Karlsruher Institut für Technologie

Algorithmen für Routenplanung

7. Termin, Sommersemester 2011

Reinhard Bauer | 12. Mai 2011

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

Karlsruher Institut für Technologie

Reach-Based Pruning

Motivation 1: Netzwerkanalyse

- Zentralitätsmaße bewerten Wichtigkeit von Knoten oder Kanten in einem Netzwerkanalyse
- Häufige Beispiele
  - Google Page Rank
  - Erdös-Zahl

Zentralitätsmaße - Beispiele

- degree centrality

$$C_D(v) = \frac{\deg(v)}{n-1}$$

- betweenness centrality

$$C_B(v) = \sum_{\substack{s \neq v \neq t \in V \\ s \neq t}} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

wobei

- σ_{st} die Anzahl der kürzesten s-t-Wege ist
- $\sigma_{st}(v)$ die Anzahl der kürzesten s-t-Wege durch v ist

Motivation 1: Netzwerkanalyse

Reach:

- Zentralitätsmaß, das groß ist, falls eine Kante in der Mitte eines langen kürzesten Weges liegt.

Motivation 2

- Zeichne um jede Kante (u, v) Kreis mit Radius $r(u, v)$, so dass gilt:
  - Für jedes Paar s, t für das (u, v) auf einem kürzesten s-t-Weg liegt gilt, dass entweder s oder t in dem Kreis liegt.

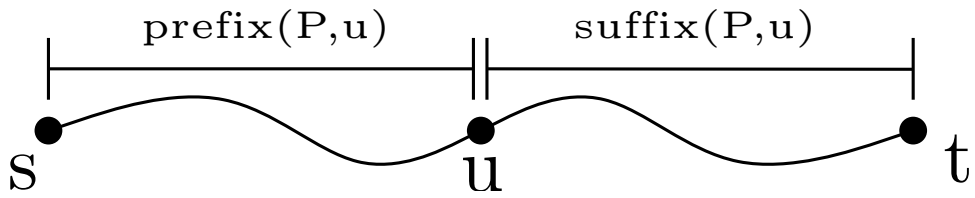
Möglichkeiten für diesen Kreis

- „geometrischer Kreis“ in euklidischer Ebene
- „graphentheoretischer Kreis“ im Eingabegraph

Strategie für Beschleunigungstechnik:

- Beachte Kante (u, v) nicht, wenn s und t nicht im Kreis um (u, v) liegen.
- Geometrischer Fall: leicht zu überprüfen
- Graphentheoretischer Fall: Bidirektionale Suche, benutze minimale Queueelemente (false positives möglich)

Das Zentralitätsmaß Reach



Definition:

- Sei $P = \langle s, \ldots, u, \ldots, t \rangle$ Pfad durch u
- dann Reach von u bezüglich P:

$$r_P(u) := \min\{\text{len}(\mathit{prefix}(P, u)), \text{len}(\mathit{suffix}(P, u))\}$$

- Reach von u:
  Maximum seiner Reachwerte bezüglich aller kürzesten Pfade durch u:

$$r(u) := \max\{r_P(u) \mid P \text{ kürzester Weg mit } u \in P\}$$

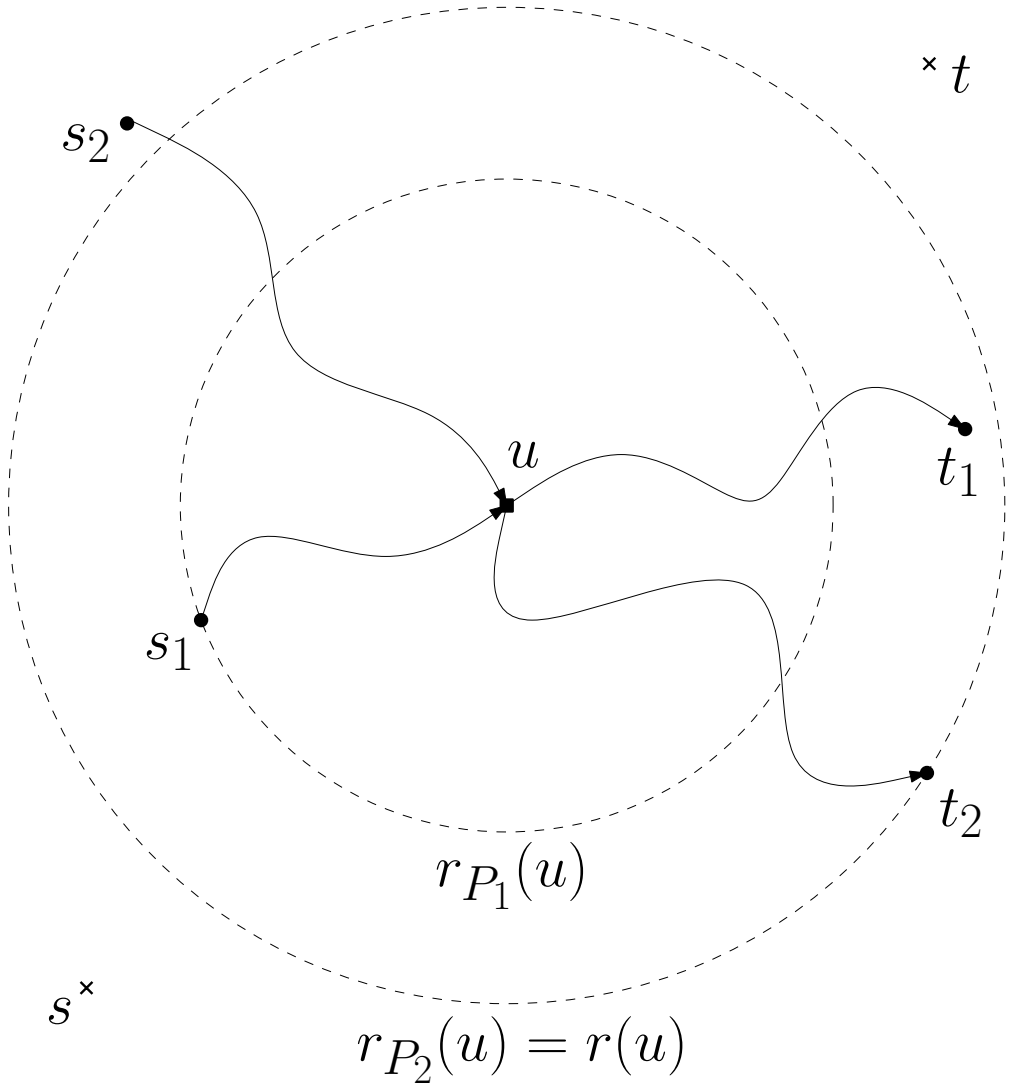
Bemerkung

Definition von Reach analog auch für Kanten möglich

Reach-Pruning

somit:

- Reach $r(u)$ von u gibt Suffix oder Prefix des längsten kürzesten Weges durch u
- wenn für u während Query $r(u) < d(s,u)$ und $r(u) < d(u,t)$ gilt, muss u nicht beachtet werden



Uni Reach Dijkstra - Pseudocode

ReachDijkstra($G = (V, E)$, s, t)

```
1  d[s] = 0
2  Q.clear(), Q.add(s, 0)
3  while !Q.empty() do
4      u ← Q.deleteMin()
5      if r(u) < d[u] and r(u) < d(u, t) then continue
6      forall the edges e = (u, v) ∈ E do
7          if d[u] + len(e) < d[v] then
8              d[v] ← d[u] + len(e)
9              if v ∈ Q then Q.decreaseKey(v, d[v])
10             else Q.insert(v, d[v])
```

Problem

Problem:

- Abfrage $r(u) < d(u,t)$

- Im geometrischen Fall ist die Überprüfung einfach
- Die geometrische Strategie wurde unabhängig voneinander in
  - Ertl: Shortest path calculation in large road networks, OR Spektrum 1998
  - R. Gutman. Reach-based Routing: A New Approach to Shortest Path Algorithms Optimized for Road Networks. In Proceedings of ALENEX'04

  entwickelt
- Graphentheoretische Strategie in [Goldberg et al, 2009]

Problem

Problem:

- Abfrage $r(u) < d(u,t)$

- Im geometrischen Fall ist die Überprüfung einfach
- Die geometrische Strategie wurde unabhängig voneinander in
  - Ertl: Shortest path calculation in large road networks, OR Spektrum 1998
  - R. Gutman. Reach-based Routing: A New Approach to Shortest Path Algorithms Optimized for Road Networks. In Proceedings of ALENEX'04

  entwickelt
- Graphentheoretische Strategie in [Goldberg et al, 2009]

Bidir. Self-Bounding Reach-Dijkstra

Idee (für Vorwärtssuche):

- ignoriere knoten u wenn $d[u] > r(u)$ gilt
- überlasse den Check $d(u,t) > r(u)$ der Rückwärtssuche
- Rückwärtssuche analog (umgekehrt)
- ändere das Stoppkriterium

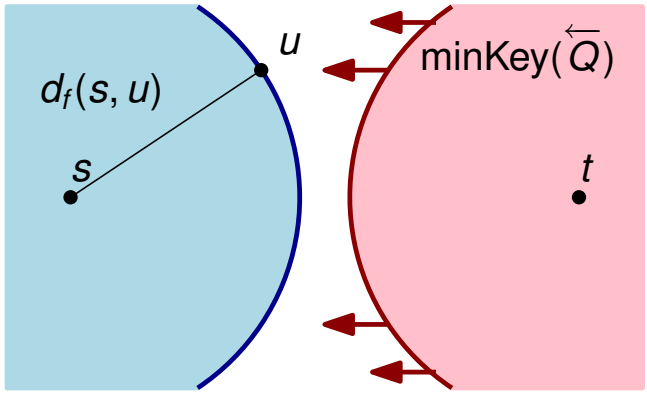
neues Stoppkriterium:

- stoppe Suche in eine Richtung wenn Queue leer oder es gilt: $\text{minKey}(Q) > \mu/2$
- stoppe Anfrage, wenn beide Suchrichtungen gestoppt haben

Bidir. Dist.-Bounding Reach-Dijkstra

Idee (für Vorwärtssuche):

- wenn u von Rückwärtssuche noch nicht erreicht, ist minKey($\overleftarrow{Q}$) eine untere Schranke für $d(u,t)$
- wenn u von Rückwärtssuche abgearbeitet, $d(u,t)$ bekannt



somit:

- ignoriere u, wenn $r(u) < d_f[u]$ und $r(u) < \min\{d_b[u], \text{minKey}(\overleftarrow{Q})\}$
- Stoppkriterium bleibt erhalten (also $\text{minKey}(\overrightarrow{Q}) + \text{minKey}(\overleftarrow{Q}) \geq \mu$)
- wenn als Alternierungsstrategie $\min\{\text{minKey}(\overrightarrow{Q}), \text{minKey}(\overleftarrow{Q})\}$ gewählt, gilt für Vorwärtssuche: $d_f[u] \leq \text{minKey}(\overleftarrow{Q})$

Optimierungen

mögliche Verbesserungen:

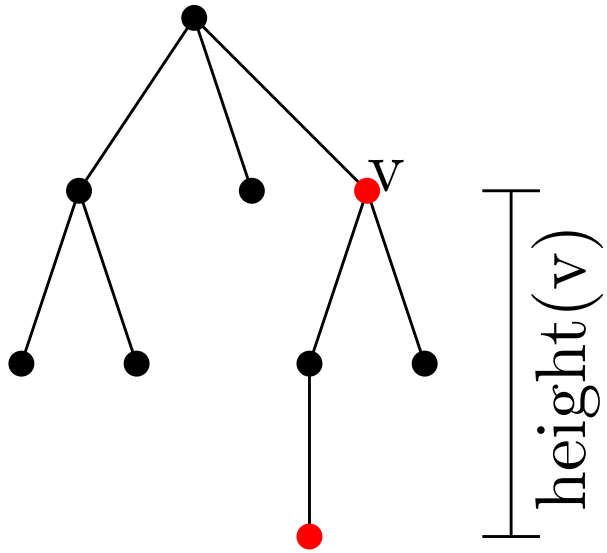
- Early (Kanten-)Pruning:
  (u, v) muss nicht relaxiert werden, wenn gilt:
  - $d[u] + \text{len}(u, v) > r(v)$
  - und $r(v) < \min\{d_b[u], \text{minKey}(\overleftarrow{Q})\}$

- Kanten sortieren:
  - sortiere ausgehende Kanten (u, v_i) absteigende nach $r(v_i)$
  - wenn Kante relaxiert wird mit $r(v_i) < \text{minKey}(\overleftarrow{Q})$ und $r(v_i) < d_f[u]$ müssen die restlichen Kanten ausgehend von u nicht relaxiert werden

Vorberechnung: Erster Ansatz

Wie kann man Reach-Werte vorberechnen?

- initialisieren $r(u) = 0$ für alle Knoten
- für jeden Knoten u
  - konstruiere kürzeste Wege-Baum
  - höhe von Knoten v: Abstand von v zum am weitesten entfernten Nachfolger
  - für jeden Knoten v:
    $r(v) = \max\{r(v), \min\{d(u,v), \text{height}(v)\}\}$



altes Problem:

- Vorberechnung basiert auf all-pair-shortest paths
- somit wieder 500 Jahre Vorberechnung

Approximation

Beobachtung:

- es genügt, für jeden Knoten eine obere Schranke des Reach-Wertes zu haben

Problem:

- untere Schranken einfach zu finden:
  - breche Konstruktion der Bäume einfach bei bestimmter Größe ab
- aber: untere Schranken sind unbrauchbar
- Berechnung von oberen Schranken deutlich schwieriger

Der RE-Algorithmus

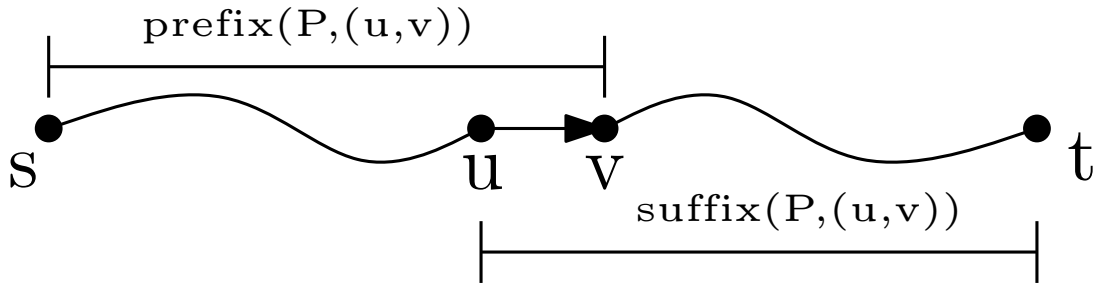
Erweiterungen gegenüber reinem Reach:

- Kanten-Reach für Vorberechnung
- obere Schranken durch iterative Berechnung
- Shortcuts

Ergebnisse:

- schnellere Vorberechnung
- deutlich schnellere Anfragen

Kanten-Reach



Definition:

- sei $P = \langle s, \ldots, u, v, \ldots, t \rangle$ Pfad durch (u, v)
- Reach von (u, v) bezüglich P:

$$r_P((u, v)) := \min\{\text{len}(P_{sv}), \text{len}(P_{ut})\}$$

- Reach von (u, v):
  Maximum seiner Reachwerte bezüglich aller kürzester Pfade durch (u, v):

$$r((u, v)) := \max\{r_P((u, v)) \mid P \text{ kürzester Weg mit } (u, v) \in P\}$$

Grobe Vorgehensweise

- Wähle ein ϵ und ein k
- Berechne Reach-Werte von allen Kanten mit Reach kleiner als ϵ
- Lösche diese Kanten
- Berechne obere Schranken für Reach-Werte von allen übrigen Kanten mit Reach kleiner als $k \cdot \epsilon$
- Lösche diese Kanten
- ....

Berechnung der Reach-Werte von allen Kanten mit Reach kleiner epsilon

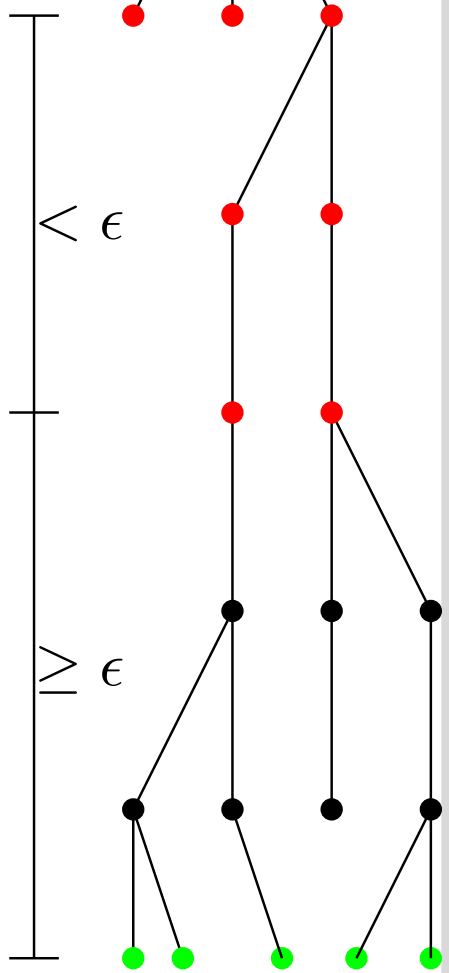
- Problem: um $r(u,v) < \epsilon$ zu garantieren, müssen alle kürzesten Wege durch (u,v) berücksichtigt werden.

Beobachtung:

- Sei $r(u,v) > \epsilon$
- Dann gibt es kürzesten Weg $P_{st} = \langle s, s', \ldots, u, v, \ldots, t', t \rangle$ Pfad durch (u,v) mit $\mathrm{dist}(s,v) > \epsilon$ und $\mathrm{dist}(u,t) > \epsilon$
- Gilt zusätzlich $\mathrm{dist}(s',v) \leq \epsilon$ und $\mathrm{dist}(u,t') \leq \epsilon$, so heißt der Weg ϵ-minimal

Approx. Berechnung von Kanten-Reach II

- partieller KW-Baum T_w von jedem $w \in V$
- stoppe, wenn garantiert, dass alle ϵ-minimalen Pfade, die bei w starten gefunden worden sind
- u ist innerer Knoten: $u = w$ oder $d(x, u) < \epsilon$ mit x erster Knoten nach w zwischen u und w
- stoppe, wenn Queue leer oder wenn Abstand von allen Knoten in der Queue zu nähestem inneren Knoten bzgl. T_w größer gleich ϵ ist
- für jeden inneren Knoten u
  - berechne Höhe bezüglich T_w
  - reach für (u, v) bezüglich T_w ist $\min\{d(w, u), height_{T_w}(u)\}$
- $r(u, v) = max_{w \in V}\{r_{T_w}(u, v)\}$
- wenn $r(u, v) \geq \epsilon$ setze $r(u, v) = \infty$



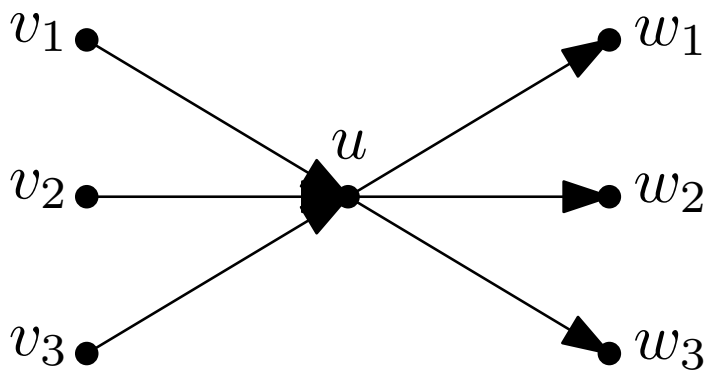
Iterative Berechnung von Knoten-Reach

Berechnen von Reach:

- wähle ϵ frei
- iterativ, solange $E \neq \emptyset$
  - berechne obere Schranken mit vorherigem Verfahren
  - entferne alle Kanten mit reach $< \epsilon$ aus Graphen
  - setze $\epsilon = k \cdot \epsilon$
- wandle Kanten-Reach in Knoten-Reach um

Umrechnung:

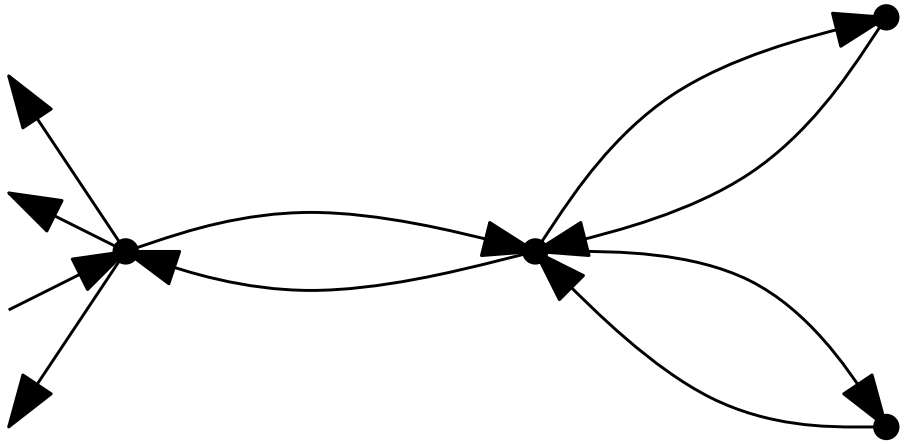
$$r(u) \leq \min \left\{ \max_i \{r(v_i, u)\}, \max_i \{r(u, w_i)\} \right\}$$



Penalties

Problem:

- durch entfernen von Knoten und Kanten ändern sich die Längen der kürzesten Wege



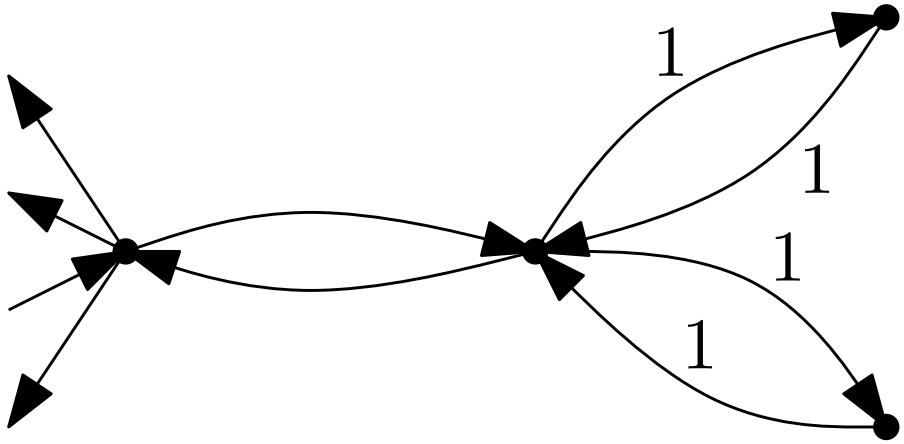
Lösung:

- halte für jeden Knoten zwei Werte vor:
  - $inPenalty(u)$: maximum der reachs entfernter (w, u) Kanten
  - $outPenalty(u)$: maximum der reachs entfernter (u, w) Kanten
- während Vorberechnung:
  - Hänge an jeden Knoten v Pseudoknoten v' mit Abstand $outPenalty(v)$
  - bestimmte neue Höhe $height_p(u)$
  - dann $r_{T_w}(u, v) = \min\{d(u, w) + inPenalty(u), height_p(u)\}$

Penalties

Problem:

- durch entfernen von Knoten und Kanten ändern sich die Längen der kürzesten Wege



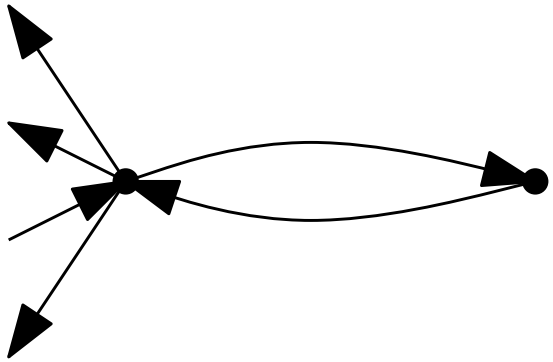
Lösung:

- halte für jeden Knoten zwei Werte vor:
  - $inPenalty(u)$: maximum der reachs entfernter (w, u) Kanten
  - $outPenalty(u)$: maximum der reachs entfernter (u, w) Kanten
- während Vorberechnung:
  - Hänge an jeden Knoten v Pseudoknoten v' mit Abstand $outPenalty(v)$
  - bestimmte neue Höhe $height_p(u)$
  - dann $r_{T_w}(u, v) = \min\{d(u, w) + inPenalty(u), height_p(u)\}$

Penalties

Problem:

- durch entfernen von Knoten und Kanten ändern sich die Längen der kürzesten Wege



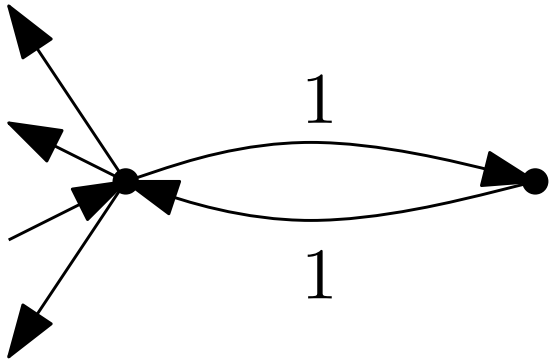
Lösung:

- halte für jeden Knoten zwei Werte vor:
  - $inPenalty(u)$: maximum der reachs entfernter (w, u) Kanten
  - $outPenalty(u)$: maximum der reachs entfernter (u, w) Kanten
- während Vorberechnung:
  - Hänge an jeden Knoten v Pseudoknoten v' mit Abstand $outPenalty(v)$
  - bestimmte neue Höhe $height_p(u)$
  - dann $r_{T_w}(u, v) = \min\{d(u, w) + inPenalty(u), height_p(u)\}$

Penalties

Problem:

- durch entfernen von Knoten und Kanten ändern sich die Längen der kürzesten Wege



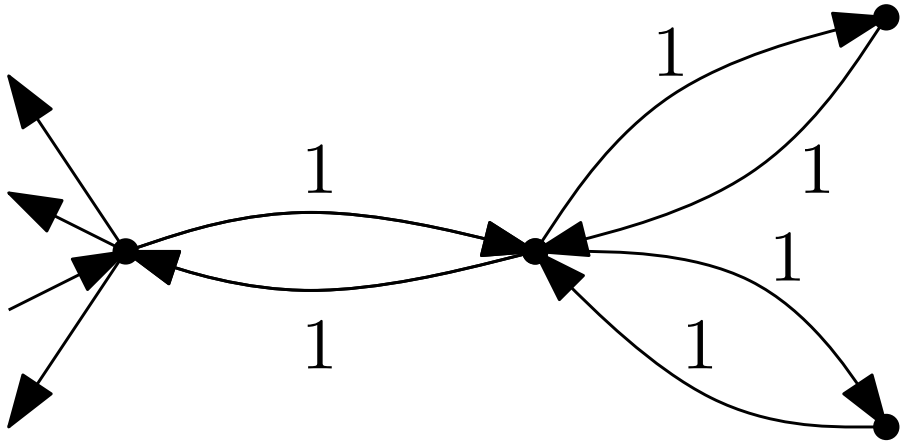
Lösung:

- halte für jeden Knoten zwei Werte vor:
  - $inPenalty(u)$: maximum der reachs entfernter (w, u) Kanten
  - $outPenalty(u)$: maximum der reachs entfernter (u, w) Kanten
- während Vorberechnung:
  - Hänge an jeden Knoten v Pseudoknoten v' mit Abstand $outPenalty(v)$
  - bestimmte neue Höhe $height_p(u)$
  - dann $r_{T_w}(u, v) = \min\{d(u, w) + inPenalty(u), height_p(u)\}$

Penalties

Problem:

- durch entfernen von Knoten und Kanten ändern sich die Längen der kürzesten Wege



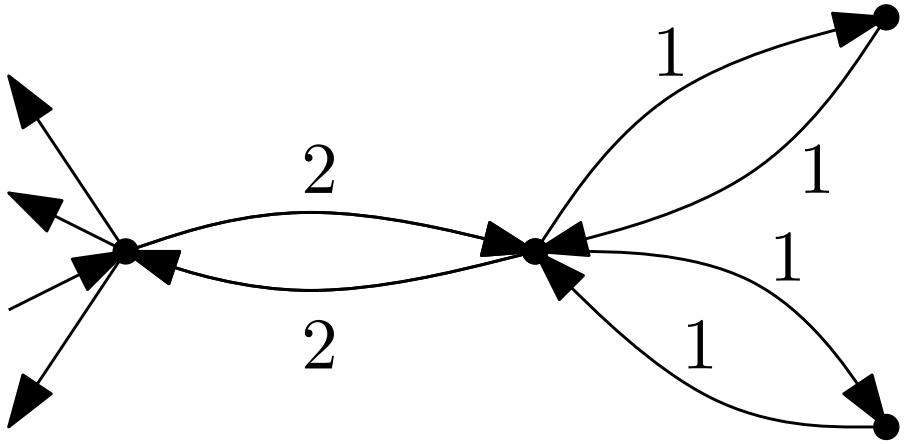
Lösung:

- halte für jeden Knoten zwei Werte vor:
  - $inPenalty(u)$: maximum der reachs entfernter (w, u) Kanten
  - $outPenalty(u)$: maximum der reachs entfernter (u, w) Kanten
- während Vorberechnung:
  - Hänge an jeden Knoten v Pseudoknoten v' mit Abstand $outPenalty(v)$
  - bestimmte neue Höhe $height_p(u)$
  - dann $r_{T_w}(u, v) = \min\{d(u, w) + inPenalty(u), height_p(u)\}$

Penalties

Problem:

- durch entfernen von Knoten und Kanten ändern sich die Längen der kürzesten Wege



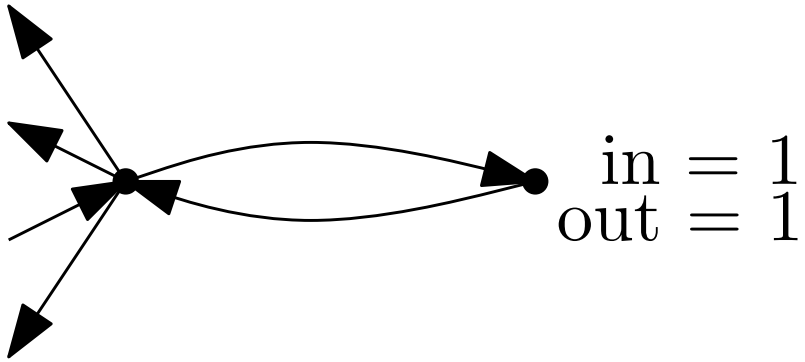
Lösung:

- halte für jeden Knoten zwei Werte vor:
  - $inPenalty(u)$: maximum der reachs entfernter (w, u) Kanten
  - $outPenalty(u)$: maximum der reachs entfernter (u, w) Kanten
- während Vorberechnung:
  - Hänge an jeden Knoten v Pseudoknoten v' mit Abstand $outPenalty(v)$
  - bestimmte neue Höhe $height_p(u)$
  - dann $r_{T_w}(u, v) = \min\{d(u, w) + inPenalty(u), height_p(u)\}$

Penalties

Problem:

- durch entfernen von Knoten und Kanten ändern sich die Längen der kürzesten Wege



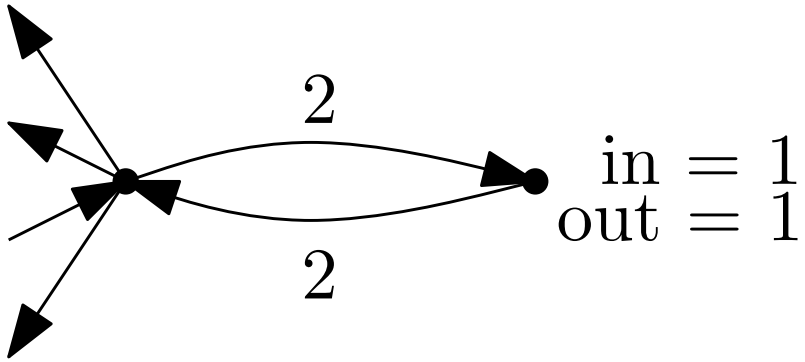
Lösung:

- halte für jeden Knoten zwei Werte vor:
  - $inPenalty(u)$: maximum der reachs entfernter (w, u) Kanten
  - $outPenalty(u)$: maximum der reachs entfernter (u, w) Kanten
- während Vorberechnung:
  - Hänge an jeden Knoten v Pseudoknoten v' mit Abstand $outPenalty(v)$
  - bestimmte neue Höhe $height_p(u)$
  - dann $r_{T_w}(u, v) = \min\{d(u, w) + inPenalty(u), height_p(u)\}$

Penalties

Problem:

- durch entfernen von Knoten und Kanten ändern sich die Längen der kürzesten Wege



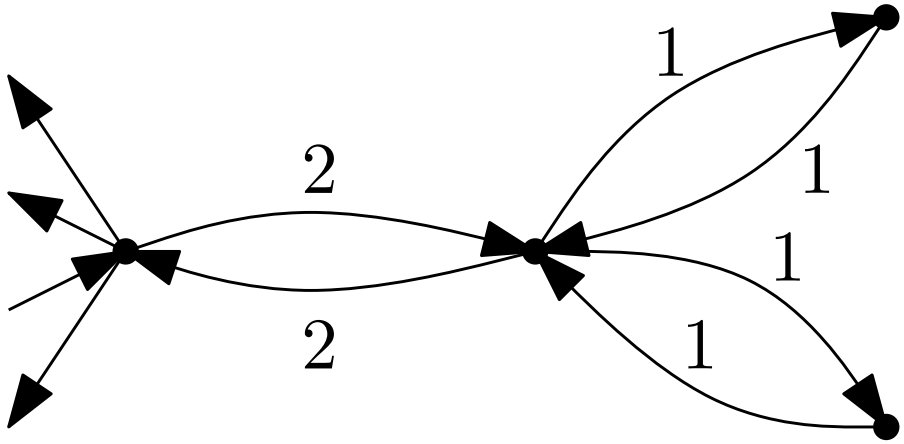
Lösung:

- halte für jeden Knoten zwei Werte vor:
  - $inPenalty(u)$: maximum der reachs entfernter (w, u) Kanten
  - $outPenalty(u)$: maximum der reachs entfernter (u, w) Kanten
- während Vorberechnung:
  - Hänge an jeden Knoten v Pseudoknoten v' mit Abstand $outPenalty(v)$
  - bestimmte neue Höhe $height_p(u)$
  - dann $r_{T_w}(u, v) = \min\{d(u, w) + inPenalty(u), height_p(u)\}$

Penalties

Problem:

- durch entfernen von Knoten und Kanten ändern sich die Längen der kürzesten Wege



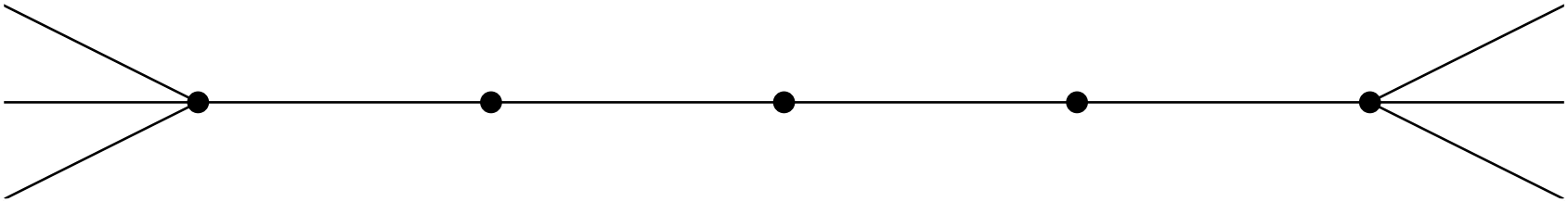
Lösung:

- halte für jeden Knoten zwei Werte vor:
  - $inPenalty(u)$: maximum der reachs entfernter (w, u) Kanten
  - $outPenalty(u)$: maximum der reachs entfernter (u, w) Kanten
- während Vorberechnung:
  - Hänge an jeden Knoten v Pseudoknoten v' mit Abstand $outPenalty(v)$
  - bestimmte neue Höhe $height_p(u)$
  - dann $r_{T_w}(u, v) = \min\{d(u, w) + inPenalty(u), height_p(u)\}$

Shortcuts / Graphkontraktion - Motivation

Beobachtung:

- lange modellierte Pfade in Straßennetzwerken
- erscheint unnötig, alle diese Knoten anzuschauen



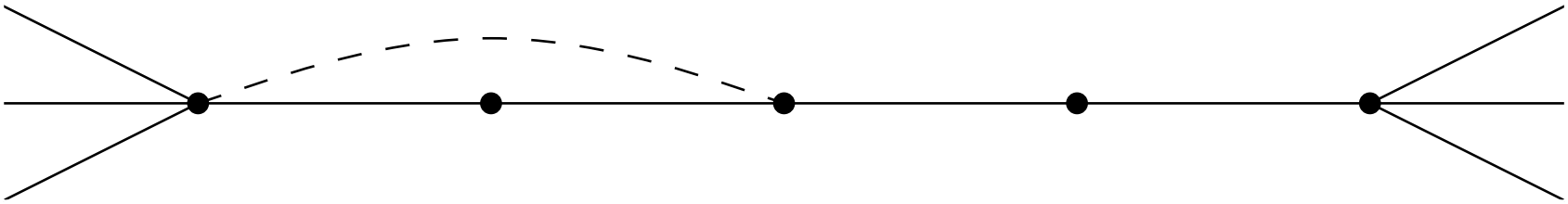
Idee:

- füge iterativ Shortcuts ein
- Shortcuts verkleinern reach Werte

Shortcuts / Graphkontraktion - Motivation

Beobachtung:

- lange modellierte Pfade in Straßennetzwerken
- erscheint unnötig, alle diese Knoten anzuschauen



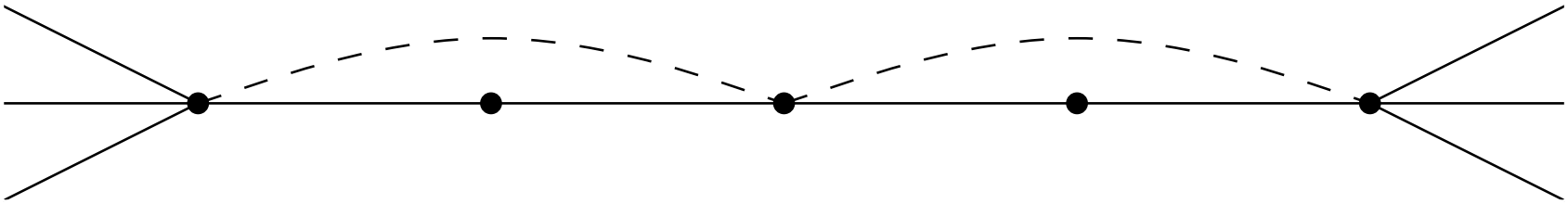
Idee:

- füge iterativ Shortcuts ein
- Shortcuts verkleinern reach Werte

Shortcuts / Graphkontraktion - Motivation

Beobachtung:

- lange modellierte Pfade in Straßennetzwerken
- erscheint unnötig, alle diese Knoten anzuschauen



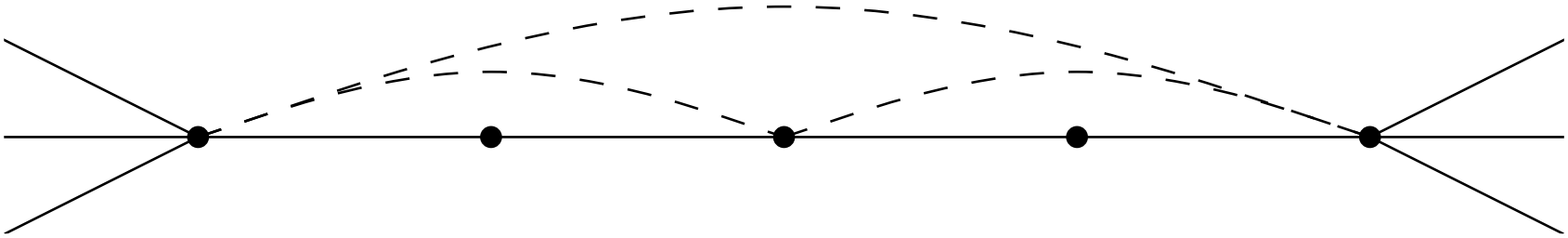
Idee:

- füge iterativ Shortcuts ein
- Shortcuts verkleinern reach Werte

Shortcuts / Graphkontraktion - Motivation

Beobachtung:

- lange modellierte Pfade in Straßennetzwerken
- erscheint unnötig, alle diese Knoten anzuschauen



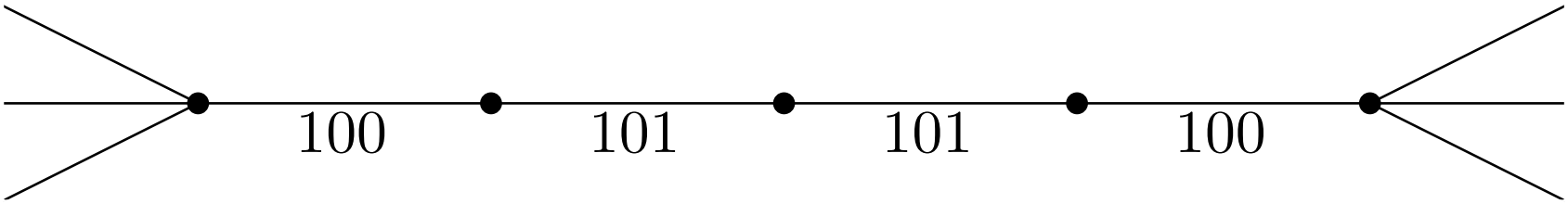
Idee:

- füge iterativ Shortcuts ein
- Shortcuts verkleinern reach Werte

Shortcuts / Graphkontraktion - Motivation

Beobachtung:

- lange modellierte Pfade in Straßennetzwerken
- erscheint unnötig, alle diese Knoten anzuschauen



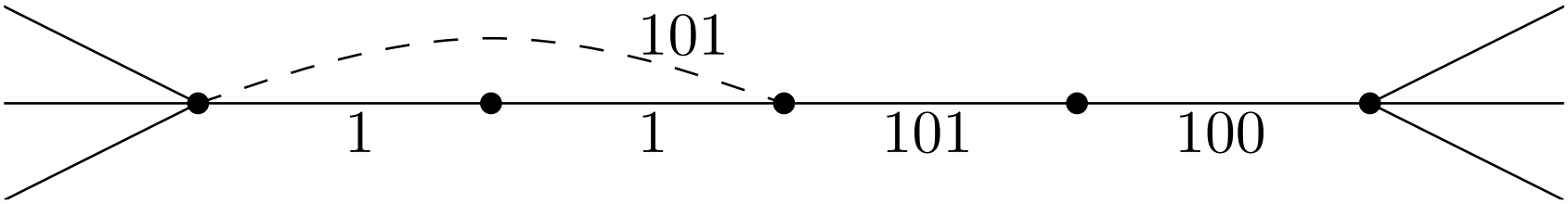
Idee:

- füge iterativ Shortcuts ein
- Shortcuts verkleinern reach Werte

Shortcuts / Graphkontraktion - Motivation

Beobachtung:

- lange modellierte Pfade in Straßennetzwerken
- erscheint unnötig, alle diese Knoten anzuschauen



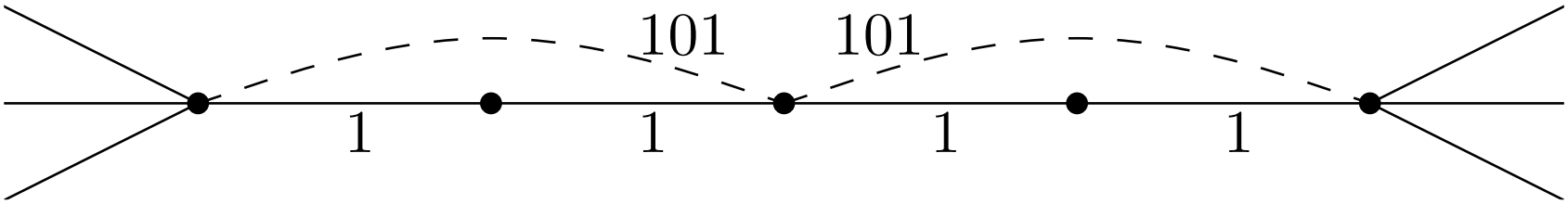
Idee:

- füge iterativ Shortcuts ein
- Shortcuts verkleinern reach Werte

Shortcuts / Graphkontraktion - Motivation

Beobachtung:

- lange modellierte Pfade in Straßennetzwerken
- erscheint unnötig, alle diese Knoten anzuschauen



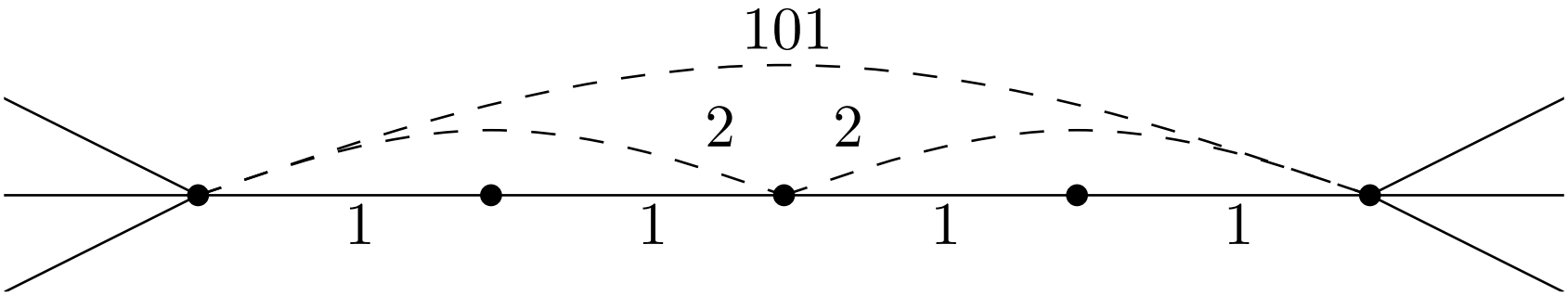
Idee:

- füge iterativ Shortcuts ein
- Shortcuts verkleinern reach Werte

Shortcuts / Graphkontraktion - Motivation

Beobachtung:

- lange modellierte Pfade in Straßennetzwerken
- erscheint unnötig, alle diese Knoten anzuschauen



Idee:

- füge iterativ Shortcuts ein
- Shortcuts verkleinern reach Werte

Kontraktion des Graphen

Vorgehen Kontraktion von Knoten v:

- genau wie bei Contraction Hierarchies
- für entfernte Kante (u, v) setze

$$r(u, v) = \text{len}(u, v) + \mathit{outPenalty}(v)$$

- für entfernte Kante (v, w) setze

$$r(v, w) = \text{len}(v, w) + \mathit{inPenalty}(v)$$

Kontraktion des Graphen

Reihenfolge:

- Reihenfolge in der Knoten kontrahiert werden ändert das Ergebnis
- Benutze priority queue zur Wahl des nächsten Knoten
- Priorität der queue ist „expansion“ $c(v)$

$$c(v) = \deg_{\text{in}}(v) \cdot \deg_{\text{out}}(v)/(\deg_{\text{in}}(v) + \deg_{\text{out}}(v))$$

- Stoppe wenn $c(v) > C$, (meist $C = 1.5$ gewählt)

Verfeinerung der Reach Werte

Problem:

- Approximation schaukelt sich auf
- Knoten mit hohem Reach werden zusätzlich massiv überschätzt
- Diese Knoten sind „wichtige“ für die Anfragen

Idee:

- nach voller Vorberechnung
- extrahiere δ Knoten mit höchstem Reach
- berechne exakten Reach mit APSP für diesen Subgraphen (mit Penalties)
- wenn neuer Reachwert kleiner, aktualsiere

Gesamtübersicht: RE-Algorithmus

Vorberechnung:

- wähle ϵ frei
- iterativ, solange $E \neq \emptyset$
  - Kontrahiere Graphen
  - berechne obere Schranken mit vorherigem Verfahren
  - entferne alle Kanten mit reach $< \epsilon$ aus Graphen
  - setze $\epsilon = k \cdot \epsilon$
- wandle Kanten-Reach in Knoten-Reach um
- verfeinere Knoten-Reach Werte

Output der Vorberechnung:

- Menge der Shortcuts die irgendwann eingefügt wurden
- Obere Schranken für die Reach-Werte aller Knoten

Gesamtübersicht: RE-Algorithmus

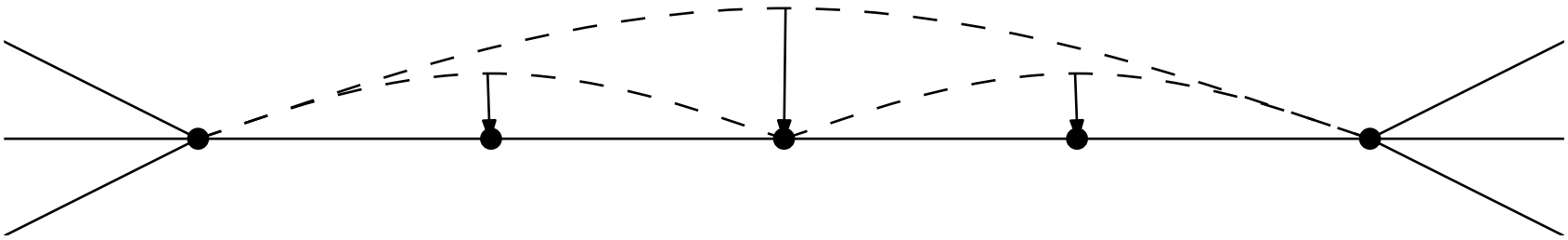
Anfrage:

- Bidirectional Distance-Bounding Reach-Dijkstra auf Graphen mit Shortcuts

Entpacken von Shortcuts

Problem:

- Anfrage auf Graphen mit Shortcuts
- Wenn der kürzeste Weg im Originalgraph erfragt werden soll: Vorgehen wie bei Contraction Hierarchies



Wiederholung: Pfadentpackung

- jeder Shortcut überspringt genau einen Knoten
- speicher Mittelknoten für jeden Shortcut
- entpacke rekursiv bei Bedarf

Optimierungen 2

REAL: RE + ALT

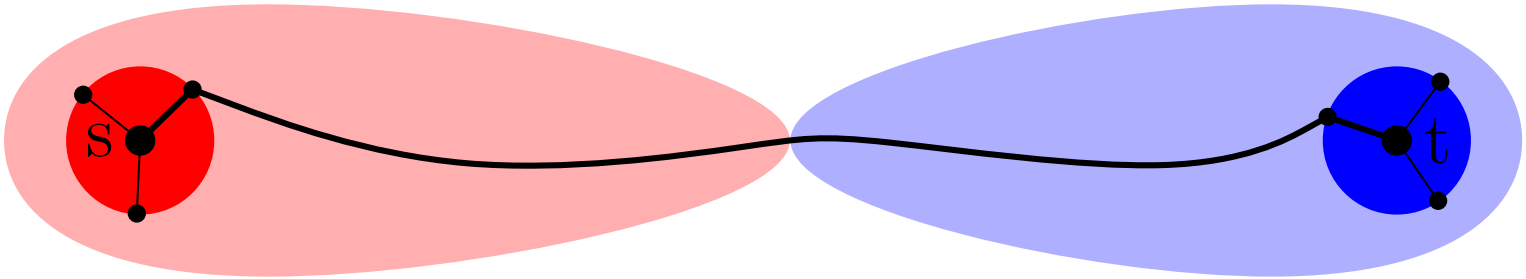
- Kombination mit ALT problemlos möglich
- Doppelter Nutzen:
  - ALT richtet die Suche stärker zum Ziel
  - Die Landmarken können für die Überprüfung $r(u) < d[u]$ und $r(u) < d(u,t)$ benutzt werden

Partielle Landmarken

Beobachtung:

- Landmarken brauchen viel Speicher
- während Anfragen werden meist Knoten mit hohem Reach betrachtet

Partielle Landmarken



Idee:

- speichere Landmarken-Informationen nur für Knoten mit Reach $> R$
- 2-phasiger Anfrage Algorithmus
  - reine Reach-Query bis minimales Element in beiden Queues $> R$
  - wenn kürzester Weg gefunden, fertig
  - sonst sortiere Queues um, so dass ALT-Potentiale enthalten
  - führe REAL Query mit geänderten Queues fort

Proxy Knoten

Problem:

- REAL braucht Potential von jedem Knoten zu t und s
- $r(s)$ und/oder $r(t)$ könnten kleiner R sein
- keine Abstandswerte von den Landmarken zu s und t gespeichert

Proxy Knoten

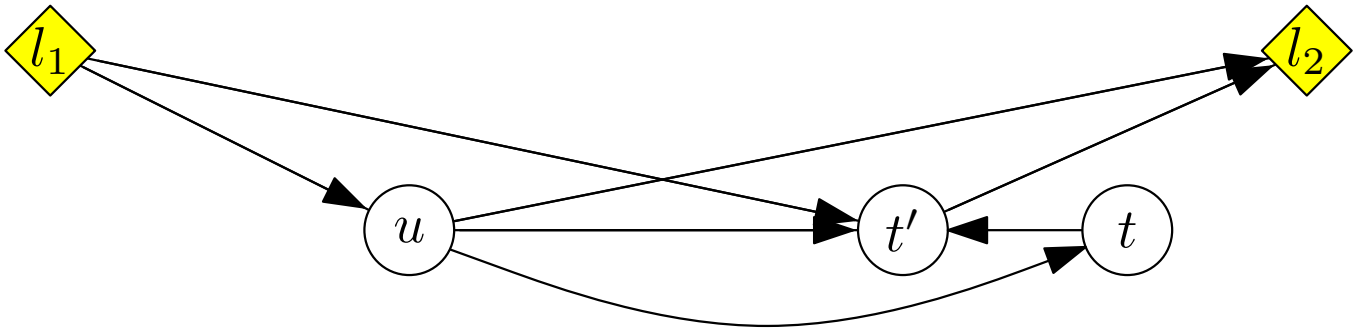
Lösung:

- Proxy-Knoten von Knoten v:

  zu v nächster Knoten v' mit $r(v') \geq R$

- bestimme Proxy-Knoten s' von s und t' von t
- neue Ungleichungen / Potentiale:

$$\begin{aligned} d(u,t) &\leq d(u,l_2) - d(t',l_2) - d(t,t') \\ d(u,t) &\leq d(l_1,t') - d(l_1,u) - d(t,t') \end{aligned}$$



Experimente

Eingaben:

- Straßennetzwerke
  - Europa: 18 Mio. Knoten, 42 Mio. Kanten
  - USA: 22 Mio. Knoten, 56 Mio. Kanten

Evaluation:

- Vorberechnung in Minuten und zusätzliche Bytes pro Knoten
- durchschnittlicher Suchraum (#abgearbeitete Knoten) und Suchzeiten (in *ms*) von 10 000 Zufallsanfragen

RE mit/ohne Shortcuts

Eingabe: BayArea: 330k Knoten

		Vorb.	Anfrage	
shortcuts	reach	time [min]	#settled	time [ms]
nein	approx	52	13369	6.44
nein	exakt	966	11194	6.05
ja	approx	3	1590	1.17
ja	exakt	980	1383	0.97

Beobachtung:

- Shortcuts reduzieren Vorberechnungszeit
- beschleunigen Anfragen
- approximative Vorberechnung deutlich schneller
- Verlust in Anfragen nicht sehr groß

RE: Verfeinerung

USA	**Vorb.**	**Anfrage**	
δ	time [min]	#settled	time
0	32	2 555	2.00
12 235	36	2 448	1.84
24 469	44	2 317	1.81
48 938	67	2 159	1.70
97 876	148	2 086	1.66

Beobachtung:

- Verfeinerung bringt bis zu 15%
- aber zu hohe Werte erhöhen Vorberechnungszeit zu massiv

Zufallsanfragen: RE und REAL

	Vorberechnung		Anfrage		
landmarks	sparsity	Zeit [min]	Platz [byte/n]	Such raum	Zeit [ms]
0	–	44	15	2 317	1.81
16	1	64	104	675	1.06
16	4	64	39	689	1.34
16	16	64	22	730	1.31
16	64	64	17	888	1.38
64	4	121	113	493	1.02
64	16	121	43	540	0.98
64	64	121	25	743	1.02

Beobachtungen:

- Zuschalten von Landmarken zahlt sich aus
- Partielle Landmarkeninformationen reduziert Platzverbrauch

Übersicht

	Vorberechnung		Anfrage		
	Zeit [h:m]	Platz [byte/n]	Such raum	Zeit [ms]	Beschl.
Dijkstra	0:00	0	9 114 385	5 591.6	1
ALT-16	1:25	128	74 669	53.6	104
ALT-64	1:08	512	25 324	19.6	285
Arc-Flags (128)	17:08	10	2 764	0.8	6 988
RE	1:22	13	4 643	3.5	1 597
REAL-(16,1)	1:36	81	814	1.2	4 588
REAL-(64,16)	2:20	35	679	1.1	5 037

Beobachtung:

- REAL ähnliche Performance wie Arc-Flags
- deutlich kürzere Vorberechnungszeiten
- höherer Speicherverbrauch

Ende

Literatur (Reach, RE, REAL):

- Andrew V. Goldberg and Haim Kaplan and Renato F. Werneck:
  Reach for A*: Shortest Path Algorithms with Preprocessing
  In: *Shortest Paths: Ninth DIMACS Implementation Challenge, 2009.*