Karlsruher Institut für Technologie

Algorithmen für Routenplanung

5. Vorlesung, Sommersemester 2021

Adrian Feilhauer | 9. Mai 2022

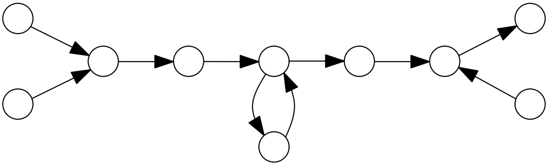
Kürzeste Wege in Straßennetzwerken

Beschleunigungstechniken (Fortsetzung)

- Shortcuts
- Bilevel-Overlay
- Multilevel-Overlay (MLO)
- Customizable Route Planning (CRP)

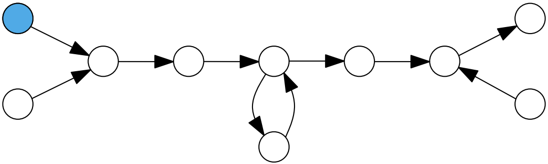
Shortcuts

Shortcut



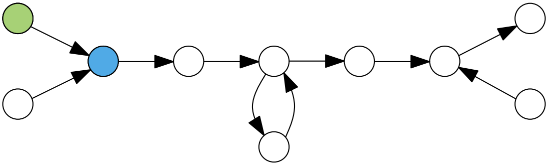
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.

Shortcut



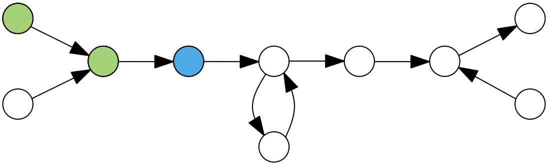
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.

Shortcut



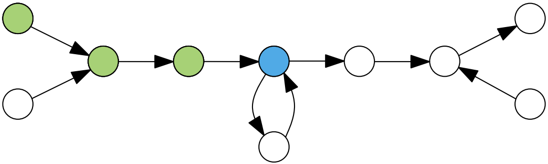
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.

Shortcut



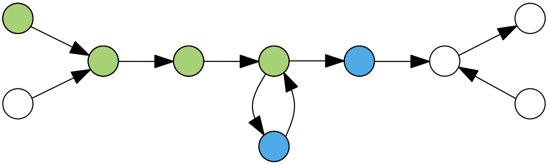
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.

Shortcut



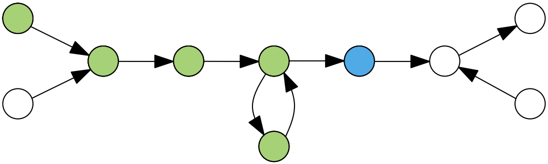
Dijkstra’s Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.

Shortcut



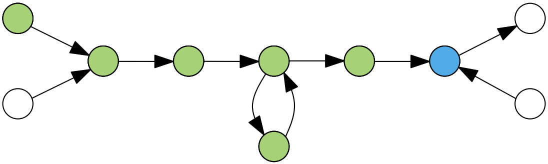
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.

Shortcut



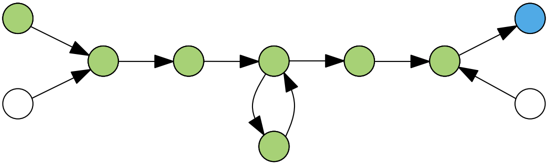
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.

Shortcut



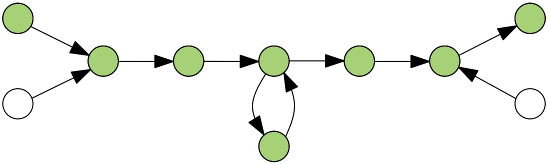
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.

Shortcut



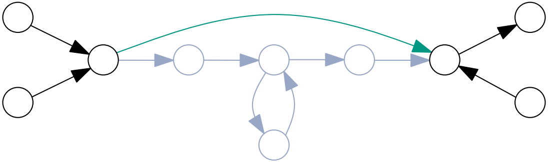
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.

Shortcut



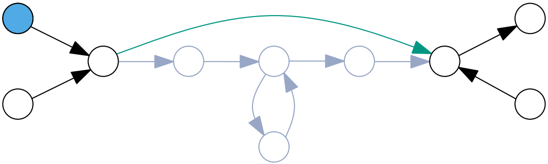
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.

Shortcut



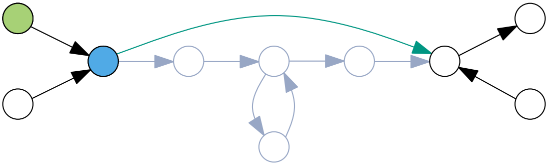
Idee: Füge Shortcut-Kante ein. Grauer Teilgraph muss nur angeschaut werden, wenn s oder t drin liegt.

Shortcut



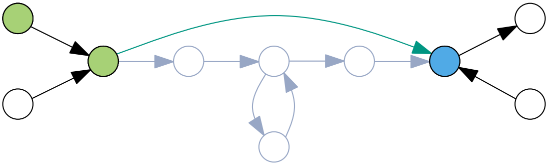
Idee: Füge Shortcut-Kante ein. Grauer Teilgraph muss nur angeschaut werden, wenn s oder t drin liegt.

Shortcut



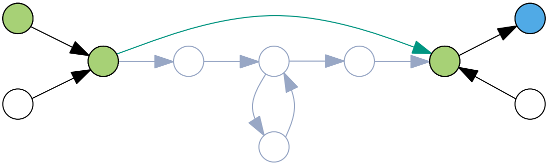
Idee: Füge Shortcut-Kante ein. Grauer Teilgraph muss nur angeschaut werden, wenn *s* oder *t* drin liegt.

Shortcut



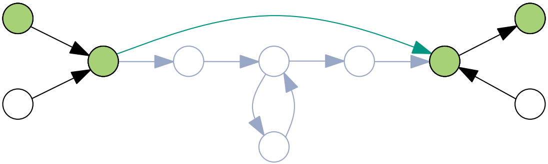
Idee: Füge Shortcut-Kante ein. Grauer Teilgraph muss nur angeschaut werden, wenn s oder t drin liegt.

Shortcut



Idee: Füge Shortcut-Kante ein. Grauer Teilgraph muss nur angeschaut werden, wenn s oder t drin liegt.

Shortcut



Idee: Füge Shortcut-Kante ein. Grauer Teilgraph muss nur angeschaut werden, wenn s oder t drin liegt.

Shortcut

- Shortcuts und bestehende Ideen können kombiniert werden:
  - ALT + Shortcuts = Core ALT (CALT)
  - ArcFlags + Shortcuts = SHARC
  - ...
- Wie weit kommen wir nur mit Shortcuts?

Bilevel-Overlays

Overlay

Karlsruher Institut für Technologie

Wiederholung:

- Ein Overlay eines Graphen $G = (V, E)$ ist ein Graph $G_O = (V_O, E_O)$, so dass:
  - $V_O \subseteq V$
  - Für alle s und t aus V_O gilt: Die st-Distanz in G ist gleich der st-Distanz in G_O

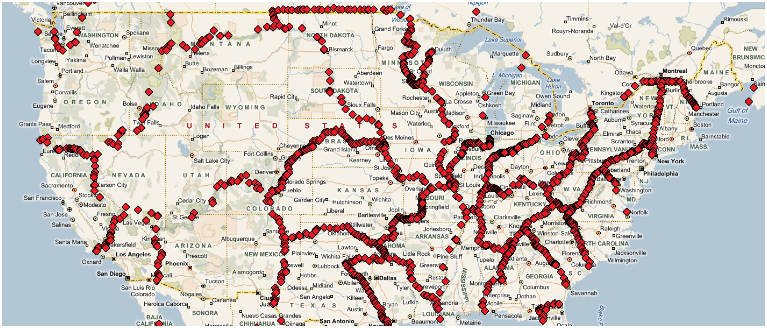
Neu:

- Berechne Overlay aus Partitionierung

Partitionierung

Karlsruher Institut für Technologie

Beobachtung: Straßengraphen haben dünne, natürliche Schnitte



- Jeder Pfad durch eine Zelle betritt/verlässt die Zelle durch einen Randknoten
- Wir wollen etwa gleich große Zellen mit wenig Kanten dazwischen
- Wie man diese Schnitte findet ist ein Thema für sich

Ausnutzung der Partition

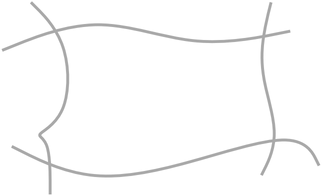
Idee: Berechne Distanzen zwischen Randknoten *in jeder Zelle*

Ausnutzung der Partition

Idee: Berechne Distanzen zwischen Randknoten *in jeder Zelle*

Overlay Graph:

- Randknoten
- Cliquen in jeder Zelle
  Gewicht eines Shortcuts entspricht
  kürzestem Weg durch die Zelle
- Schnittkanten

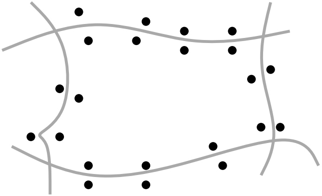


Ausnutzung der Partition

Idee: Berechne Distanzen zwischen Randknoten *in jeder Zelle*

Overlay Graph:

- Randknoten
- Cliquen in jeder Zelle
  Gewicht eines Shortcuts entspricht
  kürzestem Weg durch die Zelle
- Schnittkanten

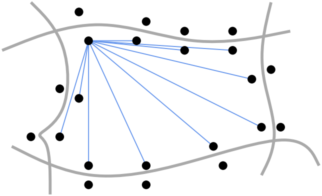


Ausnutzung der Partition

Idee: Berechne Distanzen zwischen Randknoten *in jeder Zelle*

Overlay Graph:

- Randknoten
- Cliquen in jeder Zelle
  Gewicht eines Shortcuts entspricht kürzestem Weg durch die Zelle
- Schnittkanten

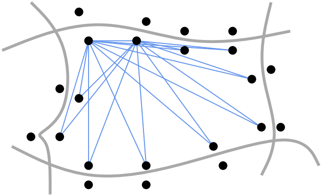


Ausnutzung der Partition

Idee: Berechne Distanzen zwischen Randknoten *in jeder Zelle*

Overlay Graph:

- Randknoten
- Cliquen in jeder Zelle
  Gewicht eines Shortcuts entspricht kürzestem Weg durch die Zelle
- Schnittkanten

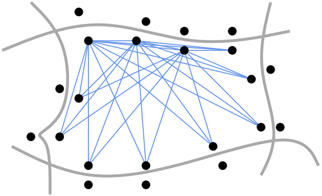


Ausnutzung der Partition

Idee: Berechne Distanzen zwischen Randknoten *in jeder Zelle*

Overlay Graph:

- Randknoten
- Cliquen in jeder Zelle
  Gewicht eines Shortcuts entspricht kürzestem Weg durch die Zelle
- Schnittkanten

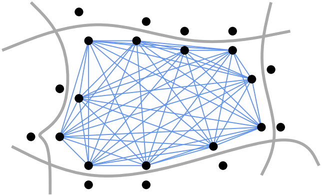


Ausnutzung der Partition

Idee: Berechne Distanzen zwischen Randknoten *in jeder Zelle*

Overlay Graph:

- Randknoten
- Cliquen in jeder Zelle
  Gewicht eines Shortcuts entspricht kürzestem Weg durch die Zelle
- Schnittkanten

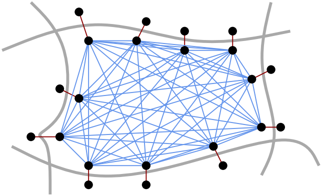


Ausnutzung der Partition

Idee: Berechne Distanzen zwischen Randknoten *in jeder Zelle*

Overlay Graph:

- Randknoten
- Cliquen in jeder Zelle
  Gewicht eines Shortcuts entspricht
  kürzestem Weg durch die Zelle
- Schnittkanten

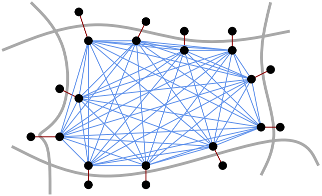


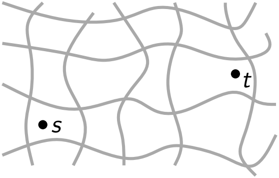
Ausnutzung der Partition

Idee: Berechne Distanzen zwischen Randknoten *in jeder Zelle*

Overlay Graph:

- Randknoten
- Cliquen in jeder Zelle
  Gewicht eines Shortcuts entspricht
  kürzestem Weg durch die Zelle
- Schnittkanten





Suchgraph:

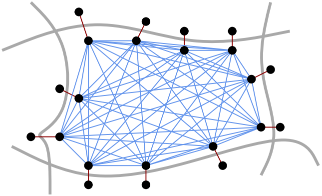
- Start- und Zielzelle...
- ...plus Overlaygraph.
- (bidirektionaler) Dijkstra

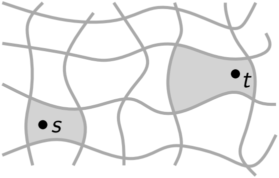
Ausnutzung der Partition

Idee: Berechne Distanzen zwischen Randknoten *in jeder Zelle*

Overlay Graph:

- Randknoten
- Cliquen in jeder Zelle
  Gewicht eines Shortcuts entspricht
  kürzestem Weg durch die Zelle
- Schnittkanten





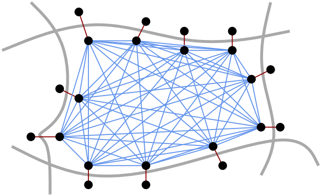
Suchgraph:

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

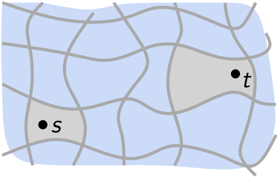
Ausnutzung der Partition

Idee: Berechne Distanzen zwischen Randknoten *in jeder Zelle*

Overlay Graph:



- Randknoten
- Cliquen in jeder Zelle
  Gewicht eines Shortcuts entspricht kürzestem Weg durch die Zelle
- Schnittkanten



Suchgraph:

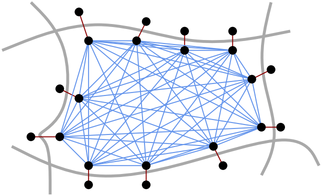
- Start- und Zielzelle...
- ...plus Overlaygraph.
- (bidirektionaler) Dijkstra

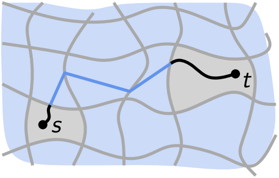
Ausnutzung der Partition

Idee: Berechne Distanzen zwischen Randknoten *in jeder Zelle*

Overlay Graph:

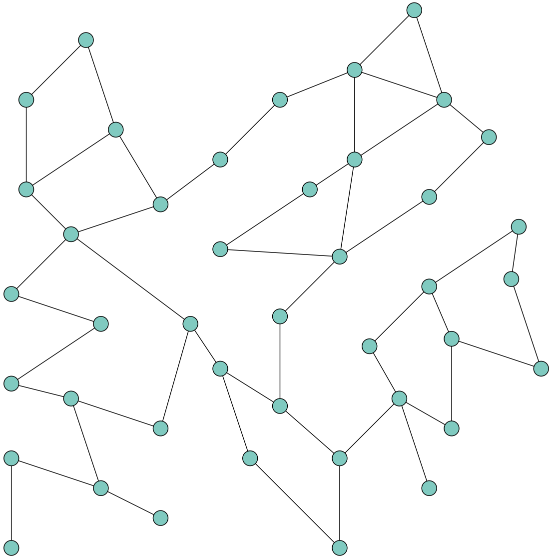
- Randknoten
- Cliquen in jeder Zelle
  Gewicht eines Shortcuts entspricht kürzestem Weg durch die Zelle
- Schnittkanten



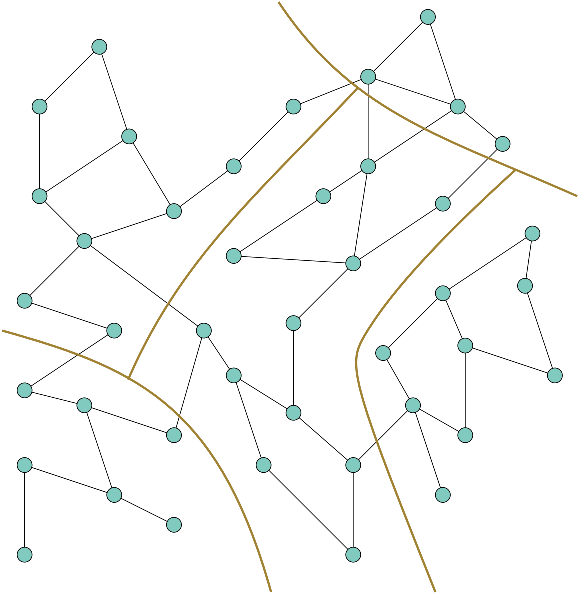


Suchgraph:

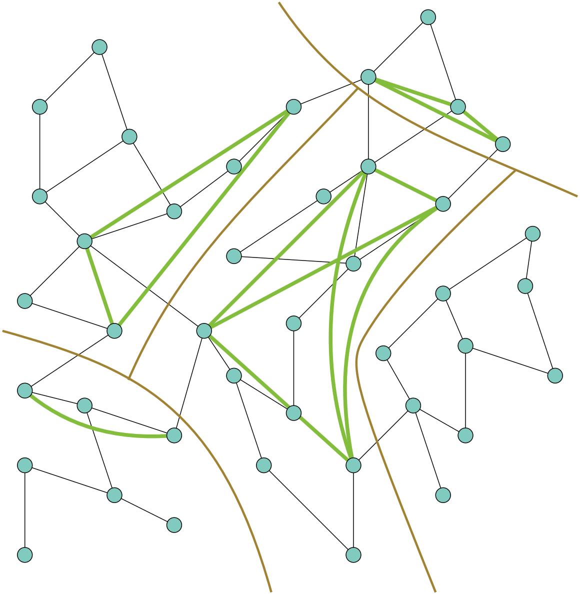
- Start- und Zielzelle...
- ...plus Overlaygraph.
- (bidirektionaler) Dijkstra



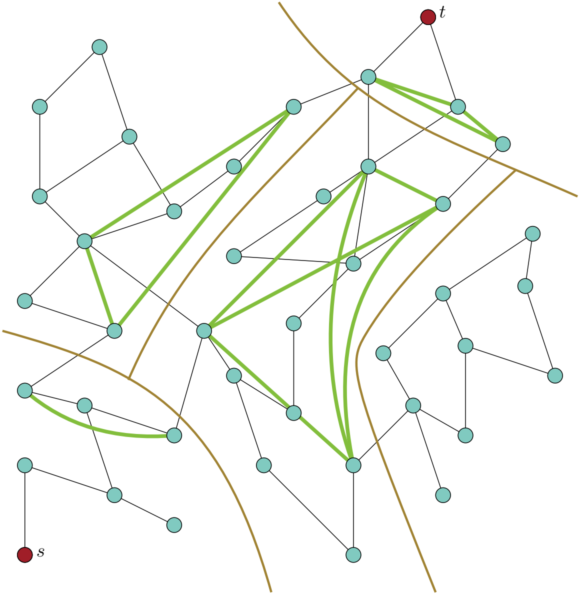
Ein Beispielgraph



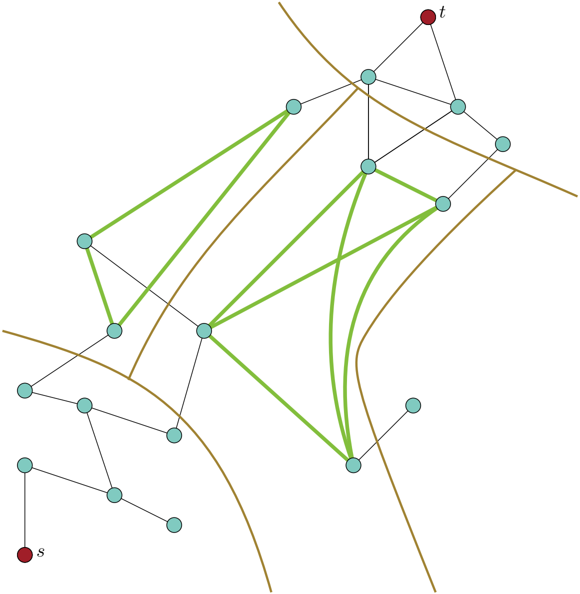
Eine Partition in 5 Zellen



Der Overlay



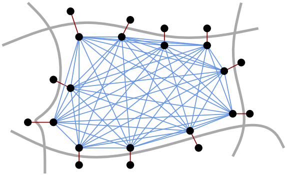
Eine Anfrage von s zu t



Die Knoten und Kanten, die während dieser Anfrage angeschaut werden

Varianten

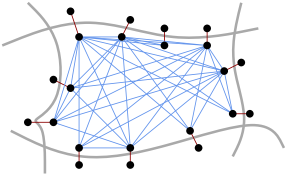
Ausdünnung des Overlaygraphen:



Varianten

Ausdünnung des Overlaygraphen:

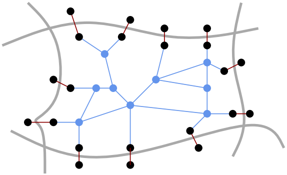
- entferne unnötige Kanten



Varianten

Ausdünnung des Overlaygraphen:

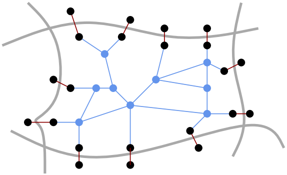
- entferne unnötige Kanten
- füge Knoten hinzu (aus Originalgraphen)



Varianten

Ausdünnung des Overlaygraphen:

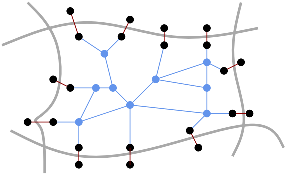
- entferne unnötige Kanten
- füge Knoten hinzu (aus Originalgraphen)
- etwas schnellere Anfragen



Varianten

Ausdünnung des Overlaygraphen:

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

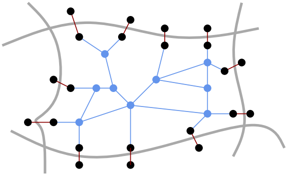


Kombination mit zielgerichteter Suche:

Varianten

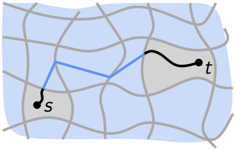
Ausdünnung des Overlaygraphen:

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



Kombination mit zielgerichteter Suche:

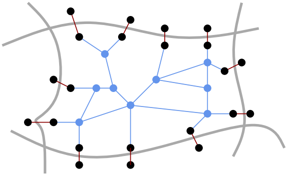
- nur auf (kleinem) Overlaygraphen
- ALT/Arc-Flags
- beschleunigt Anfragen, macht Vorberechnung und Queries komplizierter



Varianten

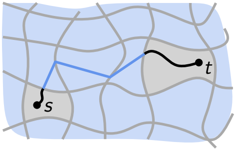
Ausdünnung des Overlaygraphen:

- 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



Aufwendig und bringt nicht so viel → wird meistens weggelassen

Multilevel-Overlays (MLO)

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

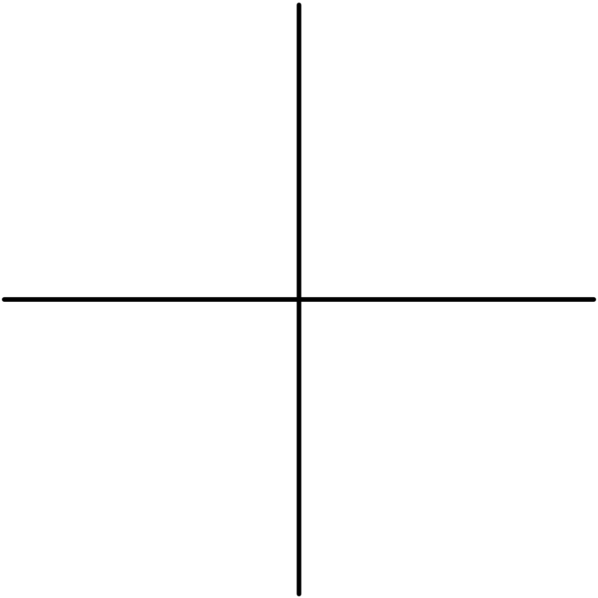
- Folge $G_0 = (V_0, E_0, \text{len}_0), \ldots, G_L = (V_L, E_L, \text{len}_L)$ von Graphen, so dass Distanzen in G_i wie in G_0
- G_i ist ein Overlay jedes G_j mit $j \leq i$

Idee

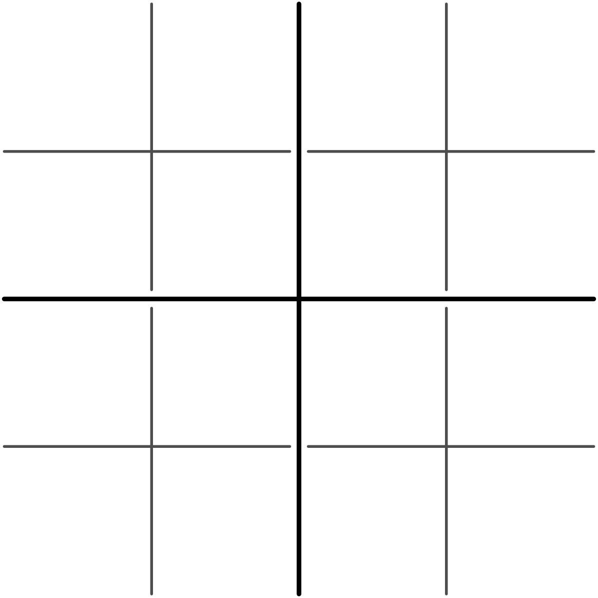
- Bidirektionale Suche
- Folge nur Kanten ins gleiche oder höhere Level

Partitionierung

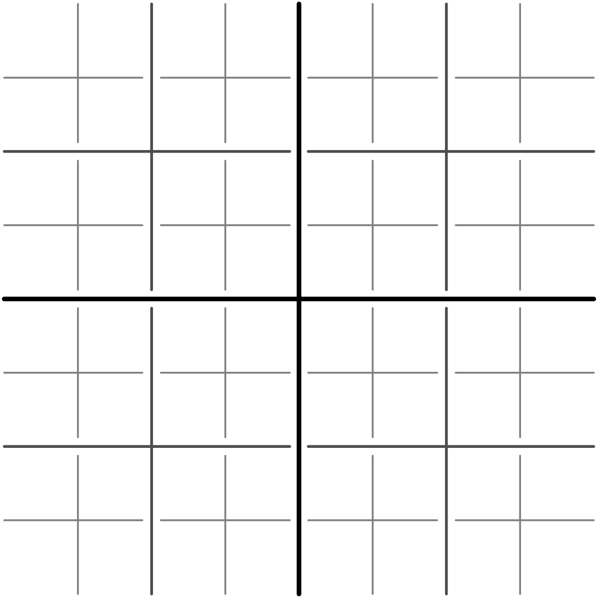
- Partitioniere Eingabegraph G
- Randknoten der Zellen bilden V_L
- Partitioniere jede Zelle in Subzellen
- Randknoten aller Subzellen bilden V_{L-1}
- ... wiederhole $L-1$ Mal



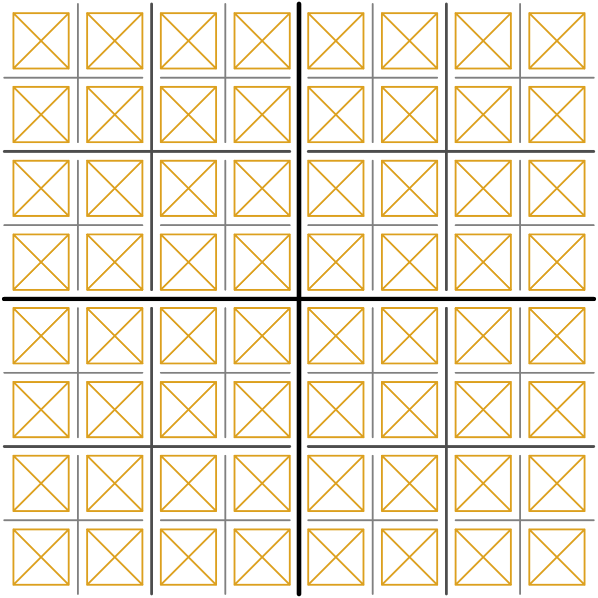
Zerteile den Graph in Zellen



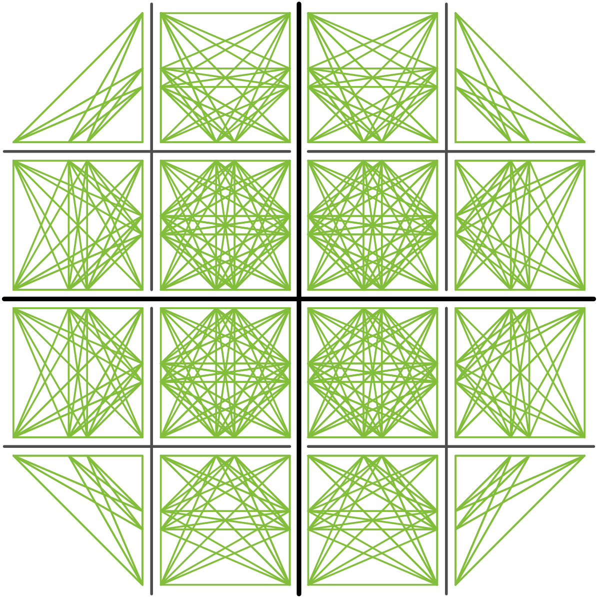
Zerteile die Zellen in Subzellen



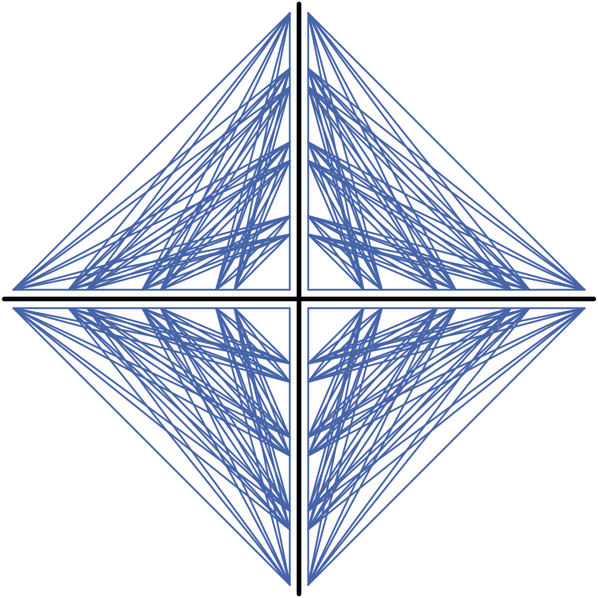
Zerteile die Subzellen in Subsubzellen



Baue für das unterste Level Overlays



Baue Overlays für das nächste höhere Level
Benutze dabei die Overlays des Levels darunter

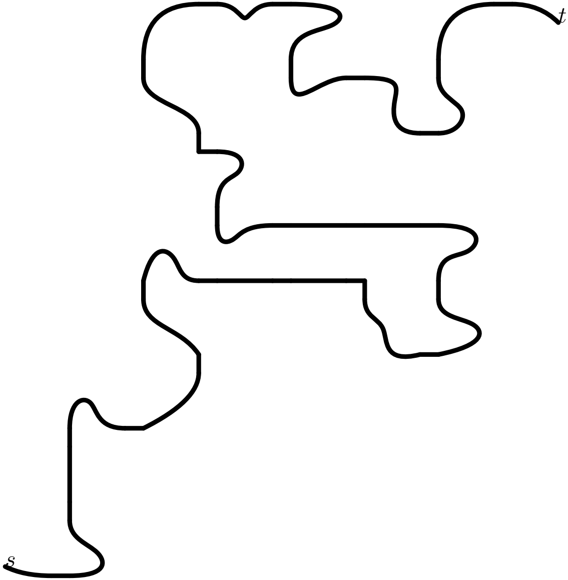


Baue Overlays für das höhste Level
Benutze dabei die Overlays des Levels darunter

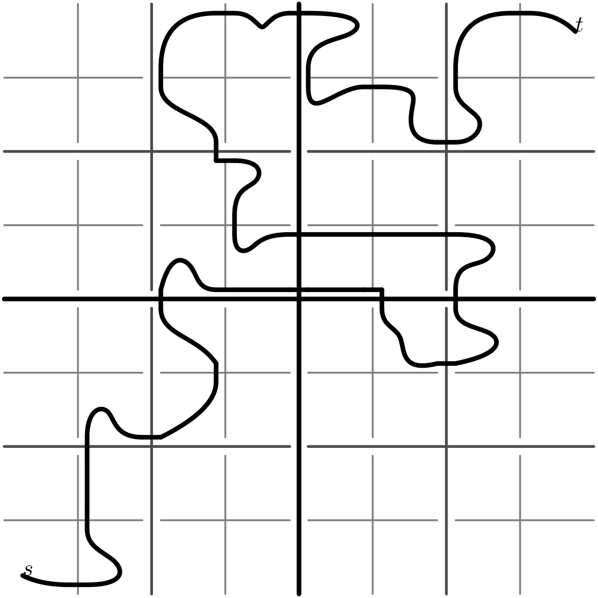
t

s

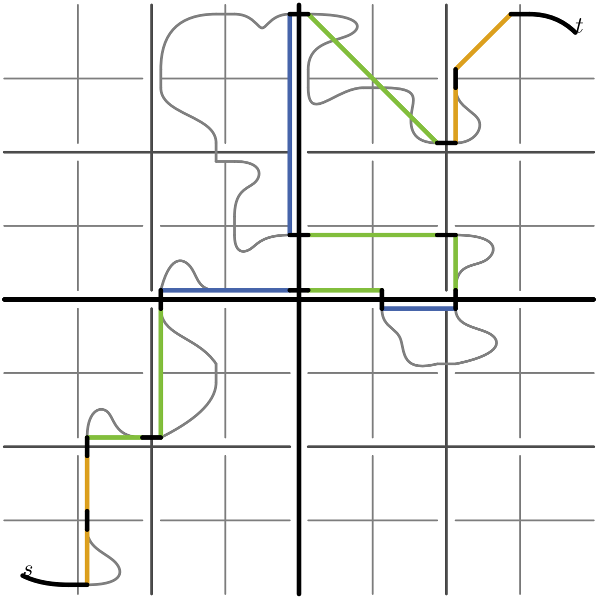
Betrachte eine *st*-Anfrage ...



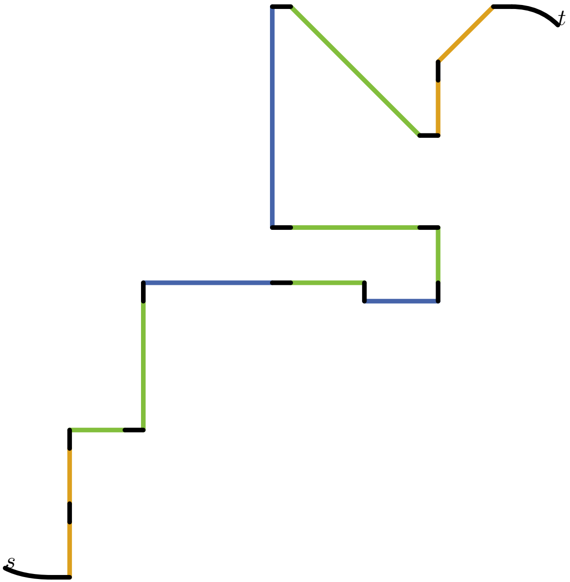
. . . und den dazu gehörigen kürzesten *st*-Pfad im Originalgraph
Achtung: Wir wissen nur, dass es ihn gibt. Wir kennen ihn noch nicht!



Betrachte nun die Partitionierung



Aus der Paritionierung können wir einen anderen gleich langen Pfad ablesen der von unserer Suche gefunden wird

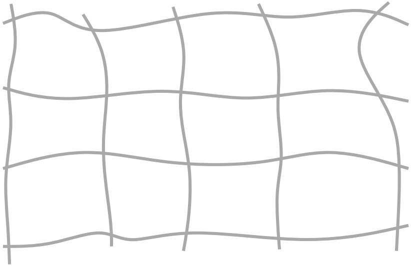


Farbe = Level der Kante: schwarz = Eingabe, orange = unterstes Level, grün = mittleres Level, blau = höchstes Level

Entpacken der Shortcuts

Von Shortcuts zu vollen Pfaden:

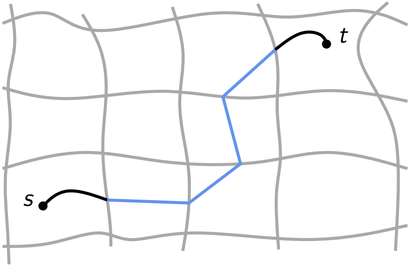
- bidirektionale Suche
- beschränkt auf die Zelle
- benutze untere Level rekursiv



Entpacken der Shortcuts

Von Shortcuts zu vollen Pfaden:

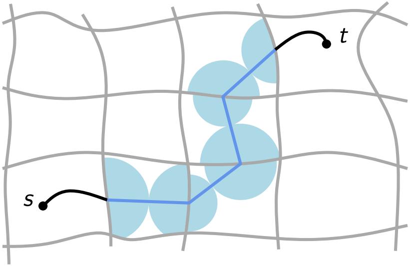
- bidirektionale Suche
- beschränkt auf die Zelle
- benutze untere Level rekursiv



Entpacken der Shortcuts

Von Shortcuts zu vollen Pfaden:

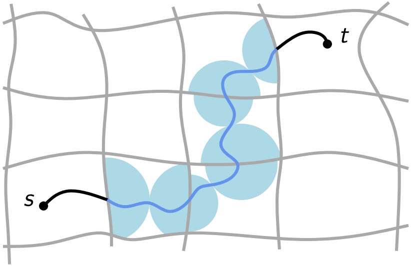
- bidirektionale Suche
- beschränkt auf die Zelle
- benutze untere Level rekursiv



Entpacken der Shortcuts

Von Shortcuts zu vollen Pfaden:

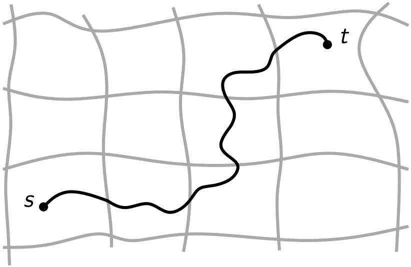
- bidirektionale Suche
- beschränkt auf die Zelle
- benutze untere Level rekursiv



Entpacken der Shortcuts

Von Shortcuts zu vollen Pfaden:

- bidirektionale Suche
- beschränkt auf die Zelle
- benutze untere Level rekursiv



Customizable Route Planning (CRP)

2-Phasen vs 3-Phasen

Bisher:

- Phase 1 (langsam): Vorberechnung
- Phase 2 (sehr schnell): *st*-Anfrage

Problem:

- Gewichte ändern sich $\rightarrow$ Vorberechnung muss neu gemacht werden

2-Phasen vs 3-Phasen

Bisher:

- Phase 1 (langsam): Vorberechnung
- Phase 2 (sehr schnell): *st*-Anfrage

Problem:

- Gewichte ändern sich → Vorberechnung muss neu gemacht werden

Neu:

- Phase 1 (langsam): Vorberechnung ohne Gewichte
- Phase 2 (schell): Gewichte/Metrik integrieren, genannt *Customization*
- Phase 3 (sehr schnell): *st*-Anfrage

3-Phasen Multilevel-Overlays

1. **Metrik-unabhängige Vorberechnung**
   - Partitionierung des Graphen
   - Erstellen der Topologie des Overlay Graphen

3-Phasen Multilevel-Overlays

1. **Metrik-unabhängige Vorberechnung**
   - Partitionierung des Graphen
   - Erstellen der Topologie des Overlay Graphen
2. **Metrik-abhängige Vorberechnung**
   - Berechnung der Gewichte der Matrix-Cliquen-Einträge
   - mit Hilfe von lokalen (hoch-parallelisierbaren) Dijkstra-Suchen

3-Phasen Multilevel-Overlays

Karlsruher Institut für Technologie

1. **Metrik-unabhängige Vorberechnung**
   - Partitionierung des Graphen
   - Erstellen der Topologie des Overlay Graphen

2. **Metrik-abhängige Vorberechnung**
   - Berechnung der Gewichte der Matrix-Cliquen-Einträge
   - mit Hilfe von lokalen (hoch-parallelisierbaren) Dijkstra-Suchen

3. **Queries**
   - benutze Graph und beide Vorberechnungen
   - bidirektionaler Dijkstra

Begrifflichkeiten

- Customizable Route Planning (CRP) $=$ 3 Phasen Problemformulierung
- Multilevel-Overlays (MLO) $=$ Algorithmus

- **Aber:** MLO war der erste 3-Phasen-Algorithmus
  $\rightarrow$ CRP wird oft als Synonym für MLO verwendet
- **Ferner:** MLO sind auch bekannt als Multilevel-Dijkstra (MLD) oder Multilevel Overlay Graphs (MOG)

Partitionierung

Wo kommt die Partitionierung her?

- Eigenes Forschungsfeld
- Viele Algorithmen/Softwarepakete
  - Jostle
  - Scotch
  - Metis
  - PUNCH
  - KaHip
  - FlowCutter
  - Inertial Flow
  - ...
- Kurzvorstellung von Inertial Flow, da:
  - sehr einfach
  - kompetitive Ergebnisse auf Straßen
    (geht aber besser)

Inertial Flow

Idee:

- Nutze geographische Einbettung
- Basiert auf Max-Flow / Min-Cut
- Berechnet eine Bipartitionierung

Inertial Flow

Idee:

- Nutze geographische Einbettung
- Basiert auf Max-Flow / Min-Cut
- Berechnet eine Bipartitionierung

Algo:

- Für beide Diagonalen, die Horizontale und die Vertikale:
  - Projiziere Knoten auf Gerade
  - Orderne Knoten nach Position
  - Mache die bn ersten/letzten Knoten zur Quelle/Senke
    (Ein typischer Wert für b ist 0.25 - 0.45)
  - Berechne einen Min-Cut
- Nimm den besten der 4 berechneten Schnitte

Inertial Flow

k-Partitionierung

- Inertial Flow teilt den Graph in zwei Teile
- Um k Teile zu erhalten gibt es folgenden einfachen Algorithmus:
  - Solange man weniger als k Teile hat:
    - Zerteile das größte Teil in zwei Teile

Datenstrukturen und Details

Details - Partitionierung

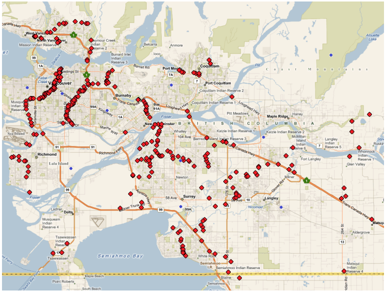
- Partitionierung wird einmal ausgeführt, Customization und Suche oft

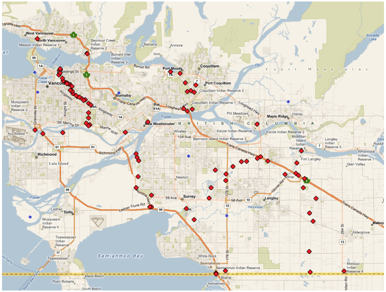
$\rightarrow$ Qualität der Partitionierung ist wichtiger als Laufzeit

Details - Partitionierung

- Partitionierung wird einmal ausgeführt, Customization und Suche oft

$\rightarrow$ Qualität der Partitionierung ist wichtiger als Laufzeit

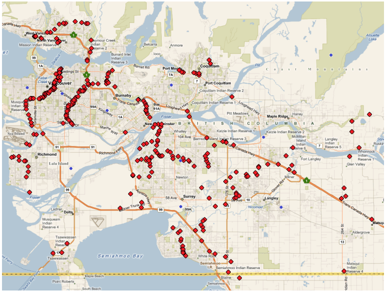


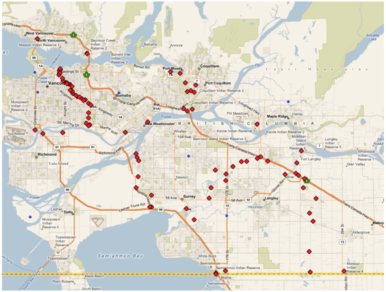


Details - Partitionierung

- Partitionierung wird einmal ausgeführt, Customization und Suche oft

$\rightarrow$ Qualität der Partitionierung ist wichtiger als Laufzeit





- PUNCH erzeugt deutlich weniger Randknoten als METIS

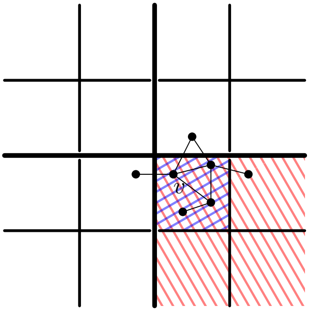
$\rightarrow$ Schnellere Customization und Suche

Details - Partitionierung

- Knoten werden pro (verschachteltem) Level einer Zelle zugewiesen

Details - Partitionierung

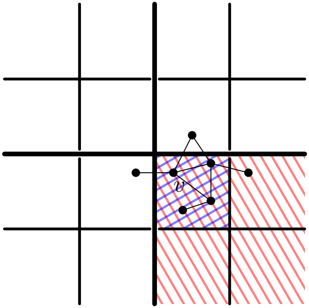
- Knoten werden pro (verschachteltem) Level einer Zelle zugewiesen
- Wähle Knoten-ID als Konkatenation der Zell-IDs (64-Bit)
- Höchstes Level zuerst, lokale ID zuletzt



$v = 110001$

Details - Partitionierung

- Knoten werden pro (verschachteltem) Level einer Zelle zugewiesen
- Wähle Knoten-ID als Konkatenation der Zell-IDs (64-Bit)
- Höchstes Level zuerst, lokale ID zuletzt
- Noch weniger Speicherbedarf durch:
  - Level 1 Zellen speichern Hiererchie
  - Knoten merken sich ihre Zelle (32-Bit)



$v = 110001$

Details - Partitionierung

In den Levels:

- Die Overlays von Zellen sind Cliquen

$\rightarrow$ Speichere die Shortcuts als Matrizen

Details - Partitionierung

In den Levels:

- Die Overlays von Zellen sind Cliquen

$\rightarrow$ Speichere die Shortcuts als Matrizen

- Die Level 1 Zellen müssen während der Customization oft explizit durchsucht werden

$\rightarrow$ Verwende lokale IDs innerhalb der Zelle für bessere Lokalität

Details - Customization

- Customization Bottom-Up (nutzt die schon erzeugten Shortcuts)

Details - Customization

- Customization Bottom-Up (nutzt die schon erzeugten Shortcuts)
- Benutzt Kopien des relevanten Subgraphen (Lokalität!)

Details - Customization

- Customization Bottom-Up (nutzt die schon erzeugten Shortcuts)
- Benutzt Kopien des relevanten Subgraphen (Lokalität!)
- Customization läuft auf kleinen Graphen

$\rightarrow$ (Multi-Source) Bellman-Ford ist einfacher und lokaler

Details - Customization

- Customization Bottom-Up (nutzt die schon erzeugten Shortcuts)
- Benutzt Kopien des relevanten Subgraphen (Lokalität!)
- Customization läuft auf kleinen Graphen

$\rightarrow$ (Multi-Source) Bellman-Ford ist einfacher und lokaler

- Einfach parallelisierbar (auch SIMD)

Details - Customization

Karlsruher Institut für Technologie

- Mehr Level beschleunigen Customization, erhöhen aber Speicherbedarf

$\rightarrow$ Füge Phantomlevel ein, die nur während der Customization existieren

Details - Customization

- Mehr Level beschleunigen Customization, erhöhen aber Speicherbedarf

$\rightarrow$ Füge Phantomlevel ein, die nur während der Customization existieren

- Wenn sehr oft wenige Gewichtsänderungen berücksichtig werden sollen

$\rightarrow$ Betroffene Zellen können lokal Customization durchführen

Details - Customization

- Mehr Level beschleunigen Customization, erhöhen aber Speicherbedarf

$\rightarrow$ Füge Phantomlevel ein, die nur während der Customization existieren

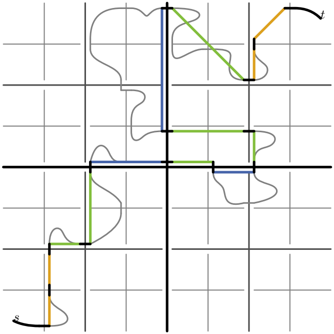
- Wenn sehr oft wenige Gewichtsänderungen berücksichtig werden sollen

$\rightarrow$ Betroffene Zellen können lokal Customization durchführen

- Wir können mehrere Customizations (für verschiedene Metriken) auf der gleichen metrikunabhängigen Struktur verwalten

Details - Query

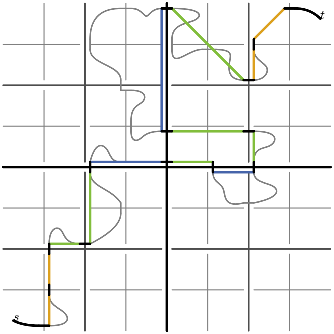
- Knoten-IDs liefern Information, welche Level wo berücksichtigt werden müssen



Details - Query

- Knoten-IDs liefern Information, welche Level wo berücksichtigt werden müssen
- Maximaler Suchraum bekannt

→ Datenstrukturen können vorausschauend alloziert werden

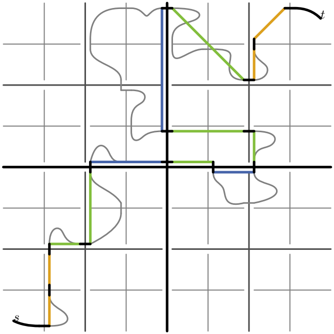


Details - Query

- Knoten-IDs liefern Information, welche Level wo berücksichtigt werden müssen
- Maximaler Suchraum bekannt

$\rightarrow$ Datenstrukturen können vorausschauend alloziert werden

- Lokale IDs für Suche in Level 1 Zellen

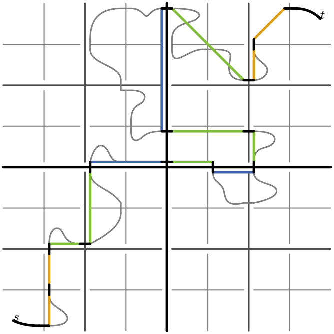


Details - Query

- Knoten-IDs liefern Information, welche Level wo berücksichtigt werden müssen
- Maximaler Suchraum bekannt

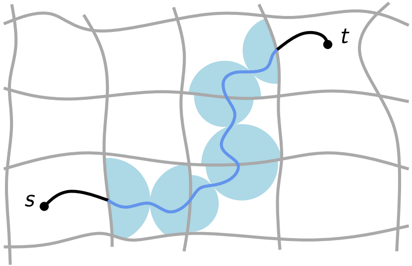
$\rightarrow$ Datenstrukturen können vorausschauend alloziert werden

- Lokale IDs für Suche in Level 1 Zellen
- Bidirektionale Suche



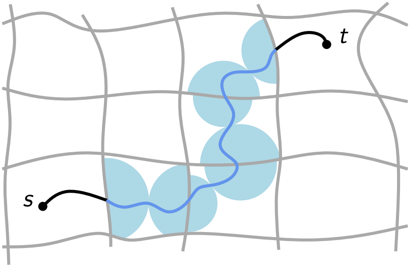
Details - Pfad Entpackung

- Bidirektionale Suche



Details - Pfad Entpackung

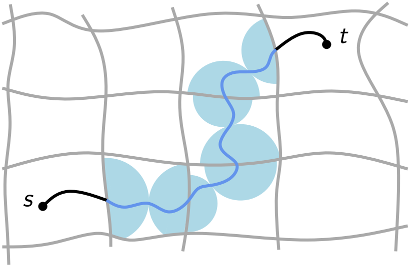
- Bidirektionale Suche
- Top Down: verwende kleinere Zellen rekursiv



Details - Pfad Entpackung

- Bidirektionale Suche
- Top Down: verwende kleinere Zellen rekursiv
- Arc-Flags verwendbar, parallelisierbar

→ Verkompliziert Algorithmus, lohnt sich nicht stark

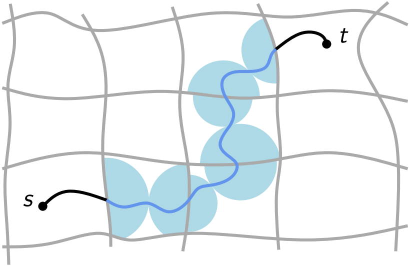


Details - Pfad Entpackung

- Bidirektionale Suche
- Top Down: verwende kleinere Zellen rekursiv
- Arc-Flags verwendbar, parallelisierbar

$\rightarrow$ Verkompliziert Algorithmus, lohnt sich nicht stark

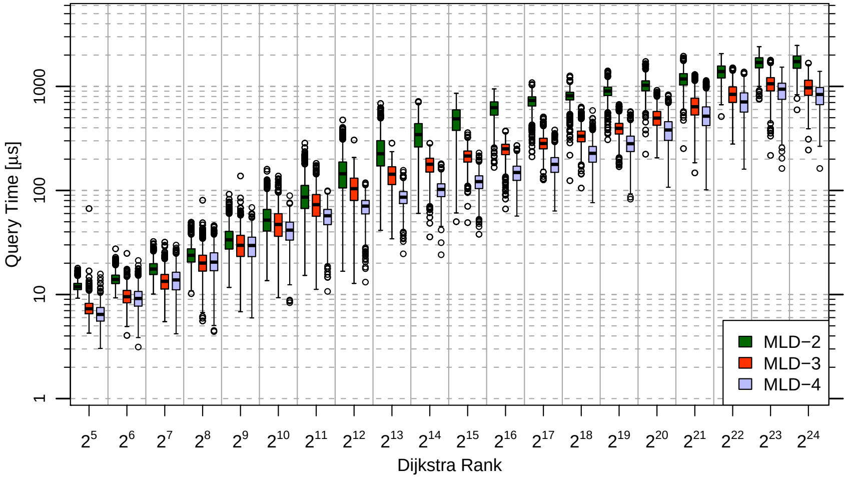
- **Besser:** Caching gefundener Shortcuts



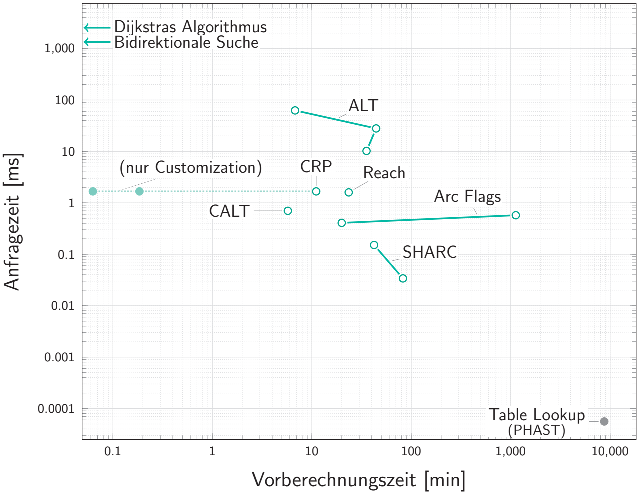
Performance

	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

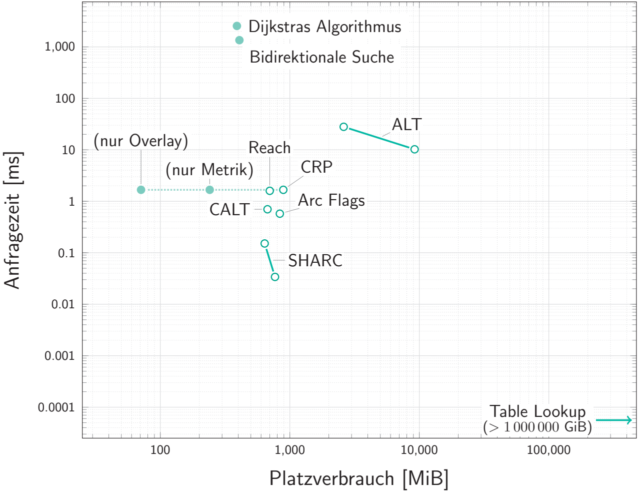
Lokale Queries



Übersicht bisherige Techniken



Übersicht bisherige Techniken



Literatur I

Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck.
Customizable Route Planning.
In Panos M. Pardalos and Steffen Rebennack, editors, *Proceedings of the 10th International Symposium on Experimental Algorithms (SEA'11)*, volume 6630 of *Lecture Notes in Computer Science*, pages 376–387. Springer, 2011.

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.

Aaron Schild and Christian Sommer.
On Balanced Separators in Road Networks.
In *Proceedings of the 14th International Symposium on Experimental Algorithms (SEA'15)*, Lecture Notes in Computer Science. Springer, 2015.

Literatur II

Frank Schulz, Dorothea Wagner, and Karsten Weihe.
Dijkstra's Algorithm On-Line: An Empirical Case Study from Public Railroad Transport.
ACM Journal of Experimental Algorithmics, 5(12):1–23, 2000.

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.