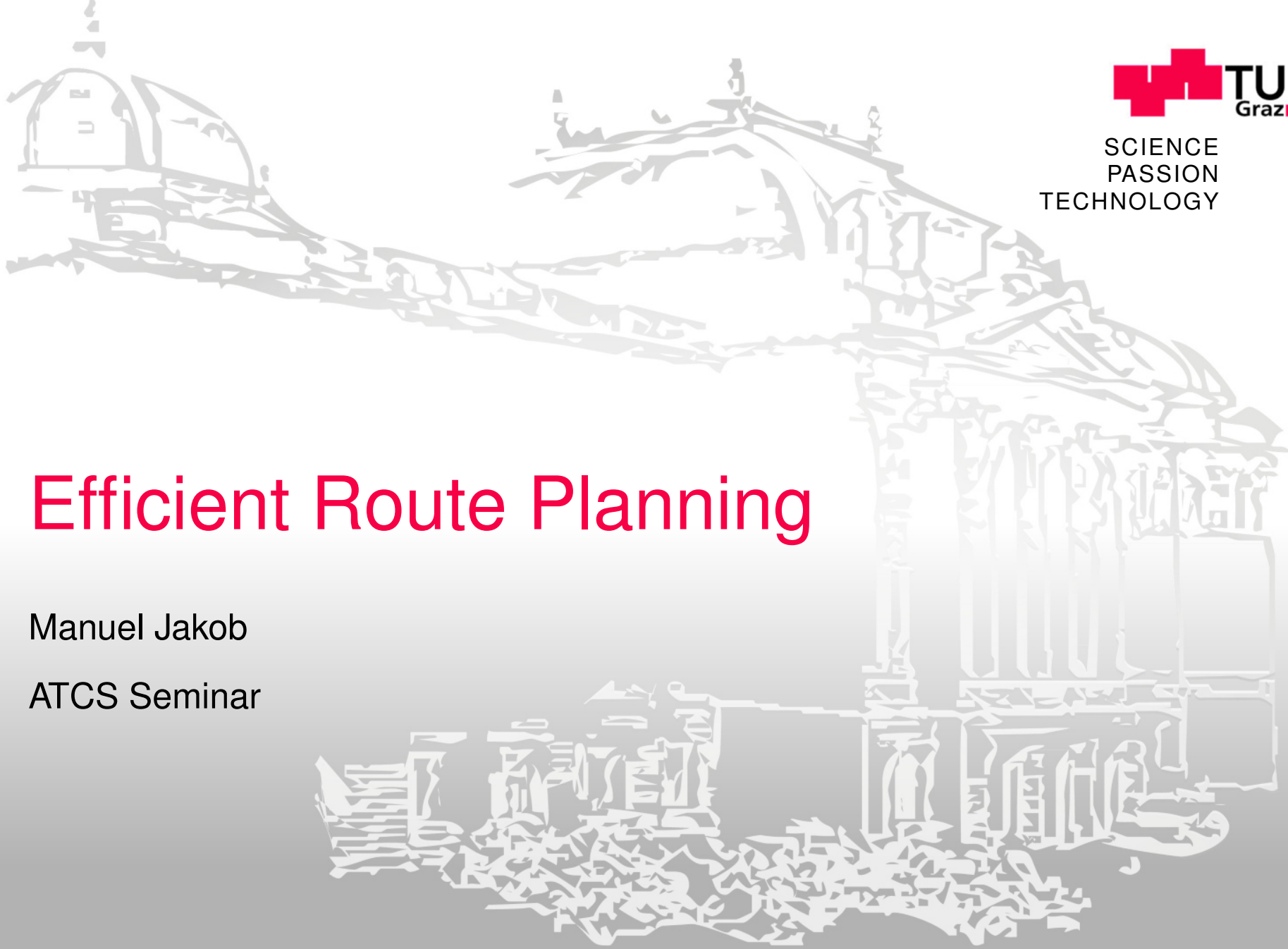


SCIENCE
PASSION
TECHNOLOGY

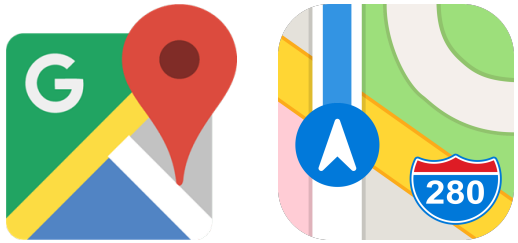
Efficient Route Planning

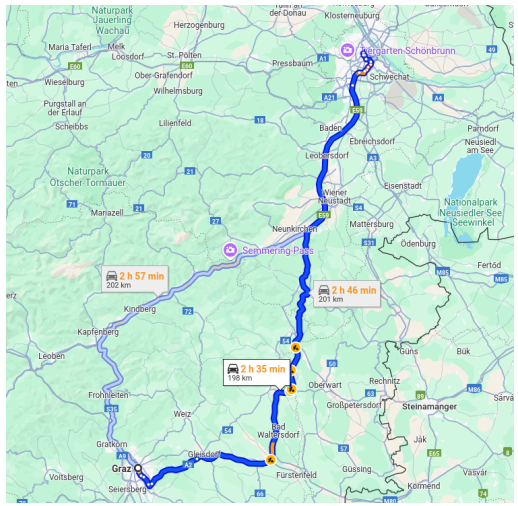
Manuel Jakob

ATCS Seminar

Motivation

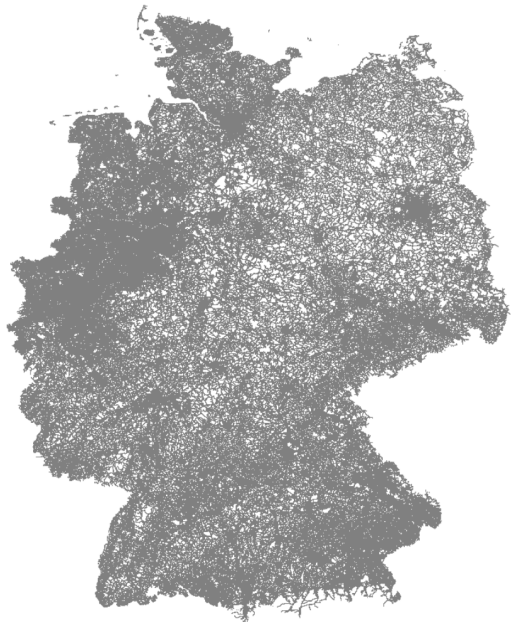
- real-time navigation
- big data
- new algorithms





Road Network

Graph $G = (V, E, \omega)$, $\omega : E \to R_+$
$|V| = n$, $|E| = m$

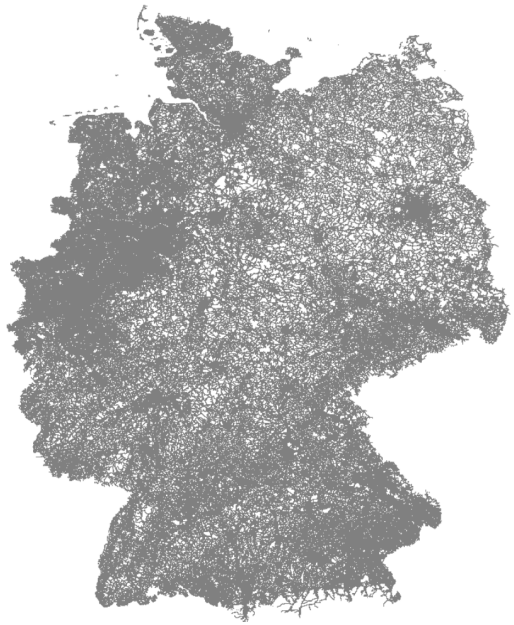


$|V| \approx 11.5M$
$|E| \approx 12.4M$

Road Network

Graph $G = (V, E, \omega)$, $\omega : E \to R_+$
$|V| = n$, $|E| = m$
Characteristics:

- almost planar
- constant max. degree
- sparse
- strongly connected
- hierarchical structure
- static

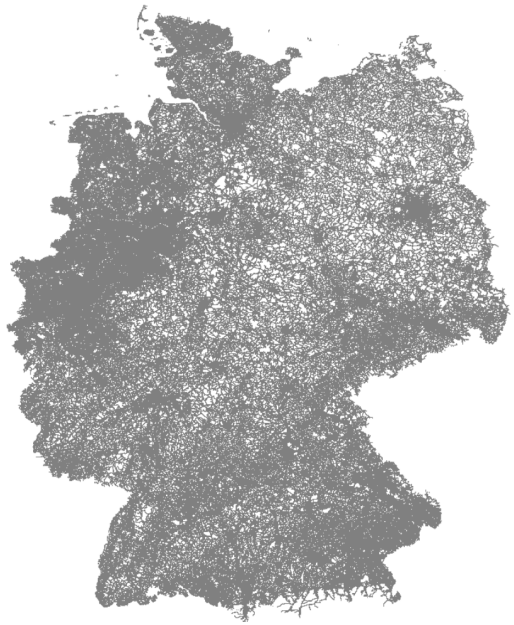


$|V| \approx 11.5M$
$|E| \approx 12.4M$

Road Network

Graph $G = (V, E, \omega)$, $\omega : E \to R_+$
$|V| = n$, $|E| = m$
Characteristics:

- almost planar
- constant max. degree
- sparse
- strongly connected
- hierarchical structure
- static



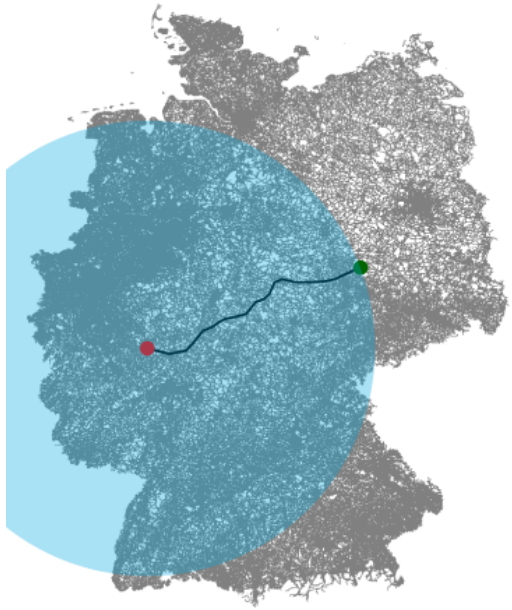
$|V| \approx 11.5M$
$|E| \approx 12.4M$

GOAL: find *shortest s-t* path (P2P)

Dijkstra's Algorithm [Dij59]

RUNTIME:

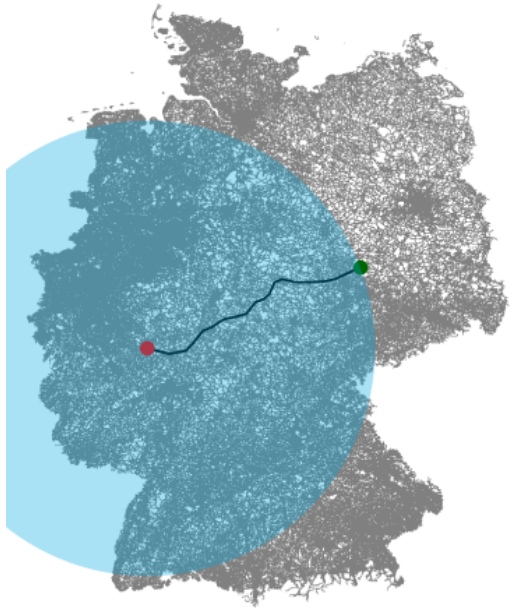
Search Space contains all vertices $v \in V$ with $\omega_s(v) < \omega_s(t)$



Dijkstra's Algorithm [Dij59]

RUNTIME: $\mathcal{O}(m + n \log n)$

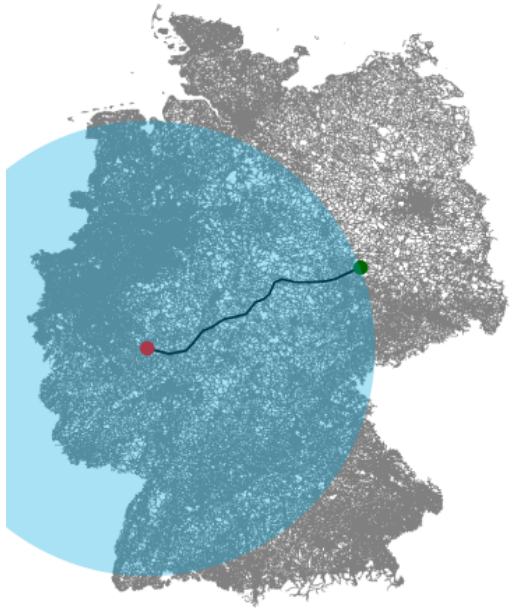
Search Space contains all vertices $v \in V$ with $\omega_s(v) < \omega_s(t)$



Dijkstra's Algorithm [Dij59]

RUNTIME: $\mathcal{O}(m + n \log n)$

Search Space contains all vertices $v \in V$ with $\omega_s(v) < \omega_s(t)$



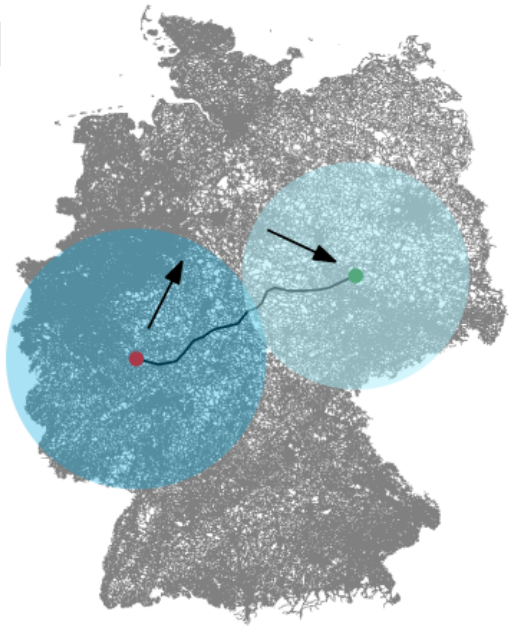
Running Dijkstra on country sized networks takes seconds!

Bidirectional Dijkstra's Algorithm

IDEA: perform the search from start and target simultaneously

Find $v \in V$ with

$$\min_{v \in V} \omega_s^{\rightarrow}(v) + \omega_t^{\leftarrow}(v)$$

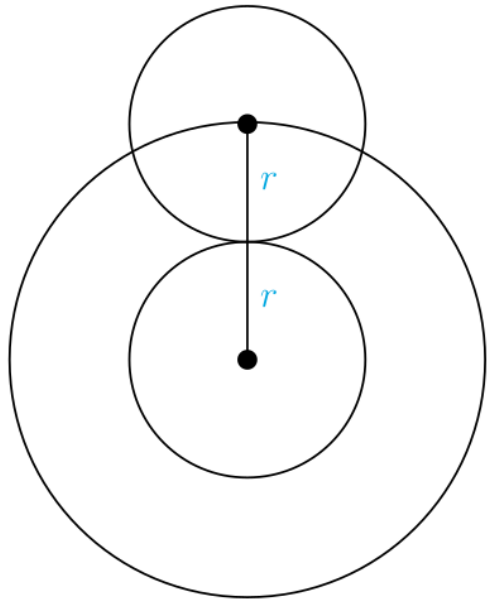


Bidirectional Dijkstra's Algorithm

Improvement:

Dijkstra: $\pi \cdot (2r)^2 = 4\pi r^2$
Bi-Dijkstra: $2 \cdot \pi r^2$

Decrease by a factor of 2

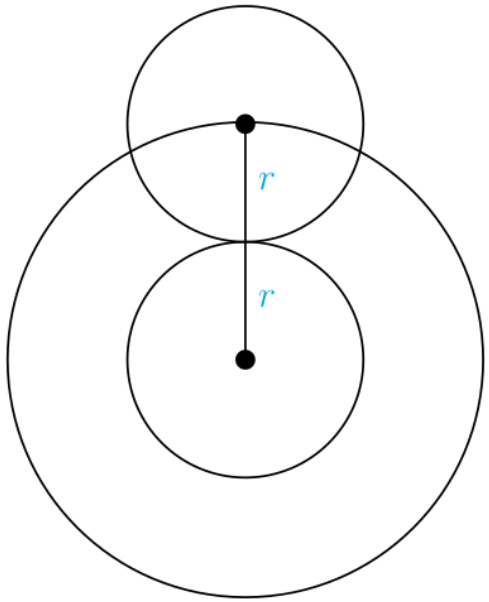


Bidirectional Dijkstra's Algorithm

Improvement:

Dijkstra: $\pi \cdot (2r)^2 = 4\pi r^2$
Bi-Dijkstra: $2 \cdot \pi r^2$

Decrease by a factor of 2



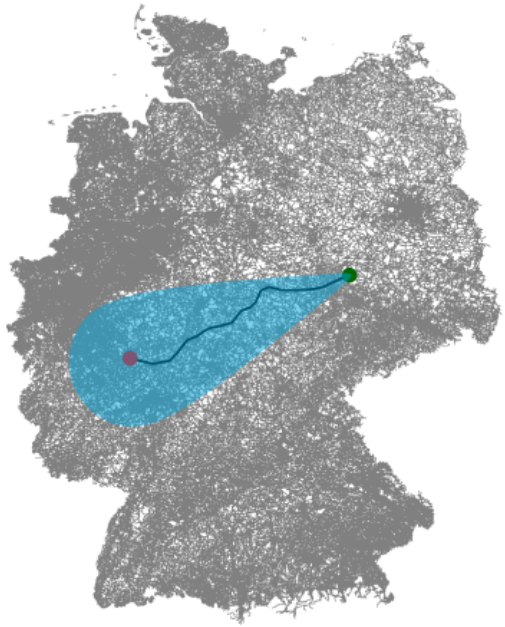
Search space still too large!

A* Algorithm

IDEA: guide search towards the target

Let $h(v)$ be an heuristic, estimates the costs from v to the target

Modify Dijkstra by setting
$\omega_s(v) + h(v)$ as key in PQ



A* Algorithm

A heuristic $h : E \rightarrow \mathcal{R}$ is...

A* Algorithm

A heuristic $h : E \to \mathcal{R}$ is...

admissible
$h(v)$ never overestimates

A* Algorithm

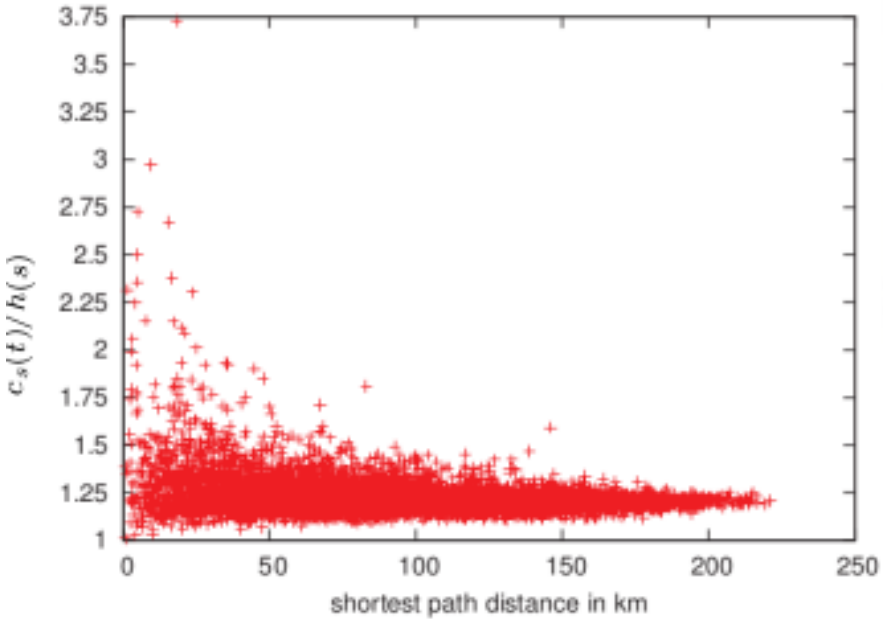
A heuristic $h : E \to \mathcal{R}$ is...

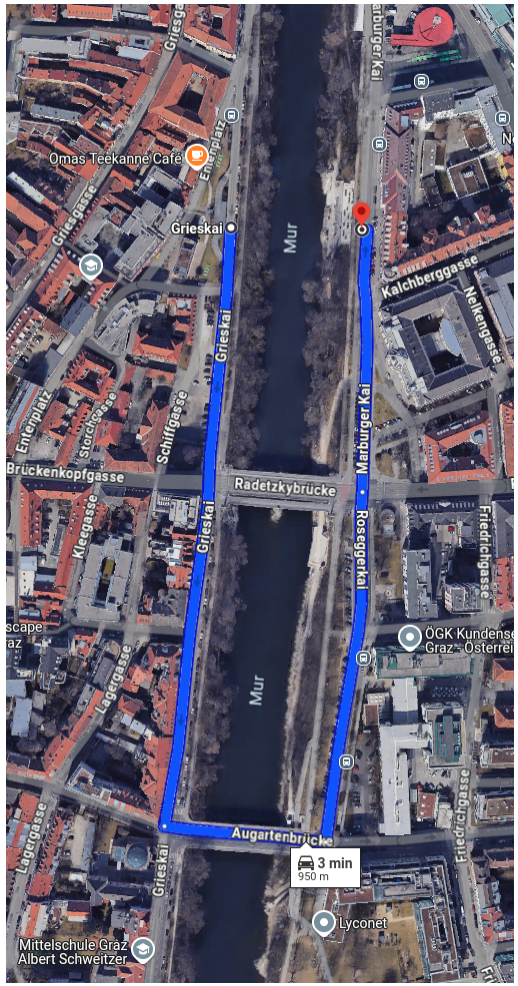
admissible
$h(v)$ never overestimates

consistent
$\forall e = (v, w) \in E : h(v) \leq \omega(e) + h(w)$

A* Algorithm

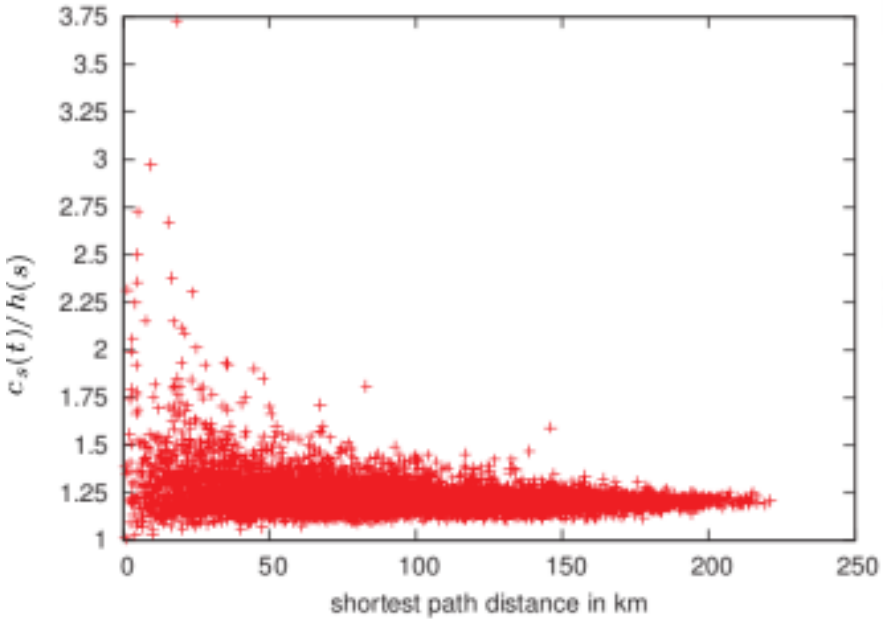
Example: bee line distance [FSS15]

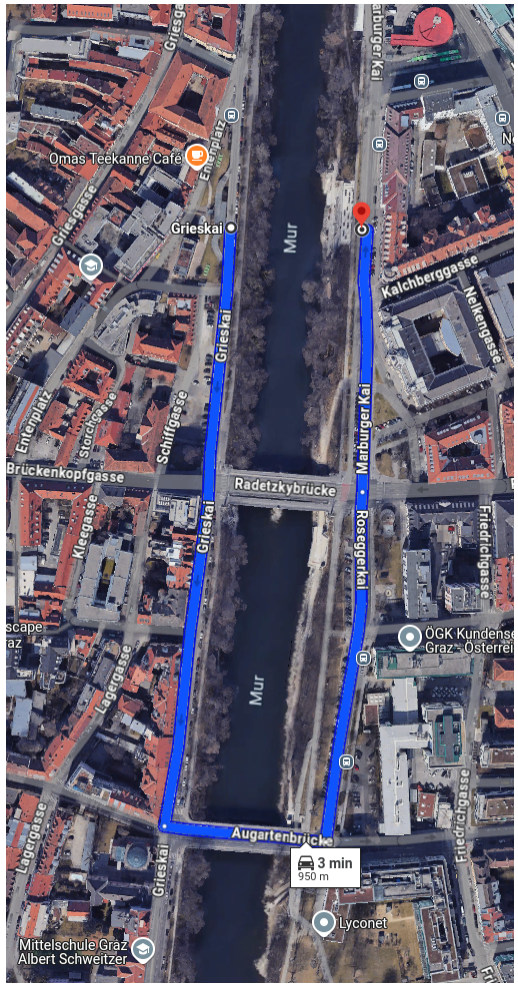




A* Algorithm

Example: bee line distance [FSS15]





Search space still too large!

How to speed up computation?

PREPROCESSING

How to speed up computation?

PREPROCESSING

1. Compute additional information in advance once

How to speed up computation?

PREPROCESSING

1. Compute additional information in advance once
2. Use data to reduce search space during query answering

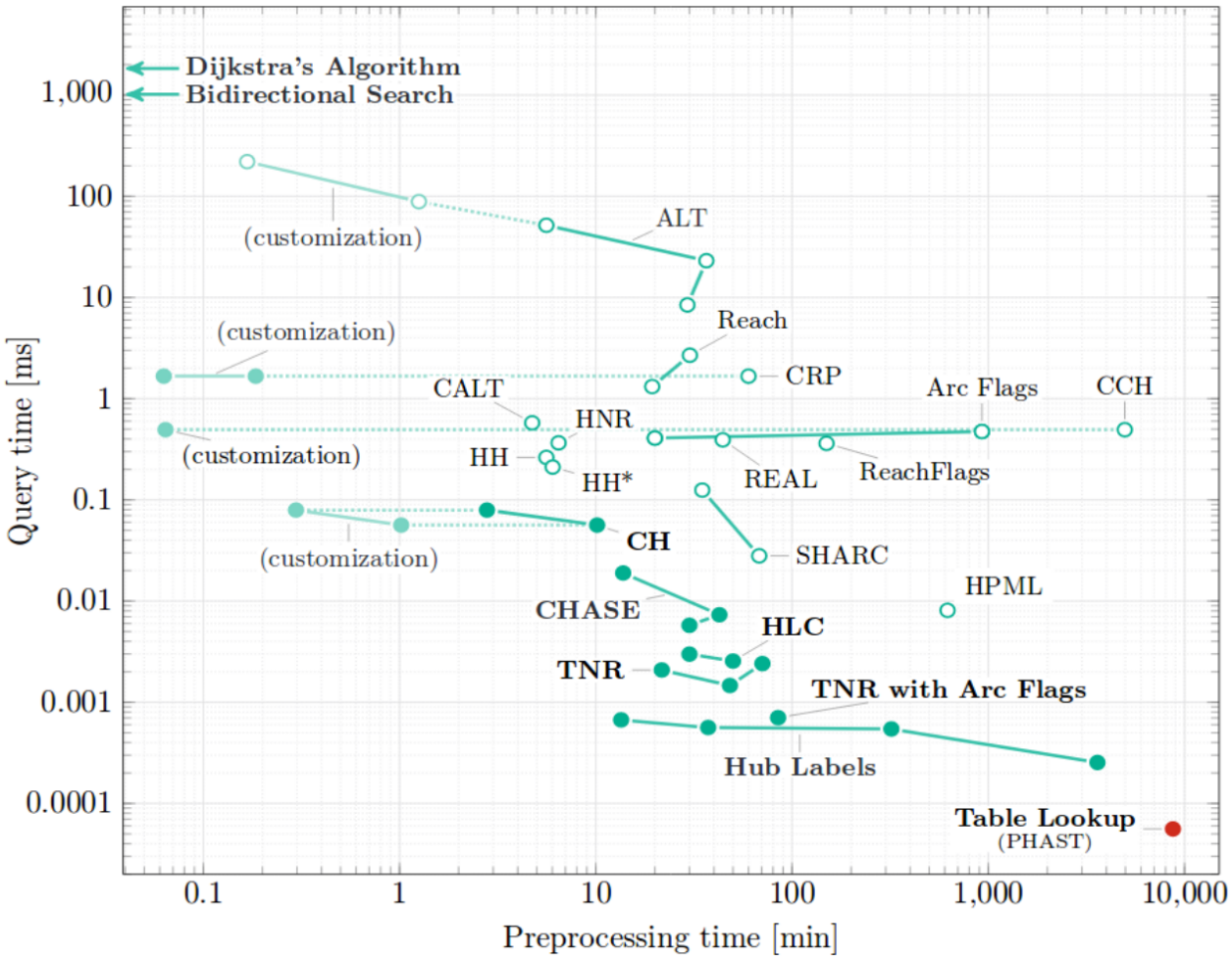
How to speed up computation?

PREPROCESSING

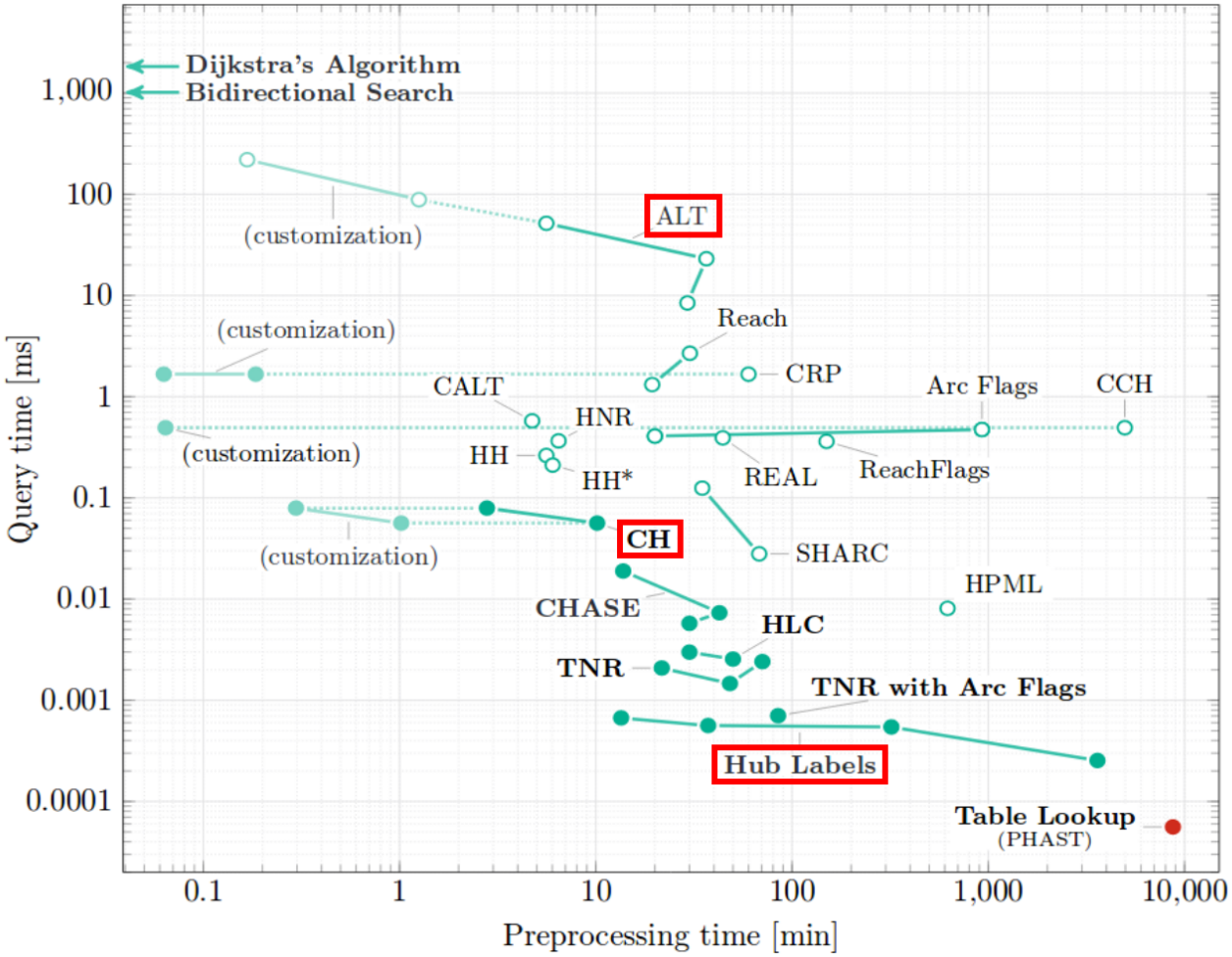
1. Compute additional information in advance once
2. Use data to reduce search space during query answering

Caution: preprocessing takes time and memory!

Big Picture


Dijkstra's Algorithm
Bidirectional Search
1,000
100
10
1
0.1
0.01
0.001
0.0001
Query time [ms]
(customization)
ALT
Reach
(customization)
CRP
CALT
Arc Flags
CCH
HNR
(customization)
HH
HH*
REAL
ReachFlags
CH
(customization)
SHARC
HPML
CHASE
HLC
TNR
TNR with Arc Flags
Hub Labels
Table Lookup
(PHAST)
0.1
1
10
100
1,000
10,000
Preprocessing time [min]

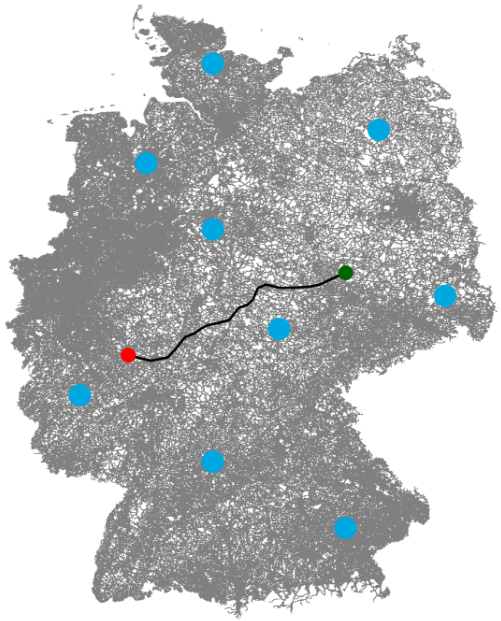
Big Picture


Dijkstra's Algorithm
Bidirectional Search
1,000
100
10
1
0.1
0.01
0.001
0.0001
Query time [ms]
(customization)
ALT
Reach
(customization)
CRP
CALT
Arc Flags
CCH
HNR
(customization)
HH
HH*
REAL
ReachFlags
(customization)
CH
SHARC
HPML
CHASE
HLC
TNR
TNR with Arc Flags
Hub Labels
Table Lookup
(PHAST)
0.1
1
10
100
1,000
10,000
Preprocessing time [min]

A* + **L**andmarks + **T**riangle inequality [GH05]

Preprocessing:

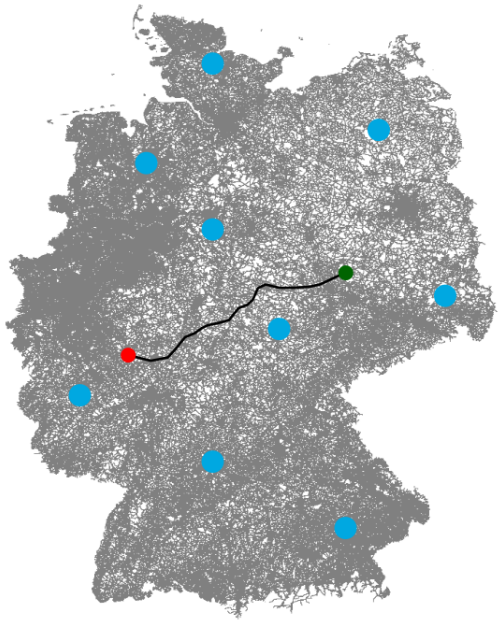
1. select set $L \subseteq V$ of landmarks
2. $\forall l \in L$: run Dijkstra and store costs to all other vertices



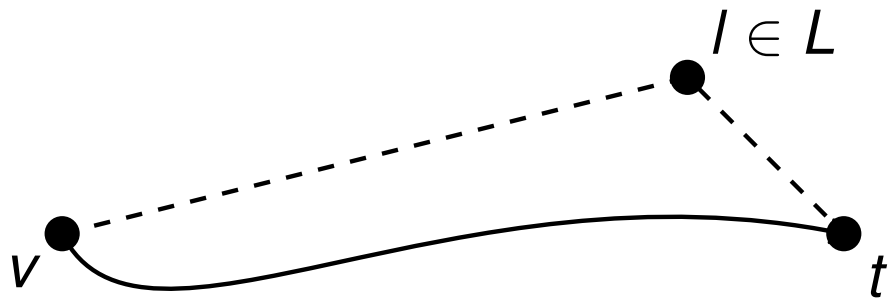
A* + **L**andmarks + **T**riangle inequality [GH05]

Preprocessing:

1. select set $L \subseteq V$ of landmarks
2. $\forall l \in L$: run Dijkstra and store costs to all other vertices



Heuristic:



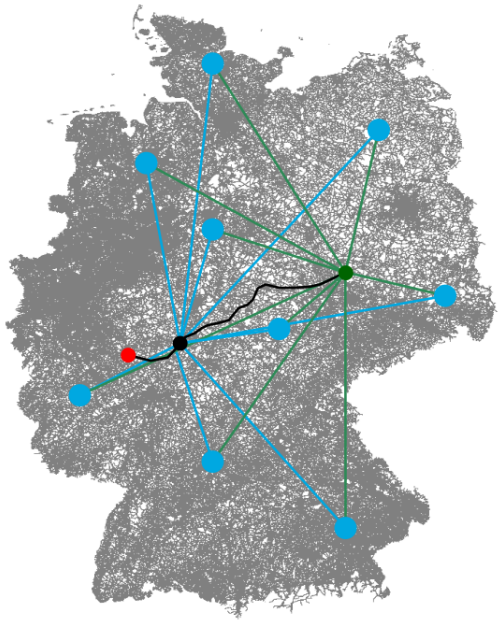
$$\omega_v(l) \leq \omega_v(t) + \omega_t(l) \Leftrightarrow$$

$$\omega_v(l) - \omega_t(l) \leq \omega_v(t)$$

A* + **L**andmarks + **T**riangle inequality

Preprocessing:

1. select set $L \subseteq V$ of landmarks
2. $\forall l \in L$: run Dijkstra and store costs to all other vertices



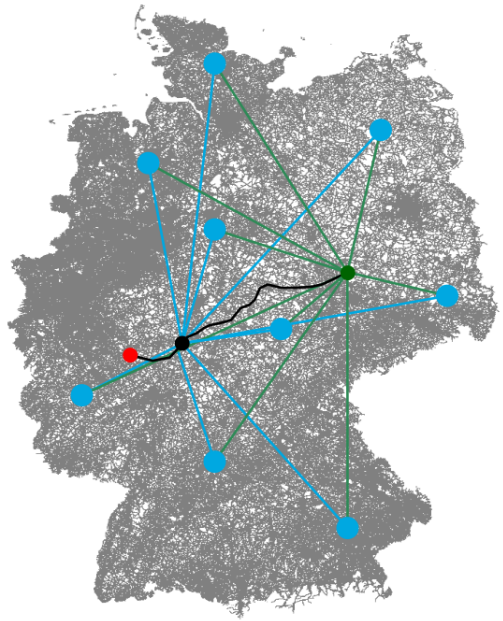
A* + **L**andmarks + **T**riangle inequality

Preprocessing:

1. select set $L \subseteq V$ of landmarks
2. $\forall l \in L$: run Dijkstra and store costs to all other vertices

Heuristic:

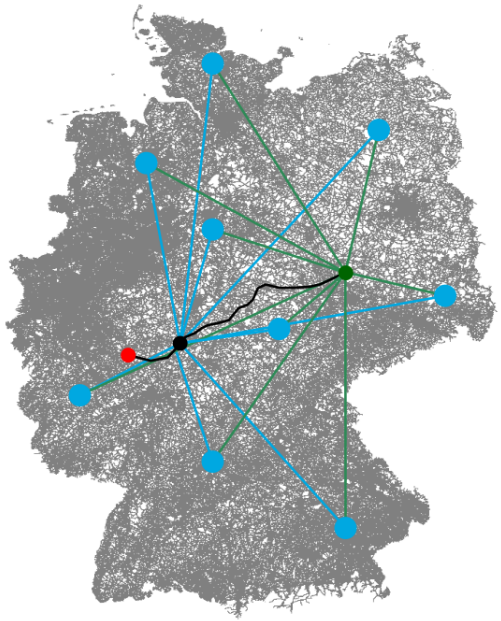
$h(v) = \max\{\omega_v(l) - \omega_t(l) \mid l \in L\}$



A* + **L**andmarks + **T**riangle inequality

Preprocessing:

1. select set $L \subseteq V$ of landmarks
2. $\forall l \in L$: run Dijkstra and store costs to all other vertices



Heuristic:

$h(v) = \max\{\omega_v(l) - \omega_t(l) \mid l \in L\}$

Small amount of landmarks is sufficient!

A* + **L**andmarks + **T**riangle inequality

Preprocessing:

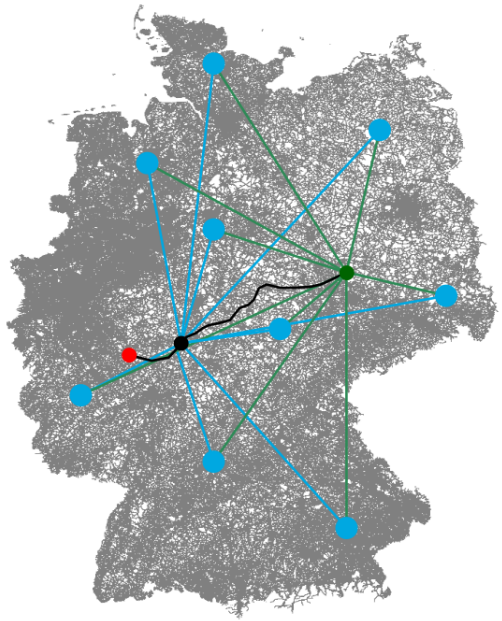
$$\mathcal{O}(|L| \cdot (n \log n + m)) + \text{select } L$$

Space Consumption:

$$\mathcal{O}(|L| \cdot n)$$

Query Time:

$$\mathcal{O}(n \log n + |L| \cdot m)$$



A* + Landmarks + Triangle inequality

Results: [GH05][Gol07]

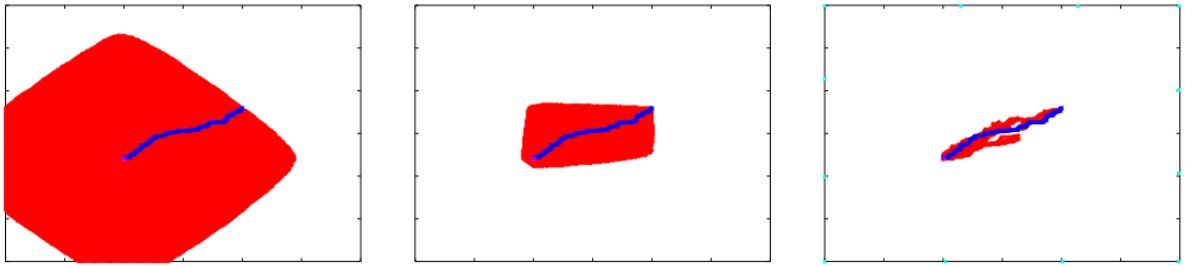


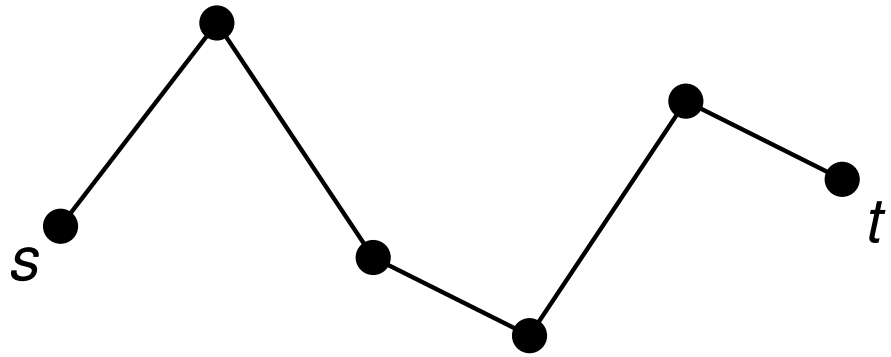
Figure 1: Vertices visited by Dijkstra's algorithm (left), A^* search with Manhattan lower bounds (middle), and an ALT algorithm (right) on the same input.

	preprocessing		query		
method	seconds	MB	avgscan	maxscan	ms
B	—	6.1	116 588	293 152	30.49
ALT	5.7	27.8	4 430	54 194	2.91
RE	45.4	12.3	668	1 697	0.55
REAL	51.1	34.0	172	982	0.28
RE (no shortcuts)	753.3	7.4	13 419	28 670	5.24
RE (no shortcuts, exact reaches)	8 629.9	7.4	11 140	24 565	4.41

Table 2. Data for the Bay Area road network.

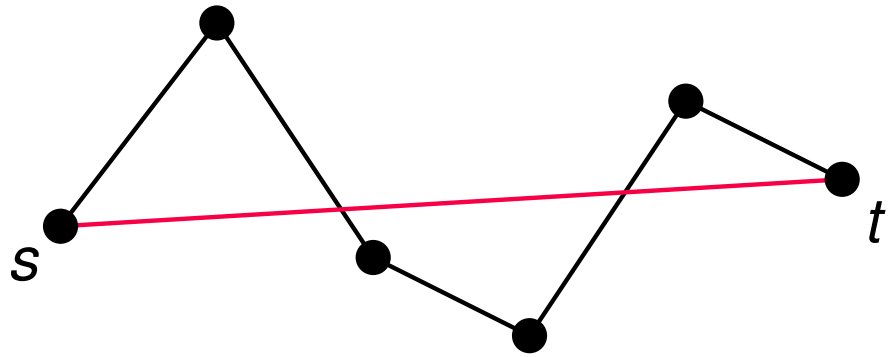
Contraction Hierarchies [GSSD08]

IDEA: Add shortcuts to the graph.



Contraction Hierarchies [GSSD08]

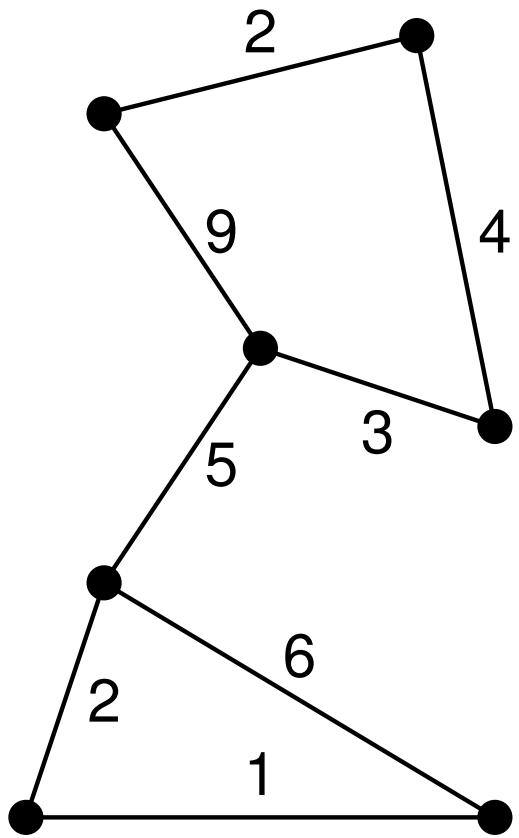
IDEA: Add shortcuts to the graph.



Contraction Hierarchies

Preprocessing:

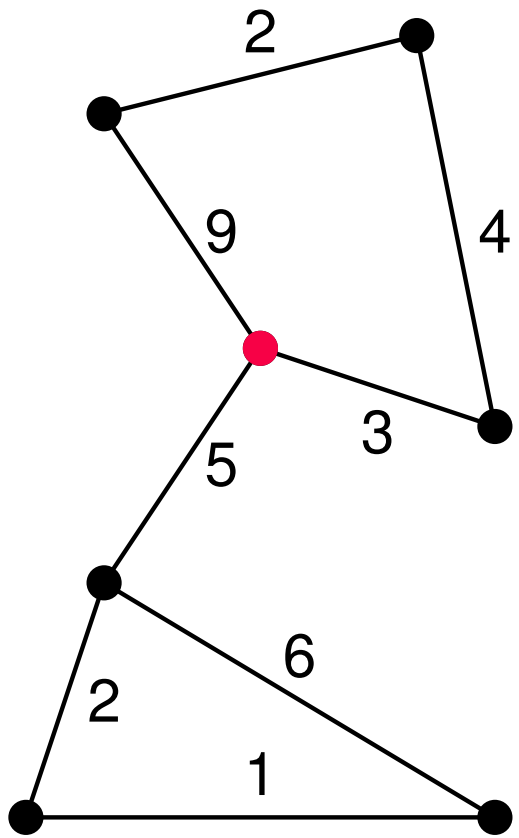
1. Define rank $r : V \to \mathbb{N}$
2. Contract nodes based on their rank
   - remove node v and incident edges
   - add shortcut between neighbors if SP goes only through v
3. Add shortcuts E^+ to G remember contraction order!



Contraction Hierarchies

Preprocessing:

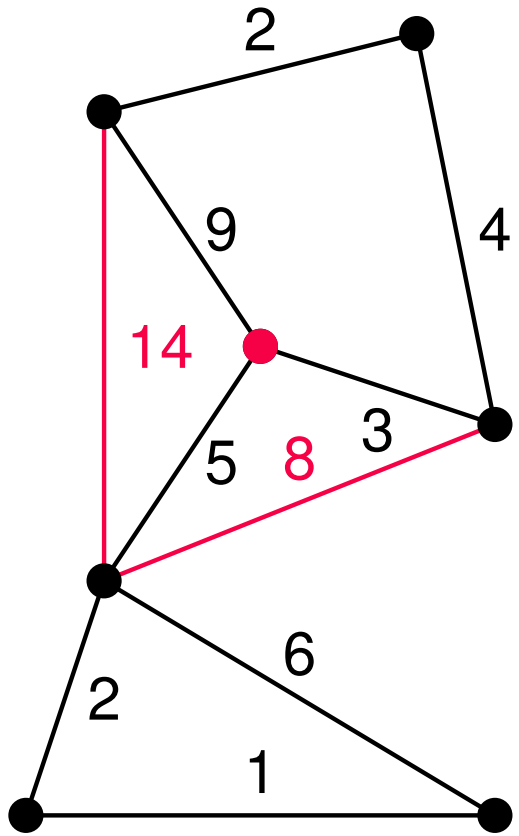
1. Define rank $r : V \to \mathbb{N}$
2. Contract nodes based on their rank
   - remove node v and incident edges
   - add shortcut between neighbors if SP goes only through v
3. Add shortcuts E^+ to G remember contraction order!



Contraction Hierarchies

Preprocessing:

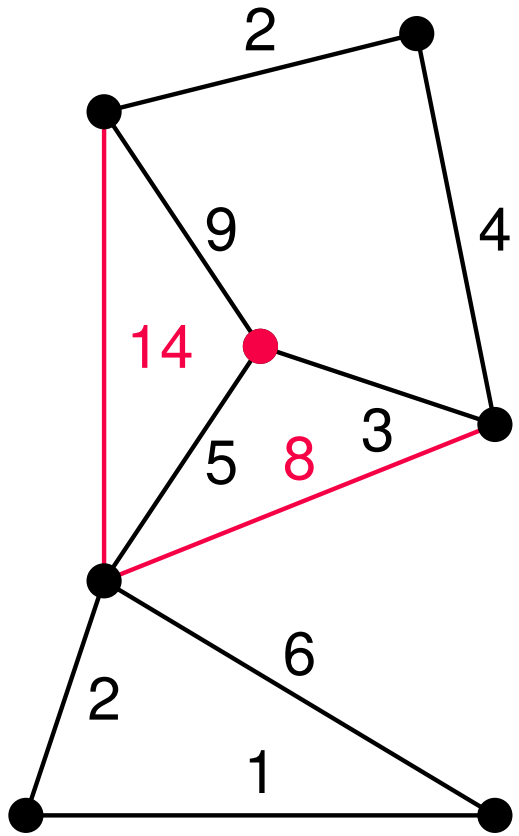
1. Define rank $r : V \to \mathbb{N}$
2. Contract nodes based on their rank
   - remove node v and incident edges
   - add shortcut between neighbors if SP goes only through v
3. Add shortcuts E^+ to G remember contraction order!



Contraction Hierarchies

Preprocessing:

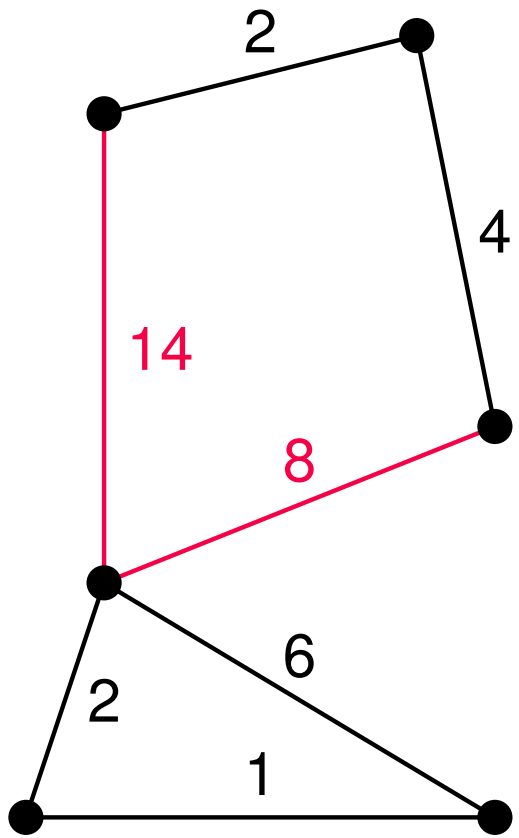
1. Define rank $r : V \to \mathbb{N}$
2. Contract nodes based on their rank
   - remove node v and incident edges
   - add shortcut between neighbors if SP goes only through v
3. Add shortcuts E^+ to G remember contraction order!



Contraction Hierarchies

Preprocessing:

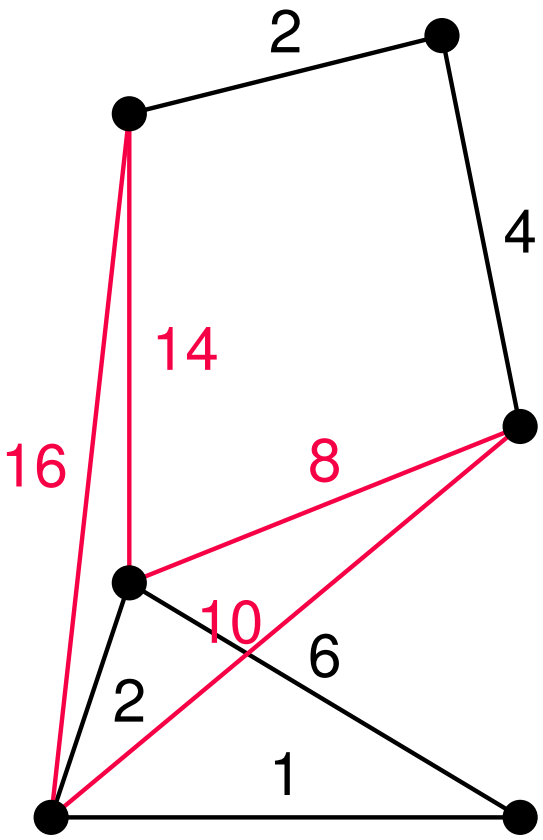
1. Define rank $r : V \to \mathbb{N}$
2. Contract nodes based on their rank
   - remove node v and incident edges
   - add shortcut between neighbors if SP goes only through v
3. Add shortcuts E^+ to G remember contraction order!



Contraction Hierarchies

Preprocessing:

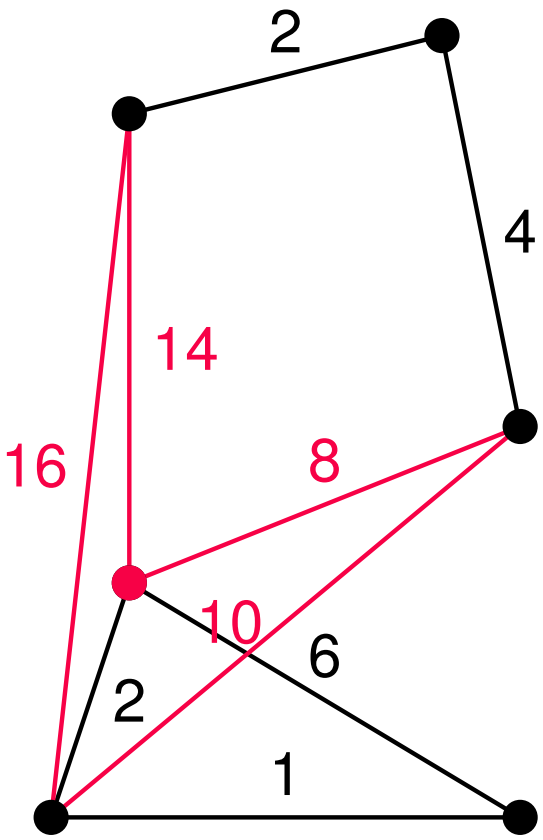
1. Define rank $r : V \to \mathbb{N}$
2. Contract nodes based on their rank
   - remove node v and incident edges
   - add shortcut between neighbors if SP goes only through v
3. Add shortcuts E^+ to G remember contraction order!



Contraction Hierarchies

Preprocessing:

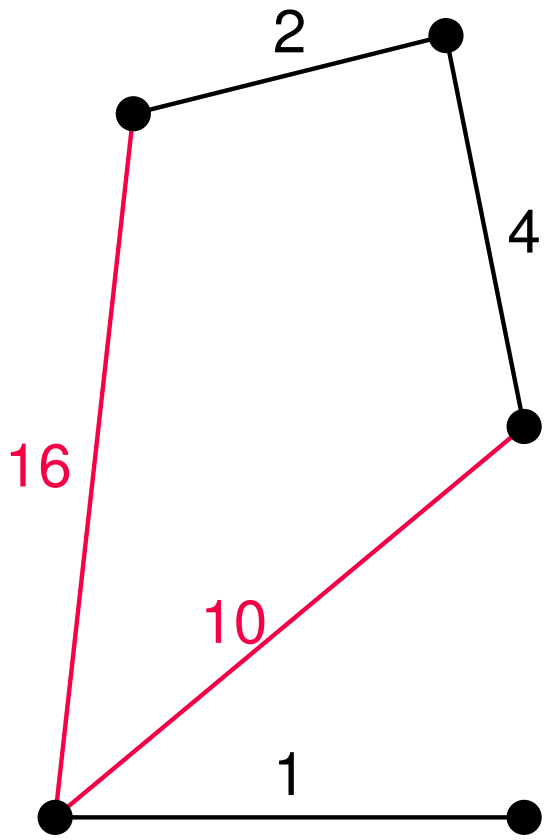
1. Define rank $r : V \to \mathbb{N}$
2. Contract nodes based on their rank
   - remove node v and incident edges
   - add shortcut between neighbors if SP goes only through v
3. Add shortcuts E^+ to G remember contraction order!



Contraction Hierarchies

Preprocessing:

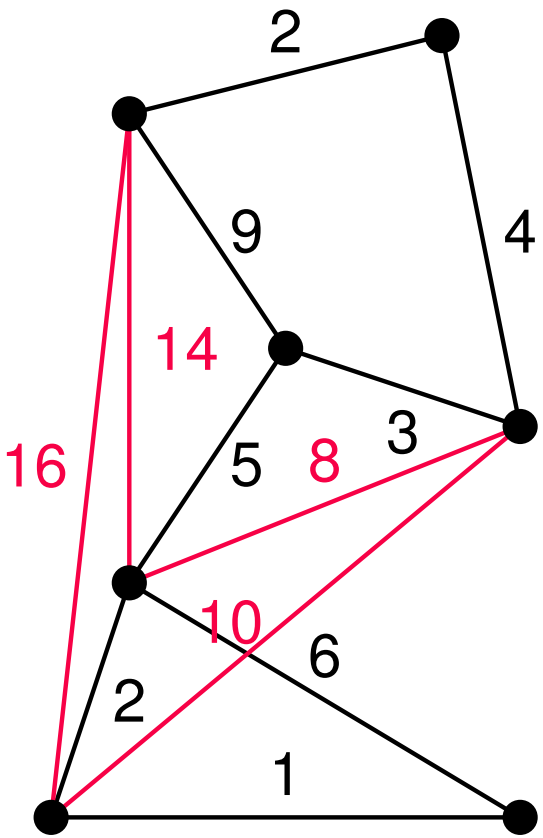
1. Define rank $r : V \to \mathbb{N}$
2. Contract nodes based on their rank
   - remove node v and incident edges
   - add shortcut between neighbors if SP goes only through v
3. Add shortcuts E^+ to G remember contraction order!



Contraction Hierarchies

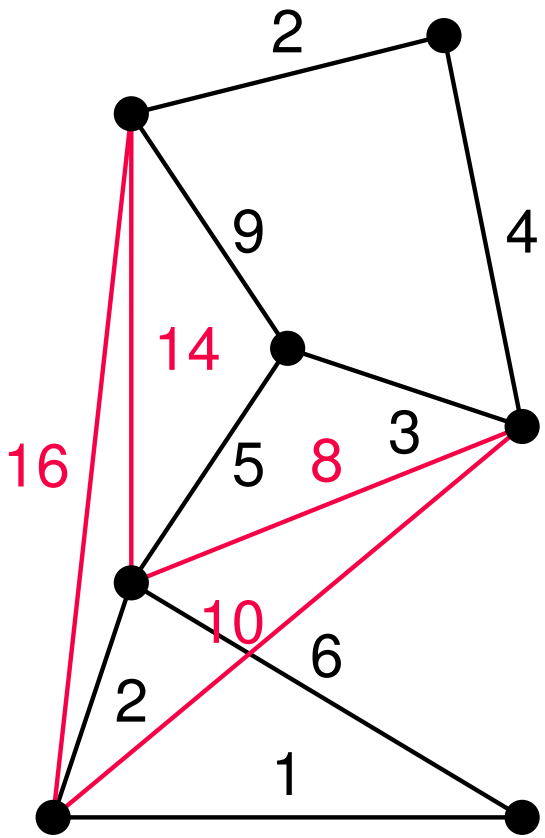
Preprocessing:

1. Define rank $r : V \to \mathbb{N}$
2. Contract nodes based on their rank
   - remove node v and incident edges
   - add shortcut between neighbors if SP goes only through v
3. Add shortcuts E^+ to G remember contraction order!



Contraction Hierarchies

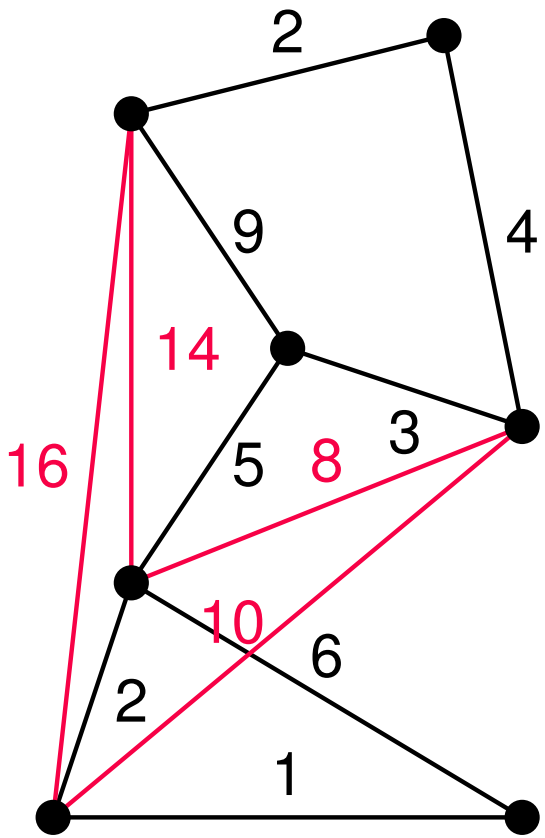
Preprocessing:
What is a good contraction order?



Contraction Hierarchies

Preprocessing:
What is a good contraction order?

Intuition: contract *important* nodes late

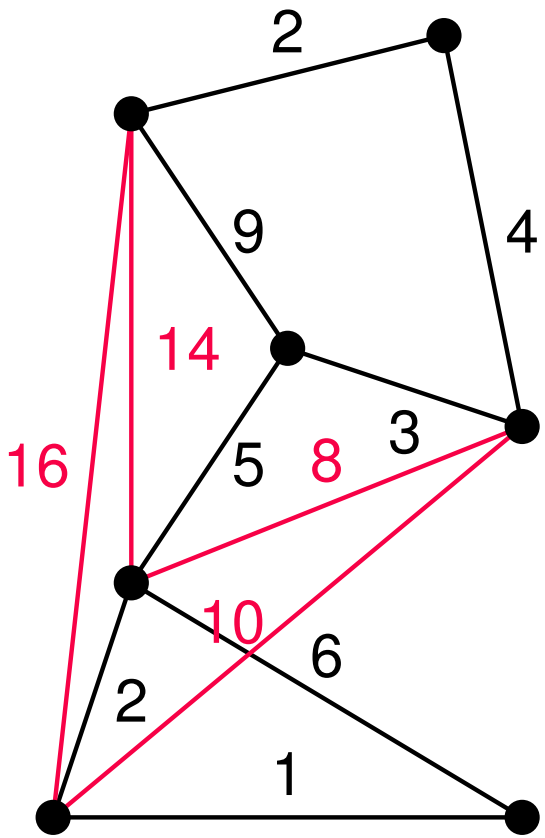


Contraction Hierarchies

Preprocessing:
What is a good contraction order?

Intuition: contract *important* nodes late

minimizing $|E^+|$ is NP-hard!



Contraction Hierarchies

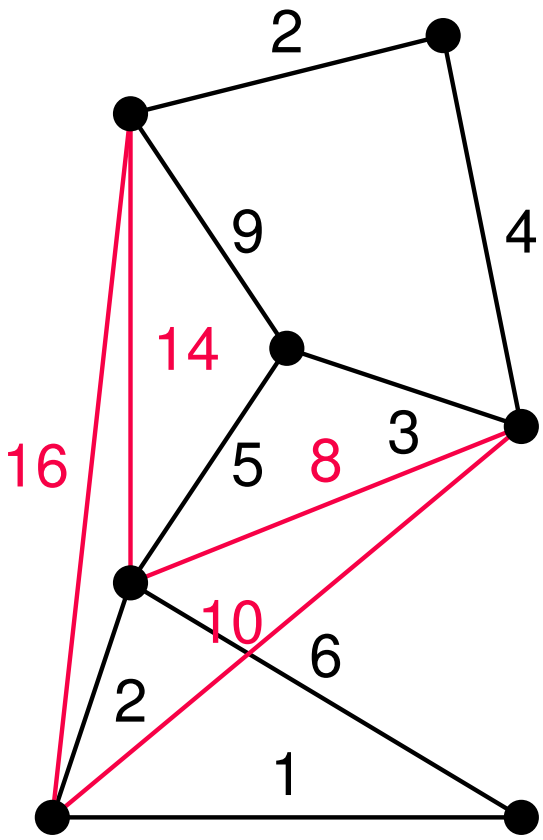
Preprocessing:
What is a good contraction order?

Intuition: contract *important* nodes late

minimizing $|E^+|$ is NP-hard!

Example: Edge difference
$ED(v) = -|N(v)| + \#\text{shortcuts of } v$

$$\Rightarrow |E^+| \approx |E|$$



Contraction Hierarchies

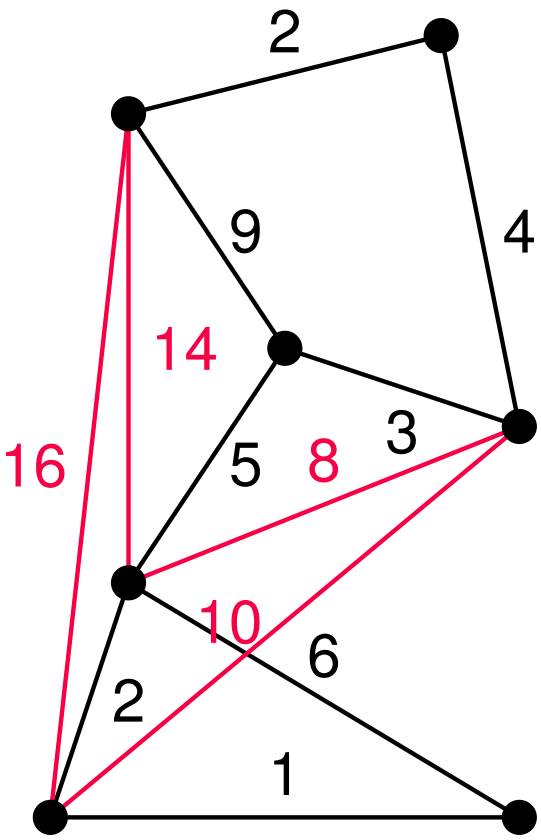
Preprocessing:
What is a good contraction order?

Intuition: contract *important* nodes late

minimizing $|E^+|$ is NP-hard!

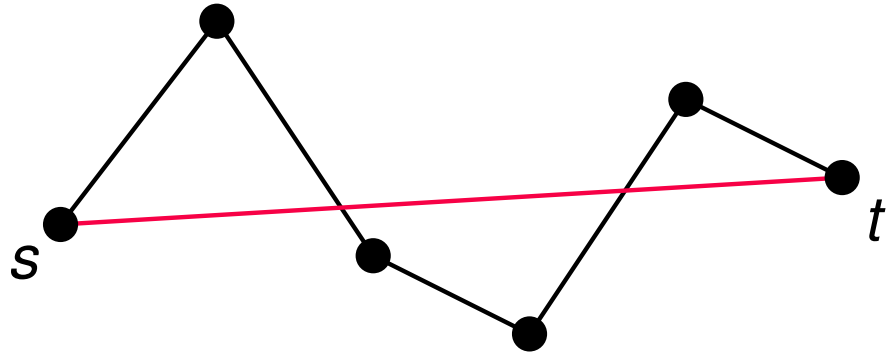
Example: Edge difference
$ED(v) = -|N(v)| + \#\text{shortcuts of } v$

$$\Rightarrow |E^+| \approx |E|$$



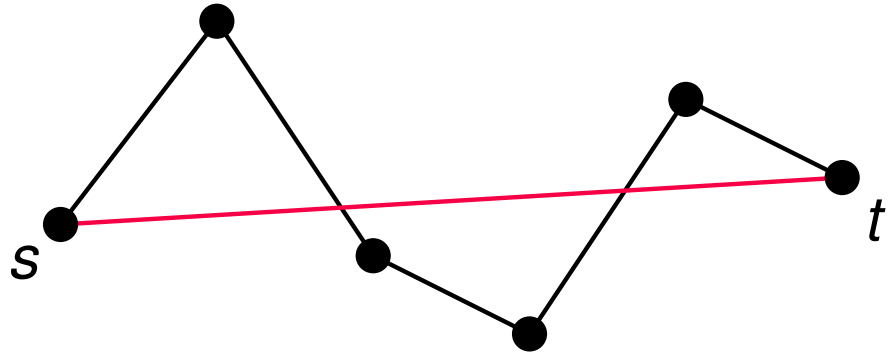
Preprocessing Time: $\mathcal{O}(n^2 \log n + nm)$ but only local search

Contraction Hierarchies



Query Answering: Convince Dijkstra to use shortcuts!

Contraction Hierarchies



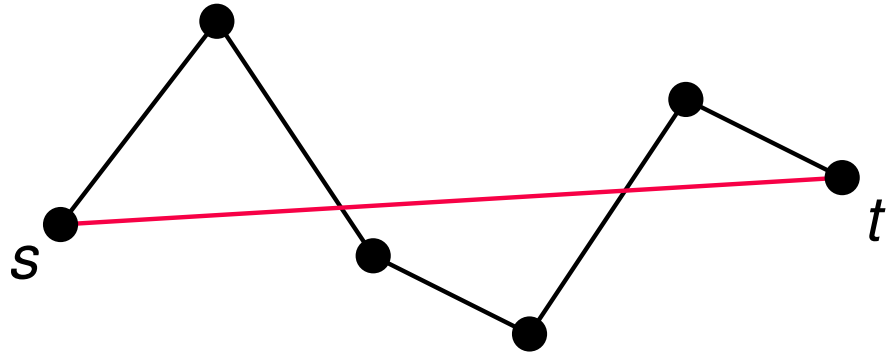
Query Answering: Convince Dijkstra to use shortcuts!
Def:

$e = (u, v)$ is called upwards, if $r(u) < r(v)$

An upwards path contains solely upward edges.

$G^{\uparrow}(v)$ is the union of all upwards path starting at v

Contraction Hierarchies



Query Answering: Convince Dijkstra to use shortcuts!
Def:

$e = (u, v)$ is called upwards, if $r(u) < r(v)$

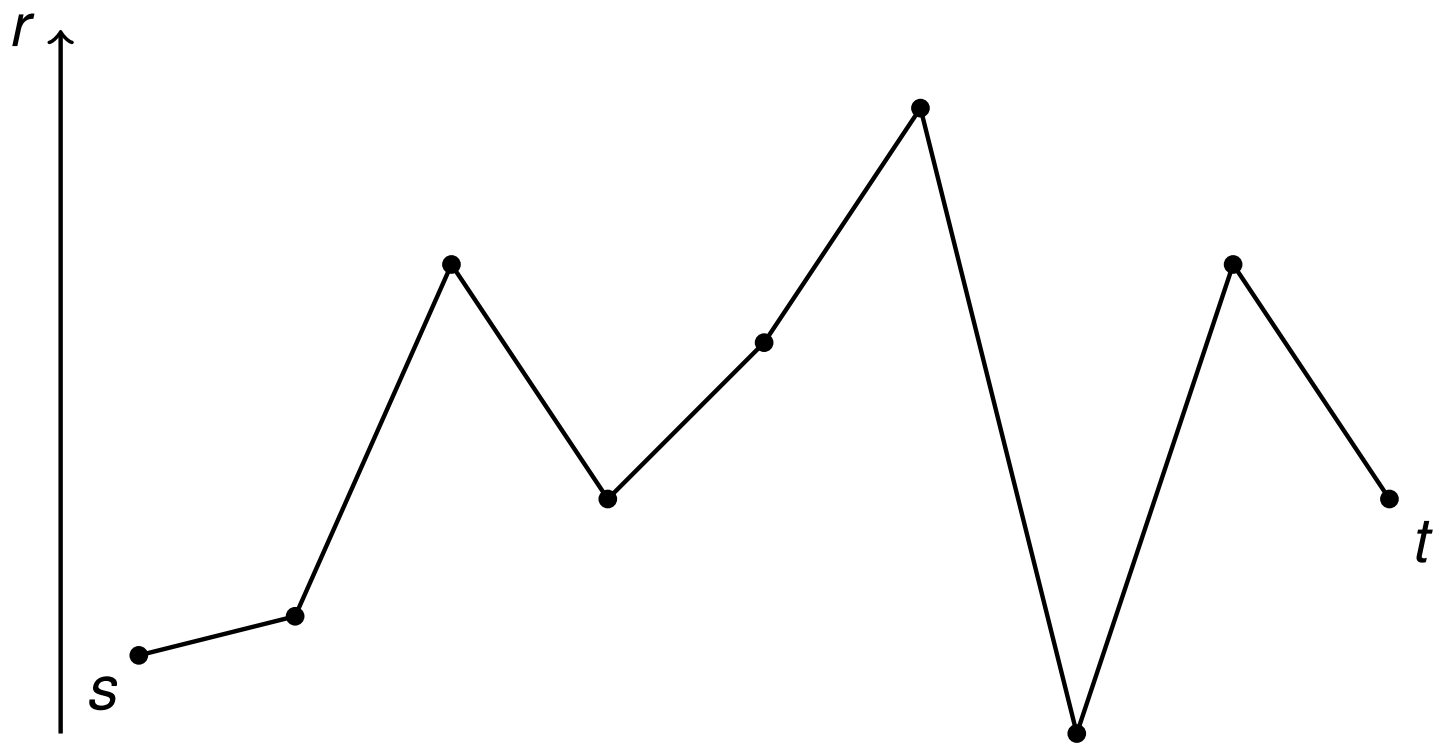
An upwards path contains solely upward edges.

$G^{\uparrow}(v)$ is the union of all upwards path starting at v

CH-Dijkstra: Bi-directional Dijkstra where forwards run is restricted to $G^{\uparrow}(s)$ and the backwards run is restricted to $G^{\downarrow}(t)$

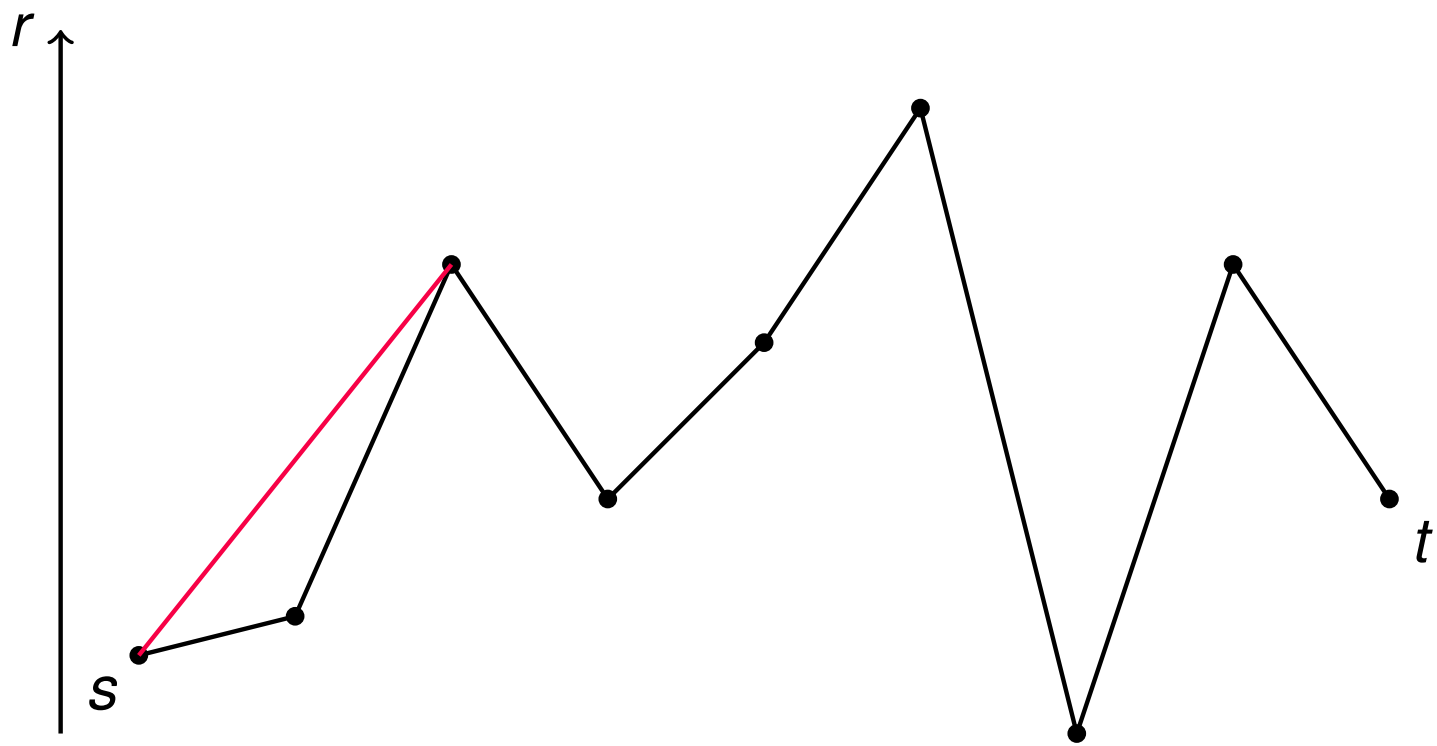
Contraction Hierarchies

Lemma: CH-Dijkstra computes the shortest *s*-*t* path



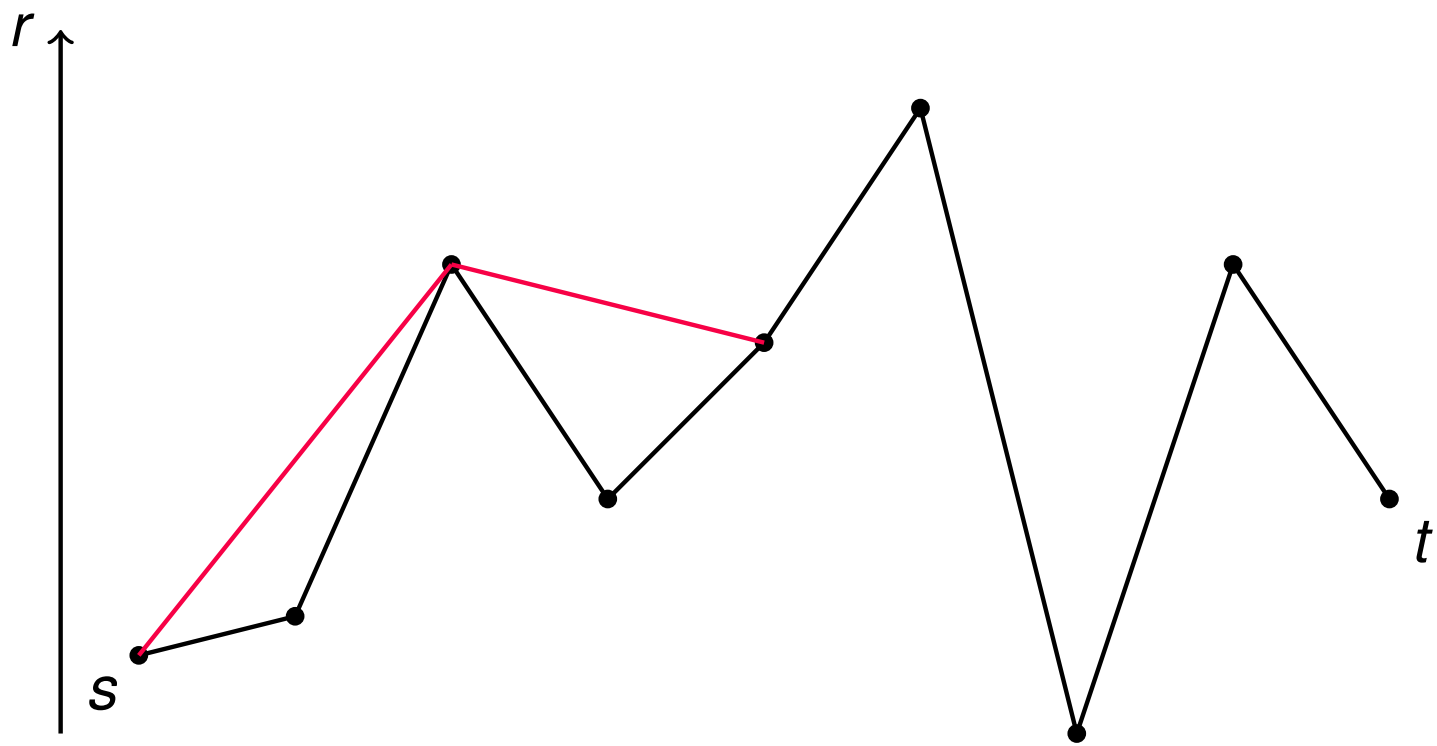
Contraction Hierarchies

Lemma: CH-Dijkstra computes the shortest *s*-*t* path



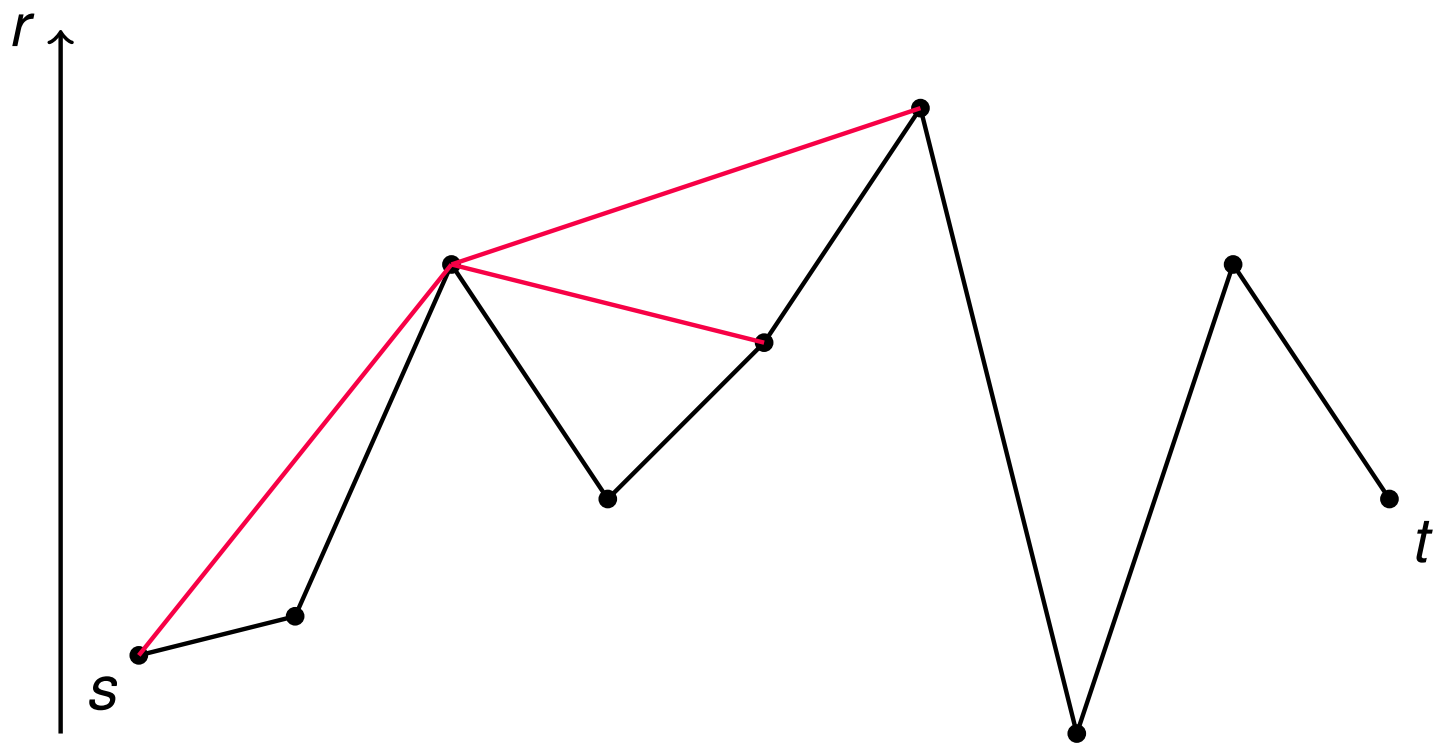
Contraction Hierarchies

Lemma: CH-Dijkstra computes the shortest *s*-*t* path



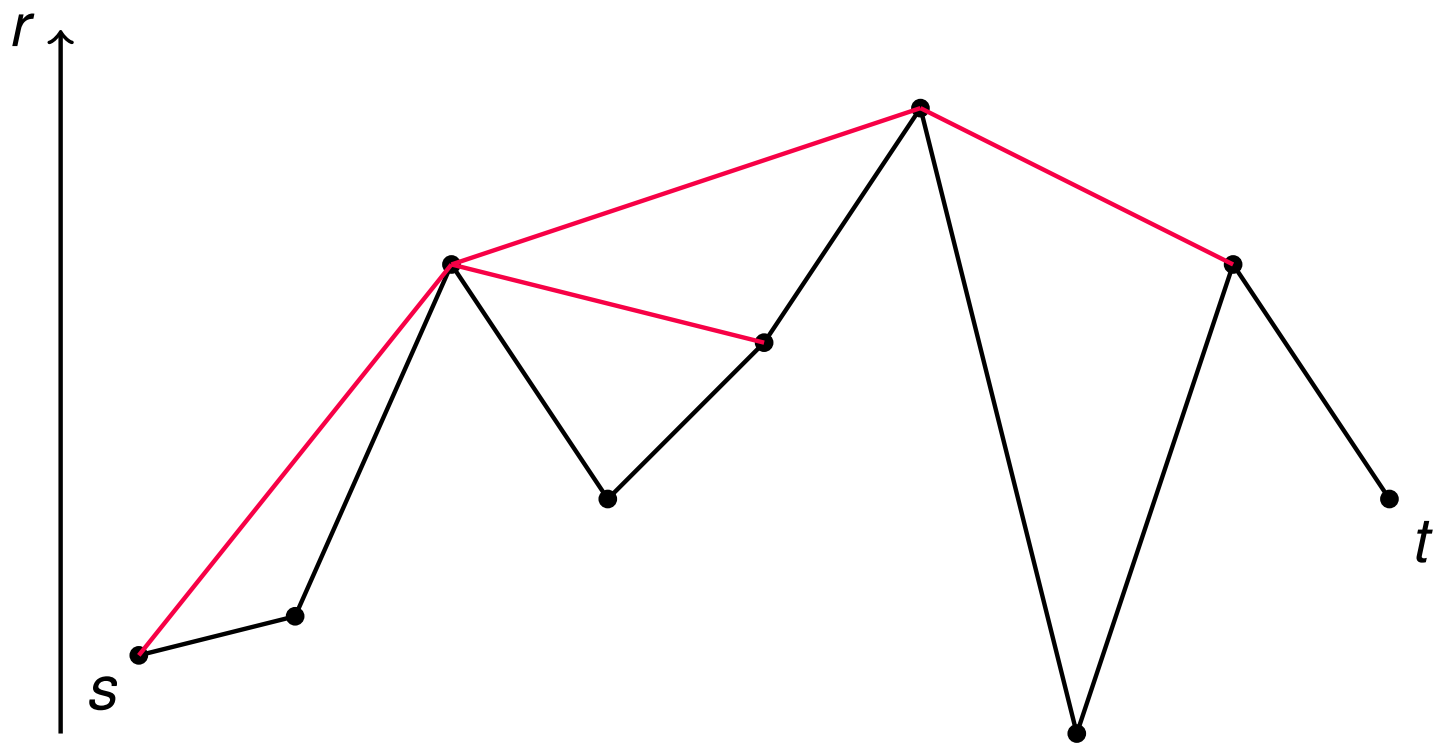
Contraction Hierarchies

Lemma: CH-Dijkstra computes the shortest *s*-*t* path



Contraction Hierarchies

Lemma: CH-Dijkstra computes the shortest *s*-*t* path



Contraction Hierarchies

Results: [GSSV12]

		prepro.		query			dom.
		time	overh.	settled	time	uses	by
method	data from	[min]	[B/n.]	nodes	[ms]	CH	CH
Dijkstra[a]	[Bauer et al. 2010b]	0	0	9.11 M	5 591		
bidir. Dijkstra[a]	[Bauer et al. 2010b]	0	0	4.76 M	2 713		
eco. CH	this paper	**8**	0.6	487	0.21	✓	
aggr. CH	this paper	27	-2.1	356	0.15	✓	
ALT-16[b]	[Goldberg et al. 2009]	13	70	82 348	120.1		✓
ALT-64[a]	[Delling and Wagner 2007]	68	512	25 234	19.6		✓
AF[c]	[Hilger et al. 2009]	2 156	25	1 593	1.1		✓
REAL[b]	[Goldberg et al. 2009]	103	36	610	0.91		✓
HH	[Schultes 2008]	13	48	709	0.61		✓
HNR	[Schultes 2008]	15	2.4	981	0.85		✓
SHARC[a]	[Bauer and Delling 2009]	81	14.5	654	0.29		✓
bidir. SHARC[a]	[Bauer and Delling 2009]	158	21.0	125	0.065		✓[d]
CALT[a]	[Bauer et al. 2010b]	11	15.4	1 394	1.34		✓
eco. CH+AF[a]	[Bauer et al. 2010b]	32	0.0	111	0.044	✓	
gen. CH+AF[a]	[Bauer et al. 2010b]	99	12	**45**	0.017	✓	
partial CH[a]	[Bauer et al. 2010b]	15	**-2.9**	965 k	53.63	✓	
TNR	this paper	46	193	N/A	0.0033	✓[e]	
TNR+AF	[Bauer et al. 2010b]	229	321	N/A	**0.0019**	✓[e]	

Europe: $|N| \approx 18M$, $|E| \approx 42M$

Hub Labels

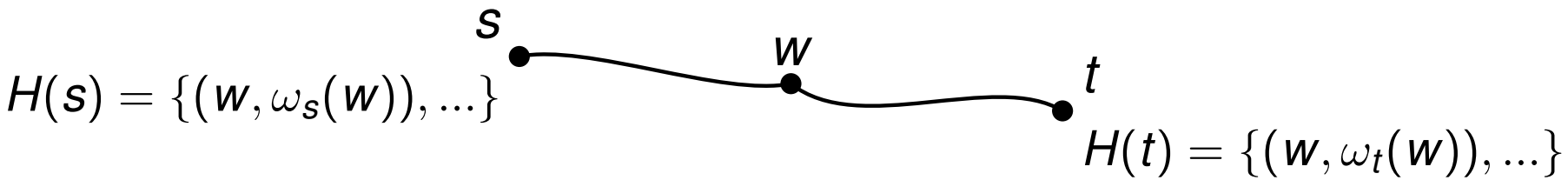
IDEA: Store distances to *important* nodes (hubs).

Hub Labels

IDEA: Store distances to *important* nodes (hubs).

Def:

$H(v) = \{(w_1, \omega_s(w_1)), (w_2, \omega_s(w_2)), ...\}$ - hub labels of v



$$\omega_s(t) = \min\{\omega_s(w) + \omega_t(w) \mid w \in H(s) \cap H(t)\}$$

Cover Property:

$\forall s, t \in V : \exists w \in H(s) \cap H(t) : \omega_s(t) = \omega_s(w) + \omega_t(w)$

Query Time: $\mathcal{O}(|H(s)| + |H(t)|)$

Hub Labels

Space Consumption: $\mathcal{O}(\Sigma_{v \in V} |H(v)|)$

Hub Labels

Space Consumption: $\mathcal{O}(\Sigma_{v \in V} |H(v)|)$

Find minimal Hub label sets is NP-Hard!

Hub Labels

Approach 1: Set Cover Approximation

Def: Set Cover

U - Universe of size n

$\mathcal{S}$ - collection of subsets S of U with weights $w(s)$

Find $\mathcal{S}' \subseteq \mathcal{S}$ of min. weight s.t. $\bigcup_{S \in \mathcal{S}'} S = U$

Hub Labels

Approach 1: Set Cover Approximation

Def: Set Cover

U - Universe of size n

$\mathcal{S}$ - collection of subsets S of U with weights $w(s)$

Find $\mathcal{S}' \subseteq \mathcal{S}$ of min. weight s.t. $\bigcup_{S \in \mathcal{S}'} S = U$

Reduction:

$U = \{(s,t) \in V^2 \mid \omega(s,t) < \infty\}$

$S(A, v, B) := \{(s,t) \mid s \in A, t \in B, v \in \pi(s,t)\}$,
with $w(S) = |A| + |B|$
$\mathcal{S}$ contains all possible $S(A, v, B)$

Hub Labels

Approach 1: Set Cover Approximation

Def: Set Cover

U - Universe of size n

$\mathcal{S}$ - collection of subsets S of U with weights $w(s)$

Find $\mathcal{S}' \subseteq \mathcal{S}$ of min. weight s.t. $\bigcup_{S \in \mathcal{S}'} S = U$

Reduction:

$U = \{(s,t) \in V^2 \mid \omega(s,t) < \infty\}$

$S(A, v, B) := \{(s,t) \mid s \in A, t \in B, v \in \pi(s,t)\}$,
with $w(S) = |A| + |B|$
$\mathcal{S}$ contains all possible $S(A, v, B)$

$\Rightarrow$ Greedy algorithm gets a $\mathcal{O}(\log n)$ approximation

Hub Labels

Approach 2: CH-based Hub Labels

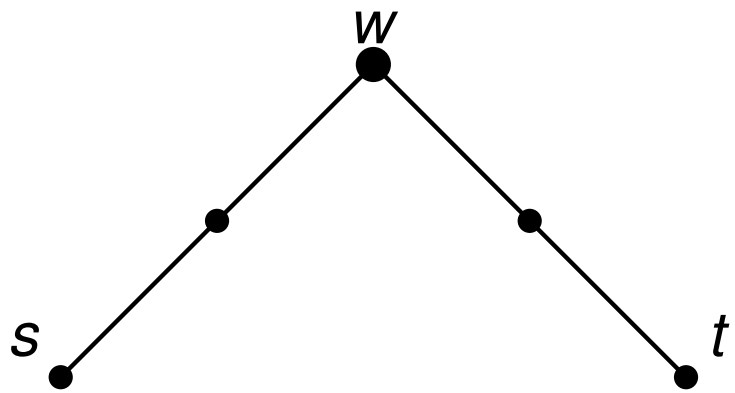
Computing CHs is very fast and memory efficient!

Hub Labels

Approach 2: CH-based Hub Labels

Computing CHs is very fast and memory efficient!

Observation: $H(v) = G^{\uparrow}(v)$ fulfills the cover property.

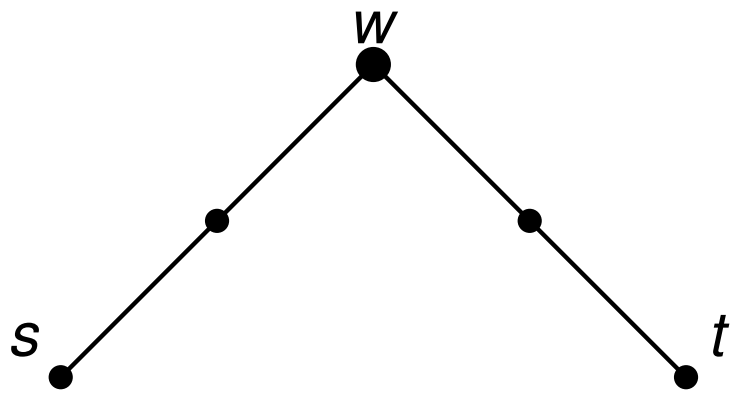


Hub Labels

Approach 2: CH-based Hub Labels

Computing CHs is very fast and memory efficient!

Observation: $H(v) = G^{\uparrow}(v)$ fulfills the cover property.



Note: not every in $G^{\uparrow}(v)$ is the peak node of some s-t path

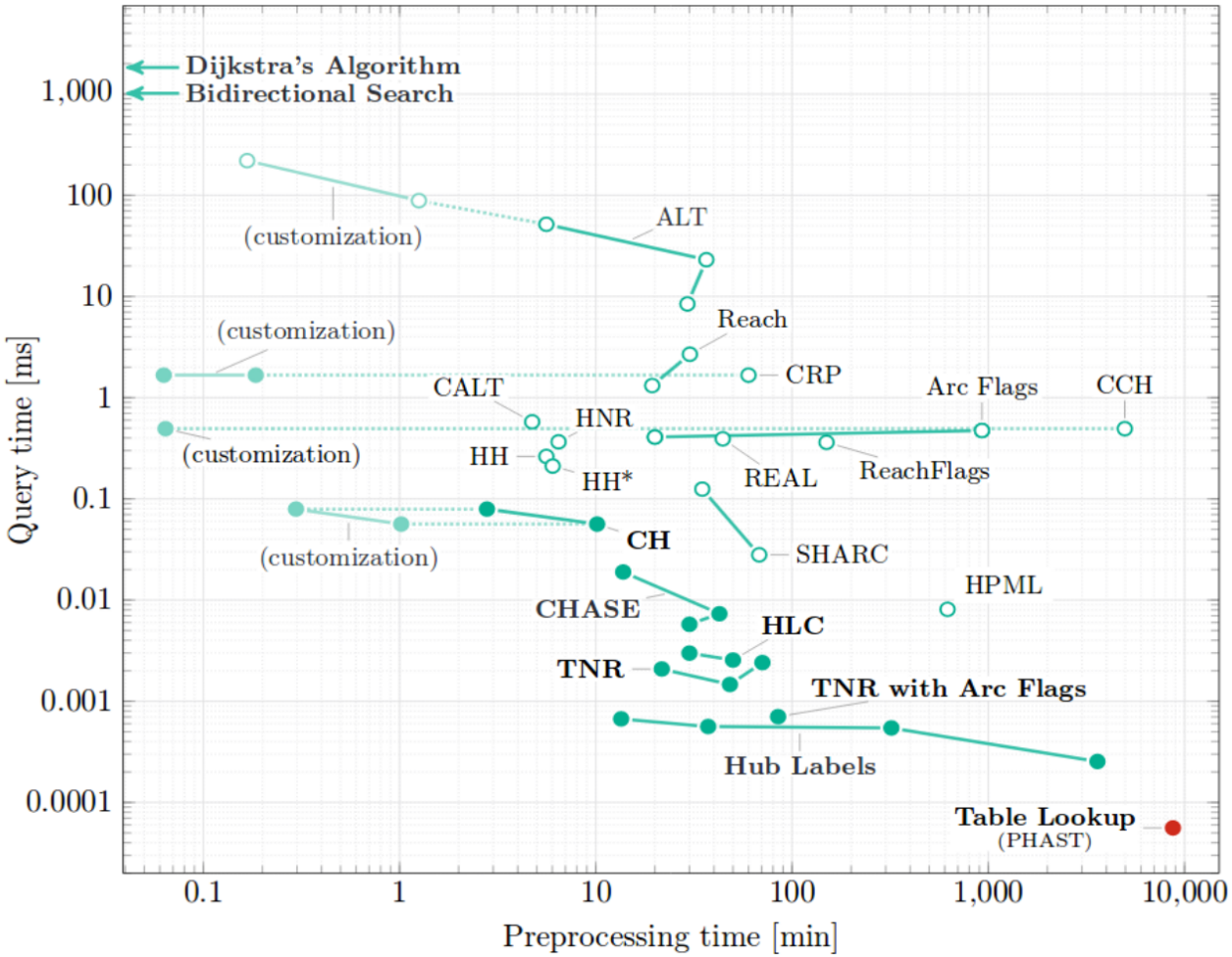
Hub Labels

Results: [ADGW11]

	Europe				USA			
	preprocessing		query		preprocessing		query	
	time	space	time [ns]		time	space	time [ns]	
method	[h:m]	[GB]	real	scaled	[h:m]	[GB]	real	scaled
CH	0:25	0.4	180000	93995	0:27	0.5	130000	67885
CHASE	1:39	0.6	17300	9034	3:48	0.7	19000	9922
HPML	≈ 24:00	3.0	18800	9817	≈ 24:00	5.1	19300	10078
TNR	1:52	3.7	3400	1775	1:30	5.4	3000	1566
TNR+AF	3:49	5.7	1900	992	2:37	6.3	1700	888
HL prefix	0:45	5.7	527	527	0:40	6.4	542	542
HL local	0:08	20.1	572	572	0:07	22.7	627	627
HL	0:14	21.3	276	276	0:18	25.4	266	266
Table Lookup	> 14:01	1 208 358.7	56	56	> 29:52	2 293 902.1	56	56

Related Topics

- One-to-Many, One-to-All Queries
- Edge cost functions
- Multicriteria route planning
- Multimodal route planning
- Public Transit Networks
- ...


Dijkstra's Algorithm
Bidirectional Search
(customization)
ALT
Reach
(customization)
CRP
CALT
Arc Flags
CCH
HNR
(customization)
HH
HH*
REAL
ReachFlags
(customization)
CH
SHARC
HPML
CHASE
HLC
TNR
TNR with Arc Flags
Hub Labels
Table Lookup
(PHAST)
Query time [ms]
Preprocessing time [min]
1,000
100
10
1
0.1
0.01
0.001
0.0001
0.1
1
10
100
1,000
10,000

Thank you!

References I

[ADGW11] Ittai Abraham, Daniel Delling, Andrew V Goldberg, and Renato F Werneck, *A hub-based labeling algorithm for shortest paths in road networks*, Experimental Algorithms: 10th International Symposium, SEA 2011, Kolimpari, Chania, Crete, Greece, May 5-7, 2011. Proceedings 10, Springer, 2011, pp. 230–241.

[Dij59] Edsger W Dijkstra, *A note on two problems in connexion with graphs*, Numerische mathematik **1** (1959), no. 1, 269–271.

References II

[FSS15] Stefan Funke, Robin Tibor Schirrmeister, and Sabine Storandt, *Automatic extrapolation of missing road network data in openstreetmap*, MUD@ICML, 2015.

[GH05] Andrew V Goldberg and Chris Harrelson, *Computing the shortest path: A search meets graph theory.*, SODA, vol. 5, 2005, pp. 156–165.

References III

[Gol07] Andrew V Goldberg, *Point-to-point shortest path algorithms with preprocessing*, International Conference on Current Trends in Theory and Practice of Computer Science, Springer, 2007, pp. 88–102.

References IV

[GSSD08] Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling, *Contraction hierarchies: Faster and simpler hierarchical routing in road networks*, Experimental Algorithms: 7th International Workshop, WEA 2008 Provincetown, MA, USA, May 30-June 1, 2008 Proceedings 7, Springer, 2008, pp. 319–333.

[GSSV12] Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter, *Exact routing in large road networks using contraction hierarchies*, Transportation Science **46** (2012), no. 3, 388–404.