Algorithmen für Routenplanung – Vorlesung 10

Daniel Delling

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

Letztes Mal: Zeitabhängige Netzwerke (Basics)

Szenario:

- Historische Daten für Verkehrssituation verfügbar
- Verkehrssituation vorhersagbar
- berechne schnellsten Weg bezüglich der erwarteten Verkehrssituation (zu einem gegebenen Startzeitpunkt)

Anfrageszenarien

Zeit-Anfrage:

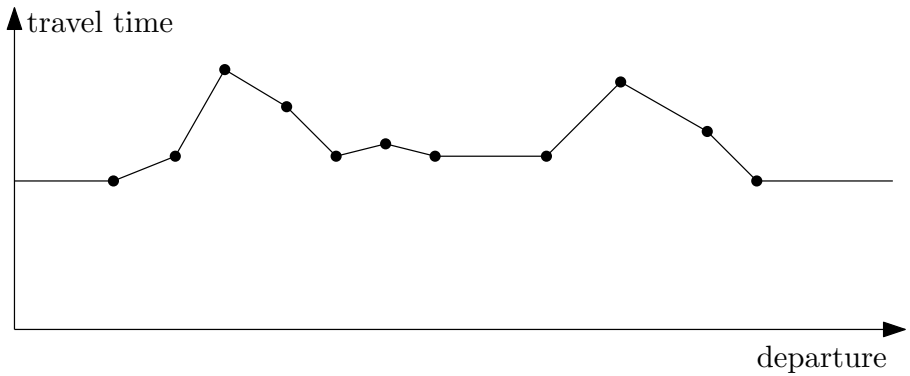
- finde kürzesten Weg für Abfahrtszeit τ
- analog zu Dijkstra?

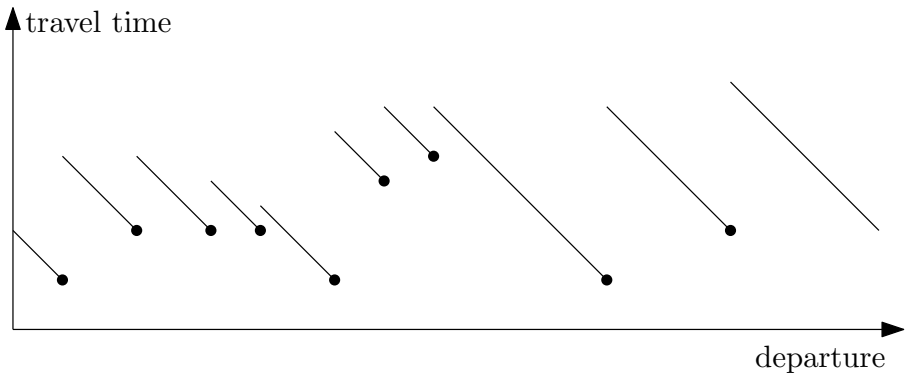
Profil-Anfrage:

- finde kürzesten Weg für alle Abfahrtszeitpunkte
- analog zu Dijkstra?

Modellierung

- hänge Funktionen an Kanten
- Wege hängen von Abfahrtszeitpunkt ab
- zwei Typen von Funktionen: Schiene und Straße





Operationen

Laufzeit Operationen

- gleich für Schiene und Straße
- $O(\log |I|)$ für Auswertung
- $O(|I^f| + |I^g|)$ für linken und minimum

Speicherverbrauch

- Public Transport geringer
- link:

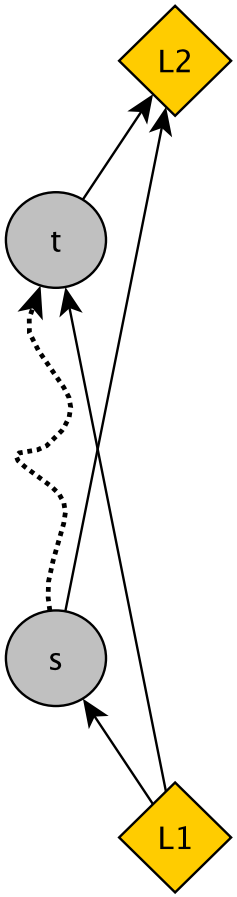
$$|I^{f \oplus g}| \leq \min\{|I^f|, |I^g|\} \quad \text{vs.} \quad |I^{f \oplus g}| \approx |I^f| + |I^g|$$

- merge:

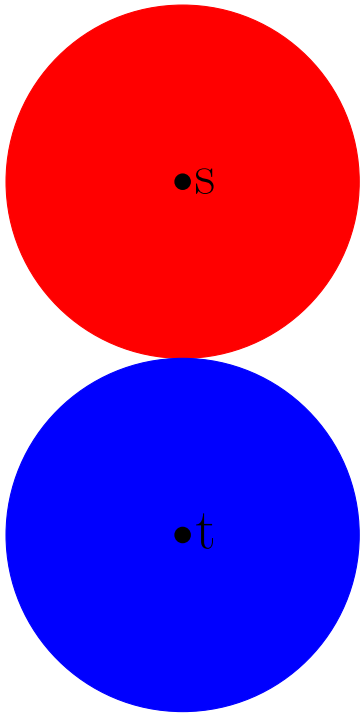
$$|I^{\min\{f,g\}}| \leq |I^f| + |I^g| \quad \text{vs. eventuell} \quad |I^{\min\{f,g\}}| > (|I^f| + |I^g|)$$

Heute: Zeitabhängige Beschleunigungstechniken

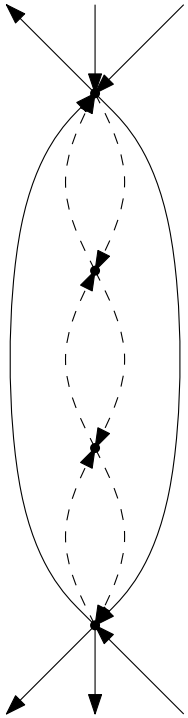
Landmarken



Bidirektionale Suche



Kontraktion



Arc-Flags

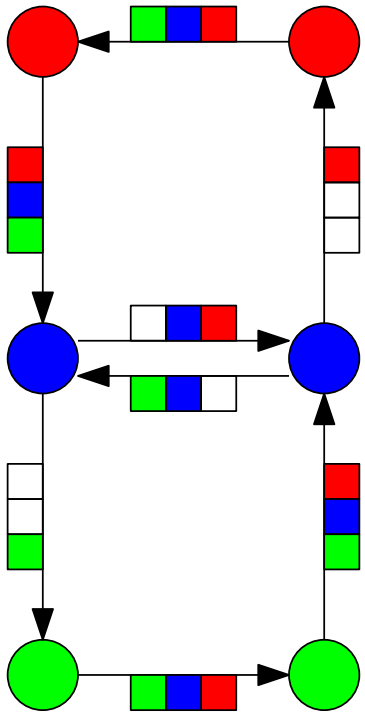
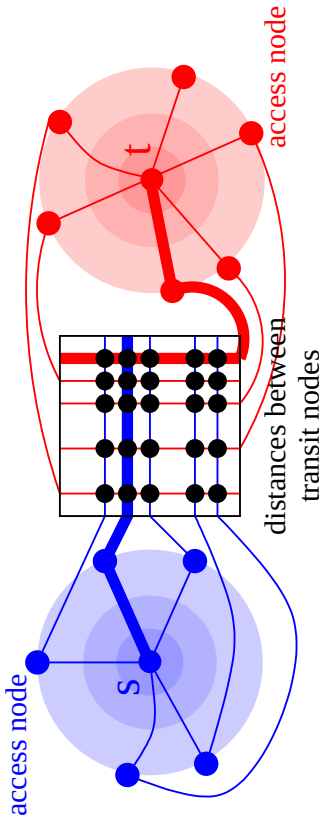


Table-Lookups



Landmarken

Vorberechnung:

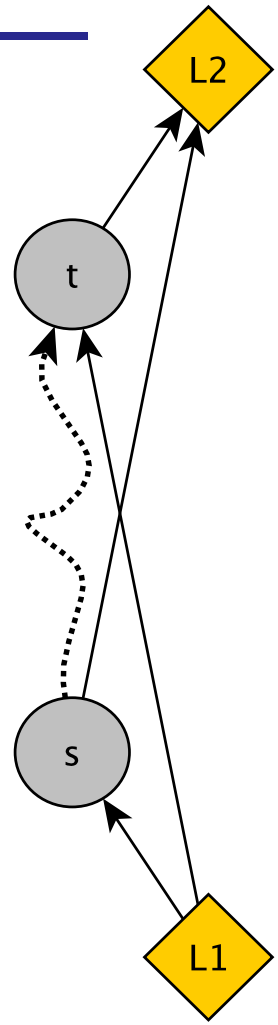
- wähle eine Hand voll (≈ 16) Knoten als Landmarken
- berechne Abstände von und zu allen Landmarken

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

Beobachtung:

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

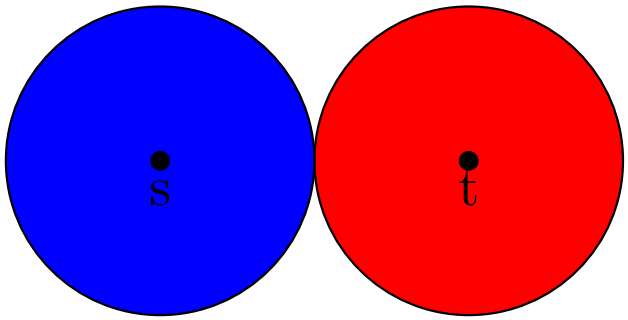
$$len_\pi(u,v) = len(u,v) - \pi(u) + \pi(v) \geq 0$$

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

Somit:

- Vorberechnung auf lowerbound Graphen
- korrekt aber eventuell langsamere Anfragezeiten

Bidirektionale Suche



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

Anpassung

Zeitanfragen:

- Ankunft unbekannt $\Rightarrow$ Rückwärtsuche?
- Rückwärtssuche nur zum Einschränken der Vorwärtssuche benutzen
- von Beschleunigungstechnik zu Beschleunigungstechnik verschieden

Profilanfragen:

- Anfrage zu allen Startzeitpunkten
- somit Rückwärtsuche kein Problem
- μ temporäre Abstandsfunktion
- breche ab, wenn $\overline{\mu} \leq \overrightarrow{Q} + \overleftarrow{Q}$

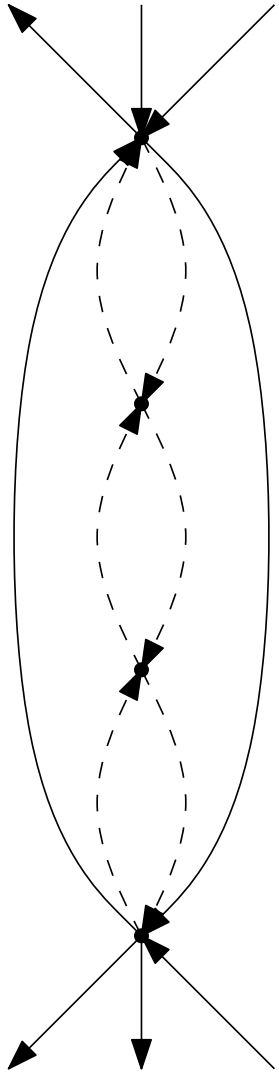
Kontraktion

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



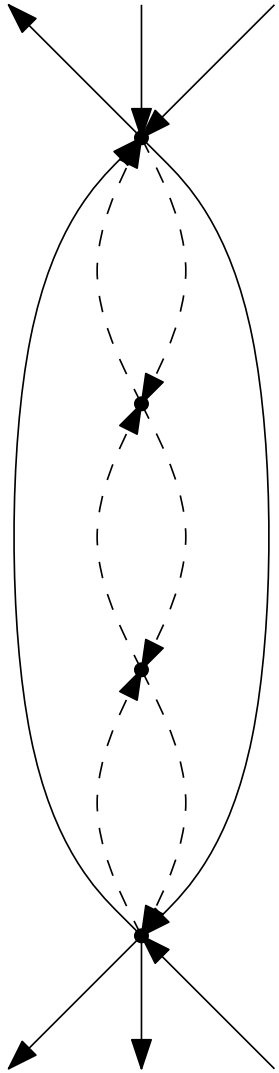
Anpassung Knoten-Reduktion

Beobachtung:

- Verfahren unabhängig von Metrik
- Shortcuts müssen dem Pfade entsprechen

Somit:

- linken der Funktionen zu einer Shortcut-Funktion
- Speicherverbrauch steigt an



Anpassung Kanten-Reduktion

Statisch:

- lösche Kante (u, v), wenn (u, v) nicht Teil des kürzesten Weges von u nach v ist, also $len(u, v) < d(u, v)$
- lokale Dijkstra-Suche von u

Zeitabhängig:

- lösche Kante (u, v), wenn (u, v) nicht Teil aller kürzesten Weges von u nach v ist, als $len(u, v) < d_*(u, v)$
- lokale Profilsuche
- Problem: deutlich langsamer

Idee:

- lösche zunächst Kanten (u, v) für die $\overline{len(u, v)} < \underline{d_*(u, v)}$ gilt
- danach lokale Profilsuche

Korridorsuche

Idee:

- führe zunächste zwei Dijkstra-Suchen mit $\underline{len}$ und $\overline{len}$ durch
- relaxiere dann nur solche Kanten (u, v), für die $\underline{d(s,u)} + \underline{(u,v)} \leq \overline{d(s,v)}$ gilt

Anmerkung:

- kann auch bei Beschleunigung einer s–t Profil-Suche genutzt werden

Approximation der Shortcuts

Problem:

- hoher Speicherbedarf der Shortcuts

Ideen:

- Shortcuts on-the-fly entpacken und dann Gewicht des Pfades berechnen
- speichere Approximationen der Funktionen, führe dann Korridorsuche bei Query auf Originalgraphen durch.
- durch speichern von Approximationen ist dies genauer.

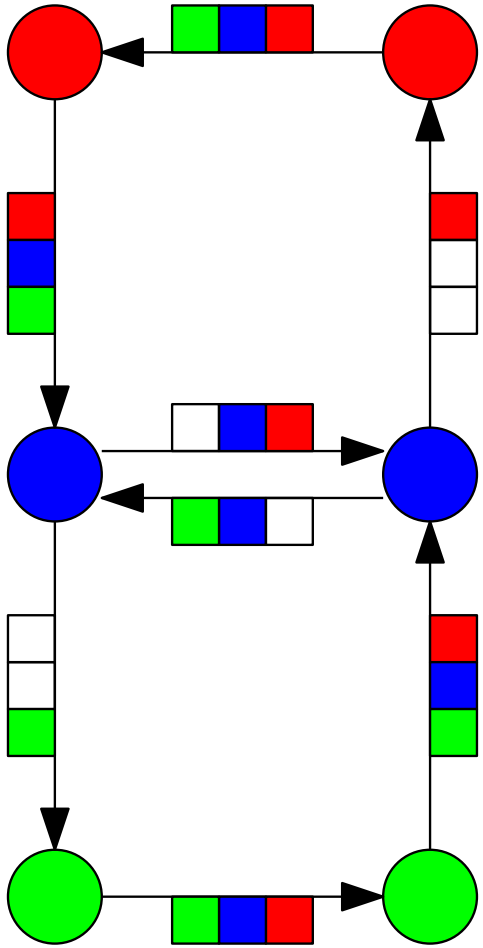
Arc-Flags

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



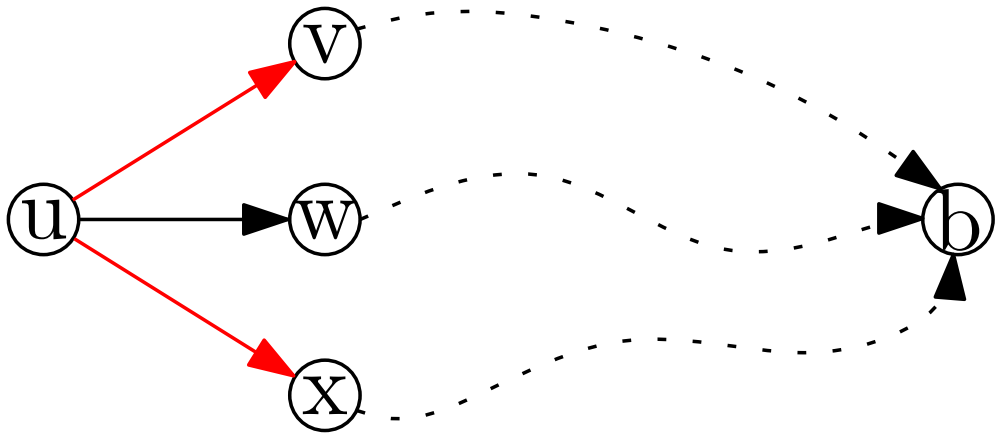
Anpassung

Idee:

- ändere Intuition einer gesetzten Flagge
- Konzept bleibt gleich: Eine Flagge pro Kante und Region
- setze Flagge
  - `zeitabhängig`: wenn Kante im Laufe des Tages "wichtig" ist

Anpassung:

- für alle Randknoten b und alle Knoten u:
- Berechne Abstandsfunktion $d_*(u, b)$
- setze Flagge wenn gilt $len(u, v) \oplus d_*(v, b) \not> d_*(u, b)$



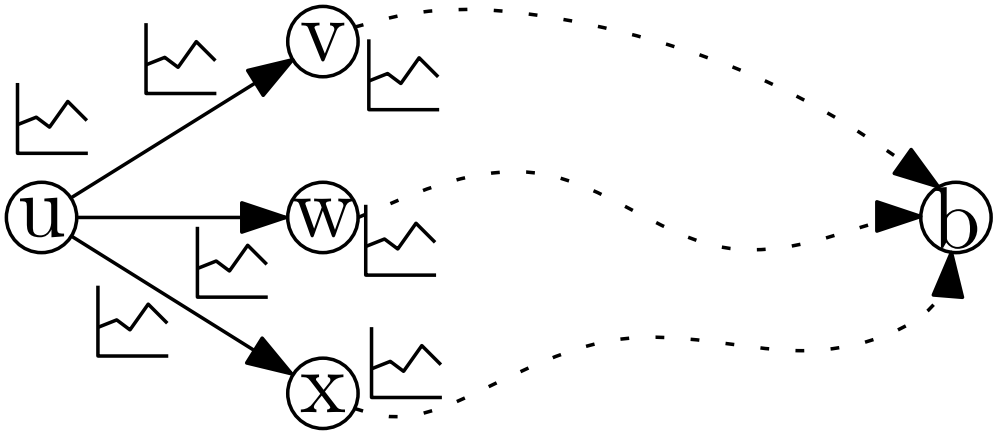
Approximation Arc-Flags (zeitabhängiges Szenario)

Beobachtung:

- viele Interpolationspunkte
- Berechnung der Abstandsfunktionen ist sehr zeitintensiv
- Laufzeit stark abhängig von der Komplexität der Funktionen

Idee:

- benutze Über- and Unterapproximation
- ⇒ schnellere Vorberechnung, langsamere Anfragen
- ⇒ aber immer noch korrekt



Heuristische Flaggen

Idee:

- führe von jedem Randknoten K Zeitanfragen aus
- mit fester Ankunftszeit
- setze Flagge, wenn Kante auf einem dem Bäume eine Baumkante ist

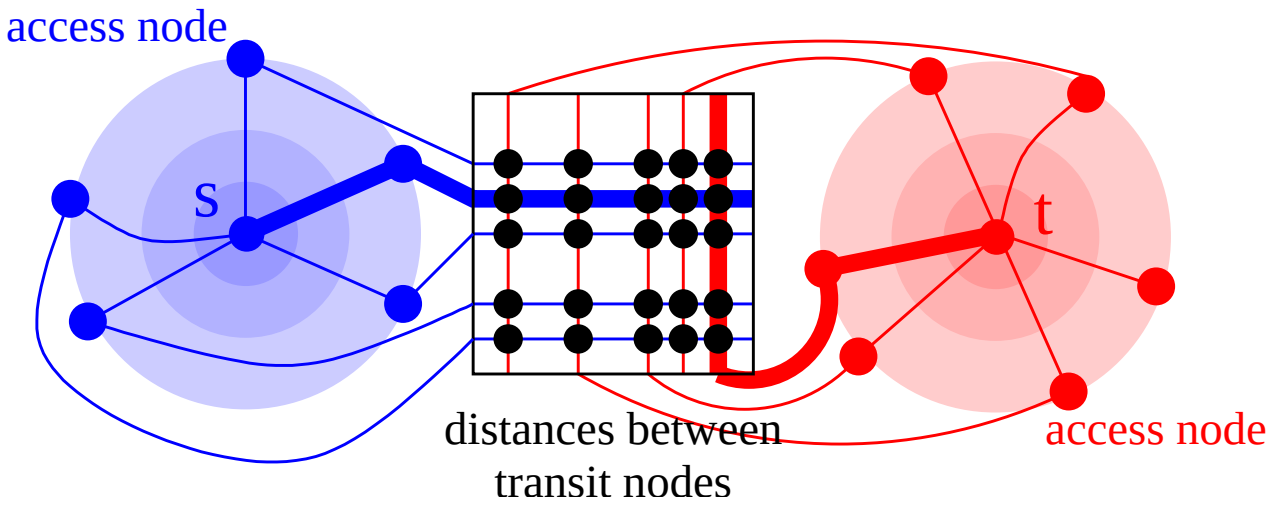
Beobachtungen:

- Flaggen eventuell nicht korrekt
- ein Pfad wird aber immer gefunden
- Fehlerrate?

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:

- Speicherverbrauch deutlich zu groß?

Diskussion Anpassung der Basismodule

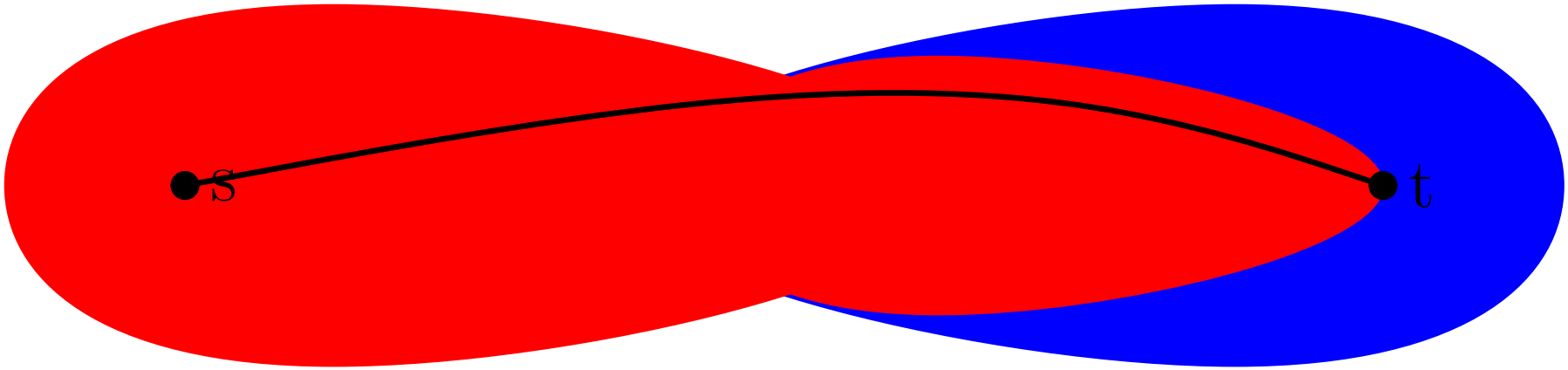
Basismodule:

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

somit folgende Algorithmen gute Kandidaten

- ≫ ALT
- ≫ Core-ALT
- ≫ SHARC
- ≫ Contraction Hierarchies

Bidirektionaler zeitabhängiger ALT



Idee - Drei Phasen:

1. Vorwärts zeitabhängig, Rückwärtssuche benutzt Minima der Funktionen, fertig wenn Suchen sich treffen
2. beide Suchen arbeiten weiter, Rückwärtssuche besucht nicht Knoten der Vorwärtssuche, fertig wenn $minKey(\overleftarrow{Q}) > \mu$
3. nur Vorwärtssuche geht weiter, bis t abgearbeitet worden ist

Approximation

Beobachtung:

- Phase 2 läuft recht lang weiter, bis $minKey(\overleftarrow{Q}) > \mu$ gilt
- insbesondere dann schlecht, wenn die lower bounds stark vom echten Wert abweichen

Approximation:

- breche Phase 2 bereits ab, wenn $minKey(\overleftarrow{Q}) * K > \mu$ gilt
- dann ist der berechnete Weg eine K-Approximation des kürzesten

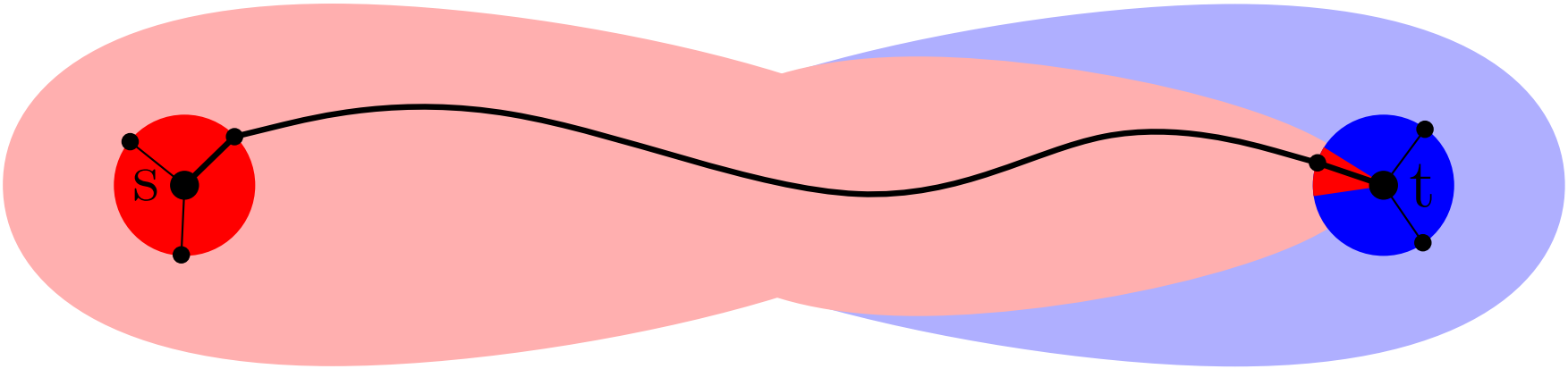
Experimente

scenario	algorithm	K	Error relative rate	relative av.	relative max	abs. max [s]	Query #sett. nodes	#rel. edges	time [ms]
midweek	uni-ALT	–	0.0%	0.000%	0.00%	0	200 236	239 112	147.20
	TDALT	1.00	0.0%	0.000%	0.00%	0	116 476	138 696	98.27
		1.15	12.4%	0.094%	14.32%	1 892	50 764	60 398	36.91
		1.50	12.5%	0.097%	27.59%	1 892	50 742	60 371	36.86
Saturday	uni-ALT	–	0.0%	0.000%	0.00%	0	148 331	177 568	100.07
	TDALT	1.00	0.0%	0.000%	0.00%	0	63 717	76 001	47.41
		1.15	10.5%	0.088%	13.97%	2 613	50 042	59 607	36.00
		1.50	10.6%	0.089%	26.17%	2 613	50 036	59 600	35.63
Sunday	uni-ALT	–	0.0%	0.000%	0.00%	0	142 631	170 670	92.79
	TDALT	1.00	0.0%	0.000%	0.00%	0	58 956	70 333	42.96
		1.15	10.4%	0.088%	14.28%	1 753	50 349	59 994	36.04
		1.50	10.5%	0.089%	32.08%	1 753	50 345	59 988	35.74

Core-ALT *(Landmarken, bidirektionale Suche, Kontraktion)*

Idee

- begrenze Beschleunigungstechnik auf kleinen Subgraphen (Kern)



Vorberechnung

- kontrahiere Graphen zu einem Kern
- Landmarken nur im Kern

Anfrage

- Initialphase: normaler Dijkstra
- benutze Landmarken nur im Kern
- zeitabhängig:
  - Rückwärtssuche ist zeitunabhängig
  - Vorwärtssuche darf alle Knoten der Rückwärtssuche besuchen

Experimente

scenario	K	Preproc. time [min]	Preproc. space [B/n]	Error rate	Error relative av.	Error relative max	Error abs. max [s]	Query #settled nodes	Query #relaxed edges	Query time [ms]
	1.00	9	50.3	0.0%	0.000%	0.00%	0	2 984	11 316	4.84
Monday	1.15	9	50.3	8.3%	0.051%	11.00%	1 618	1 588	5 303	1.84
	1.50	9	50.3	8.3%	0.052%	17.25%	1 618	1 587	5 301	1.84
	1.00	9	50.3	0.0%	0.000%	0.00%	0	3 190	12 255	5.36
midweek	1.15	9	50.3	8.2%	0.051%	13.84%	2 408	1 593	5 339	1.87
	1.50	9	50.3	8.2%	0.052%	13.84%	2 408	1 592	5 337	1.86
	1.00	8	44.9	0.0%	0.000%	0.00%	0	3 097	12 162	5.21
Friday	1.15	8	44.9	7.8%	0.052%	11.29%	2 348	1 579	5 376	1.82
	1.50	8	44.9	7.8%	0.054%	21.19%	2 348	1 579	5 374	1.82
	1.00	6	27.8	0.0%	0.000%	0.00%	0	1 856	7 188	2.42
Saturday	1.15	6	27.8	4.4%	0.031%	11.50%	1 913	1 539	5 542	1.71
	1.50	6	27.8	4.4%	0.031%	24.17%	1 913	1 539	5 541	1.71
	1.00	5	19.1	0.0%	0.000%	0.00%	0	1 773	6 712	2.13
Sunday	1.15	5	19.1	4.0%	0.029%	12.72%	1 400	1 551	5 541	1.68
	1.50	5	19.1	4.1%	0.029%	17.84%	1 400	1 550	5 540	1.68

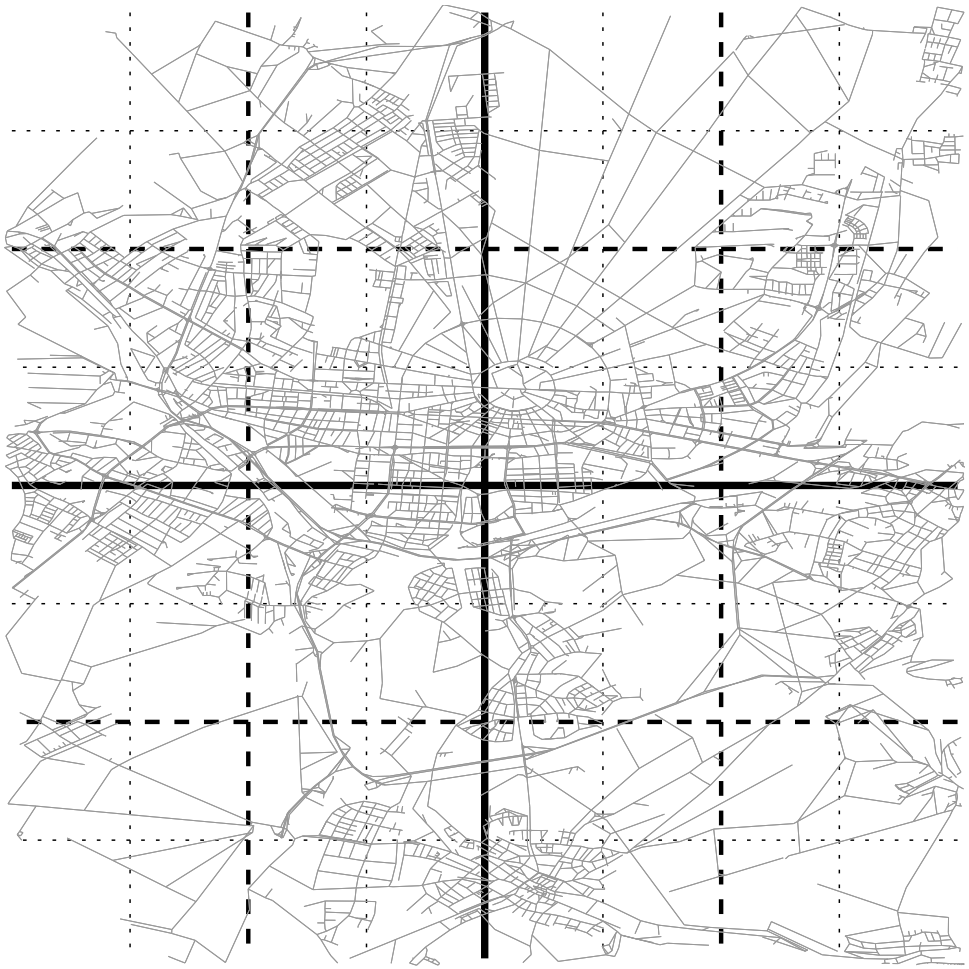
SHARC (Kontraktion, Arc-Flags)

Vorberechnung:

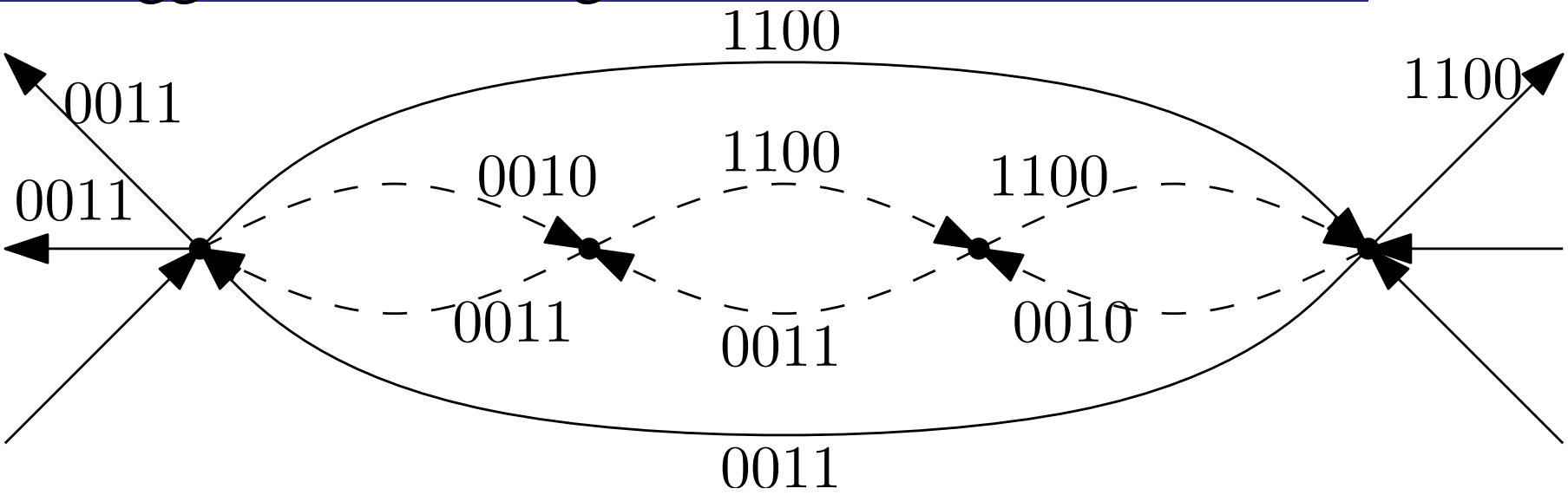
- Multi-Level-Partition
- iterativer Prozess:
  - kontrahiere Subgraphen
  - berechne Flaggen
- Flaggenverfeinerung

Anpassung:

- Kontraktion und Flaggen berechnung anpassen
- Verfeinerung?



Flaggenverfeinerung



Vorgehen:

- verfeinere Flaggen
- propagiere Flaggen von wichtigen zu unwichtigen Kanten
- statisch: mittels lokaler Suche
- zeitabhängig: mittels lokaler Profilsuche

Experimente

		Preprocessing				Time-Queries					
		time	space	edge	points	#del.	speed	#rel.	speed	time	speed
scenario	algom	[h:m]	[B/n]	inc.	inc.	mins	up	edges	up	[ms]	up
Monday	eco	1:16	156.6	25.4%	366.8%	19 136	124	101 176	54	24.55	63
midweek	eco	1:16	154.9	25.4%	363.8%	19 425	119	104 947	51	25.06	60
Friday	eco	1:10	142.0	25.4%	358.0%	17 412	134	92 473	58	22.07	69
Saturday	eco	0:42	90.3	25.0%	283.6%	5 284	441	19 991	269	5.34	276
	agg	48:57	84.3	24.5%	264.4%	721	3 229	1 603	3 349	0.58	2 554
Sunday	eco	0:30	64.6	24.6%	215.8%	2 142	1 097	6 549	826	1.86	787
	agg	27:20	60.7	24.1%	202.6%	670	3 504	1 439	3 759	0.50	2 904
no traffic	static	0:06	13.5	23.9%	23.9%	591	3 790	1 837	2 810	0.30	4 075

Daniel Delling – Algorithmen für Routenplanung – Vorlesung 10
Lehrstuhl für Algorithmik I
Institut für theoretische Informatik

Approximation

		Prepro		Error			Time-Queries					
		time	space	error	max	max	#del.	spd	#rel.	spd	time	spd
scenario	algo	[h:m]	[B/n]	-rate	rel.	abs.[s]	mins	up	edges	up	[ms]	up
Monday	heu	3:30	138.2	0.46%	0.54%	39.3	810	2 935	1 593	3 439	0.69	2 253
midweek	heu	3:26	137.2	0.82%	0.61%	48.3	818	2 820	1 611	3 297	0.69	2 164
Friday	heu	3:14	125.2	0.50%	0.50%	50.3	769	3 044	1 522	3 543	0.64	2 358
Saturday	heu	2:13	80.4	0.18%	0.23%	16.9	666	3 499	1 336	4 018	0.51	2 887
Sunday	heu	1:48	58.8	0.09%	0.36%	14.9	635	3 699	1 271	4 255	0.46	3 163
no	static	0:06	13.5	0.00%	0.00%	0.0	591	3 790	1 837	2 810	0.30	4 075

Beobachtung:

- ≫ Fehler sehr gering
- ≫ hoher Speicherverbrauch

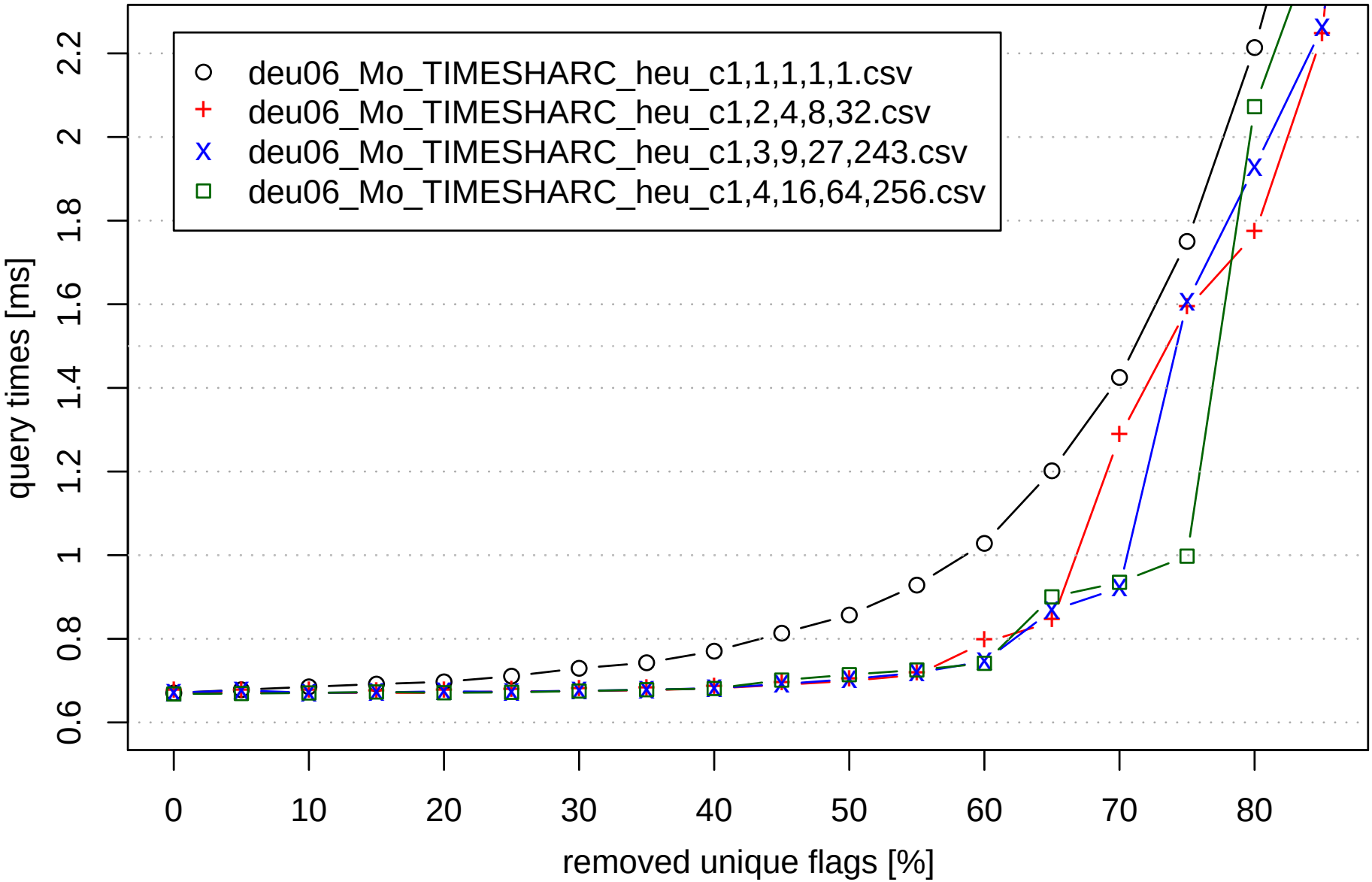
Profilsuchen

traffic	var.	Time-Queries #del. mins	#rel. edges	time [ms]	Profile-Queries #del. mins	profile /time	#re-ins.	#rel. edges	time [ms]	profile /time
Monday	eco	19 136	101 176	24.55	19 768	1.03	402	208 942	51 122	2 082.6
	heu	810	1 593	0.69	1 071	1.32	24	3 597	1 008	1 460.9
midweek	eco	19 425	104 947	25.06	20 538	1.06	432	222 066	60 147	2 400.3
	heu	818	1 611	0.69	1 100	1.35	27	3 731	1 075	1 548.4
Friday	eco	17 412	92 473	22.07	19 530	1.12	346	204 545	52 780	2 391.9
	heu	769	1 522	0.64	1 049	1.36	21	3 551	832	1 293.2
Saturday	eco	5 284	19 991	5.34	5 495	1.04	44	41 956	3 330	624.0
	agg	721	1 603	0.58	865	1.20	9	3 269	134	232.5
	heu	666	1 336	0.51	798	1.20	8	2 665	98	191.9
Sunday	eco	2 142	6 549	1.86	2 294	1.07	12	13 563	536	288.1
	agg	670	1 439	0.50	781	1.17	5	2 824	57	113.5
	heu	635	1 271	0.46	738	1.16	5	2 449	45	97.9

Kompression

Gründe für Overhead:

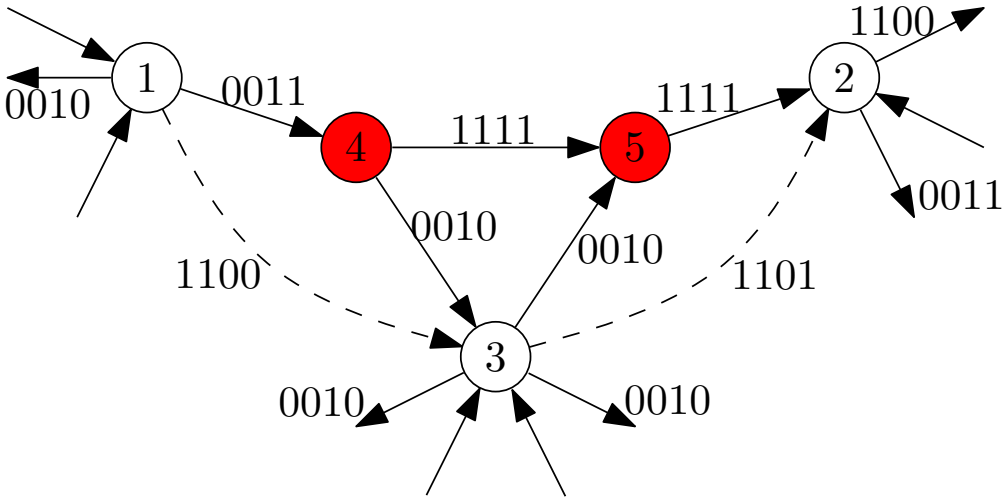
- Regionsinformation
- Flaggen speichern
- Shortcuts (Einträge im Kantenarray)
- zusätzliche Interpolationspunkte


deu06_Mo_TIMESHARC_heu_c1,1,1,1,1.csv
deu06_Mo_TIMESHARC_heu_c1,2,4,8,32.csv
deu06_Mo_TIMESHARC_heu_c1,3,9,27,243.csv
deu06_Mo_TIMESHARC_heu_c1,4,16,64,256.csv
query times [ms]
0.6
0.8
1
1.2
1.4
1.6
1.8
2
2.2
0
10
20
30
40
50
60
70
80
removed unique flags [%]

Shortcut Kompression

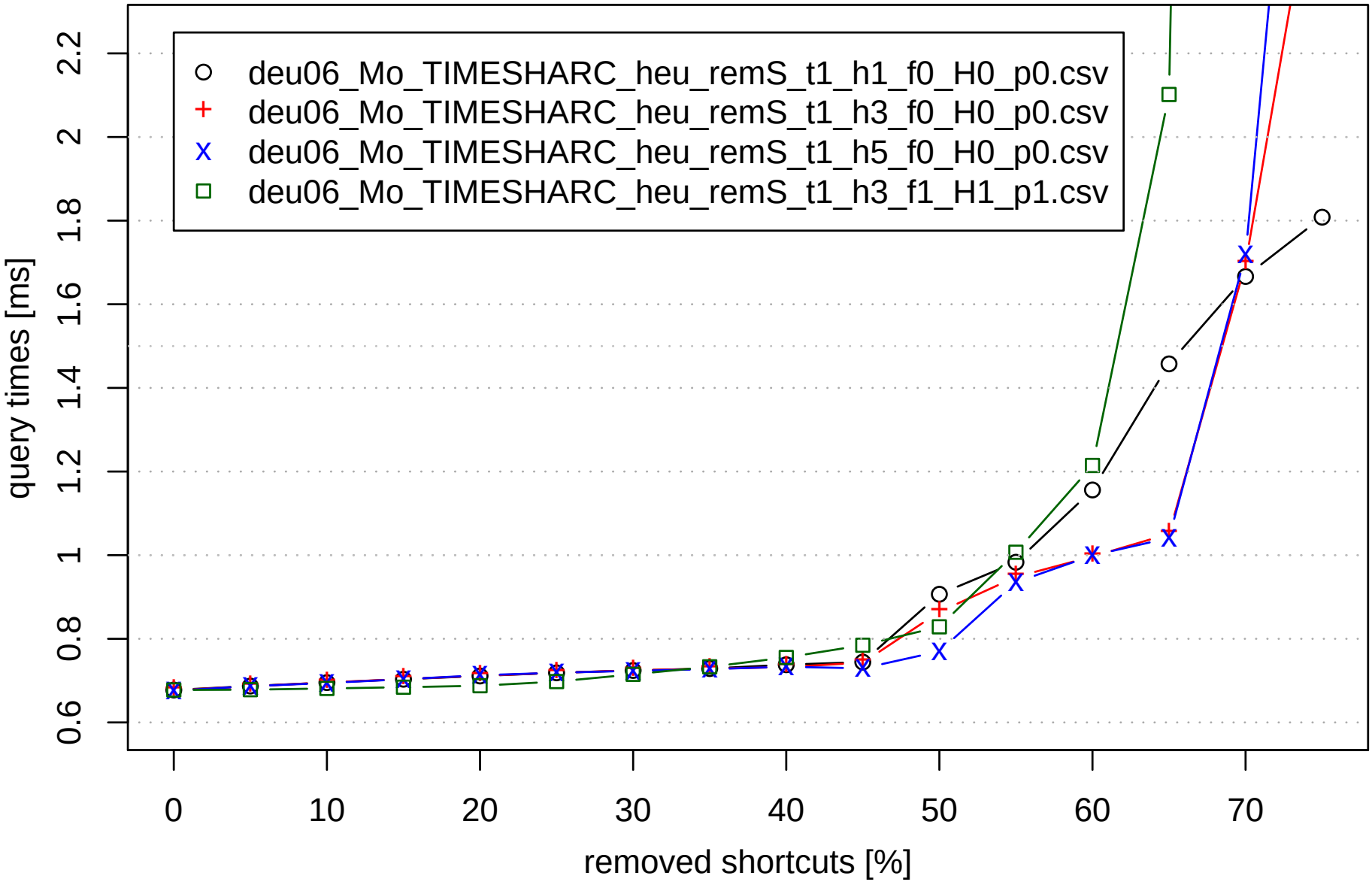
Beobachtung:

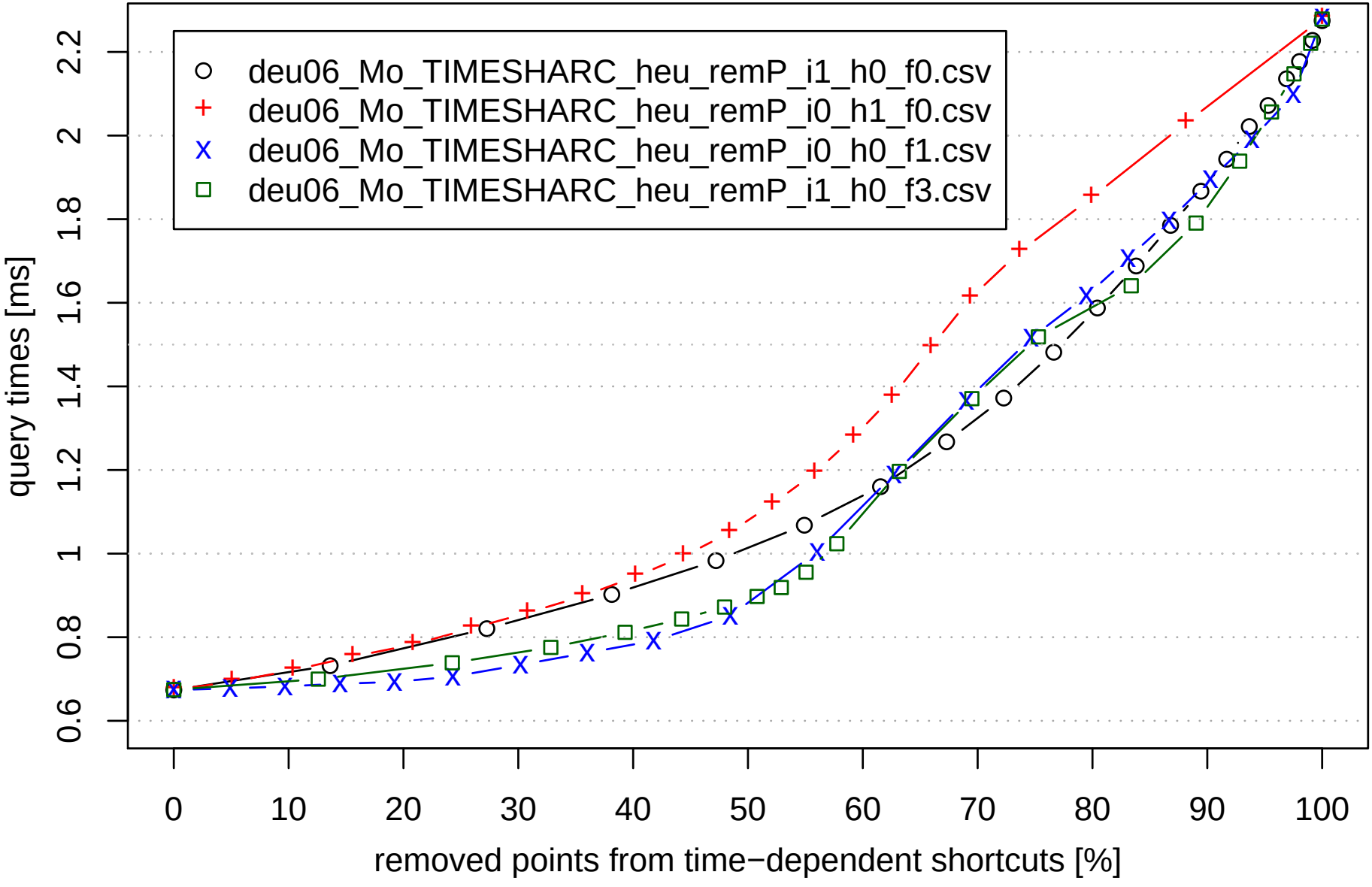
- Shortcut entspricht einem Pfad
- manche Shortcuts erscheinen unwichtig



Idee:

- entferne (manche) Shortcuts nach Vorberechnung
- vererbe Flaggen an erste Kante des Pfades
- welche sind wichtig?
- außerdem: entferne Interpolationspunkte und entpacke on-the-fly


deu06_Mo_TIMESHARC_heu_remS_t1_h1_f0_H0_p0.csv
deu06_Mo_TIMESHARC_heu_remS_t1_h3_f0_H0_p0.csv
deu06_Mo_TIMESHARC_heu_remS_t1_h5_f0_H0_p0.csv
deu06_Mo_TIMESHARC_heu_remS_t1_h3_f1_H1_p1.csv
query times [ms]
0.6
0.8
1
1.2
1.4
1.6
1.8
2
2.2
0
10
20
30
40
50
60
70
removed shortcuts [%]



Ergebnis:

Speicherverbrauch kann auf 10 - 15 bytes pro Knoten gedrückt werden

Contraction Hierarchies

Vorberechnung:

- benutze gleiche Knotenordnung
- kontrahiere zeitabhängig
- erzeugt Suchgraphen $G' = (V, \uparrow E \cup \downarrow E)$

Anfrage

- Rückwärts aufwärts mittels min-max Suche
- markiere alle Kanten (u, v) aus $\downarrow E$ mit $\underline{d(u, v)} + \underline{d(v, t)} \leq \overline{d(u, v)}$
- diese Menge sei $\downarrow E'$
- zeitabhängige Vorwärtsuche in $(V, \uparrow E \cup \downarrow E'$

Experimente

input	type of ordering	Contr. ordering [h:m]	Contr. const. [h:m]	Contr. space [B/n]	Queries #delete mins	Queries time [ms]	Queries speed up
Monday	static min	0:05	0:20	1 035	518	1.19	1 240
	timed	1:47	0:14	750	546	1.19	1 244
midweek	static min	0:05	0:20	1 029	528	1.22	1 212
	timed	1:48	0:14	743	551	1.19	1 242
Friday	static min	0:05	0:16	856	497	1.11	1 381
	timed	1:30	0:12	620	526	1.13	1 362
Saturday	static min	0:05	0:08	391	428	0.81	1 763
	timed	0:52	0:08	282	529	1.09	1 313
Sunday	static min	0:05	0:06	248	407	0.71	1 980
	timed	0:38	0:07	177	541	1.07	1 321

Kompression

Idee:

- Speichere nur Approximationen der Shortcuts
- dadurch weniger Speicherverbrauch
- aber auch langsamere Queryzeiten
- ongoing work

Ergebnisse:

- 2 bis 3 mal langsamer als normale TCH
- Speicherverbrauch: ca. 140 Bytes pro Knoten
- somit immer noch höher als bei SHARC

Zusammenfassung Zeitabhängige Beschleunigungst.

Basismodule:

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

somit folgende Algorithmen gut in zeitabhängigen Szenarien verwendbar

- ≫ ALT
- ≫ Core-ALT
- ≫ SHARC
- ≫ Contraction Hierarchies

Literatur

Zeitabhängige Beschleunigungstechniken:

- Nannicini et al. 08
- Delling 08,09
- Delling/Nannicini 08
- Batz et al. 09

Anmerkung:

- wird auf der Homepage verlinkt