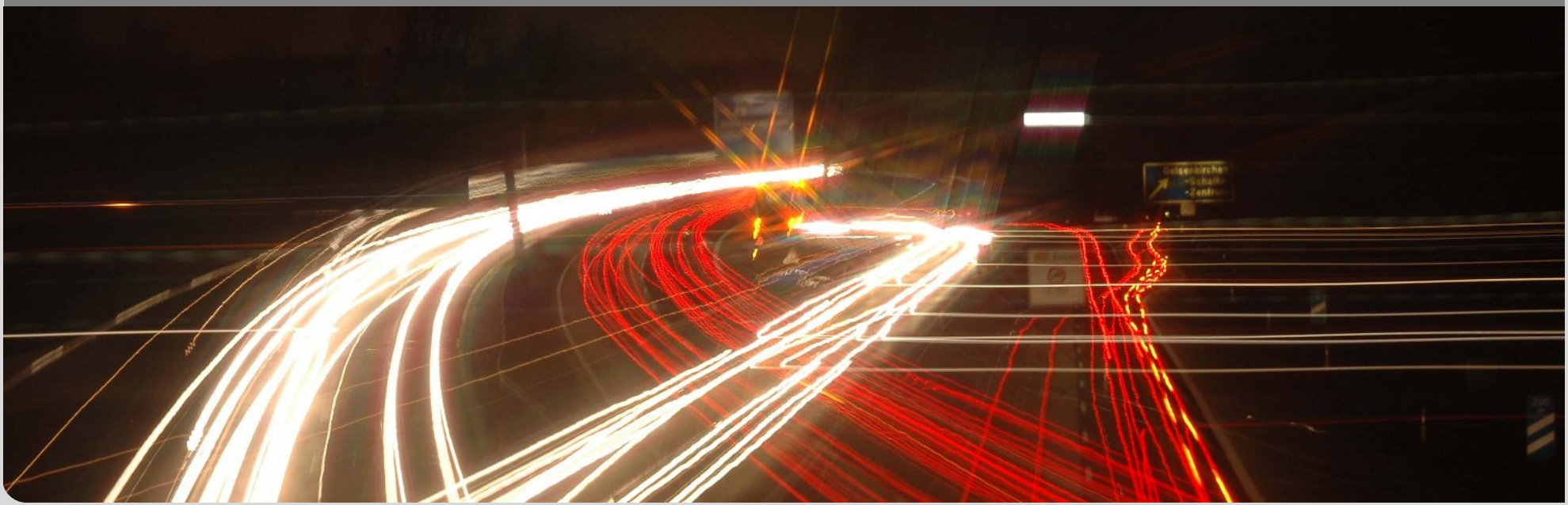
Karlsruher Institut für Technologie

Algorithmen für Routenplanung

5. Vorlesung, Sommersemester 2018

Valentin Buchhold | 2. Mai 2018

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



Kürzeste Wege in Straßennetzwerken

Beschleunigungstechniken (Fortsetzung)

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

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Reach

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Reach Idee:

Intuition:

- Schreibe an jeden Knoten seine “Wichtigkeit”
  - Eine Sackgasse ist unwichtig
  - Eine Autobahn ist wichtig
- **Idee:** Vermeide es unwichtige Knoten abzusuchen

Reach Idee:

Intuition:

- Schreibe an jeden Knoten seine “Wichtigkeit”
  - Eine Sackgasse ist unwichtig
  - Eine Autobahn ist wichtig
- **Idee:** Vermeide es unwichtige Knoten abzusuchen

Wann ist ein Knoten wichtig?

Reach Idee:

Intuition:

- Schreibe an jeden Knoten seine “Wichtigkeit”
  - Eine Sackgasse ist unwichtig
  - Eine Autobahn ist wichtig
- **Idee:** Vermeide es unwichtige Knoten abzusuchen

Wann ist ein Knoten wichtig?

- **Bei Reach**:
  - Mitte eines langen kürzesten Pfads ist wichtig
- **Später in der Vorlesung**:
  - Andere Techniken haben andere “Wichtigkeits”-Begriffe
  - Bitte nicht verwechseln
  - Reach-“Wichtigkeit” eines Knoten heißt der *Reach eines Knoten* oder der *Reach-Wert eines Knoten*

Reach

Idee:

- Bestimme für jeden Knoten u einen Kreis, so dass:
  - für jedes Paar s, t gilt: wenn u auf kürzestem s–t-Weg liegt, ist entweder s oder t im Kreis.
  - Der Radius dieses Kreises ist der Reachwert von u.

Reach

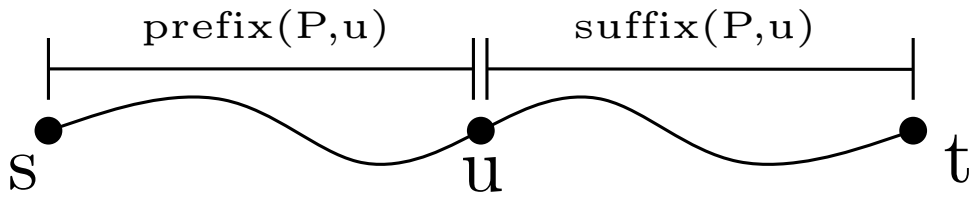
Idee:

- Bestimme für jeden Knoten u einen Kreis, so dass:
  - für jedes Paar s, t gilt: wenn u auf kürzestem s–t-Weg liegt, ist entweder s oder t im Kreis.
  - Der Radius dieses Kreises ist der Reachwert von u.

Strategie für Beschleunigungstechnik:

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

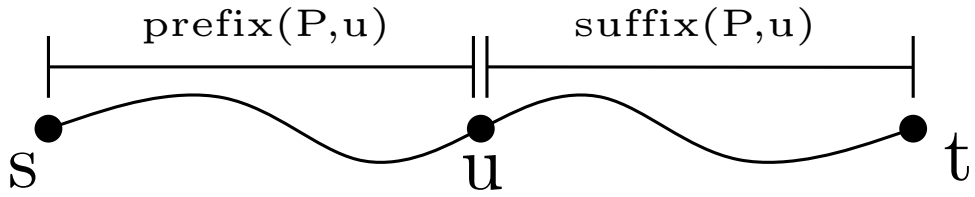
Reach



Definition:

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

Reach



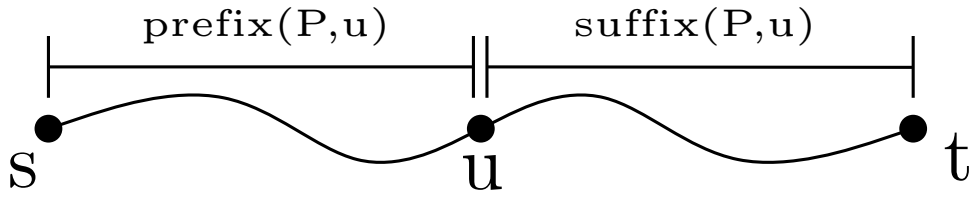
Definition:

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

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

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

Reach



Definition:

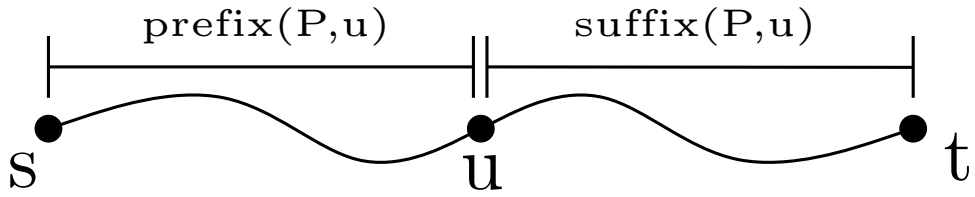
- Sei $P = \langle s, \ldots, u, \ldots, t \rangle$ Pfad durch u
- dann Reach von u bezüglich P:

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

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

- Reach von u:

Reach



Definition:

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

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

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.

Reach-Pruning

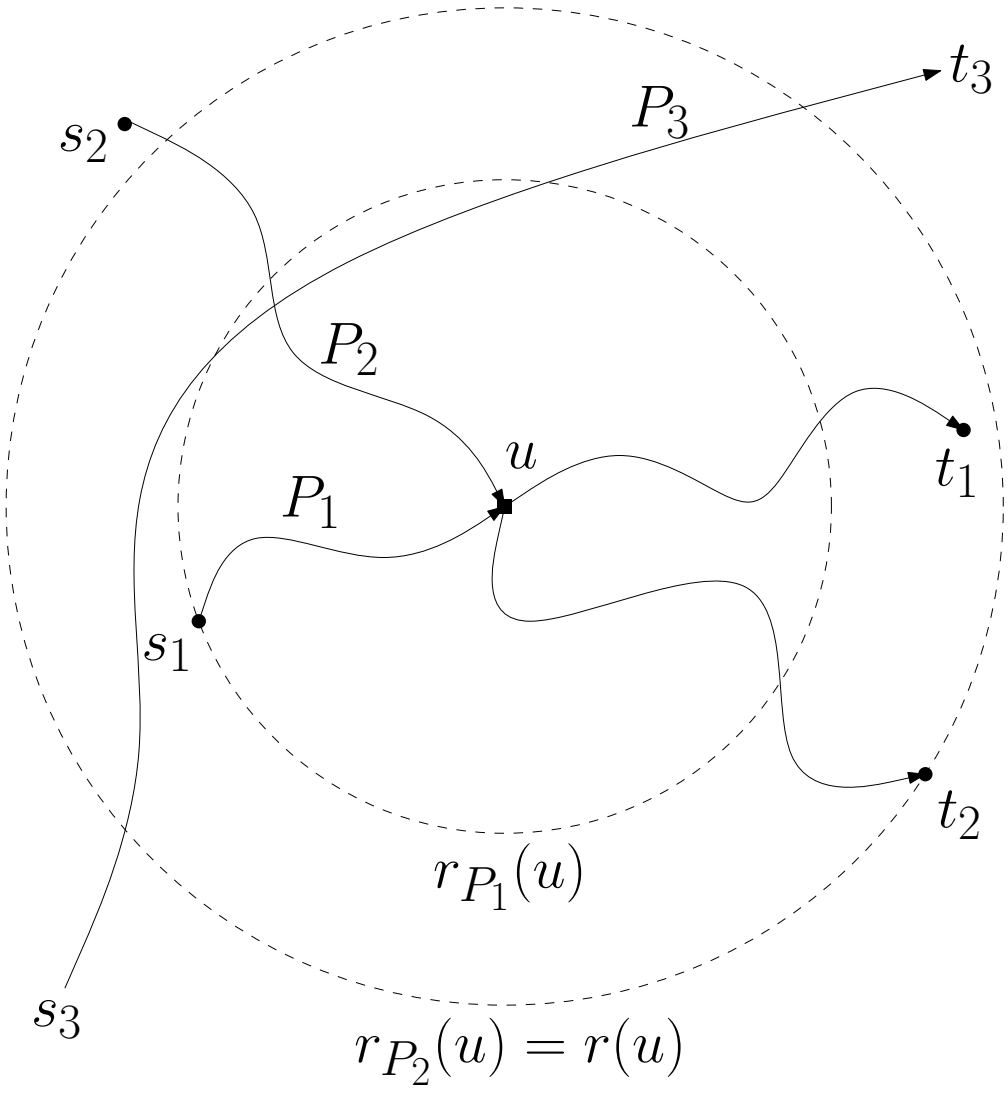
- Reach $r(u)$ von u gibt Suffix oder Prefix des längsten kürzesten Weges durch u

Im Beispiel:

- P_1 und P_2 gehen durch u $\rightarrow r_{P_1}(u) \leq r(u)$ und $r_{P_2}(u) \leq r(u)$
- P_3 geht nicht durch u

Pruning Regel:

- wenn für u während Query $r(u) < d(s,u)$ und $r(u) < d(u,t)$ gilt, muss u nicht beachtet werden



Unidirectional Reach Dijkstra

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 u = t then break                          // Stoppkriterium
6      if r(u) < d[u] and r(u) < d(u, t) then continue
7      forall edges e = (u, v) ∈ E do
8          if d[u] + len(e) < d[v] then
9              d[v] ← d[u] + len(e)
10             if v ∈ Q then Q.decreaseKey(v, d[v])
11             else Q.insert(v, d[v])
```

Problem

Problem?

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Problem

Problem:

- Abfrage $r(u) < d(u,t)$ geht nicht so einfach.

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Problem

Problem:

- Abfrage $r(u) < d(u,t)$ geht nicht so einfach.

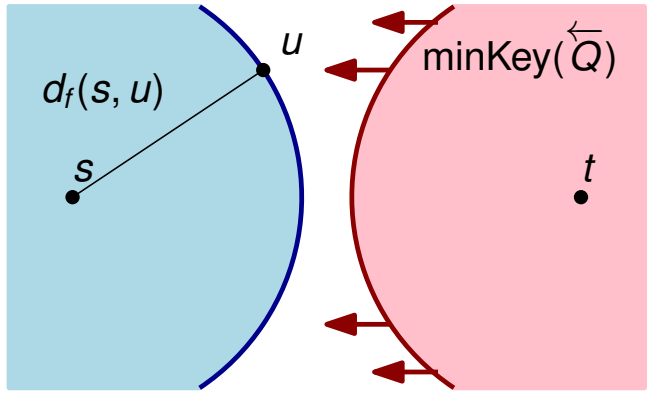
Lösung:

- Nutze bidirektionale Suche
- Zwei Varianten:
  - Bidir. Dist.-Bounding Reach-Dijkstra
  - Bidir. Self-Bounding Reach-Dijkstra

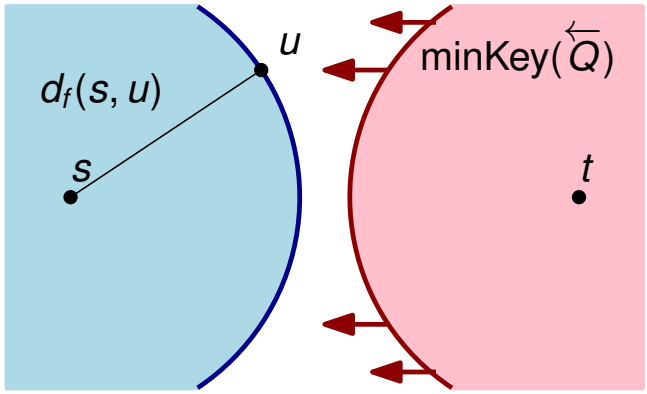
Bidir. Dist.-Bounding Reach-Dijkstra

Erinnerung:

- Prunen erlaubt wenn
  $r(u) < d(s,u)$ und $r(u) < d(u,t)$



Bidir. Dist.-Bounding Reach-Dijkstra



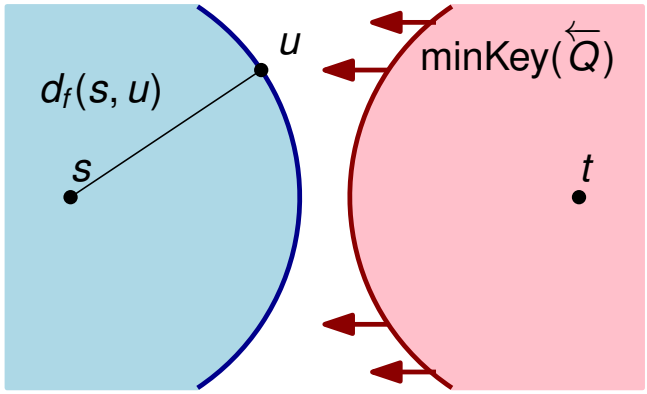
Erinnerung:

- Prunen erlaubt wenn
  $r(u) < d(s,u)$ und $r(u) < d(u,t)$

Idee:

- Bestimme untere Schranke $d_\ell(u,t)$ für $d(u,t)$
- Prune wenn $r(u) < d(s,u)$ und $r(u) < d_\ell(u,t)$
- Wir prunen selterner als erlaubt

Bidir. Dist.-Bounding Reach-Dijkstra



Erinnerung:

- Prunen erlaubt wenn $r(u) < d(s,u)$ und $r(u) < d(u,t)$

Idee:

- Bestimme untere Schranke $d_\ell(u,t)$ für $d(u,t)$
- Prune wenn $r(u) < d(s,u)$ und $r(u) < d_\ell(u,t)$
- Wir prunen selterner als erlaubt

Untere Schranke:

- Wenn u bereits aus der Rückwärts-Queue genommen wurde: $d_b[u] = d(u,t) \rightarrow$ setze $d_\ell(u,t) = d(u,t)$
- Sonst: minKey($\overleftarrow{Q}$) $\leq d(u,t)$ setze $d_\ell(u,t) =$ minKey($\overleftarrow{Q}$)

Bidir. Self-Bounding Reach-Dijkstra

Erinnerung:

- Prunen erlaubt wenn
  $r(u) < d(s,u)$ und $r(u) < d(u,t)$

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Bidir. Self-Bounding Reach-Dijkstra

Erinnerung:

- Prunen erlaubt wenn
  $r(u) < d(s,u)$ und $r(u) < d(u,t)$

Idee:

- Prune wenn $r(u) < d(s,u)$
- Wir prunen öfter als erlaubt
  $\rightarrow$ Algorithmus erstmal inkorrekt

Bidir. Self-Bounding Reach-Dijkstra

Erinnerung:

- Prunen erlaubt wenn
  $r(u) < d(s,u)$ und $r(u) < d(u,t)$

Idee:

- Prune wenn $r(u) < d(s,u)$
- Wir prunen öfter als erlaubt
  $\rightarrow$ Algorithmus erstmal inkorrekt

Neues Stoppkriterium:

- **Idee:** Rückwärtssuche soll Knoten besuchen, die Vorwärtssuche zu viel geprunt hat
- stoppe eine Suchrichtung, wenn minKey$(Q) \geq \mu/2$ oder Queue leer
- stoppe Anfrage, wenn beide Suchrichtungen gestoppt haben
- **Korrektheit:** gute Fingerübung

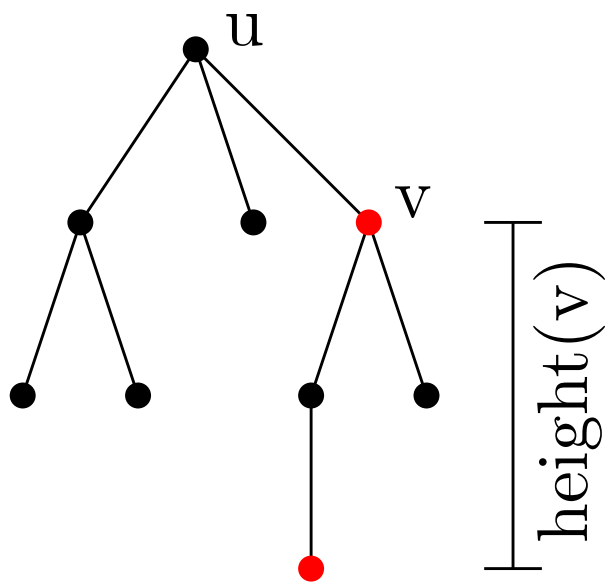
Was ist besser?

- Bidir. Dist.-Bounding Reach-Dijkstra prunt weniger als möglich
- Bidir. Self-Bounding Reach-Dijkstra hat schlechteres Stoppkriterium
- Was besser ist hängt vom Graph ab

Vorberechnung:

Wie kann man Reach-Werte vorberechnen?

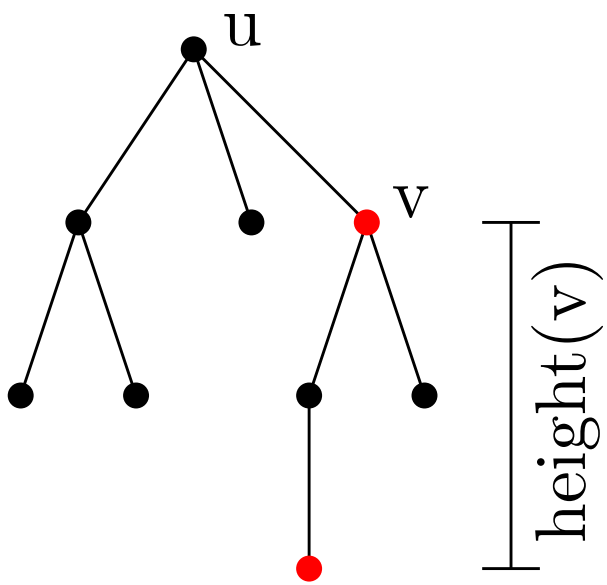
- initialisiere $r(u) = 0$ für alle Knoten
- für jeden Knoten u
  - konstruiere kürzeste Wege-Baum
  - Höhe von Knoten v: Abstand von v zum am weitesten entfernten Nachfolger
  - für jeden Knoten v:
    $r(v) = \max\{r(v), \min\{d(u,v), \text{height}(v)\}\}$



Vorberechnung:

Wie kann man Reach-Werte vorberechnen?

- initialisiere $r(u) = 0$ 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

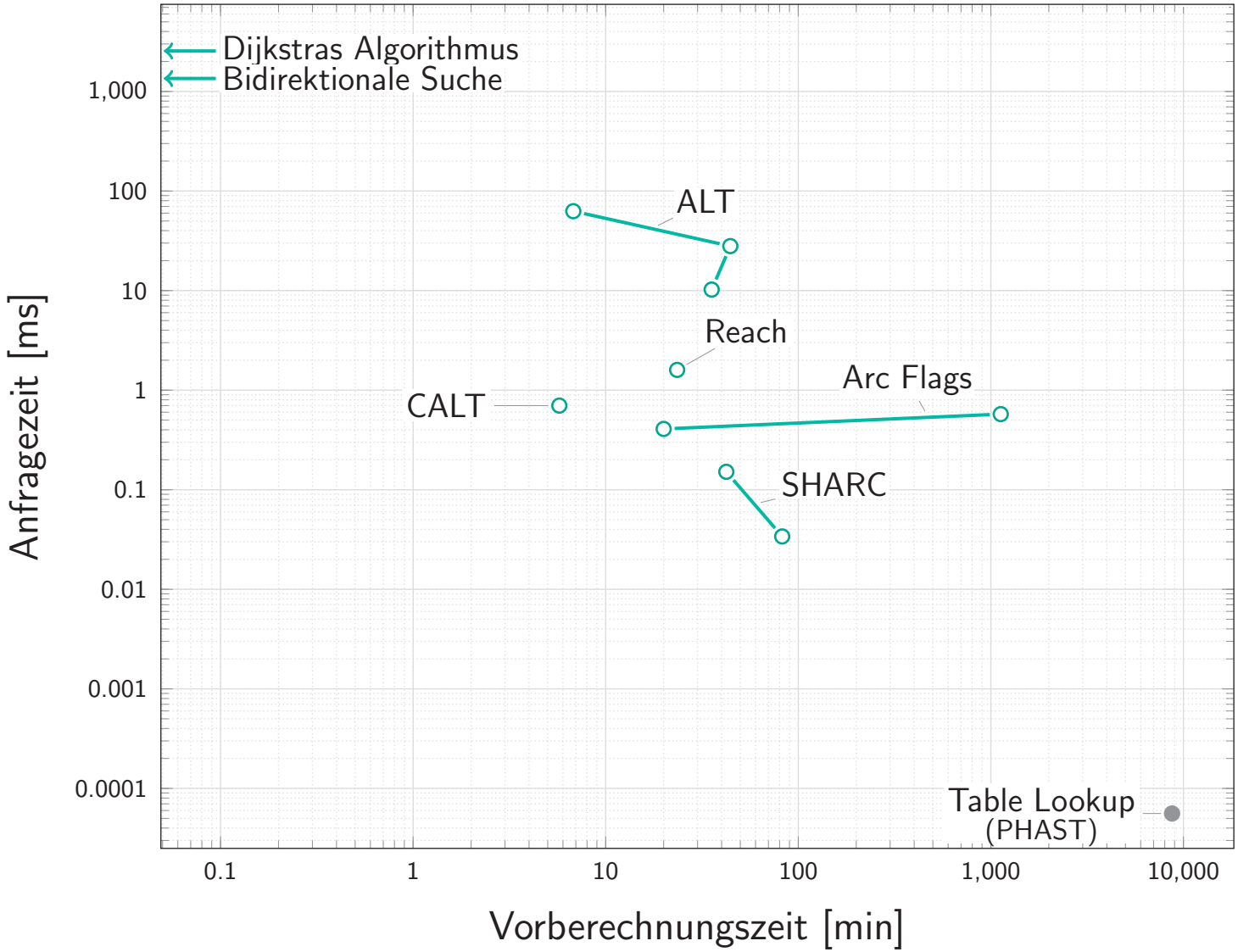
Beobachtung:

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

Problem:

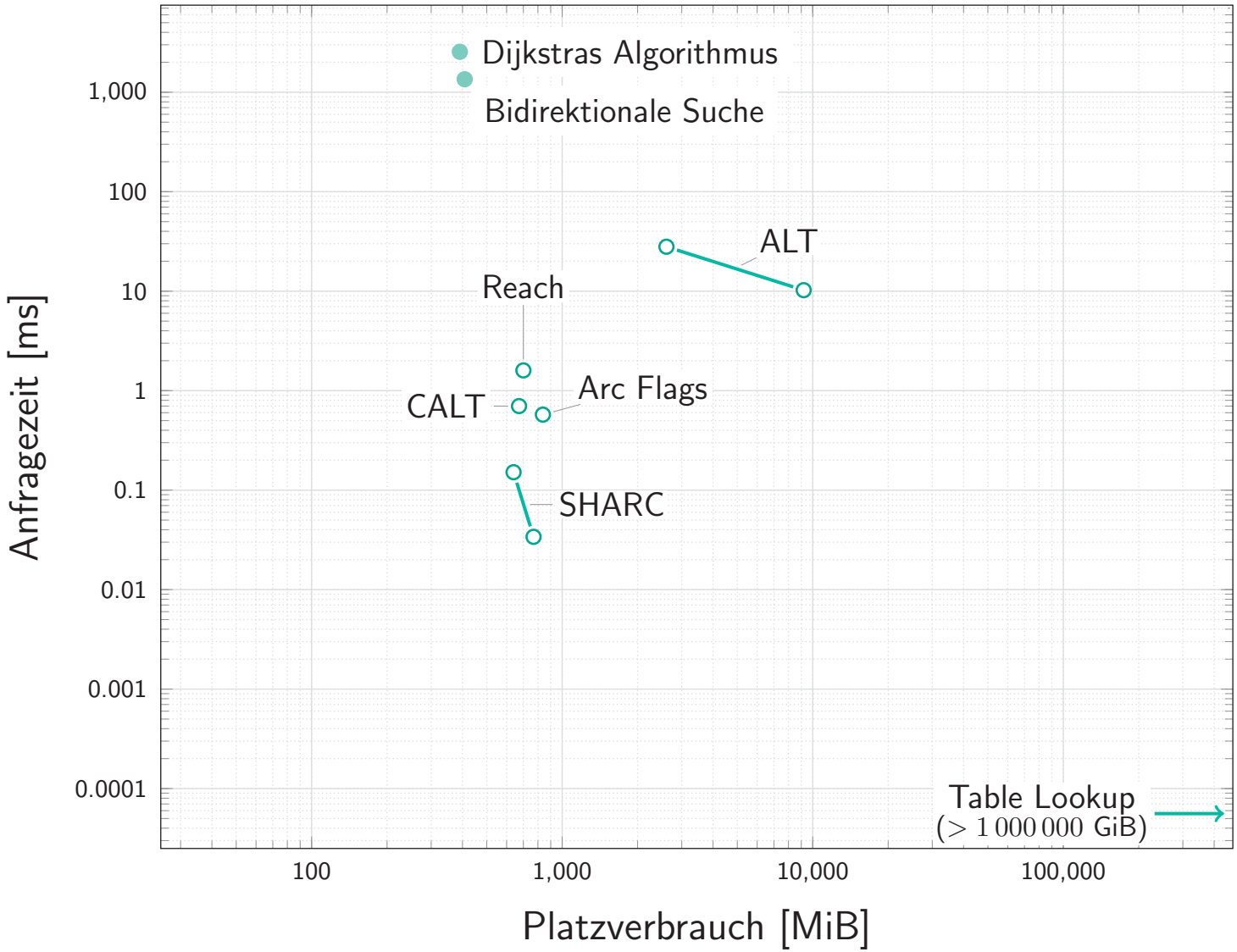
- untere Schranken einfach zu finden:
  - breche Konstruktion der Bäume einfach bei bestimmter Größe ab
- aber: untere Schranken sind unbrauchbar
- Berechnung von oberen Schranken deutlich schwieriger
- möglich, aber sehr aufwendig
- nicht (mehr) Bestandteil der Vorlesung

Übersicht bisherige Techniken



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Übersicht bisherige Techniken

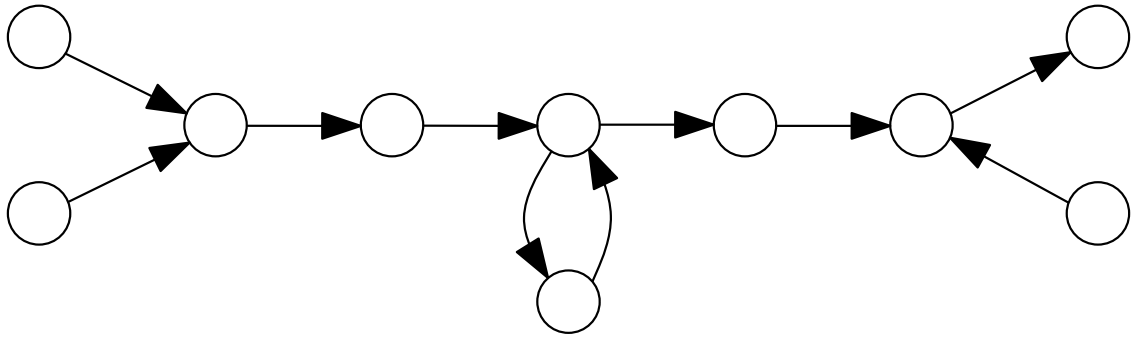


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Shortcuts

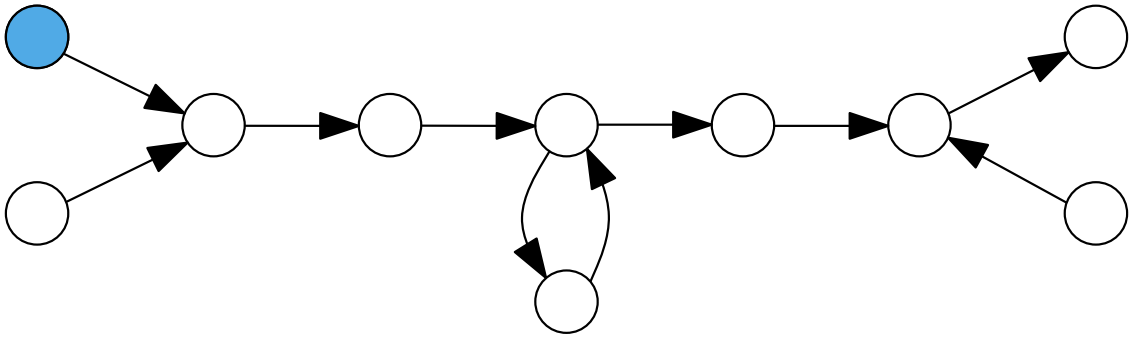
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Shortcut



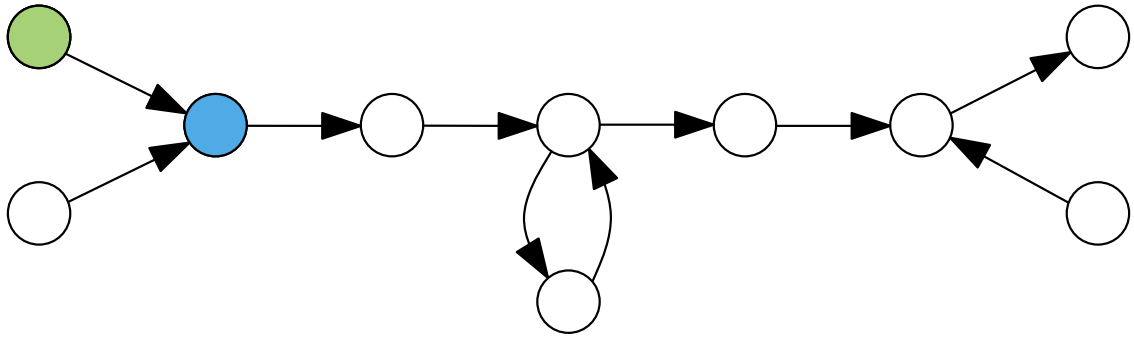
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.

Shortcut



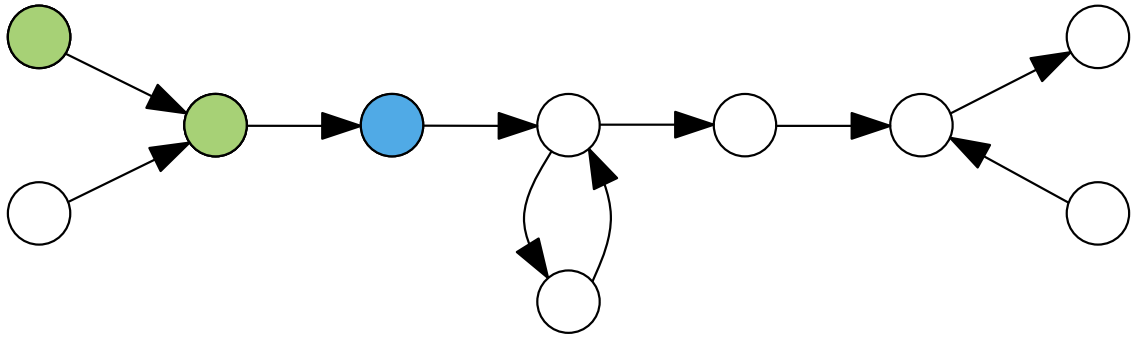
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.

Shortcut



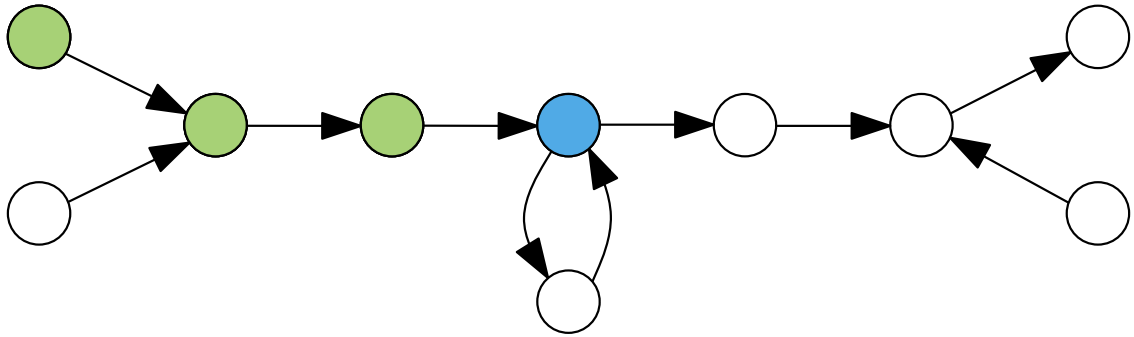
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.

Shortcut



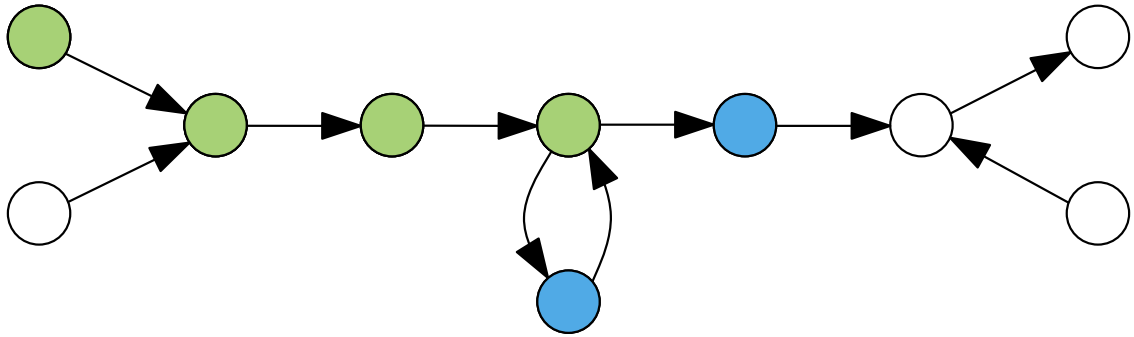
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.

Shortcut



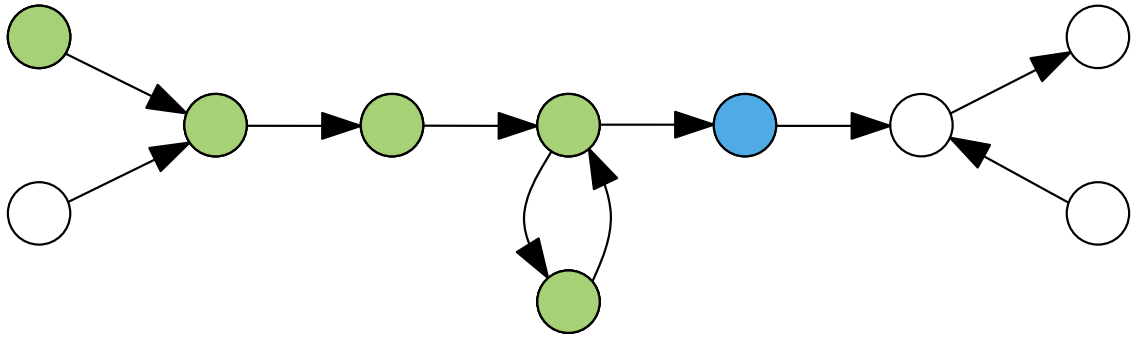
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.

Shortcut



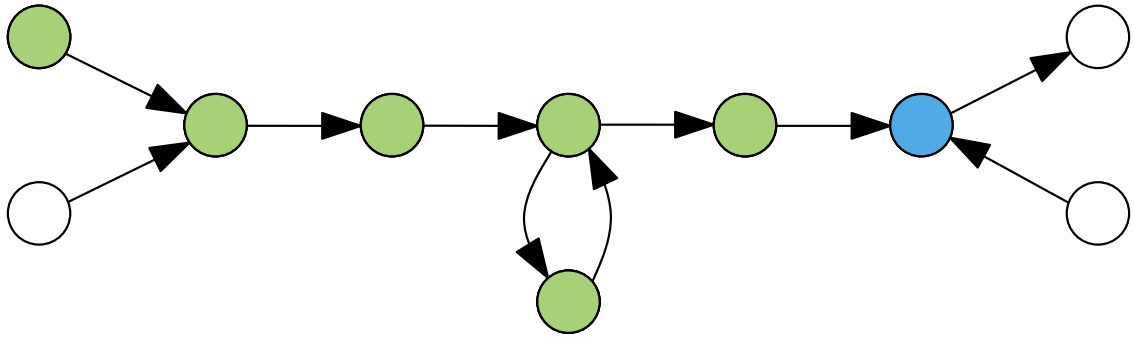
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.

Shortcut



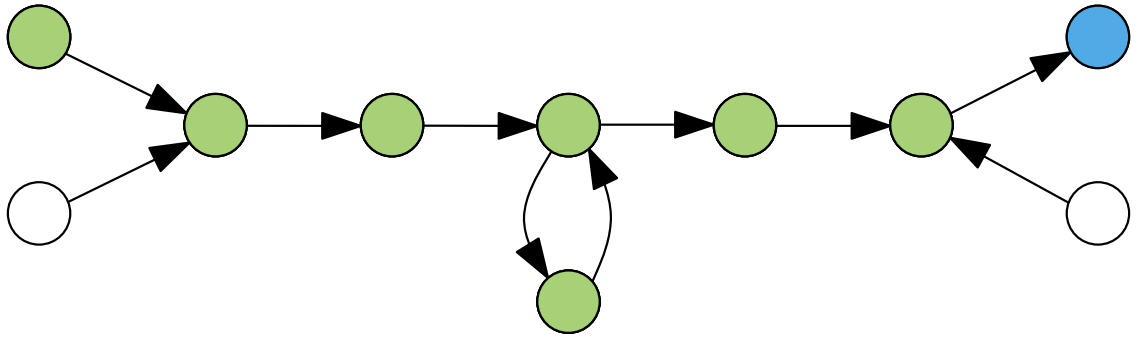
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.

Shortcut



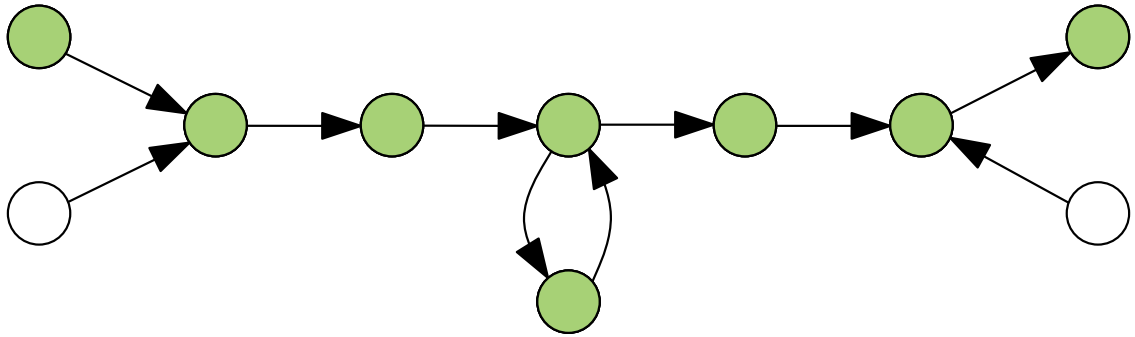
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.

Shortcut



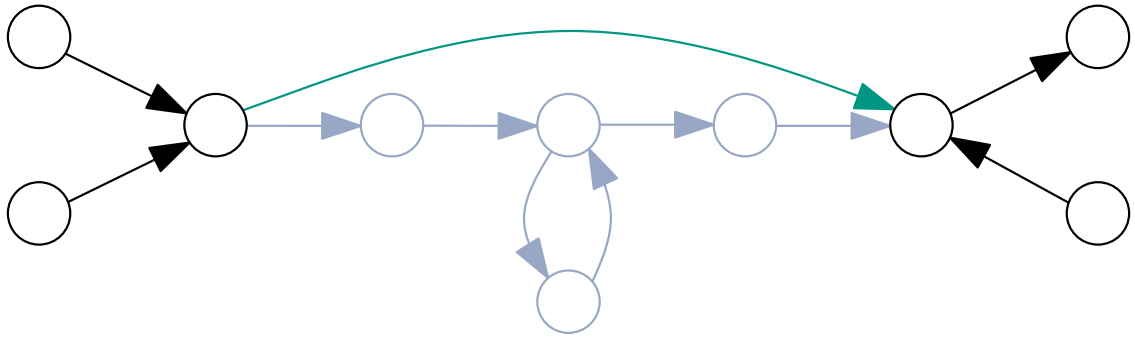
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.

Shortcut



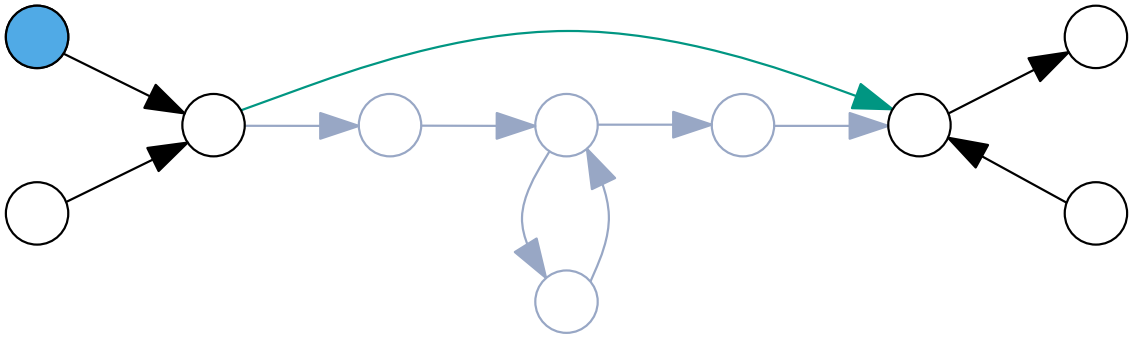
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.

Shortcut



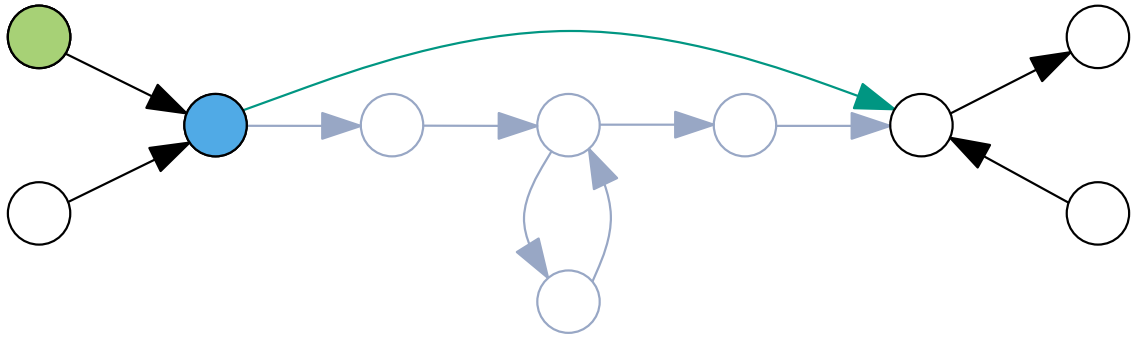
Idee: Füge Shortcut-Kante ein. Grauer Teilgraph muss nur angeschaut werden, wenn *s* oder *t* drin liegt.

Shortcut



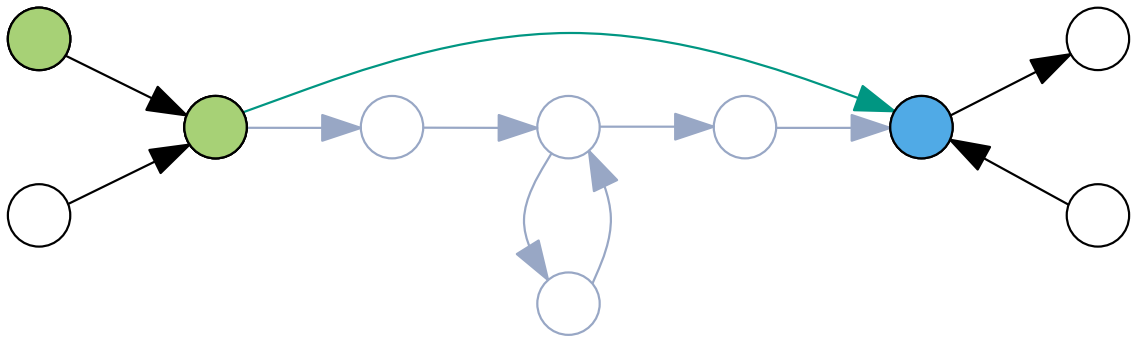
Idee: Füge Shortcut-Kante ein. Grauer Teilgraph muss nur angeschaut werden, wenn *s* oder *t* drin liegt.

Shortcut



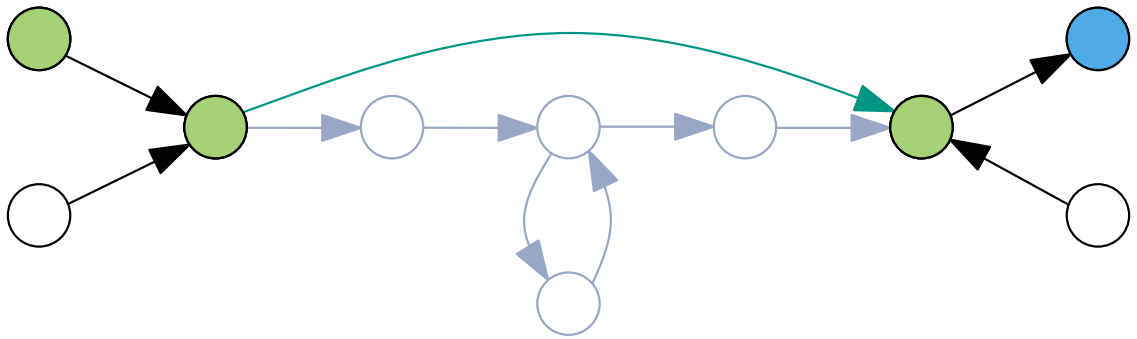
Idee: Füge Shortcut-Kante ein. Grauer Teilgraph muss nur angeschaut werden, wenn *s* oder *t* drin liegt.

Shortcut



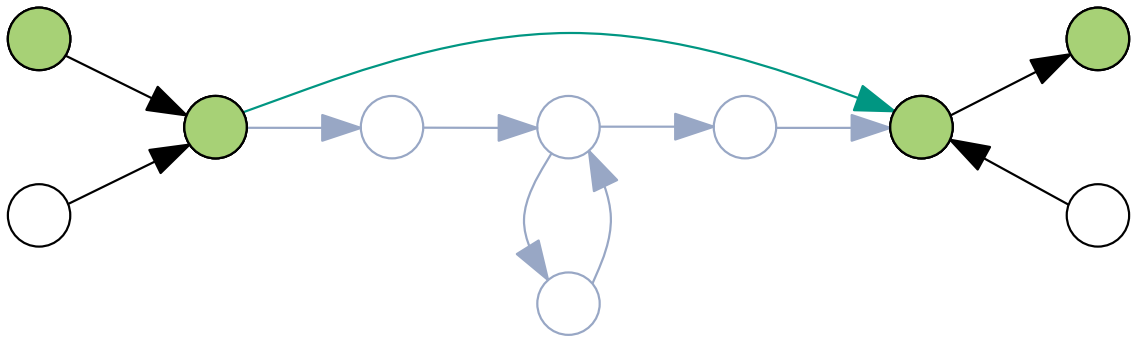
Idee: Füge Shortcut-Kante ein. Grauer Teilgraph muss nur angeschaut werden, wenn *s* oder *t* drin liegt.

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)
  - ALT + Reach + Shortcuts = REAL
  - ArcFlags + Shortcuts = SHARC
  - . . .
- Wie weit kommen wir nur mit Shortcuts?

Bilevel-Overlays

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Overlay

Wiederholung:

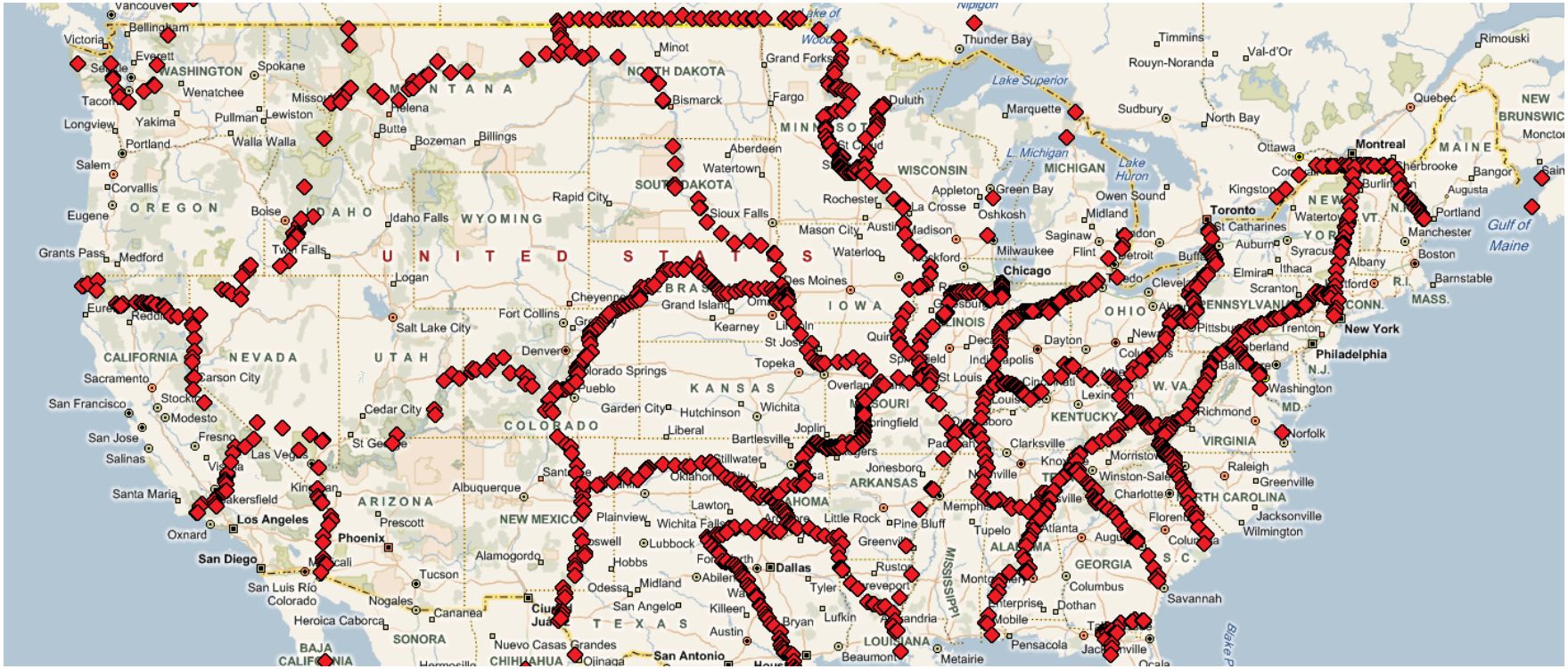
- Ein Overlay eines Graphen $G = (V, E)$ ist ein Graph $G_O = (V_O, E_O)$, so dass:
  - $V_O \subseteq V$
  - Für alle s und t aus V_O gilt: Die st-Distanz in G ist gleich der st-Distanz in G_O

Neu:

- Berechne Overlay aus Partitionierung

Partitionierung

Beobachtung: Straßengraphen haben dünne, natürliche Schnitte



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

Ausnutzung der Partition

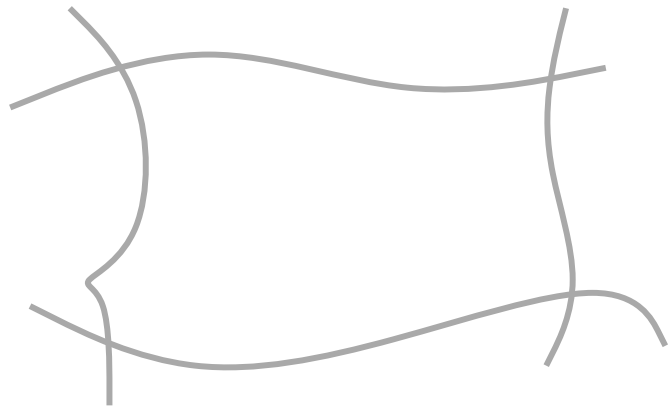
Idee: Berechne Distanzen zwischen Randknoten *in jeder Zelle*

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

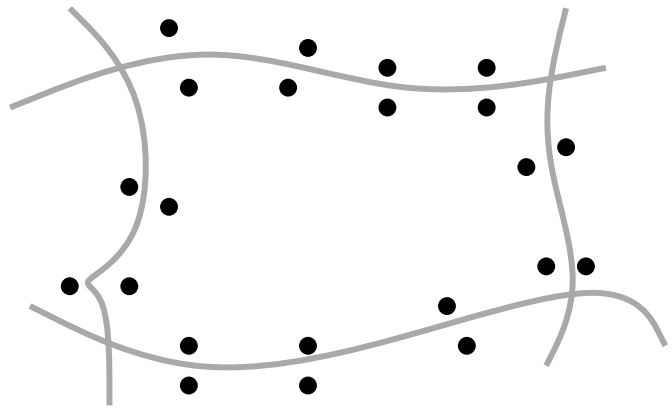


Ausnutzung der Partition

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

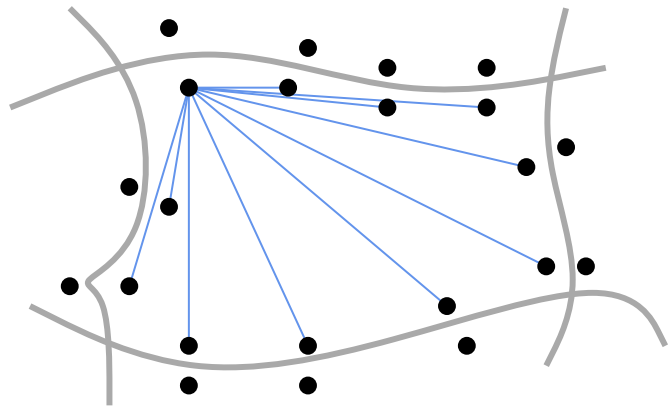


Ausnutzung der Partition

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

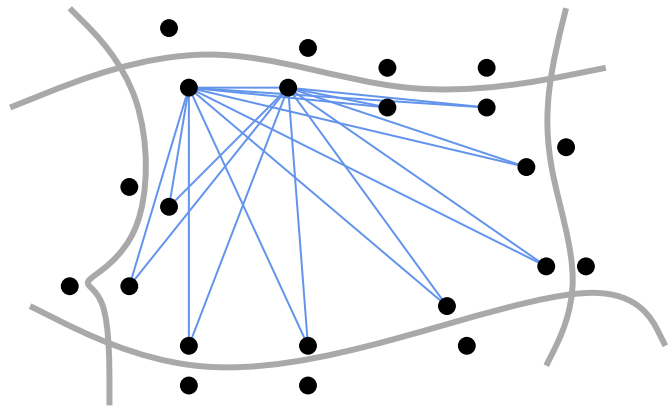


Ausnutzung der Partition

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

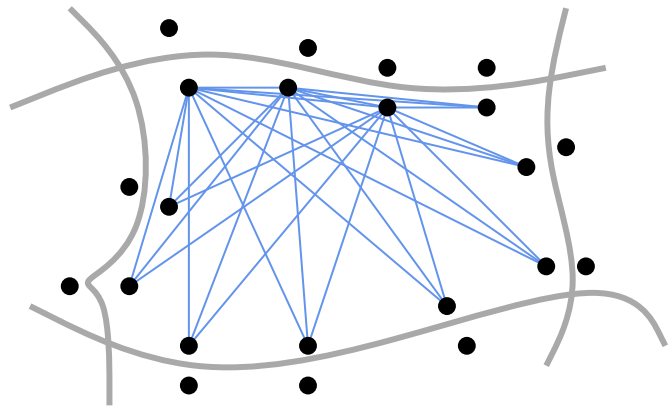


Ausnutzung der Partition

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

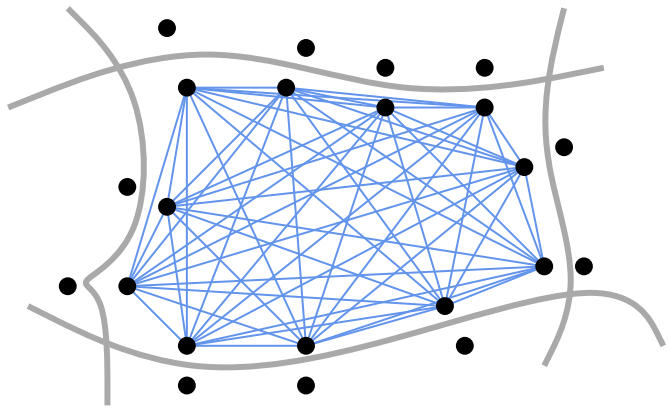


Ausnutzung der Partition

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

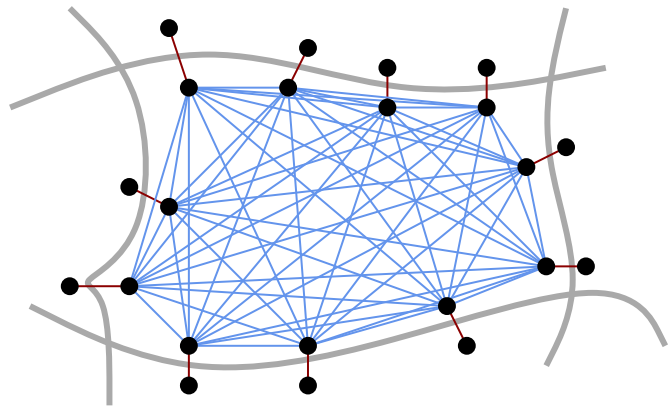


Ausnutzung der Partition

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

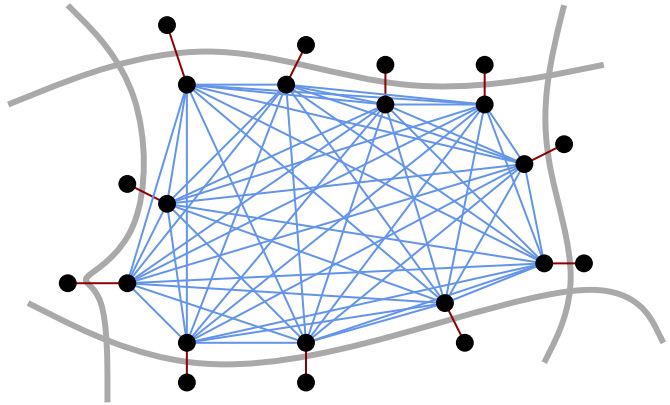


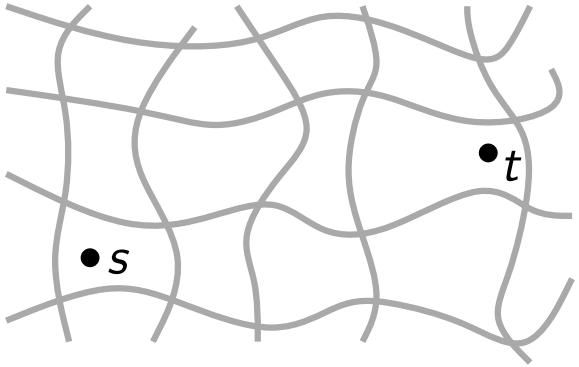
Ausnutzung der Partition

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:

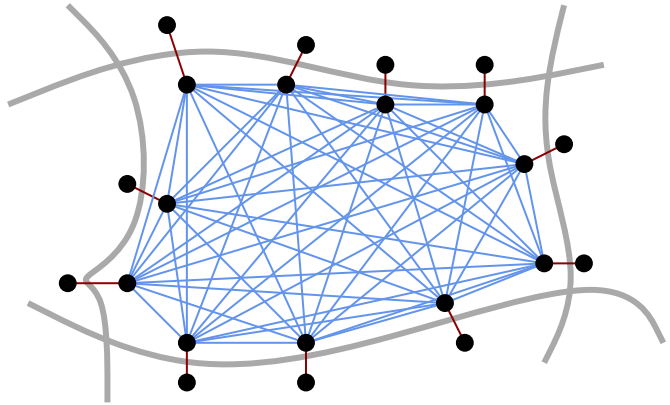
- Start- und Zielzelle...
- ...plus Overlaygraph.
- (bidirektionaler) Dijkstra

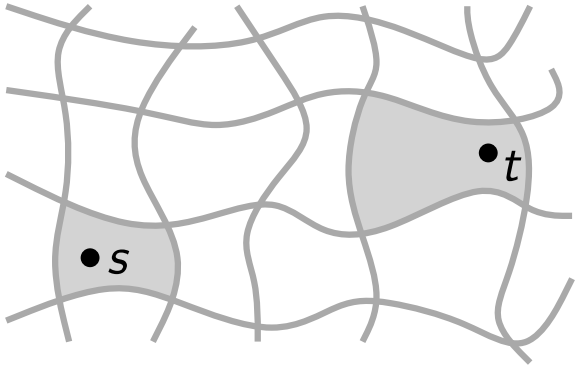
Ausnutzung der Partition

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:

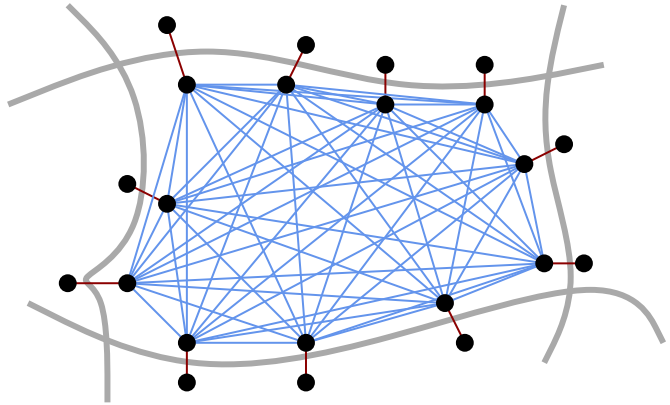
- Start- und Zielzelle...
- ...plus Overlaygraph.
- (bidirektionaler) Dijkstra

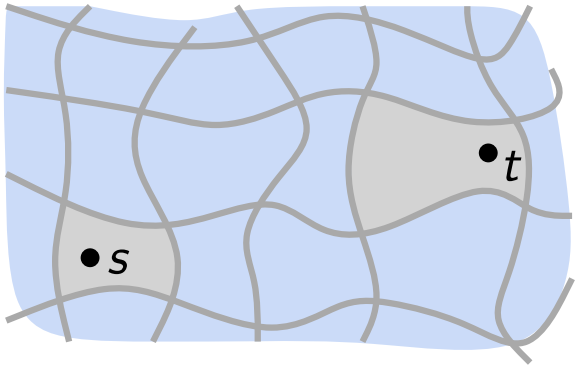
Ausnutzung der Partition

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:

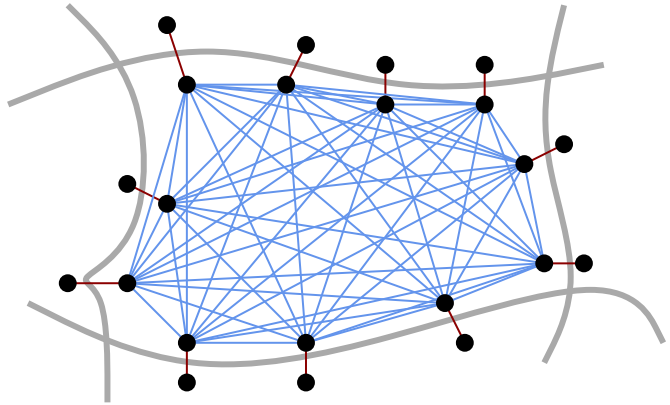
- Start- und Zielzelle...
- ...plus Overlaygraph.
- (bidirektionaler) Dijkstra

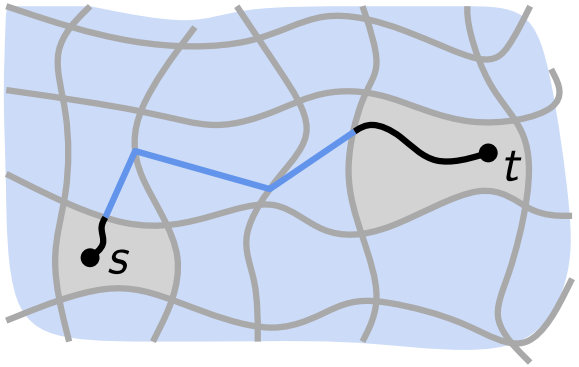
Ausnutzung der Partition

Idee: Berechne Distanzen zwischen Randknoten *in jeder Zelle*

Overlay Graph:

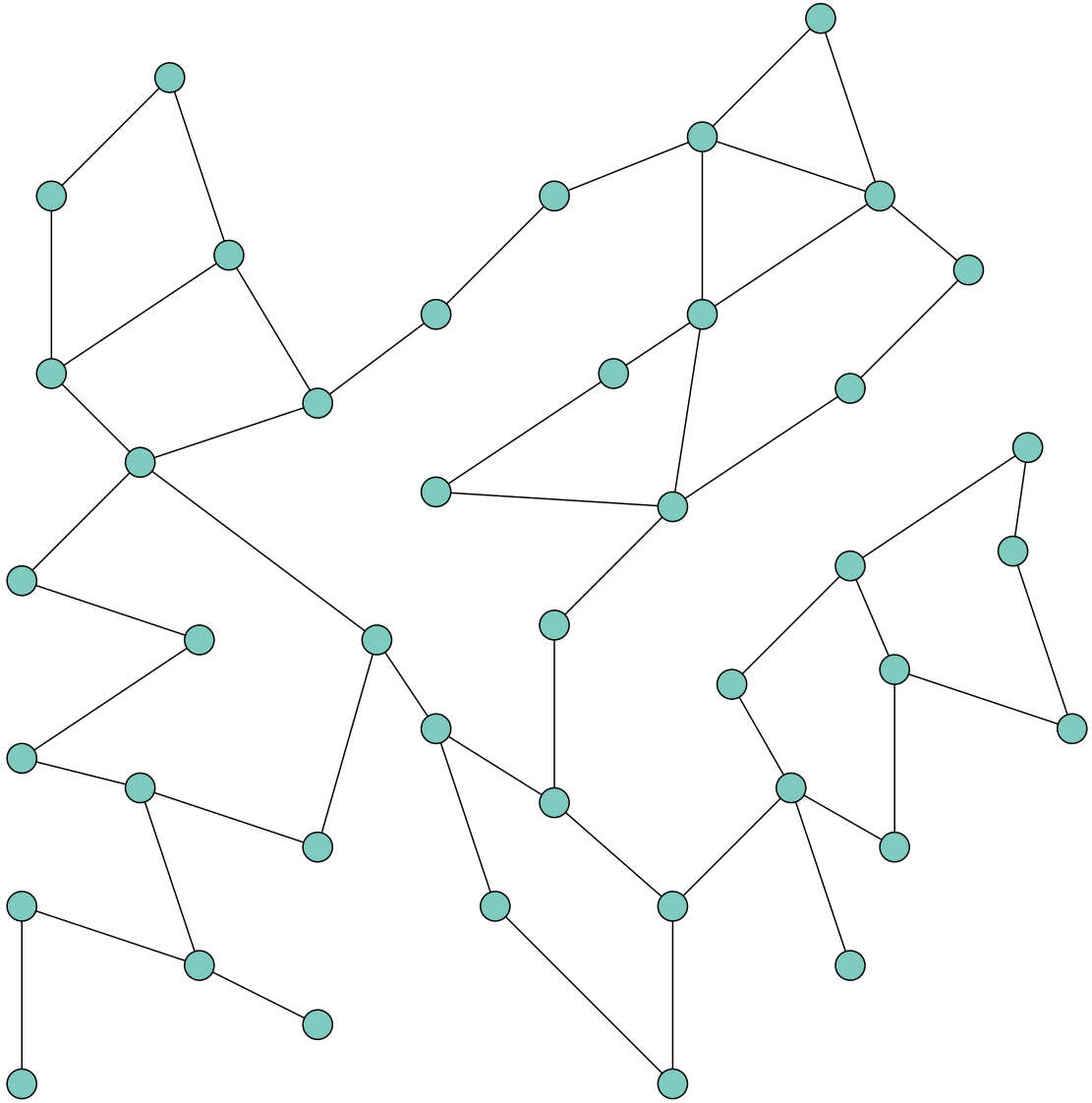
- Randknoten
- Cliquen in jeder Zelle
  Gewicht eines Shortcuts entspricht kürzestem Weg durch die Zelle
- Schnittkanten





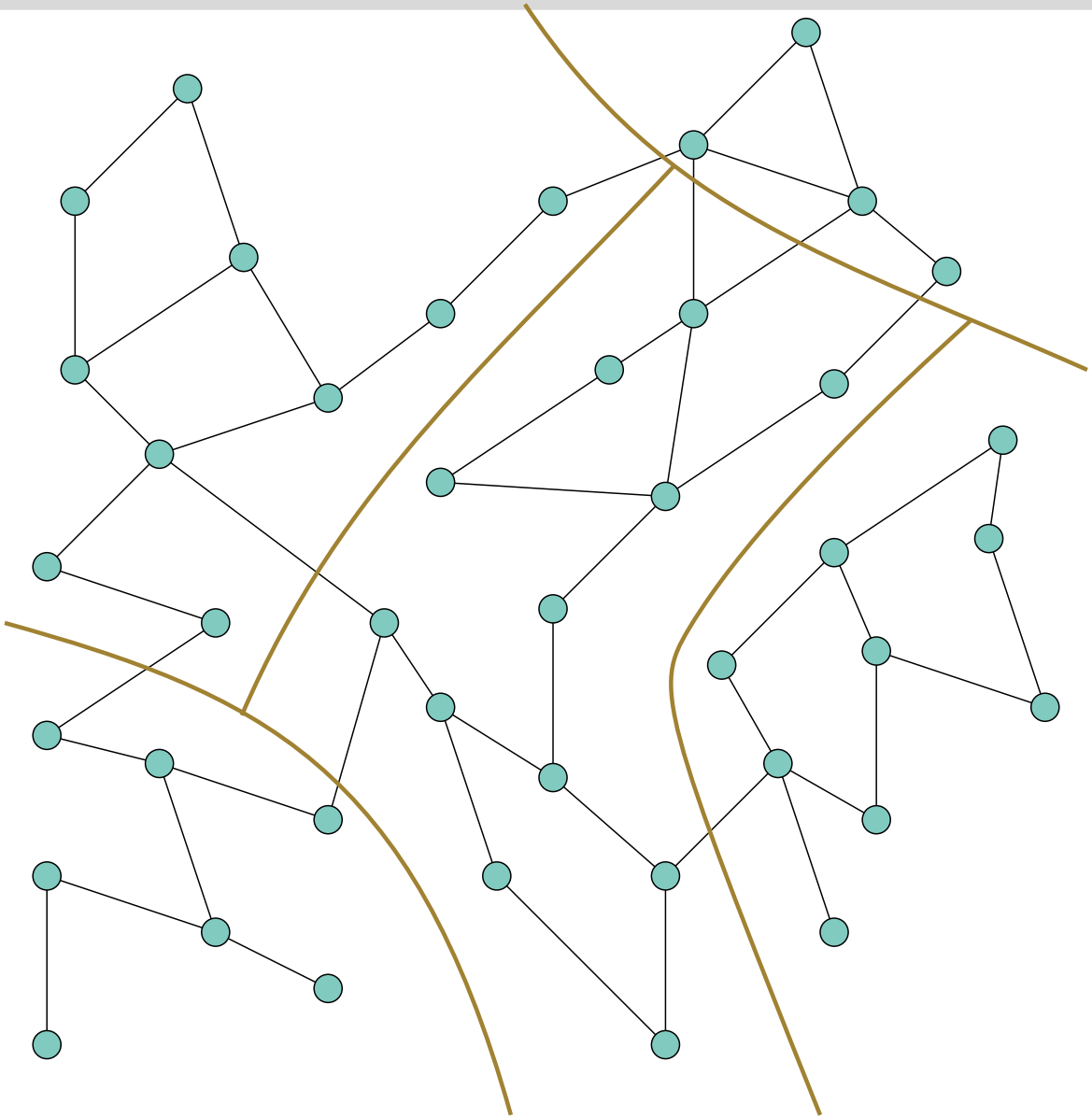
Suchgraph:

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



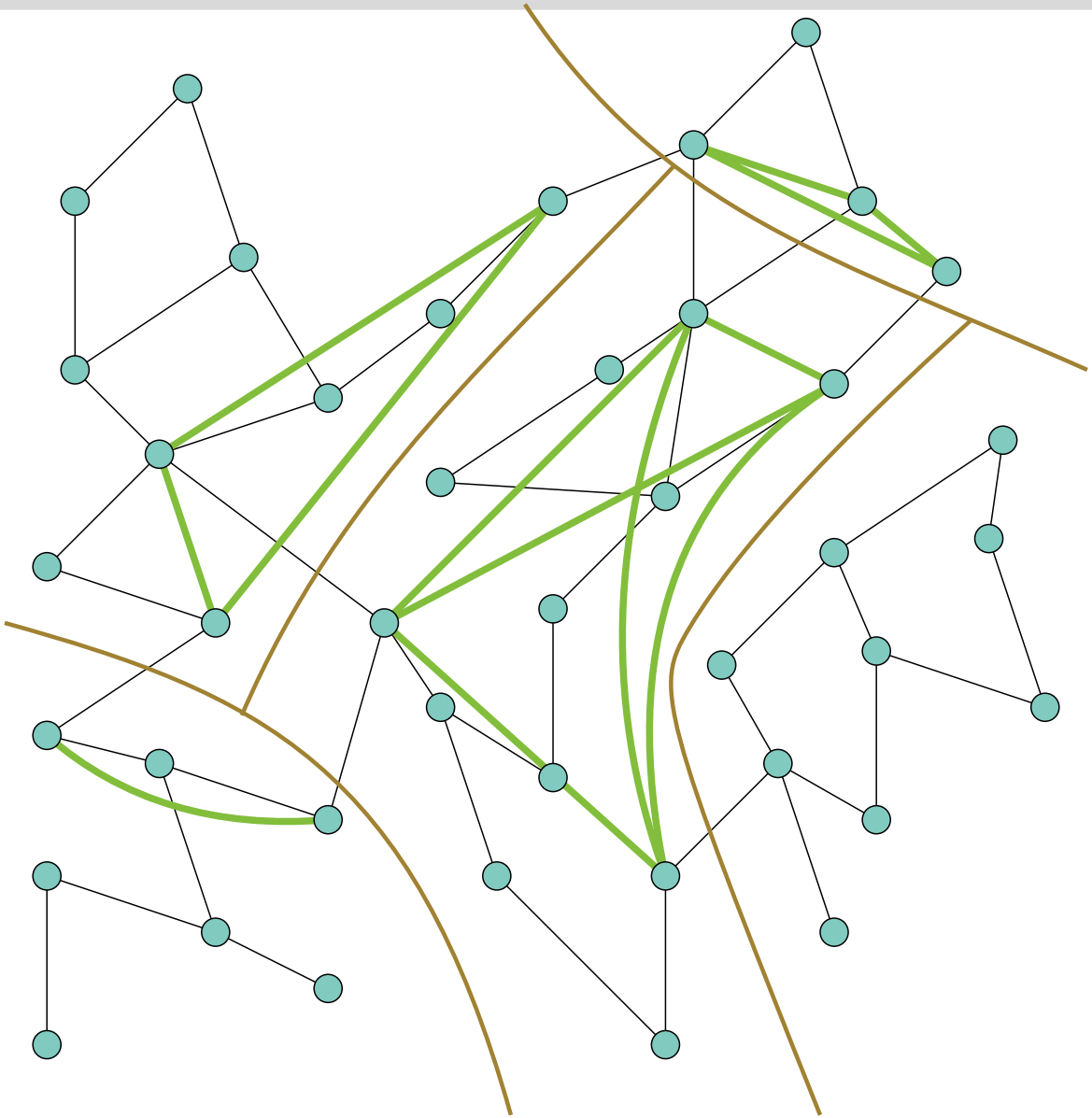
Ein Beispielgraph

Institut für Theoretische Informatik
Lehrstuhl Algorithmik



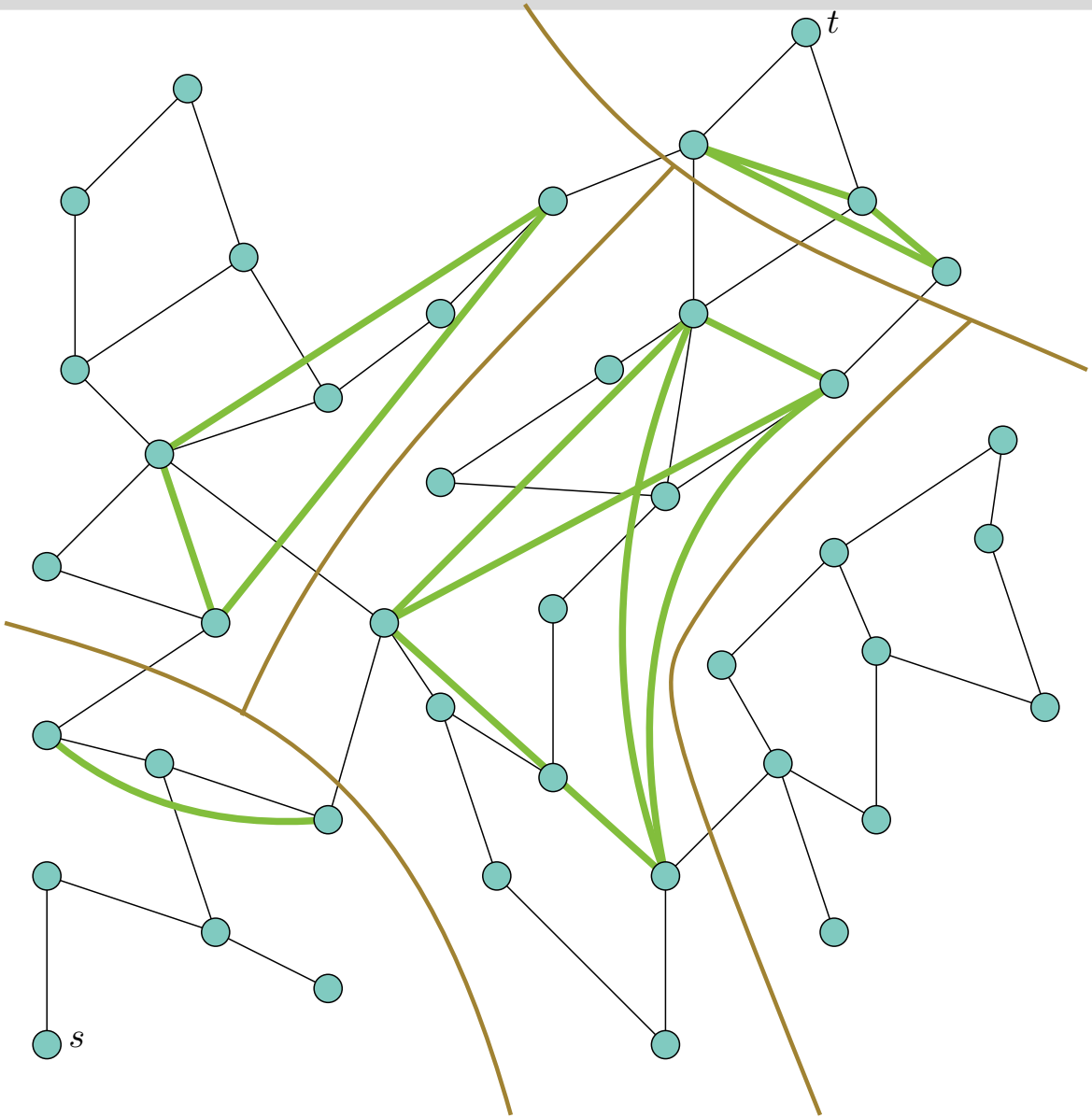
Eine Partition in 5 Zellen

Institut für Theoretische Informatik
Lehrstuhl Algorithmik



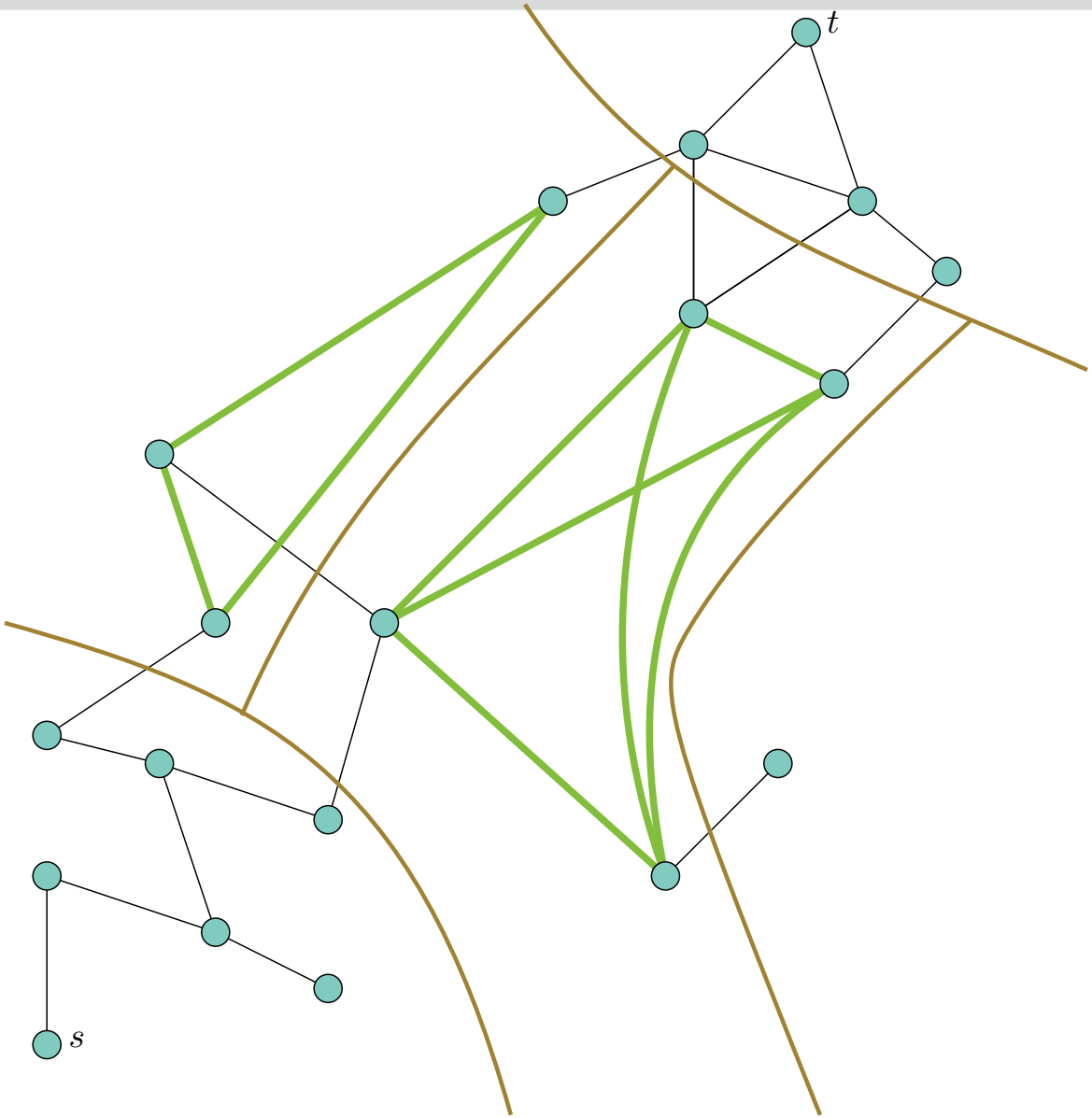
Der Overlay

Institut für Theoretische Informatik
Lehrstuhl Algorithmik



Eine Anfrage von s zu t

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

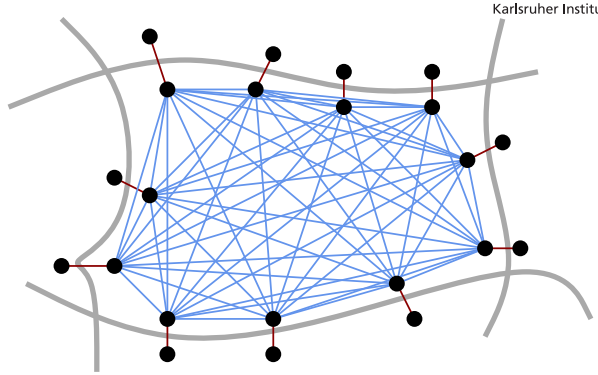


Die Knoten und Kanten, die während dieser Anfrage angeschaut werden

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Varianten

Ausdünnung des Overlaygraphen:

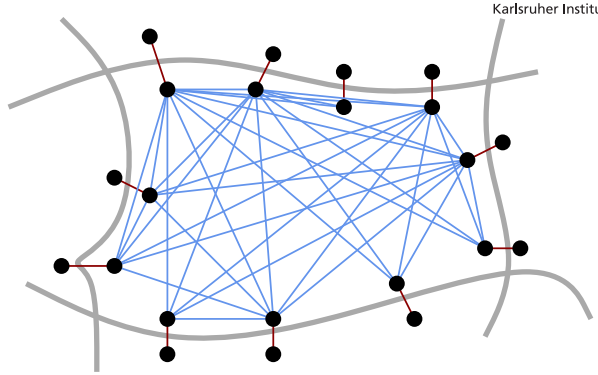


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Varianten

Ausdünnung des Overlaygraphen:

- entferne unnötige Kanten

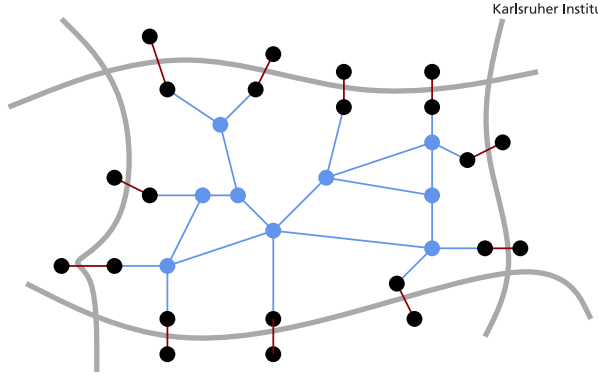


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Varianten

Ausdünnung des Overlaygraphen:

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

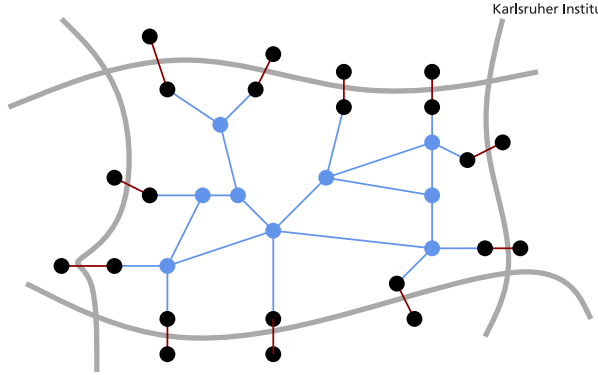


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Varianten

Ausdünnung des Overlaygraphen:

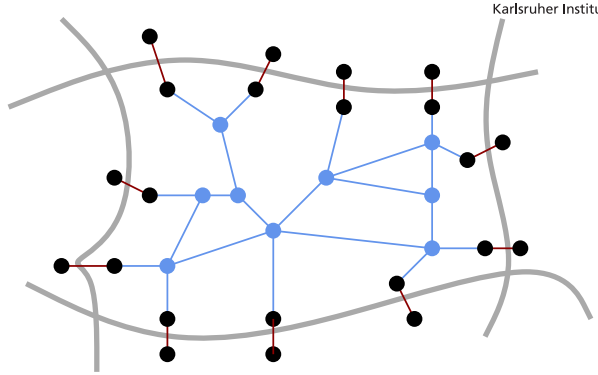
- entferne unnötige Kanten
- füge Knoten hinzu (aus Originalgraphen)
- etwas schnellere Anfragen



Varianten

Ausdünnung des Overlaygraphen:

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

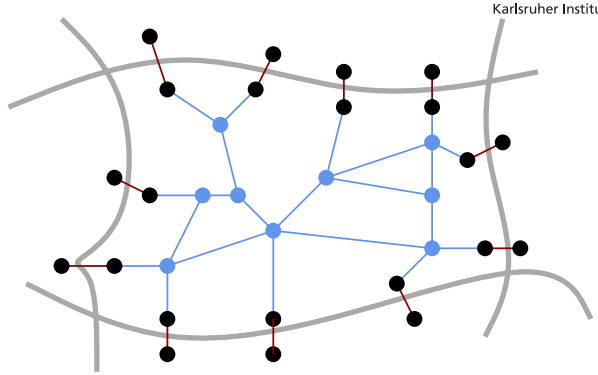


Kombination mit zielgerichteter Suche:

Varianten

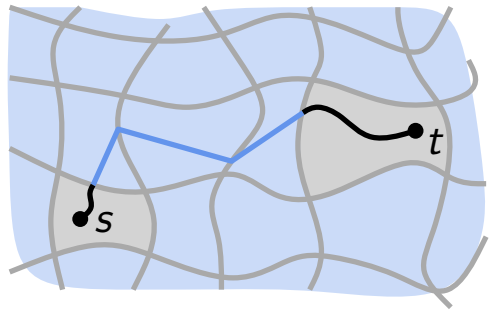
Ausdünnung des Overlaygraphen:

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



Kombination mit zielgerichteter Suche:

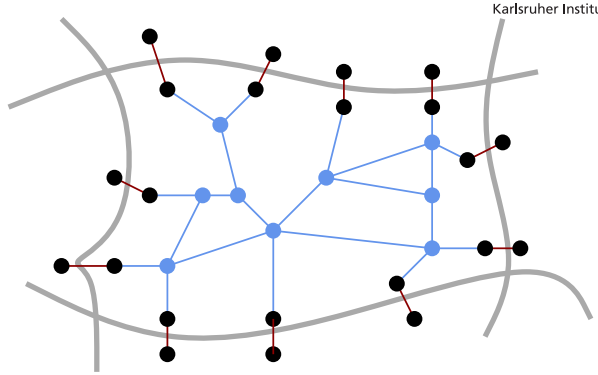
- nur auf (kleinem) Overlaygraphen
- ALT/Arc-Flags
- beschleunigt Anfragen, macht Vorberechnung und Queries komplizierter



Varianten

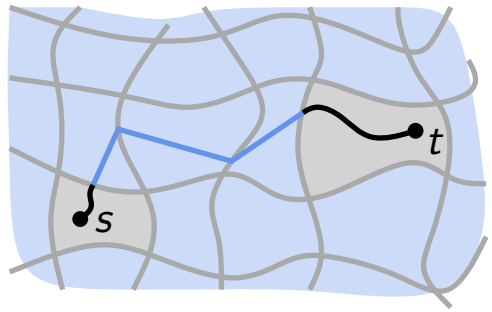
Ausdünnung des Overlaygraphen:

- 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 $\rightarrow$ wird meistens weggelassen

Multilevel-Overlays (MLO)

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

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

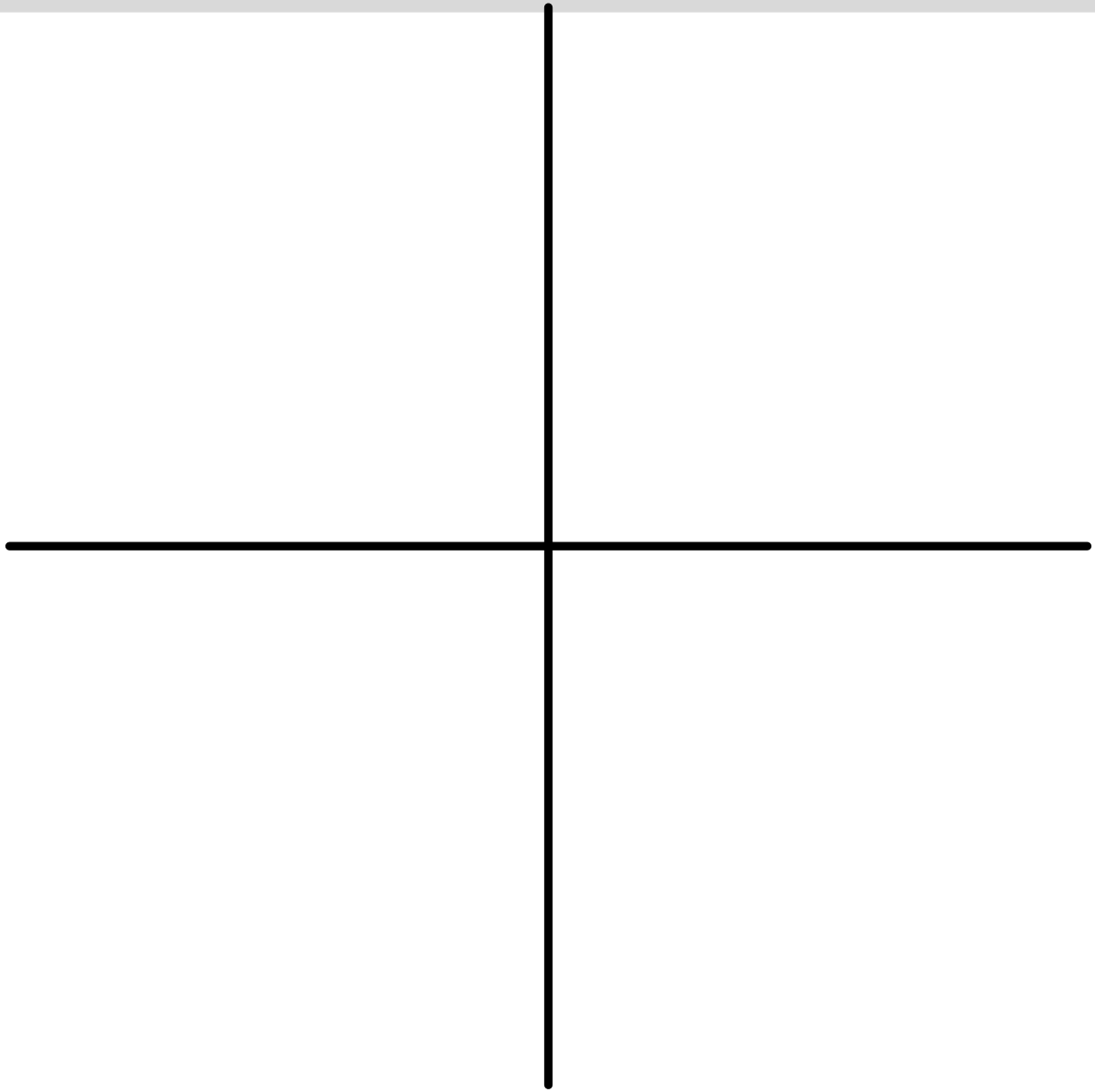
- Folge $G_0 = (V_0, E_0, \text{len}_0), \ldots, G_L = (V_L, E_L, \text{len}_L)$ von Graphen, so dass Distanzen in G_i wie in G_0
- G_i ist ein Overlay jedes G_j mit $j \leq i$

Idee

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

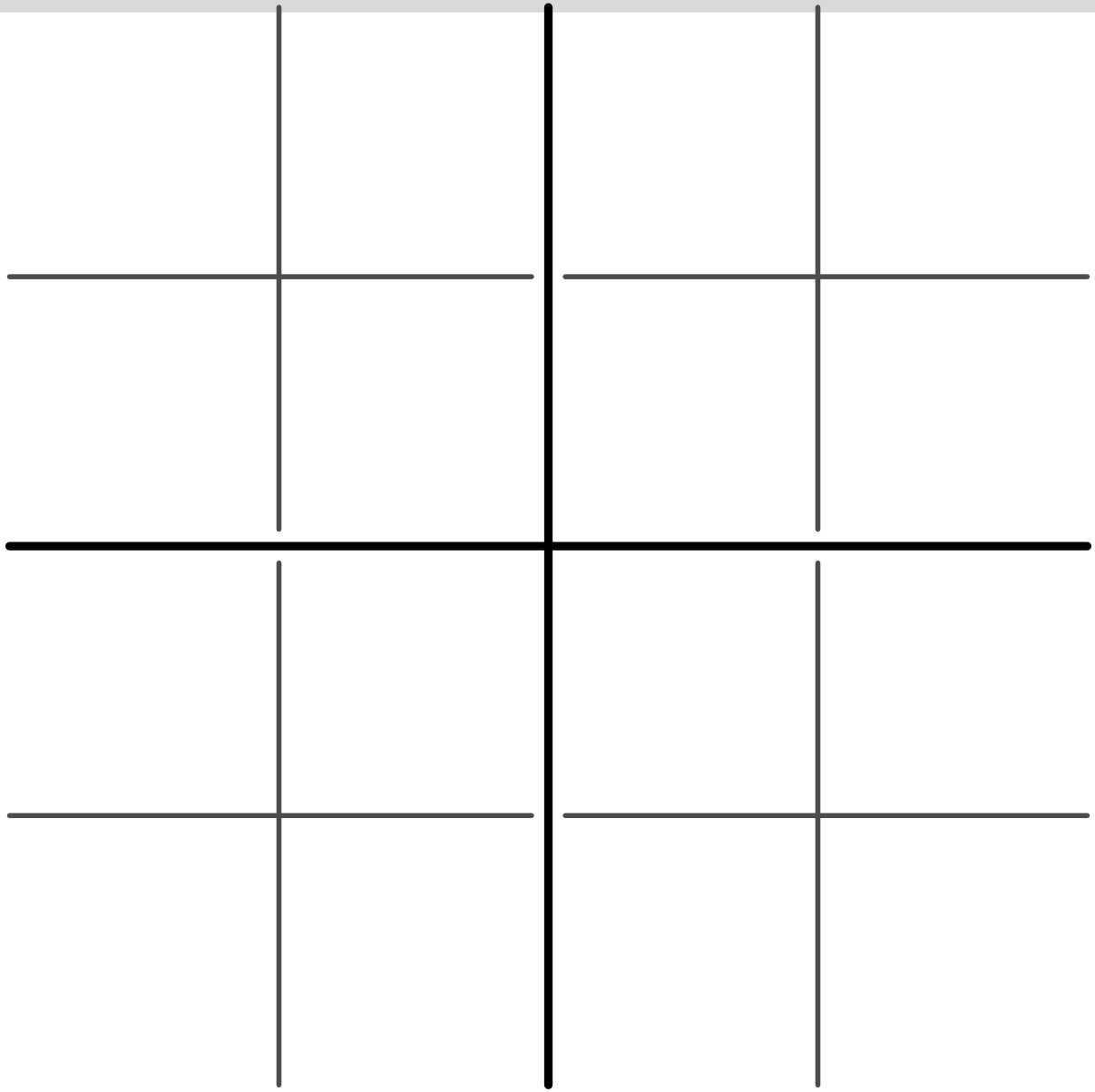
Partitionierung

- 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



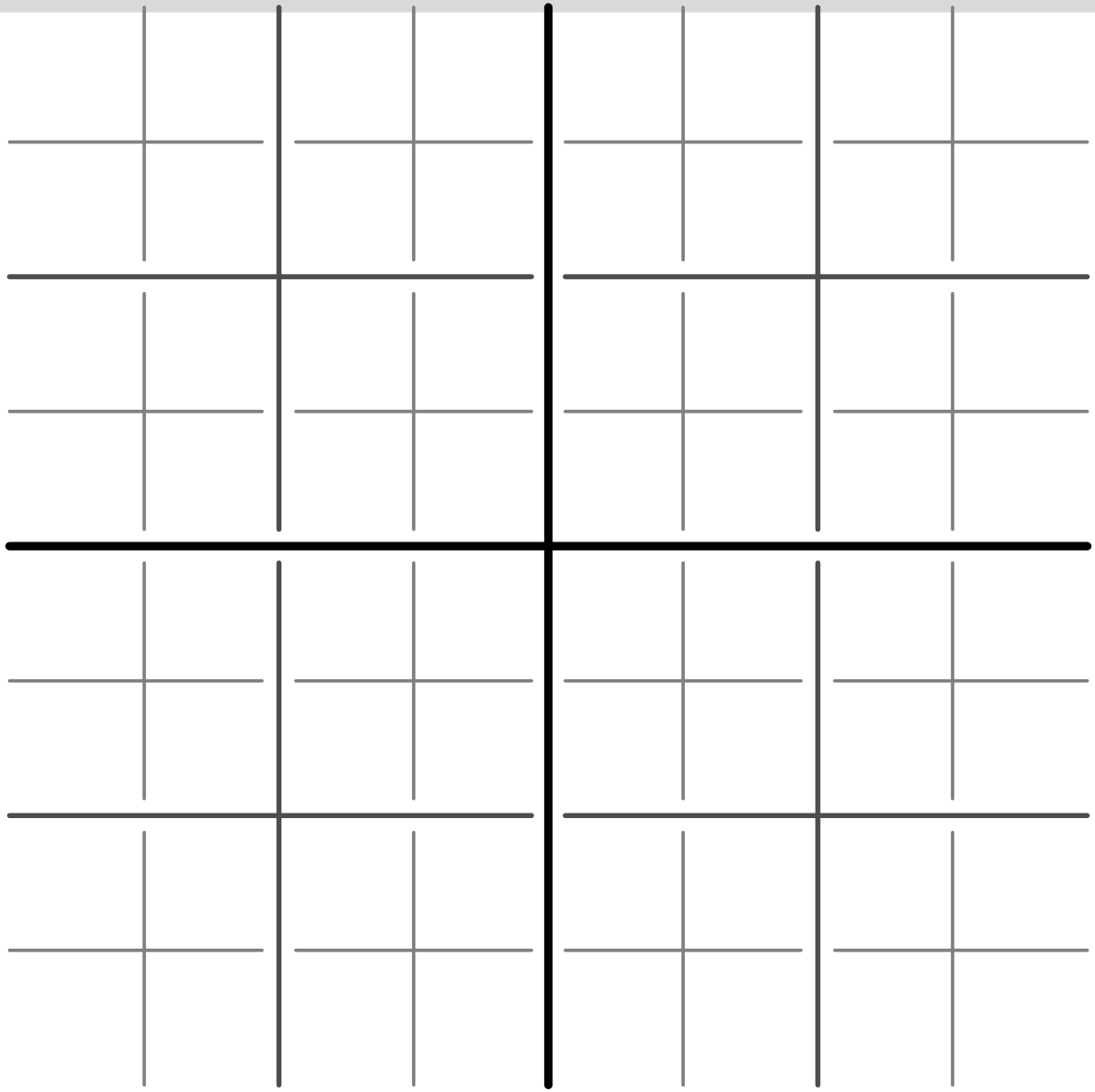
Zerteile den Graph in Zellen

Institut für Theoretische Informatik
Lehrstuhl Algorithmik



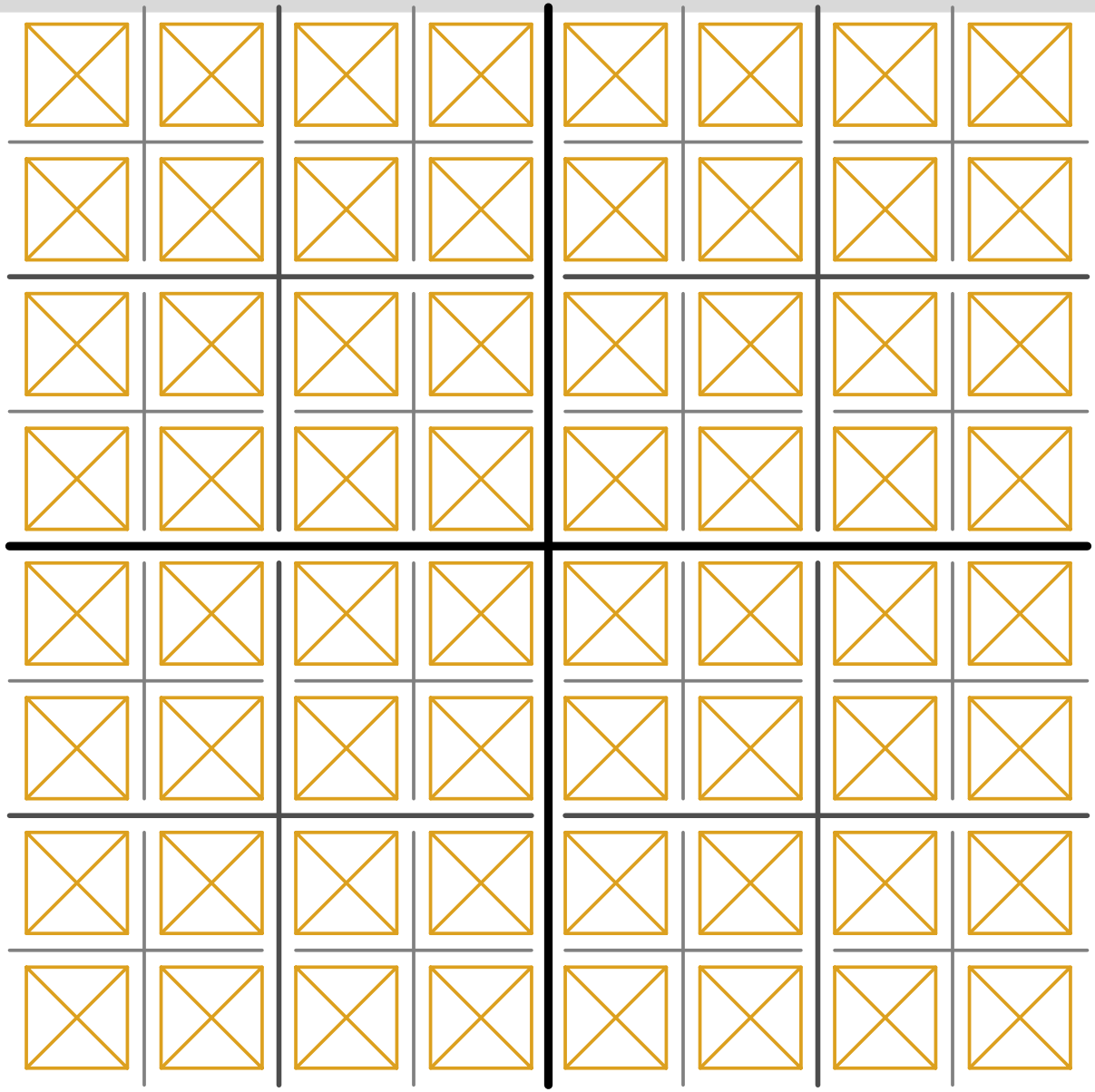
Zerteile die Zellen in Subzellen

Institut für Theoretische Informatik
Lehrstuhl Algorithmik



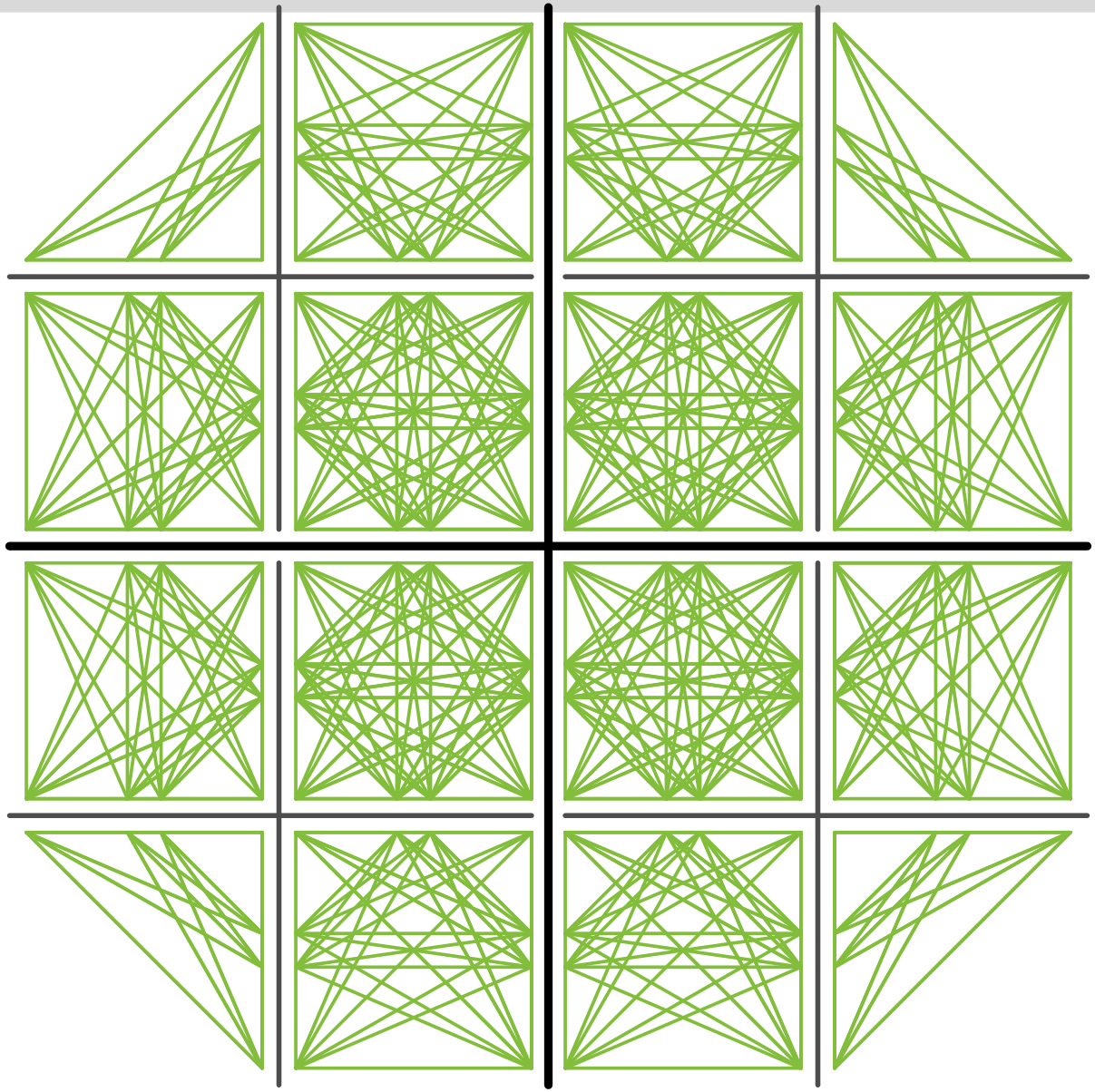
Zerteile die Subzellen in Subsubzellen

Institut für Theoretische Informatik
Lehrstuhl Algorithmik



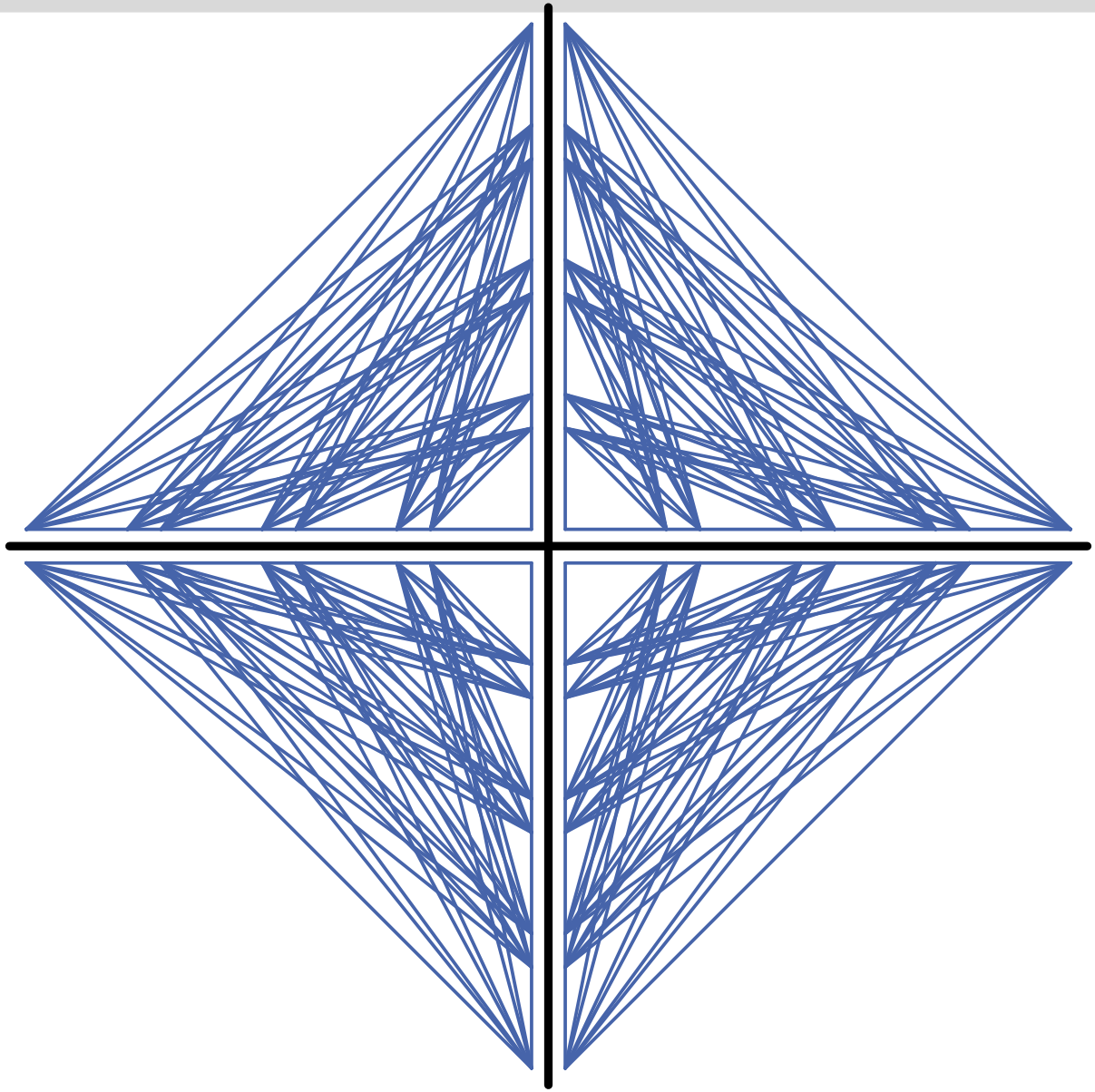
Baue für das unterste Level Overlays

Institut für Theoretische Informatik
Lehrstuhl Algorithmik



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

Institut für Theoretische Informatik
Lehrstuhl Algorithmik



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

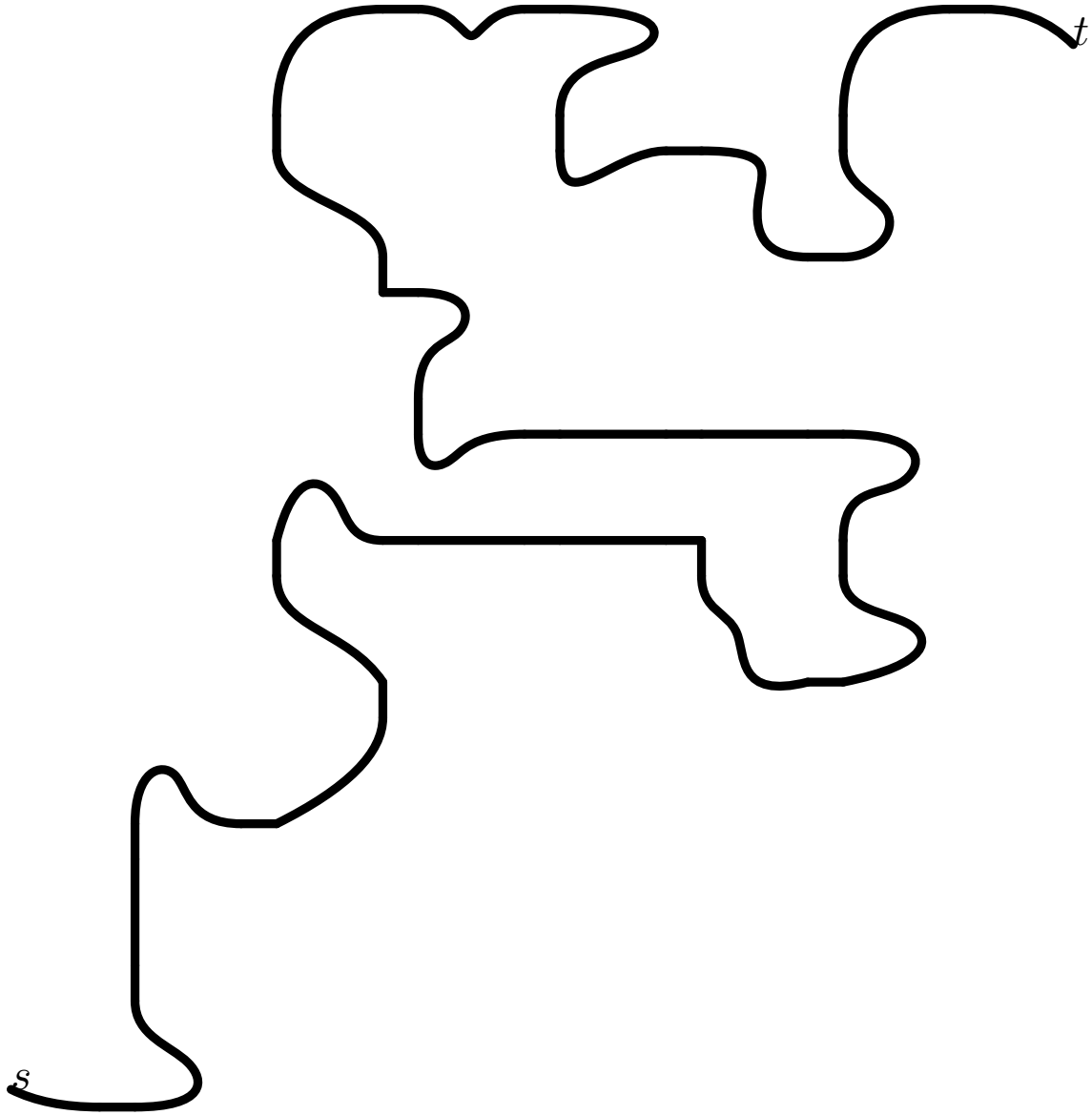
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

t

s

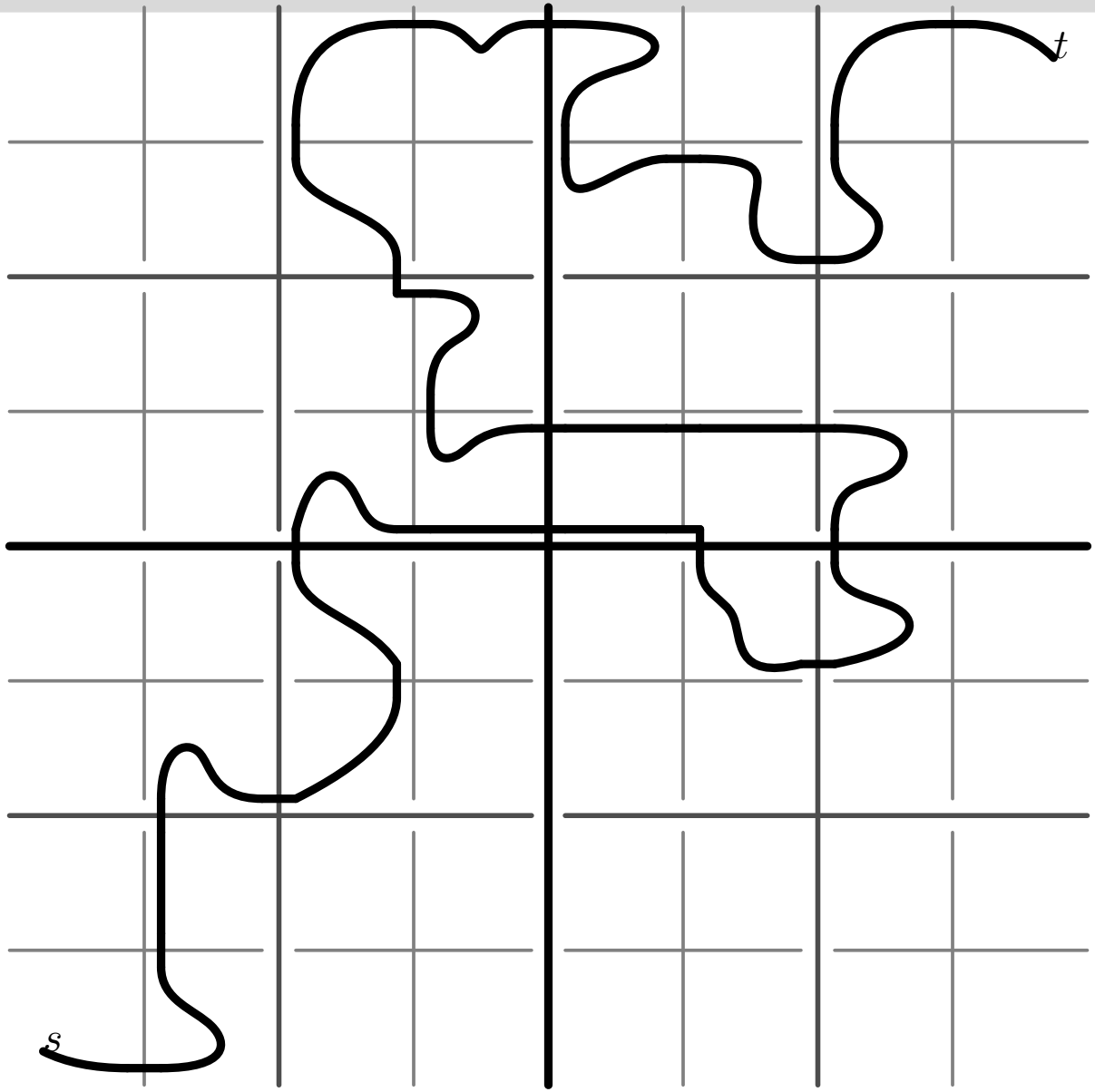
Betrachte eine *st*-Anfrage . . .

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

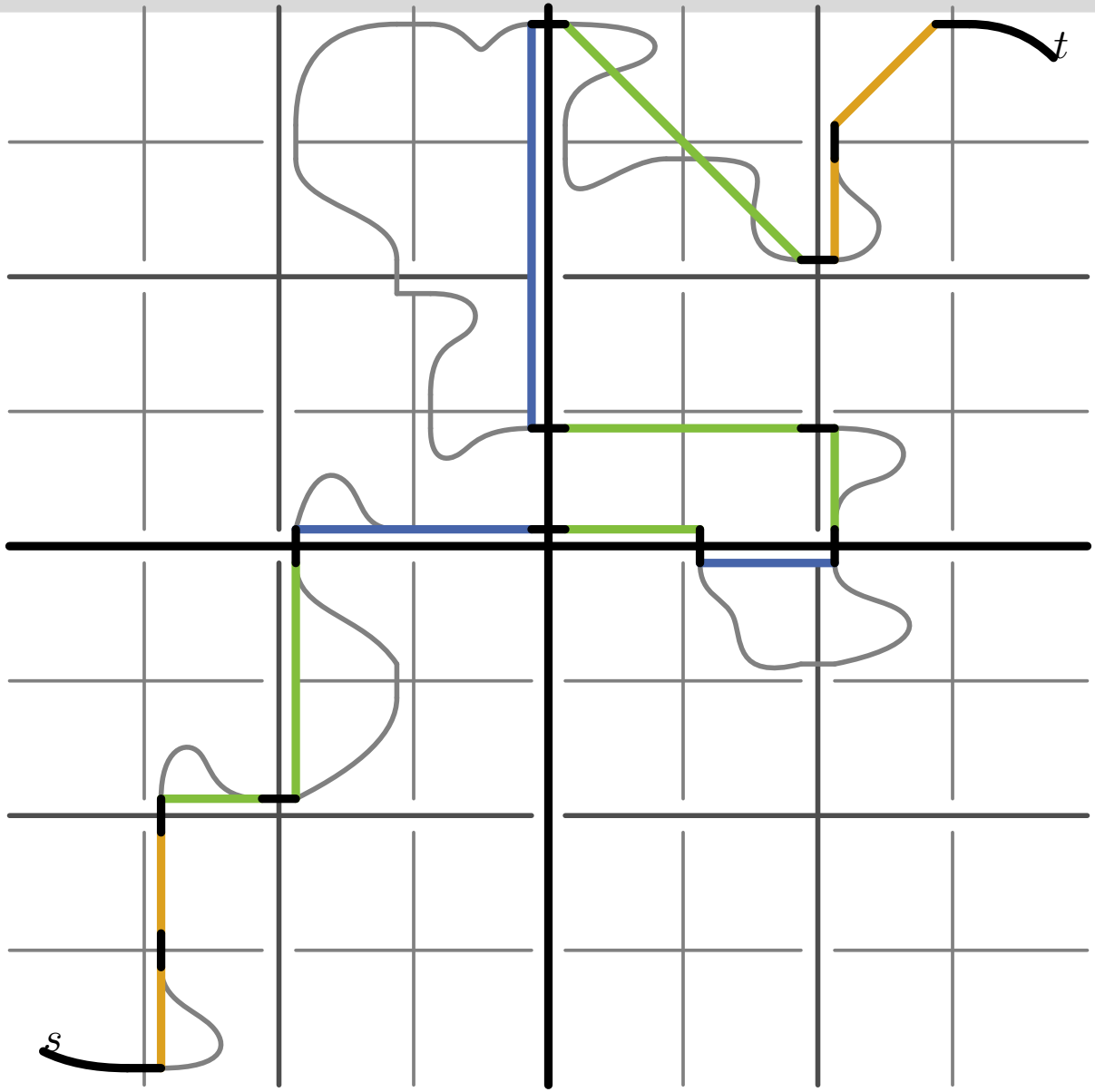


. . . und den dazu gehörigen kürzesten *st*-Pfad im Originalgraph

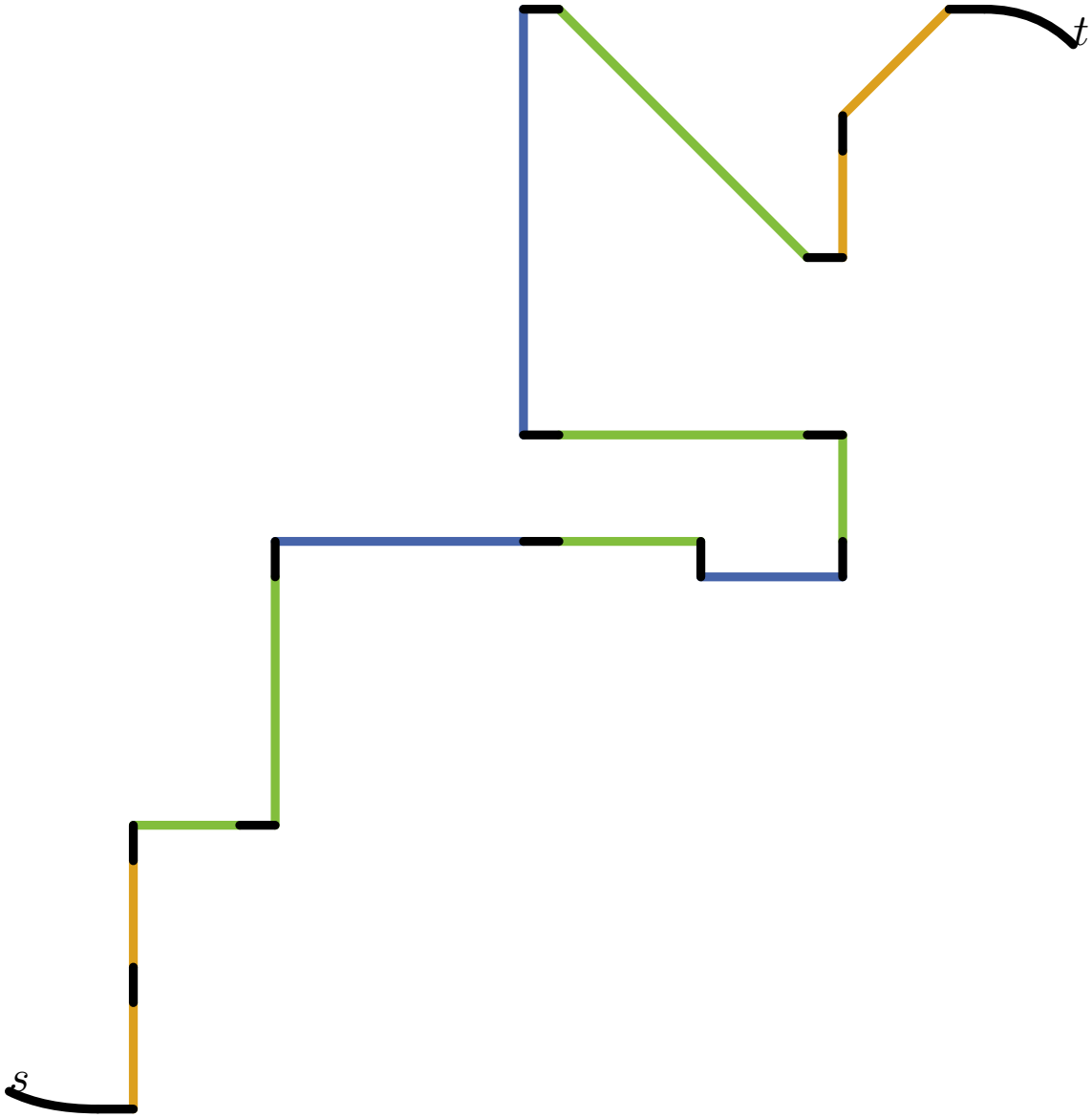
Achtung: Wir wissen nur, dass es ihn gibt. Wir kennen ihn noch nicht!



Betrachte nun die Partitionierung



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

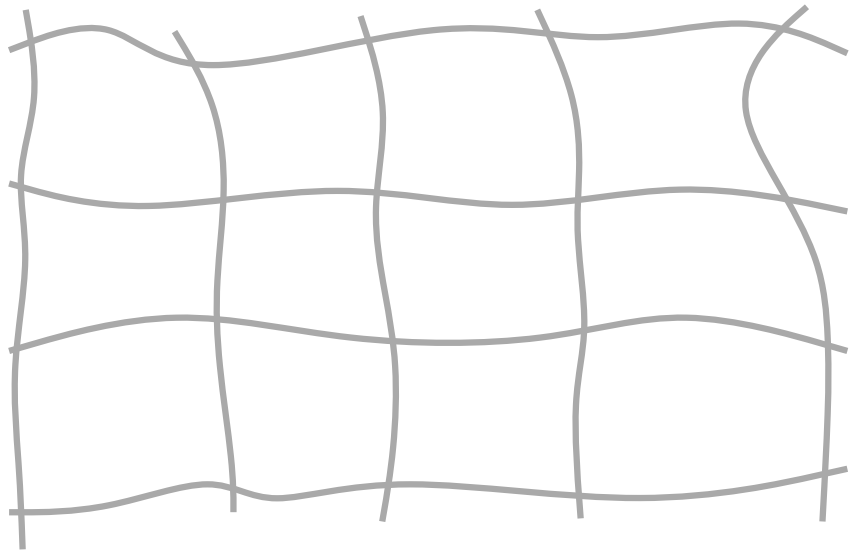


Farbe = Level der Kante: schwarz = Eingabe, orange = unterstes Level, grün = mittleres Level, blau = höchstes Level

Entpacken der Shortcuts

Von Shortcuts zu vollen Pfaden:

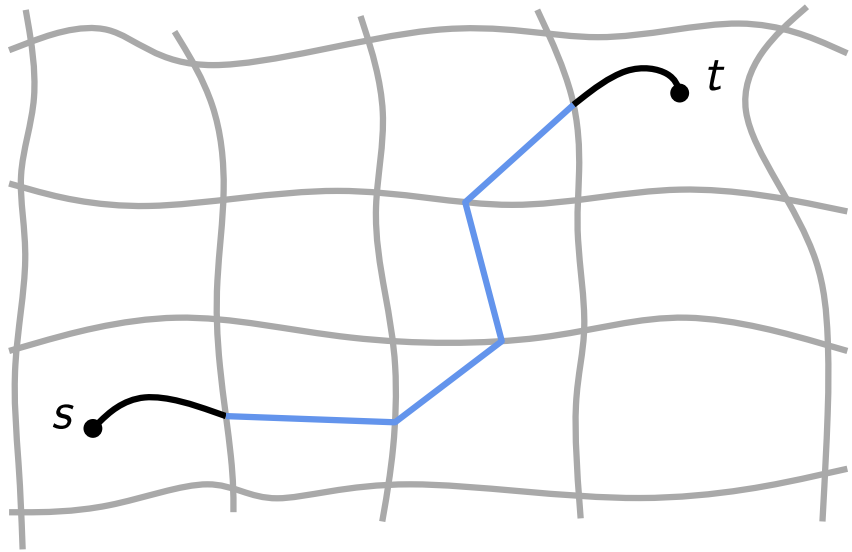
- bidirektionale Suche
- beschränkt auf die Zelle
- benutze untere Level rekursiv



Entpacken der Shortcuts

Von Shortcuts zu vollen Pfaden:

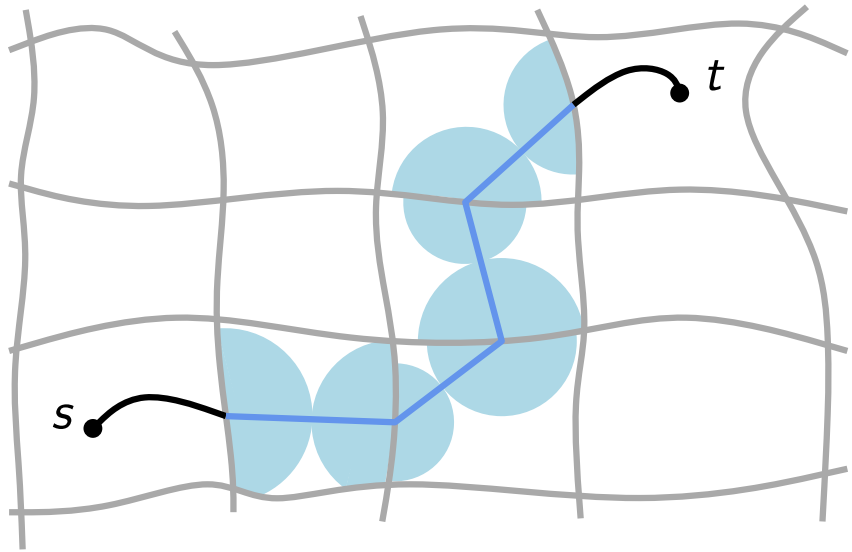
- bidirektionale Suche
- beschränkt auf die Zelle
- benutze untere Level rekursiv



Entpacken der Shortcuts

Von Shortcuts zu vollen Pfaden:

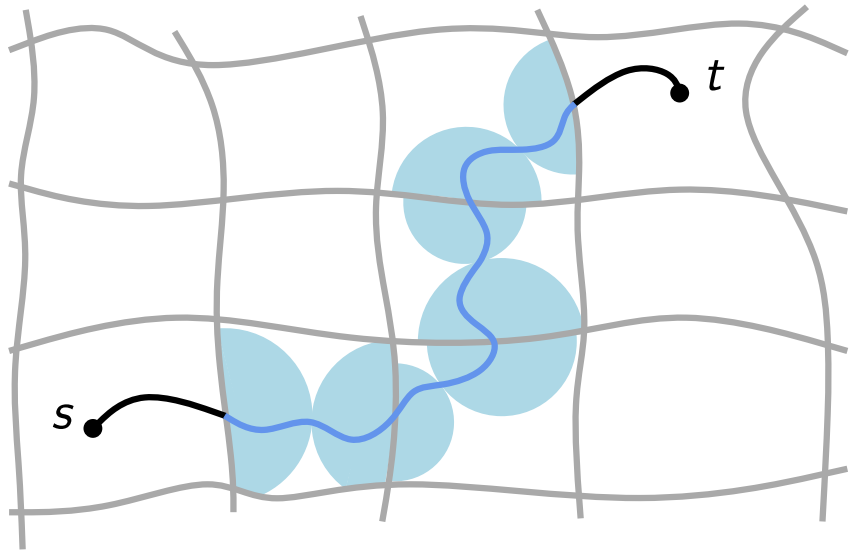
- bidirektionale Suche
- beschränkt auf die Zelle
- benutze untere Level rekursiv



Entpacken der Shortcuts

Von Shortcuts zu vollen Pfaden:

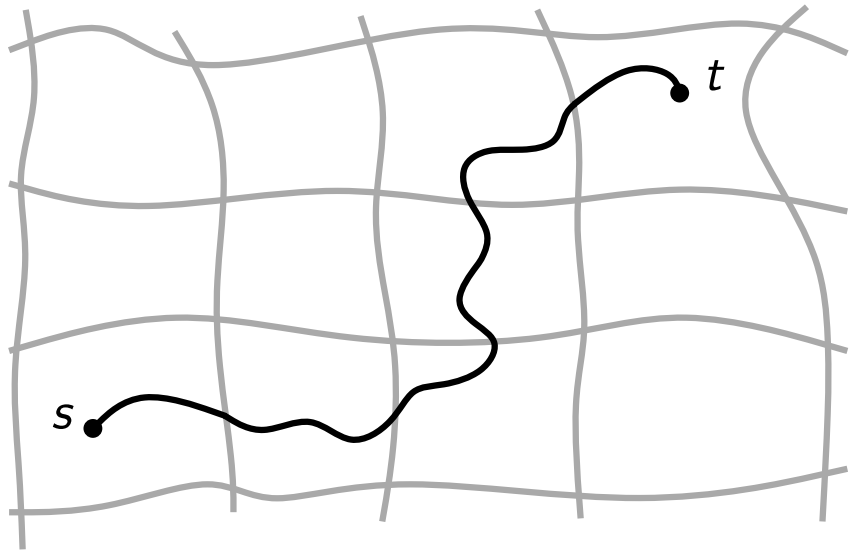
- bidirektionale Suche
- beschränkt auf die Zelle
- benutze untere Level rekursiv



Entpacken der Shortcuts

Von Shortcuts zu vollen Pfaden:

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



Wo kommt die Partitionierung her?

- Eigenes Forschungsfeld
- Gibt eine eigene Vorlesung von Sanders/Schulz/Schlag zu dem Thema
- Viele Algorithmen/Software Pakete
  - 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

Customizable Route Planning (CRP)

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

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

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

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

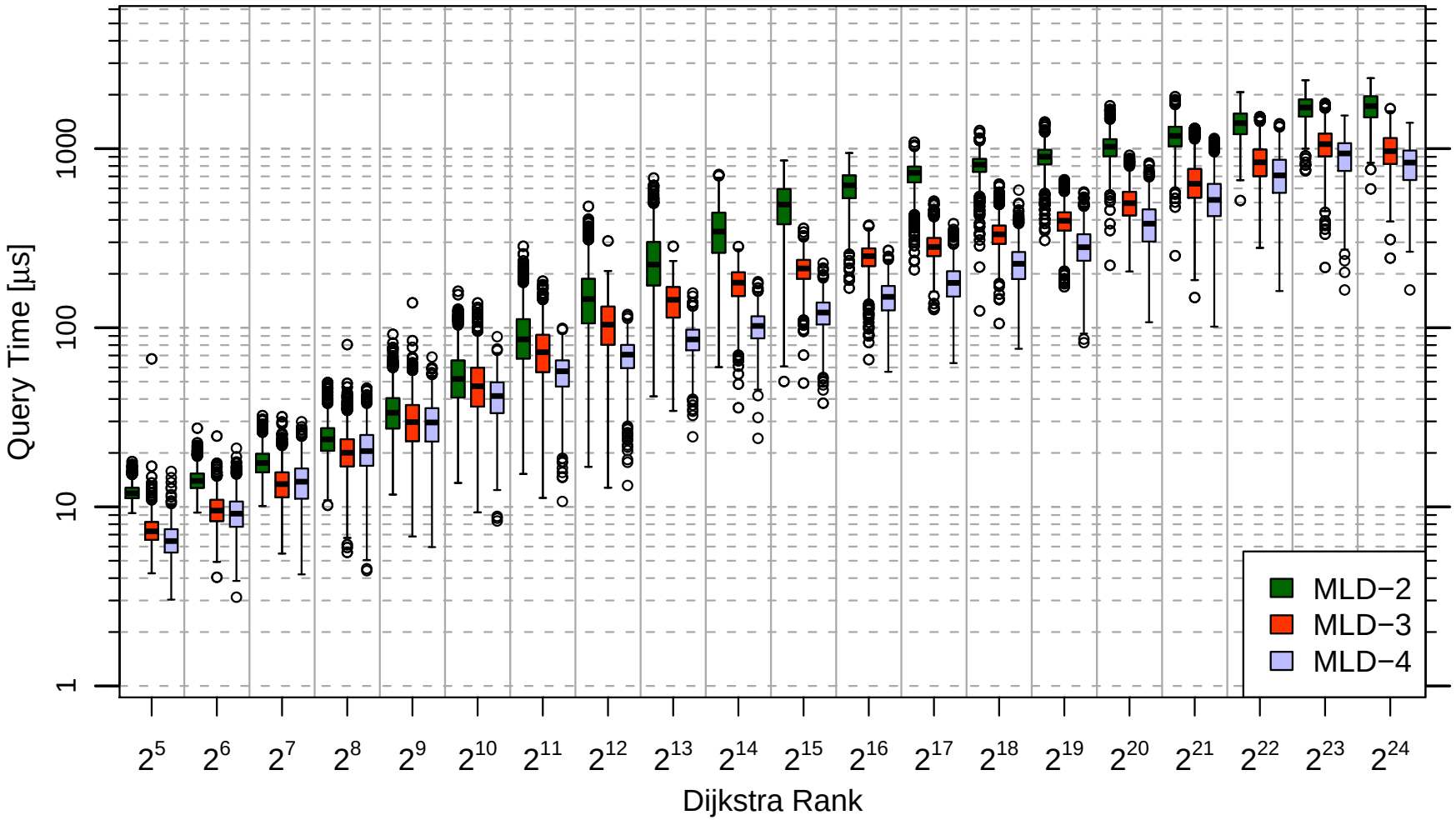
- Customizable Route Planning (CRP) = 3 Phasen Problemformulierung
- Multilevel-Overlays (MLO) = Algorithmus

- **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)

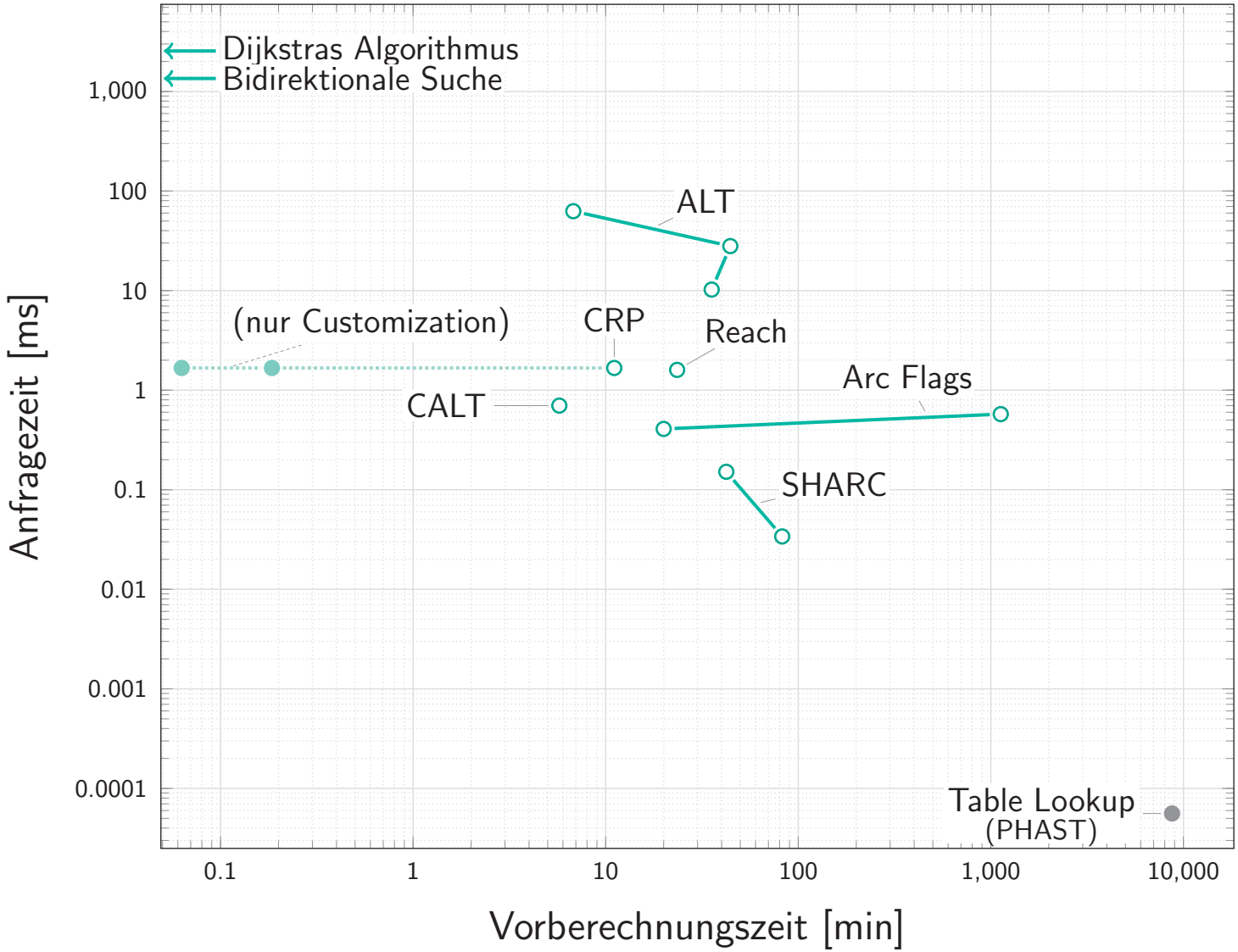
Performance

	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

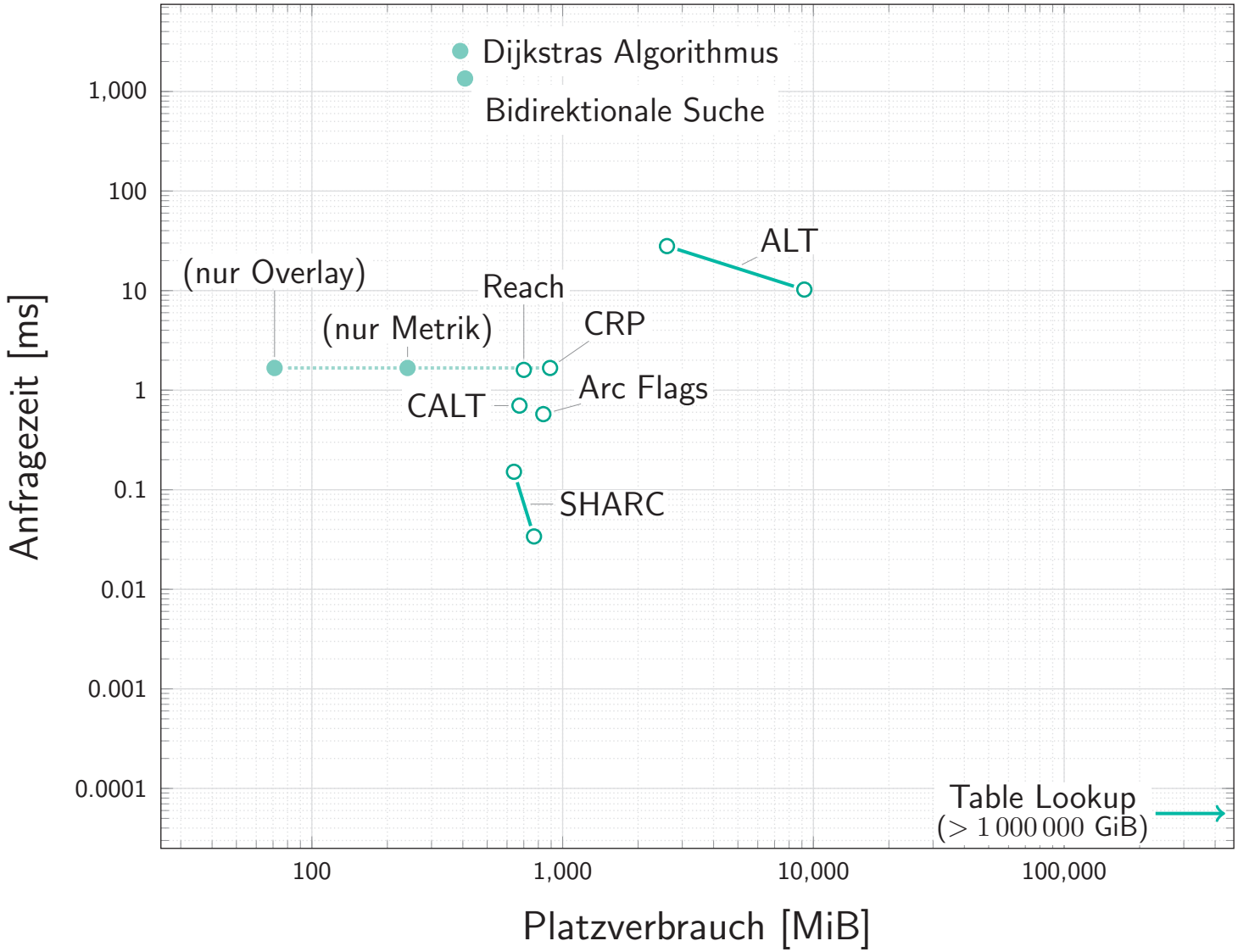


Übersicht bisherige Techniken



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Übersicht bisherige Techniken



Nächste Termine

Montag, 7.5.2018
Mittwoch, 9.5.2018
Montag, 14.5.2018

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.

Andrew V. Goldberg, Haim Kaplan, and Renato F. Werneck.

Reach for A*: Shortest Path Algorithms with Preprocessing.

In Camil Demetrescu, Andrew V. Goldberg, and David S. Johnson, editors, *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.

Literatur II


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.


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.