Karlsruher Institut für Technologie

Algorithmen für Routenplanung

6. Vorlesung, Sommersemester 2012

Daniel Delling | 14. Mai 2012

MICROSOFT RESEARCH SILICON VALLEY

Hierarchische Techniken

Ideensammlung:

- identifiziere wichtige Knoten mit Zentralitätsmaß
- überspringe unwichtige Teile des Graphen

Motivation 1: Netzwerkanalyse

- Zentralitätsmaße bewerten Wichtigkeit von Knoten oder Kanten in einem Netzwerkanalyse
- Häufige Beispiele
  - Google Page Rank
  - Erdös-Zahl

Zentralitätsmaße - Beispiele

- degree centrality

$$C_D(v) = \frac{\deg(v)}{n-1}$$

- betweenness centrality

$$C_B(v) = \sum_{\substack{s \neq v \neq t \in V \\ s \neq t}} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

wobei

- σ_{st} die Anzahl der kürzesten s-t-Wege ist
- $\sigma_{st}(v)$ die Anzahl der kürzesten s-t-Wege durch v ist

Motivation 1: Netzwerkanalyse

Reach:

- Zentralitätsmaß, das groß ist, falls eine Kante in der Mitte eines langen kürzesten Weges liegt.

Motivation 2

- Zeichne um jede Kante (u, v) Kreis mit Radius $r(u, v)$, so dass gilt:
  - Für jedes Paar s, t für das (u, v) auf einem kürzesten s-t-Weg liegt gilt, dass entweder s oder t in dem Kreis liegt.

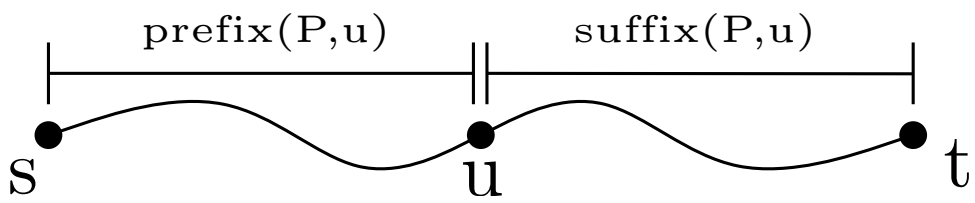
Möglichkeiten für diesen Kreis

- „geometrischer Kreis“ in euklidischer Ebene
- „graphentheoretischer Kreis“ im Eingabegraph

Strategie für Beschleunigungstechnik:

- Beachte Kante (u, v) nicht, wenn s und t nicht im Kreis um (u, v) liegen.
- wie überprüfen?

Das Zentralitätsmaß Reach



Definition:

- Sei $P = \langle s, \ldots, u, \ldots, t \rangle$ Pfad durch u
- dann Reach von u bezüglich P:

$$r_P(u) := \min\{\text{len}(\mathit{prefix}(P, u)), \text{len}(\mathit{suffix}(P, u))\}$$

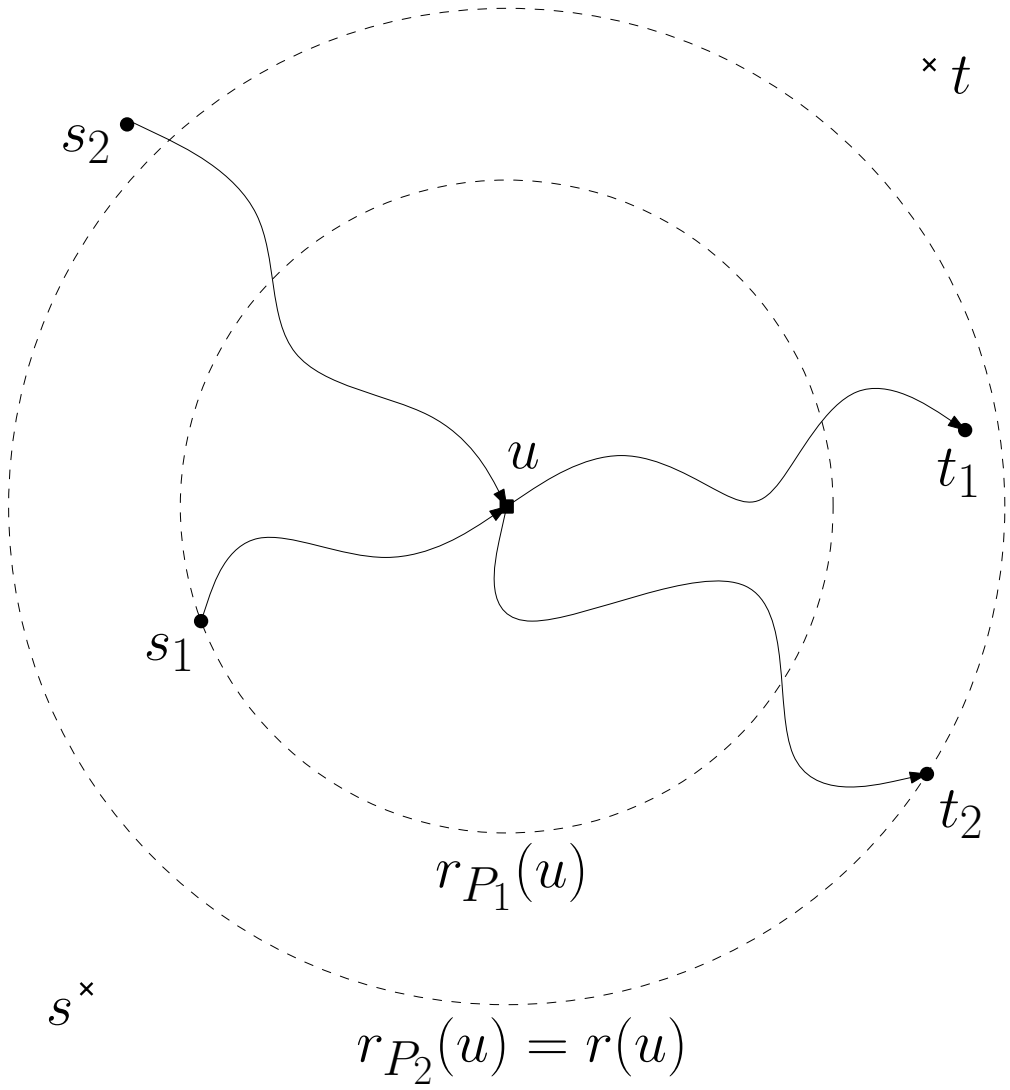
- Reach von u:
  Maximum seiner Reachwerte bezüglich aller kürzesten Pfade durch u:

$$r(u) := \max\{r_P(u) \mid P \text{ kürzester Weg mit } u \in P\}$$

Reach-Pruning

somit:

- Reach $r(u)$ von u gibt Suffix oder Prefix des längsten kürzesten Weges durch u
- wenn für u während Query $r(u) < d(s,u)$ und $r(u) < d(u,t)$ gilt, muss u nicht beachtet werden



Uni Reach Dijkstra - Pseudocode

ReachDijkstra($G = (V, E)$, s, t)

```
1  d[s] = 0
2  Q.clear(), Q.add(s, 0)
3  while !Q.empty() do
4      u ← Q.deleteMin()
5      if r(u) < d[u] and r(u) < d(u, t) then continue
6      forall edges e = (u, v) ∈ E do
7          if d[u] + len(e) < d[v] then
8              d[v] ← d[u] + len(e)
9              if v ∈ Q then Q.decreaseKey(v, d[v])
10             else Q.insert(v, d[v])
```

Problem

Problem:

- Abfrage $r(u) < d(u,t)$

- Im geometrischen Fall ist die überprüfung einfach
- benutze Landmarken
- weiteres?

Bidir. Self-Bounding Reach-Dijkstra

Idee (für Vorwärtssuche):

- ignoriere knoten u wenn $d[u] > r(u)$ gilt
- überlasse den Check $d(u,t) > r(u)$ der Rückwärtssuche
- Rückwärtssuche analog (umgekehrt)
- ändere das Stoppkriterium

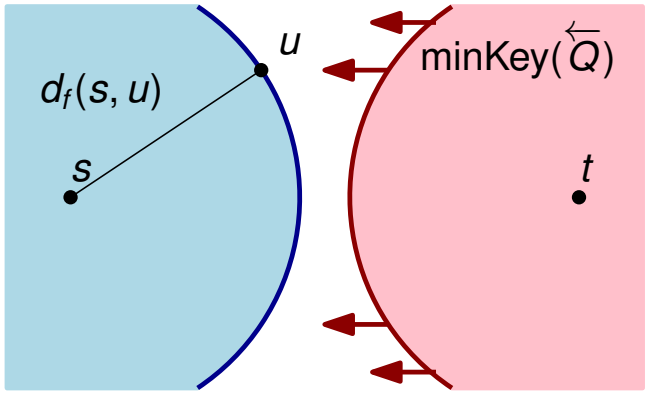
neues Stoppkriterium:

- stoppe Suche in eine Richtung wenn Queue leer oder es gilt: $\text{minKey}(Q) > \mu/2$
- stoppe Anfrage, wenn beide Suchrichtungen gestoppt haben
- Korrektheit gute Fingerübung

Bidir. Dist.-Bounding Reach-Dijkstra

Idee (für Vorwärtssuche):

- wenn u von Rückwärtssuche noch nicht erreicht, ist minKey($\overleftarrow{Q}$) eine untere Schranke für $d(u,t)$
- wenn u von Rückwärtssuche abgearbeitet, $d(u,t)$ bekannt



somit:

- ignoriere u, wenn $r(u) < d_f[u]$ und $r(u) < \min\{d_b[u], \text{minKey}(\overleftarrow{Q})\}$
- Stoppkriterium bleibt erhalten (also $\text{minKey}(\overrightarrow{Q}) + \text{minKey}(\overleftarrow{Q}) \geq \mu$)
- wenn als Alternierungsstrategie $\min\{\text{minKey}(\overrightarrow{Q}), \text{minKey}(\overleftarrow{Q})\}$ gewählt, gilt für Vorwärtssuche: $d_f[u] \leq \text{minKey}(\overleftarrow{Q})$

Optimierungen

mögliche Verbesserungen:

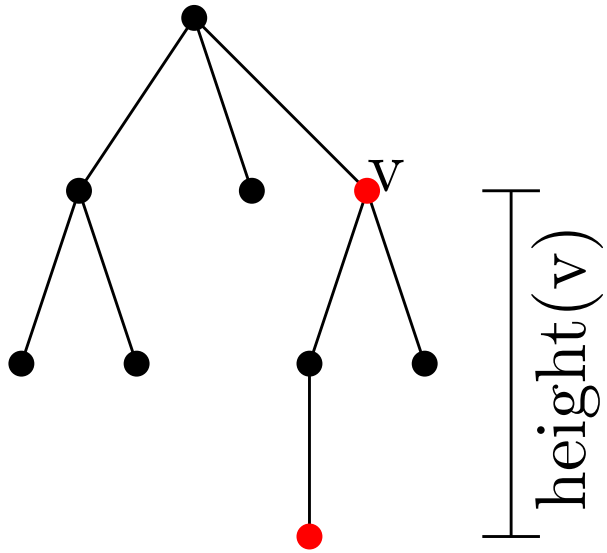
- Early (Kanten-)Pruning:
  (u, v) muss nicht relaxiert werden, wenn gilt:
  - $d[u] + \text{len}(u, v) > r(v)$
  - und $r(v) < \min\{d_b[u], \text{minKey}(\overleftarrow{Q})\}$

- Kanten sortieren:
  - sortiere ausgehende Kanten (u, v_i) absteigende nach $r(v_i)$
  - wenn Kante relaxiert wird mit $r(v_i) < \text{minKey}(\overleftarrow{Q})$ und $r(v_i) < d_f[u]$ müssen die restlichen Kanten ausgehend von u nicht relaxiert werden

Vorberechnung: Erster Ansatz

Wie kann man Reach-Werte vorberechnen?

- initialisieren $r(u) = 0$ für alle Knoten
- für jeden Knoten u
  - konstruiere kürzeste Wege-Baum
  - höhe von Knoten v: Abstand von v zum am weitesten entfernten Nachfolger
  - für jeden Knoten v:
    $r(v) = \max\{r(v), \min\{d(u,v), \text{height}(v)\}\}$



altes Problem:

- Vorberechnung basiert auf all-pair-shortest paths

Approximation

Beobachtung:

- es genügt, für jeden Knoten eine obere Schranke des Reach-Wertes zu haben

Problem:

- untere Schranken einfach zu finden:
  - breche Konstruktion der Bäume einfach bei bestimmter Größe ab
- aber: untere Schranken sind unbrauchbar
- Berechnung von oberen Schranken deutlich schwieriger
- möglich, aber sehr aufwendig
- nicht (mehr) Bestandteil der Vorlesung

Literatur

Literatur (Reach, RE, REAL):

- R. Gutman:
  Reach-based Routing: A New Approach to Shortest Path Algorithms Optimized for Road Networks
  In: *ALENEX, 2004*
- Andrew V. Goldberg and Haim Kaplan and Renato F. Werneck:
  Reach for A*: Shortest Path Algorithms with Preprocessing
  In: *Shortest Paths: Ninth DIMACS Implementation Challenge, 2009.*

Hierarchische Techniken

Ideensammlung:

- identifiziere wichtige Knoten mit Zentralitätsmaß
- überspringe unwichtige Teile des Graphen

Overlay Graph

Gegeben

- Eingabegraph $G = (V, E, \text{len})$
- Knotenmenge $V \supseteq V_L$

Berechne

- Berechne $G_L = (V_L, E_L, \text{len}_L)$, so dass Distanzen in G_L wie in G

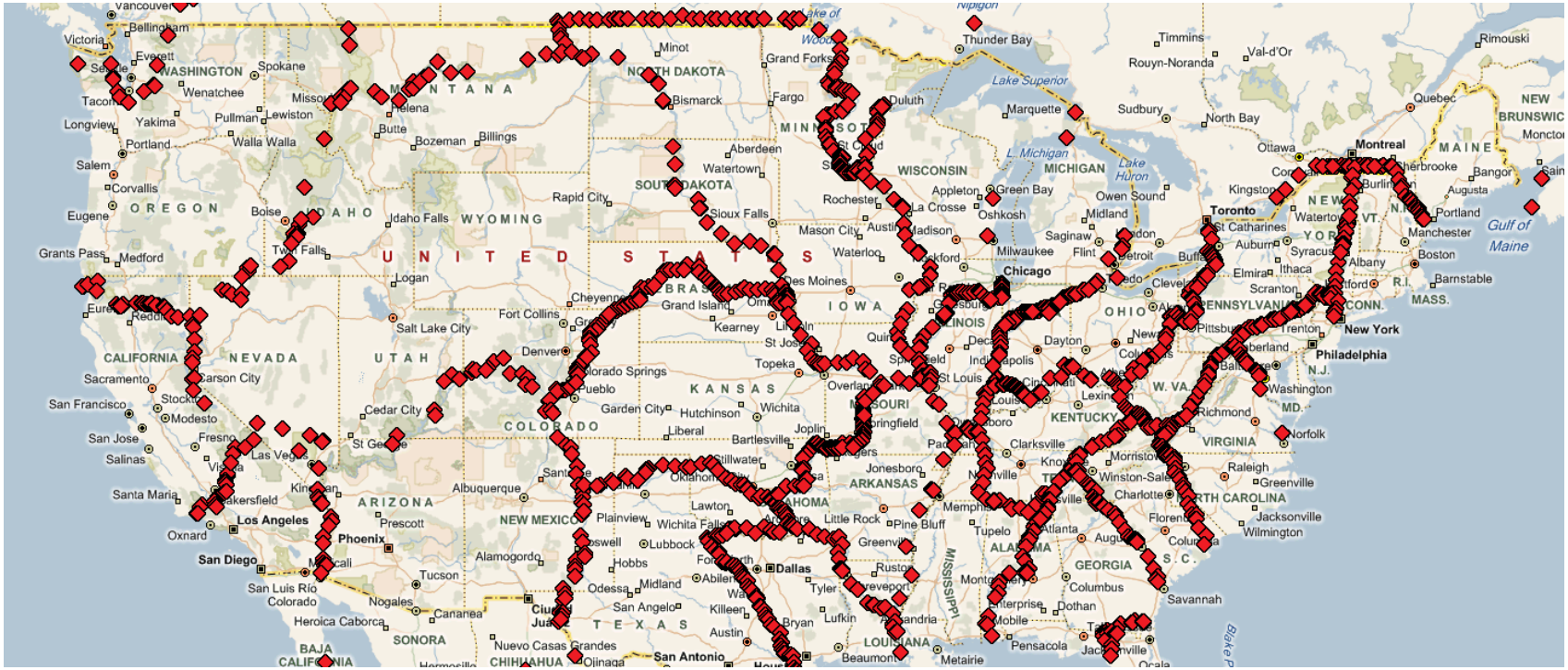
Wahl der Knotenmenge

Ideensammlung

- Randknoten
- (approximiertes) Zentralitätsmaß

Partitionierung

Letztes Mal: Strassengraphen haben natürliche Schnitte



- Jeder Pfad durch eine Zelle betritt/verlässt die Zelle durch einen Randknoten

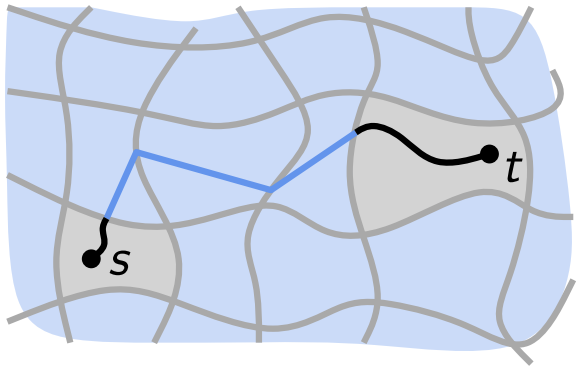
$\Rightarrow$ Minimiere # Schnittkanten mit Zellgrösse $\leq U$ (Eingabeparameter)

Ausnutzung der Partition

Idee: Berechne Distanzen zwischen Randknoten *in jeder Zelle*

Overlay Graph:

- Randknoten
- Cliquen in jeder Zelle
- Schnittkanten



Suchgraph:

- Start- und Zielzelle...
- ...plus Overlaygraph.
- (bidirektionaler) Dijkstra

Example

2^{15} Knoten pro Zelle, 626 Zellen $\Rightarrow$ 34 k Knoten im Overlaygraphen

Ergebnisse

Worst-Case:

- Kanten scans: $O(\sum \text{cliques} + 2 \cdot \text{cell size})$.
  Grösse des Overlaygraphen is metrikunabhängig

Beispiel:

	Metric Customization		Queries	
metric	time [s]	space [MB]	scans	time [ms]
Travel time	20	10	45134	10
Distance	20	10	47127	11

(partition: $\leq 2^{14}$ nodes/cell)

West Europa (18 M nodes, 42 M edges)
Intel Core-i7 920 (four cores at 2.67 GHz)

Diskussion

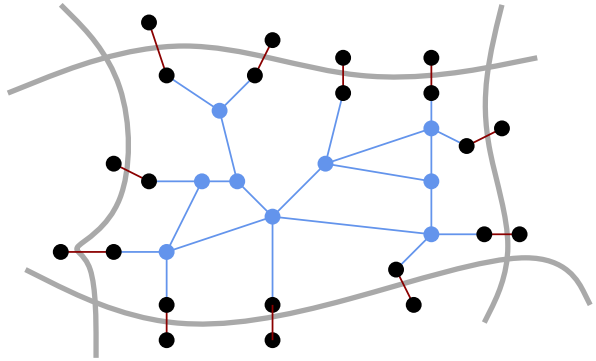
Verbesserungen?

- Ausdünnung der Graphen
- Kombination mit zielgerichteter Suche
- Multi-Level Partitionierung

Naheliegendes

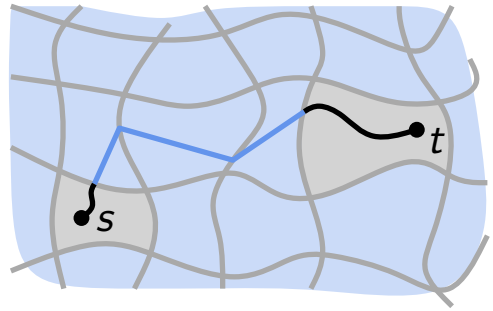
Ausdünnung des Overlaygraphen:

- entferne unnötige Kanten
- füge Knoten hinzu (aus Originalgraphen)
- etwas schnellere Anfragen



Kombination mit zielgerichteter Suche:

- nur auf (kleinem) Overlaygraphen
- ALT/Arc-Flags
- beschleunigt Anfragen, macht Vorberechnung und Queries komplizierter



Es geht besser (und einfacher!)

Multilevel-Overlay Graphs

Gegeben

- Eingabegraph $G = (V, E, \text{len})$
- Folge $V := V_0 \supseteq V_1 \supseteq \ldots \supseteq V_L$ von Teilmengen von V.

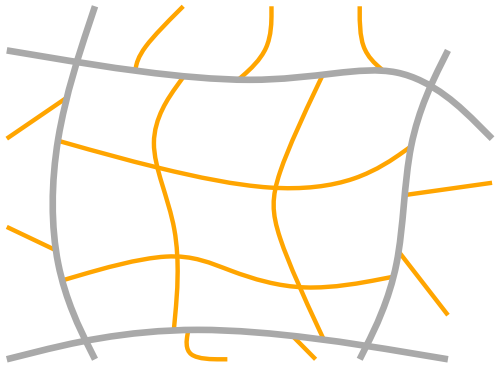
Berechne

- Folge $G_0 = (V_0, E_0, \text{len}_0), \ldots, G_L = (V_L, E_L, \text{len}_L)$ von Graphen, so dass Distanzen in G_i wie in G_0

Multi-Level Ansatz

Multi-Level Partition:

- benutze mehrere Partitionen (mit PUNCH)
- berechne Cliquen bottom-up
- benutze G_{i-1} um G_i zu bestimmen
- trade-off zwischen Platz und Suchzeiten



Suchgraph:

- Overlay auf obersten Level (G_L) ...
- und alle Ziel- und Startzellen (auf jedem Level)
- (bidirektionaler) Dijkstra

Phantom Level

Beobachtung: Viele Level $\Rightarrow$ schnellere Vorberechnung, mehr Platz

- Benutze mehr Level für Vorberechnung (32 Knoten pro Zelle)
- Speichere die unteren Level aber nicht dauerhaft
- Schnellere Vorberechnung
- Werden aber nicht für Queries benutzt

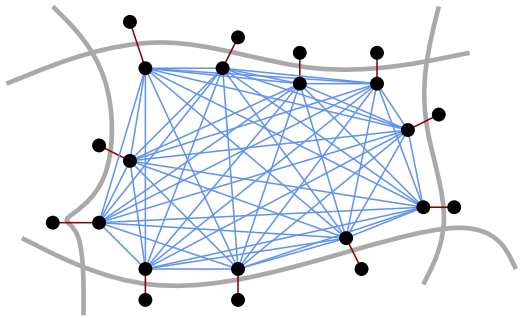
Cell Size	Cust Time
$[2^{14}]$	20.0 s
$[2^8 : 2^{14}]$	4.9 s

(metric: travel time)

Implementation

Repräsentation des Overlaygraphen:

- wirklich Graph-DS nötig?
- speicher Cliquen als Matrizen
- Beschleunigt Vorberechnung und Queries um einen Faktor 2
- Entkoppelung von Metrik und Topology
- reduziert Platzverbrauch: 32 Bit pro Cliquenkante (pro Metrik) + 1 x 32 Bit fuer Topology



3-Phasen Algorithmus

1. Metrik-unabhängige Vorberechnung

- partitionierung des Graphen
- bauen der Topology des Overlay Graphen
- linearer Platz für Partition, kann ein wenig dauern (nur einmal)
- PUNCH: 15-30 Minuten (mit aggressiven Parametern)

2. Metrik-abhängige Vorberechnung

- Berechnung der Länge der Matrix-Einträge
- Mit Hilfe von lokalen (hoch-parallelisierbaren) Dijkstra-Suchen
- Overhead pro Metrik gering (ein Array)

3. Queries

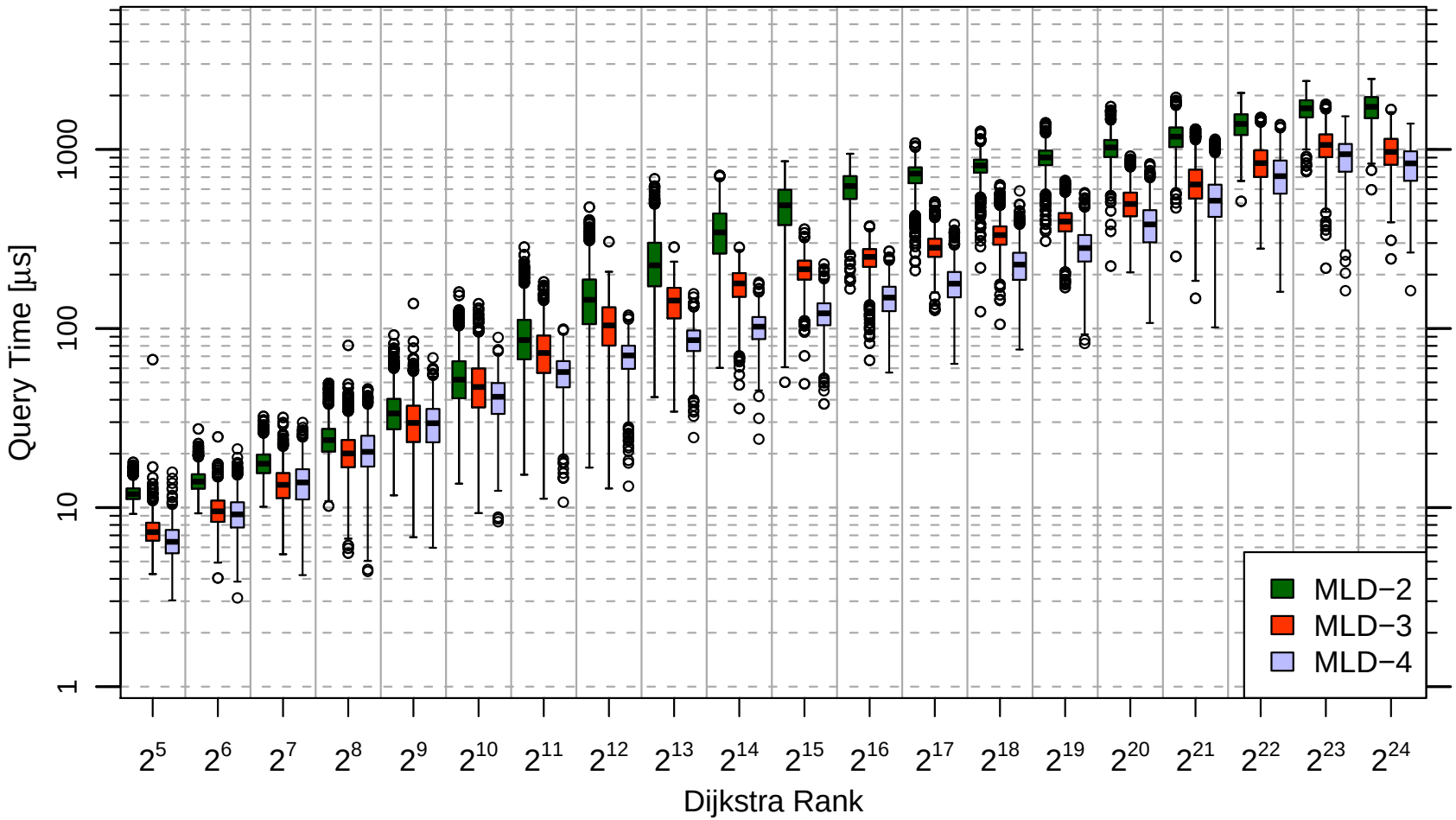
- Benutze Graph und beide Vorberechnungen
- (paralleler) bidirektionaler Dijkstra

Performance

	Algorithm	Customization time [s]	Customization space [MB]	Queries scans	Queries time [ms]
travel time	MLD-1 [2^{14}]	4.9	9.8	45134	5.67
	MLD-2 [$2^{12} : 2^{18}$]	5.0	18.4	12722	1.79
	MLD-3 [$2^{10} : 2^{15} : 2^{20}$]	5.2	32.3	6074	0.91
	MLD-4 [$2^{8} : 2^{12} : 2^{16} : 2^{20}$]	5.2	59.0	3897	0.71
distances	MLD-1 [2^{14}]	4.7	9.8	47127	6.19
	MLD-2 [$2^{12} : 2^{18}$]	4.9	18.4	13114	1.85
	MLD-3 [$2^{10} : 2^{15} : 2^{20}$]	5.1	32.3	6315	1.01
	MLD-4 [$2^{8} : 2^{12} : 2^{16} : 2^{20}$]	4.7	59.0	4102	0.77

MLD: fast customization / compact / real-time queries / robust

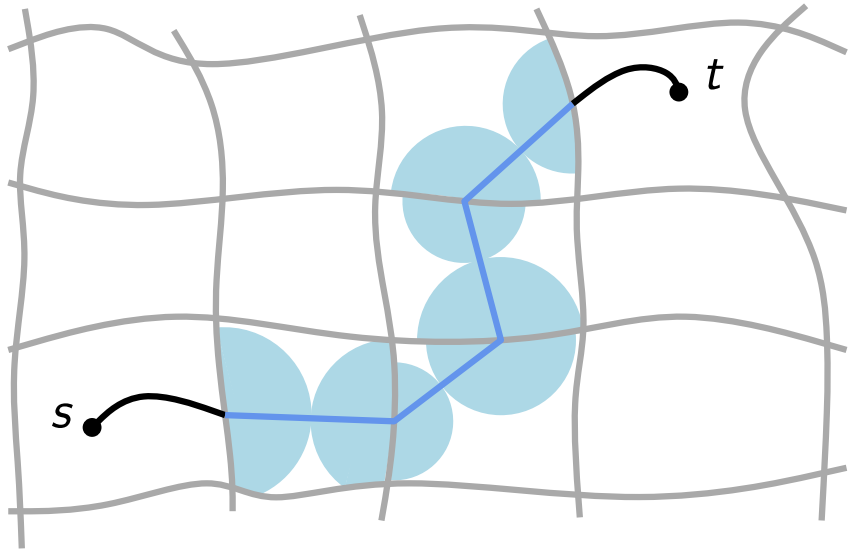
Local Queries



Entpacken der Shortcuts

Von Shortcuts zu vollen Pfaden:

- bidirektionale Suche
- beschraänkt auf die Zelle
- benutze untere Level rekursiv
- parallelisierbar
- benutze LRU-cache



Kaum zusätzlicher Speicher, 20-30% Zeit-Overhead.

Customizable Route Planning

Eigenschaften:

- viele Metriken
- schnelle lokale and globale updates
  Zelle in Millisekunden, gesamter Graph in Sekunden
- real-time queries (5000$\times$ Dijkstra)
- Bing Maps Routing Engine

Partition ist der Schlüssel.

Literatur

Literatur (Multilevel Overlay Graph):

- Frank Schulz, Dorothea Wagner, Christos Zaroliagis
  Using Multi-Level Graphs for Timetable Information in Railway Systems
  In : *Proceedings of the 4th Workshop on Algorithm Engineering and Experiments (ALENEX'02)*, 2002
- Peter Sanders, Dominik Schultes
  Dynamic Highway Node Routing
  In: *Proceedings of the 6th Workshop on Experimental Algorithms (WEA '07)*, 2007
- Daniel Delling, Andrew V. Goldberg, Thomas Pajor, Renato Werneck
  Customizable Route Planning
  In: *Proceedings of the 10th International Symposium on Experimental Algorithms (SEA '11)*, 2011

Nächste Termine

Karlsruher Institut für Technologie

Mittwoch, 16.5.2012

Montag, 21.5.2012 (Thomas)

Mittwoch, 30.5.2012 (Thomas)