Karlsruher Institut für Technologie

Algorithmen für Routenplanung

7. Sitzung, Sommersemester 2013

Julian Dibbelt | 13. Mai 2013

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

Hierarchische Techniken

Ideensammlung:

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

Overlay Graph

Gegeben

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

Berechne

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

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

Wahl der Knotenmenge

- Randknoten
- können auch beliebige Knotenmengen sein

Contraction Hierarchies

Zutaten:

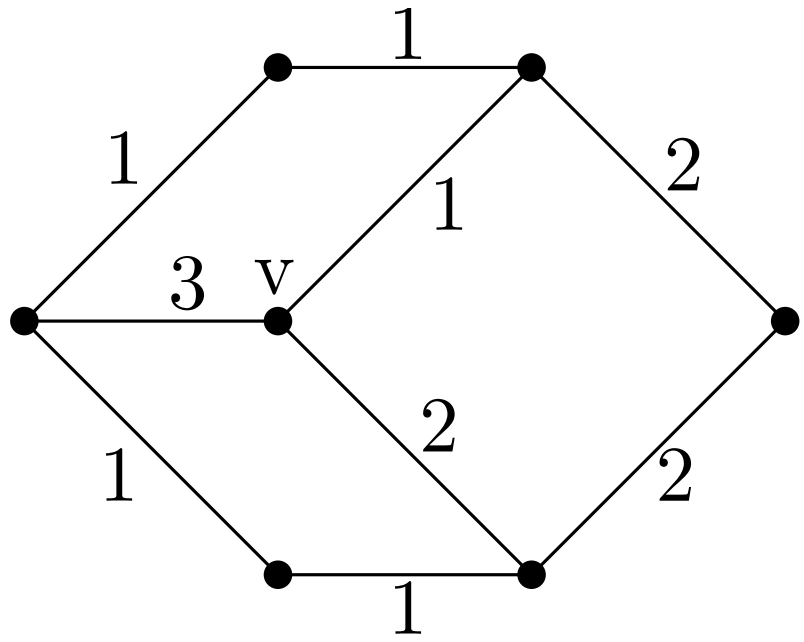
- Ordne Knoten nach Wichtigkeit
  $r : V \to [0, \ldots, n-1]$
- Shortcut Operation

Contraction Hierarchies

Zutaten:

- Ordne Knoten nach Wichtigkeit $r : V \to [0, \ldots, n-1]$
- Shortcut Operation

Shortcut Operation (von Knoten v):



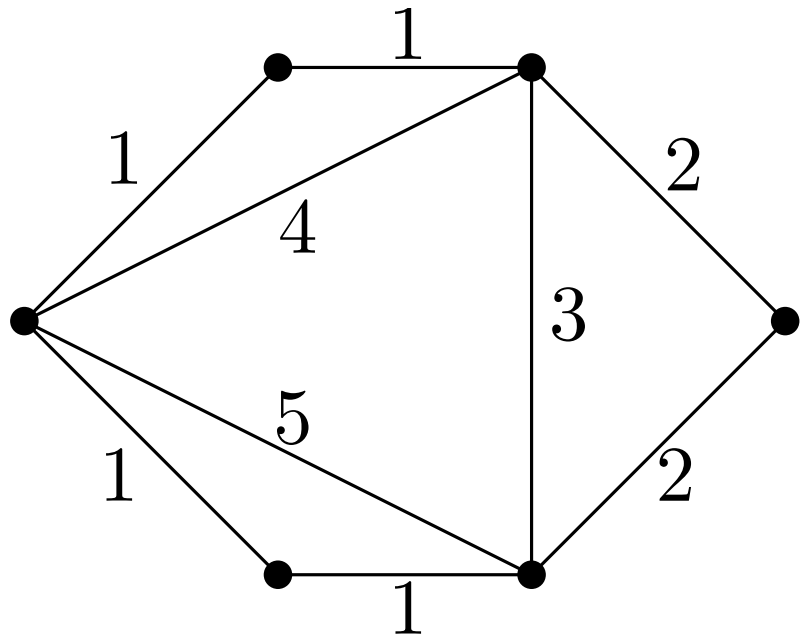
Contraction Hierarchies

Zutaten:

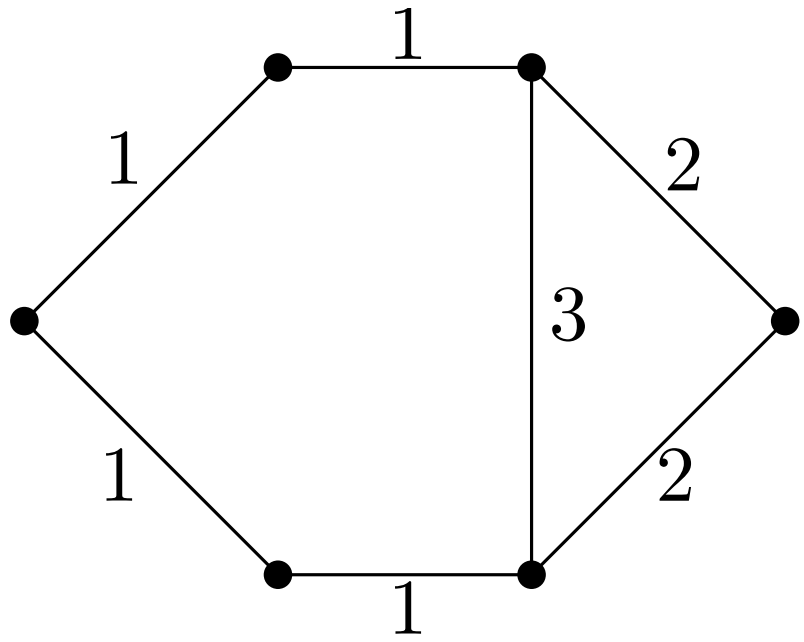
- Ordne Knoten nach Wichtigkeit $r : V \to [0, \ldots, n-1]$
- Shortcut Operation

Shortcut Operation (von Knoten v):

- entferne v



Contraction Hierarchies



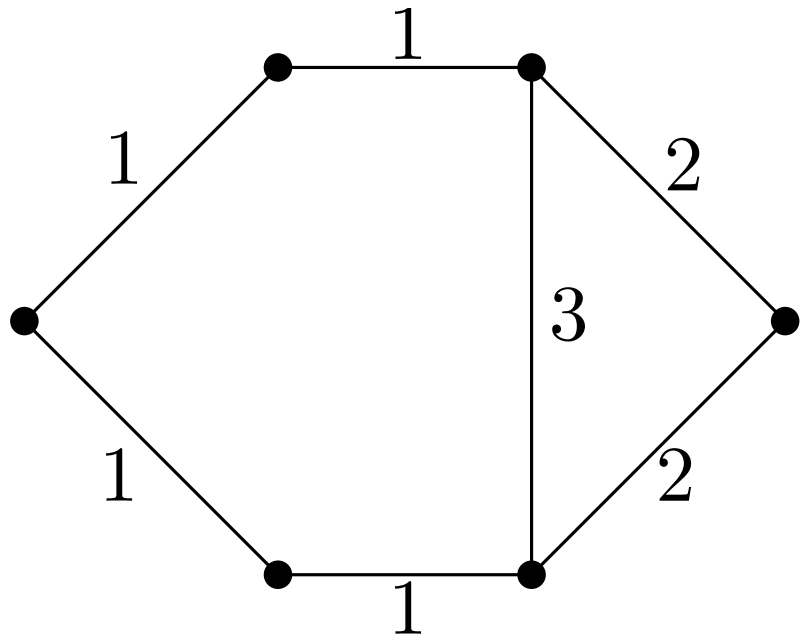
Zutaten:

- Ordne Knoten nach Wichtigkeit $r : V \to [0, \ldots, n-1]$
- Shortcut Operation

Shortcut Operation (von Knoten v):

- entferne v
- für jedes Paar u, w füge Kanten (u, w) zum Graphen wenn v auf dem einzigen kürzesten Weg von u nach w liegt

Contraction Hierarchies



Zutaten:

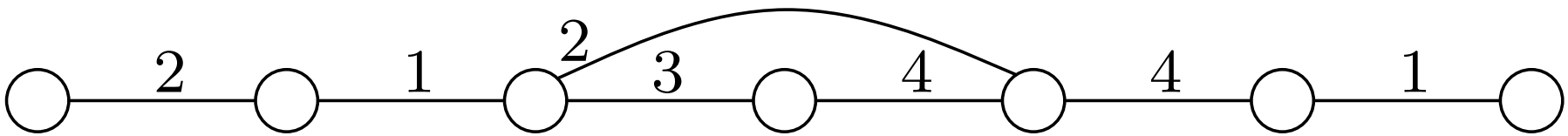
- Ordne Knoten nach Wichtigkeit $r : V \to [0, \ldots, n-1]$
- Shortcut Operation

Shortcut Operation (von Knoten v):

- entferne v
- für jedes Paar u, w füge Kanten (u, w) zum Graphen wenn v auf dem einzigen kürzesten Weg von u nach w liegt
- kann durch lokale Dijkstra Suchen von den Nachbarn berechnet werden

Contraction Hierarchies

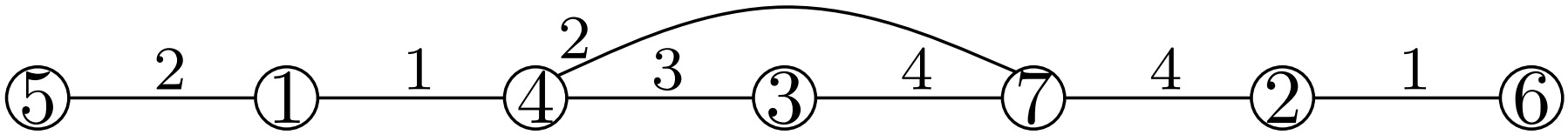
Vorberechnung:



Contraction Hierarchies

Vorberechnung:

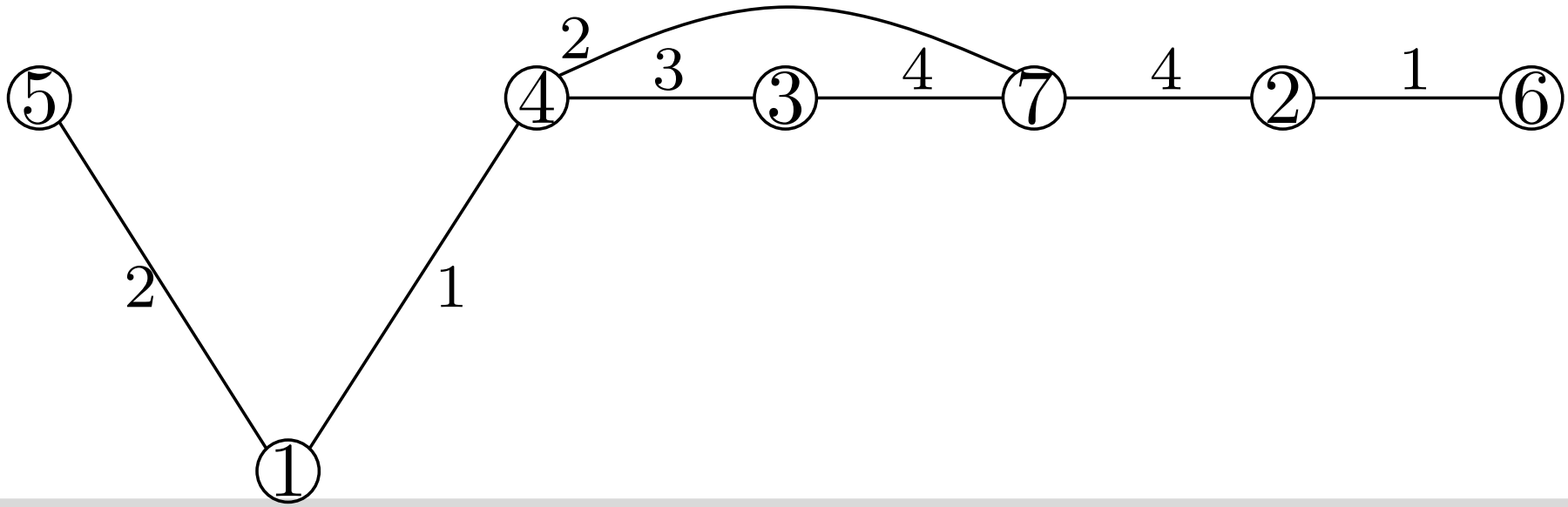
- ordne Knoten



Contraction Hierarchies

Vorberechnung:

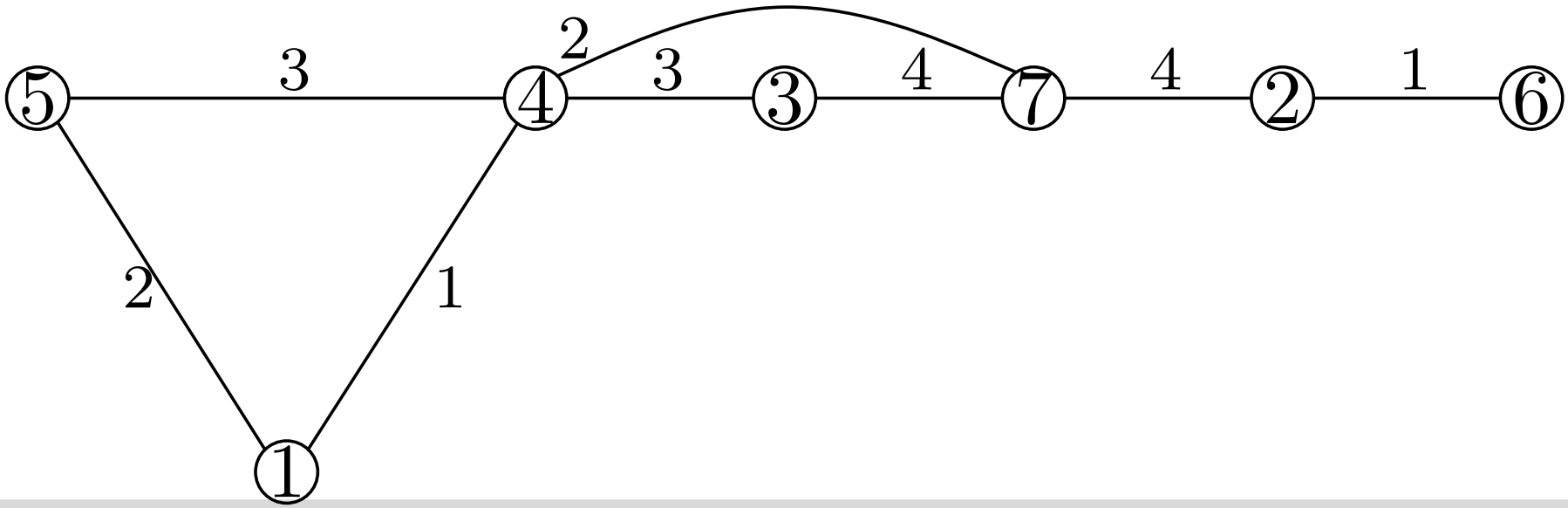
- ordne Knoten
- bearbeite in der Reihenfolge
- shortcutten des Knoten, hinzufügen von Shortcuts (A^+)



Contraction Hierarchies

Vorberechnung:

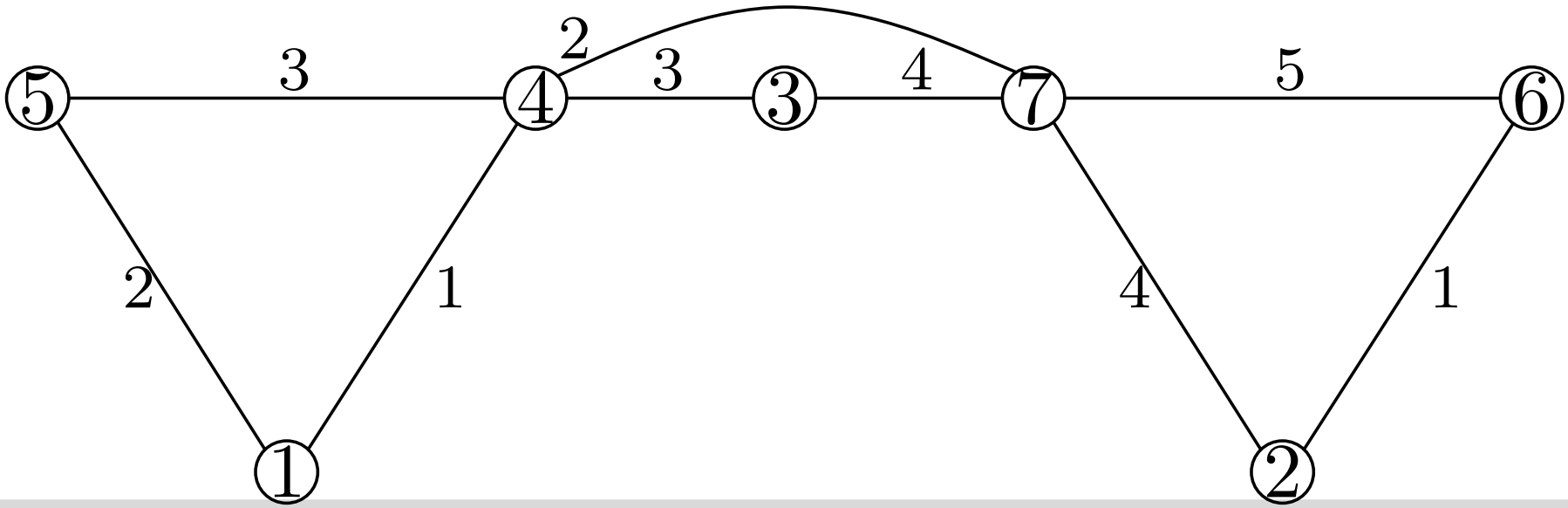
- ordne Knoten
- bearbeite in der Reihenfolge
- shortcutten des Knoten, hinzufügen von Shortcuts (A^+)



Contraction Hierarchies

Vorberechnung:

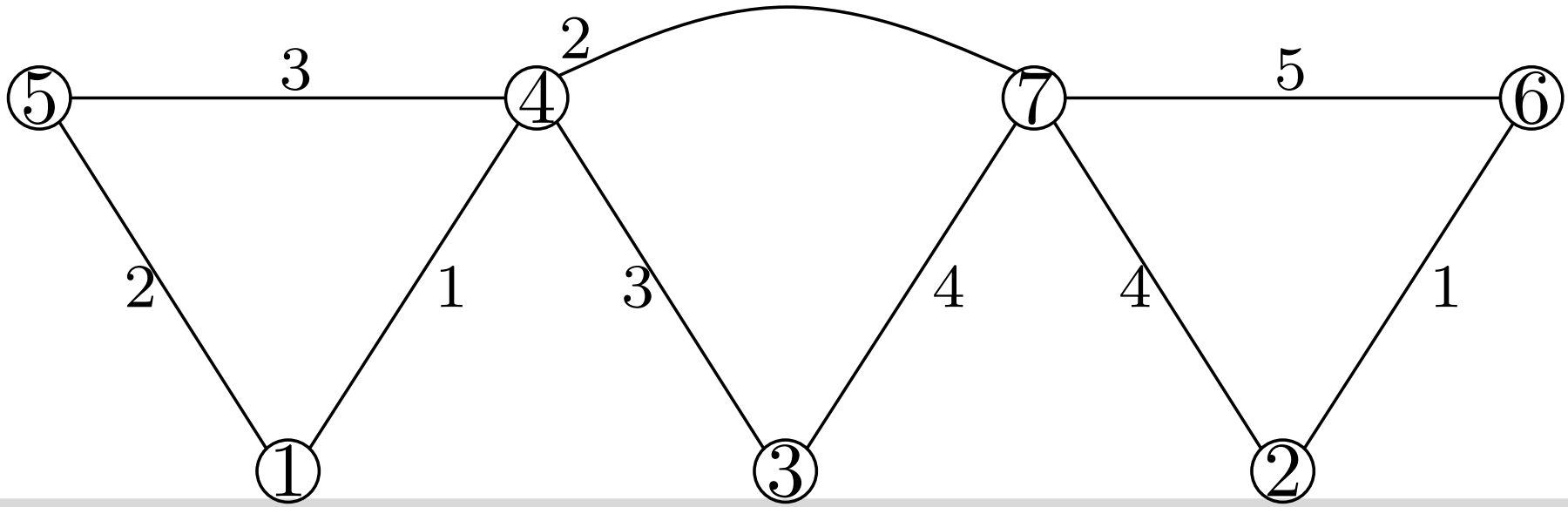
- ordne Knoten
- bearbeite in der Reihenfolge
- shortcutten des Knoten, hinzufügen von Shortcuts (A^+)



Contraction Hierarchies

Vorberechnung:

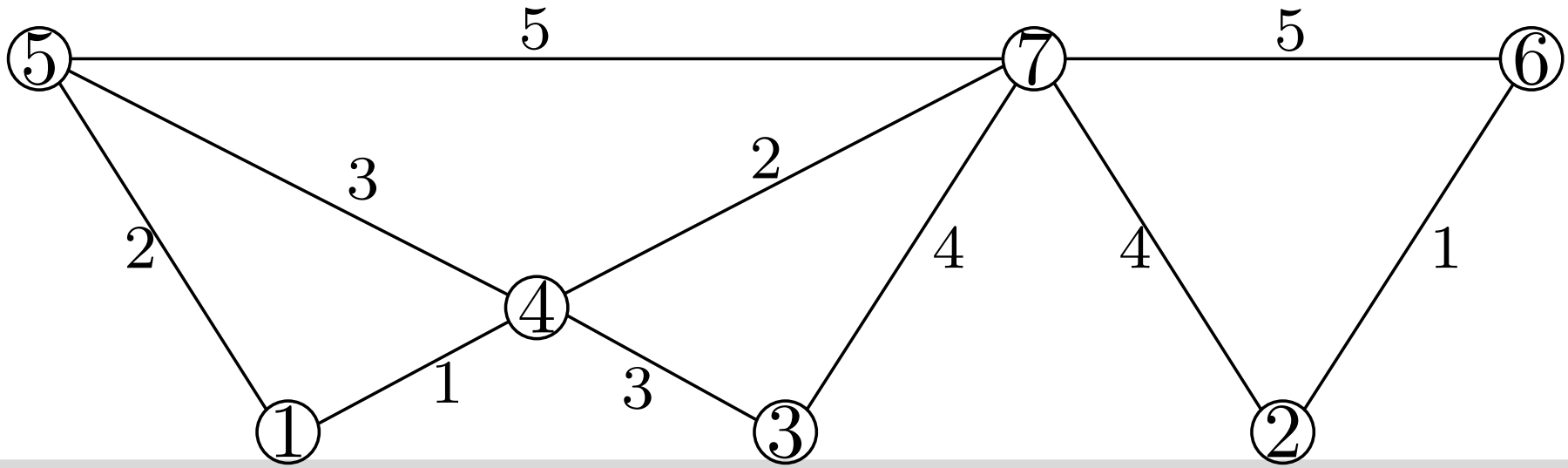
- ordne Knoten
- bearbeite in der Reihenfolge
- shortcutten des Knoten, hinzufügen von Shortcuts (A^+)



Contraction Hierarchies

Vorberechnung:

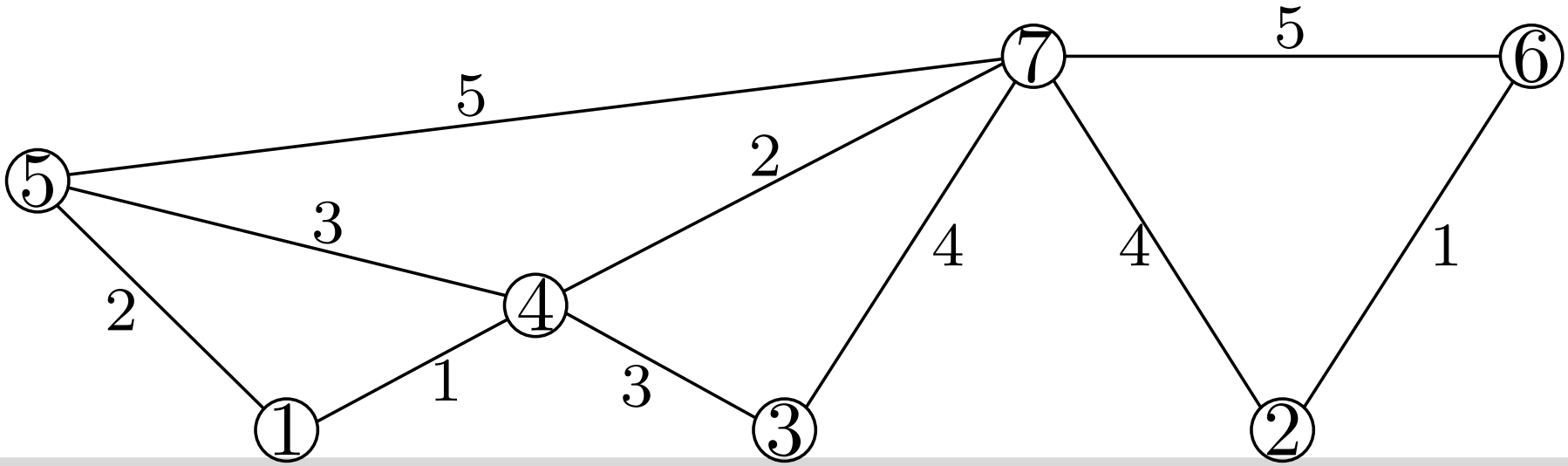
- ordne Knoten
- bearbeite in der Reihenfolge
- shortcutten des Knoten, hinzufügen von Shortcuts (A^+)



Contraction Hierarchies

Vorberechnung:

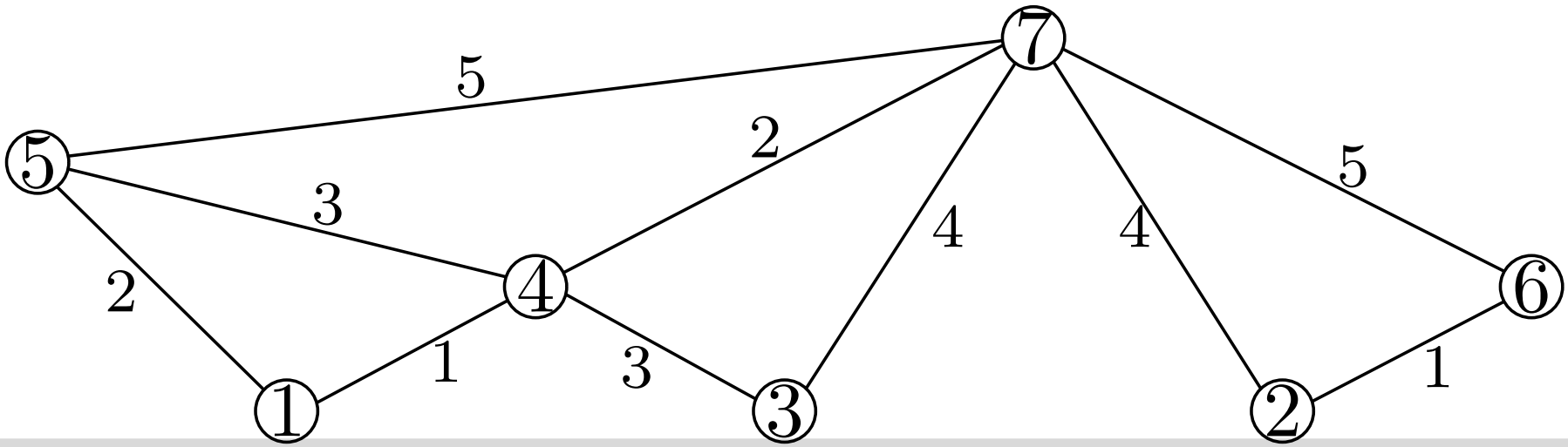
- ordne Knoten
- bearbeite in der Reihenfolge
- shortcutten des Knoten, hinzufügen von Shortcuts (A^+)



Contraction Hierarchies

Vorberechnung:

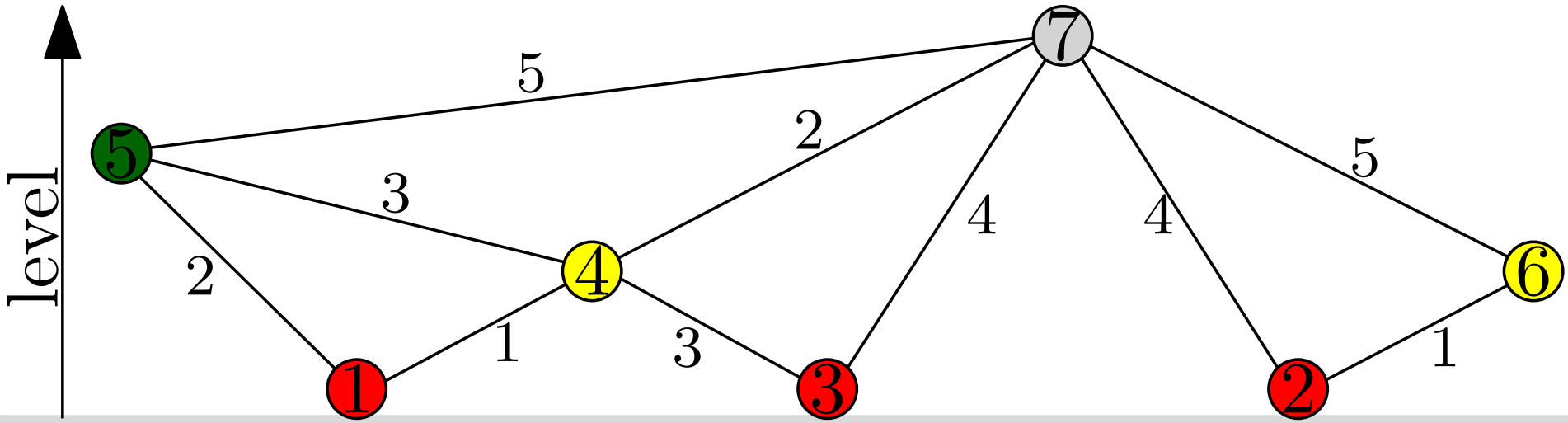
- ordne Knoten
- bearbeite in der Reihenfolge
- shortcutten des Knoten, hinzufügen von Shortcuts (A^+)



Contraction Hierarchies

Vorberechnung:

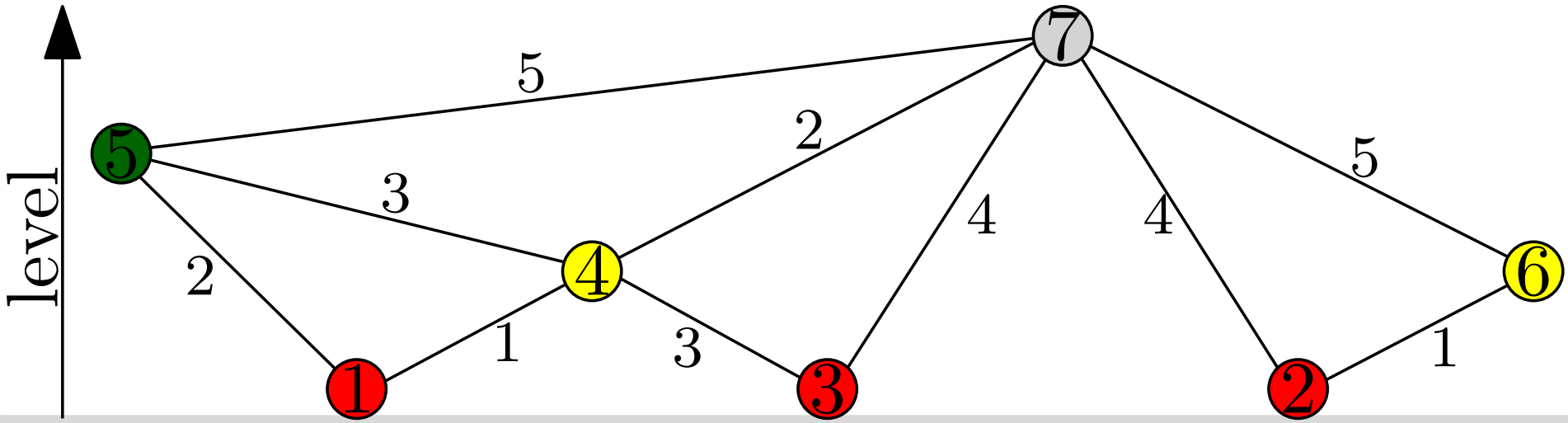
- ordne Knoten
- bearbeite in der Reihenfolge
- shortcutten des Knoten, hinzufügen von Shortcuts (A^+)
- bestimme Level des Knoten



Contraction Hierarchies

Vorberechnung:

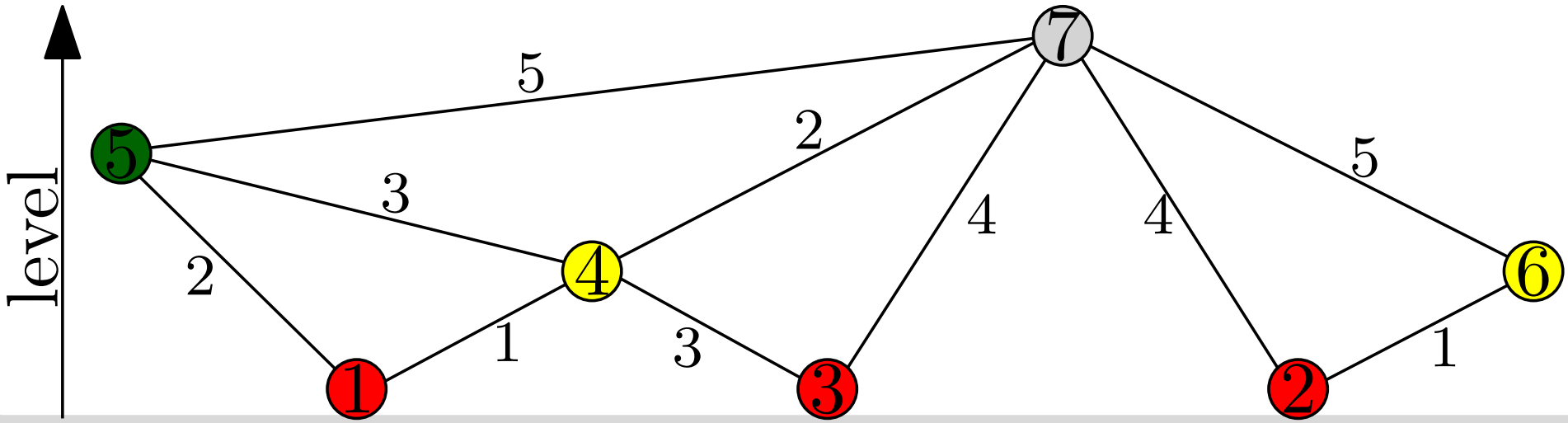
- ordne Knoten
- bearbeite in der Reihenfolge
- shortcutten des Knoten, hinzufügen von Shortcuts (A^+)
- bestimme Level des Knoten
- erzeugt einen Graph $G^+ = (V, A \cup A^+)$



Contraction Hierarchies

Anfrage

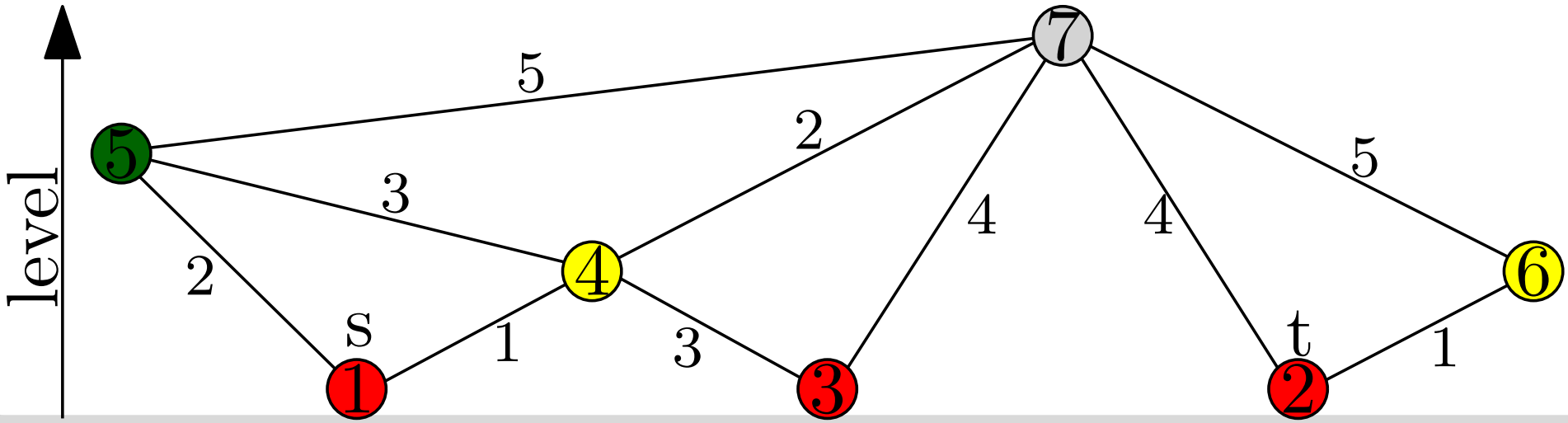
- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten
  Aufwärtsgraph $G_\uparrow := (V, E_\uparrow)$ with $E_\uparrow := \{(u, v) \in E : u < v\}$
  Abwärtsgraph $G_\downarrow := (V, E_\downarrow)$ with $E_\downarrow := \{(u, v) \in E : u > v\}$
- Vorwärtssuche in $G_\uparrow$ und Rückwärtssuche in $G_\downarrow$
- konservatives Stoppkriterium
- jede Suche stoppt wenn Queue leer oder $\mathit{minKey}(Q) > \mu$



Contraction Hierarchies

Anfrage

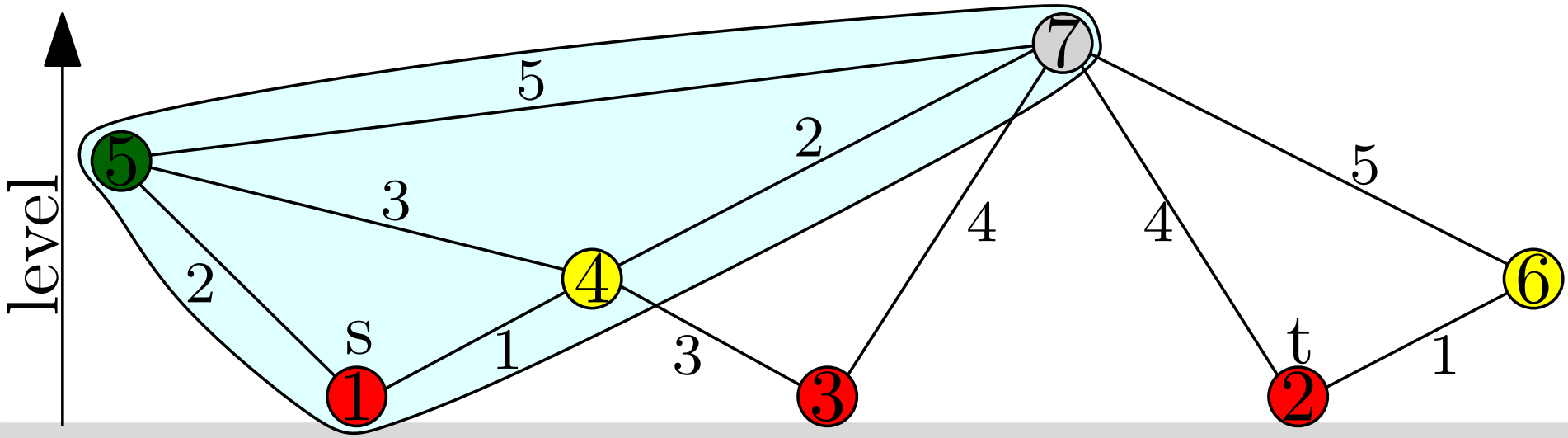
- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten
  Aufwärtsgraph $G_\uparrow := (V, E_\uparrow)$ with $E_\uparrow := \{(u, v) \in E : u < v\}$
  Abwärtsgraph $G_\downarrow := (V, E_\downarrow)$ with $E_\downarrow := \{(u, v) \in E : u > v\}$
- Vorwärtssuche in $G_\uparrow$ und Rückwärtssuche in $G_\downarrow$
- konservatives Stoppkriterium
- jede Suche stoppt wenn Queue leer oder $\mathit{minKey}(Q) > \mu$



Contraction Hierarchies

Anfrage

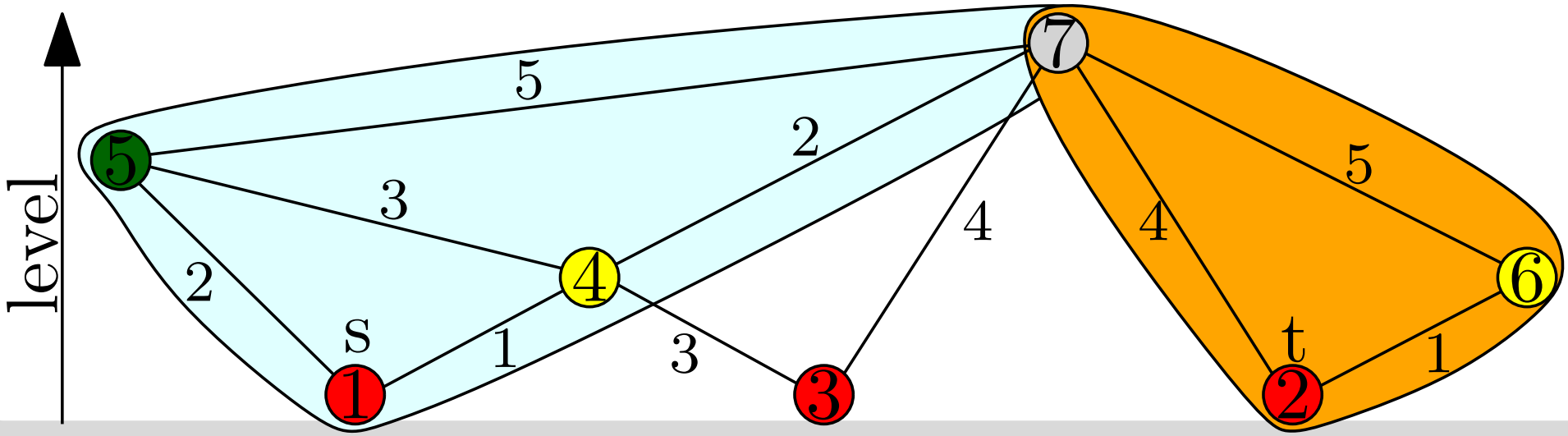
- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten
  Aufwärtsgraph $G_\uparrow := (V, E_\uparrow)$ with $E_\uparrow := \{(u,v) \in E : u < v\}$
  Abwärtsgraph $G_\downarrow := (V, E_\downarrow)$ with $E_\downarrow := \{(u,v) \in E : u > v\}$
- Vorwärtssuche in $G_\uparrow$ und Rückwärtssuche in $G_\downarrow$
- konservatives Stoppkriterium
- jede Suche stoppt wenn Queue leer oder $\mathit{minKey}(Q) > \mu$



Contraction Hierarchies

Anfrage

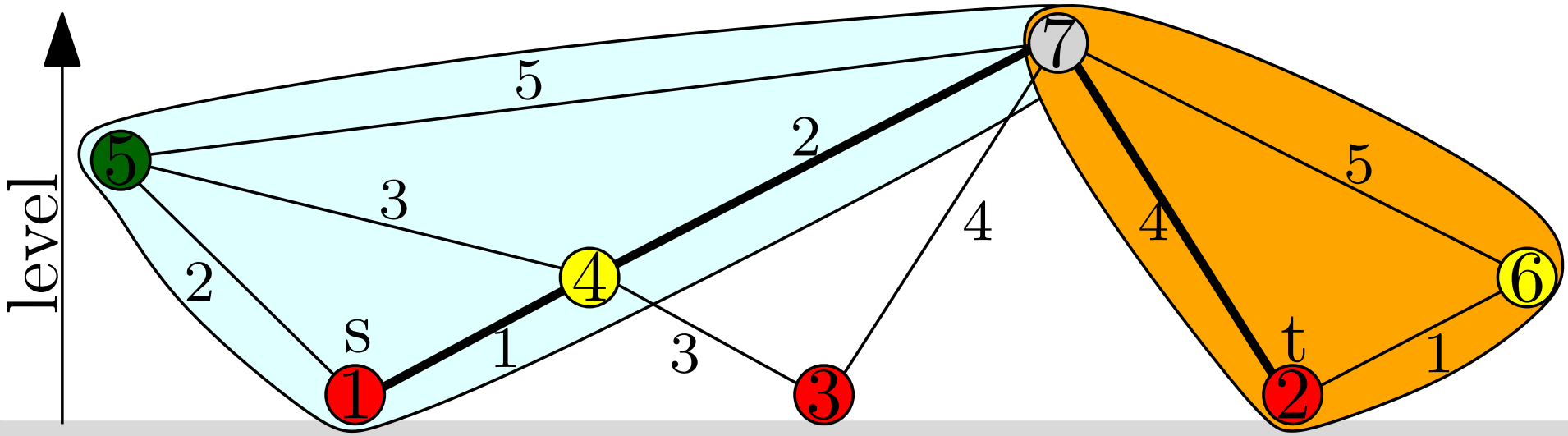
- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten
  Aufwärtsgraph $G_\uparrow := (V, E_\uparrow)$ with $E_\uparrow := \{(u,v) \in E : u < v\}$
  Abwärtsgraph $G_\downarrow := (V, E_\downarrow)$ with $E_\downarrow := \{(u,v) \in E : u > v\}$
- Vorwärtssuche in $G_\uparrow$ und Rückwärtssuche in $G_\downarrow$
- konservatives Stoppkriterium
- jede Suche stoppt wenn Queue leer oder $\mathit{minKey}(Q) > \mu$



Contraction Hierarchies

Anfrage

- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten
  Aufwärtsgraph $G_\uparrow := (V, E_\uparrow)$ with $E_\uparrow := \{(u,v) \in E : u < v\}$
  Abwärtsgraph $G_\downarrow := (V, E_\downarrow)$ with $E_\downarrow := \{(u,v) \in E : u > v\}$
- Vorwärtssuche in $G_\uparrow$ und Rückwärtssuche in $G_\downarrow$
- konservatives Stoppkriterium
- jede Suche stoppt wenn Queue leer oder $\mathit{minKey}(Q) > \mu$



Korrektheit

Beobachtung:

Karlsruher Institut für Technologie
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Korrektheit

Beobachtung:

- der kürzeste s–t Weg P existiert auch in G^+
  (wir fügen nur Kanten hinzu)

Korrektheit

Beobachtung:

- der kürzeste s–t Weg P existiert auch in G^+
  (wir fügen nur Kanten hinzu)
- aber wir suchen nur aufwärts

Korrektheit

Beobachtung:

- der kürzeste s–t Weg P existiert auch in G^+ (wir fügen nur Kanten hinzu)
- aber wir suchen nur aufwärts
- es gibt einen wichtigsten Knoten v auf P

Korrektheit

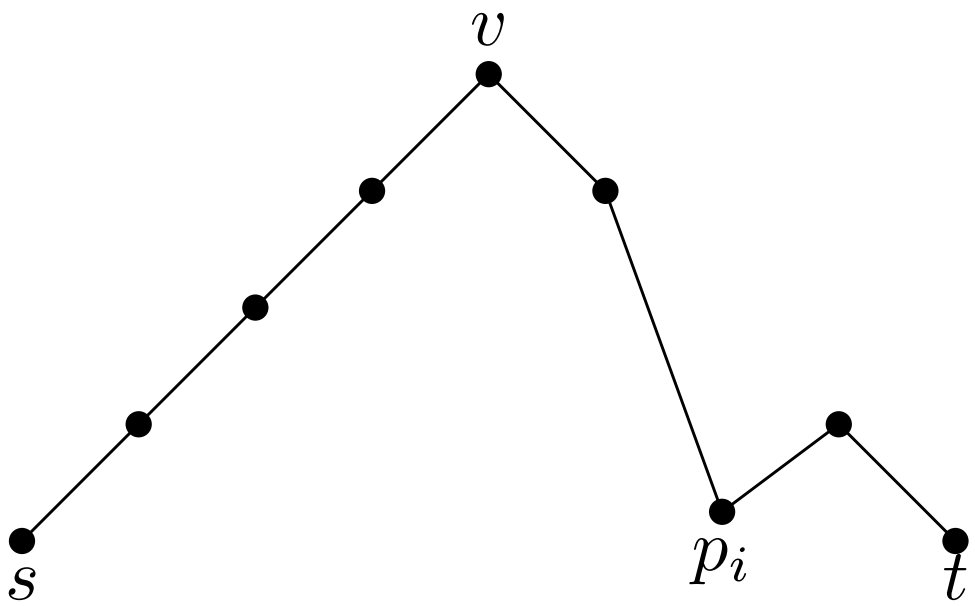
Beobachtung:

- der kürzeste s–t Weg P existiert auch in G^+
  (wir fügen nur Kanten hinzu)
- aber wir suchen nur aufwärts
- es gibt einen wichtigsten Knoten v auf P
- wir müssen zeigen, dass es einen s–aufwärts–v–abwärts–t Pfad gibt

Korrektheit

Beobachtung:

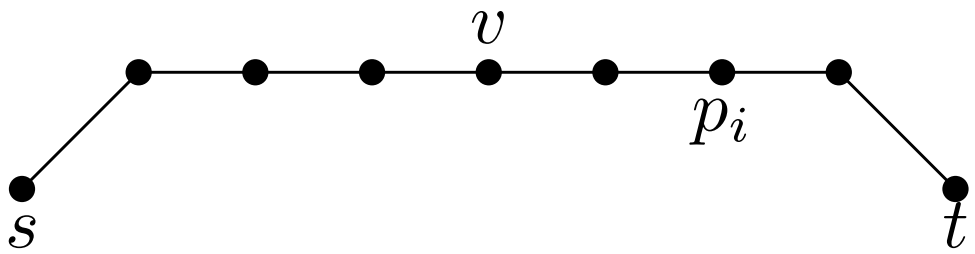
- der kürzeste s–t Weg P existiert auch in G^+
  (wir fügen nur Kanten hinzu)
- aber wir suchen nur aufwärts
- es gibt einen wichtigsten Knoten v auf P
- wir müssen zeigen, dass es einen s–aufwärts–v–abwärts–t Pfad gibt



Korrektheit

Beobachtung:

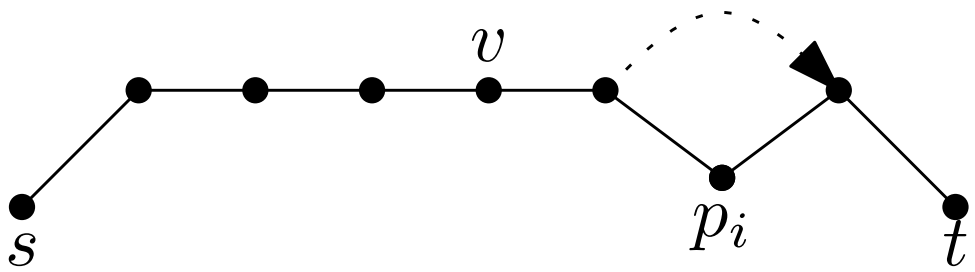
- der kürzeste s–t Weg P existiert auch in G^+
  (wir fügen nur Kanten hinzu)
- aber wir suchen nur aufwärts
- es gibt einen wichtigsten Knoten v auf P
- wir müssen zeigen, dass es einen s–aufwärts–v–abwärts–t Pfad gibt



Korrektheit

Beobachtung:

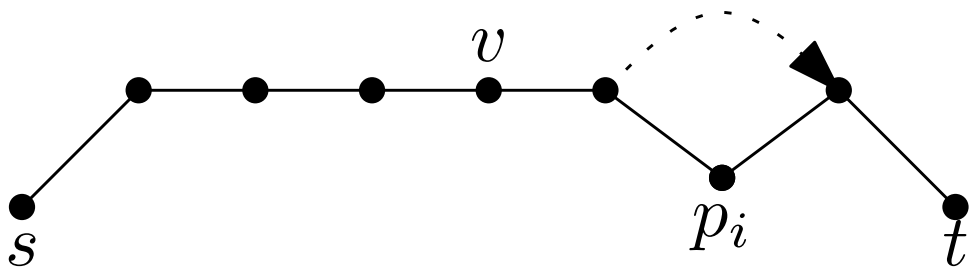
- der kürzeste s–t Weg P existiert auch in G^+
  (wir fügen nur Kanten hinzu)
- aber wir suchen nur aufwärts
- es gibt einen wichtigsten Knoten v auf P
- wir müssen zeigen, dass es einen s–aufwärts–v–abwärts–t Pfad gibt



Korrektheit

Beobachtung:

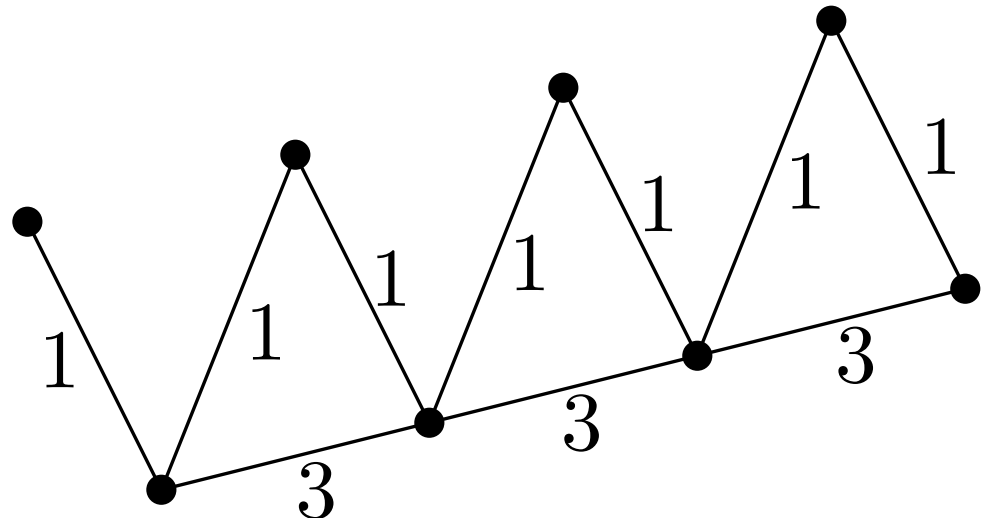
- der kürzeste s–t Weg P existiert auch in G^+
  (wir fügen nur Kanten hinzu)
- aber wir suchen nur aufwärts
- es gibt einen wichtigsten Knoten v auf P
- wir müssen zeigen, dass es einen s–aufwärts–v–abwärts–t Pfad gibt
- verändertes Stoppkriterium:
  es kann $v = s$ oder $v = t$ sein



Stall-On-Demand

Beobachtung:

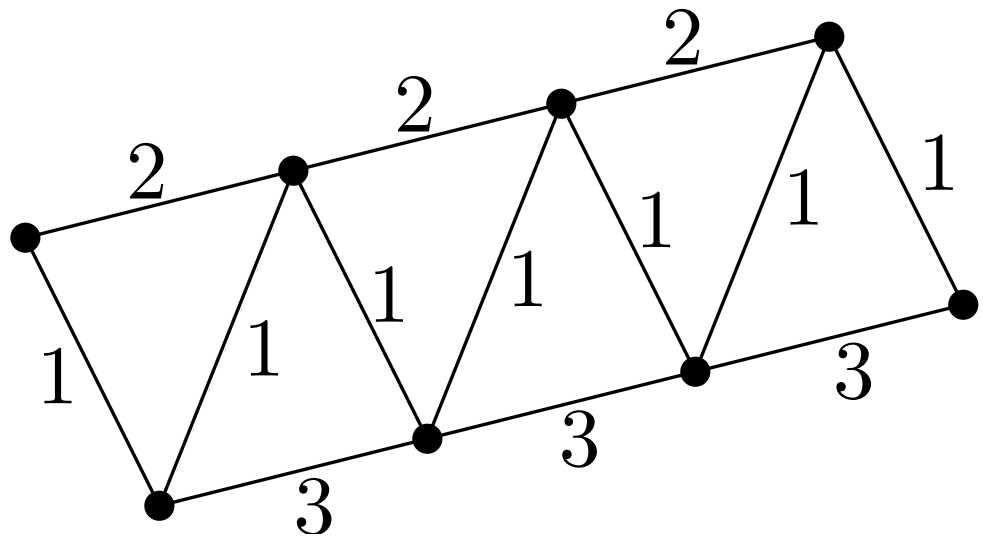
- Aufwärtssuchen können Knoten mit falscher Distanz scannen



Stall-On-Demand

Beobachtung:

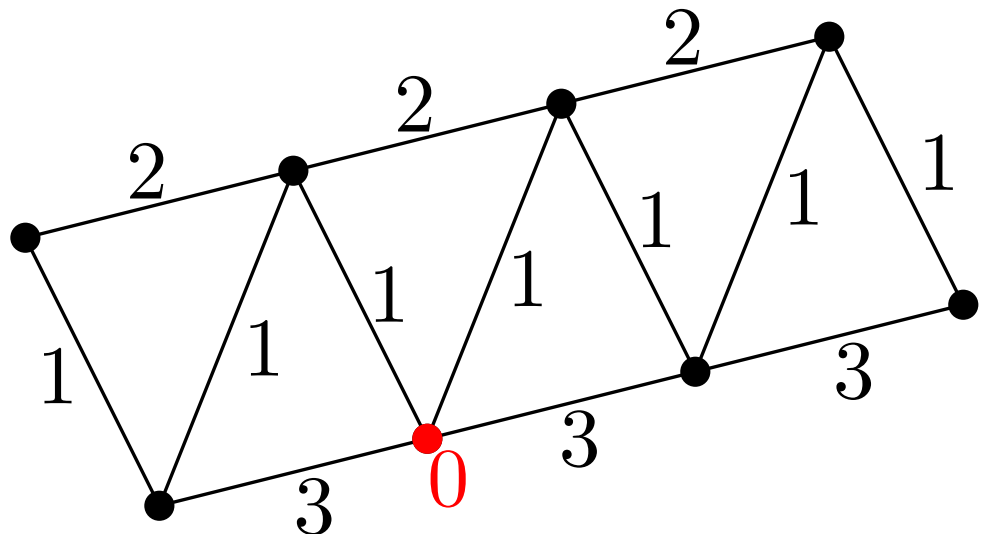
- Aufwärtssuchen können Knoten mit falscher Distanz scannen



Stall-On-Demand

Beobachtung:

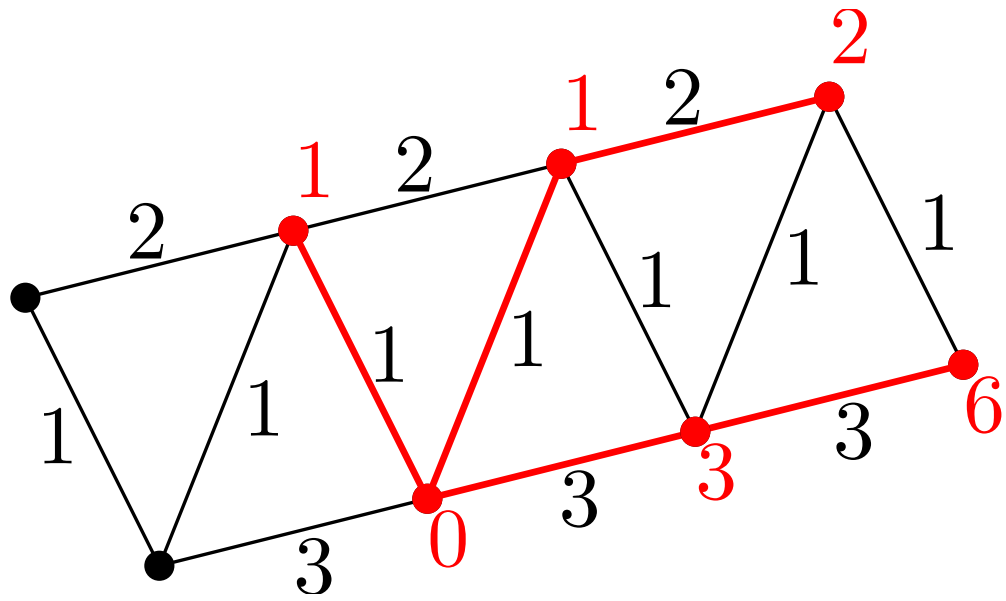
- Aufwärtssuchen können Knoten mit falscher Distanz scannen



Stall-On-Demand

Beobachtung:

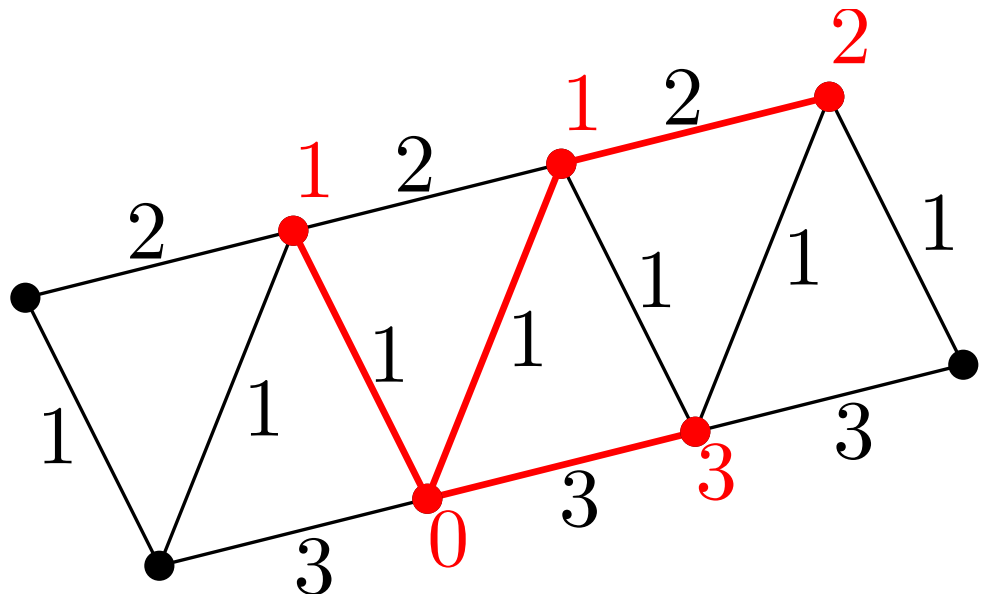
- Aufwärtssuchen können Knoten mit falscher Distanz scannen



Stall-On-Demand

Beobachtung:

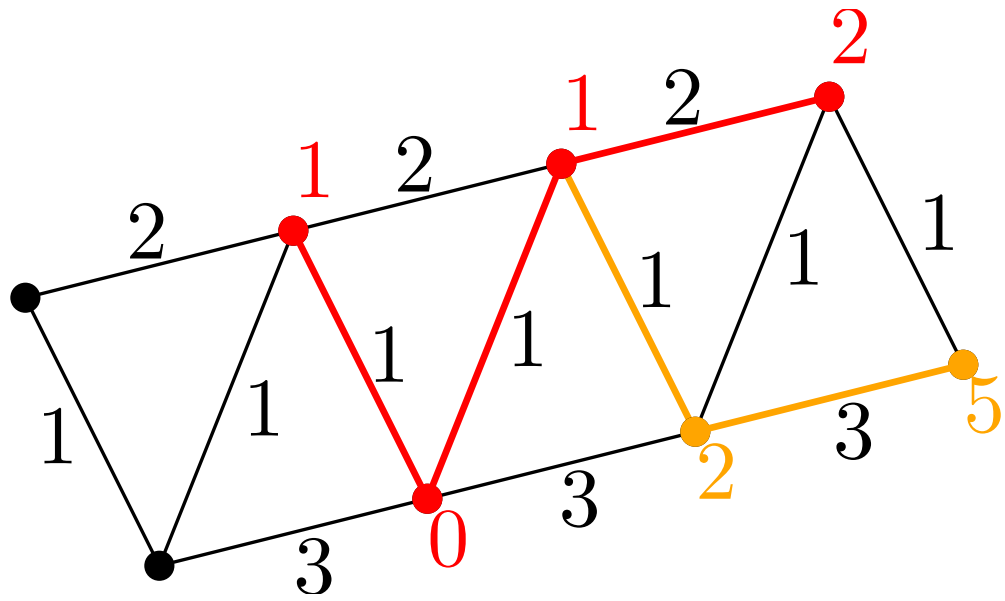
- Aufwärtssuchen können Knoten mit falscher Distanz scannen
- Heuristik versucht das durch scannen von Abwärtskanten zu verhindern
- diese Knoten werden deaktiviert, können aber wieder aktiviert werden
  (für den Fall, dass das Stall (lokal!) propagiert wird)



Stall-On-Demand

Beobachtung:

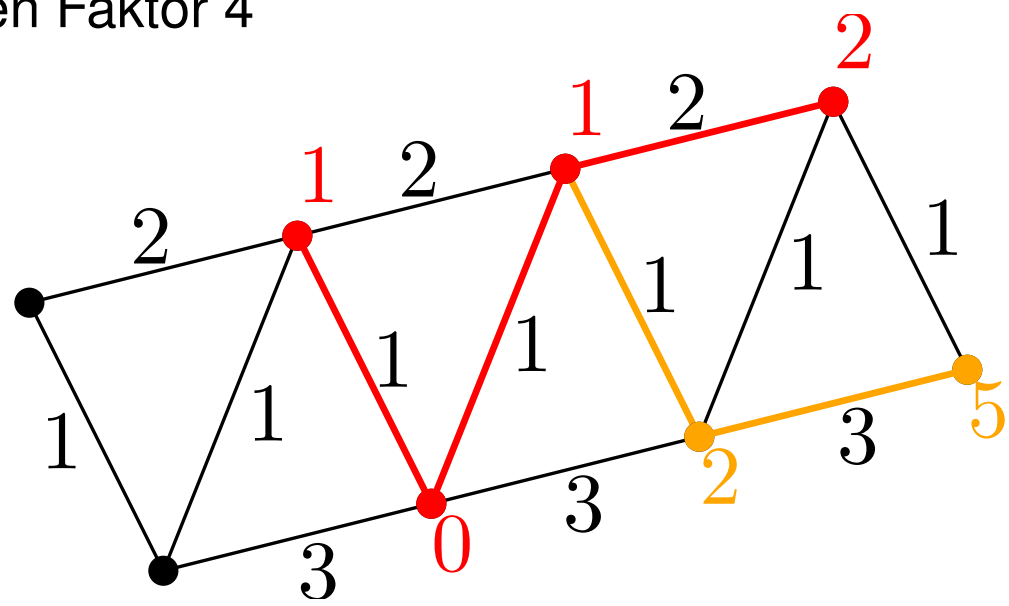
- Aufwärtssuchen können Knoten mit falscher Distanz scannen
- Heuristik versucht das durch scannen von Abwärtskanten zu verhindern
- diese Knoten werden deaktiviert, können aber wieder aktiviert werden
  (für den Fall, dass das Stall (lokal!) propagiert wird)



Stall-On-Demand

Beobachtung:

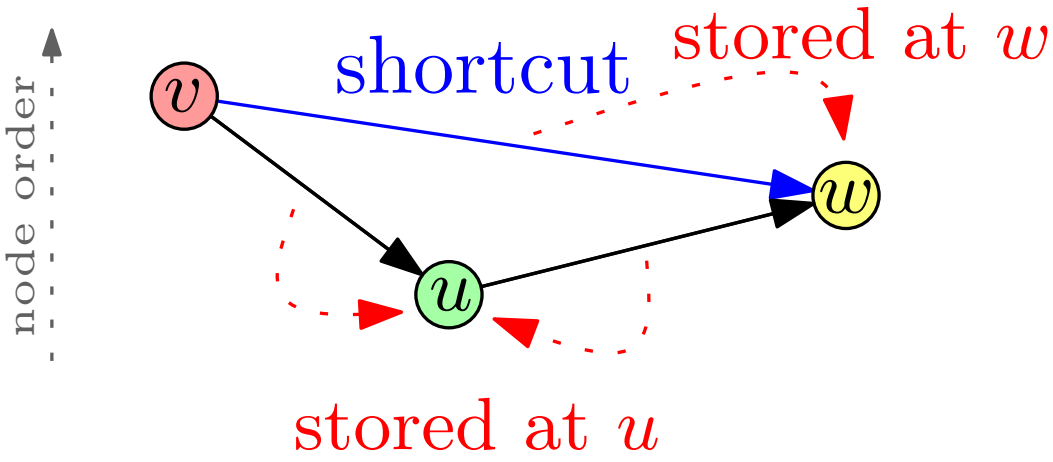
- Aufwärtssuchen können Knoten mit falscher Distanz scannen
- Heuristik versucht das durch scannen von Abwärtskanten zu verhindern
- diese Knoten werden deaktiviert, können aber wieder aktiviert werden
  (für den Fall, dass das Stall (lokal!) propagiert wird)
- reduziert Suchraum um einen Faktor 4



Datenstruktur

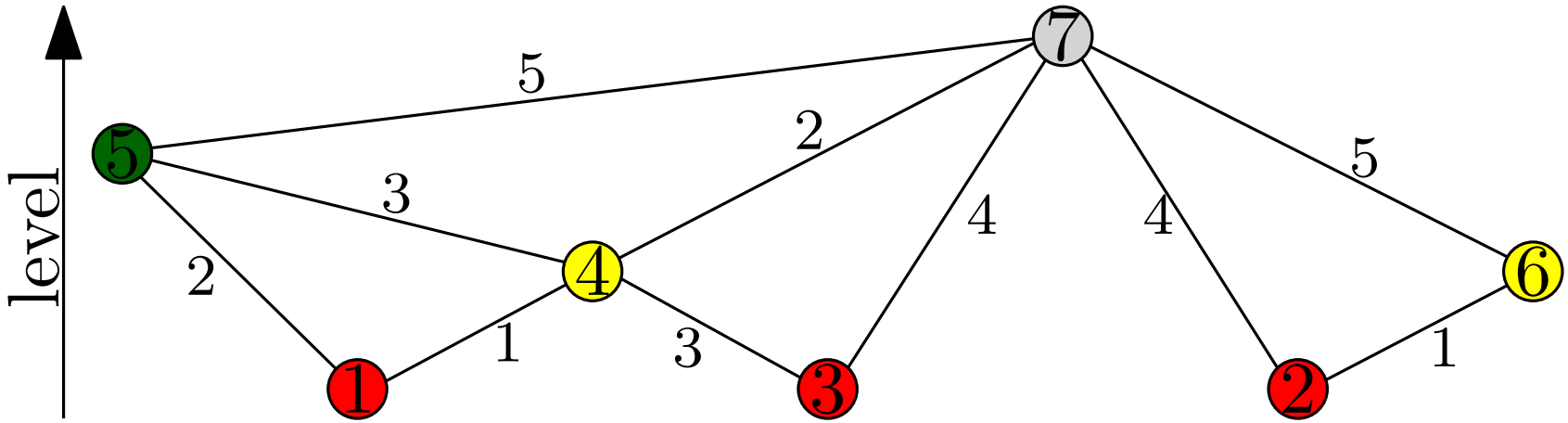
Suchgraph:

- normalerweise: speichere Kanten (v, w) in den Adjacenz-Arrays von v und w
- für die CH-Suche reicht es aus, die Kante nur an den Knoten $\min\{r(v), r(w)\}$ zu speichern
- durch ungerichtete Kanten negativer Speicheroverhead möglich



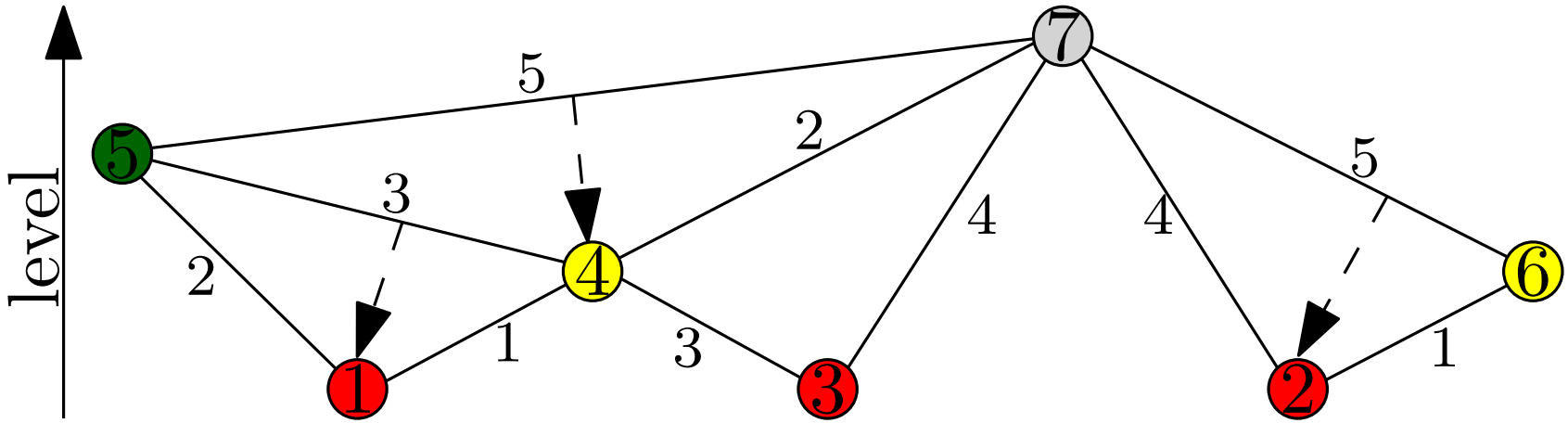
Pfadentpackung

- 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?

- top-down
- bottom-up

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Top-Down

Vorgehen (Greedy Shortest Path Cover):

- bestimme alle kürzesten Wege U, die noch nicht von bereits kontrahierten Knoten überdeckt sind

Top-Down

Vorgehen (Greedy Shortest Path Cover):

- bestimme alle kürzesten Wege U, die noch nicht von bereits kontrahierten Knoten überdeckt sind
- U_v: alle kürzesten Wege, die v enthalten

Top-Down

Vorgehen (Greedy Shortest Path Cover):

- bestimme alle kürzesten Wege U, die noch nicht von bereits kontrahierten Knoten überdeckt sind
- U_v: alle kürzesten Wege, die v enthalten
- S_v: Menge der Startknoten von U_v, T_v der Zielknoten

Top-Down

Vorgehen (Greedy Shortest Path Cover):

- bestimme alle kürzesten Wege U, die noch nicht von bereits kontrahierten Knoten überdeckt sind
- U_v: alle kürzesten Wege, die v enthalten
- S_v: Menge der Startknoten von U_v, T_v der Zielknoten
- kontrahiere Knoten v mit maximalem $|U_v|/(|S_v| + |T_v|)$, balanciert

Top-Down

Vorgehen (Greedy Shortest Path Cover):

- bestimme alle kürzesten Wege U, die noch nicht von bereits kontrahierten Knoten überdeckt sind
- U_v: alle kürzesten Wege, die v enthalten
- S_v: Menge der Startknoten von U_v, T_v der Zielknoten
- kontrahiere Knoten v mit maximalem $|U_v|/(|S_v| + |T_v|)$, balanciert
- aktualisiere $U = U \setminus U_v$

Top-Down

Vorgehen (Greedy Shortest Path Cover):

- bestimme alle kürzesten Wege U, die noch nicht von bereits kontrahierten Knoten überdeckt sind
- U_v: alle kürzesten Wege, die v enthalten
- S_v: Menge der Startknoten von U_v, T_v der Zielknoten
- kontrahiere Knoten v mit maximalem $|U_v|/(|S_v| + |T_v|)$, balanciert
- aktualisiere $U = U \setminus U_v$
- wiederhole bis U leer

Top-Down

Vorgehen (Greedy Shortest Path Cover):

- bestimme alle kürzesten Wege U, die noch nicht von bereits kontrahierten Knoten überdeckt sind
- U_v: alle kürzesten Wege, die v enthalten
- S_v: Menge der Startknoten von U_v, T_v der Zielknoten
- kontrahiere Knoten v mit maximalem $|U_v|/(|S_v| + |T_v|)$, balanciert
- aktualisiere $U = U \setminus U_v$
- wiederhole bis U leer

Diskussion

- Greedy, denn Hitting-Set ist NP-schwer
- n mal APSP ($O(n)$ space)
- kann auf APSP Zeit gedrückt werden ($O(n^2)$ space, inkrementell)
- sehr gute Qualität der Ordnung

Bottom-Up

Vorgehen:

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Bottom-Up

Vorgehen:

- identifiziere aktuell unwichtigsten Knoten

Bottom-Up

Vorgehen:

- identifiziere aktuell unwichtigsten Knoten
- kontrahiere den Knoten

Bottom-Up

Vorgehen:

- identifiziere aktuell unwichtigsten Knoten
- kontrahiere den Knoten
- wiederhole

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Bottom-Up

Vorgehen:

- identifiziere aktuell unwichtigsten Knoten
- kontrahiere den Knoten
- wiederhole

Kriterien zum Bestimmen dieses Knotens:

- Differenz Hinzugefügter und Gelöschter Kanten
- Differenz der Originalkanten, die diese Kanten repräsentieren
- Level des Knoten

Bottom-Up

Vorgehen:

- identifiziere aktuell unwichtigsten Knoten
- kontrahiere den Knoten
- wiederhole

Kriterien zum Bestimmen dieses Knotens:

- Differenz Hinzugefügter und Gelöschter Kanten
- Differenz der Originalkanten, die diese Kanten repräsentieren
- Level des Knoten

Diskussion

- schnell, da alles lokale Information
- kann on-the-fly berechnet werden
- aber nur, solange der Knotengrad gering bleibt
- Qualität der Ordnung ist schlechter (grössere Suchräume)

Range Optimierung (Hybrid)

Idee

- Knotenordnung (z.B. durch bottom-up) gegeben
- zwei Parameter X, Y
- kontrahiere alle Knoten mit $r(v) \leq X$
- bestimme alle kürzesten Wege $U_{\bar{Y}}$ die nicht von Knoten mit $r(v) > Y$ überdeckt sind
- ordne Knoten mit $X < r(v) \leq Y$ mit top-down bzgl. $U_{\bar{Y}}$

Diskussion

- verbessert Qualität der Ordnung
- schnell genug bei geschickter Wahl von X und Y
- kann ineinader verschachtelt, überlappend ausgeführt werden
  somit kann ein schlecht bottom-up als wichtig gerankter Knoten noch wieder ganz nach unten getauscht werden

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-∞	5:42	0.4	200	73.9

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

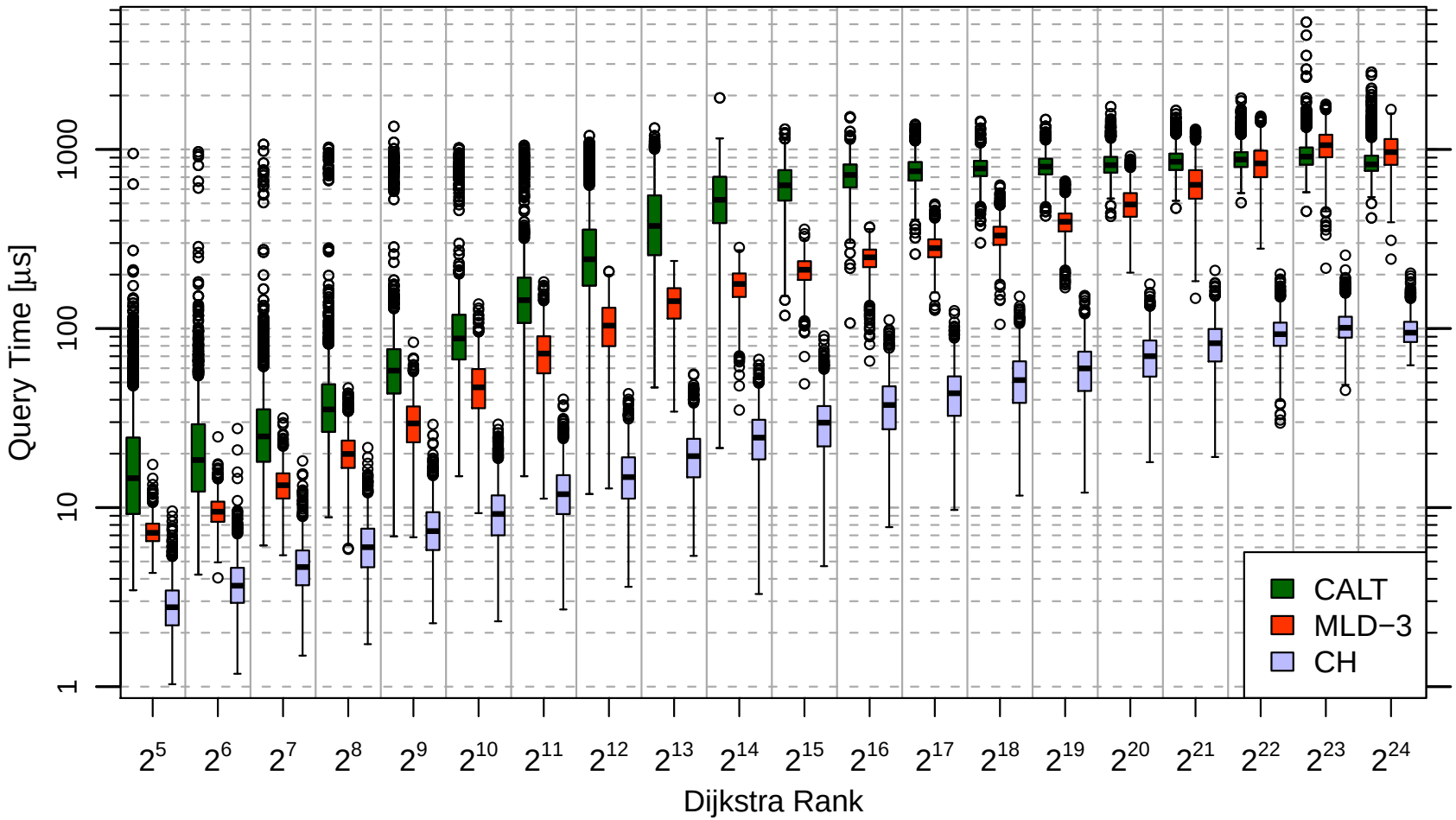
Ergebnisse

	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-∞	5:42	0.4	200	73.9

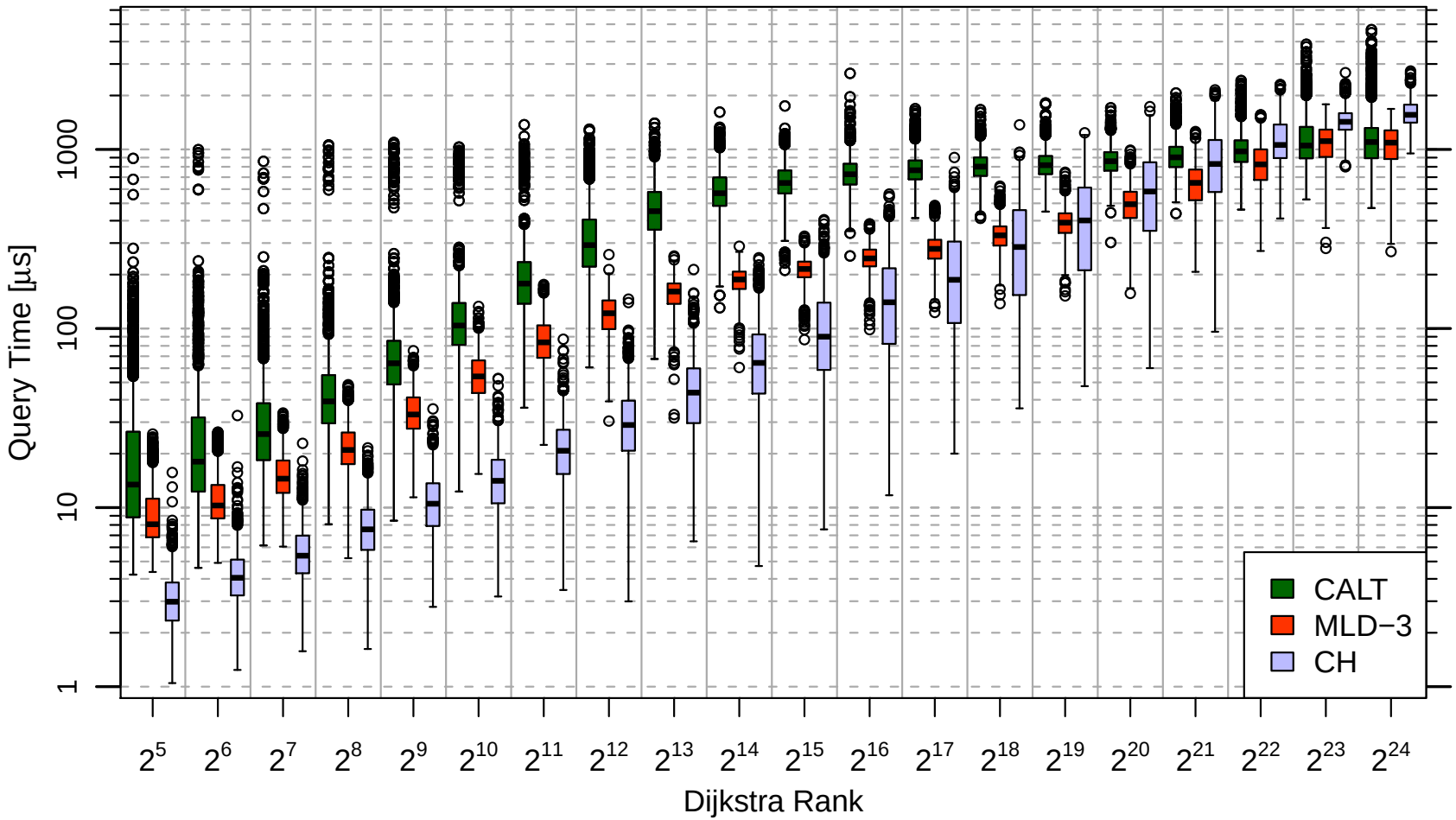
CH-x mit 2^x top-down ordnung, ∞ mit Hybrid Ordnung
zur Erinnerung: Graph hat $> 2^{24}$ Knoten

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

Local Queries: Reisezeiten



Local Queries: Reisedistanzen



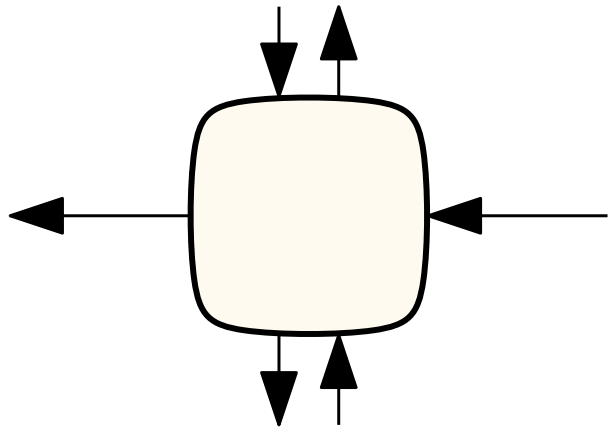
Zusammenfassung CH

- Knotenordnung + Shortcutting
- erzeugt n Overlay graphen
- bidirektionaler Anfragealgorithmus
- einfache und schnelle Vorberechnung
- negativer (?) Speicheroverhead
- Performance stark metrikabhängig
- hohe Beschleunigung für Reisezeiten
- in Produktion von vielen (?) Firmen

Abbiegeverbote/-kosten

bisher:

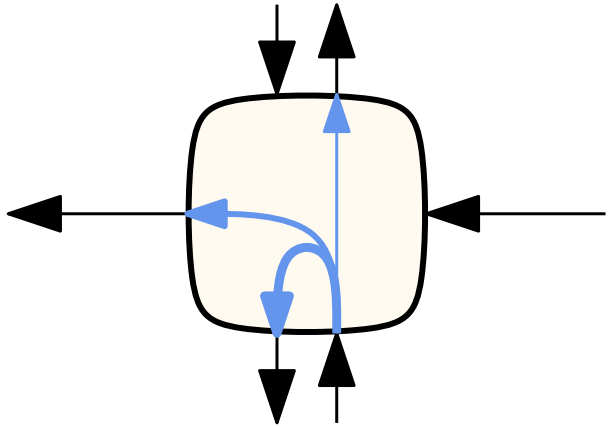
- Kreuzungen $\rightarrow$ Knoten
- Strassen $\rightarrow$ Kanten



Abbiegeverbote/-kosten

bisher:

- Kreuzungen $\rightarrow$ Knoten
- Strassen $\rightarrow$ Kanten



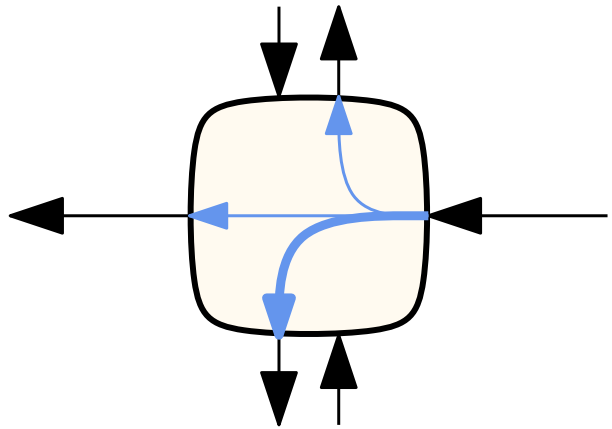
aber:

- Abbiegen manchmal verboten
- Linksabbiegen teurer als rechts
- Kosten U-Turns hoch
- wurde als einfaches Modellierungsdetail abgetan

Abbiegeverbote/-kosten

bisher:

- Kreuzungen $\rightarrow$ Knoten
- Strassen $\rightarrow$ Kanten



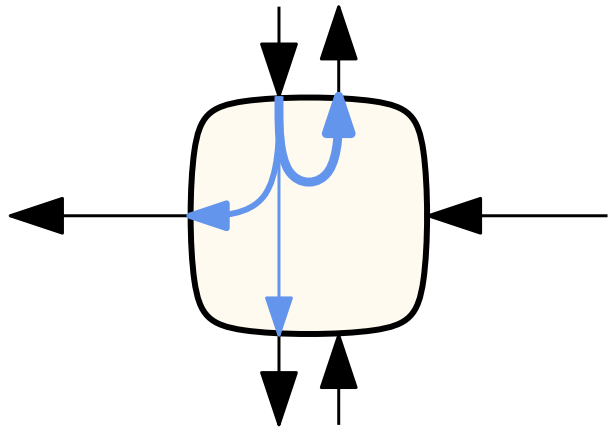
aber:

- Abbiegen manchmal verboten
- Linksabbiegen teurer als rechts
- Kosten U-Turns hoch
- wurde als einfaches Modellierungsdetail abgetan

Abbiegeverbote/-kosten

bisher:

- Kreuzungen $\rightarrow$ Knoten
- Strassen $\rightarrow$ Kanten



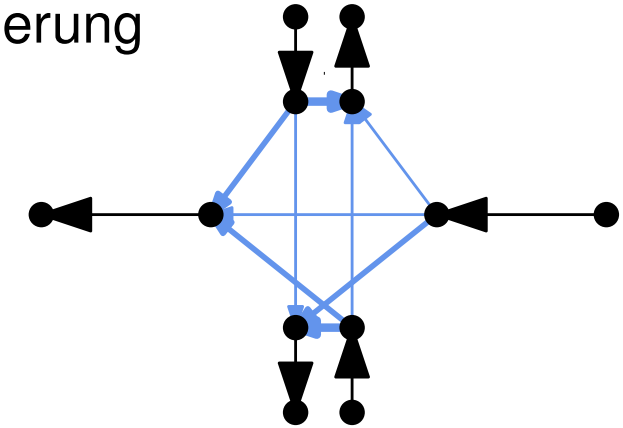
aber:

- Abbiegen manchmal verboten
- Linksabbiegen teurer als rechts
- Kosten U-Turns hoch
- wurde als einfaches Modellierungsdetail abgetan

Modellierung

Möglichkeit I:

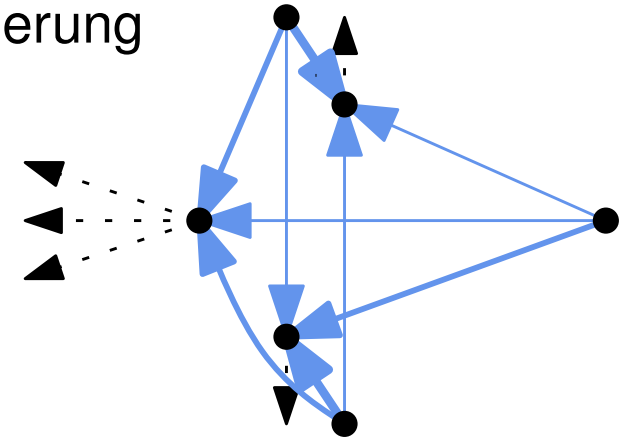
- Vergrössern des Graphen durch Ausmodellierung
- redundante Information



Modellierung

Möglichkeit I:

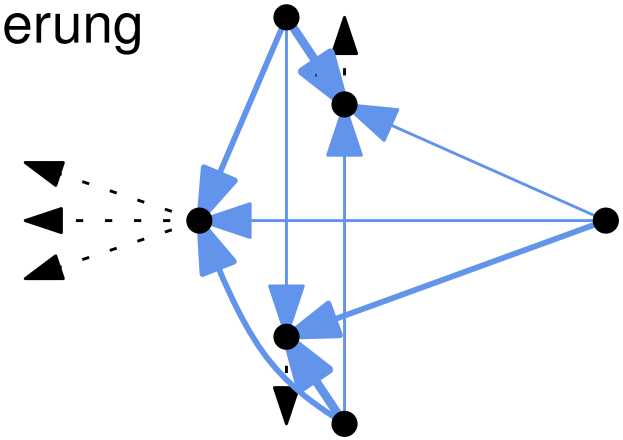
- Vergrössern des Graphen durch Ausmodellierung
- redundante Information
- entferne einen Knoten pro Strasse
- kantenbasierter Graph da
  - Strassen $\rightarrow$ Knoten
  - Turns $\rightarrow$ Kanten



Modellierung

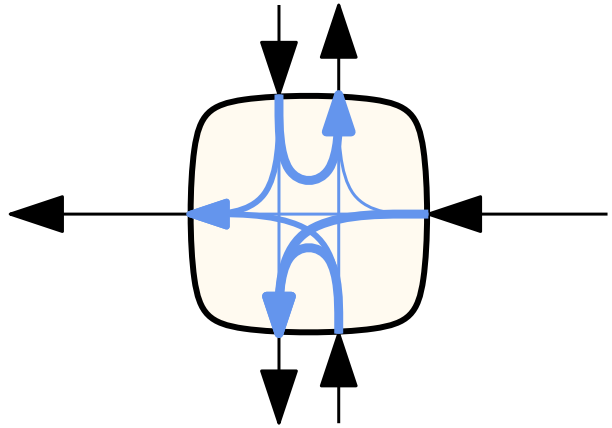
Möglichkeit I:

- Vergrössern des Graphen durch Ausmodellierung
- redundante Information
- entferne einen Knoten pro Strasse
- kantenbasierter Graph da
  - Strassen $\rightarrow$ Knoten
  - Turns $\rightarrow$ Kanten



Möglichkeit II:

- behalte Kreuzungen als Knoten
- speicher Abbiegetabelle
  Abb. Eingangs- $\times$ Ausgangspunkte $\rightarrow$ Kosten
- Beobachtung: viele Knoten haben die gleiche Abbiegetabelle
- also speicher jede Tabelle einmal, Knoten speichern Tabellen-ID



Kantenbasierter Graph

Dijkstra:

- funktioniert ohne Anpassung
- mehr Knoten zu scannen
- Faktor 3-4 langsamer

Kantenbasierter Graph

Dijkstra:

- funktioniert ohne Anpassung
- mehr Knoten zu scannen
- Faktor 3-4 langsamer

CH

- funktioniert ohne Anpassung
- aber grössere Anzahl Knoten/Kanten erhöht Vorberechungszeit

Kantenbasierter Graph

Dijkstra:

- funktioniert ohne Anpassung
- mehr Knoten zu scannen
- Faktor 3-4 langsamer

CH

- funktioniert ohne Anpassung
- aber grössere Anzahl Knoten/Kanten erhöht Vorberechungszeit

MLD

- Anzahl Schnittkanten erhöht sich
- Schnittkanten $=$ Schnittknoten
- (eventuell Wechsel zu Knotenseparatoren sinnvoll?)

Kompaktes Modell

Dijkstra:

- Turns müssen in den Suchalgorithmus integriert werden
- Kreuzungen können mehrfach gescannt werden
  label-correcting bzgl. Kreuzung, label-setting bzgl. Eingangs-/Ausgangspunkte
- jede Kante wird höchstens einmal gescannt
- Suchraum gleich zu kantenbasiertem Modell
  simuliert Dijkstra auf kantenbasiertem Graphen
- Vorteil: weniger Speicher für den Graphen

Optimierung:

- berechne für jede Kreuzung Schranken
- reduziert Suchraum wieder um ca einen Faktor 2
  aber immer noch langsamer als ohne Turns

Kompaktes Modell

CH

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Kompaktes Modell

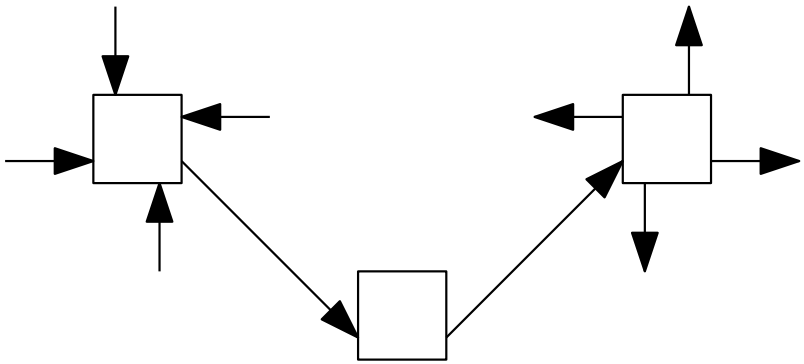
CH

- Zeugensuche wird komplizierter

Kompaktes Modell

CH

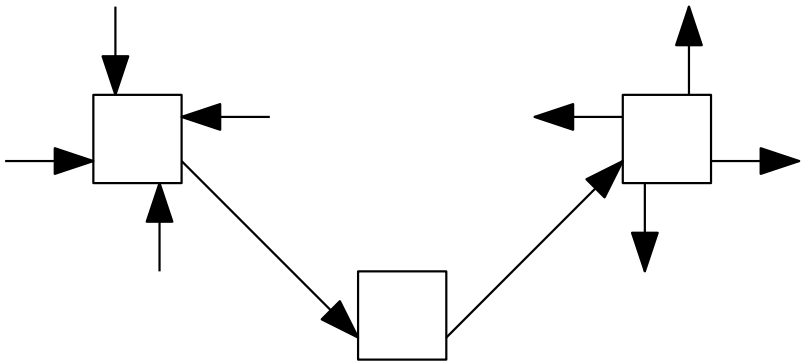
- Zeugensuche wird komplizierter
  - für jedes Paar eingehender und ausgehender Kanten muss eine Zeugensuche durchgeführt werden



Kompaktes Modell

CH

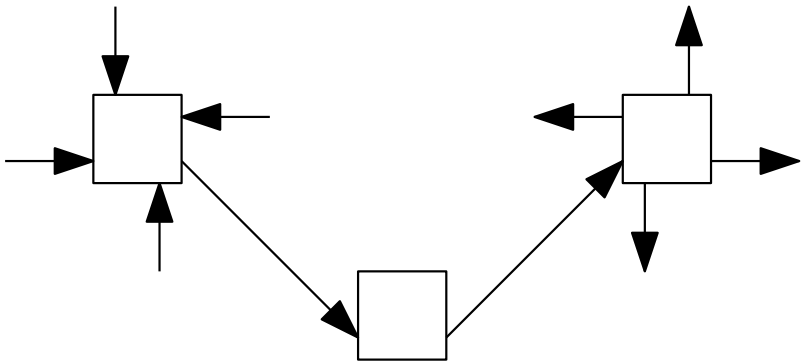
- Zeugensuche wird komplizierter
  - für jedes Paar eingehender und ausgehender Kanten muss eine Zeugensuche durchgeführt werden
  - es können Self-Loops entstehen

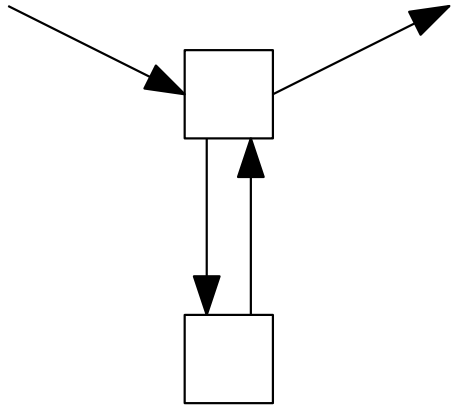


Kompaktes Modell

CH

- Zeugensuche wird komplizierter
  - für jedes Paar eingehender und ausgehender Kanten muss eine Zeugensuche durchgeführt werden
  - es können Self-Loops entstehen

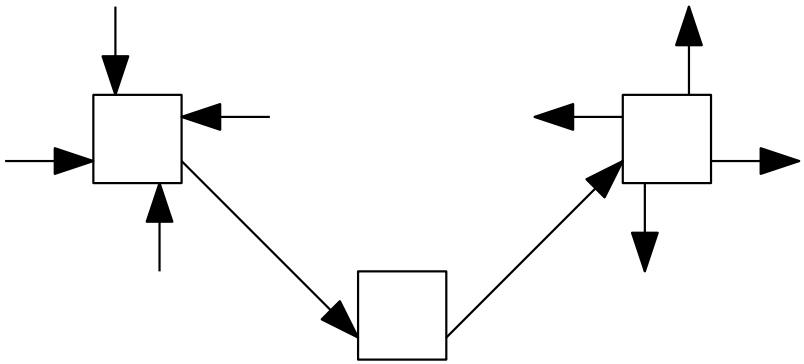


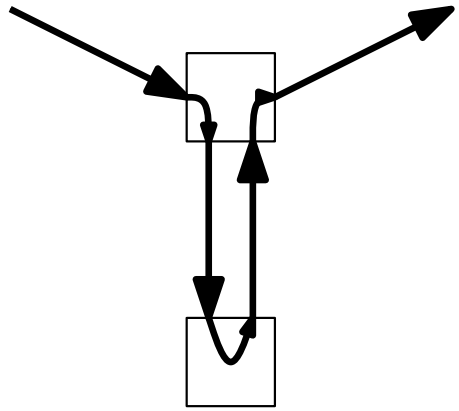


Kompaktes Modell

CH

- Zeugensuche wird komplizierter
  - für jedes Paar eingehender und ausgehender Kanten muss eine Zeugensuche durchgeführt werden
  - es können Self-Loops entstehen

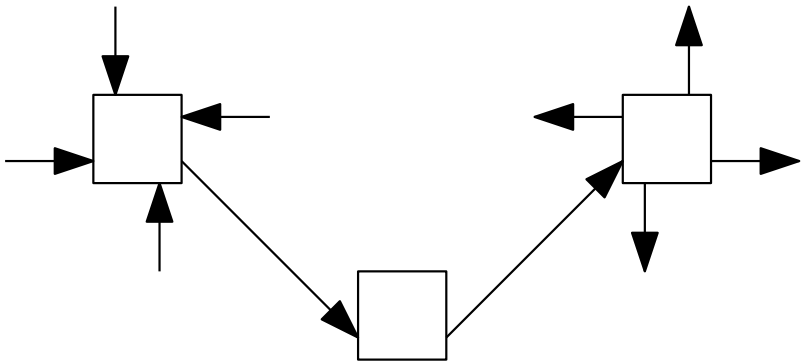


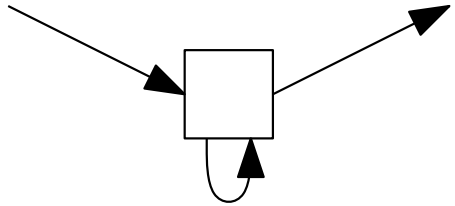


Kompaktes Modell

CH

- Zeugensuche wird komplizierter
  - für jedes Paar eingehender und ausgehender Kanten muss eine Zeugensuche durchgeführt werden
  - es können Self-Loops entstehen





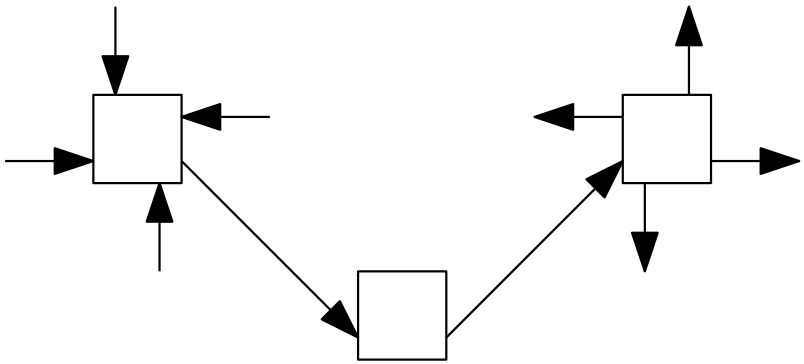
Kompaktes Modell

CH

- Zeugensuche wird komplizierter
  - für jedes Paar eingehender und ausgehender Kanten muss eine Zeugensuche durchgeführt werden
  - es können Self-Loops entstehen

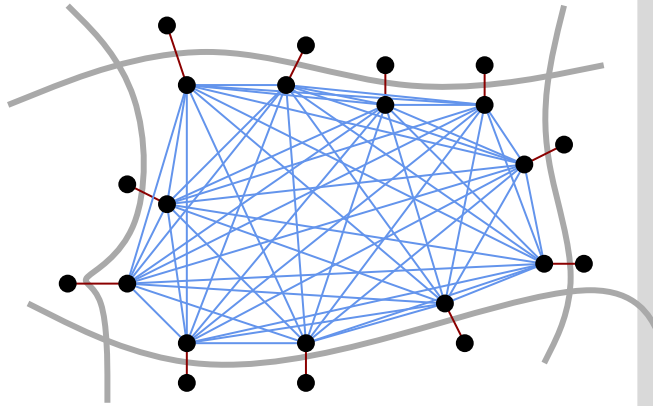
$\Rightarrow$ Anpassung schwierig

$\Rightarrow$ Zeugensuche schlägt häufig fehl



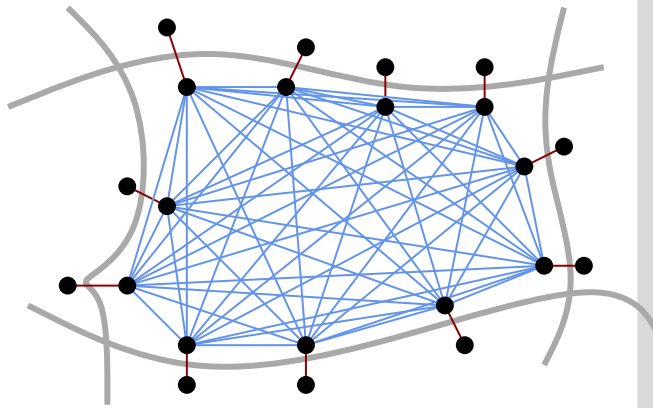
Kompaktes Modell

MLD



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Kompaktes Modell



MLD

- Schnittkanten bleiben erhalten
- Schnittkante $\rightarrow$ 2 Knoten auf Overlay
- Turns müssen nur auf unterstem beachtet werden
- auf Overlaygraphen: normaler Dijkstra

$\Rightarrow$ einfache Anpassung, aber Query wird komplizierter

Ergebnisse

	Algorithm	Customization time [s]	Customization [MB]	Queries #scans	Queries time [ms]
1 s	MLD-4 [$2^8 : 2^{12} : 2^{16} : 2^{20}$]	5.8	61.7	3556	1.18
	CH expanded	3407.4	880.6	550	0.18
	CH compact	849.0	132.5	905	0.19
100 s	MLD-4 [$2^8 : 2^{12} : 2^{16} : 2^{20}$]	7.5	61.7	3813	1.28
	CH expanded	5799.2	931.1	597	0.21
	CH compact	23774.8	304.0	5585	2.11

Beobachtung:

- CH hat massive Probleme mit Turns
- MLD kaum (einer der Hauptgründe für Entwicklung von MLD)

Labeling

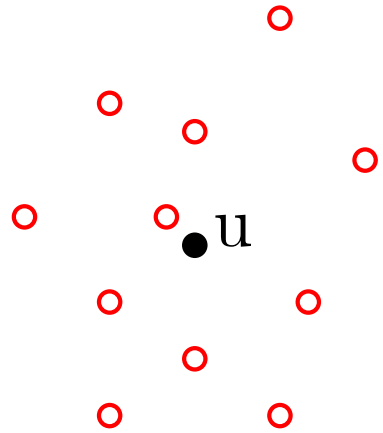
Vorberechung:

- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$

Labeling

Vorberechung:

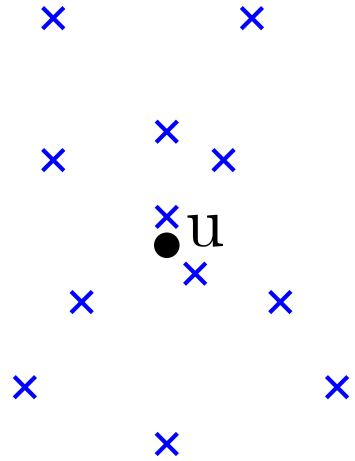
- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
  - dist(u, v) für jeden Hub $v \in L_f(u)$



Labeling

Vorberechung:

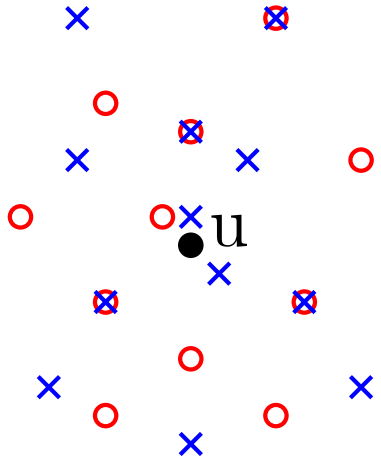
- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
  - $\text{dist}(u, v)$ für jeden Hub $v \in L_f(u)$
  - $\text{dist}(v, u)$ für jeden Hub $v \in L_b(u)$



Labeling

Vorberechung:

- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
  - $\text{dist}(u, v)$ für jeden Hub $v \in L_f(u)$
  - $\text{dist}(v, u)$ für jeden Hub $v \in L_b(u)$



Labeling

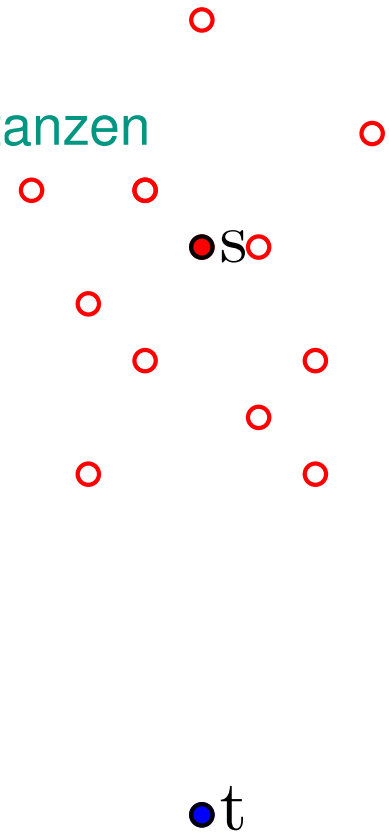
Vorberechung:

- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
  - dist(u, v) für jeden Hub $v \in L_f(u)$
  - dist(v, u) für jeden Hub $v \in L_b(u)$
- die Label müssen die cover property einhalten:
  $\forall s, t$, $L_f(s) \cap L_b(t)$ überdeckt den kürzesten s–t Pfad

Labeling

Vorberechung:

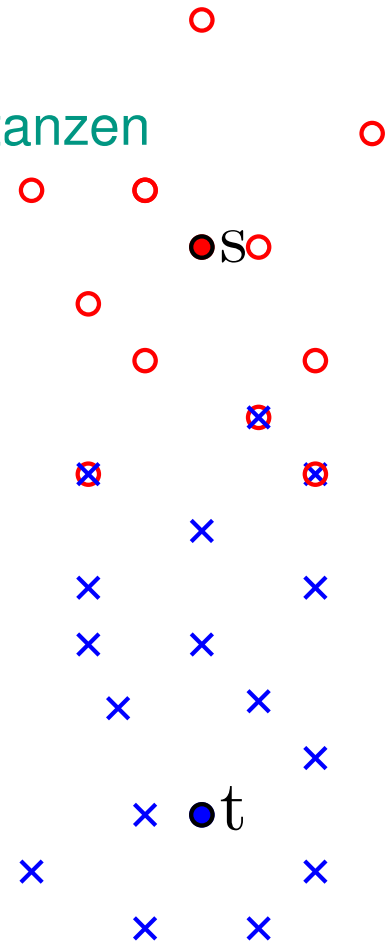
- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
  - $\text{dist}(u, v)$ für jeden Hub $v \in L_f(u)$
  - $\text{dist}(v, u)$ für jeden Hub $v \in L_b(u)$
- die Label müssen die cover property einhalten:
  $\forall s, t,\ L_f(s) \cap L_b(t)$ überdeckt den kürzesten s–t Pfad



Labeling

Vorberechung:

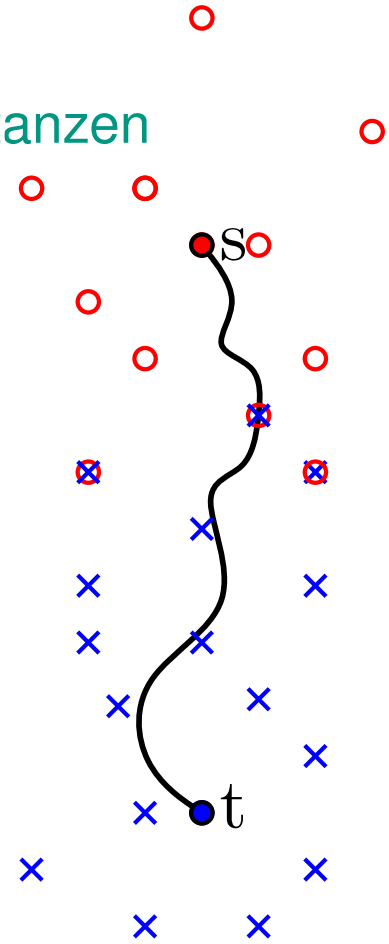
- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
  - $\text{dist}(u, v)$ für jeden Hub $v \in L_f(u)$
  - $\text{dist}(v, u)$ für jeden Hub $v \in L_b(u)$
- die Label müssen die cover property einhalten:
  $\forall s, t,\ L_f(s) \cap L_b(t)$ überdeckt den kürzesten s–t Pfad



Labeling

Vorberechung:

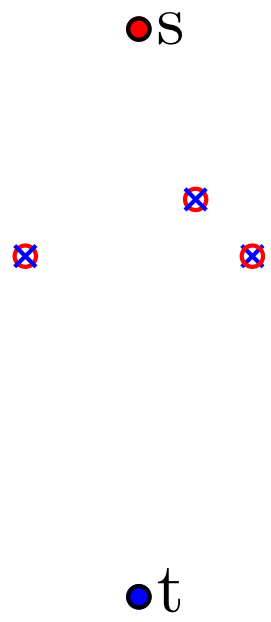
- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
  - $\text{dist}(u, v)$ für jeden Hub $v \in L_f(u)$
  - $\text{dist}(v, u)$ für jeden Hub $v \in L_b(u)$
- die Label müssen die cover property einhalten:
  $\forall s, t,\ L_f(s) \cap L_b(t)$ überdeckt den kürzesten s–t Pfad



Labeling

Vorberechung:

- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
  - $\text{dist}(u, v)$ für jeden Hub $v \in L_f(u)$
  - $\text{dist}(v, u)$ für jeden Hub $v \in L_b(u)$
- die Label müssen die cover property einhalten:
  $\forall s, t$, $L_f(s) \cap L_b(t)$ überdeckt den kürzesten s–t Pfad



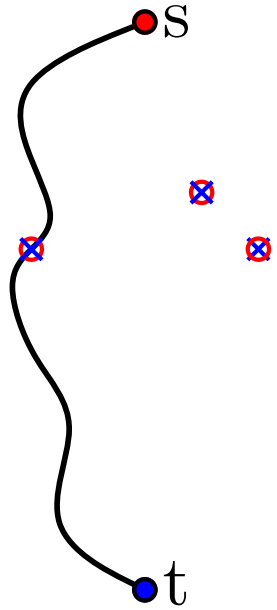
s–t Anfrage:

- finde Knoten $v \in L_f(s) \cap L_b(t)$ ...

Labeling

Vorberechung:

- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
  - $\text{dist}(u, v)$ für jeden Hub $v \in L_f(u)$
  - $\text{dist}(v, u)$ für jeden Hub $v \in L_b(u)$
- die Label müssen die cover property einhalten:
  $\forall s, t$, $L_f(s) \cap L_b(t)$ überdeckt den kürzesten s–t Pfad



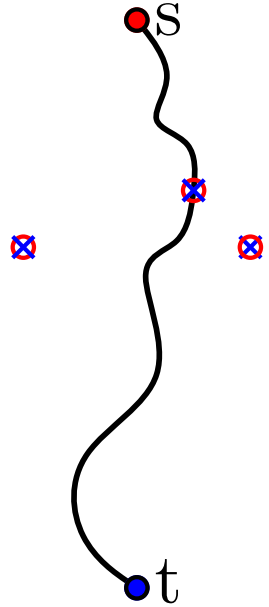
s–t Anfrage:

- finde Knoten $v \in L_f(s) \cap L_b(t)$ ...
- ... der $\text{dist}(s, v) + \text{dist}(v, t)$ minimiert

Labeling

Vorberechung:

- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
  - $\text{dist}(u, v)$ für jeden Hub $v \in L_f(u)$
  - $\text{dist}(v, u)$ für jeden Hub $v \in L_b(u)$
- die Label müssen die cover property einhalten:
  $\forall s, t$, $L_f(s) \cap L_b(t)$ überdeckt den kürzesten s–t Pfad



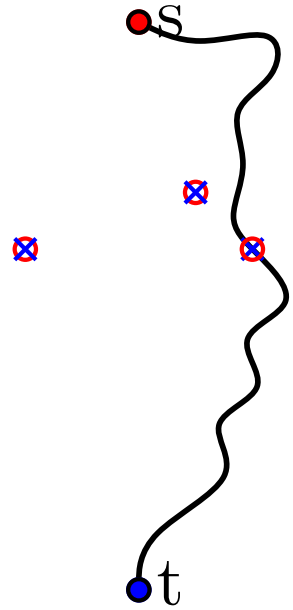
s–t Anfrage:

- finde Knoten $v \in L_f(s) \cap L_b(t)$ ...
- ... der $\text{dist}(s, v) + \text{dist}(v, t)$ minimiert

Labeling

Vorberechung:

- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
  - $\text{dist}(u, v)$ für jeden Hub $v \in L_f(u)$
  - $\text{dist}(v, u)$ für jeden Hub $v \in L_b(u)$
- die Label müssen die cover property einhalten:
  $\forall s, t$, $L_f(s) \cap L_b(t)$ überdeckt den kürzesten s–t Pfad



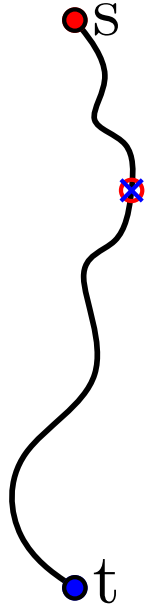
s–t Anfrage:

- finde Knoten $v \in L_f(s) \cap L_b(t)$ ...
- ... der $\text{dist}(s, v) + \text{dist}(v, t)$ minimiert

Labeling

Vorberechung:

- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
  - $\text{dist}(u, v)$ für jeden Hub $v \in L_f(u)$
  - $\text{dist}(v, u)$ für jeden Hub $v \in L_b(u)$
- die Label müssen die cover property einhalten:
  $\forall s, t$, $L_f(s) \cap L_b(t)$ überdeckt den kürzesten s–t Pfad



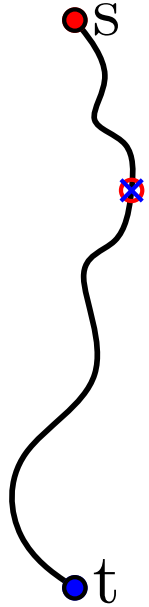
s–t Anfrage:

- finde Knoten $v \in L_f(s) \cap L_b(t)$ ...
- ... der $\text{dist}(s, v) + \text{dist}(v, t)$ minimiert

Labeling

Vorberechung:

- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
  - $\text{dist}(u, v)$ für jeden Hub $v \in L_f(u)$
  - $\text{dist}(v, u)$ für jeden Hub $v \in L_b(u)$
- die Label müssen die cover property einhalten:
  $\forall s, t$, $L_f(s) \cap L_b(t)$ überdeckt den kürzesten s–t Pfad



s–t Anfrage:

- finde Knoten $v \in L_f(s) \cap L_b(t)$ ...
- ... der $\text{dist}(s, v) + \text{dist}(v, t)$ minimiert

Beobachtungen:

- Laufzeit hängt von Labelgrösse ab
- wie effizient berechnen?

Hub Labels

Speichern der Labels:

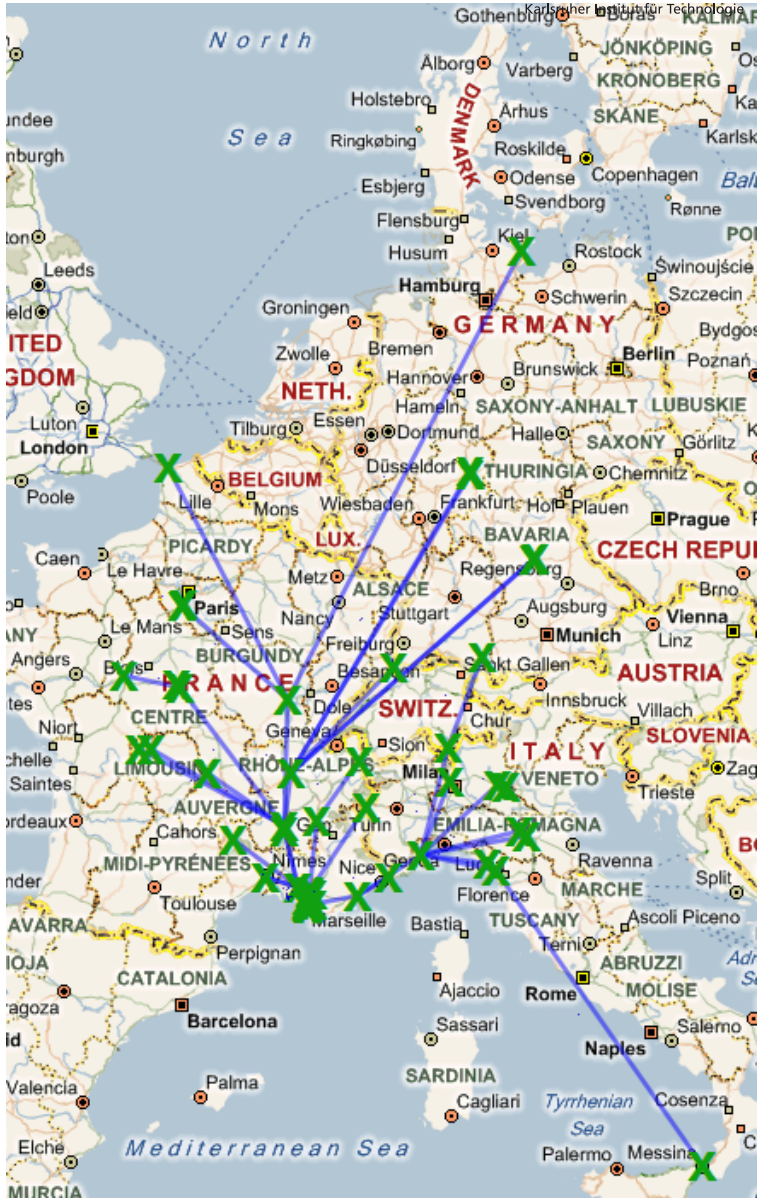
- als Menge von Hub,Distanz Paaren

Hub Labels

Speichern der Labels:

- als Menge von Hub,Distanz Paaren

$L_f(s)$ | 1,0 | 4,1 | 5,2 | 7,3 |



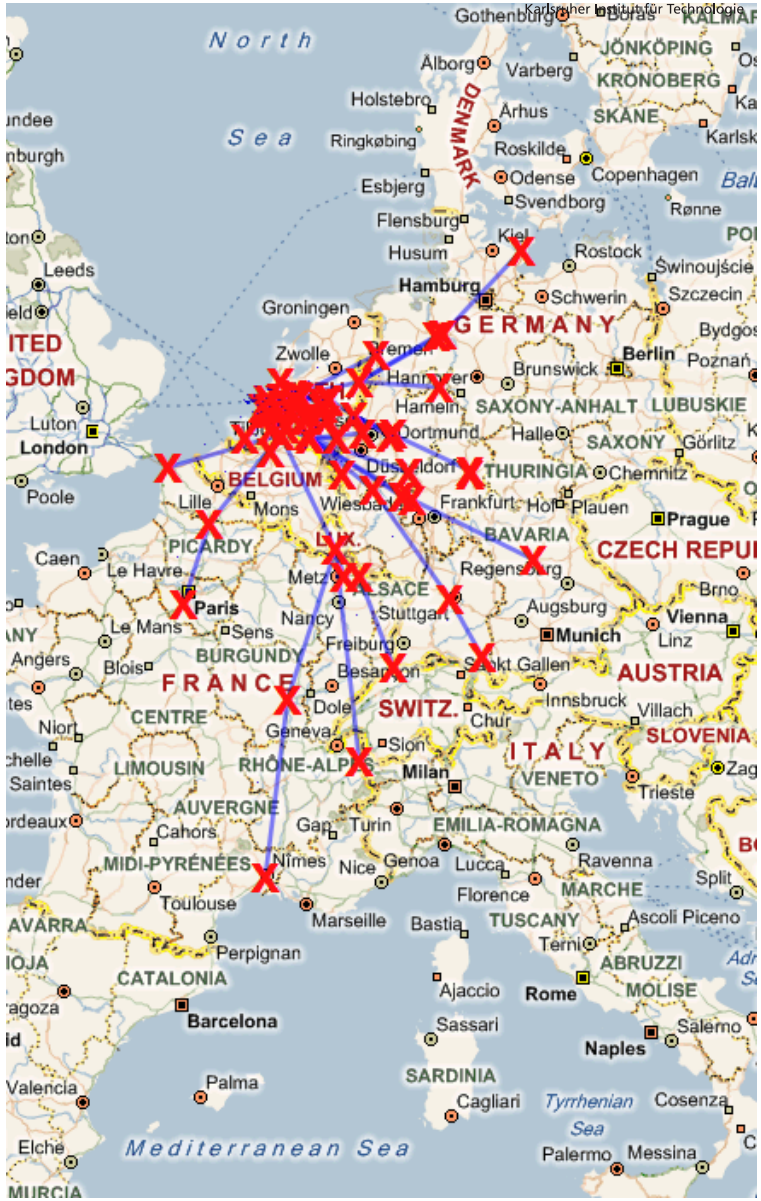
Hub Labels

Speichern der Labels:

- als Menge von Hub,Distanz Paaren

$L_f(s)$ | 1,0 | 4,1 | 5,2 | 7,3 |

$L_b(t)$ | 2,0 | 6,1 | 7,4 | 8,1 | 9,3 |



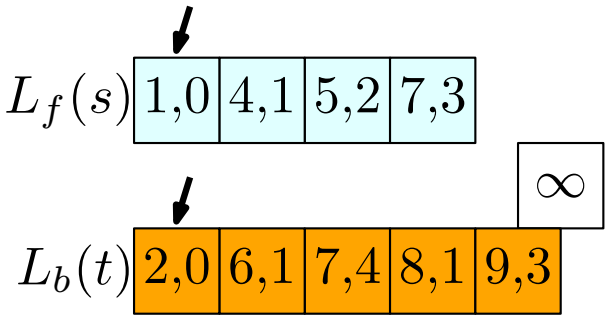
Hub Labels

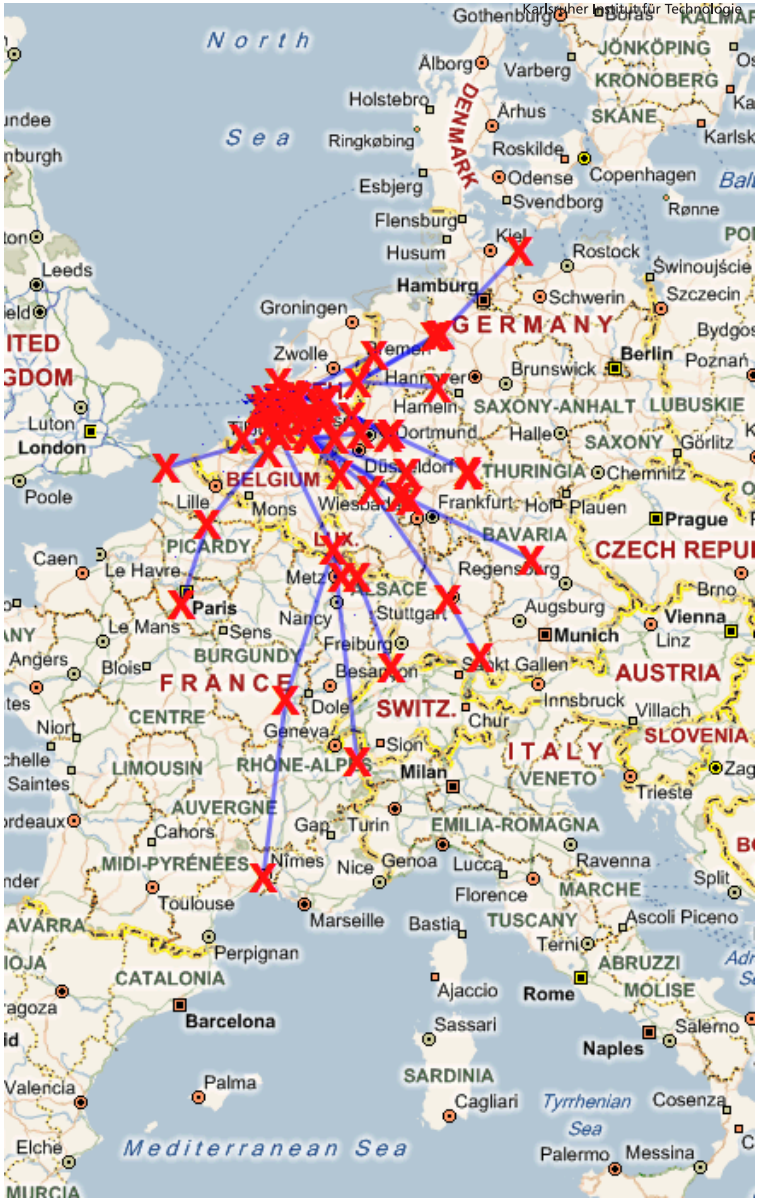
Speichern der Labels:

- als Menge von Hub,Distanz Paaren

Anfrage:

- scannen von zwei Arrays
- nur einige Speicherzugriffe nötig





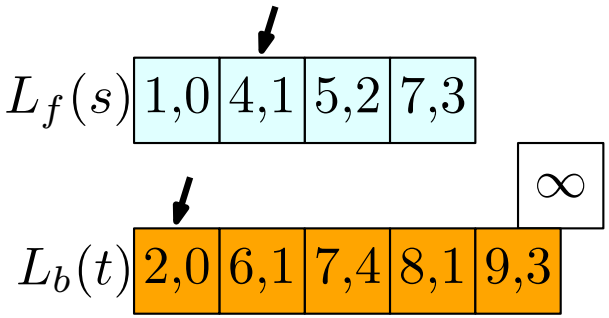
Hub Labels

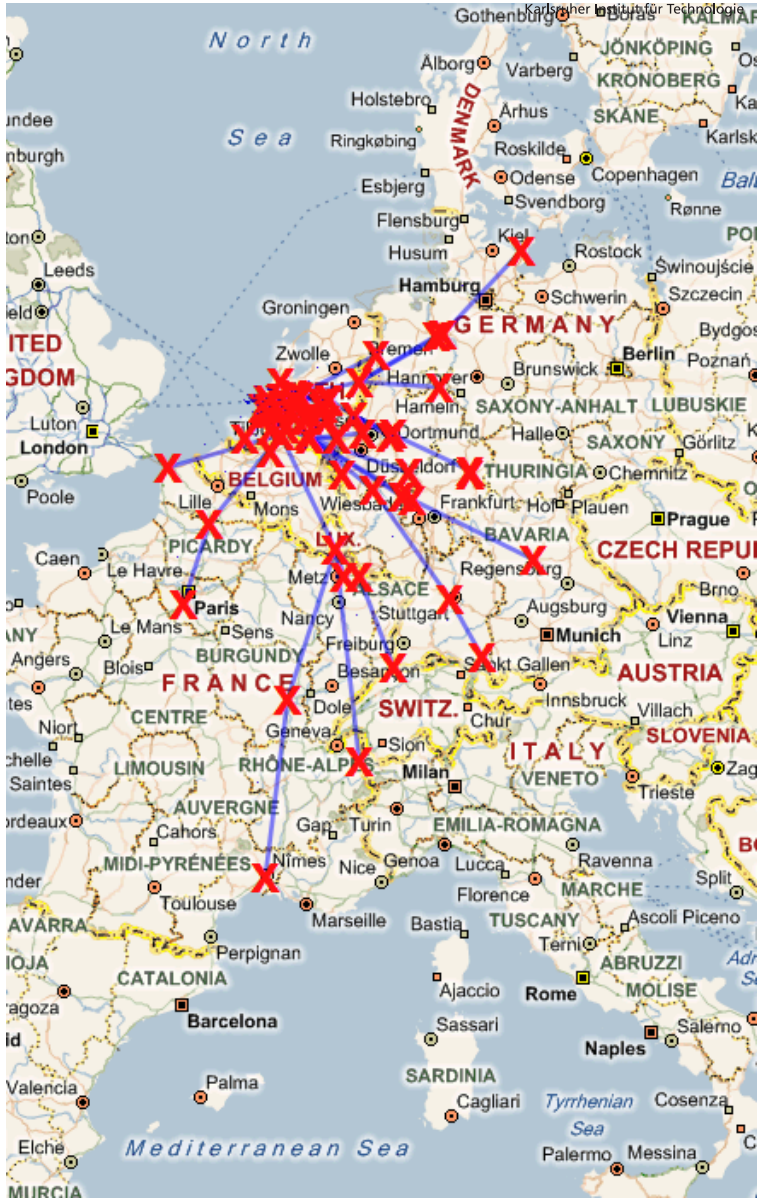
Speichern der Labels:

- als Menge von Hub,Distanz Paaren

Anfrage:

- scannen von zwei Arrays
- nur einige Speicherzugriffe nötig





Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Hub Labels

Speichern der Labels:

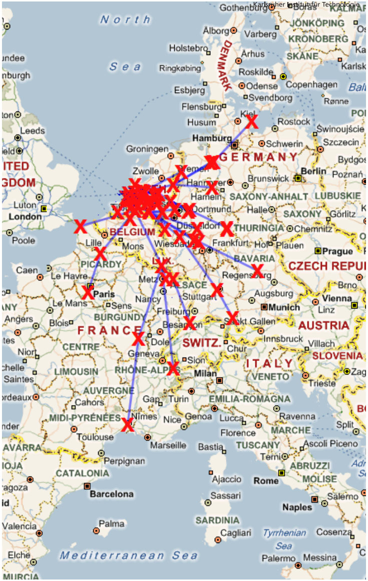
- als Menge von Hub,Distanz Paaren

Anfrage:

- scannen von zwei Arrays
- nur einige Speicherzugriffe nötig

$L_f(s)$	1,0	4,1	5,2	7,3	∞

$L_b(t)$	2,0	6,1	7,4	8,1	9,3



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

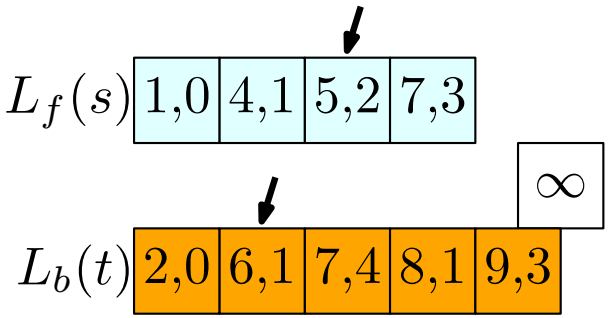
Hub Labels

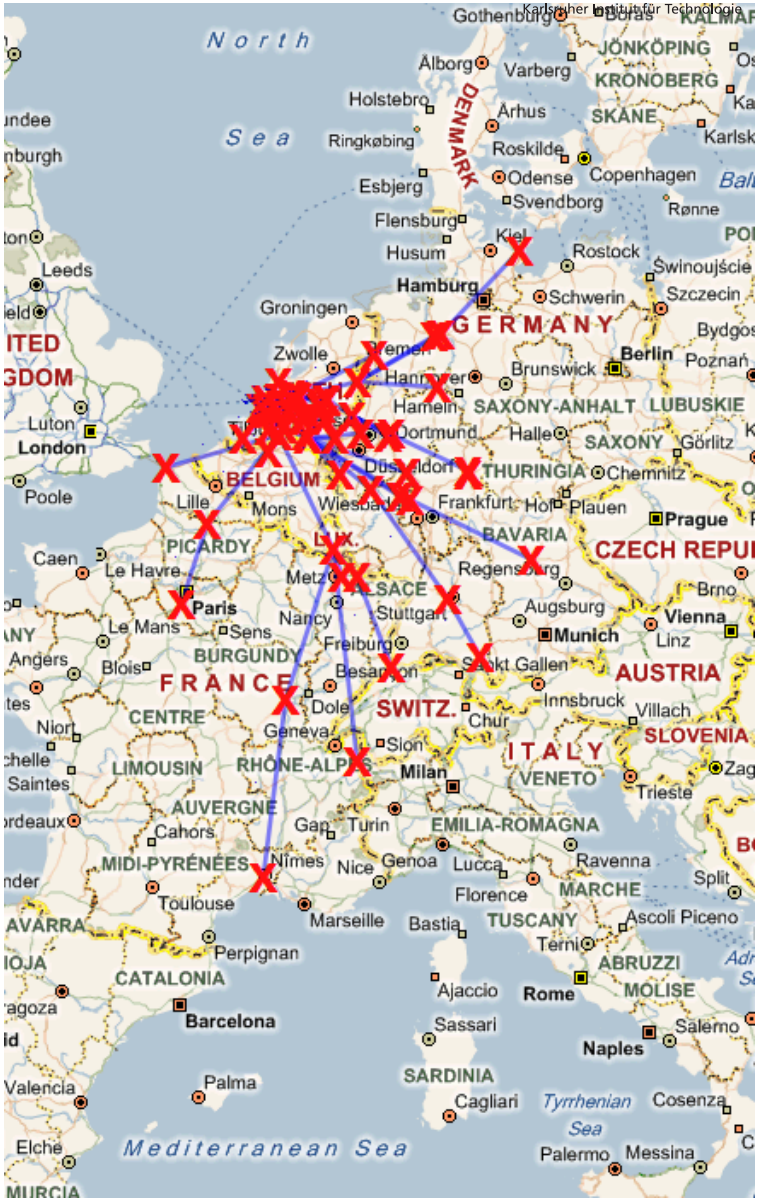
Speichern der Labels:

- als Menge von Hub,Distanz Paaren

Anfrage:

- scannen von zwei Arrays
- nur einige Speicherzugriffe nötig





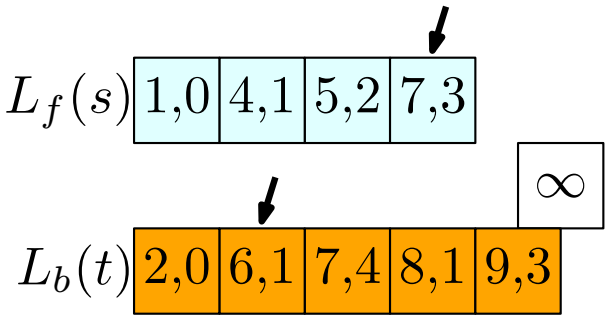
Hub Labels

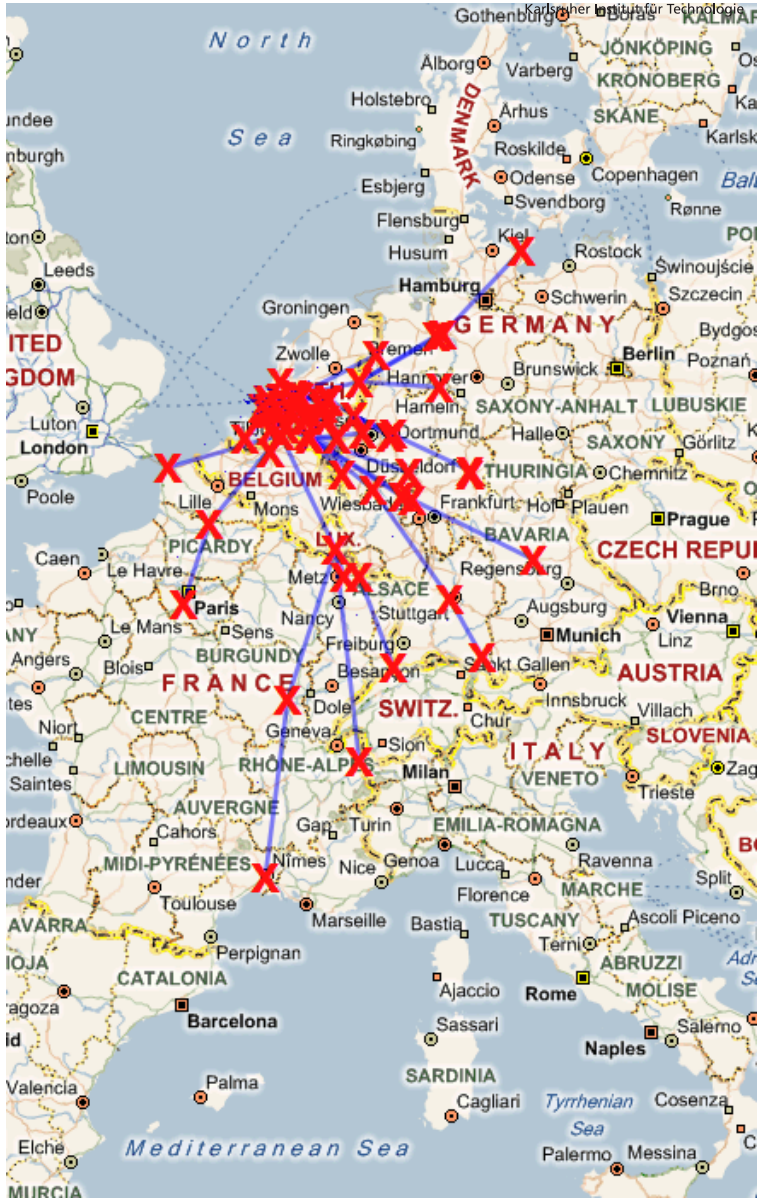
Speichern der Labels:

- als Menge von Hub,Distanz Paaren

Anfrage:

- scannen von zwei Arrays
- nur einige Speicherzugriffe nötig





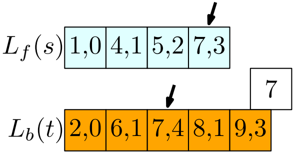
Hub Labels

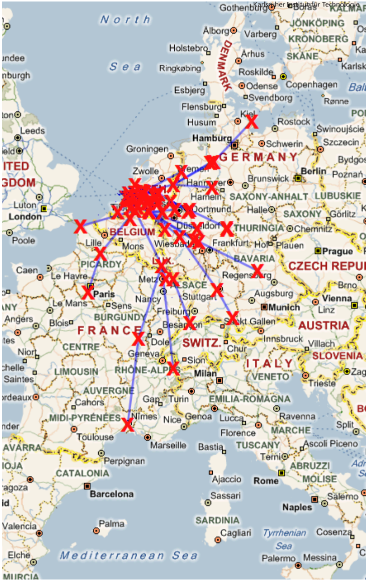
Speichern der Labels:

- als Menge von Hub,Distanz Paaren

Anfrage:

- scannen von zwei Arrays
- nur einige Speicherzugriffe nötig





Institut für Theoretische Informatik
Lehrstuhl Algorithmik

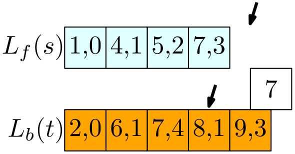
Hub Labels

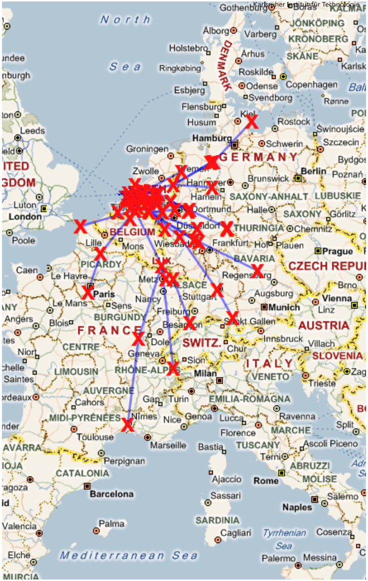
Speichern der Labels:

- als Menge von Hub,Distanz Paaren

Anfrage:

- scannen von zwei Arrays
- nur einige Speicherzugriffe nötig





Hub Labels

Speichern der Labels:

- als Menge von Hub,Distanz Paaren

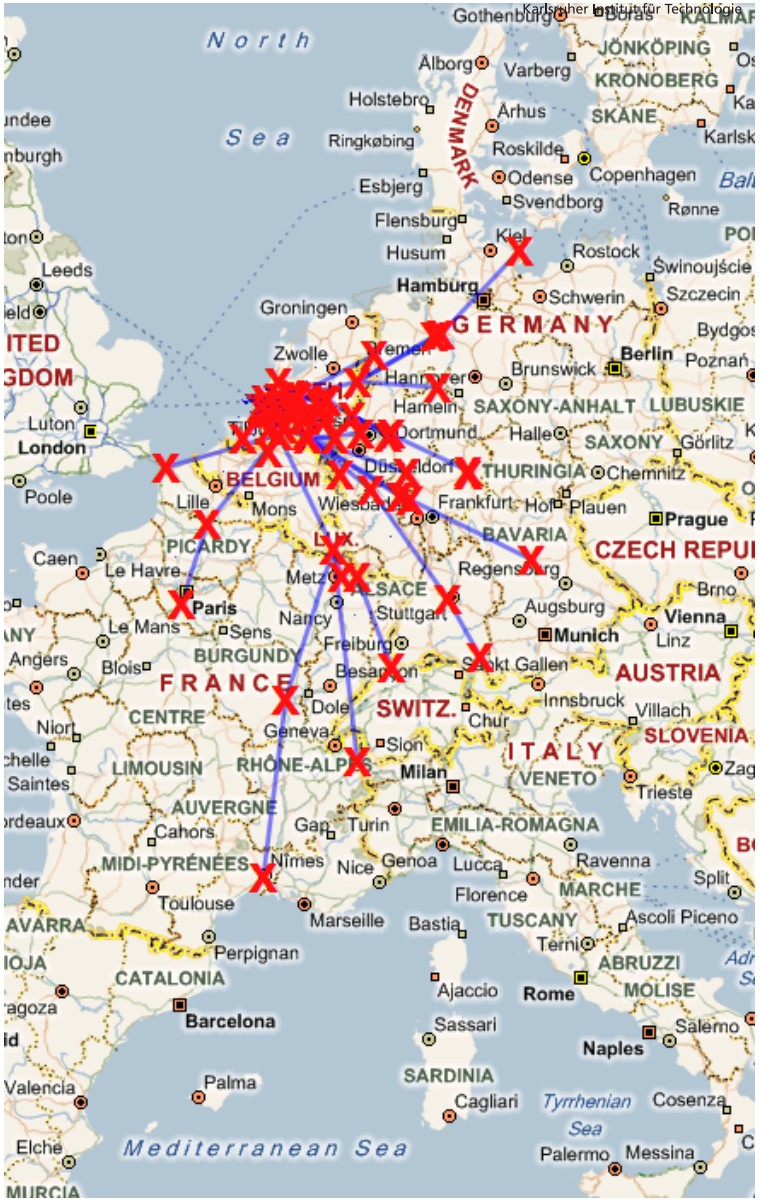
Anfrage:

- scannen von zwei Arrays
- nur einige Speicherzugriffe nötig
- sehr hohe Lokalität

$L_f(s)$ | 1,0 | 4,1 | 5,2 | 7,3 |

7

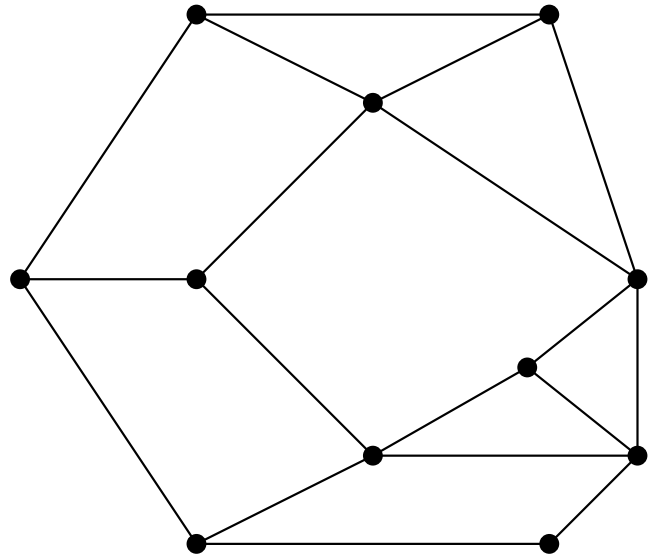
$L_b(t)$ | 2,0 | 6,1 | 7,4 | 8,1 | 9,3 |



Vorberechnung

Idee:

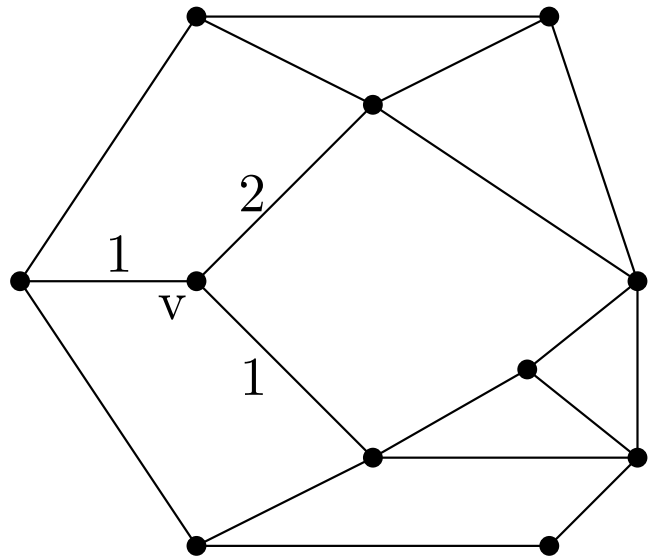
- benutze Knotenordnung



Vorberechnung

Idee:

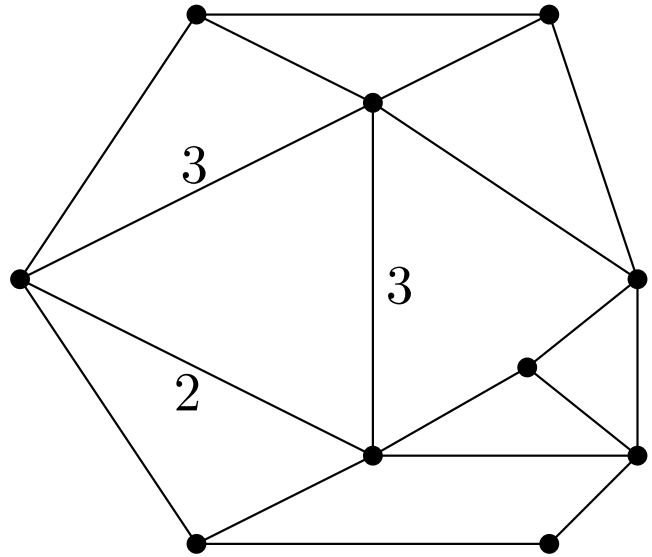
- benutze Knotenordnung
- kontrahiere Knoten v



Vorberechnung

Idee:

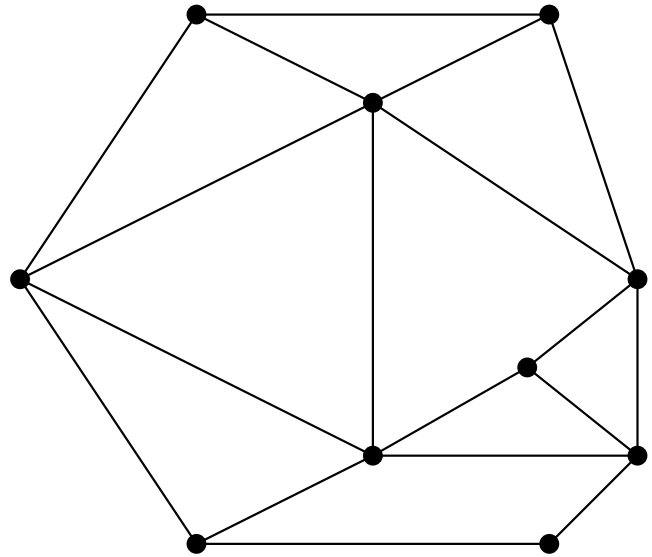
- benutze Knotenordnung
- kontrahiere Knoten v



Vorberechnung

Idee:

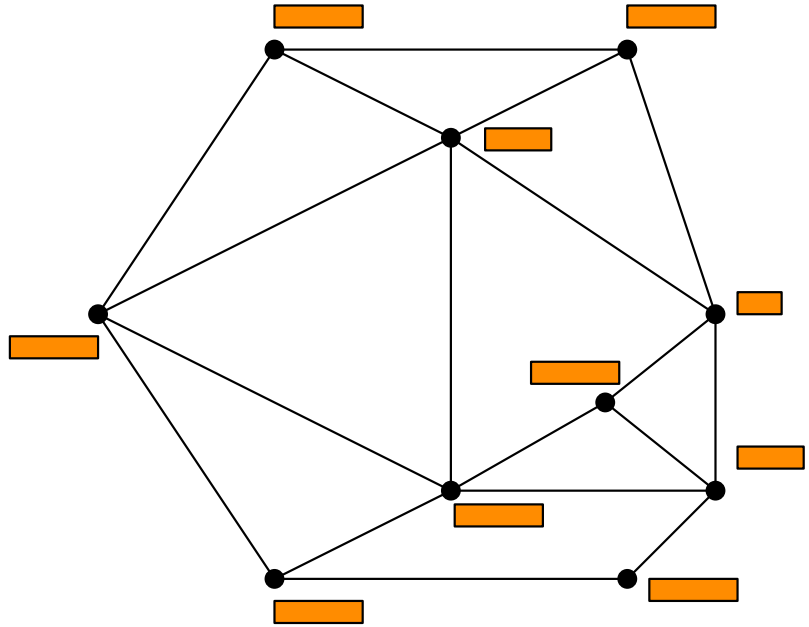
- benutze Knotenordnung
- kontrahiere Knoten v



Vorberechnung

Idee:

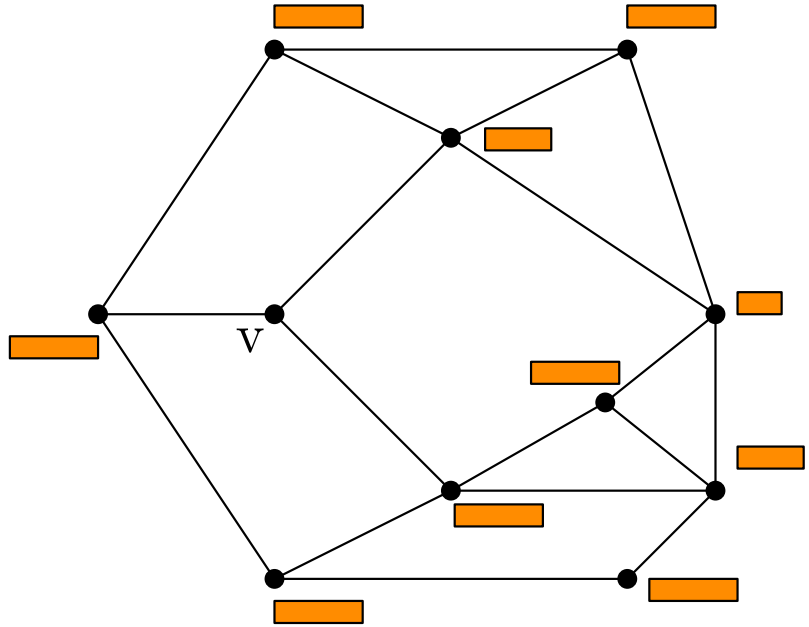
- benutze Knotenordnung
- kontrahiere Knoten v
- berechne Labels rekursiv



Vorberechnung

Idee:

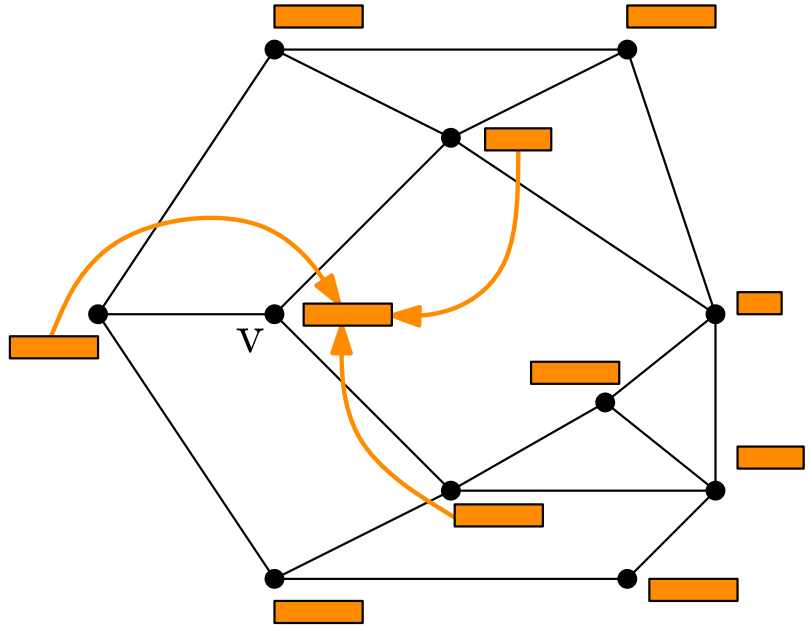
- benutze Knotenordnung
- kontrahiere Knoten v
- berechne Labels rekursiv



Vorberechnung

Idee:

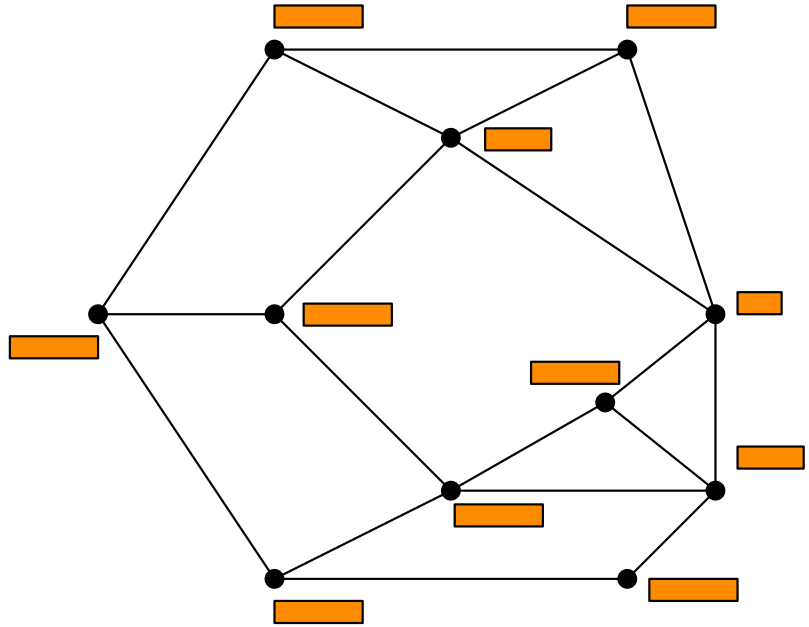
- benutze Knotenordnung
- kontrahiere Knoten v
- berechne Labels rekursiv
- merge Labels der Aufwärts-Nachbarn von v
- dünne Label aus



Vorberechnung

Idee:

- benutze Knotenordnung
- kontrahiere Knoten v
- berechne Labels rekursiv
- merge Labels der Aufwärts-Nachbarn von v
- dünne Label aus



Korrektheit:

- analog zu Korrektheit von CH
- Argumentation über den wichtigsten Knoten auf dem Pfad
- dieser ist im Vorwärtslabel von s und im Rückwärtslabel von t

Mergen von Labels

Generell:

- $L_f(v) = \bigcup_{(v,u) \in G^+} (L_f(u) + (v,u))$
- wenn ein Hub mehrfach im resultierendem Label, behalte nur den mit minimaler Distanz

Mergen von Labels

Generell:

- $L_f(v) = \bigcup_{(v,u) \in G^+} (L_f(u) + (v,u))$
- wenn ein Hub mehrfach im resultierendem Label, behalte nur den mit minimaler Distanz

Ausdünnen:

- manche Knoten im Label haben falschen Distanzwert
- können entfernt werden (analog zu stall-on-demand von CH)
- prunen durch HL-Queries nach Mergen
  bootstrapping

Weitere Beschleunigung

globale Anfragen:

- benachbarte Knoten haben ähnliche Hubs
- der verantwortliche Hub ist wichtig

Weitere Beschleunigung

globale Anfragen:

- benachbarte Knoten haben ähnliche Hubs
- der verantwortliche Hub ist wichtig

idea:

- permutiere hub IDs so, dass wichtige hubs kleine IDs haben

$\Rightarrow$ sind somit am Anfang des Labels

- partitioniere die Eingabe (wieder mal...)
- für jedes Paar Regionen
  - bestimme den am wenigsten wichtigen Hub
  - speichere dessen ID in einer Tabelle (Oracle)

Weitere Beschleunigung

globale Anfragen:

- benachbarte Knoten haben ähnliche Hubs
- der verantwortliche Hub ist wichtig

idea:

- permutiere hub IDs so, dass wichtige hubs kleine IDs haben

$\Rightarrow$ sind somit am Anfang des Labels

- partitioniere die Eingabe (wieder mal...)
- für jedes Paar Regionen
  - bestimme den am wenigsten wichtigen Hub
  - speichere dessen ID in einer Tabelle (Oracle)

query:

- bestimme ID des hubs in der Tabelle
- stoppe Iteration der Labels nach dieser ID

$\Rightarrow$ beschleunigt globale Anfragen

$L(s)$	2	3	6	35	37	102	155	172
$L(t)$	2	6	8	43	45	85		

$Or(s,t)$: 10

Weitere Beschleunigung

globale Anfragen:

- benachbarte Knoten haben ähnliche Hubs
- der verantwortliche Hub ist wichtig

idea:

- permutiere hub IDs so, dass wichtige hubs kleine IDs haben

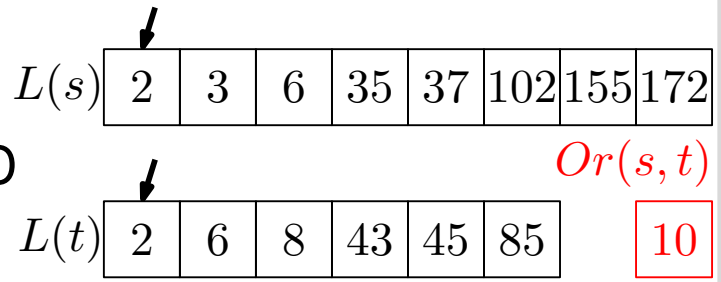
$\Rightarrow$ sind somit am Anfang des Labels

- partitioniere die Eingabe (wieder mal...)
- für jedes Paar Regionen
  - bestimme den am wenigsten wichtigen Hub
  - speichere dessen ID in einer Tabelle (Oracle)

query:

- bestimme ID des hubs in der Tabelle
- stoppe Iteration der Labels nach dieser ID

$\Rightarrow$ beschleunigt globale Anfragen



Weitere Beschleunigung

globale Anfragen:

- benachbarte Knoten haben ähnliche Hubs
- der verantwortliche Hub ist wichtig

idea:

- permutiere hub IDs so, dass wichtige hubs kleine IDs haben

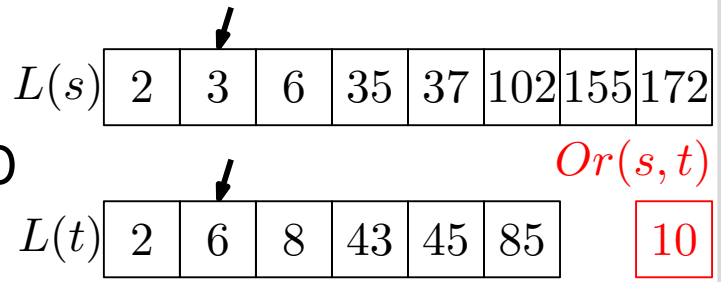
$\Rightarrow$ sind somit am Anfang des Labels

- partitioniere die Eingabe (wieder mal...)
- für jedes Paar Regionen
  - bestimme den am wenigsten wichtigen Hub
  - speichere dessen ID in einer Tabelle (Oracle)

query:

- bestimme ID des hubs in der Tabelle
- stoppe Iteration der Labels nach dieser ID

$\Rightarrow$ beschleunigt globale Anfragen



Weitere Beschleunigung

globale Anfragen:

- benachbarte Knoten haben ähnliche Hubs
- der verantwortliche Hub ist wichtig

idea:

- permutiere hub IDs so, dass wichtige hubs kleine IDs haben

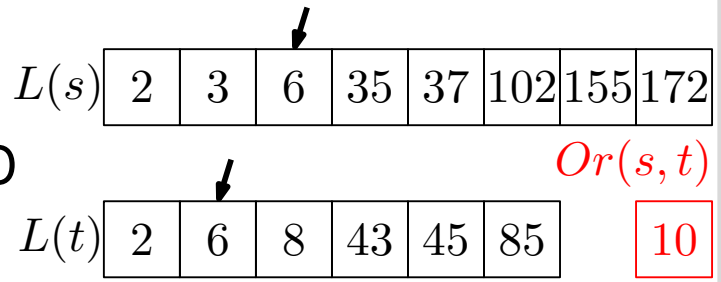
$\Rightarrow$ sind somit am Anfang des Labels

- partitioniere die Eingabe (wieder mal...)
- für jedes Paar Regionen
  - bestimme den am wenigsten wichtigen Hub
  - speichere dessen ID in einer Tabelle (Oracle)

query:

- bestimme ID des hubs in der Tabelle
- stoppe Iteration der Labels nach dieser ID

$\Rightarrow$ beschleunigt globale Anfragen



Weitere Beschleunigung

globale Anfragen:

- benachbarte Knoten haben ähnliche Hubs
- der verantwortliche Hub ist wichtig

idea:

- permutiere hub IDs so, dass wichtige hubs kleine IDs haben

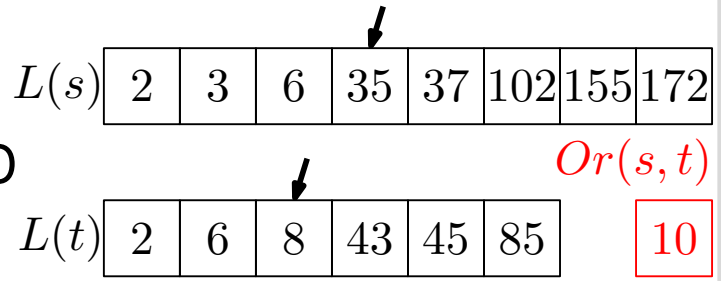
$\Rightarrow$ sind somit am Anfang des Labels

- partitioniere die Eingabe (wieder mal...)
- für jedes Paar Regionen
  - bestimme den am wenigsten wichtigen Hub
  - speichere dessen ID in einer Tabelle (Oracle)

query:

- bestimme ID des hubs in der Tabelle
- stoppe Iteration der Labels nach dieser ID

$\Rightarrow$ beschleunigt globale Anfragen



Ergebnisse

	preprocessing		query
method	time [h:m]	space [GB]	time [μs]
MLD-3	$<$ 0:01	0.4	912
CH	0:02	0.4	96.3
HL-0	0:03	22.5	0.700
HL-15	0:05	18.8	0.556
HL-17	0:25	18.4	0.545
HL-∞	5:43	16.8	0.508
HL-∞ + Oracle	6:12	17.7	0.254
Table Lookup	???	1 208 358.7	0.056

Ergebnisse

	preprocessing		query
method	time [h:m]	space [GB]	time [μs]
MLD-3	$<$ 0:01	0.4	912
CH	0:02	0.4	96.3
HL-0	0:03	22.5	0.700
HL-15	0:05	18.8	0.556
HL-17	0:25	18.4	0.545
HL-∞	5:43	16.8	0.508
HL-∞ + Oracle	6:12	17.7	0.254
Table Lookup	???	1 208 358.7	0.056

HL-x mit 2^x top-down ordnung, ∞ mit Hybrid Ordnung

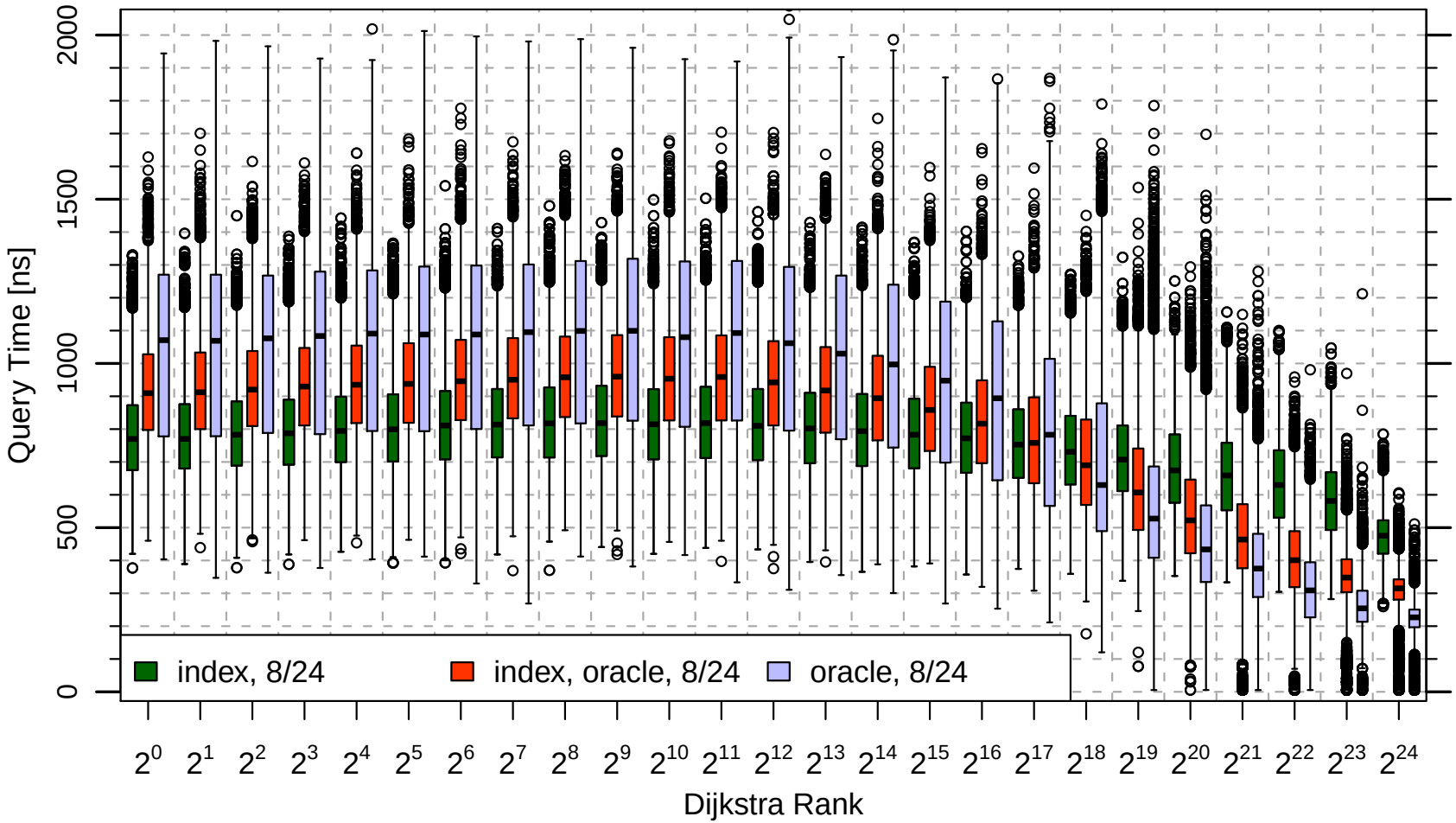
Ergebnisse

	preprocessing		query
method	time [h:m]	space [GB]	time [μs]
MLD-3	$<$ 0:01	0.4	912
CH	0:02	0.4	96.3
HL-0	0:03	22.5	0.700
HL-15	0:05	18.8	0.556
HL-17	0:25	18.4	0.545
HL-∞	5:43	16.8	0.508
HL-∞ + Oracle	6:12	17.7	0.254
Table Lookup	???	1 208 358.7	0.056

HL-x mit 2^x top-down ordnung, ∞ mit Hybrid Ordnung

- HL ist Faktor 100 schneller als CH (Speedup 10 Mio)
- hoher Speicherverbrauch (durch Kompression reduzierbar)
- nur 5 mal langsamer als ein Speicherzugriff

Lokale Queries



Zusammenfassung

- Knotenordnung definiert Labeling
- Beschleunigung gegenüber CH von Faktor mehr als 100
- durch besser Lokalität
- nur 5 mal langsamer als ein Speicherzugriff
- schnellster Algorithmus momentan
- beschleunigt lokale und globale Anfragen
- aber Speicherverbrauch sehr hoch
- wird zu einem späterem Zeitpunkt noch einmal wichtig

Literatur

Contraction Hierarchies:

- Robert Geisberger, Peter Sanders, Dominik Schultes, Christian Vetter
  Exact Routing in Large Road Networks Using Contraction Hierarchies
  In: *Transportation Science*, 2012

HubLabels:

- Ittai Abraham, Daniel Delling, Andrew V. Goldberg, Renato Werneck
  A Hub-Based Labeling Algorithm for Shortest Paths on Road Networks
  In: *Proceedings of the 10th International Symposium on Experimental Algorithms (SEA '11)*, 2011
- Ittai Abraham, Daniel Delling, Andrew V. Goldberg, Renato Werneck
  Hierarchical Hub Labelings for Shortest Paths
  MSR Technical Report, 2012

Nächste Termine

Mittwoch, 22.05.2013 (Julian)

Montag, 27.05.2013 (Thomas)

Mittwoch, 29.05.2013 (Julian)

Montag, 03.06.2013 (Thomas)