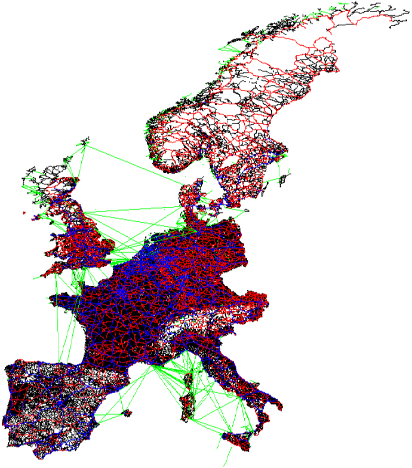
Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks

R. Geisberger P. Sanders D. Schultes D. Delling

7th International Workshop on Experimental Algorithms

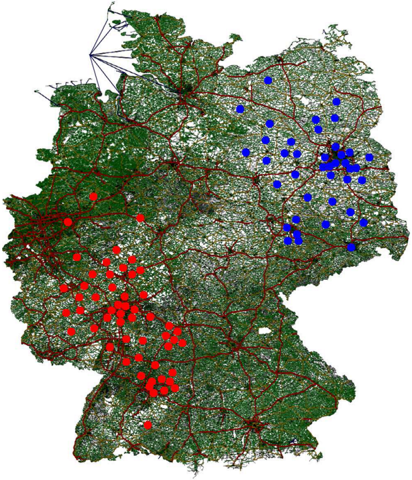
Motivation

- exact shortest paths calculation in large road networks
- minimize:
  - query time
  - preprocessing time
  - space consumption
- + simplicity



Mobile Navigation

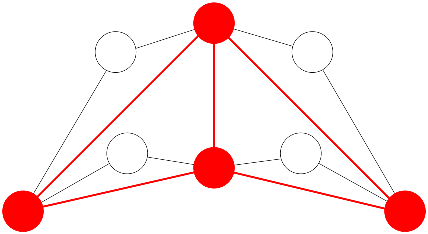
Logistics



Highway-Node Routing (HNR)

[Sanders and Schultes, WEA 07]

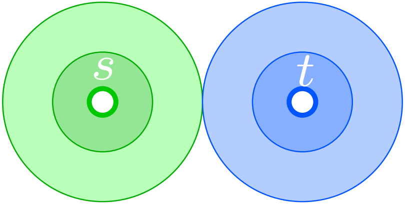
- general approach
- \+ adopt correctness proofs
- \- relies on another method to create hierarchies



Highway Hierarchies (HH)

[Sanders and Schultes, ESA 05, 06]

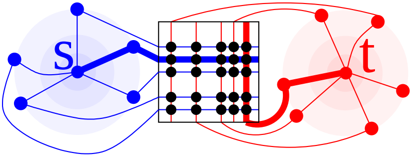
- two construction steps that are iteratively applied
  - removal of low degree nodes
  - removal of edges that only appear on shortest paths close to source or target
- Contraction Hierarchies (CH) are a radical simplification



Speedup Techniques

Transit-Node Routing (TNR) [BFSS07]

- fastest speedup technique known today
- higher preprocessing time

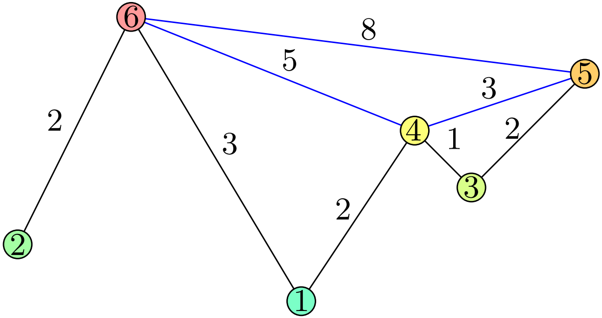


Goal-Directed Routing

- can be combined with hierarchical speedup techniques [DDSSSW08]

$\Rightarrow$ both techniques benefit from Contraction Hierarchies (CH)

Contraction Hierarchies (CH)



Main Idea

Contraction Hierarchies (CH)

- contract only one node at a time
  $\Rightarrow$ local and cache-efficient operation

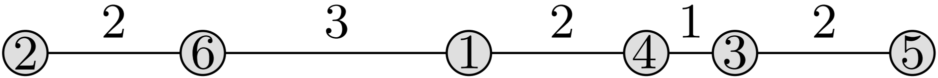
in more detail:

- order nodes by “importance”, $V = \{1, 2, \ldots, n\}$
- contract nodes in this order, node v is contracted by

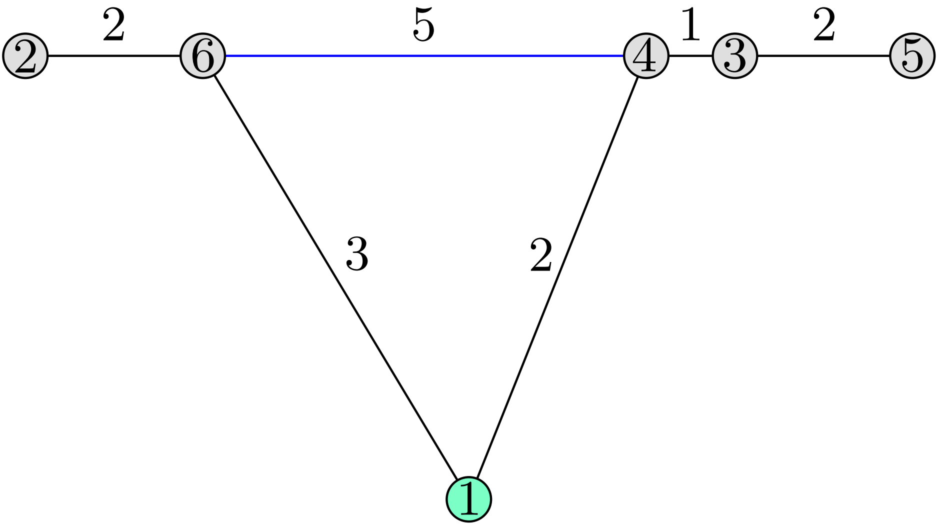
  foreach *pair* (u, v) *and* (v, w) *of edges* **do**
      if $\langle u, v, w\rangle$ *is a unique shortest path* **then**
          add shortcut (u, w) with weight $w(\langle u, v, w\rangle)$

- query relaxes only edges to more “important” nodes
  $\Rightarrow$ valid due to shortcuts

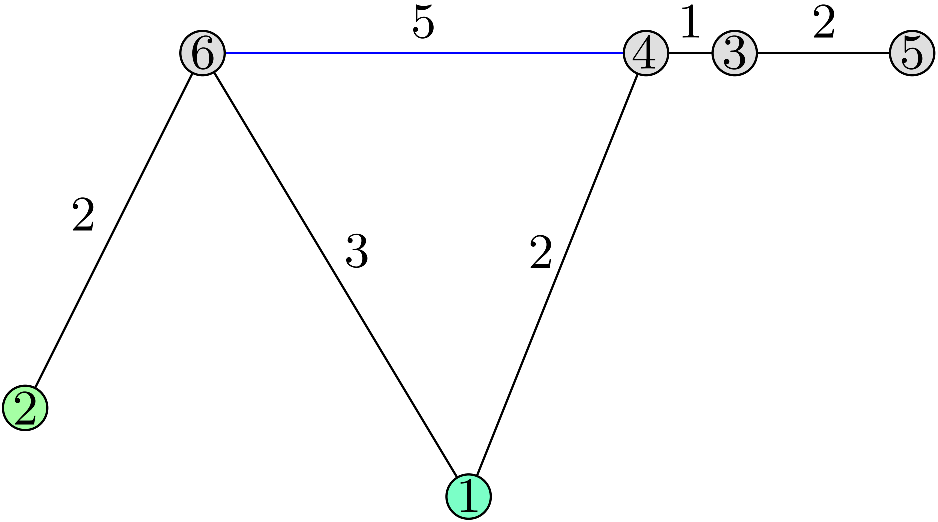
Example: Construction



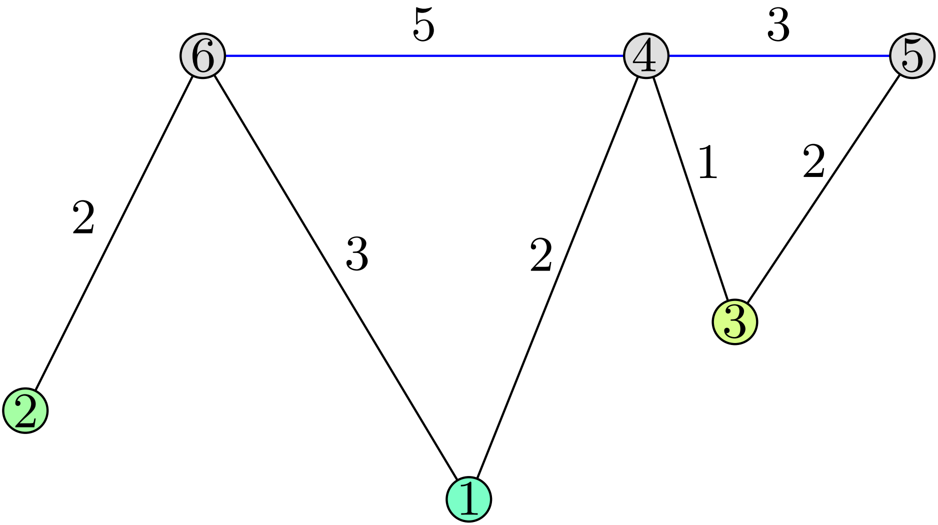
Example: Construction



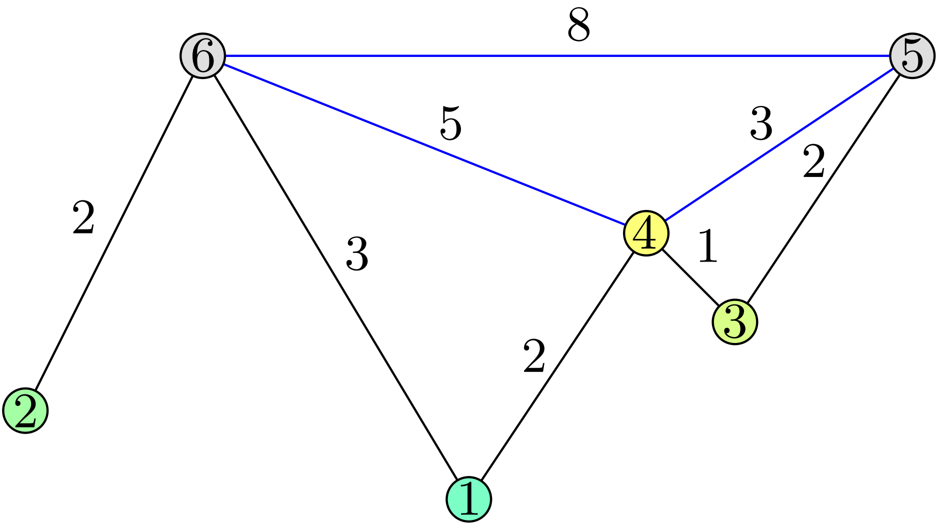
Example: Construction



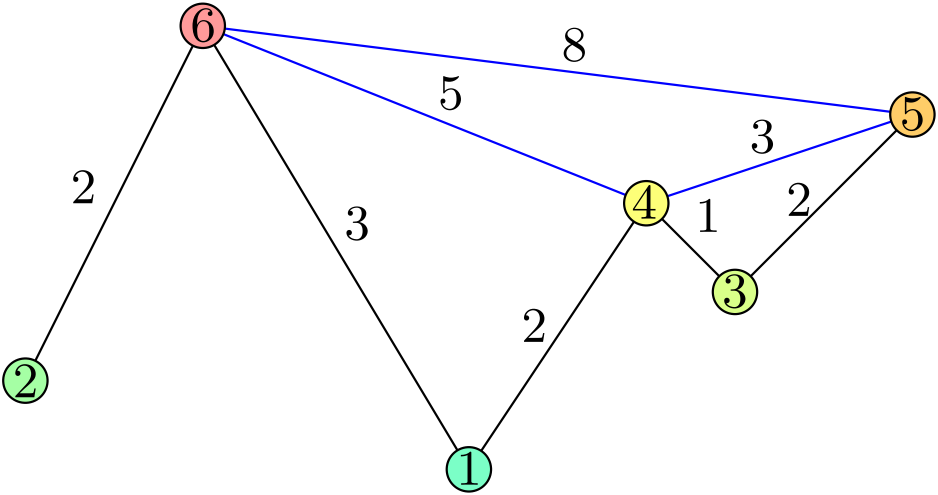
Example: Construction



Example: Construction



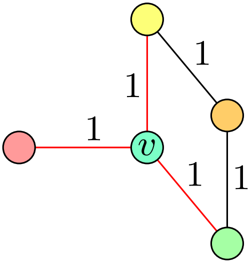
Example: Construction



Construction

to identify necessary shortcuts

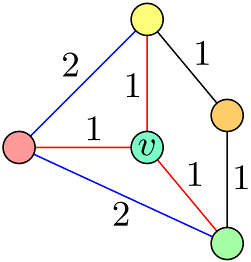
- local searches from all nodes u with incoming edge (u, v)
- ignore node v at search
- add shortcut (u, w) iff found distance $d(u, w) > w(u, v) + w(v, w)$



Construction

to identify necessary shortcuts

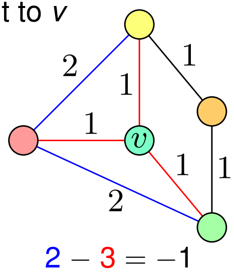
- local searches from all nodes u with incoming edge (u, v)
- ignore node v at search
- add shortcut (u, w) iff found distance $d(u, w) > w(u, v) + w(v, w)$



Node Order

use priority queue of nodes, node *v* is weighted with a linear combination of:

- **edge difference** #shortcuts – #edges incident to *v*
- **uniformity** e.g. #deleted neighbors
- …

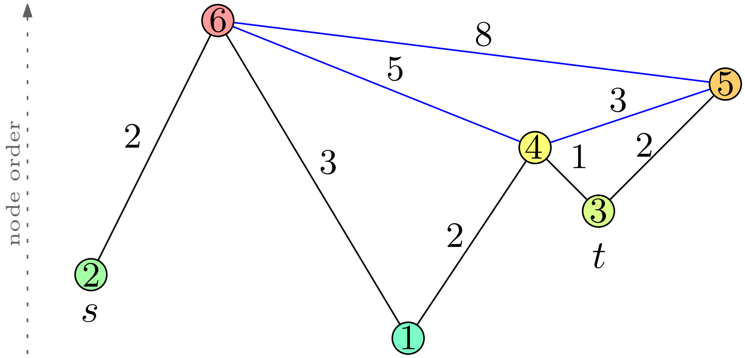


integrated construction and ordering:

1. remove node *v* on top of the priority queue
2. contract node *v*
3. update weights of remaining nodes

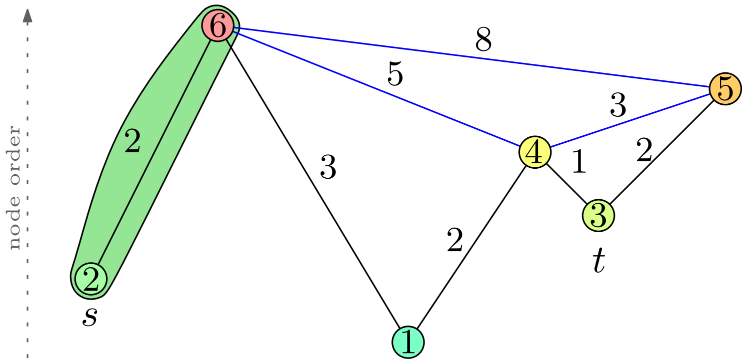
Query

- modified bidirectional Dijkstra algorithm
- upward graph $G_\uparrow := (V, E_\uparrow)$ with $E_\uparrow := \{(u, v) \in E : u < v\}$
  downward graph $G_\downarrow := (V, E_\downarrow)$ with $E_\downarrow := \{(u, v) \in E : u > v\}$
- forward search in $G_\uparrow$ and backward search in $G_\downarrow$



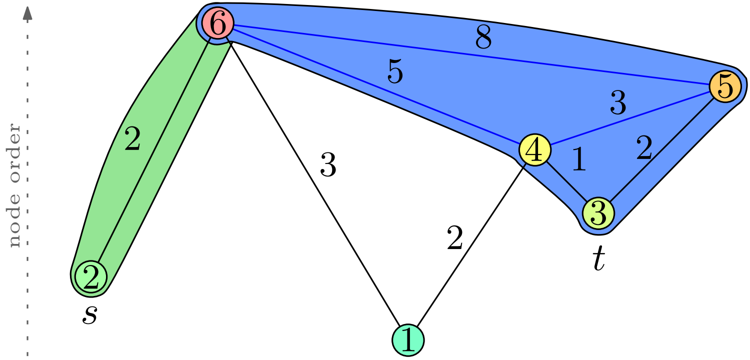
Query

- modified bidirectional Dijkstra algorithm
- upward graph $G_\uparrow := (V, E_\uparrow)$ with $E_\uparrow := \{(u,v) \in E : u < v\}$
  downward graph $G_\downarrow := (V, E_\downarrow)$ with $E_\downarrow := \{(u,v) \in E : u > v\}$
- forward search in $G_\uparrow$ and backward search in $G_\downarrow$



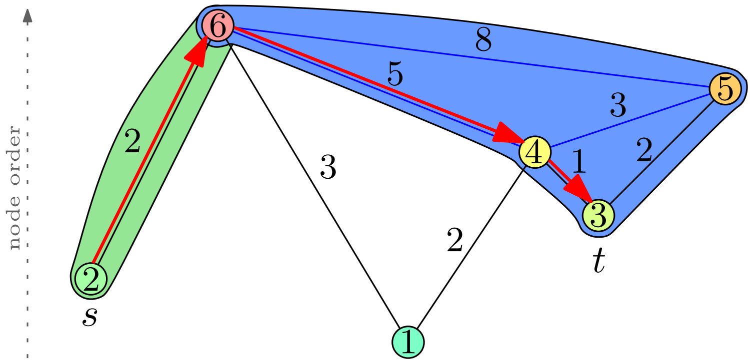
Query

- modified bidirectional Dijkstra algorithm
- upward graph $G_{\uparrow} := (V, E_{\uparrow})$ with $E_{\uparrow} := \{(u, v) \in E : u < v\}$
  downward graph $G_{\downarrow} := (V, E_{\downarrow})$ with $E_{\downarrow} := \{(u, v) \in E : u > v\}$
- forward search in $G_{\uparrow}$ and backward search in $G_{\downarrow}$



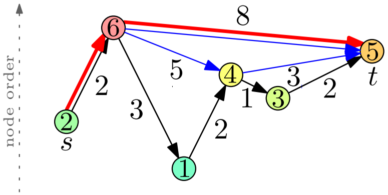
Query

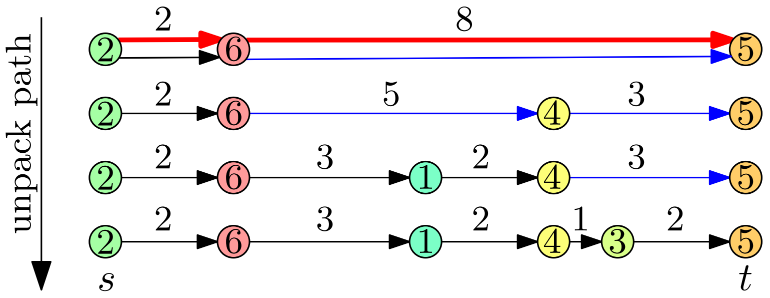
- modified bidirectional Dijkstra algorithm
- upward graph $G_\uparrow := (V, E_\uparrow)$ with $E_\uparrow := \{(u, v) \in E : u < v\}$
  downward graph $G_\downarrow := (V, E_\downarrow)$ with $E_\downarrow := \{(u, v) \in E : u > v\}$
- forward search in $G_\uparrow$ and backward search in $G_\downarrow$



Outputting Paths

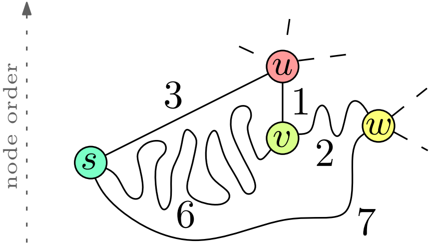
- for a shortcut (u, w) of a path $\langle u, v, w \rangle$, store middle node v with the edge
- expand path by recursively replacing a shortcut with its originating edges





Stall-on-Demand

- v can be "stalled" by u (if $d(u) + w(u, v) < d(v)$)
- stalling can propagate to adjacent nodes
- search is not continued from stalled nodes



- does not invalidate correctness (only suboptimal paths are stalled)

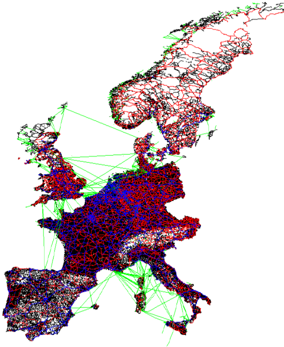
Experiments

environment

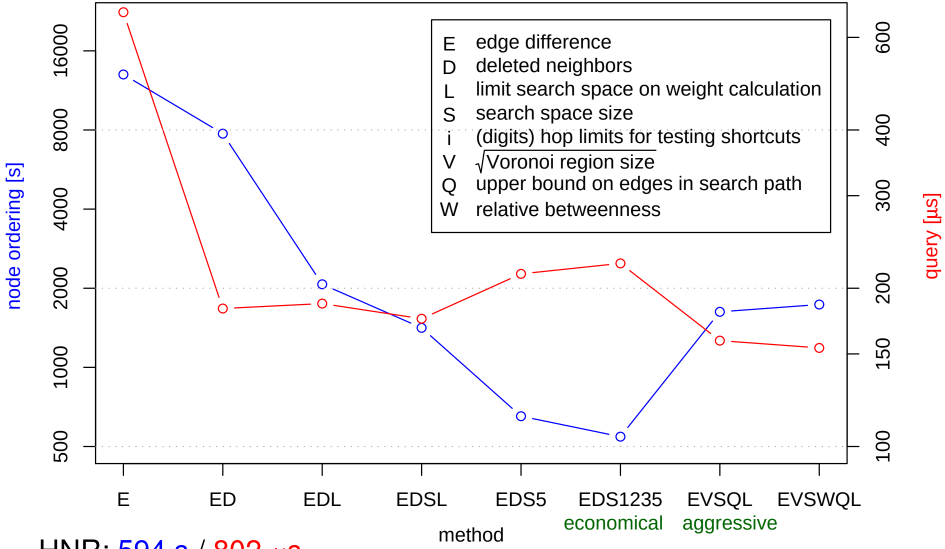
- AMD Opteron Processor 270 at 2.0 GHz
- 8 GB main memory
- GNU C++ compiler 4.2.1

test instance

- road network of Western Europe (PTV)
- 18 029 721 nodes
- 42 199 587 directed edges

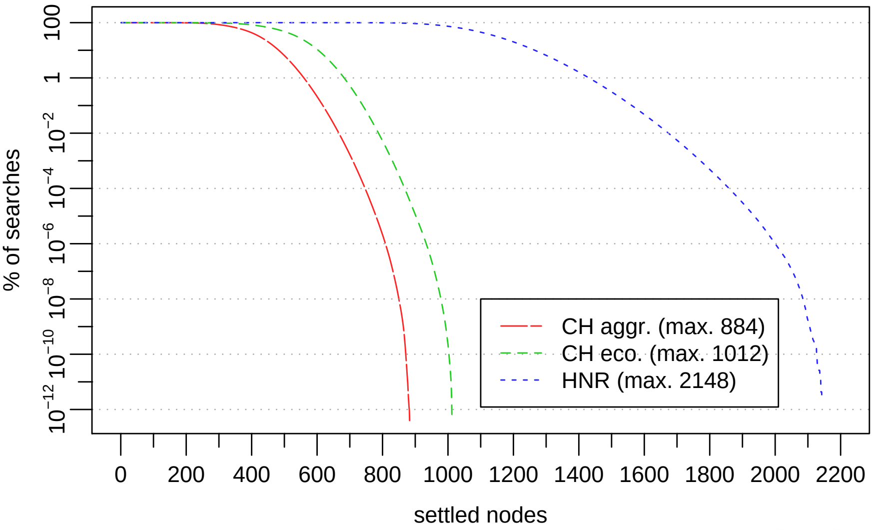


Performance



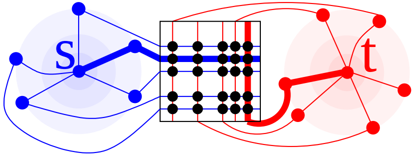
HNR: 594 s / 802 μs

Worst Case Costs



Additional Results

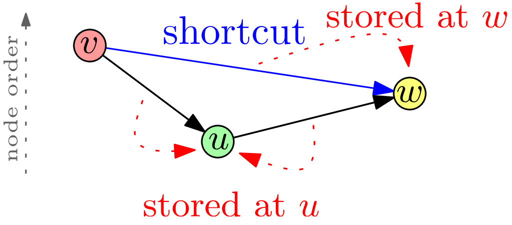
space overhead	
HNR	9.5 B/node
CH economical	0.6 B/node
CH aggressive	-2.7 B/node
Many-to-Many Shortest Paths [KSSSW07]	
$10\,000 \times 10\,000$ table	$23.2 \rightarrow 10.2$ s
Transit Node Routing [BFSS07]	
query time	$4.3 \rightarrow 3.4\ \mu$s



Graph Representation

search graph

- usually store edge (v, w) in the adjacency array of v and w
- for the search, we need to store it only at node $\min\{v, w\}$
- possibly negative space overhead



Summary

- Contraction Hierarchies are simple
- less space overhead
- 5× faster queries than the best previous hierarchical Dijkstra-based speedup techniques
- new foundation for other routing algorithms like Transit-Node Routing

- Future work
  - test other priority terms
  - time-dependent routing
  - dynamization

$$f(x) =$$