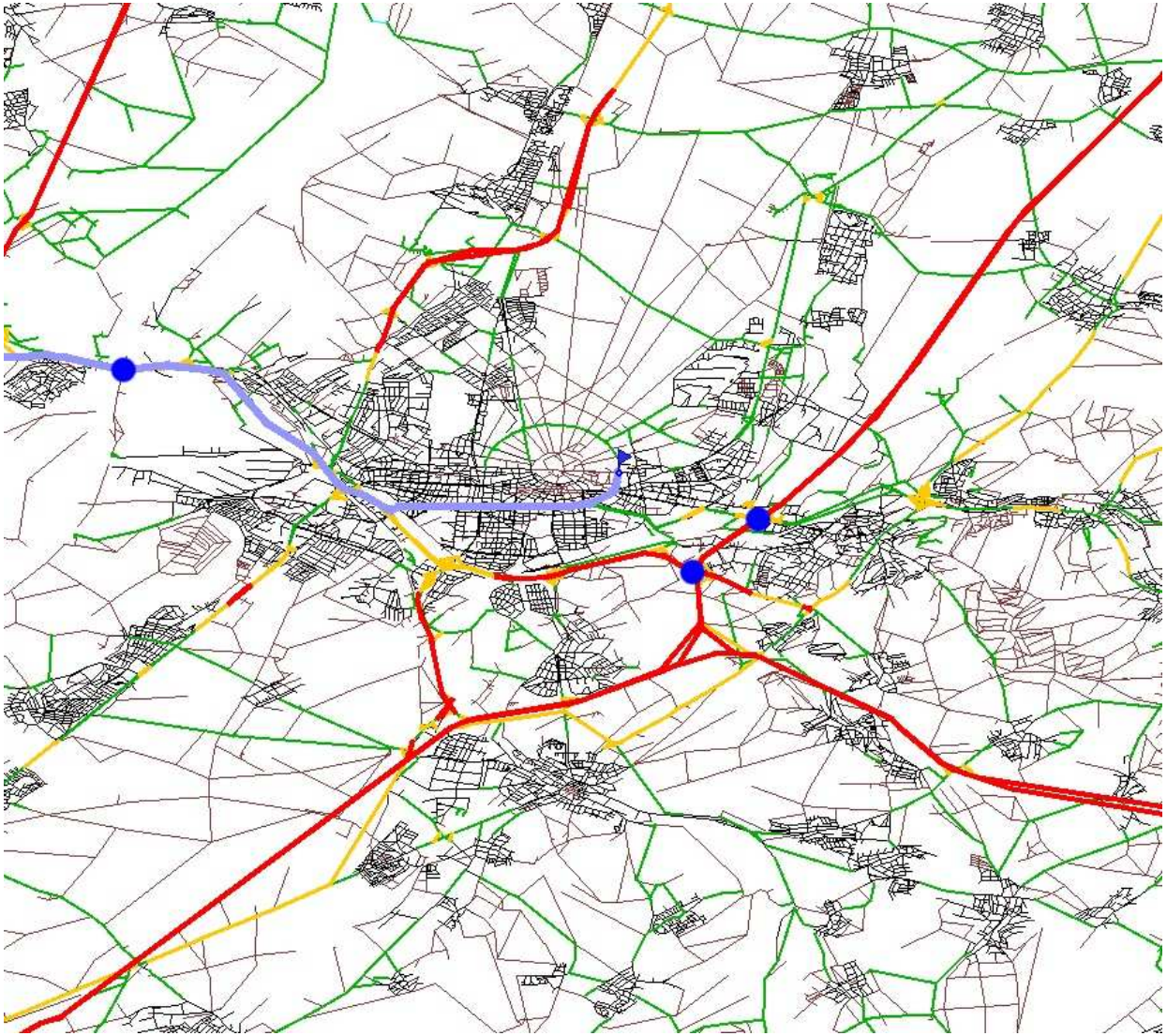


Beobachtung:

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

Karlsruhe nach. . .
Paris

Transit-Node Routing

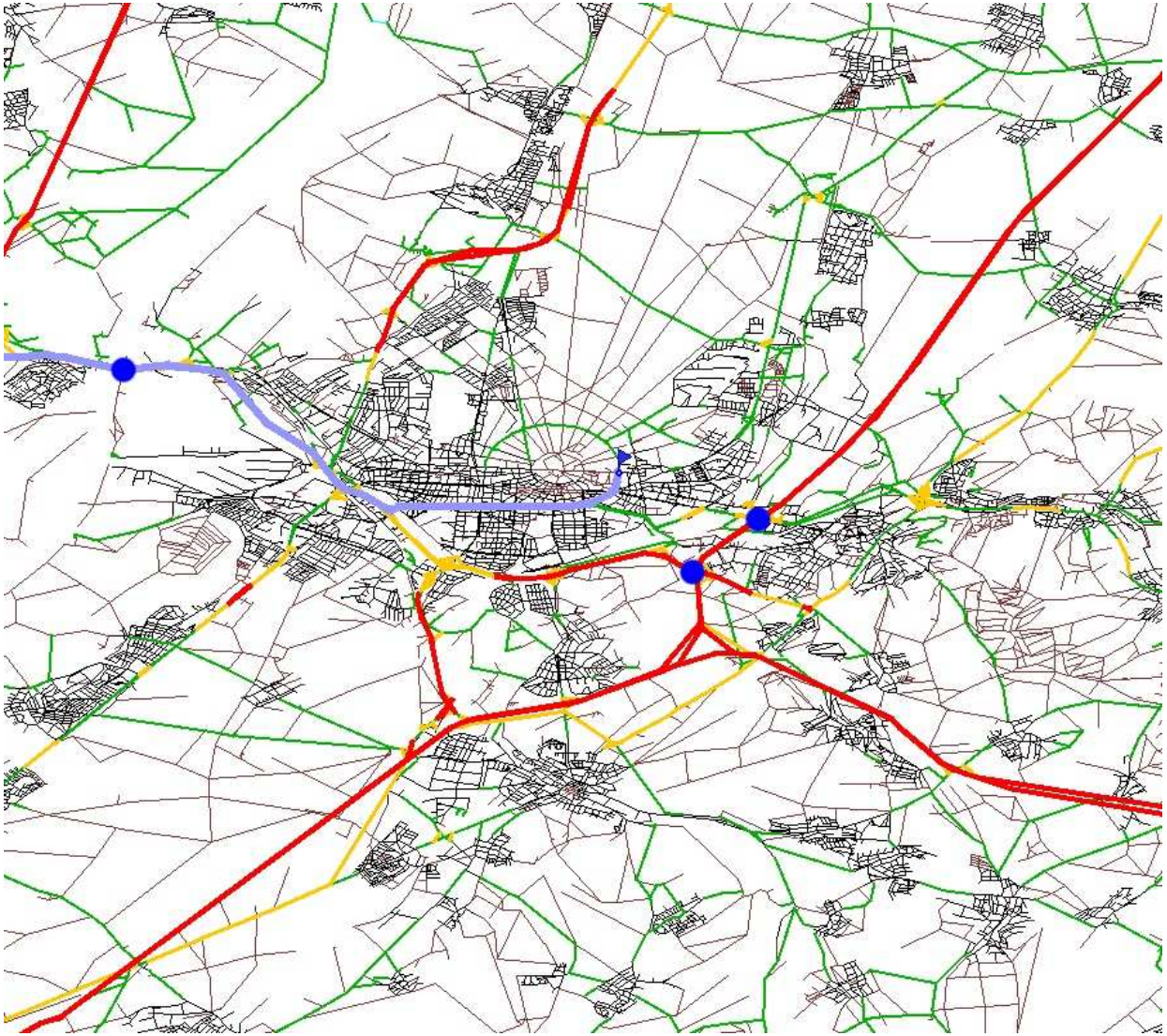


Beobachtung:

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

Karlsruhe nach. . .
London

Transit-Node Routing



Beobachtung:

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

Karlsruhe nach. . .
Brüssel

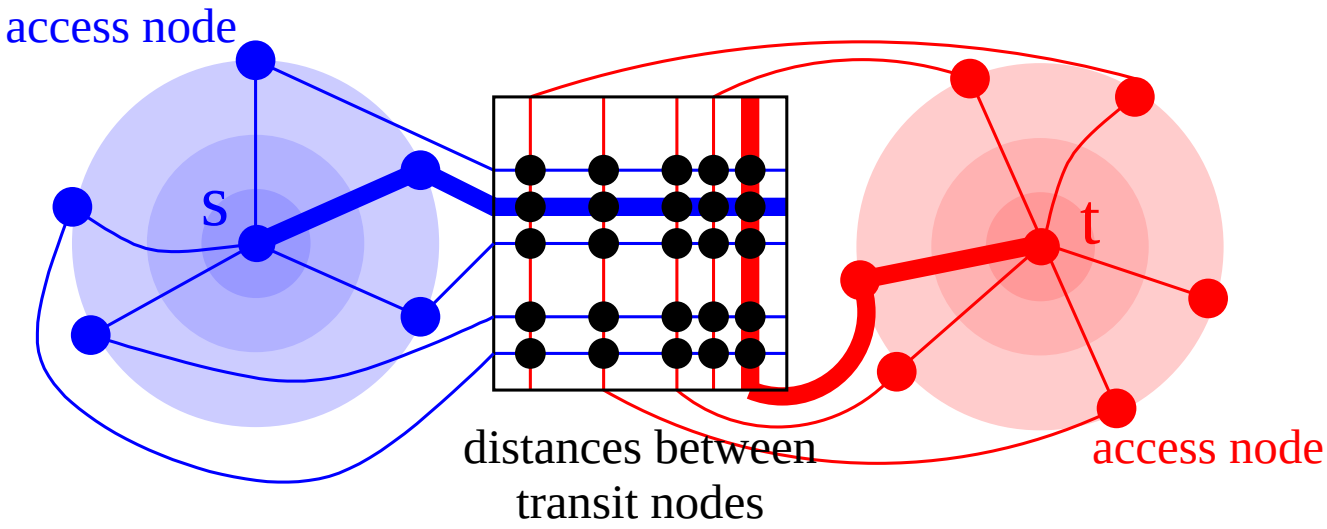
Transit-Node Routing

Idee:

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

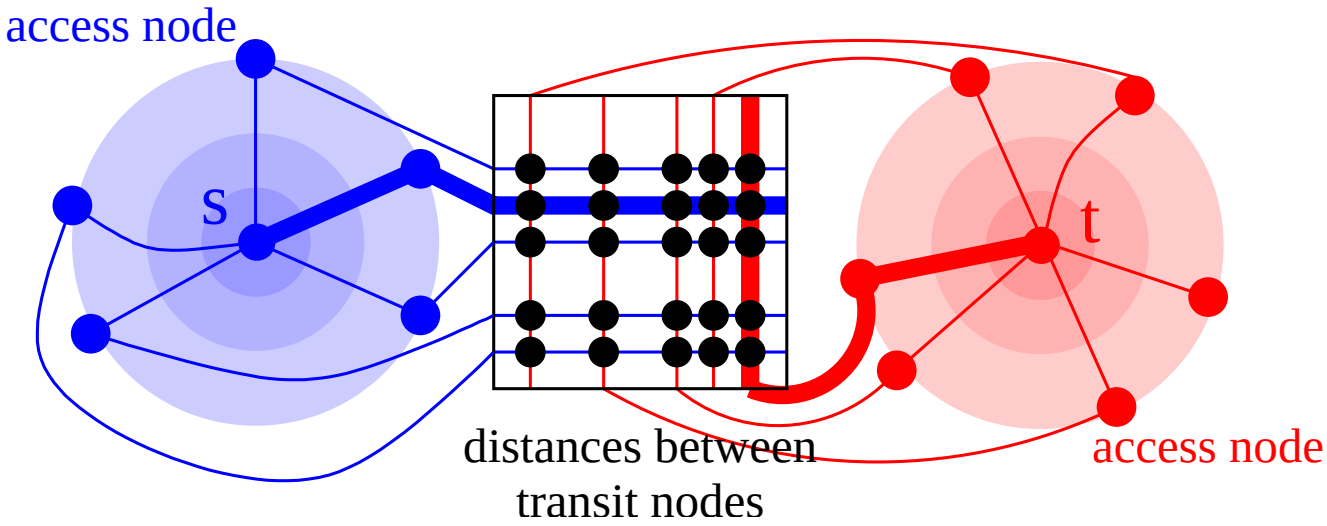
Probleme:

- Speicherverbrauch
- nahe Anfragen



Notation

- Einfachheit: Ungerichtete Graphen $G = (G, E, \text{len})$
- Sei $T \subseteq V$ eine Menge von **Transitknoten**
- Sei $A : V \to 2^T$ eine Zuordnung von Knoten zu ihren **Zugangsknoten**
- Sei $L : V \times V \to \{true, false\}$ ein **Lokalitätsfilter**

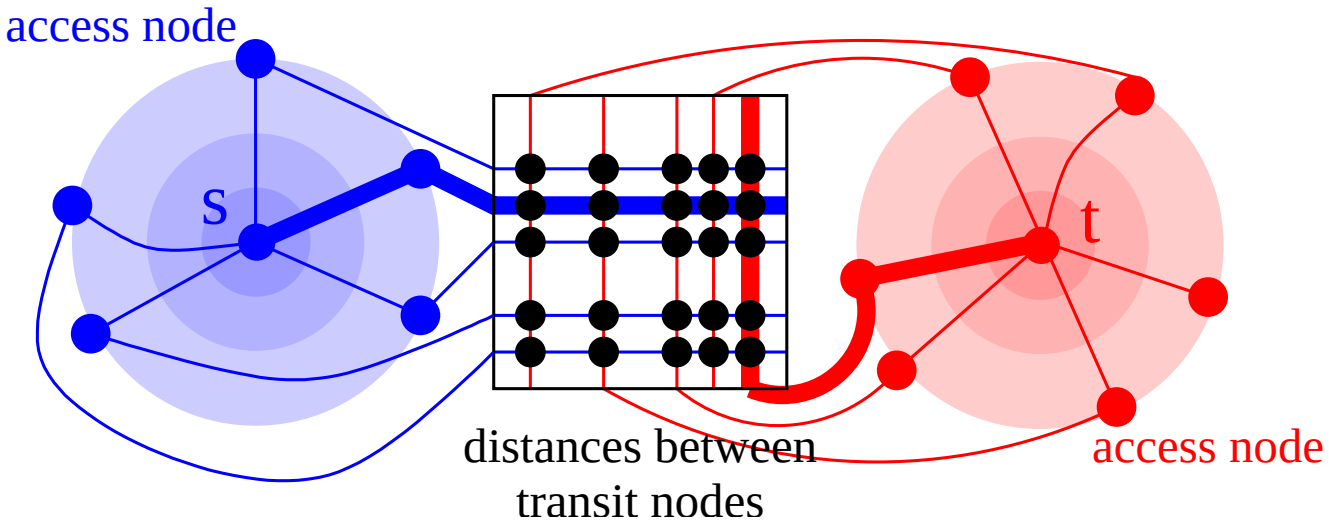


Idee

- Nichtlokale Anfragen müssen über Zugangsknoten führen:
- Formalisierung:

$$\neg L(s,t) \Rightarrow d(s,t) = \min\{d(s,u)+d(u,v)+d(v,t) : u \in A(s), v \in A(t)\}$$

- Berechne Distanzen zwischen allen Transitknoten und zu allen Zugangsknoten vor
- Nichtlokale Anfrage kann dann direkt mit obiger Formel beantwortet werden

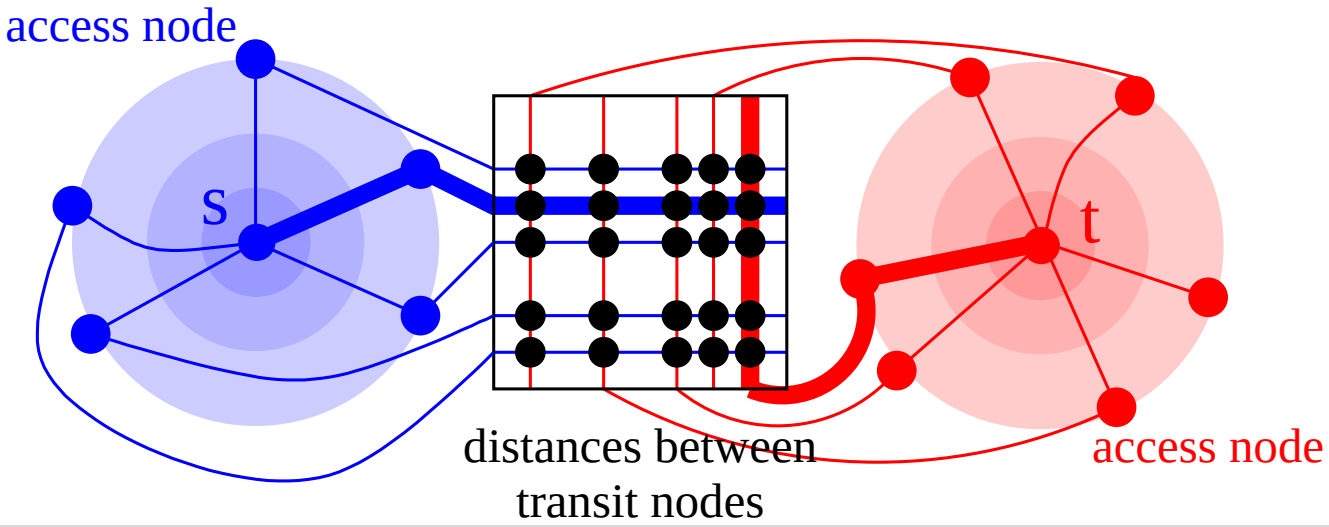


Idee

- Nichtlokale Anfragen müssen über Zugangsknoten führen:
- Formalisierung:

$$\neg L(s,t) \Rightarrow d(s,t) = \min\{d(s,u)+d(u,v)+d(v,t) : u \in A(s), v \in A(t)\}$$

- Berechne Distanzen zwischen allen Transitknoten und zu allen Zugangsknoten vor
- Nichtlokale Anfrage kann dann direkt mit obiger Formel beantwortet werden
- Lokale Anfrage: Dafür brauchen wir noch Plan B



Frage

- Gegeben: Transitknoten T und Lokalitätsfilter L
- Wie berechnet man die entsprechenden Zugangsknoten $A(v) \subseteq 2^T$ für jeden Knoten v?

Frage

- Gegeben: Transitknoten T und Lokalitätsfilter L
- Wie berechnet man die entsprechenden Zugangsknoten $A(v) \subseteq 2^T$ für jeden Knoten v?

Wir müssen $A(v)$ so berechnen, dass

$$\neg L(s,t) \Rightarrow d(s,t) = \min\{d(s,u)+d(u,v)+d(v,t) : u \in A(s), v \in A(t)\}$$

Wähle

$$A(v) := \{w \in T \mid w \text{ ist der 1. Knoten aus } T \text{ auf einem kürz. } s\text{-}w\text{-Weg}\}$$

Berechnung von Zugangsknoten

- Sei $G[T]$ der leere Graph
- für jeden Transitknoten $v \in T$
  - starte Dijkstra's Algorithmus mit Wurzel v auf umgedrehtem Graphen
    Stoppe die Suche, wenn alle Pfade zu Knoten in der Queue einen Knoten aus T enthalten.
  - Speichere v als zusätzichen Zugangsknoten für jeden Knoten u auf einem kürzesten Weg der keinen anderen Knoten aus T enthält
  - Speichere zusätzlich die Kante (v, u) mit Länge $\mathrm{dist}(v, u)$ in den Graphen $G[T]$
- Berechne APSP in $G[T]$

Eine geometrische Transit-Node Routing Implementation

- **Vorgehensweise:** Erst Lokalitätsfilter L festlegen, dann dafür gute Wahl von Transitknoten T finden
- **Annahme:** Geokoordinaten gegeben
- **Vorarbeit 1:** Partitioniere Graphen in $g \times g$ Gitter, bestehend aus quadratischen Zellen (für ein g)
- **Vorarbeit 2:** Sortiere die Knotenmenge beliebig. Ergebnis: Ordnung $\prec$.

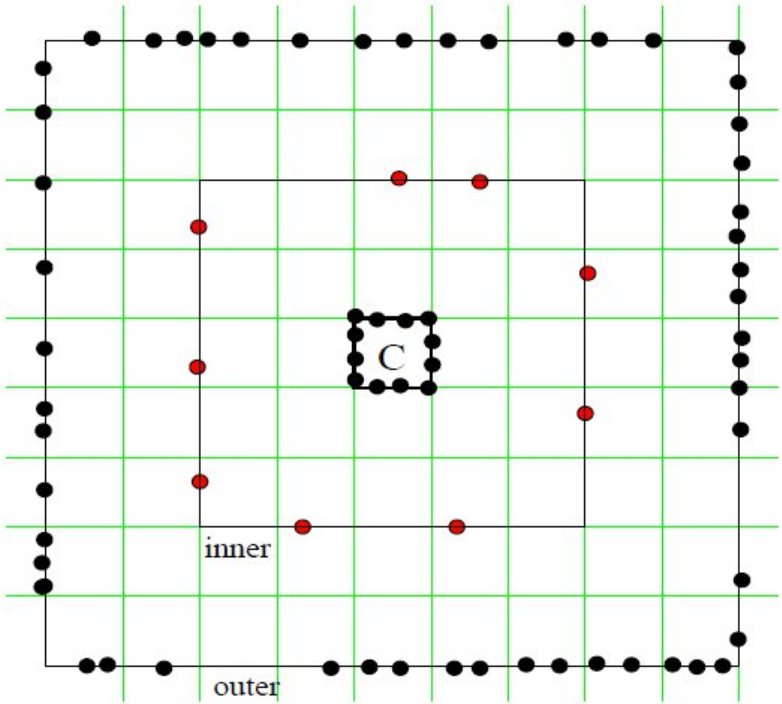
Eine geometrische Transit-Node Routing Implementation

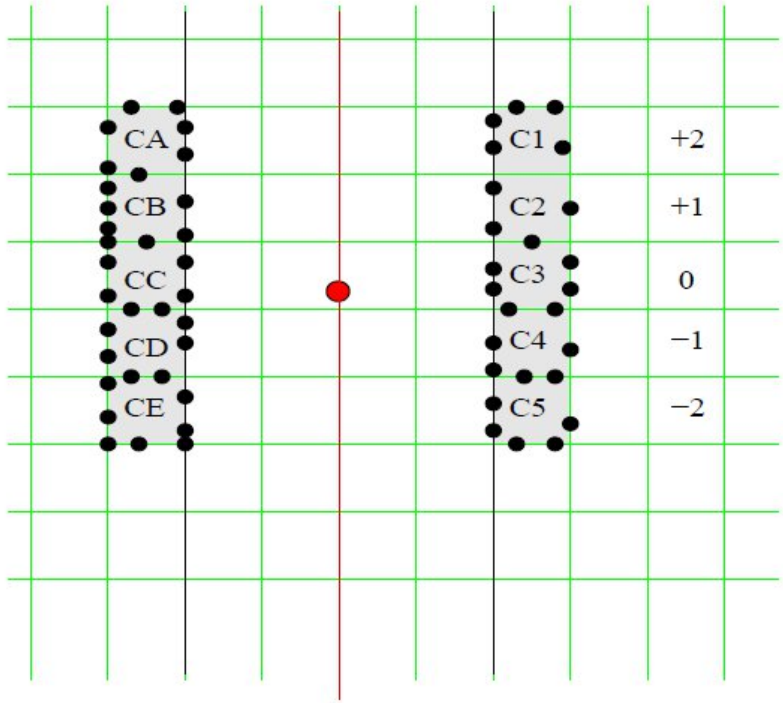
Lokalitätsfilter:
s und t sind mindestens 5 Zellen entfernt (horizontal oder vertikal)

- Sei C eine Zelle
- Sei S_{inner} das 5x5 Quadrat mit Zentrum c
- Sei S_{outer} das 9x9 Quadrat mit Zentrum c
- Sei E_C die Kantenmenge mit genaum einem Endknoten in C
- Sei $V_C := \{u \in V \mid \{u, v\} \in E_C, u \prec v\}$
- Sei V_{inner}, V_{outer} analog definiert

Transitknoten (von Zelle C induziert):
Alle Knoten aus V_{inner} die auf mindestens einem kürzesten Weg von V_C nach V_{outer} liegen

Eine geometrische Transit-Node Routing Implementation





Eine einfache Art die Transitknoten zu berechnen

- Für jede Zelle C
  - Für jeden Knoten $v \in V_C$
    - Dijkstra's Suche mit Wurzel v bis alle Knoten aus V_{outer} besucht sind
    - Ergänze die Transitknoten entsprechend

Nachteil: Dauert mehrere Tage (Europagraph) schon für ein sehr grobes 128x128 Gitter.

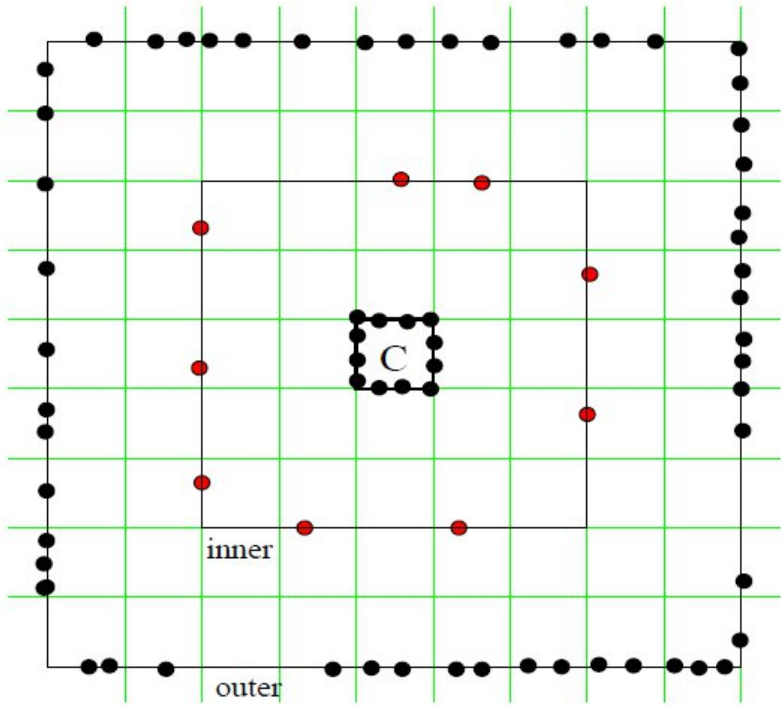
Ein Sweepline-Verfahren um die Transitknoten zu berechnen

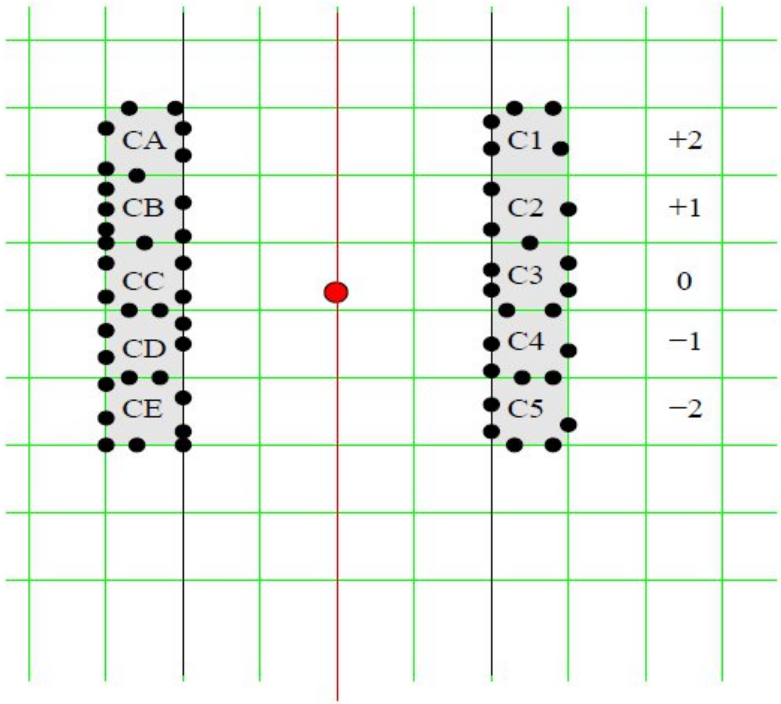
Sweepline-Verfahren

Verbreitete geometrische Technik, bei der eine imaginäre horizontale oder vertikale Linie von links nach rechts (oder oben nach unten) über das betrachtete Objekt bewegt wird.

Ein Sweepline-Verfahren um die Transitknoten zu berechnen

Notation für nächste Folie





Ein Sweepline-Verfahren um die Transitknoten zu berechnen

Ziel: Reduziere die Tiefe der berechneten Kürzeste-Wege-Bäume von etwa 5 auf etwa 3 Zellen

- Für eine vertikale Sweepline SL von links nach rechts
  - Für jede Kante $\{u, v\}$ die SL kreuzt
    - sei u der Knoten mit $u \prec v$
    - Dijkstra's Suche mit Wurzel u bis alle Knoten die maximal 3 Zellen links oder rechts neben v sind besucht wurden
  - Für jede Kante $\{u, v\}$ die SL kreuzt
    - sei u der Knoten mit $u \prec v$
    - wenn $\exists x \in C\{A, B, C, D, E\}, y \in C\{1, 2, 3, 4, 5\}$ so dass $\text{dist}(x, u) + \text{dist}(u, y) = \text{dist}(x, y)$ dann setze u als Transitknoten
- Wiederhole entsprechend für horizontale Sweepline von oben nach unten

Ein Sweepline-Verfahren um die Transitknoten zu berechnen

Ziel: Reduziere die Tiefe der berechneten Kürzeste-Wege-Bäume von etwa 5 auf etwa 3 Zellen

- Für eine vertikale Sweepline SL von links nach rechts
  - Für jede Kante $\{u, v\}$ die SL kreuzt
    - sei u der Knoten mit $u \prec v$
    - Dijkstra's Suche mit Wurzel u bis alle Knoten die maximal 3 Zellen links oder rechts neben v sind besucht wurden
  - Für jede Kante $\{u, v\}$ die SL kreuzt
    - sei u der Knoten mit $u \prec v$
    - wenn $\exists x \in C\{A, B, C, D, E\}, y \in C\{1, 2, 3, 4, 5\}$ so dass $\mathrm{dist}(x, u) + \mathrm{dist}(u, y) = \mathrm{dist}(x, y)$ dann setze u als Transitknoten
- Wiederhole entsprechend für horizontale Sweepline von oben nach unten

Bemerkung $\mathrm{dist}(x, y)$ kann leicht aus der vorberechneten Information gewonnen werden.

Zwischenzusammenfassung

- Einteilung in geometrisches Gitter
- Anfrage ist lokal, wenn Start und Ziel höchstens 4 Zellen auseinanderliegen
- Lokale Anfrage: Benutze Dijkstra
- Nichtlokale Anfrage: Berechne Distanz mit Tablelookups über

$A(v) := \{w \in T \mid w \text{ ist der 1. Knoten aus } T \text{ auf einem kürz. } s\text{-}w\text{-Weg}\}$

Distanzen vs. Kürzeste Wege

- Bis jetzt berechnet das Vorgehen nur $\mathrm{dist}(s, t)$
- Für manche Anwendungen genügt dies
- Wie kann man zusätzlich einen kürzesten s-t-Weg ausgeben lassen?

Möglichkeit 1: Allgemeines Verfahren für schnelle Techniken

- Eine Technik die Distanzanfragen beantworten kann heißt auch **Distanzorakel**
- Straßengraphen haben beschränkten Knotengrad

Möglichkeit 1: Allgemeines Verfahren für schnelle Techniken

Karlsruher Institut für Technologie

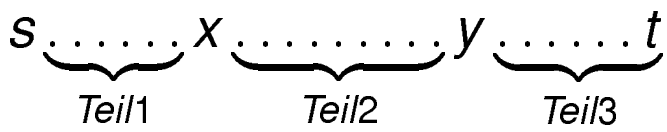
Algorithmus

- Kürzester Weg $P =< s >$
- aktuellerKnoten $:= s$
- Solange aktuellerKnoten $\neq t$
  - finde ausgehenden Kante $(aktuellerKnoten, w)$ von aktuellerKnoten mit $\text{dist}(s, t) = \text{dist}(s, aktuellerKnoten) + len(aktuellerKnoten, w) + \text{dist}(w, t)$
  - $P := P \oplus < w >$
  - aktuellerKnoten=w

Fazit: Mit (fast) Konstantzeit-Distanzorakel ist die Angabe eines kürzesten Wegen in (fast) konstanter Zeit pro Kante des kürzesten Weges möglich.

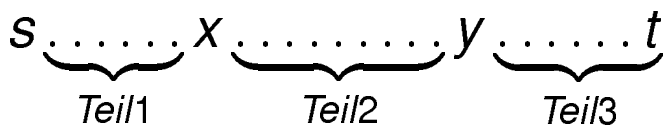
Möglichkeit 2: Mit vorgespeicherter Information

Betrachte kürzesten *s*-*t*-Weg mit Zugangsknoten *x* und *y*



Möglichkeit 2: Mit vorgespeicherter Information

Betrachte kürzesten s-t-Weg mit Zugangsknoten x und y

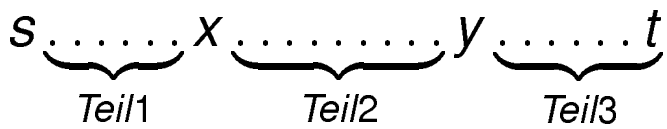


Teil 2:

- **Speichere alle kürzesten Wege zwischen Transitknoten (rekursiv)**
- Speichere dazu für jedes Paar $u, v \in T$ einen Knoten w auf einem Kürzesten u-v-Weg (oder null falls (u, v) Kante)
- Speichere dann für u, w einen Knoten auf einem kürzesten u-w-Weg (oder null falls (u, w) Kante)
- Speichere dann für w, t einen Knoten auf einem kürzesten w-v-Weg (oder null falls (w, v) Kante)
- Führe dies rekursiv fort, bis kürzester Weg gespeichert

Möglichkeit 2: Mit vorgespeicherter Information

Betrachte kürzesten s-t-Weg mit Zugangsknoten x und y



Teil 1 und 3

- Wie Möglichkeit 1, mit vorberechneter Distanz zum Verbindungsknoten als Orakel

Wie kann man lokale Anfragen beschleunigen?

Wie kann man lokale Anfragen beschleunigen?

- Benutze andere Beschleunigungstechnik für lokale Anfragen
- Mehrstufige Transitknoten

Mehrstufige Transitknoten

Mehrstufige Transitknoten

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

Mehrstufige Transitknoten

- **Anfrage:**
  Benutze höchsten Level in dem die Anfrage nicht lokal ist
- Speichere Distanzen (außer im höchsten Level) in Hashtabellen anstatt in Matrizen
- Berechne für in einen level $dist(u, v)$ nur, wenn u-v im nächsthöherem level lokal ist

Experimente I

	$\lvert\mathcal{T}\rvert$	$\lvert\mathcal{T}\rvert \times \lvert\mathcal{T}\rvert$/node	avg. $\lvert A\rvert$	% global queries	preprocessing
64×64	2 042	0.1	11.4	91.7%	498 min
128×128	7 426	1.1	11.4	97.4%	525 min
256×256	24 899	12.8	10.6	99.2%	638 min
512×512	89 382	164.6	9.7	99.8%	859 min
$1\,024 \times 1\,024$	351 484	2 545.5	9.1	99.9%	964 min

Experimente I

non-local (99%)	local (1%)	all queries	preprocessing	space per node
12 μs	5112 μs	63 μs	20 h	**21** bytes

Experimente II

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
- Wichtig: Die Zahlen beziehen sich auf eine andere Variante von Transit-Node Routing

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

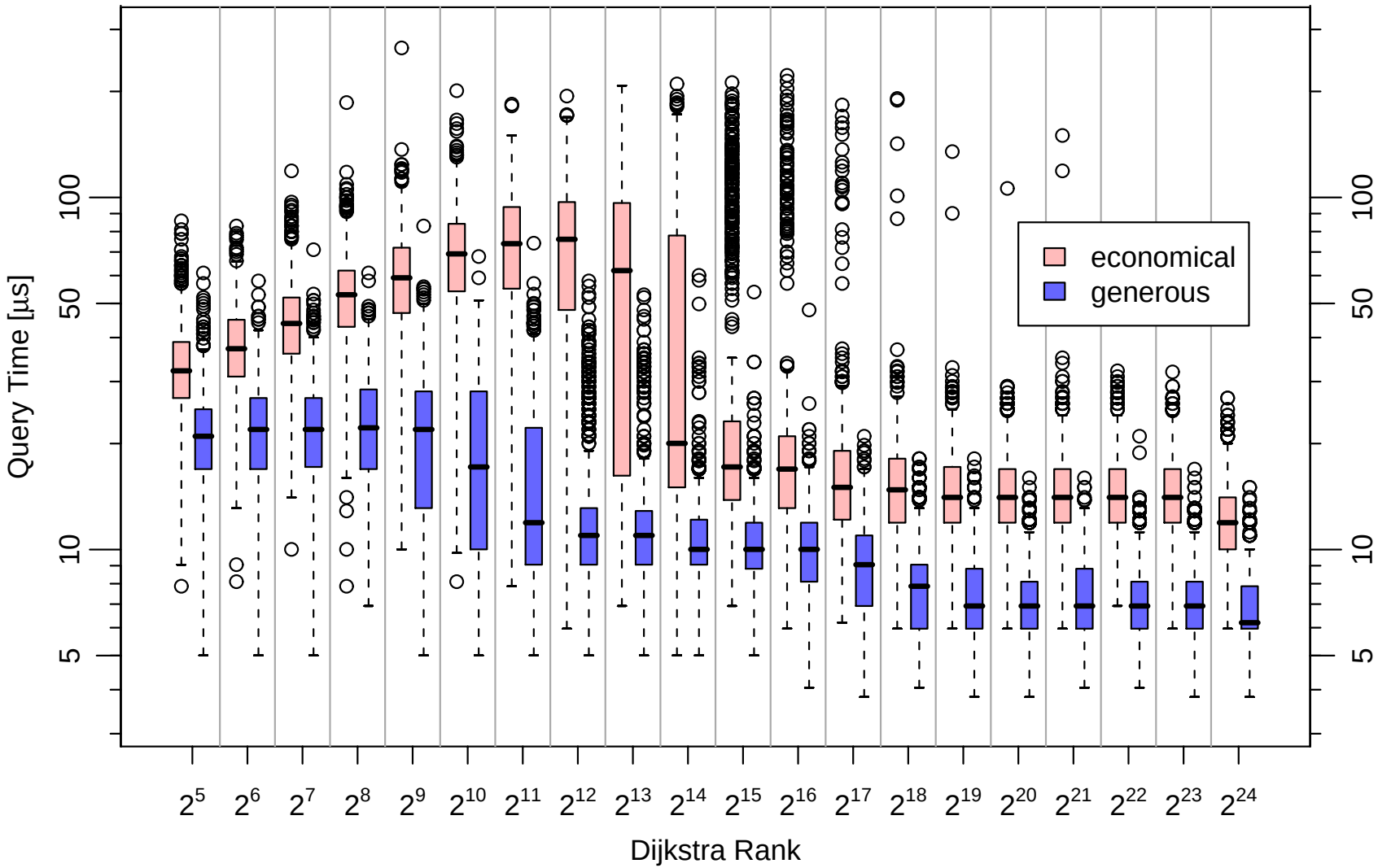
TNR: Anfragen

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
	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
	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
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

Dijkstra Rank



Ende

Literatur (Transit-Node Routing):

- Holger Bast, Stefan Funke, Domagoj Matijevic, Peter Sanders, Dominik Schultes:
  In Transit to Constant Time Shortest-Path Queries in Road Networks
  In: *Proceedings of the 9th Workshop on Algorithm Engineering and Experiments (ALENEX'07)*, 2009