

Algorithmen für Routenplanung

4. Vorlesung, Sommersemester 2014

Moritz Baum | 04. Mai 2015

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



Kürzeste Wege in Straßennetzwerken

Beschleunigungstechniken (Fortsetzung)

- Reach
- CRP

Bisher: bidirektionale Suche, zielgerichtete Suche (ALT, Arc-Flags)

Heute: Nutze “Hierarchie” im Graphen

in etwa: Seiten-, Haupt-, Land-, Bundesstraßen, Autobahnen

Ideensammlung:

Bisher: bidirektionale Suche, zielgerichtete Suche (ALT, Arc-Flags)

Heute: Nutze “Hierarchie” im Graphen

in etwa: Seiten-, Haupt-, Land-, Bundesstraßen, Autobahnen

Ideensammlung:

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

- Zentralitätsmaße bewerten Wichtigkeit von Knoten oder Kanten in einem Netzwerk
- Geläufige Beispiele
 - Google Page Rank
 - Erdős-Zahl

- degree centrality

$$C_D(v) = \frac{\deg(v)}{n-1} \in [0, 1]$$

- betweenness centrality

$$C_B(v) = \sum_{\substack{s \neq v \neq t \in V \\ s \neq t}} \frac{\sigma_{st}(v)}{\sigma_{st}} \in [0, 1]$$

wobei

- σ_{st} die Anzahl der kürzesten s - t -Wege ist (meistens 1)
- $\sigma_{st}(v)$ die Anzahl solcher kürzesten Wege, die durch v gehen

Geeignetes Maß für Routenplanung?

Reach



Zentralitätsmaß, das groß ist, falls ein Knoten in der Mitte eines langen kürzesten Weges liegt.

Vorbereitung:

- Zeichne um jeden Knoten u einen Kreis mit Radius $r(u)$, so dass:
 - für jedes Paar s, t gilt: wenn u auf kürzestem s - t -Weg liegt, ist entweder s oder t im Kreis.

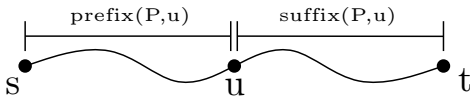
Zentralitätsmaß, das groß ist, falls ein Knoten in der Mitte eines langen kürzesten Weges liegt.

Vorbereitung:

- Zeichne um jeden Knoten u einen Kreis mit Radius $r(u)$, so dass:
 - für jedes Paar s, t gilt: wenn u auf kürzestem s - t -Weg liegt, ist entweder s oder t im Kreis.

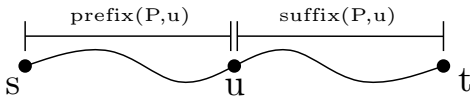
Strategie für Beschleunigungstechnik:

- Beachte Knoten u nicht, wenn s und t nicht im Kreis um u liegen.
- wie überprüfen?



Definition:

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

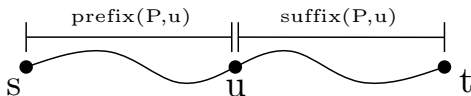


Definition:

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

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

Knoten, die zentral auf P liegen, haben hohen Reach.



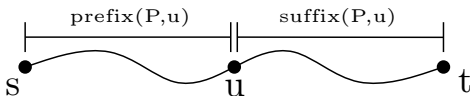
Definition:

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

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

Knoten, die zentral auf P liegen, haben hohen Reach.

- Reach von u :



Definition:

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

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

Knoten, die zentral auf P liegen, haben hohen Reach.

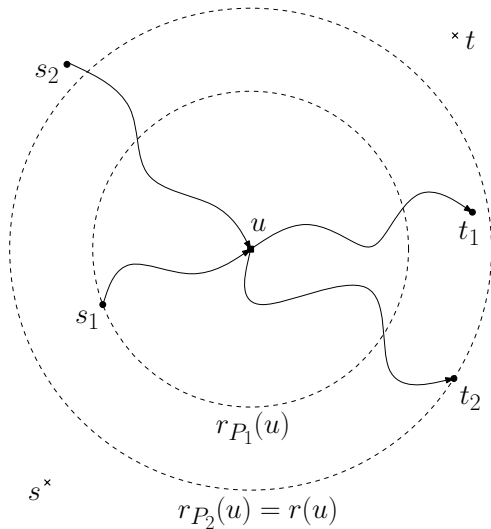
- Reach von u :
Maximum der Reachwerte bezüglich **aller** kürzesten Pfade durch u :

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

Knoten mit hohem Reach liegen zentral auf langen kürzesten Pfaden.

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



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

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

Problem

Problem?

Problem:

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

Problem:

- Abfrage $r(u) < d(u, t)$
- Im geometrischen Fall ist die Überprüfung einfach
- Auch möglich: benutze Landmarken
- Weiteres?

Bidir. Self-Bounding Reach-Dijkstra

Idee: Bidirektionale Suche

$$r(u) < d(s, u) \quad \rightsquigarrow \quad r(u) < d_f[u]$$

$$r(u) < d(u, t) \quad \rightsquigarrow \quad r(u) < d_b[u]$$

Vorwärtssuche:

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

Bidir. Self-Bounding Reach-Dijkstra

Idee: Bidirektionale Suche

$$r(u) < d(s, u) \rightsquigarrow r(u) < d_f[u]$$

$$r(u) < d(u, t) \rightsquigarrow r(u) < d_b[u]$$

Vorwärtssuche:

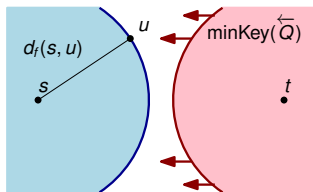
- ignoriere knoten u , wenn $r(u) < d_f[u]$ gilt
- überlasse den Check $r(u) < d(u, t)$ der Rückwärtssuche
- Rückwärtssuche analog
- ändere das Stoppkriterium

neues Stoppkriterium:

- stoppe eine Suchrichtung, wenn $\min\text{Key}(Q) > \mu/2$ oder Queue leer
- stoppe Anfrage, wenn **beide** Suchrichtungen gestoppt haben
- Korrektheit: gute Fingerübung

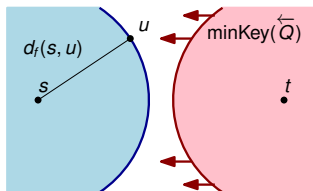
Idee (für Vorwärtssuche):

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



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, ist $d(u, t)$ bekannt

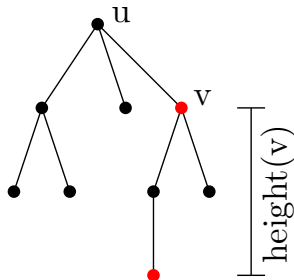


somit:

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

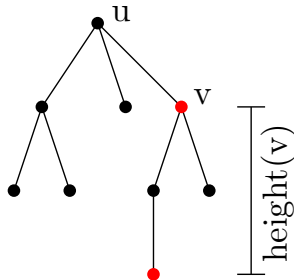
Wie kann man Reach-Werte vorberechnen?

- 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 :
 $r(v) = \max\{r(v), \min\{d(u, v), \text{height}(v)\}\}$



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:

- Vorbereitung basiert auf all-pair-shortest paths

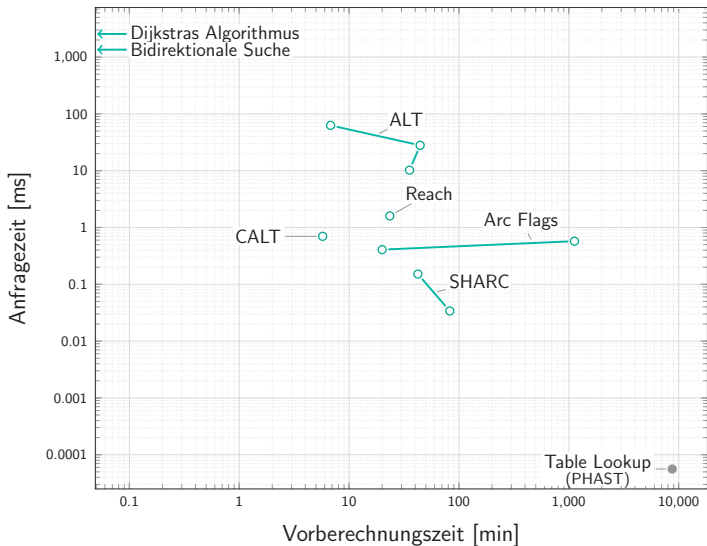
Beobachtung:

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

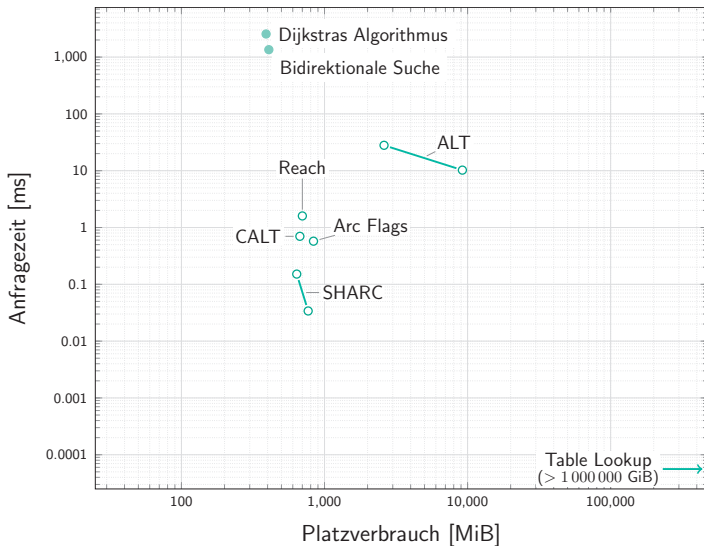
Problem:

- 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

Übersicht bisherige Techniken



Übersicht bisherige Techniken



CRP



Ideensammlung:

- Identifiziere wichtige Knoten mit Zentralitätsmaß
- Überspringe unwichtige Teile des Graphen

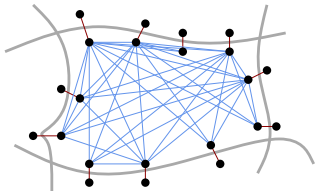
Ideensammlung:

- Identifiziere wichtige Knoten mit Zentralitätsmaß
- Überspringe unwichtige Teile des Graphen

(Multi-Level) Overlays [SWZ02, HSW08]

Beobachtung: Viele (lange) Routen in Teilen identisch

Idee: Berechne Teillösungen vor

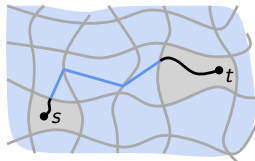


Overlay-Graph:

- Wähle **wichtige Knoten** (Separatoren, Pfadüberdeckung, heuristisch)
- Berechne **Shortcut-Kanten**:
 - Überspringen unwichtige Knoten
 - Erhalten Distanz

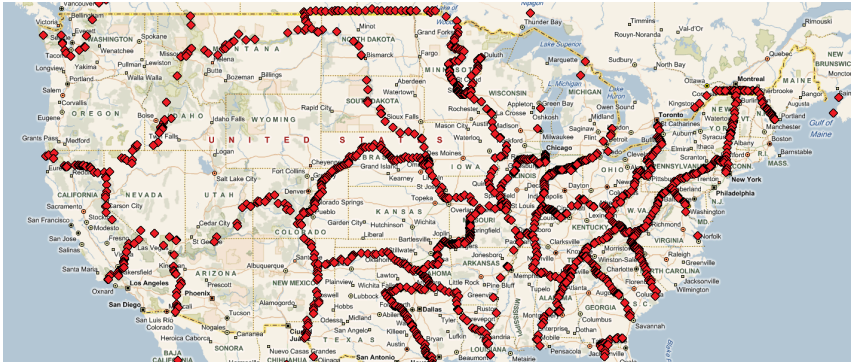
Anfragen:

- **Multi-Level Dijkstra**-Variante
- Ignoriert Kanten zu **weniger** wichtigen Knoten



Analog: Hierarchie mehrerer Level immer wichtigerer Knoten

Beobachtung: Strassengraphen haben dünne, natürliche Schnitte



- Jeder Pfad durch eine Zelle betritt/verlässt die Zelle durch einen Randknoten
- ⇒ Minimiere # Schnittkanten mit Zellgröße $\leq U$ (Eingabeparameter)

Ausnutzung der Partition

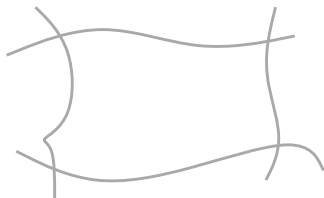
Idee: Berechne Distanzen zwischen Randknoten *in jeder Zelle*

Ausnutzung der Partition

Idee: Berechne Distanzen zwischen Randknoten *in jeder Zelle*

Overlay Graph:

- Randknoten
- Cliques in jeder Zelle
- Schnittkanten

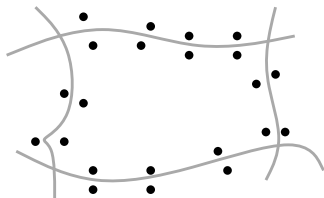


Ausnutzung der Partition

Idee: Berechne Distanzen zwischen Randknoten *in jeder Zelle*

Overlay Graph:

- Randknoten
- Cliques in jeder Zelle
- Schnittkanten

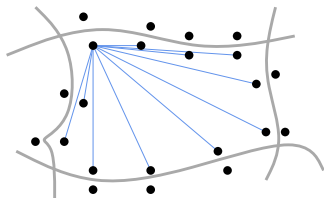


Ausnutzung der Partition

Idee: Berechne Distanzen zwischen Randknoten *in jeder Zelle*

Overlay Graph:

- Randknoten
- Cliques in jeder Zelle
- Schnittkanten

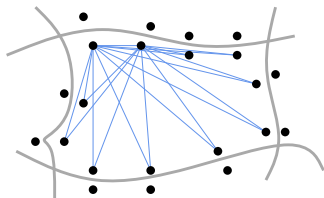


Ausnutzung der Partition

Idee: Berechne Distanzen zwischen Randknoten *in jeder Zelle*

Overlay Graph:

- Randknoten
- Cliques in jeder Zelle
- Schnittkanten

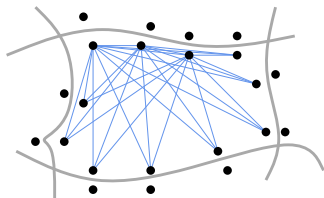


Ausnutzung der Partition

Idee: Berechne Distanzen zwischen Randknoten *in jeder Zelle*

Overlay Graph:

- Randknoten
- Cliques in jeder Zelle
- Schnittkanten

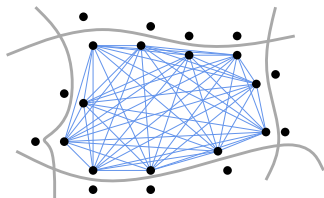


Ausnutzung der Partition

Idee: Berechne Distanzen zwischen Randknoten *in jeder Zelle*

Overlay Graph:

- Randknoten
- Cliques in jeder Zelle
- Schnittkanten

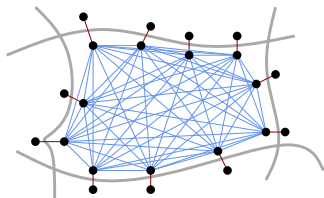


Ausnutzung der Partition

Idee: Berechne Distanzen zwischen Randknoten *in jeder Zelle*

Overlay Graph:

- Randknoten
- Cliques in jeder Zelle
- Schnittkanten

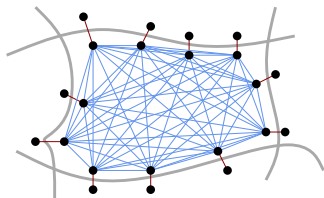


Ausnutzung der Partition

Idee: Berechne Distanzen zwischen Randknoten *in jeder Zelle*

Overlay Graph:

- Randknoten
- Cliques in jeder Zelle
- Schnittkanten



Suchgraph:

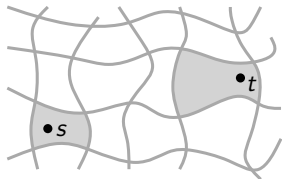
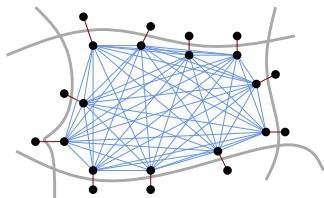
- Start- und Zielzelle...
- ...plus Overlaygraph.
- (bidirektionaler) Dijkstra

Ausnutzung der Partition

Idee: Berechne Distanzen zwischen Randknoten *in jeder Zelle*

Overlay Graph:

- Randknoten
- Cliques in jeder Zelle
- Schnittkanten



Suchgraph:

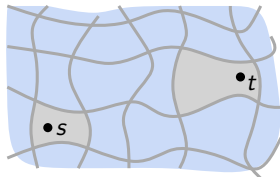
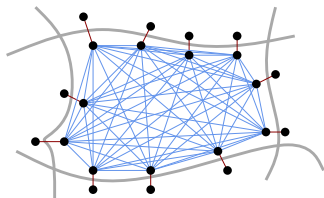
- Start- und Zielzelle...
- ...plus Overlaygraph.
- (bidirektionaler) Dijkstra

Ausnutzung der Partition

Idee: Berechne Distanzen zwischen Randknoten *in jeder Zelle*

Overlay Graph:

- Randknoten
- Cliques in jeder Zelle
- Schnittkanten



Suchgraph:

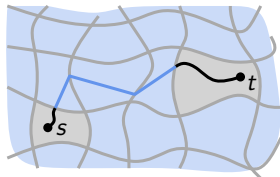
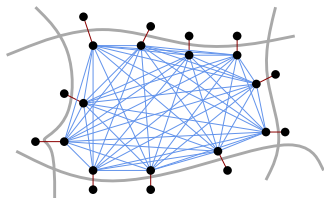
- Start- und Zielzelle...
- ...plus Overlaygraph.
- (bidirektionaler) Dijkstra

Ausnutzung der Partition

Idee: Berechne Distanzen zwischen Randknoten *in jeder Zelle*

Overlay Graph:

- Randknoten
- Cliques in jeder Zelle
- Schnittkanten



Suchgraph:

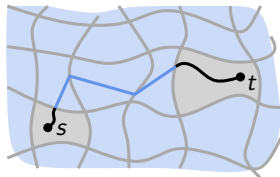
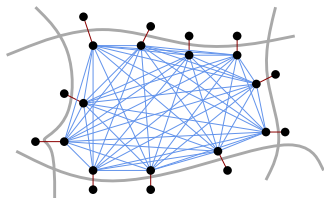
- Start- und Zielzelle...
- ...plus Overlaygraph.
- (bidirektionaler) Dijkstra

Ausnutzung der Partition

Idee: Berechne Distanzen zwischen Randknoten *in jeder Zelle*

Overlay Graph:

- Randknoten
- Cliques in jeder Zelle
- Schnittkanten



Suchgraph:

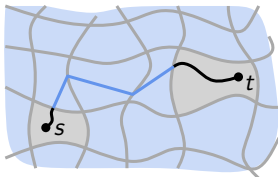
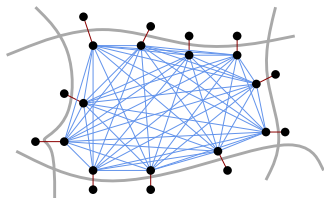
- Start- und Zielzelle...
- ...plus Overlaygraph.
- (bidirektionaler) Dijkstra

Ausnutzung der Partition

Idee: Berechne Distanzen zwischen Randknoten *in jeder Zelle*

Overlay Graph:

- Randknoten
- Cliques in jeder Zelle
- Schnittkanten



Suchgraph:

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

Beispiel: Europa (18 M. Knoten)

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

Worst-Case:

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

Worst-Case:

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

Beispiel:

metric	Metric Customization		Queries	
	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 Knoten, 42 M Kanten)
Intel Core-i7 920 (four cores at 2.67 GHz)

Worst-Case:

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

Beispiel:

metric	Metric Customization		Queries	
	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 Knoten, 42 M Kanten)
Intel Core-i7 920 (four cores at 2.67 GHz)

Diskussion: Unterschied zu ArcFlags?

Worst-Case:

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

Beispiel:

metric	Metric Customization		Queries	
	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 Knoten, 42 M Kanten)
Intel Core-i7 920 (four cores at 2.67 GHz)

Unterschied zu ArcFlags:

Vorberechnung außerhalb der Zelle vs. eingeschränkt auf Zelle

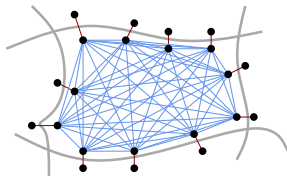
Verbesserungen?

Verbesserungen?

- Ausdünnung der Graphen (unnötig viele Kanten in Cliques?)
- Kombination mit zielgerichteter Suche
- Multi-Level Partitionierung

Naheliegenderes

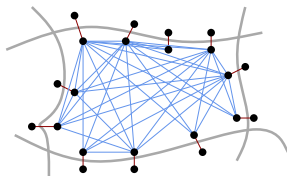
Ausdünnung des Overlaygraphen:



Naheliegendes

Ausdünnung des Overlaygraphen:

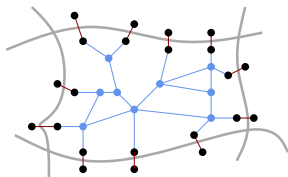
- entferne unnötige Kanten



Naheliegenderes

Ausdünnung des Overlaygraphen:

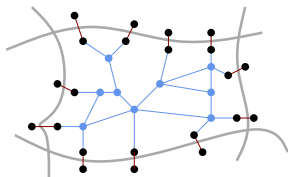
- entferne unnötige Kanten
- füge Knoten hinzu (aus Originalgraphen)



Naheliegenderes

Ausdünnung des Overlaygraphen:

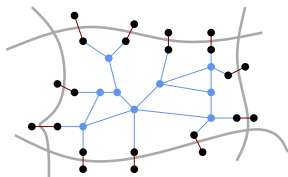
- entferne unnötige Kanten
- füge Knoten hinzu (aus Originalgraphen)
- etwas schnellere Anfragen



Naheliegenderes

Ausdünnung des Overlaygraphen:

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

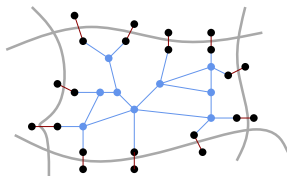


Kombination mit zielgerichteter Suche:

Naheliegendes

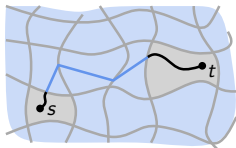
Ausdünnung des Overlaygraphen:

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



Kombination mit zielgerichteter Suche:

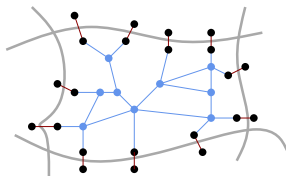
- nur auf (kleinem) Overlaygraphen
- ALT/Arc-Flags
- beschleunigt Anfragen, macht Vorberechnung und Queries komplizierter



Naheliegendes

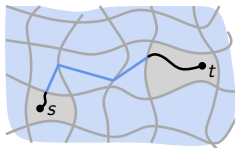
Ausdünnung des Overlaygraphen:

- 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!)

Gegeben

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

Berechne

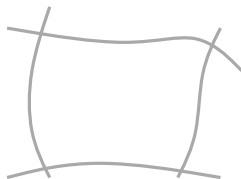
- Folge $G_0 = (V_0, E_0, \text{len}_0), \dots, G_L = (V_L, E_L, \text{len}_L)$ von Graphen, so dass Distanzen in G_i wie in G_0

Multi-Level Partition:

- benutze mehrere, geschachtelte Partitionen

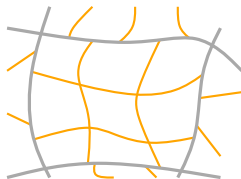
Multi-Level Partition:

- benutze mehrere, geschachtelte Partitionen



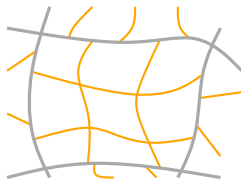
Multi-Level Partition:

- benutze mehrere, geschachtelte Partitionen



Multi-Level Partition:

- benutze mehrere, geschachtelte Partitionen
- berechne Cliques bottom-up
- benutze G_{i-1} um G_i zu bestimmen
- Trade-off zwischen Platz und Suchzeiten



Multi-Level Partition:

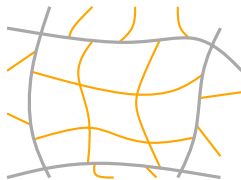
- benutze mehrere, geschachtelte Partitionen
- berechne Cliques bottom-up
- benutze G_{i-1} um G_i zu bestimmen
- Trade-off zwischen Platz und Suchzeiten



Suchgraph:

Multi-Level Partition:

- benutze mehrere, geschachtelte Partitionen
- berechne Cliques 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

Beobachtung: Viele Level \Rightarrow schnellere Vorberechnung, mehr Platz

Beobachtung: Viele Level \Rightarrow schnellere Vorberechnung, mehr Platz

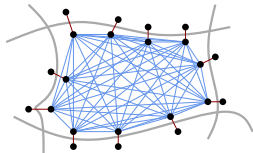
- benutze mehr Level für Vorberechnung (32 Knoten pro Zelle)
- speichere die unteren Level aber nicht dauerhaft
- schnellere Vorberechnung
- werden aber nicht für Queries benutzt

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

(metric: travel time)

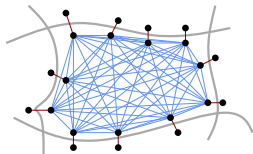
Repräsentation des Overlaygraphen:

- wirklich Graph-DS nötig?



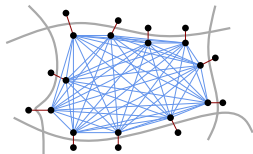
Repräsentation des Overlaygraphen:

- wirklich Graph-DS nötig?
- speicher Cliques als Matrizen
- beschleunigt Vorberechnung und Queries um einen Faktor 2
Schneller als ausgedünnter Overlay!



Repräsentation des Overlaygraphen:

- wirklich Graph-DS nötig?
- speicher Cliques als Matrizen
- beschleunigt Vorberechnung und Queries um einen Faktor 2
Schneller als ausgedünnter Overlay!
- Entkopplung von Metrik und Topologie
- reduziert Platzverbrauch pro Metrik!



3-Phasen Algorithmus

1. Metrik-unabhängige Vorberechnung

- Partitionierung des Graphen
- Erstellen der Topologie des Overlay Graphen
- Linearer Platz für Partition, kann ein wenig dauern (nur einmal)
- PUNCH: 15-30 Minuten

1. Metrik-unabhängige Vorberechnung

- Partitionierung des Graphen
- Erstellen der Topologie des Overlay Graphen
- Linearer Platz für Partition, kann ein wenig dauern (nur einmal)
- PUNCH: 15-30 Minuten

2. Metrik-abhängige Vorberechnung

- Berechnung der Gewichte der Matrix-Einträge
- mit Hilfe von lokalen (hoch-parallelisierbaren) Dijkstra-Suchen
- Overhead pro Metrik klein und speicher-lokal (ein Array)

1. Metrik-unabhängige Vorberechnung

- Partitionierung des Graphen
- Erstellen der Topologie des Overlay Graphen
- Linearer Platz für Partition, kann ein wenig dauern (nur einmal)
- PUNCH: 15-30 Minuten

2. Metrik-abhängige Vorberechnung

- Berechnung der Gewichte der Matrix-Einträge
- mit Hilfe von lokalen (hoch-parallelisierbaren) Dijkstra-Suchen
- Overhead pro Metrik klein und speicher-lokal (ein Array)

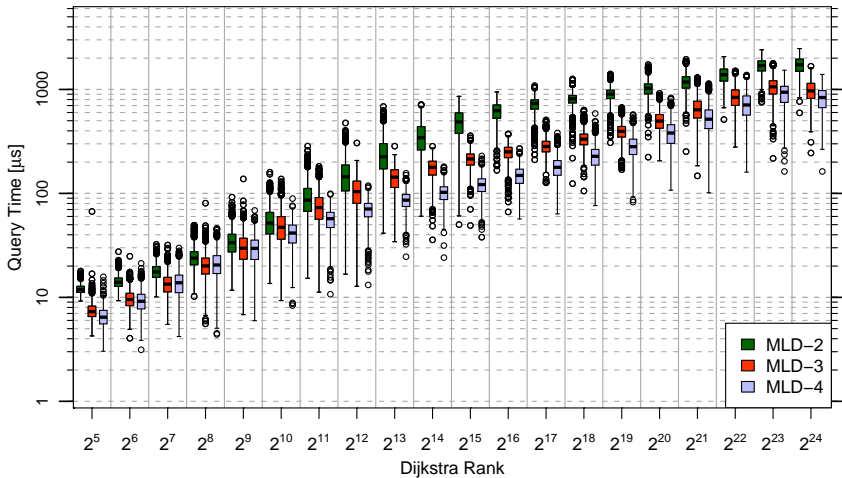
3. Queries

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

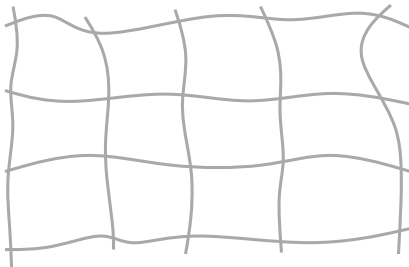
	Algorithm	Customization		Queries	
		time [s]	space [MB]	scans	time [ms]
travel time	MLD-1 [2^{14}]	4.9	9.8	45134	5.67
	MLD-2 [$2^{12} : 2^{18}$]	5.0	18.4	12722	1.79
	MLD-3 [$2^{10} : 2^{15} : 2^{20}$]	5.2	32.3	6074	0.91
	MLD-4 [$2^8 : 2^{12} : 2^{16} : 2^{20}$]	5.2	59.0	3897	0.71
distances	MLD-1 [2^{14}]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

	Algorithm	Customization		Queries	
		time [s]	space [MB]	scans	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

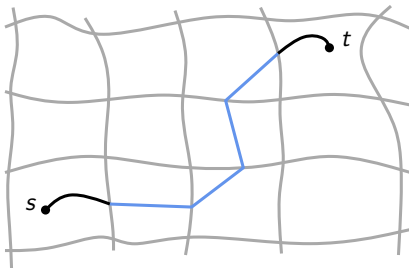


Von Shortcuts zu vollen Pfaden:

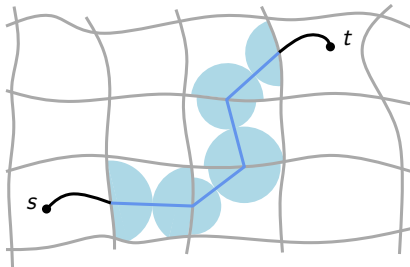


Entpacken der Shortcuts

Von Shortcuts zu vollen Pfaden:

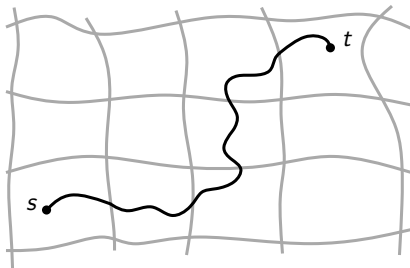


Von Shortcuts zu vollen Pfaden:



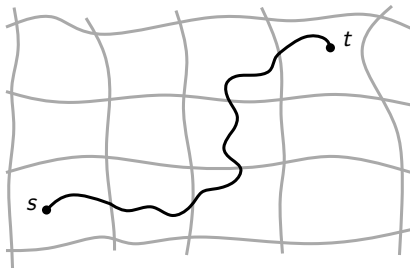
Von Shortcuts zu vollen Pfaden:

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



Von Shortcuts zu vollen Pfaden:

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



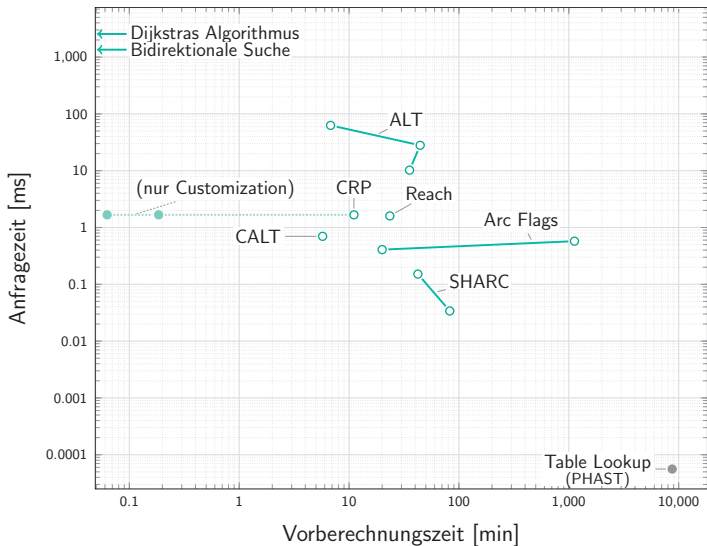
Kaum zusätzlicher Speicher, 20-30% Zeit-Overhead.

Eigenschaften:

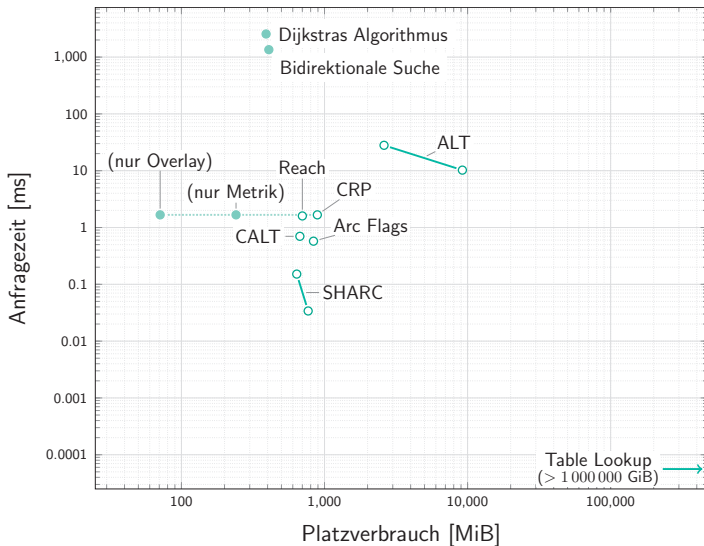
- viele Metriken
- schnelle lokale and globale Updates
Zelle in Millisekunden, gesamter Graph in Sekunden
- (hinreichend) schnelle Anfragen (1000× Dijkstra)
- Bing Maps Routing Engine

Gute Partition ist der Schlüssel!





Übersicht bisherige Techniken



Übersicht bisherige Techniken



Mittwoch, 6.5.2014

-  Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck.
Customizable route planning.
In Proceedings of the 10th International Symposium on Experimental Algorithms (SEA'11), volume 6630 of *Lecture Notes in Computer Science*, pages 376–387. Springer, 2011.
-  Andrew V. Goldberg, Haim Kaplan, and Renato F. Werneck.
Reach for A*: Shortest path algorithms with preprocessing.
In The Shortest Path Problem: Ninth DIMACS Implementation Challenge, volume 74 of *DIMACS Book*, pages 93–139. American Mathematical Society, 2009.
-  Ronald J. Gutman.
Reach-based routing: A new approach to shortest path algorithms optimized for road networks.
In Proceedings of the 6th Workshop on Algorithm Engineering and Experiments (ALENEX'04), pages 100–111. SIAM, 2004.
-  Martin Holzer, Frank Schulz, and Dorothea Wagner.
Engineering multilevel overlay graphs for shortest-path queries.
ACM Journal of Experimental Algorithmics, 13(2.5):1–26, December 2008.



Frank Schulz, Dorothea Wagner, and 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)*, volume 2409 of *Lecture Notes in Computer Science*, pages 43–59. Springer, 2002.