Algorithmen für Routenplanung

6. Vorlesung, Sommersemester 2010

Reinhard Bauer | 09. Mai 2011

INSTITUT FÜR THEORETISCHE INFORMATIK · ALGORITHMIK I · PROF. DR. DOROTHEA WAGNER

Multilevel-Overlay Graphs

Gegeben

- Eingabegraph $G = (V, E, \text{len})$
- Folge $V := V_0 \supseteq V_1 \supseteq \ldots \supseteq V_L$ von Teilmengen von V.

Berechne

- Folge $G_0 = (V_0, E_0, \text{len}_0), \ldots, G_L = (V_L, E_L, \text{len}_L)$ von Graphen, so dass Distanzen in G_i wie in G_0

Notion

- Der **level** eines Knoten v ist das größte i, so dass $v \in V_i$

Pseudocode Vorberechnung

Algorithm 1: Preprocessing-Phase of the MOL-Technique

input : graph $G = (V, E, \text{len})$, number of levels $L + 1$, sequence $V := V_0 \supseteq V_1 \supseteq \ldots \supseteq V_L$

output: multilevel overlay graph G', *level* : $V \to Z_{\geq 0}$

$G_0 \leftarrow G$

for $l = 1, \ldots, L$ **do**

- $E_l \leftarrow \{(s,t) \in V_l \times V_l \mid \forall$shortest s-t-paths $(s, u_1, \ldots, u_k, t)$ in G_{l-1} it is $u_1, \ldots, u_k \notin V_l\}$
- $\text{len}_l \leftarrow$ function from E_l to $\mathbb{R}_{\geq 0}$ such that $\text{len}_l(u,v) = \mathrm{dist}_G(u,v)$
- $G_l \leftarrow (V_i, E_i, \text{len}_i)$

$G' \leftarrow G[E_1 \cup \ldots \cup E_L]$

for $v \in V$ **do**

- $\mathit{level}(v) \leftarrow \max\{i \mid 0 \leq i \leq L \text{ and } v \in V_i\}$

Anfrage

- Anfrage: Bidirektionaler Dijkstra
- Ausgehend von einem Knoten u, relaxiere nur Kanten (u, v), so dass $\mathit{level}(u) \leq \mathit{level}(v)$

Pseudocode - Anfrage - Eine Richtung

Algorithm 2: Unidirectional Pruned Search of a MOL-Query

input : graph $G' = (V, E \cup E', \text{len})$, node $x \in V$, $\mathit{level} : V \to Z_{\geq 0}$

output: distance label $d()$

for $v \in V$ **do** $d(v) \leftarrow \infty$ `/* Initialization Phase */`

$d(x) \leftarrow 0$; $Q.\textsc{Insert}(x, 0)$

while not $Q.\textsc{IsEmpty}$ **do**   `/* Main Phase */`

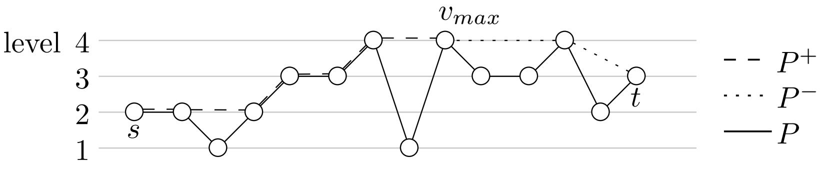
    $v \leftarrow Q.\textsc{ExtractMin}$ **for** $(v, w) \in E \cup E'$ *with* $\mathit{level}(w) \geq \mathit{level}(v)$ **do**

        if $d(v) + \mathit{len}(v, w) < d(w)$ **then**

            $d(w) \leftarrow d(v) + \mathit{len}(v, w)$

            $Q.\textsc{InsertOrUpdate}(w, d(w))$

Korrektheit



- P: Kürzester Weg in Originalgraph
- P^+: Weg von Vorwärtssuche
- P^-: Weg von Rückwärtssuche

Geschichte

- [Schulz, Wagner, Zaroliagis 02]
  Technik eingeführt
- [Sanders, Schultes 07]
  Variante als Highway Node Routing entwickelt
- [Delling Goldberg, Pajor, Werneck 11]
  Re-Implementierung und Tuning der ursprünglichen Technik

Wahl der Knoten

- Variante 1: Möglichst „zentrale“ Knoten
- Variante 2: Gute Separatoren

Verschiedene Metriken / Dynamische Anpassung

Szenario:

- Topologie des Netzwerkes vorher bekannt
- Metrik kann sich ändern (verschiedene Fahrprofile, Staus, ...)

Strategie

- Vor-Vorberechnung: Finde Gute Separatoren im Netzwerk
- Wichtig: Wenige Randknoten!
- Separatoren bleiben immer gleich
- Wenn sich die Metrik ändert, berechne den Overlay-Graphen neu
- Dies geht schnell, wenn es hinreichend wenige Randknoten und kleine Zellen gibt

Experimente

[cell sizes]	Vorberechnung time [s]	Vorberechnung space [MB]	Anfrage vertex scans	Anfrage time [ms]
$[2^{14}]$	4.9	10.1	45420	5.81
$[2^{12} : 2^{18}]$	5.0	18.8	12683	1.82
$[2^{10} : 2^{15} : 2^{20}]$	5.2	32.7	6099	0.91
$[2^{8} : 2^{12} : 2^{16} : 2^{20}]$	4.7	59.5	3828	0.72

Literatur

Literatur (Multilevel Overlay Graph):

- Daniel Delling, Andrew V. Goldberg, Thomas Pajor, Renato Werneck
  Customizable Route Planning
  In: *Proceedings of the 10th International Symposium on Experimental Algorithms (SEA '11)*, 2011

- Peter Sanders, Dominik Schultes
  Dynamic Highway Node Routing
  In: *Proceedings of the 6th Workshop on Experimental Algorithms (WEA '07)*, 2007

Contraction Hierarchies

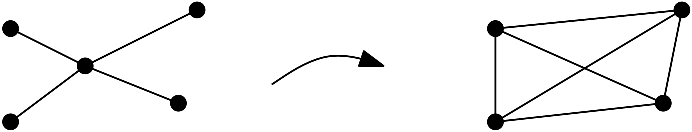
Idee:

- Multilevel-Ansatz mit sovielen levels wie Knoten

Contraction Hierarchies

How to contract vertex v

delete v
for *each pair* u, w *of neighbours of* v **do**
  use a limited local search heuristic to find a witness that (u, v, w) is *not* the only shortest path
  if (u, v, w) *'might be' the only shortest* u*-*w*-path* **then**
    insert edge (u, w) with $\text{len}(u, w) = \text{len}(u, v) + \text{len}(v, w)$



Contraction Hierarchies

The CH-Preprocessing

- edgeDiff(u) gives the difference of edges in the graph after contraction u
- betweenness(u) measures the centrality of u
- estCostOfQuery(u) estimates the cost of the query when now contracting u
- #delNeighbours(u) count the number of already removed neighbours of u

---

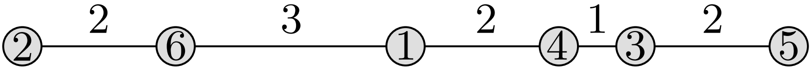
while *there are vertices* **do**
    choose a vertex v with minimal α edgeDiff(u) $+$ β betweenness(u) $+$ γ estCostOfQuery(u) $+$ δ #delNeighbours(u)
    contract v
output: set E^{+} of edges inserted during contraction
output: contraction order

Wiederholung: CH

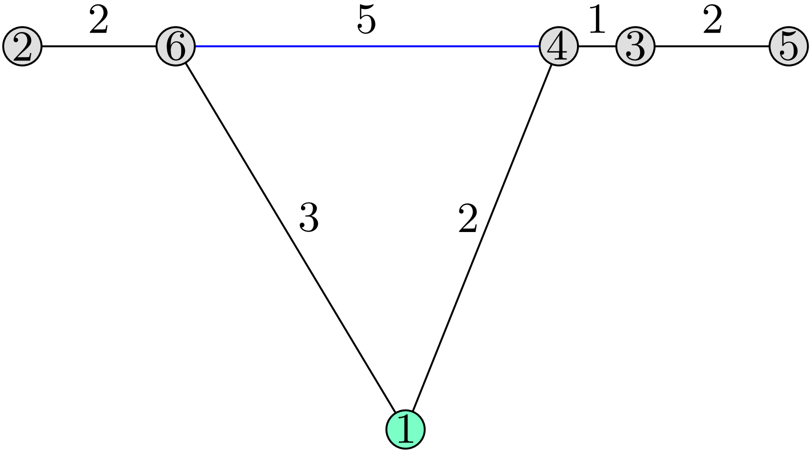
The CH-Query

- split search graph $G = (V, E)$ into
  - $G_\uparrow = (V, E_\uparrow)$ with $E_\uparrow = \{(u, v) \in E | v \text{ contracted after } u\}$
  - $G_\downarrow = (V, E_\downarrow)$ with $E_\downarrow = \{(u, v) \in E | v \text{ contracted before } u\}$
- Bidirectional Search
  - forward search on $G_\uparrow$
  - backward search on $G_\downarrow$
- Distance Balanced
- use stall on demand to additionally prune the search space
- stop search in one direction, when smallest element in queue is at least the length of the shortest path found so far

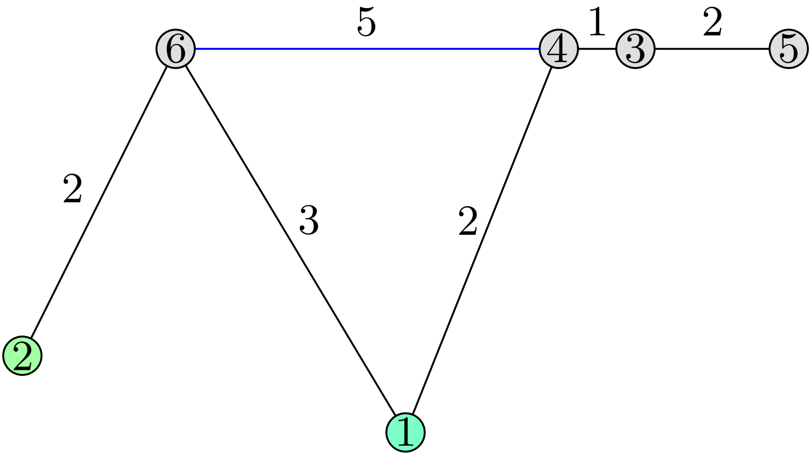
Beispiel: Konstruktion



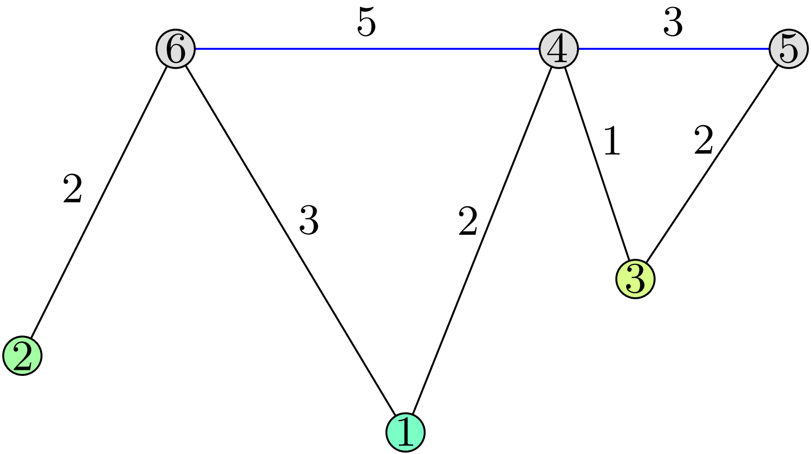
Beispiel: Konstruktion



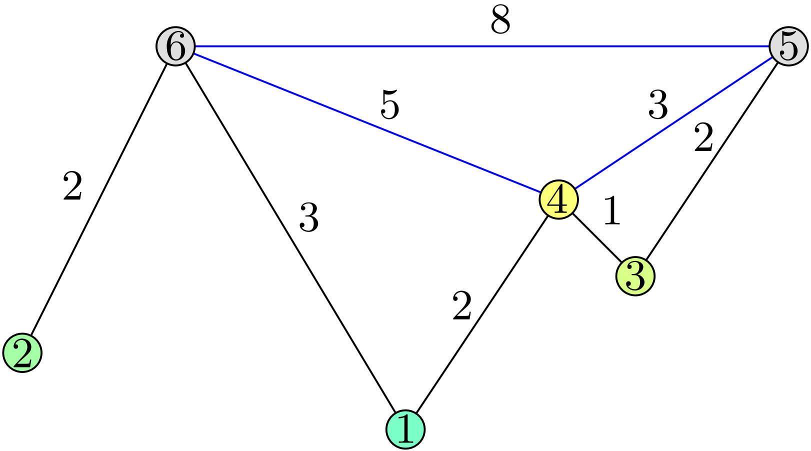
Beispiel: Konstruktion



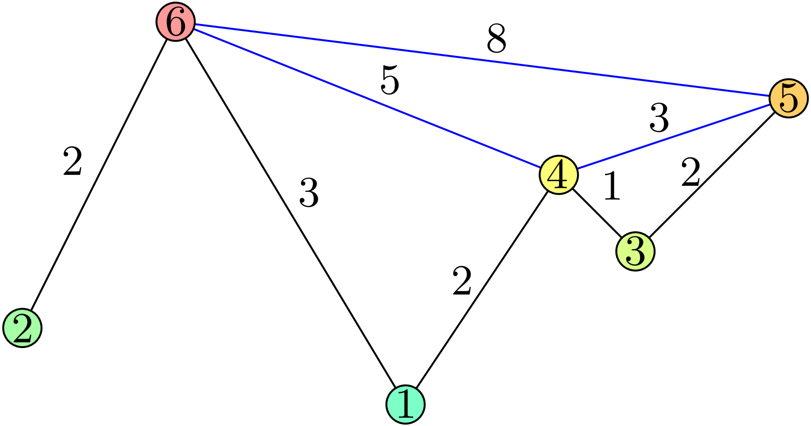
Beispiel: Konstruktion



Beispiel: Konstruktion



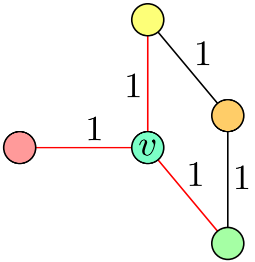
Beispiel: Konstruktion



Konstruktion

wie identifiziert man nötige Shortcuts?

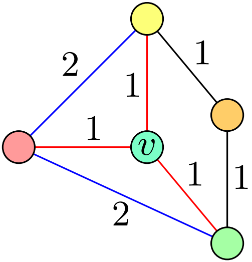
- lokale Suchen von allen Knoten u der eingehenden Kanten (u, v)
- ignoriere Knoten v während der Suche
- füge shortcut (u, w) ein wenn $d(u, w) > w(u, v) + w(v, w)$



Konstruktion

wie identifiziert man nötige Shortcuts?

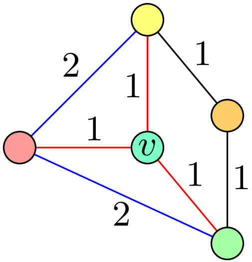
- lokale Suchen von allen Knoten u der eingehenden Kanten (u, v)
- ignoriere Knoten v während der Suche
- füge shortcut (u, w) ein wenn $d(u, w) > w(u, v) + w(v, w)$



Konstruktion

wie identifiziert man nötige Shortcuts?

- lokale Suchen von allen Knoten u der eingehenden Kanten (u, v)
- ignoriere Knoten v während der Suche
- füge shortcut (u, w) ein wenn $d(u, w) > w(u, v) + w(v, w)$

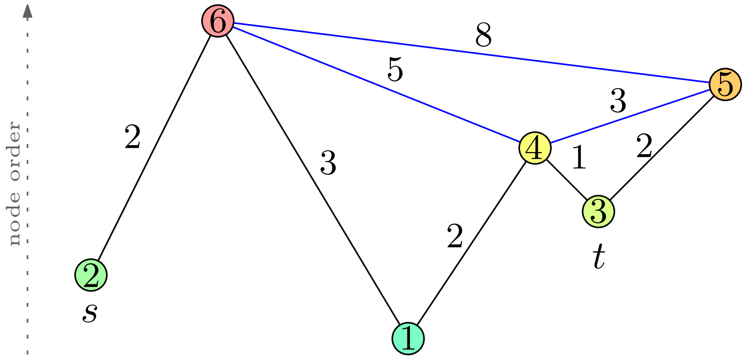


Optimierungen:

- limitiere Suchräume der lokalen Suchen
- limitiere hop-zahl der Suchen
- Spezialfälle: 1-hop-Suche, 2-hop-Suche

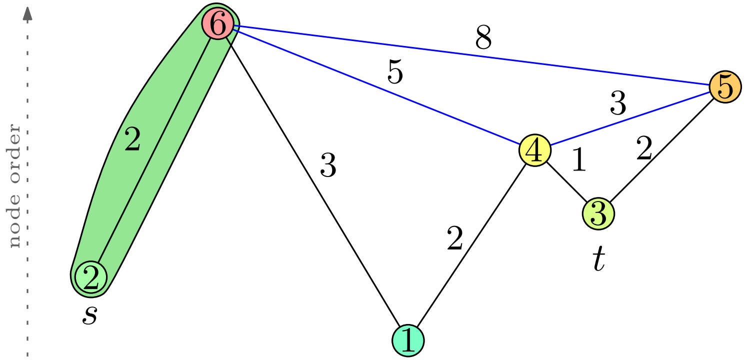
Anfragen

- modifizierter bidirektionaler Dijkstra algorithmus
- upward graph $G_\uparrow := (V, E_\uparrow)$ with $E_\uparrow := \{(u, v) \in E : u < v\}$
  downward graph $G_\downarrow := (V, E_\downarrow)$ with $E_\downarrow := \{(u, v) \in E : u > v\}$
- Vorwärts-Suche in $G_\uparrow$ and Rückwärtssuche in $G_\downarrow$



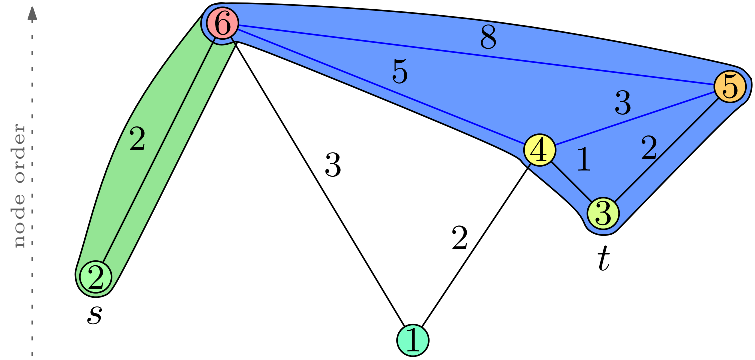
Anfragen

- modifizierter bidirektionaler Dijkstra algorithmus
- upward graph $G_\uparrow := (V, E_\uparrow)$ with $E_\uparrow := \{(u, v) \in E : u < v\}$
  downward graph $G_\downarrow := (V, E_\downarrow)$ with $E_\downarrow := \{(u, v) \in E : u > v\}$
- Vorwärts-Suche in $G_\uparrow$ and Rückwärtssuche in $G_\downarrow$



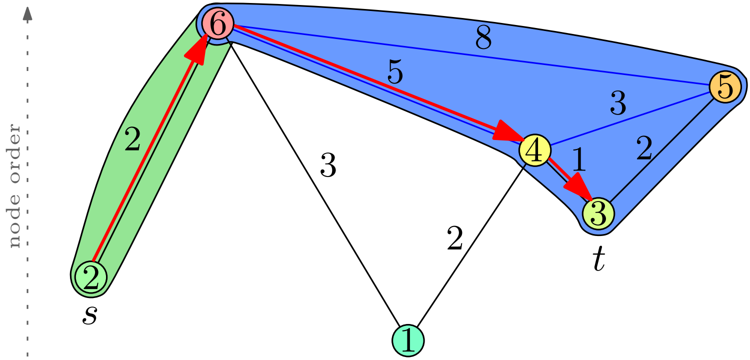
Anfragen

- modifizierter bidirektionaler Dijkstra algorithmus
- upward graph $G_\uparrow := (V, E_\uparrow)$ with $E_\uparrow := \{(u,v) \in E : u < v\}$
  downward graph $G_\downarrow := (V, E_\downarrow)$ with $E_\downarrow := \{(u,v) \in E : u > v\}$
- Vorwärts-Suche in $G_\uparrow$ and Rückwärtssuche in $G_\downarrow$



Anfragen

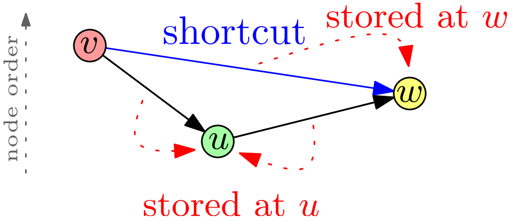
- modifizierter bidirektionaler Dijkstra algorithmus
- upward graph $G_\uparrow := (V, E_\uparrow)$ with $E_\uparrow := \{(u,v) \in E : u < v\}$
  downward graph $G_\downarrow := (V, E_\downarrow)$ with $E_\downarrow := \{(u,v) \in E : u > v\}$
- Vorwärts-Suche in $G_\uparrow$ and Rückwärtssuche in $G_\downarrow$



Datenstruktur

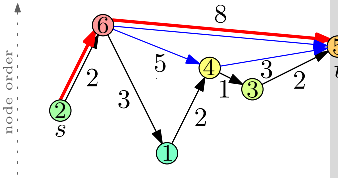
Suchgraph:

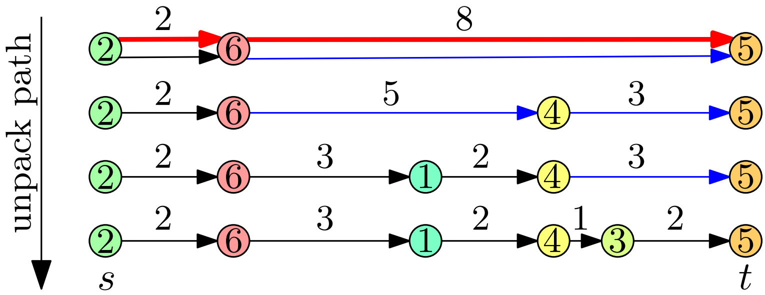
- normalerweise: speichere Kanten (v, w) in den Adjazenz-Arrays von v und w
- für die Suche reicht es aus, die Kante nur an den Knoten $\min\{v, w\}$ zu speichern



Ausgabe der Pfade

- für jeden Shortcut (u, w) eines Pfades (u, v, w), speichere Mittelknoten v an der Kante
- expandiere Pfade mittels Rekursion





Experimente

Eingaben:

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

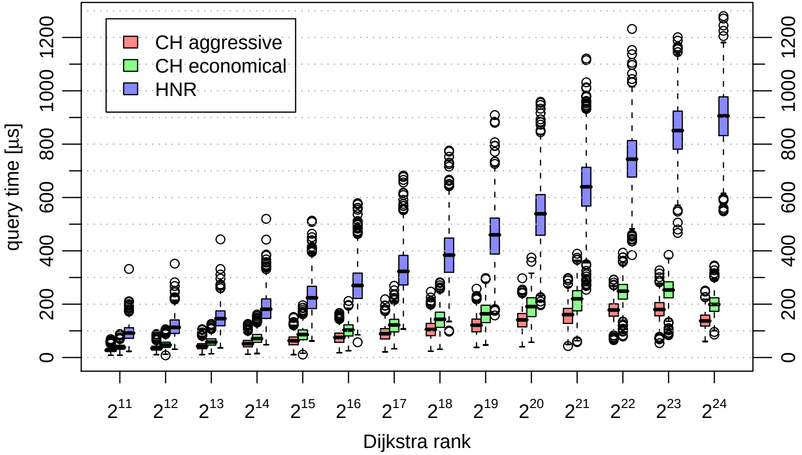
Evaluation:

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

Übersicht: bisherige Techniken

	Vorberechnung		Anfrage		
	Zeit [h:m]	Platz [byte/n]	Such raum	Zeit [ms]	Beschl.
Dijkstra	0:00	0	9 114 385	5 591.6	1
Bi-Dijkstra	0:00	0	4 764 110	2 713.2	2
Uni-ALT-16	1:25	128	815 639	327.6	17.1
Uni-ALT-64	1:08	512	604 968	288.5	19.4
ALT-16	1:25	128	74 669	53.6	104
ALT-64	1:08	512	25 324	19.6	285
Uni Arc-Flags (128)	8:34	20	92 545	31.9	175
Arc-Flags (128)	17:08	10	2 764	0.8	6 988
eco CH	0:10	0.6	459	0.22	25 413
agg CH	0:32	-3.0	359	0.15	37 273

Dijkstra Rank



Ende

Literatur (Contraction Hierarchies):

- Daniel Delling, Robert Geisberger, Peter Sanders, Dominik Schultes:
  Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks
  In: *Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08), volume 5038 of Lecture Notes in Computer Science, pages 319-333. Springer, June 2008.*