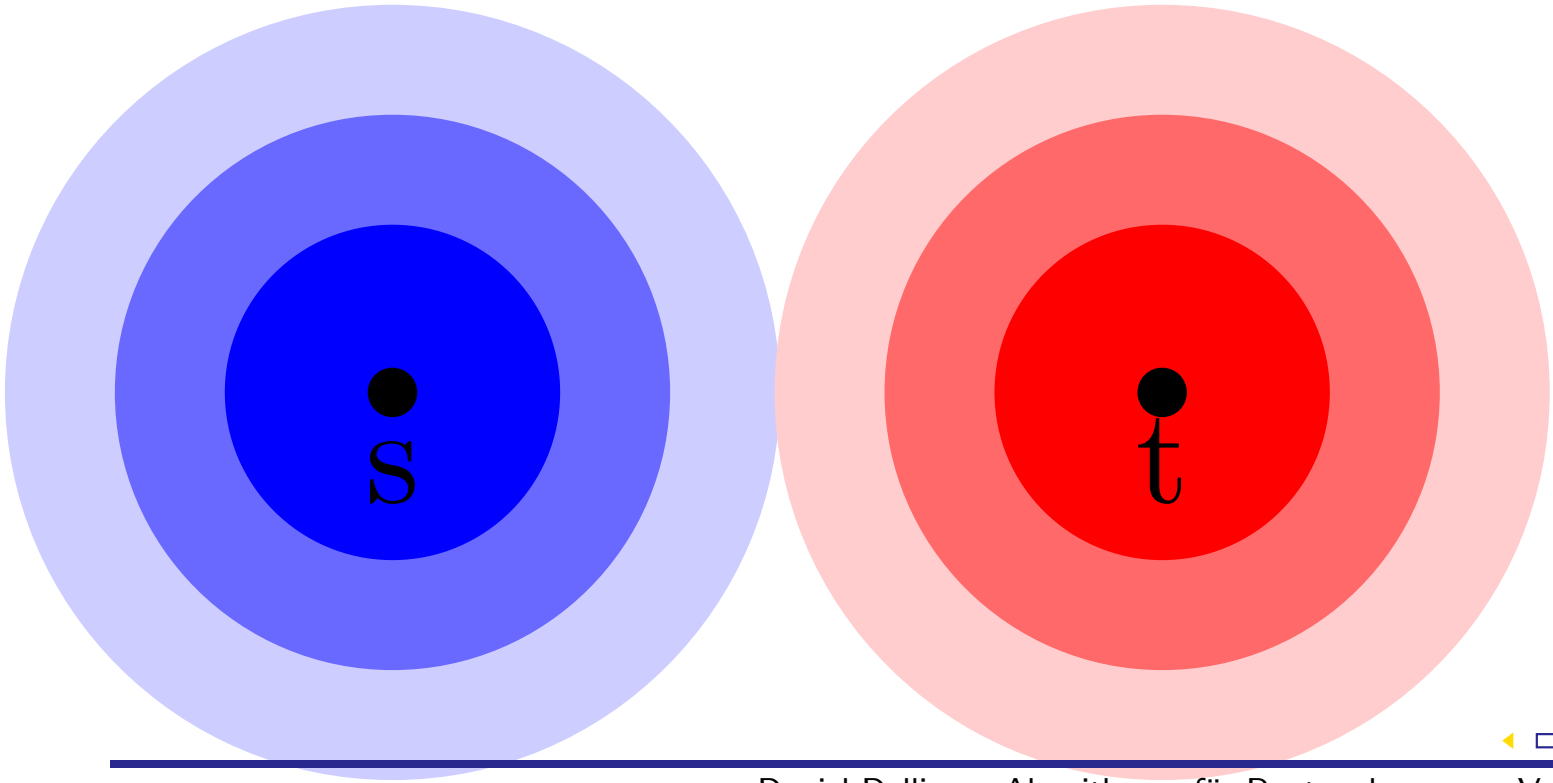
Algorithmen für Routenplanung – Vorlesung 6

Daniel Delling

Lehrstuhl für Algorithmik I
Institut für theoretische Informatik
Universität Karlsruhe (TH)
Forschungsuniversität · gegründet 1825

Letztes Mal

- Highway Hierarchies
- Highway-Node Routing
- Contraction Hierarchies



Zusammenfassung letztes Mal

- Highway Hierarchies
- Highway-Node Routing
- Contraction Hierarchies

Gemeinsamkeiten:

- hierarchisch
- suchen nur aufwärts

Themen Heute

- Many-to-Many Route Planning
  - berechnet Distanz-Matrix
- Transit-Node Routing
  - baut auf Many-to-Many auf
  - schnellste Technik für Routeplanung
  - wurde veröffentlicht in Science

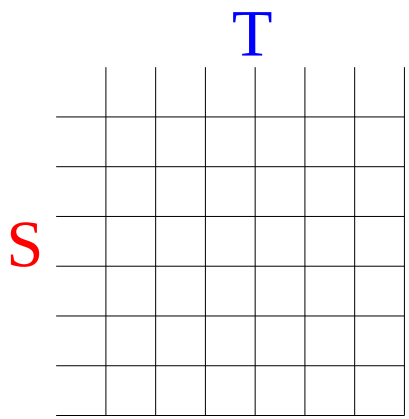
Many-to-Many Kürzeste Wege

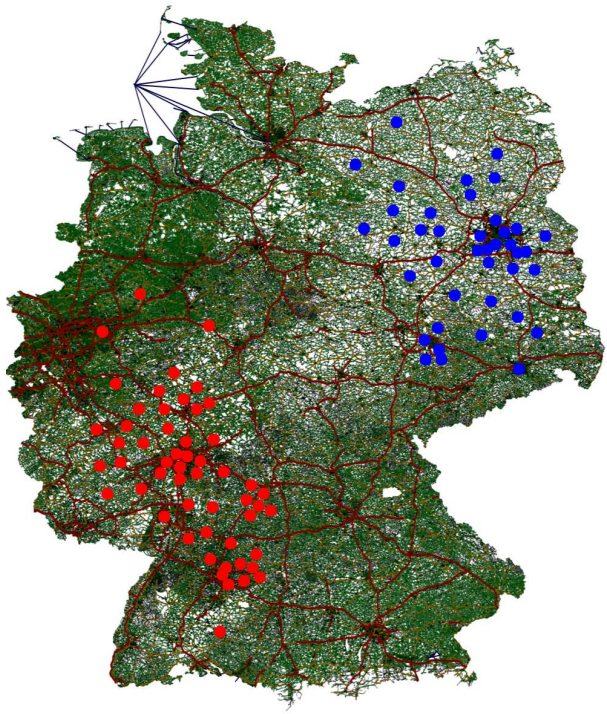
Gegeben:

- Graph
- Knotenmengen $S, T \in V$

Gesucht:

- Distanzmatrix D





Anwendungen:

- vehicle routing
- traveling salesman
- Vorberechnung Transit-Node Routing (später mehr)

Naiver Ansatz

Erster Ansatz:

- führe $|S|$ mal Dijkstra aus
- führe $|S| \times |T|$ mal die Query einer Beschleunigungstechnik aus

Problem:

- typische Eingabegröße (Europa, 10 000 x 10 000 Einträge)
- Variante 1 dauert ca. 1 Tag
- Variante 2 dauert ca. 2-3 Stunden

Ungerichtete Bidirektionale Techniken

Definition:

- $\overrightarrow{\sigma}(s,t)$: Suchraum der Vorwärtssuche von s nach t
- $\overleftarrow{\sigma}(s,t)$ analog
- eine bidirektionale Suche ist Ziel-unabhängig, gdw.

$$\forall (s,t_1,t_2) \in V^3 : \overrightarrow{\sigma}(s,t_1) = \overrightarrow{\sigma}(s,t_2) \quad \text{und}$$
$$\forall (s_1,s_2,t) \in V^3 : \overleftarrow{\sigma}(s_1,t) = \overleftarrow{\sigma}(s_2,t)$$

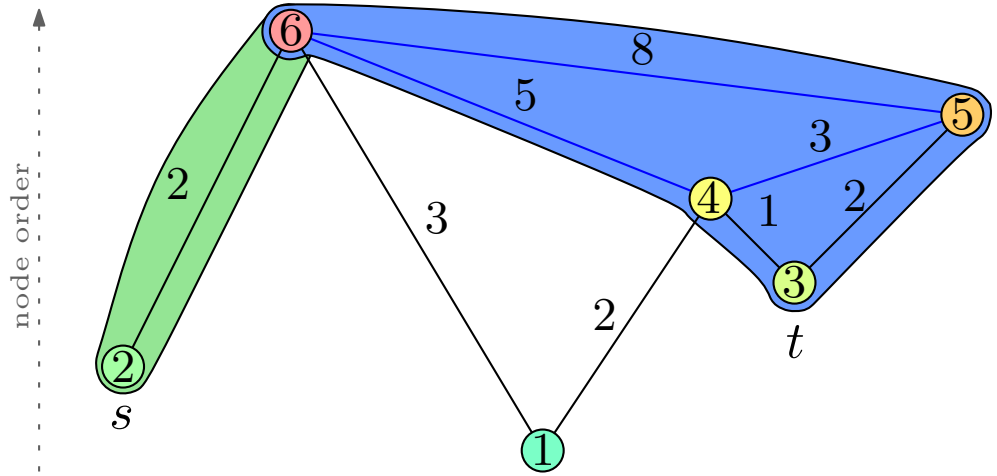
Beispiele:

- Bidirektionaler Dijkstra
- ohne Stoppkriterium, lass laufen bis Queues leer sind

Hierarchische Techniken

Beobachtung:

- suchen nur aufwärts
- sind nicht zielgerichtet



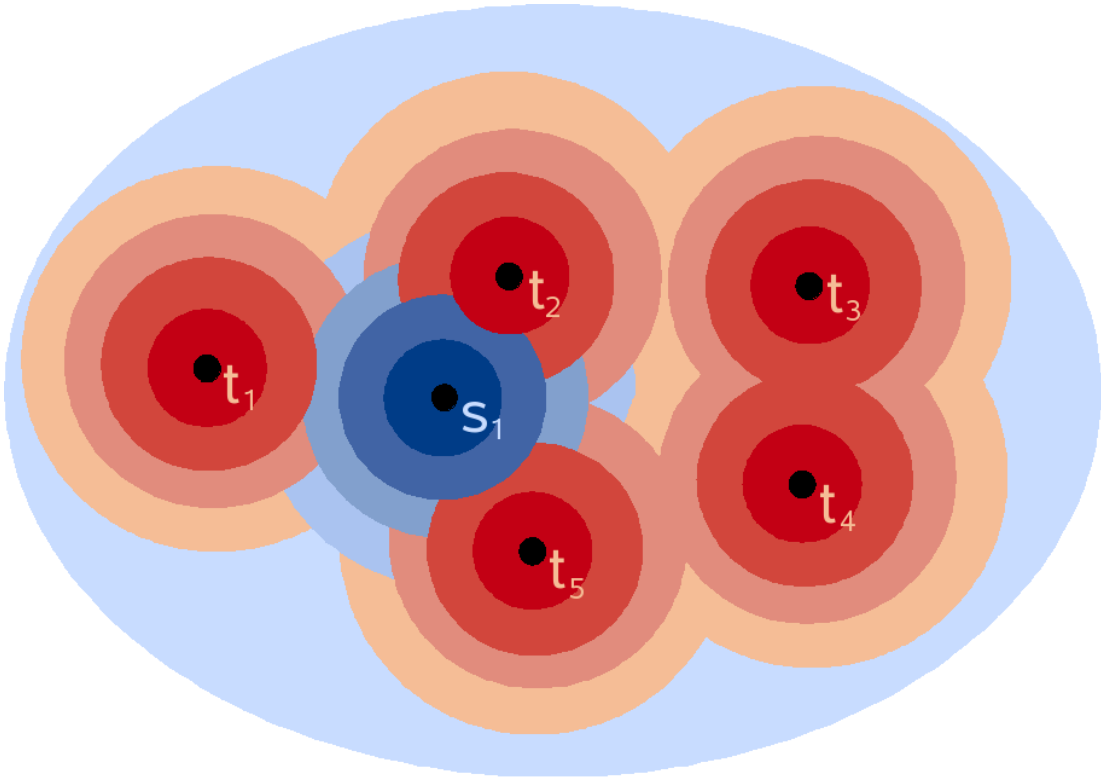
somit:

- Bidirektionaler Dijkstra
- Reach/RE
- Highway Hierarchies
- Highway-Node Routing
- Contraction Hierarchies
- ohne Stoppkriterium, lass laufen bis Queues leer sind

Vorgehen

Idee:

- führe $|T|$ Rückwärtssuchen aus
- speicher für jedes u die Abstände zu allen $t \in T$
- verwalte temporäre Distanztabelle D
- führe $|S|$ Vorwärtssuchen aus
- aktualisiere Einträge in D



Problem:

- verwalten der Suchräume?

Schneiden der Suchräume

während Rückwärtssuchen:

- breche nicht ab
- für jedes erreichte u:
  - füge Element $(u, t, d(u, t))$ in einen Vektor ein

Kompression:

- sortiere Einträge nach u
- speichere für jeden Knoten u einen Bucket $\beta(u)$ mit allen $(t, d(u, t))$ ab
- mittels Adjazenzarray

während Vorwärtssuchen:

- breche nicht ab
- für jedes erreichte u:
  - durchsuche Bucket $\beta(u)$
  - aktualisiere Distanzmatrix

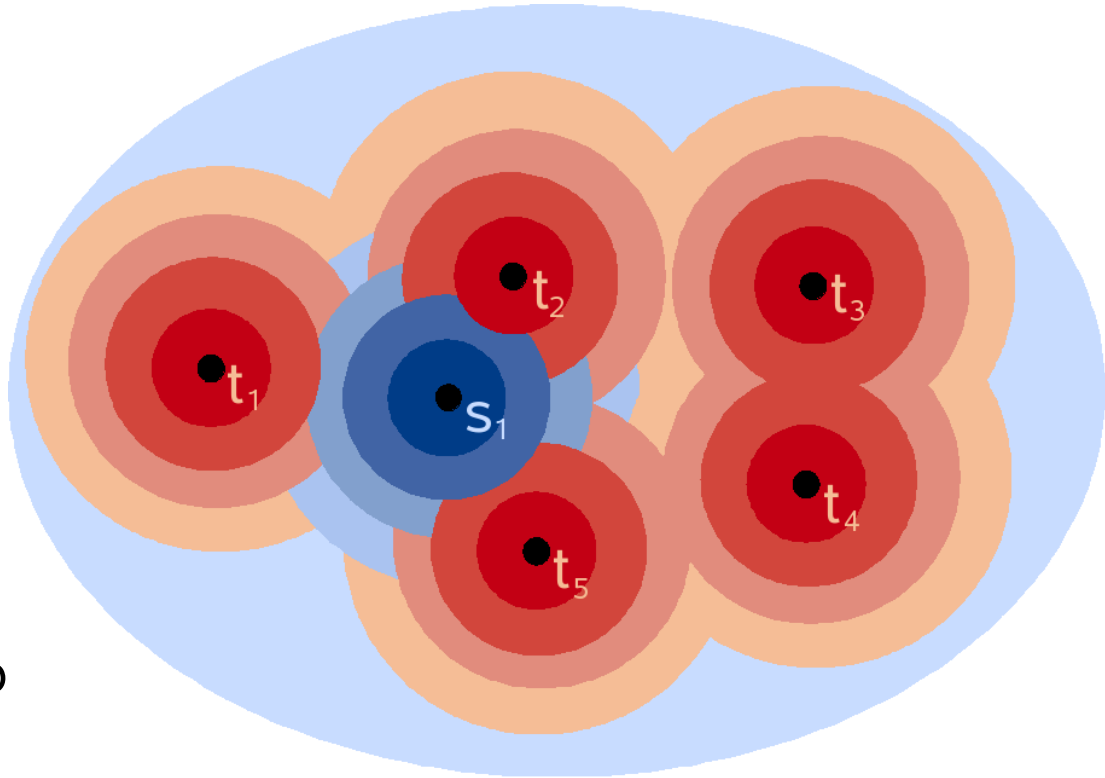
Asymmetrie

Beobachtungen:

- Durchsuchen der Buckets dominiert die Laufzeit

Idee:

- Rückwärtssuche durchsucht nicht obersten Level
- schneide Hierarchie oben ab
- dadurch weniger Buckets
- aber: Vorwärtssuche dauern länger



Pfadentpackung

Vorgehen:

- speichere jeden Suchbaum von $t \in T$
- speichere jeden Suchbaum von $s \in S$
- für jeden Eintrag in D halte Zeiger auf Knoten in den beiden Suchbäumen, die zu s und t gehören, vor

Optimierung:

- schneide Teile der Suchbäume weg, die man nicht benötigt

Ergebnisse M2M Europa

Setup:

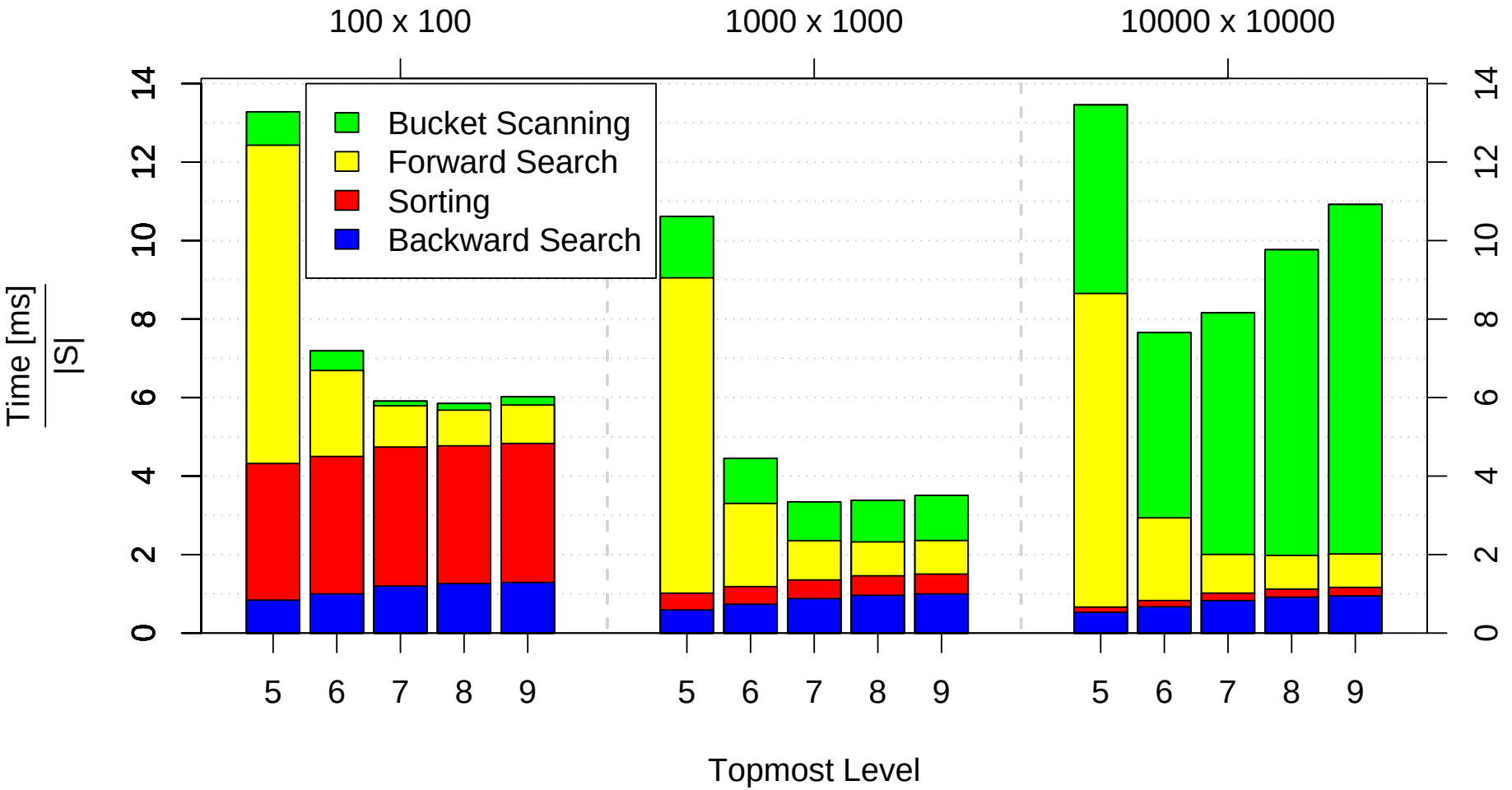
- Eingabe Europa
- berechne $S \times S$ Distanztabelle
- Laufzeit in Sekunden

$\lvert S \rvert$	100	500	1 000	5 000	10 000	20 000
HH	0.6	1.7	3.3	26.3	76.6	247.7
HNR	0.4	0.8	1.4	8.5	23.2	75.1
CH	0.4	0.5	0.6	3.3	10.2	36.6

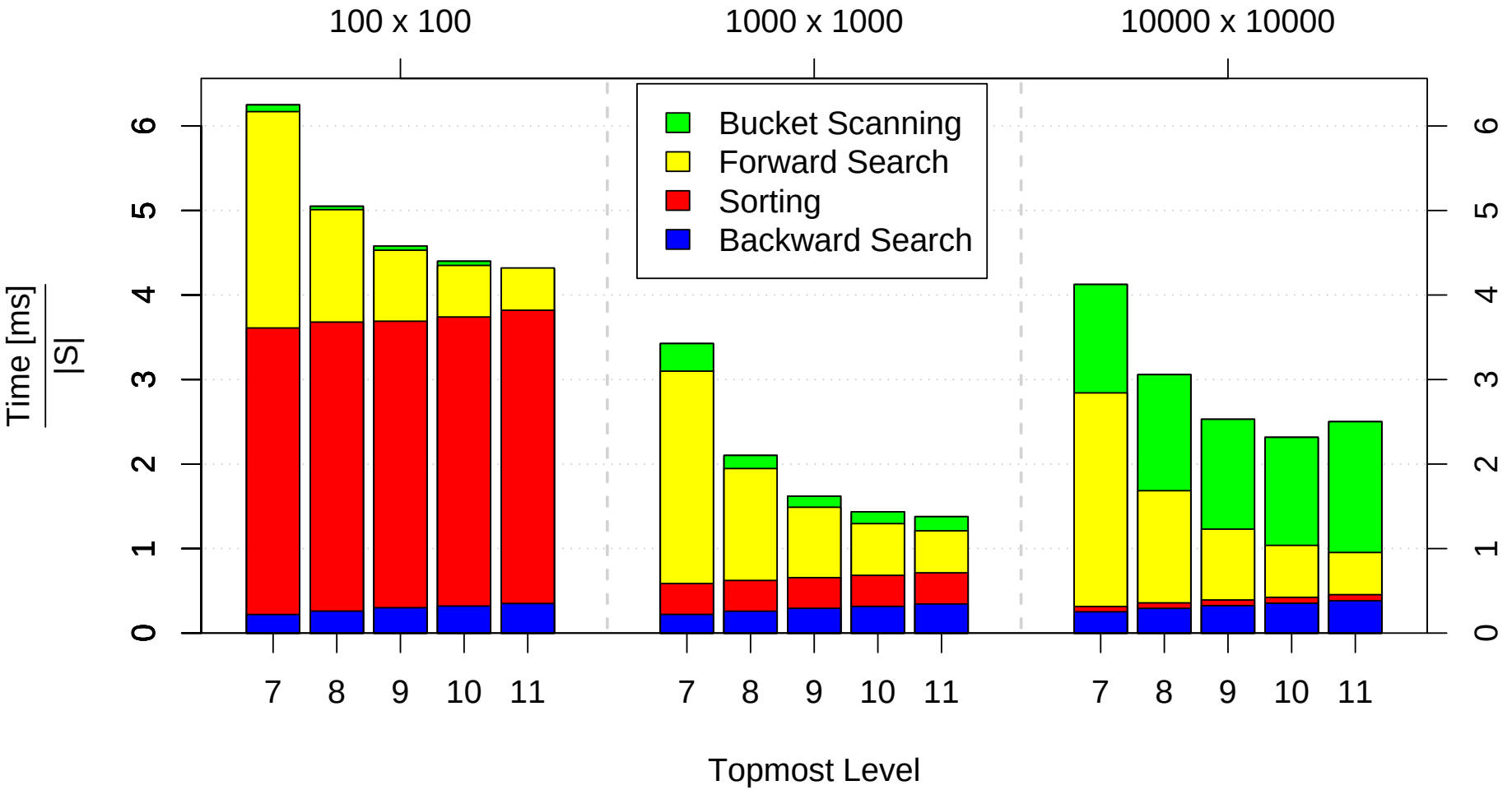
Beobachtung:

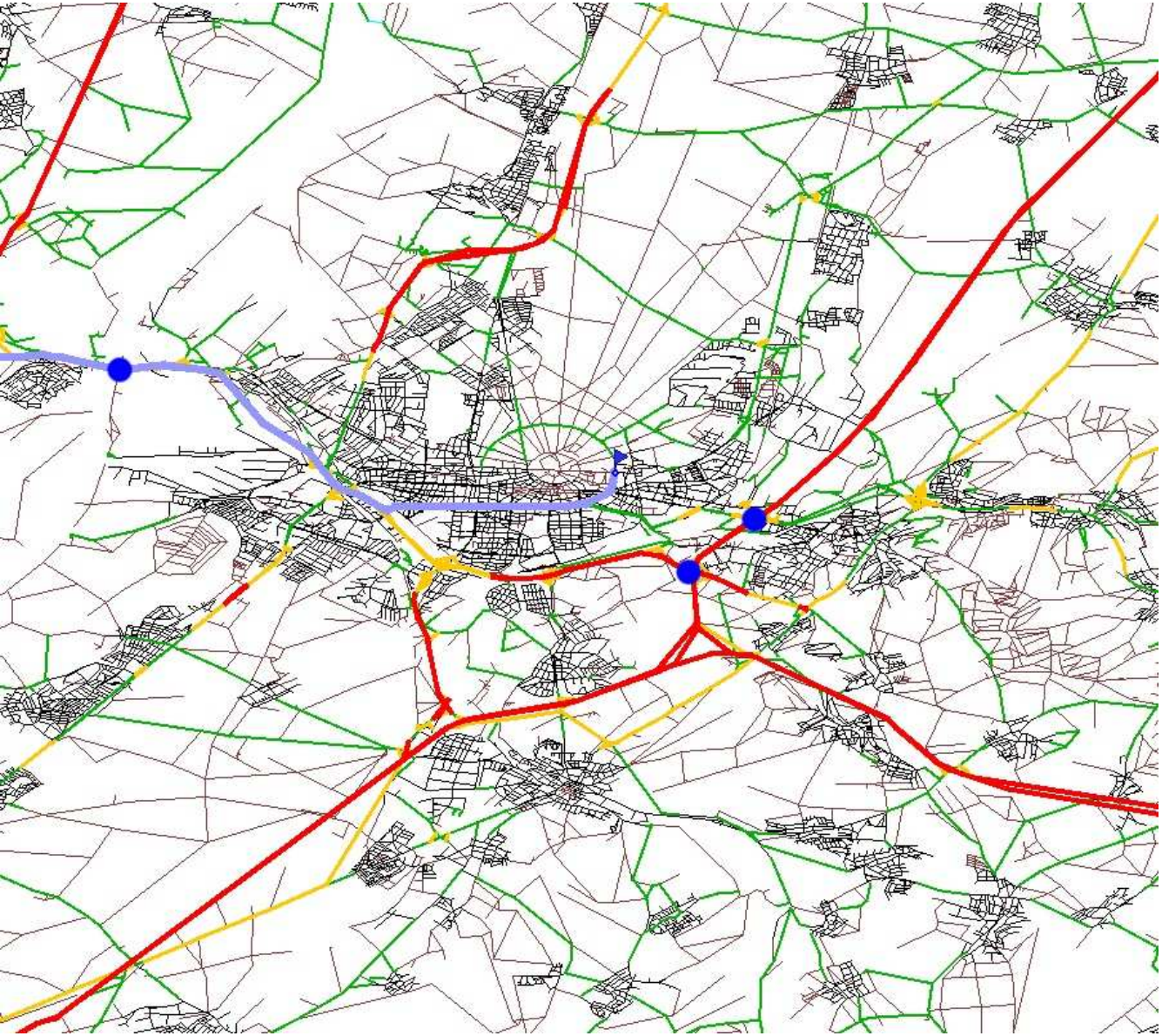
- CH schneller als HNR schneller als HH
- 10 000 Matrix in 10,2 Sekunden berechenbar
- pro Eintrag: 0.1 μs

Ergebnisse M2M HH



Ergebnisse M2M HNR





Beobachtung:

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

Transit-Node Routing

Ziel:

- reduziere Anfragen auf Table-Lookups

Idee:

- erweitere den HH + Distanztabellen Ansatz

Probleme:

- Speicherverbrauch
- nahe Anfragen?

Erster Ansatz: HH + Distanztabelle

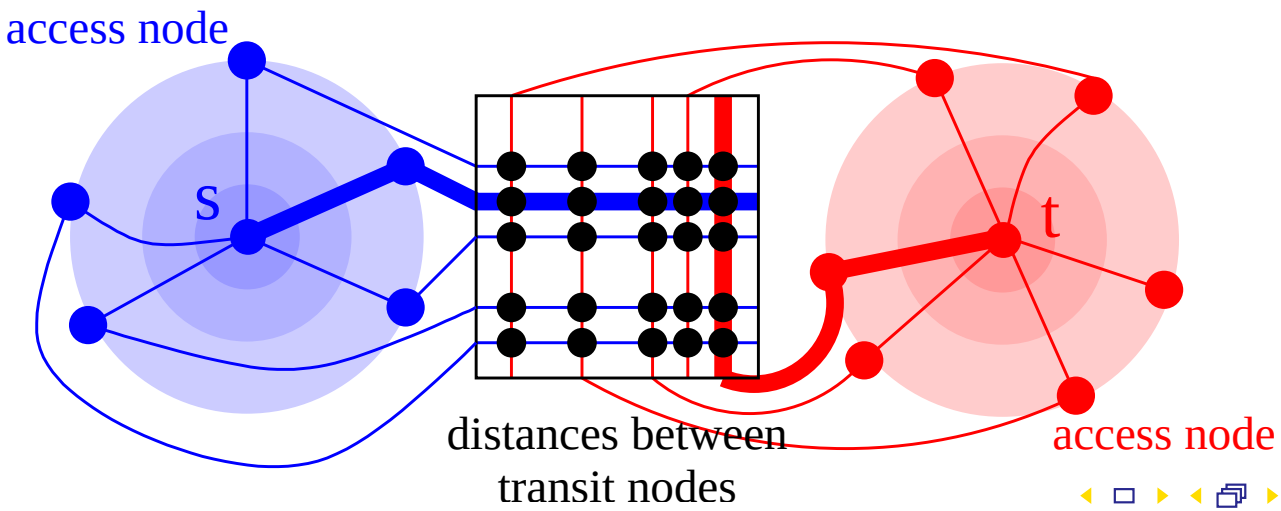
Idee:

- speichere Distanzmatrix für letzte Level
- speichere Abstand von und zu allen relevanten Knoten (ca. 55 pro Knoten)
- Abstand:

$$d_T(s,t) = \min\{d(s,u) + d(u,v) + d(v,t) \mid u \in \overrightarrow{A}(u), v \in \overleftarrow{A}(v)\}$$

Probleme:

- Speicherverbrauch
- nahe Anfragen?



Transit-Node Routing

gegeben:

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

betrachte für jeden Level $\ell, 0 \leq \ell \leq L$**:**

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

Eigenschaften

Bemerkungen:

- In der Distanztabelle werden nur relevante Informationen abgelegt

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

- d_ℓ gibt die Distanz in dem entsprechenden Level an

TNR Anfragen

TNR-Query(s,t)

1 $d' = \infty$
2 **for** $\ell = L$ **down to** 0 **do**
3 $\quad d' := \min\{d', d_\ell(s,t)\}$
4 $\quad$ **if** $\neg L_\ell(s,t)$ **then break**
5 **return** d'

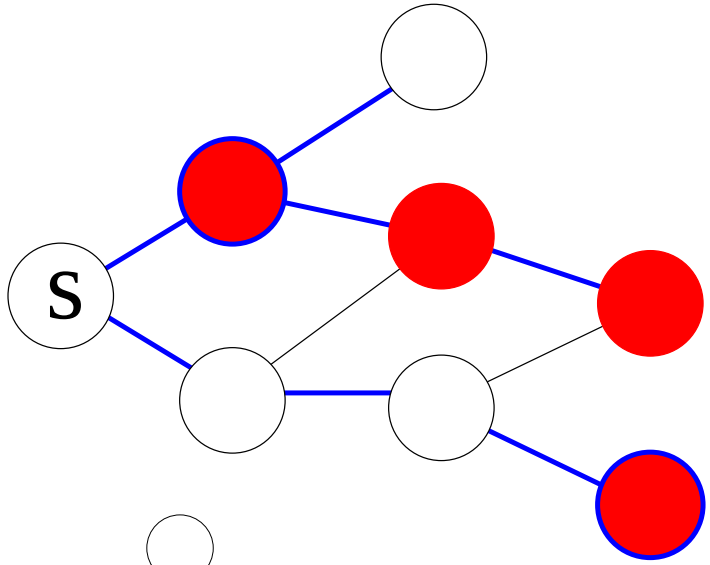
Bemerkungen:

- speichere in D_ℓ nur nicht ∞ Einträge (Hash-Tabelle)
- benutze HNR/CH für Level 0 Anfragen

Access-Nodes berechnen

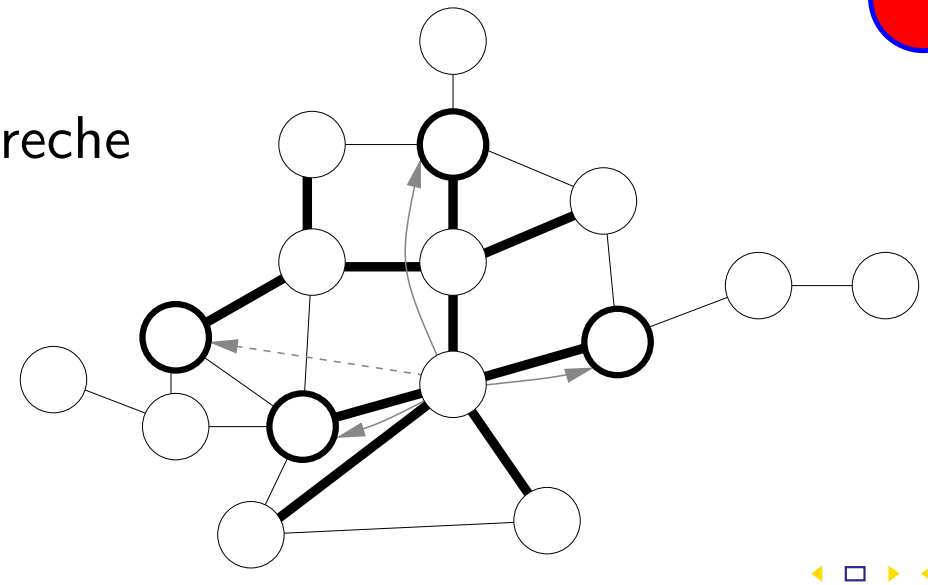
Vorgehen:

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



Beschleunigung:

- benutze HNR Anfrage
- benutze aggressives stalling (breche Zweig bei Transit-Knoten ab)
- dünne dann mit Hilfe der Transit-Tabelle aus



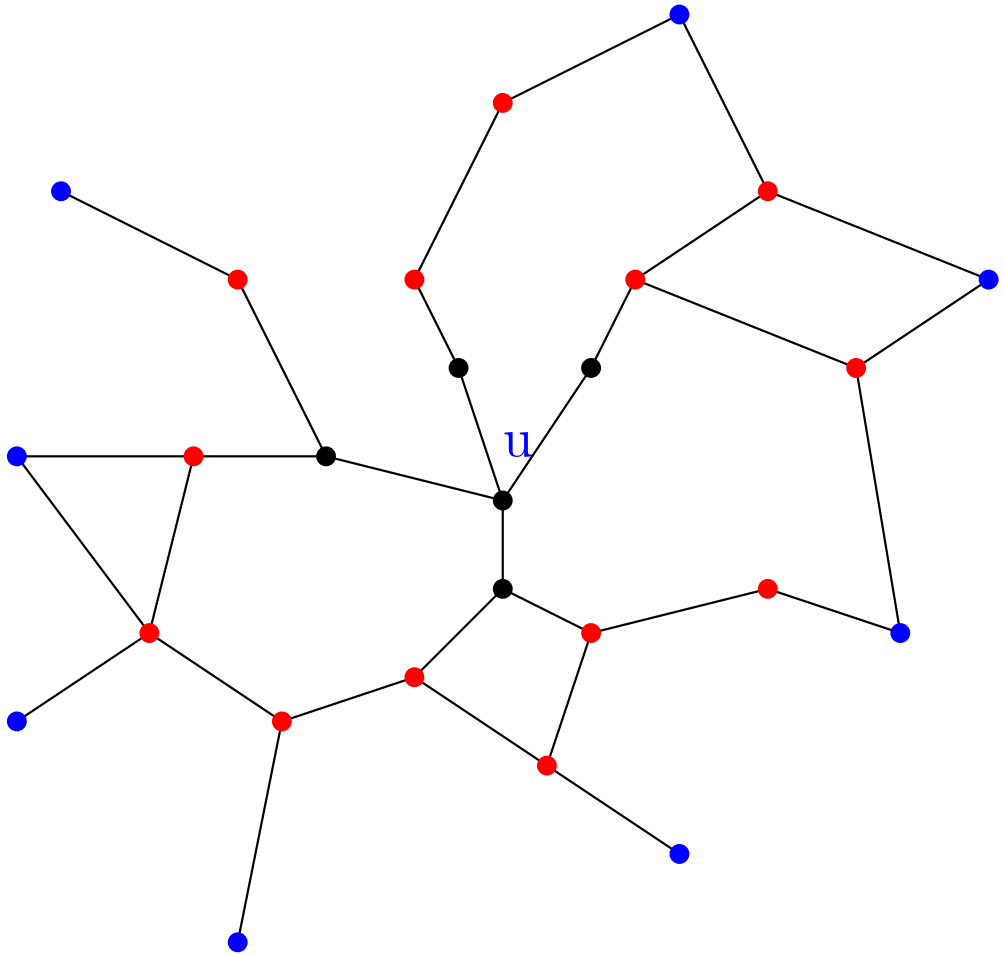
Runterpropagieren von Access-Nodes:

Idee:

- Access-Nodes können propagiert werden

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

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



Distanztabellen

Vorgehen:

- benutze Many-to-Many Ansatz

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:

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

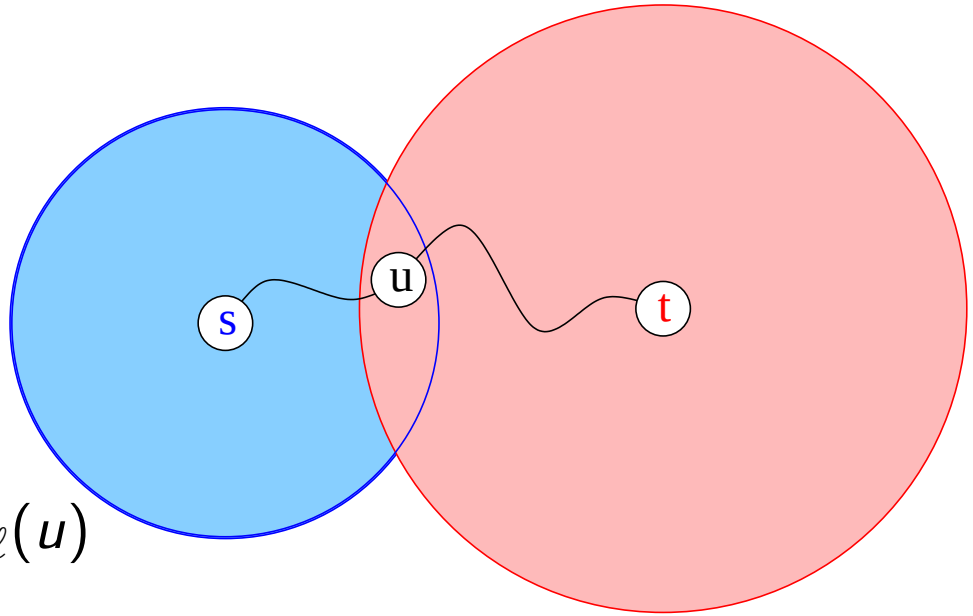
Locality Filter

Vorgehen:

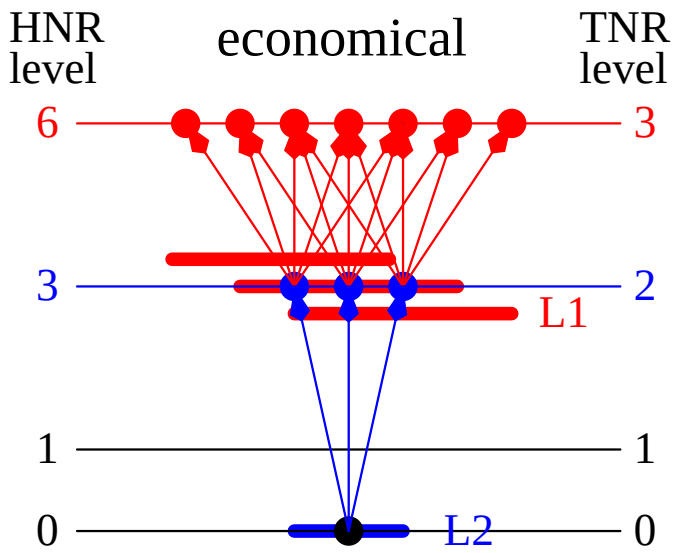
- speicher für jeden Knoten den (euklidischen) Abstand zum am weitesten entfernten Access Node ab, also $\overrightarrow{K}(u)$ und $\overleftarrow{K}(u)$
- für jeden Level, also $\overrightarrow{K}_\ell(u)$, $\overleftarrow{K}_\ell(u)$
- geht während Vorberechnung
- locality filter schlägt zu, wenn

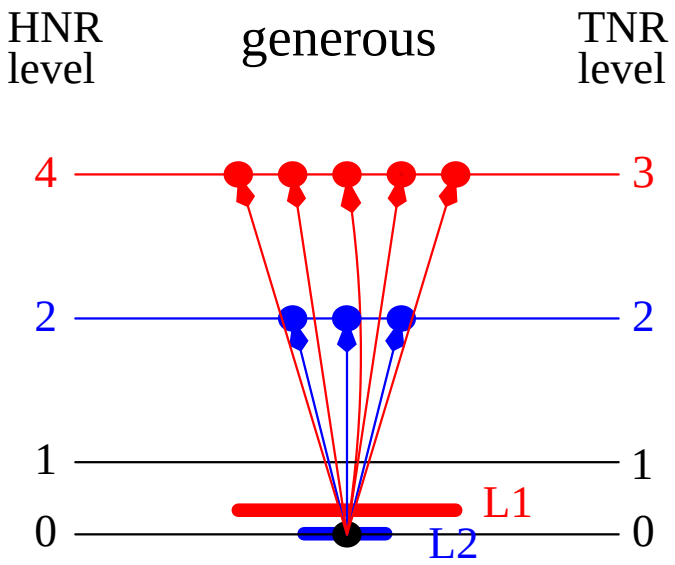
$$||s - t|| < K(s) + K(t)$$

- dadurch manchmal falscher Alarm



2 Varianten





economical:

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

generous:

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

Pfadentpackung

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

Vergleich zu HH/HNR/CH

Gemeinsamkeiten:

- Level-Einteilung
- Pfadentpackung ähnlich
- TNR basiert auf HH/HNR/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

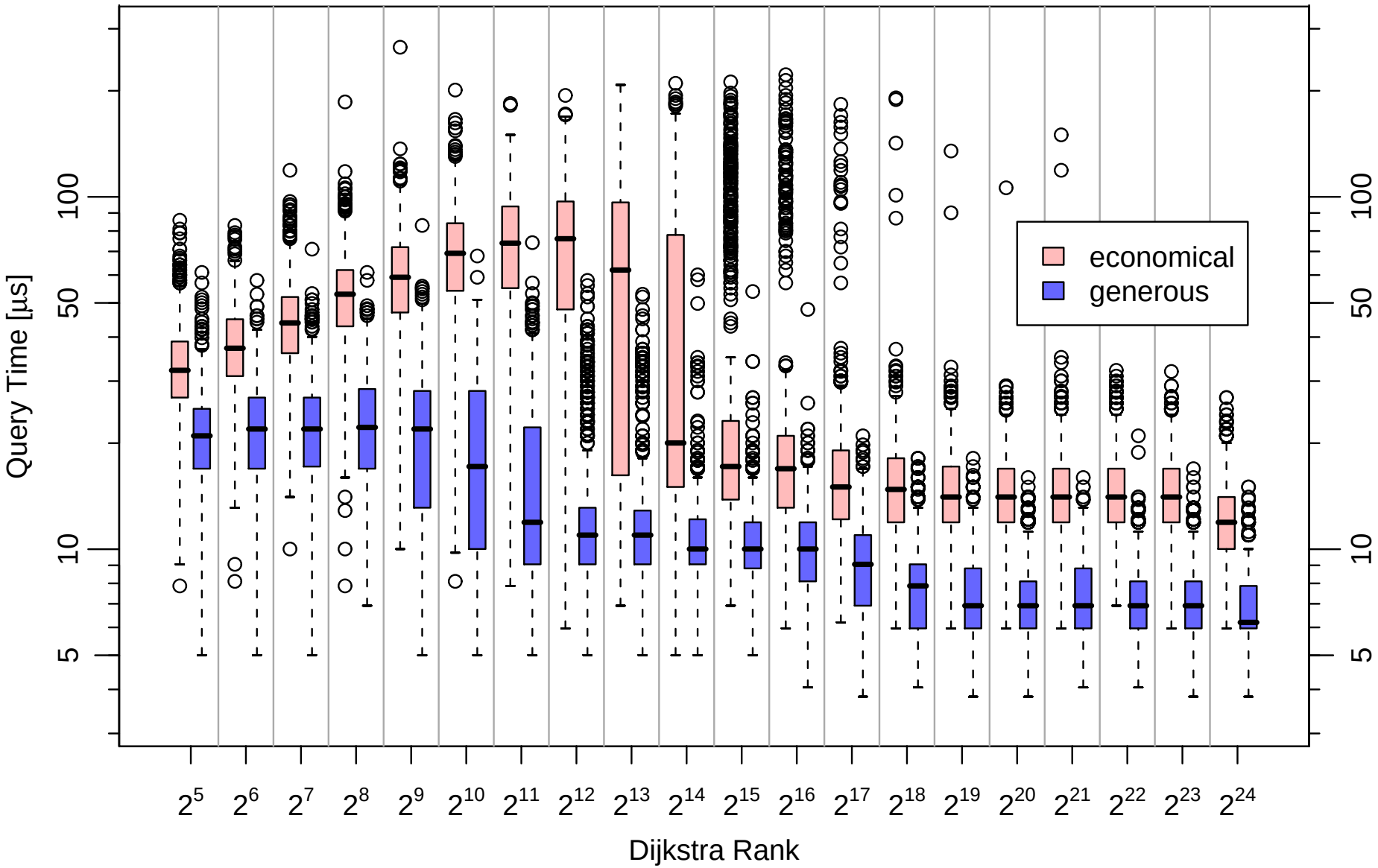
metric	variant	level 3 $\lvert T_3 \rvert$	level 3 $\lvert A_3 \rvert$	level 2 $\lvert T_2 \rvert$	level 2 $\lvert D_2 \rvert$	level 2 $\lvert A_2 \rvert$	overhead [B/node]	time [h]
Europe								
time	eco	9 355	11.4	151 450	0.15%	5.3	99	0:25
	gen	9 458	11.3	293 209	0.14%	4.4	226	1:15
USA (Tiger)								
time	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

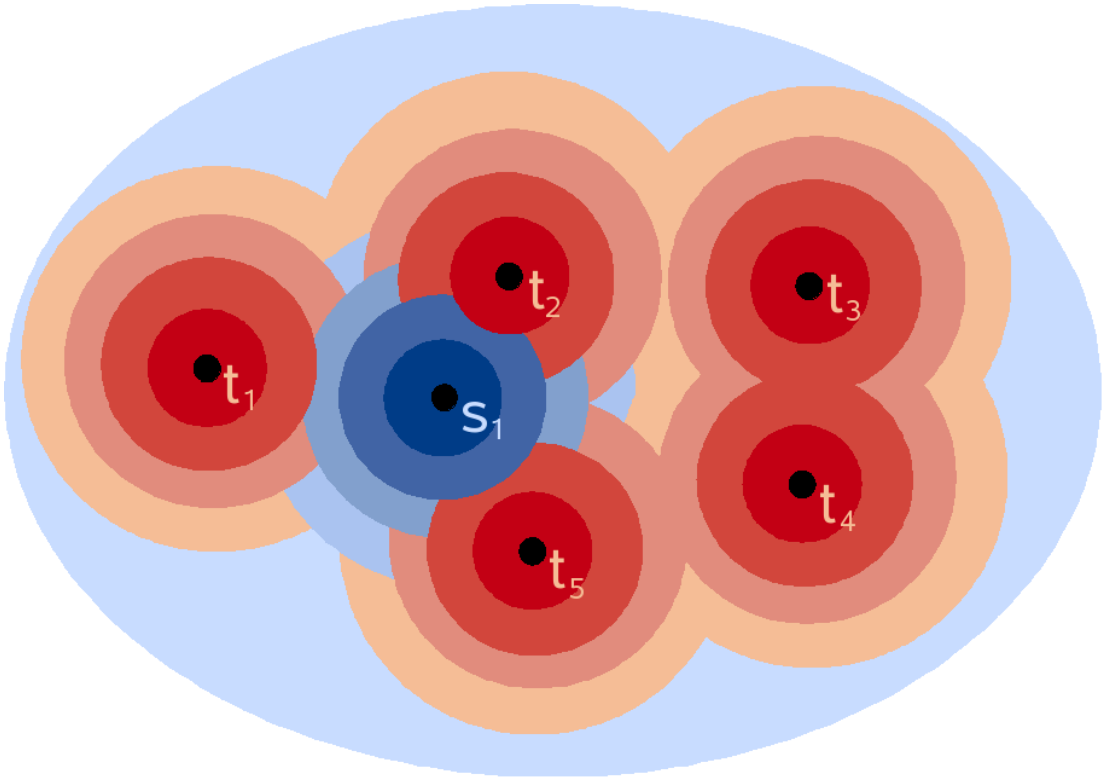
Übersicht: bisherige Techniken

	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
RE	1:22	13	4 643	3.5	1 597
REAL-(64,16)	2:20	35	679	1.1	5 037
HH	0:13	48	709	0.61	9 165
HNR	0:15	2.4	981	0.85	6 577
CH	0:32	-3.0	359	0.15	37 273
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


Query Time [µs]
5
10
50
100
economical
generous
2^5 2^6 2^7 2^8 2^9 2^{10} 2^{11} 2^{12} 2^{13} 2^{14} 2^{15} 2^{16} 2^{17} 2^{18} 2^{19} 2^{20} 2^{21} 2^{22} 2^{23} 2^{24}
Dijkstra Rank

Zusammenfassung Many-to-Many

- berechnet Distanztabellen
- durch $|S| + |T|$ zielunabhängige hierarchische Suchen
- erzeuge Buckets $(u, d(u, t)$ während der Rückwärtssuchen
- durchsuche Buckets bei Vorwärtssuche
- pro Eintrag ca. 0.1 μs



Zusammenfassung Transit-Node Routing

- ersetzt Suche (fast) komplett durch Table-lookups
- 3 Zutaten:
  - Distanztabelle
  - Access Nodes
  - locality filter
- Suchzeit von unter 5 μs
- offenes Problem: Variante, die komplett auf CH aufbaut (erste Ergebnisse: 4.3 $\mu s \rightsquigarrow$ 3.3 μs

Literatur

Many-to-Many:

- Knopp et al. 07
- Schultes 08 (Kapitel 5)

Transit-Node Routing:

- Bast et al. 07
- Schultes 08 (Kapitel 6)

Anmerkung:

- wird auf der Homepage verlinkt
- user: routePlanning / pass: ss09