KIT
Karlsruher Institut für Technologie

Algorithmen für Routenplanung

9. Vorlesung, Sommersemester 2016

Ben Strasser | 18. Mai 2016

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

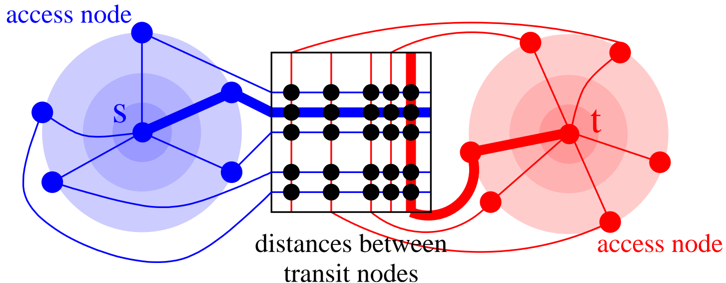
Kürzeste Wege in Straßennetzwerken

Beschleunigungstechniken (Fortsetzung)

- Transit-Node Routing (TNR)
- (G)PHAST

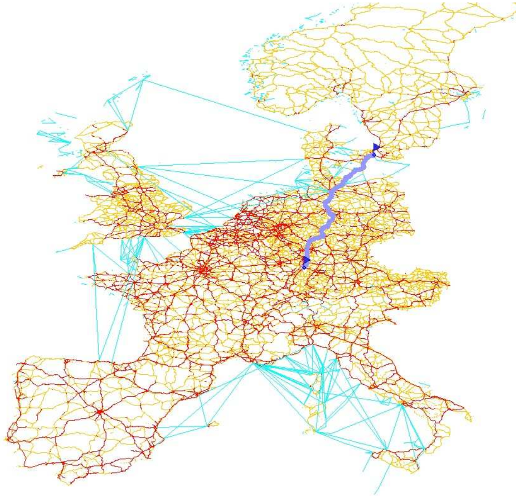
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Transit-Node Routing



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Transit-Node Routing

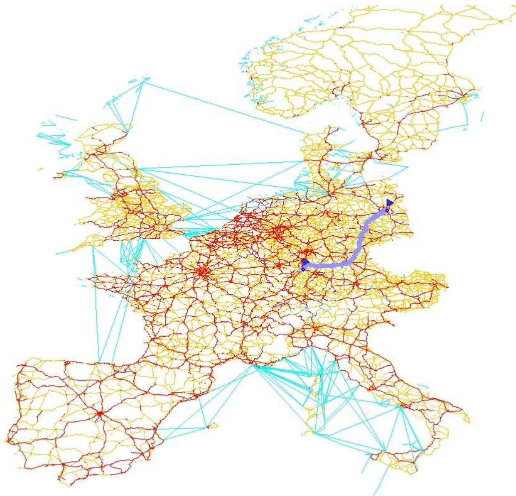


Beobachtung:

- wenn man weit weg fährt, fährt man immer an bestimmten Punkten vorbei
- hier: von Karlsruhe aus, an drei relevanten Stellen

Karlsruhe nach. . .
Kopenhagen

Transit-Node Routing

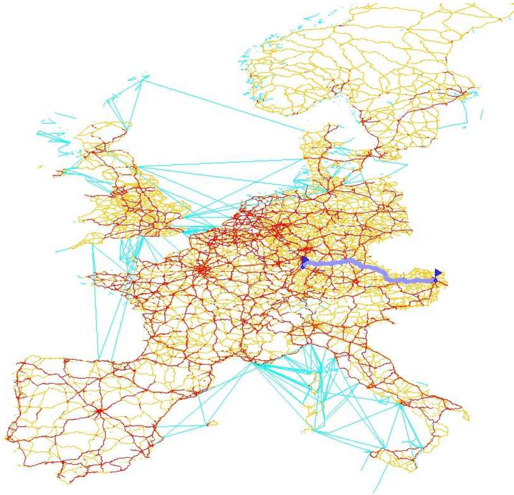


Beobachtung:

- wenn man weit weg fährt, fährt man immer an bestimmten Punkten vorbei
- hier: von Karlsruhe aus, an drei relevanten Stellen

Karlsruhe nach...
Berlin

Transit-Node Routing



Beobachtung:

- wenn man weit weg fährt, fährt man immer an bestimmten Punkten vorbei
- hier: von Karlsruhe aus, an drei relevanten Stellen

Karlsruhe nach...
Wien

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Transit-Node Routing

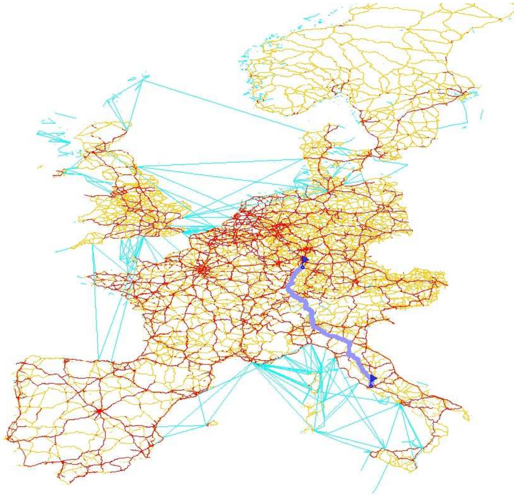
Beobachtung:

- wenn man weit weg fährt, fährt man immer an bestimmten Punkten vorbei
- hier: von Karlsruhe aus, an drei relevanten Stellen

Karlsruhe nach…
München

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Transit-Node Routing



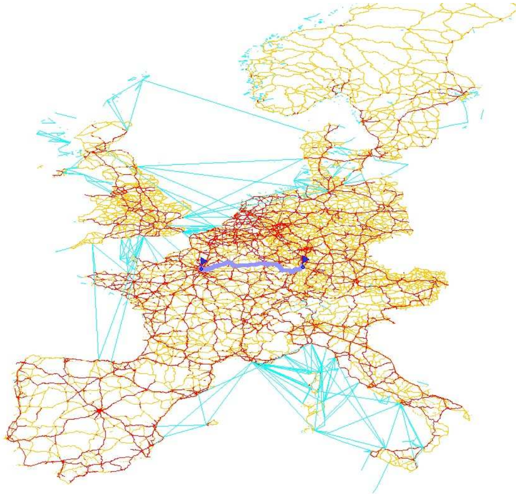
Beobachtung:

- wenn man weit weg fährt, fährt man immer an bestimmten Punkten vorbei
- hier: von Karlsruhe aus, an drei relevanten Stellen

Karlsruhe nach…
Rom

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Transit-Node Routing



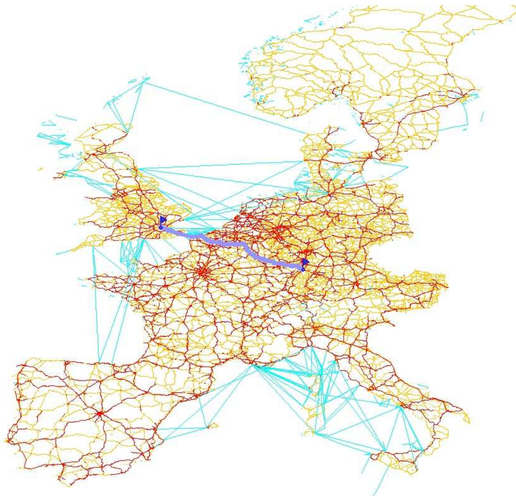
Beobachtung:

- wenn man weit weg fährt, fährt man immer an bestimmten Punkten vorbei
- hier: von Karlsruhe aus, an drei relevanten Stellen

Karlsruhe nach. . .
Paris

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Transit-Node Routing



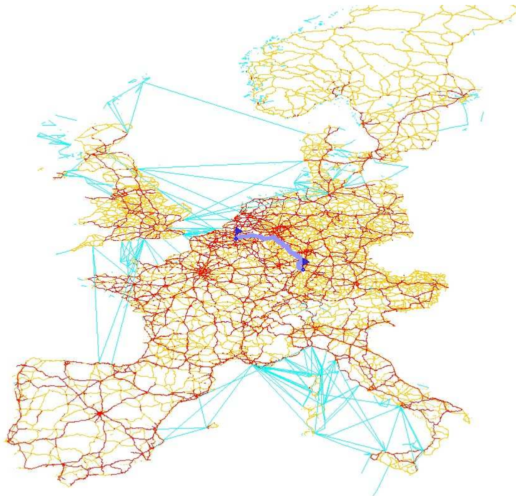
Beobachtung:

- wenn man weit weg fährt, fährt man immer an bestimmten Punkten vorbei
- hier: von Karlsruhe aus, an drei relevanten Stellen

Karlsruhe nach…
London

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Transit-Node Routing

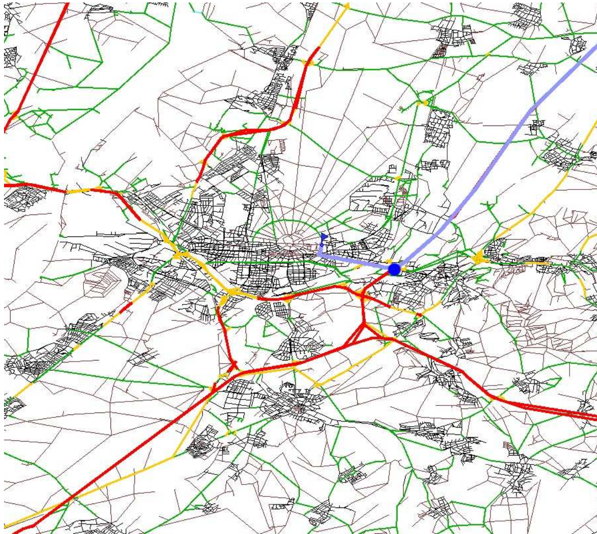


Beobachtung:

- wenn man weit weg fährt, fährt man immer an bestimmten Punkten vorbei
- hier: von Karlsruhe aus, an drei relevanten Stellen

Karlsruhe nach…
Brüssel

Transit-Node Routing

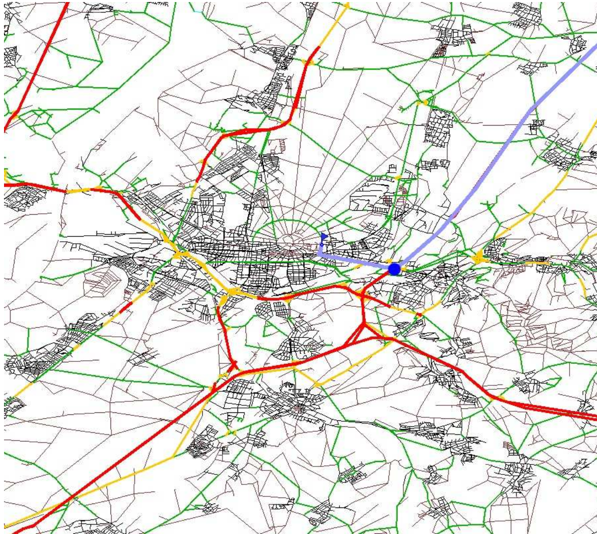


Beobachtung:

- wenn man weit weg fährt, fährt man immer an bestimmten Punkten vorbei
- hier: von Karlsruhe aus, an drei relevanten Stellen

Karlsruhe nach. . .
Kopenhagen

Transit-Node Routing



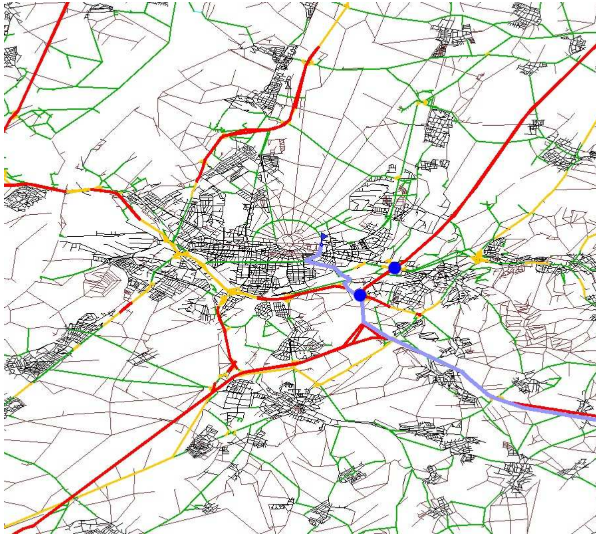
Beobachtung:

- wenn man weit weg fährt, fährt man immer an bestimmten Punkten vorbei
- hier: von Karlsruhe aus, an drei relevanten Stellen

Karlsruhe nach. . .
Berlin

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Transit-Node Routing

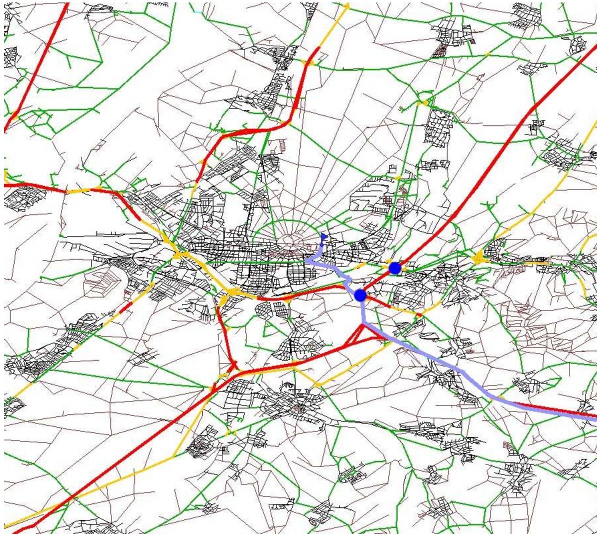


Beobachtung:

- wenn man weit weg fährt, fährt man immer an bestimmten Punkten vorbei
- hier: von Karlsruhe aus, an drei relevanten Stellen

Karlsruhe nach...
Wien

Transit-Node Routing

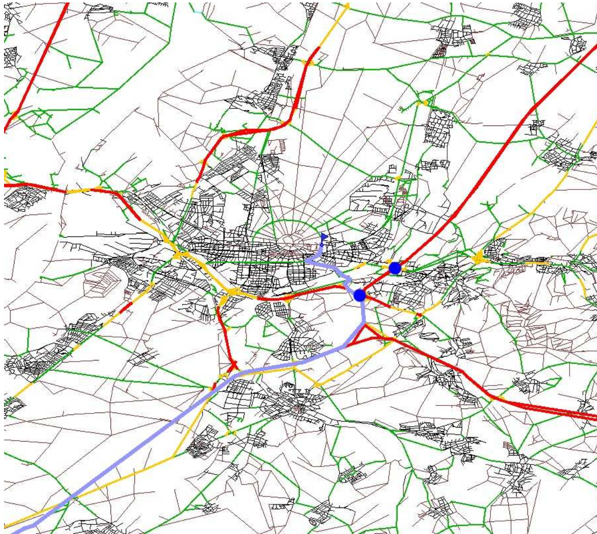


Beobachtung:

- wenn man weit weg fährt, fährt man immer an bestimmten Punkten vorbei
- hier: von Karlsruhe aus, an drei relevanten Stellen

Karlsruhe nach. . .
München

Transit-Node Routing

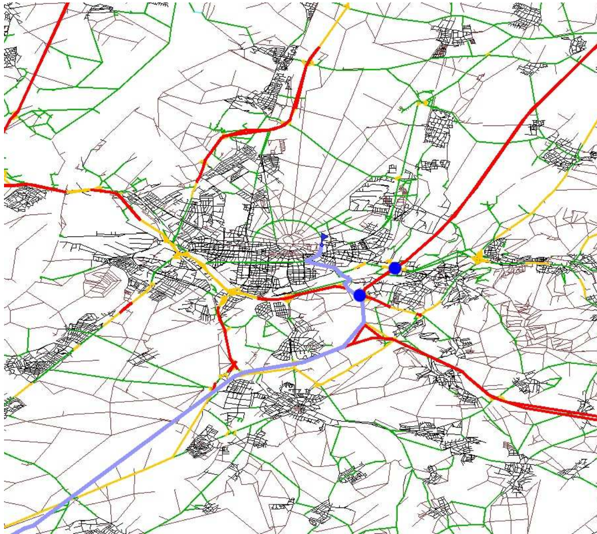


Beobachtung:

- wenn man weit weg fährt, fährt man immer an bestimmten Punkten vorbei
- hier: von Karlsruhe aus, an drei relevanten Stellen

Karlsruhe nach. . .
Rom

Transit-Node Routing



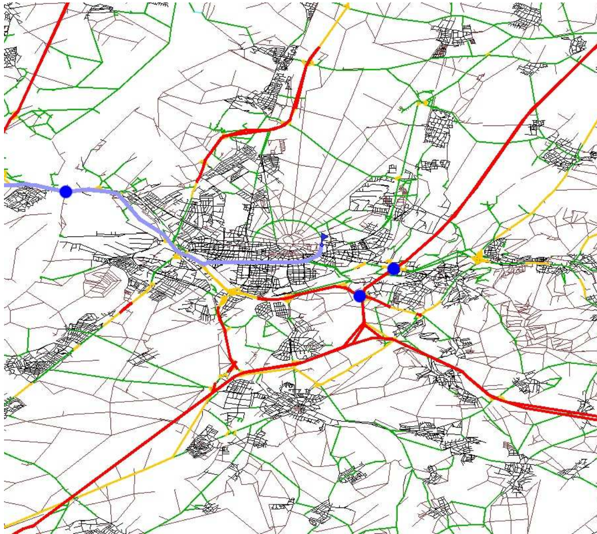
Beobachtung:

- wenn man weit weg fährt, fährt man immer an bestimmten Punkten vorbei
- hier: von Karlsruhe aus, an drei relevanten Stellen

Karlsruhe nach…
Paris

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Transit-Node Routing

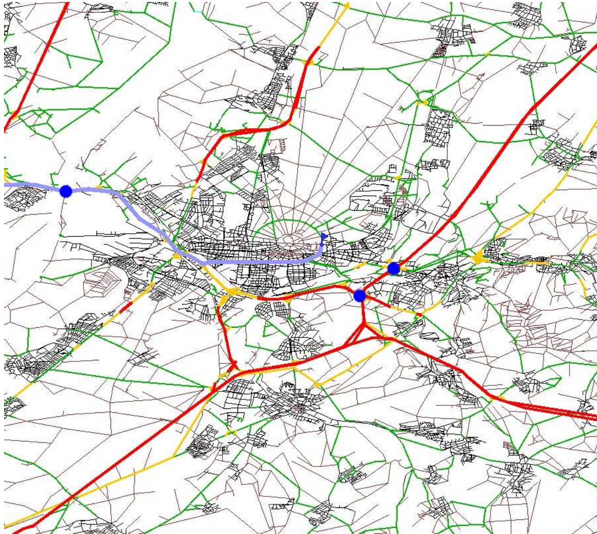


Beobachtung:

- wenn man weit weg fährt, fährt man immer an bestimmten Punkten vorbei
- hier: von Karlsruhe aus, an drei relevanten Stellen

Karlsruhe nach. . .
London

Transit-Node Routing



Beobachtung:

- wenn man weit weg fährt, fährt man immer an bestimmten Punkten vorbei
- hier: von Karlsruhe aus, an drei relevanten Stellen

Karlsruhe nach. . .
Brüssel

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

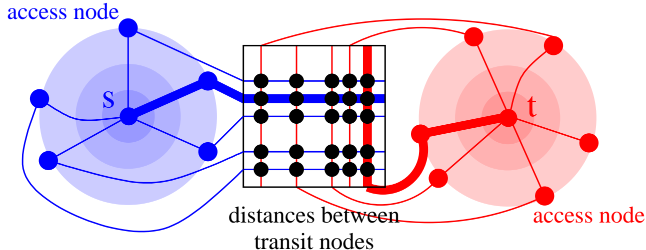
Transit-Node Routing

Idee:

- reduziere Anfragen auf Zugriffe in eine quadratische Tabellen
- identifiziere “wichtige” Knoten
- vollständige Distanztabellen zwischen diesen Knoten

Probleme:

- Speicherverbrauch
- nahe Anfragen



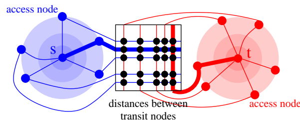
Generelles TNR-Framework

Karlsruher Institut für Technologie

- Wähle Transit-Knoten: $T \subseteq V$
- Bestimme für jeden Knoten v eine Menge von Vorwärts $\overrightarrow{A}(v)$ und Rückwärts $\overleftarrow{A}(v)$ Access-Knoten
- Vorberechnete Distanzen: D_T und d_A

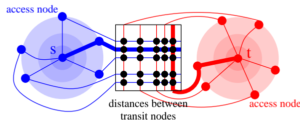
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Generelles TNR-Framework



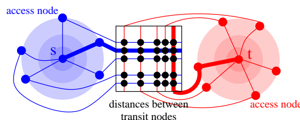
- Wähle Transit-Knoten: $T \subseteq V$
- Bestimme für jeden Knoten v eine Menge von Vorwärts $\overrightarrow{A}(v)$ und Rückwärts $\overleftarrow{A}(v)$ Access-Knoten
- Vorberechnete Distanzen: D_T und d_A

Generelles TNR-Framework



- Wähle Transit-Knoten: $T \subseteq V$
- Bestimme für jeden Knoten v eine Menge von Vorwärts $\overrightarrow{A}(v)$ und Rückwärts $\overleftarrow{A}(v)$ Access-Knoten
- Vorberechnete Distanzen: D_T und d_A
- $\text{dist}(s,t) \stackrel{?}{=} \min_{u \in \overrightarrow{A}(s), v \in \overleftarrow{A}(t)} \{d_A(s,u) + D_T(u,v) + d_A(v,t)\}$

Generelles TNR-Framework



- Wähle Transit-Knoten: $T \subseteq V$
- Bestimme für jeden Knoten v eine Menge von Vorwärts $\overrightarrow{A}(v)$ und Rückwärts $\overleftarrow{A}(v)$ Access-Knoten
- Vorberechnete Distanzen: D_T und d_A
- $\text{dist}(s,t) \stackrel{?}{=} \min_{u \in \overrightarrow{A}(s), v \in \overleftarrow{A}(t)} \{d_A(s,u) + D_T(u,v) + d_A(v,t)\}$

Berechnete Distanz nur für hinreichend weite Anfragen korrekt

- Locality filter: $L : V \times V \to \{\text{true}, \text{false}\}$
- true $\longrightarrow$ Fallback-Routine für lokale Anfragen
- Einseitige Fehler erlaubt

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Zutaten TNR-Verfahren

Also:

- Wie Transit-Knoten bestimmen?
- Wie Access-Knoten und deren Distanz bestimmen?
- Wie Distanztabelle zwischen Transit-Knoten berechnen?
- Welcher Lokalitätsfilter?
- Wie lokale Anfragen berechnen?

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Zutaten TNR-Verfahren

Karlsruher Institut für Technologie

Also:

- Wie Transit-Knoten bestimmen?
- Wie Access-Knoten und deren Distanz bestimmen?
- Wie Distanztabelle zwischen Transit-Knoten berechnen?
- Welcher Lokalitätsfilter?
- Wie lokale Anfragen berechnen?

Ideen?

Zutaten TNR-Verfahren

Karlsruher Institut für Technologie

Also:

- Wie Transit-Knoten bestimmen?
- Wie Access-Knoten und deren Distanz bestimmen?
- Wie Distanztabelle zwischen Transit-Knoten berechnen?
- Welcher Lokalitätsfilter?
- Wie lokale Anfragen berechnen?

Ideen? Verschiedene Ansätze: Grid-based TNR [BFM06],

Hierarchie-basiertes TNR mit geometrischem Lokalitätsfilter [BFM+07, GSSV12], CH-TNR [ALS13]

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

CH-TNR

Karlsruher Institut für Technologie

Transit Node Routing aufbauend auf CH

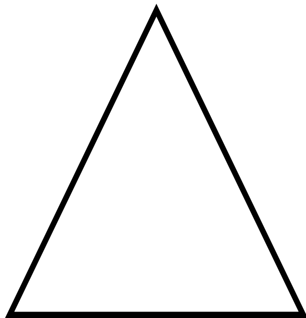
- CH für Vorberechnung und lokale Anfrage
- Top-*k* Knoten sind Transit-Knoten...

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

CH-TNR

Transit Node Routing aufbauend auf CH

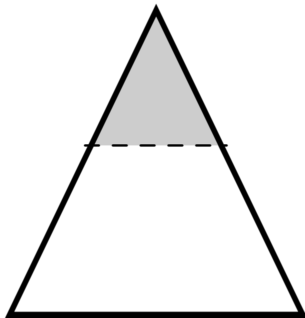
- CH für Vorberechnung und lokale Anfrage
- Top-*k* Knoten sind Transit-Knoten…



CH-TNR

Transit Node Routing aufbauend auf CH

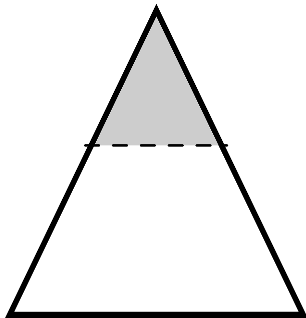
- CH für Vorberechnung und lokale Anfrage
- Top-k Knoten sind Transit-Knoten. . .



CH-TNR

Transit Node Routing aufbauend auf CH

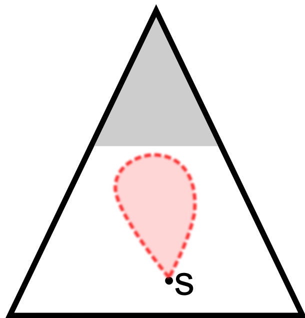
- CH für Vorberechnung und lokale Anfrage
- Top-k Knoten sind Transit-Knoten…
- …und damit die Access-Nodes berechnen



CH-TNR

Transit Node Routing aufbauend auf CH

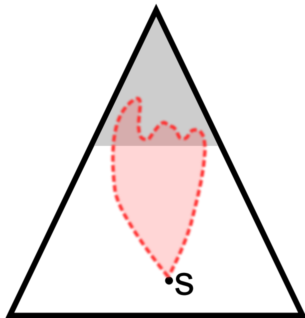
- CH für Vorberechnung und lokale Anfrage
- Top-k Knoten sind Transit-Knoten…
- …und damit die Access-Nodes berechnen



CH-TNR

Transit Node Routing aufbauend auf CH

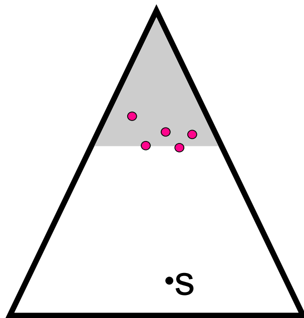
- CH für Vorberechnung und lokale Anfrage
- Top-k Knoten sind Transit-Knoten…
- …und damit die Access-Nodes berechnen



CH-TNR

Transit Node Routing aufbauend auf CH

- CH für Vorberechnung und lokale Anfrage
- Top-k Knoten sind Transit-Knoten…
- …und damit die Access-Nodes berechnen

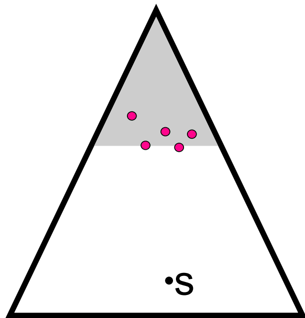


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

CH-TNR

Transit Node Routing aufbauend auf CH

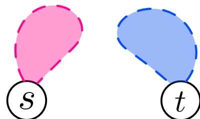
- CH für Vorberechnung und lokale Anfrage
- Top-*k* Knoten sind Transit-Knoten…
- …und damit die Access-Nodes berechnen
- Lokalitätsfilter!?

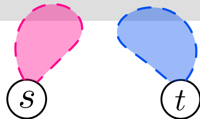


CH-basiertes TNR

Eigenschaften einer lokalen Anfrage

- Betrachte den höchsten Knoten *m* auf einem kürzesten CH hoch-runter *st*-Pfad

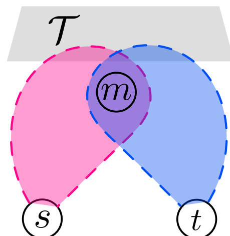


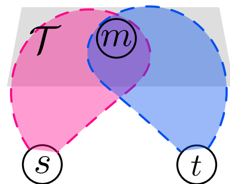


CH-basiertes TNR

Eigenschaften einer lokalen Anfrage

- Betrachte den höchsten Knoten *m* auf einem kürzesten CH hoch-runter *st*-Pfad

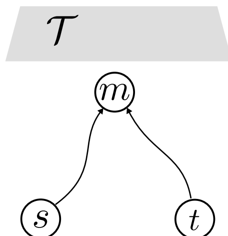


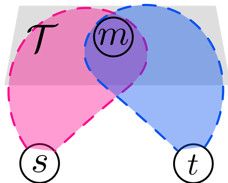


CH-basiertes TNR

Eigenschaften einer lokalen Anfrage

- Betrachte den höchsten Knoten *m* auf einem kürzesten CH hoch-runter *st*-Pfad

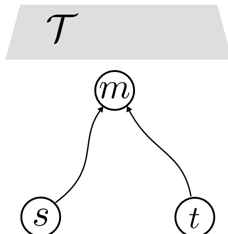


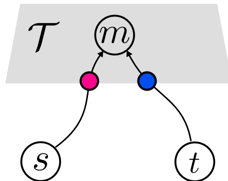


CH-basiertes TNR

Eigenschaften einer lokalen Anfrage

- Betrachte den höchsten Knoten m auf einem kürzesten CH hoch-runter st-Pfad
- $m \notin \mathcal{T} \iff$ lokale Anfrage

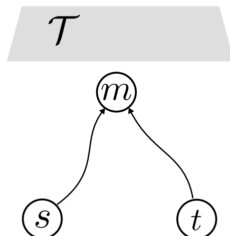




CH-basiertes TNR

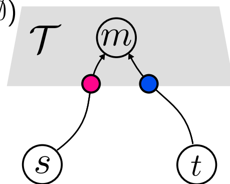
Eigenschaften einer lokalen Anfrage

- Betrachte den höchsten Knoten m auf einem kürzesten CH hoch-runter st-Pfad
- $m \notin \mathcal{T} \iff$ lokale Anfrage

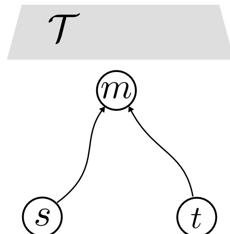


Suchraum-basierter Lokalitätsfilter

- Speichere Suchraum unterhalb der Transit-Knoten $S : V \to V \setminus \mathcal{T}$ explizit
- Fällt bei der Access-Knoten-Berechnung als Beiprodukt ab
- Während der Anfrage: $\mathcal{L} = (S(s) \cap S(t) \neq \emptyset)$
- Braucht viel Speicher!



CH-basiertes TNR

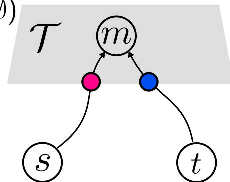


Eigenschaften einer lokalen Anfrage

- Betrachte den höchsten Knoten m auf einem kürzesten CH hoch-runter st-Pfad
- $m \notin \mathcal{T} \iff$ lokale Anfrage

Suchraum-basierter Lokalitätsfilter

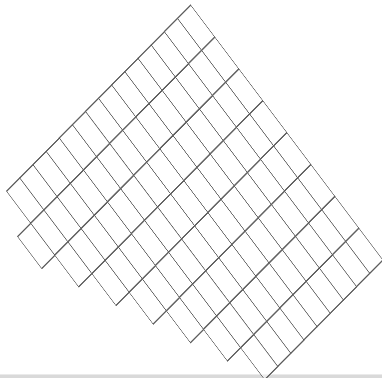
- Speichere Suchraum unterhalb der Transit-Knoten $S : V \to V \setminus \mathcal{T}$ explizit
- Fällt bei der Access-Knoten-Berechnung als Beiprodukt ab
- Während der Anfrage: $\mathcal{L} = (S(s) \cap S(t) \neq \emptyset)$
- Braucht viel Speicher!
- Einseitiger Fehler erlaubt



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Lokalitätsfilter

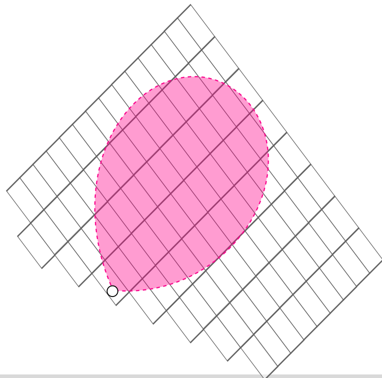
- Paritioniere Graph in Regionen
- Überapproximation des Suchraum mittels berührter Regionen.
- Wenn x im Suchraum $S(s)$ ist dann ist die Region $R(x)$ im approximierten Suchraum $S'(s)$



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

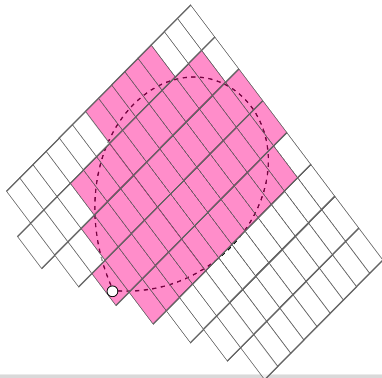
Lokalitätsfilter

- Paritioniere Graph in Regionen
- Überapproximation des Suchraum mittels berührter Regionen.
- Wenn x im Suchraum $S(s)$ ist dann ist die Region $R(x)$ im approximierten Suchraum $S'(s)$



Lokalitätsfilter

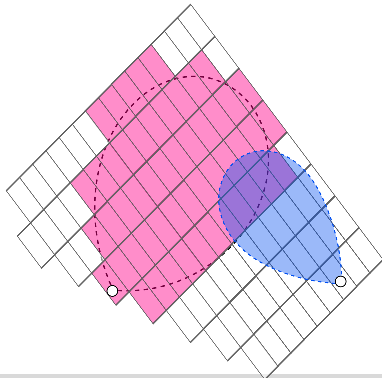
- Paritioniere Graph in Regionen
- Überapproximation des Suchraum mittels berührter Regionen.
- Wenn x im Suchraum $S(s)$ ist dann ist die Region $R(x)$ im approximierten Suchraum $S'(s)$



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

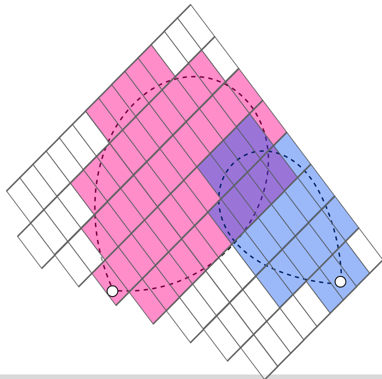
Lokalitätsfilter

- Paritioniere Graph in Regionen
- Überapproximation des Suchraum mittels berührter Regionen.
- Wenn x im Suchraum $S(s)$ ist dann ist die Region $R(x)$ im approximierten Suchraum $S'(s)$
- $m \in S(s) \cap S(t) \implies R(m) \in S'(s) \cap S'(t)$



Lokalitätsfilter

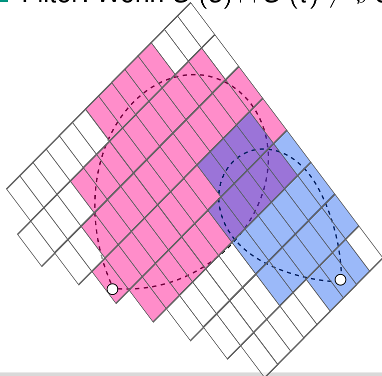
- Paritioniere Graph in Regionen
- Überapproximation des Suchraum mittels berührter Regionen.
- Wenn x im Suchraum $S(s)$ ist dann ist die Region $R(x)$ im approximierten Suchraum $S'(s)$
- $m \in S(s) \cap S(t) \implies R(m) \in S'(s) \cap S'(t)$



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

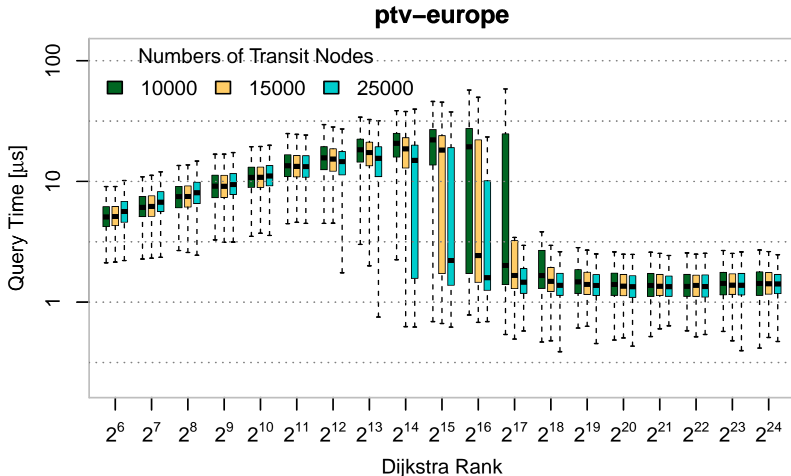
Lokalitätsfilter

- Paritioniere Graph in Regionen
- Überapproximation des Suchraum mittels berührter Regionen.
- Wenn x im Suchraum $S(s)$ ist dann ist die Region $R(x)$ im approximierten Suchraum $S'(s)$
- $m \in S(s) \cap S(t) \implies R(m) \in S'(s) \cap S'(t)$
- Filter: Wenn $S'(s) \cap S'(t) \neq \emptyset$ dann lokal



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Dijkstra Rank



Frage: Welche durchschnittliche Laufzeit ergibt sich?

Zusammenfassung

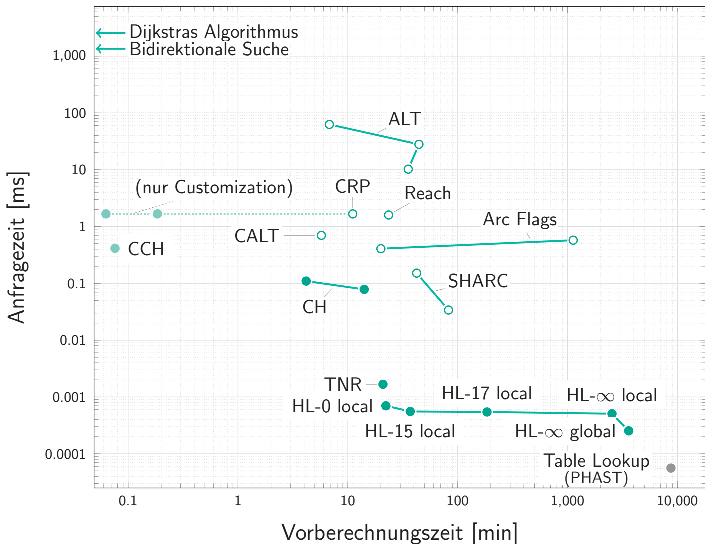
Karlsruher Institut für Technologie

Transit-Node Routing

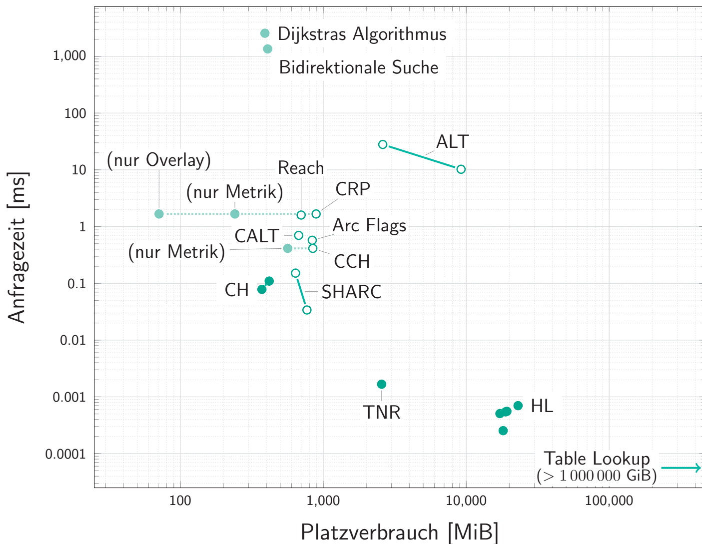
- ersetzt Suche (fast) komplett durch Table-Lookups
- 4 Zutaten:
  - Transit-Nodes
  - Distanztabelle
  - Access-Nodes
  - Locality-Filter

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Übersicht bisherige Techniken

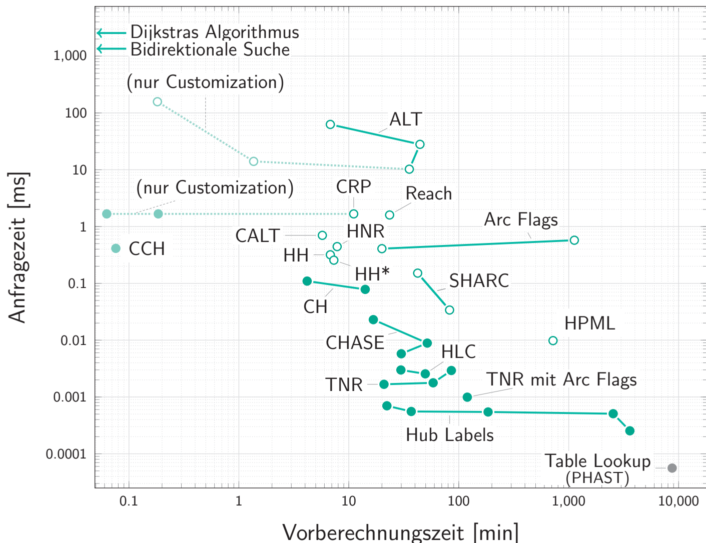


Übersicht bisherige Techniken



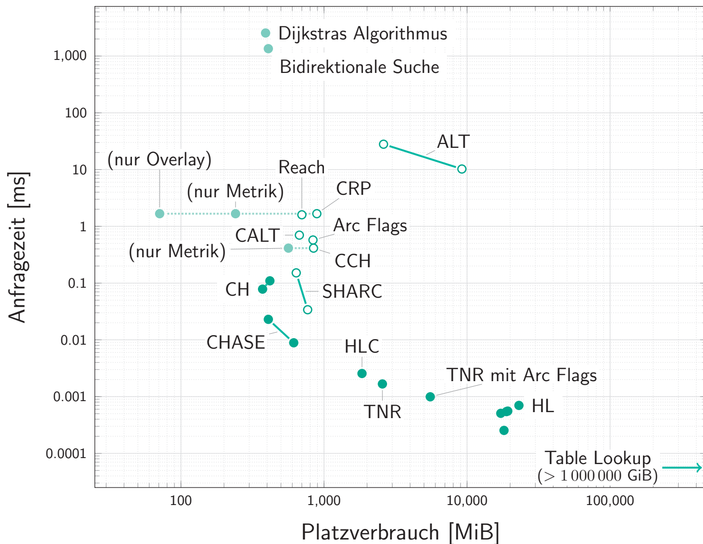
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

“Komplett”übersicht One-to-One



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

“Komplett”übersicht One-to-One



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

PHAST

Andere Problemstellung

Karlsruher Institut für Technologie

bisher

- Finde einen *st*-Pfad

jetzt

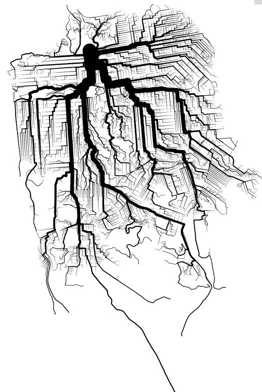
- one-to-all: Von *s* zu allen Knoten
- all-to-all: Berechne alle kürzesten Wege

Kürzeste-Wege Bäume

Karlsruher Institut für Technologie

Anfrage:

- gegeben ein nicht negativ gewichteter gerichteter Graph und Knoten s
- berechne Distanzen von s zu *allen* anderen



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Kürzeste-Wege Bäume

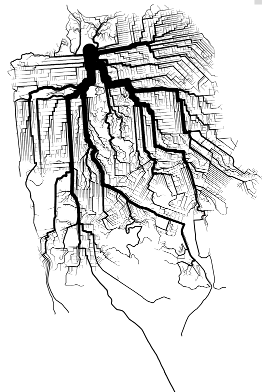
Karlsruher Institut für Technologie

Anfrage:

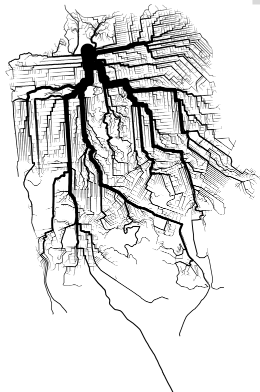
- gegeben ein nicht negativ gewichteter gerichteter Graph und Knoten *s*
- berechne Distanzen von *s* zu *allen* anderen

Lösung:

- Dijkstra [Dij59]



Kürzeste-Wege Bäume



Anfrage:

- gegeben ein nicht negativ gewichteter gerichteter Graph und Knoten s
- berechne Distanzen von s zu *allen* anderen

Lösung:

- Dijkstra [Dij59]

Fakten:

- $O(m + n \log n)$ mit Fibonacci Heaps [FT87]
- linear (mit kleiner Konstanten) in Praxis [Gol01]
- Ausnutzung von moderner Hardware schwierig

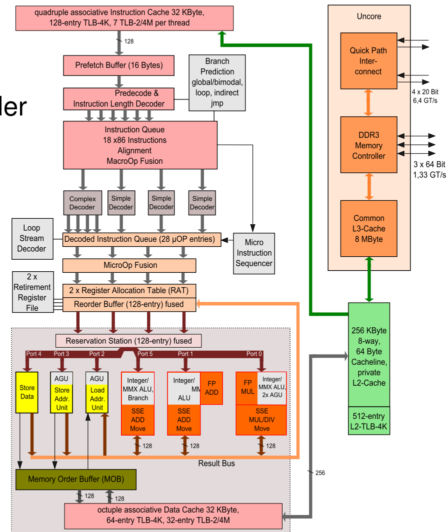
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Moderne CPU Architektur

Karlsruher Institut für Technologie

Einige Fakten:

- viele Kerne
- mehr Kerne als Speicherkontroller
- Hyperthreading
- Multi-Sockel System
- steile Speicherhierarchie
- Cache coherency



GT/s: gigatransfers per second

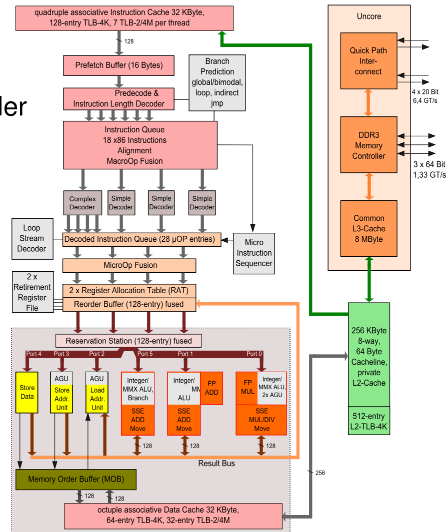
Moderne CPU Architektur

Einige Fakten:

- viele Kerne
- mehr Kerne als Speicherkontroller
- Hyperthreading
- Multi-Sockel System
- steile Speicherhierarchie
- Cache coherency

Hauptherausforderungen:

- Parallelisierung
- Speicherzugriff

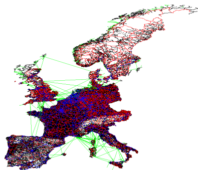


Ausnutzen von Moderner Hardware

Daten Lokalität

- Eingabe: West Europa
- 18M Knoten, 23M Strassen

  Dijkstra: $\approx$ 3.0 s $\Rightarrow$ nicht real-time



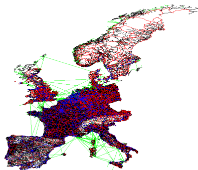
Core-i7 workstation (2.66 GHz)

Ausnutzen von Moderner Hardware

Daten Lokalität

- Eingabe: West Europa
- 18M Knoten, 23M Strassen

Dijkstra:	$\approx$ 3.0 s	$\Rightarrow$ nicht real-time
$n + m$ clock cycles:	$\approx$ 15 ms	$\Rightarrow$ viel schneller

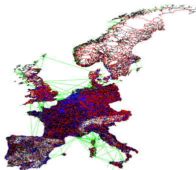


Core-i7 workstation (2.66 GHz)

Ausnutzen von Moderner Hardware

Karlsruher Institut für Technologie

Daten Lokalität



- Eingabe: West Europa
- 18M Knoten, 23M Strassen

Dijkstra:	$\approx$ 3.0 s	$\Rightarrow$ nicht real-time
$n + m$ clock cycles:	$\approx$ 15 ms	$\Rightarrow$ viel schneller
BFS:	$\approx$ 2.0 s	

- Verlangsamung kommt nicht durch Priorityqueue allein

Core-i7 workstation (2.66 GHz)

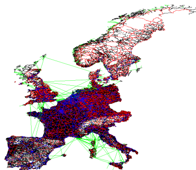
Ausnutzen von Moderner Hardware

Karlsruher Institut für Technologie

Daten Lokalität

- Eingabe: West Europa
- 18M Knoten, 23M Strassen

Dijkstra:	$\approx$ 3.0 s	$\Rightarrow$ nicht real-time
$n + m$ clock cycles:	$\approx$ 15 ms	$\Rightarrow$ viel schneller
BFS:	$\approx$ 2.0 s	



- Verlangsamung kommt nicht durch Priorityqueue allein

Core-i7 workstation (2.66 GHz)

Parallelisierung:

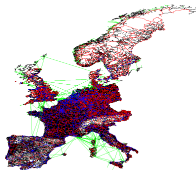
- Spekulation
- Δ-stepping [MS03]
- mehr Operationen als Dijkstra
- keine grosse Beschleunigung auf dünnen Graphen

Ausnutzen von Moderner Hardware

Daten Lokalität

- Eingabe: West Europa
- 18M Knoten, 23M Strassen

Dijkstra:	$\approx$ 3.0 s	$\Rightarrow$ nicht real-time
$n + m$ clock cycles:	$\approx$ 15 ms	$\Rightarrow$ viel schneller
BFS:	$\approx$ 2.0 s	



- Verlangsamung kommt nicht durch Priorityqueue allein

Core-i7 workstation (2.66 GHz)

Parallelisierung:

- Spekulation
- Δ-stepping [MS03]
- mehr Operationen als Dijkstra
- keine grosse Beschleunigung auf dünnen Graphen
- Berechnen von mehreren Bäumen ist einfach

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Ansatz 1

Karlsruher Institut für Technologie

Idee:

- Umordnen der Knoten im Graphen

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Ansatz 1

KIT Karlsruher Institut für Technologie

Idee:

- Umordnen der Knoten im Graphen

		time per tree [ms]		
algorithm	details	random	input	DFS
Dijkstra	binary heap	11159	5859	5180
	Dial	7767	3538	2908
	smart queue	7991	3556	2826
BFS	—	6060	2445	2068

(**Achtung:** One-to-All & Älterer Rechner)

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Ansatz 1

Karlsruher Institut für Technologie

Idee:

- Umordnen der Knoten im Graphen

		time per tree [ms]		
algorithm	details	random	input	DFS
Dijkstra	binary heap	11159	5859	5180
	Dial	7767	3538	2908
	smart queue	7991	3556	2826
BFS	—	6060	2445	2068

(**Achtung:** One-to-All & Älterer Rechner)

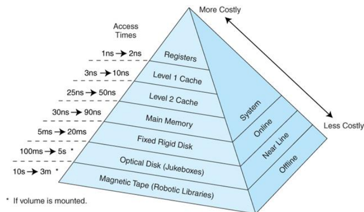
$\Rightarrow$ keine grosse Beschleunigung

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Ansatz 2

Dijkstra's Algorithmus:

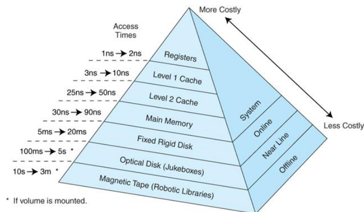
- moderne Hardware nicht ausgenutzt
- Hauptprobleme:
  - Daten Lokalität
  - Parallelisierung



Ansatz 2

Dijkstra's Algorithmus:

- moderne Hardware nicht ausgenutzt
- Hauptprobleme:
  - Daten Lokalität
  - Parallelisierung



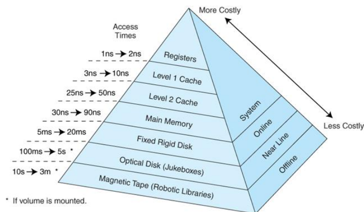
Fragen:

- hilft Vorberechnung?
- wie?
- Ansatzpunkt?

Ansatz 2

Dijkstra's Algorithmus:

- moderne Hardware nicht ausgenutzt
- Hauptprobleme:
  - Daten Lokalität
  - Parallelisierung



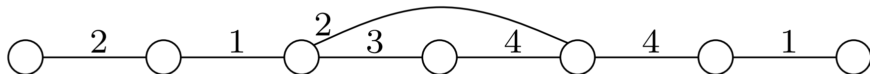
Fragen:

- hilft Vorberechnung?
- wie?
- Ansatzpunkt?

PHAST: Hardware-Accelerated Shortest path Trees

Contraction Hierarchies

preprocessing:

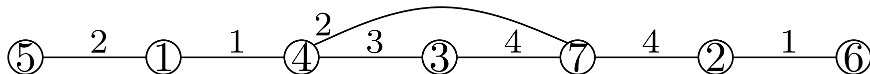


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Contraction Hierarchies

preprocessing:

- ordne Knoten nach Wichtigkeit

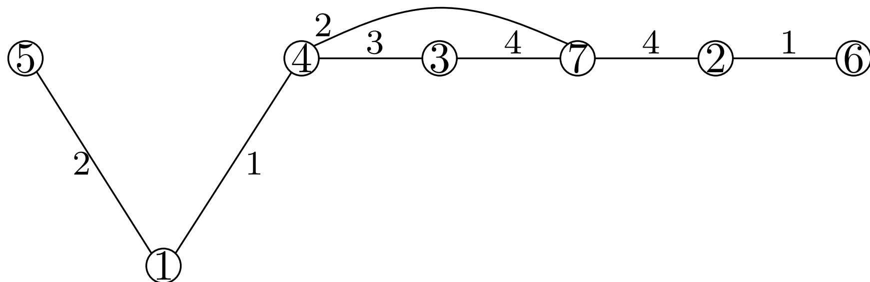


Contraction Hierarchies

Karlsruher Institut für Technologie

preprocessing:

- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu

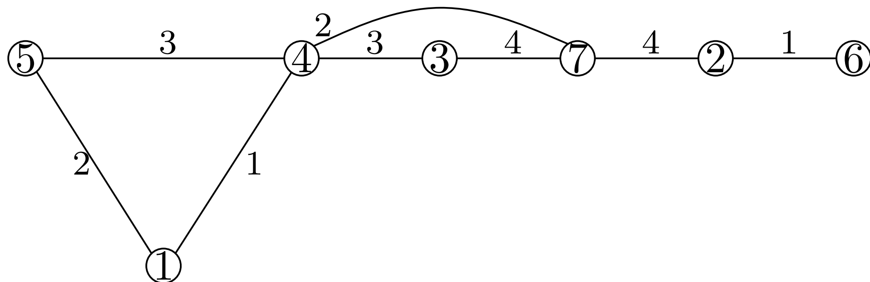


Contraction Hierarchies

Karlsruher Institut für Technologie

preprocessing:

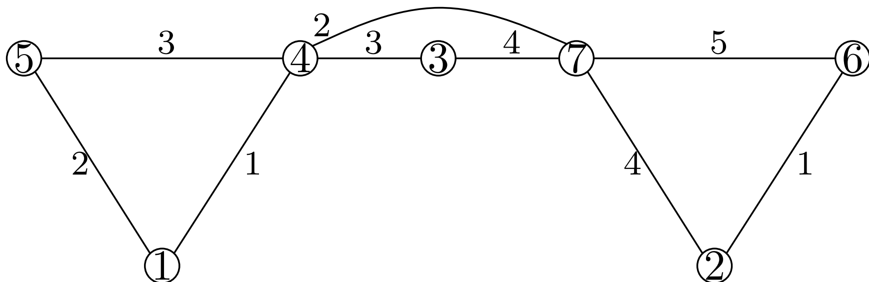
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu



Contraction Hierarchies

preprocessing:

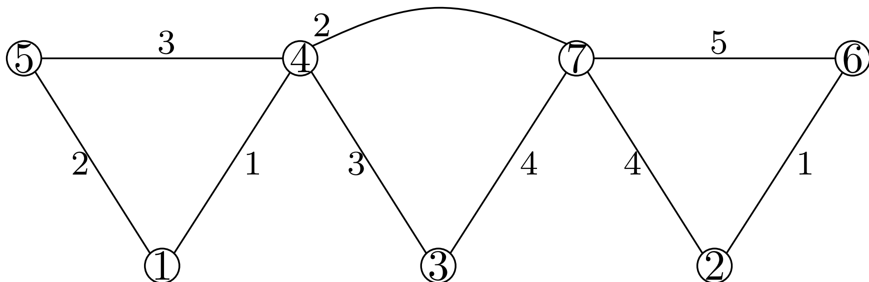
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu



Contraction Hierarchies

preprocessing:

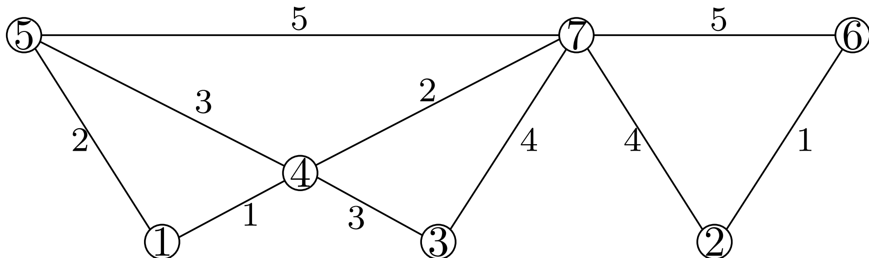
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu



Contraction Hierarchies

preprocessing:

- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu

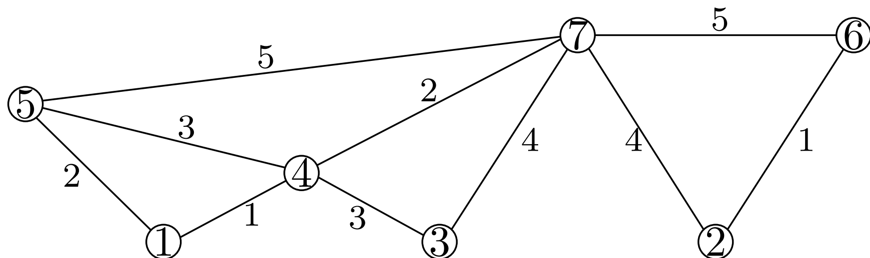


Contraction Hierarchies

Karlsruher Institut für Technologie

preprocessing:

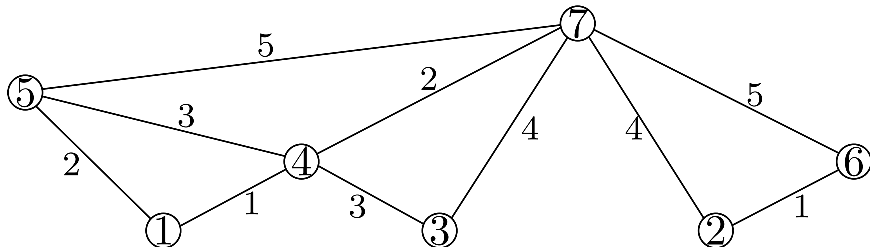
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu



Contraction Hierarchies

preprocessing:

- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu

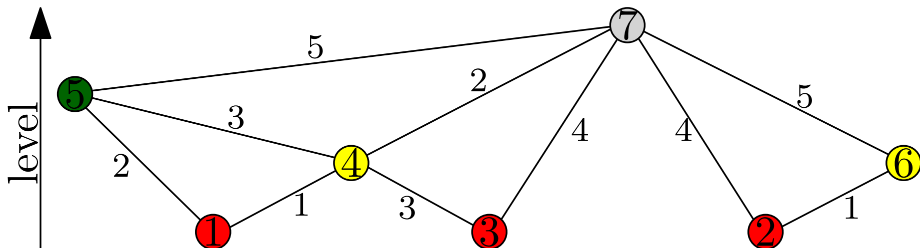


Contraction Hierarchies

Karlsruher Institut für Technologie

preprocessing:

- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu
- Levelzuordnung (ca. 150 in Strassennetzwerken)

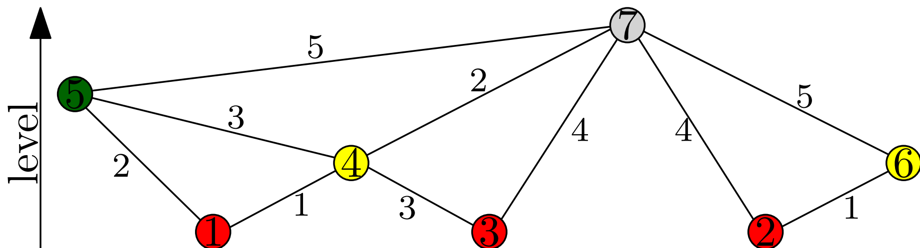


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Contraction Hierarchies

Punkt-zu-Punkt Anfragen

- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten
- besucht nur 500 Knoten

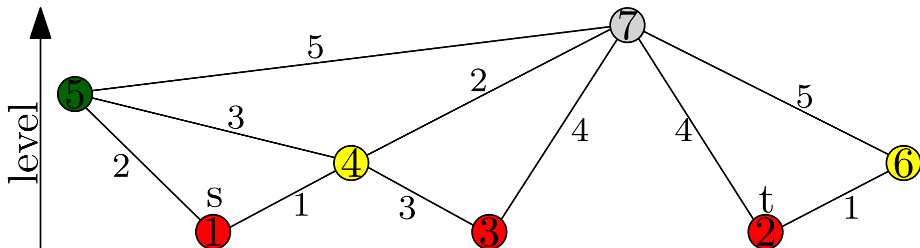


Contraction Hierarchies

Karlsruher Institut für Technologie

Punkt-zu-Punkt Anfragen

- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten
- besucht nur 500 Knoten

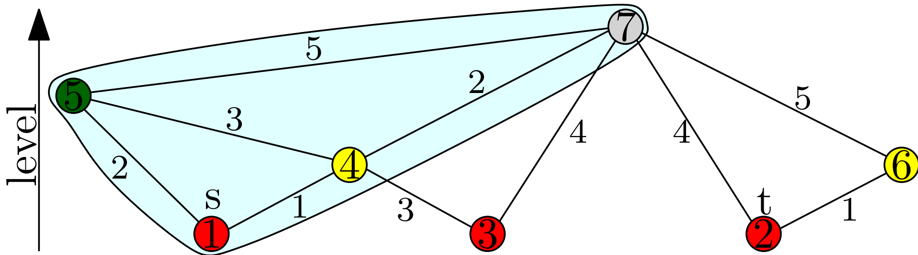


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Contraction Hierarchies

Punkt-zu-Punkt Anfragen

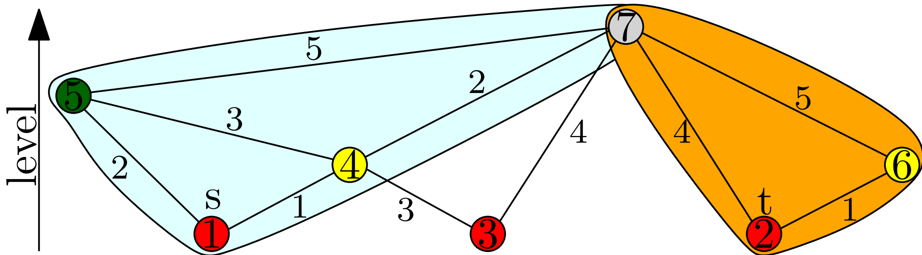
- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten
- besucht nur 500 Knoten



Contraction Hierarchies

Punkt-zu-Punkt Anfragen

- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten
- besucht nur 500 Knoten

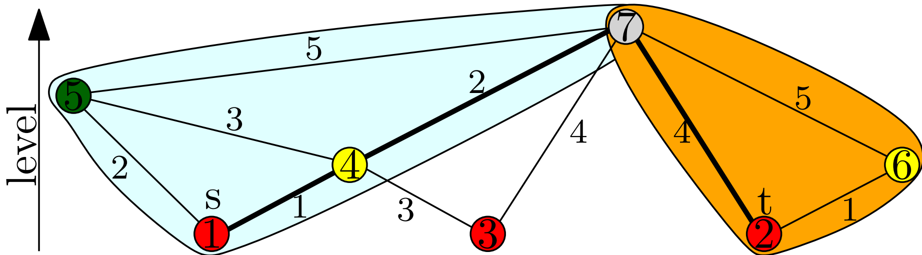


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Contraction Hierarchies

Punkt-zu-Punkt Anfragen

- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten
- besucht nur 500 Knoten



Contraction Hierarchies

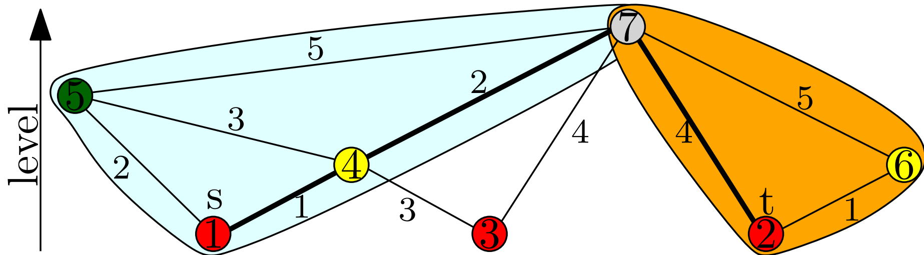
Karlsruher Institut für Technologie

Punkt-zu-Punkt Anfragen

- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten
- besucht nur 500 Knoten

Korrektheit:

- es gibt einen wichtigsten Knoten auf dem Pfad
- dieser wird von Vorwärts- und Rückwärtssuche gescannt

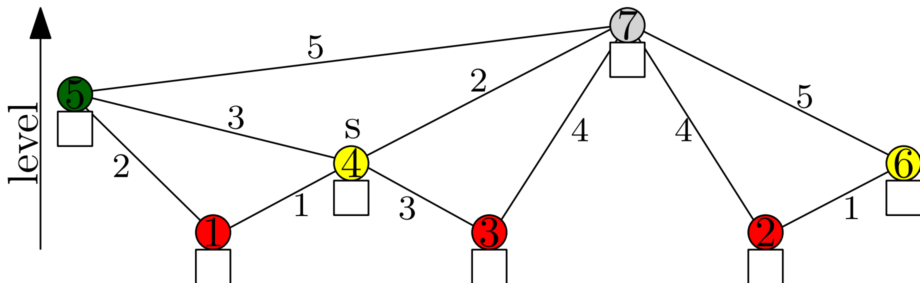


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Neuer Anfragealgorithmus

Karlsruher Institut für Technologie

one-to-all Suche von *s*:

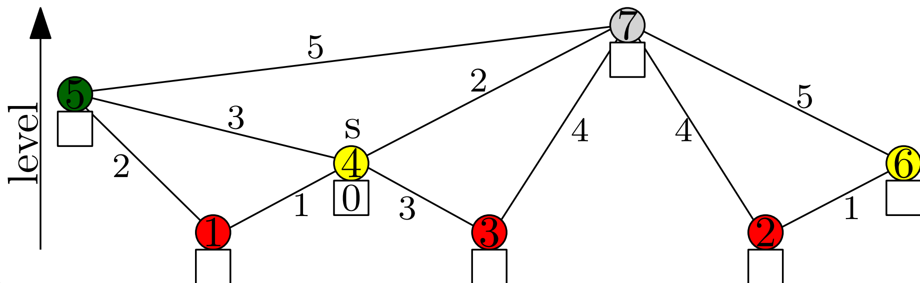


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Neuer Anfragealgorithmus

one-to-all Suche von *s*:

- vorwärts CH Suche von *s* ($\approx$ 0.05 ms)

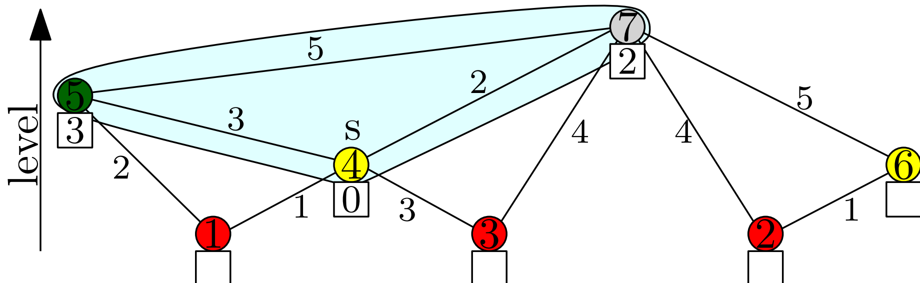


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Neuer Anfragealgorithmus

one-to-all Suche von *s*:

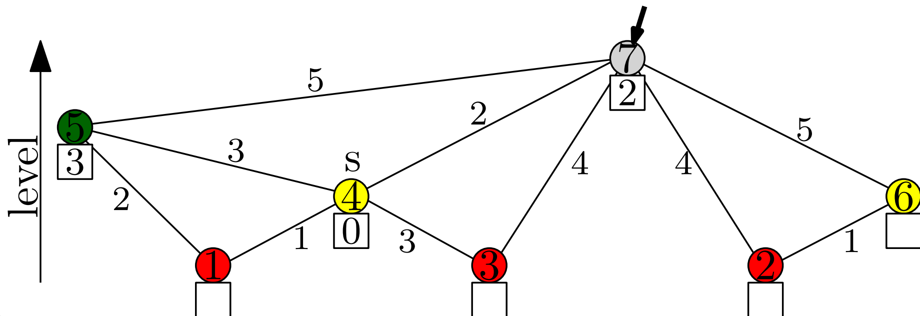
- vorwärts CH Suche von *s* ($\approx$ 0.05 ms)
- setze Distanzen *d* für alle erreichten Knoten



Neuer Anfragealgorithmus

one-to-all Suche von *s*:

- vorwärts CH Suche von s ($\approx$ 0.05 ms)
- setze Distanzen d für alle erreichten Knoten
- bearbeite alle Knoten u in inverser Levelordnung:
  - checke eingehende Kanten (v, u) mit $lev(v) > lev(u)$
  - setze $d(u) = \min\{d(u), d(v) + w(v, u)\}$

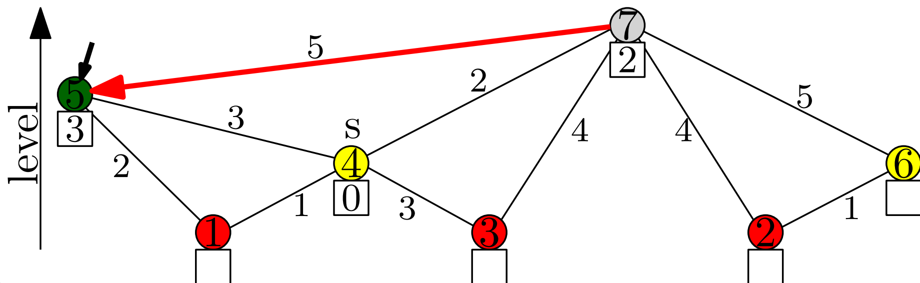


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Neuer Anfragealgorithmus

one-to-all Suche von *s*:

- vorwärts CH Suche von *s* ($\approx$ 0.05 ms)
- setze Distanzen *d* für alle erreichten Knoten
- bearbeite alle Knoten *u* in inverser Levelordnung:
  - checke eingehende Kanten (v, u) mit $lev(v) > lev(u)$
  - setze $d(u) = \min\{d(u), d(v) + w(v, u)\}$

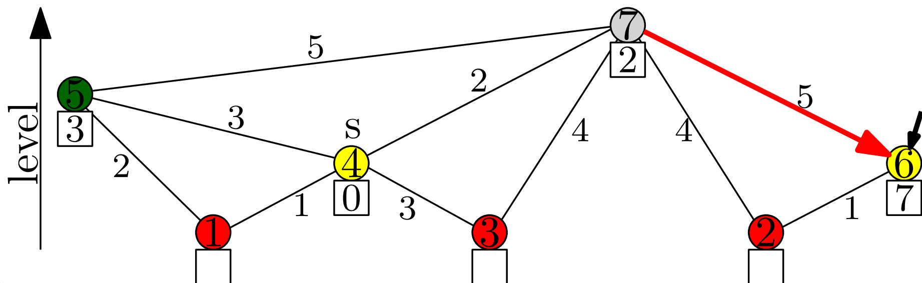


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Neuer Anfragealgorithmus

one-to-all Suche von *s*:

- vorwärts CH Suche von s ($\approx$ 0.05 ms)
- setze Distanzen d für alle erreichten Knoten
- bearbeite alle Knoten u in inverser Levelordnung:
  - checke eingehende Kanten (v, u) mit $lev(v) > lev(u)$
  - setze $d(u) = \min\{d(u), d(v) + w(v, u)\}$

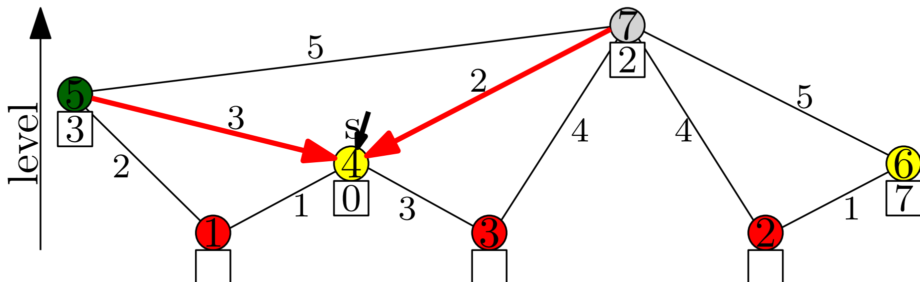


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Neuer Anfragealgorithmus

one-to-all Suche von s:

- vorwärts CH Suche von s ($\approx$ 0.05 ms)
- setze Distanzen d für alle erreichten Knoten
- bearbeite alle Knoten u in inverser Levelordnung:
  - checke eingehende Kanten (v, u) mit $lev(v) > lev(u)$
  - setze $d(u) = \min\{d(u), d(v) + w(v, u)\}$

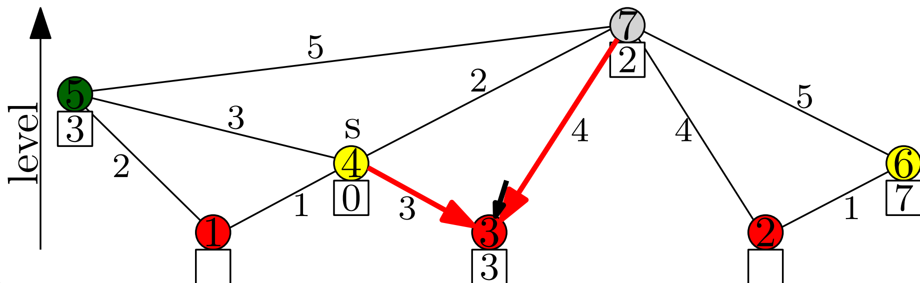


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Neuer Anfragealgorithmus

one-to-all Suche von s:

- vorwärts CH Suche von s ($\approx$ 0.05 ms)
- setze Distanzen d für alle erreichten Knoten
- bearbeite alle Knoten u in inverser Levelordnung:
  - checke eingehende Kanten (v, u) mit $lev(v) > lev(u)$
  - setze $d(u) = \min\{d(u), d(v) + w(v, u)\}$

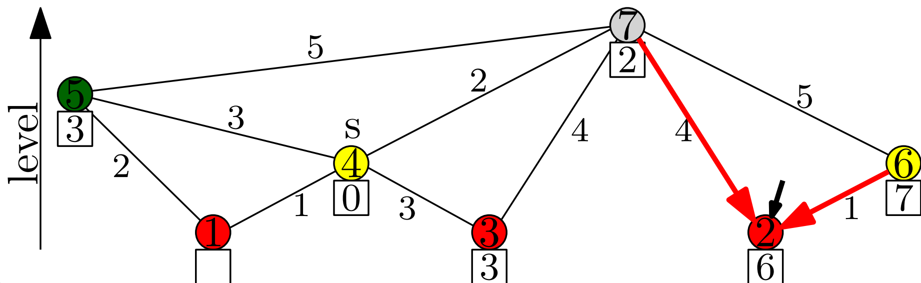


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Neuer Anfragealgorithmus

one-to-all Suche von *s*:

- vorwärts CH Suche von *s* ($\approx$ 0.05 ms)
- setze Distanzen *d* für alle erreichten Knoten
- bearbeite alle Knoten *u* in inverser Levelordnung:
  - checke eingehende Kanten (v, u) mit $lev(v) > lev(u)$
  - setze $d(u) = \min\{d(u), d(v) + w(v, u)\}$

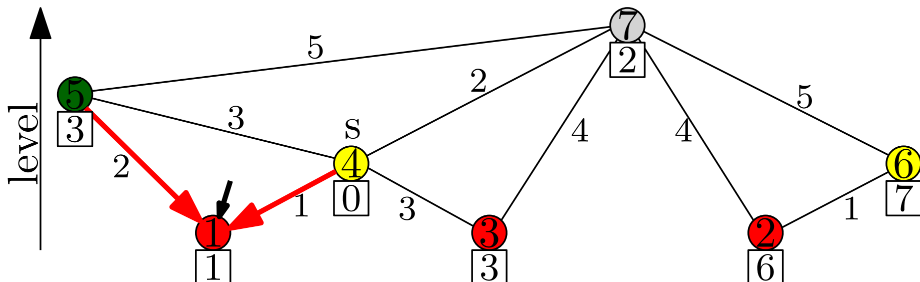


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Neuer Anfragealgorithmus

one-to-all Suche von s:

- vorwärts CH Suche von s ($\approx$ 0.05 ms)
- setze Distanzen d für alle erreichten Knoten
- bearbeite alle Knoten u in inverser Levelordnung:
  - checke eingehende Kanten (v, u) mit $lev(v) > lev(u)$
  - setze $d(u) = \min\{d(u), d(v) + w(v, u)\}$

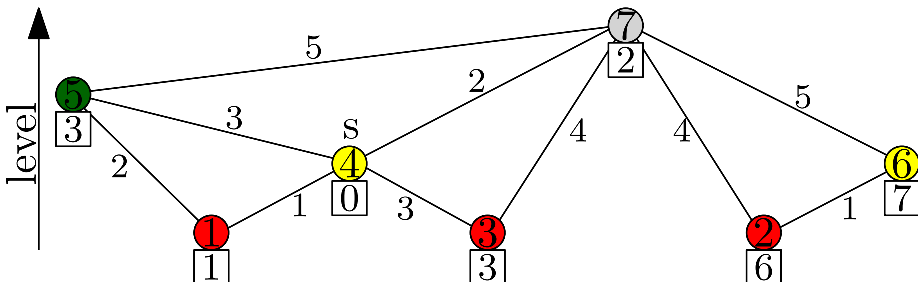


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Neuer Anfragealgorithmus

one-to-all Suche von *s*:

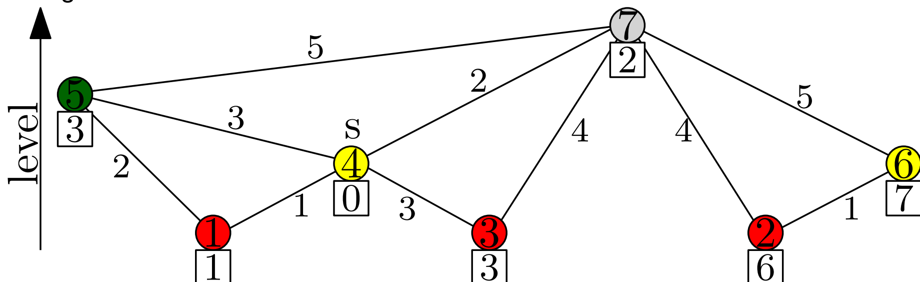
- vorwärts CH Suche von s ($\approx$ 0.05 ms)
- setze Distanzen d für alle erreichten Knoten
- bearbeite alle Knoten u in inverser Levelordnung:
  - checke eingehende Kanten (v, u) mit $lev(v) > lev(u)$
  - setze $d(u) = \min\{d(u), d(v) + w(v, u)\}$
- top-down Bearbeitung ohne Priority Queue (ca. 2.0 s)



Neuer Anfragealgorithmus

one-to-all Suche von s:

- vorwärts CH Suche von s ($\approx$ 0.05 ms)
- setze Distanzen d für alle erreichten Knoten
- bearbeite alle Knoten u in inverser Levelordnung:
  - checke eingehende Kanten (v, u) mit $lev(v) > lev(u)$
  - setze $d(u) = \min\{d(u), d(v) + w(v, u)\}$
- top-down Bearbeitung ohne Priority Queue (ca. 2.0 s)
- genauso schnell wie BFS. Warum das Ganze?

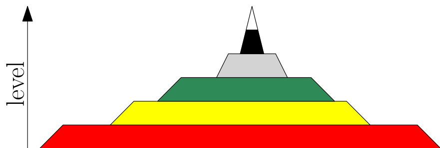


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Analyse

Beobachtung:

- top-down Prozess ist der Flaschenhals

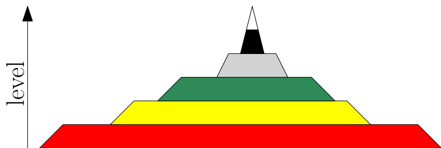


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Analyse

Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient

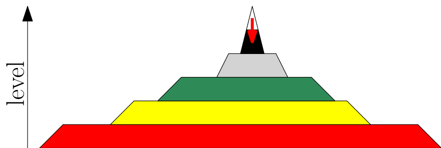


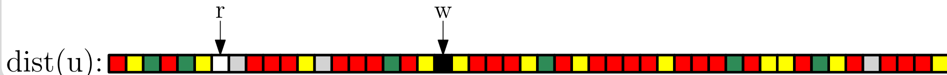
$\mathrm{dist}(u)$:

Analyse

Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient



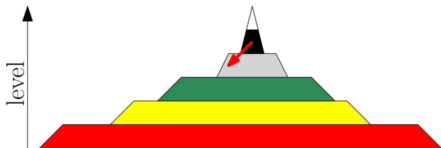


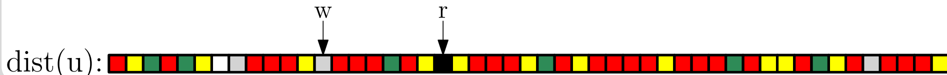
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Analyse

Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient

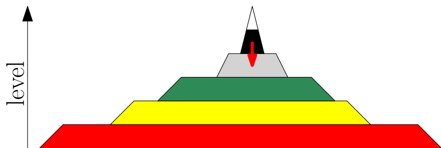


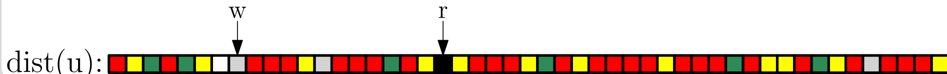


Analyse

Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient



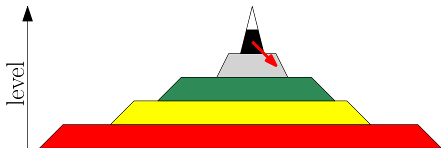


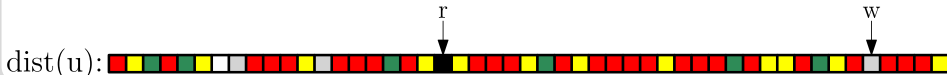
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Analyse

Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient

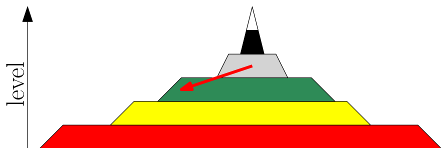




Analyse

Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient

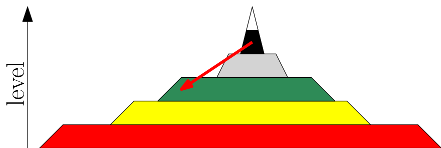


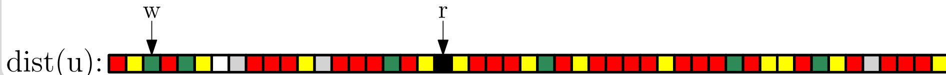
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Analyse

Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient



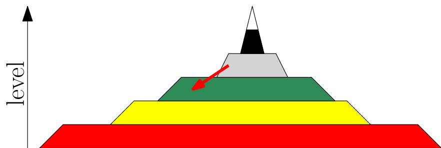


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Analyse

Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient
- Zugriffsmuster sind unabhängig von *s*



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Analyse

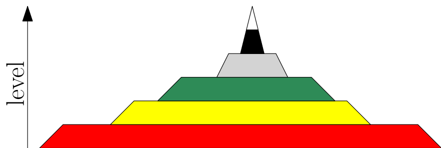
Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient
- Zugriffsmuster sind unabhängig von s

Idee:

- speicher $G_\uparrow$ und $G_\downarrow$ separat
- Umordnung der Knoten, Kanten, und Distanzlabel nach Level

$\Rightarrow$ lesen der Kanten und schreiben der Distanzen wird zu einem sequenziellen Sweep



Analyse

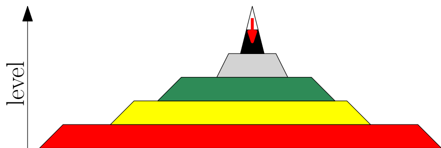
Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient
- Zugriffsmuster sind unabhängig von s

Idee:

- speicher $G_\uparrow$ und $G_\downarrow$ separat
- Umordnung der Knoten, Kanten, und Distanzlabel nach Level

$\Rightarrow$ lesen der Kanten und schreiben der Distanzen wird zu einem sequenziellen Sweep



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Analyse

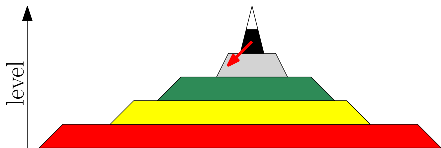
Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient
- Zugriffsmuster sind unabhängig von s

Idee:

- speicher $G_\uparrow$ und $G_\downarrow$ separat
- Umordnung der Knoten, Kanten, und Distanzlabel nach Level

$\Rightarrow$ lesen der Kanten und schreiben der Distanzen wird zu einem sequenziellen Sweep



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Analyse

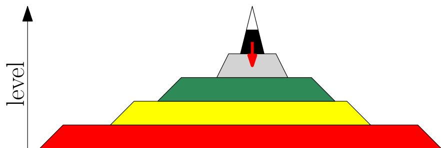
Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient
- Zugriffsmuster sind unabhängig von s

Idee:

- speicher $G_\uparrow$ und $G_\downarrow$ separat
- Umordnung der Knoten, Kanten, und Distanzlabel nach Level

$\Rightarrow$ lesen der Kanten und schreiben der Distanzen wird zu einem sequenziellen Sweep



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Analyse

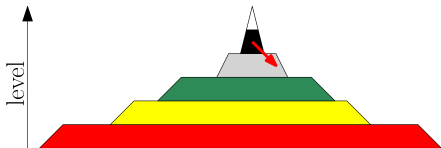
Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient
- Zugriffsmuster sind unabhängig von s

Idee:

- speicher $G_{\uparrow}$ und $G_{\downarrow}$ separat
- Umordnung der Knoten, Kanten, und Distanzlabel nach Level

$\Rightarrow$ lesen der Kanten und schreiben der Distanzen wird zu einem sequenziellen Sweep



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Analyse

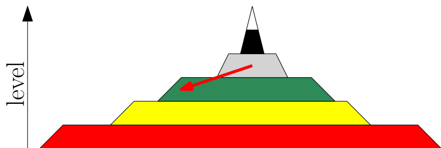
Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient
- Zugriffsmuster sind unabhängig von s

Idee:

- speicher $G_\uparrow$ und $G_\downarrow$ separat
- Umordnung der Knoten, Kanten, und Distanzlabel nach Level

$\Rightarrow$ lesen der Kanten und schreiben der Distanzen wird zu einem sequenziellen Sweep



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Analyse

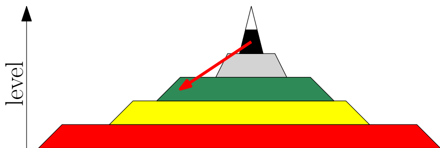
Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient
- Zugriffsmuster sind unabhängig von s

Idee:

- speicher $G_{\uparrow}$ und $G_{\downarrow}$ separat
- Umordnung der Knoten, Kanten, und Distanzlabel nach Level

$\Rightarrow$ lesen der Kanten und schreiben der Distanzen wird zu einem sequenziellen Sweep



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Analyse

Beobachtung:

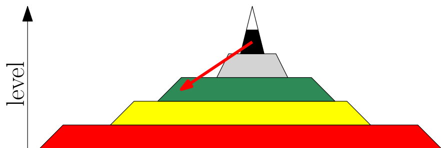
- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient
- Zugriffsmuster sind unabhängig von s

Idee:

- speicher $G_\uparrow$ und $G_\downarrow$ separat
- Umordnung der Knoten, Kanten, und Distanzlabel nach Level

$\Rightarrow$ lesen der Kanten und schreiben der Distanzen wird zu einem sequenziellen Sweep

$\Rightarrow$ 172 ms pro Baum



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Analyse

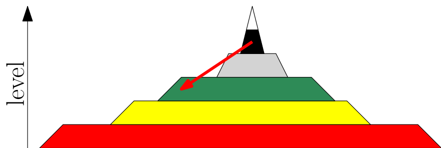
Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient
- Zugriffsmuster sind unabhängig von s

Idee:

- speicher $G_\uparrow$ und $G_\downarrow$ separat
- Umordnung der Knoten, Kanten, und Distanzlabel nach Level

⇒ lesen der Kanten und schreiben der Distanzen wird zu einem sequenziellen Sweep

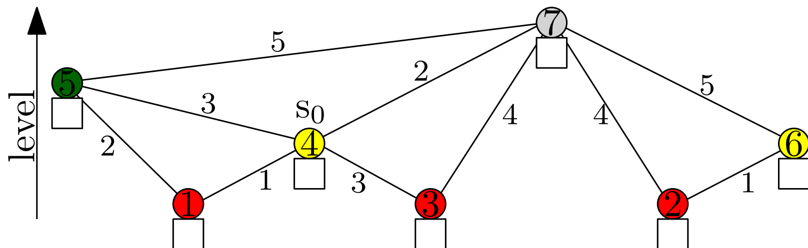


⇒ 172 ms pro Baum

- aber lesen der Distanzen immer noch ineffizient

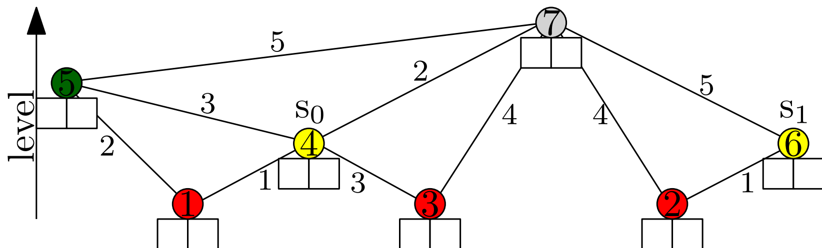
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Szenario: Multiple Startknoten



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

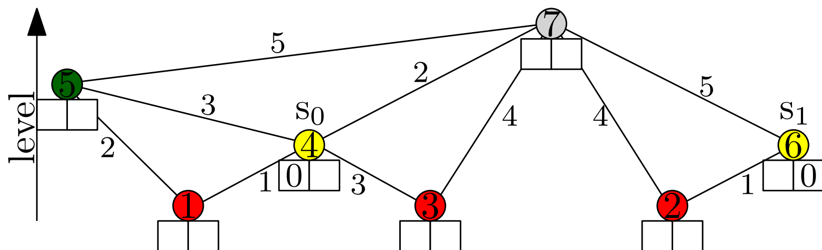
Szenario: Multiple Startknoten



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

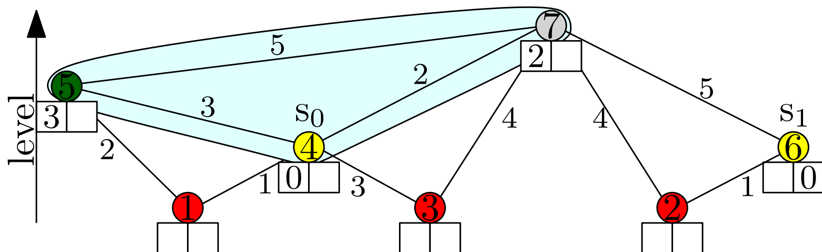
Szenario: Multiple Startknoten

- *k* Vorwärtssuchen



Szenario: Multiple Startknoten

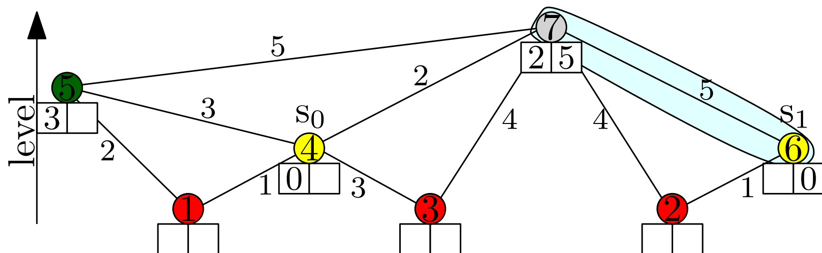
- *k* Vorwärtssuchen



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Szenario: Multiple Startknoten

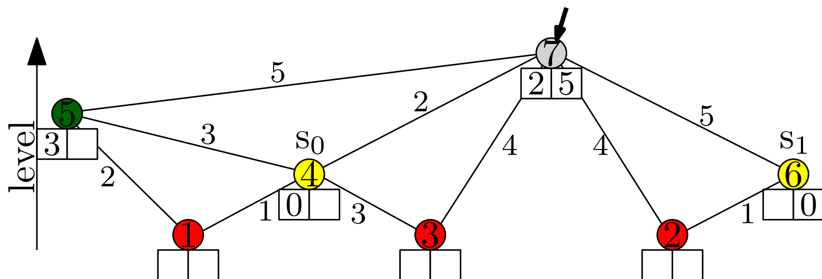
- *k* Vorwärtssuchen



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

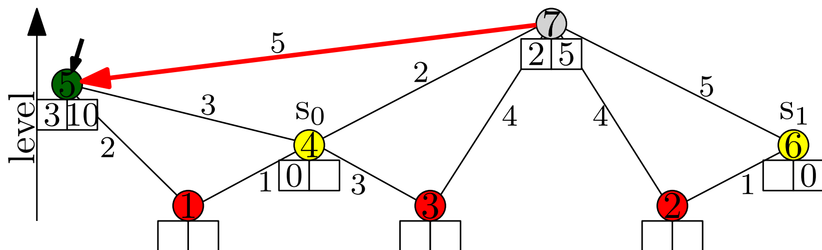
Szenario: Multiple Startknoten

- *k* Vorwärtssuchen
- ein sweep (update aller *k* Werte)
- speicher Distanzlabel pro Knoten



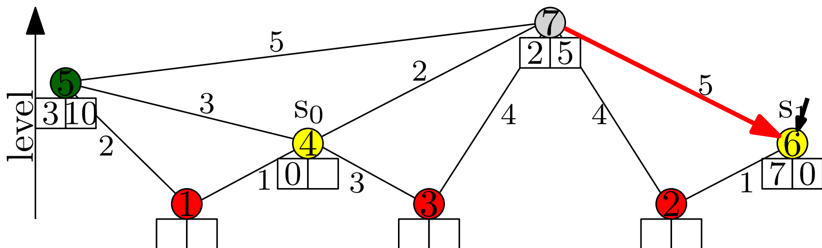
Szenario: Multiple Startknoten

- *k* Vorwärtssuchen
- ein sweep (update aller *k* Werte)
- speicher Distanzlabel pro Knoten



Szenario: Multiple Startknoten

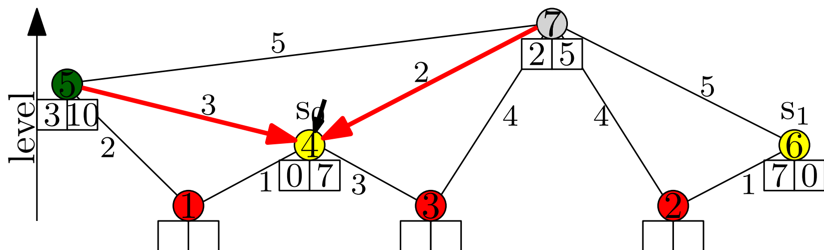
- *k* Vorwärtssuchen
- ein sweep (update aller *k* Werte)
- speicher Distanzlabel pro Knoten



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

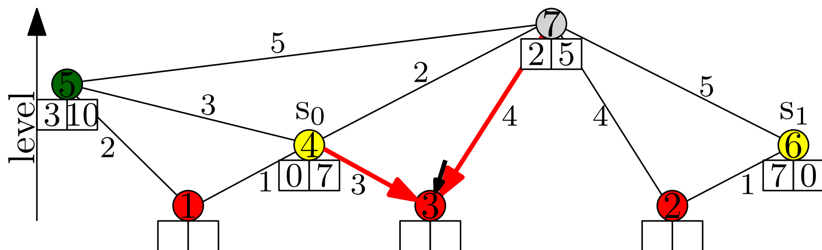
Szenario: Multiple Startknoten

- *k* Vorwärtssuchen
- ein sweep (update aller *k* Werte)
- speicher Distanzlabel pro Knoten



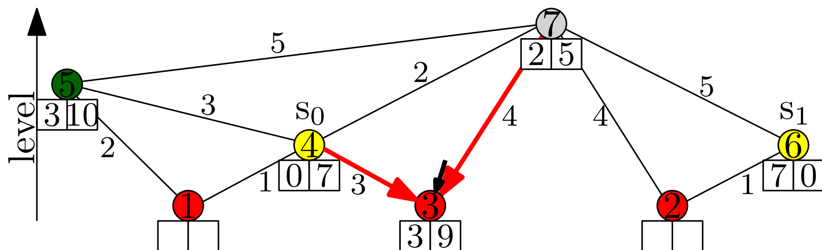
Szenario: Multiple Startknoten

- *k* Vorwärtssuchen
- ein sweep (update aller *k* Werte)
- speicher Distanzlabel pro Knoten



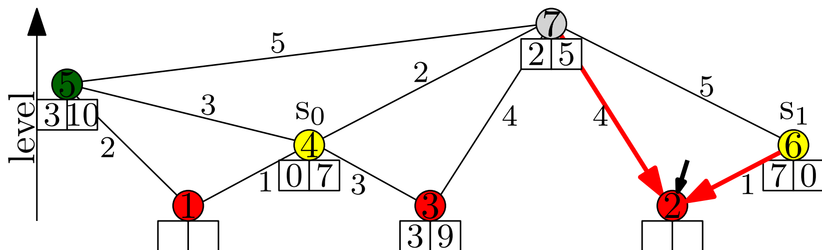
Szenario: Multiple Startknoten

- *k* Vorwärtssuchen
- ein sweep (update aller *k* Werte)
- speicher Distanzlabel pro Knoten



Szenario: Multiple Startknoten

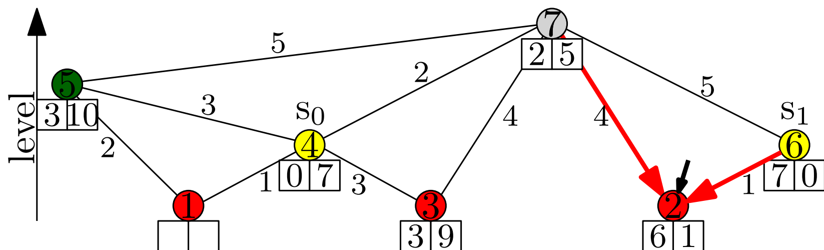
- *k* Vorwärtssuchen
- ein sweep (update aller *k* Werte)
- speicher Distanzlabel pro Knoten



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Szenario: Multiple Startknoten

- *k* Vorwärtssuchen
- ein sweep (update aller *k* Werte)
- speicher Distanzlabel pro Knoten
- Vorteil: Mehr Daten pro Lesezugriff
- 96.8 ms pro Baum ($k = 16$)



SIMD/SSE

Karlsruher Institut für Technologie

- SIMD : Single Instruction Multiple Data
- Spezielle CPU-Instruktion die mit Vektoren fester Länge arbeiten und nur einen CPU-Takt benötigen

- Gibt es in allen modernen x86 Prozessoren
- 128 Bit Registers à 4×32 Bit oder à 8×16 Bit Ganzzahlen

- Die 128-Bit x86 Implementierung heißt SSE
- SIMD ist das generelle Konzept

- Neuerdings gibt es auch
  - Die 256-Bit x86 Implementierung heißt AVX
  - Die 512-Bit x86 Implementierung wird kommen
    aktuell nur in Xeon-Phi zu haben

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

SIMD/SSE

Karlsruher Institut für Technologie

Ohne SSE:

```
int a[8],b[8],c[8];
c[0]=a[0]+b[0];    c[1]=a[1]+b[1];    c[2]=a[2]+b[2];
c[3]=a[3]+b[3];    c[4]=a[4]+b[4];    c[5]=a[5]+b[5];
c[6]=a[6]+b[6];    c[7]=a[7]+b[7];
```

8 Zeiteinheiten

Mit SSE:

```
__m128i a[2],b[2],c[2];
c[0]=_mm_add_epi32(a[0],b[0]);
c[1]=_mm_add_epi32(a[1],b[1]);
```

2 Zeiteinheiten

Aber: Beschleunigung von 4 gibt es nur bei compute-bound Algorithmen.

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

SIMD/SSE

Ohne SSE:

```
short a[8],b[8],c[8];
c[0]=a[0]+b[0];    c[1]=a[1]+b[1];    c[2]=a[2]+b[2];
c[3]=a[3]+b[3];    c[4]=a[4]+b[4];    c[5]=a[5]+b[5];
c[6]=a[6]+b[6];    c[7]=a[7]+b[7];
```

8 Zeiteinheiten

Mit SSE:

```
__m128i a,b,c;
c=_mm_add_epi16(a,b);
```

Eine Zeiteinheit

Aber: Beschleunigung von 8 gibt es nur bei compute-bound Algorithmen.

SIMD/SSE

Einfache if/else-Konstrukte gibt es auch.

Ohne SSE:

```
short a[8],b[8],c[8];
for(int i=0; i<8; ++i)
    c[i] = a[i] < b[i] ? a[i] : b[i];
```

Mit SSE:

```
__m128i a,b,c;
c=_mm_blendv_epi8(a,b,_mm_cmplt_epi16(a,b));
```


Karlsruher Institut für Technologie
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

SIMD/SSE

Einfache if/else-Konstrukte gibt es auch.

Ohne SSE:

```
short a[8],b[8],c[8];
for(int i=0; i<8; ++i)
    c[i] = a[i] < b[i] ? a[i] : b[i];
```

Mit SSE:

```
__m128i a,b,c;
c=_mm_blendv_epi8(a,b,_mm_cmplt_epi16(a,b));
```

Oder einfach die Minimumanweisung verwenden:

```
__m128i a,b,c;
c=_mm_min_epi16(a,b);
```


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

SIMD/SSE

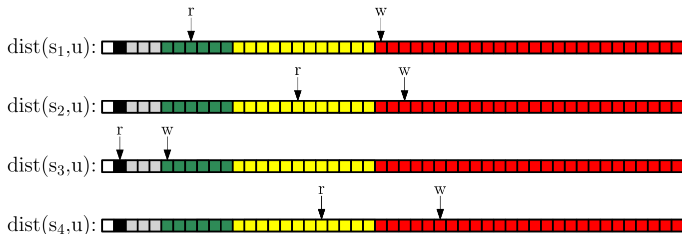
Karlsruher Institut für Technologie

- Bei mehreren Bäumen: PHAST Operationen können mit SSE umgesetzt werden
- scanne 4 Distanzlabel auf einmal
- 37.1 ms pro Baum ($k = 16$)

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

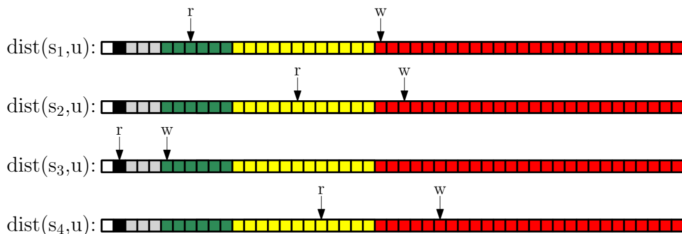
Parallelisierung

- Jeder Thread arbeitet unterschiedliche Startknoten ab



Parallelisierung

- Jeder Thread arbeitet unterschiedliche Startknoten ab



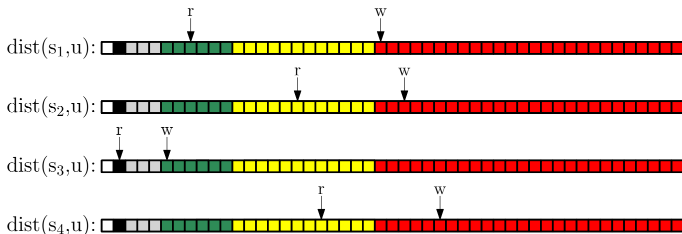
Ergebnisse:

- 16 Startenknoten pro Sweep (updates via SSE)
- multi-core nach Startknoten ⇒ 64 Startknoten parallel (4 cores)
- 18.8 ms per Baum

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Parallelisierung

- Jeder Thread arbeitet unterschiedliche Startknoten ab



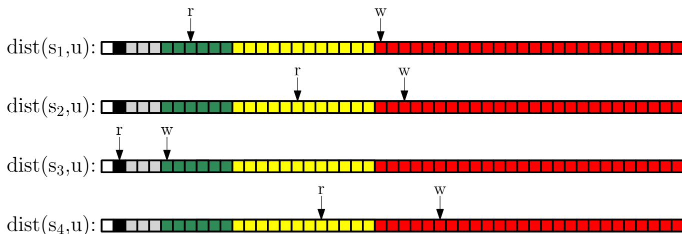
Ergebnisse:

- 16 Startenknoten pro Sweep (updates via SSE)
- multi-core nach Startknoten ⇒ 64 Startknoten parallel (4 cores)
- 18.8 ms per Baum
- Warum kein perfekter Speedup?

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Parallelisierung

- Jeder Thread arbeitet unterschiedliche Startknoten ab



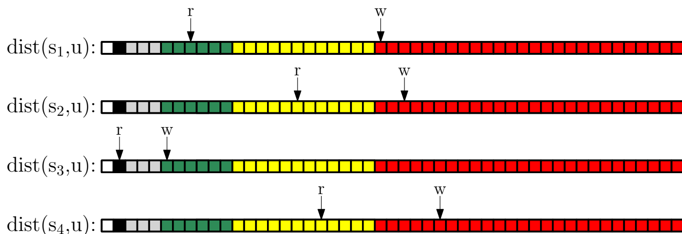
Ergebnisse:

- 16 Startenknoten pro Sweep (updates via SSE)
- multi-core nach Startknoten ⇒ 64 Startknoten parallel (4 cores)
- 18.8 ms per Baum
- Warum kein perfekter Speedup?
- lower bound tests zeigen: nah an Speicherbandbreite

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Parallelisierung

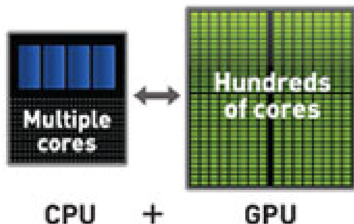
- Jeder Thread arbeitet unterschiedliche Startknoten ab



Ergebnisse:

- 16 Startenknoten pro Sweep (updates via SSE)
- multi-core nach Startknoten ⇒ 64 Startknoten parallel (4 cores)
- 18.8 ms per Baum
- Warum kein perfekter Speedup?
- lower bound tests zeigen: nah an Speicherbandbreite
- kann eine GPU helfen?

GPU Architektur



Intel Xeon X5680:

- 3.33 GHz, oft $\geq$ 10 GB RAM
- 32 GB/s Speicherbandbreite
- 6 Kerne
- SIMD/SSE mit 4 floats/ints per Vektor

NVIDIA GTX 580:

- 772 MHz, 1.5 GB RAM
- 192 GB/s Speicherbandbreite
- 16 Kerne, 32 parallele Threads pro Kern $\Rightarrow$ 512 parallele Threads
- eingeschränkte Berechnungen

512 Threads?

Karlsruher Institut für Technologie

Gemeinsamer Codezeiger:

- GPU Threads innerhalb eines Kerns haben einen gemeinsamen Codezeiger.
- If-Bedinung wahr für ein Thread $\rightarrow$ Alle Threads durchlaufen den Wahr-Block.
  - Aber für die falsch-Threads werden die Anweisungen ausmaskiert.
- Schleifen dauern immer so lange wie der längste Thread braucht.
- Stichwort: Single Instruction Multiple Threads

Verwandt mit SIMD:

- Jede Vektorkomponent ist ein ”Thread”.
- Beim komponentenweisen if-else wird auch eine Seite ausmaskiert.

512 Threads?

Gemeinsamer Codezeiger:

- GPU Threads innerhalb eines Kerns haben einen gemeinsamen Codezeiger.
- If-Bedinung wahr für ein Thread $\rightarrow$ Alle Threads durchlaufen den Wahr-Block.
  - Aber für die falsch-Threads werden die Anweisungen ausmaskiert.
- Schleifen dauern immer so lange wie der längste Thread braucht.
- Stichwort: Single Instruction Multiple Threads

Verwandt mit SIMD:

- Jede Vektorkomponent ist ein ”Thread”.
- Beim komponentenweisen if-else wird auch eine Seite ausmaskiert.

Fairerer Vergleich:

- 772 MHz $\times$ 16 Kerne $\times$ 32 ”Threads”$=$ 395264
- 3.33 GHz $\times$ 6 Kerne $\times$ 4 SIMD-Vektor $=$ 79992
- In optimalen Bedinungen bis zu 5-mal mehr Berechnungen

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Weitere GPU Einschränkungen

Karlsruher Institut für Technologie

- Der Transfer CPU-Speicher $\leftrightarrow$ GPU-Speicher ist langsam.
- GPU-Threads zwischen Kernen synchronisieren ist sehr teuer. Geht nur indem die CPU die Arbeit in mehere Jobs aufteilt.
- Keine Cache-Kohärenz zwischen Cores:
  1. Core A schreibt einen neuen Wert and die Adresse 42.
  2. Core B ließt die Adresse 42 aus, aber erhält den alten Wert.

  Erklärung: Der neue Wert steckt noch im Cache von A fest.
- Langsam, wenn GPU-Threads durcheinander auf den Speicher zugreifen.

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

GPHAST - Ideen

Karlsruher Institut für Technologie

Beobachtungen:

- Aufwärtsuche ist schnell
- Flaschenhals ist der lineare Sweep
- Speicherbandbreite das Problem

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

GPHAST - Ideen

Karlsruher Institut für Technologie

Beobachtungen:

- Aufwärtsuche ist schnell
- Flaschenhals ist der lineare Sweep
- Speicherbandbreite das Problem

Idee:

- speicher CH und Distanzarray auf der GPU
- Aufwärtssuche auf der CPU
- kopiere besuchte Knoten zur GPU (weniger als 2 kB)
- **Wichtig**: Nur die besuchten kopieren, nicht alle
- linearen Sweep auf der GPU

GPHAST - Ideen

Beobachtungen:

- Aufwärtsuche ist schnell
- Flaschenhals ist der lineare Sweep
- Speicherbandbreite das Problem

Idee:

- speicher CH und Distanzarray auf der GPU
- Aufwärtssuche auf der CPU
- kopiere besuchte Knoten zur GPU (weniger als 2 kB)
- **Wichtig**: Nur die besuchten kopieren, nicht alle
- linearen Sweep auf der GPU

Problem:

- nicht genug Speicher auf GPU um Tausende von Bäumen parallel zu bearbeiten
- wir müssen eine einzelne Baumberechnung parallelisieren

Paralleler Linearer Sweep

Beobachtung:

- beim Scannen von Level i:
  - nur eingehende Kanten von level $> i$ wichtig
  - schreiben von Distanzlabeln in Level i, lesen von Level $> i$
  - Distanzlabel für Level $> i$ sind korrekt
- scannen eines Level-i Knoten ist unabhängig von anderen Level-i Knoten

Paralleler Linearer Sweep

Beobachtung:

- beim Scannen von Level i:
  - nur eingehende Kanten von level $> i$ wichtig
  - schreiben von Distanzlabeln in Level i, lesen von Level $> i$
  - Distanzlabel für Level $> i$ sind korrekt
- scannen eines Level-i Knoten ist unabhängig von anderen Level-i Knoten

Paralleler Linearer Sweep

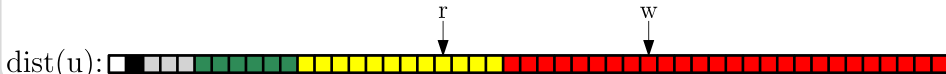
Beobachtung:

- beim Scannen von Level i:
  - nur eingehende Kanten von level $> i$ wichtig
  - schreiben von Distanzlabeln in Level i, lesen von Level $> i$
  - Distanzlabel für Level $> i$ sind korrekt
- scannen eines Level-i Knoten ist unabhängig von anderen Level-i Knoten

Paralleler Linearer Sweep

Beobachtung:

- beim Scannen von Level i:
  - nur eingehende Kanten von level $> i$ wichtig
  - schreiben von Distanzlabeln in Level i, lesen von Level $> i$
  - Distanzlabel für Level $> i$ sind korrekt
- scannen eines Level-i Knoten ist unabhängig von anderen Level-i Knoten



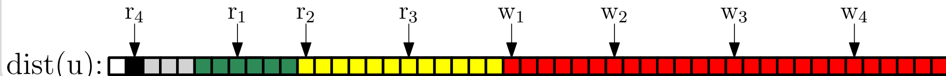
Paralleler Linearer Sweep

Beobachtung:

- beim Scannen von Level *i*:
  - nur eingehende Kanten von level $> i$ wichtig
  - schreiben von Distanzlabeln in Level *i*, lesen von Level $> i$
  - Distanzlabel für Level $> i$ sind korrekt
- scannen eines Level-*i* Knoten ist unabhängig von anderen Level-*i* Knoten

Idee:

- scanne alle Knoten auf Level *i* parallel
- Synchronization nach jedem Level
- ein Thread pro Knoten



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

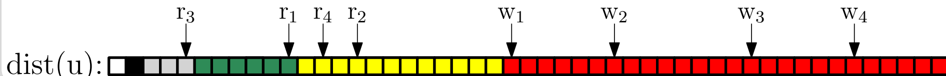
Paralleler Linearer Sweep

Beobachtung:

- beim Scannen von Level i:
  - nur eingehende Kanten von level $> i$ wichtig
  - schreiben von Distanzlabeln in Level i, lesen von Level $> i$
  - Distanzlabel für Level $> i$ sind korrekt
- scannen eines Level-i Knoten ist unabhängig von anderen Level-i Knoten

Idee:

- scanne alle Knoten auf Level i parallel
- Synchronization nach jedem Level
- ein Thread pro Knoten



Paralleler Linearer Sweep

Beobachtung:

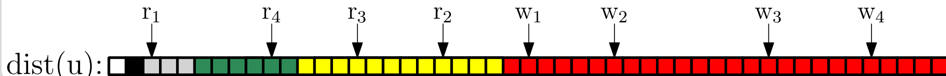
- beim Scannen von Level i:
  - nur eingehende Kanten von level $> i$ wichtig
  - schreiben von Distanzlabeln in Level i, lesen von Level $> i$
  - Distanzlabel für Level $> i$ sind korrekt
- scannen eines Level-i Knoten ist unabhängig von anderen Level-i Knoten

Idee:

- scanne alle Knoten auf Level i parallel
- Synchronization nach jedem Level
- ein Thread pro Knoten

results:

- 5.5 ms auf NVIDIA GTX 580
- Beschleunigung von 511 gegenüber Dijkstra



Paralleler Linearer Sweep

Beobachtung:

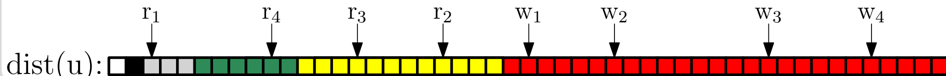
- beim Scannen von Level *i*:
  - nur eingehende Kanten von level $> i$ wichtig
  - schreiben von Distanzlabeln in Level *i*, lesen von Level $> i$
  - Distanzlabel für Level $> i$ sind korrekt
- scannen eines Level-*i* Knoten ist unabhängig von anderen Level-*i* Knoten

Idee:

- scanne alle Knoten auf Level *i* parallel
- Synchronization nach jedem Level
- ein Thread pro Knoten

results:

- 5.5 ms auf NVIDIA GTX 580
- Beschleunigung von 511 gegenüber Dijkstra
- (mehrere Bäume: 2.2 ms)



PHAST

PHAST auf 4-Kern Workstation (Core-i7 920)

sources/ sweep	time per tree [ms]					
	1 core		2 cores		4 cores	
1	171.9		86.7		47.1	
4	121.8	(67.6)	61.5	(35.5)	32.5	(24.4)
8	105.5	(51.2)	53.5	(28.0)	28.3	(20.8)
16	96.8	(37.1)	49.4	(22.1)	25.9	(18.8)

Werte in Klammern mit SSE aktiviert

GPHAST

Karlsruher Institut für Technologie

PHAST auf Nvidia GTX 580

trees / sweep	memory [MB]	time [ms]
1	395	5.53
2	464	3.93
4	605	3.02
8	886	2.52
16	1448	2.21

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

weitere Eingaben

Karlsruher Institut für Technologie

algorithm	device	Europe time	Europe distance	USA time	USA distance
Dijkstra	4-core workstation	947.72	609.19	1269.12	947.75
	12-core server	288.81	177.58	380.40	280.17
	48-core server	168.49	108.58	229.00	167.77
PHAST	4-core workstation	18.81	22.25	27.11	28.81
	12-core server	7.20	8.27	10.42	10.71
	48-core server	4.03	5.03	6.18	6.58
GPHAST	GTX 580	2.21	3.88	3.41	4.65

Beobachtung:

- Beschleunigung für Distanzmetrik geringer

All-Pairs Shortest Paths

Karlsruher Institut für Technologie

Eingabe: Europa mit Reisezeiten

algorithm	device	time	energy [MJ]
Dijkstra	4-core workstation	197d	
	12-core server	60d	
	48-core server	35d	
PHAST	4-core workstation	94h	
	12-core server	36h	
	48-core server	20h	
GPHAST	GTX 580		

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

All-Pairs Shortest Paths

Karlsruher Institut für Technologie

Eingabe: Europa mit Reisezeiten

algorithm	device	time	energy [MJ]
Dijkstra	4-core workstation	197d	
	12-core server	60d	
	48-core server	35d	
PHAST	4-core workstation	94h	
	12-core server	36h	
	48-core server	20h	
GPHAST	GTX 580	11h	

All-Pairs Shortest Paths

Karlsruher Institut für Technologie

Eingabe: Europa mit Reisezeiten

algorithm	device	time	energy [MJ]
Dijkstra	4-core workstation	197d	
	12-core server	60d	
	48-core server	35d	
PHAST	4-core workstation	94h	
	12-core server	36h	
	48-core server	20h	
GPHAST	GTX 580	11h	

4-core workstation without GPU: 163 watts
4-core workstation with GPU: 375 watts
12-core server: 332 watts
48-core server: 747 watts

All-Pairs Shortest Paths

KIT
Karlsruher Institut für Technologie

Eingabe: Europa mit Reisezeiten

algorithm	device	time	energy [MJ]
Dijkstra	4-core workstation	197d	2780.6
	12-core server	60d	1725.9
	48-core server	35d	2265.5
PHAST	4-core workstation	94h	55.2
	12-core server	36h	43.0
	48-core server	20h	54.2
GPHAST	GTX 580	11h	14.9

4-core workstation without GPU: 163 watts
4-core workstation with GPU: 375 watts
12-core server: 332 watts
48-core server: 747 watts

Baumrekonstruktion

Karlsruher Institut für Technologie

bis jetzt:

- nur Distanzen berechnet, nicht Bäume

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Baumrekonstruktion

Karlsruher Institut für Technologie

bis jetzt:

- nur Distanzen berechnet, nicht Bäume

Idee:

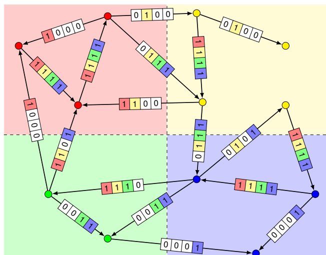
- iteration über alle Kanten (1 Thread pro Kante)
- Wenn $d(v) + \text{len}(v, u) = d(u)$ dann ist v der Vorgänger von u (sofern kürzeste Wege eindeutig sind)

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Arc-Flags

Idee:

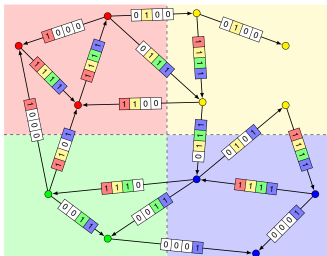
- benutze GPHAST zum Berechnen der Bäume von den Randknoten aus



Arc-Flags

Idee:

- benutze GPHAST zum Berechnen der Bäume von den Randknoten aus
- setze Flaggen durch zusätzlichen Sweep auf GPU
  Wichtig, weil alle Bäume “auf CPU kopieren” teuer ist



Arc-Flags

Karlsruher Institut für Technologie

Idee:

- benutze GPHAST zum Berechnen der Bäume von den Randknoten aus
- setze Flaggen durch zusätzlichen Sweep auf GPU
  Wichtig, weil alle Bäume “auf CPU kopieren” teuer ist
- Vorberechnung sinkt von 17 Stunden auf 3 Minuten

Hybrid GPHAST

Beobachtung:

- Synchronization des Level kostet Zeit auf der GPU (5 μs pro Level)
- obere Level sind klein

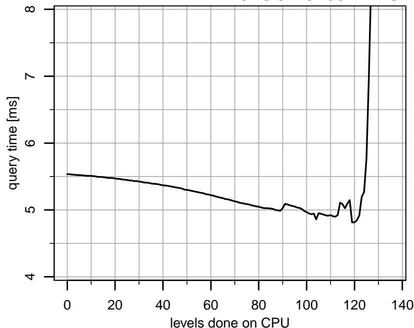
Idee:

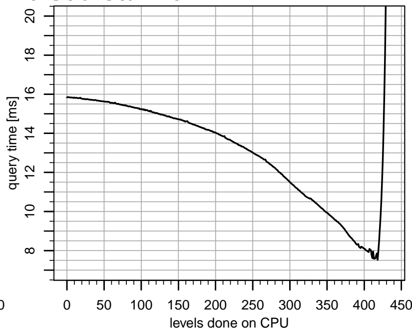
- beginne linearen Sweep auf CPU (bis Level k)
- kopiere Suchraum und alle Distanzlabel für Knoten oberhalb k zur GPU
- restlicher Scan auf der GPU

Hybrid GPHAST: Ergebnisse

Karlsruher Institut für Technologie

Reisezeiten vs. Reisedistanzen





Es lohnt sich, ein Paar Level auf der CPU zu berechnen.

Zusammenfassung

Karlsruher Institut für Technologie

- neuer Algorithmus für kürzeste Wege Bäume
- skaliert auf Modern Architektur
- ein Baum auf GPU: 5.5 ms
  (ungefähr 0.31 ns pro Eintrag)
- real-time Berechnung von kompletten Bäumen
- 16 Bäume auf einer GPU auf einmal: 2.2 ms pro Baum
  (ungefähr 0.13 ns pro Eintrag)
- APSP in 11 Stunden (auf workstation mit einer GPU),
  anstellen von 6 Monaten (auf 4 Kernen)
- erlaubt APSP-basierte Berechnungen
- 150 mal Energie-effizienter als Dijkstras Algorithmus
- funktioniert nur, wenn CH funktioniert

Übungsfragen

Karlsruher Institut für Technologie

Wie kann man folgende Anwendungen mit PHAST beschleunigen? Welche sind auf der GPU schneller?

- Stau-Updates für Techniken eine APSP-basierte Vorberechnung haben
- APSP-Matrix berechnen und im Speicher halten
- Diameter berechnen
- ALT
- Reach

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Übungsfragen

Karlsruher Institut für Technologie

Wie kann man folgende Anwendungen mit PHAST beschleunigen? Welche sind auf der GPU schneller?

- Stau-Updates für Techniken eine APSP-basierte Vorberechnung haben
  Geht nicht, da 11h Reaktionszeit zu langsam sind
- APSP-Matrix berechnen und im Speicher halten
  Geht nicht, da es nicht genug RAM gibt um die Ergebnisse im Speicher zu halten.
- Diameter berechnen
  Geht. **CPU**: Straight-Forward **GPU**: Geht, aber man darf nur eine parallele Maximumbildung über alle n Einträge machen. Sonst ist es langsamer als auf der CPU.
- ALT
  Geht auf der CPU, GPU -¿ CPU Transfer zu teuer
- Reach
  Geht. **CPU**: Straight-Forward **GPU**: Baumhöhenberechnung dominiert. Auf der CPU ist es oft schneller.

Literatur I

Julian Arz, Dennis Luxen, and Peter Sanders.
Transit node routing reconsidered.
In *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13)*, volume 7933 of *Lecture Notes in Computer Science*, pages 55–66. Springer, 2013.

Holger Bast, Stefan Funke, and Domagoj Matijevic.
Transit - ultrafast shortest-path queries with linear-time preprocessing.
In *The Shortest Path Problem: Ninth DIMACS Implementation Challenge -*, November 2006.

Holger Bast, Stefan Funke, Domagoj Matijevic, Peter Sanders, and Dominik Schultes.
In transit to constant shortest-path queries in road networks.
In *Proceedings of the 9th Workshop on Algorithm Engineering and Experiments (ALENEX'07)*, pages 46–59. SIAM, 2007.

Edsger W. Dijkstra.
A note on two problems in connexion with graphs.
Numerische Mathematik, 1(1):269–271, 1959.

Literatur II

Karlsruher Institut für Technologie

Michael L. Fredman and Robert Tarjan.
Fibonacci heaps and their uses in improved network optimization algorithms.
Journal of the ACM, 1987.

Andrew V. Goldberg.
A simple shortest path algorithm with linear average time.
In *Proceedings of the 9th Annual European Symposium on Algorithms (ESA'01)*, volume 2161 of *Lecture Notes in Computer Science*, pages 230–241, 2001.

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.

Ulrich Meyer and Peter Sanders.
δ-stepping: A parallelizable shortest path algorithm.
Journal of Algorithms, 49(1):114–152, 2003.