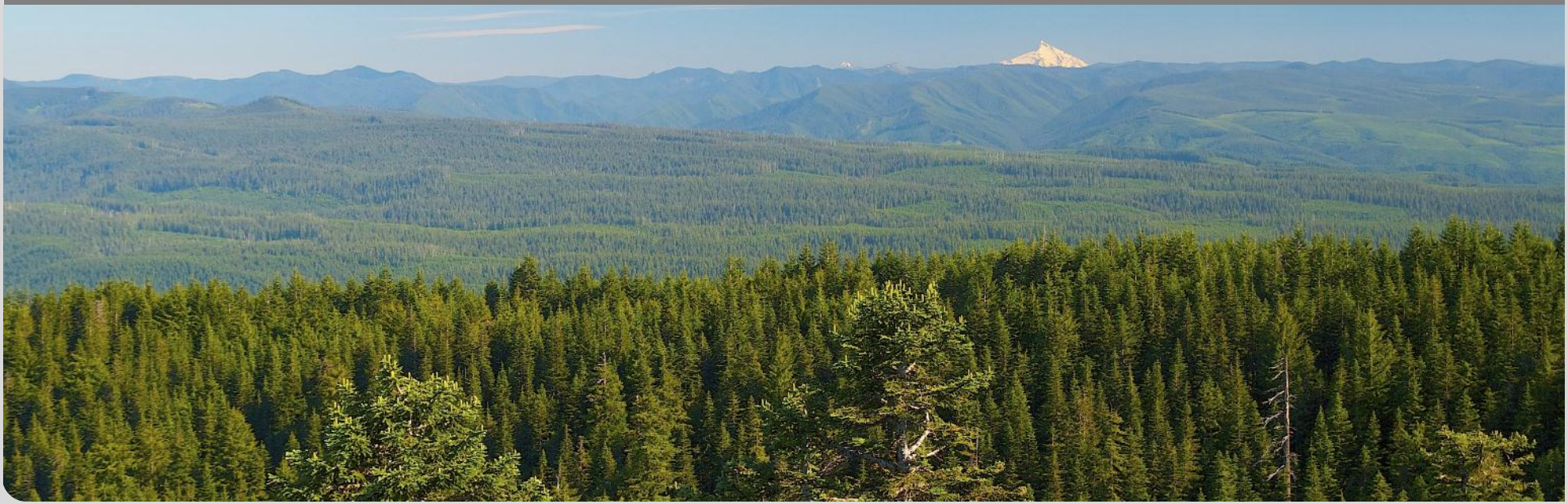
Karlsruher Institut für Technologie

Algorithmen für Routenplanung

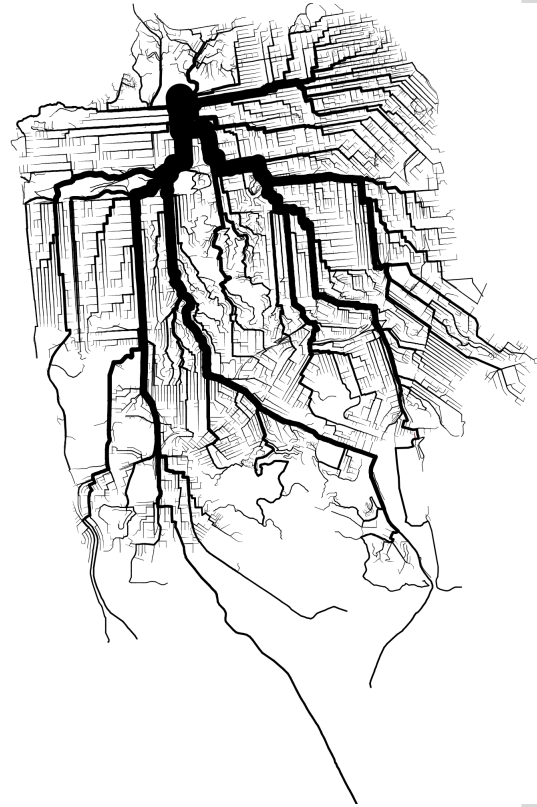
10. Sitzung, Sommersemester 2013

Julian Dibbelt | 29. Mai 2013

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



Kürzeste-Wege Bäume



Anfrage:

- gegeben ein nicht negativ gewichteter gerich
  Knoten s
- berechne Distanzen von s zu *allen* anderen

Lösung:

- Dijkstra [Dij59]

Fakten:

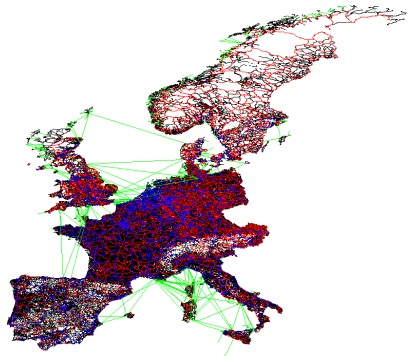
- $O(m + n \log n)$ mit Fibonacci Heaps [FT87]
- linear (mit kleiner Konstanten) in Praxis [Gol01]
- Ausnutzung von moderner Hardware schwierig

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],[MBBC09]
- mehr Operationen als Dijkstra
- keine grosse Beschleunigung auf dünnen Graphen
- Berechnen von mehreren Bäumen ist einfach

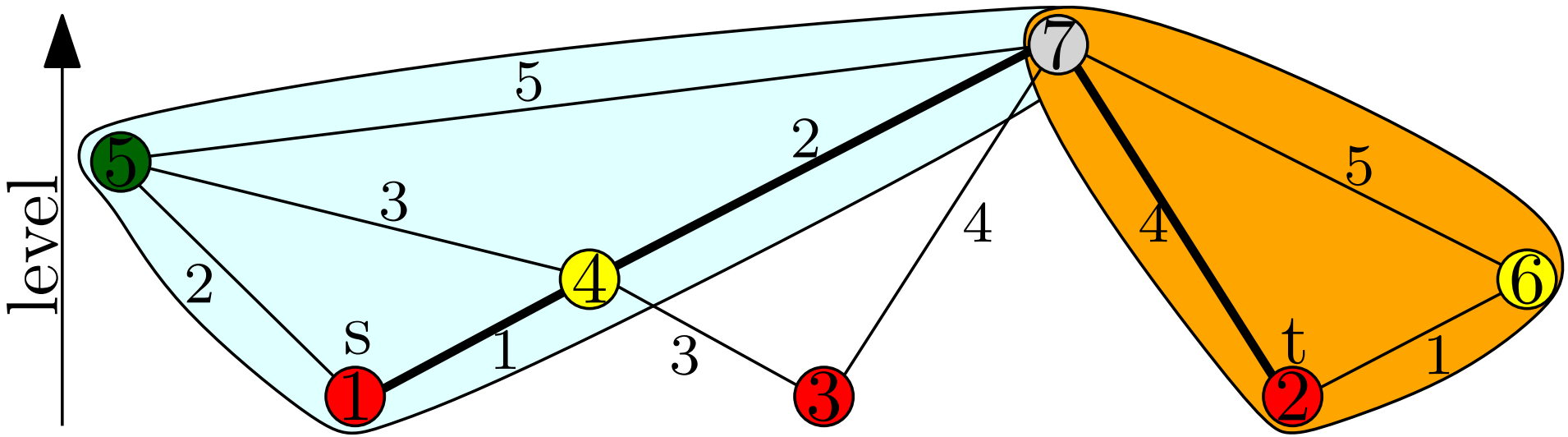
Contraction Hierarchies

Punkt-zu-Punkt Anfragen

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

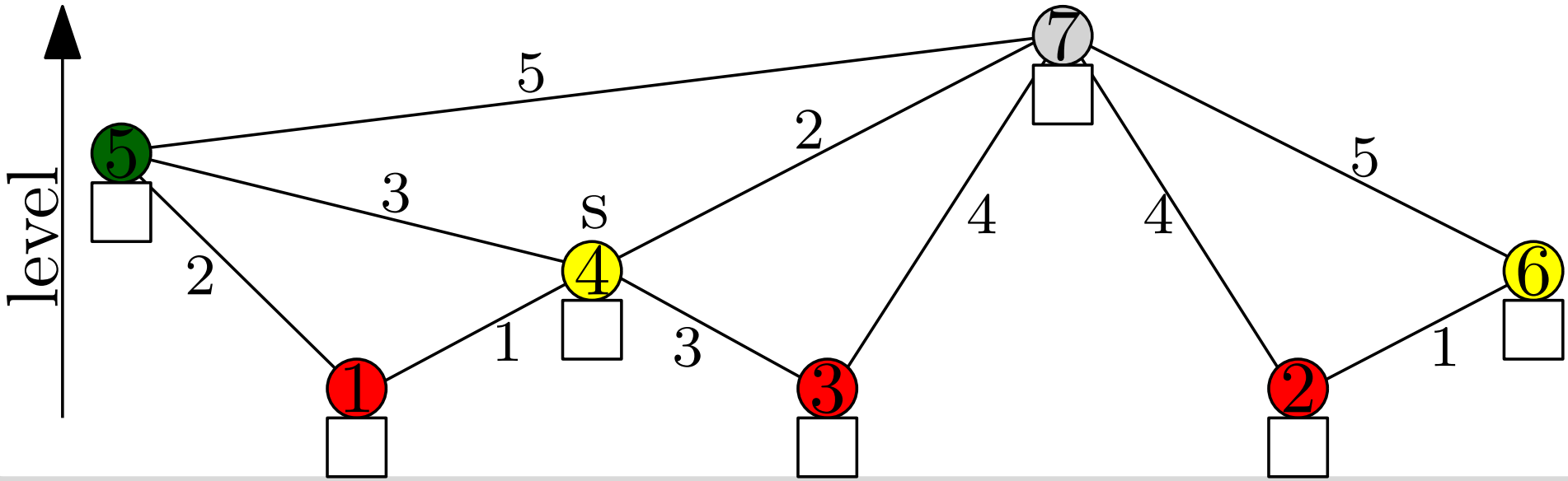
Korrektheit:

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



Neuer Anfragealgorithmus

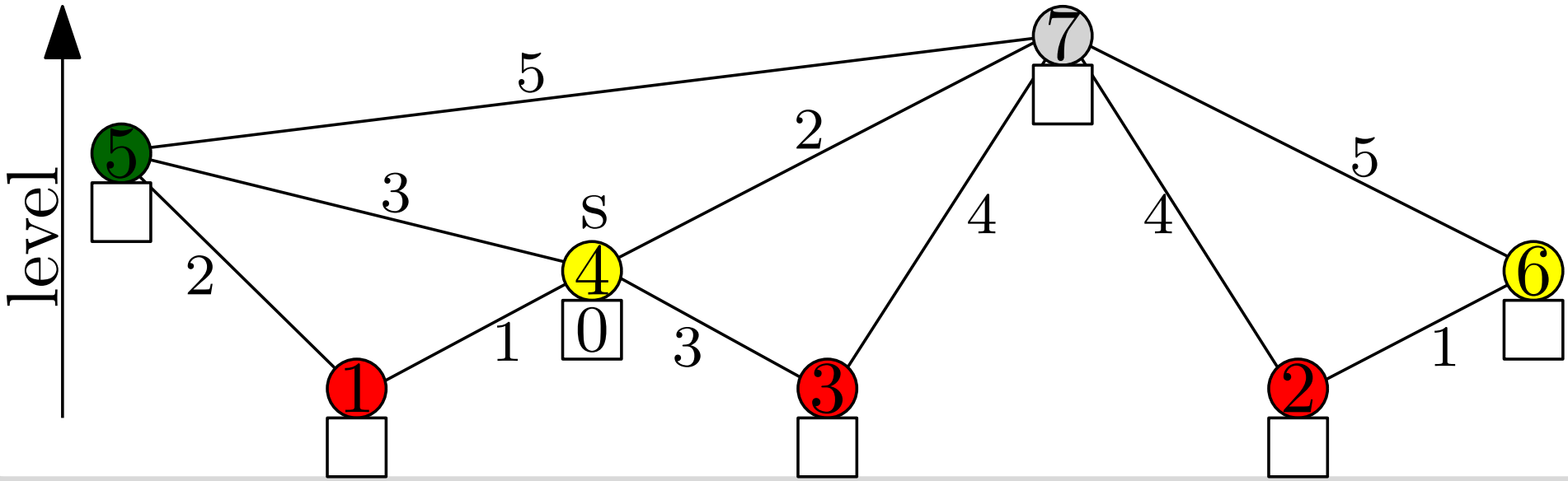
one-to-all Suche von s:



Neuer Anfragealgorithmus

one-to-all Suche von s:

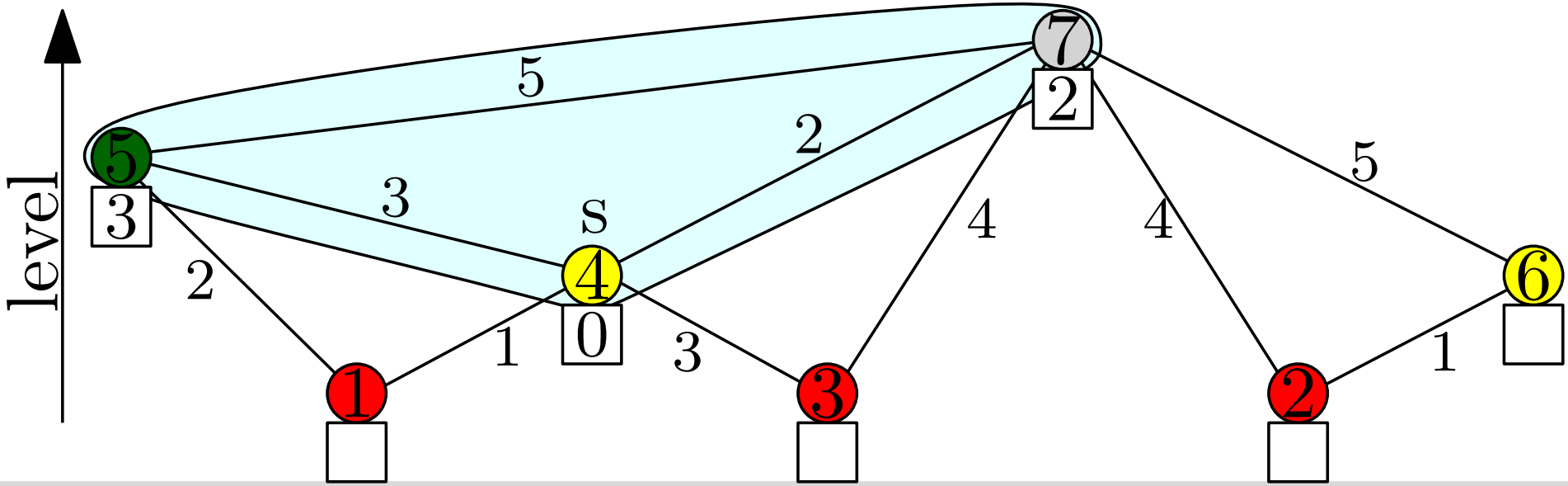
- vorwärts CH Suche von s ($\approx$ 0.05 ms)



Neuer Anfragealgorithmus

one-to-all Suche von *s*:

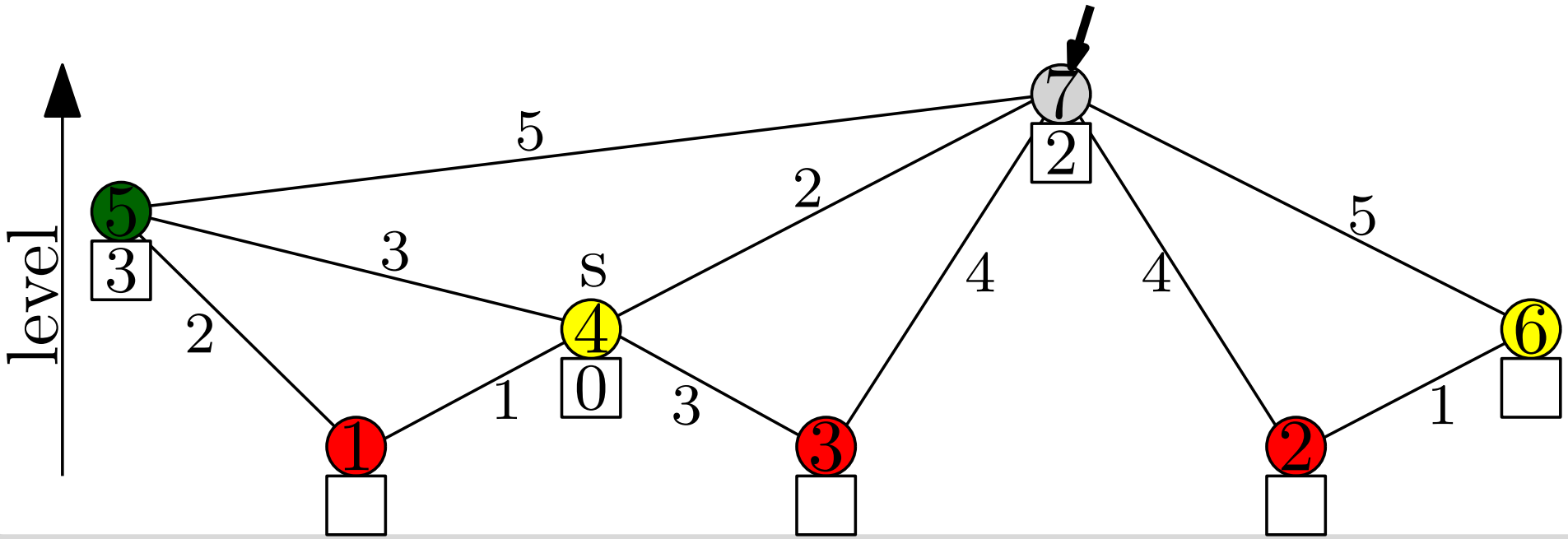
- vorwärts CH Suche von *s* ($\approx$ 0.05 ms)
- setze Distanzen *d* für alle erreichten Knoten



Neuer Anfragealgorithmus

one-to-all Suche von s:

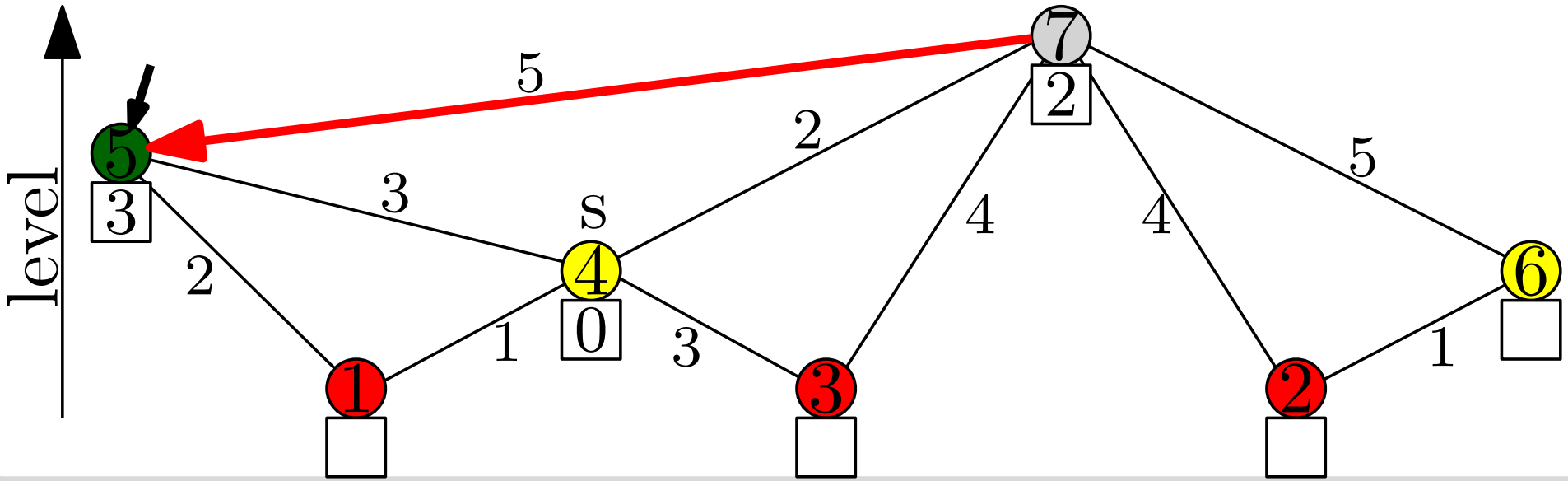
- vorwärts CH Suche von s (≈ 0.05 ms)
- setze Distanzen d für alle erreichten Knoten
- bearbeite alle Knoten u in inverser Levelordnung:
  - checke eingehende Kanten (v, u) mit $lev(v) > lev(u)$
  - setze $d(u) = \min\{d(u), d(v) + w(v, u)\}$



Neuer Anfragealgorithmus

one-to-all Suche von s:

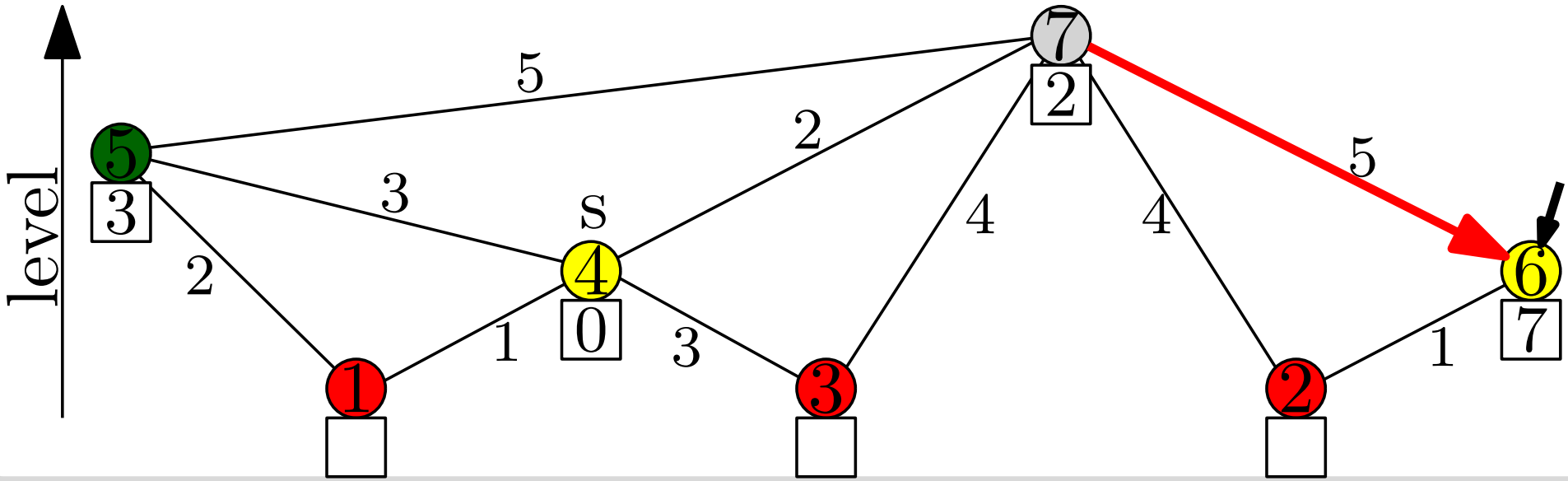
- vorwärts CH Suche von s ($\approx$ 0.05 ms)
- setze Distanzen d für alle erreichten Knoten
- bearbeite alle Knoten u in inverser Levelordnung:
  - checke eingehende Kanten (v, u) mit $lev(v) > lev(u)$
  - setze $d(u) = \min\{d(u), d(v) + w(v, u)\}$



Neuer Anfragealgorithmus

one-to-all Suche von s:

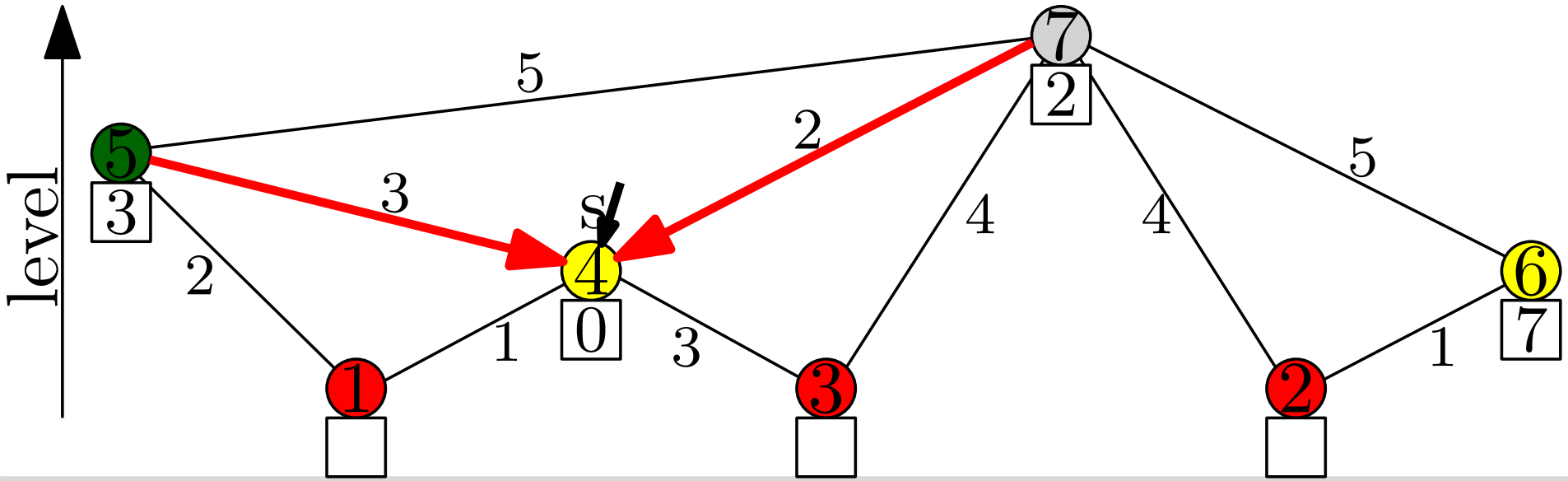
- vorwärts CH Suche von s ($\approx$ 0.05 ms)
- setze Distanzen d für alle erreichten Knoten
- bearbeite alle Knoten u in inverser Levelordnung:
  - checke eingehende Kanten (v, u) mit $lev(v) > lev(u)$
  - setze $d(u) = \min\{d(u), d(v) + w(v, u)\}$



Neuer Anfragealgorithmus

one-to-all Suche von s:

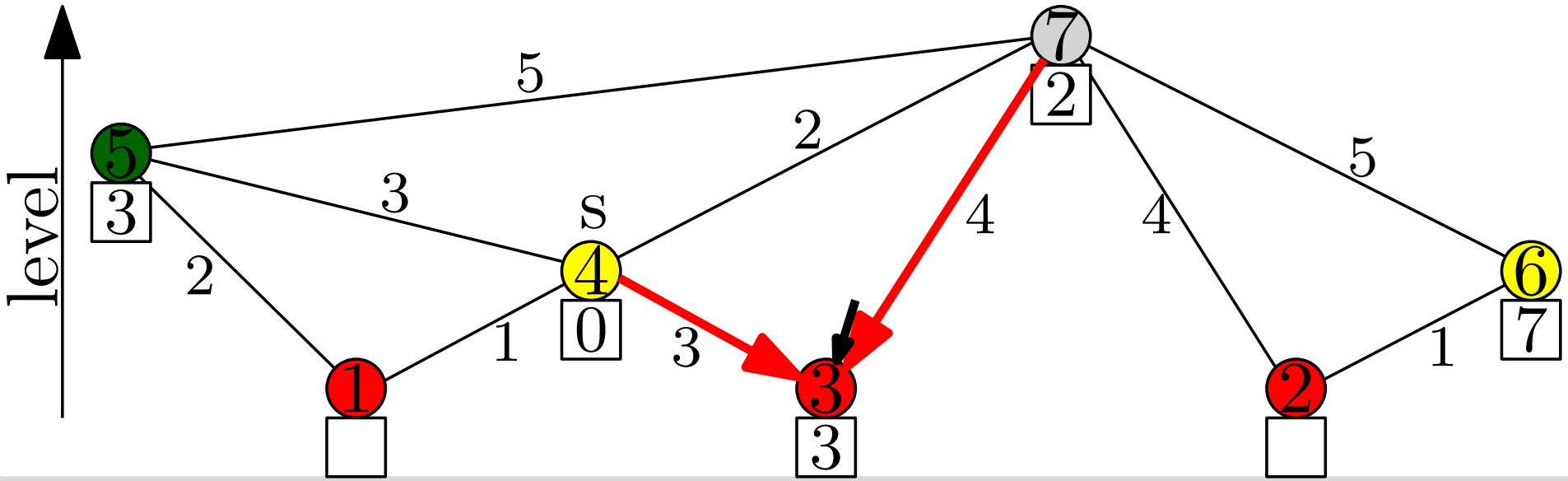
- vorwärts CH Suche von s (≈ 0.05 ms)
- setze Distanzen d für alle erreichten Knoten
- bearbeite alle Knoten u in inverser Levelordnung:
  - checke eingehende Kanten (v, u) mit $lev(v) > lev(u)$
  - setze $d(u) = \min\{d(u), d(v) + w(v, u)\}$



Neuer Anfragealgorithmus

one-to-all Suche von s:

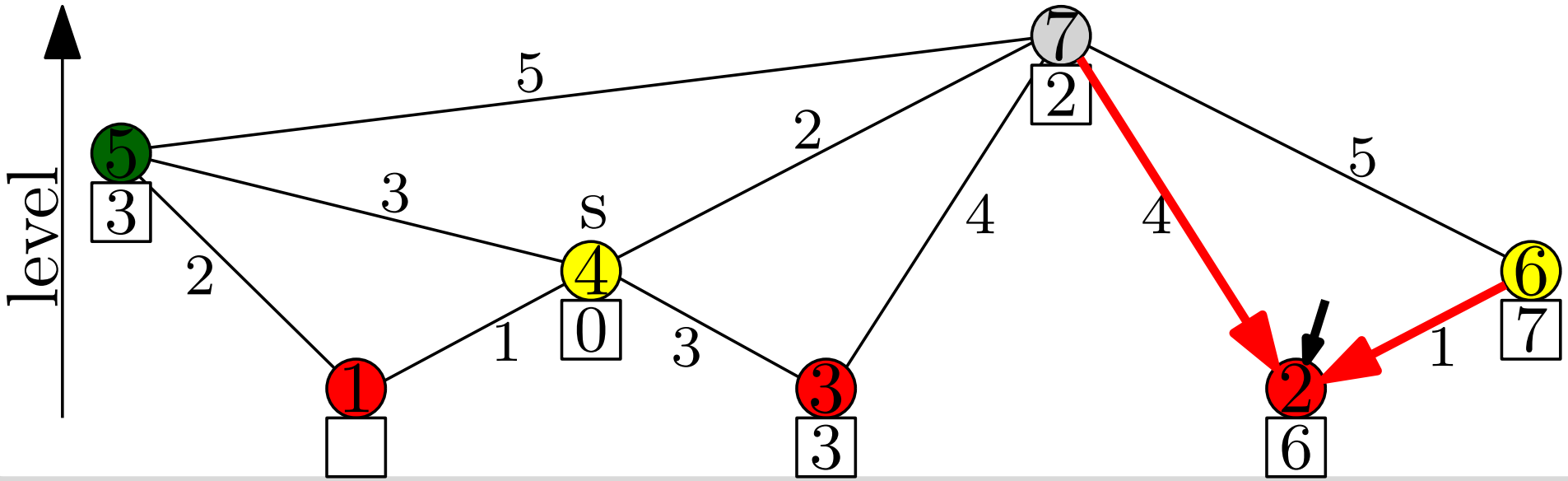
- vorwärts CH Suche von s ($\approx$ 0.05 ms)
- setze Distanzen d für alle erreichten Knoten
- bearbeite alle Knoten u in inverser Levelordnung:
  - checke eingehende Kanten (v, u) mit $lev(v) > lev(u)$
  - setze $d(u) = \min\{d(u), d(v) + w(v, u)\}$



Neuer Anfragealgorithmus

one-to-all Suche von s:

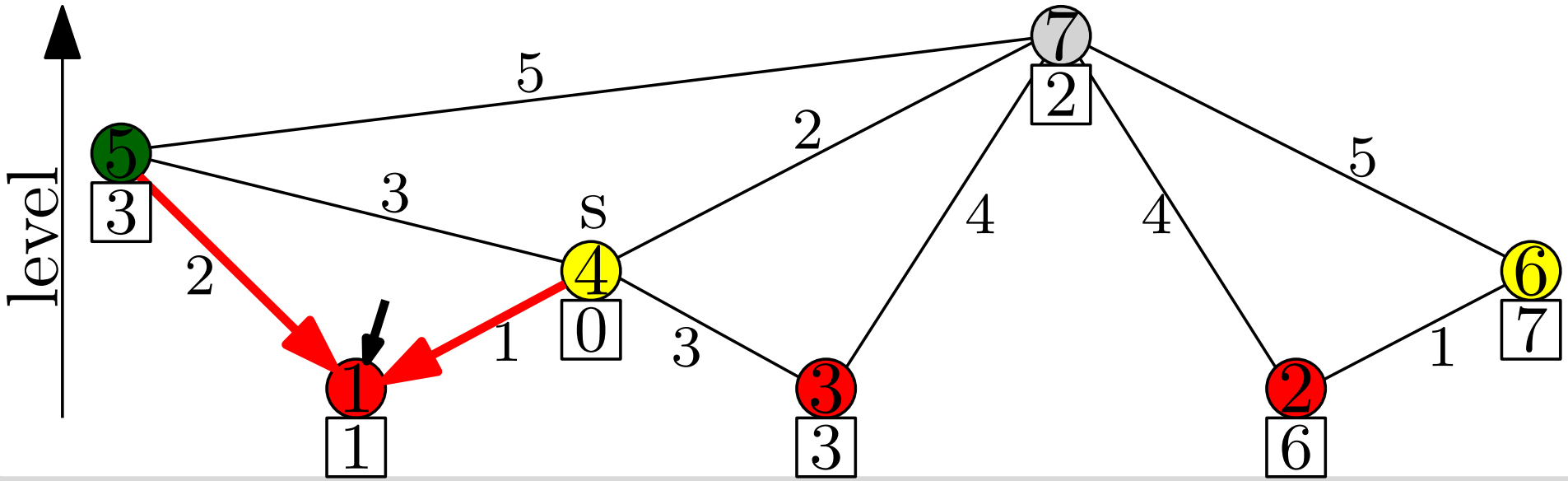
- vorwärts CH Suche von s ($\approx$ 0.05 ms)
- setze Distanzen d für alle erreichten Knoten
- bearbeite alle Knoten u in inverser Levelordnung:
  - checke eingehende Kanten (v, u) mit $lev(v) > lev(u)$
  - setze $d(u) = \min\{d(u), d(v) + w(v, u)\}$



Neuer Anfragealgorithmus

one-to-all Suche von s:

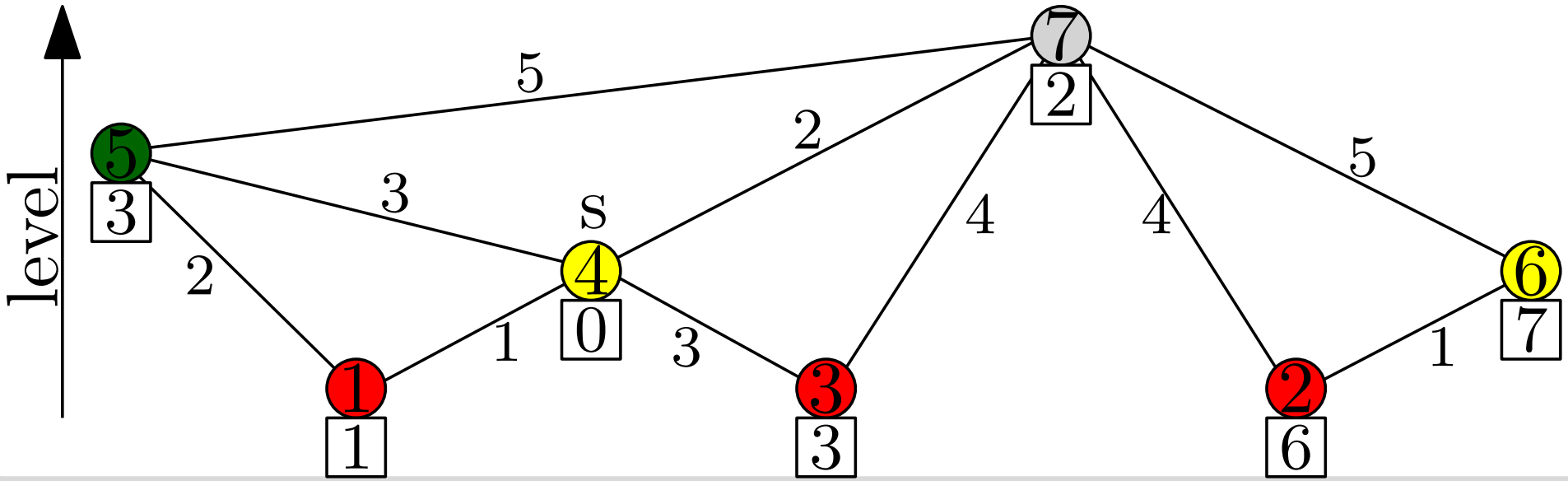
- vorwärts CH Suche von s ($\approx$ 0.05 ms)
- setze Distanzen d für alle erreichten Knoten
- bearbeite alle Knoten u in inverser Levelordnung:
  - checke eingehende Kanten (v, u) mit $lev(v) > lev(u)$
  - setze $d(u) = \min\{d(u), d(v) + w(v, u)\}$



Neuer Anfragealgorithmus

one-to-all Suche von s:

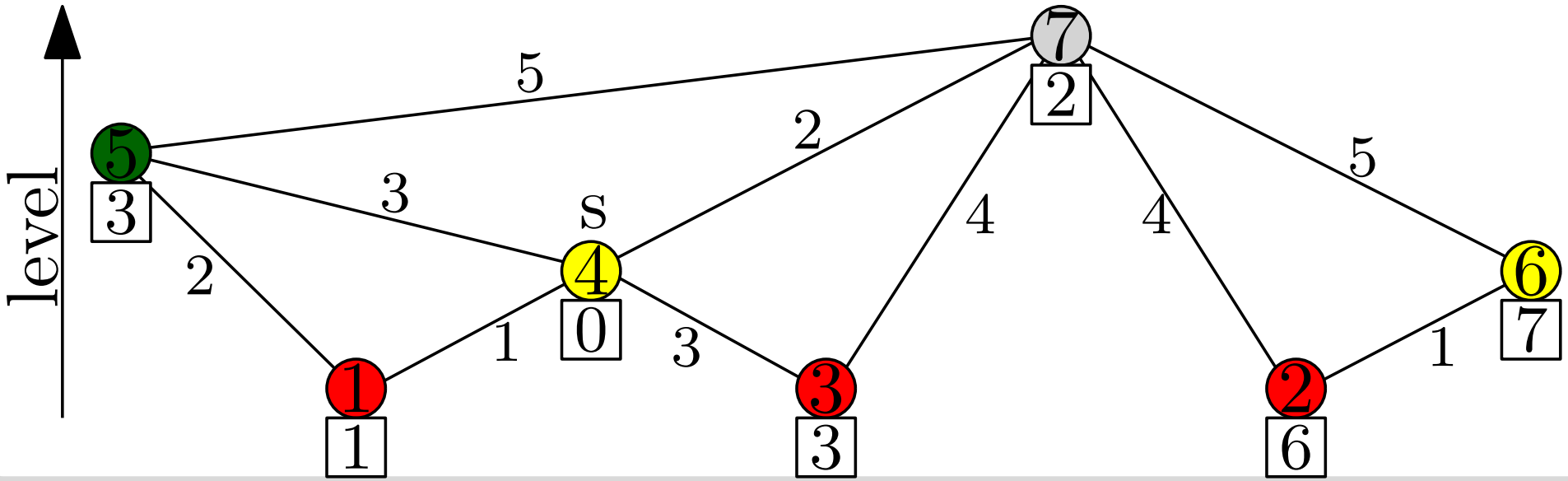
- vorwärts CH Suche von s ($\approx$ 0.05 ms)
- setze Distanzen d für alle erreichten Knoten
- bearbeite alle Knoten u in inverser Levelordnung:
  - checke eingehende Kanten (v, u) mit $lev(v) > lev(u)$
  - setze $d(u) = \min\{d(u), d(v) + w(v, u)\}$
- top-down Bearbeitung ohne Priority Queue (ca. 2.0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



Neuer Anfragealgorithmus

one-to-all Suche von s:

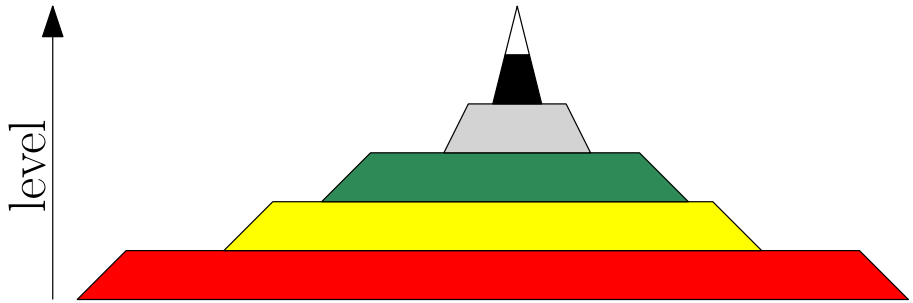
- vorwärts CH Suche von s ($\approx$ 0.05 ms)
- setze Distanzen d für alle erreichten Knoten
- bearbeite alle Knoten u in inverser Levelordnung:
  - checke eingehende Kanten (v, u) mit $lev(v) > lev(u)$
  - setze $d(u) = \min\{d(u), d(v) + w(v, u)\}$
- top-down Bearbeitung ohne Priority Queue (ca. 2.0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



Analyse

Beobachtung:

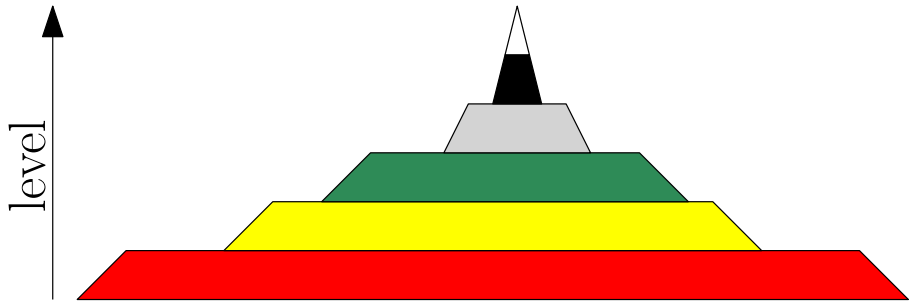
- top-down Prozess ist der Flaschenhals



Analyse

Beobachtung:

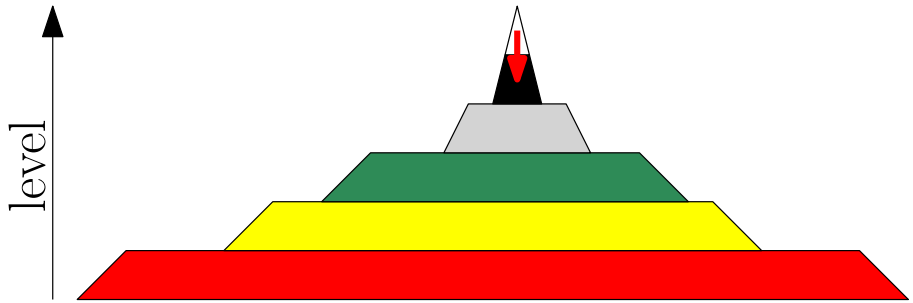
- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient

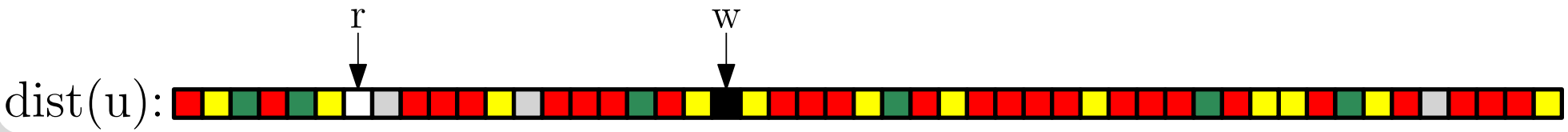


Analyse

Beobachtung:

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

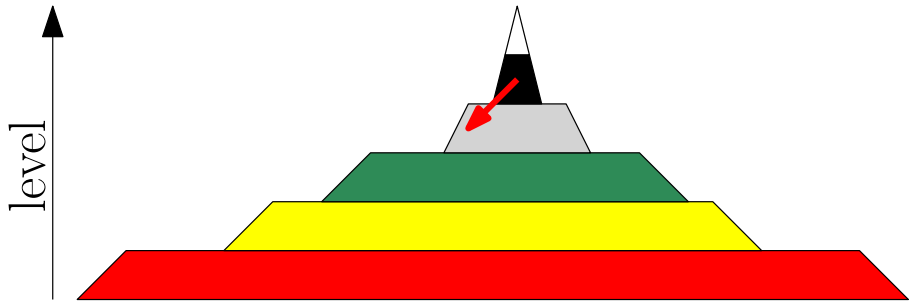


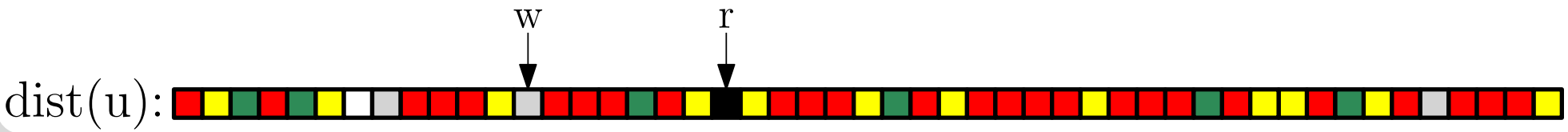


Analyse

Beobachtung:

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

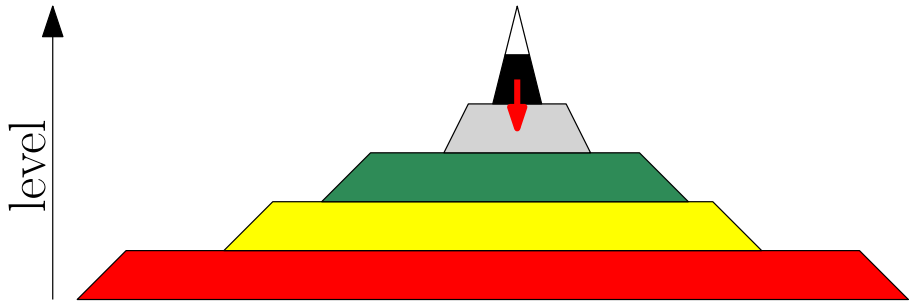


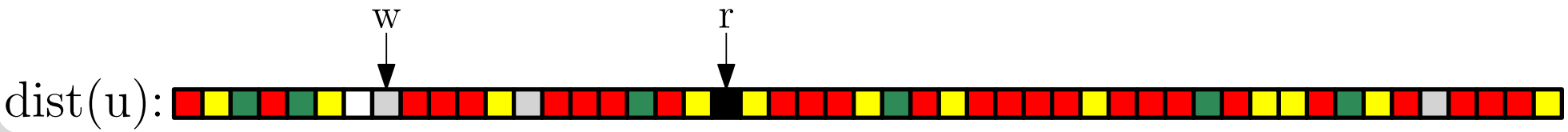


Analyse

Beobachtung:

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

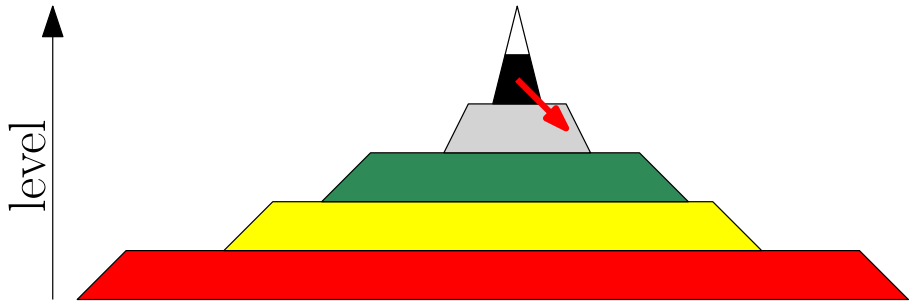


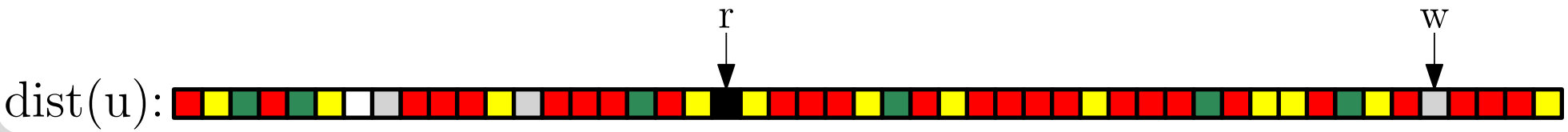


Analyse

Beobachtung:

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

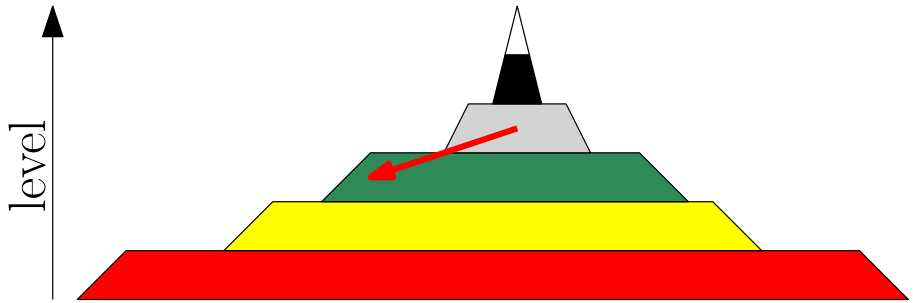




Analyse

Beobachtung:

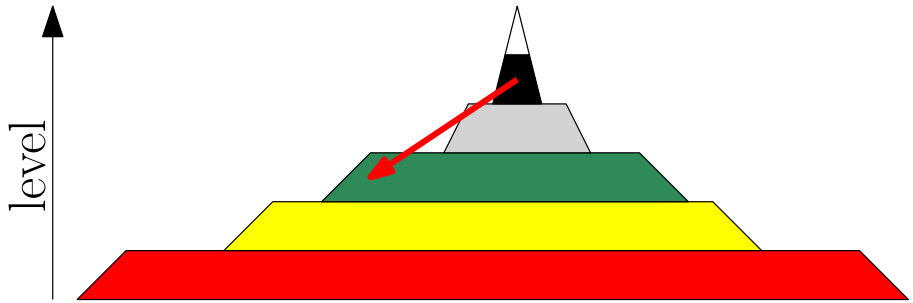
- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient

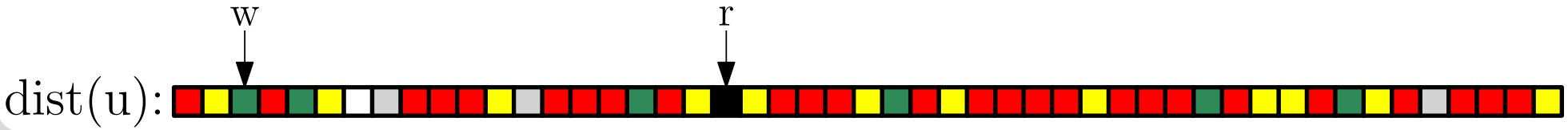


Analyse

Beobachtung:

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

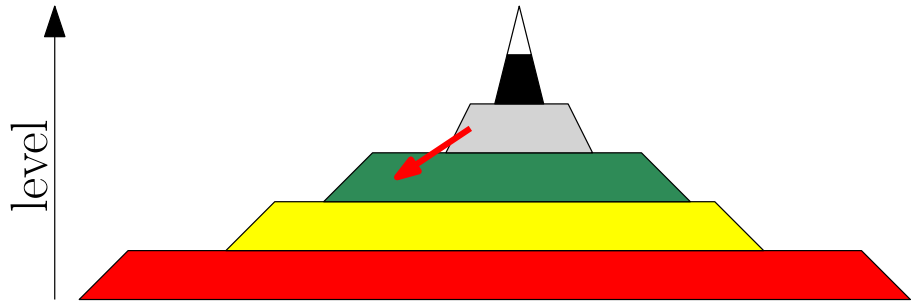




Analyse

Beobachtung:

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



Analyse

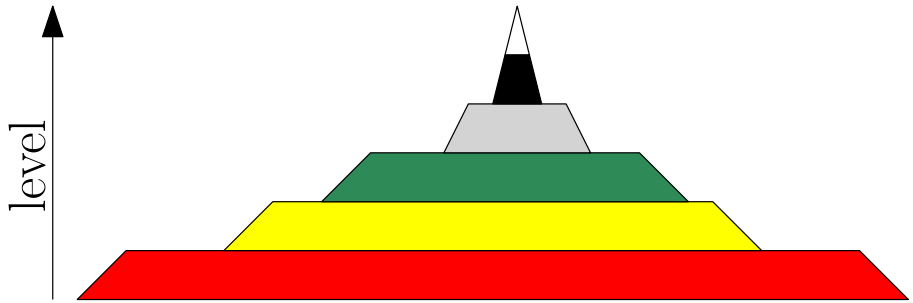
Beobachtung:

- 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

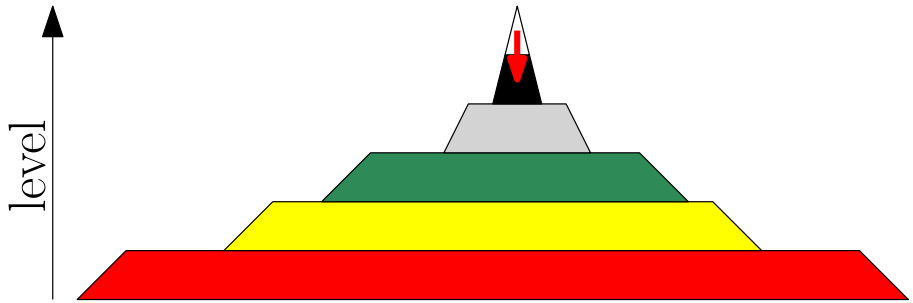
Beobachtung:

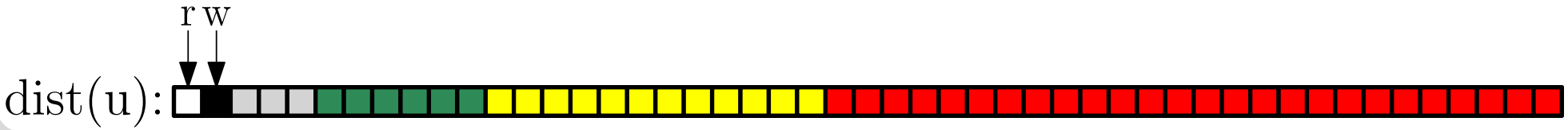
- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient
- Zugriffsmuster sind unabhängig von s

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





Analyse

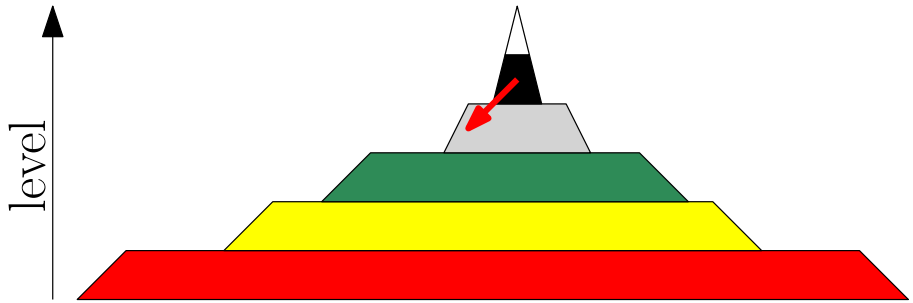
Beobachtung:

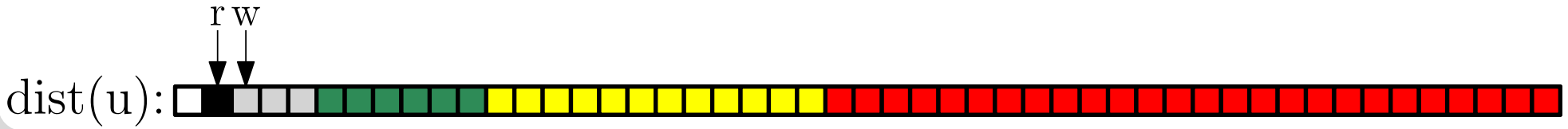
- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient
- Zugriffsmuster sind unabhängig von s

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

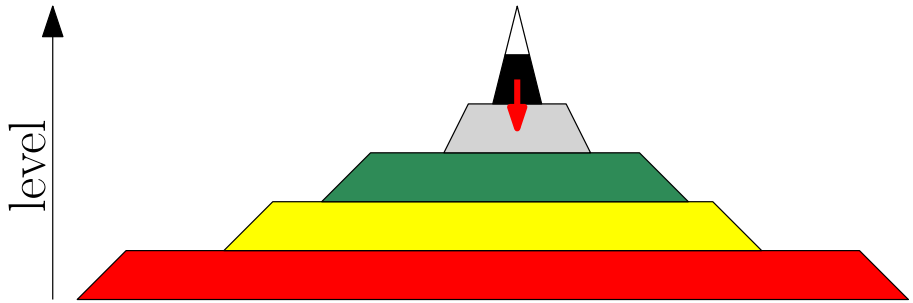
Beobachtung:

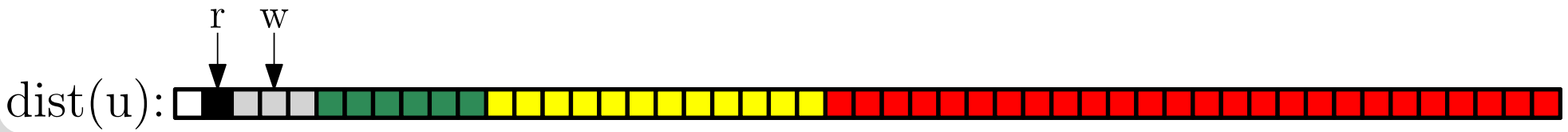
- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient
- Zugriffsmuster sind unabhängig von s

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





Analyse

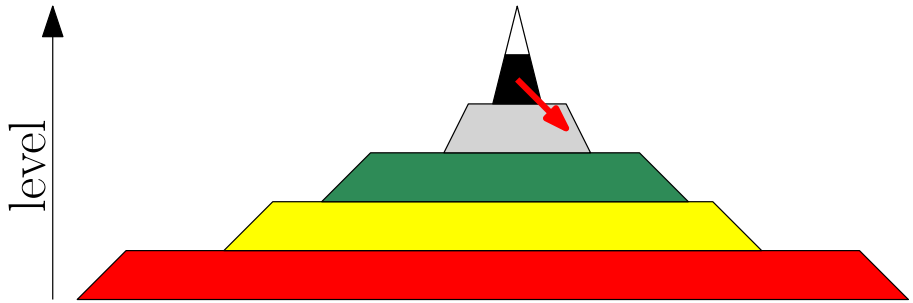
Beobachtung:

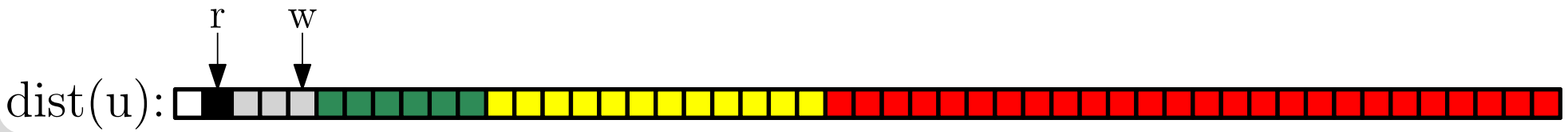
- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient
- Zugriffsmuster sind unabhängig von s

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

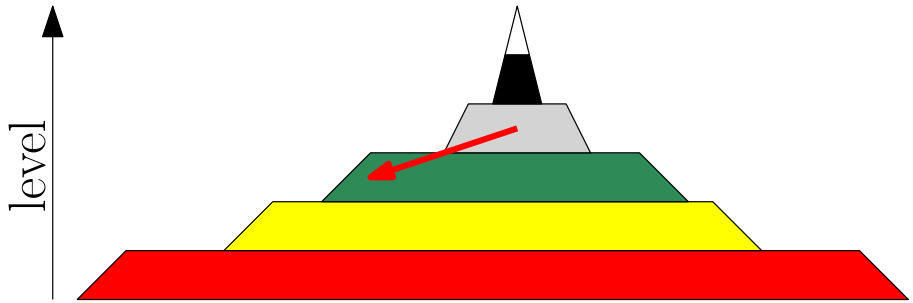
Beobachtung:

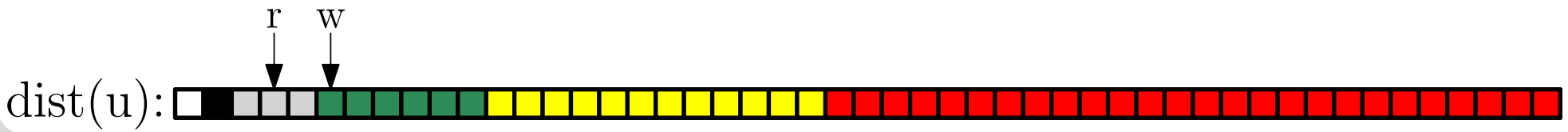
- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient
- Zugriffsmuster sind unabhängig von s

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





Analyse

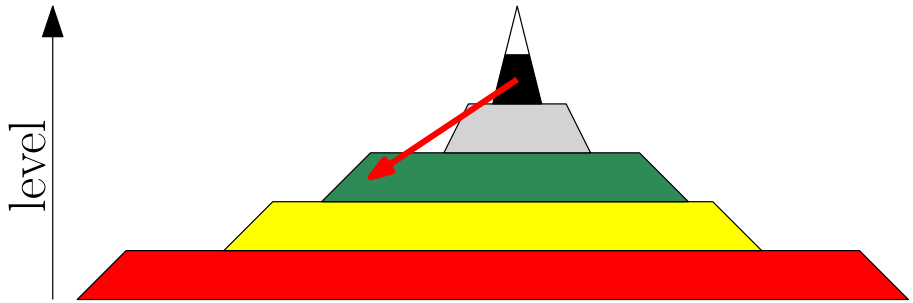
Beobachtung:

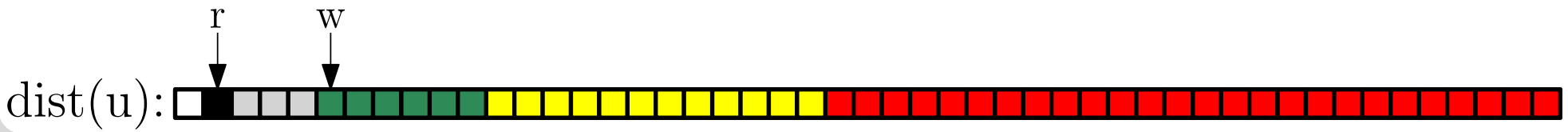
- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient
- Zugriffsmuster sind unabhängig von s

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:

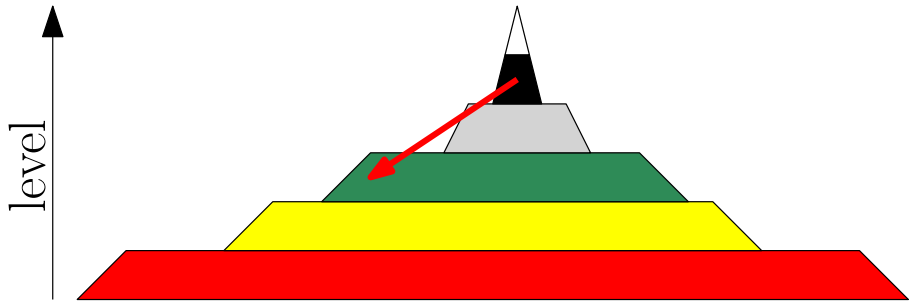
- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient
- Zugriffsmuster sind unabhängig von s

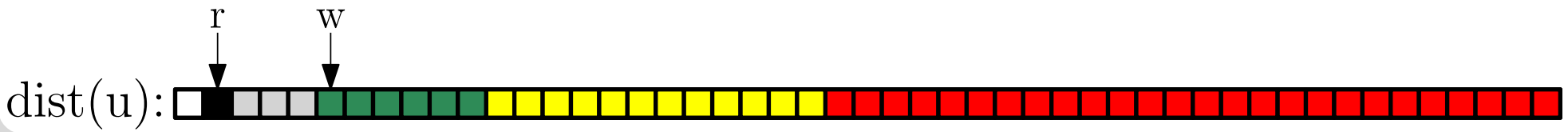
Idee:

- 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

$\Rightarrow$ 172 ms pro Baum





Analyse

Beobachtung:

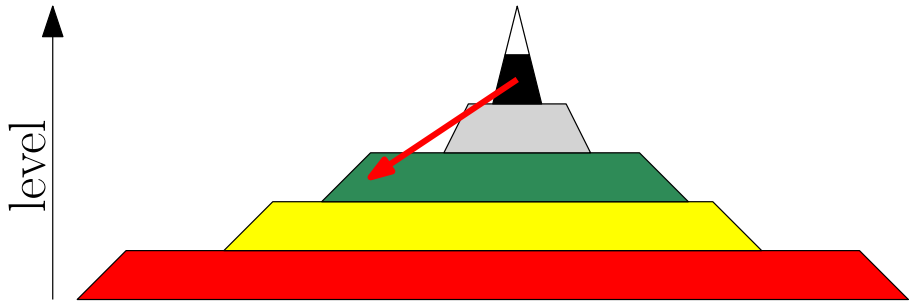
- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient
- Zugriffsmuster sind unabhängig von s

Idee:

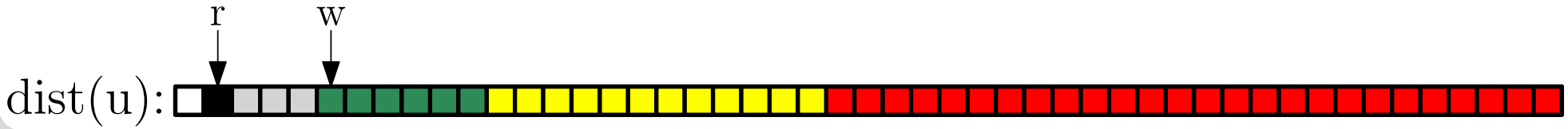
- speicher $G_\uparrow$ und $G_\downarrow$ separat
- Umordnung der Knoten, Kanten, und Distanzlabel nach Level

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

$\Rightarrow$ 172 ms pro Baum



- aber lesen der Distanzen immer noch ineffizient



GPHAST - Ideen

Beobachtungen:

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

GPHAST - Ideen

Beobachtungen:

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

Idee:

- speicher CH und Distanzarray auf der GPU
- Auswärtssuche auf der CPU
- kopiere Suchraum zur GPU (weniger als 2 kB)
- linearen Sweep auf der GPU

GPHAST - Ideen

Beobachtungen:

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

Idee:

- speicher CH und Distanzarray auf der GPU
- Auswärtssuche auf der CPU
- kopiere Suchraum zur GPU (weniger als 2 kB)
- linearen Sweep auf der GPU

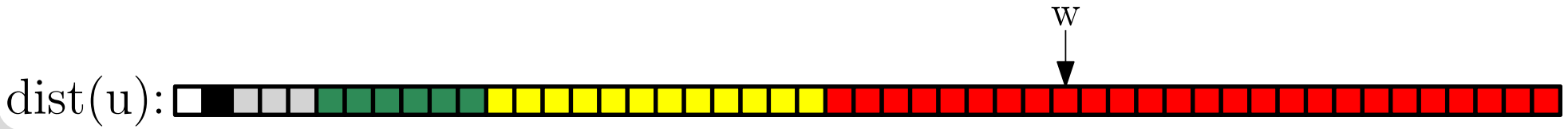
Problem:

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

Paralleler Linearer Sweep

Beobachtung:

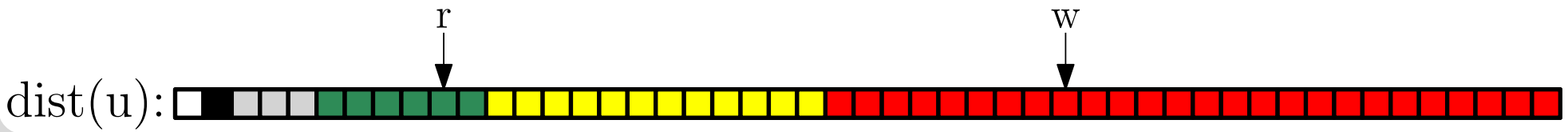
- beim Scannen von Level i:
  - nur eingehende Kanten von level $> i$ wichtig
  - schreiben von Distanzlabeln in Level i, lesen von Level $> i$
  - Distanzlabel für Level $> i$ sind korrekt
- scannen eines Level-i Knoten ist unabhängig von anderen Level-i Knoten



Paralleler Linearer Sweep

Beobachtung:

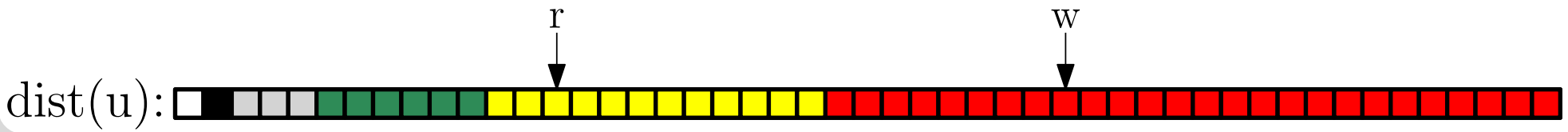
- beim Scannen von Level i:
  - nur eingehende Kanten von level $> i$ wichtig
  - schreiben von Distanzlabeln in Level i, lesen von Level $> i$
  - Distanzlabel für Level $> i$ sind korrekt
- scannen eines Level-i Knoten ist unabhängig von anderen Level-i Knoten



Paralleler Linearer Sweep

Beobachtung:

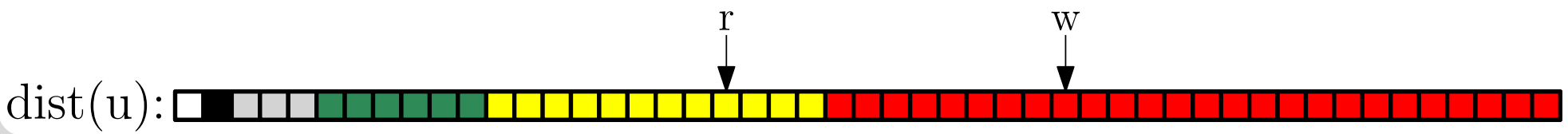
- beim Scannen von Level i:
  - nur eingehende Kanten von level $> i$ wichtig
  - schreiben von Distanzlabeln in Level i, lesen von Level $> i$
  - Distanzlabel für Level $> i$ sind korrekt
- scannen eines Level-i Knoten ist unabhängig von anderen Level-i Knoten



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



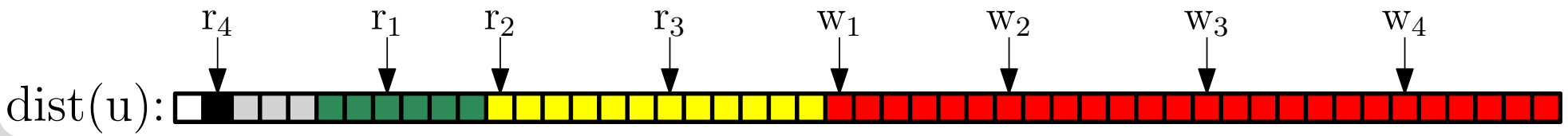
Paralleler Linearer Sweep

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



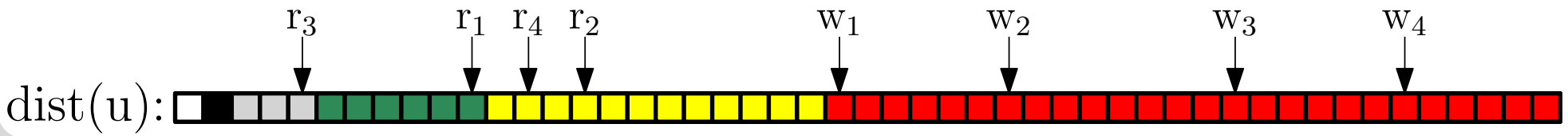
Paralleler Linearer Sweep

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:

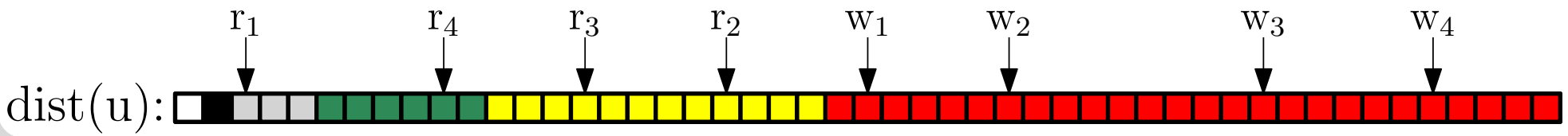
- beim Scannen von Level i:
  - nur eingehende Kanten von level $> i$ wichtig
  - schreiben von Distanzlabeln in Level i, lesen von Level $> i$
  - Distanzlabel für Level $> i$ sind korrekt
- scannen eines Level-i Knoten ist unabhängig von anderen Level-i Knoten

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:

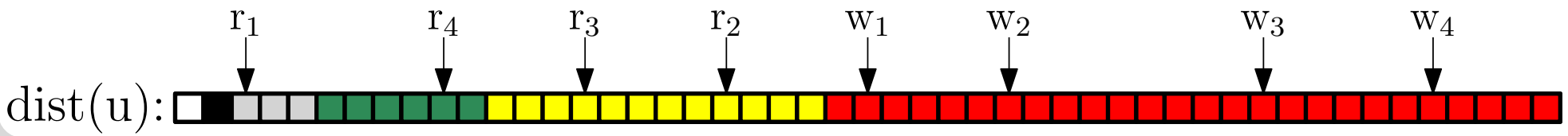
- beim Scannen von Level i:
  - nur eingehende Kanten von level $> i$ wichtig
  - schreiben von Distanzlabeln in Level i, lesen von Level $> i$
  - Distanzlabel für Level $> i$ sind korrekt
- scannen eines Level-i Knoten ist unabhängig von anderen Level-i Knoten

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

weitere Eingaben

		Europe		USA	
algorithm	device	time	distance	time	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
- APSP in 11h!

Baumrekonstruktion

bis jetzt:

- nur Distanzen berechnet, nicht Bäume

Baumrekonstruktion

bis jetzt:

- nur Distanzen berechnet, nicht Bäume

Idee:

- iteration über alle Kanten
- setze $p(u) = v$ wenn $d(v) + \text{len}(v, u) = d(u)$

Baumrekonstruktion

bis jetzt:

- nur Distanzen berechnet, nicht Bäume

Idee:

- iteration über alle Kanten
- setze $p(u) = v$ wenn $d(v) + \text{len}(v, u) = d(u)$
- speicher Kanten als Triples $(u, v, \text{len}(u, v))$
- ein Thread pro Kante

Baumrekonstruktion

bis jetzt:

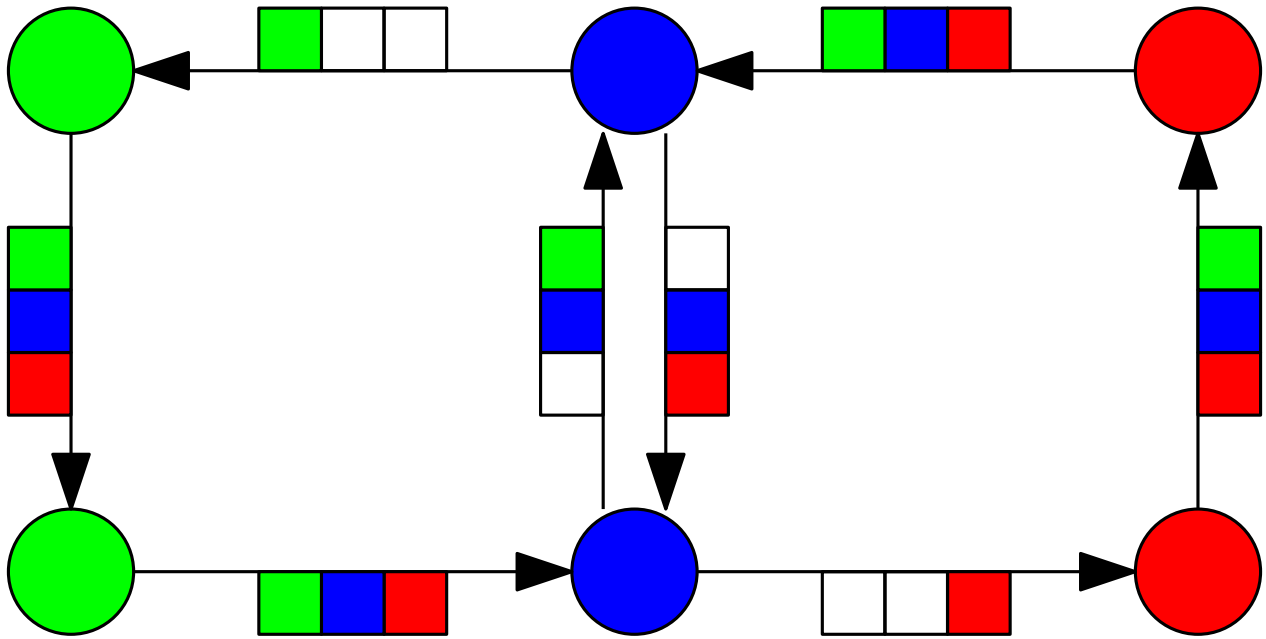
- nur Distanzen berechnet, nicht Bäume

Idee:

- iteration über alle Kanten
- setze $p(u) = v$ wenn $d(v) + \text{len}(v, u) = d(u)$
- speicher Kanten als Triples $(u, v, \text{len}(u, v))$
- ein Thread pro Kante
- ein *linearer* Sweep über den Graphen
- erhöht Berechnungszeit um einen Faktor 2

Arc-Flags

Idee:

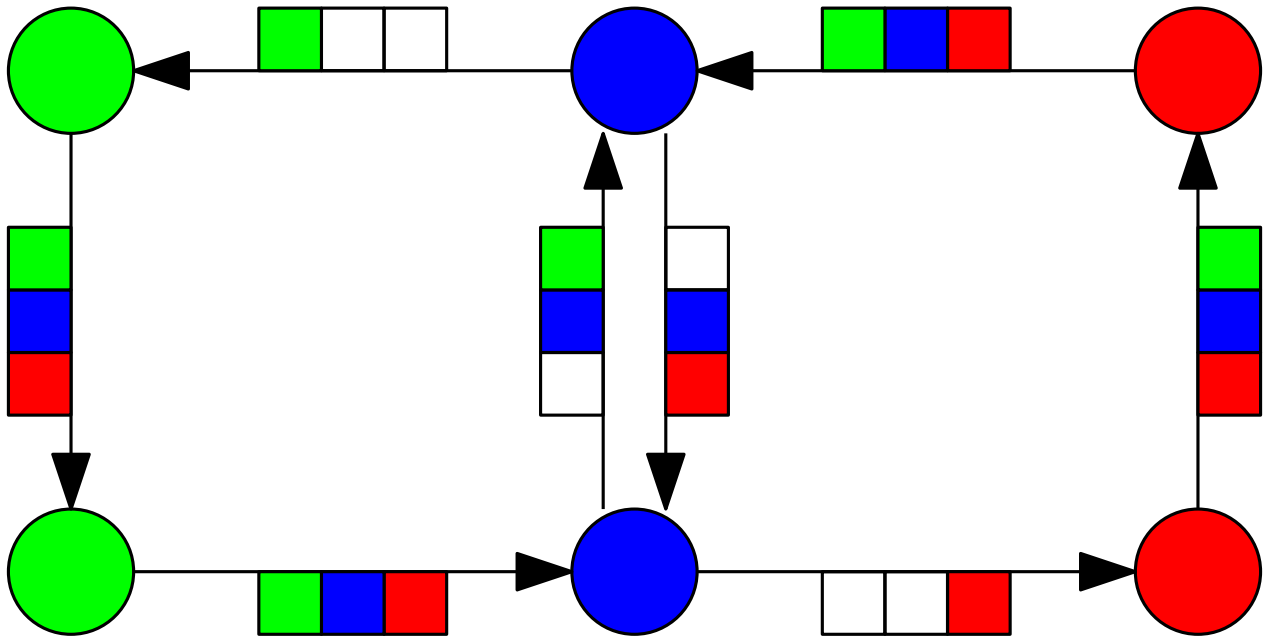


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Arc-Flags

Idee:

- setze 1-shell Kanten automatisch

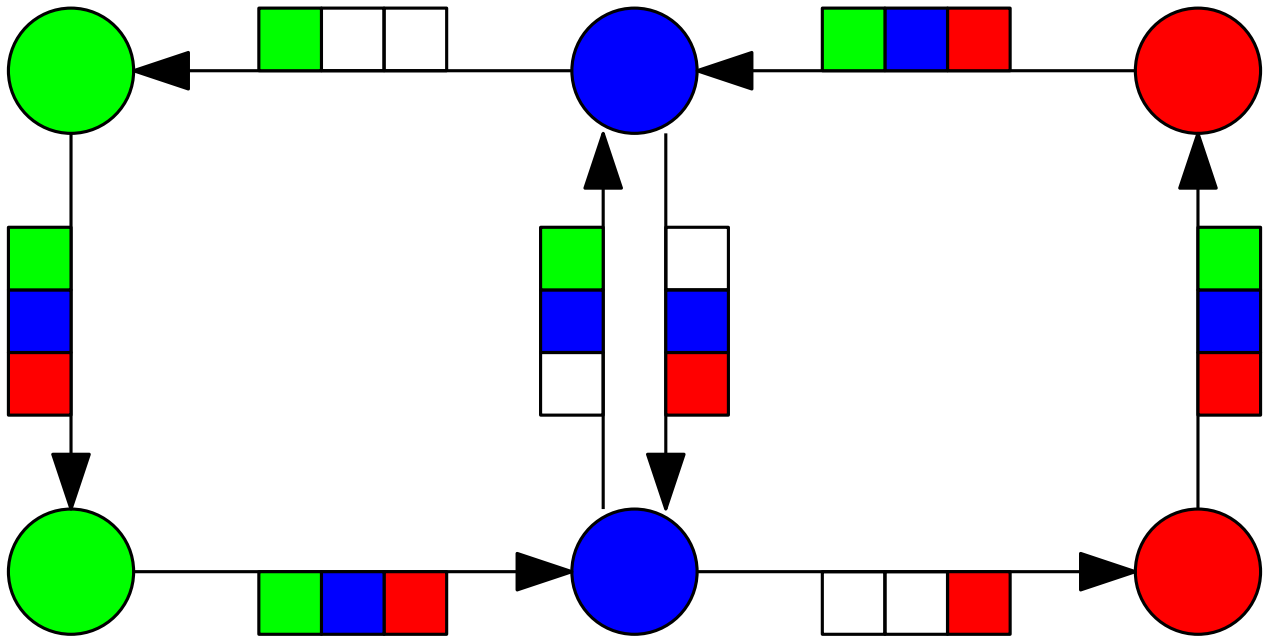


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Arc-Flags

Idee:

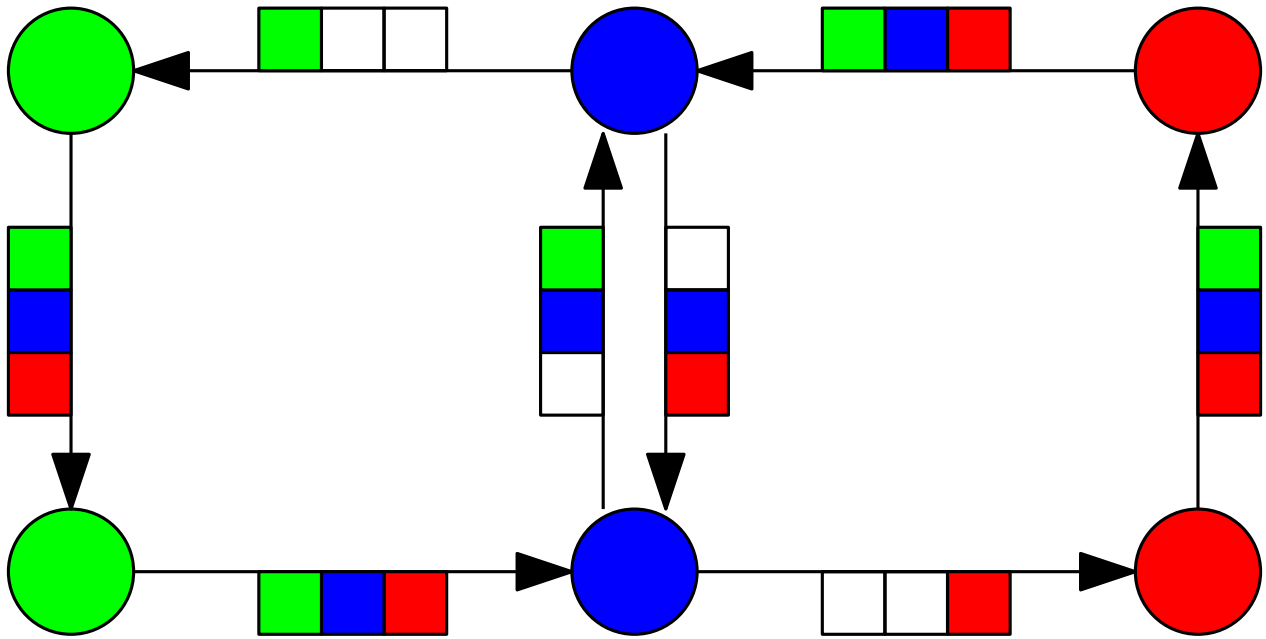
- setze 1-shell Kanten automatisch
- benutze GPHAST zum Berechnen der Bäume



Arc-Flags

Idee:

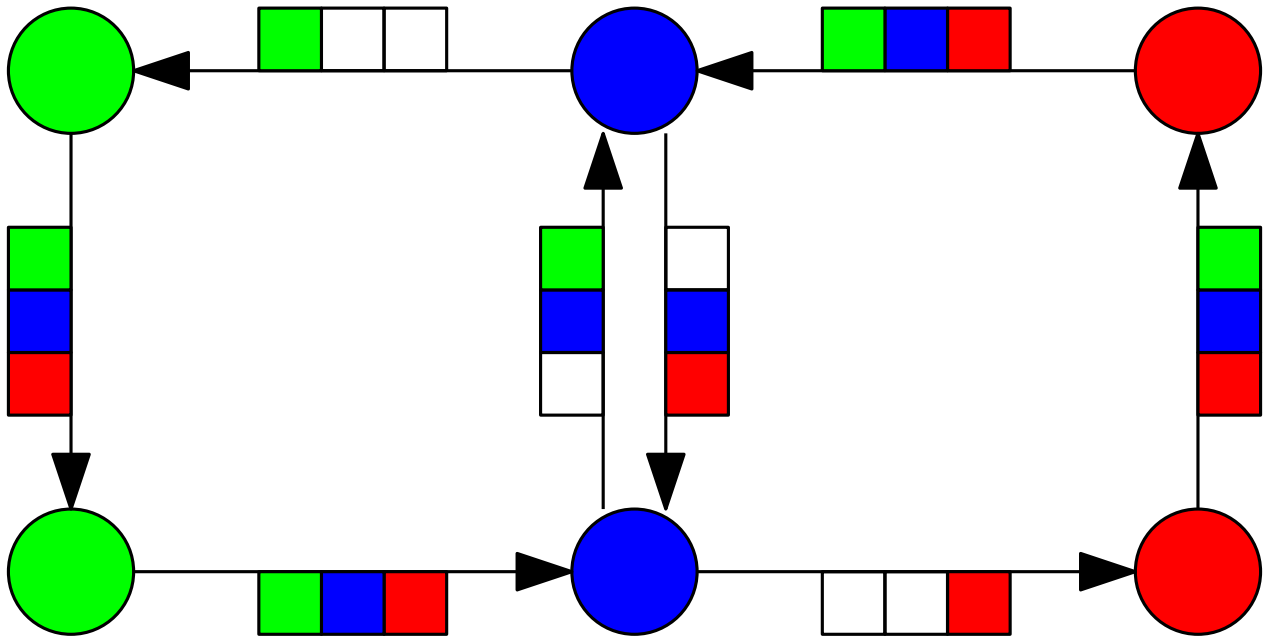
- setze 1-shell Kanten automatisch
- benutze GPHAST zum Berechnen der Bäume
- setze Flaggen durch zusätzlichen Sweep auf GPU



Arc-Flags

Idee:

- setze 1-shell Kanten automatisch
- benutze GPHAST zum Berechnen der Bäume
- setze Flaggen durch zusätzlichen Sweep auf GPU
- Vorberechnung sinkt von 17 Stunden auf 3 Minuten
- ohne Partitionierung (2 Minuten mit PUNCH) und CH Vorberechnung (2 Minuten)



CHASE

Eigenschaften

- Setzen von Flaggen nur für wichtige Kanten (5% wichtigsten Knoten)
- Flaschenhals: Baumberechnung

CHASE

Eigenschaften

- Setzen von Flaggen nur für wichtige Kanten (5% wichtigsten Knoten)
- Flaschenhals: Baumberechnung

Idee:

- benutze GPHAST zum Berechnen von Bäumen
- setze Flaggen für alle Kanten
- verzögere Kontraktion von Original-Randknoten

CHASE

Eigenschaften

- Setzen von Flaggen nur für wichtige Kanten (5% wichtigsten Knoten)
- Flaschenhals: Baumberechnung

Idee:

- benutze GPHAST zum Berechnen von Bäumen
- setze Flaggen für alle Kanten
- verzögere Kontraktion von Original-Randknoten

$\Rightarrow$ durch Kontraktion erhöht sich Anzahl Randknoten nicht zu sehr

CHASE

Eigenschaften

- Setzen von Flaggen nur für wichtige Kanten (5% wichtigsten Knoten)
- Flaschenhals: Baumberechnung

Idee:

- benutze GPHAST zum Berechnen von Bäumen
- setze Flaggen für alle Kanten
- verzögere Kontraktion von Original-Randknoten

$\Rightarrow$ durch Kontraktion erhöht sich Anzahl Randknoten nicht zu sehr

$\Rightarrow$ Anzahl Randknoten: 22k (11k in G)

CHASE

Eigenschaften

- Setzen von Flaggen nur für wichtige Kanten (5% wichtigsten Knoten)
- Flaschenhals: Baumberechnung

Idee:

- benutze GPHAST zum Berechnen von Bäumen
- setze Flaggen für alle Kanten
- verzögere Kontraktion von Original-Randknoten

$\Rightarrow$ durch Kontraktion erhöht sich Anzahl Randknoten nicht zu sehr

$\Rightarrow$ Anzahl Randknoten: 22k (11k in G)

$\Rightarrow$ 15 Minuten Vorberechnung, 28 gescannte Knoten, 5.4 μs Anfragzeit

Hybrid GPHAST

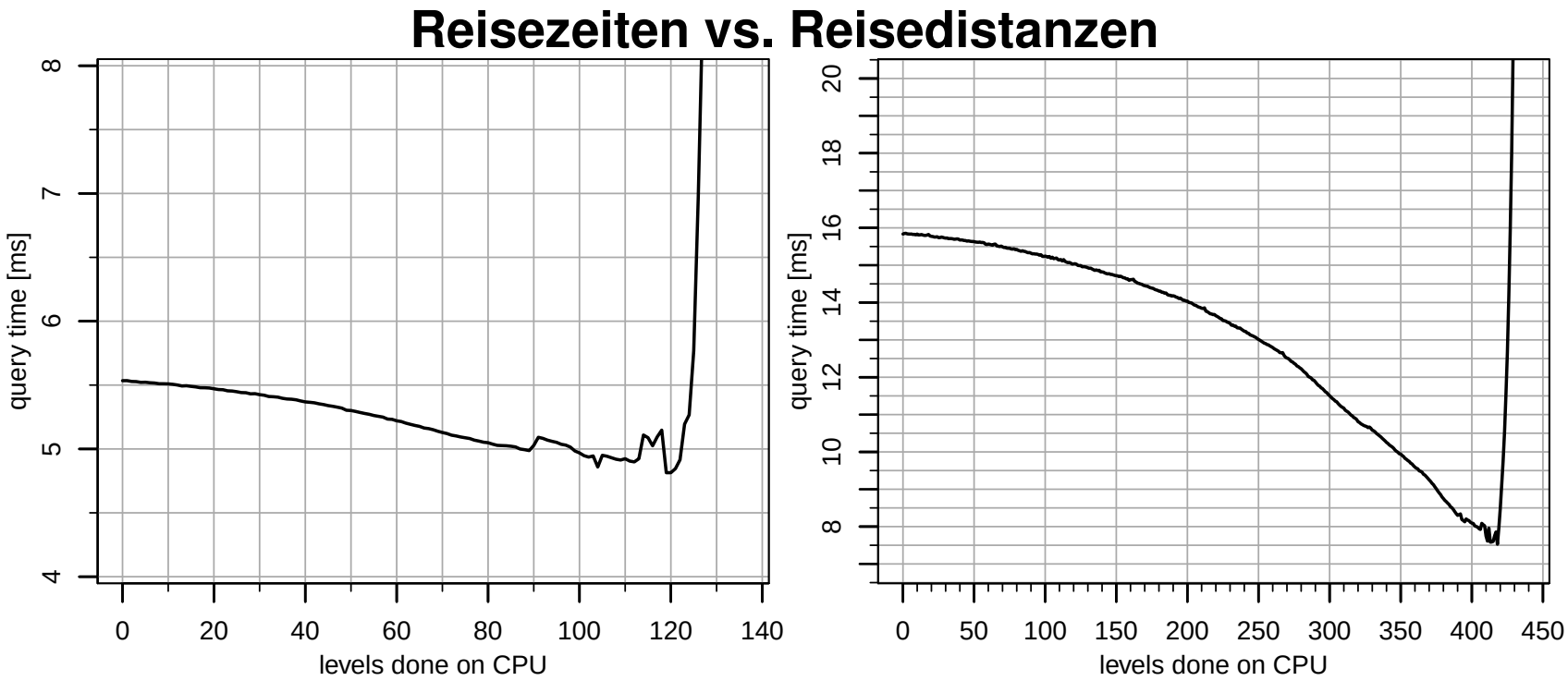
Beobachtung:

- 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



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

Zusammenfassung

- 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)
- erlaub APSP-basierte Berechnungen
- 150 mal Energie-effizienter als Dijkstras Algorithmus
- funktioniert nur, wenn CH funktioniert

Literatur

Literatur:

- Daniel Delling, Andrew V. Goldberg, Andreas Nowatzyk, Renato F. Werneck
  PHAST: Hardware-Accelerated Shortest path Trees
  In: *Journal of Parallel and Distributed Computing*, 2012

Anfrageszenarien

Punkt-zu-Punkt

- zwei Punkte $\rightarrow$ kürzester Weg
- wird für Routenplanung benutzt
- Beschleunigungstechniken
- HubLabels 10Mx schneller

Anfrageszenarien

Punkt-zu-Punkt

- zwei Punkte $\rightarrow$ kürzester Weg
- wird für Routenplanung benutzt
- Beschleunigungstechniken
- HubLabels 10Mx schneller

One-to-All

- ein Knoten $\rightarrow$ Distanzen zu allen Knoten
- wird für Vorberechnung benutzt
- PHAST 500x schneller (auf GPU)
- nutzt Hardware aus

Anfrageszenarien

Punkt-zu-Punkt

- zwei Punkte $\rightarrow$ kürzester Weg
- wird für Routenplanung benutzt
- Beschleunigungstechniken
- HubLabels 10Mx schneller

One-to-Many

- ein (variierender Knoten) und eine (feste) Menge $\rightarrow$ Distanz zu allen Knoten in der Menge
- wichtig für POI

One-to-All

- ein Knoten $\rightarrow$ Distanzen zu allen Knoten
- wird für Vorberechnung benutzt
- PHAST 500x schneller (auf GPU)
- nutzt Hardware aus

Anfrageszenarien

Punkt-zu-Punkt

- zwei Punkte $\rightarrow$ kürzester Weg
- wird für Routenplanung benutzt
- Beschleunigungstechniken
- HubLabels 10Mx schneller

One-to-All

- ein Knoten $\rightarrow$ Distanzen zu allen Knoten
- wird für Vorberechnung benutzt
- PHAST 500x schneller (auf GPU)
- nutzt Hardware aus

One-to-Many

- ein (variierender Knoten) und eine (feste) Menge $\rightarrow$ Distanz zu allen Knoten in der Menge
- wichtig für POI

Many-to-Many

- zwei Mengen $\rightarrow$ Distanztabelle
- wichtig für Vehicle Routing

Das One-to-Many Problem

Problem Definition:

- Eingabe: eine Knoten s und eine Menge T
- Ausgabe: Distanz von s zu allen $t \in T$
- Annahme: wir fixieren T und variieren s

Das One-to-Many Problem

Problem Definition:

- Eingabe: eine Knoten s und eine Menge T
- Ausgabe: Distanz von s zu allen $t \in T$
- Annahme: wir fixieren T und variieren s

offensichtliche Lösungen:

- Dijkstras Algorithmus (mit Stoppkriterium)
  - $\Rightarrow$ Performance stark abhängig von $|T|$ und Verteilung von T

Das One-to-Many Problem

Problem Definition:

- Eingabe: eine Knoten s und eine Menge T
- Ausgabe: Distanz von s zu allen $t \in T$
- Annahme: wir fixieren T und variieren s

offensichtliche Lösungen:

- Dijkstras Algorithmus (mit Stoppkriterium)
  $\Rightarrow$ Performance stark abhängig von $|T|$ und Verteilung von T
- $|T|$ p2p Anfragen (z.B. HL)
  $\Rightarrow$ Performance stark abhängig von $|T|$

Das One-to-Many Problem

Problem Definition:

- Eingabe: eine Knoten s und eine Menge T
- Ausgabe: Distanz von s zu allen $t \in T$
- Annahme: wir fixieren T und variieren s

offensichtliche Lösungen:

- Dijkstras Algorithmus (mit Stoppkriterium)
  $\Rightarrow$ Performance stark abhängig von $|T|$ und Verteilung von T
- $|T|$ p2p Anfragen (z.B. HL)
  $\Rightarrow$ Performance stark abhängig von $|T|$
- benutze PHAST (kein Stoppkriterium!)
  $\Rightarrow$ Overkill (vor allem für kleine T)

Erste Ideen

Vorschläge?

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Ungerichtete Bidir. Techniken

Definition:

- $\overrightarrow{\sigma}(s,t)$: Suchraum der Vorwärtssuche von s nach t
- $\overleftarrow{\sigma}(s,t)$ analog
- eine bidirektionale Suche ist Ziel-unabhängig, gdw.

$$\forall (s,t_1,t_2) \in V^3 : \overrightarrow{\sigma}(s,t_1) = \overrightarrow{\sigma}(s,t_2) \quad \text{und}$$
$$\forall (s_1,s_2,t) \in V^3 : \overleftarrow{\sigma}(s_1,t) = \overleftarrow{\sigma}(s_2,t)$$

Ungerichtete Bidir. Techniken

Definition:

- $\overrightarrow{\sigma}(s,t)$: Suchraum der Vorwärtssuche von s nach t
- $\overleftarrow{\sigma}(s,t)$ analog
- eine bidirektionale Suche ist Ziel-unabhängig, gdw.

$$\forall (s,t_1,t_2) \in V^3 : \overrightarrow{\sigma}(s,t_1) = \overrightarrow{\sigma}(s,t_2) \quad \text{und}$$
$$\forall (s_1,s_2,t) \in V^3 : \overleftarrow{\sigma}(s_1,t) = \overleftarrow{\sigma}(s_2,t)$$

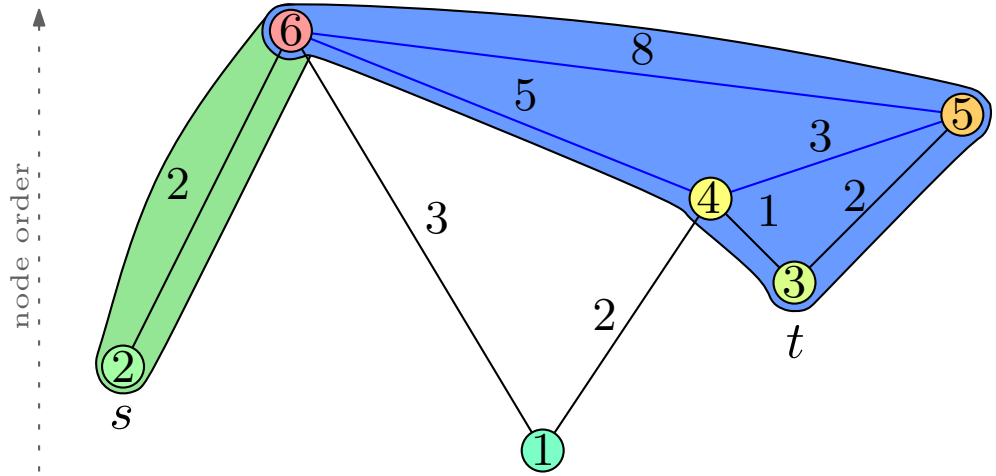
Beispiele:

- Bidirektionaler Dijkstra
- ohne Stoppkriterium, lass laufen bis Queues leer sind

Hierarchische Techniken

Beobachtung:

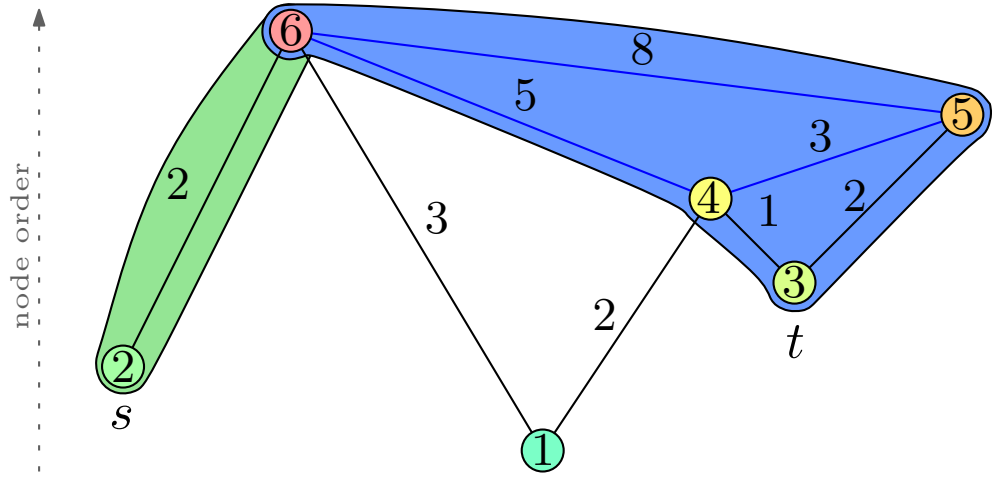
- suchen nur aufwärts
- sind nicht zielgerichtet



Hierarchische Techniken

Beobachtung:

- suchen nur aufwärts
- sind nicht zielgerichtet



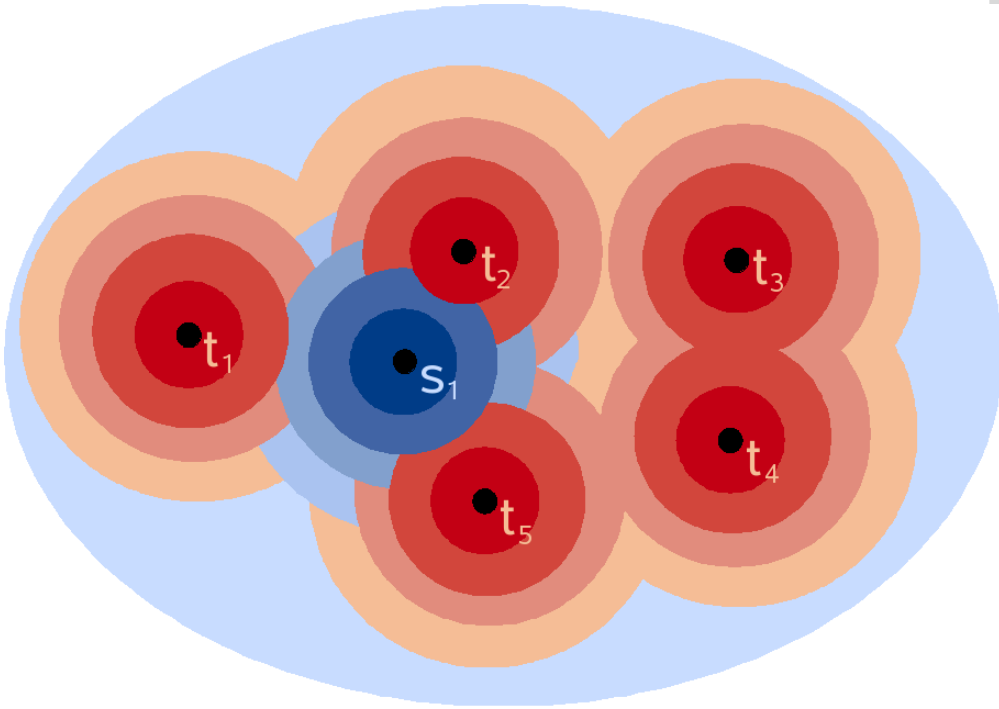
somit:

- Bidirektionaler Dijkstra
- Reach
- Contraction Hierarchies
- ohne Stoppkriterium, lass laufen bis Queues leer sind
- HL

Vorgehen

Idee:

- führe $|T|$ Rückwärtssuchen aus
- speicher für jedes besuchte u Abstände zu allen $t \in T$
- verwalte temporäres Distanzarray D_T
- führe Vorwärtssuche aus
- aktualisiere Einträge in D_T



Problem:

- verwalten der Suchräume?

Schneiden der Suchräume

während Rückwärtssuchen:

- breche nicht ab
- für jedes erreichte u:
  - füge Element $(u, t, d(u, t))$ in einen Vektor ein

Kompression:

- sortiere Einträge nach u
- speichere für jeden Knoten u einen Bucket $\beta(u)$ mit allen $(t, d(u, t))$ ab
- mittels Adjazenzarray

während Vorwärtssuche:

- breche nicht ab
- für jedes erreichte u:
  - durchsuche Bucket $\beta(u)$
  - aktualisiere Distanzarray

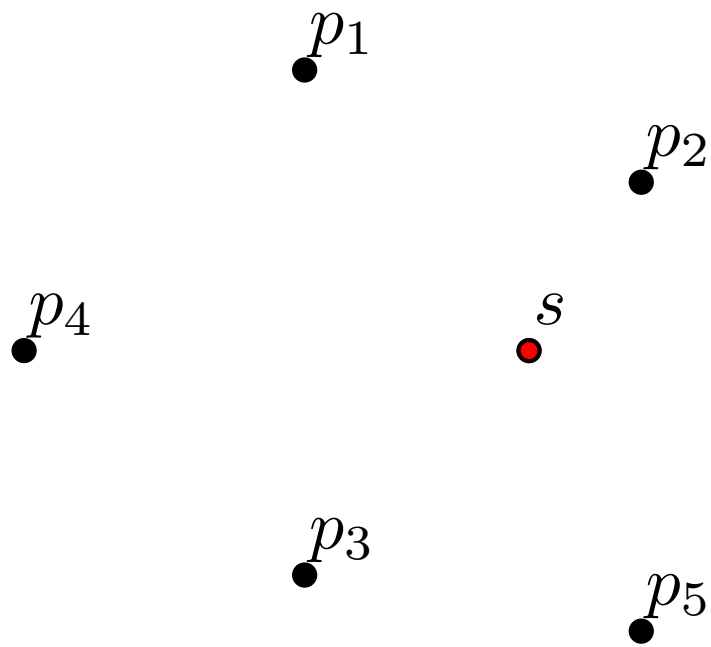
Schneiden der Suchräume

während Rückwärtssuchen:

- breche nicht ab
- für jedes erreichte u:
  - füge Element $(u, t, d(u, t))$ in einen Vektor ein

Kompression:

- sortiere Einträge nach u
- speichere für jeden Knoten u einen Bucket $\beta(u)$ mit allen $(t, d(u, t))$ ab
- mittels Adjazenzarray



während Vorwärtssuche:

- breche nicht ab
- für jedes erreichte u:
  - durchsuche Bucket $\beta(u)$
  - aktualisiere Distanzarray

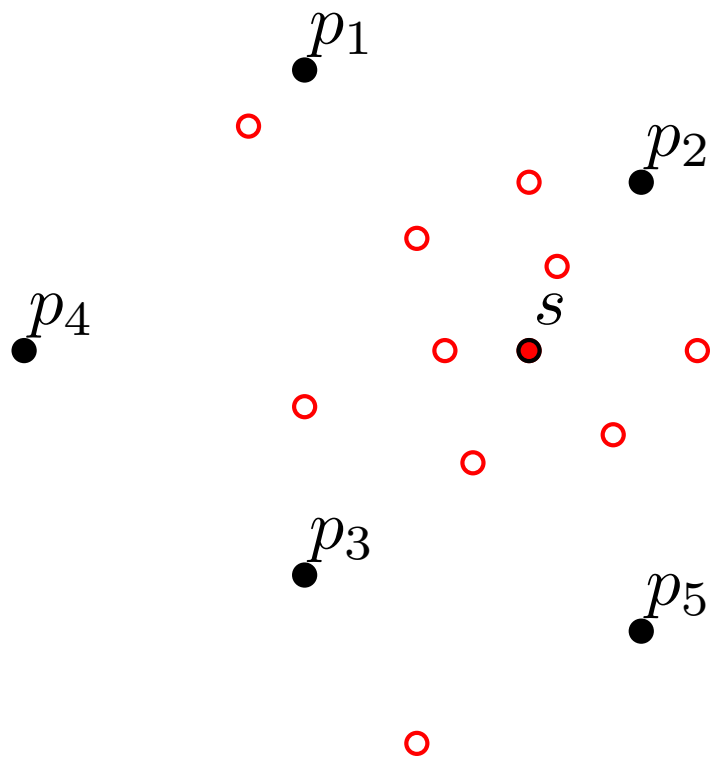
Schneiden der Suchräume

während Rückwärtssuchen:

- breche nicht ab
- für jedes erreichte u:
  - füge Element $(u, t, d(u, t))$ in einen Vektor ein

Kompression:

- sortiere Einträge nach u
- speichere für jeden Knoten u einen Bucket $\beta(u)$ mit allen $(t, d(u, t))$ ab
- mittels Adjazenzarray



während Vorwärtssuche:

- breche nicht ab
- für jedes erreichte u:
  - durchsuche Bucket $\beta(u)$
  - aktualisiere Distanzarray

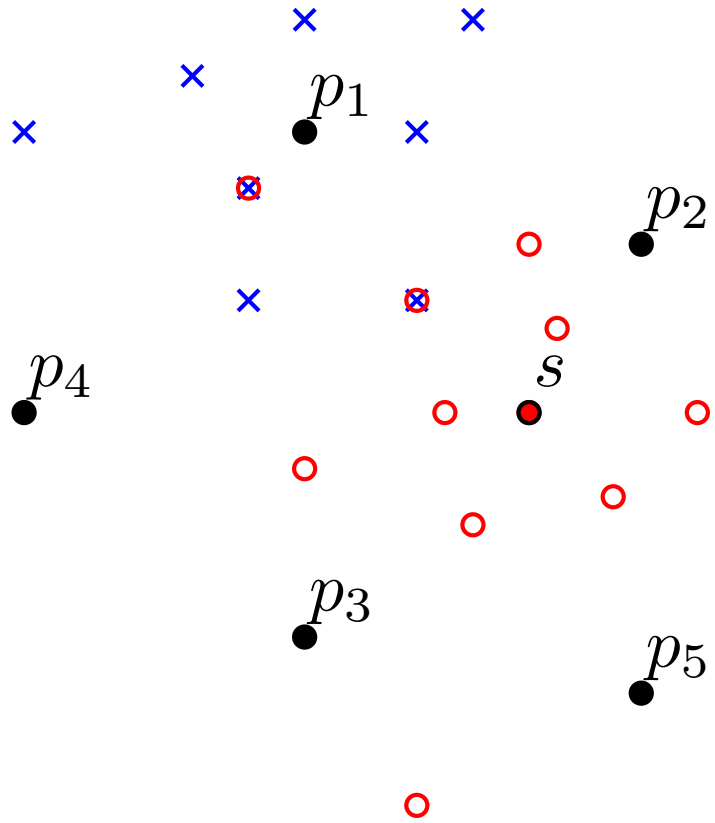
Schneiden der Suchräume

während Rückwärtssuchen:

- breche nicht ab
- für jedes erreichte u:
  - füge Element $(u, t, d(u, t))$ in einen Vektor ein

Kompression:

- sortiere Einträge nach u
- speichere für jeden Knoten u einen Bucket $\beta(u)$ mit allen $(t, d(u, t))$ ab
- mittels Adjazenzarray



während Vorwärtssuche:

- breche nicht ab
- für jedes erreichte u:
  - durchsuche Bucket $\beta(u)$
  - aktualisiere Distanzarray

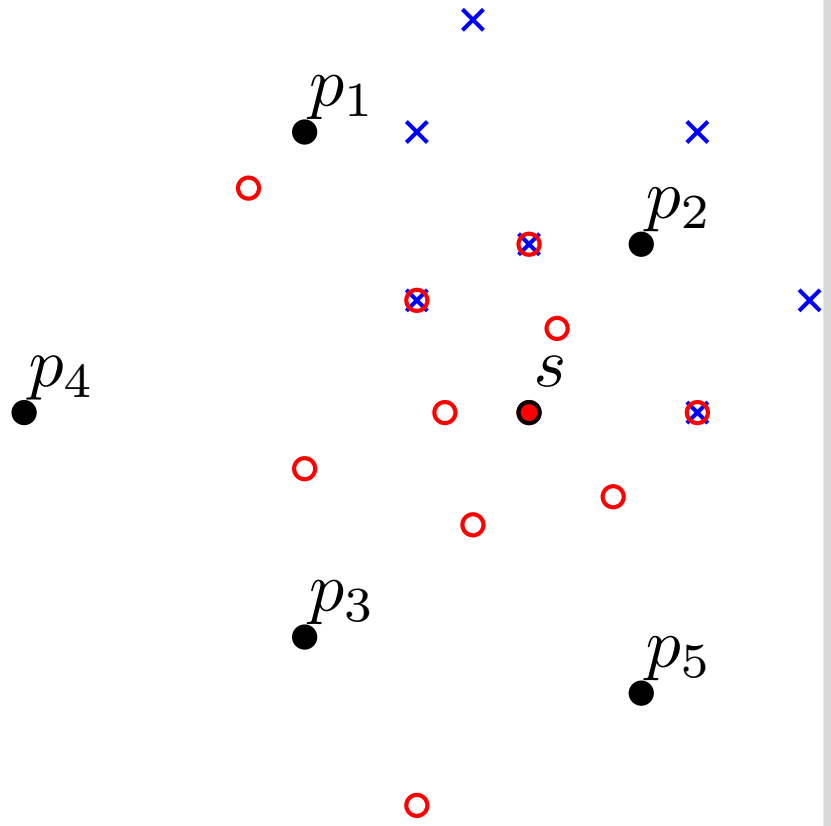
Schneiden der Suchräume

während Rückwärtssuchen:

- breche nicht ab
- für jedes erreichte u:
  - füge Element $(u, t, d(u, t))$ in einen Vektor ein

Kompression:

- sortiere Einträge nach u
- speichere für jeden Knoten u einen Bucket $\beta(u)$ mit allen $(t, d(u, t))$ ab
- mittels Adjazenzarray



während Vorwärtssuche:

- breche nicht ab
- für jedes erreichte u:
  - durchsuche Bucket $\beta(u)$
  - aktualisiere Distanzarray

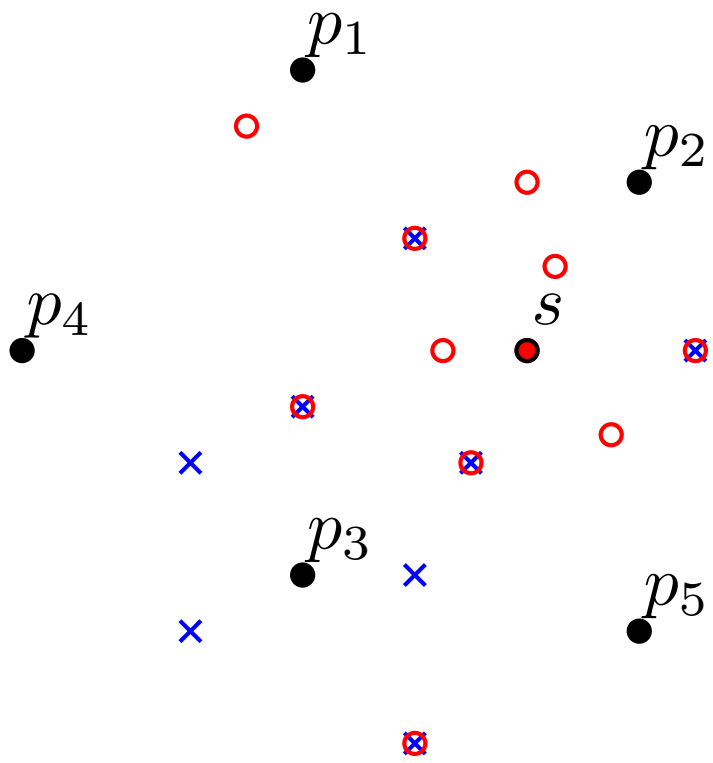
Schneiden der Suchräume

während Rückwärtssuchen:

- breche nicht ab
- für jedes erreichte u:
  - füge Element $(u, t, d(u, t))$ in einen Vektor ein

Kompression:

- sortiere Einträge nach u
- speichere für jeden Knoten u einen Bucket $\beta(u)$ mit allen $(t, d(u, t))$ ab
- mittels Adjazenzarray



während Vorwärtssuche:

- breche nicht ab
- für jedes erreichte u:
  - durchsuche Bucket $\beta(u)$
  - aktualisiere Distanzarray

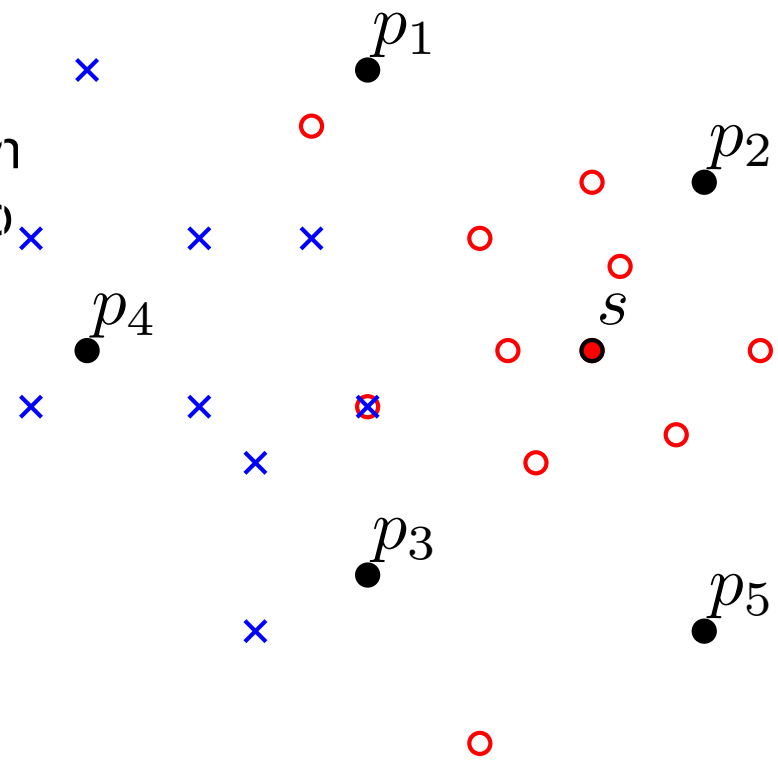
Schneiden der Suchräume

während Rückwärtssuchen:

- breche nicht ab
- für jedes erreichte u:
  - füge Element $(u, t, d(u, t))$ in einen Vektor ein

Kompression:

- sortiere Einträge nach u
- speichere für jeden Knoten u einen Bucket $\beta(u)$ mit allen $(t, d(u, t))$ ab
- mittels Adjazenzarray



während Vorwärtssuche:

- breche nicht ab
- für jedes erreichte u:
  - durchsuche Bucket $\beta(u)$
  - aktualisiere Distanzarray

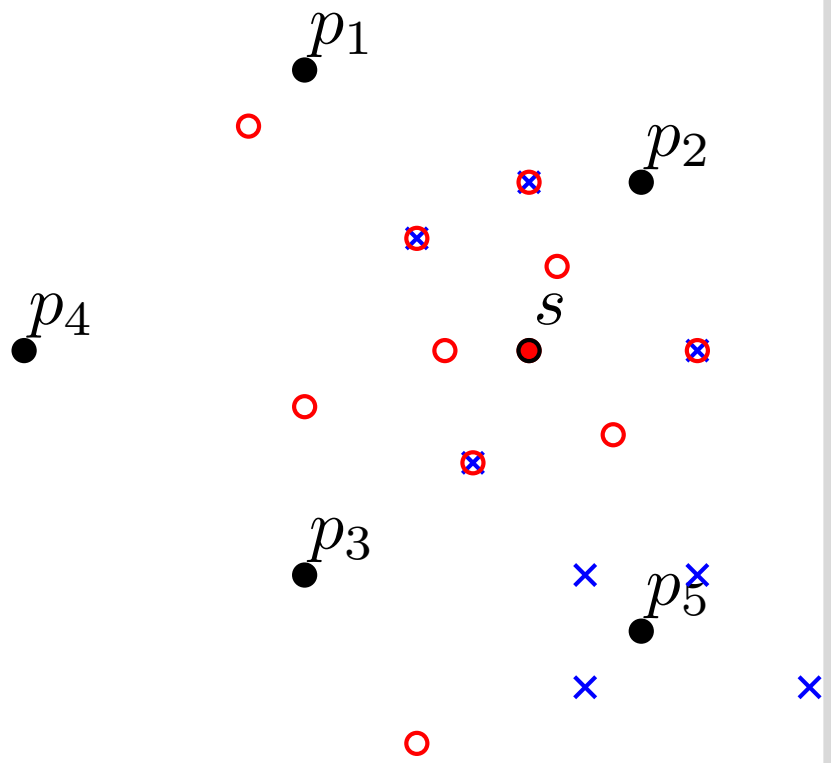
Schneiden der Suchräume

während Rückwärtssuchen:

- breche nicht ab
- für jedes erreichte u:
  - füge Element $(u, t, d(u, t))$ in einen Vektor ein

Kompression:

- sortiere Einträge nach u
- speichere für jeden Knoten u einen Bucket $\beta(u)$ mit allen $(t, d(u, t))$ ab
- mittels Adjazenzarray



während Vorwärtssuche:

- breche nicht ab
- für jedes erreichte u:
  - durchsuche Bucket $\beta(u)$
  - aktualisiere Distanzarray

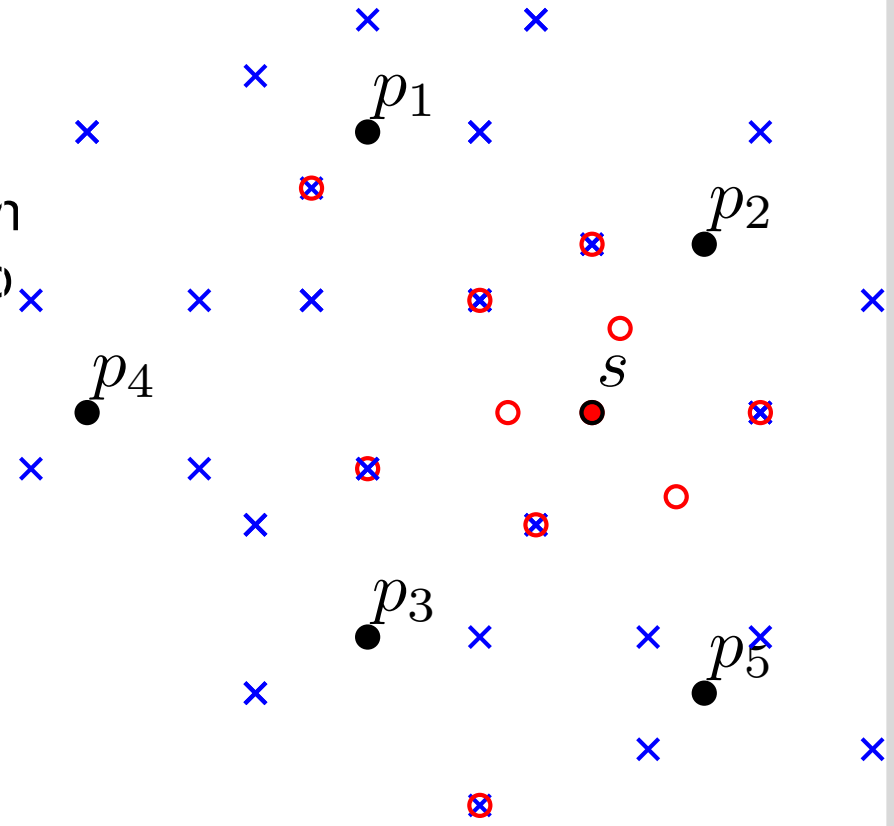
Schneiden der Suchräume

während Rückwärtssuchen:

- breche nicht ab
- für jedes erreichte u:
  - füge Element $(u, t, d(u, t))$ in einen Vektor ein

Kompression:

- sortiere Einträge nach u
- speichere für jeden Knoten u einen Bucket $\beta(u)$ mit allen $(t, d(u, t))$ ab
- mittels Adjazenzarray



während Vorwärtssuche:

- breche nicht ab
- für jedes erreichte u:
  - durchsuche Bucket $\beta(u)$
  - aktualisiere Distanzarray

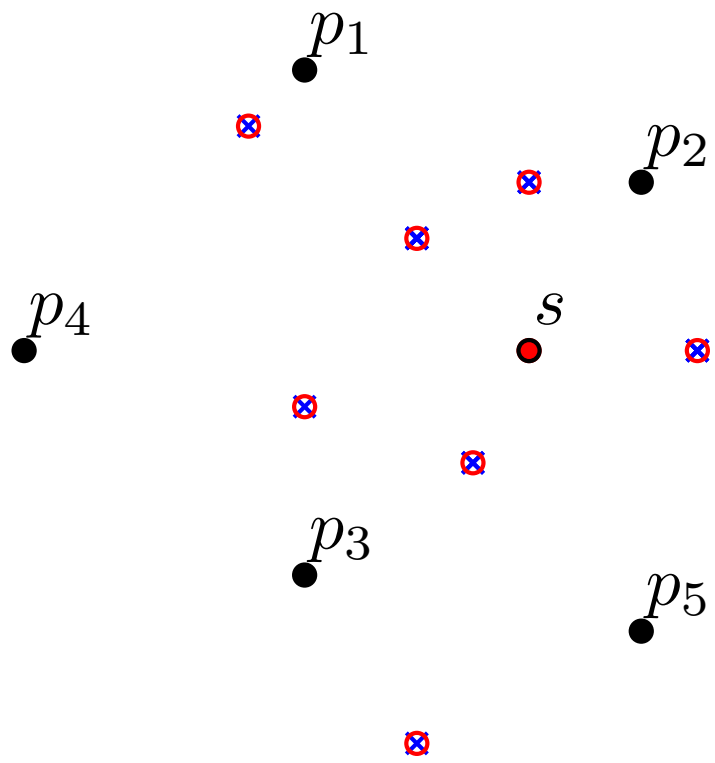
Schneiden der Suchräume

während Rückwärtssuchen:

- breche nicht ab
- für jedes erreichte u:
  - füge Element $(u, t, d(u, t))$ in einen Vektor ein

Kompression:

- sortiere Einträge nach u
- speichere für jeden Knoten u einen Bucket $\beta(u)$ mit allen $(t, d(u, t))$ ab
- mittels Adjazenzarray



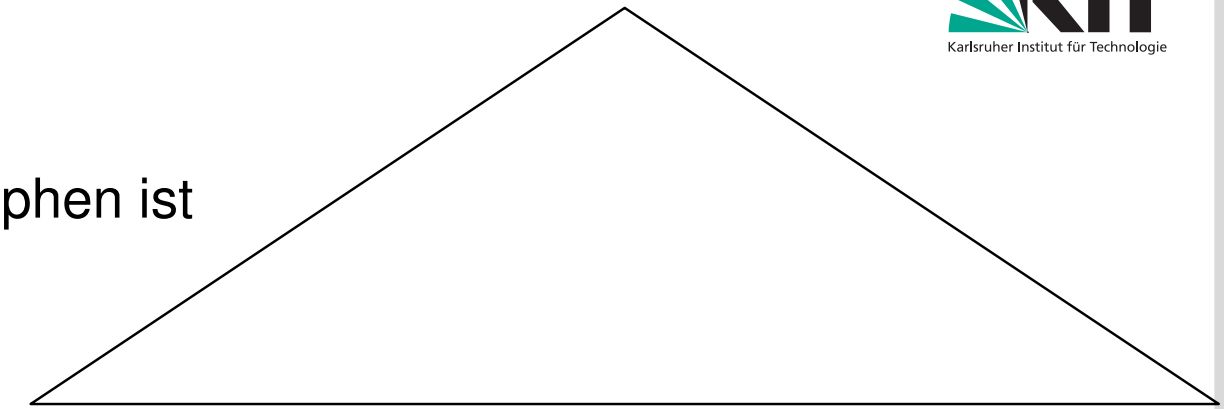
während Vorwärtssuche:

- breche nicht ab
- für jedes erreichte u:
  - durchsuche Bucket $\beta(u)$
  - aktualisiere Distanzarray

RPHAST

Beobachtung:

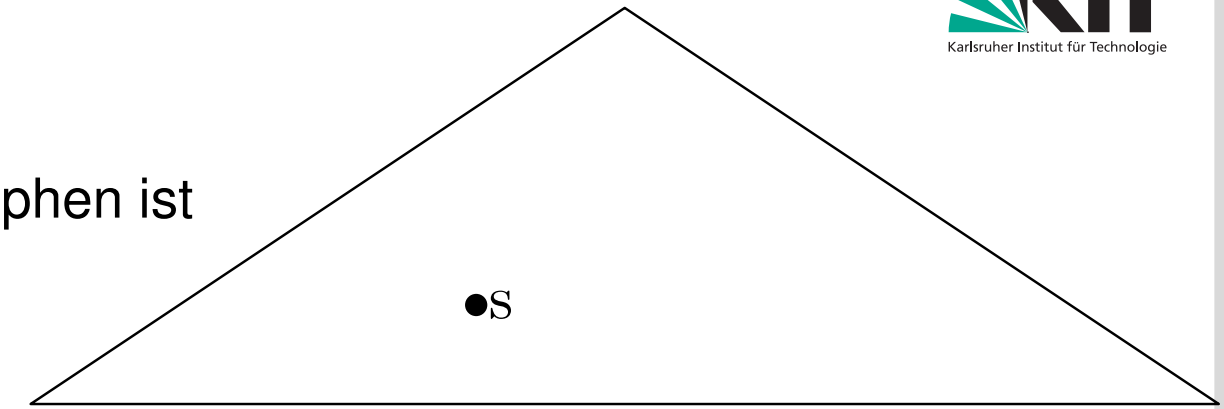
- Sweep über den Graphen ist der Flaschenhals



RPHAST

Beobachtung:

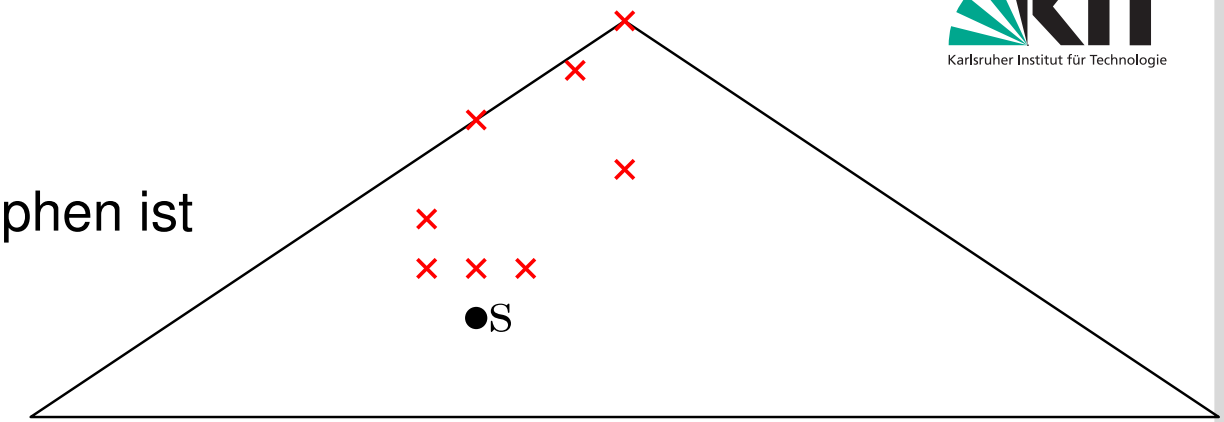
- Sweep über den Graphen ist der Flaschenhals



RPHAST

Beobachtung:

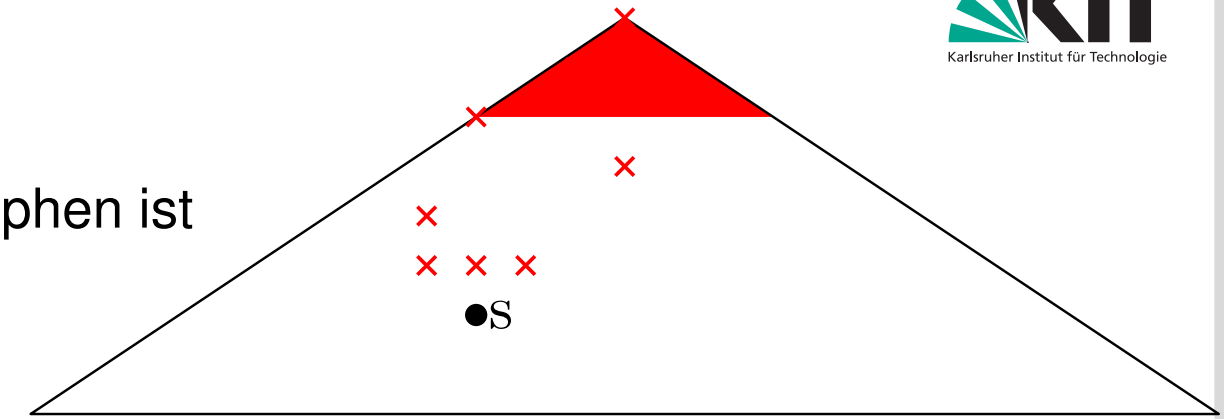
- Sweep über den Graphen ist der Flaschenhals



RPHAST

Beobachtung:

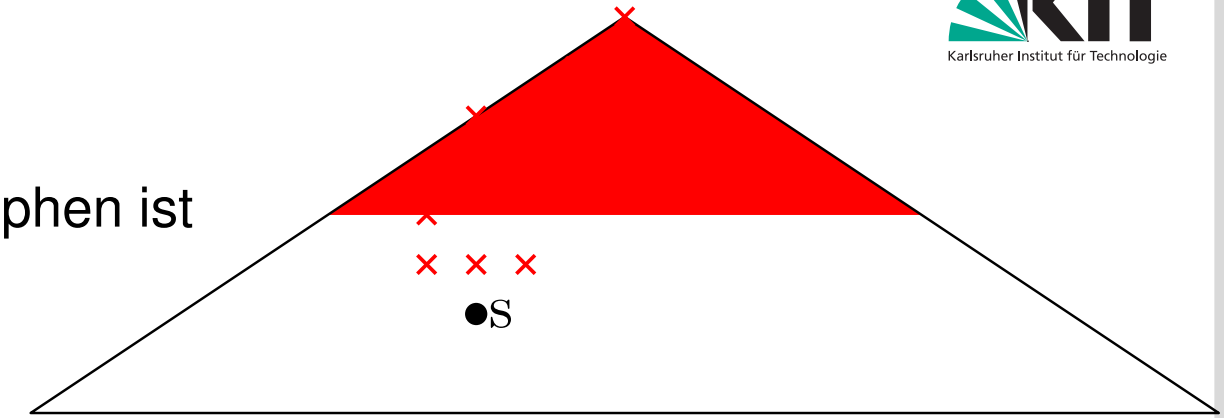
- Sweep über den Graphen ist der Flaschenhals



RPHAST

Beobachtung:

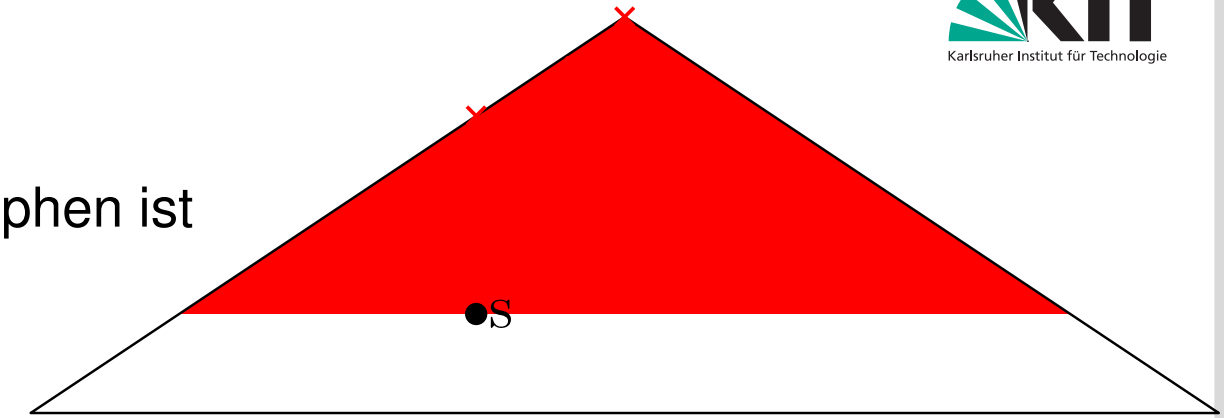
- Sweep über den Graphen ist der Flaschenhals



RPHAST

Beobachtung:

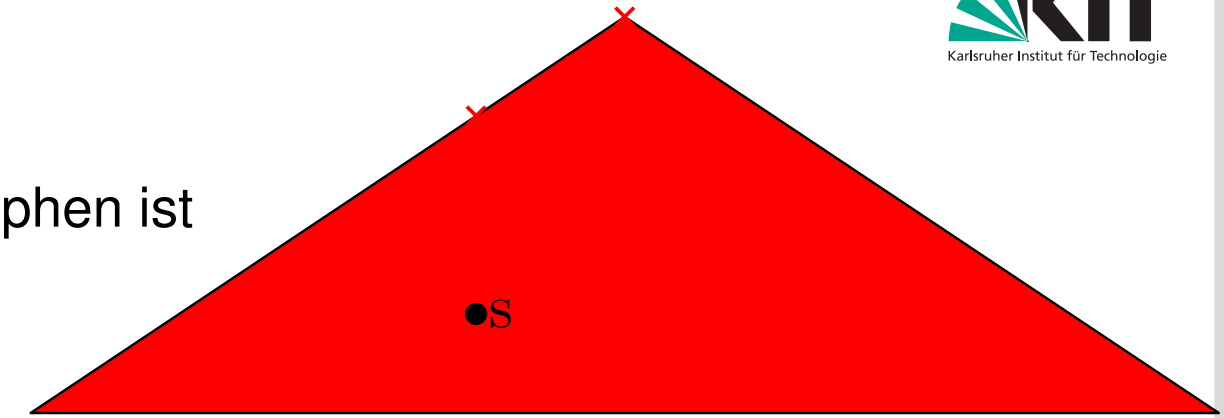
- Sweep über den Graphen ist der Flaschenhals



RPHAST

Beobachtung:

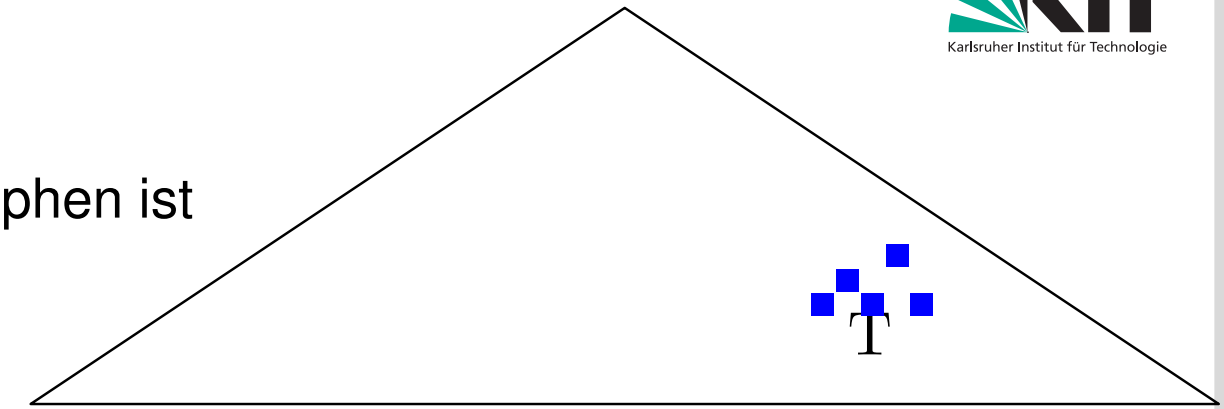
- Sweep über den Graphen ist der Flaschenhals



RPHAST

Beobachtung:

- Sweep über den Graphen ist der Flaschenhals



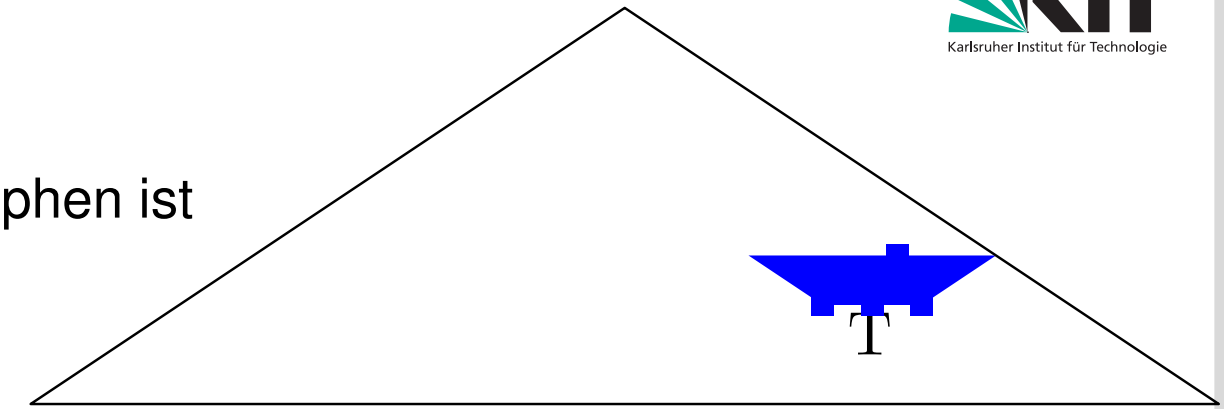
Idee:

- extrahiere relevanten Teil des Graphen (Ziel Selektion)

RPHAST

Beobachtung:

- Sweep über den Graphen ist der Flaschenhals



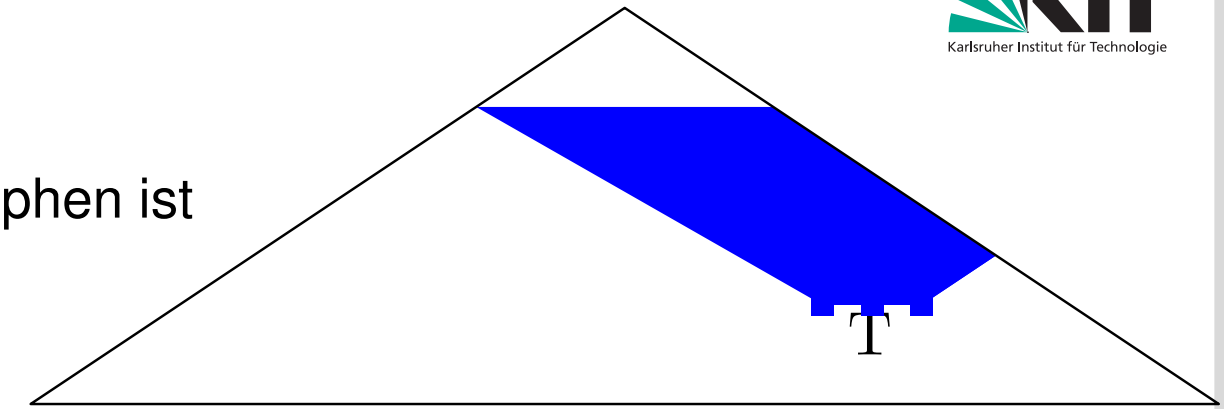
Idee:

- extrahiere relevanten Teil des Graphen (Ziel Selektion)

RPHAST

Beobachtung:

- Sweep über den Graphen ist der Flaschenhals



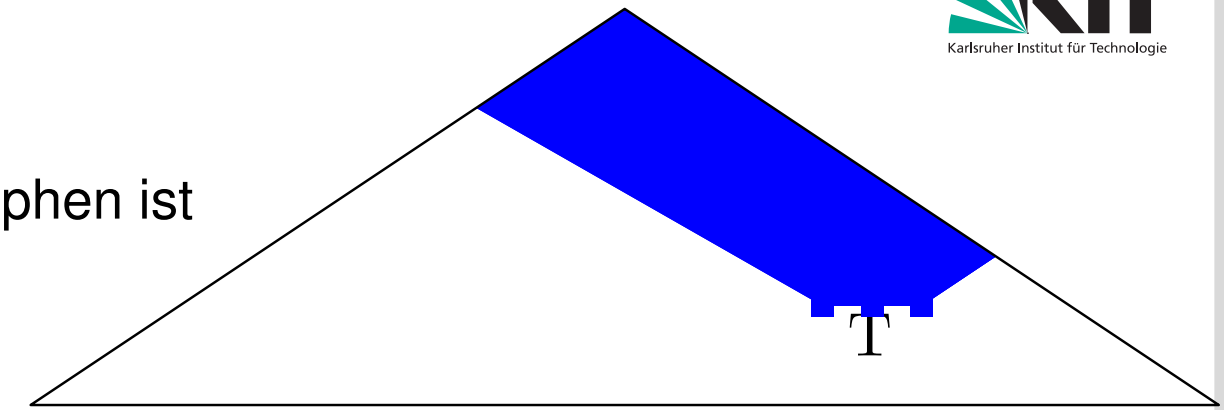
Idee:

- extrahiere relevanten Teil des Graphen (Ziel Selektion)

RPHAST

Beobachtung:

- Sweep über den Graphen ist der Flaschenhals



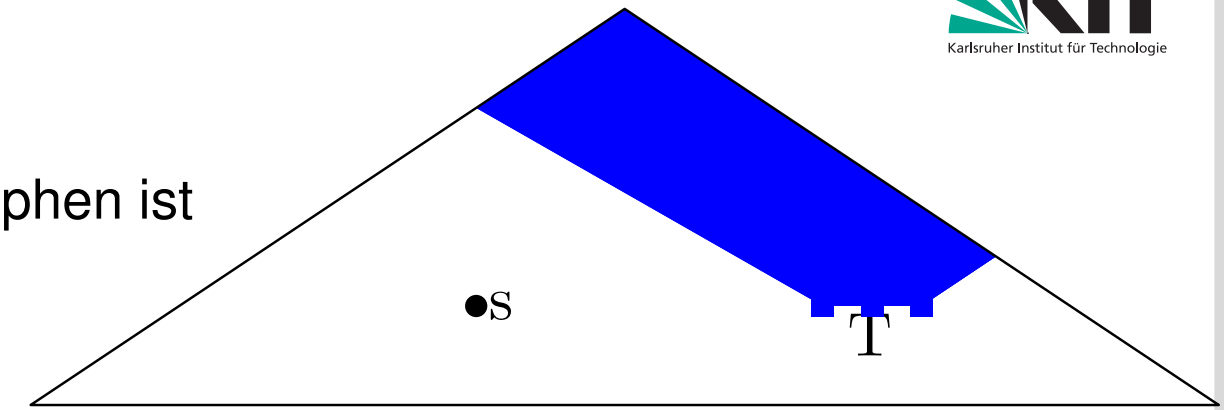
Idee:

- extrahiere relevanten Teil des Graphen (Ziel Selektion)

RPHAST

Beobachtung:

- Sweep über den Graphen ist der Flaschenhals



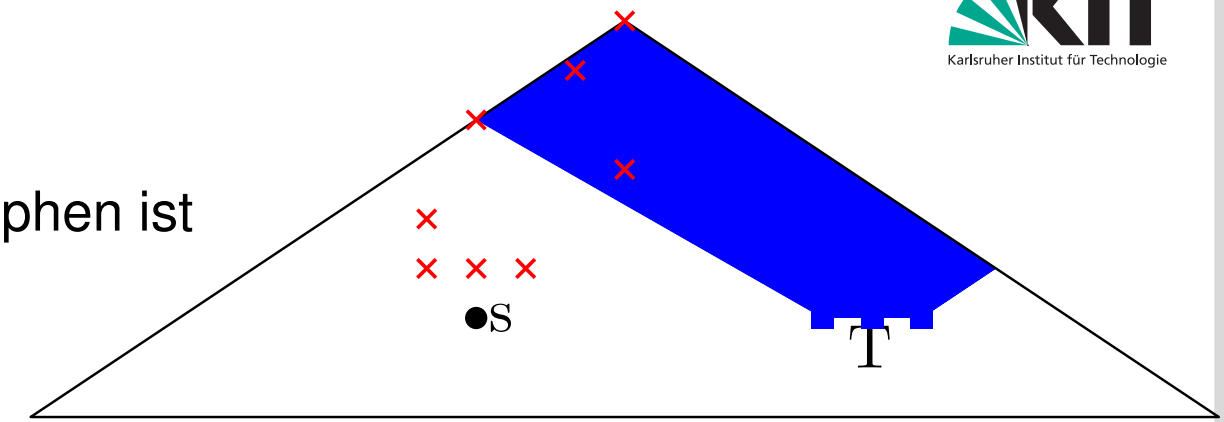
Idee:

- extrahiere relevanten Teil des Graphen (Ziel Selektion)
- Aufwärtssuche im vollen Graphen

RPHAST

Beobachtung:

- Sweep über den Graphen ist der Flaschenhals



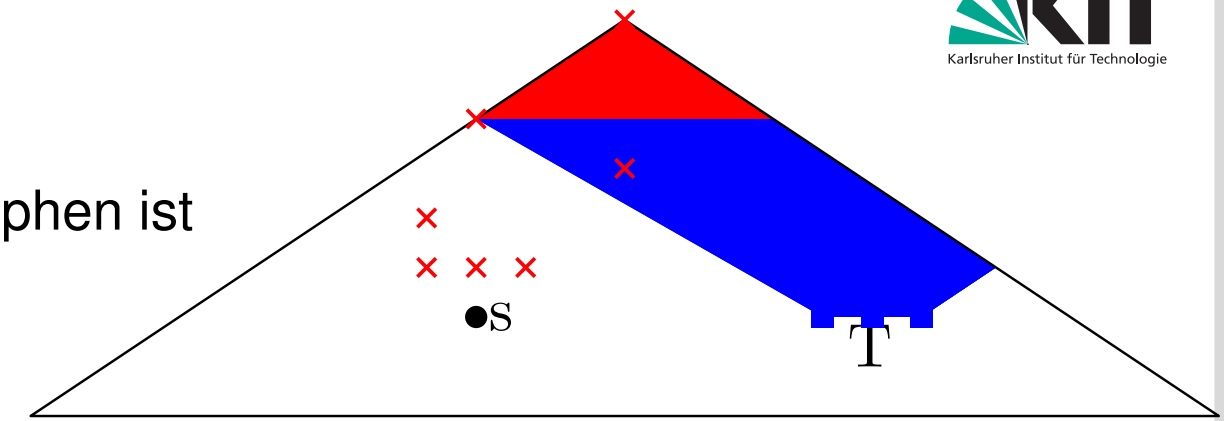
Idee:

- extrahiere relevanten Teil des Graphen (Ziel Selektion)
- Aufwärtssuche im vollen Graphen

RPHAST

Beobachtung:

- Sweep über den Graphen ist der Flaschenhals



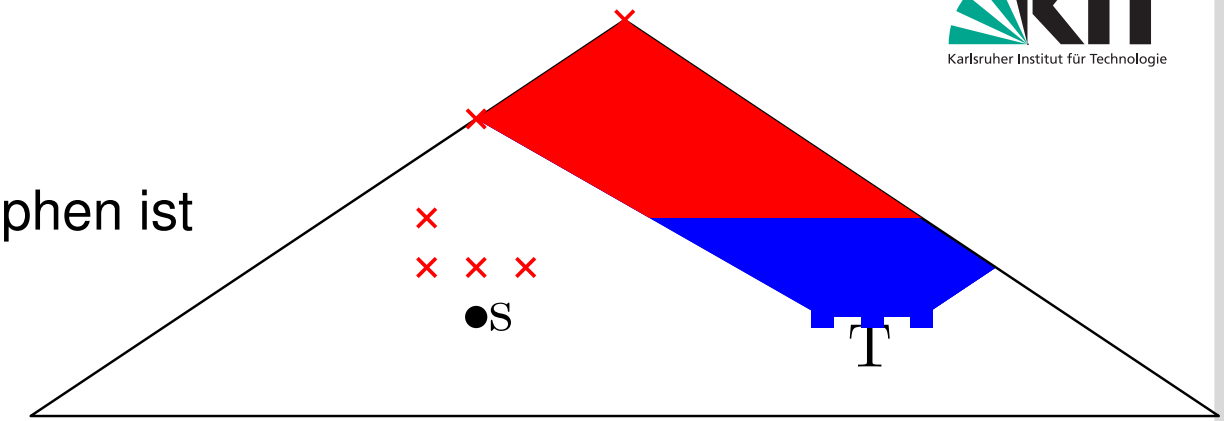
Idee:

- extrahiere relevanten Teil des Graphen (Ziel Selektion)
- Aufwärtssuche im vollen Graphen
- Sweep auf extrahiertem Graphen

RPHAST

Beobachtung:

- Sweep über den Graphen ist der Flaschenhals



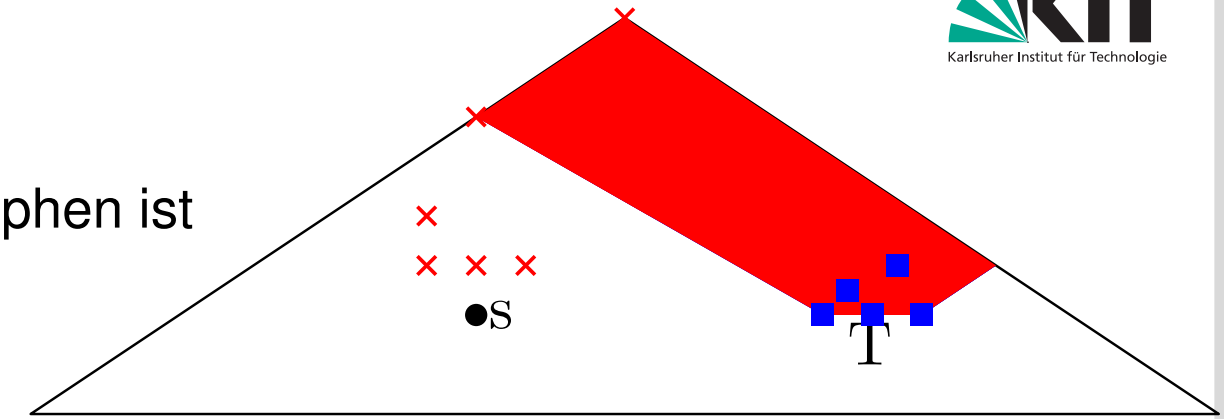
Idee:

- extrahiere relevanten Teil des Graphen (Ziel Selektion)
- Aufwärtssuche im vollen Graphen
- Sweep auf extrahiertem Graphen

RPHAST

Beobachtung:

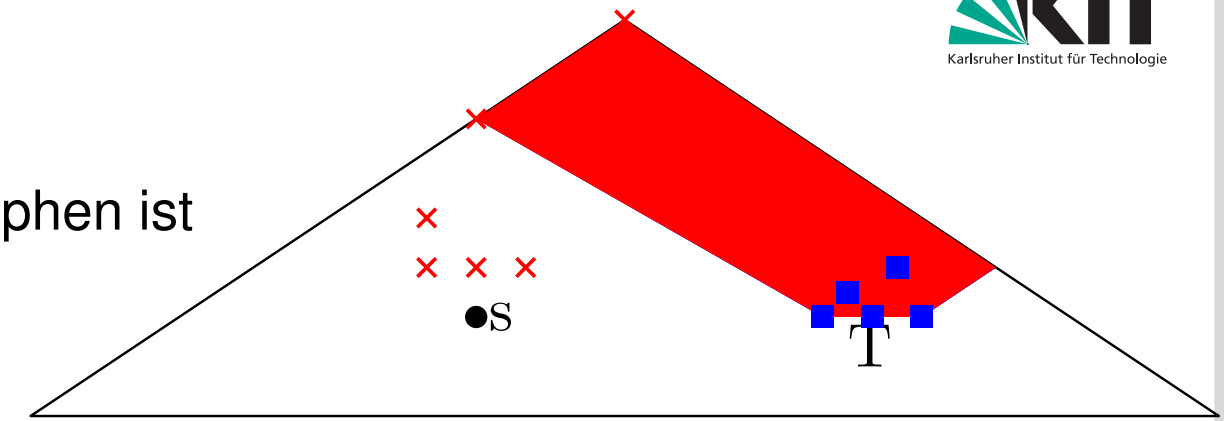
- Sweep über den Graphen ist der Flaschenhals



Idee:

- extrahiere relevanten Teil des Graphen (Ziel Selektion)
- Aufwärtssuche im vollen Graphen
- Sweep auf extrahiertem Graphen

RPHAST



Beobachtung:

- Sweep über den Graphen ist der Flaschenhals

Idee:

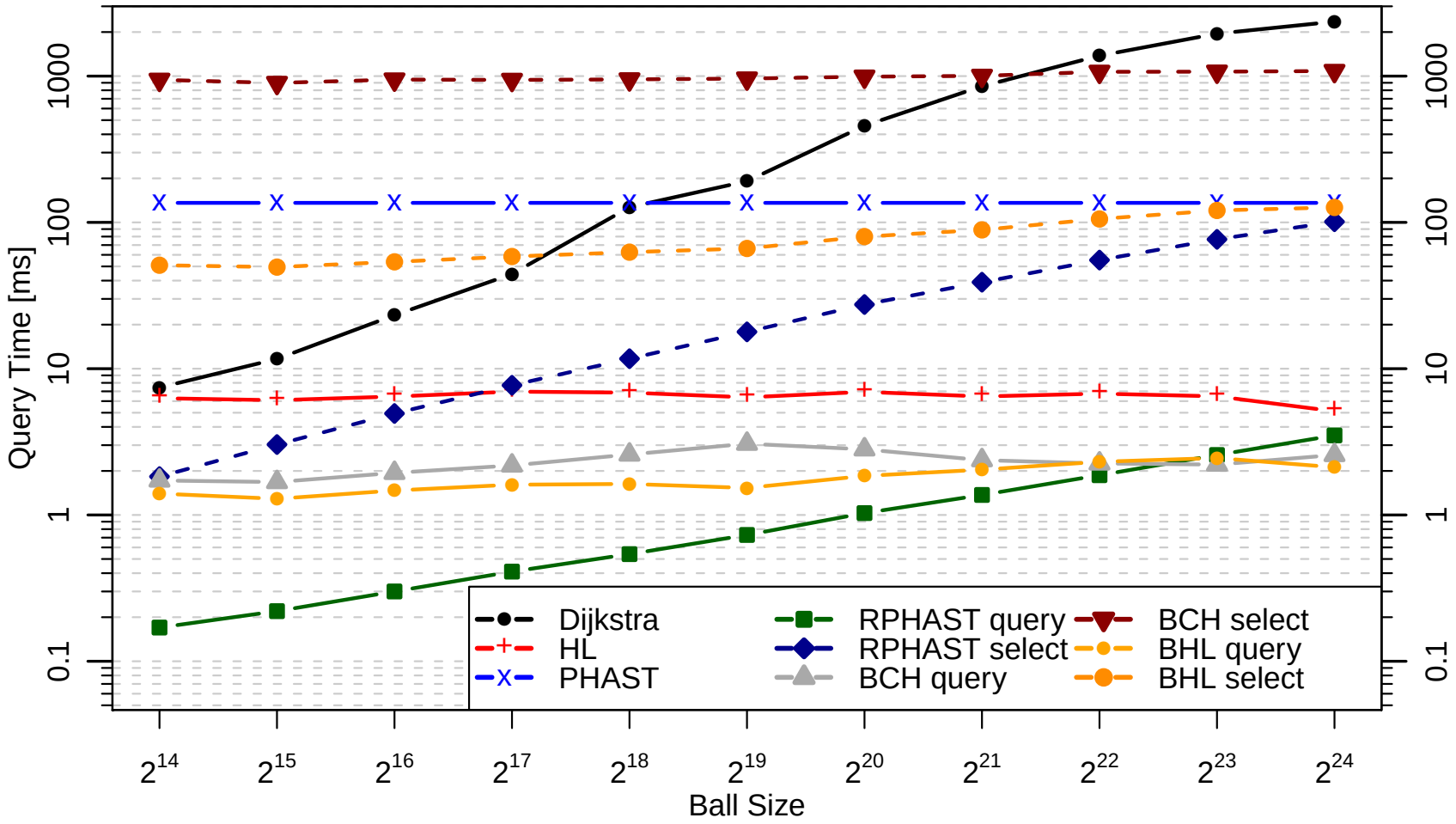
- extrahiere relevanten Teil des Graphen (Ziel Selektion)
- Aufwärtssuche im vollen Graphen
- Sweep auf extrahiertem Graphen

$\Rightarrow$

- Startknoten kann im ganzem Graphen liegen
- Grösse des extrahierten Graphen hängt von Verteilung und Anzahl T ab
- kann wie PHAST parallelisiert werden
- GPU implementation möglich

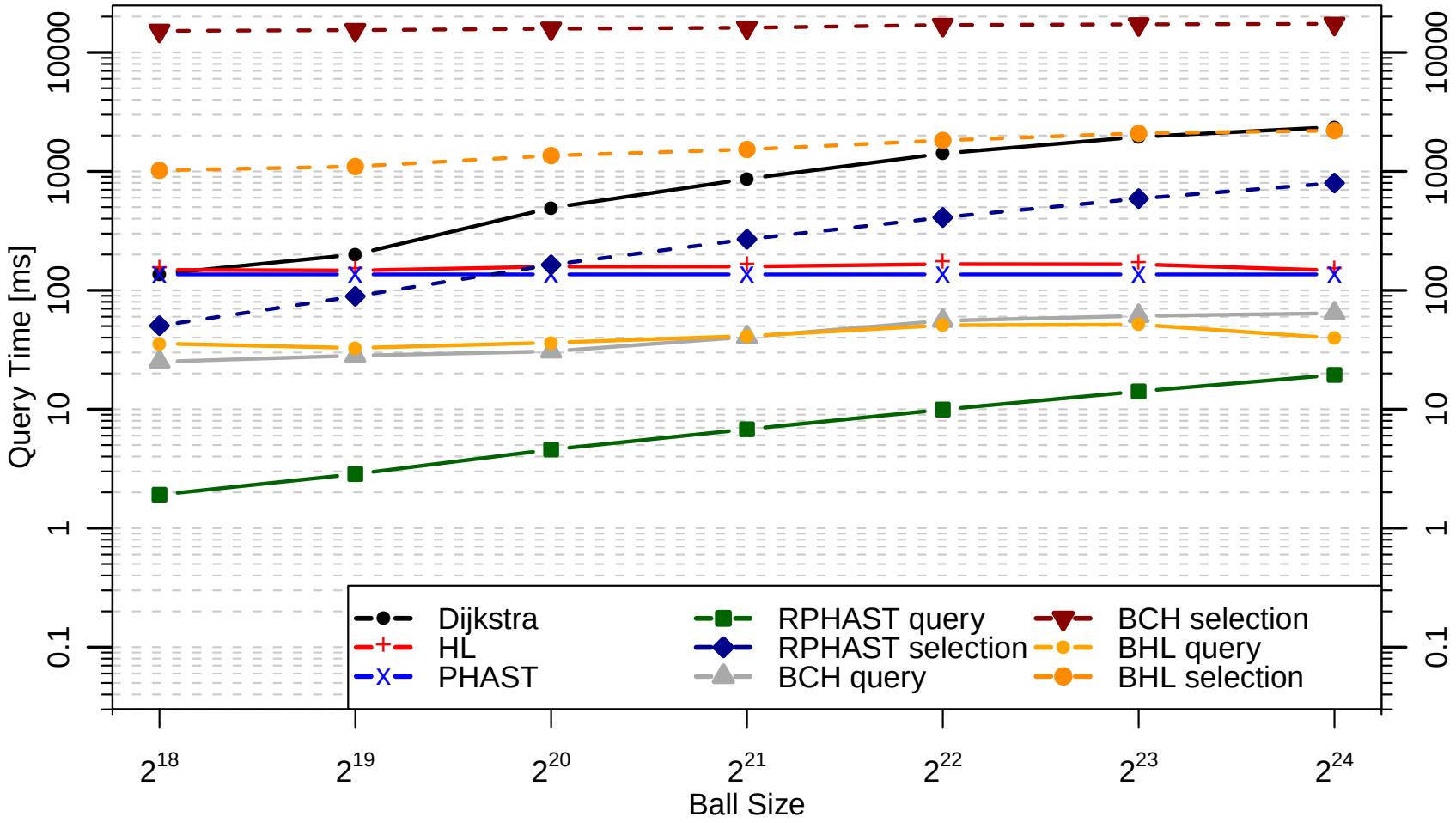
Experimente I

input: Westeuropa (18M Knoten), $|T| = 2^{14}$



Experimente II

input: Westeuropa (18M Knoten), $|T| = 2^{18}$



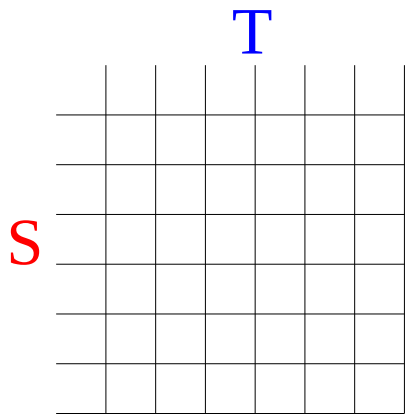
Many-to-Many Kürzeste Wege

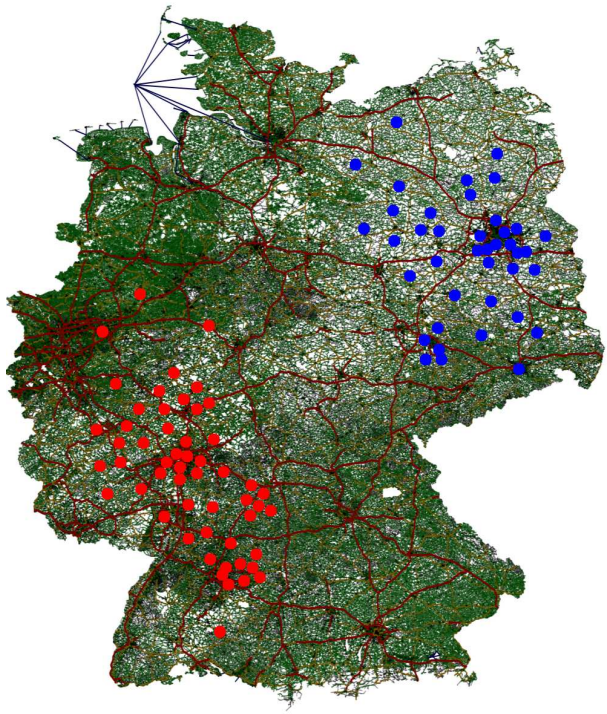
Gegeben:

- Graph
- Knotenmengen $S, T \in V$

Gesucht:

- Distanzmatrix D





Many-to-Many Kürzeste Wege

Gegeben:

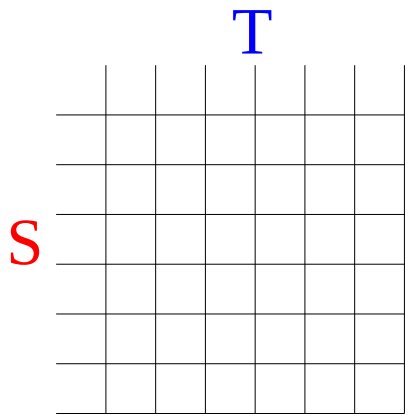
- Graph
- Knotenmengen $S, T \in V$

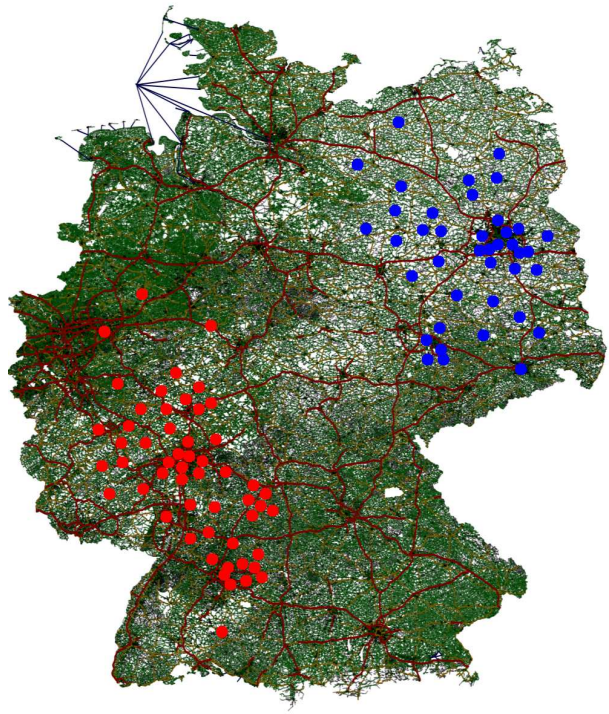
Gesucht:

- Distanzmatrix D

Anwendungen:

- vehicle routing
- traveling salesman





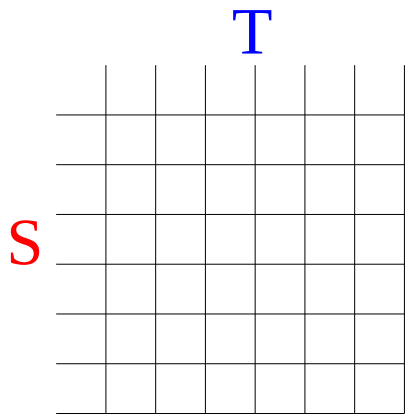
Many-to-Many Kürzeste Wege

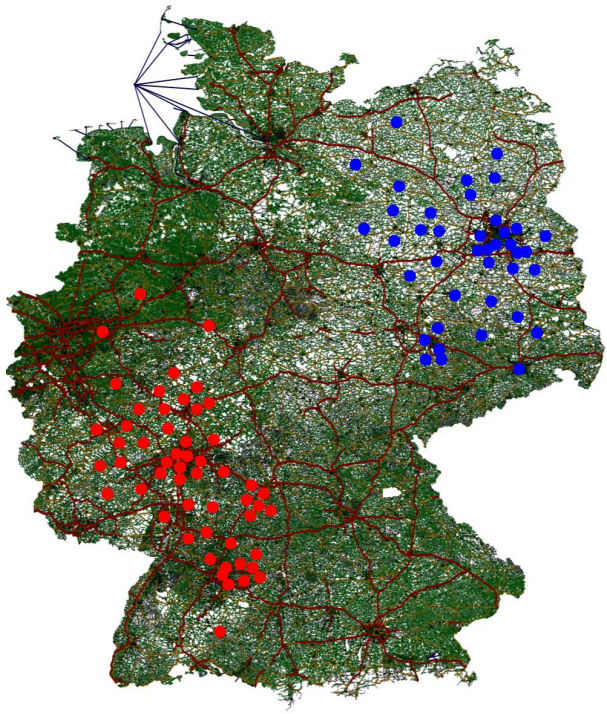
Gegeben:

- Graph
- Knotenmengen $S, T \in V$

Gesucht:

- Distanzmatrix D





Anwendungen:

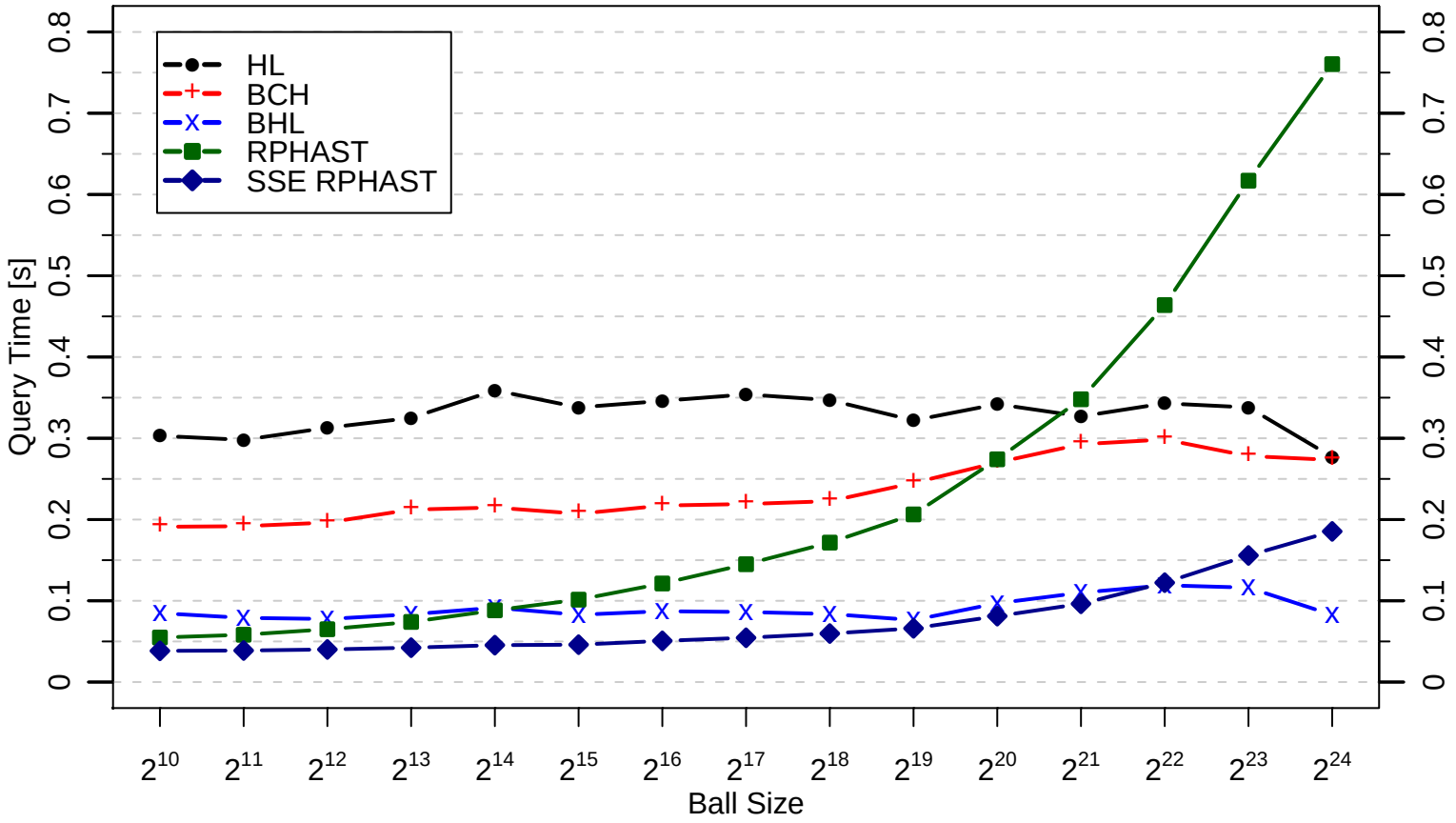
- vehicle routing
- traveling salesman

Lösung:

- $|S|$ one-to-many Anfragen
- speicher Distanzen in der Tabelle
- RPHAST kann multiples Setup (SSE) nutzen

Experimente I

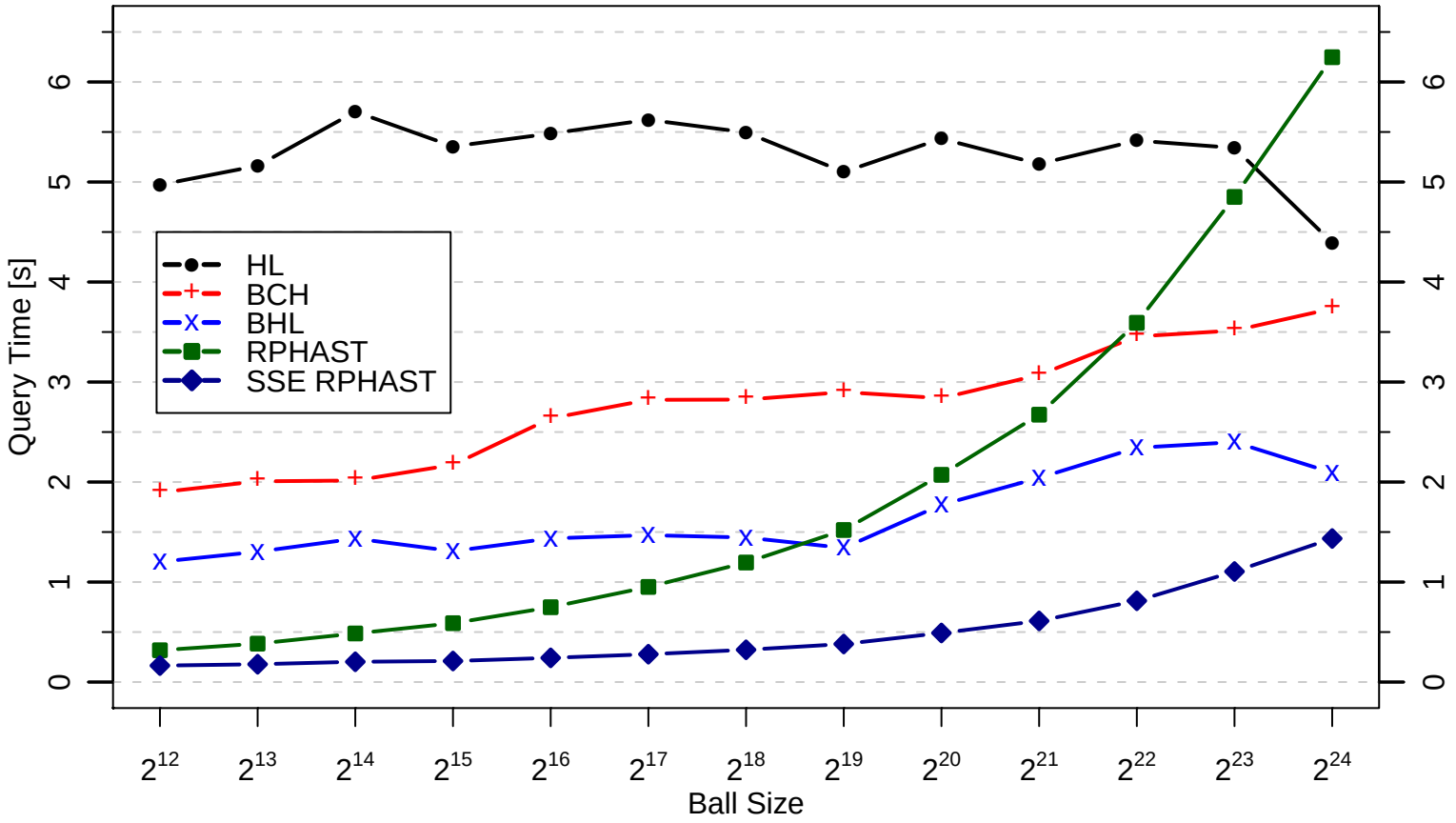
input: Westeuropa (18M Knoten), $|S| = |T| = 2^{10}$



Beobachtung: alle Techniken unter einer Sekunde

Experimente II

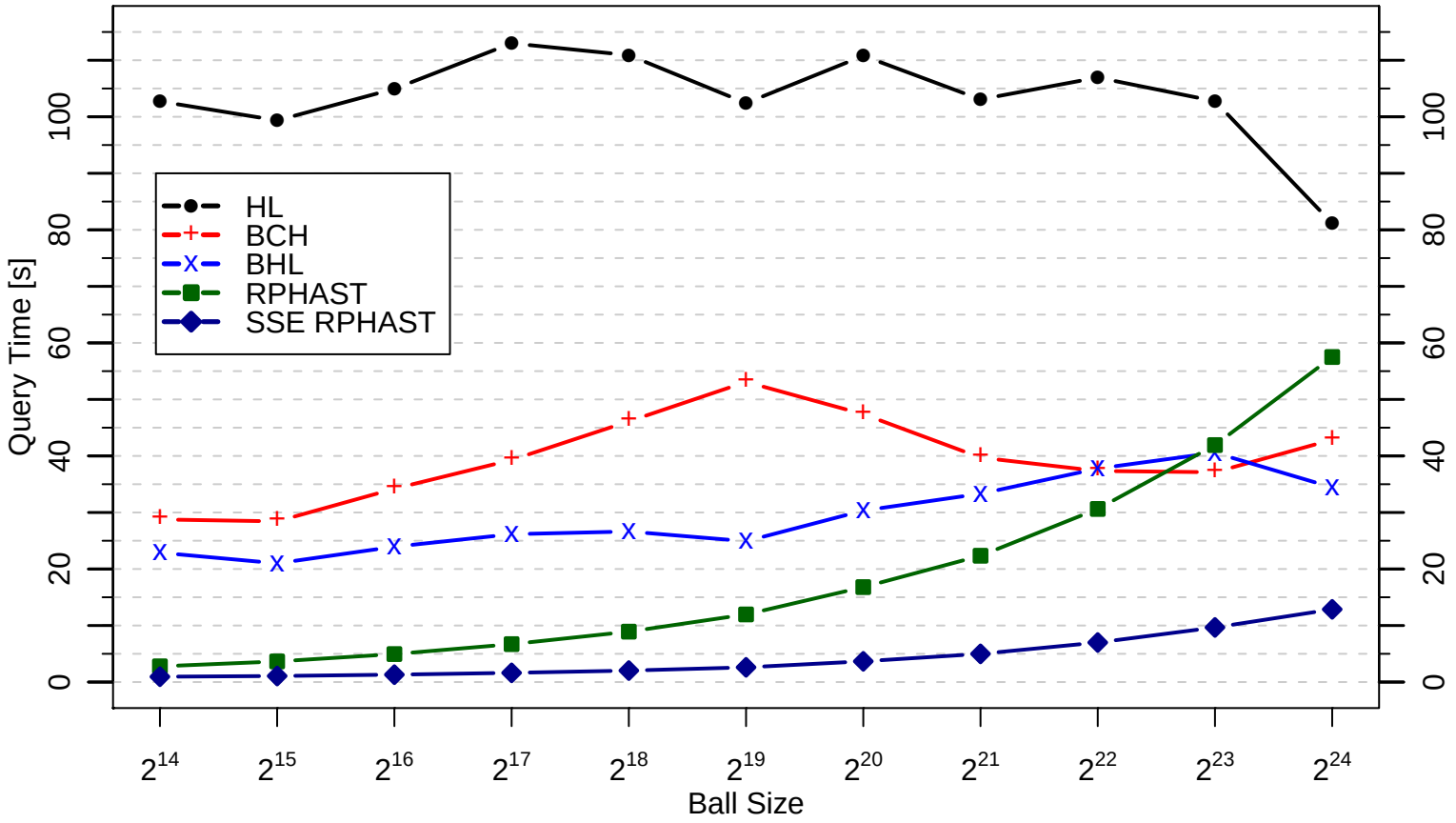
input: Westeuropa (18M Knoten), $|S| = |T| = 2^{12}$



Beobachtung: SSE PHAST am schnellsten

Experimente III

input: Westeuropa (18M Knoten), $|S| = |T| = 2^{14}$



Beobachtung: SSE PHAST am schnellsten

Nächster POI

Szenario:

- Zielknoten sind POIs
- finde k nächste POIs von einem Startknoten s

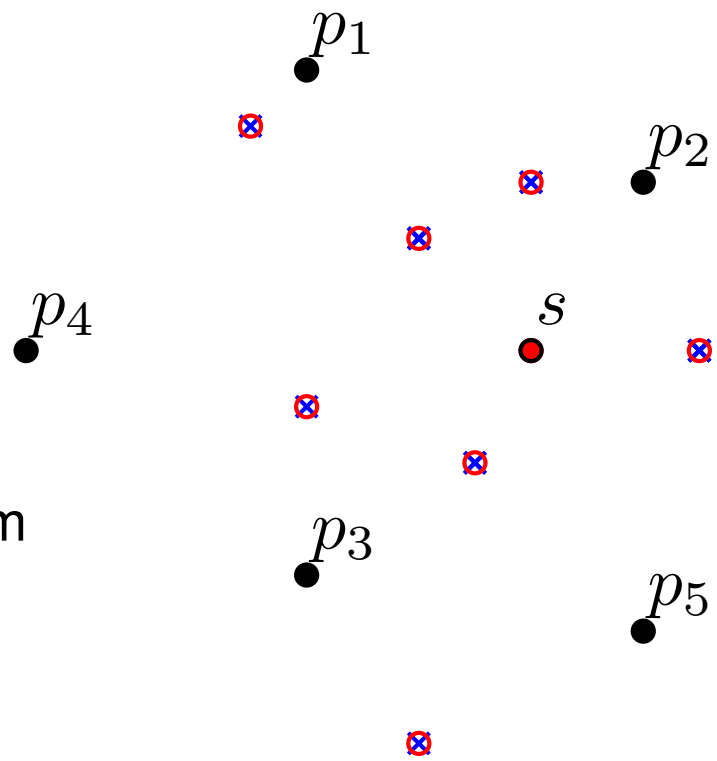
Nächster POI

Szenario:

- Zielknoten sind POIs
- finde k nächste POIs von einem Startknoten s

Lösung:

- wie one-to-many
- ordne die buckets pro Knoten auch nach aufsteigender Distanz
- in jedem Bucket müssen nur die k nächsten POIs durchsucht werden
- Laufzeit für POI Query nicht abhängig von Anzahl POIs im System
- Laufzeit: Suchraum $\cdot$ k



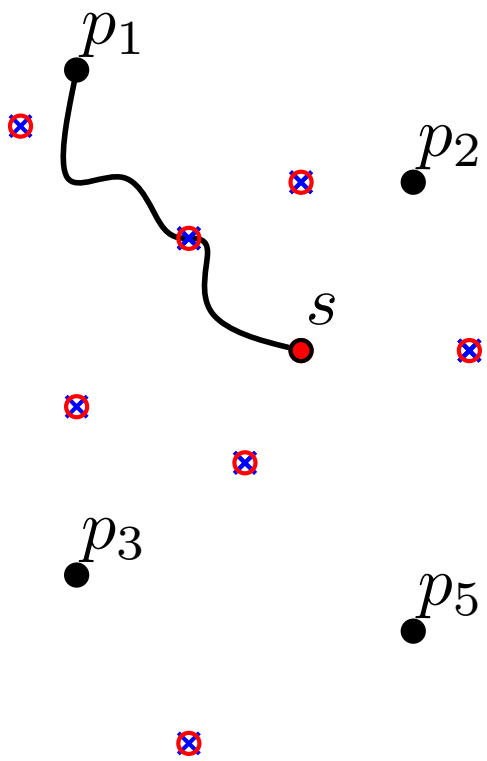
Nächster POI

Szenario:

- Zielknoten sind POIs
- finde k nächste POIs von einem Startknoten s

Lösung:

- wie one-to-many
- ordne die buckets pro Knoten auch nach aufsteigender Distanz
- in jedem Bucket müssen nur die k nächsten POIs durchsucht werden
- Laufzeit für POI Query nicht abhängig von Anzahl POIs im System
- Laufzeit: Suchraum $\cdot$ k



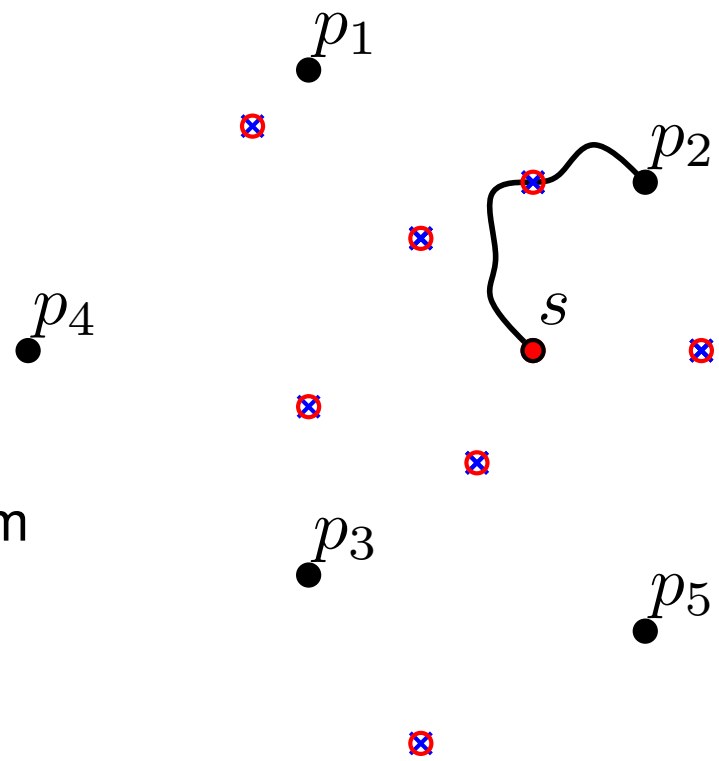
Nächster POI

Szenario:

- Zielknoten sind POIs
- finde k nächste POIs von einem Startknoten s

Lösung:

- wie one-to-many
- ordne die buckets pro Knoten auch nach aufsteigender Distanz
- in jedem Bucket müssen nur die k nächsten POIs durchsucht werden
- Laufzeit für POI Query nicht abhängig von Anzahl POIs im System
- Laufzeit: Suchraum $\cdot$ k



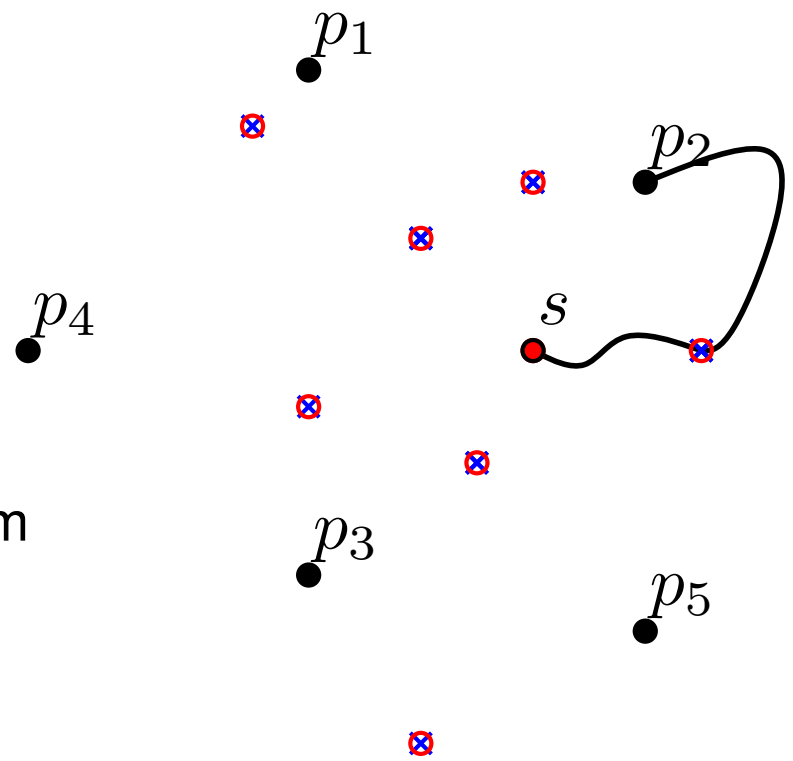
Nächster POI

Szenario:

- Zielknoten sind POIs
- finde k nächste POIs von einem Startknoten s

Lösung:

- wie one-to-many
- ordne die buckets pro Knoten auch nach aufsteigender Distanz
- in jedem Bucket müssen nur die k nächsten POIs durchsucht werden
- Laufzeit für POI Query nicht abhängig von Anzahl POIs im System
- Laufzeit: Suchraum $\cdot$ k



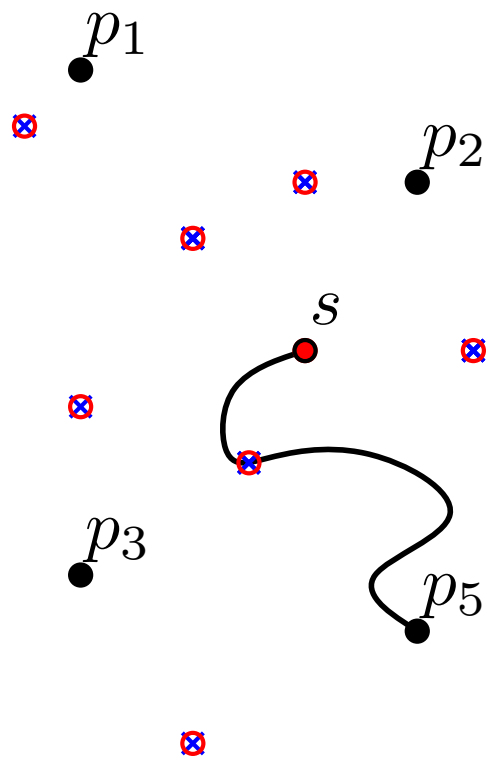
Nächster POI

Szenario:

- Zielknoten sind POIs
- finde k nächste POIs von einem Startknoten s

Lösung:

- wie one-to-many
- ordne die buckets pro Knoten auch nach aufsteigender Distanz
- in jedem Bucket müssen nur die k nächsten POIs durchsucht werden
- Laufzeit für POI Query nicht abhängig von Anzahl POIs im System
- Laufzeit: Suchraum $\cdot$ k



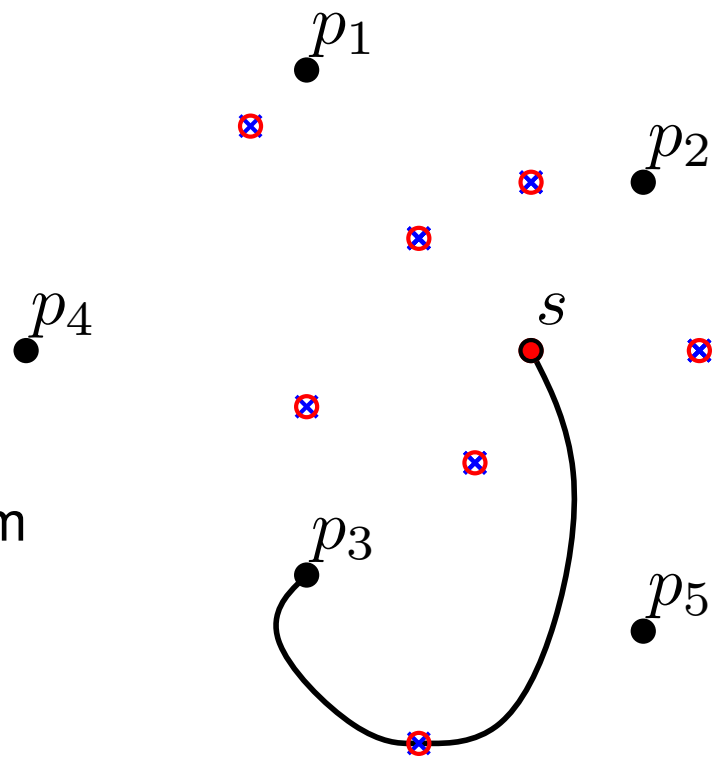
Nächster POI

Szenario:

- Zielknoten sind POIs
- finde k nächste POIs von einem Startknoten s

Lösung:

- wie one-to-many
- ordne die buckets pro Knoten auch nach aufsteigender Distanz
- in jedem Bucket müssen nur die k nächsten POIs durchsucht werden
- Laufzeit für POI Query nicht abhängig von Anzahl POIs im System
- Laufzeit: Suchraum $\cdot$ k



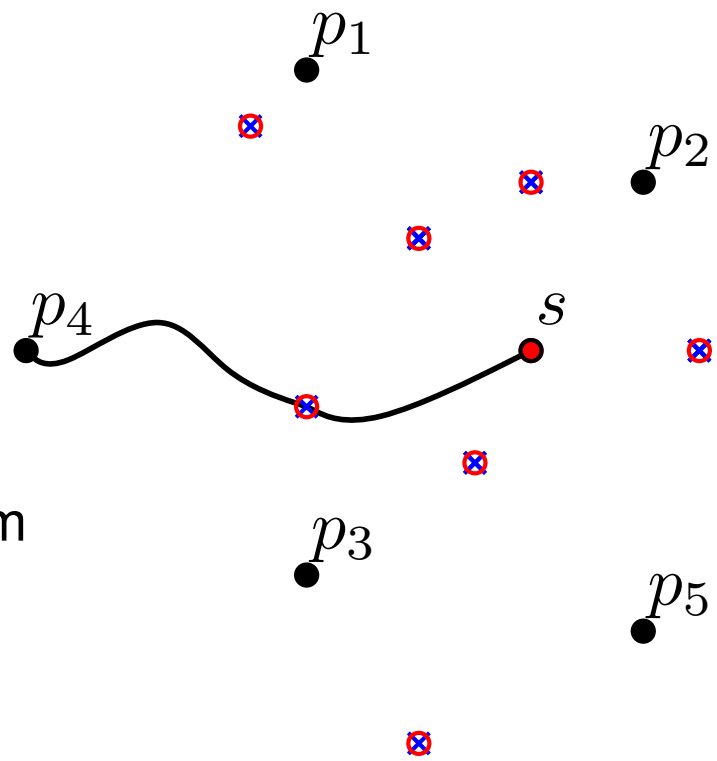
Nächster POI

Szenario:

- Zielknoten sind POIs
- finde k nächste POIs von einem Startknoten s

Lösung:

- wie one-to-many
- ordne die buckets pro Knoten auch nach aufsteigender Distanz
- in jedem Bucket müssen nur die k nächsten POIs durchsucht werden
- Laufzeit für POI Query nicht abhängig von Anzahl POIs im System
- Laufzeit: Suchraum $\cdot$ k



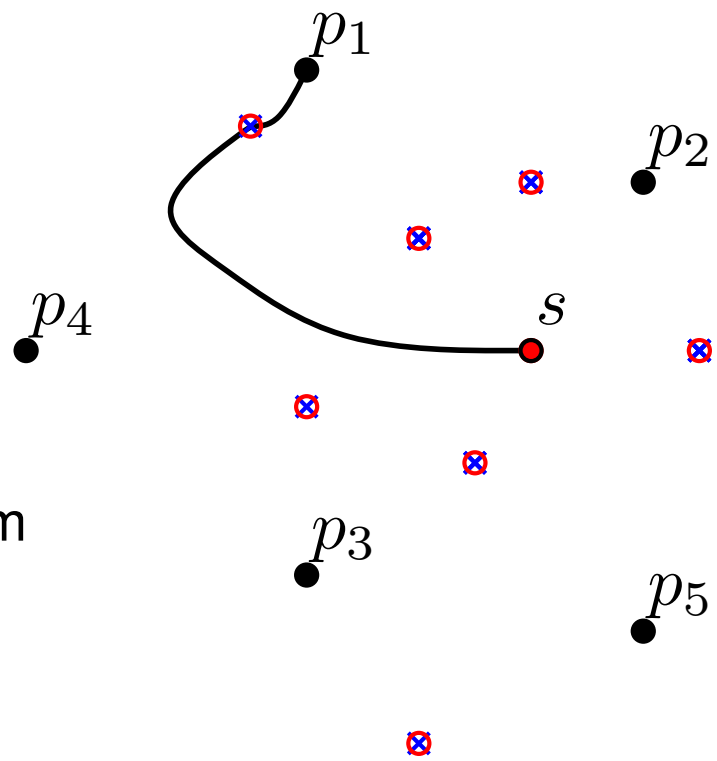
Nächster POI

Szenario:

- Zielknoten sind POIs
- finde k nächste POIs von einem Startknoten s

Lösung:

- wie one-to-many
- ordne die buckets pro Knoten auch nach aufsteigender Distanz
- in jedem Bucket müssen nur die k nächsten POIs durchsucht werden
- Laufzeit für POI Query nicht abhängig von Anzahl POIs im System
- Laufzeit: Suchraum $\cdot$ k



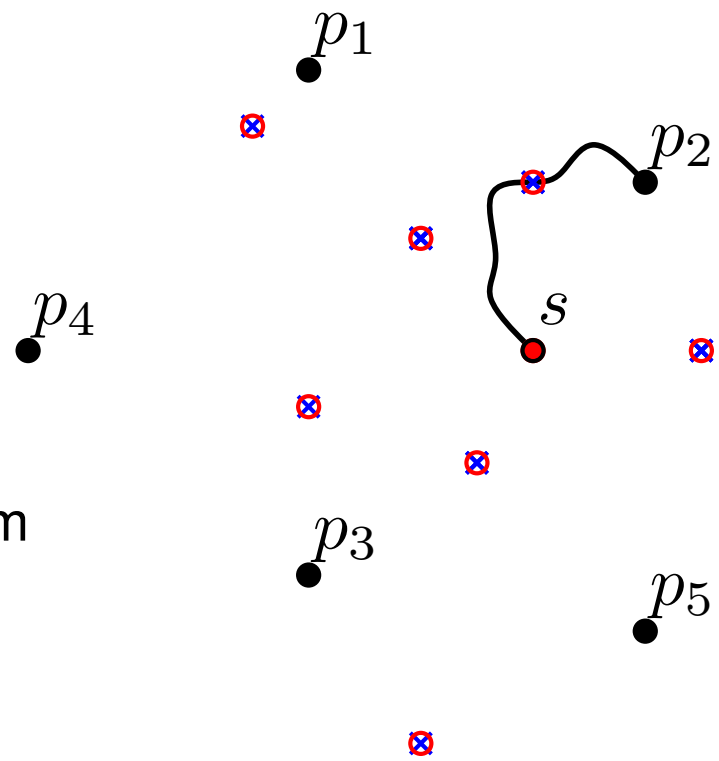
Nächster POI

Szenario:

- Zielknoten sind POIs
- finde k nächste POIs von einem Startknoten s

Lösung:

- wie one-to-many
- ordne die buckets pro Knoten auch nach aufsteigender Distanz
- in jedem Bucket müssen nur die k nächsten POIs durchsucht werden
- Laufzeit für POI Query nicht abhängig von Anzahl POIs im System
- Laufzeit: Suchraum $\cdot$ k



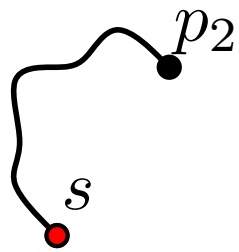
Nächster POI

Szenario:

- Zielknoten sind POIs
- finde k nächste POIs von einem Startknoten s

Lösung:

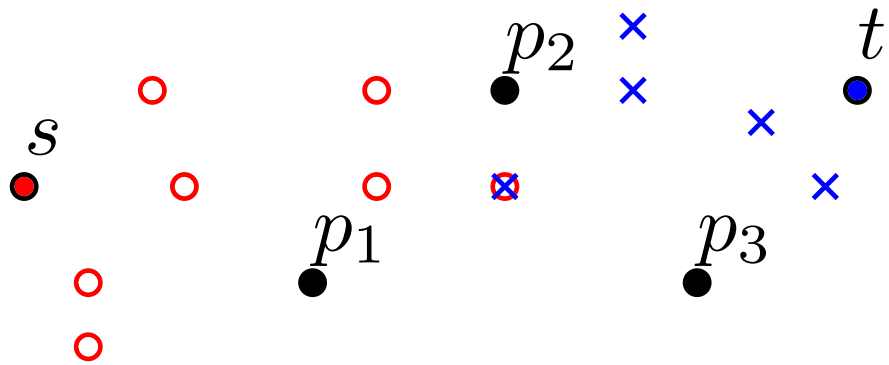
- wie one-to-many
- ordne die buckets pro Knoten auch nach aufsteigender Distanz
- in jedem Bucket müssen nur die k nächsten POIs durchsucht werden
- Laufzeit für POI Query nicht abhängig von Anzahl POIs im System
- Laufzeit: Suchraum $\cdot$ k



Bester Via Knoten

Szenario:

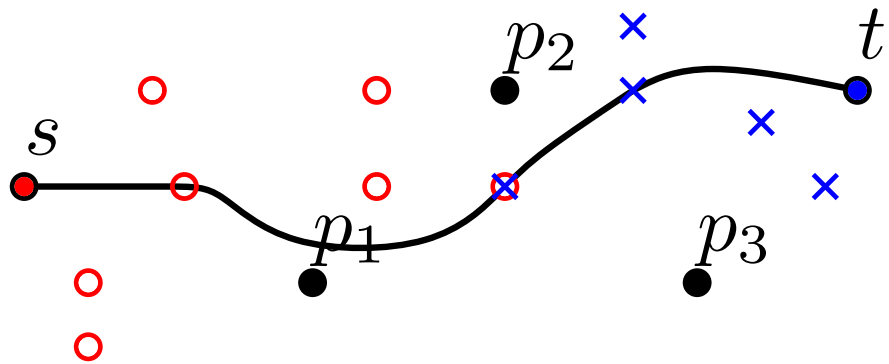
- Zielknoten sind POIs
- finde k best Via Knoten POIs von einem Startknoten s zu einem Startknoten t
- minimiere $\text{dist}(s, p) + \text{dist}(p, t)$ über alle POIs p



Bester Via Knoten

Szenario:

- Zielknoten sind POIs
- finde k best Via Knoten POIs von einem Startknoten s zu einem Startknoten t
- minimiere $\text{dist}(s, p) + \text{dist}(p, t)$ über alle POIs p



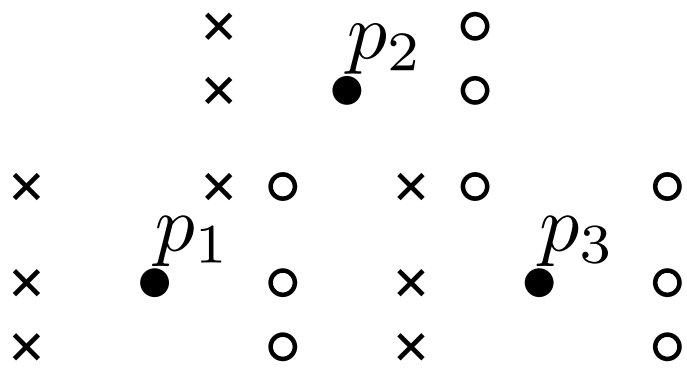
Bester Via Knoten

Szenario:

- Zielknoten sind POIs
- finde k best Via Knoten POIs von einem Startknoten s zu einem Startknoten t
- minimiere $\text{dist}(s, p) + \text{dist}(p, t)$ über alle POIs p

Lösung:

- Vorwärts- und Rückwartssuche von jedem POI
- speicher Kreuzprodukt der beiden Suchräume mit Distanz durch den POI
- Such von s und t: evaluiere jedes Paar
- Laufzeit: Suchraum$^2 \cdot k$



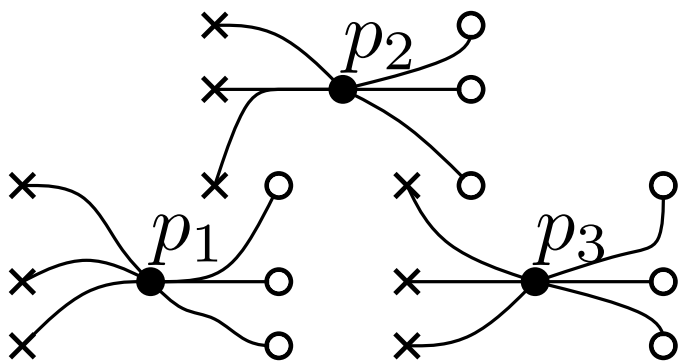
Bester Via Knoten

Szenario:

- Zielknoten sind POIs
- finde k best Via Knoten POIs von einem Startknoten s zu einem Startknoten t
- minimiere $\text{dist}(s, p) + \text{dist}(p, t)$ über alle POIs p

Lösung:

- Vorwärts- und Rückwartssuche von jedem POI
- speicher Kreuzprodukt der beiden Suchräume mit Distanz durch den POI
- Such von s und t: evaluiere jedes Paar
- Laufzeit: $\text{Suchraum}^2 \cdot k$



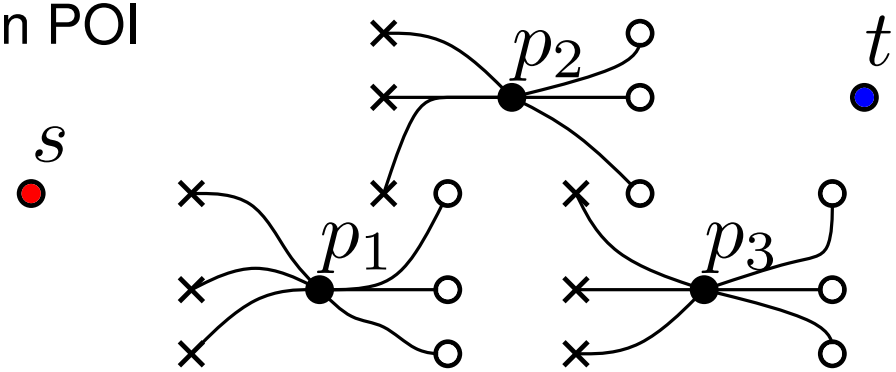
Bester Via Knoten

Szenario:

- Zielknoten sind POIs
- finde k best Via Knoten POIs von einem Startknoten s zu einem Startknoten t
- minimiere $\text{dist}(s, p) + \text{dist}(p, t)$ über alle POIs p

Lösung:

- Vorwärts- und Rückwartssuche von jedem POI
- speicher Kreuzprodukt der beiden Suchräume mit Distanz durch den POI
- Such von s und t: evaluiere jedes Paar
- Laufzeit: Suchraum$^2 \cdot k$



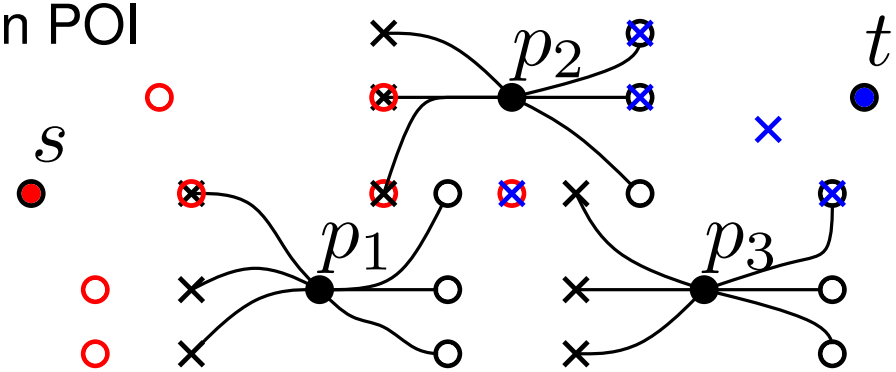
Bester Via Knoten

Szenario:

- Zielknoten sind POIs
- finde k best Via Knoten POIs von einem Startknoten s zu einem Startknoten t
- minimiere $\text{dist}(s, p) + \text{dist}(p, t)$ über alle POIs p

Lösung:

- Vorwärts- und Rückwartssuche von jedem POI
- speicher Kreuzprodukt der beiden Suchräume mit Distanz durch den POI
- Such von s und t: evaluiere jedes Paar
- Laufzeit: Suchraum$^2 \cdot k$



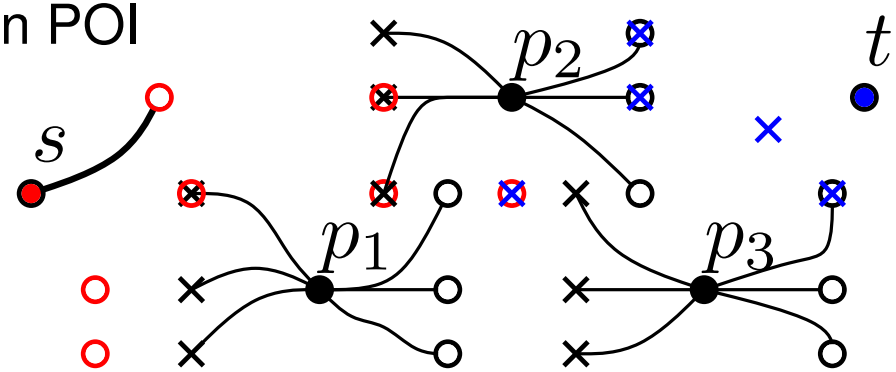
Bester Via Knoten

Szenario:

- Zielknoten sind POIs
- finde k best Via Knoten POIs von einem Startknoten s zu einem Startknoten t
- minimiere $\text{dist}(s, p) + \text{dist}(p, t)$ über alle POIs p

Lösung:

- Vorwärts- und Rückwartssuche von jedem POI
- speicher Kreuzprodukt der beiden Suchräume mit Distanz durch den POI
- Such von s und t: evaluiere jedes Paar
- Laufzeit: Suchraum$^2 \cdot k$



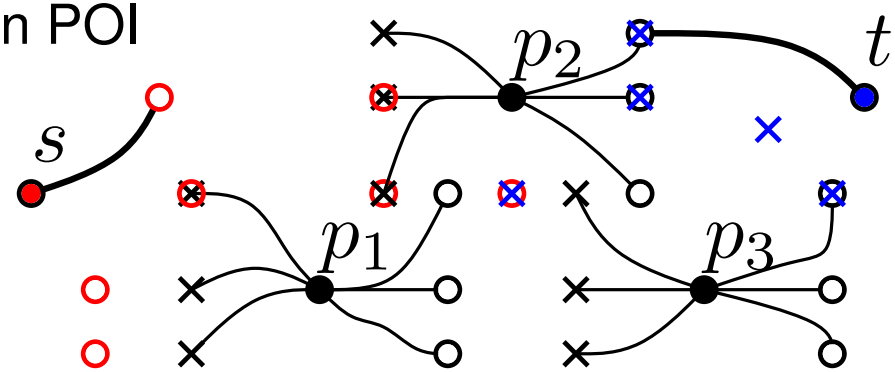
Bester Via Knoten

Szenario:

- Zielknoten sind POIs
- finde k best Via Knoten POIs von einem Startknoten s zu einem Startknoten t
- minimiere $\text{dist}(s, p) + \text{dist}(p, t)$ über alle POIs p

Lösung:

- Vorwärts- und Rückwartssuche von jedem POI
- speicher Kreuzprodukt der beiden Suchräume mit Distanz durch den POI
- Such von s und t: evaluiere jedes Paar
- Laufzeit: Suchraum$^2 \cdot k$



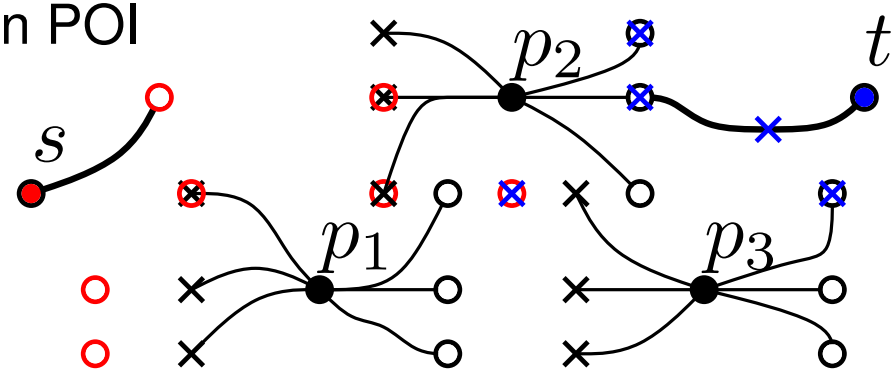
Bester Via Knoten

Szenario:

- Zielknoten sind POIs
- finde k best Via Knoten POIs von einem Startknoten s zu einem Startknoten t
- minimiere $\text{dist}(s, p) + \text{dist}(p, t)$ über alle POIs p

Lösung:

- Vorwärts- und Rückwartssuche von jedem POI
- speicher Kreuzprodukt der beiden Suchräume mit Distanz durch den POI
- Such von s und t: evaluiere jedes Paar
- Laufzeit: Suchraum$^2 \cdot k$



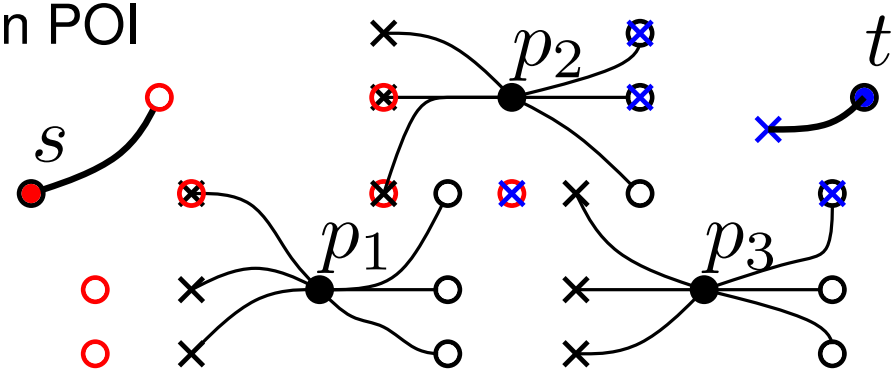
Bester Via Knoten

Szenario:

- Zielknoten sind POIs
- finde k best Via Knoten POIs von einem Startknoten s zu einem Startknoten t
- minimiere $\text{dist}(s, p) + \text{dist}(p, t)$ über alle POIs p

Lösung:

- Vorwärts- und Rückwartssuche von jedem POI
- speicher Kreuzprodukt der beiden Suchräume mit Distanz durch den POI
- Such von s und t: evaluiere jedes Paar
- Laufzeit: Suchraum$^2 \cdot k$



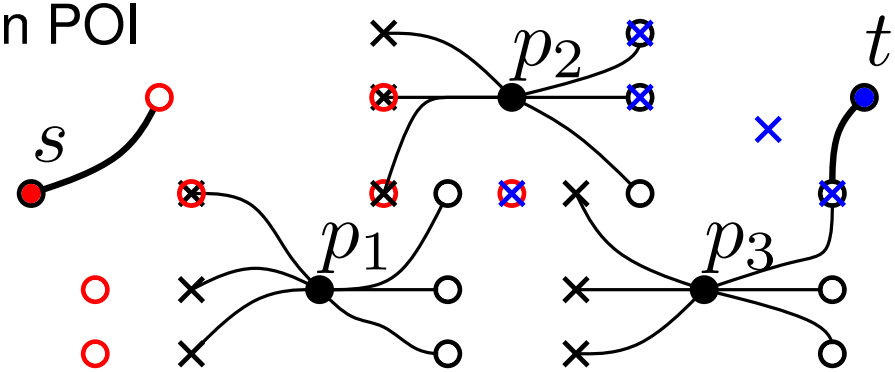
Bester Via Knoten

Szenario:

- Zielknoten sind POIs
- finde k best Via Knoten POIs von einem Startknoten s zu einem Startknoten t
- minimiere $\text{dist}(s, p) + \text{dist}(p, t)$ über alle POIs p

Lösung:

- Vorwärts- und Rückwartssuche von jedem POI
- speicher Kreuzprodukt der beiden Suchräume mit Distanz durch den POI
- Such von s und t: evaluiere jedes Paar
- Laufzeit: Suchraum$^2 \cdot k$



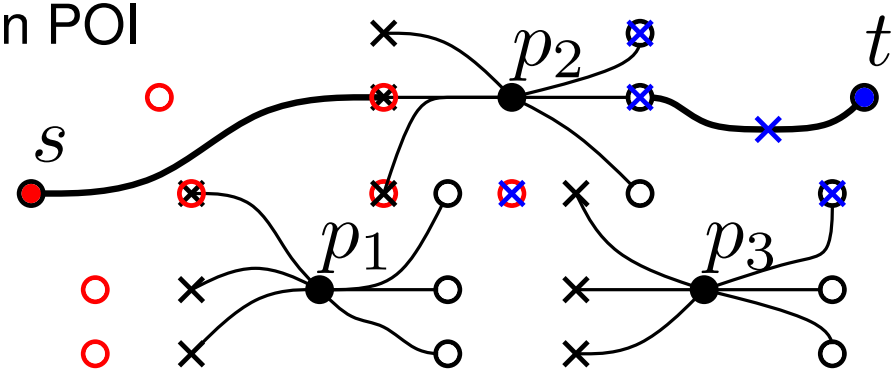
Bester Via Knoten

Szenario:

- Zielknoten sind POIs
- finde k best Via Knoten POIs von einem Startknoten s zu einem Startknoten t
- minimiere $\text{dist}(s, p) + \text{dist}(p, t)$ über alle POIs p

Lösung:

- Vorwärts- und Rückwartssuche von jedem POI
- speicher Kreuzprodukt der beiden Suchräume mit Distanz durch den POI
- Such von s und t: evaluiere jedes Paar
- Laufzeit: Suchraum$^2 \cdot k$



Aufgabenstellung

Beobachtung:

- Queries sind schnell genug
- Visualisierung und Netzwerklatenz der Flaschenhals
- schwieriger und hoch optimierter Code

Aufgabenstellung

Beobachtung:

- Queries sind schnell genug
- Visualisierung und Netzwerklatenz der Flaschenhals
- schwieriger und hoch optimierter Code

Können wir Geschwindigkeit gegen einfachere Bedienbarkeit eintauschen?

Aufgabenstellung

Beobachtung:

- Queries sind schnell genug
- Visualisierung und Netzwerklatenz der Flaschenhals
- schwieriger und hoch optimierter Code

Können wir Geschwindigkeit gegen einfachere Bedienbarkeit eintauschen?

Idee:

- Implementier Routenplanung direkt in SQL
- auch die Erweiterungen

Location Services in SQL

Vorteile:

- einfach zu nutzen
- Daten meist eh schon in SQL
- skalieren einfach (bestehende Datenbanksysteme, Cloud SQL)
- auch für Nicht-Routing-Experten zu nutzen
- External Memory Implementation “umsonst”

Location Services in SQL

Vorteile:

- einfach zu nutzen
- Daten meist eh schon in SQL
- skalieren einfach (bestehende Datenbanksysteme, Cloud SQL)
- auch für Nicht-Routing-Experten zu nutzen
- External Memory Implementation “umsonst”

Nachteile:

- SQL viel langsamer als optimierter C++ Code
- keine aufwändigen Datenstrukturen (Graph, Priority Queue)
- Dijkstra-basierte Techniken sind keine Option

Ideensammlung

Welcher Ansatz?

Ideensammlung

Welcher Ansatz?

- keine Priority Queue?

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Ideensammlung

Welcher Ansatz?

- keine Priority Queue?
- keine Graphdatenstruktur?

Ideensammlung

Welcher Ansatz?

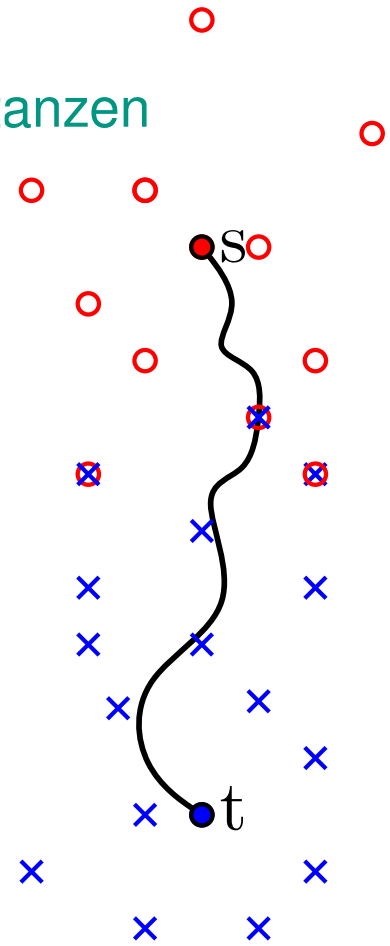
- keine Priority Queue?
- keine Graphdatenstruktur?

Idee: Hub Labeling

Hub 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:

- Label der Grösse ≈ 70 für Europe
- sehr schnelle Queryzeiten
- Set operationen

Speichern der Label

Idee:

- berechne Label mit C++ (wie bei HubLabels)
- aber speicher die Label direkt in der Datenbank
- ein Vorwärtslabel von Knoten v mit k Hubs:
  - erzeugt k Triples $(v, u, d(v, u))$ in Tabelle `forward`
- Rückwärtslabel genauso in `backward`
- ungefähr 1.35 Milliarden Zeilen pro Tabelle (ca. 19 GB pro Richtung)

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

$L_b(2)$ | 2,0 | 6,1 | 7,4 |

forward			backward		
node	hub	dist	node	hub	dist
1	1	0	1	1	0
1	4	1	1	4	4
1	5	2	2	2	0
1	7	3	2	6	1
2	14	2	2	7	4
⋮	⋮	⋮	⋮	⋮	⋮

Speichern der Label

Idee:

- berechne Label mit C++ (wie bei HubLabels)
- aber speicher die Label direkt in der Datenbank
- ein Vorwärtslabel von Knoten v mit k Hubs:
  - erzeugt k Triples $(v, u, d(v, u))$ in Tabelle `forward`
- Rückwärtslabel genauso in `backward`
- ungefähr 1.35 Milliarden Zeilen pro Tabelle (ca. 19 GB pro Richtung)
- indiziere nach `node` (primary) und `hub` (secondary)

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

$L_b(2)$ | 2,0 | 6,1 | 7,4 |

forward			backward		
node	hub	dist	node	hub	dist
1	1	0	1	1	0
1	4	1	1	4	4
1	5	2	2	2	0
1	7	3	2	6	1
2	14	2	2	7	4
⋮	⋮	⋮	⋮	⋮	⋮

SQL Query

Algorithm 1: SQL_DIST

Input: source $s \in V$, target $t \in V$

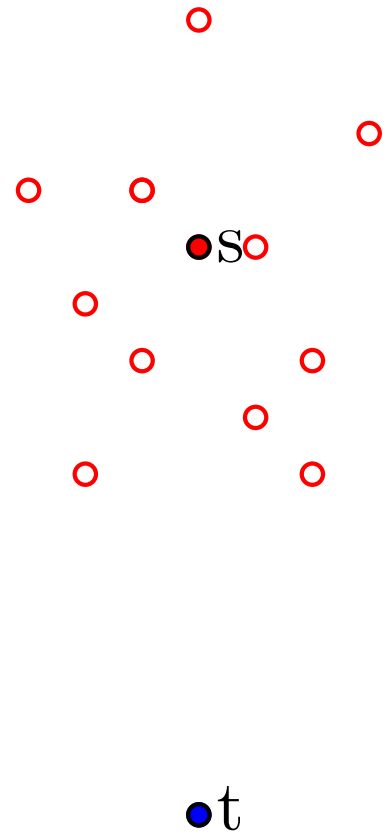
```
1 SELECT
2       MIN(forward.dist+backward.dist)
3 FROM forward,backward
4 WHERE
5       forward.node = s AND
6       backward.node = t AND
7       forward.hub = backward.hub
```

SQL Query

Algorithm 1: SQL_DIST

Input: source $s \in V$, target $t \in V$

```
1 SELECT
2       MIN(forward.dist+backward.dist)
3 FROM forward,backward
4 WHERE
5       forward.node = s AND
6       backward.node = t AND
7       forward.hub = backward.hub
```

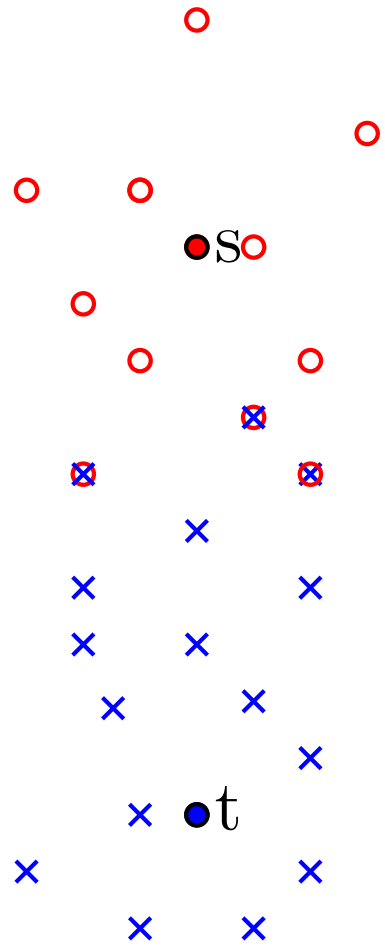


SQL Query

Algorithm 1: SQL_DIST

Input: source $s \in V$, target $t \in V$

```
1 SELECT
2       MIN(forward.dist+backward.dist)
3 FROM forward,backward
4 WHERE
5       forward.node = s AND
6       backward.node = t AND
7       forward.hub = backward.hub
```

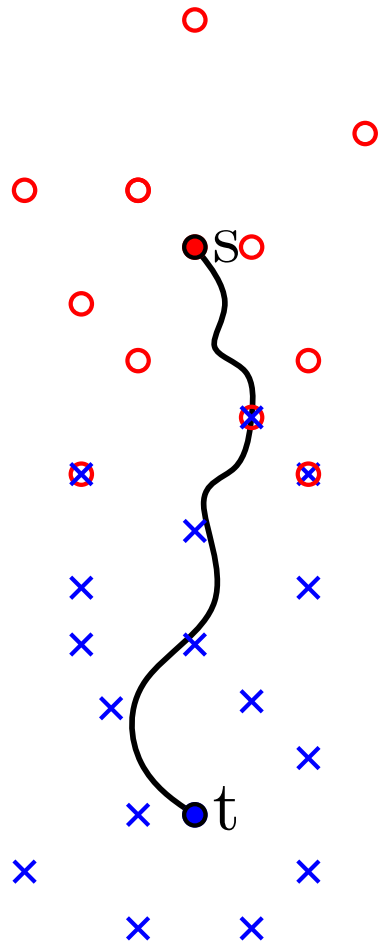


SQL Query

Algorithm 1: SQL_DIST

Input: source $s \in V$, target $t \in V$

```
1 SELECT
2       MIN(forward.dist+backward.dist)
3 FROM forward,backward
4 WHERE
5       forward.node = s AND
6       backward.node = t AND
7       forward.hub = backward.hub
```

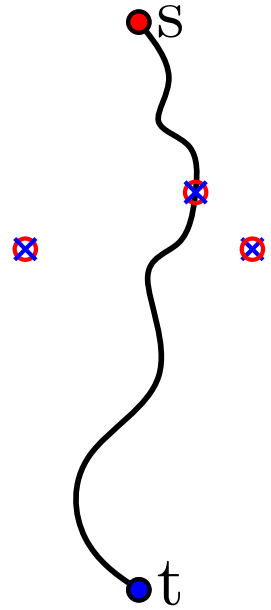


SQL Query

Algorithm 1: SQL_DIST

Input: source $s \in V$, target $t \in V$

```
1 SELECT
2       MIN(forward.dist+backward.dist)
3 FROM forward,backward
4 WHERE
5       forward.node = s AND
6       backward.node = t AND
7       forward.hub = backward.hub
```

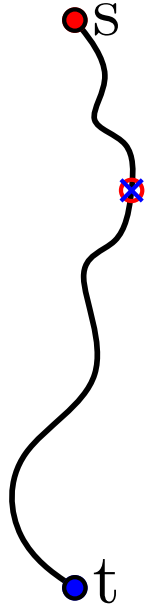


SQL Query

Algorithm 1: SQL_DIST

Input: source $s \in V$, target $t \in V$

```
1 SELECT
2       MIN(forward.dist+backward.dist)
3 FROM forward,backward
4 WHERE
5       forward.node = s AND
6       backward.node = t AND
7       forward.hub = backward.hub
```

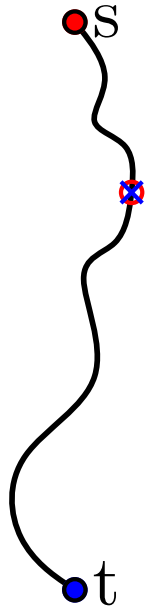


SQL Query

Algorithm 1: SQL_DIST

Input: source $s \in V$, target $t \in V$

```
1 SELECT
2       MIN(forward.dist+backward.dist)
3 FROM forward,backward
4 WHERE
5       forward.node = s AND
6       backward.node = t AND
7       forward.hub = backward.hub
```



Bemerkung:

- berechnet nur die Distanz

Pfadentpackung

Idee:

- 2 Phasen
- speicher jeden Shortcut aus G^+ explizit (als Sequenz von Kanten IDs) in Tabelle `shortcuts`
- ca. 5 GB in Tabelle

Phase 1:

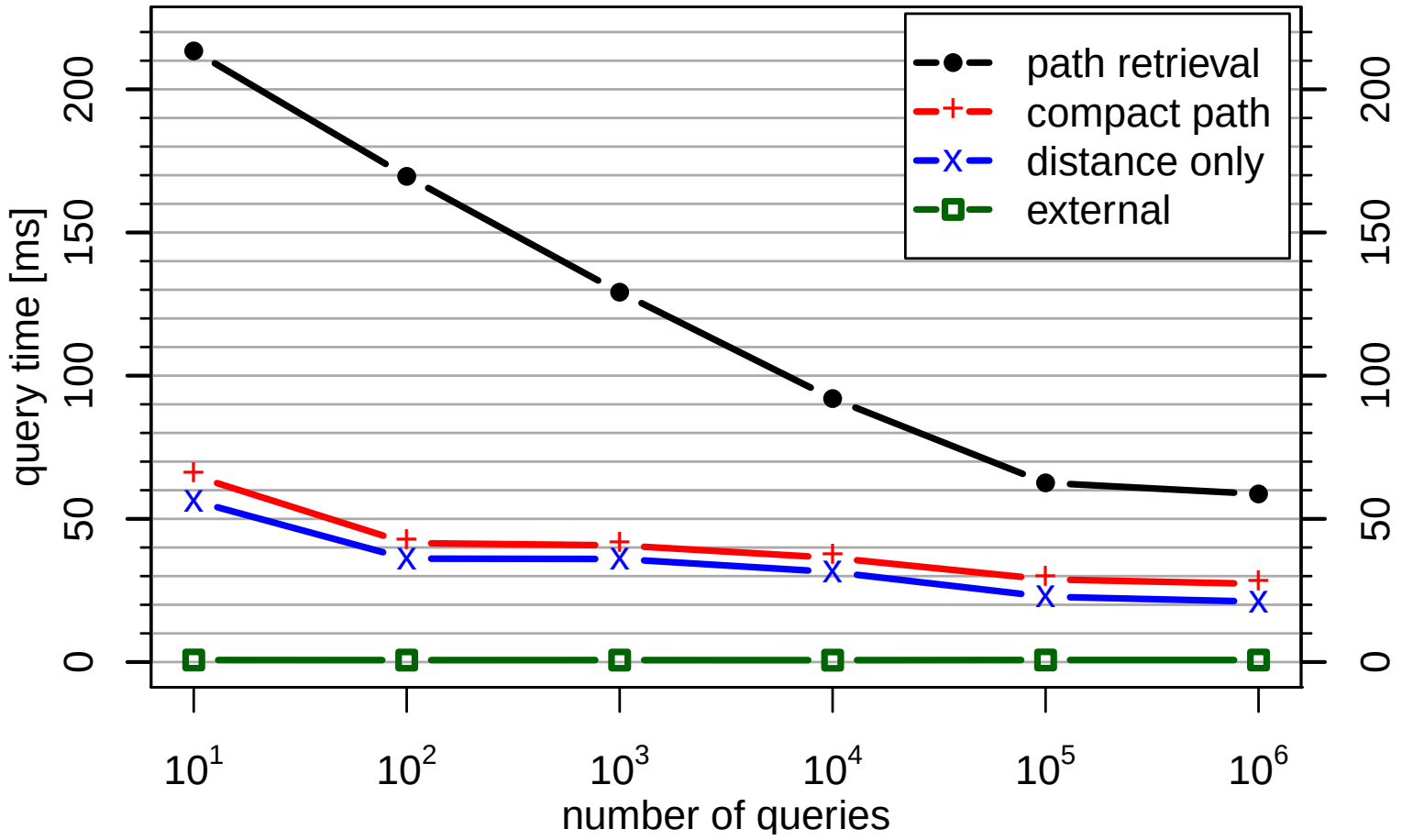
- erzeuge Pfad in G^+ durch Hubs auf dem Pfad
- erweitere Tabellen `forward` und `backward` um 2 Spalten: Parent und Shortcut
- erhöht Speicherverbrauch der Tabelle von 19 auf 32 GB

Phase 2:

- erzeuge Pfad in G durch matchen von G^+ mit `shortcuts`

Ergebnisse

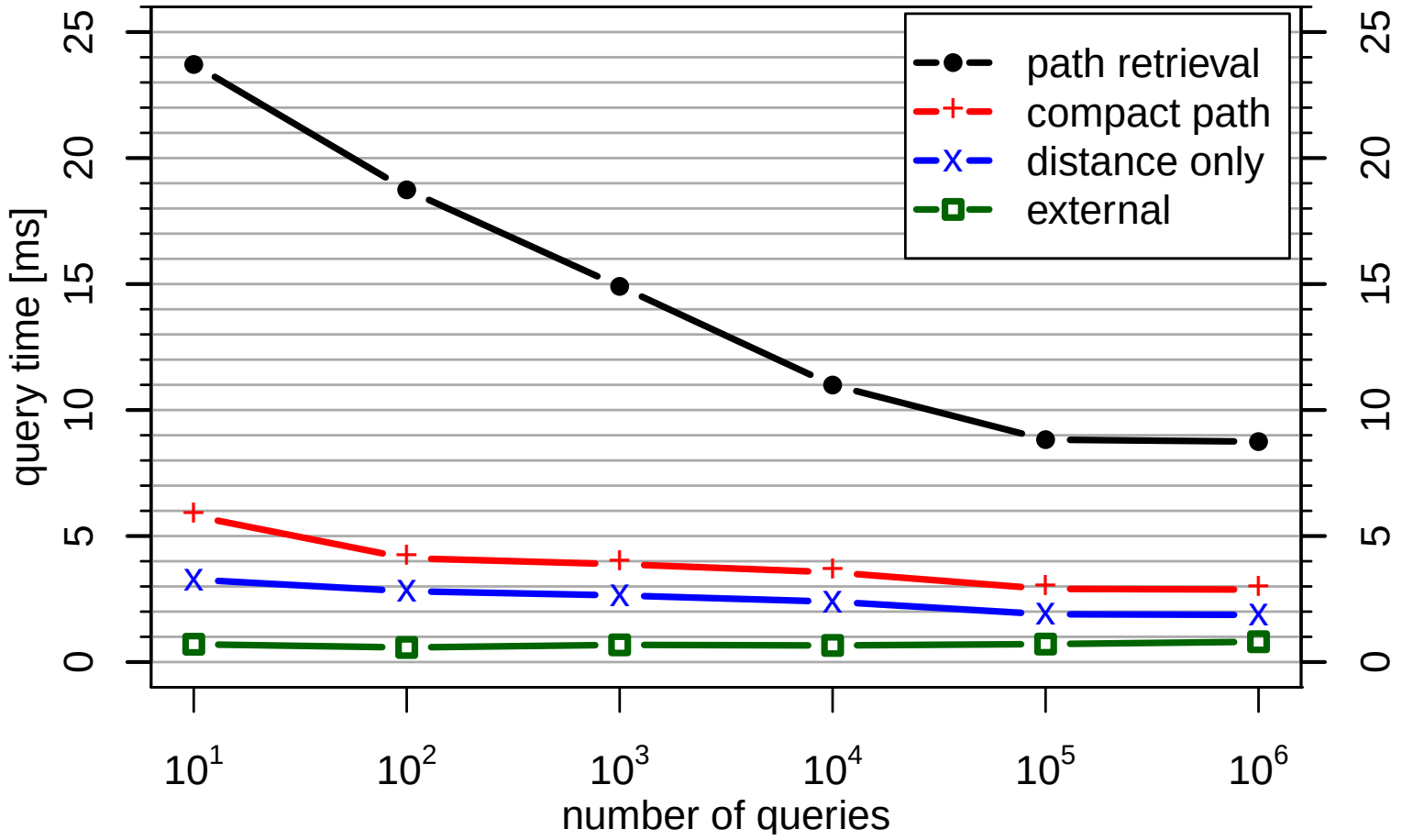
Setup: MS SQL Server 2008 R2 mit Daten auf HDD, kalter Cache



Beobachtung: Nicht schnell genug

Ergebnisse (SSD)

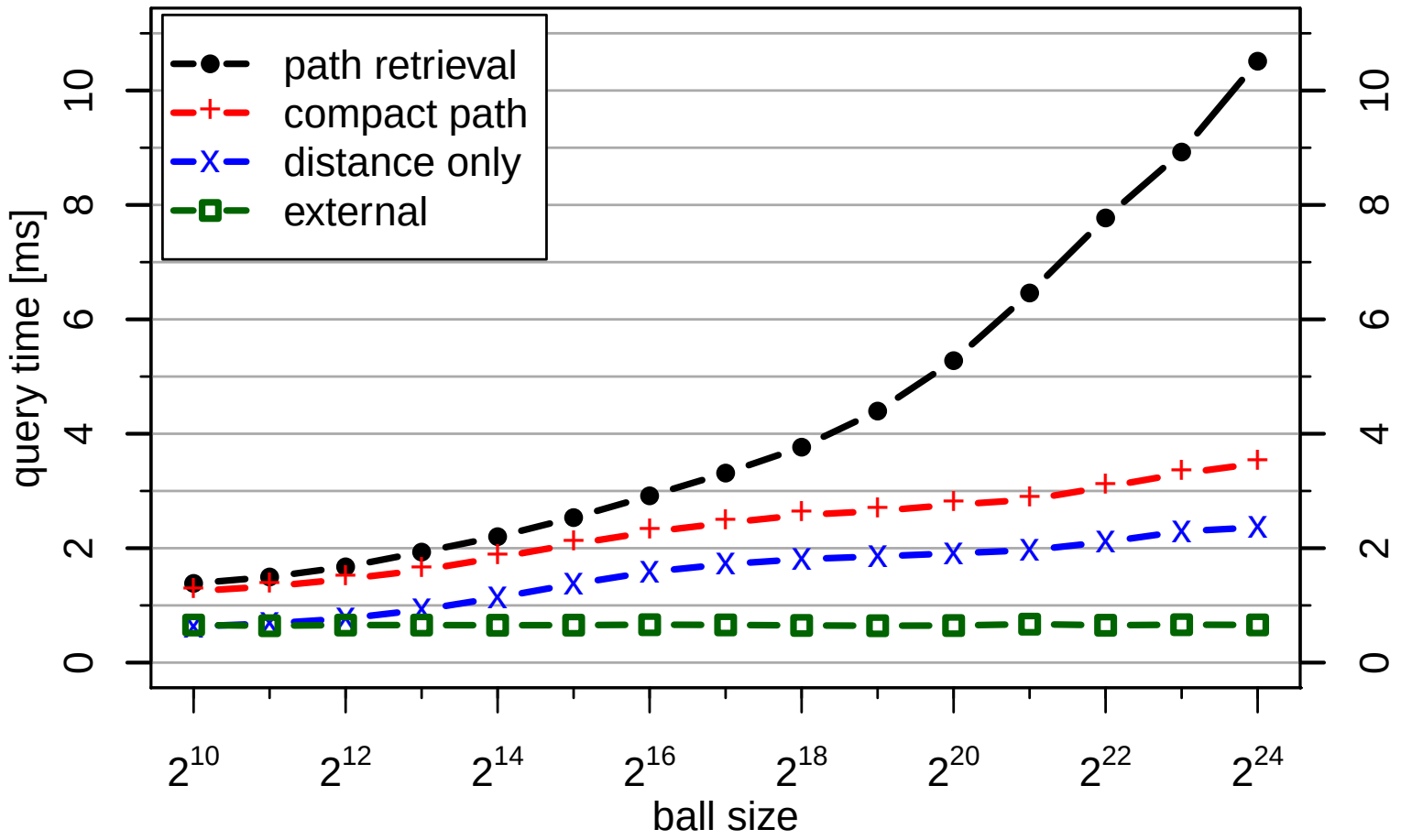
Setup: MS SQL Server 2008 R2 mit Daten auf SSD, kalter Cache



Beobachtung: SSD macht Queries schnell genug

Lokale Anfrage

Setup: Anfragen mit verschiedenem Rank, 10000 Anfragen, kalter Cache

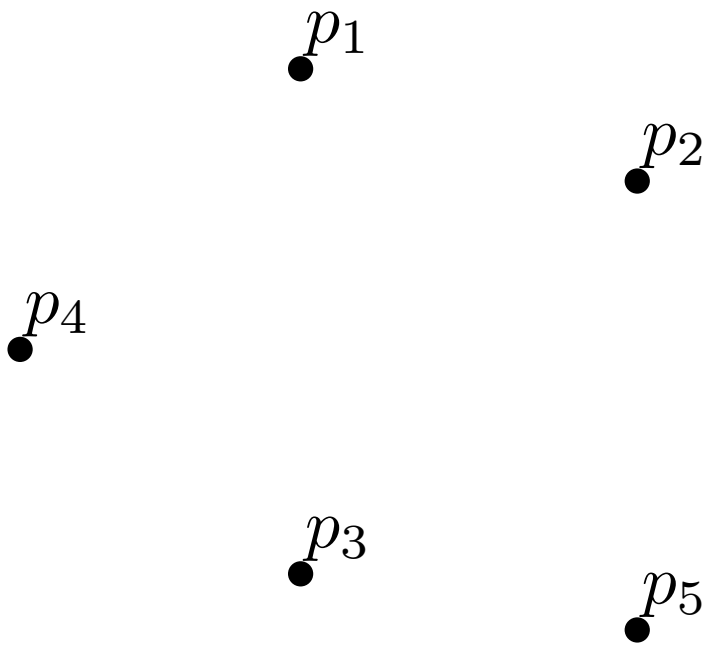


Beobachtung: praxisrelevante Anfragen sehr schnell

POI Anfragen in SQL

Idee:

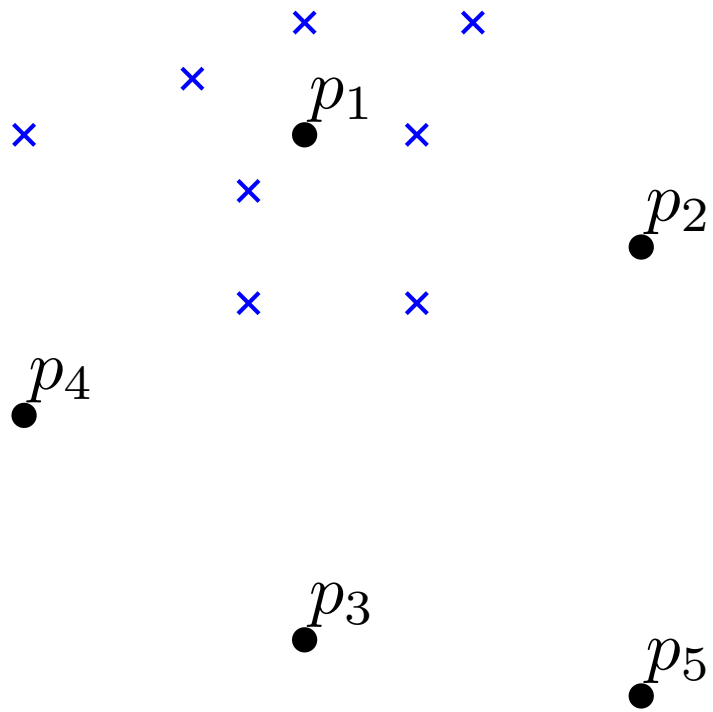
- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`



POI Anfragen in SQL

Idee:

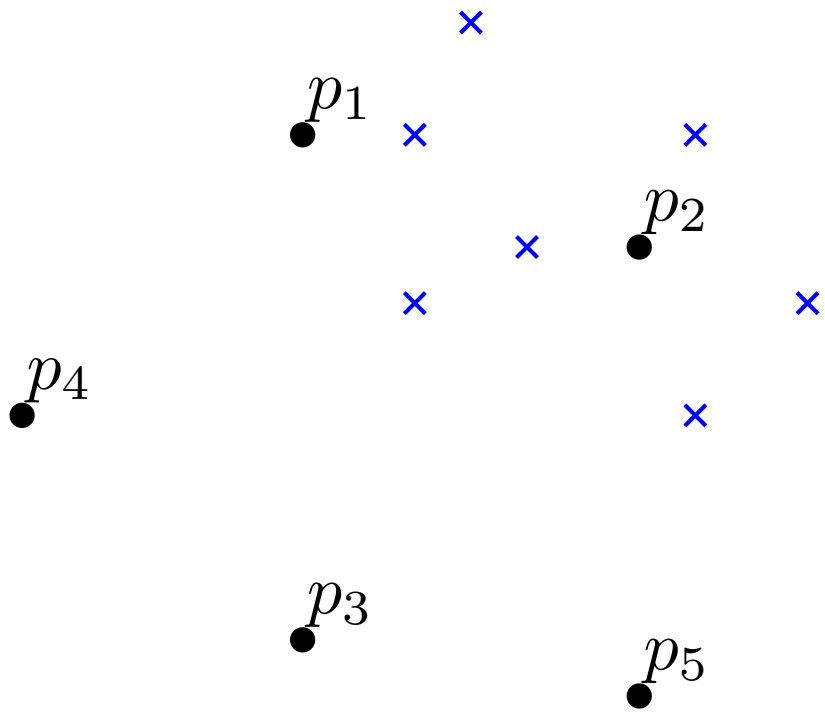
- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`



POI Anfragen in SQL

Idee:

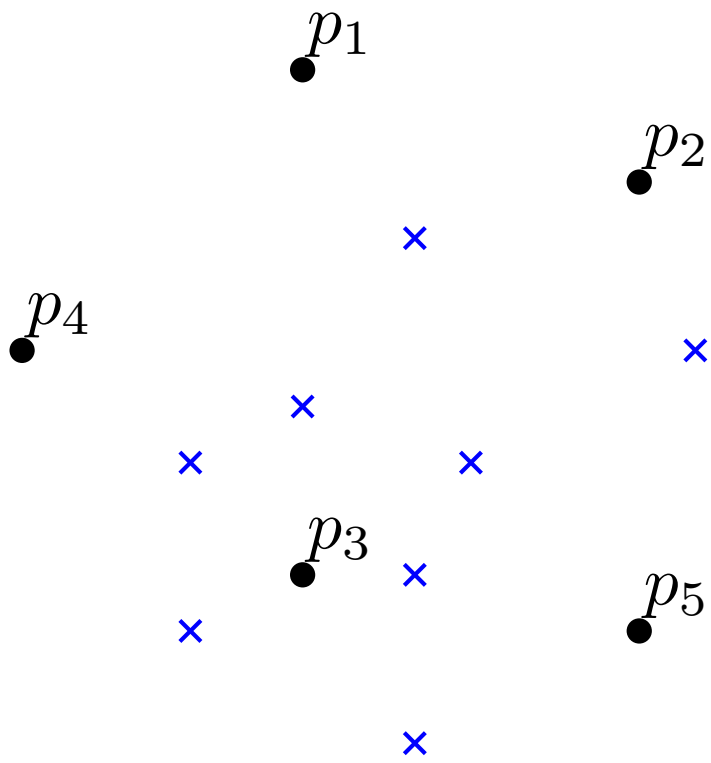
- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`



POI Anfragen in SQL

Idee:

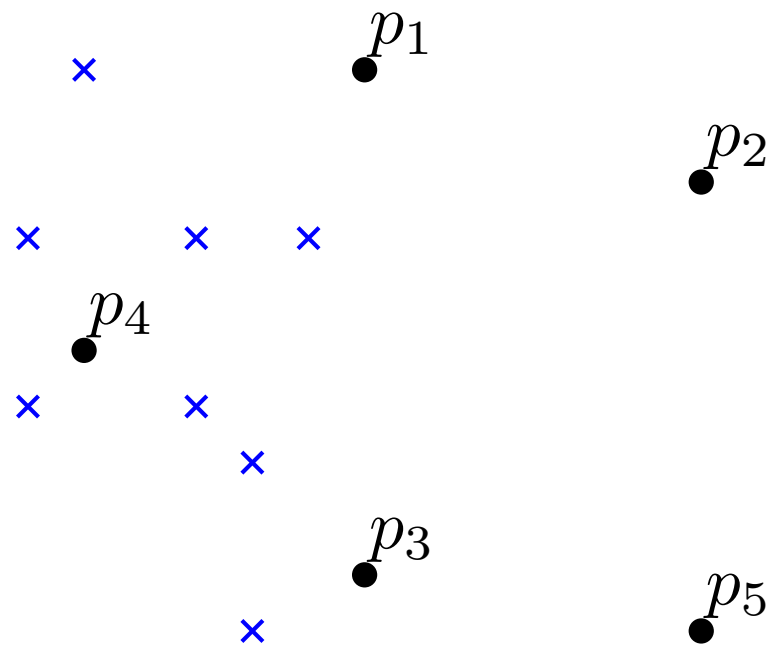
- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`



POI Anfragen in SQL

Idee:

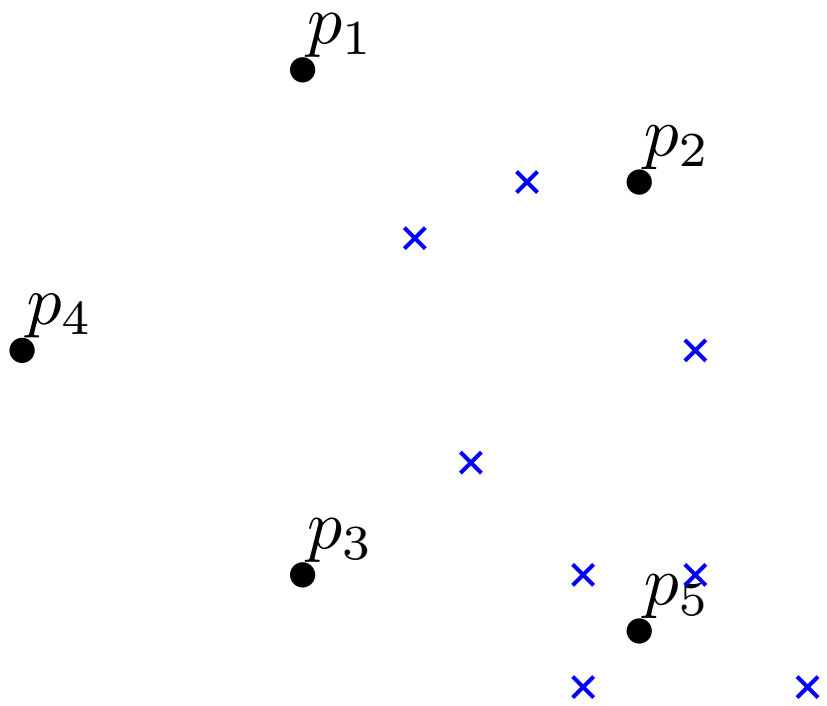
- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`



POI Anfragen in SQL

Idee:

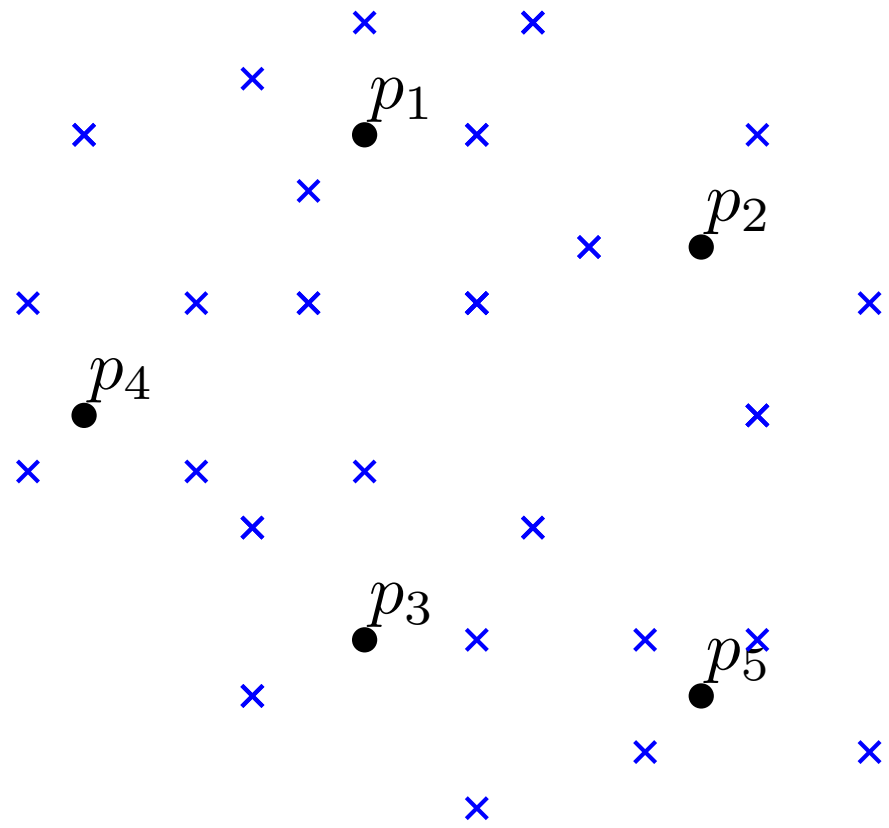
- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`



POI Anfragen in SQL

Idee:

- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`



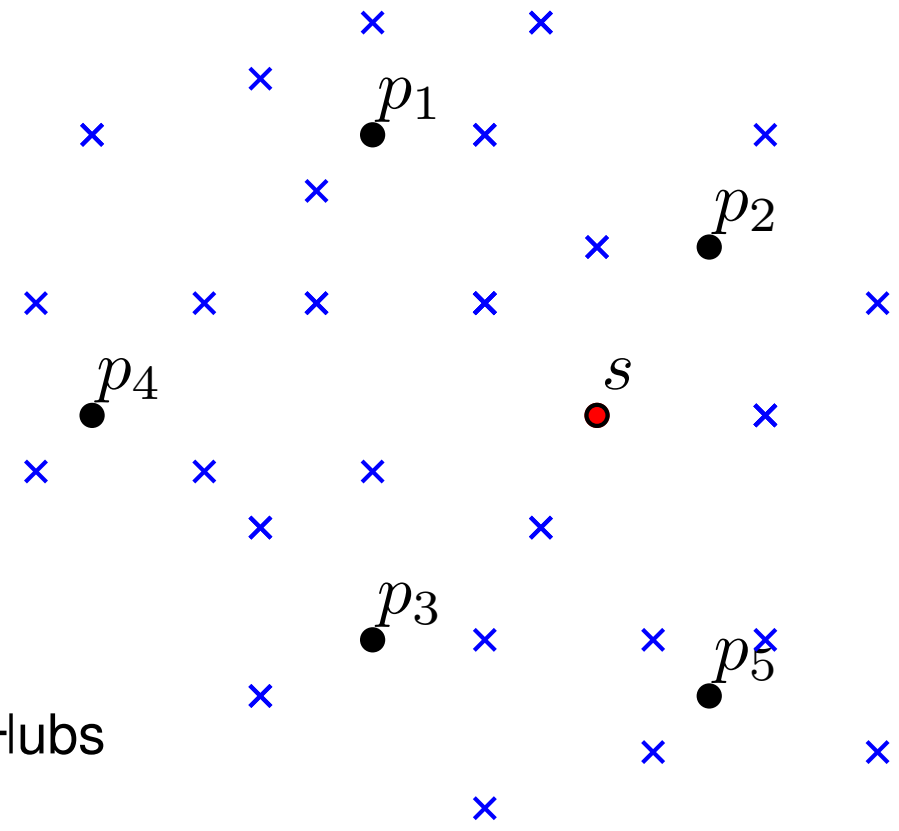
POI Anfragen in SQL

Idee:

- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`

Query:

- iteriert über alle ausgehenden Hubs
- für jeden Hub werden nur die (k) nächsten POIs betrachtet
- Antwort: die k insgesamt nächsten POIs



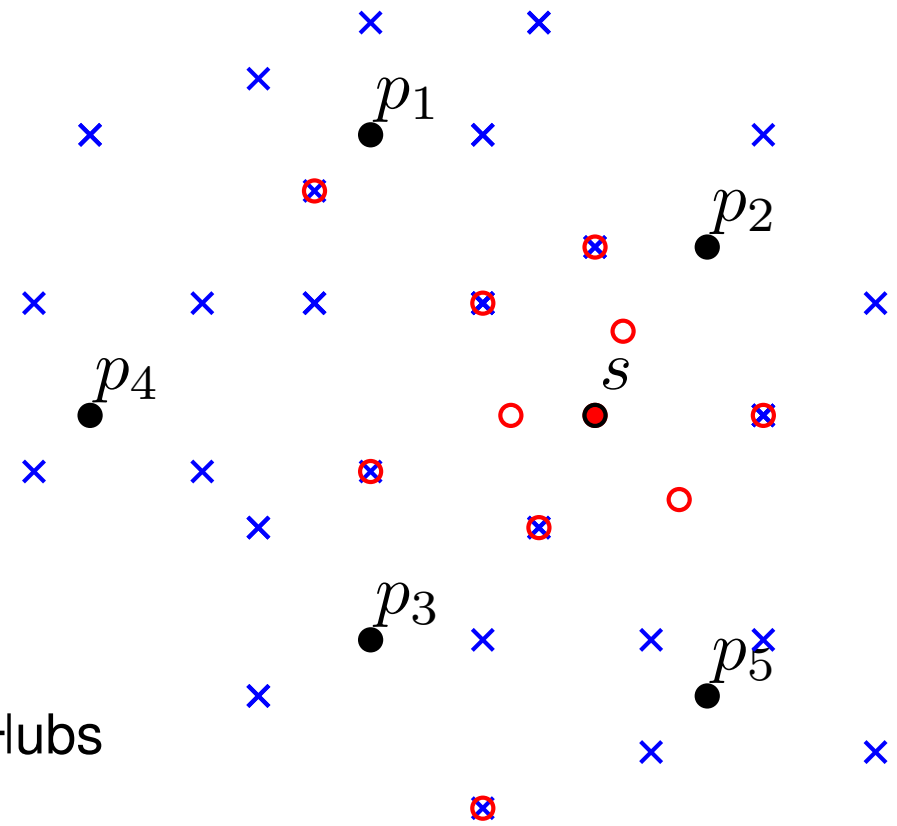
POI Anfragen in SQL

Idee:

- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`

Query:

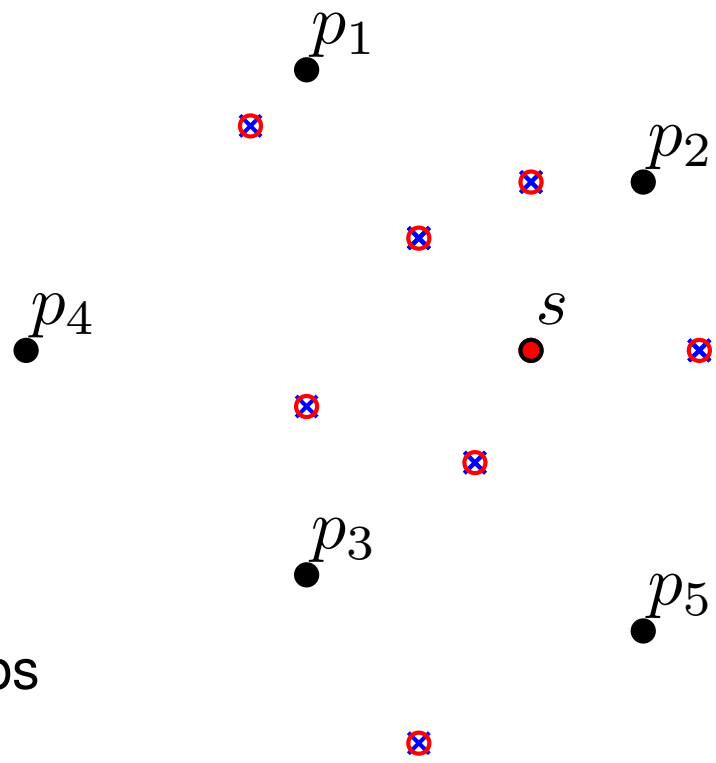
- iteriert über alle ausgehenden Hubs
- für jeden Hub werden nur die (k) nächsten POIs betrachtet
- Antwort: die k insgesamt nächsten POIs



POI Anfragen in SQL

Idee:

- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`



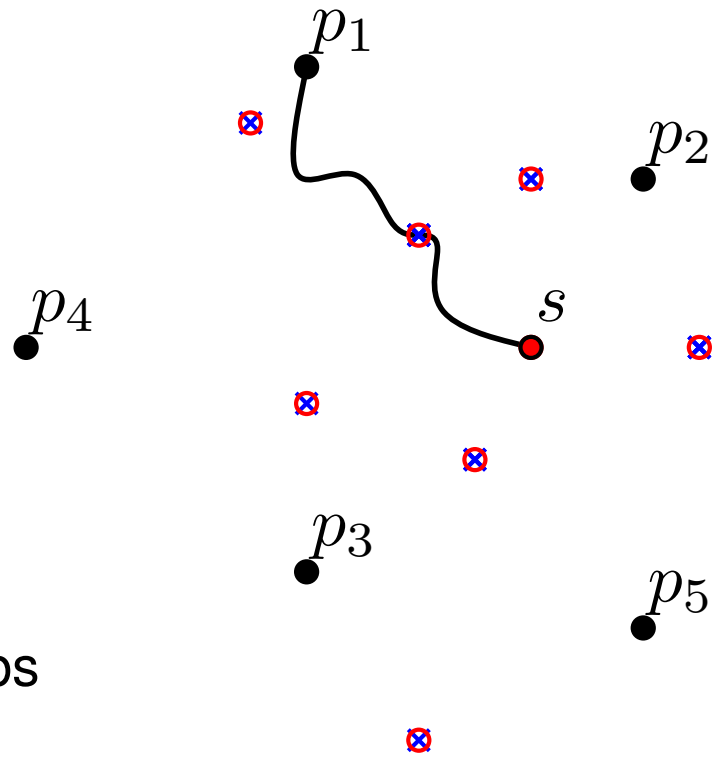
Query:

- iteriert über alle ausgehenden Hubs
- für jeden Hub werden nur die (k) nächsten POIs betrachtet
- Antwort: die k insgesamt nächsten POIs

POI Anfragen in SQL

Idee:

- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`



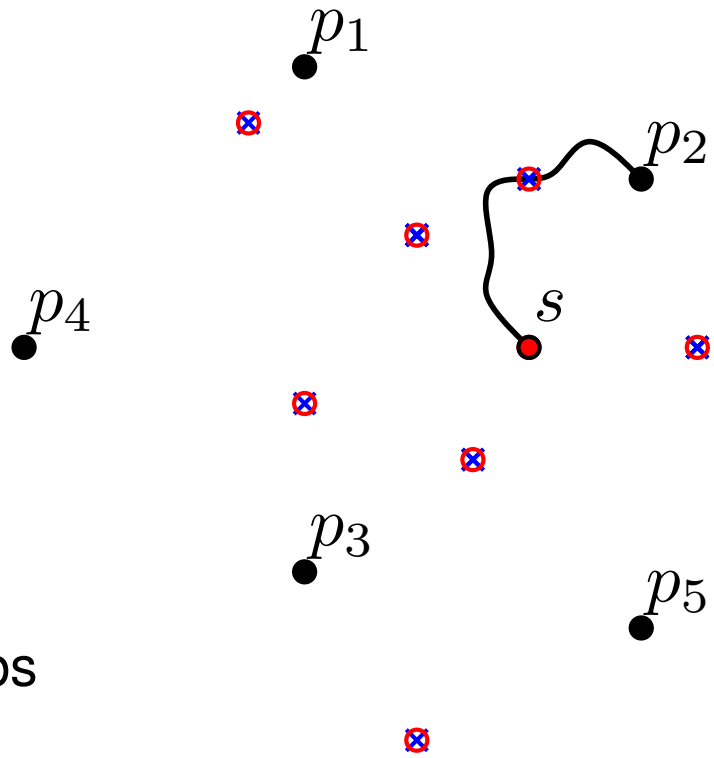
Query:

- iteriert über alle ausgehenden Hubs
- für jeden Hub werden nur die (k) nächsten POIs betrachtet
- Antwort: die k insgesamt nächsten POIs

POI Anfragen in SQL

Idee:

- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`



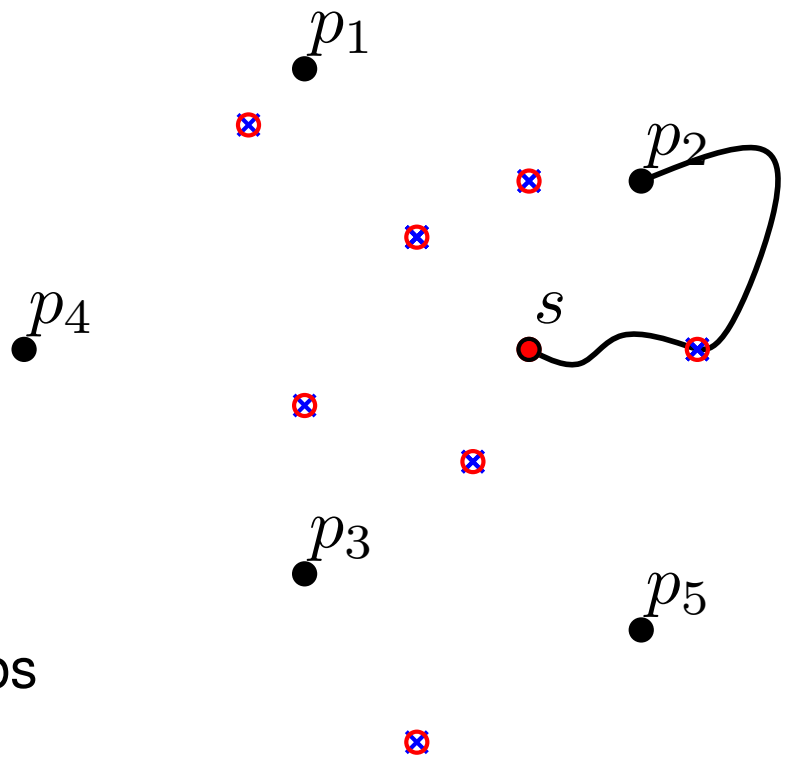
Query:

- iteriert über alle ausgehenden Hubs
- für jeden Hub werden nur die (k) nächsten POIs betrachtet
- Antwort: die k insgesamt nächsten POIs

POI Anfragen in SQL

Idee:

- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`



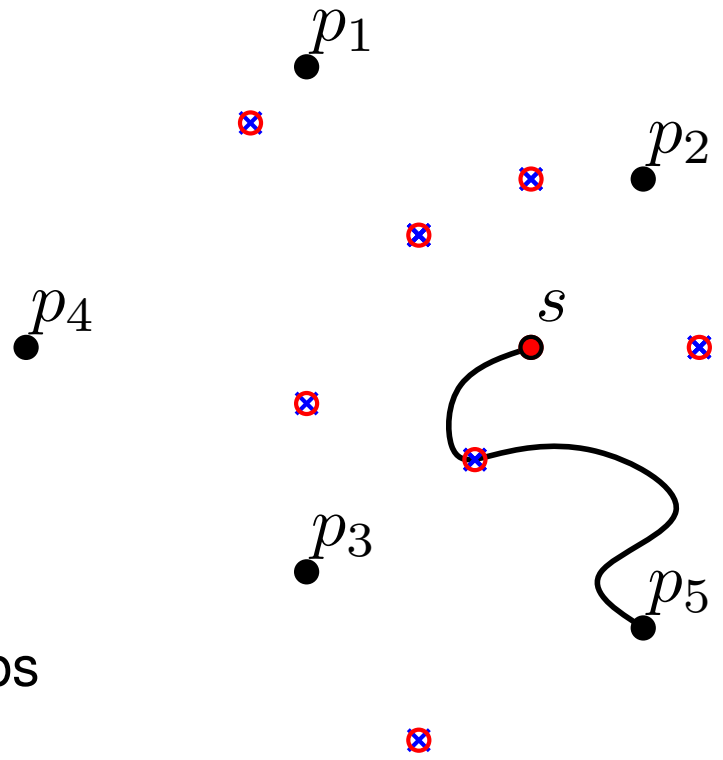
Query:

- iteriert über alle ausgehenden Hubs
- für jeden Hub werden nur die (k) nächsten POIs betrachtet
- Antwort: die k insgesamt nächsten POIs

POI Anfragen in SQL

Idee:

- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`



Query:

- iteriert über alle ausgehenden Hubs
- für jeden Hub werden nur die (k) nächsten POIs betrachtet
- Antwort: die k insgesamt nächsten POIs

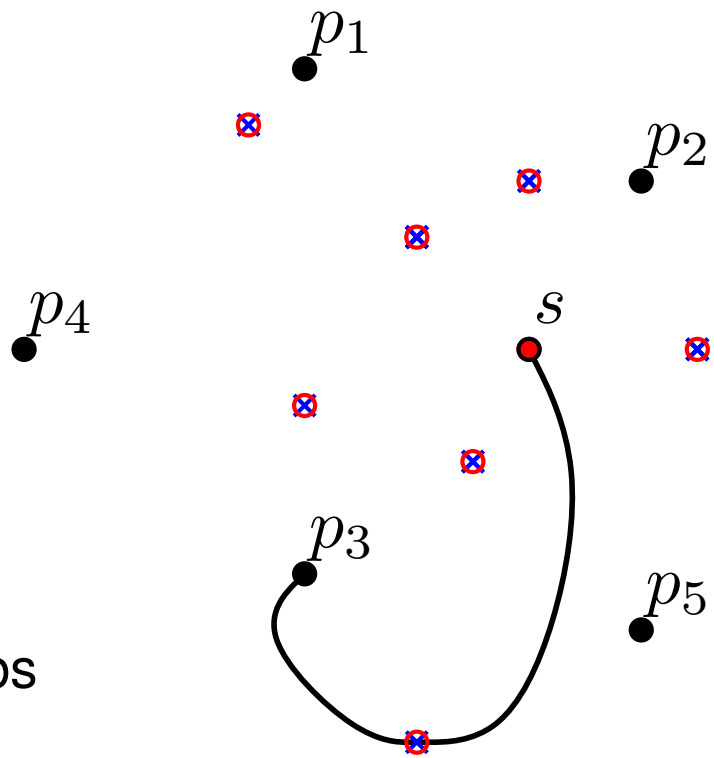
POI Anfragen in SQL

Idee:

- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`

Query:

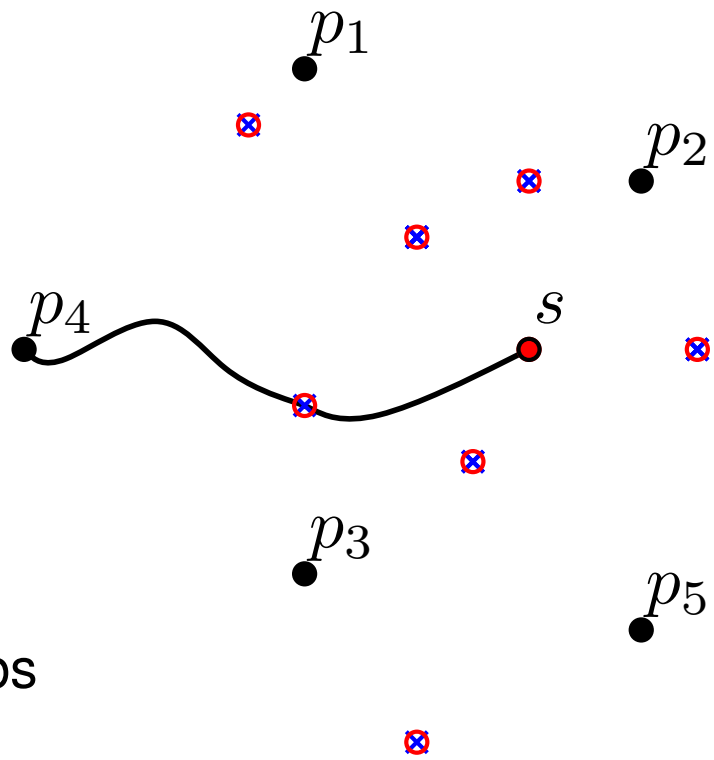
- iteriert über alle ausgehenden Hubs
- für jeden Hub werden nur die (k) nächsten POIs betrachtet
- Antwort: die k insgesamt nächsten POIs



POI Anfragen in SQL

Idee:

- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`



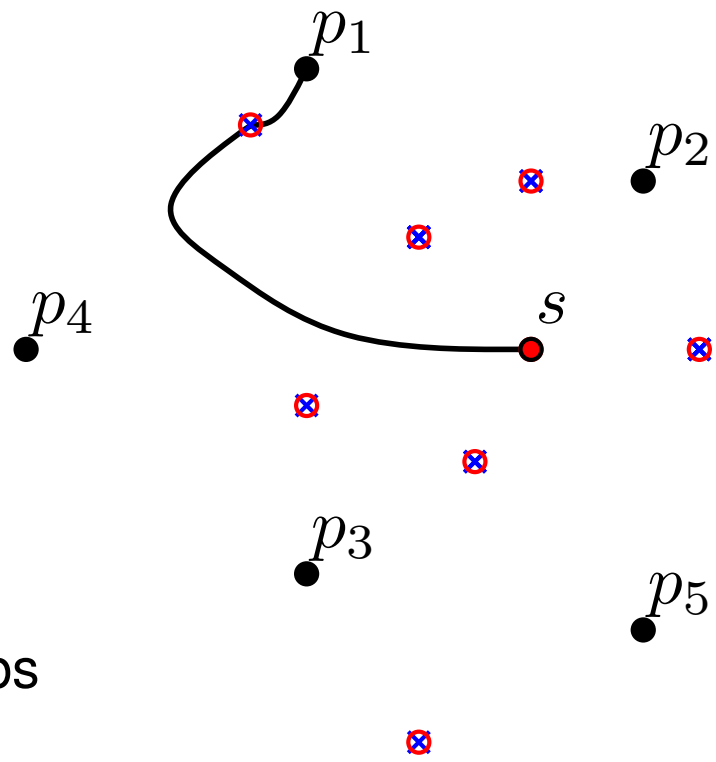
Query:

- iteriert über alle ausgehenden Hubs
- für jeden Hub werden nur die (k) nächsten POIs betrachtet
- Antwort: die k insgesamt nächsten POIs

POI Anfragen in SQL

Idee:

- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`



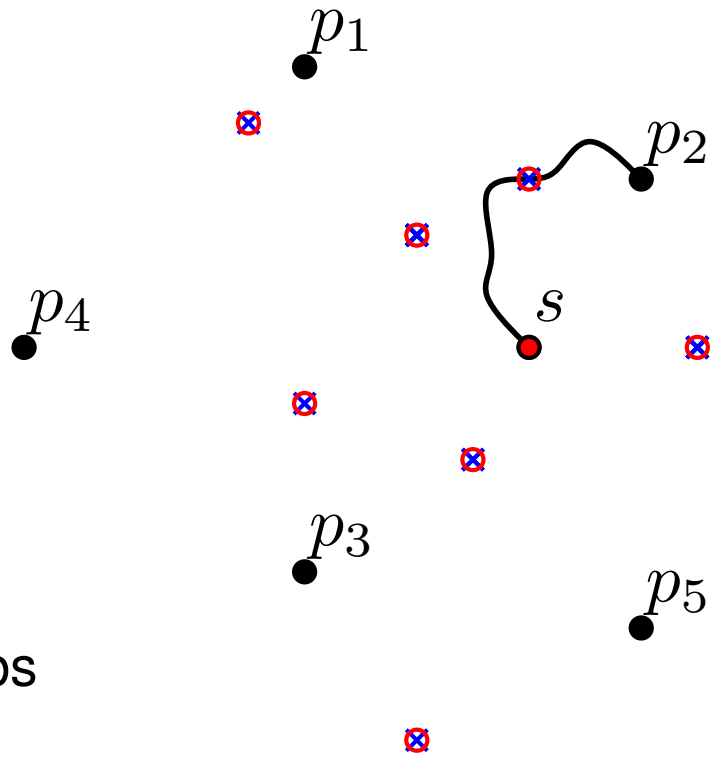
Query:

- iteriert über alle ausgehenden Hubs
- für jeden Hub werden nur die (k) nächsten POIs betrachtet
- Antwort: die k insgesamt nächsten POIs

POI Anfragen in SQL

Idee:

- extrahiere die Rückwartslabel aus `backward`
- speicher sie in neuer Tabelle `poilab`
- indiziere nach `hub` und `dist`

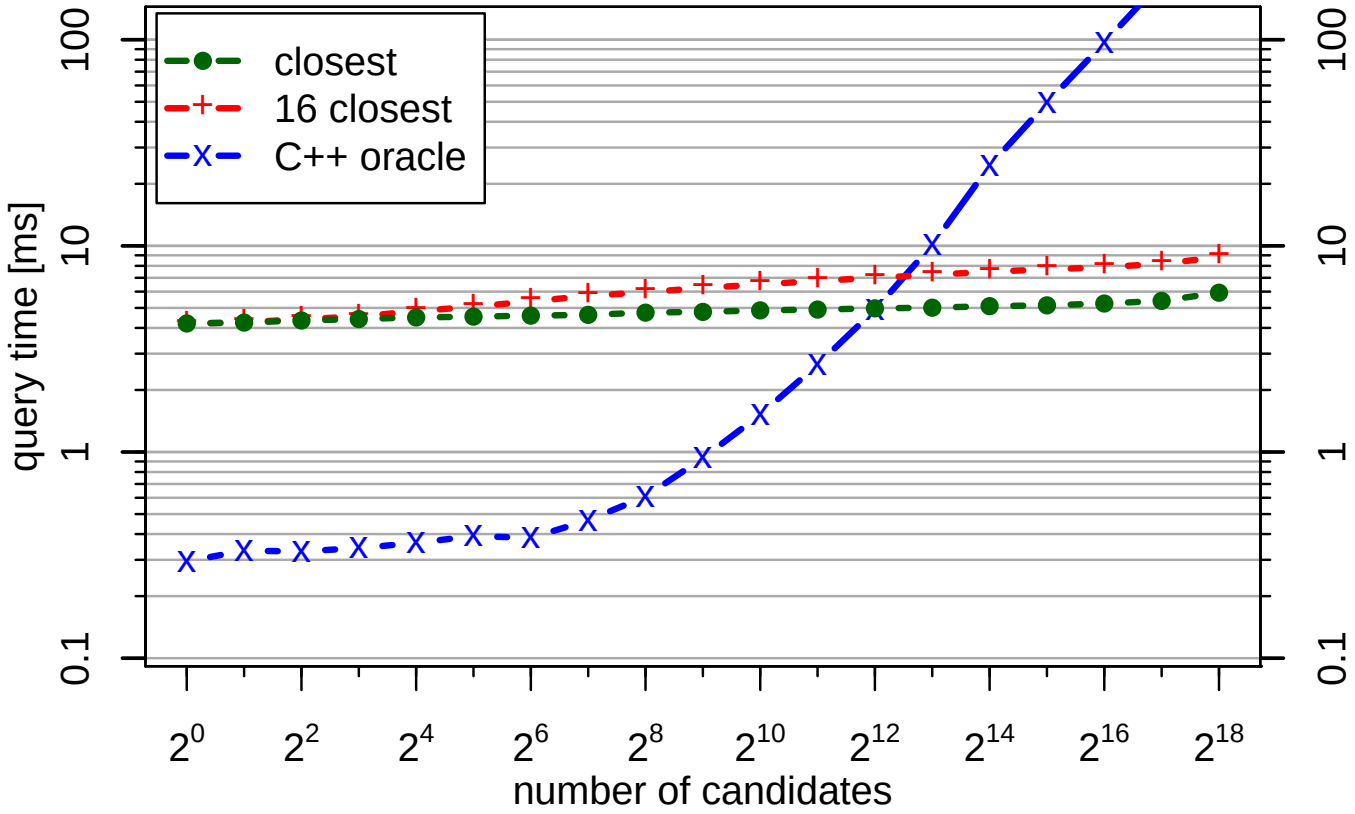


Query:

- iteriert über alle ausgehenden Hubs
- für jeden Hub werden nur die (k) nächsten POIs betrachtet
- Antwort: die k insgesamt nächsten POIs

Ergebnisse POI

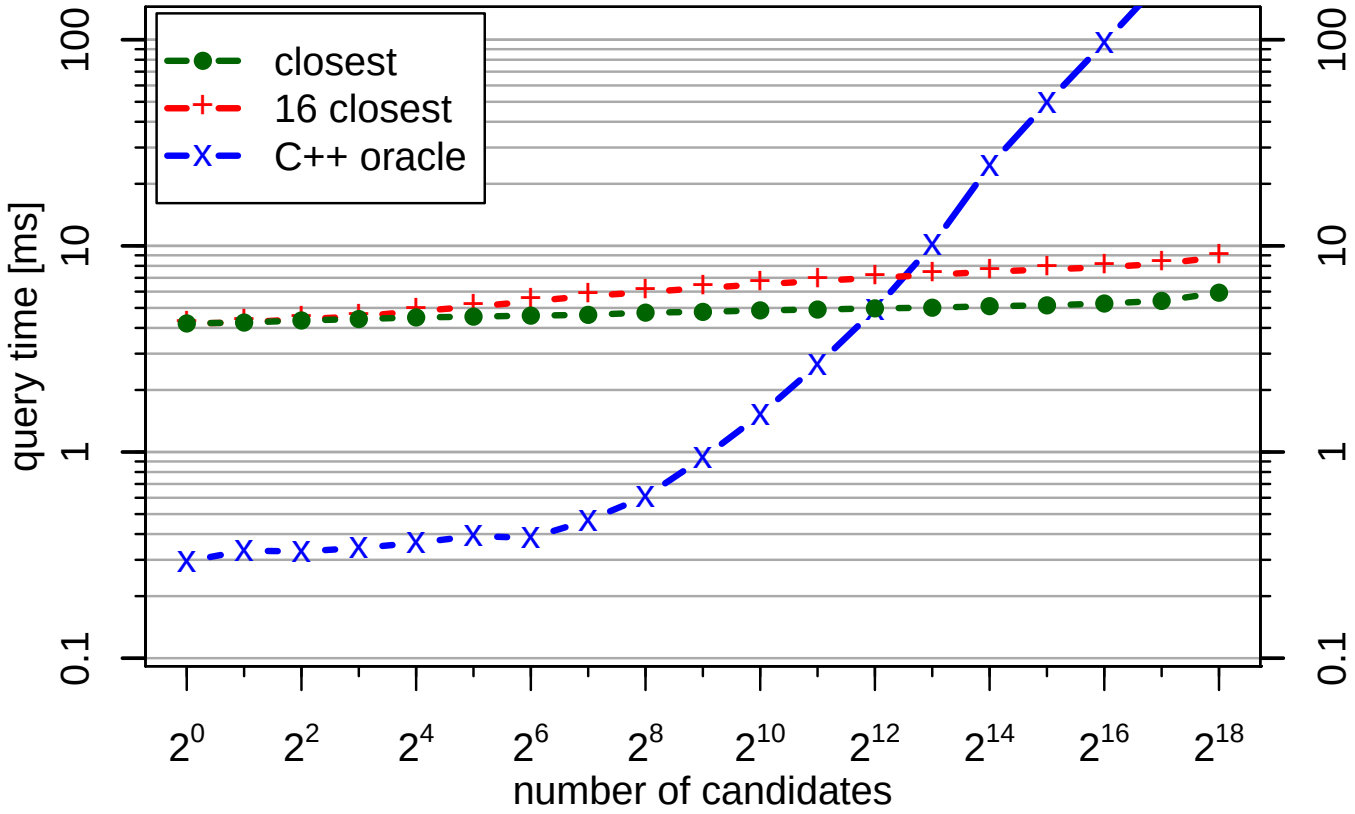
Setup: verschiedene Anzahl POIs, zufällig gewählt



- externes Punkt-zu-Punkt Orakel: skaliert schlecht

Ergebnisse POI

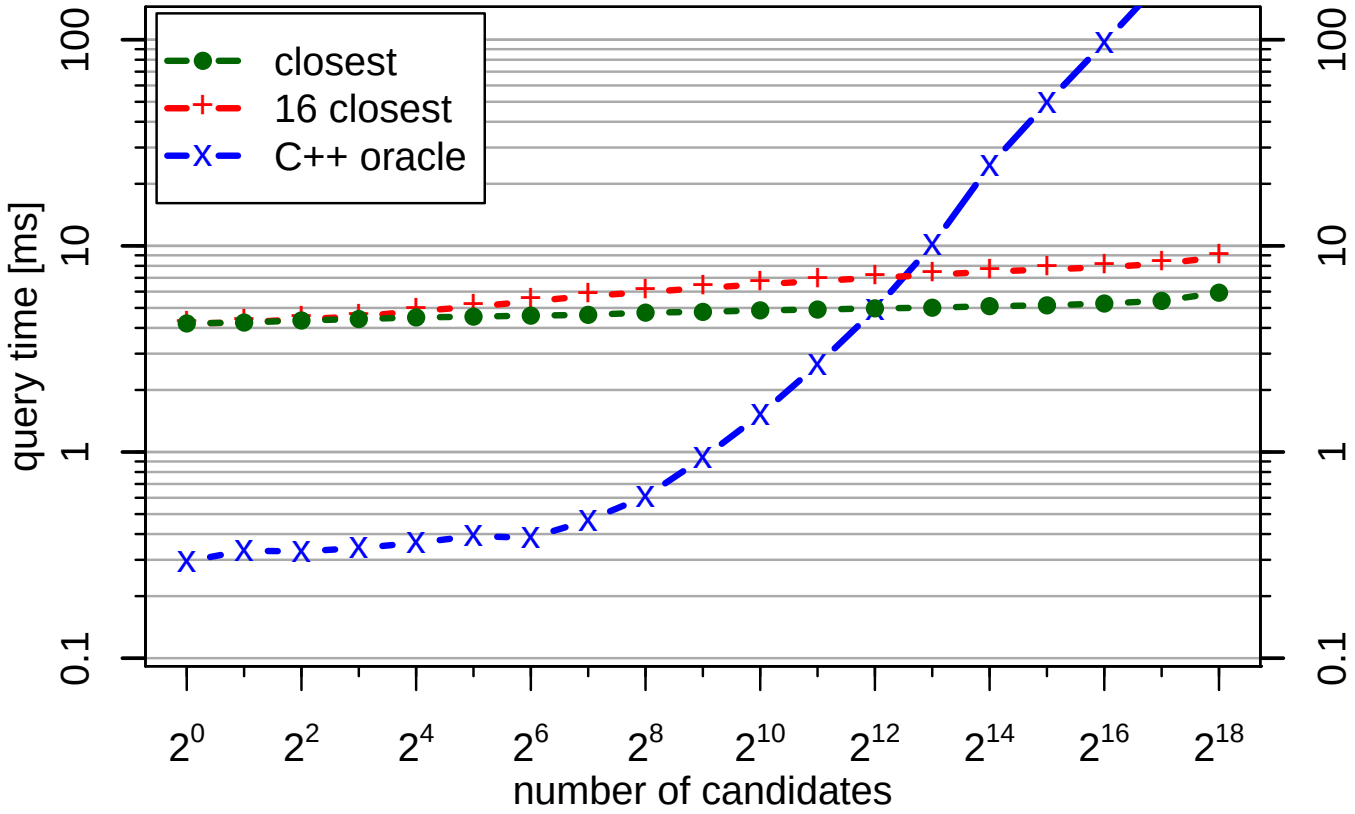
Setup: verschiedene Anzahl POIs, zufällig gewählt



- externes Punkt-zu-Punkt Orakel: skaliert schlecht
- SQL Anfragen unabhängig von Anzahl POIs

Ergebnisse POI

Setup: verschiedene Anzahl POIs, zufällig gewählt



- externes Punkt-zu-Punkt Orakel: skaliert schlecht
- SQL Anfragen unabhängig von Anzahl POIs
- weitere Constraints einfach (“jetzt geöffnet”)

Demo

Zusammenfassung

- one-to-many shortest paths
- many-to-many
- POI Anfragen
- bester Via Knoten Anfragen
- Location Services in SQL

Ende

Literatur:

- Sebastian Knopp, Peter Sanders, Dominik Schultes, Frank Schulz, and Dorothea Wagner:
  Computing Many-to-Many Shortest Paths Using Highway Hierarchies
  In: *Proceedings of the 9th Workshop on Algorithm Engineering and Experiments (ALENEX'07)*, pages 36-45, 2007.
- Daniel Delling, Andrew V. Goldberg, Renato F. Werneck
  Faster Batched Shortest Paths in Road Networks
  In: *Proceedings of the 11th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'11)*, pages 52-63, 2011
- Ittai Abraham, Daniel Delling, Andrew V. Goldberg, Renato F. Werneck
  HLDB: Location-Based Services in Databases
  bald verfügbar

Nächste Termine

Karlsruher Institut für Technologie

Montag, 10.6.2013

Montag, 17.6.2013

Institut für Theoretische Informatik
Lehrstuhl Algorithmik