Algorithmen für Routenplanung

6. Vorlesung, Sommersemester 2018

Valentin Buchhold | 7. Mai 2018

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

Kürzeste Wege in Straßennetzwerken

Beschleunigungstechniken (Fortsetzung)

Thema: Contraction Hierarchies (CH)

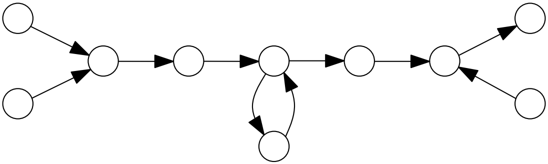
- CH Basisvariante
- CH im Detail
    - Stall-on-demand
    - Pfadentpackung
    - Top-Down-Knotenordnungen

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

CH Basisvariante

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

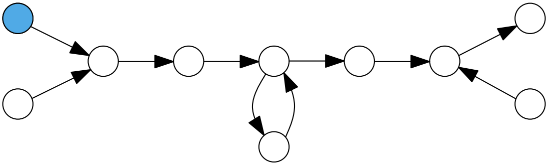
Shortcut



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

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

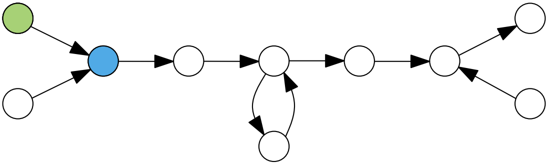
Shortcut



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

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

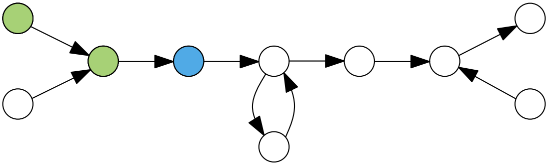
Shortcut



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

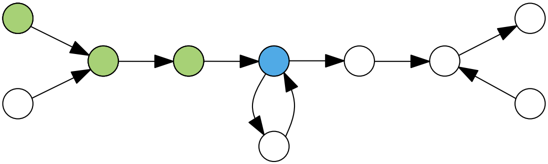
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Shortcut



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

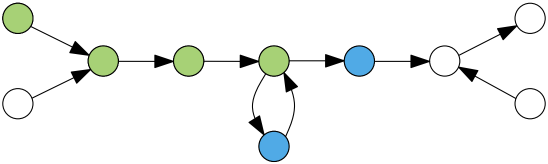
Shortcut



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

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

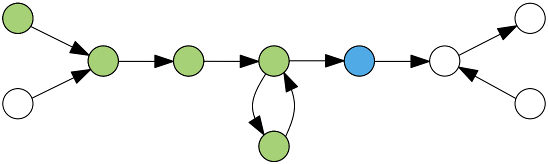
Shortcut



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

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

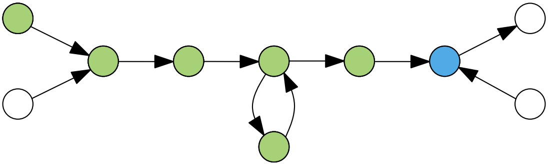
Shortcut



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

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

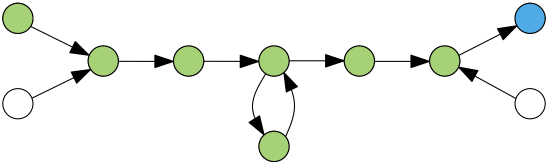
Shortcut



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

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Shortcut

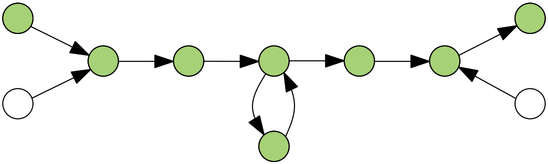


Dijkstra’s Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Shortcut

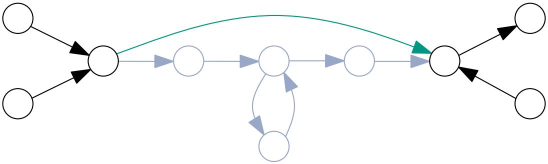
KIT
Karlsruher Institut für Technologie



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

Shortcut

Karlsruher Institut für Technologie

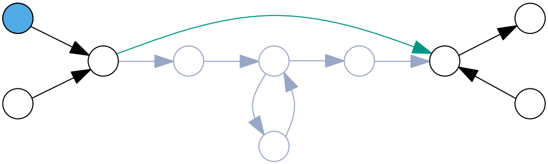


Idee: Füge Shortcut-Kante ein. Grauer Teilgraph muss nur angeschaut werden, wenn *s* oder *t* drin liegt.

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Shortcut

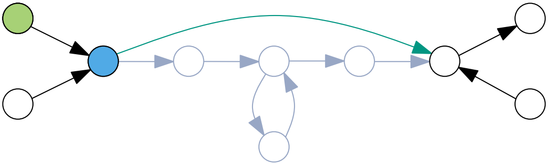
Karlsruher Institut für Technologie



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

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

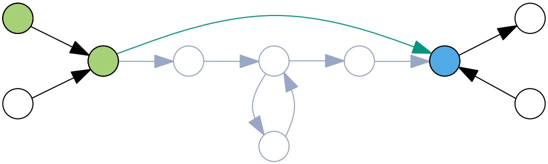
Shortcut



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

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

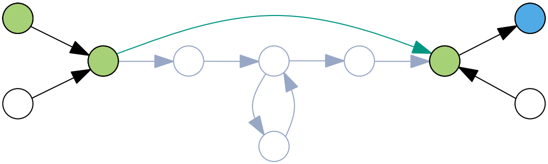
Shortcut



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

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

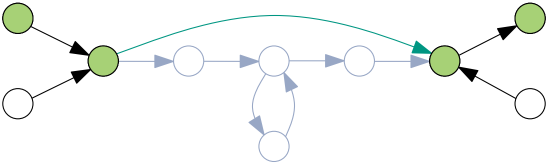
Shortcut



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

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

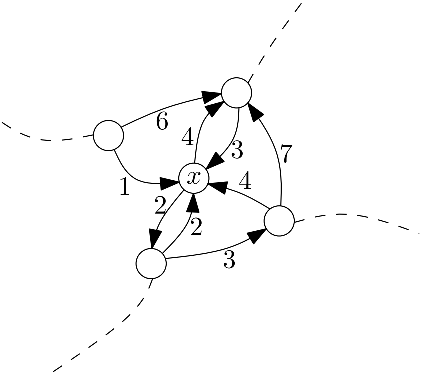
Shortcut



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

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

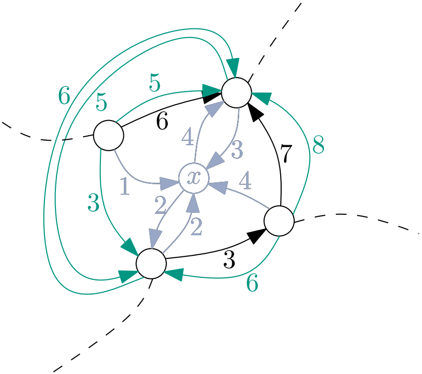
Knotenkontraktion von x



Kontraktion von x: Lösche x und füge Shortcuts zwischen Nachbarn ein, um die Distanzen zwischen allen Knoten zu erhalten

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

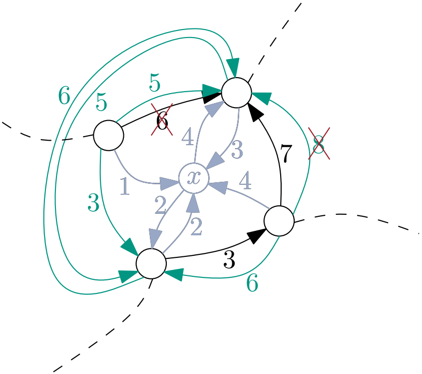
Knotenkontraktion von *x*



Kontraktion von *x*: Lösche *x* und füge Shortcuts zwischen Nachbarn ein, um die Distanzen zwischen allen Knoten zu erhalten

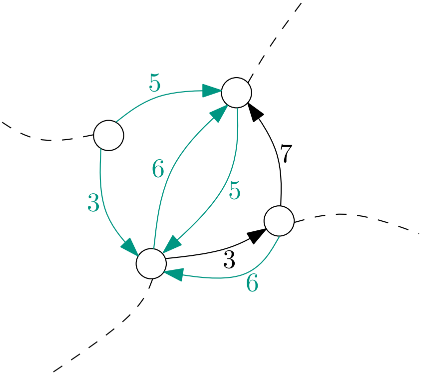
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Knotenkontraktion von *x*

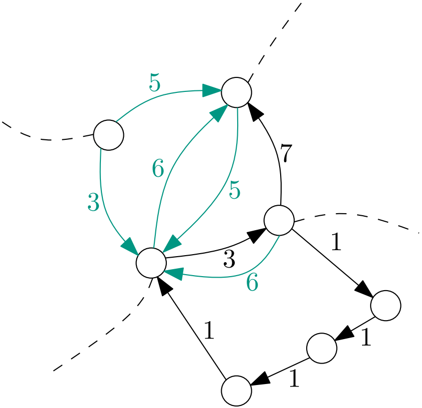


Bei Mehrfachkanten: Längere Kanten verwerfen

Knotenkontraktion von x

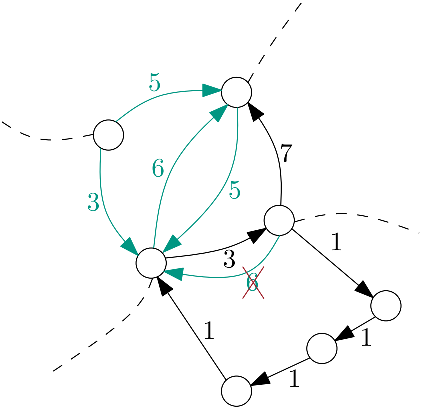


Knotenkontraktion von x



Falls es einen kürzeren Pfad durch den Restgraphen gibt, dann kann man einen Shortcut auch verwerfen.
Suche nach solchem Pfad heißt Zeugensuche/Witness Search

Knotenkontraktion von x



Falls es einen kürzeren Pfad durch den Restgraphen gibt, dann kann man einen Shortcut auch verwerfen.
Suche nach solchem Pfad heißt Zeugensuche/Witness Search

Zeugensuche

- Es seien y und z zwei Nachbarn des kontrahierten Knoten x
- Wir fügen einen Shortcut (y, z) mit Gewicht $\text{len}(y, x) + \text{len}(x, z)$ ein, wenn $y \rightarrow x \rightarrow z$ der einzige kürzeste $y - z$-Weg ist
- Zum Überprüfen, ob es einen kürzeren Weg gibt, startet man einen Dijkstra von y aus nach z. Diese Suche kann teuer sein. Mögliche Optimierungen:
  - Suche darf nicht über den Knoten x gehen
  - Bidirektionale Variante von Dijkstras Algorithmus
  - Wenn die Suchen sich treffen, kann man abbrechen
  - Wenn die Suchfront größer wird als $\text{len}(y, x) + \text{len}(x, z)$ kann man abbrechen
- Wenn das immer noch zu langsam ist: Suche nach k Schritten abbrechen. Eventuell gibt es einen Pfad, den wir nicht finden. Das führt zu zusätzlichen Shortcuts, aber das ist kein Problem bzgl. der Korrektheit.

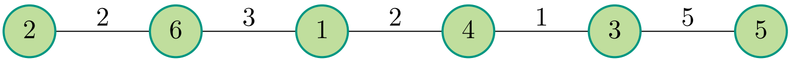
Contraction Hierarchy

Grundidee

- Eingabegraph G
- Ordne Knoten von G nach "Wichtigkeit": $v_1 \dots v_n$
- Kontrahiere Knoten iterativ aus G heraus
  - zuerst den "unwichtigsten" Knoten v_1
  - den "wichtigsten" Knoten v_n als letztes
- Graph mit Shortcuts heißt augmentierter Graph

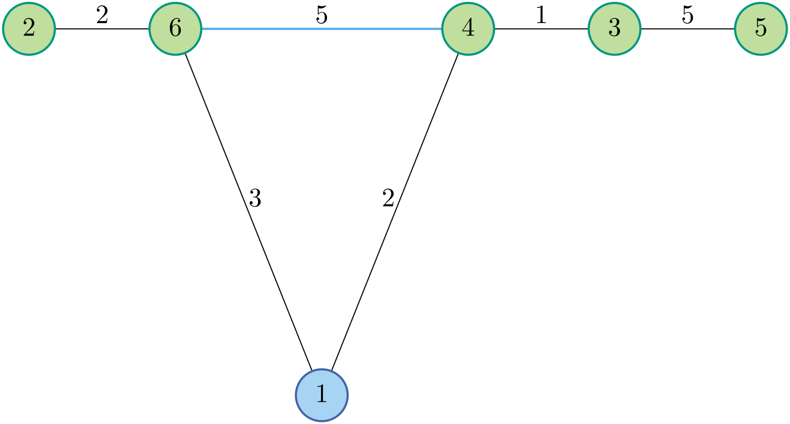
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Contraction Hierarchy



Knoten nummeriert nach “Wichtigkeit”

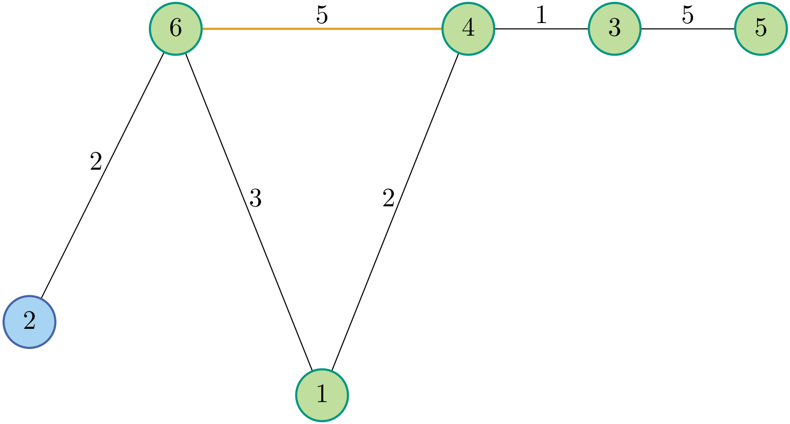
Contraction Hierarchy



Knoten nummeriert nach “Wichtigkeit”

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

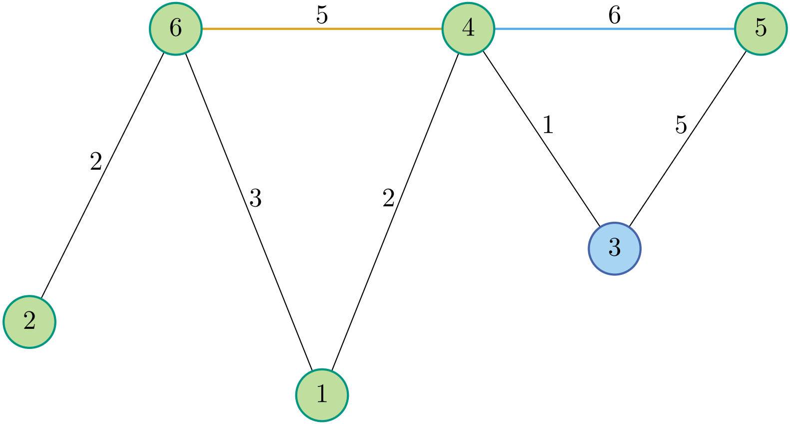
Contraction Hierarchy



Knoten nummeriert nach “Wichtigkeit”

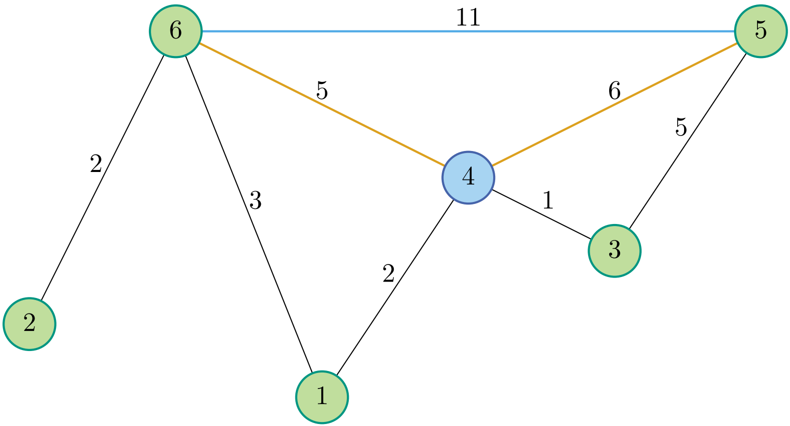
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Contraction Hierarchy



Knoten nummeriert nach “Wichtigkeit”

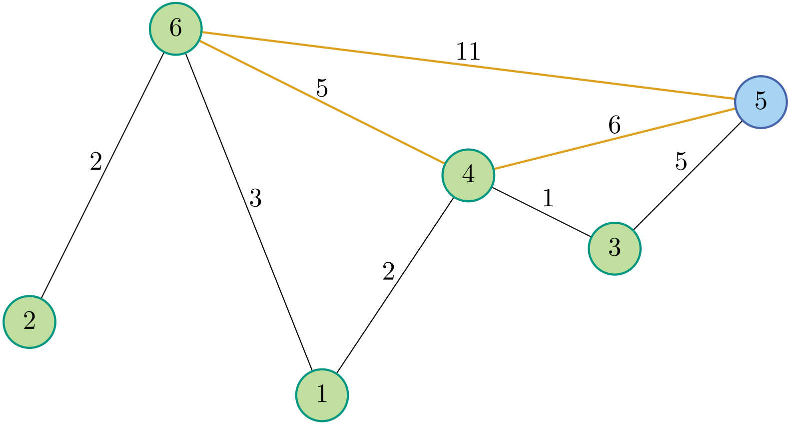
Contraction Hierarchy



Knoten nummeriert nach "Wichtigkeit"

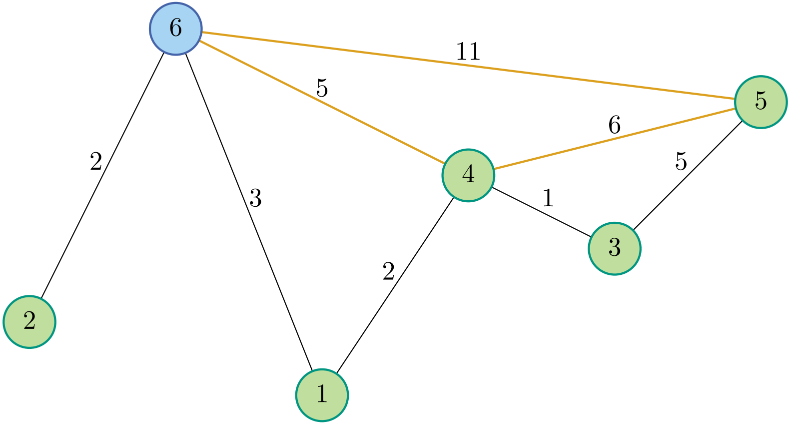
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Contraction Hierarchy



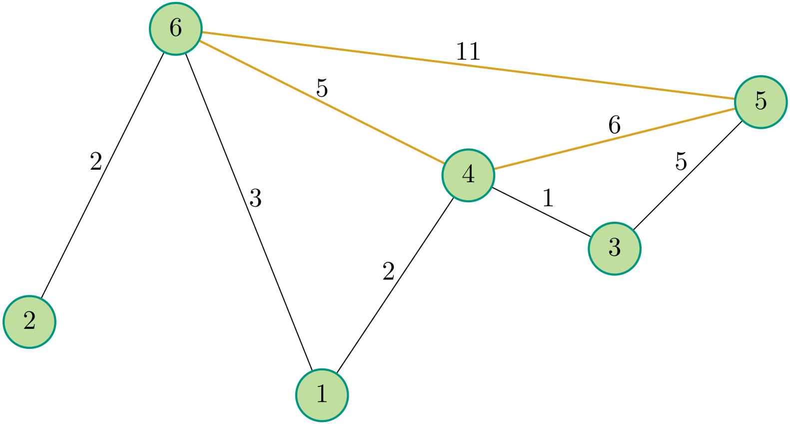
Knoten nummeriert nach “Wichtigkeit”

Contraction Hierarchy



Knoten nummeriert nach “Wichtigkeit”

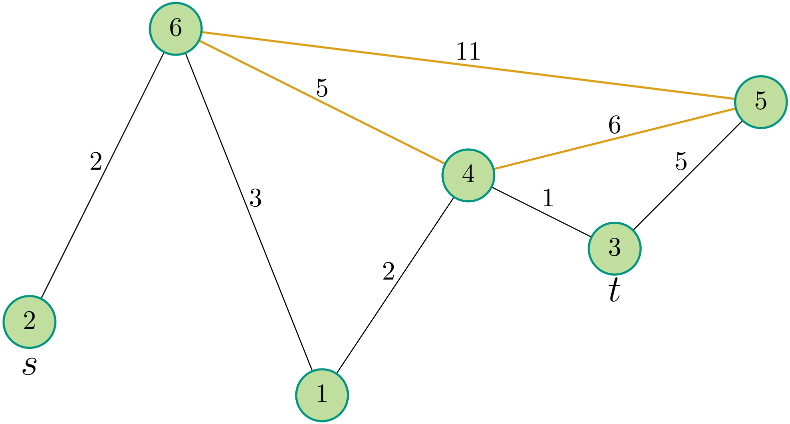
Contraction Hierarchy



Knoten nummeriert nach “Wichtigkeit”

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

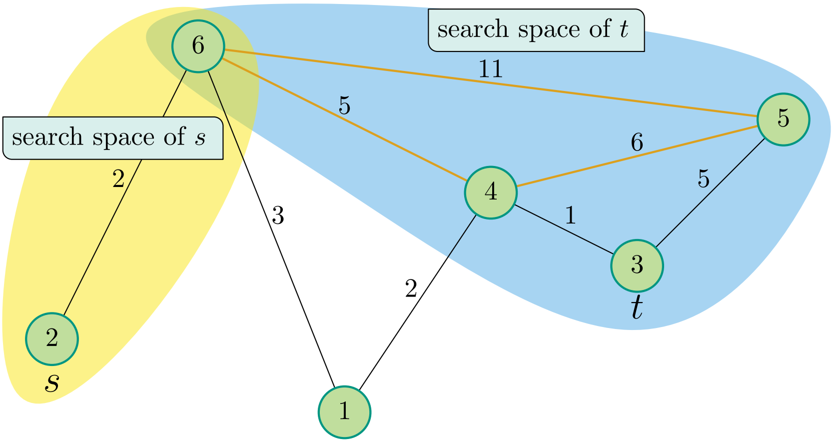
Contraction Hierarchy



Knoten nummeriert nach “Wichtigkeit”

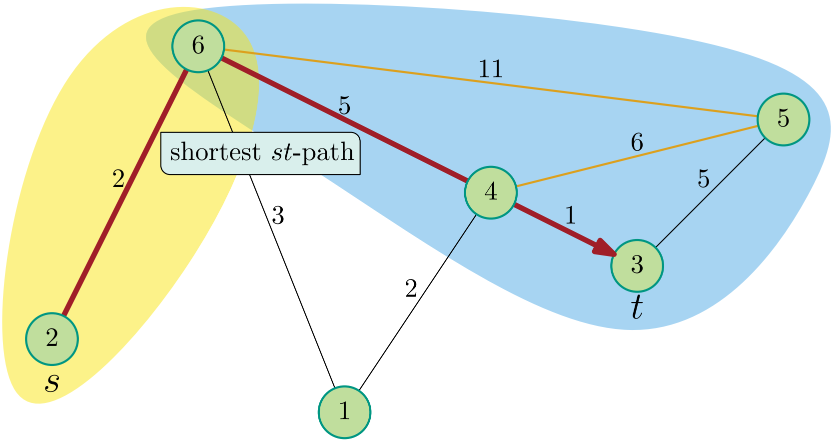
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Contraction Hierarchy



Knoten nummeriert nach “Wichtigkeit”

Contraction Hierarchy



Für jeden ursprünglichen kürzesten Weg gibt es einen hoch-runter-Pfad

Anfrage

- Bidirektionale Variante von Dijkstras Algorithmus
- Verfolge nur Kanten zu wichtigeren Knoten
- Vorwärtssuche findet den “hoch”-Teil des Pfads
- Rückwärtssuche findet den “runter”-Teil des Pfads
- Abbruch, wenn der min-key beider Queues größer ist als der bisher kürzeste gefundene Pfad

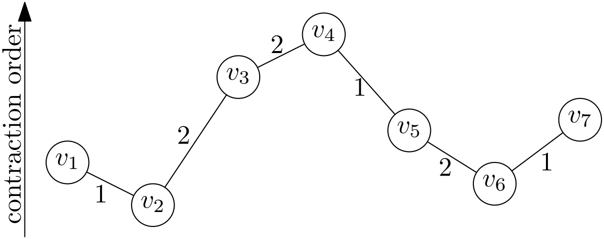
Korrektheit

- Die Vorwärtssuche findet nur Aufwärtspfade.
- Die Rückwärtssuche findet nur Abwärtspfade.
- Gemeinsam werden nur Pfade gefunden die hoch und dann wieder runter gehen.
- Wir müssen also beweisen, dass es in G^+ immer einen kürzesten hoch-runter Pfad gibt.

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Korrektheit

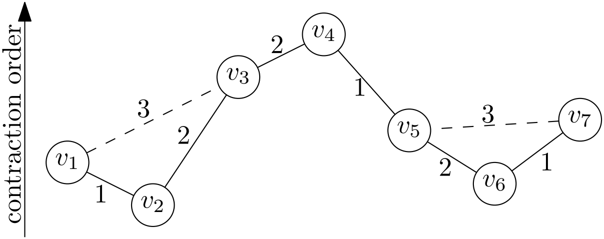
- Die Vorwärtssuche findet nur Aufwärtspfade.
- Die Rückwärtssuche findet nur Abwärtspfade.
- Gemeinsam werden nur Pfade gefunden die hoch und dann wieder runter gehen.
- Wir müssen also beweisen, dass es in G^+ immer einen kürzesten hoch-runter Pfad gibt.



- Es gibt einen kürzesten Weg P
- Wenn P kein hoch-runter Pfad ist, dann gibt es einen Knoten der höhere Nachbarn hat
- Dann gibt es auch Shortcut oder Zeugen zw. den Nachbarn

Korrektheit

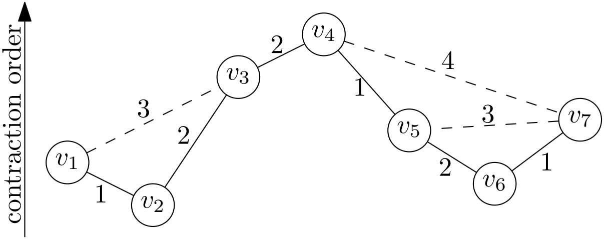
- Die Vorwärtssuche findet nur Aufwärtspfade.
- Die Rückwärtssuche findet nur Abwärtspfade.
- Gemeinsam werden nur Pfade gefunden die hoch und dann wieder runter gehen.
- Wir müssen also beweisen, dass es in G^+ immer einen kürzesten hoch-runter Pfad gibt.



- Es gibt einen kürzesten Weg P
- Wenn P kein hoch-runter Pfad ist, dann gibt es einen Knoten der höhere Nachbarn hat
- Dann gibt es auch Shortcut oder Zeugen zw. den Nachbarn

Korrektheit

- Die Vorwärtssuche findet nur Aufwärtspfade.
- Die Rückwärtssuche findet nur Abwärtspfade.
- Gemeinsam werden nur Pfade gefunden die hoch und dann wieder runter gehen.
- Wir müssen also beweisen, dass es in G^+ immer einen kürzesten hoch-runter Pfad gibt.

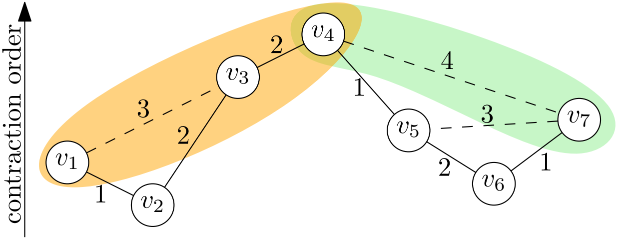


- Es gibt einen kürzesten Weg P
- Wenn P kein hoch-runter Pfad ist, dann gibt es einen Knoten der höhere Nachbarn hat
- Dann gibt es auch Shortcut oder Zeugen zw. den Nachbarn

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Korrektheit

- Die Vorwärtssuche findet nur Aufwärtspfade.
- Die Rückwärtssuche findet nur Abwärtspfade.
- Gemeinsam werden nur Pfade gefunden die hoch und dann wieder runter gehen.
- Wir müssen also beweisen, dass es in G^+ immer einen kürzesten hoch-runter Pfad gibt.



- Es gibt einen kürzesten Weg P
- Wenn P kein hoch-runter Pfad ist, dann gibt es einen Knoten der höhere Nachbarn hat
- Dann gibt es auch Shortcut oder Zeugen zw. den Nachbarn

Nach “Wichtigkeit” Ordnen

Grund-Idee:

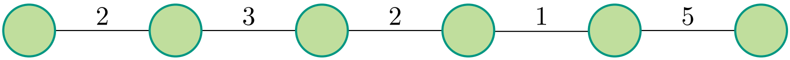
- Wir wollen wenig Shortcuts
- Ein Knoten ist “unwichtig”, wenn er wenig Shortcuts erzeugt
- $\rightarrow$ simuliere Knotenkontraktion, um Knoten zu gewichten

Algorithmus:

- Baue eine große Warteschlange mit allen Knoten gewichtet nach ihrer “Wichtigkeit”
- Kontrahiere iterativ unwichtigsten Knoten
- Kontraktion eines Knoten kann “Wichtigkeit” der Nachbarn beeinflussen
- $\rightarrow$ “Wichtigkeit” der Nachbarn neu berechnen

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

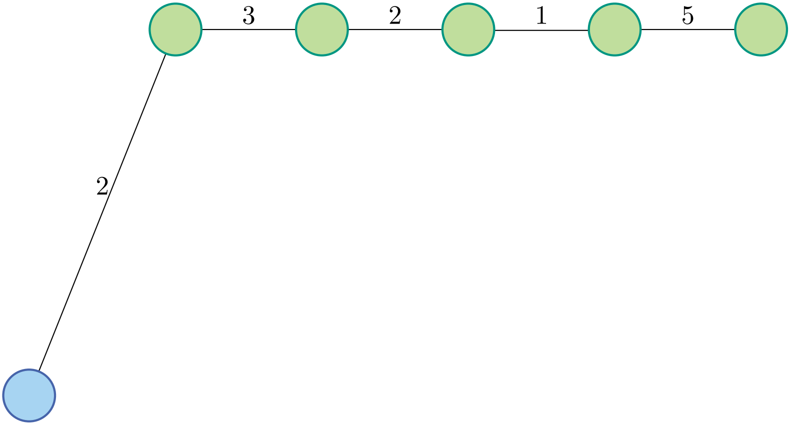
Problemfall: Pfad



Linker Knoten kontrahieren erzeugt keine Shortcuts

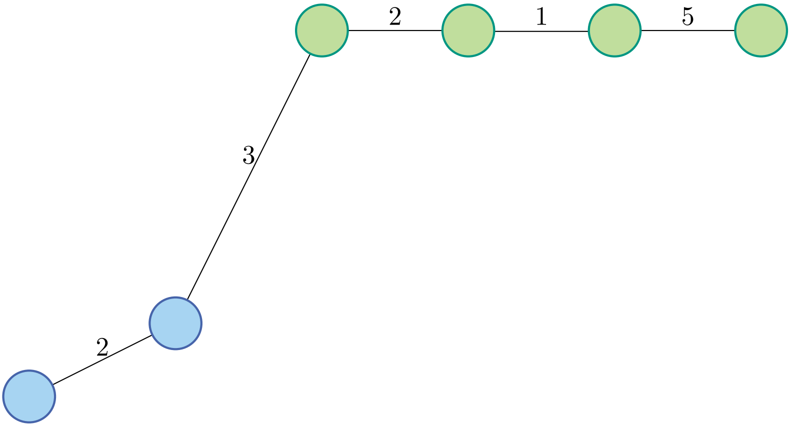
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Problemfall: Pfad



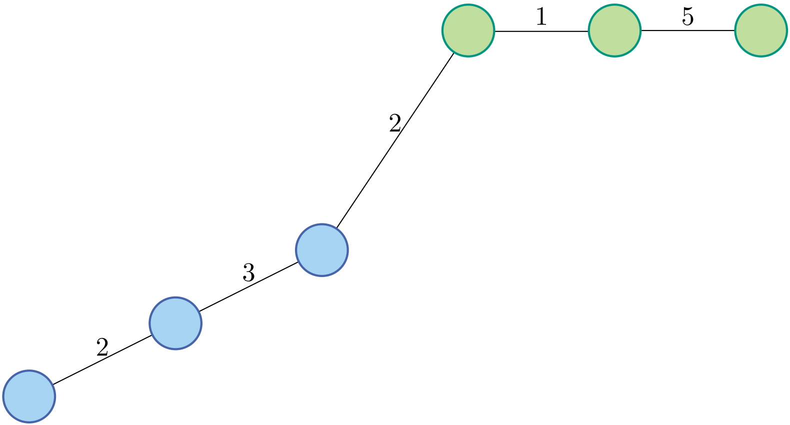
Linker Knoten kontrahieren erzeugt keine Shortcuts

Problemfall: Pfad



Linker Knoten kontrahieren erzeugt keine Shortcuts

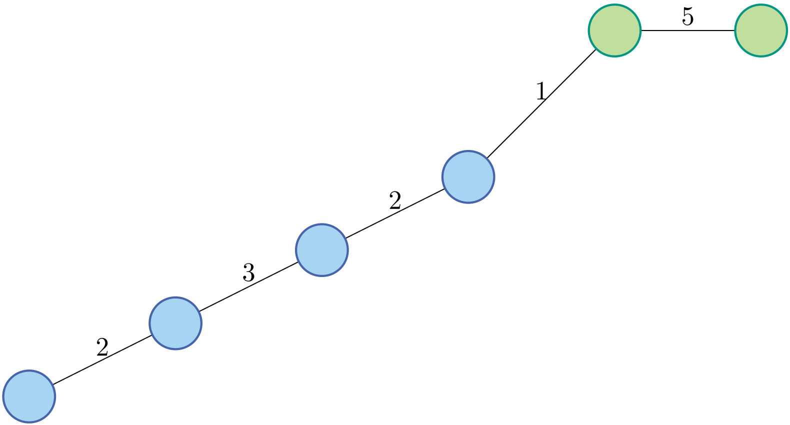
Problemfall: Pfad



Linker Knoten kontrahieren erzeugt keine Shortcuts

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

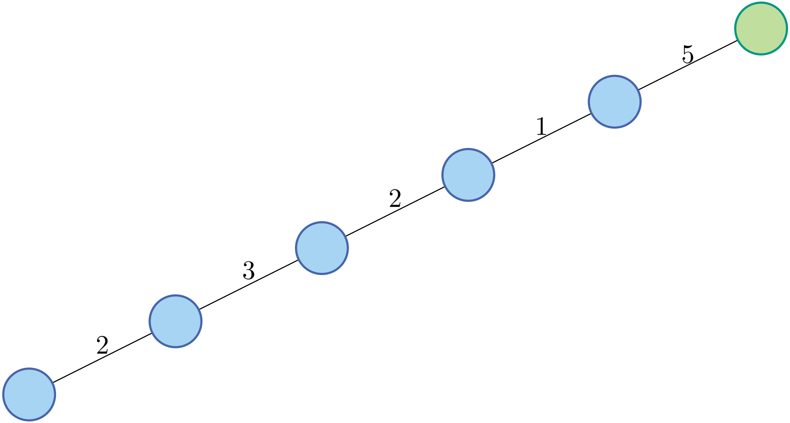
Problemfall: Pfad



Linker Knoten kontrahieren erzeugt keine Shortcuts

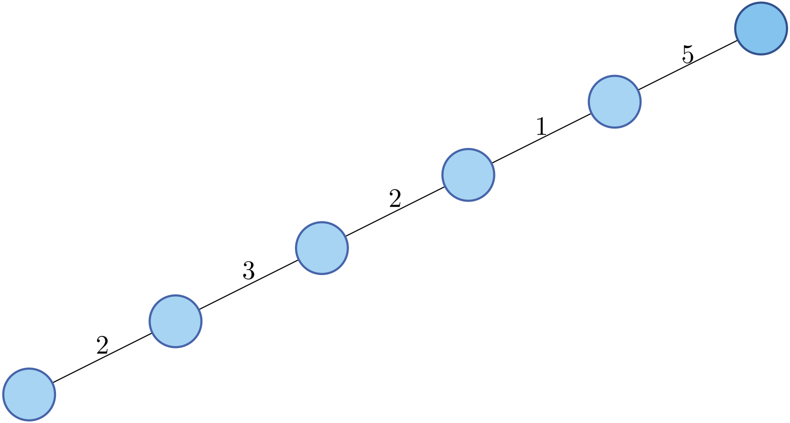
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Problemfall: Pfad



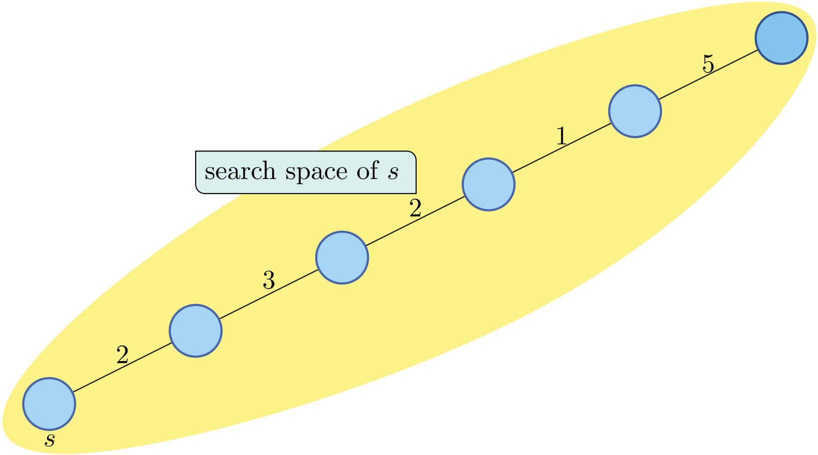
Linker Knoten kontrahieren erzeugt keine Shortcuts

Problemfall: Pfad



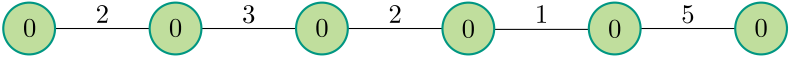
Linker Knoten kontrahieren erzeugt keine Shortcuts

Problemfall: Pfad



Suchraum von s ist der ganze Graph $\rightarrow$ keine Beschleunigung

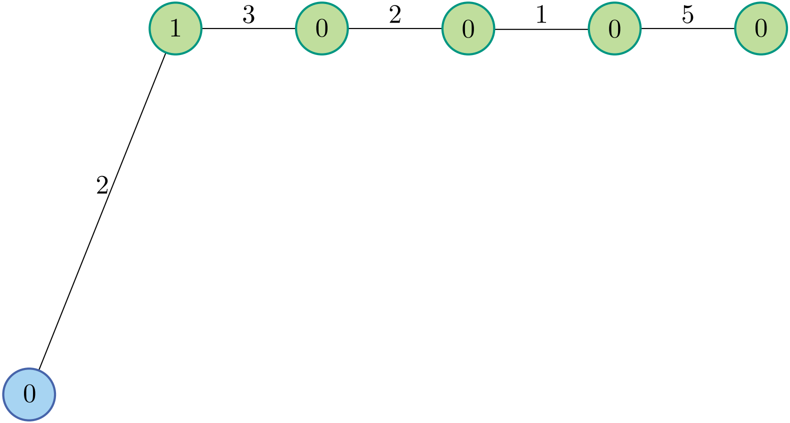
Problemfall: Pfad



2-tes Kriterium: Das geschätzte Level $\ell(x)$ eines Knoten x kontrahiert $\rightarrow$ für alle Nachbarn y: $\ell(y) \leftarrow \max\{\ell(y), \ell(x) + 1\}$

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

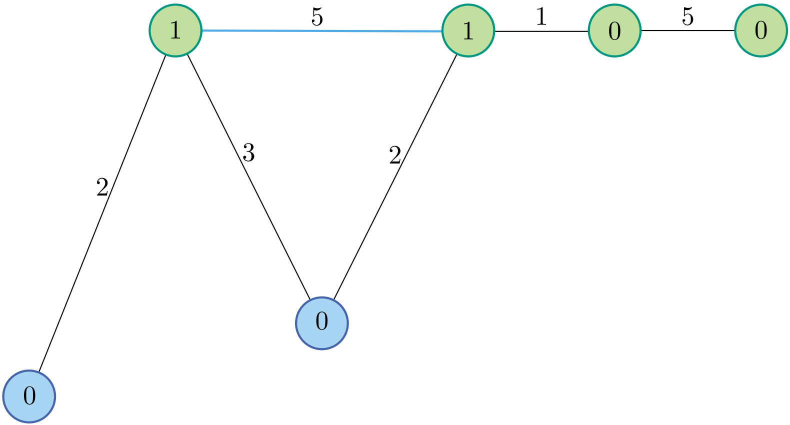
Problemfall: Pfad



2-tes Kriterium: Das geschätzte Level $\ell(x)$ eines Knoten
x kontrahiert $\rightarrow$ für alle Nachbarn y: $\ell(y) \leftarrow \max\{\ell(y), \ell(x) + 1\}$

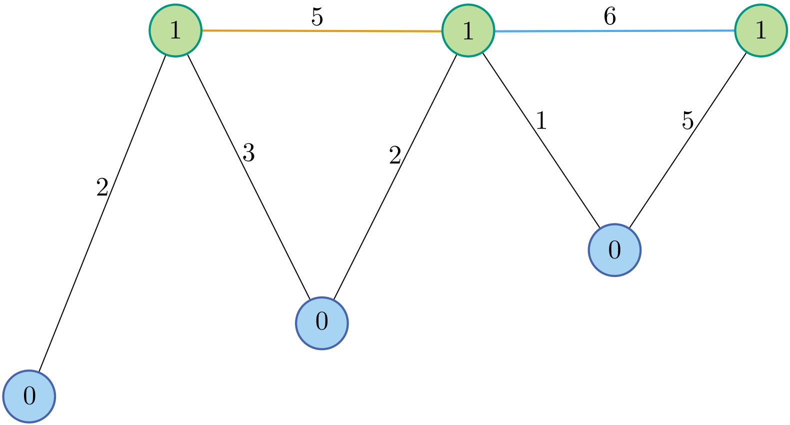
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Problemfall: Pfad



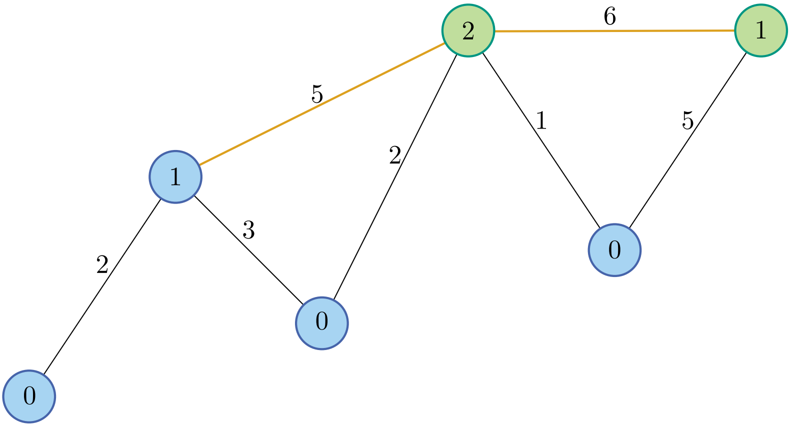
2-tes Kriterium: Das geschätzte Level $\ell(x)$ eines Knoten x kontrahiert $\rightarrow$ für alle Nachbarn y: $\ell(y) \leftarrow \max\{\ell(y), \ell(x) + 1\}$

Problemfall: Pfad



2-tes Kriterium: Das geschätzte Level $\ell(x)$ eines Knoten x kontrahiert $\rightarrow$ für alle Nachbarn y: $\ell(y) \leftarrow \max\{\ell(y), \ell(x) + 1\}$

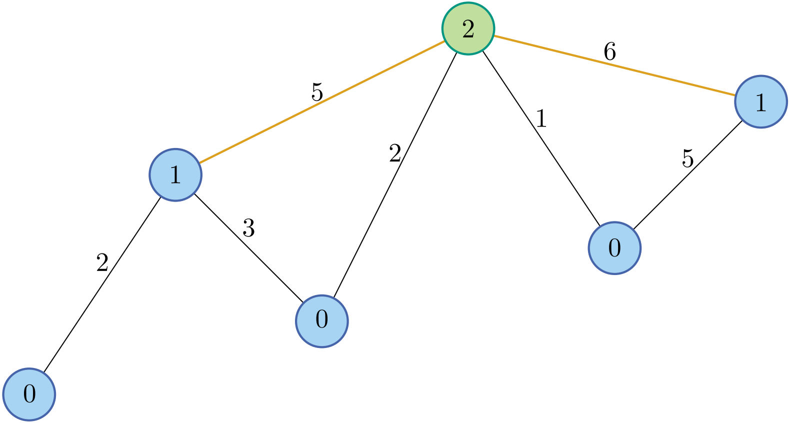
Problemfall: Pfad



2-tes Kriterium: Das geschätzte Level $\ell(x)$ eines Knoten
x kontrahiert $\rightarrow$ für alle Nachbarn y: $\ell(y) \leftarrow \max\{\ell(y), \ell(x) + 1\}$

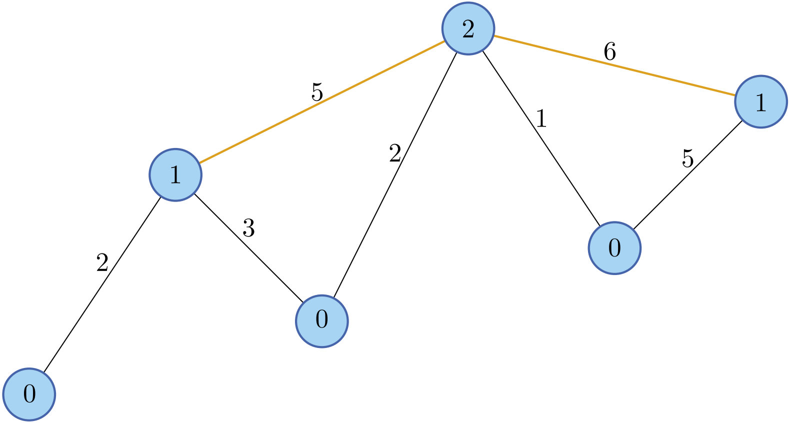
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Problemfall: Pfad



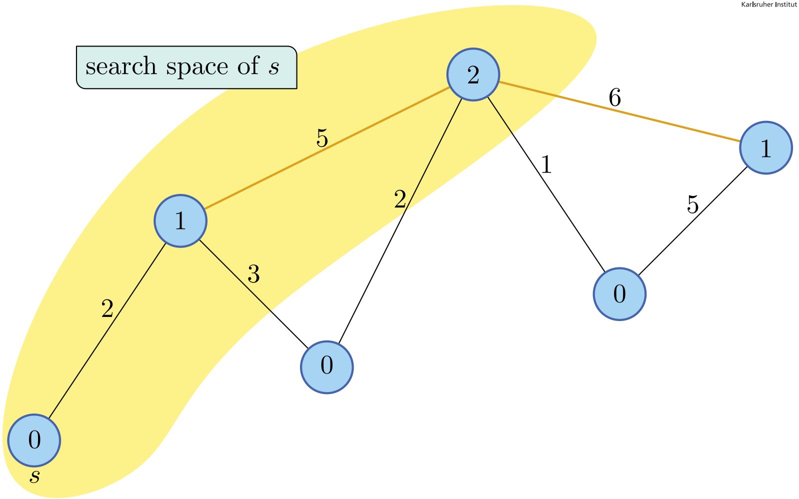
2-tes Kriterium: Das geschätzte Level $\ell(x)$ eines Knoten x kontrahiert $\rightarrow$ für alle Nachbarn y: $\ell(y) \leftarrow \max\{\ell(y), \ell(x) + 1\}$

Problemfall: Pfad



2-tes Kriterium: Das geschätzte Level $\ell(x)$ eines Knoten
x kontrahiert $\rightarrow$ für alle Nachbarn y: $\ell(y) \leftarrow \max\{\ell(y), \ell(x) + 1\}$

Problemfall: Pfad



2-tes Kriterium: Das geschätzte Level $\ell(x)$ eines Knoten
x kontrahiert $\rightarrow$ für alle Nachbarn y: $\ell(y) \leftarrow \max\{\ell(y), \ell(x) + 1\}$

Kombination mehrerer Kriterien

- Speichere für jede Kante e die Anzahl $h(e)$ der Originalkanten, aus denen sie besteht
- Es sei $A(x)$ die Menge der eingefügten Shortcuts, wenn x kontrahiert werden würde
- Analog: $D(x)$ die Menge der gelöschten Kanten
- Es sei $I(x)$ die "Wichtigkeit" von x

Eine funktionierende Definition von $I(x)$ ist

$$I(x) := \ell(x) + \frac{|A(x)|}{|D(x)|} + \frac{\sum_{e \in A(x)} h(e)}{\sum_{e \in D(x)} h(e)}$$

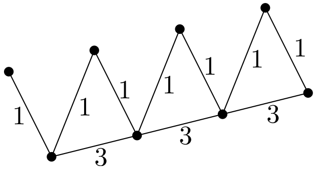
Hinweis: Es gibt sehr viele unterschiedliche Definitionen für I. Das ist nur ein Kochrezept, das sich bewährt hat und jeder würzt leicht anders

CH im Detail

Stall-On-Demand

Beobachtung:

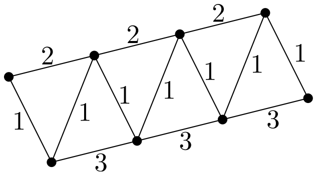
- Suchen können Knoten mit zu großer Distanz besuchen



Stall-On-Demand

Beobachtung:

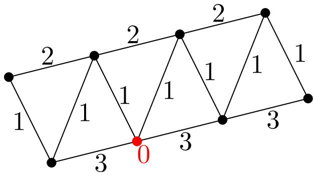
- Suchen können Knoten mit zu großer Distanz besuchen



Stall-On-Demand

Beobachtung:

- Suchen können Knoten mit zu großer Distanz besuchen

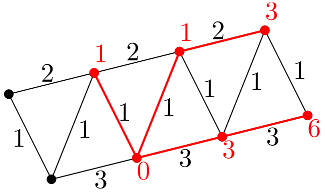


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Stall-On-Demand

Beobachtung:

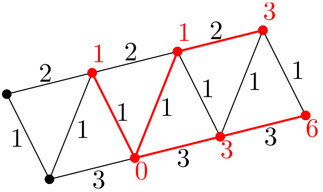
- Suchen können Knoten mit zu großer Distanz besuchen



Stall-On-Demand

Beobachtung:

- Suchen können Knoten mit zu großer Distanz besuchen

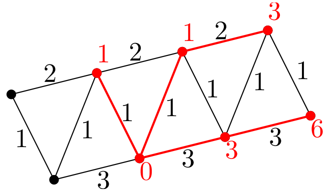


- Kann man zum prunen verwenden

Stall-On-Demand

Beobachtung:

- Suchen können Knoten mit zu großer Distanz besuchen

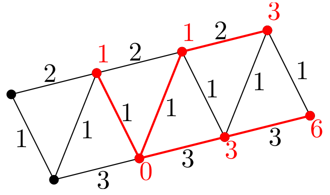


- Kann man zum prunen verwenden
- Jeder Teilpfad eines kürzesten hoch-runter-Pfads muss ein kürzester Pfad sein
- Ehe man einen Knoten settelt, sucht man nach kürzeren hoch-runter-Pfaden, gibt es diese kann man prunen

Stall-On-Demand

Beobachtung:

- Suchen können Knoten mit zu großer Distanz besuchen

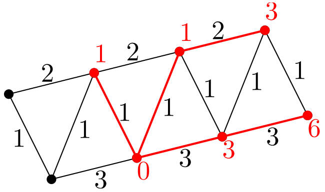


- Kann man zum prunen verwenden
- Jeder Teilpfad eines kürzesten hoch-runter-Pfads muss ein kürzester Pfad sein
- Ehe man einen Knoten settelt, sucht man nach kürzeren hoch-runter-Pfaden, gibt es diese kann man prunen
- Knoten v wird geprunt, wenn es Aufwärtsnachbar u von v gibt, so dass $d(u) + w(u, v) < d(v)$

Stall-On-Demand

Beobachtung:

- Suchen können Knoten mit zu großer Distanz besuchen

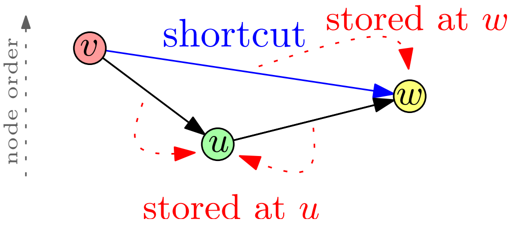


- Kann man zum prunen verwenden
- Jeder Teilpfad eines kürzesten hoch-runter-Pfads muss ein kürzester Pfad sein
- Ehe man einen Knoten settelt, sucht man nach kürzeren hoch-runter-Pfaden, gibt es diese kann man prunen
- Knoten v wird geprunt, wenn es Aufwärtsnachbar u von v gibt, so dass $d(u) + w(u, v) < d(v)$
- (Dies ist eine vereinfachte Version des ursprünglichen "Stall-On-Demand".)

Datenstruktur

Suchgraph:

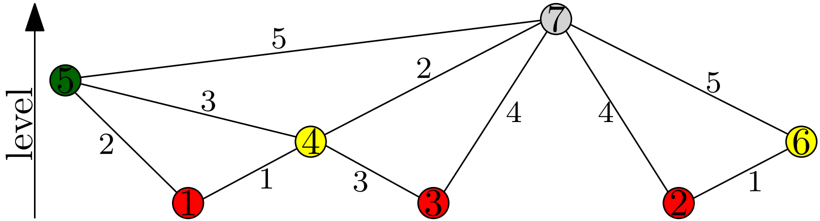
- normalerweise: speichere Kanten (v, w) in den Adjazenz-Arrays von v und w um bidirektionale Suche zu erlauben
- für die CH-Suche reicht es aus, die Kante nur an den Knoten $\min\{r(v), r(w)\}$ zu speichern



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

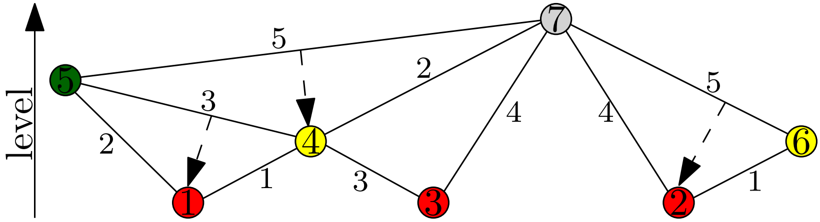
Pfadentpackung

- für jeden Shortcut (u, w) eines Pfades (u, v, w), speichere Mittelknoten v an der Kante
- expandiere Pfade mittels Rekursion



Pfadentpackung

- für jeden Shortcut (u, w) eines Pfades (u, v, w), speichere Mittelknoten v an der Kante
- expandiere Pfade mittels Rekursion



Knotenordnung

Wie Knoten ordnen?

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Knotenordnung

Wie Knoten ordnen?

- Bottom-Up, suche nach unwichtigen Knoten [GSSV12], haben wir schon gemacht
- Top-Down, suche nach wichtigen Knoten [ADGW12]
- Sampling Path-Greedy, Variante von Top-Down [DGPW14]

Top-Down

Karlsruher Institut für Technologie

Path-Greedy

- Idee: Baue Ordnung von wichtig nach unwichtig
- Knoten wichtig, wenn er auf vielen kürzesten Wegen liegt

- initial ist die Ordnung leer
- Pfad ist überdeckt, wenn er einen Knoten in der Ordnung enthält
- sei U die Menge der nicht überdeckten kürzesten Pfade
- initial ist U die Menge aller kürzesten Pfade

- Algo:
  - Solange die Ordnung nicht voll:
  - Packe Knoten v oben in die Ordnung, so dass v möglichst viele Pfade aus U überdeckt
  - Entferne diese Pfade aus U

Top-Down

Path-Greedy: Diskussion

- n mal All-Pair-Shortest-Path
  $\rightarrow n^3 \log n$ mit Dijkstra auf dünnen Graphen
- linear Speicherverbrauch
- Kann auf $n^2 \log n$ gedrückt werden mit $O(n^2)$ Speicher
  $O(n^2)$ kommt daher, dass n Kürzeste-Wege-Bäume verwaltet werden
- gute Qualität der Ordnung, aber langsam

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Sampling Path-Greedy [DGPW14]

- Problem von Top-Down: $O(n^2)$ Speicherverbrauch, da n Kürzeste-Wege-Bäume verwaltet werden
- Ansatz von Sampling Path-Greedy: speichere nur ein “paar” kürzeste Wege Bäume
- d.h., der Algorithmus sampelt Bäume

Sampling Path-Greedy [DGPW14]

Karlsruher Institut für Technologie

Idee

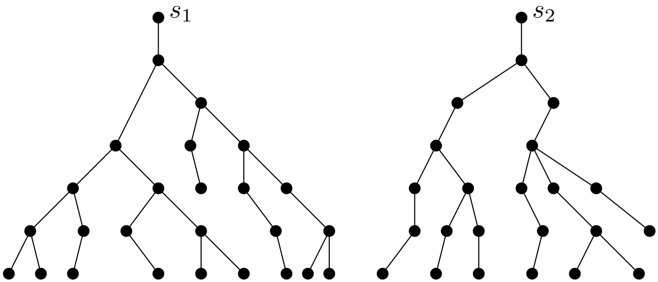
- Verwalte nur wenige zufällige Kürzeste-Wege-Bäume
  - Quellknoten: $s_1, s_2 \dots$
- Wähle v, der auf den meisten Pfaden in diesen Bäumen liegt

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Sampling Path-Greedy [DGPW14]

Idee

- Verwalte nur wenige zufällige Kürzeste-Wege-Bäume
  - Quellknoten: s_1, s_2 ...
- Wähle v, der auf den meisten Pfaden in diesen Bäumen liegt

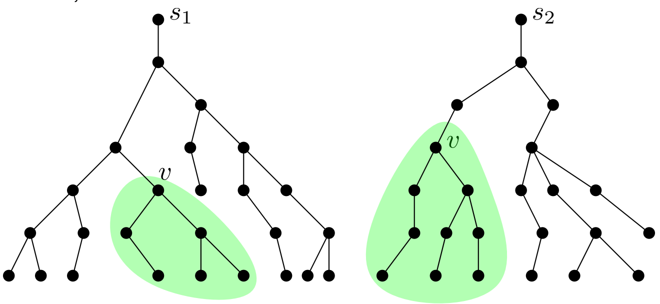


- Die kürzeste Wege Bäume von s_1 und s_2

Sampling Path-Greedy [DGPW14]

Idee

- Verwalte nur wenige zufällige Kürzeste-Wege-Bäume
  - Quellknoten: s_1, s_2 ...
- Wähle v, der auf den meisten Pfaden in diesen Bäumen liegt



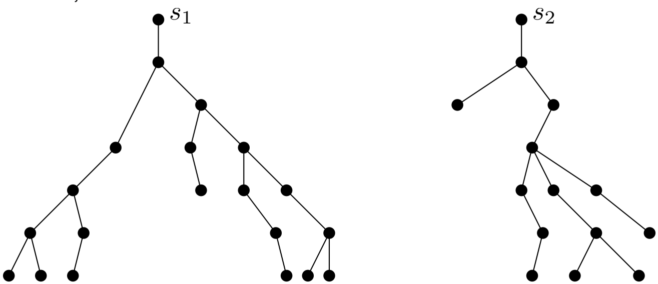
- Die kürzeste Wege Bäume von s_1 und s_2
- v liegt auf so vielen Pfaden, die bei s_1 beginnen, wie sein s_1-Teilbaum groß ist (hier 6)

Sampling Path-Greedy [DGPW14]

Karlsruher Institut für Technologie

Idee

- Verwalte nur wenige zufällige Kürzeste-Wege-Bäume
  - Quellknoten: s_1, s_2 ...
- Wähle v, der auf den meisten Pfaden in diesen Bäumen liegt



- Die kürzeste Wege Bäume von s_1 und s_2
- v liegt auf so vielen Pfaden, die bei s_1 beginnen, wie sein s_1-Teilbaum groß ist (hier 6)
- Entferne Teilbäume in allen verwalteten Bäumen

Sampling Path-Greedy [DGPW14]

Experimente zeigen:

- Nur wenige Bäume notwendig um die wichtigsten Knoten zu finden
- Für das Mittelfeld und die unwichtigen Knoten werden mehr Bäume gebraucht

Beobachtung:

- Durch Löschen der Teilbäume wird Speicher frei

Idee:

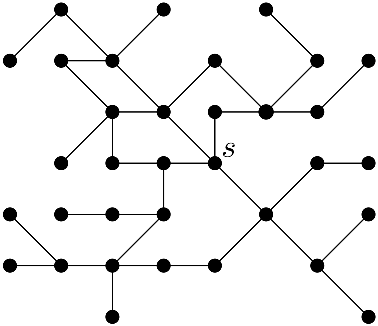
- Fülle frei gewordenen Speicher mit neuen Bäumen

Neue Bäume Aufbauen [DGPW14]

- Durch entfernen von Teilbäumen schrumpfen die Bäume
- $\rightarrow$ Baue neue Bäume auf
- Wie machen wir das ohne uns den ganzen Graph anzuschauen?

Neue Bäume Aufbauen [DGPW14]

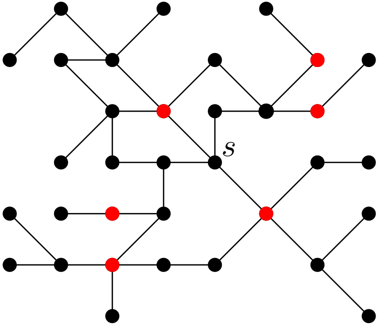
- Durch entfernen von Teilbäumen schrumpfen die Bäume
- $\rightarrow$ Baue neue Bäume auf
- Wie machen wir das ohne uns den ganzen Graph anzuschauen?



s ist neue Baumwurzel

Neue Bäume Aufbauen [DGPW14]

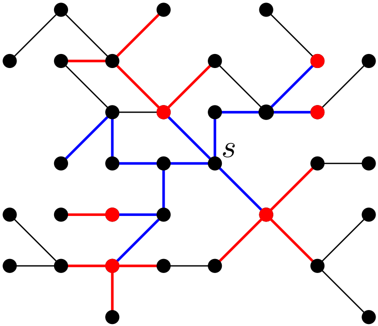
- Durch entfernen von Teilbäumen schrumpfen die Bäume
- $\rightarrow$ Baue neue Bäume auf
- Wie machen wir das ohne uns den ganzen Graph anzuschauen?



rote Knoten sind bereits in der Ordnung

Neue Bäume Aufbauen [DGPW14]

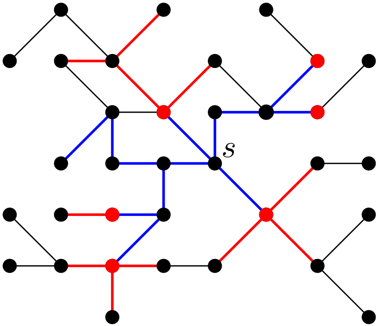
- Durch entfernen von Teilbäumen schrumpfen die Bäume
- $\rightarrow$ Baue neue Bäume auf
- Wie machen wir das ohne uns den ganzen Graph anzuschauen?



lasse Dijkstra laufen

Neue Bäume Aufbauen [DGPW14]

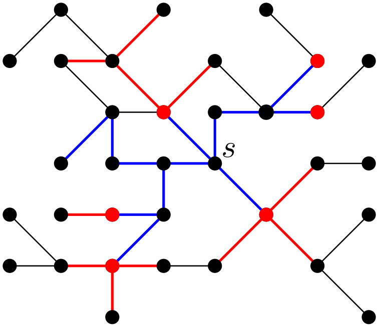
- Durch entfernen von Teilbäumen schrumpfen die Bäume
- $\rightarrow$ Baue neue Bäume auf
- Wie machen wir das ohne uns den ganzen Graph anzuschauen?



speichere welche Knoten über rote Knoten erreicht wurden

Neue Bäume Aufbauen [DGPW14]

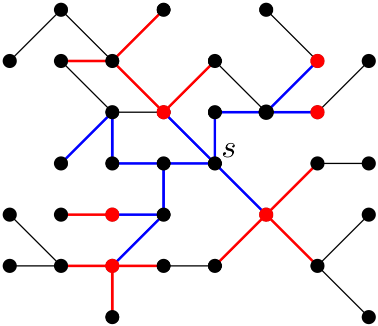
- Durch entfernen von Teilbäumen schrumpfen die Bäume
- $\rightarrow$ Baue neue Bäume auf
- Wie machen wir das ohne uns den ganzen Graph anzuschauen?



breche ab, wenn die Queue nur noch Knoten enthält die über rote Knoten erreicht wurden

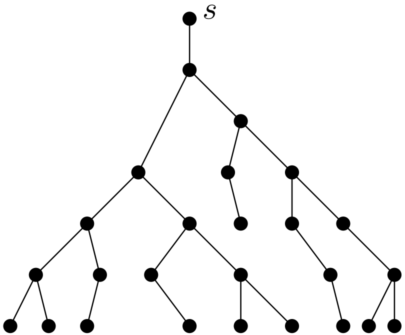
Neue Bäume Aufbauen [DGPW14]

- Durch entfernen von Teilbäumen schrumpfen die Bäume
- $\rightarrow$ Baue neue Bäume auf
- Wie machen wir das ohne uns den ganzen Graph anzuschauen?



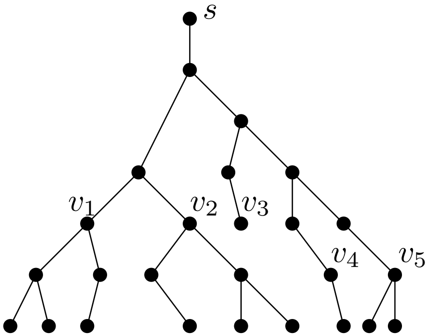
der blaue Teilgraph ist der neue Baum

Neue Bäume Aufbauen (alternative Sichtweise) [DGPW14]



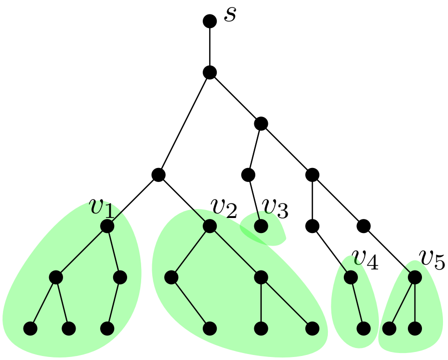
s ist neue Baumwurzel
Oben abgebildet ist der vollständige Kürzeste-Wege Baum

Neue Bäume Aufbauen (alternative Sichtweise) [DGPW14]



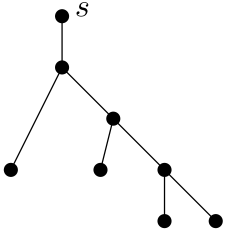
Die Knoten v_i wurden bereits ausgewählt

Neue Bäume Aufbauen (alternative Sichtweise) [DGPW14]



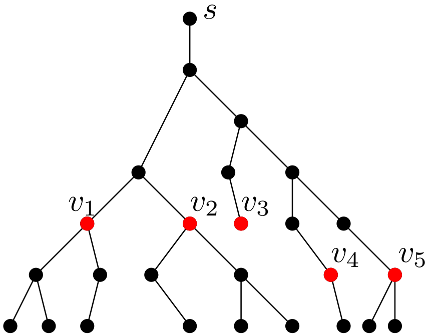
Ihre Teilbäume werden nicht gebraucht
$\rightarrow$ Wir wollen diese Teile gar nicht erst aufbauen

Neue Bäume Aufbauen (alternative Sichtweise) [DGPW14]



Dies ist der Baum den wir haben wollen

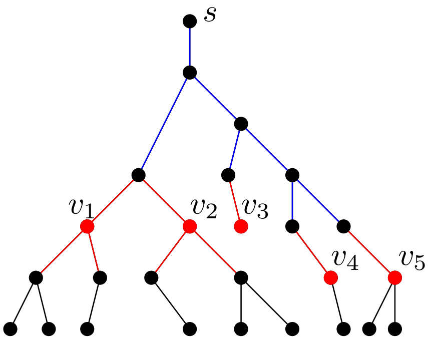
Neue Bäume Aufbauen (alternative Sichtweise) [DGPW14]



Die Knoten v_i sind bereits in der Ordnung und rot markiert

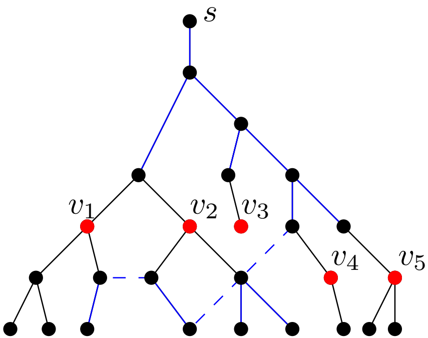
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Neue Bäume Aufbauen (alternative Sichtweise) [DGPW14]



Um den Baum aufzubauen lassen wir Dijkstras Algorithmus von s aus laufen und brechen ab sobald alle Äste über rote Knoten gehen

Neue Bäume Aufbauen (alternative Sichtweise) [DGPW14]



Dies ist nicht das selbe wie an allen $v_1 \ldots$ zu prunen!
Wenn wir prunen findet die Suche neue Wege die gar nicht entlang des Kürzeste-Wege Baum von s entlang gehen.

Sampling Path-Greedy [DGPW14]

Karlsruher Institut für Technologie

Algo:

1. Baue Bäume auf mit zufälliger Wurzel, bis $k \cdot n$ Knoten in allen Bäumen sind
2. Wähle v aus
3. Lösche Teilbäume unter v
4. Gehe zu 1

- k ist ein Parameter der die Qualität steuert

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Sampling Path-Greedy [DGPW14]

Vorteile von Sampling Path-Greedy:

- Funktioniert auf den meisten Graphen
  Auch auf Graphen mit hohen Knotengraden, wo Bottom-Up sich schwer tut

Auf Straße

- Ordnungsqualität vergleichbar mit Bottom-Up
- Aber langsamer als Bottom-Up
- $\rightarrow$ Nehmt Bottom-Up

Ergebnisse

	preprocessing		query	
method	time [h:m]	space [GB]	scans	time [μs]
MLD-3	$<$ 0:01	0.4	6074	912
MLD-4	$<$ 0:01	0.4	3897	707
CH	0:02	0.4	284	96.3
CH-15	0:04	0.4	231	85.0
CH-17	0:24	0.4	217	79.7

CH-x mit 2^x top-down Ordnung,
zur Erinnerung: Graph hat $> 2^{24}$ Knoten

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Ergebnisse

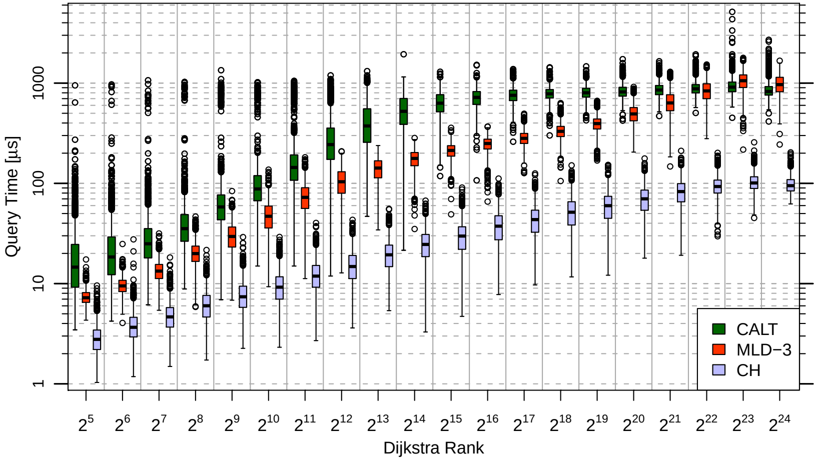
	preprocessing		query	
method	time [h:m]	space [GB]	scans	time [μs]
MLD-3	$<$ 0:01	0.4	6074	912
MLD-4	$<$ 0:01	0.4	3897	707
CH	0:02	0.4	284	96.3
CH-15	0:04	0.4	231	85.0
CH-17	0:24	0.4	217	79.7

CH-x mit 2^x top-down Ordnung,
zur Erinnerung: Graph hat $> 2^{24}$ Knoten

- CH etwas langsamere Vorberechnung
- Faktor 10 schneller als MLD
- bottom-up Knotenordnung gut genug

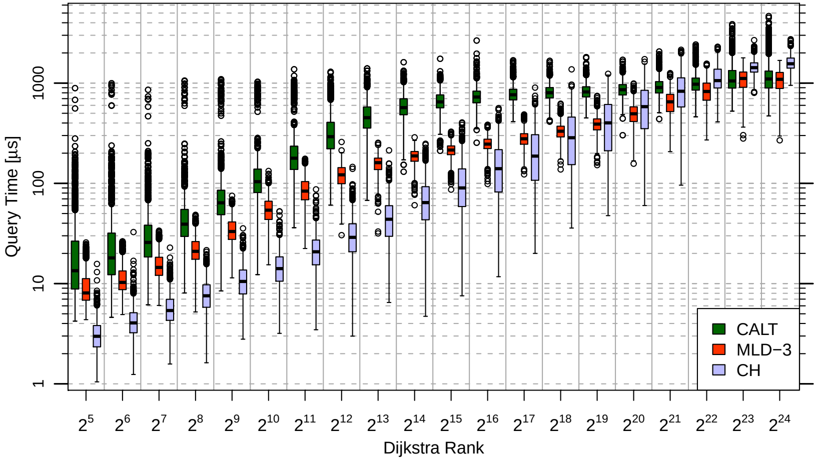
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Local Queries: Reisezeiten



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Local Queries: Reisedistanzen



Nächste Termine

Mittwoch, 9.5.2018
Montag, 14.5.2018
Mittwoch, 16.5.2018

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Literatur I

Karlsruher Institut für Technologie

Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck.
Hierarchical hub labelings for shortest paths.
In *Proceedings of the 20th Annual European Symposium on Algorithms (ESA'12)*, volume 7501 of *Lecture Notes in Computer Science*, pages 24–35. Springer, 2012.

Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck.
Robust distance queries on massive networks.
In *Proceedings of the 22nd Annual European Symposium on Algorithms (ESA'14)*, volume 8737 of *Lecture Notes in Computer Science*, pages 321–333. Springer, September 2014.

Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter.
Exact routing in large road networks using contraction hierarchies.
Transportation Science, 46(3):388–404, August 2012.