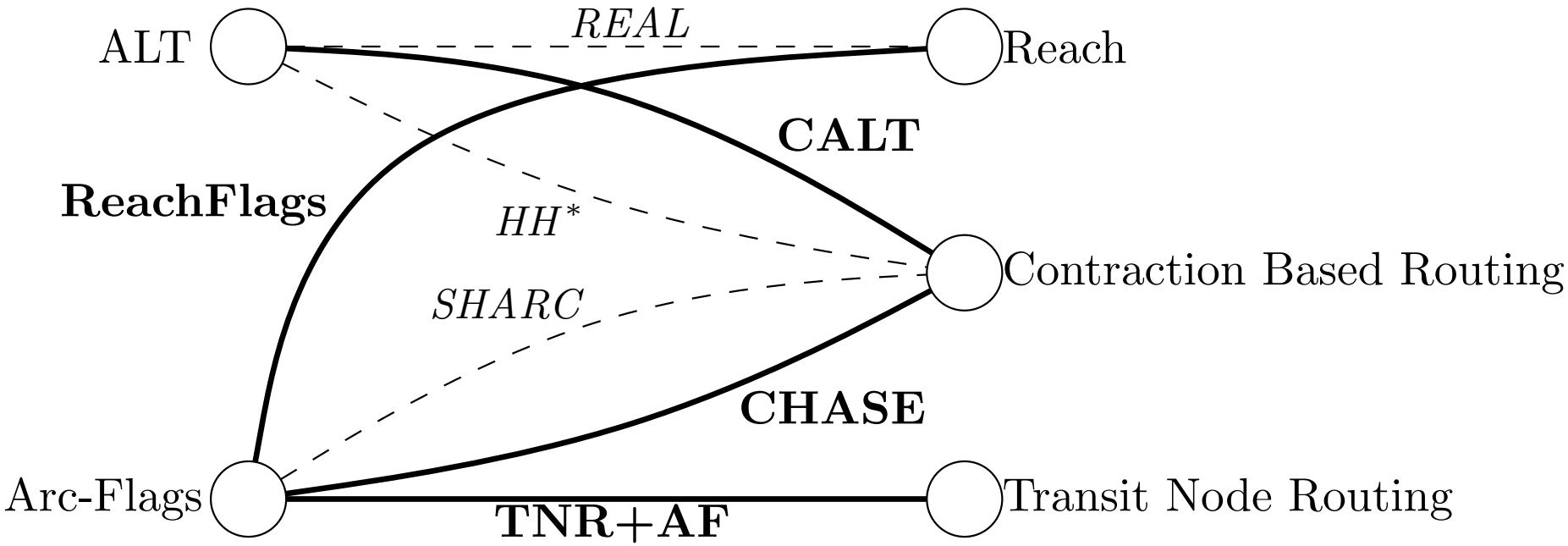
Algorithmen für Routenplanung – Vorlesung 8

Daniel Delling

Lehrstuhl für Algorithmik I
Institut für theoretische Informatik
Universität Karlsruhe (TH)
Forschungsuniversität · gegründet 1825

Letztes Mal: Kombinationen



Thema Heute: Dynamische Szenarien

Szenario:

- Unfall auf einer Straße
- Reisezeit ändert sich auf dieser Straße
- berechne schnellsten Weg bezüglich der aktualisierten Reisezeiten

Herausforderung

Hauptproblem:

- Kantengewichte ändern sich
- Vorberechnung basiert auf ursprünglichen Kantengewichten
- komplette Vorberechnung für jeden Stau wenig sinnvoll

Vorgehen I:

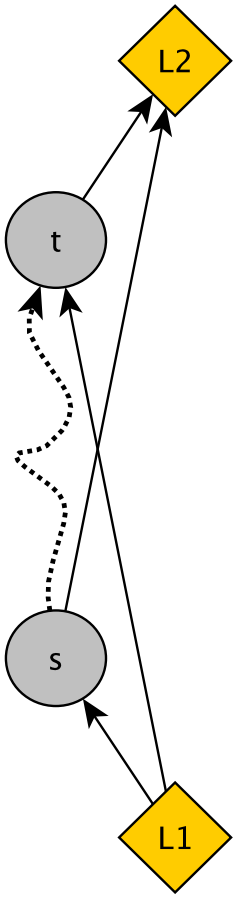
- identifiziere den Teil der beeinträchtigten Vorberechnung
- aktualisiere diesen Teil
- Serverszenario

Vorgehen II:

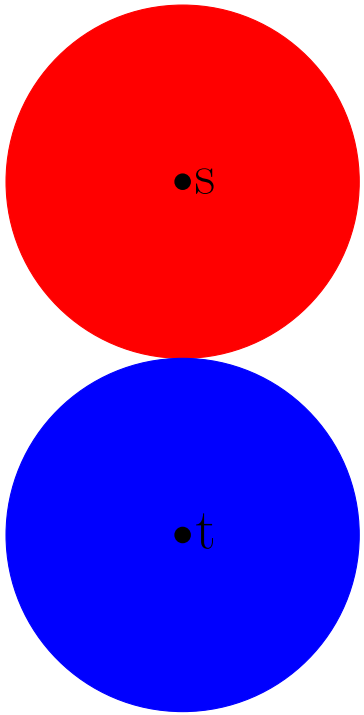
- robuste Vorberechnung
- immer noch korrekt, wenn Kantengewichte sich erhöhen
- dadurch eventuell Anfragen langsamer
- mobiles Szenario

Vier Bausteine

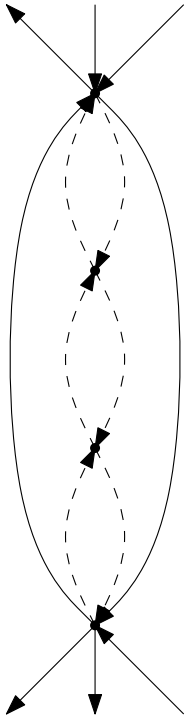
Landmarken



Bidirektionale Suche



Kontraktion



Arc-Flags

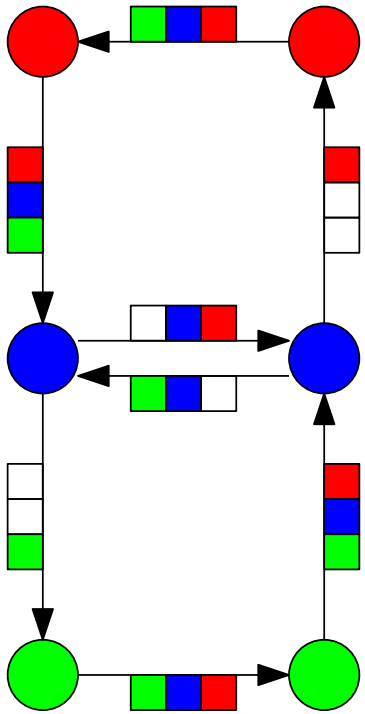
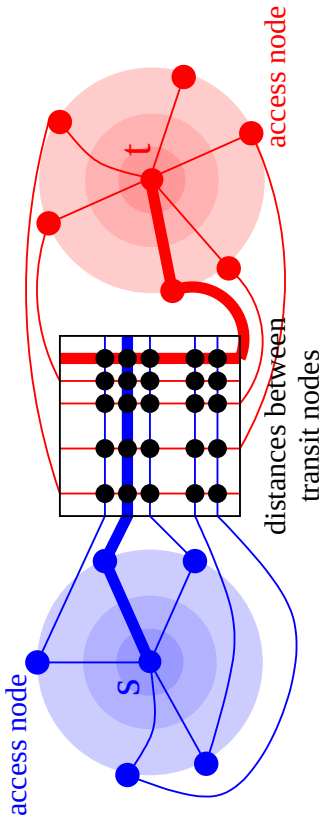
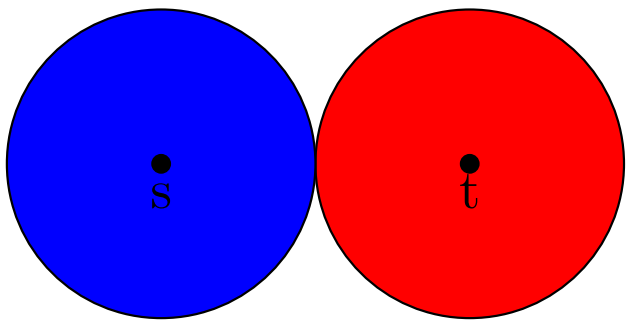


Table-Lookups



Bidirektionale Suche

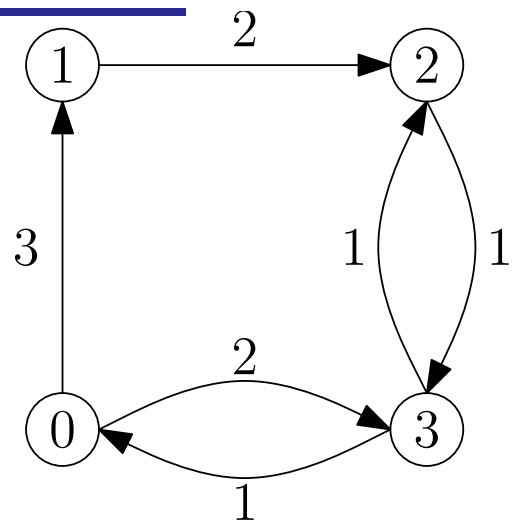


- starte zweite Suche von t
- relaxiere rückwärts nur eingehende Kanten
- stoppe die Suche, wenn beide Suchräume sich treffen

Anpassung

Beobachtung:

- Rückwärtskante muss aktualisiert werden
- sonst keine Vorberechnung
- somit (fast) keine Anpassung nötig



somit:

- bidirektionale Suche ohne Probleme anpassbar
- kein Verlust in Geschwindigkeit

`firstEdge`	0	3	5	7	10					
`targetNode`	1	3	3	0	2	1	3	0	0	2
`weight`	3	2	1	3	2	2	1	1	2	1
`isIncoming`	–	–	√	√	–	√	√	–	√	√
`isOutgoing`	√	√	–	–	√	–	√	√	–	√

Landmarken [Goldberg, Harrelson 05]

Vorberechnung:

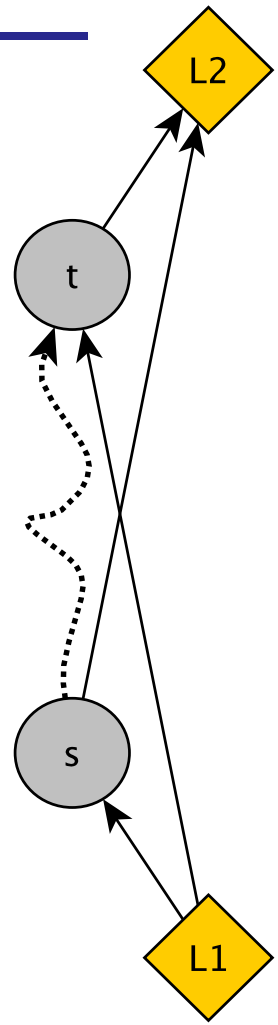
- wähle eine Hand voll (≈ 16) Knoten als Landmarken
- berechne Abstände von und zu allen Landmarken

Anfrage:

- benutze Landmarken und Dreiecksungleichung um eine untere Schranke für den Abstand zum Ziel zu bestimmen

$$\begin{aligned} d(s,t) &\geq d(L_1,t) - d(L_1,s) \\ d(s,t) &\geq d(s,L_2) - d(t,L_2) \end{aligned}$$

- verändert Reihenfolge der besuchten Knoten



Anpassung I

Beobachtung:

- Landmarkenwahl ist Heuristik, also nicht ändern
- Distanzen von und zu Landmarken sind nicht korrekt

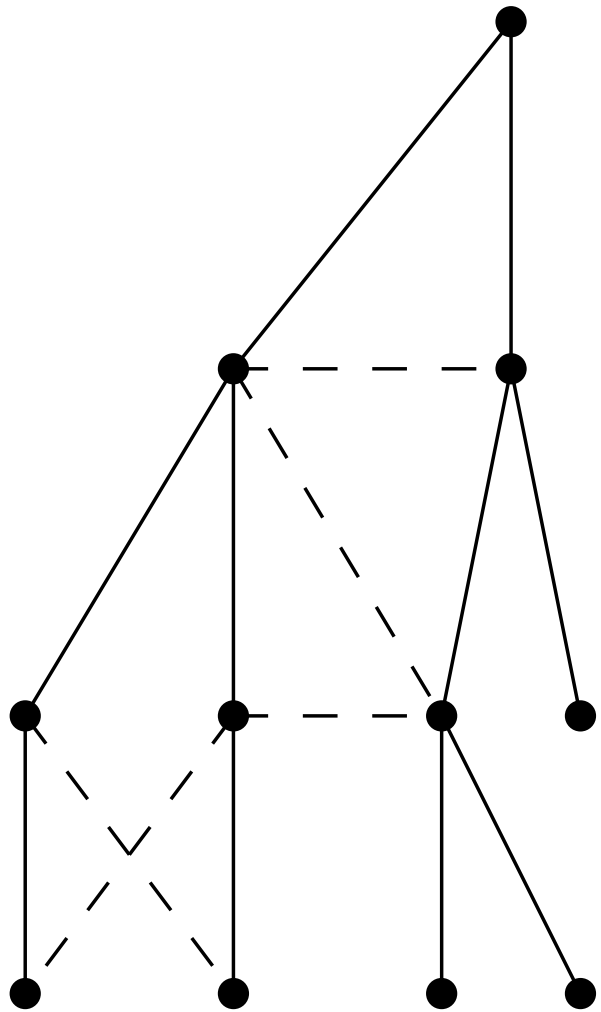
Vorgehen:

- aktualisiere diese Distanzen
- benutze dynamischen Dijkstra Algorithmus um Distanzen zu aktualisieren
- dann Anfragen korrekt und so schnell wie bei kompletter neuer Vorberechnung

Dynamischer Dijkstra

Erhöhung:

- wenn geänderte Kante (u, v) keine Baumkante, keine Änderung am Baum
- sonst:
  - markiere alle Kinder von (u, v) als ungültig und erhöhe Abstand
  - füge Nachbarn der Kinder in Queue ein (Priorität: Abstand)
  - starte Dijkstra
  - füge Knoten nur in Queue ein, wenn Abstand echt kleiner wird



Anmerkung:

- kann beliebig aufwendig werden (wenn update nahe des Wurzelknoten)

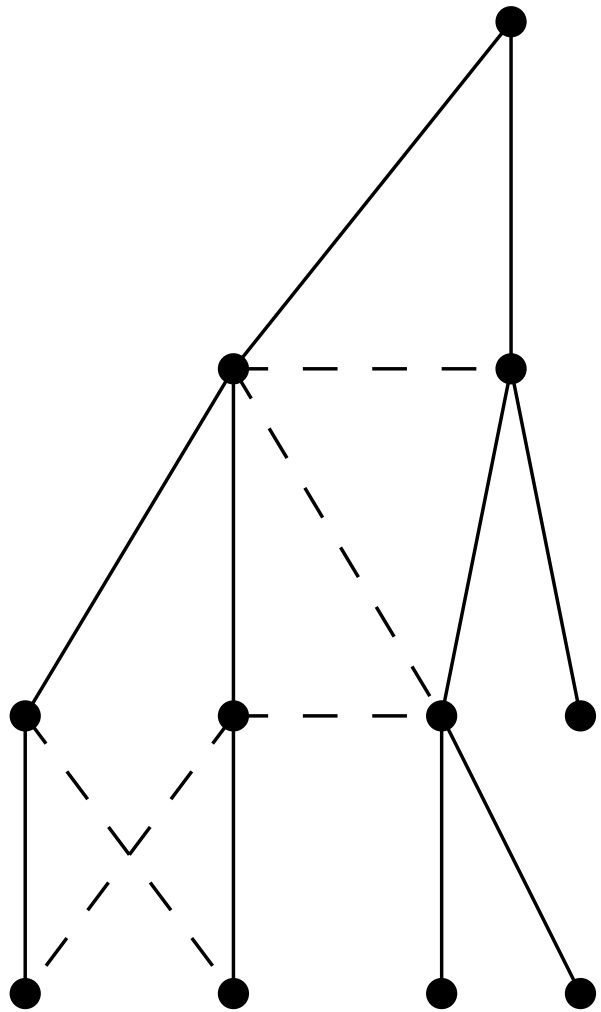
Dynamischer Dijkstra

Verringerung:

- sei geänderte Kante (u, v), wenn $d(u) + len_\delta(u, v) \geq d(v)$, fertig
- sonst
  - starte Dijkstra von v mit $d(u) + len_\delta(u, v)$ als Priorität
  - füge Knoten nur in Queue ein, wenn Abstand echt kleiner wird

Anmerkung:

- kann beliebig aufwendig werden (wenn update nahe des Wurzelknoten)



Anpassung II

Beobachtung:

- Korrektheit von ALT basiert darauf, dass reduzierten Kantengewichte größer gleich 0 sind

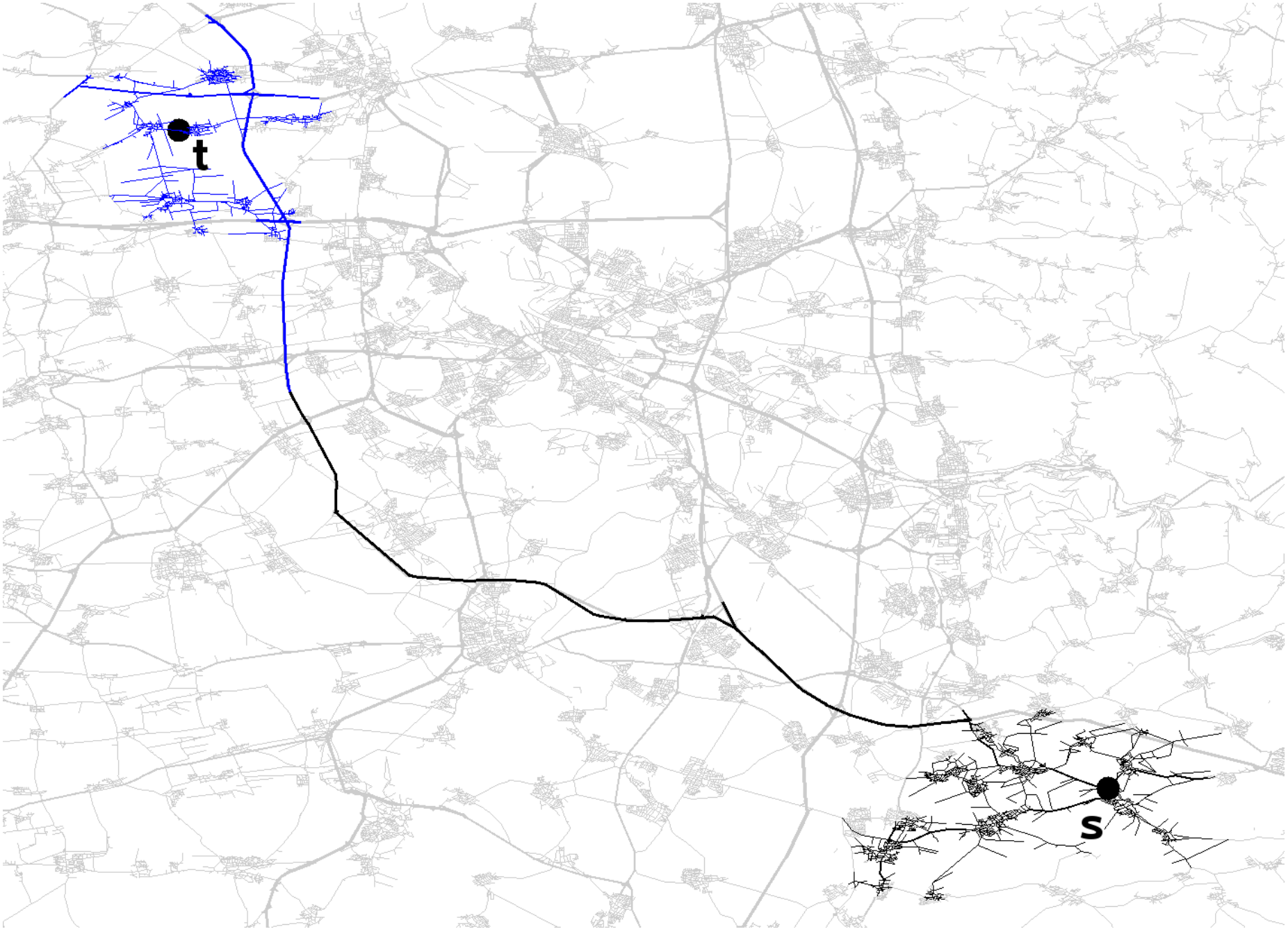
$$len_\pi(u,v) = len(u,v) - \pi(u) + \pi(v) \geq 0$$

- durch Erhöhen der Kantengewichte wird dies nicht verletzt
- durch Staus können Reisezeiten nicht unter Initialwert fallen

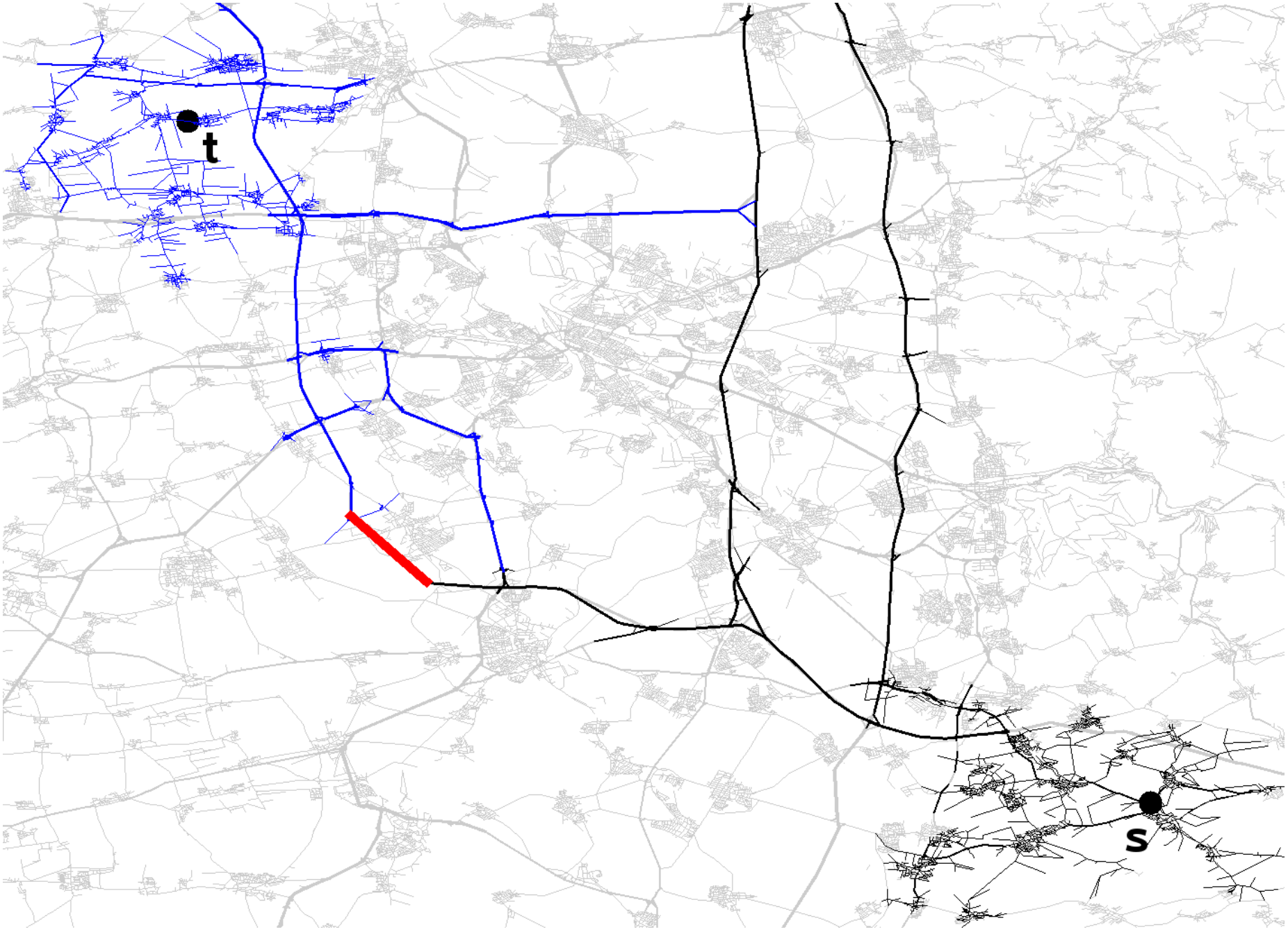
No-update Variante:

- Vorberechnung auf staufreiem Graphen
- Suchanfragen ohne Aktualisierung
- korrekt aber eventuell langsamere Anfragezeiten

Beispiel

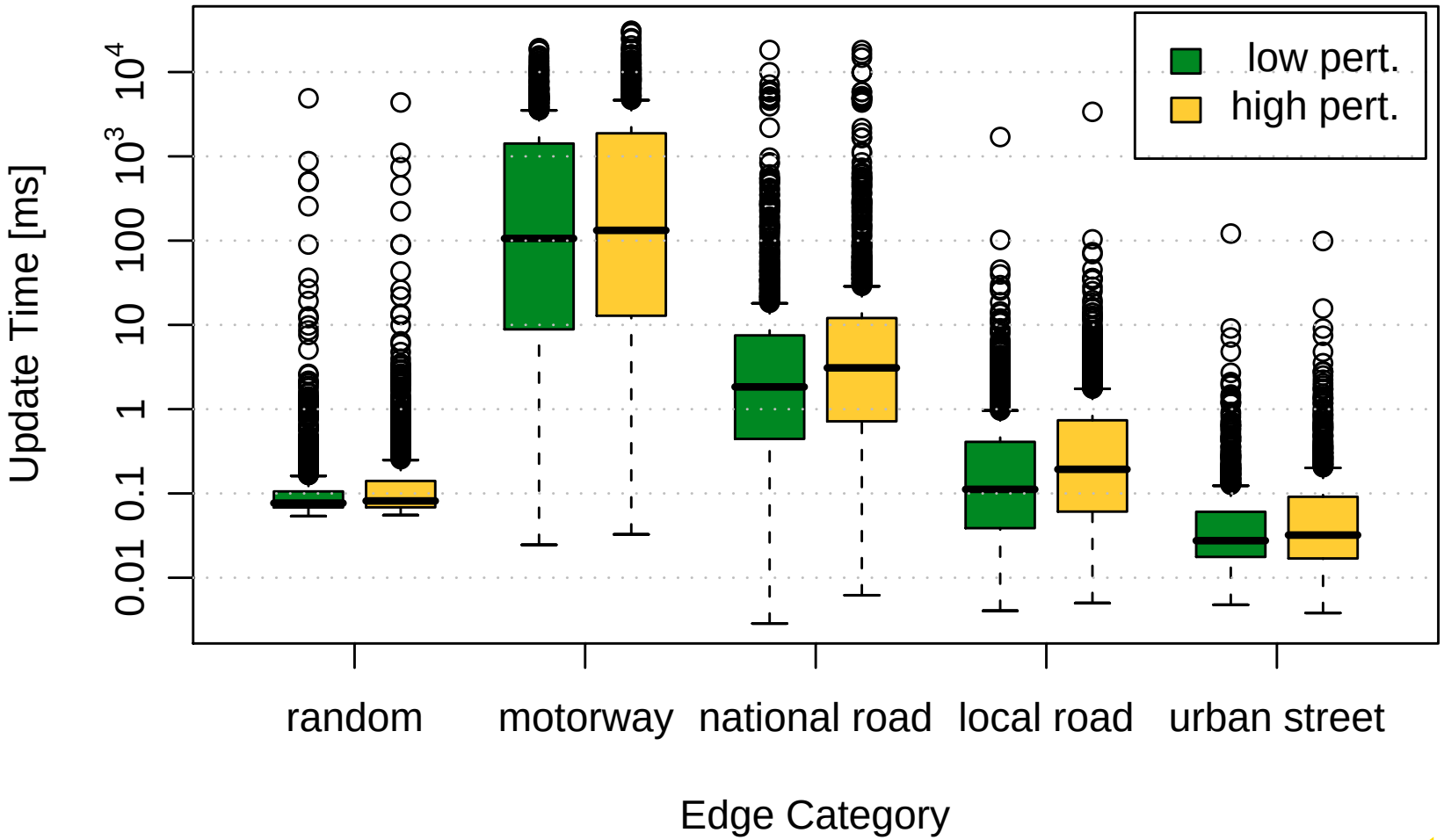


Beispiel



Aktualisierung der Vorberechnung

Aktualisieren von 32 kürzeste-Wege Bäumen für 16 Landmarken:



No-update Variante: Typ des Updates

- Zufallsanfragen nach 1 000 updates (x2, x10 in Klammern)
- verschiedene Straßenkategorien

road type	affected queries	16 landmarks search space		32 landmarks search space	
All	7.5 %	74 700	(77 759)	41 044	(43 919)
urban	0.8 %	74 796	(74 859)	40 996	(41 120)
local	1.5 %	74 659	(74 669)	40 949	(40 995)
national	28.1 %	74 920	(75 777)	41 251	(42 279)
motorway	95.3 %	97 249	(265 472)	59 550	(224 268)

Beobachtung:

- nur Autobahnen haben Einfluß auf Suchraumgröße

No-update Variante: Anzahl Updates

- Zufallsanfragen nach Autobahn updates (x2, x10 in Klammern)
- verschiedene Anzahl Updates

updates	affected queries	16 landmarks search space		32 landmarks search space	
100	39.9 %	75 691	(91 610)	41 725	(56 349)
200	64.7 %	78 533	(107 084)	44 220	(69 906)
500	87.1 %	86 284	(165 022)	50 007	(124 712)
1 000	95.3 %	97 249	(265 472)	59 550	(224 268)
2 000	97.8 %	154 112	(572 961)	115 111	(531 801)
5 000	99.1 %	320 624	(1 286 317)	279 758	(1 247 628)
10 000	99.5 %	595 740	(2 048 455)	553 590	(1 991 297)

Beobachtung:

- $\leq$ 1000 gut verkraftbar

Kontraktion [Sanders,Schultes 05]

Beobachtung:

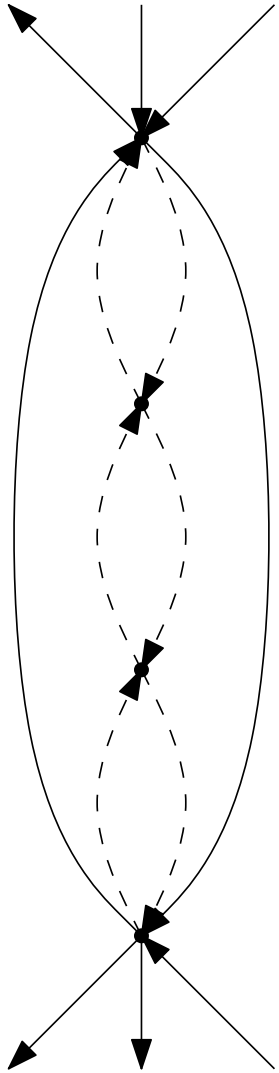
- Knoten mit niedrigem Grad sind unwichtig

Knoten-Reduktion:

- entferne diese Knoten iterativ
- füge neue Kanten (Abkürzungen) hinzu, um die Abstände zwischen verbleibenden Knoten zu erhalten

Kanten-Reduktion:

- behalte nur relevante Shortcuts
- lokale Suche während oder nach Knoten-reduktion



Anpassung

Beobachtung:

- entfernte Knoten sind “unwichtig”
- meist ändert sich diese Klassifikation nicht durch ein Update
- Knotenordnung bleibt gleich
- Länge der Shortcuts (u, v) entspricht der Länge des kürzesten Weges von u nach v
- dieser kann sich ändern(!)

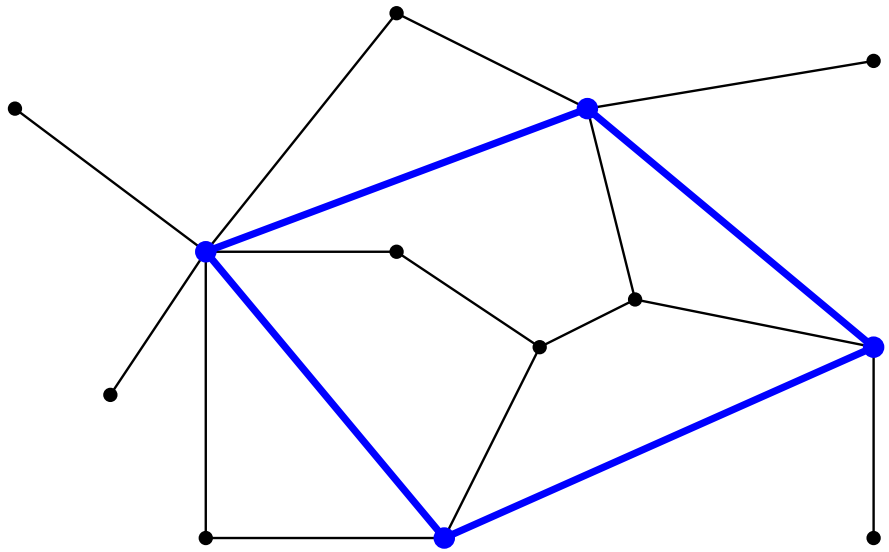
somit:

- shortcuts müssen aktualisiert werden
- wie?

Variante I

Idee:

- Kanten steuern zu Shortcuts bei
- aktualisiere nur beeinträchtigte Shortcuts
- berechne mit Dijkstra (im bereits aktualisierten Graphen) neue Länge des Shortcuts



Problem:

- klappt nur gut wenn shortcuts nicht zu lange

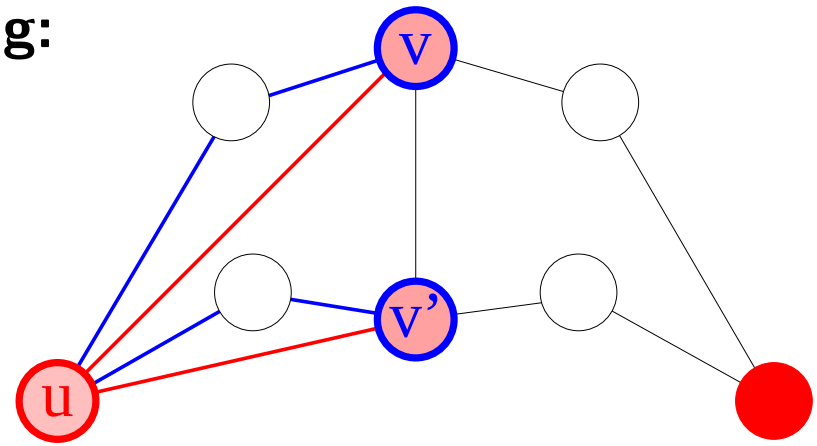
Variante II

Idee:

- wiederhole Kontraktion von allen Knoten, die durch update beeinflusst werden

Vorberechnung Highway-Node Routing:

- von jedem Knoten $u \in S$:
- lokale Suche in G
- bis alle Zweige abgedeckt
- füge Kante von u zu abdeckenden Knoten ein



somit:

- verwalte für jeden Knoten w ein set A_w
- füge u zu A_w jedes abgearbeiteten Knoten w
- wenn update auf (w, v): wiederhole Vorberechnung für alle $u \in A_w$

Variante III

Beobachtung (mobiles Szenario):

- nur Staus auf dem kürzesten Weg haben Einfluss
- Idee: aktualisiere nur diese Shortcuts (iterativ)

Vorgehen:

- invalidiere Shortcuts (aber aktualisiere sie nicht)
- berechne kürzesten Weg
- checke, ob gefundener Weg invalide shortcuts enthält
- aktualisiere nur diese shortcuts
- berechne wieder kürzesten Weg
- solange bis Weg keinen invaliden shortcut mehr enthält
- schnellere updates, langsamere Queries

Experimente: Variante II

setup

- Zeit (in ms) bei zufälligen Updates (normiert pro Update)
- $+$ 30 Minuten, $-$ 30 Minuten, auf ∞ setzen, Gewicht $\times$ 10

	any road type				motorway			
|change set|	$+$	$-$	∞	$\times$	$+$	$-$	∞	$\times$
1	2.7	2.5	2.8	2.6	40.0	40.0	40.1	37.3
1000	2.4	2.3	2.4	2.4	8.4	8.1	8.3	8.1

	national			regional			urban		
|change set|	$+$	$-$	∞	$+$	$-$	∞	$+$	$-$	∞
1	19.9	19.6	20.3	8.4	7.9	8.6	2.1	2.0	2.1
1000	7.1	6.7	7.1	5.3	5.0	5.3	2.0	1.9	2.0

Experimente: Variante III

|change set|	affected queries	iterative query time [ms]		iterative #iterations average		iterative #iterations max	
		+ 30 minutes (× 10)					
1	0.4 %	1.5	(1.5)	1.0	(1.0)	2	(2)
10	5.7 %	1.7	(1.7)	1.1	(1.1)	3	(3)
100	40.0 %	3.4	(3.3)	1.4	(1.4)	5	(5)
1 000	83.7 %	22.9	(17.6)	2.7	(2.4)	9	(8)
10 000	97.9 %	492.0	(323.3)	7.9	(6.3)	27	(22)

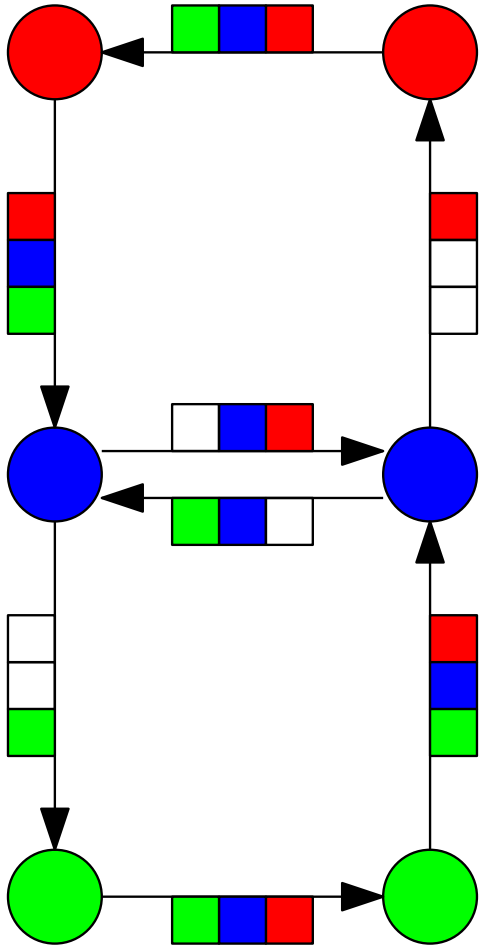
Arc-Flags [Lauther04],[Möhring et al. 05,06]

Idee:

- partitioniere den Graph in k Zellen
- hänge ein Label mit k Bits an jede Kante
- zeigt ob e wichtig für die Zielzelle ist
- modifizierter Dijkstra überspringt unwichtige Kanten

Beobachtung:

- Partition wird auf ungewichtetem Grahen durchgeführt
- Flaggen müssen allerdings aktualisiert werden



Anpassung: Dynamische Bäume

Idee:

- halte für jeden Randknoten den Baum im Speicher
- benutze dynamischen Dijkstra zum Aktualisieren

Problem:

- 15000 Randknoten
- pro Randknoten und pro Knoten 4 Byte
- 1 TB Speicherbedarf....

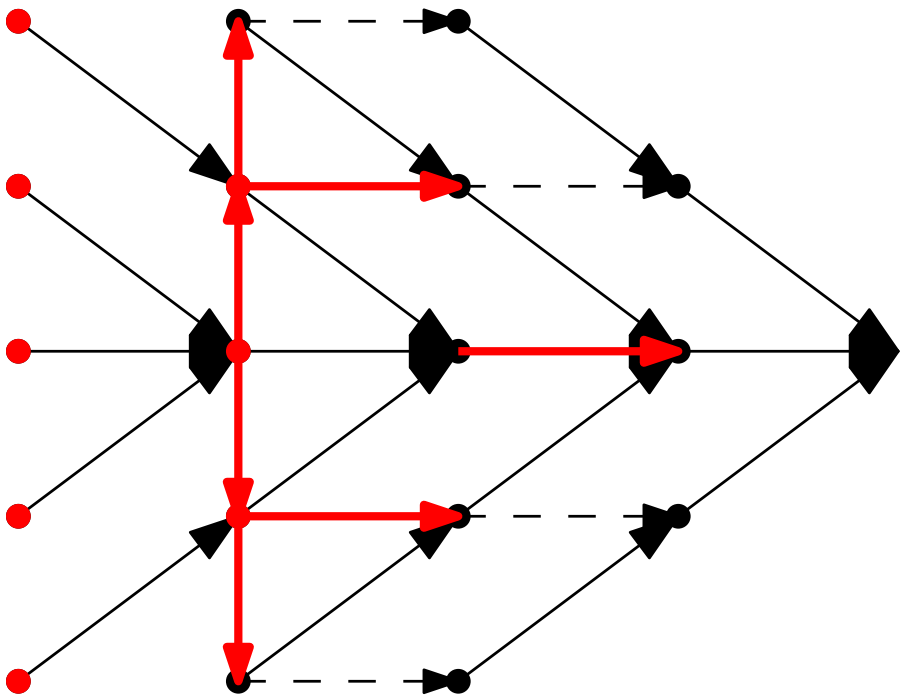
Anpassung: Konservativ

Vorgehen ((u, v) wird um δ erhöht):

- für jede gesetzte Regions-Flagge R von (u, v)
  - Tiefensuche auf induziertem Baum, startend von u
  - jede ausgehende Kante setzt R auf 1

Problem:

- setzt sehr viele Flaggen auf 1



Anpassung: Kompromiss

Beobachtung:

- für eine nicht-gesetzte Flagge einer Kante gilt $d(u,b) < len(u,v) + d(u,v)$ für alle Randknoten
- Grenzwert $\gamma_b(u,v) = len(u,v) + d(v,b) - d(u,b)$
- gibt an, wieviel kleiner die Kante sein müsste, damit sie eine 1 bekommen hätte

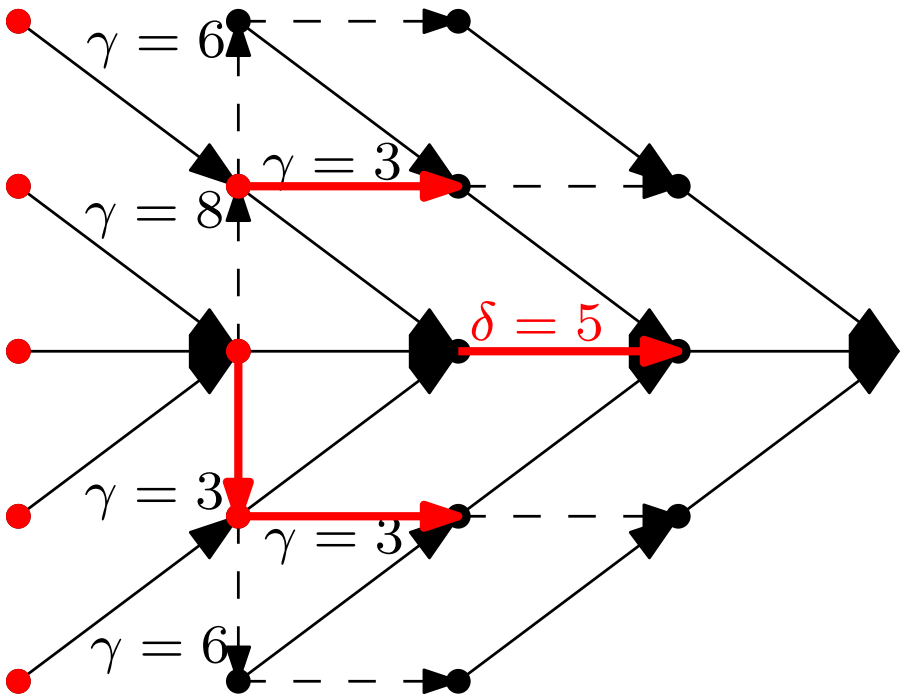
Idee:

- speicher für jede Kante und jede Region den Minimal-Wert γ_R der Region R
- nur die Kanten müssen aktualisiert werden, für die $\gamma_R < \delta$

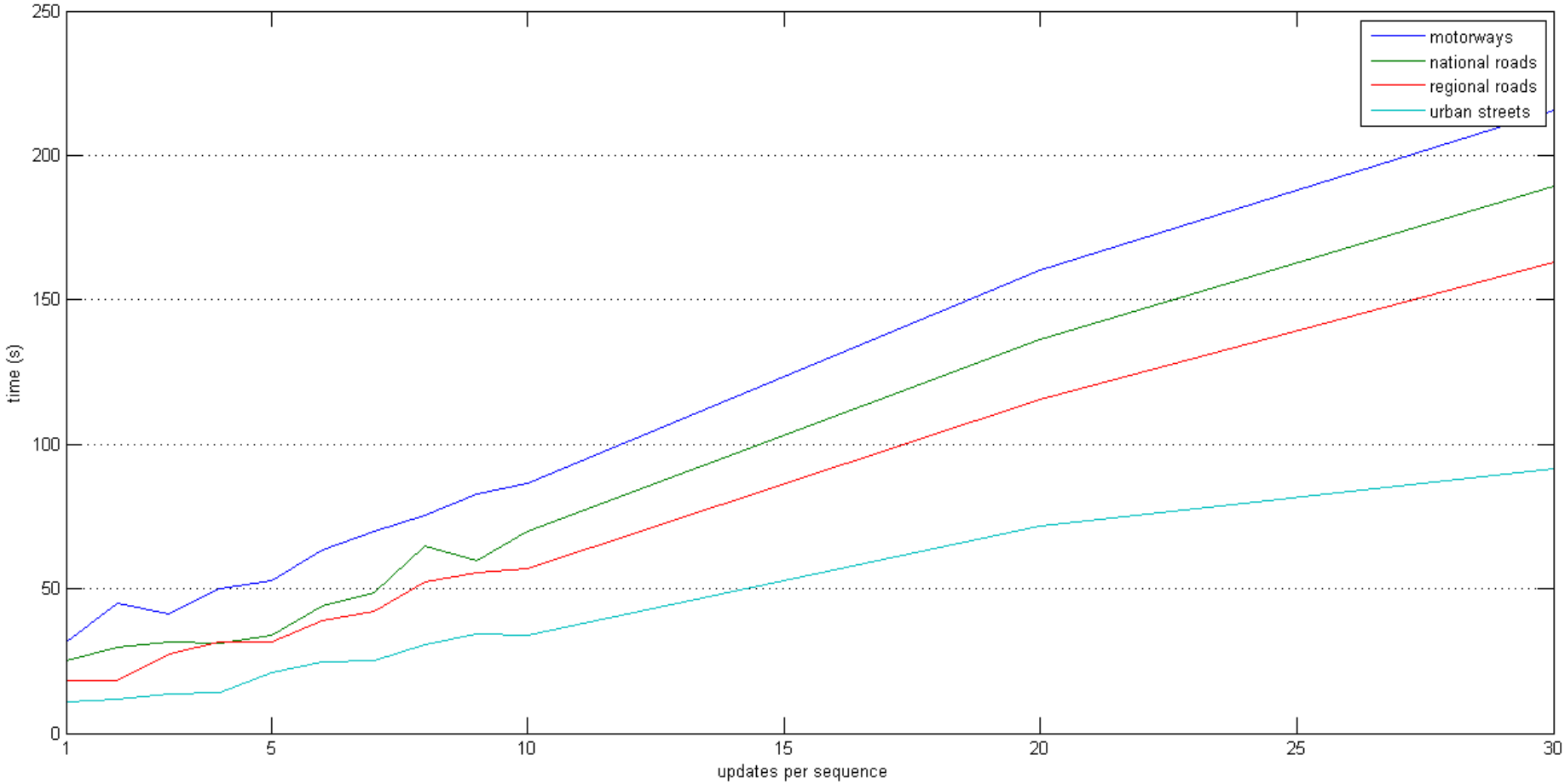
Anpassung: Kompromiss

Vorgehen:

- analog zu konservativ
- setze Flagge auf 1 nur dann, wenn δ des updates größer als threshold ist



Experimente: Update des Preprocessings



Experimente: Laufzeiten

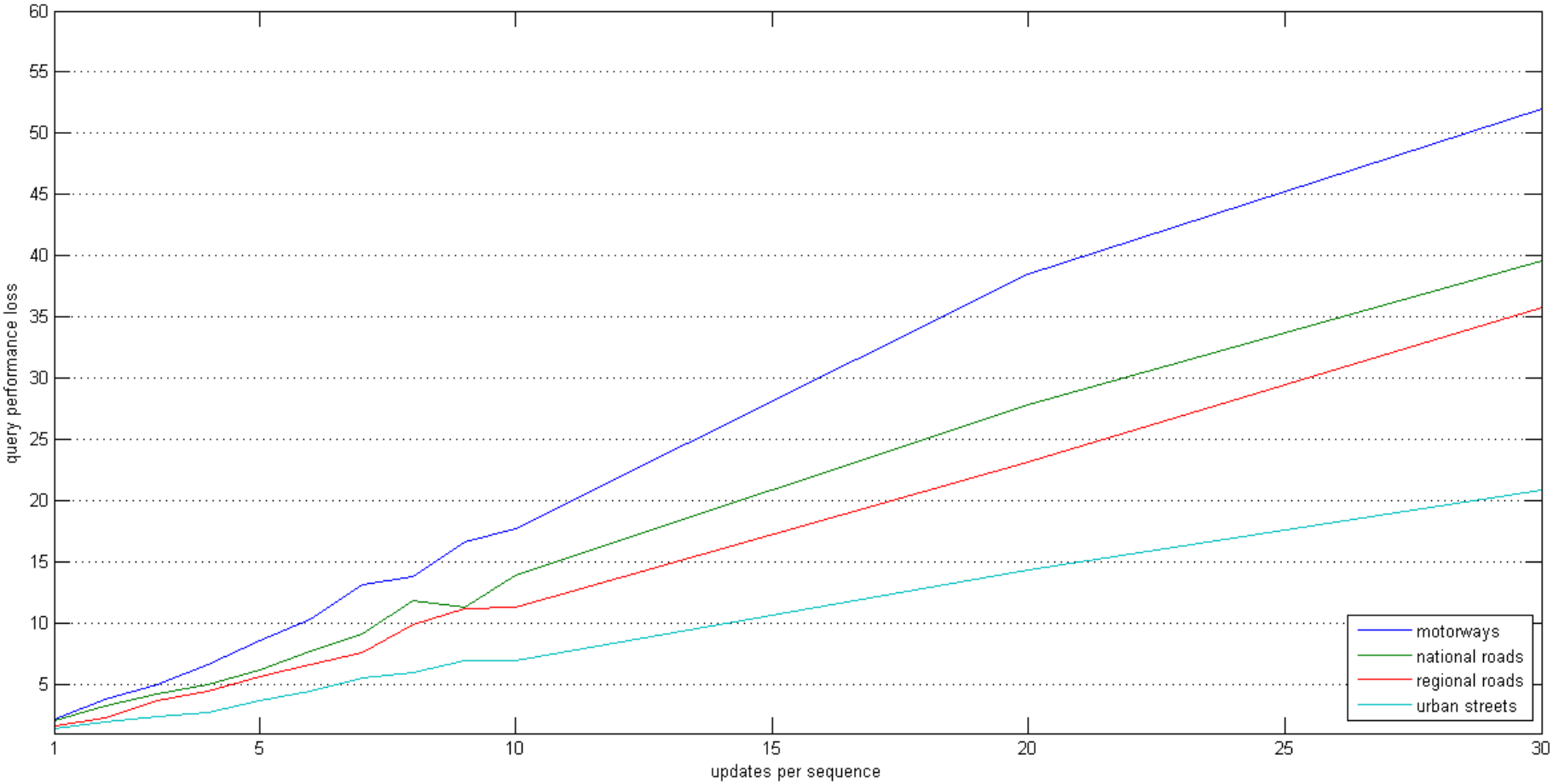
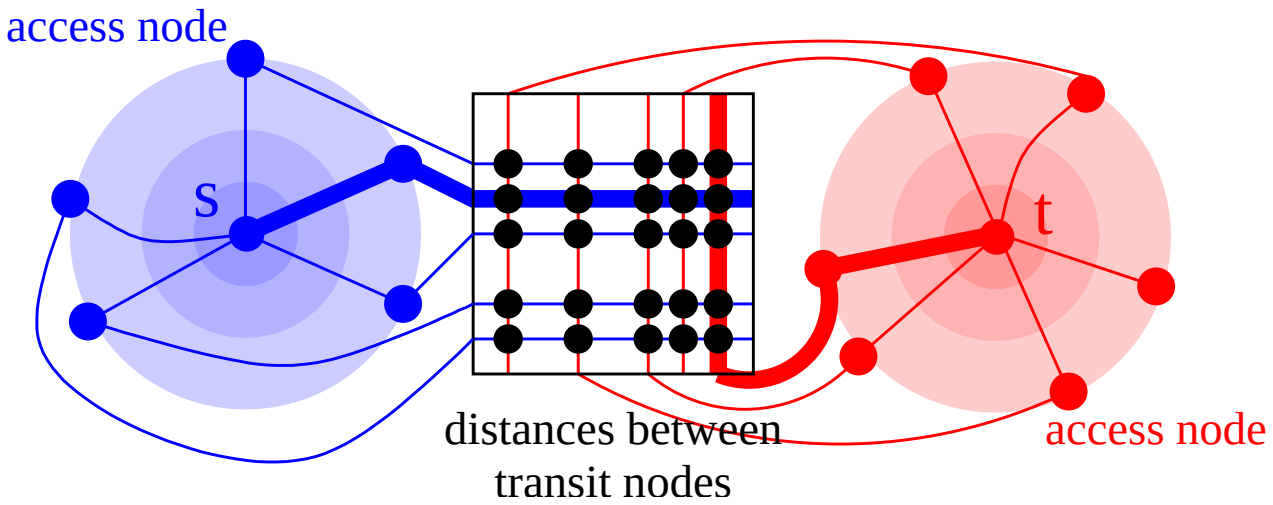


Table-Lookups

Idee:

- speichere Distanztabellen
- nur für "wichtige" Teile des Graphen
- Suchen laufen nur bis zur Tabelle
- harmoniert gut mir hierarchischen Techniken



Anpassung

Beobachtung:

- Distanz-Tabelle muss aktualisiert werden
- ein Eintrag entspricht effektiv einem Shortcut
- vollständiger Overlay-Graph
- viele Shortcuts

also:

- Techniken der Kontraktion anwendbar
- bloß so nicht effektiv
- lange Updatezeiten
- momentan nicht effizient implementierbar
- offenes Problem

Zusammenfassung Dynamische Szenarien

Dynamisierbare Basismodule:

- \+ bidirektionale Suche
- \+ landmarken
- \+ Kontraktion
- – arc-flags
- – Table Look-ups

somit folgende Algorithmen gut in dynamischen Szenarien verwendbar

- ≫ ALT
- ≫ Core-ALT
- ≫ Highway-Node Routing
- ≫ Contraction Hierarchies (Übung)

Literatur

Dynamisierung der Module:

- ALT: Delling/Wagner 07
- Kontraktion: Schultes 08, Delling et al. 09
- Arc-Flags: Berretini et al. 09
- Core-ALT: Delling/Nannicini 08

Anmerkung:

- wird auf der Homepage verlinkt