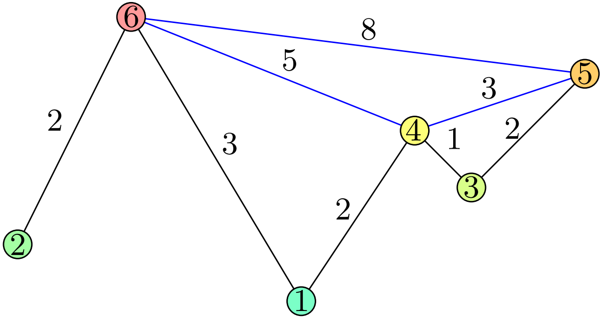
# 2) Multi-Criteria
# 1) Contraction Hierarchies
# 3) for Ride Sharing

Robert Geisberger

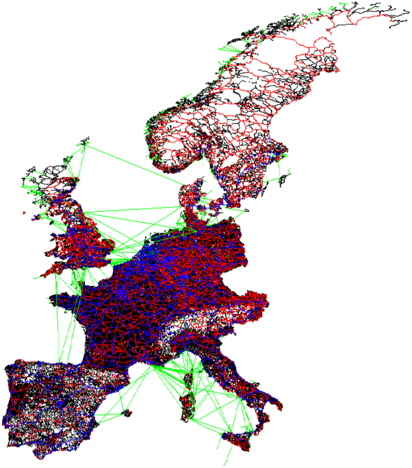
Bad Herrenalb, 4th and 5th December 2008

# Contraction Hierarchies (CH)



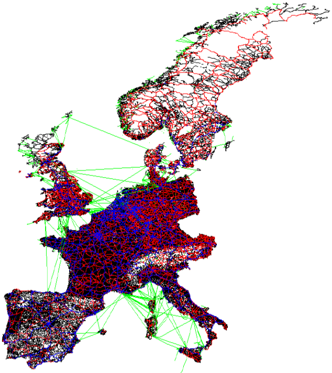
# Motivation

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



# Hierarchy

- find the hierarchy
- the more hierarchy is available the more you can find
- exploit it to speedup your algorithm



# Main Idea

**Contraction Hierarchies (CH)**

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

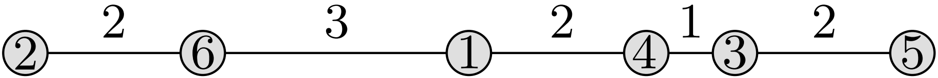
in more detail:

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

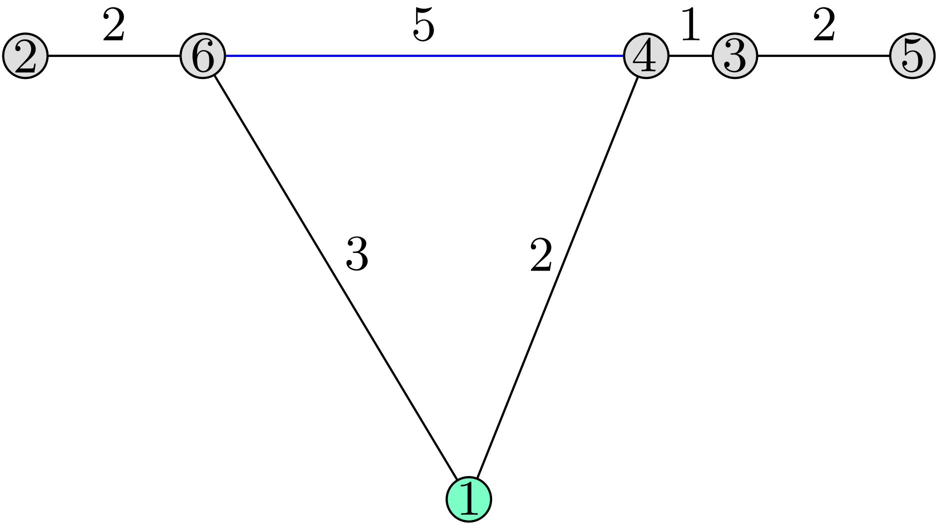
  **foreach** *pair* $(u, v)$ *and* $(v, w)$ *of edges* **do**
      **if** $\langle u, v, w \rangle$ *is a unique shortest path* **then**
          add shortcut $(u, w)$ with weight $w(\langle u, v, w \rangle)$

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

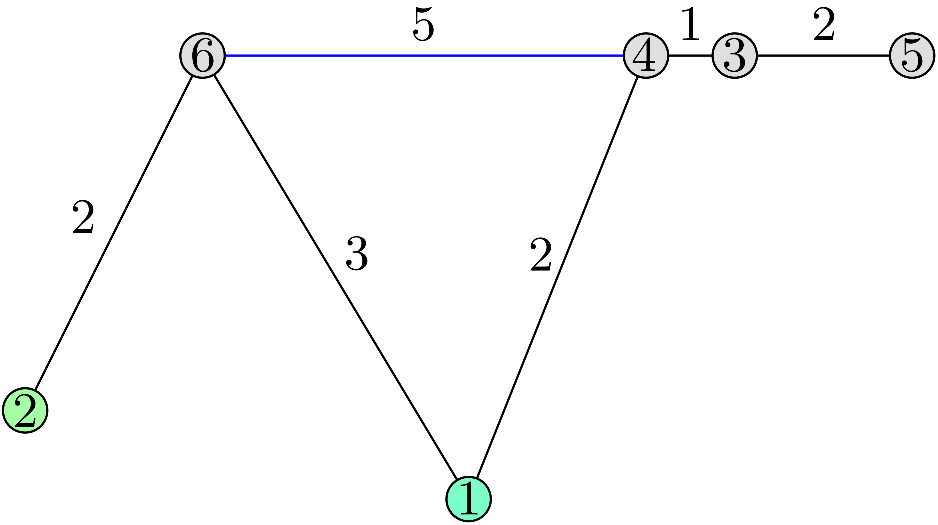
# Example: Construction



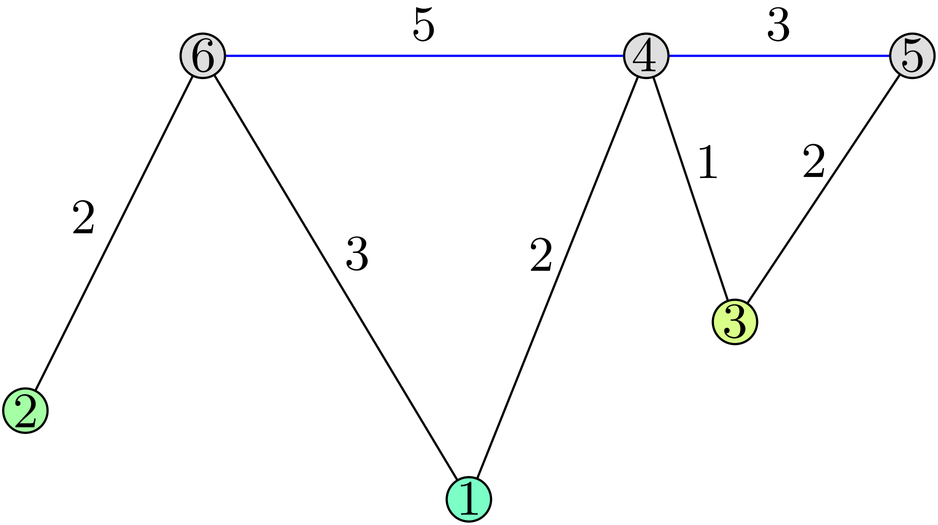
# Example: Construction



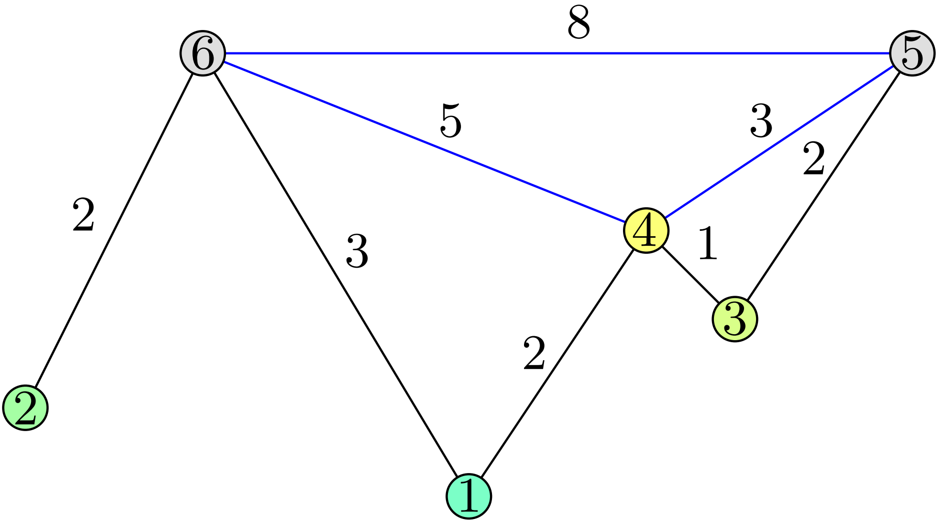
# Example: Construction



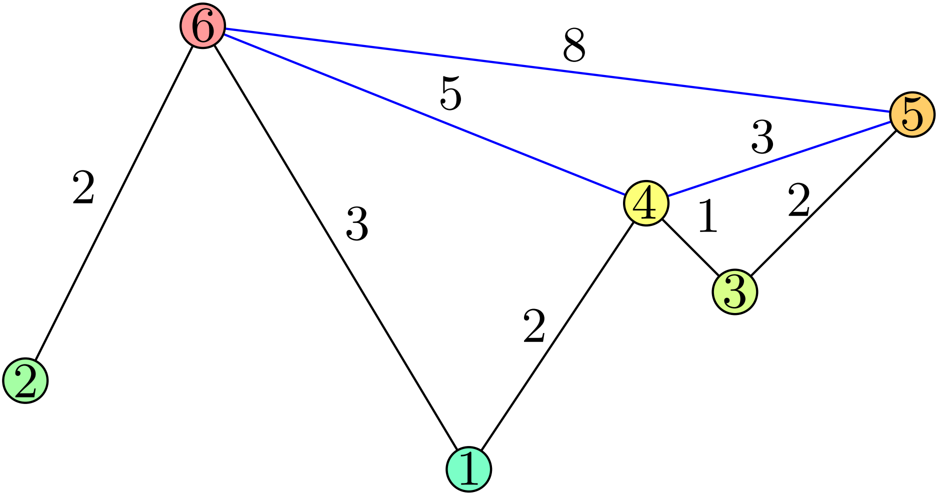
# Example: Construction



# Example: Construction



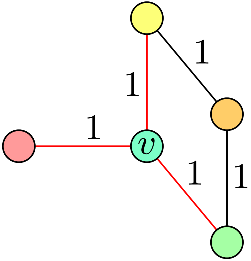
# Example: Construction



# Construction

to identify necessary shortcuts

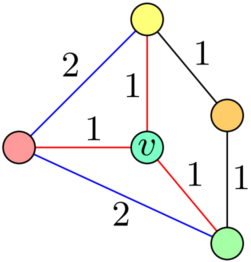
- local searches from all nodes $u$ with incoming edge $(u, v)$
- ignore node $v$ at search
- add shortcut $(u, w)$ iff found distance $d(u, w) > w(u, v) + w(v, w)$



# Construction

to identify necessary shortcuts

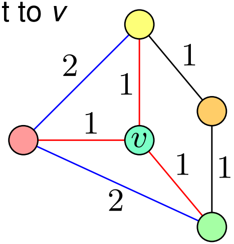
- local searches from all nodes $u$ with incoming edge $(u, v)$
- ignore node $v$ at search
- add shortcut $(u, w)$ iff found distance $d(u, w) > w(u, v) + w(v, w)$



# 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
- …



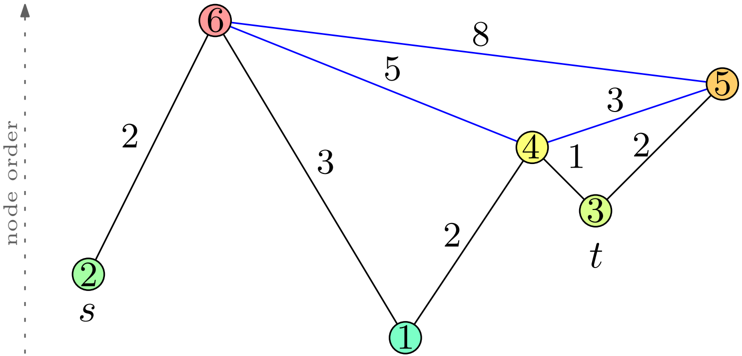

$2 - 3 = -1$

integrated construction and ordering:

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

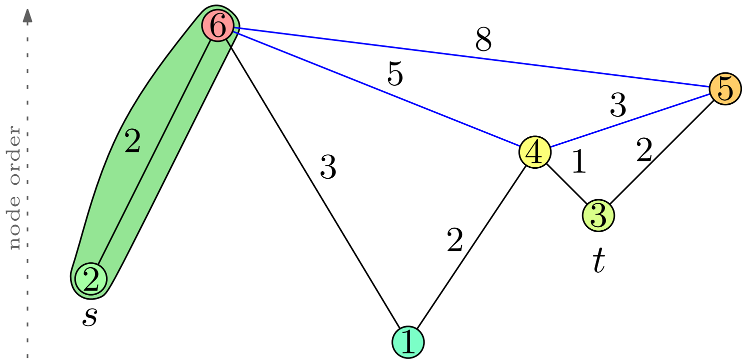
# Query

- 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 $G_\downarrow$



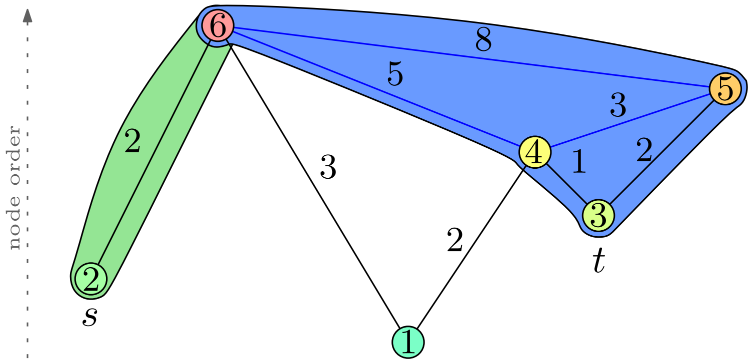
# Query

- 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 $G_\downarrow$



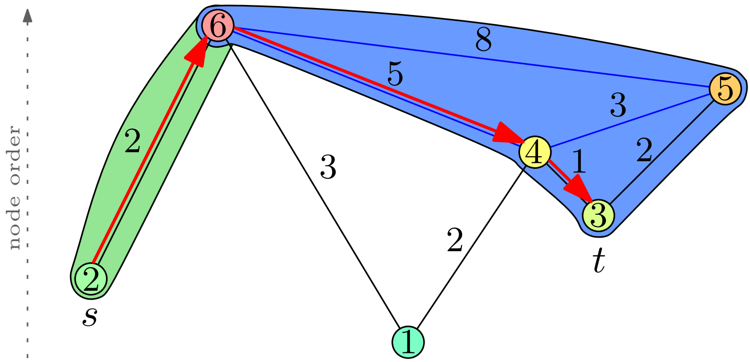
# Query

- 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 $G_\downarrow$



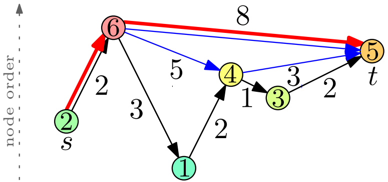
# Query

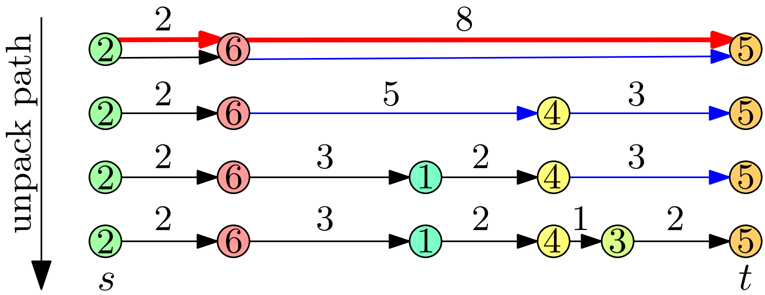
- modified bidirectional Dijkstra algorithm
- upward graph $G_\uparrow := (V, E_\uparrow)$ with $E_\uparrow := \{(u, v) \in E : u < v\}$
  downward graph $G_\downarrow := (V, E_\downarrow)$ with $E_\downarrow := \{(u, v) \in E : u > v\}$
- forward search in $G_\uparrow$ and backward search in $G_\downarrow$



# Outputting Paths

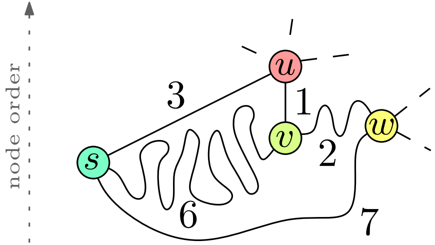
- for a shortcut $(u, w)$ of a path $\langle u, v, w \rangle$, store middle node $v$ with the edge
- expand path by recursively replacing a shortcut with its originating edges



# 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)

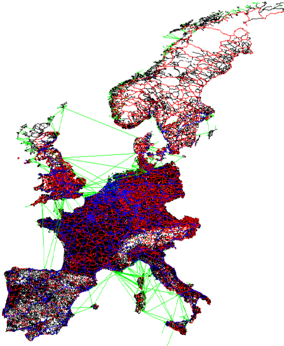
# Experiments

**environment**

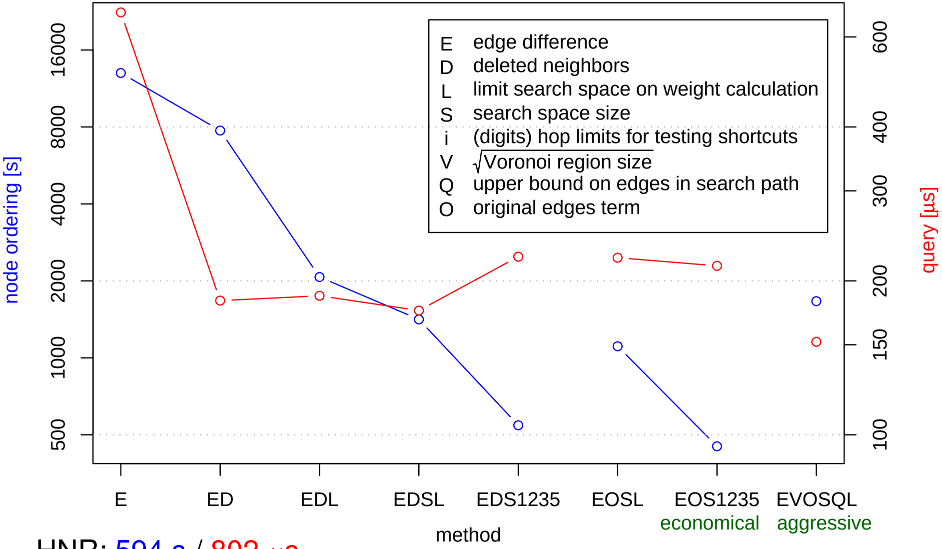
- AMD Opteron Processor 270 at 2.0 GHz
- 8 GB main memory
- GNU C++ compiler 4.2.1

**test instance**

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

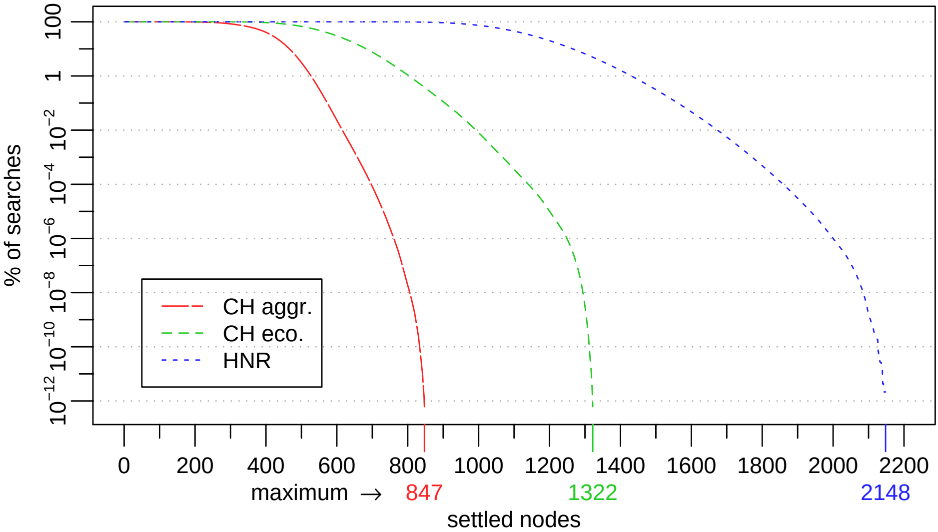


# Performance




HNR: 594 s / 802 $\mu$s

# Worst Case Costs



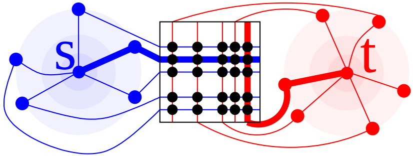
# Hierarchy is not hierarchy

**Many-to-Many Shortest Paths** [KSSSW07]

| method | query [$\mu$s] | settled nodes | non-stalled nodes | $10\,000 \times 10\,000$ table |
|---|---|---|---|---|
| EVSQL | 159 | 368 | 209 | 10.2 s |
| EVOSQL | 152 | 356 | 207 | 11.0 s |

**Transit Node Routing** [BFSS07]

preprocessing time with method EDOSQ1235 46 min
query time still at 3.3 $\mu$s



# Summary

- Contraction Hierarchies are simple and fast
- 7.5 min preprocessing results in 0.21 ms queries
- foundation for other methods
- definition of "best" hierarchy selection depends on application

# Part II

# Multi-Criteria Contraction Hierarchies

# Feasibility Study of Young Scientist (FYS)

- Do you have a very good master-/diploma-/phd-thesis?
- Is there a interesting question left?
- Then they may give you money.

# Goals

- multiple optimization criterias
  e.g.: distance, time, costs
- flexibility at route calculation time
  e.g.: individual vehicle speeds
- diversity of results
  e.g.: calculate Pareto-optimal results
- roundtrips with scenic value
  e.g.: for tourists

# Challenges and New Insight

- Every optimization criterion has a specific influence on the hierarchy of a road network.
  e.g.: Finding the fastest route contains more hierarchy than finding the shortest route.
- **Challenge:** Multiple criteria interfere with hierarchy, but the algorithm should work fast on large graphs.
  e.g.: Motorways drop in the hierarchy because of road tolls.
- **New insight:** Combinations und weightings of optimization criteria that preserve hierarchy (and which not).

# Algorithmic Ideas

- modifiy the contraction so the query stays simple
- add all necessary shortcuts during contraction
- do this by modifying the local search
  - linear combination of two: $x + ay$ with $a \in [l, u]$
    label is now a function of $x$ (see timedependent CH)
  - linear combination of more: $a_1 x_1 + \cdots + a_n x_n$ with $a_i \in [l_i, u_i]$
  - Pareto-optimal (may add too many shortcuts)
- let us see what works best ;-)

Part III

# Fahrtenfinder

# Ride Sharing

**Current approaches:**

- match only ride offers with identical start/destination (perfect fit)
- sometimes radial search around start/destination

**Our approach:**

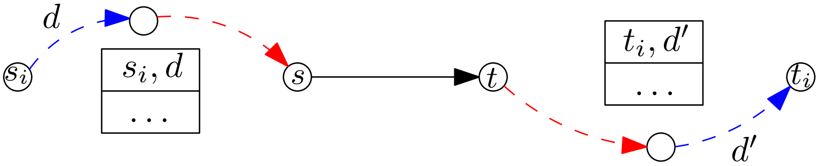
- driver picks passenger up and gives him a ride to his destination
- find the driver with the minimal detour (reasonable fit)

**Efficient algorithm:**

- adaption of the many-to-many algorithm

# Efficient Algorithm

- store forward search space from each start node $s_i$ in a bucket
- request from $s$ to $t$ needs to calculate all distances $d(s_i, s)$
- scan buckets of all reached nodes
- analogously for distances $d(t, t_i)$



# Experiments

- matching a request is really fast $\approx 25\,\text{ms}$
- it can sort by detour and output e.g. the best ten offers
- departure windows as selection criterion
- online adding/removal of offers supported $\approx 0.2\,/\,2\,\text{ms}$