

# Algorithmen für Routenplanung

16. Vorlesung, Sommersemester 2022

Tim Zeitz | 22. Juni 2022



## Zeitabhängige Netzwerke (Basics)

- Funktionen statt Konstanten an Kanten
- Operationen werden teurer
  - $\mathcal{O}(\log |I|)$  für Auswertung
  - $\mathcal{O}(|I^f| + |I^g|)$  für Linken und Minimum
  - Speicherverbrauch explodiert
- Zeitanfragen:
  - Normaler Dijkstra
  - Kaum langsamer (lediglich Auswertung)
- Profilanfragen
  - nicht zu handhaben

		Nodes	Arcs [ $\cdot 10^6$ ]	Avg. $ f $	Rel. Delay [%]		Size
		[ $\cdot 10^6$ ]	(TD [%])	TD arcs	All	TD	[GB]
Ber	Tuesday	0.4	1.0 (27.4)	75.0	3.1	17.6	0.2
	Saturday	0.4	1.0 (20.2)	69.1	2.1	14.8	0.1
Ger06	midweek	4.7	10.8 (7.2)	19.5	1.7	33.1	0.3
	Saturday	4.7	10.8 (3.9)	15.8	0.8	28.5	0.2
SynEur	Low	18.0	42.2 (0.1)	13.2	0.3	125.2	0.8
	Medium	18.0	42.2 (1.0)	13.2	0.8	124.9	0.8
	High	18.0	42.2 (6.2)	13.2	4.6	124.8	1.0
Ger17	Tuesday	7.2	15.8 (29.2)	31.6	3.5	20.8	1.3
Eur17	Tuesday	25.8	55.5 (27.2)	29.5	2.7	19.0	4.2
Eur20	Tuesday	28.5	60.9 (76.3)	22.5	21.0	34.9	8.7

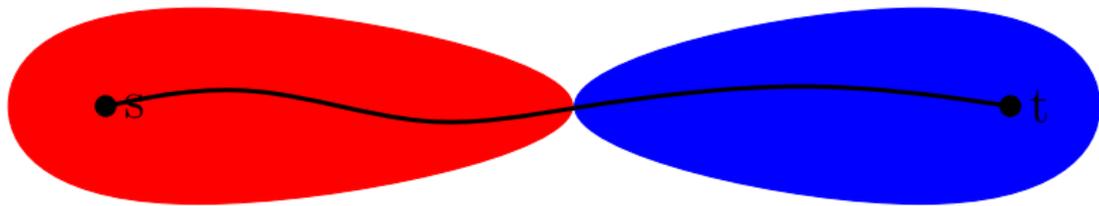
# Beschleunigungstechniken

# Bidirektionaler zeitabhängiger ALT

● S

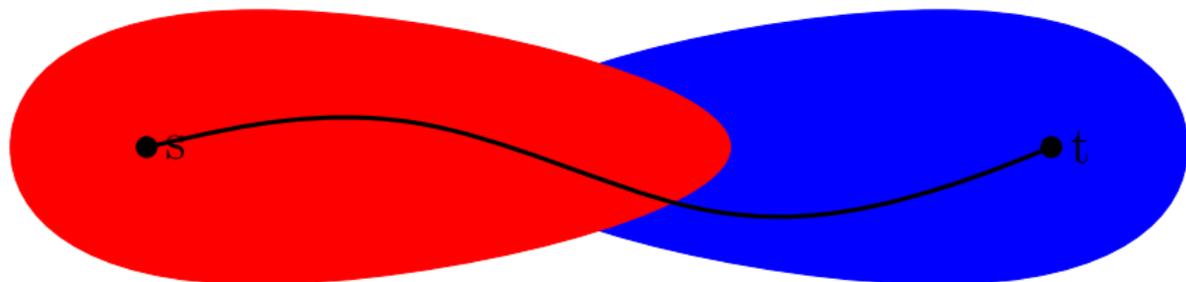
● t

Idee - Drei Phasen:



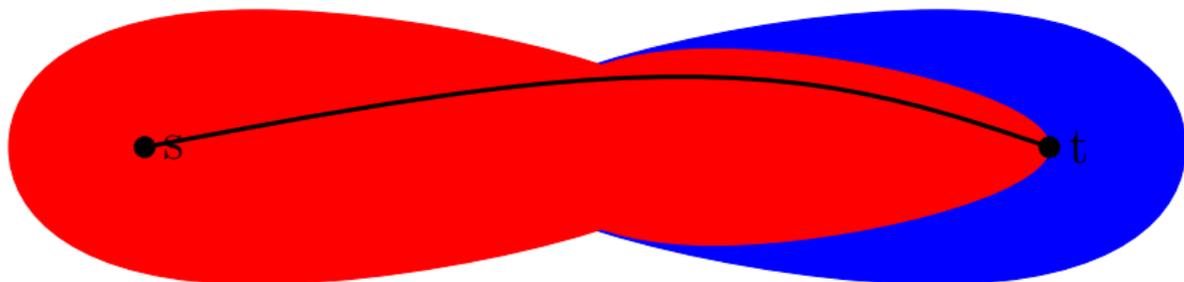
## Idee - Drei Phasen:

- 1 Vorwärts zeitabhängig, Rückwärtssuche benutzt **Minima der Funktionen**.  
Fertig wenn Suchen sich treffen. Berechne zeitabhängige tentative Distanz  $\mu$ 
  - Distanz der Rückwärtssuche: untere Schranke  $\Rightarrow$  nicht geeignet
  - Variante 1: Durch Auswerten des gefundenen Pfades
  - Variante 2: Rückwärtssuche schleift auch Maxima durch



## Idee - Drei Phasen:

- 1 Vorwärts zeitabhängig, Rückwärtssuche benutzt **Minima der Funktionen**.  
Fertig wenn Suchen sich treffen. Berechne zeitabhängige tentative Distanz  $\mu$ 
  - Distanz der Rückwärtssuche: untere Schranke  $\Rightarrow$  nicht geeignet
  - Variante 1: Durch Auswerten des gefundenen Pfades
  - Variante 2: Rückwärtssuche schleift auch Maxima durch
- 2 Rückwärtssuche weiter bis  $\min\text{Key}(\overleftarrow{Q}) > \mu$



## Idee - Drei Phasen:

- 1 Vorwärts zeitabhängig, Rückwärtssuche benutzt **Minima der Funktionen**.  
Fertig wenn Suchen sich treffen. Berechne zeitabhängige tentative Distanz  $\mu$ 
  - Distanz der Rückwärtssuche: untere Schranke  $\Rightarrow$  nicht geeignet
  - Variante 1: Durch Auswerten des gefundenen Pfades
  - Variante 2: Rückwärtssuche schleift auch Maxima durch
- 2 Rückwärtssuche weiter bis  $\min\text{Key}(\overleftarrow{Q}) > \mu$
- 3 Vorwärtssuche arbeitet weiter bis  $t$  abgearbeitet worden ist und besucht nur Knoten, die die Rückwärtssuche zuvor besucht hat

## Beobachtung:

- Phase 2 läuft recht lange 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  $\minKey(\overleftarrow{Q}) \cdot K > \mu$  gilt
- dann ist der berechnete Weg eine  $K$ -Approximation des kürzesten Weges

## Idee

- begrenze Beschleunigungstechnik auf kleinen Subgraphen (**Kern**)

s ●

● t

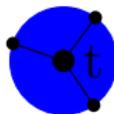
## Vorbereitung

- kontrahiere Graphen zu einem Kern
- Landmarken nur im Kern

## Anfrage

## Idee

- begrenze Beschleunigungstechnik auf kleinen Subgraphen (**Kern**)



## Vorbereitung

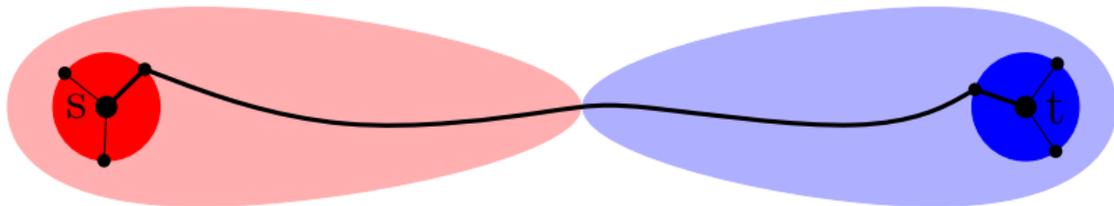
- kontrahiere Graphen zu einem Kern
- Landmarken nur im Kern

## Anfrage

- Initialphase: normaler Dijkstra

## Idee

- begrenze Beschleunigungstechnik auf kleinen Subgraphen (**Kern**)



## Vorbereitung

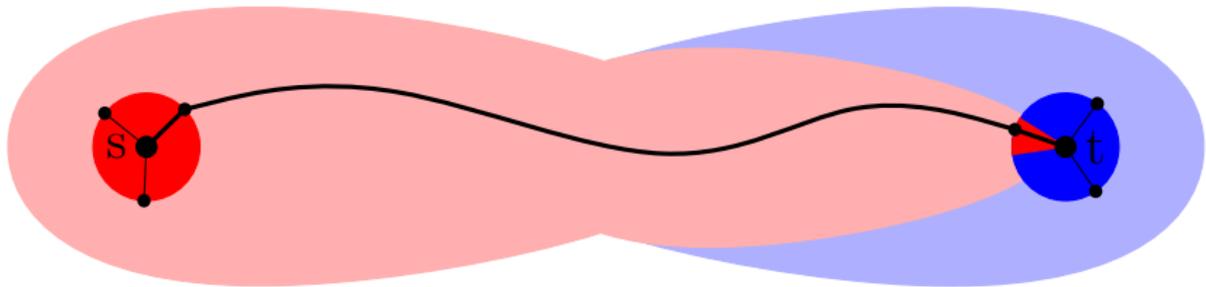
- kontrahiere Graphen zu einem Kern
- Landmarken nur im Kern

## Anfrage

- Initialphase: normaler Dijkstra
- benutze Landmarken nur im Kern

## Idee

- begrenze Beschleunigungstechnik auf kleinen Subgraphen (**Kern**)



## Vorbereitung

- 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

# SHARC (Kontraktion, Arc-Flags)

## Vorbereitung:

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

## Anpassung (grob):

- Kontraktion und Flaggenberechnung anpassen
- Verfeinerung durch (lokale) Profilsuchen



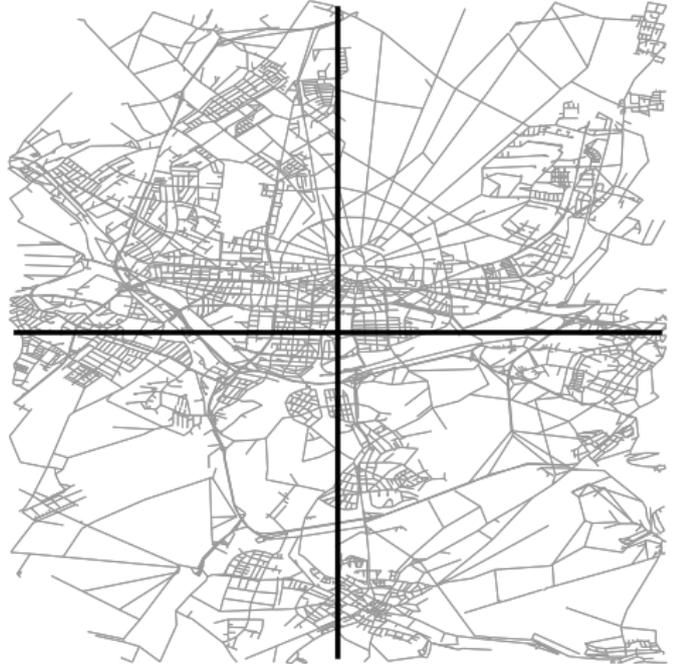
# SHARC (Kontraktion, Arc-Flags)

## Vorbereitung:

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

## Anpassung (grob):

- Kontraktion und Flaggenberechnung anpassen
- Verfeinerung durch (lokale) Profilsuchen



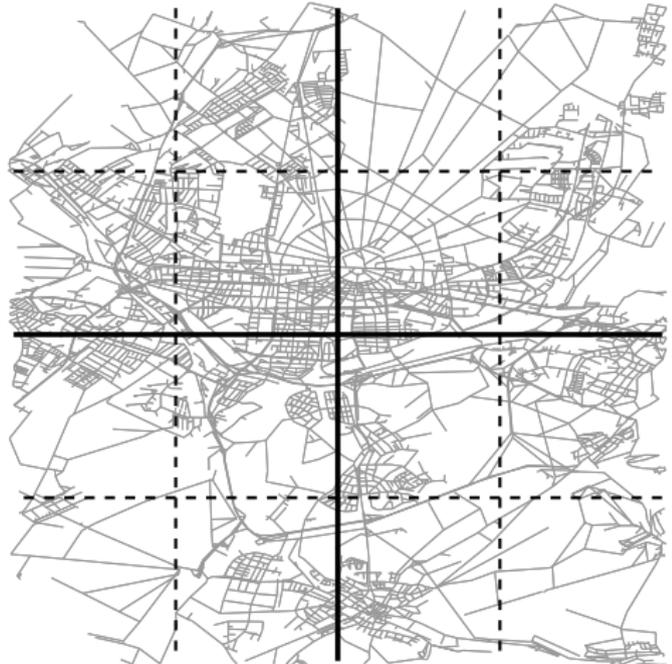
# SHARC (Kontraktion, Arc-Flags)

## Vorbereitung:

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

## Anpassung (grob):

- Kontraktion und Flaggenberechnung anpassen
- Verfeinerung durch (lokale) Profilsuchen



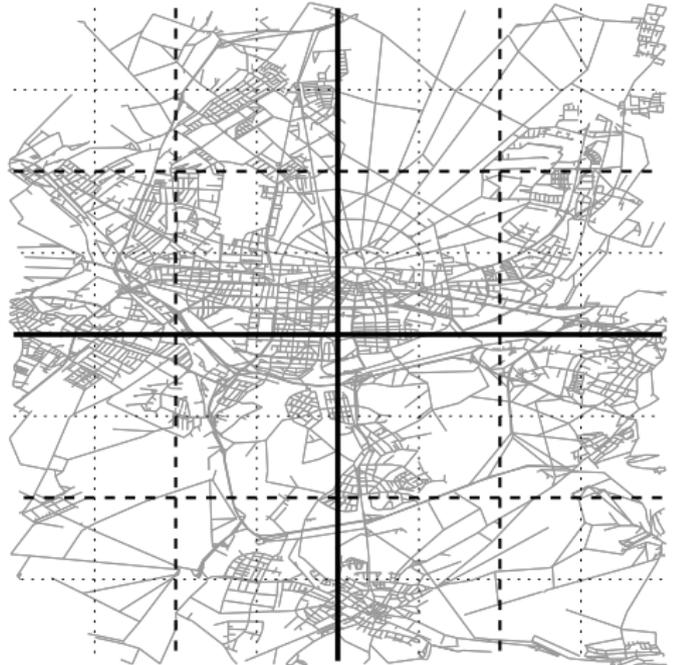
# SHARC (Kontraktion, Arc-Flags)

## Vorbereitung:

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

## Anpassung (grob):

- Kontraktion und Flaggenberechnung anpassen
- Verfeinerung durch (lokale) Profilsuchen



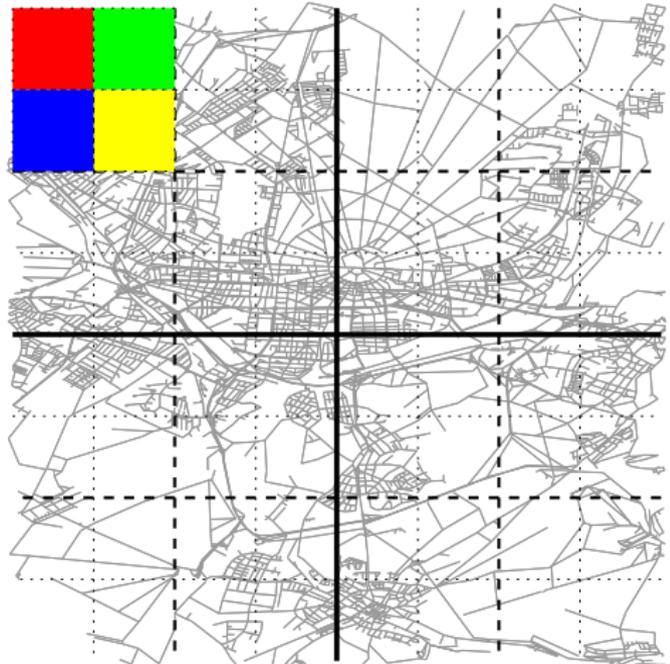
# SHARC (Kontraktion, Arc-Flags)

## Vorbereitung:

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

## Anpassung (grob):

- Kontraktion und Flaggenberechnung anpassen
- Verfeinerung durch (lokale) Profilsuchen



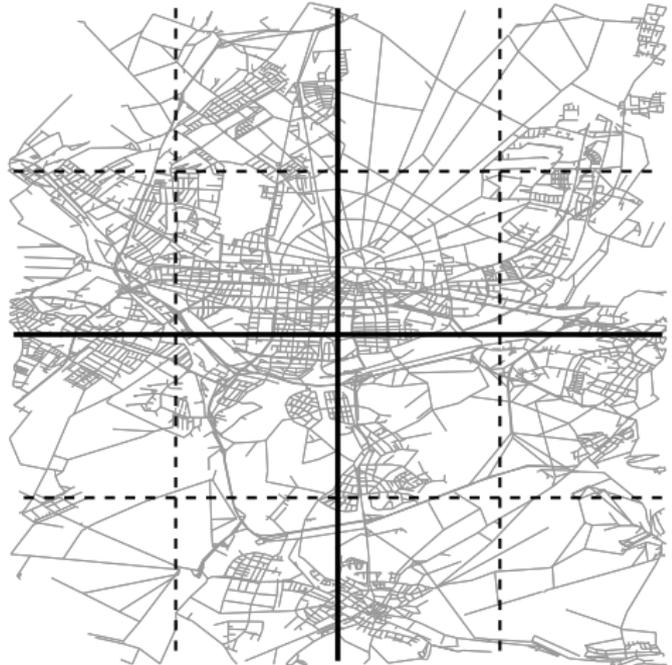
# SHARC (Kontraktion, Arc-Flags)

## Vorbereitung:

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

## Anpassung (grob):

- Kontraktion und Flaggenberechnung anpassen
- Verfeinerung durch (lokale) Profilsuchen



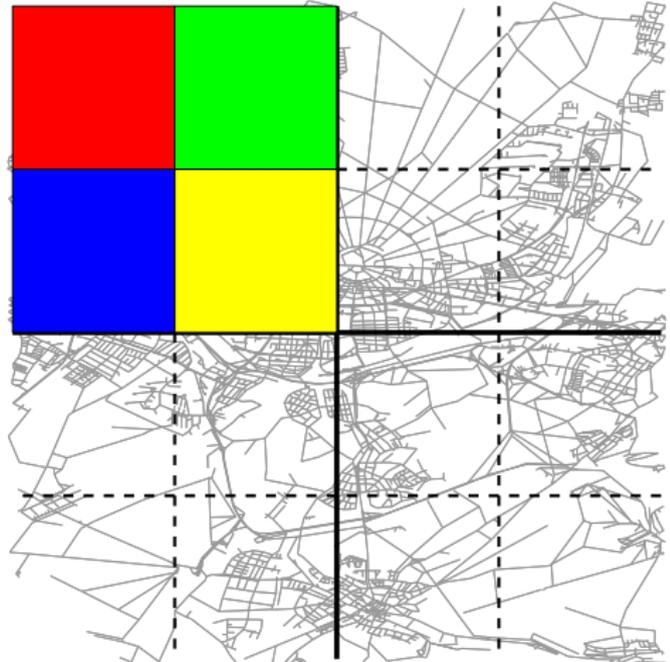
# SHARC (Kontraktion, Arc-Flags)

## Vorbereitung:

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

## Anpassung (grob):

- Kontraktion und Flaggenberechnung anpassen
- Verfeinerung durch (lokale) Profilsuchen



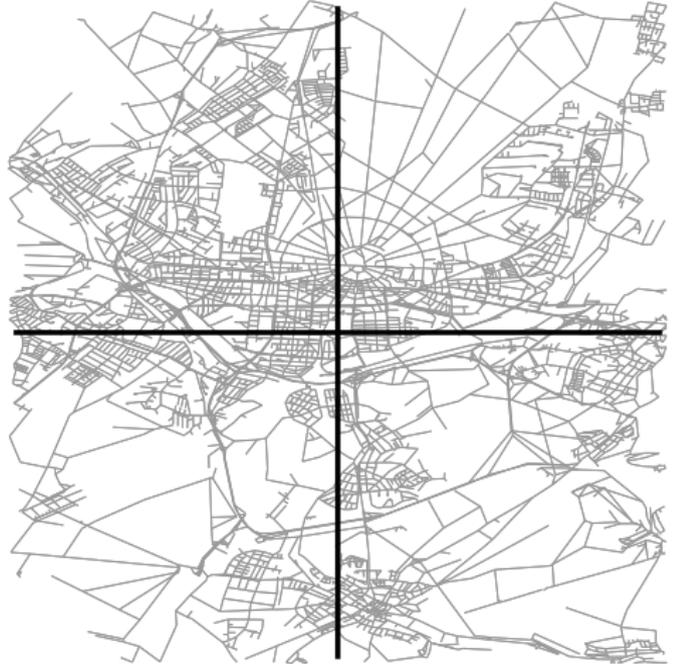
# SHARC (Kontraktion, Arc-Flags)

## Vorbereitung:

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

## Anpassung (grob):

- Kontraktion und Flaggenberechnung anpassen
- Verfeinerung durch (lokale) Profilsuchen



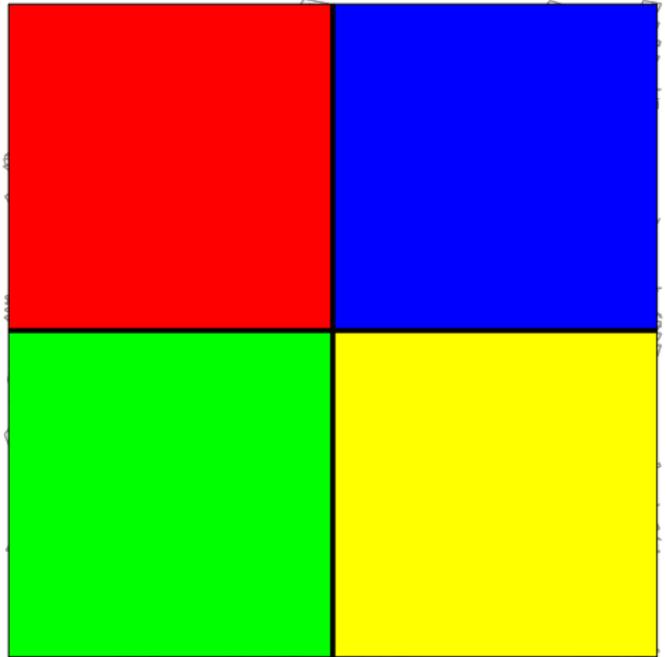
# SHARC (Kontraktion, Arc-Flags)

## Vorbereitung:

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

## Anpassung (grob):

- Kontraktion und Flaggenberechnung anpassen
- Verfeinerung durch (lokale) Profilsuchen



## Vorbereitung:

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

## Vorbereitung:

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

## Anfrage

- Rückwärtssuche schwierig (Ankunftszeit unbekannt)
- Kompletten Rückwärtsaufwärtssuchraum markieren?

## Vorbereitung:

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

## Anfrage

- Rückwärtssuche schwierig (Ankunftszeit unbekannt)
- Kompletten Rückwärtsaufwärtssuchraum markieren?
- Rückwärts aufwärts mittels min-max Suche (Phase 1)
  - Intervallsuche: jeder Knoten bekommt eine untere und obere Reisezeitschranke
  - markiere alle Kanten  $(u, v)$  aus  $\downarrow E$  mit  $\underline{d}(u, v) + \underline{d}(v, t) \leq \overline{d}(u, t)$
  - diese Menge sei  $\downarrow E'$
- zeitabhängige Vorwärtsuche in  $(V, \uparrow E \cup \downarrow E')$  (Phase 2)

# Experimente (TD Germany)

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

## Idee:

- speicher jeden Shortcut als Approximation
- reduziert Speicher um bis zu Faktor 10
- Aber: selbst die Vorwärtssuche liefert jetzt nur noch (Earliest Arrival) Approximationsintervalle!

## Idee:

- speicher jeden Shortcut als Approximation
- reduziert Speicher um bis zu Faktor 10
- Aber: selbst die Vorwärtssuche liefert jetzt nur noch (Earliest Arrival) Approximationsintervalle!

## Query (viele Phasen!):

- Phase 1: Rück-auf: Min/max-Intervall, Vor-auf: Ankunfts-Intervall
- Phase 2: Vor-ab: Ankunftsintervall
- Phase 3: Rück-auf: Reisezeit-Intervall (verschärft Schranken)
- alle Knoten an denen Suchen sich treffen: Kandidaten  $C$
- entpacke alle (approximierten) Shortcuts auf  $s-C-t$  Pfaden
- erzeugt (exakten weil Originalkanten) Subgraphen (Korridor)
- Time-dependent Dijkstra im Korridor (Phase 4)
- nicht viel langsamer (Faktor 2)
- ist **exakt**

## Variante 1:

- normale Profilsuche in der CH
- langsam

## Variante 1:

- normale Profilsuche in der CH
- langsam

## Variante 2:

- normale Profilsuche im Korridor (min/max, Approximation)
- besser, aber es geht noch besser

## Variante 1:

- normale Profilsuche in der CH
- langsam

## Variante 2:

- normale Profilsuche im Korridor (min/max, Approximation)
- besser, aber es geht noch besser

## Variante 3:

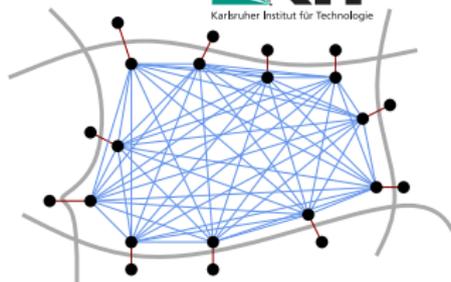
- Kontraktion des Korridors:
  - halte Start- und Zielknoten fest
  - Führe exakte Kontraktion durch (Linken von Kanten, keine Approximation)
  - Priorisiere Knoten mit unkomplexen inzidenten Kanten
- Balancierte Berechnung

⇒ ca. 30 ms

# MLD (CRP)

## Beobachtungen

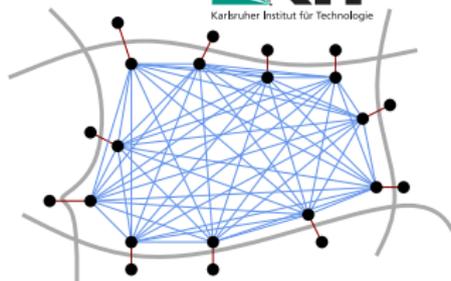
- Partitionierung metrik-unabhängig
- Viele Shortcuts / Overlay-Kanten
- Großer Vorteil von MLD (eigentlich):  
Komplettes Speicherlayout nach Partitionierung bekannt
- Uni-direktionale Anfragen möglich (Partition steuert Suchlevel)



# MLD (CRP)

## Beobachtungen

- Partitionierung metrik-unabhängig
- Viele Shortcuts / Overlay-Kanten
- Großer Vorteil von MLD (eigentlich):  
Komplettes Speicherlayout nach Partitionierung bekannt
- Uni-direktionale Anfragen möglich (Partition steuert Suchlevel)



## TDCRP

- Speicherlayout hängt an Komplexität der Overlay-Kanten-Funktionen; die ist aber vorab unbekannt
- Lohnt sich noch das Speichern der kompletten Cliques?
- Hilft geschickte Approximation?

# Approximation (TDCRP)

## Approximation hilft auch hier (sehr viel)

$\varepsilon$ [%]		Lvl 1	Lvl 2	Lvl 3	Lvl 4	Lvl 5	Lvl 6
0	breakpoints	99 M	397 M	813 M	1 356 M	—	—
	td.arc.cplx.	21	69	188	507	—	—
0.1	breakpoints	65 M	126 M	142 M	121 M	68 M	26 M
	td.arc.cplx.	14	22	33	45	50	47
1.0	breakpoints	51 M	73 M	62 M	41 M	21 M	8 M
	td.arc.cplx.	11	13	14	15	15	14
10.0	breakpoints	28 M	28 M	19 M	12 M	6 M	1 M
	td.arc.cplx.	6	5	5	5	4	2

Approximation nach jedem Level.

## TDCRP robuster gegen Veränderungen in der Eingabe

Network	TCH		TDCRP	
	Pre. [s]	Q. [ms]	Cust. [s]	Q. [ms]
Europe	1 479	1.37	109	5.75
Europe, bad traffic	7 772	5.87	208	8.01
Europe, avoid highways	8 956	19.54	127	8.29

## Problem:

- hoher Speicherbedarf der Shortcuts

## Ideen:

- Shortcuts nur approximieren, **inexakte Anfragen**
- Keine Gewichte am Shortcut speichern, stattdessen on-the-fly entpacken und Pfad linken: **spart Speicher, kostet Laufzeit**
- speichere auf Shortcuts Über- und Unterapproximation der Funktionen
  - induzieren wieder Korridor (aber genaueren als nur Min/Max!)
  - entpacke Shortcuts im Korridor, dies gibt einen Teil des Originalgraphen
  - benutze nun die nicht-approximierten Originalkanten für eine **exakte Suche**

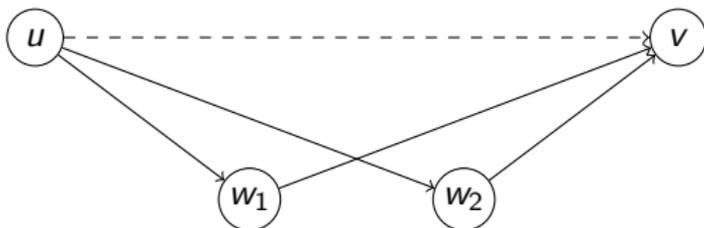
## Problem:

- hoher Speicherbedarf der Shortcuts

## Ideen:

- Keine Gewichte am Shortcut speichern, stattdessen on-the-fly entpacken und Pfad linken: **spart Speicher, kostet Laufzeit**

- Beobachtung: Kürzeste Reisezeit ändert sich häufiger als Pfad
- Shortcuts speichern
  - zeitabhängige kürzeste Pfade - **effiziente Entpackung**
  - zeitunabhängige Bounds - **Pruning**
- CATCHUP = CCH + Unpacking Shortcuts + viel Engineering



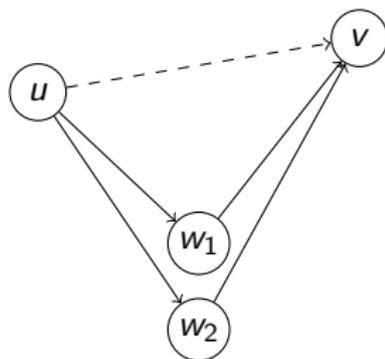
00:00	$uw_1$	$w_1v$
07:32	$uw_2$	$w_2v$
15:42	$uw_1$	$w_1v$

## Idee

- CCH Customization + Optimierungen
- Reisezeitfunktionen temporär speichern

## Pruning mit Bounds

- Precustomization
- Dreiecke sortieren

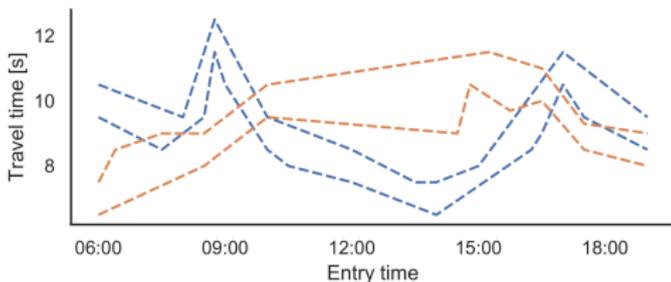


## Idee

- CCH Customization + Optimierungen
- Reisezeitfunktionen temporär speichern

## Pruning mit Bounds

- Precustomization
- Dreiecke sortieren



## Approximation

- Hüllkurven nutzen wenn Funktionen zu komplex
- Lazy exakte Profile rekonstruieren für exaktes Mergen

## Idee

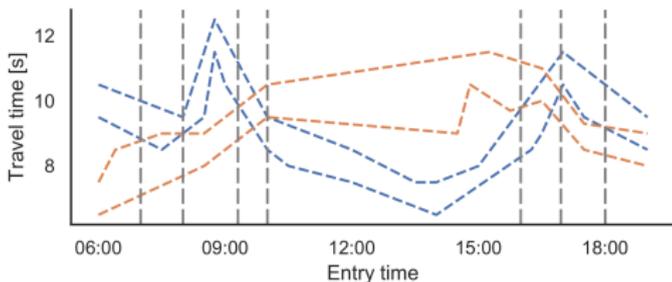
- CCH Customization + Optimierungen
- Reisezeitfunktionen temporär speichern

## Pruning mit Bounds

- Precustomization
- Dreiecke sortieren

## Approximation

- Hüllkurven nutzen wenn Funktionen zu komplex
- Lazy exakte Profile rekonstruieren für exaktes Mergen



## Idee

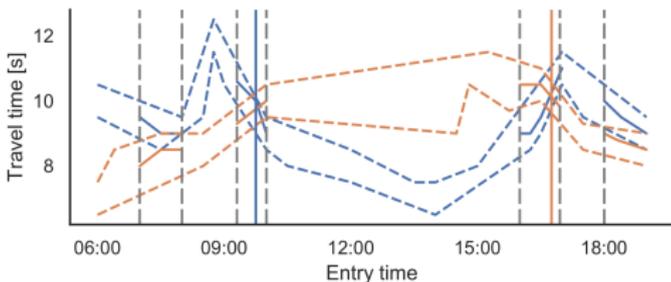
- CCH Customization + Optimierungen
- Reisezeitfunktionen temporär speichern

## Pruning mit Bounds

- Precustomization
- Dreiecke sortieren

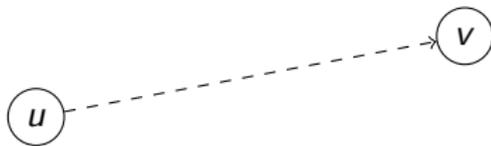
## Approximation

- Hüllkurven nutzen wenn Funktionen zu komplex
- Lazy exakte Profile rekonstruieren für exaktes Mergen



## 2 Phasen

- Elimination Tree Interval Query
- Lazy Corridor Dijkstra



## 2 Phasen

- Elimination Tree Interval Query
- Lazy Corridor Dijkstra



## Lazy Auspacken/Relaxieren

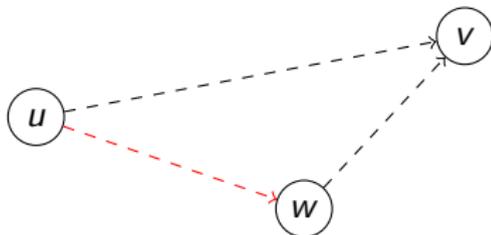
- Shortcuts lazy auspacken
- Erste Kante auf dem Shortcut relaxieren
- Rest zum Korridor hinzufügen

## 2 Phasen

- Elimination Tree Interval Query
- Lazy Corridor Dijkstra

## Lazy Auspacken/Relaxieren

- Shortcuts lazy auspacken
- Erste Kante auf dem Shortcut relaxieren
- Rest zum Korridor hinzufügen

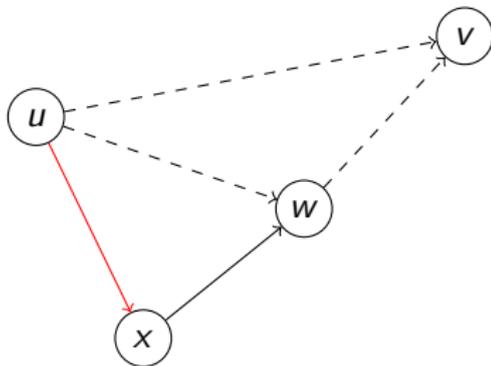


## 2 Phasen

- Elimination Tree Interval Query
- Lazy Corridor Dijkstra

## Lazy Auspacken/Relaxieren

- Shortcuts lazy auspacken
- Erste Kante auf dem Shortcut relaxieren
- Rest zum Korridor hinzufügen

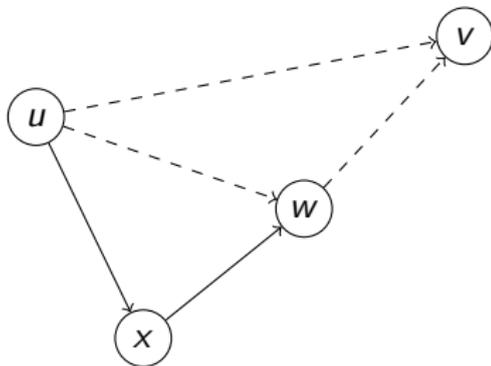


## 2 Phasen

- Elimination Tree Interval Query
- Lazy Corridor Dijkstra

## Lazy Auspacken/Relaxieren

- Shortcuts lazy auspacken
- Erste Kante auf dem Shortcut relaxieren
- Rest zum Korridor hinzufügen



## Zielgerichtet

- Lower Bounds aus Intervallquery als  $A^*$  Potentiale
- Beim auspacken Bounds propagieren

# CATCHUp: Preprocessing Resultate

	CCH arcs [ $\cdot 10^3$ ]	Expansions per arc			Index [GB]	Running time [s]	
		Avg.	Max.	= 1 [%]		Phase 1	Phase 2
Ber	1 977	1.039	31	98.6	0.09	1.5	6.2
Ger06	22 519	1.075	44	98.4	1.06	30.1	21.6
Ger17	31 488	1.090	107	98.5	1.50	35.0	107.4
Eur17	114 857	1.099	115	98.4	5.47	189.6	557.0
Eur20	128 921	1.191	109	96.9	6.32	209.6	1 039.5

# Einfacher?

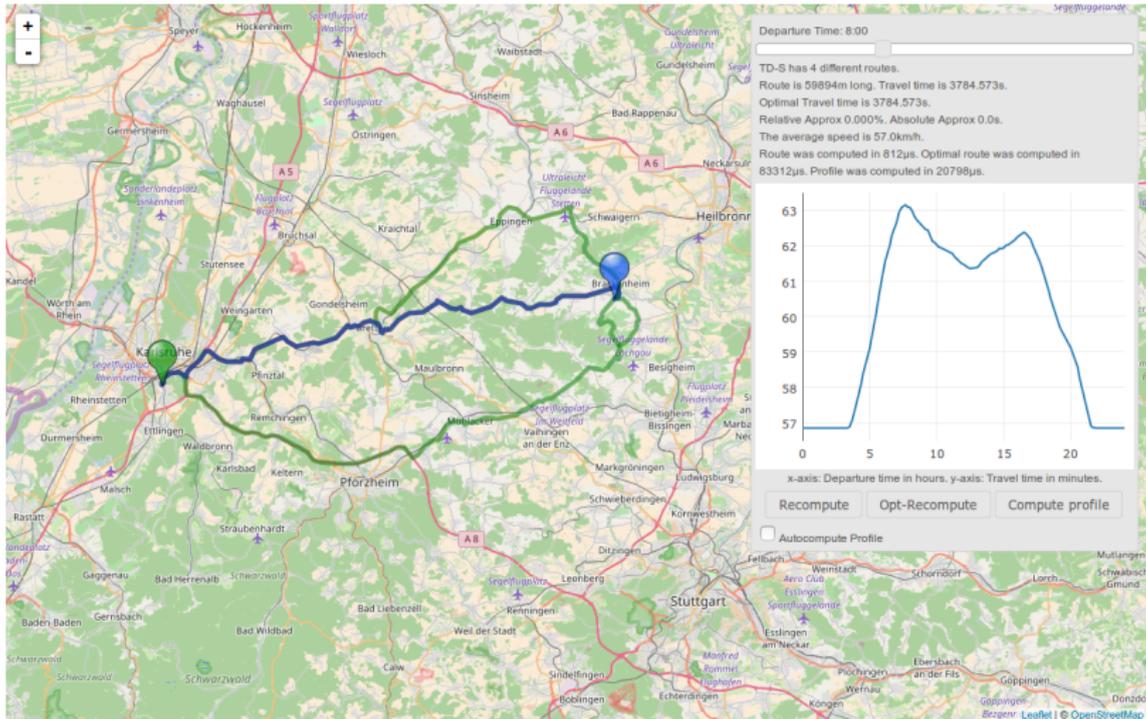
- Bis hierhin: Viele komplizierte Algorithmen
- Geht das nicht einfacher?

- Bis hierhin: Viele komplizierte Algorithmen
- Geht das nicht einfacher?
- Ein sinnvoller Grundalgorithmus:
  - Berechne eine CH auf dem untere-Schranken-Graph
  - Berechne zeitunabhängigen Pfad
  - Rechne zeitabhängige Fahrzeit entlang des Pfads nach
- heißt Freeflow

Etwas ausgefeilter:

- Tag in charakteristische Abschnitte aufteilen  
*Rush-hour Morgens, Mittags, Rush-hour Nachmittags, Nachts, (Live)*
- Zeit-**unabhängiger** Graph für jeden Abschnitt
  - Gewichte: Durchschnittlich travel time pro Abschnitt
- Eine (C)CH pro Abschnitt
- Zeit-**unabhängiger** kürzester Pfad pro Abschnitt
- Pfade zu Subgraph vereinigen
- **Earliest arrival**: TD-Dijkstra in Subgraph
- **Profil**: Alle 10 min. TD-Dijkstra's algo. in Subgraph

# Time-Dependent Sampling



- Vorbereitung: CH auf lower-bound-Graph
- Query: A\* mit exakten lower-bound-Distanzen als Potential
  - Lazy RPHAST für Many-to-one Distanzen

# Performance EA Queries: Ger06

	Prepro.	Custom.	Space	Query	Rel. error [%]	
	Cores × [s]	Cores × [s]	[GB]	[ms]	Avg.	Max.
TD-Dijkstra	-	-	-	525.48	-	-
TDCALT	540	-	0.23	5.36	-	-
TDCALT-K1.15	540	-	0.23	1.87	0.050	13.840
eco L-SHARC	4 680	-	1.03	6.31	-	-
heu SHARC	12 360	-	0.64	0.69	n/r	0.610
KaTCH	16 × 170	-	4.66	0.63	-	-
TCH	8 × 378	8 × 74	4.66	0.75	-	-
ATCH (1.0)	8 × 378	8 × 74	1.12	1.24	-	-
ATCH ( $\infty$ )	8 × 378	8 × 74	0.55	1.66	-	-
inex. TCH (0.1)	8 × 378	8 × 74	1.34	0.70	0.020	0.100
inex. TCH (1.0)	8 × 378	8 × 74	1.00	0.69	0.270	1.010
TD-CRP (0.1)	16 × 273	16 × 16	0.78	1.92	0.050	0.250
TD-CRP (1.0)	16 × 273	16 × 8	0.36	1.66	0.680	2.850
TD-S+9	547	-	3.61	1.67	0.001	1.523
CATCHUp	16 × 31	16 × 18	1.06	0.70	-	-
CH-Potentials	59	-	0.19	4.50	-	-

	Run T. [ms]
eco SHARC	47 388
heu SHARC	847
TCH	95.5
ATCH (2.5)	38.7
TD-S+P4	19.5
TD-S+P9	22.2
CATCHUp	83.6

# Performance EA Queries: Ger17

	Prepro.	Custom.	Space	Query	Rel. error [%]	
	Cores × [s]	Cores × [s]	[GB]	[ms]	Avg.	Max.
TD-Dijkstra	-	-	-	869.79	-	-
KaTCH	16 × 874	-	42.81	1.38	-	-
TD-S+9	617	-	5.28	2.28	0.001	0.963
CATCHUp	16 × 35	16 × 92	1.50	1.87	-	-

# Performance EA Queries: Eur17

	Prepro.	Custom.	Space	Query	Rel. error [%]	
	Cores × [s]	Cores × [s]	[GB]	[ms]	Avg.	Max.
TD-Dijkstra	-	-	-	2 581.16	-	-
KaTCH	16 × 3 089	-	146.97	OOM	-	-
TD-S+9	3 368	-	18.84	4.03	0.002	1.159
CATCHUp	16 × 196	16 × 479	5.48	4.50	-	-
CH-Potentials	293	-	1.10	89.70	-	-

# Montag, 27. Juni 2022

- Daniel Delling:  
**Engineering and Augmenting Route Planning Algorithms**  
Ph.D. Thesis, Universität Karlsruhe (TH), 2009.
- Gernot Veit Batz, Robert Geisberger, Peter Sanders, Christian Vetter:  
**Minimum Time-Dependent Travel Times with Contraction Hierarchies**  
In: *Journal of Experimental Algorithmics*, 2013.
- Moritz Baum, Julian Dibbelt, Thomas Pajor, Dorothea Wagner:  
**Dynamic Time-Dependent Route Planning in Road Networks with User Preferences**  
In: *Proceedings of the 9th International Symposium on Experimental Algorithms (SEA'16)*, 2016.

- Ben Strasser: **Dynamic Time-Dependent Routing in Road Networks Through Sampling**  
In: *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'17)*, 2017.
- Ben Strasser, Dorothea Wagner, Tim Zeitz: **Space-efficient, Fast and Exact Routing in Time-dependent Road Networks**  
In: *Proceedings of the 28th Annual European Symposium on Algorithms (ESA'20)*
- Ben Strasser, Tim Zeitz: **A Fast and Tight Heuristic for A\* in Road Networks**  
In: *Proceedings of the 19th International Symposium on Experimental Algorithms (SEA'21)*