Karlsruhe Institute of Technology

Route Planning in Transportation Networks – When the Metric Matters

EuroCG'16, Lugano, Switzerland

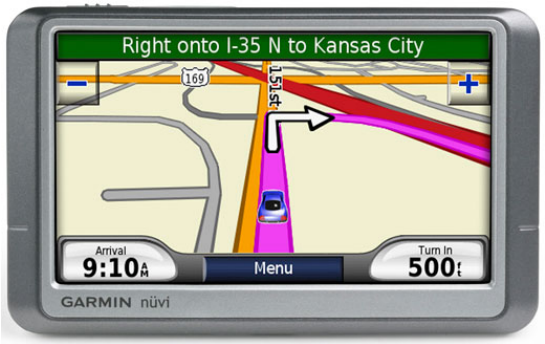
Prof. Dr. Dorothea Wagner | March 31, 2016

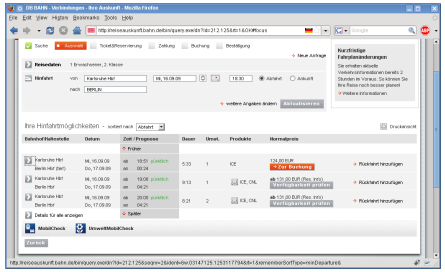
KARLSRUHE INSTITUTE OF TECHNOLOGY – INSTITUTE OF THEORETICAL INFORMATICS – GROUP ALGORITHMICS

Motivation

Important application

- Navigation systems for cars
- Google Maps, Bing Maps, . . .
- Timetable information





Fundamental research on

- Clean mathematical models
- With provable quality guarantees
- Rigorous performance evaluation

Problem Setting

Query

- Find the quickest journey in a transportation network between source and target

Modeling

- Network as graph $G = (V, E)$
- Edge weights are travel times
- Shortest path in $G \equiv$ quickest journey
- Classic problem (Dijkstra's algorithm)

Challenge

- Transport networks are huge
- Dijkstra's algorithm too slow ($>$ 1 second)

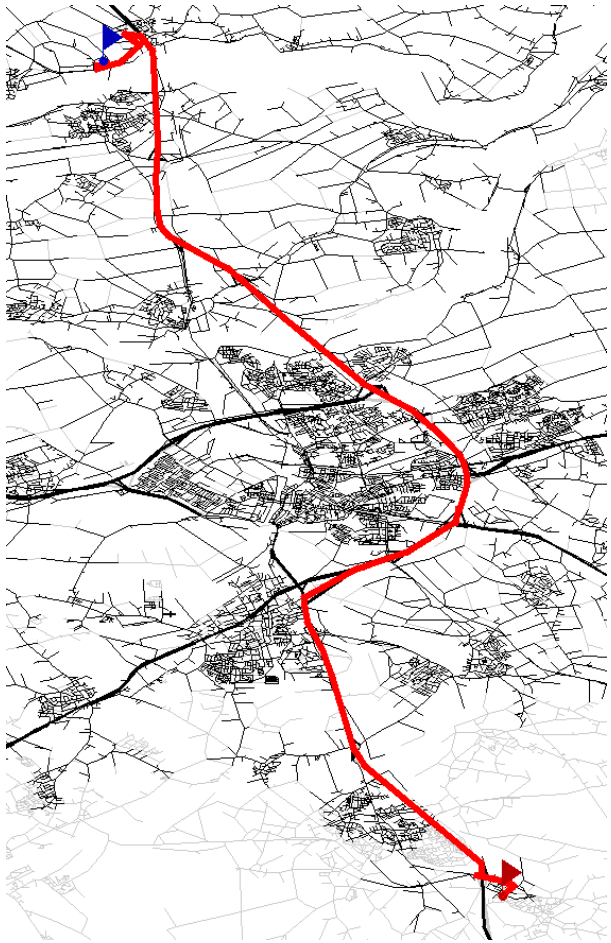
Speedup Techniques

Observation

- Dijkstra visits all nodes closer than the target
- Bidirectional search does not help much

$\Rightarrow$ Unnecessary computations

- Many requests in a hardly changing network



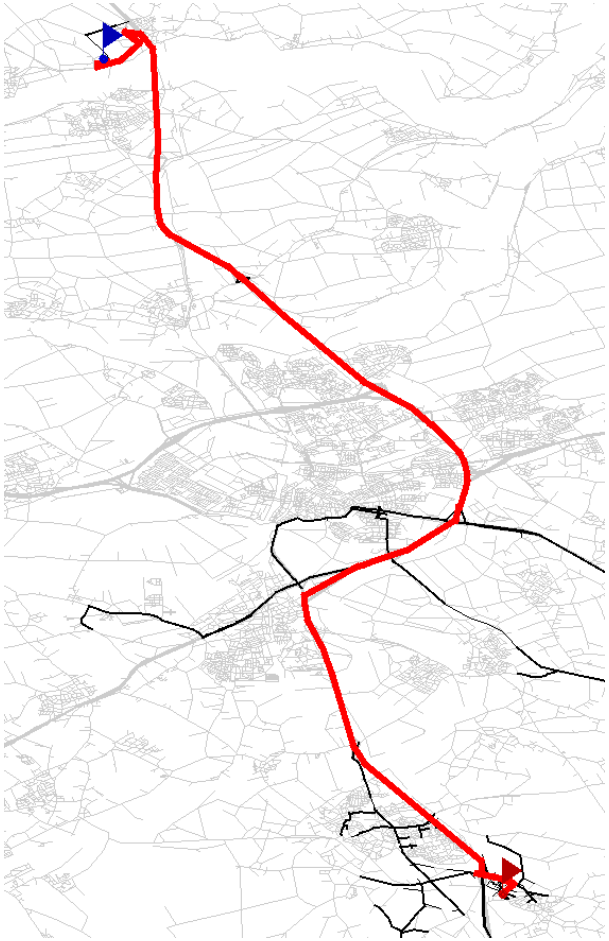
Speedup Techniques

Observation

- Dijkstra visits all nodes closer than the target
- Bidirectional search does not help much

$\Rightarrow$ Unnecessary computations

- Many requests in a hardly changing network

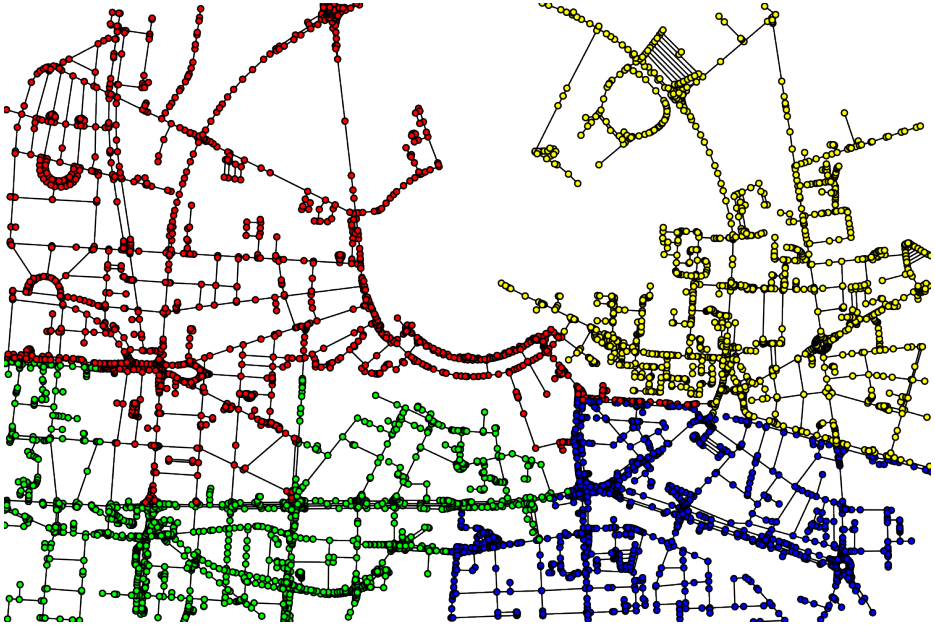


Idea

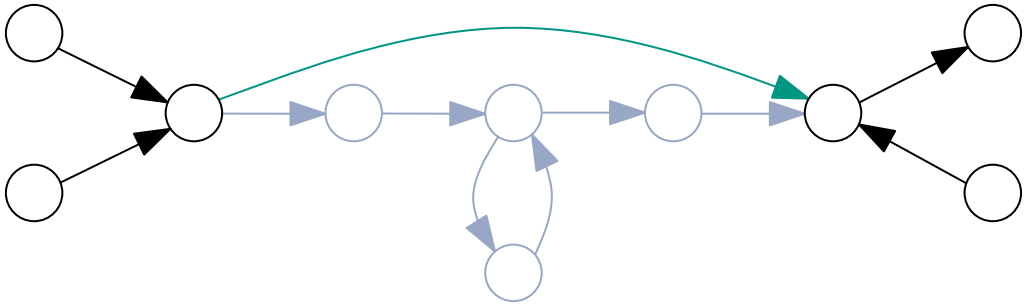
- Two-phase algorithm:
  - Offline: Preprocessing of additional data
  - Online: Speed up subsequent queries with this data
- Aspects:
  - Preprocessing time and space
  - Speedup over Dijkstra
  - Ease of implementation

Some Ideas

- Partition Network

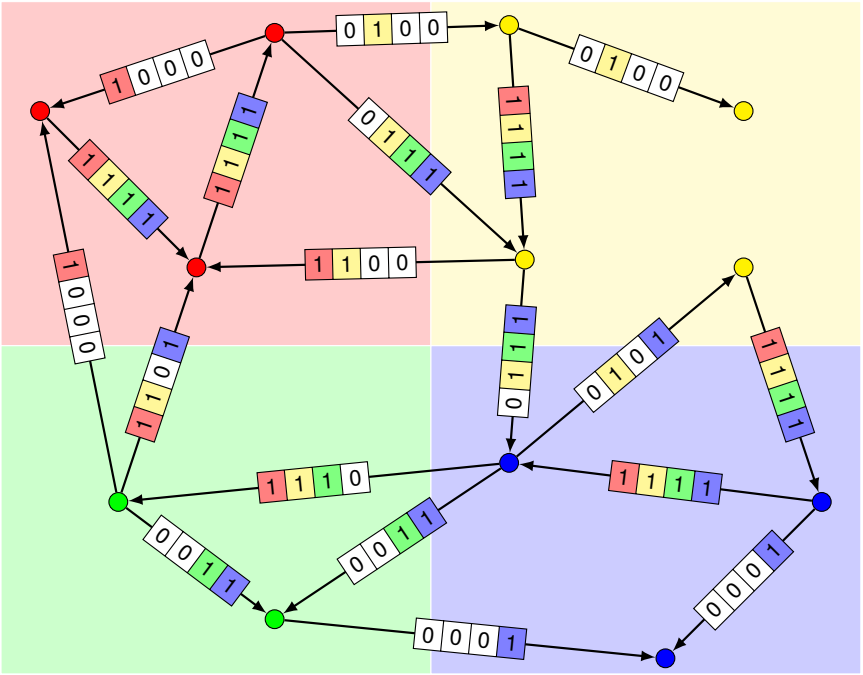


- Shortcuts

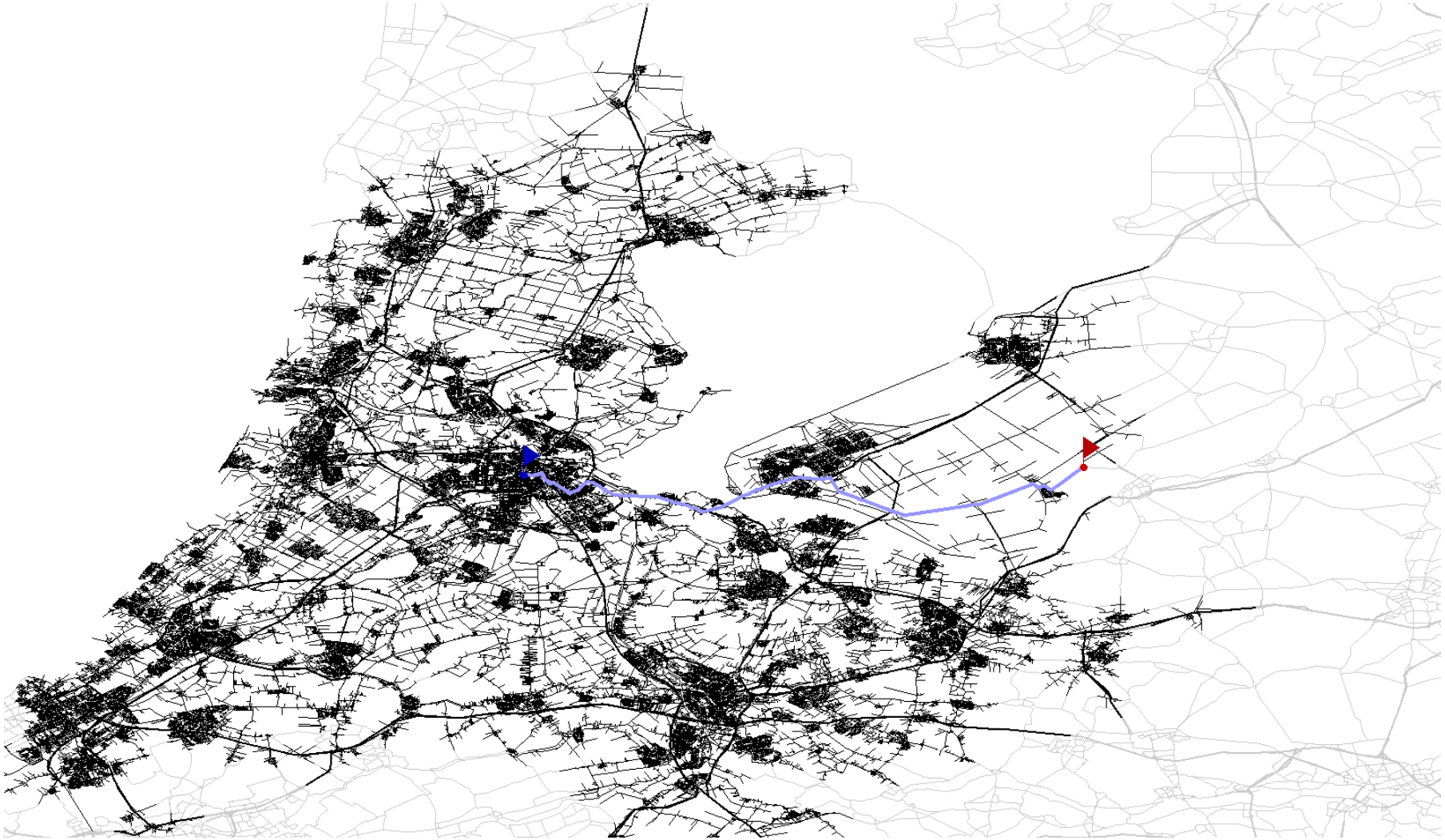


Arc-Flags [Lau04, BD09]

- Partition network into k cells
- k-bit vector for each edge
- indicates if edge required for shortest path into cell

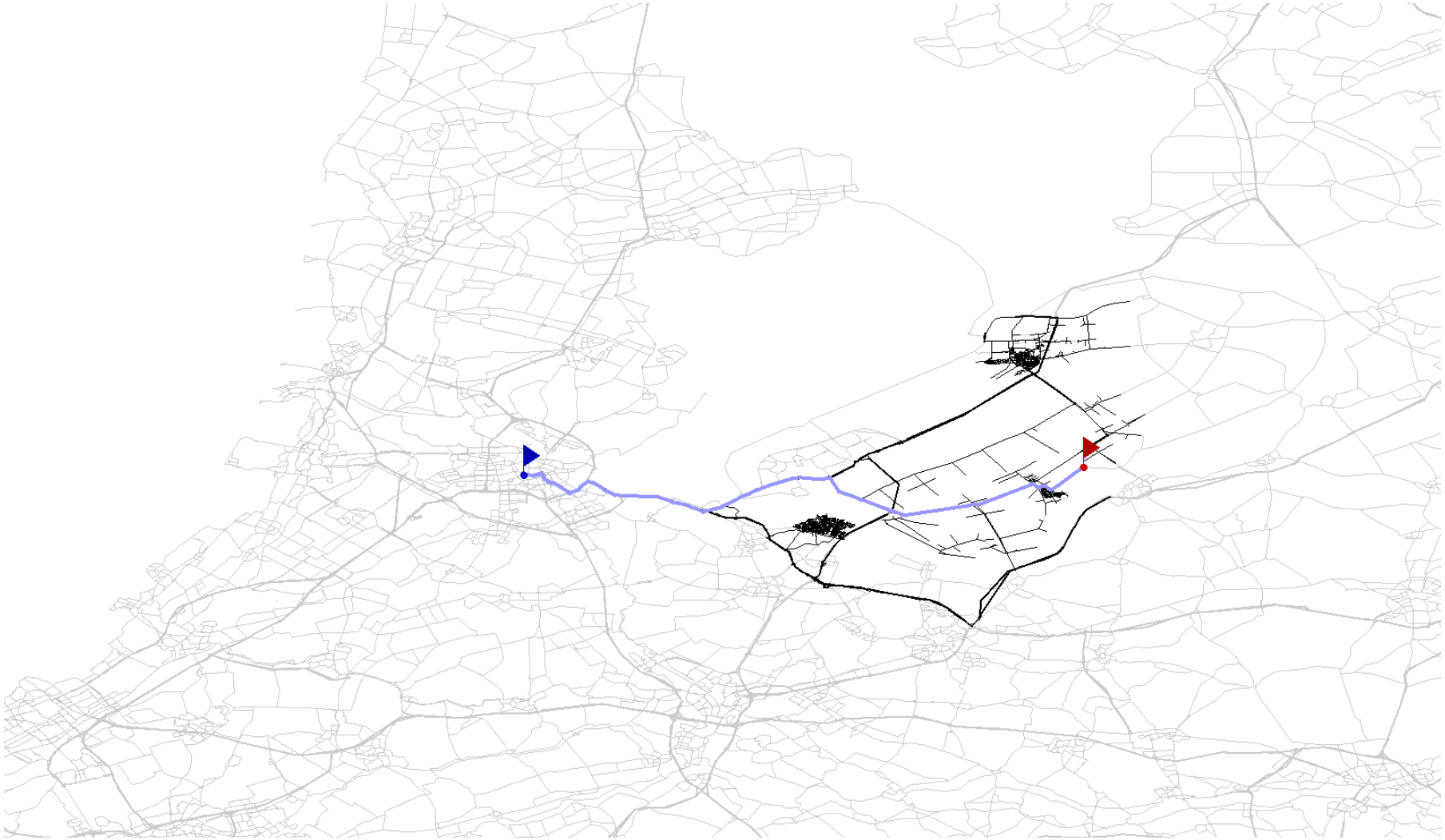


Search Space - Dijkstra's Algorithm



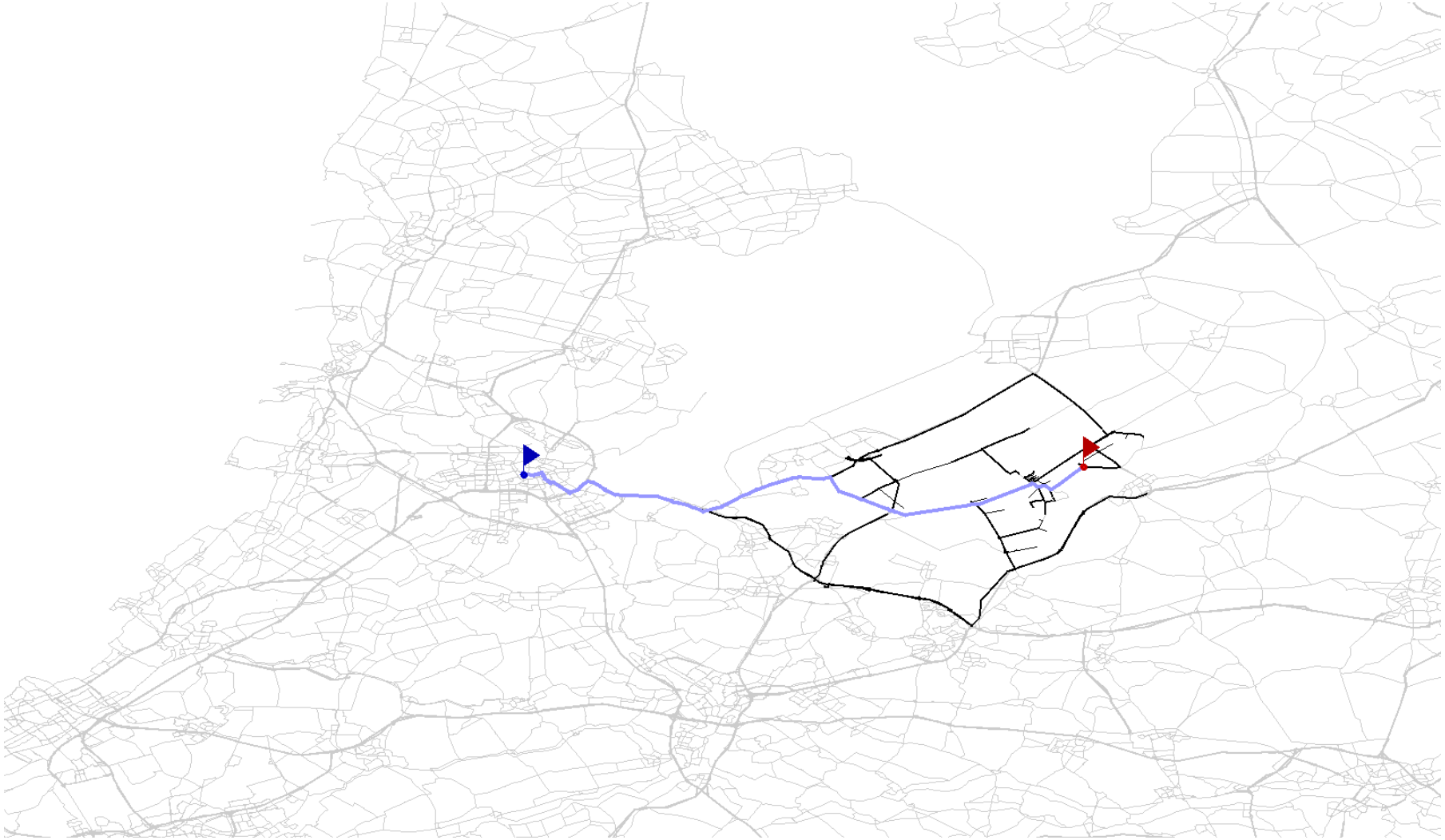
Institute for Theoretical Informatics
Chair Algorithmics

Search Space - Arc-Flags



Institute for Theoretical Informatics
Chair Algorithmics

Search Space - SHARC

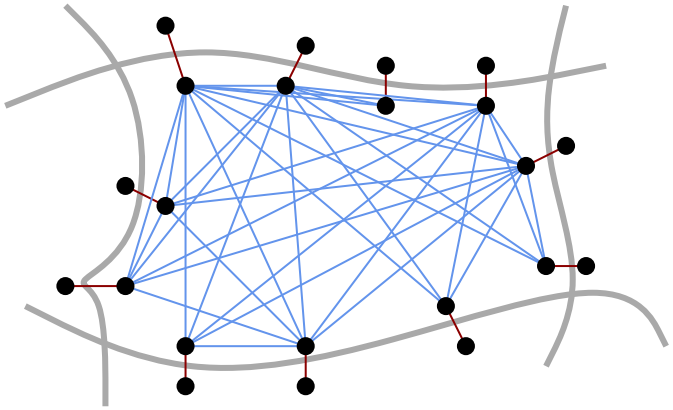


Institute for Theoretical Informatics
Chair Algorithmics

Multilevel Overlays [SWW00, HSW08]

Observation: Many paths share long subpaths

Idea: Precompute partial solutions

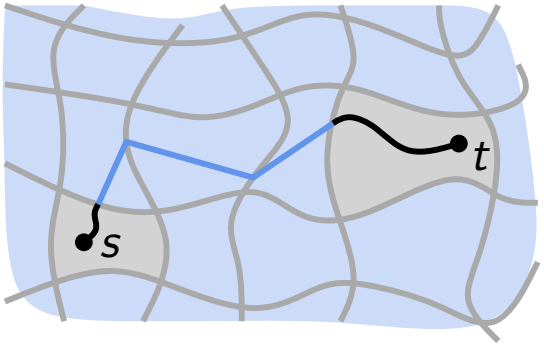


Overlay graph

- Select important nodes (separators, path coverage, heuristic)
- Compute shortcut edges:
  - Skip unimportant nodes
  - Conserve distances between important nodes

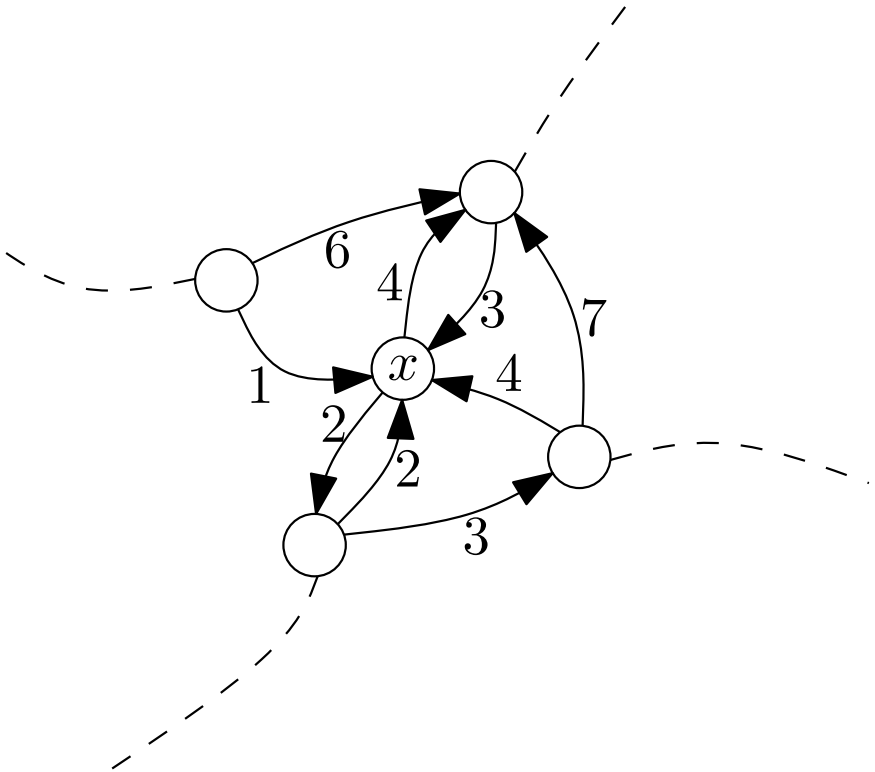
Queries

- Multilevel Dijkstra variant
- Ignore edges towards less important nodes



Contraction Hierarchies [GSSD08, GSSV12]

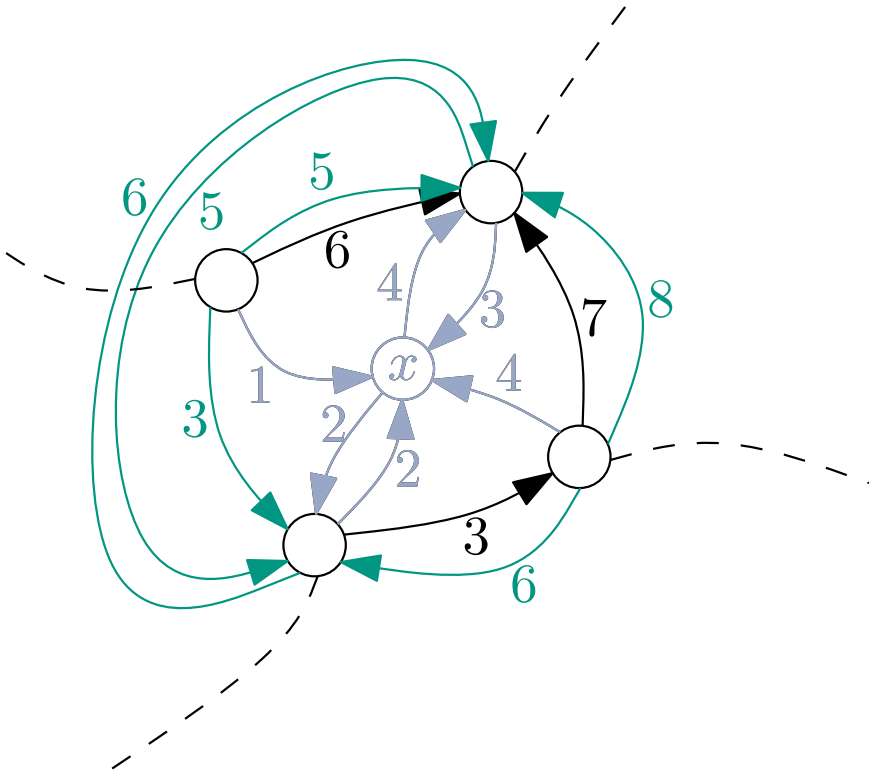
Idea: Compute shortcuts by iteratively contracting nodes



Contraction of x:
Remove x, add shortcuts among neighbors to maintain distances

Contraction Hierarchies [GSSD08, GSSV12]

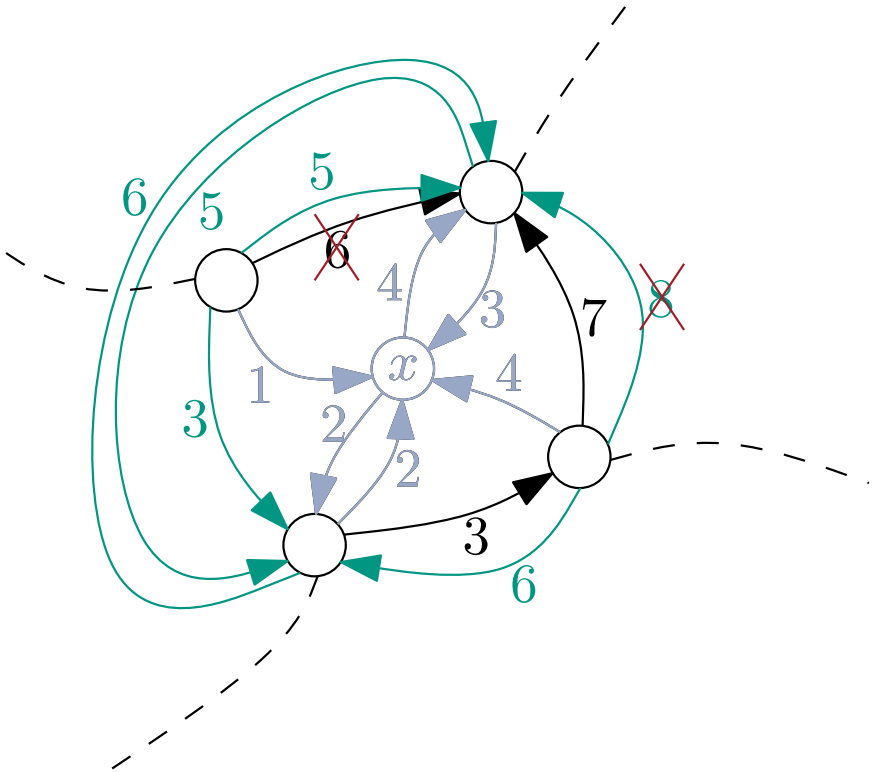
Idea: Compute shortcuts by iteratively contracting nodes



Contraction of x:
Remove x, add shortcuts among neighbors to maintain distances

Contraction Hierarchies [GSSD08, GSSV12]

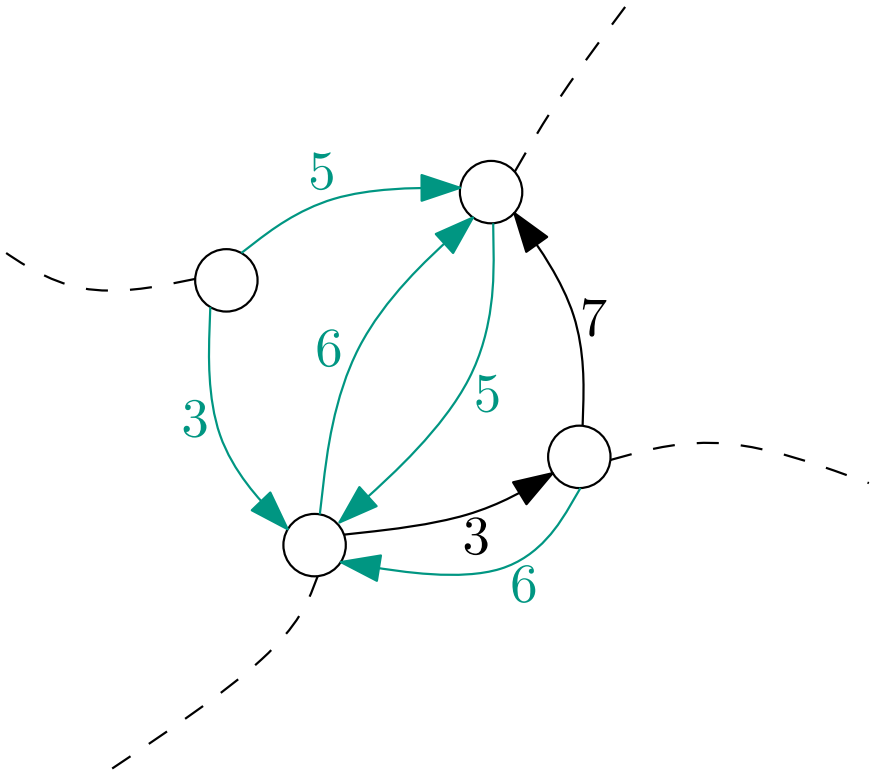
Idea: Compute shortcuts by iteratively contracting nodes



Delete longer edge in case of multi-edges

Contraction Hierarchies [GSSD08, GSSV12]

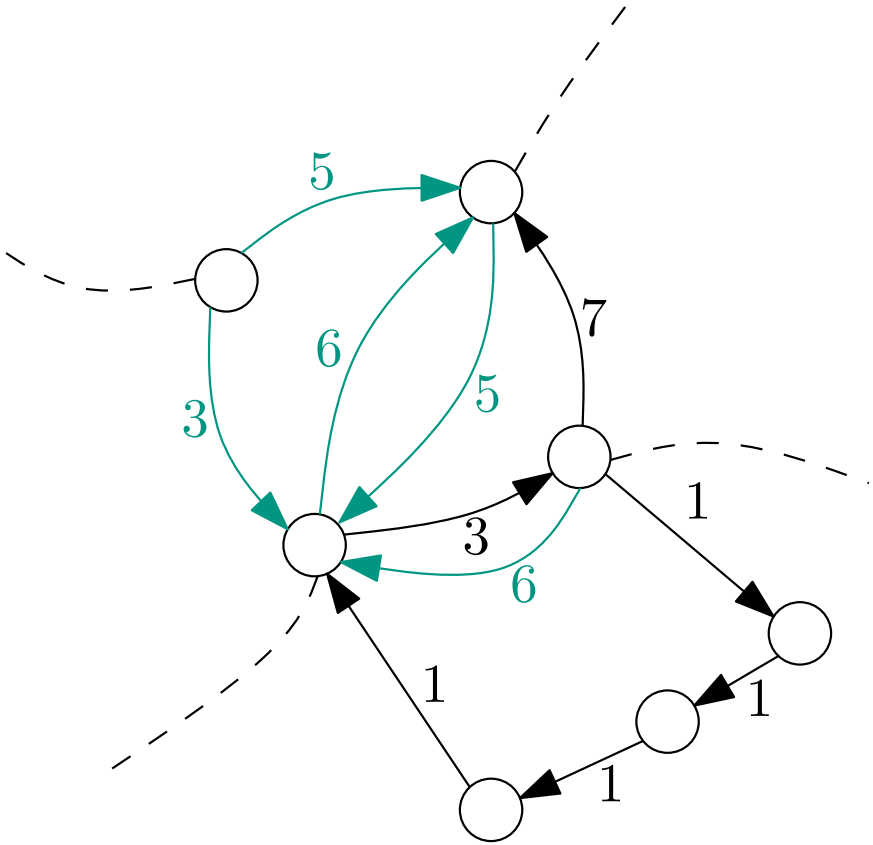
Idea: Compute shortcuts by iteratively contracting nodes



Resulting shortcuts

Contraction Hierarchies [GSSD08, GSSV12]

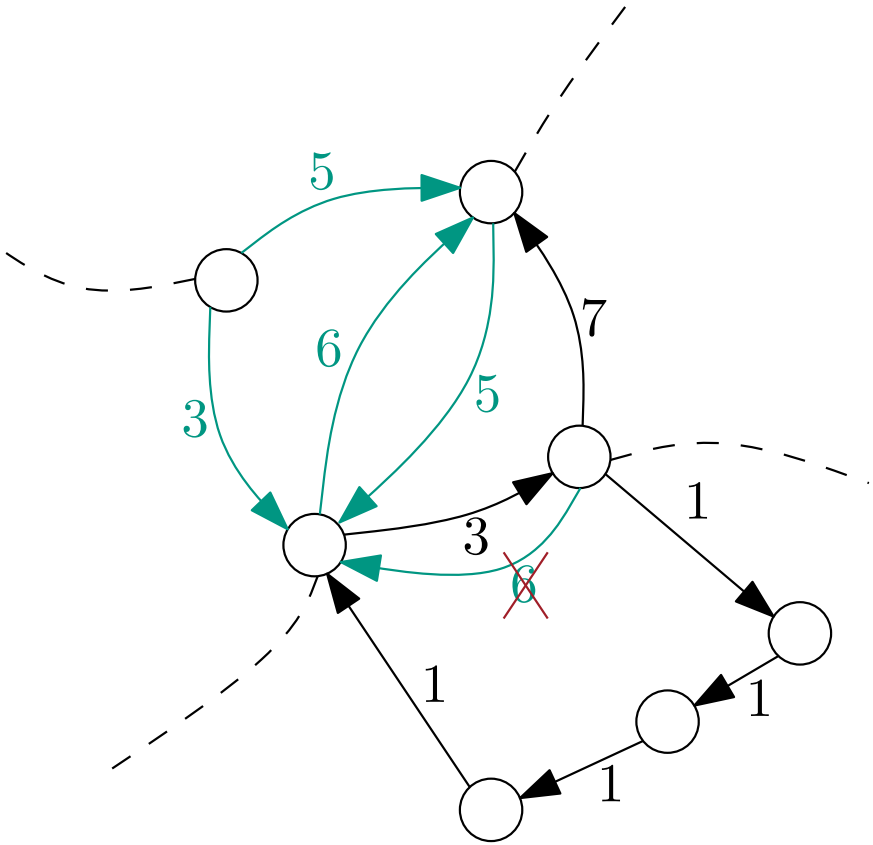
Idea: Compute shortcuts by iteratively contracting nodes



If shorter path through remaining graph exists, remove shortcut

Contraction Hierarchies [GSSD08, GSSV12]

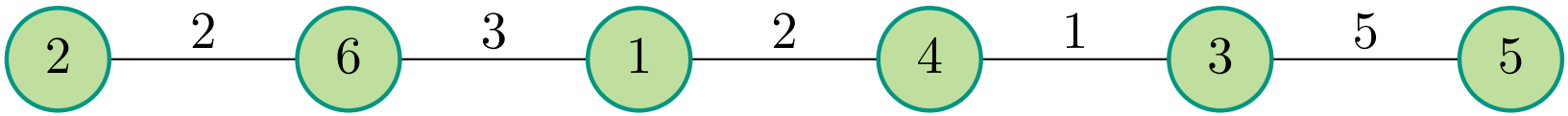
Idea: Compute shortcuts by iteratively contracting nodes



If shorter path through remaining graph exists, remove shortcut
Search for such shorter paths is called witness search

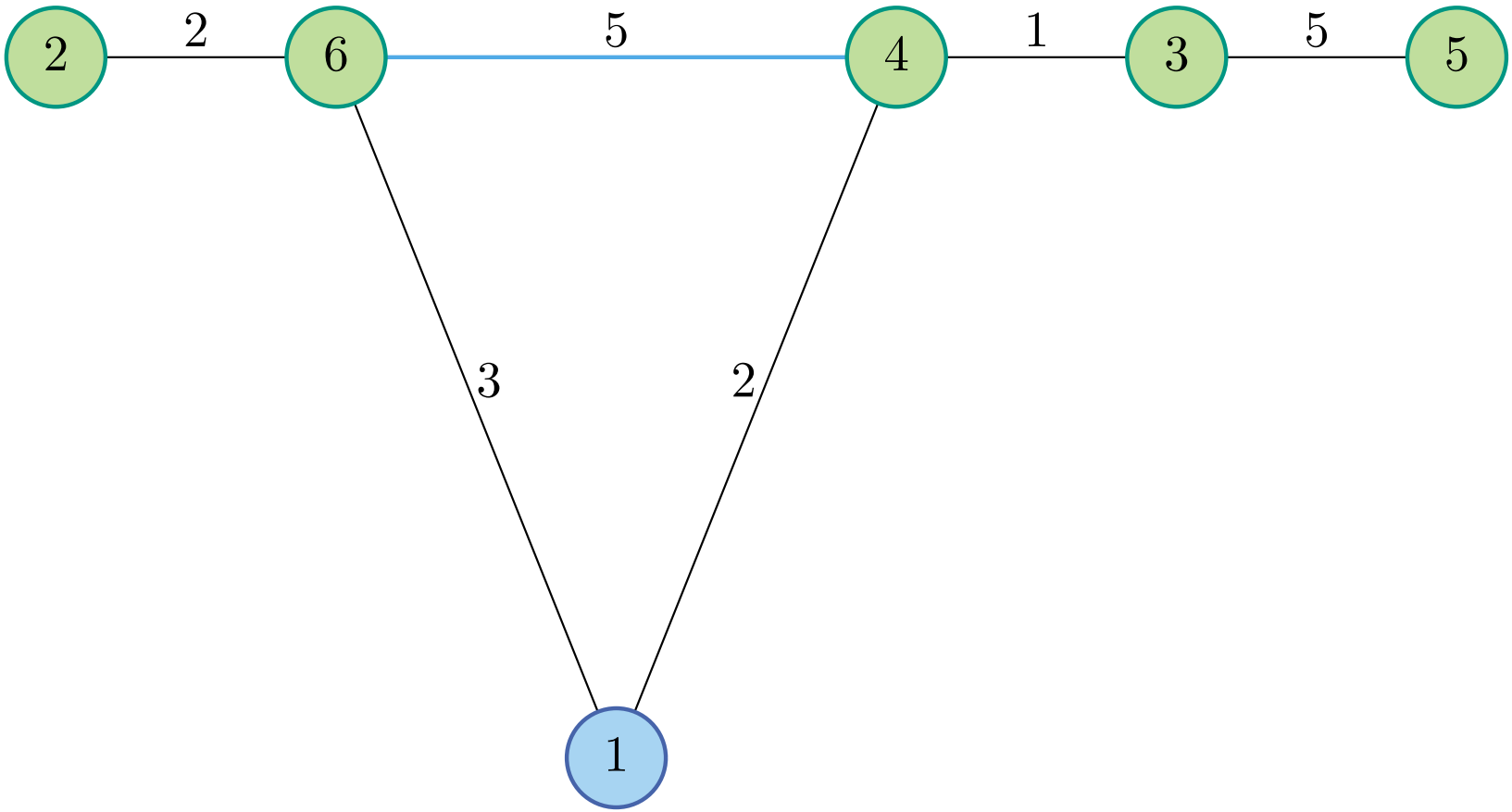
Contraction Hierarchies [GSSD08, GSSV12]

Preprocessing example: Iteratively contract nodes



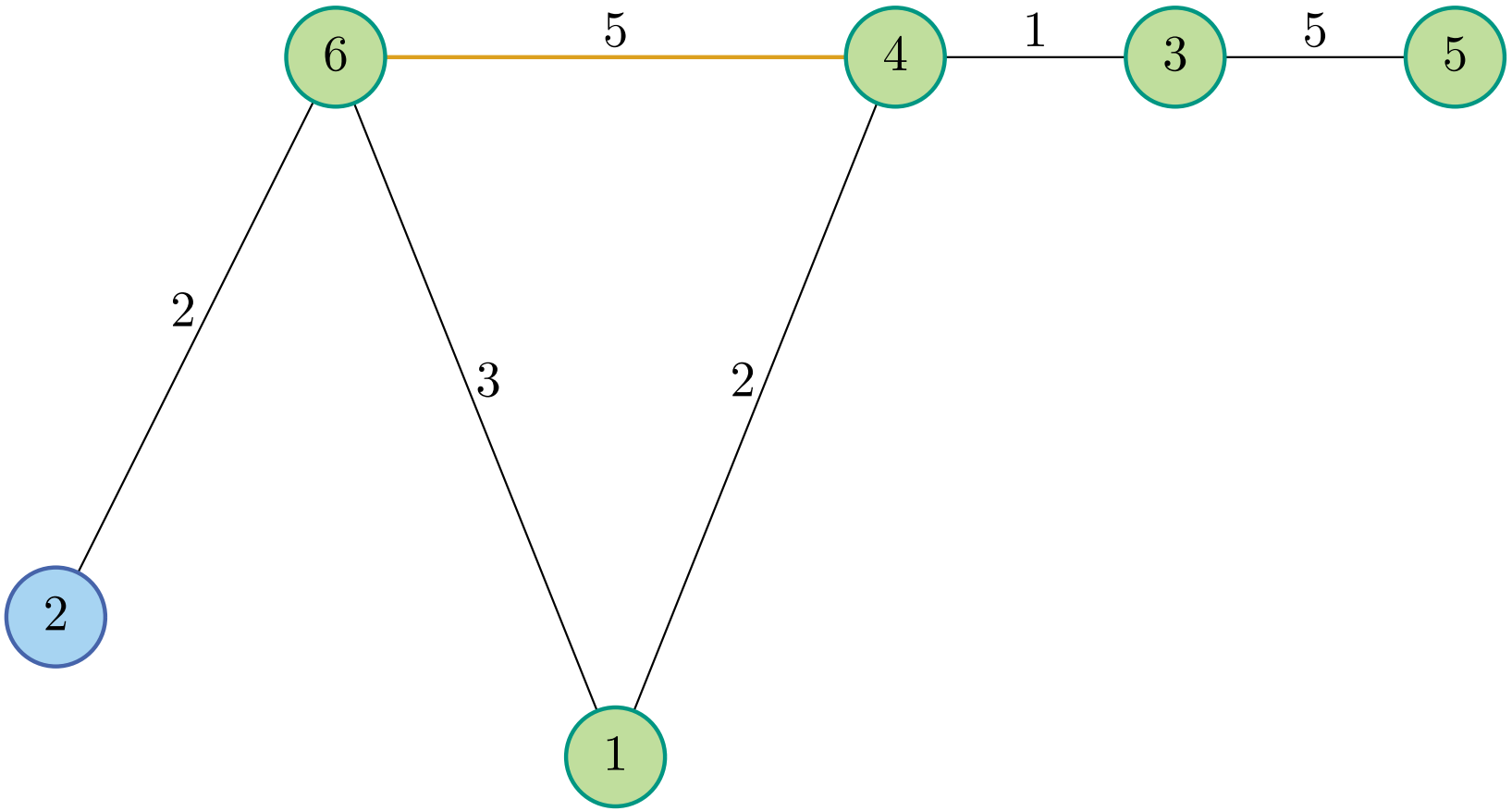
Contraction Hierarchies [GSSD08, GSSV12]

Preprocessing example: Iteratively contract nodes



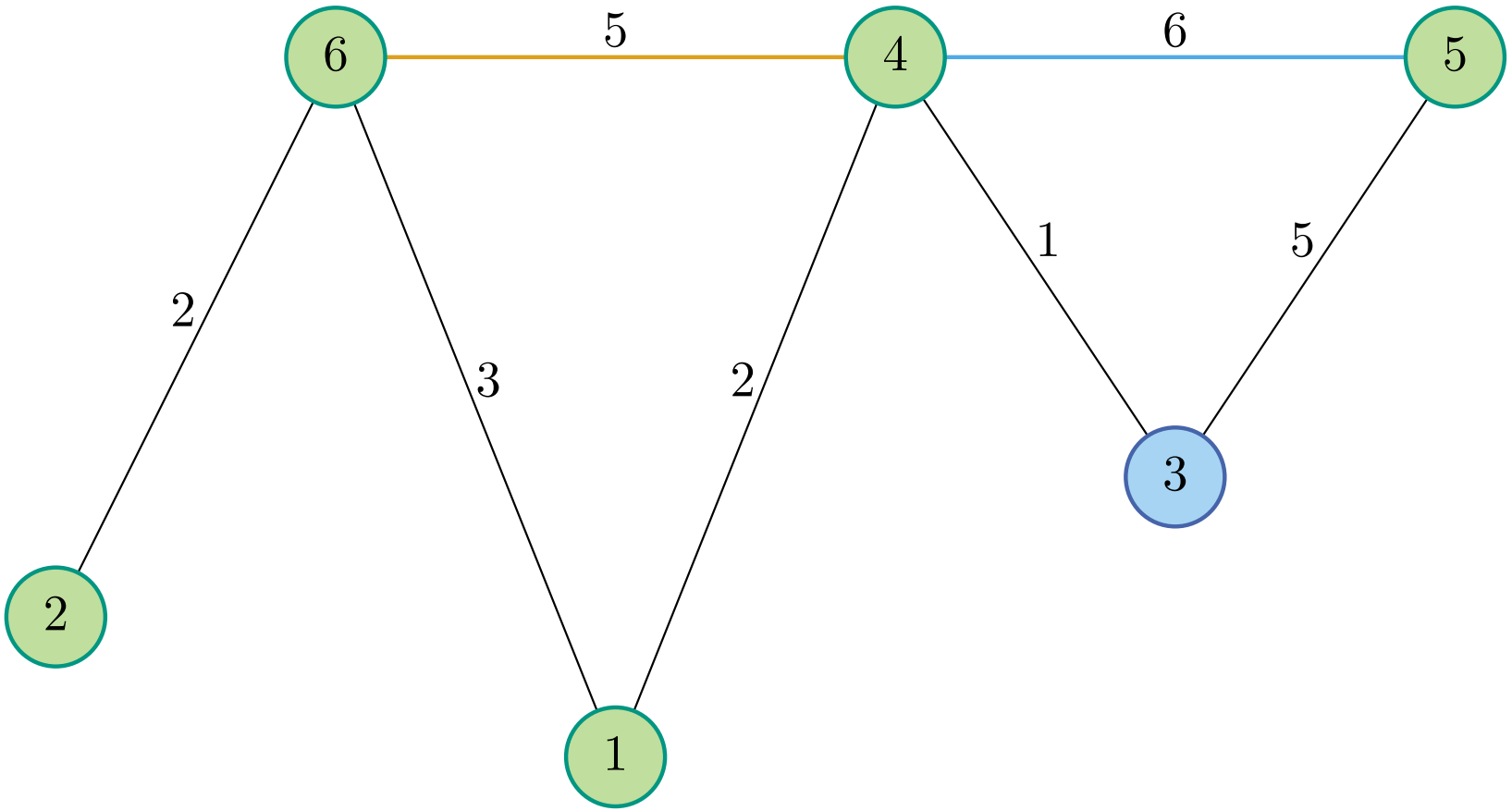
Contraction Hierarchies [GSSD08, GSSV12]

Preprocessing example: Iteratively contract nodes



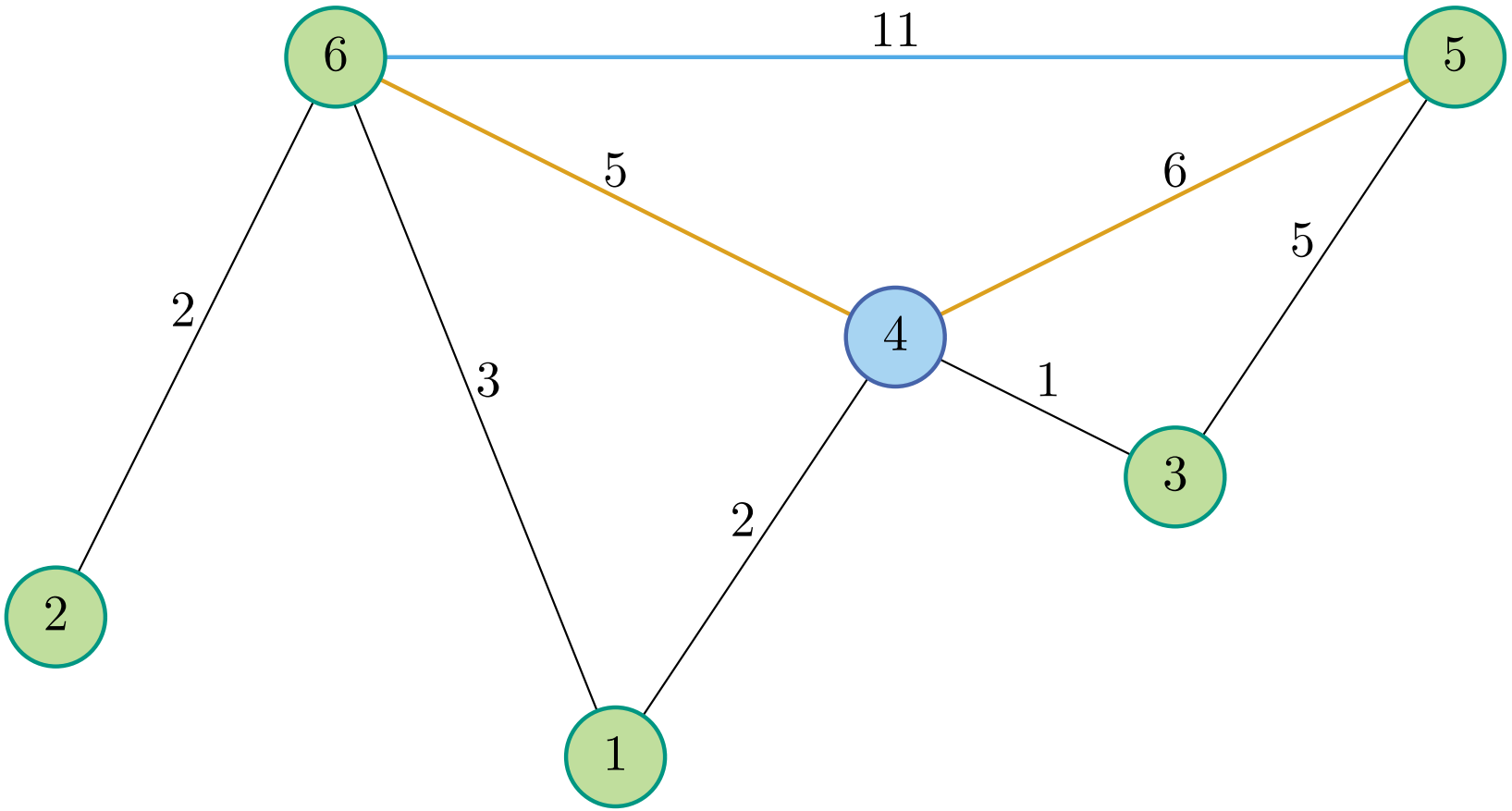
Contraction Hierarchies [GSSD08, GSSV12]

Preprocessing example: Iteratively contract nodes



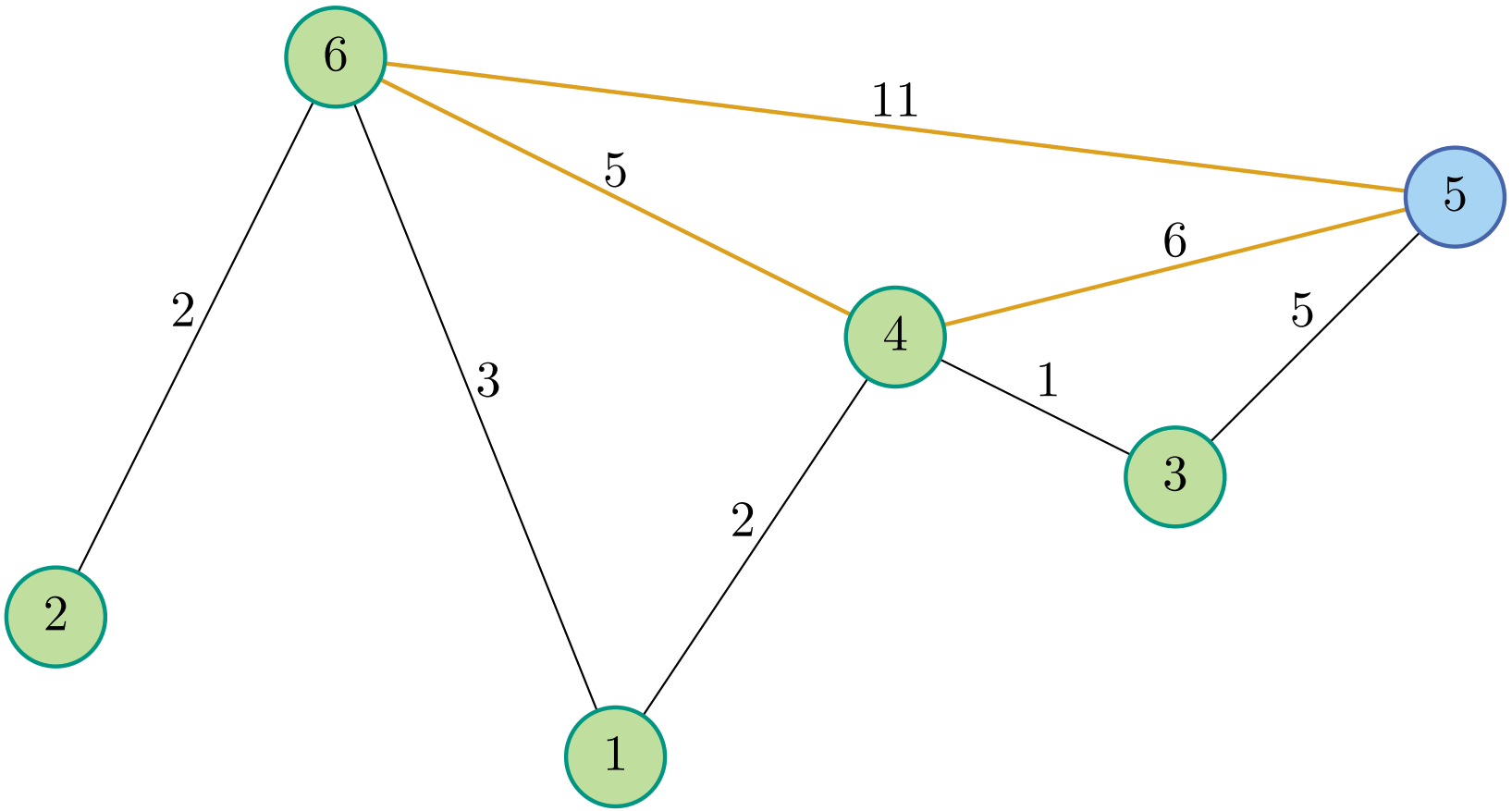
Contraction Hierarchies [GSSD08, GSSV12]

Preprocessing example: Iteratively contract nodes



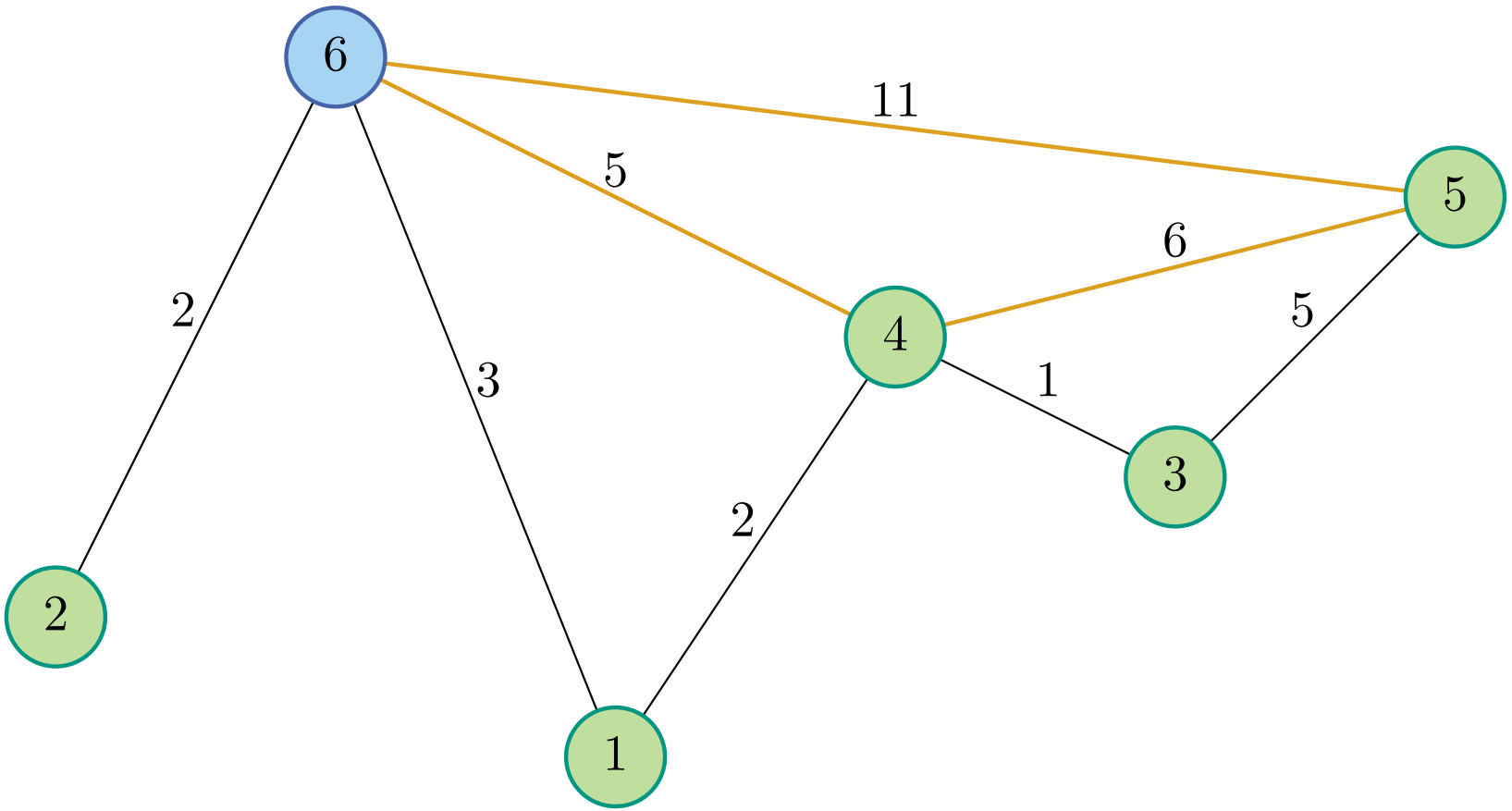
Contraction Hierarchies [GSSD08, GSSV12]

Preprocessing example: Iteratively contract nodes



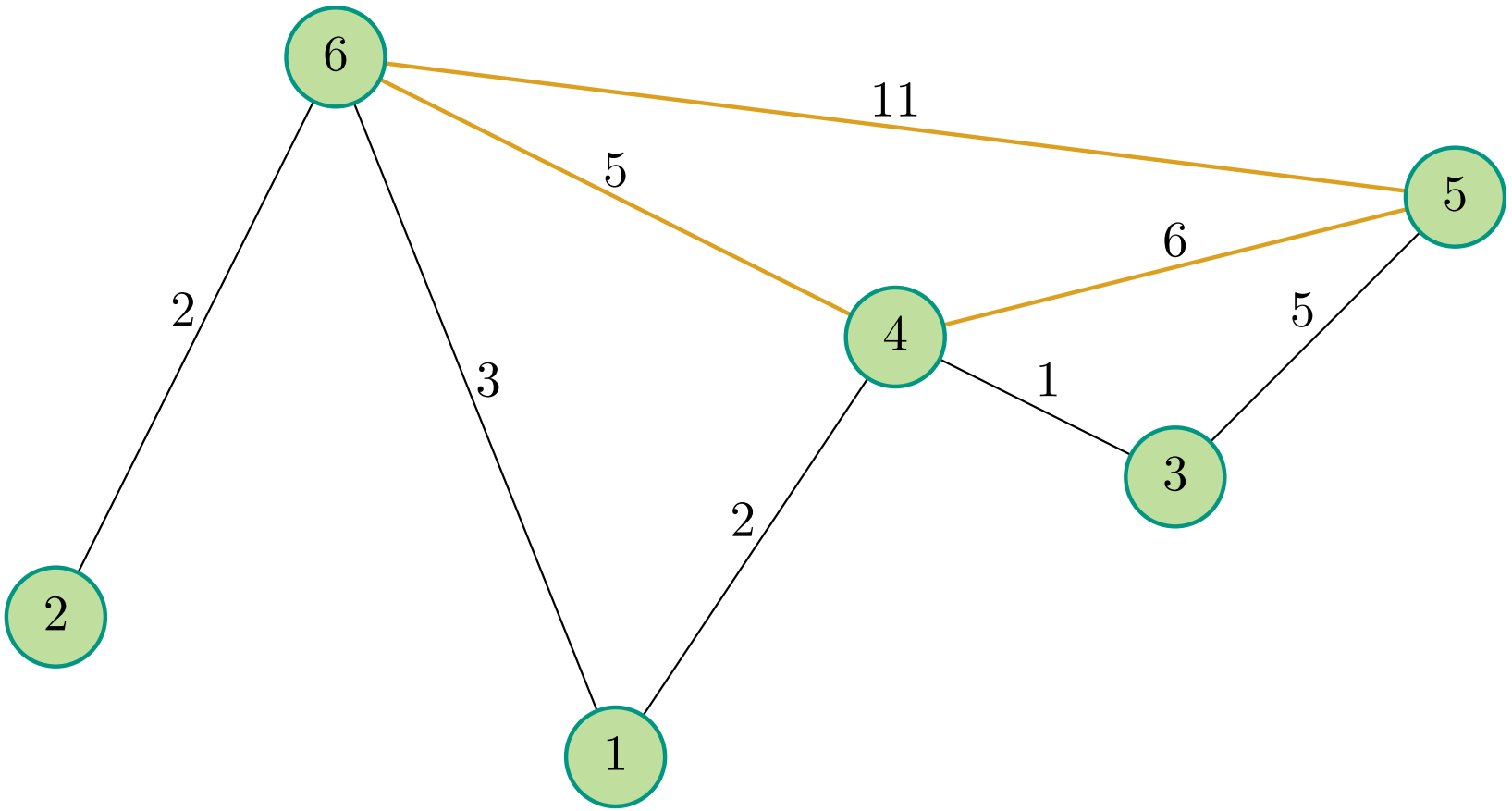
Contraction Hierarchies [GSSD08, GSSV12]

Preprocessing example: Iteratively contract nodes



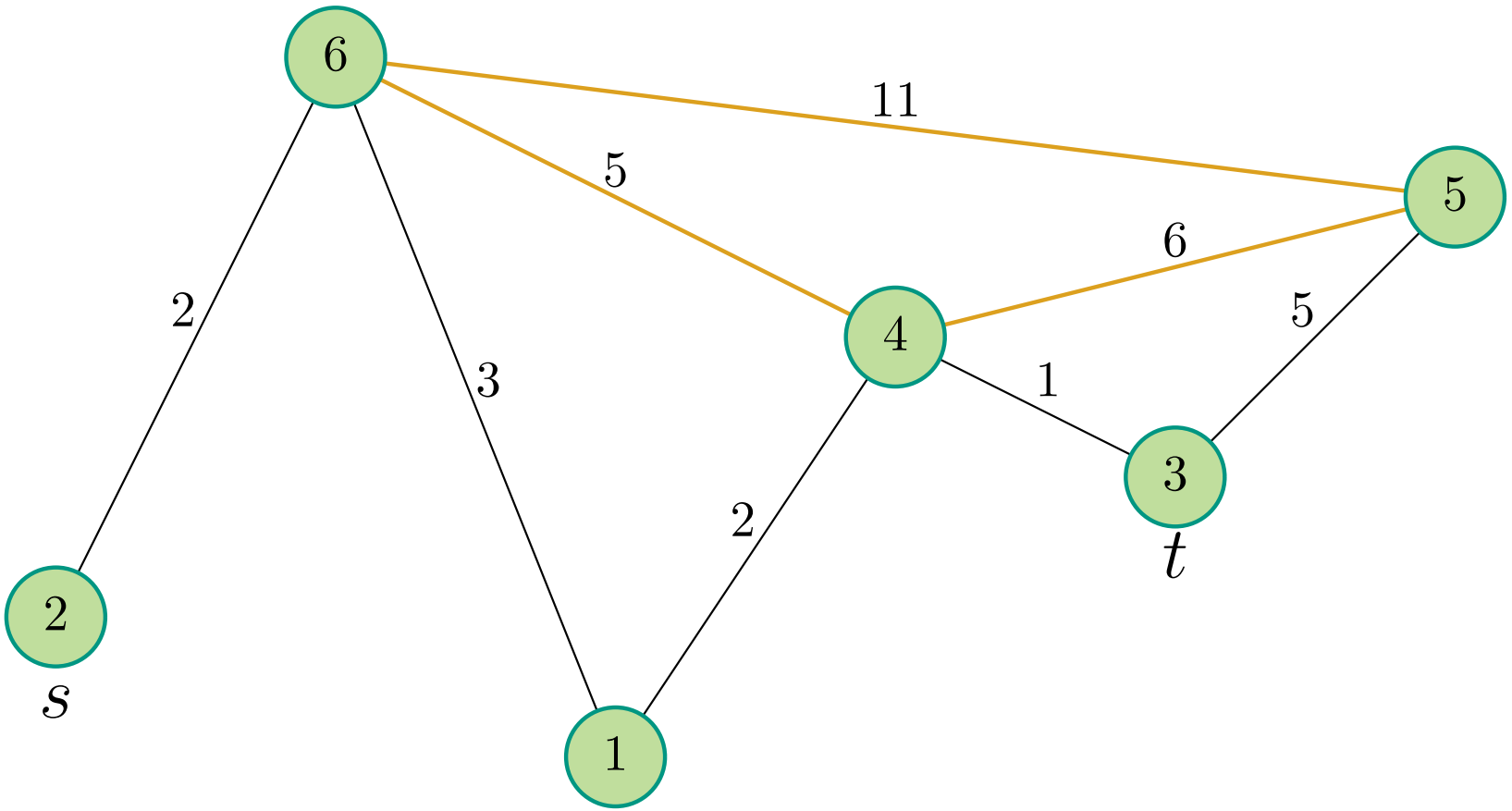
Contraction Hierarchies [GSSD08, GSSV12]

Preprocessing example: Iteratively contract nodes



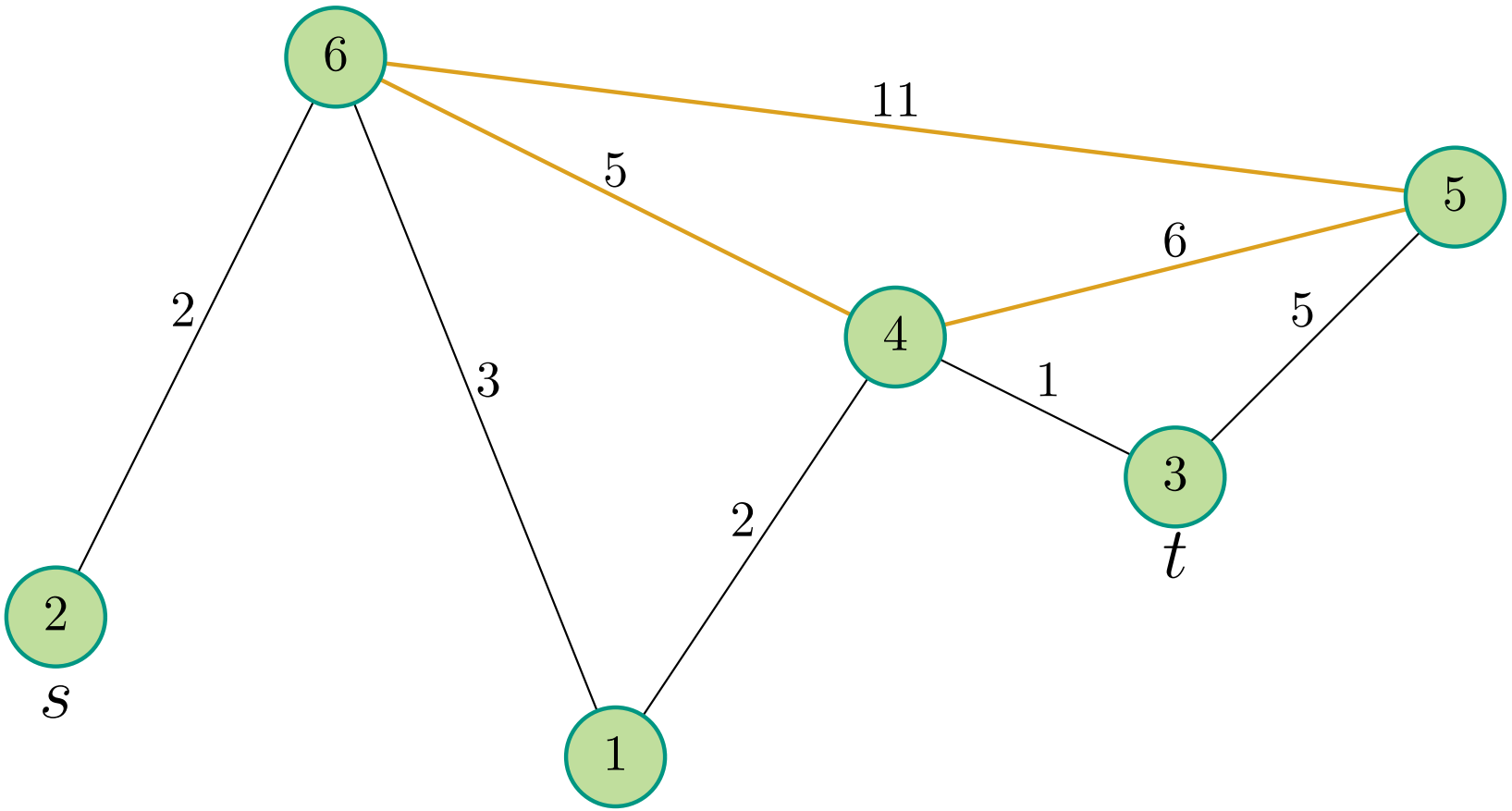
Contraction Hierarchies [GSSD08, GSSV12]

Preprocessing example: Iteratively contract nodes



Contraction Hierarchies [GSSD08, GSSV12]

Preprocessing example: Iteratively contract nodes

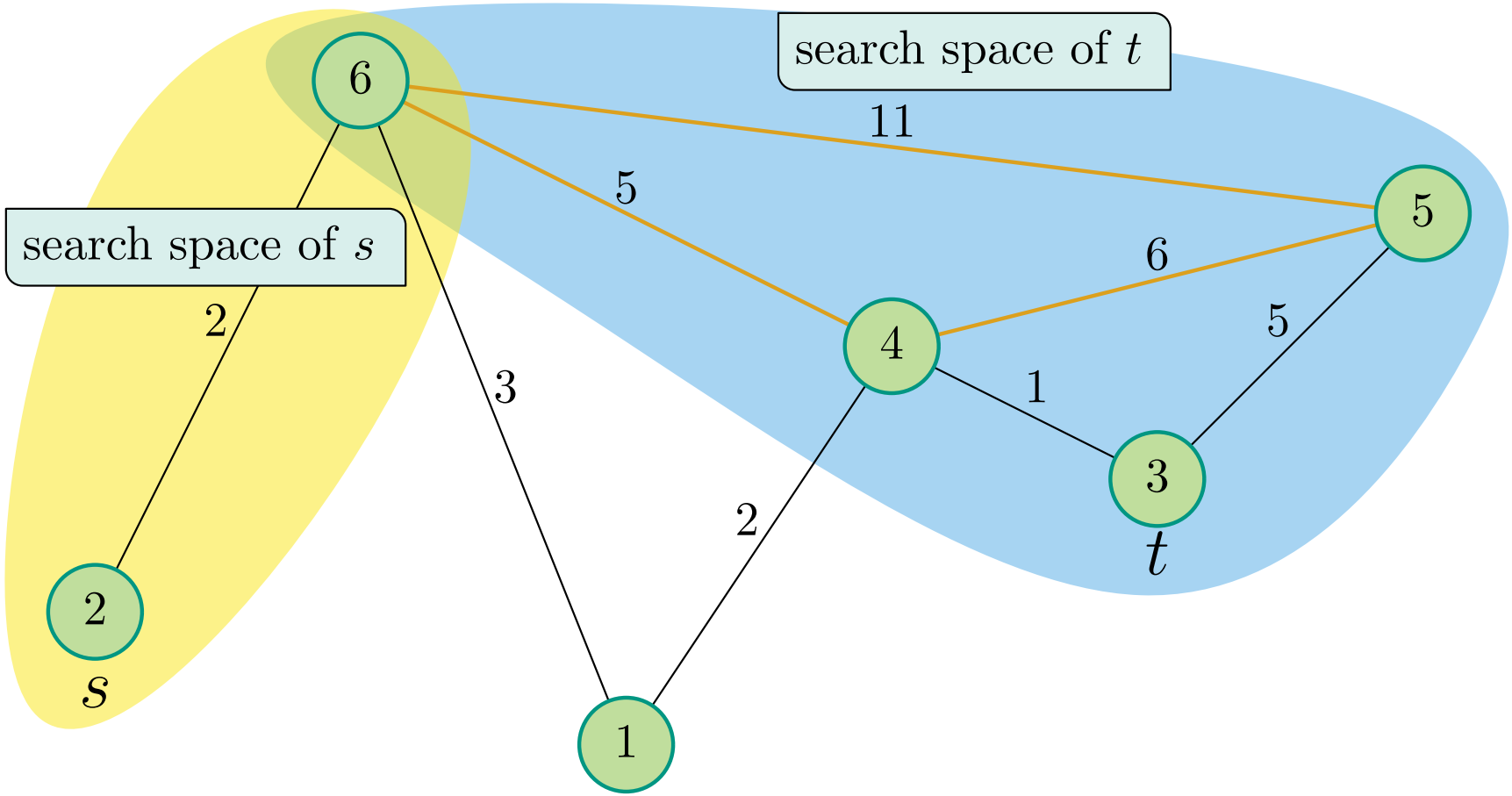


Order nodes by “importance”

Intuition: Nodes on more shortest paths are more important

Contraction Hierarchies [GSSD08, GSSV12]

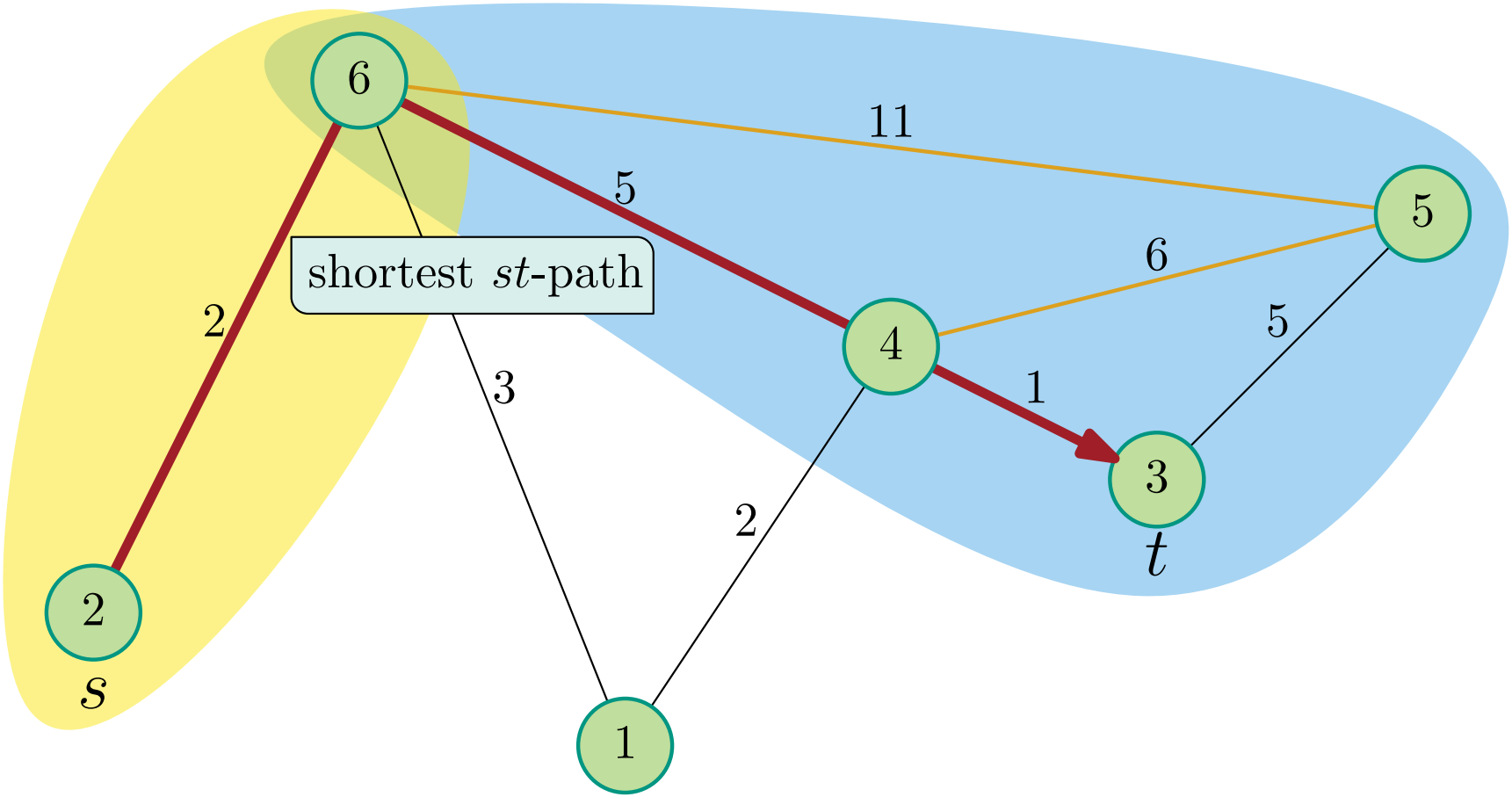
Query example: Bidirectional upward search



Institute for Theoretical Informatics
Chair Algorithmics

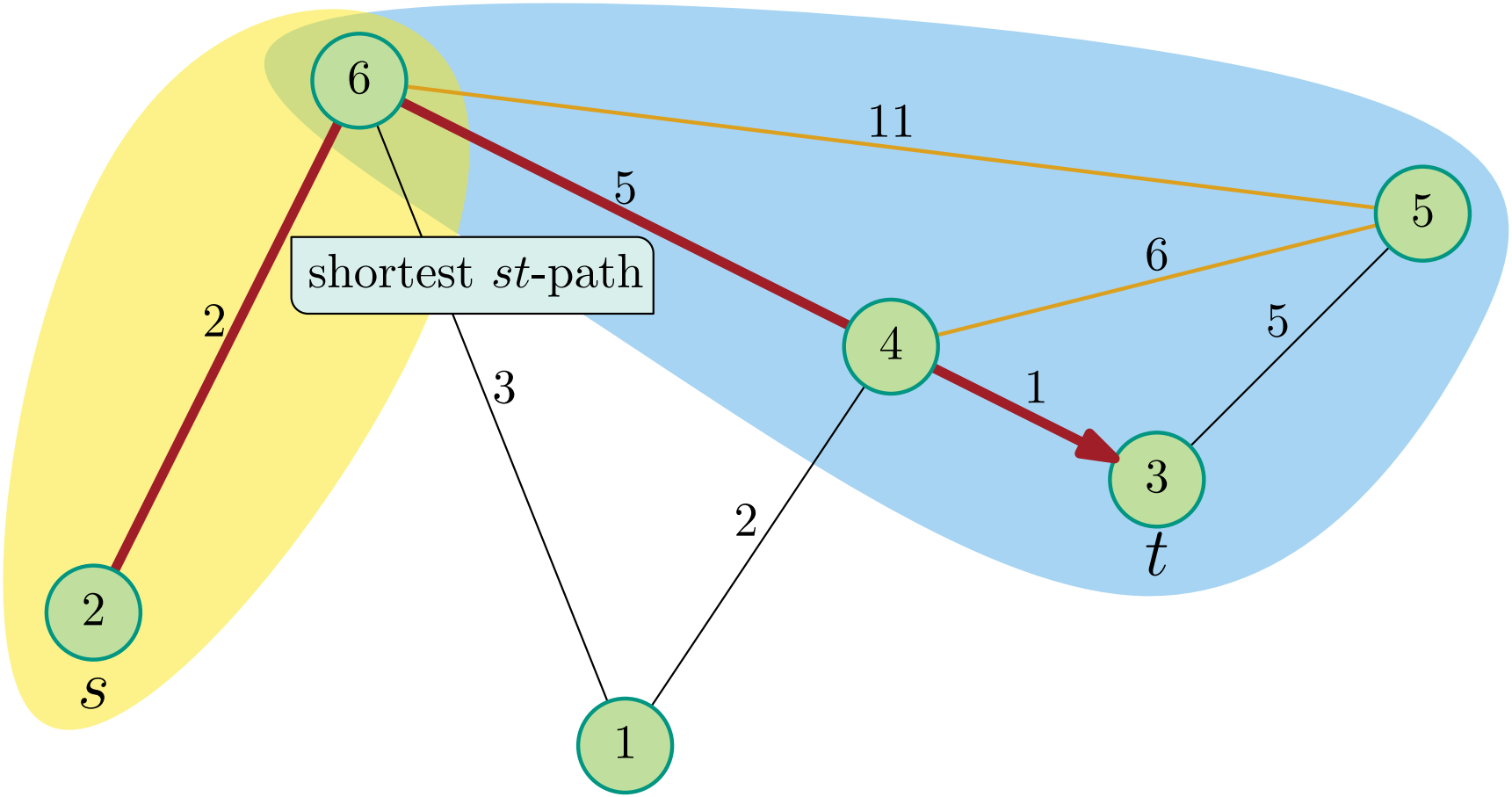
Contraction Hierarchies [GSSD08, GSSV12]

Query example: Bidirectional upward search



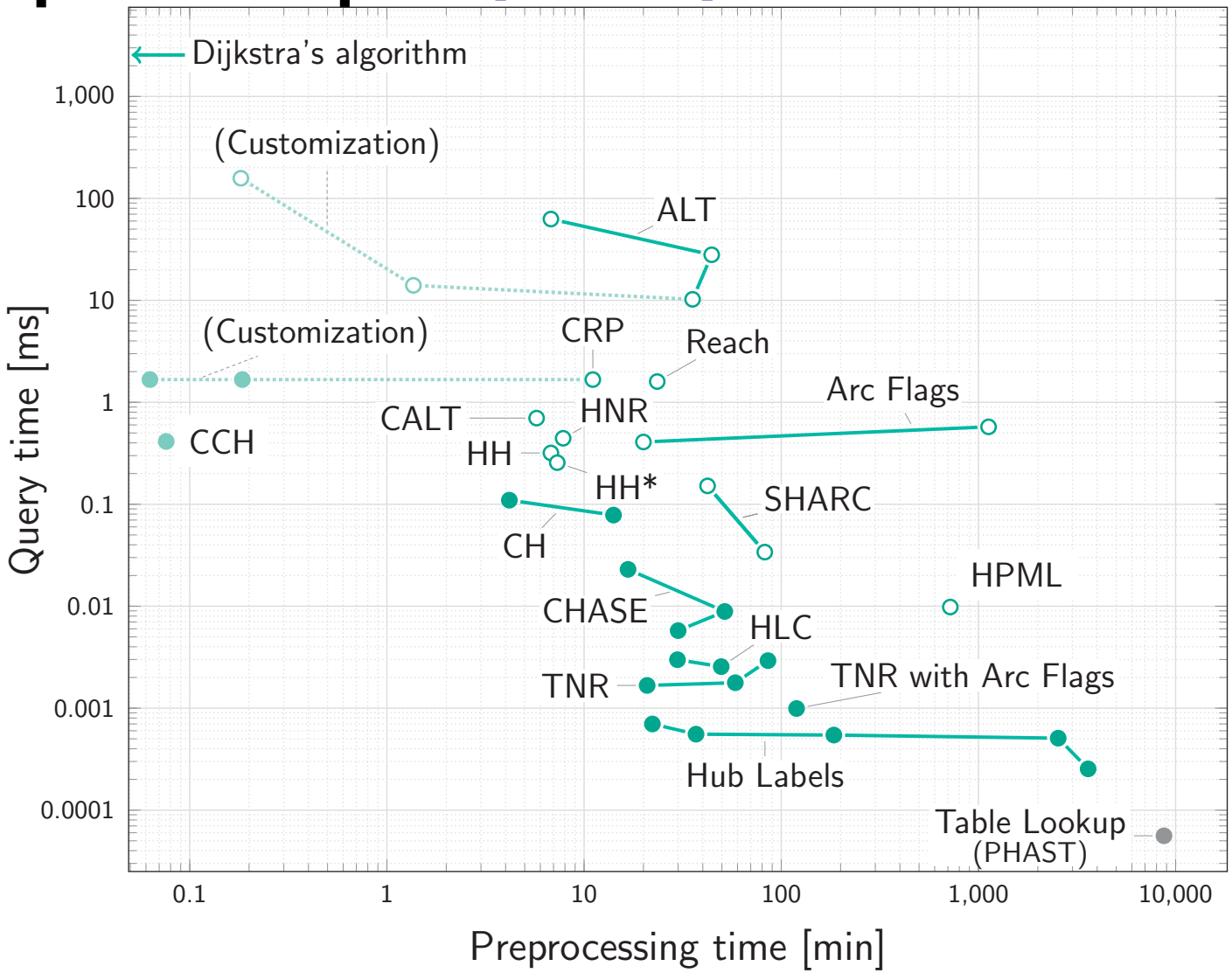
Contraction Hierarchies [GSSD08, GSSV12]

Query example: Bidirectional upward search



For every original shortest path, there is a shortest up-down path

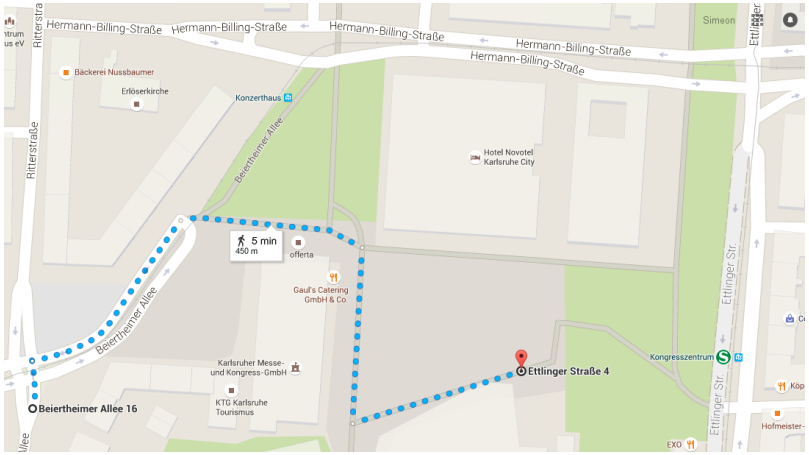
Speedup Techniques [BDG+15a]



In use at Apple, Bing, Google, TomTom, . . .

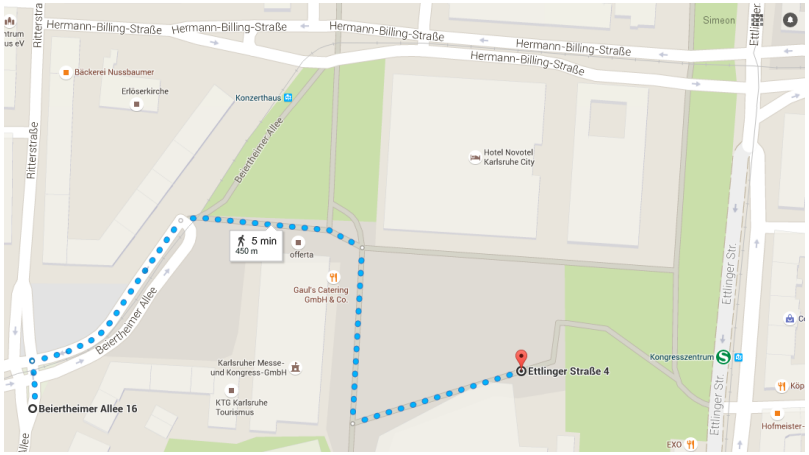
The Metric Matters

- Travel time
- Distance
- Turn costs
- Points of interests
  (nearest POIs, shortest via-POIs)
- Electromobility
- User customizable metrics
  e.g., height restrictions, avoid freeways, eco-friendliness, . . .
- Pedestrian routing



The Metric Matters

- Travel time
- Distance
- Turn costs
- Points of interests
  (nearest POIs, shortest via-POIs)
- Electromobility
- User customizable metrics
  e.g., height restrictions, avoid freeways, eco-friendliness, . . .
- Pedestrian routing



Route Planning for Electric Vehicles

Route Planning for Electric Vehicles

Electric vehicles:

- Future means of transportation
- Run on regenerative energy sources

But:

- Restricted battery capacity
- Long recharging times
- “Range anxiety”

$\Rightarrow$ Consider energy consumption in route planning applications

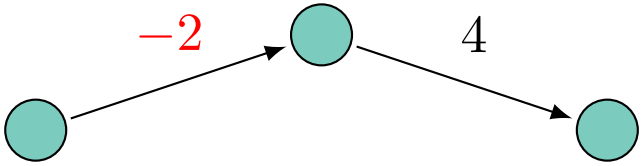
Task: Given start and destination in a road network,
find the route that minimizes energy consumption.

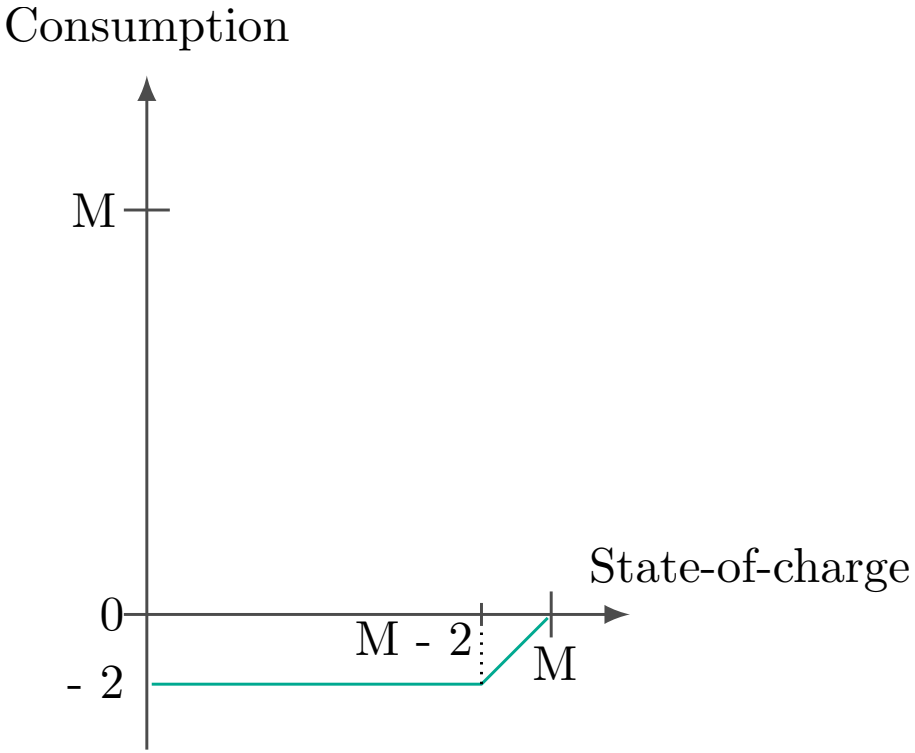
Energy-Optimal Routes

Challenges:

- Negative edge weights (recuperation)
- Battery constraints (no over-, undercharging)

Energy consumption depends on battery state-of-charge (at the start):



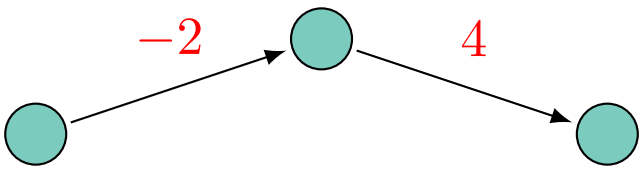


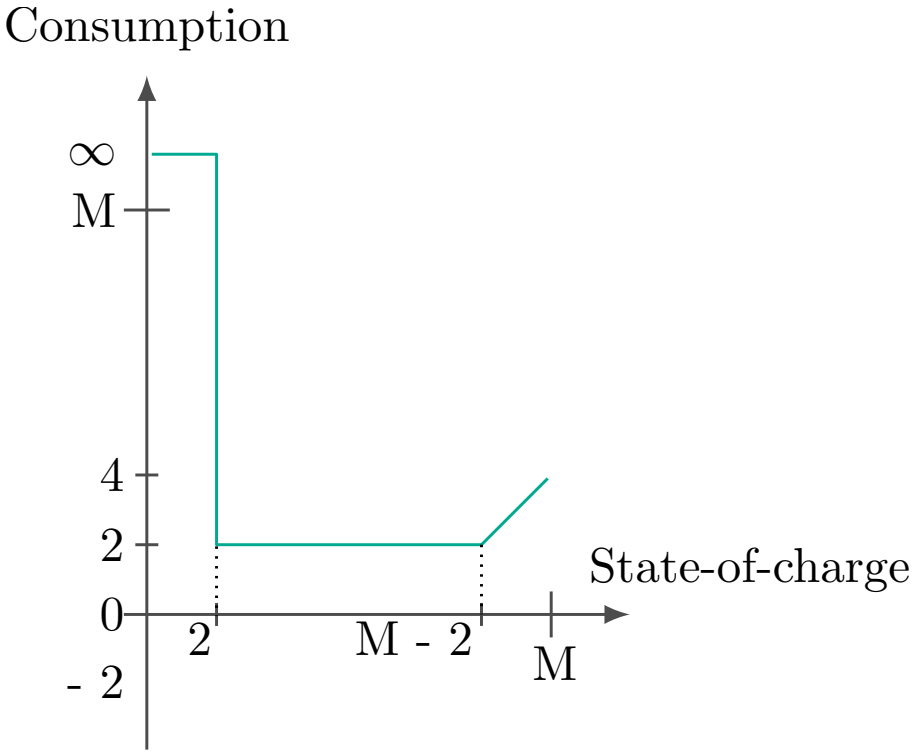
Energy-Optimal Routes

Challenges:

- Negative edge weights (recuperation)
- Battery constraints (no over-, undercharging)

Energy consumption depends on battery state-of-charge (at the start):



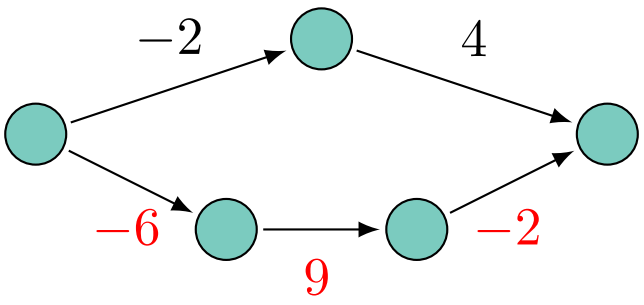


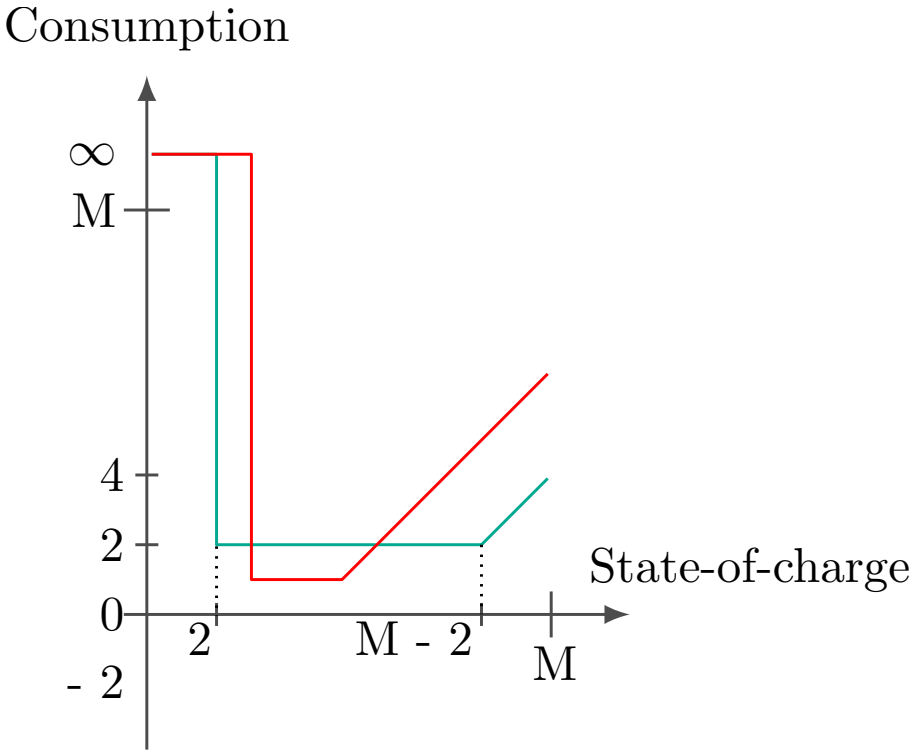
Energy-Optimal Routes

Challenges:

- Negative edge weights (recuperation)
- Battery constraints (no over-, undercharging)

Energy consumption depends on battery state-of-charge (at the start):



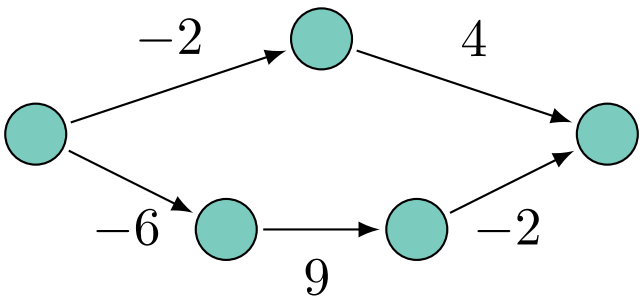


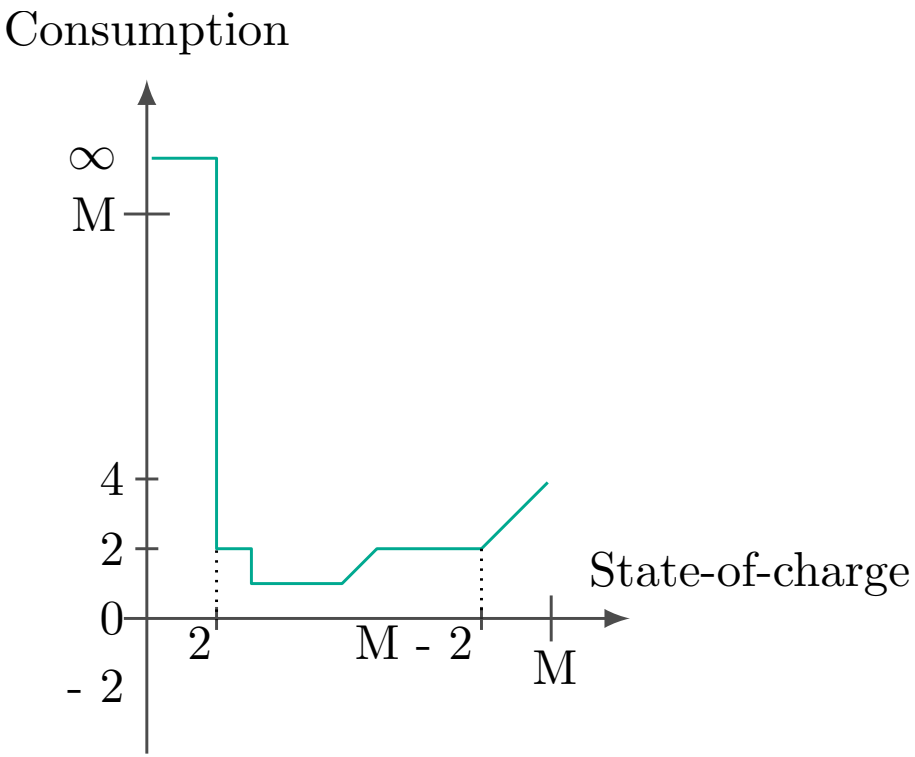
Energy-Optimal Routes

Challenges:

- Negative edge weights (recuperation)
- Battery constraints (no over-, undercharging)

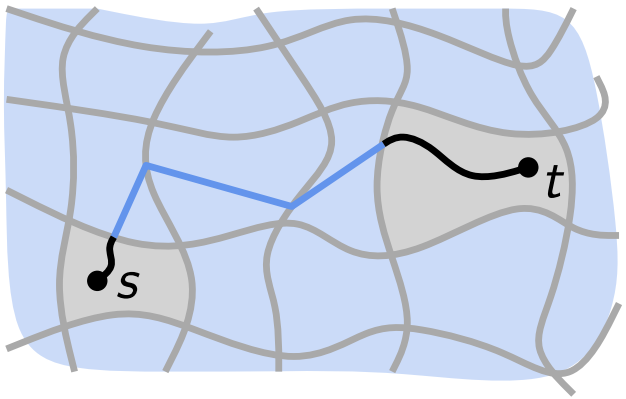
Energy consumption depends on battery state-of-charge (at the start):

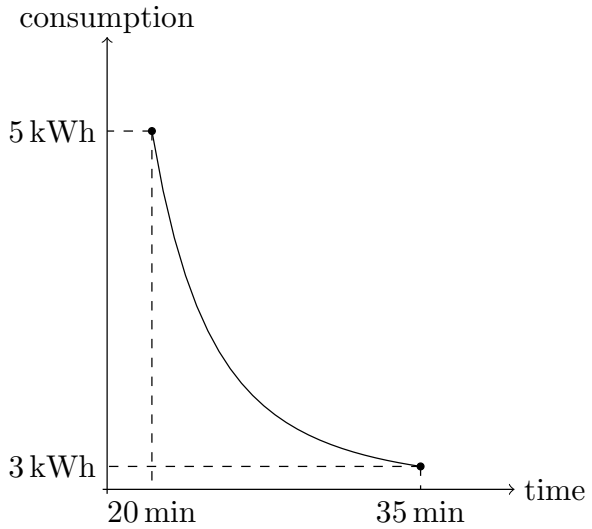




Energy-Optimal Routes [BDPW13]

- Shortcuts are functions, not scalar values
- Bidirectional search more complicated (unknown state-of-charge at target)
- User-dependent consumption profiles ($\Rightarrow$ custom metrics)





Experiments:

- Fast queries (few milliseconds)
- Fast customization (few seconds)

But: Energy-optimal routes follow slow roads

- Energy-optimal paths: 63 % extra time
- Fastest paths: 62 % extra energy

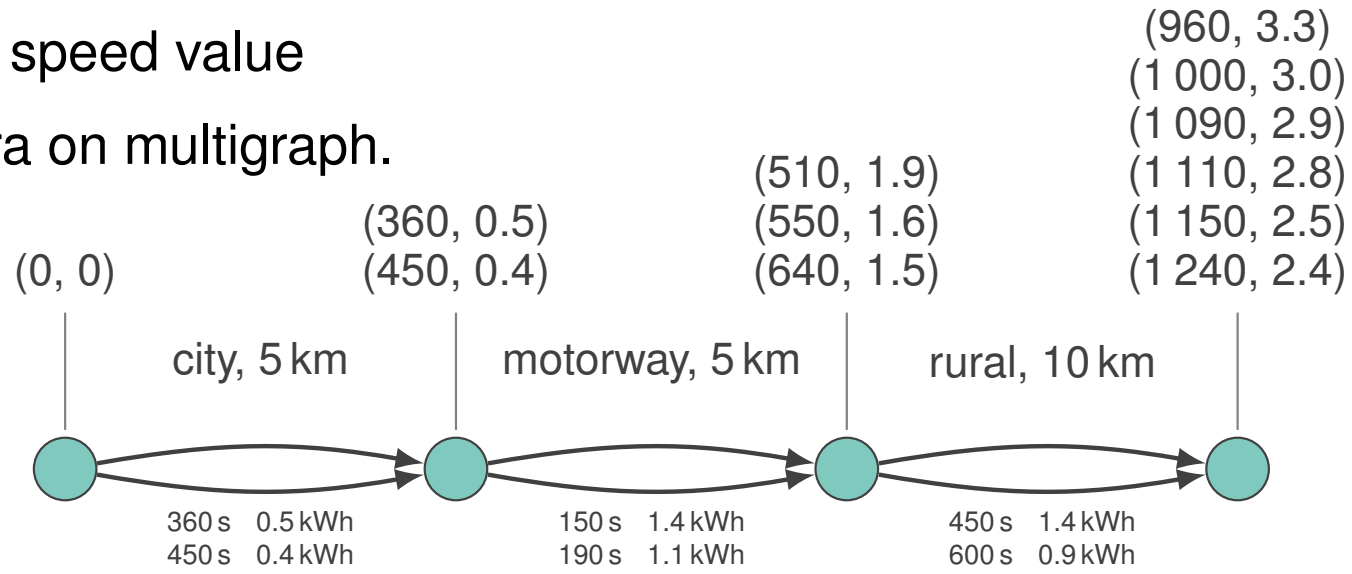
$\Rightarrow$ Consider tradeoff between speed and energy consumption

Find the fastest path such that the battery does not run out: $\mathcal{NP}$**-hard**

Constrained Shortest Paths [BDHS+14]

- Energy can be saved driving below speed limit
- Additional instructions to the driver
- One edge per speed value

⇒ Bicriteria Dijkstra on multigraph.



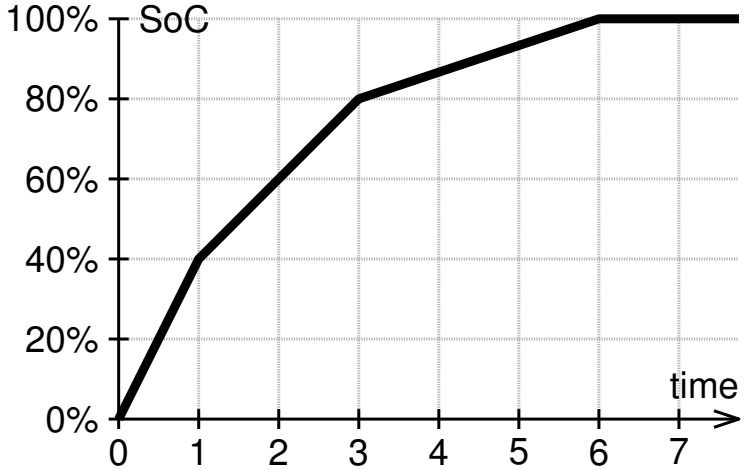
Worst case: n vertices with k parallel edges produce $\Theta(k^n)$ solutions

In practice:

- Slow basic algorithm (hours)
- Fast heuristics (seconds and below)

Including Charging Stops [BDG+15b]

- Recharging allowed at some nodes (but requires charging time).
- Realistic models of charging stations:
  - Charging power varies
  - Super chargers
  - Battery swapping stations



Challenges:

1. Recuperation, battery constraints
2. Energy efficient driving vs. time consuming charging stops
   - Detour for reaching a charging station
3. Charging is not uniform
   - Interrupt charging and take another station later

Observations

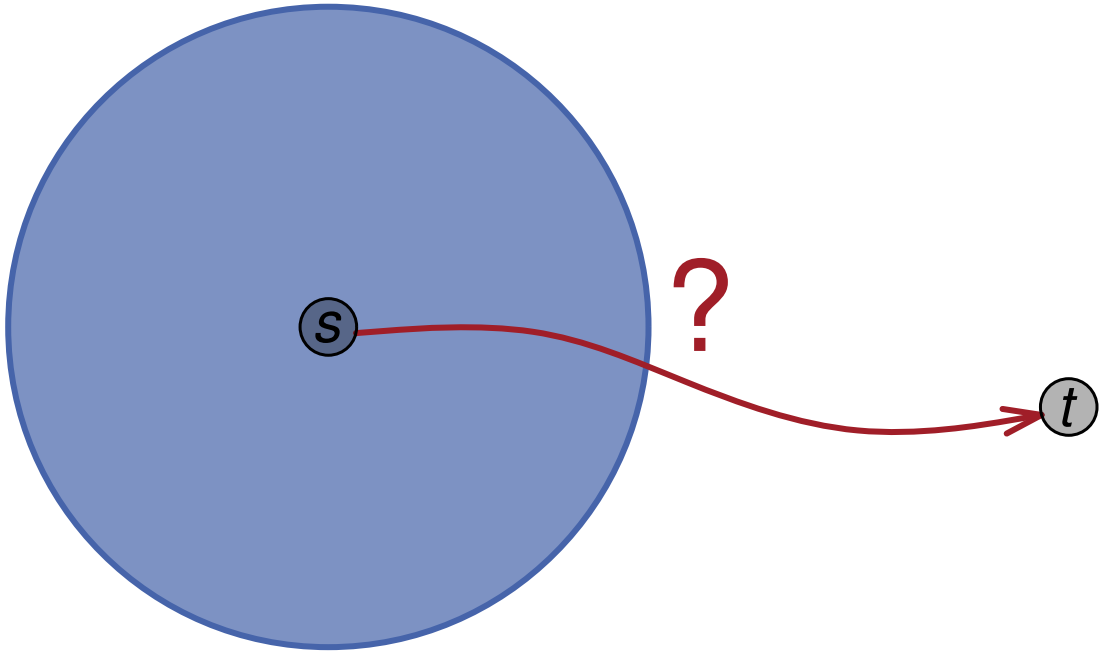
Find the fastest route from s to t:

Reachable area

Charging station

Observations

Find the fastest route from s to t:

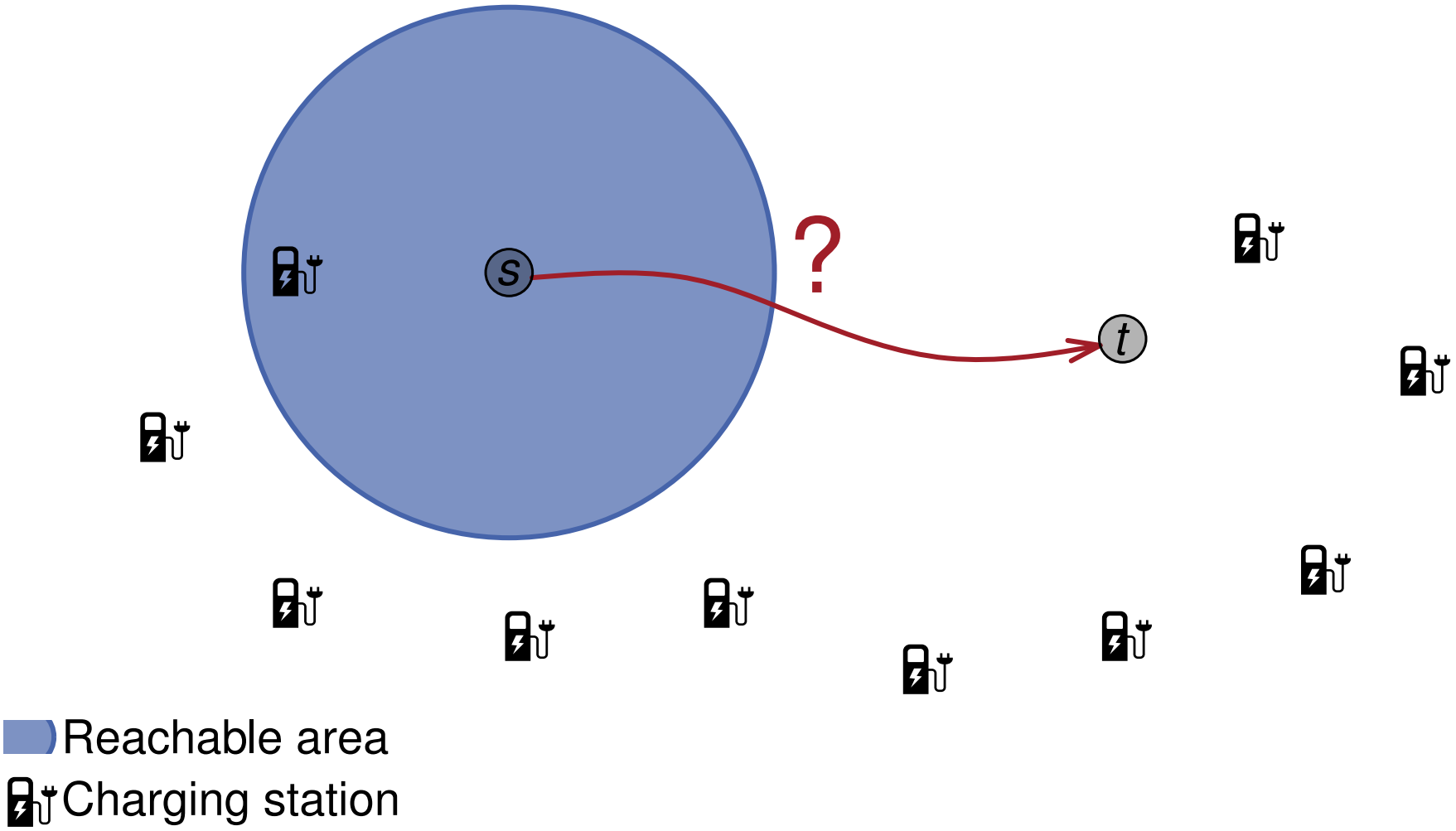


Reachable area

Charging station

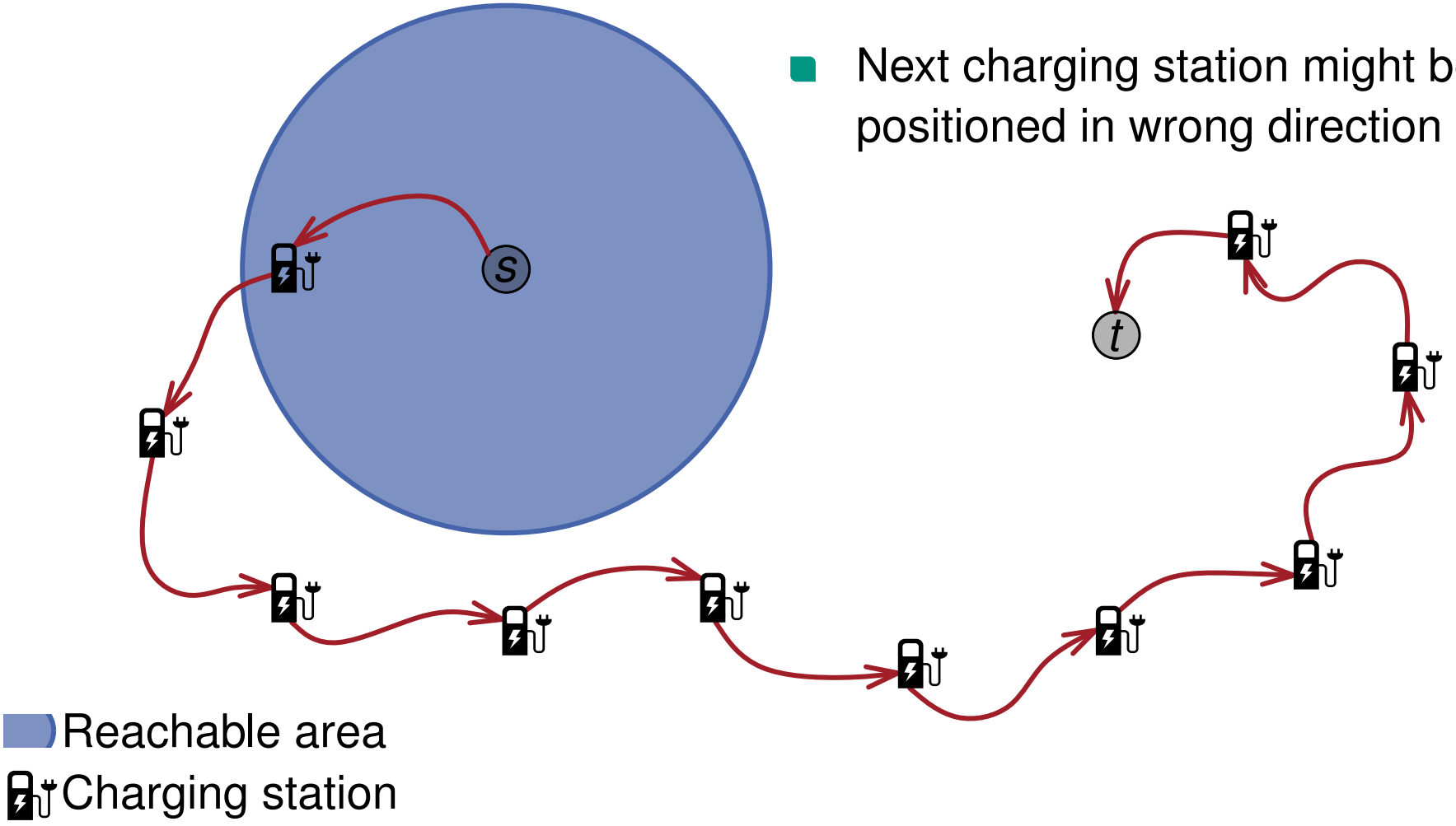
Observations

Find the fastest route from s to t:



Observations

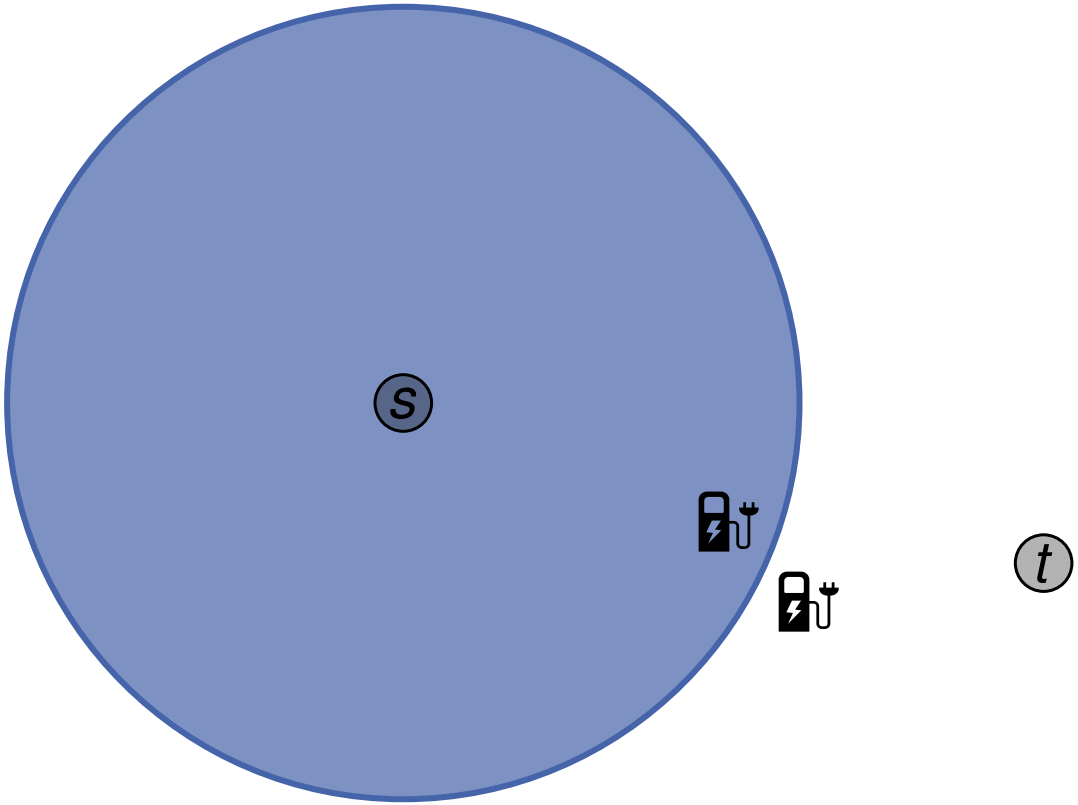
Find the fastest route from s to t:



- Next charging station might be positioned in wrong direction

Observations

Find the fastest route from s to t:

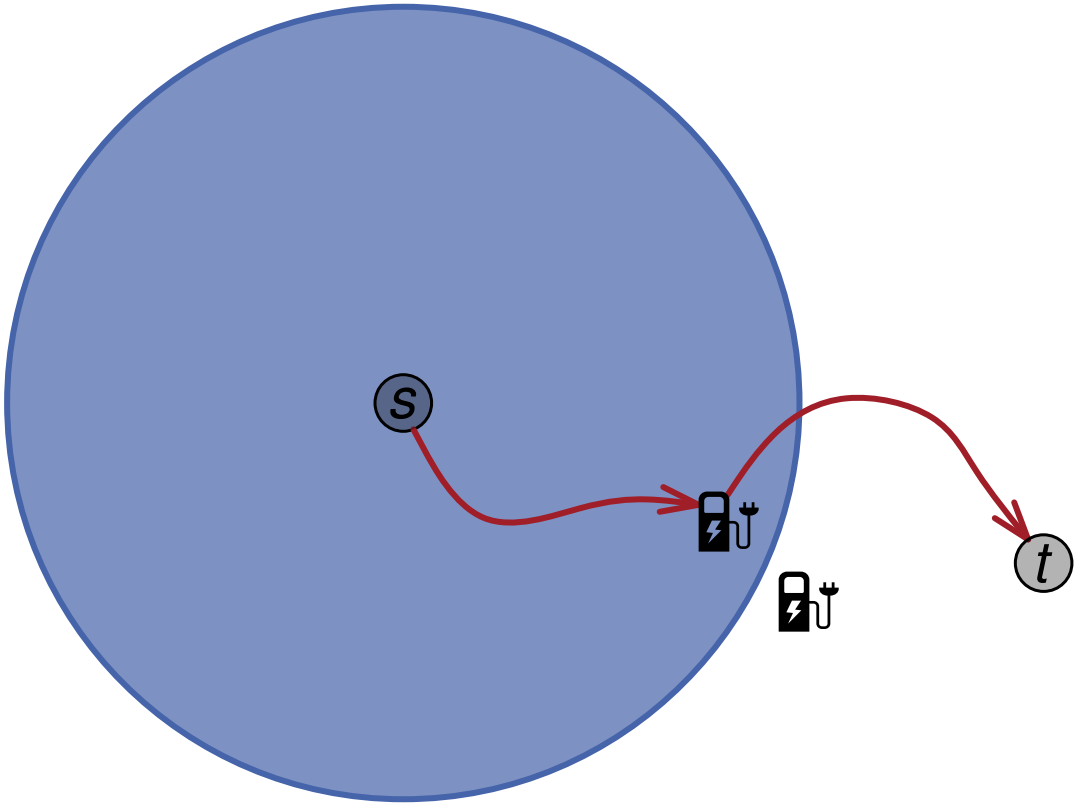


Reachable area

Charging station

Observations

Find the fastest route from s to t:

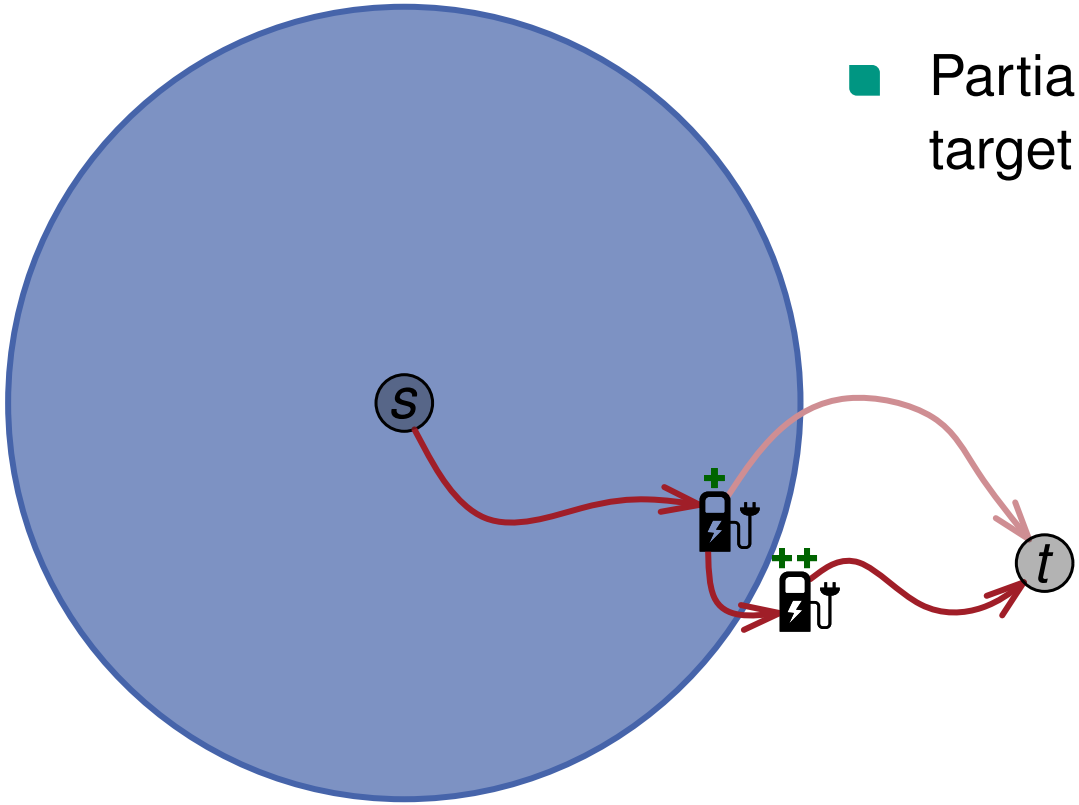


Reachable area

Charging station

Observations

Find the fastest route from s to t:



- Partial recharging, even if the target is already reachable

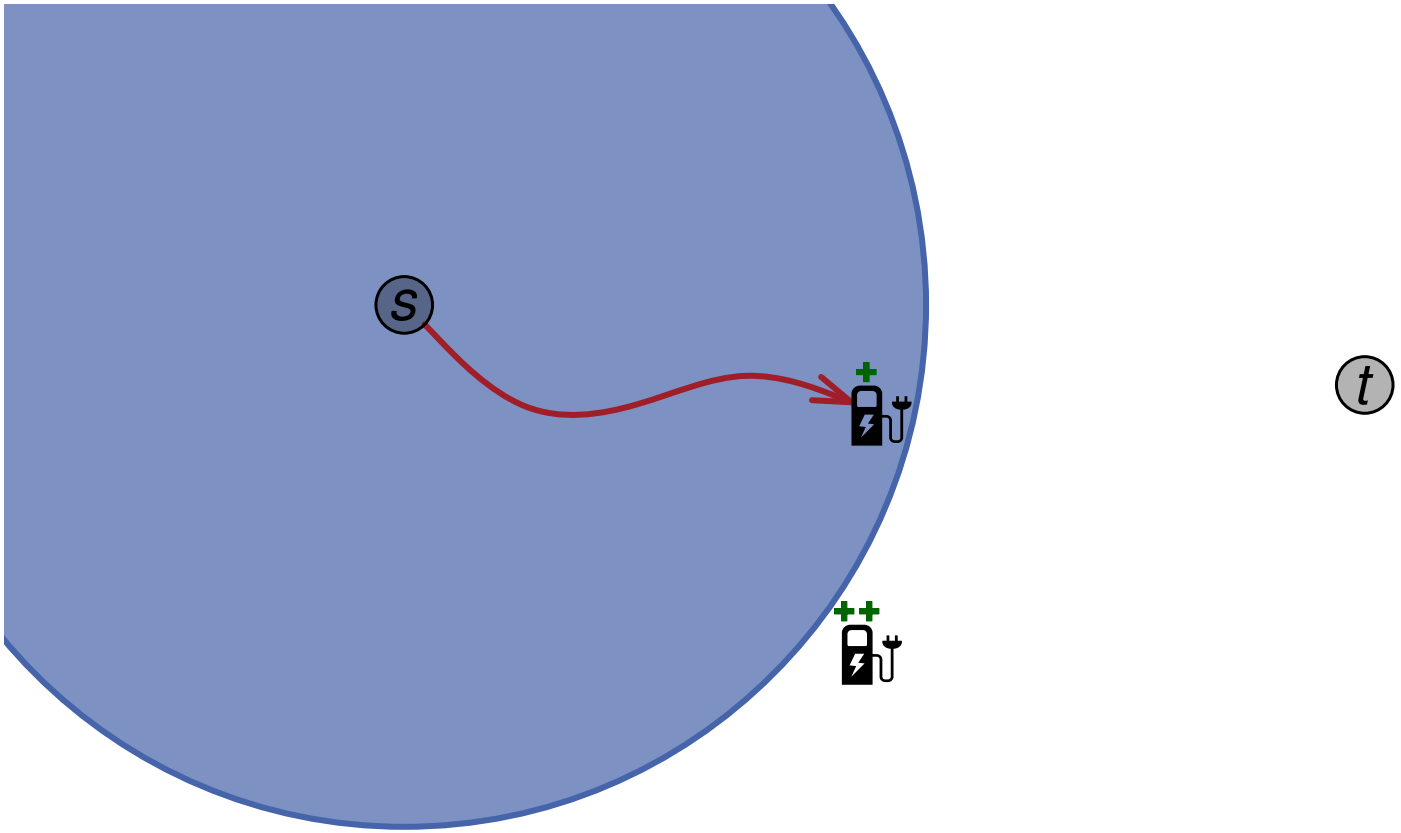
Reachable area

Charging station

Fast charging station / swapping station

Observations

Find the fastest route from s to t:



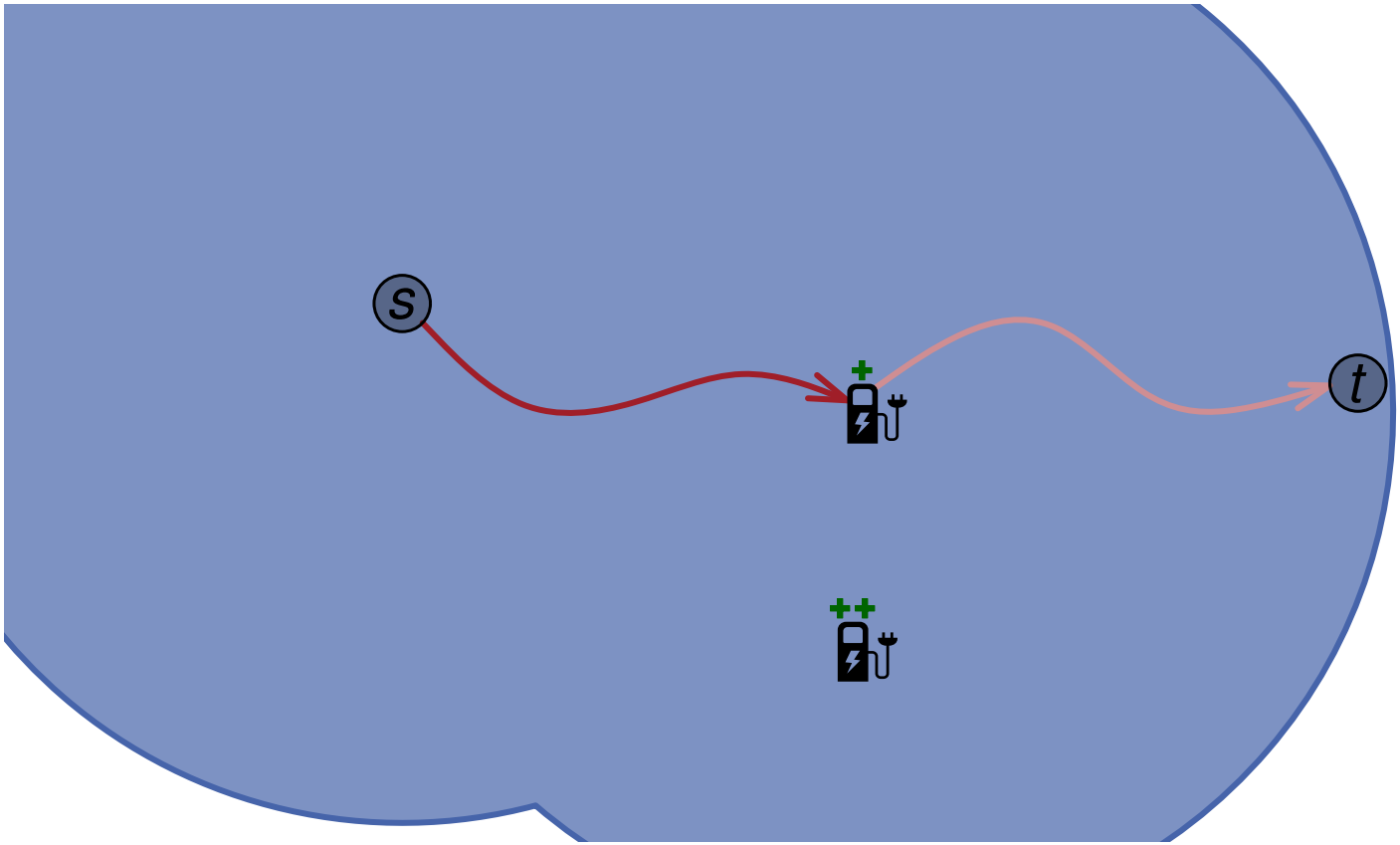
Reachable area

Charging station

Fast charging station / swapping station

Observations

Find the fastest route from s to t:



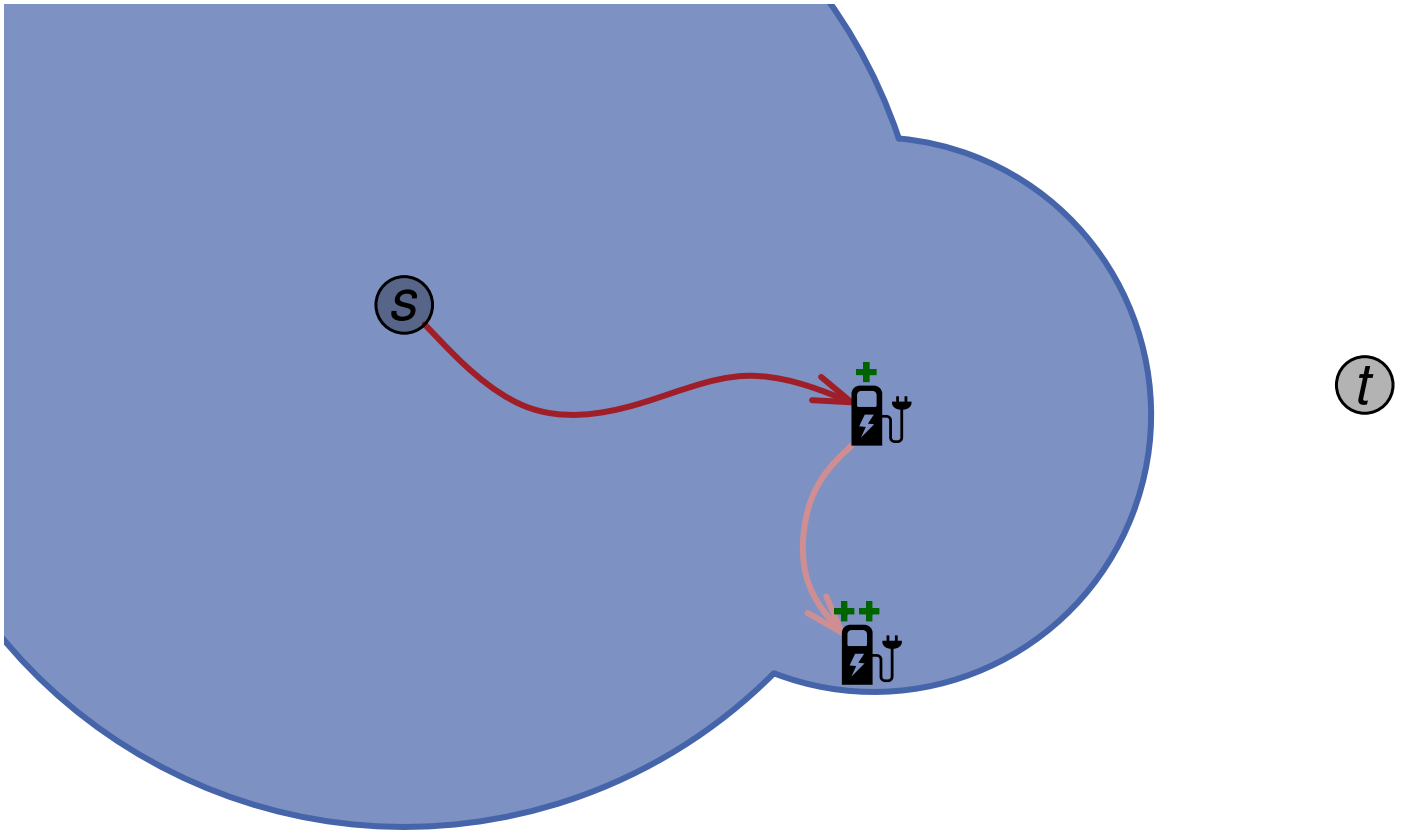
Reachable area

Charging station

Fast charging station / swapping station

Observations

Find the fastest route from s to t:



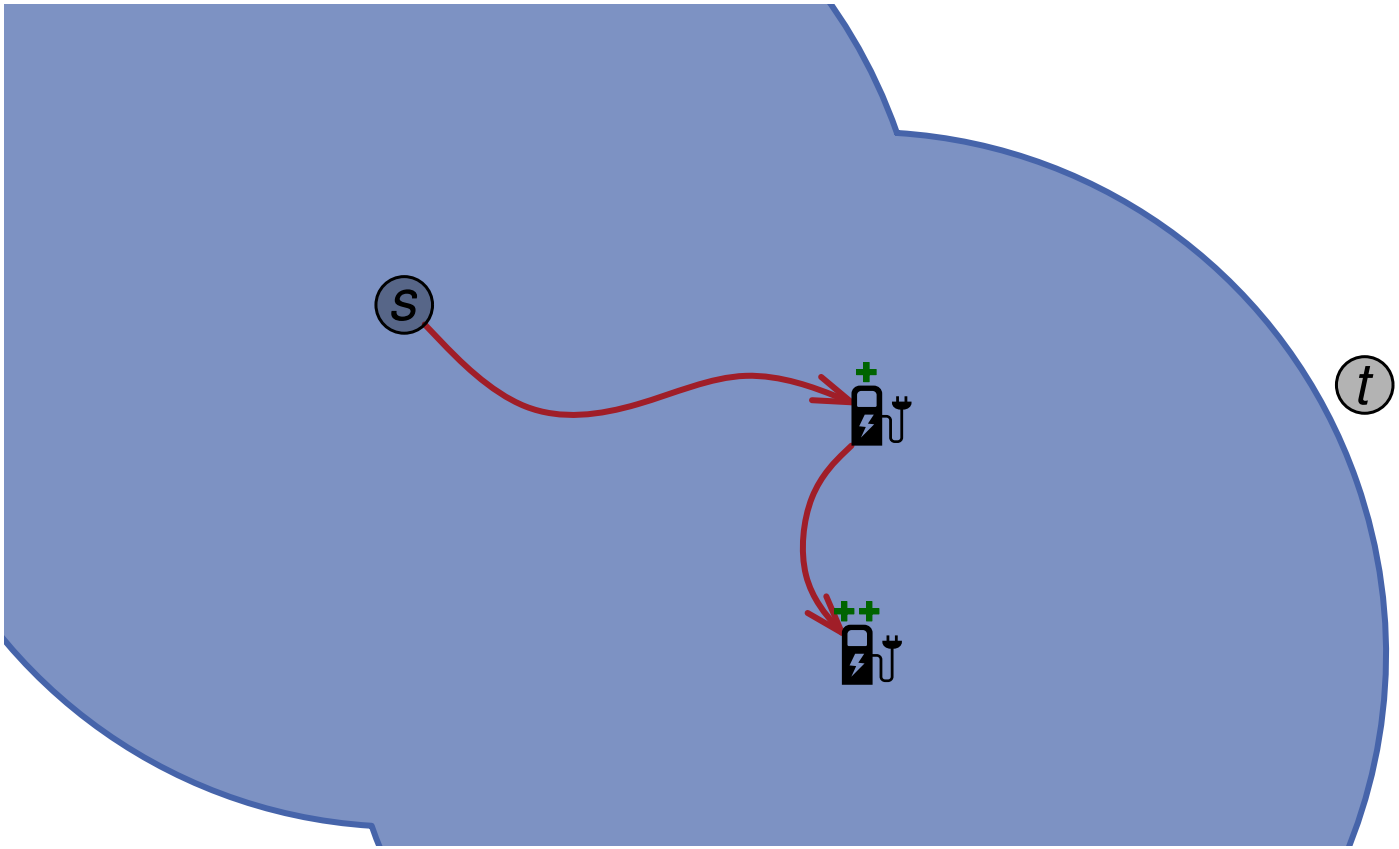
Reachable area

Charging station

Fast charging station / swapping station

Observations

Find the fastest route from s to t:



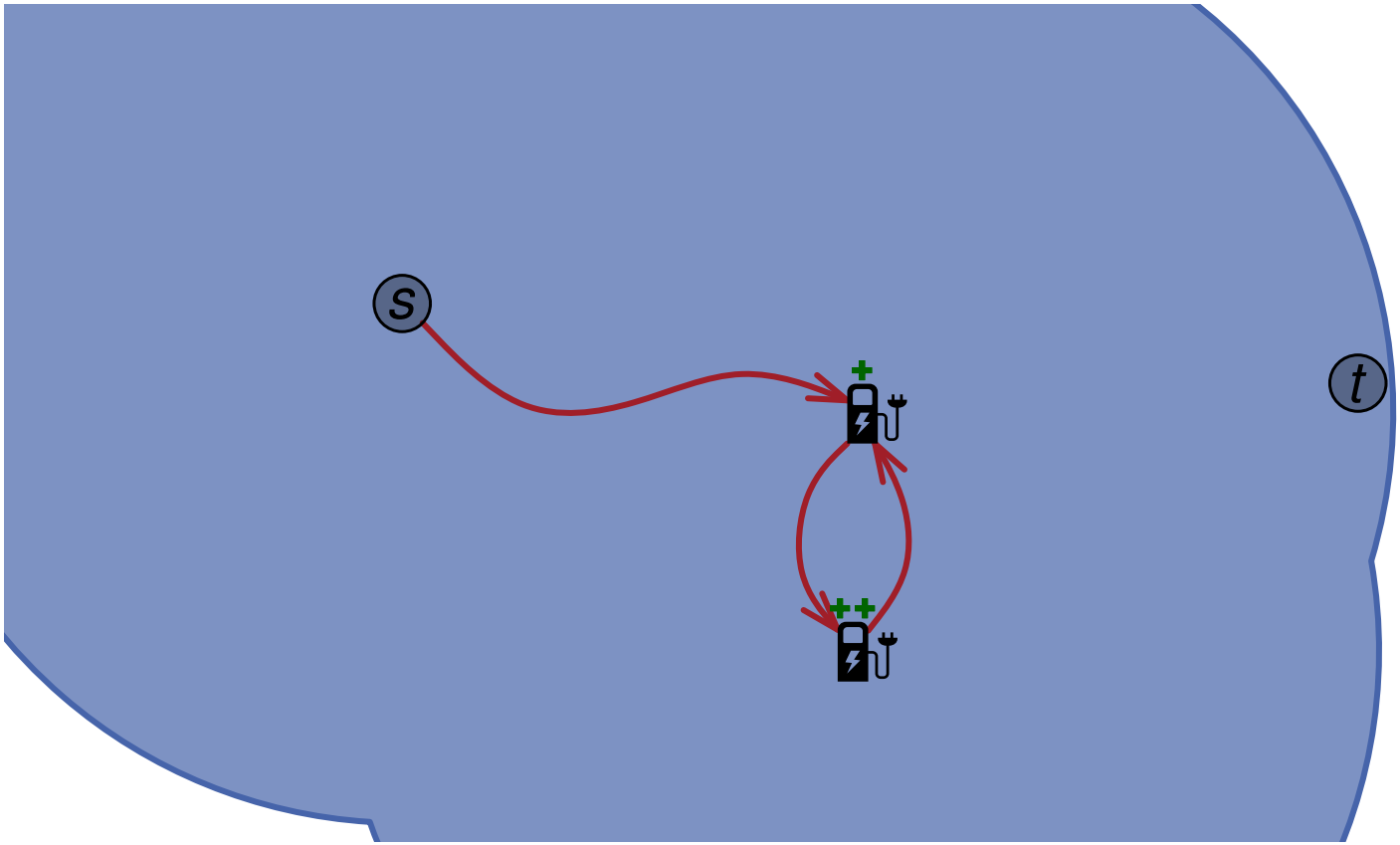
Reachable area

Charging station

Fast charging station / swapping station

Observations

Find the fastest route from s to t:



Reachable area

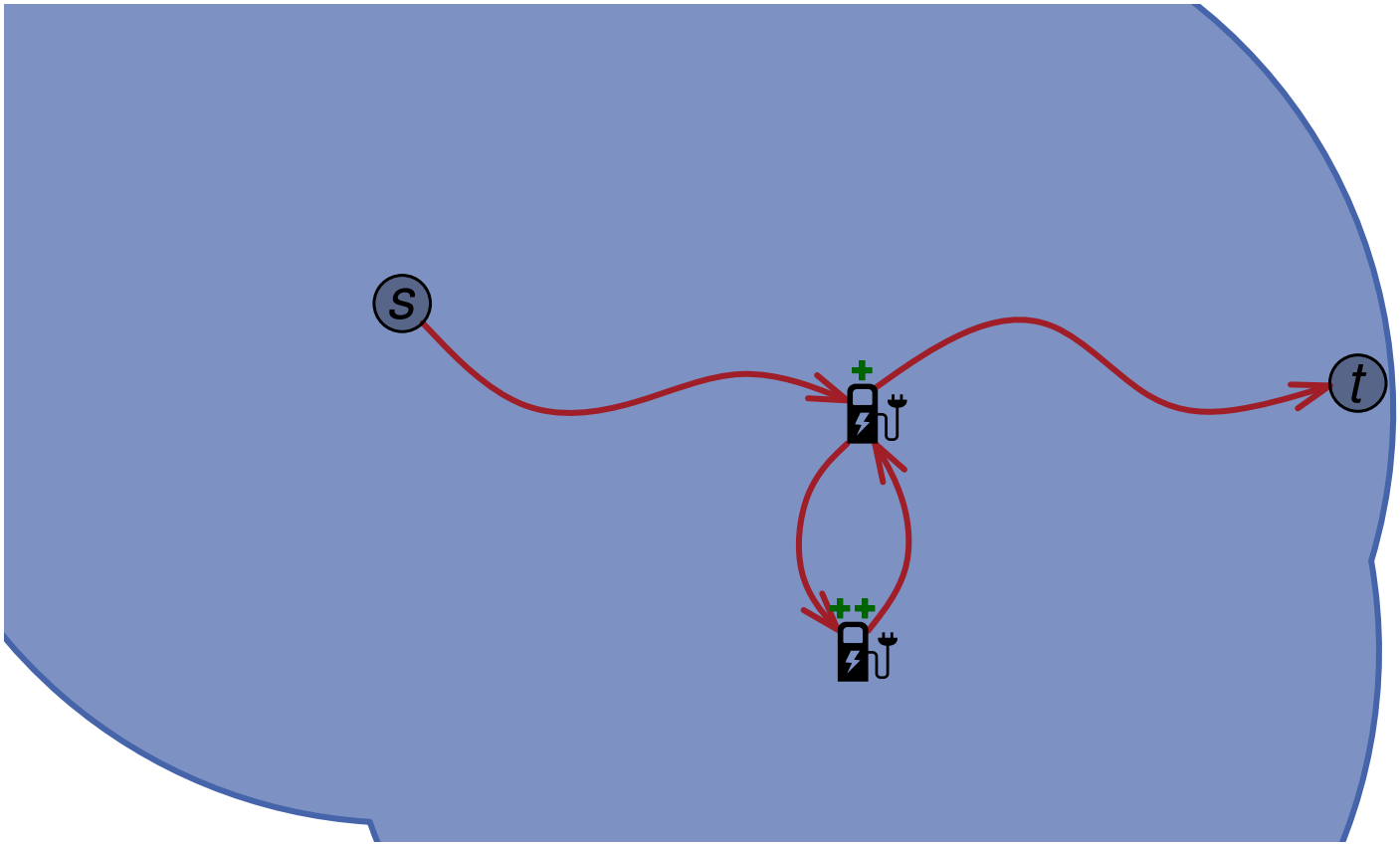
Charging station

Fast charging station / swapping station

Observations

Find the fastest route from s to t:

- Fastest route may contain cycles



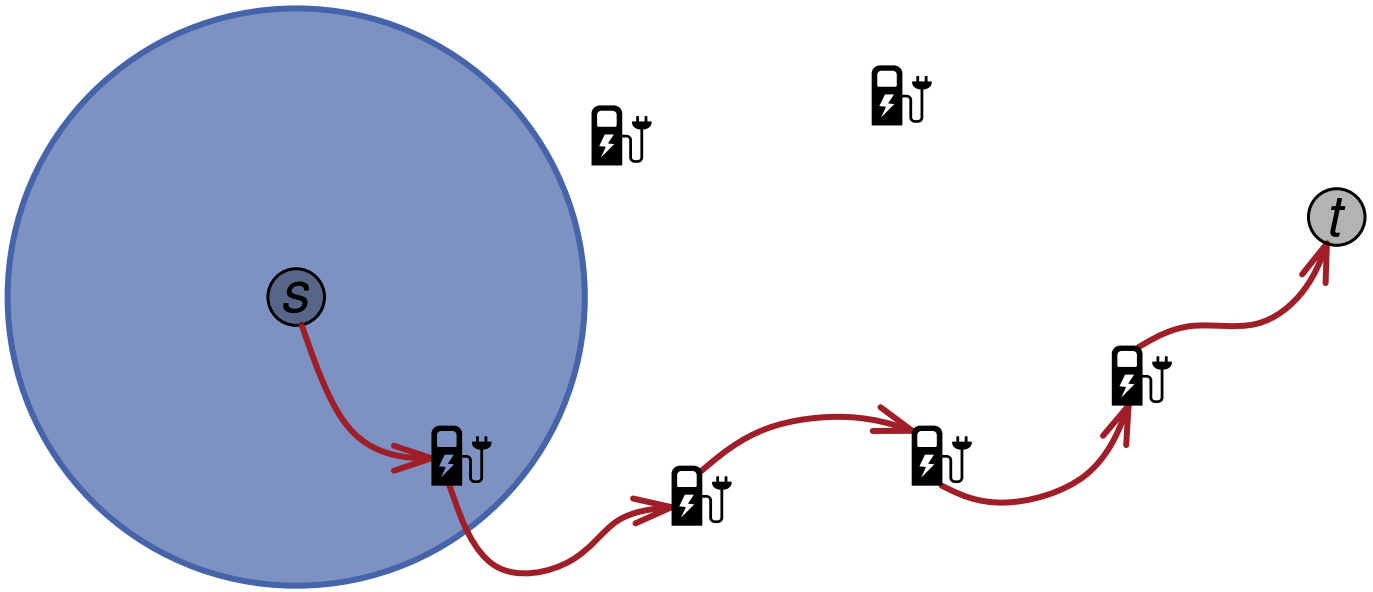
Reachable area

Charging station

Fast charging station / swapping station

Observations

Find the fastest route from s to t:



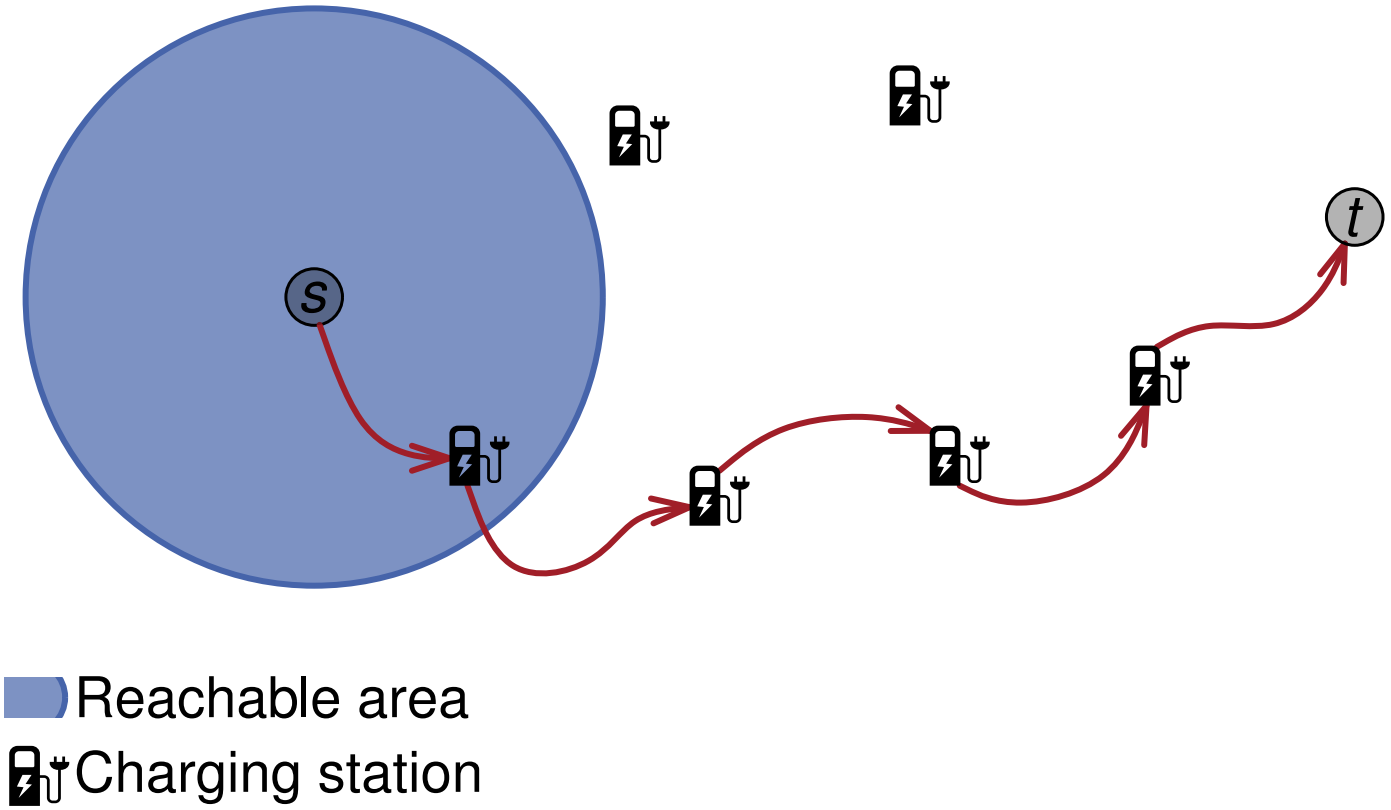
Reachable area

Charging station

Observations

Find the fastest route from s to t:

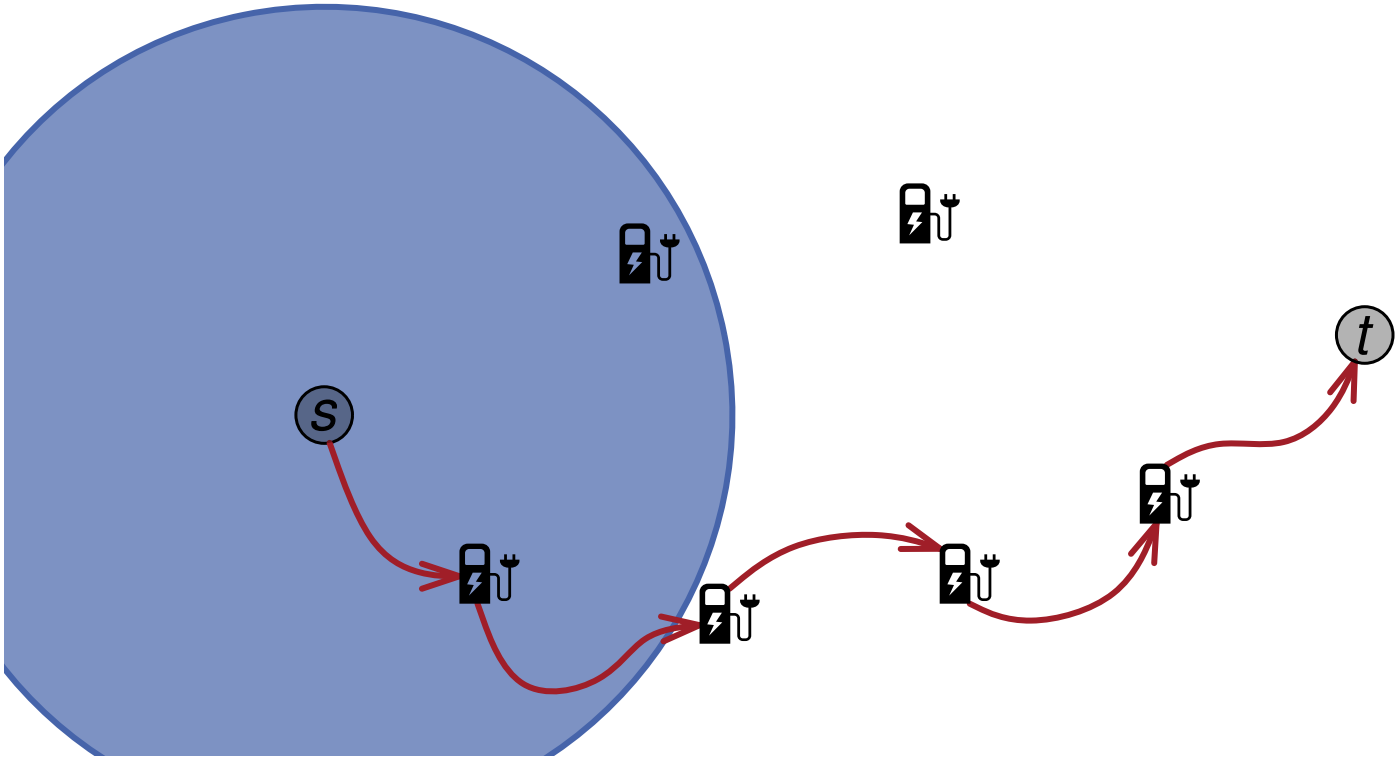
- Larger battery $\Rightarrow$ simpler problem ?



Observations

Find the fastest route from s to t:

- Larger battery $\Rightarrow$ simpler problem ?



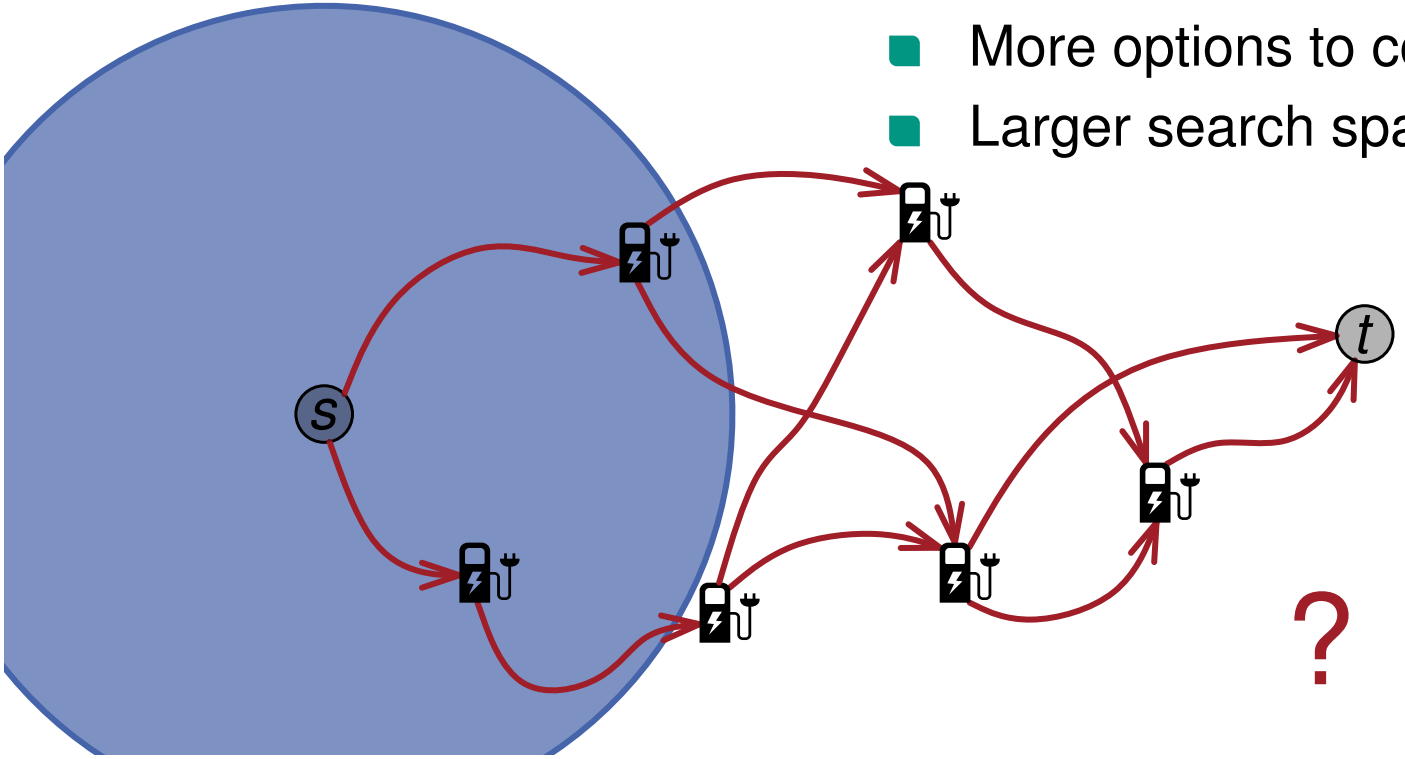
Reachable area

Charging station

Observations

Find the fastest route from s to t:

- Larger battery $\Rightarrow$ simpler problem ?
- More options to consider
- Larger search space



Reachable area

Charging station

Charging Function Propagation

CFP Algorithm

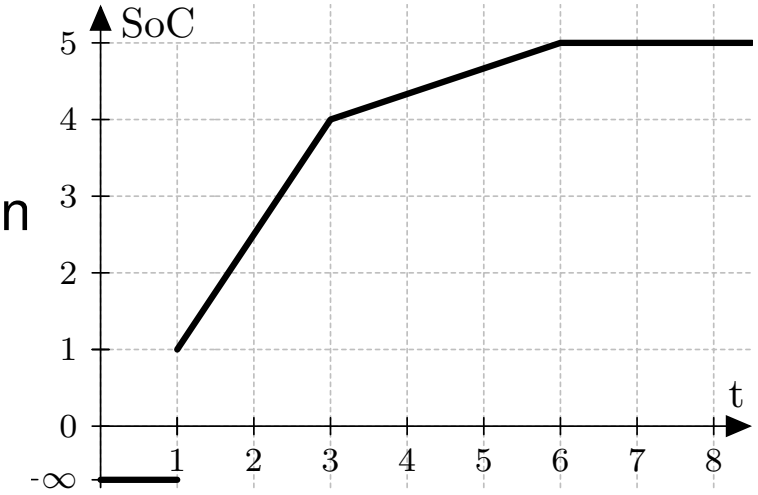
- Based on bicriteria Dijkstra
- If no charging station has been used: label = tuple (travel time, SoC)
- Per vertex: Maintain set of Pareto-optimal labels

Problem: When reaching a charging station: How long to stay?

- Depends on the remaining path to target
- Optimal state-of-charge for departure yet unknown

Solution:

- Delay this decision!
- Keep track of last passed charging station
- Labels represent charging tradeoffs



CHArge

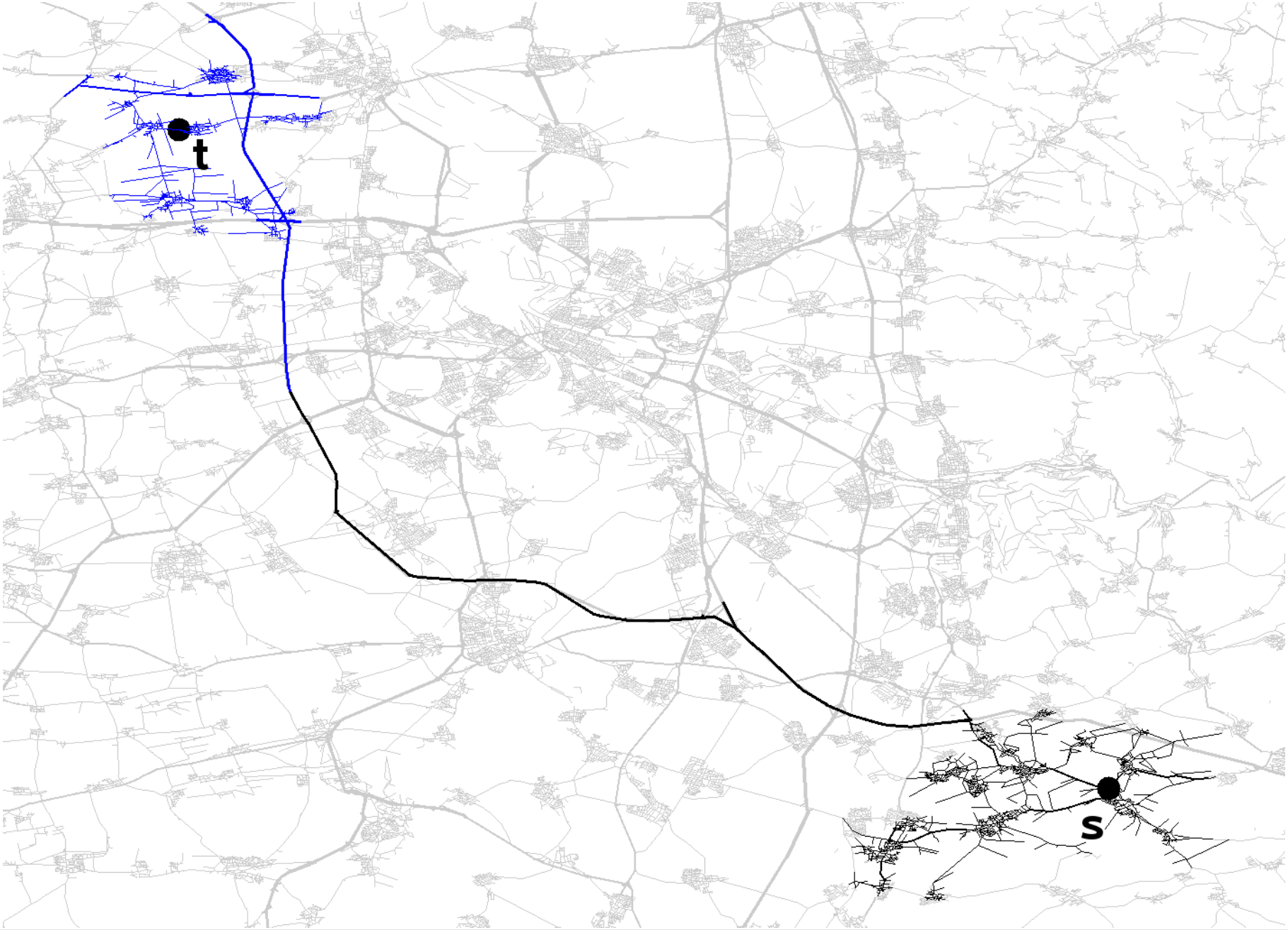
CHArge = CH & A* & CFP:

- Combines CFP with speedup techniques
- Can handle arbitrary charging station types

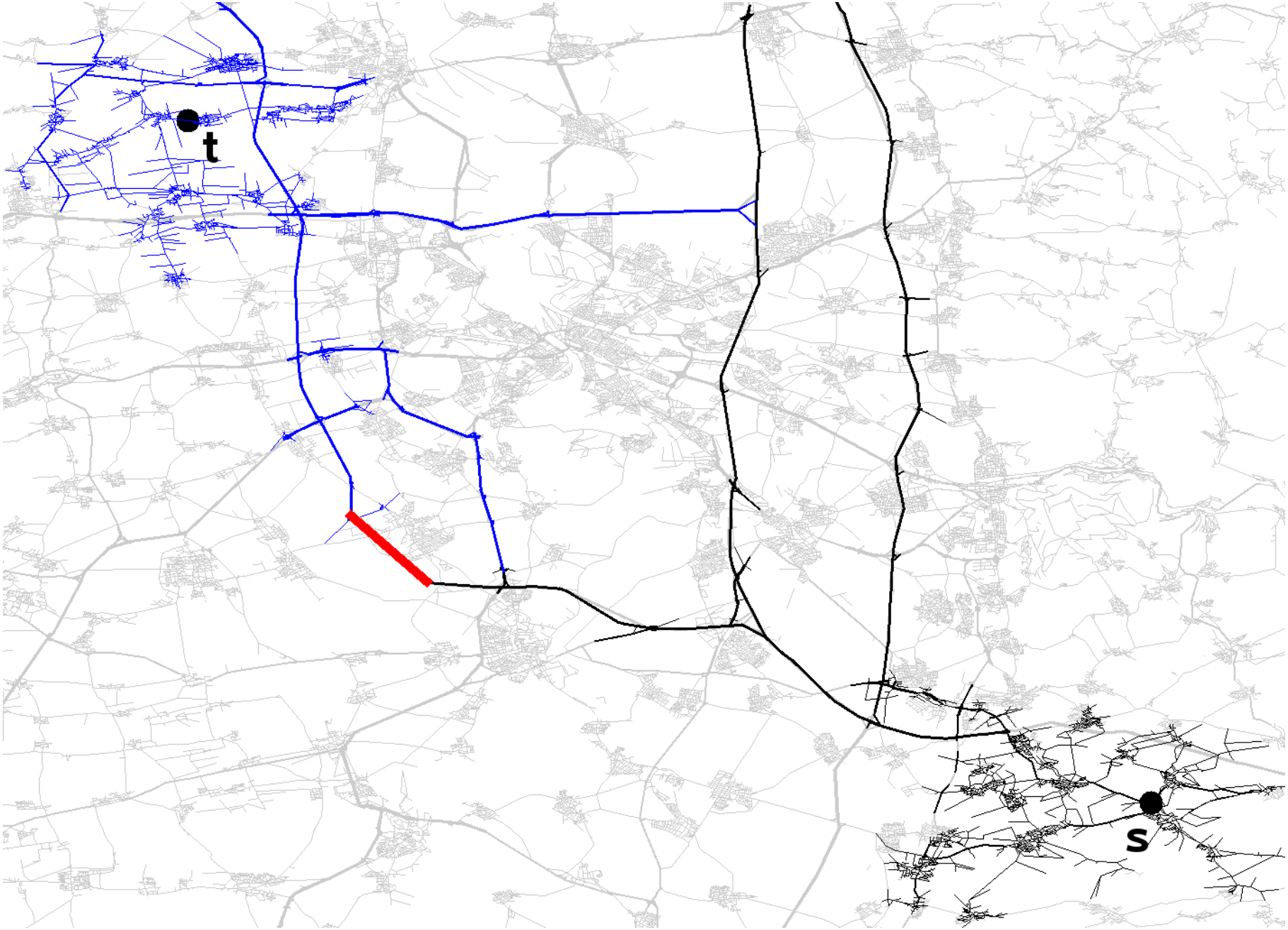
Experiments:

- Moderate preprocessing times
  Europe ∼30 min; Germany ∼5 min
- Fast queries on continental-sized networks
  Europe ∼1 min; Germany ∼1 sec
- Even better results possible, using heuristics
  Europe ∼0.1–1 sec; Germany ∼20–100 ms
  often optimal solutions, mean error ∼1%

Customizable Route Planning



Customizable Route Planning



Real-World Metrics

- Distance
- Pedestrian
- Travel time, but don't use toll roads
- Travel time, avoid left turns, height restrictions, . . .
- Traffic Congestion, accidents, . . .

Problem

- Preprocessing is metric-dependent
- State-of-the-art algorithms tailored to travel time
  heavily exploit 'hierarchy' of road categories

Naive solution

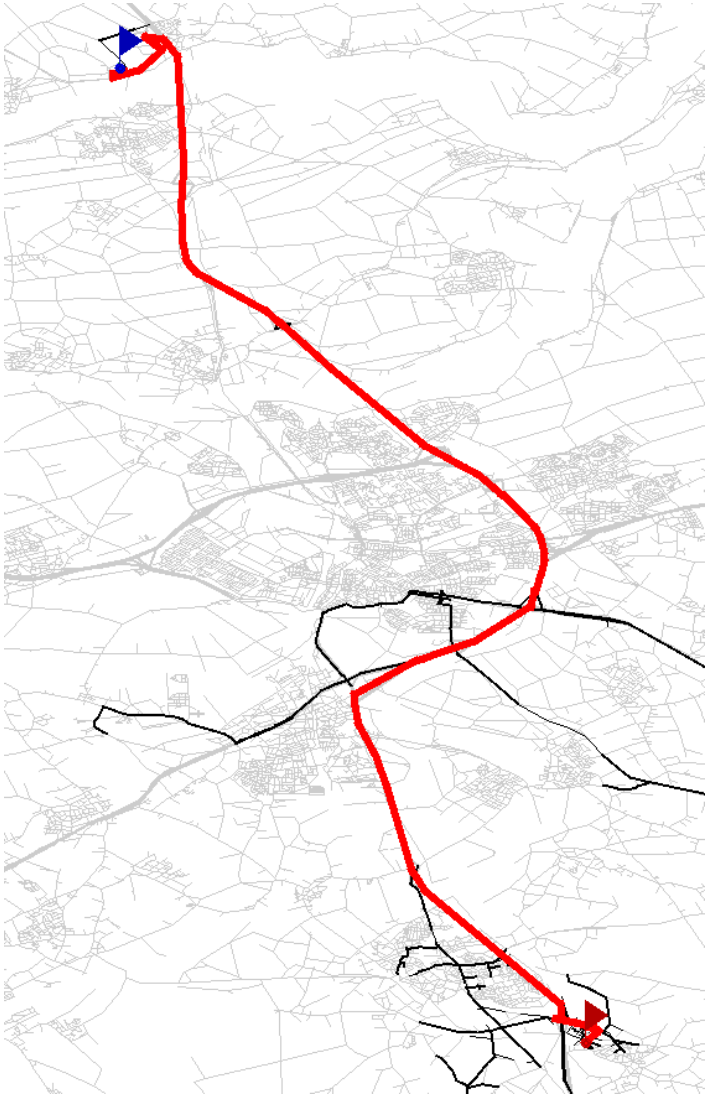
- Compute preprocessing for each metric
- Preprocessing and query time increase significantly
- Higher space overhead

$\Rightarrow$ **Metric customization**

Shortest Path Computation

Two-phase:

- Preprocessing (slow): compute additional data
- Query (fast): answer *st*-queries using data from preprocessing



Institute for Theoretical Informatics
Chair Algorithmics

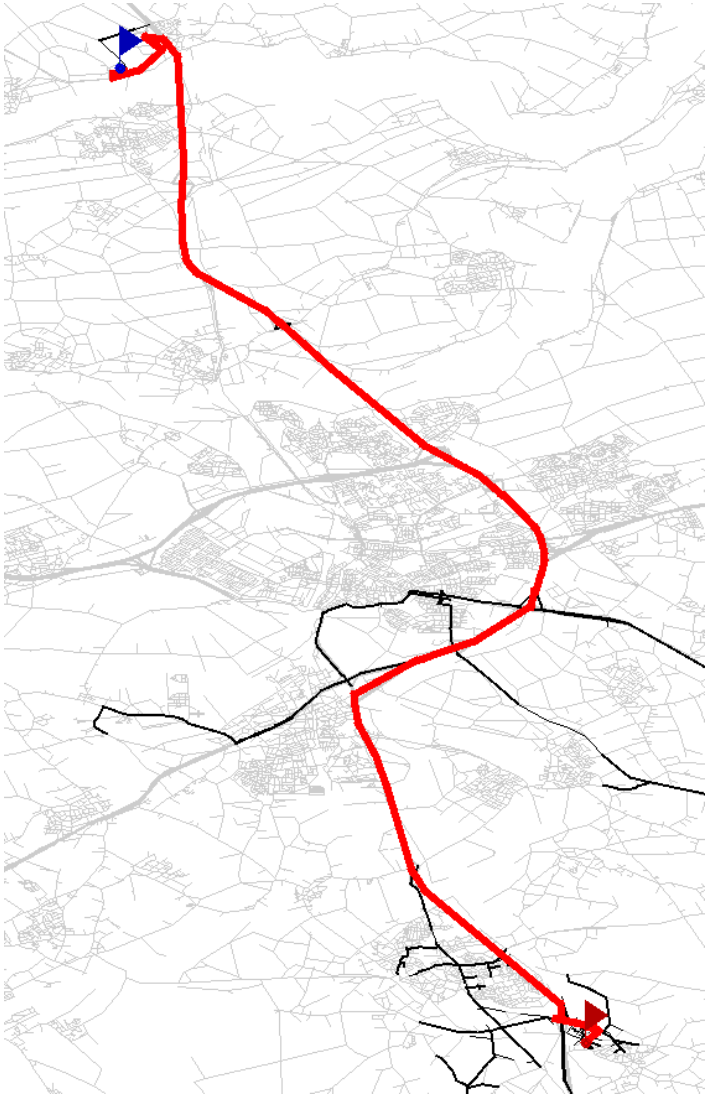
Shortest Path Computation

Two-phase:

- Preprocessing (slow): compute additional data
- Query (fast): answer *st*-queries using data from preprocessing

Three-phase:

- Preprocessing (slow): compute additional weight-independent data
- Customization (reasonably fast): introduce weights
- Query (fast): answer *st*-queries using data from preprocessing and customization



CH Revisited

Metric-dependent orders:

- Node order determines CH performance
- Many ordering algorithms exist
- Some fast, some slow, some specific to certain graph classes, . . .
- **But**: Best order depends on the weights

CH Revisited

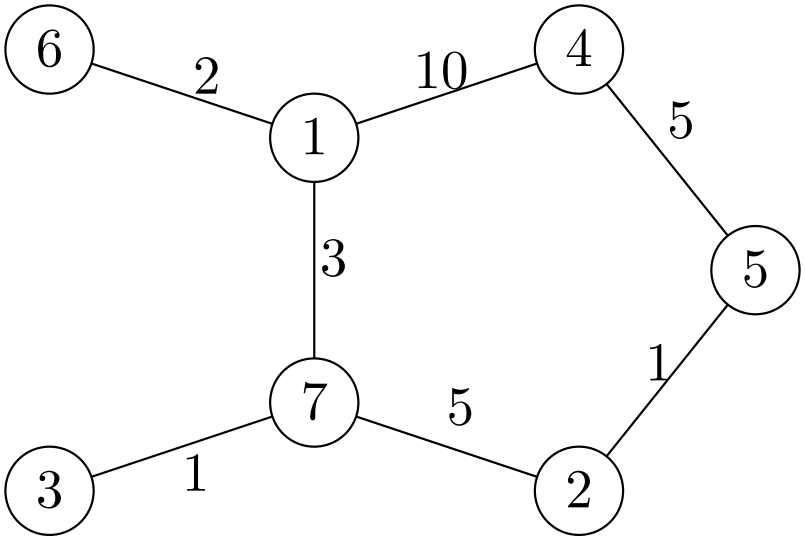
Metric-dependent orders:

- Node order determines CH performance
- Many ordering algorithms exist
- Some fast, some slow, some specific to certain graph classes, . . .
- **But**: Best order depends on the weights

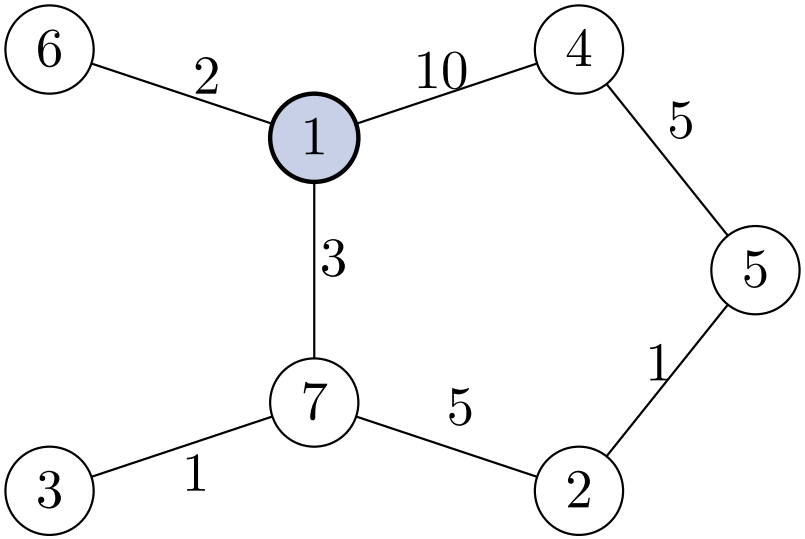
Metric-independent orders:

- Is there an order that is good for every weight?
  (but not necessarily best)
- Core idea of 3-phase CH

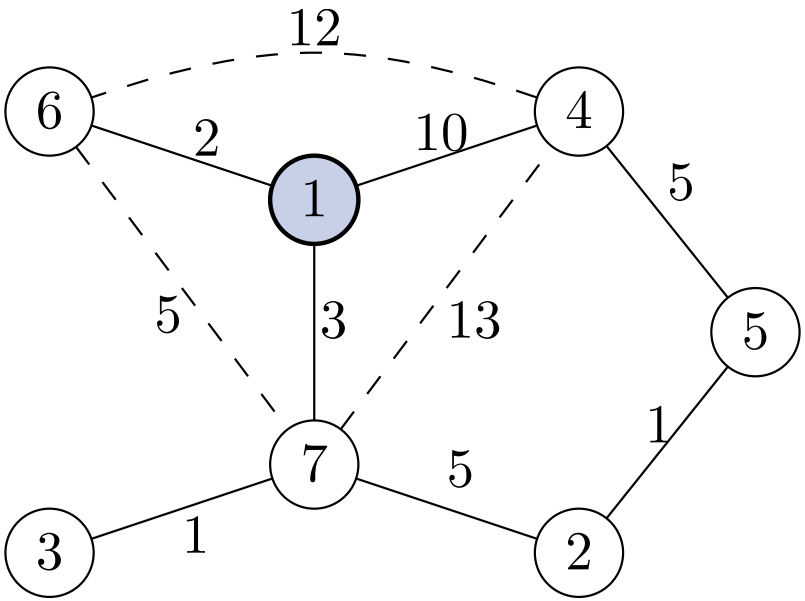
Contraction Hierarchies



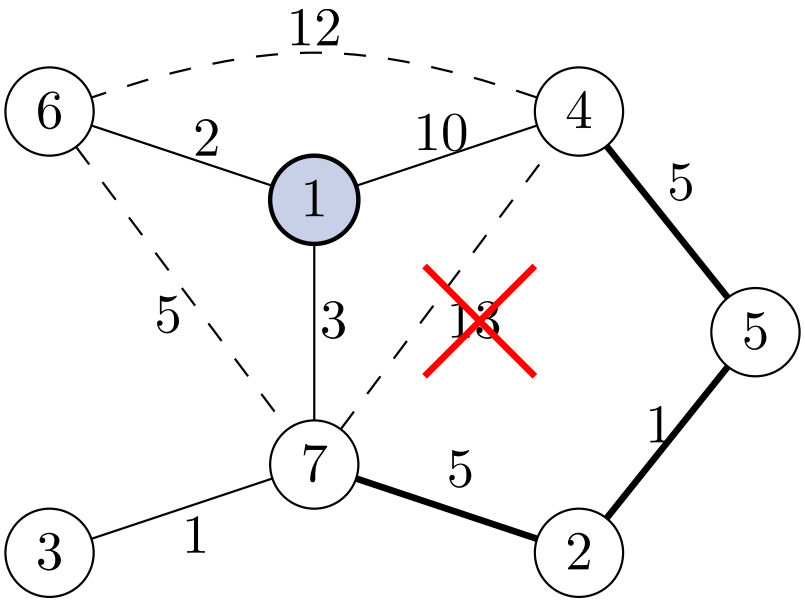
Contraction Hierarchies



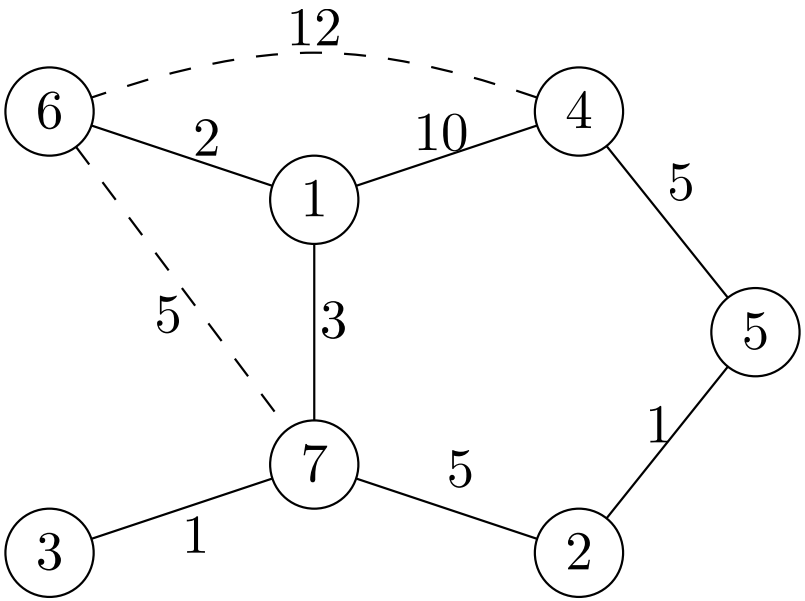
Contraction Hierarchies



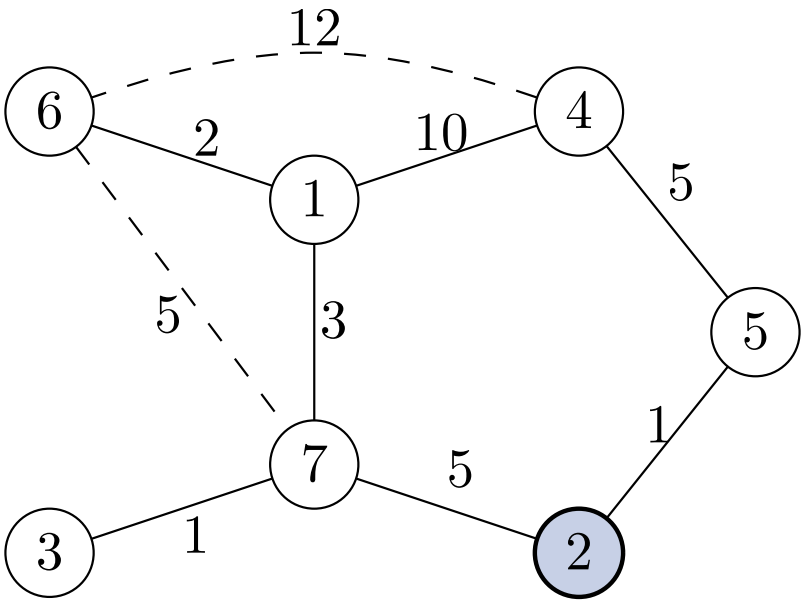
Contraction Hierarchies



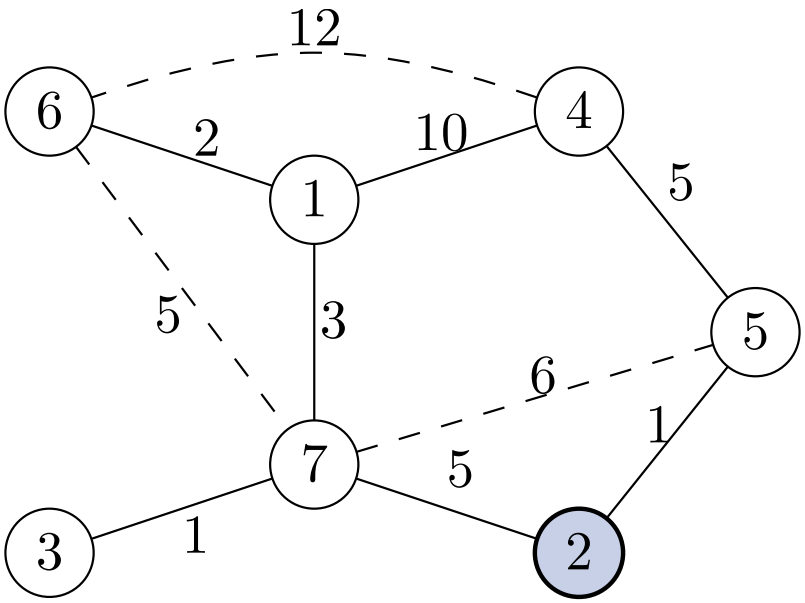
Contraction Hierarchies



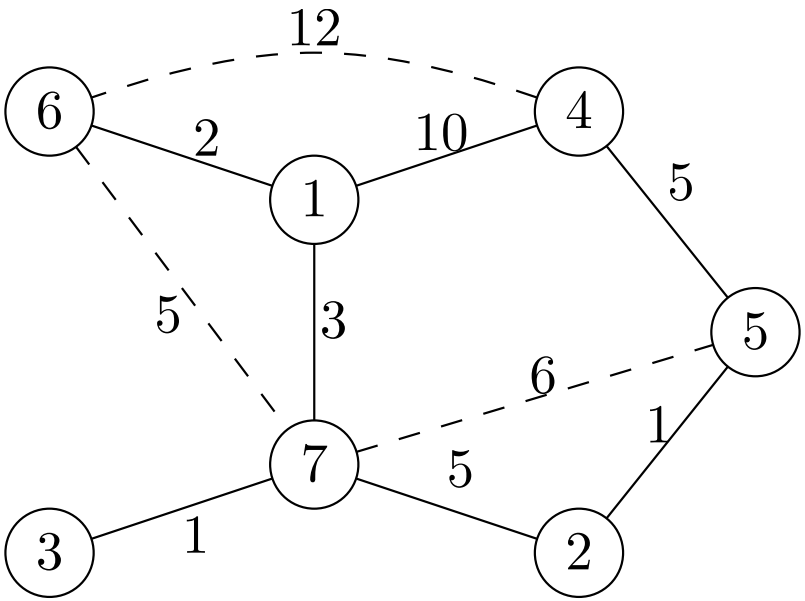
Contraction Hierarchies



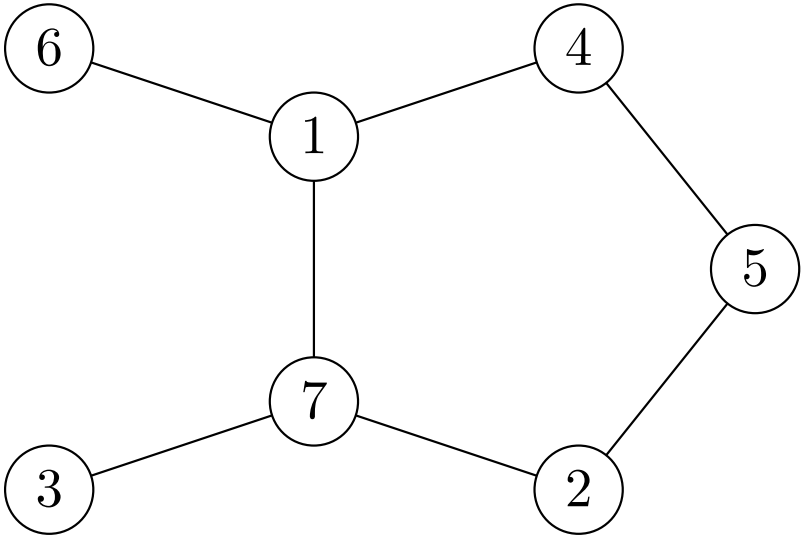
Contraction Hierarchies



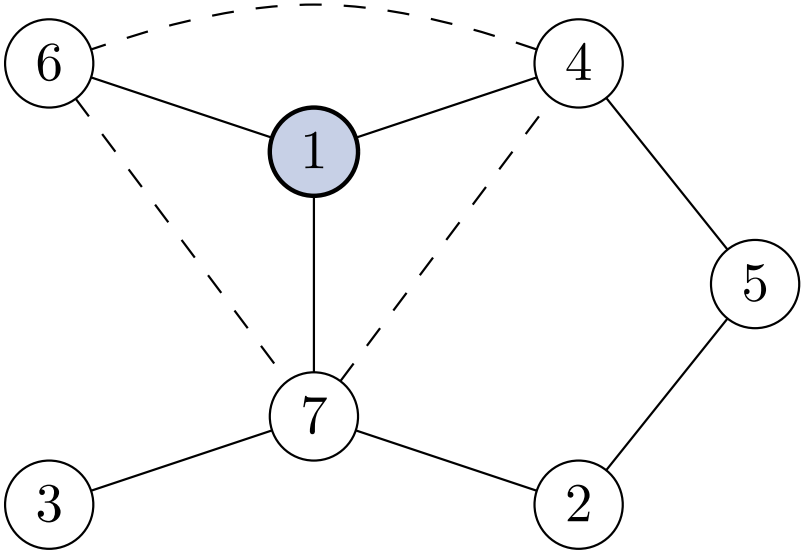
Contraction Hierarchies



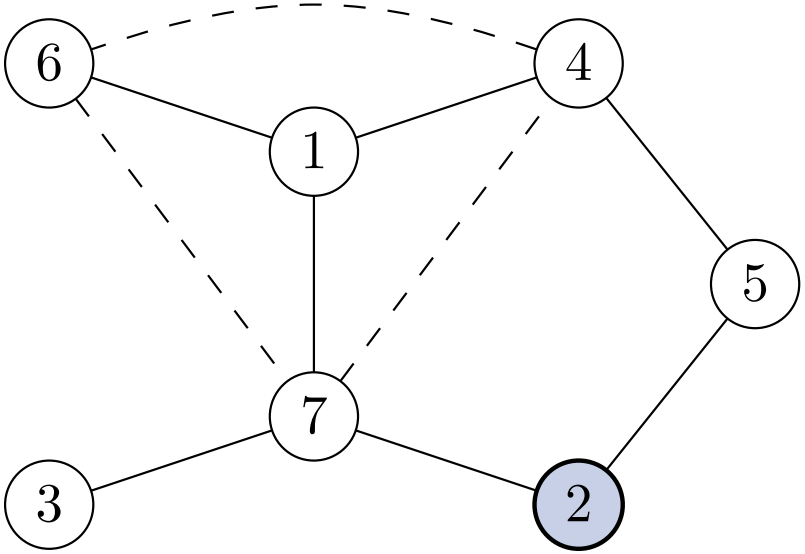
Graph Fill-In



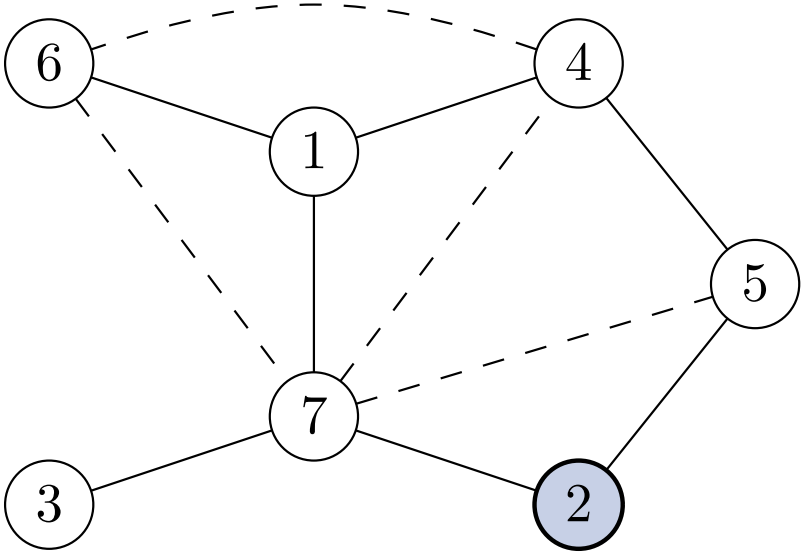
Graph Fill-In



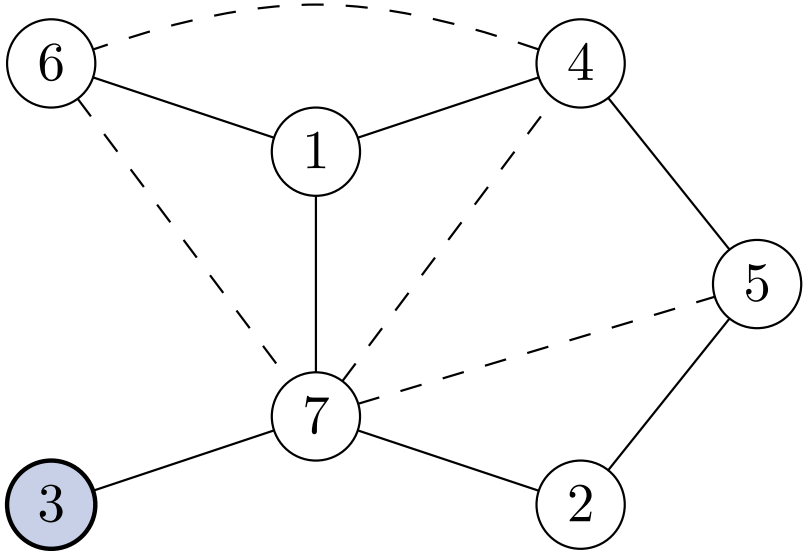
Graph Fill-In



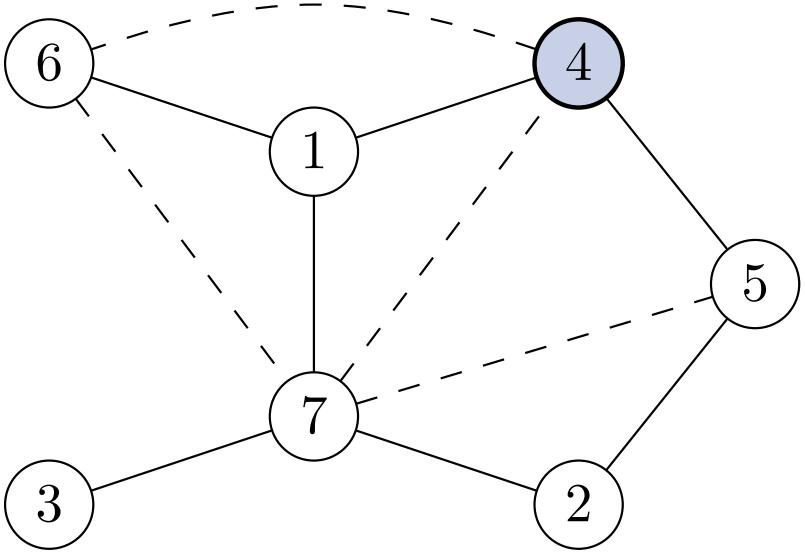
Graph Fill-In



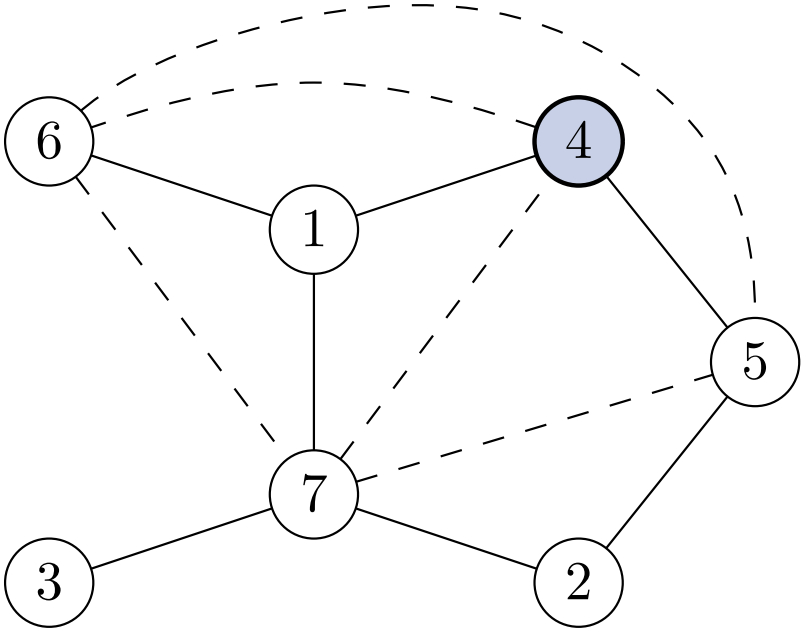
Graph Fill-In



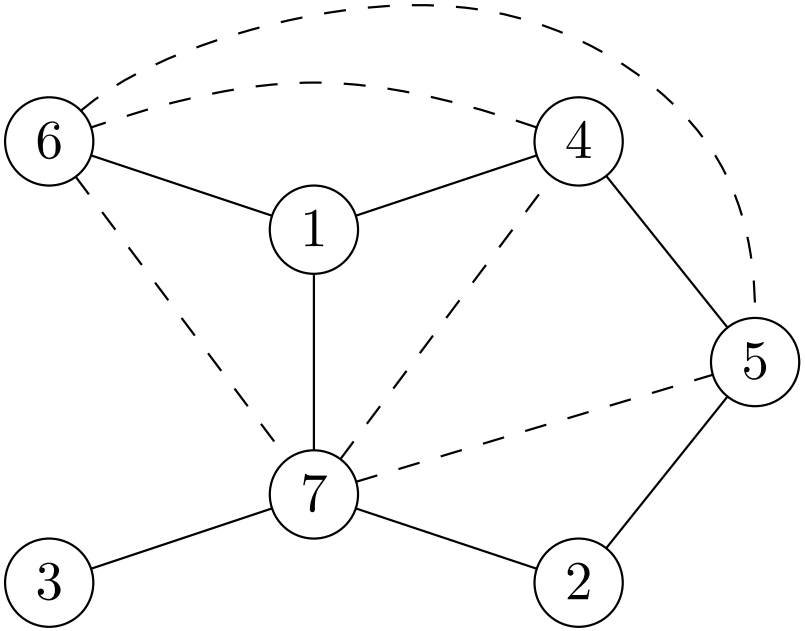
Graph Fill-In



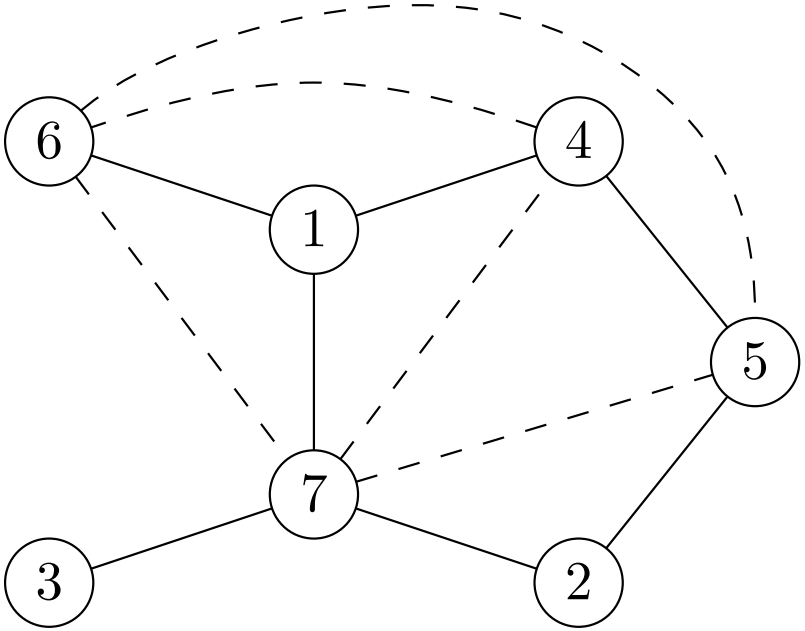
Graph Fill-In

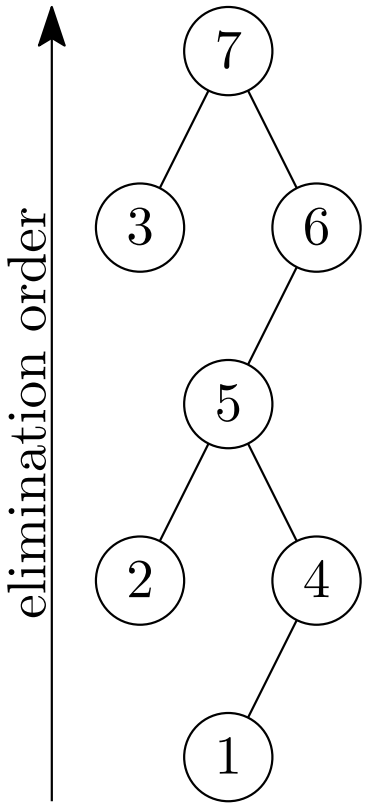


Graph Fill-In



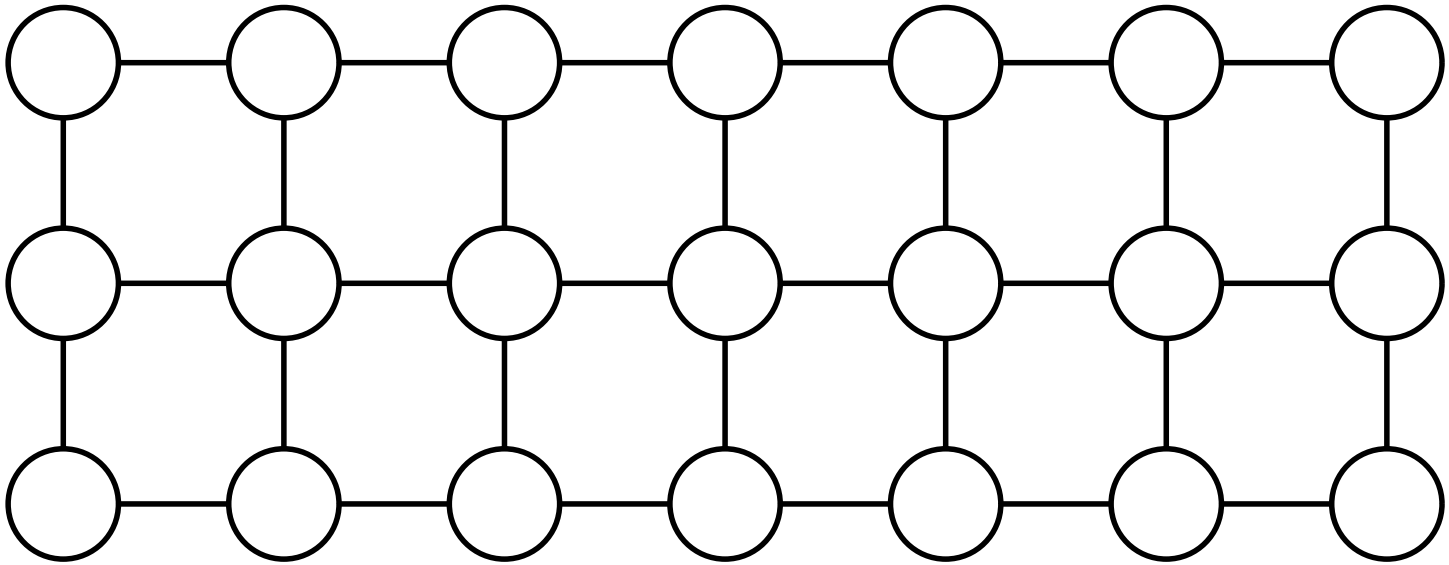
Graph Fill-In





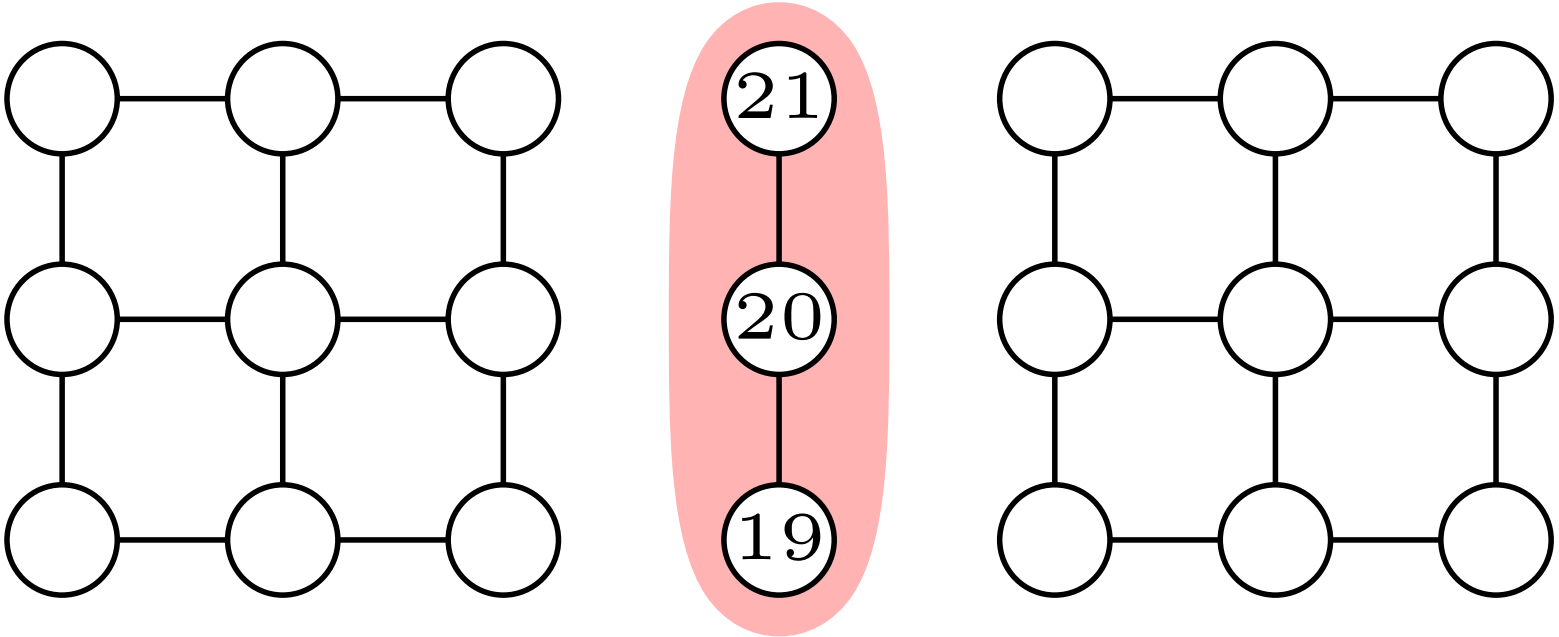
elimination tree

Nested Dissection (ND)

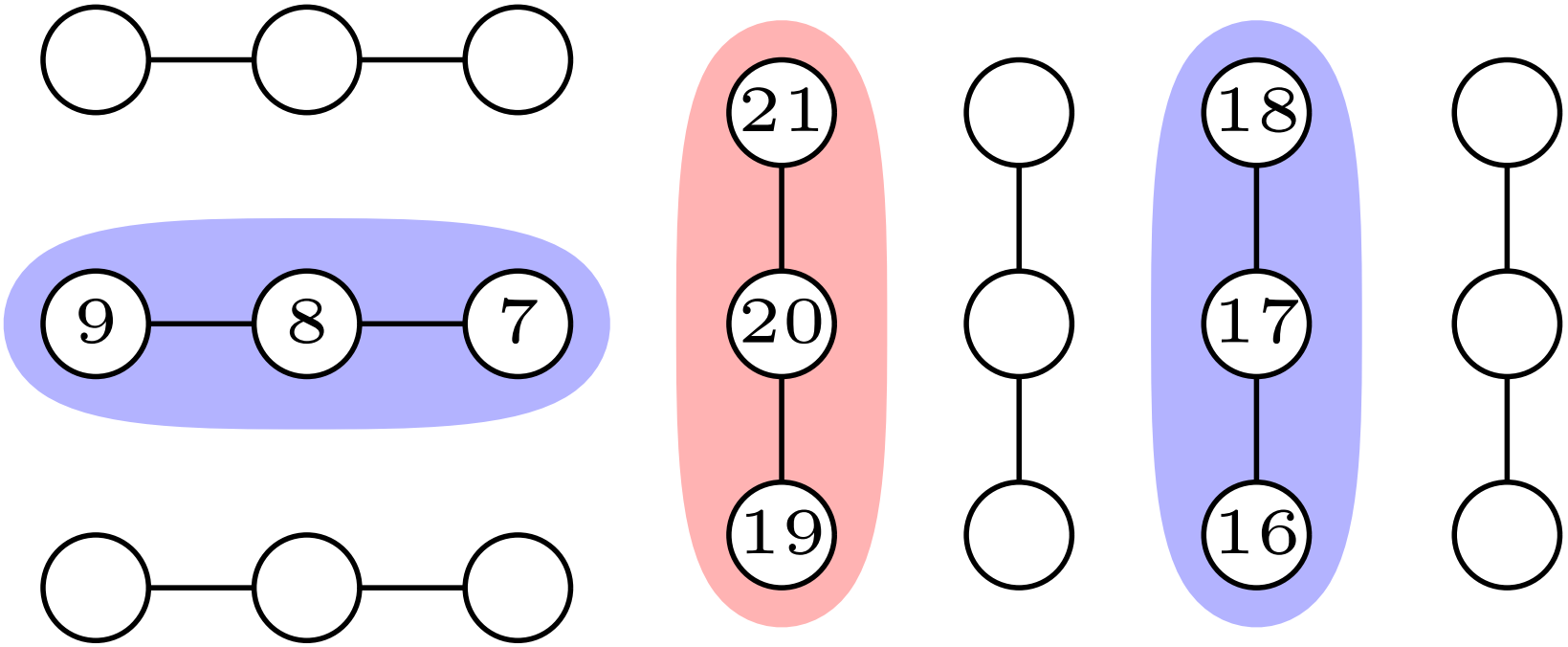


Institute for Theoretical Informatics
Chair Algorithmics

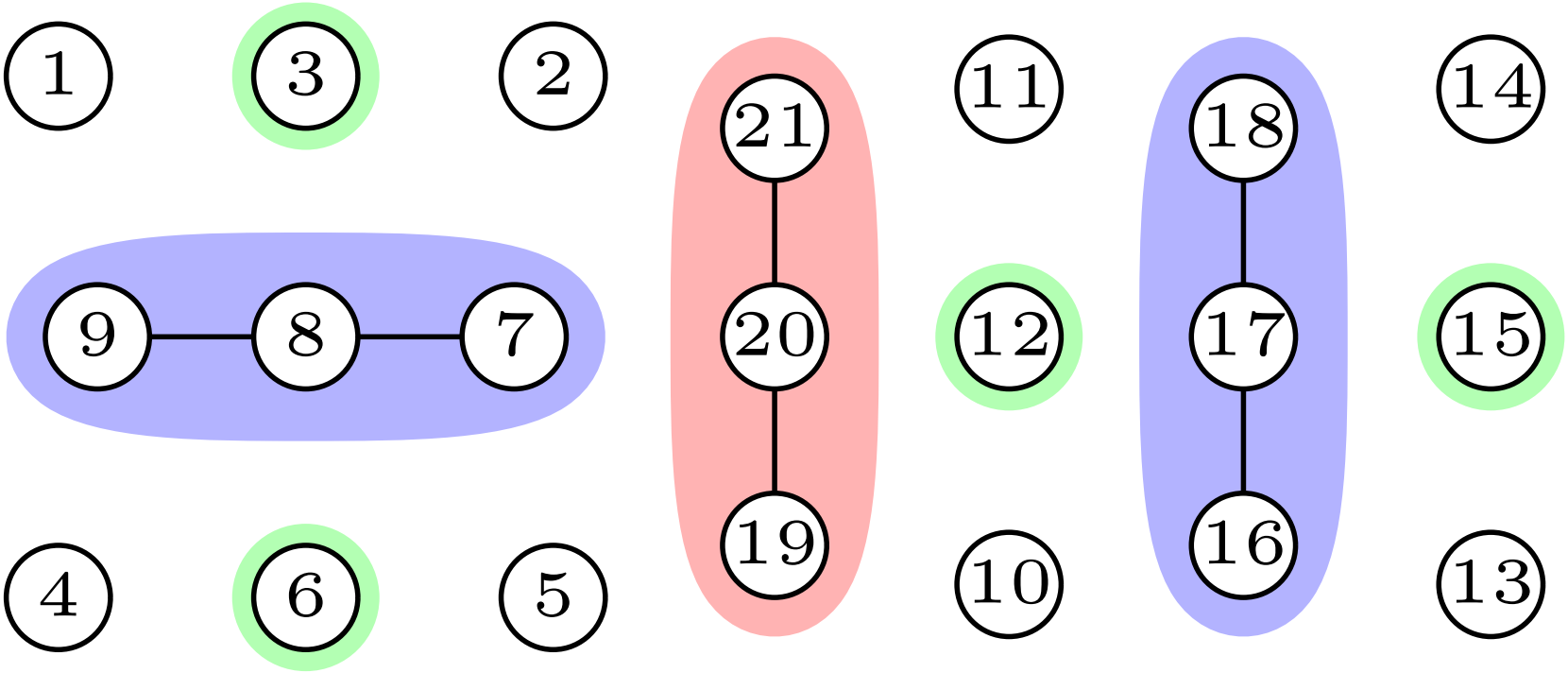
Nested Dissection (ND)



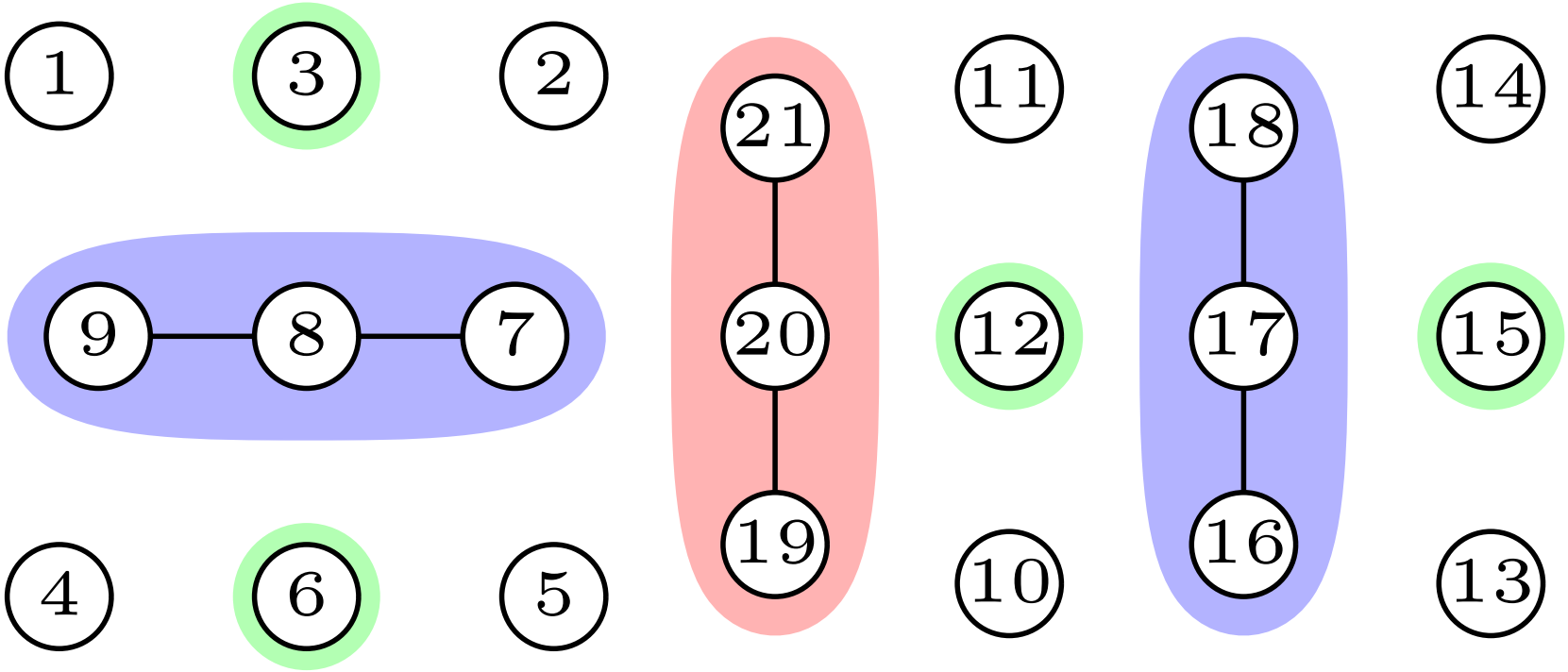
Nested Dissection (ND)



Nested Dissection (ND)



Nested Dissection (ND)



ND ordering from recursive $O(n^\beta)$ balanced separators
yields elimination tree of height $O(n^\beta)$

Consequences [BCRW13, DSW16]

Theoretical guarantee:

- ND-ordering yields search space guarantee of $O(n^\beta)$ nodes
- $O(\sqrt{n})$ rec. balanced separators yield guarantee of $O(n)$ edges
- Planar graphs have $O(\sqrt{n})$ recursive balanced separators

Consequences [BCRW13, DSW16]

Theoretical guarantee:

- ND-ordering yields search space guarantee of $O(n^{\beta})$ nodes
- $O(\sqrt{n})$ rec. balanced separators yield guarantee of $O(n)$ edges
- Planar graphs have $O(\sqrt{n})$ recursive balanced separators

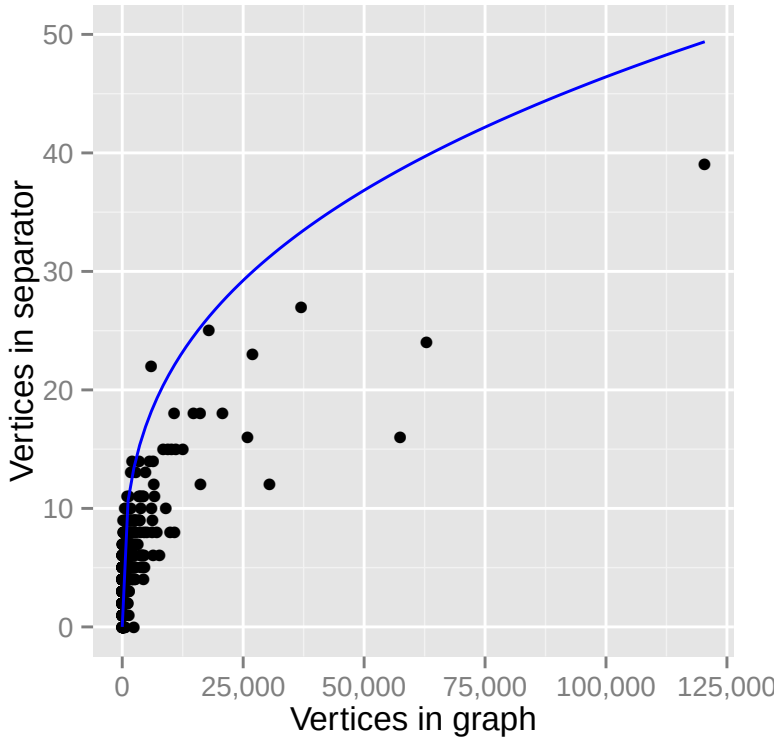
Practical impact:

- Contraction ordering that is weight-independent
- Minimum vs maximum contraction hierarchies
- Customizable contraction hierarchies (CCH)

CCH : Three-Phase Approach

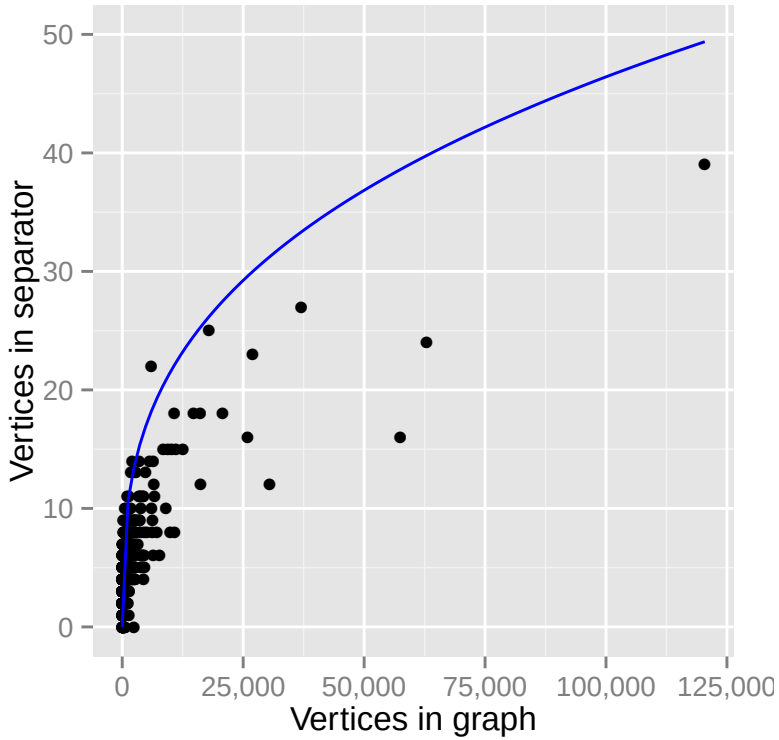
- **Preprocessing**
  - Compute ND-order
  - Solve balanced graph bisection subproblem
  - Compute fill-in (shortcuts)
- **Customization**
  - Add weights to shortcuts
    - Enumerate lower triangles in CH
- **Query**
  - Existing CH-query works unmodified
  - **Alternative**: Elimination-tree query

Node Separators



Separators for graph of Karlsruhe and ≈50 km vicinity
Blue function is $y = \sqrt[3]{x}$

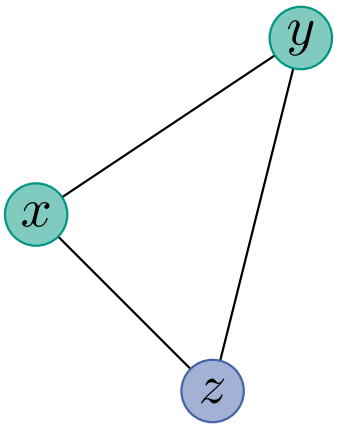
Node Separators



Separators for graph of Karlsruhe and ≈50 km vicinity
Blue function is $y = \sqrt[3]{x}$

Assumption: Road graphs have $O(\sqrt[3]{x})$ rec., balanced separators

Customization



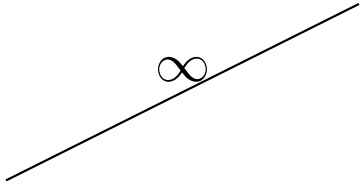
- Lower triangle inequality holds if for each edge (x, y) and all its lower triangles $\{x, y, z\}$:

$$w(x, y) \leq w(x, z) + w(z, y)$$

- Initialization: Weights of shortcuts are set ∞
- Customization: Adjust weights on shortcuts to fulfill lower triangle inequality

Customization

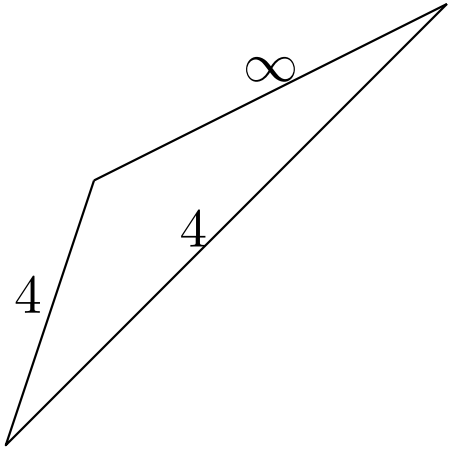
Basic Customization



Shortcut edge

Customization

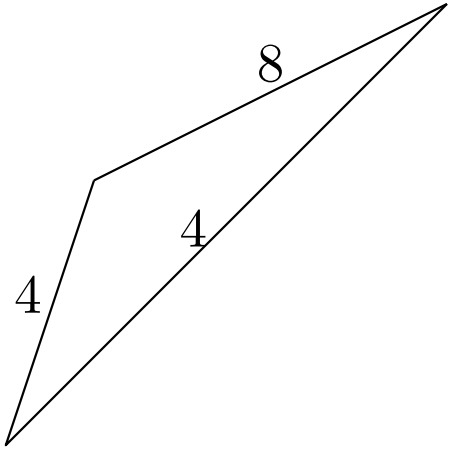
Basic Customization



Lower triangle inequality must hold for every lower triangle

Customization

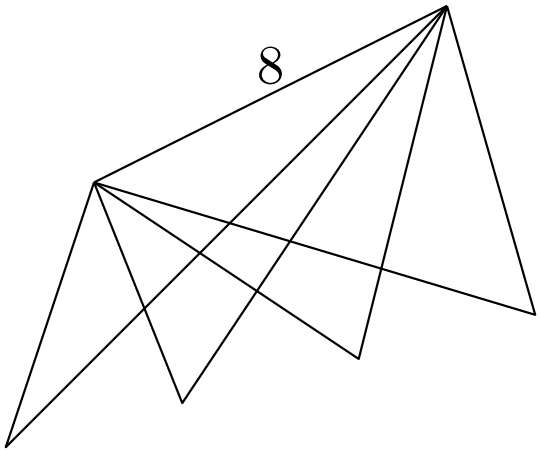
Basic Customization



Decrease weight until the inequality holds

Customization

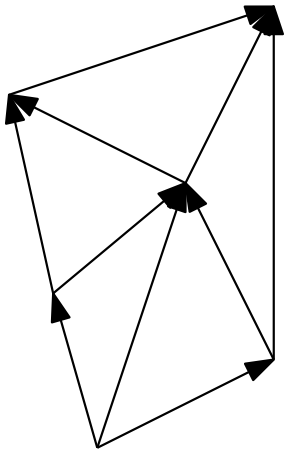
Basic Customization



Do this for every lower triangle

Customization

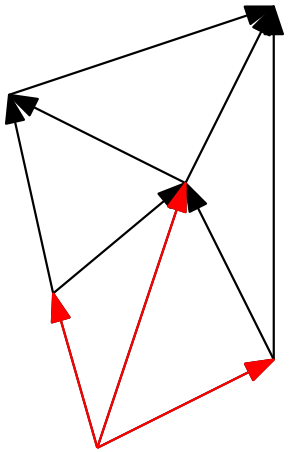
Basic Customization



In which order do we process the edges?

Customization

Basic Customization

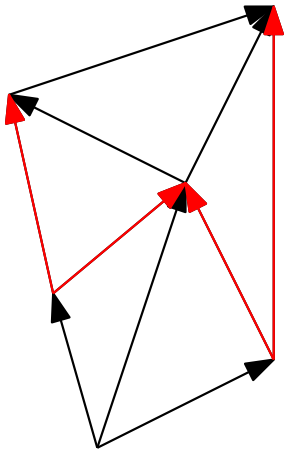


Iterate over edges by increasing level

Institute for Theoretical Informatics
Chair Algorithmics

Customization

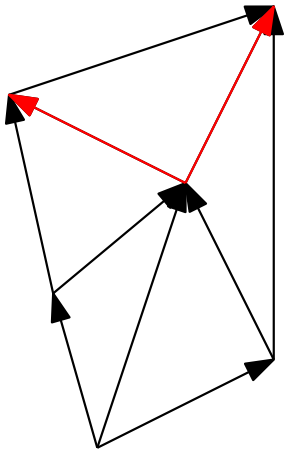
Basic Customization



Iterate over edges by increasing level

Customization

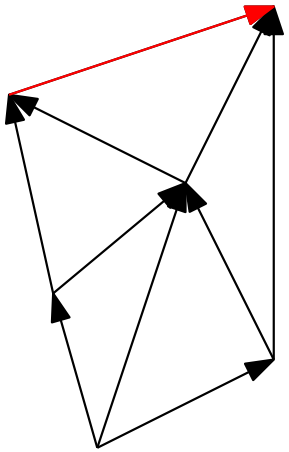
Basic Customization



Iterate over edges by increasing level

Customization

Basic Customization

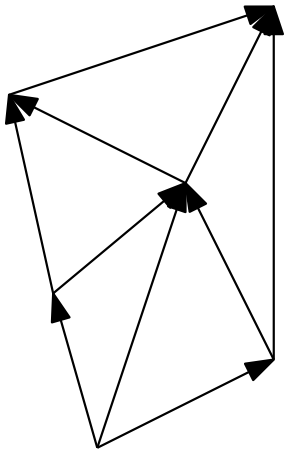


Iterate over edges by increasing level

Institute for Theoretical Informatics
Chair Algorithmics

Customization

Basic Customization



Lower triangle inequality holds $\rightarrow$ CH-Query correct

Elimination-Tree-Query

While not at the root do:

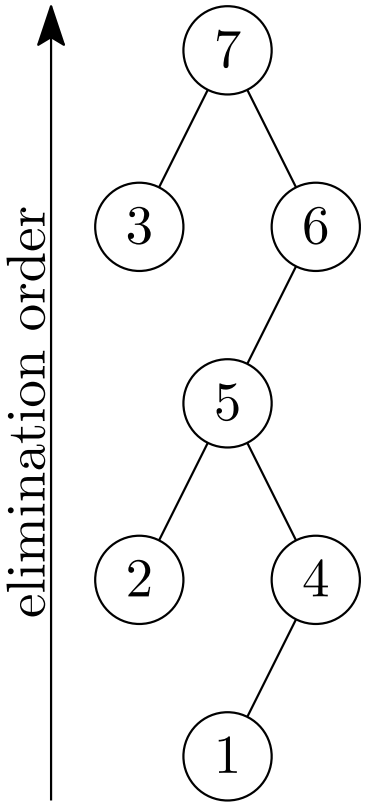
- If s comes before t in the order:
  - Relax outgoing arcs of s in its search space
  - $s \leftarrow \text{parent}(s)$
- Else:
  - Relax outgoing arcs of t in its search space
  - $t \leftarrow \text{parent}(t)$

Advantage:

- No queue
- Works with negative weights

But:

- Local queries are not faster than long distance queries



Elimination-Tree

The elimination tree is defined as following:

$$\mathrm{parent}(x) = \text{first neighbor of } x \text{ in order contracted after } x$$

Theorem: Ancestors of x = Nodes in search space of x

Elimination-Tree

The elimination tree is defined as following:

$$\mathrm{parent}(x) = \text{first neighbor of } x \text{ in order contracted after } x$$

Theorem: Ancestors of x = Nodes in search space of$x$

- $V(x)$ = ancestors of x
- $S(x)$ = nodes in search space x

Proof:

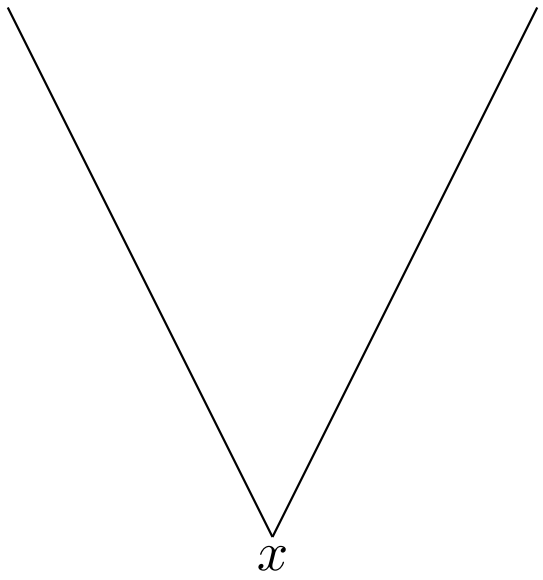
- $V(x) \subseteq S(x)$
- Holds, because path along tree is in search space
- Remains to show that $V(x) \supseteq S(x)$
- Proof by contraction

Elimination-Tree

The elimination tree is defined as following:

$$\mathrm{parent}(x) = \text{first neighbor of } x \text{ in order contracted after } x$$

Theorem: Ancestors of x = Nodes in search space of x



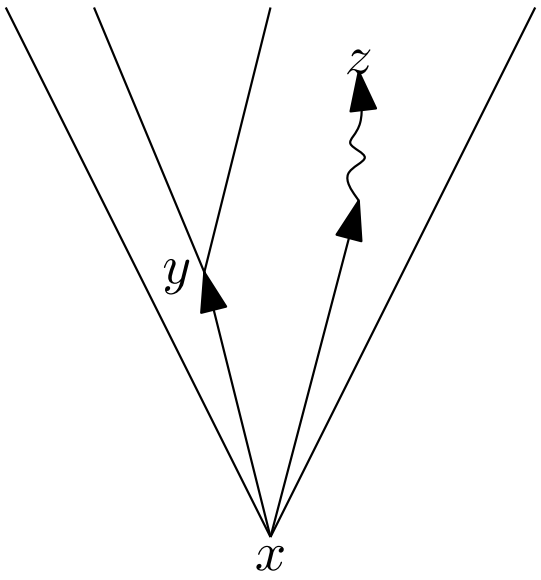
The search space of x

Elimination-Tree

The elimination tree is defined as following:

$$\mathrm{parent}(x) = \text{first neighbor of } x \text{ in order contracted after } x$$

Theorem: Ancestors of x = Nodes in search space of x



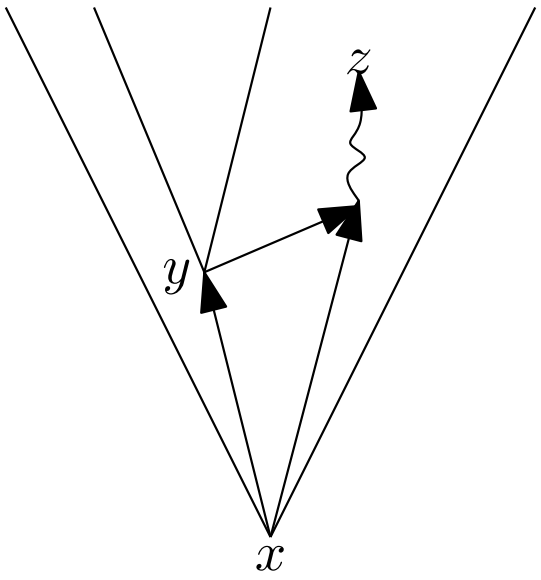
Let $\mathrm{parent}(x) = y$.
Assumption: z in the search space of x but not in the search space of y.

Elimination-Tree

The elimination tree is defined as following:

$$\mathrm{parent}(x) = \text{first neighbor of } x \text{ in order contracted after } x$$

Theorem: Ancestors of x = Nodes in search space of x



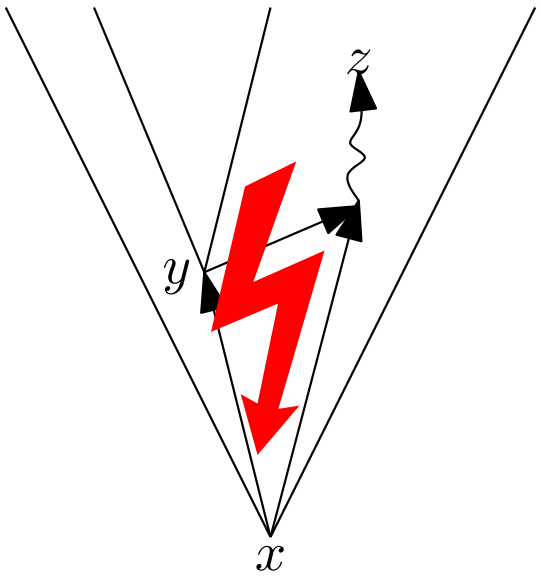
This edge must exist by construction.

Elimination-Tree

The elimination tree is defined as following:

$$\mathrm{parent}(x) = \text{first neighbor of } x \text{ in order contracted after } x$$

Theorem: Ancestors of x = Nodes in search space of x



z is in the search space of y.
Contraction

Experimental Evaluation

Instance:

- Standard DIMACS Europe benchmark, travel time metric
- $\approx$ 18M nodes, $\approx$42M directed edges
- 26.5% degree 1, 18.7% degree 2

Experimental Evaluation

Instance:

- Standard DIMACS Europe benchmark, travel time metric
- $\approx$ 18M nodes, $\approx$42M directed edges
- 26.5% degree 1, 18.7% degree 2

Results:

- Plain Dijkstra: $\approx$ 2s
- CH-preprocessing: $\approx$ 5min - 6h
- CH-query: $\approx$ 0.107ms
- CCH-customization (16 threads): $\approx$ 420ms
- CCH-query: $\approx$ 0.413ms
- CCH-query (+perfect witness search): $\approx$ 0.161ms
- CRP-customization (12 threads): $\approx$ 370ms
- CRP-query: $\approx$ 1.65ms

Route Planning for Pedestrians

Institute for Theoretical Informatics
Chair Algorithmics

Route Planning for Pedestrians [ADN+15]

Problem

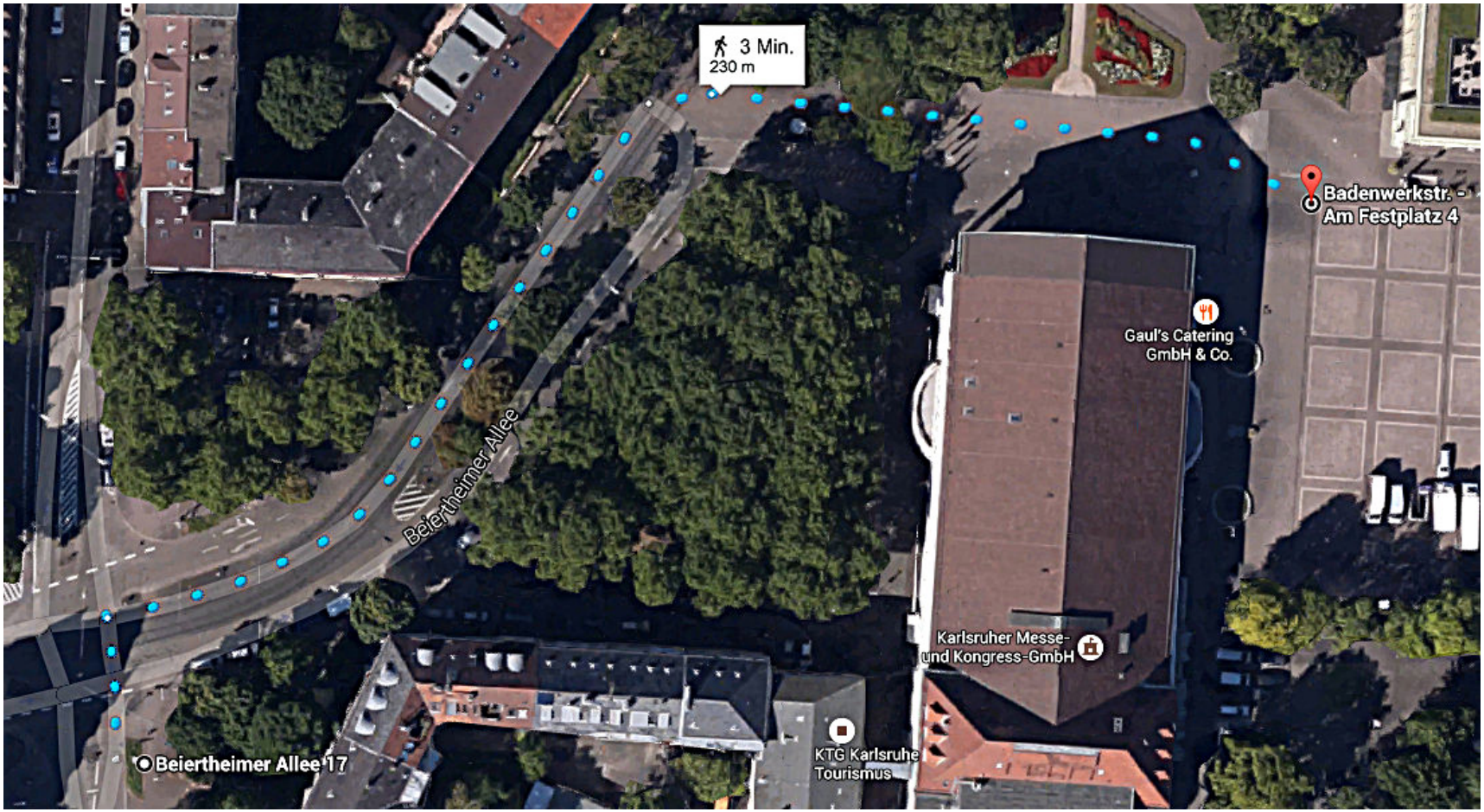
- Given: Street map
- Goal: Fastest route from A to B, feasible for pedestrians

State-of-the-art approaches

1. Use routing system for cars with different cost function:
   - Uniform walking speed (no different speed limits)
   - No direction of traffic (no one-way streets)
   - Exclude certain road categories (e. g., no highways)
   - Include certain others (e. g., add park walkways)
2. Use road lanes to draw routes

Shortcomings of Existing Solutions

Unnecessary crossing of large street



Germany, Karlsruhe, Beiertheimer Allee (Source: Google Maps)

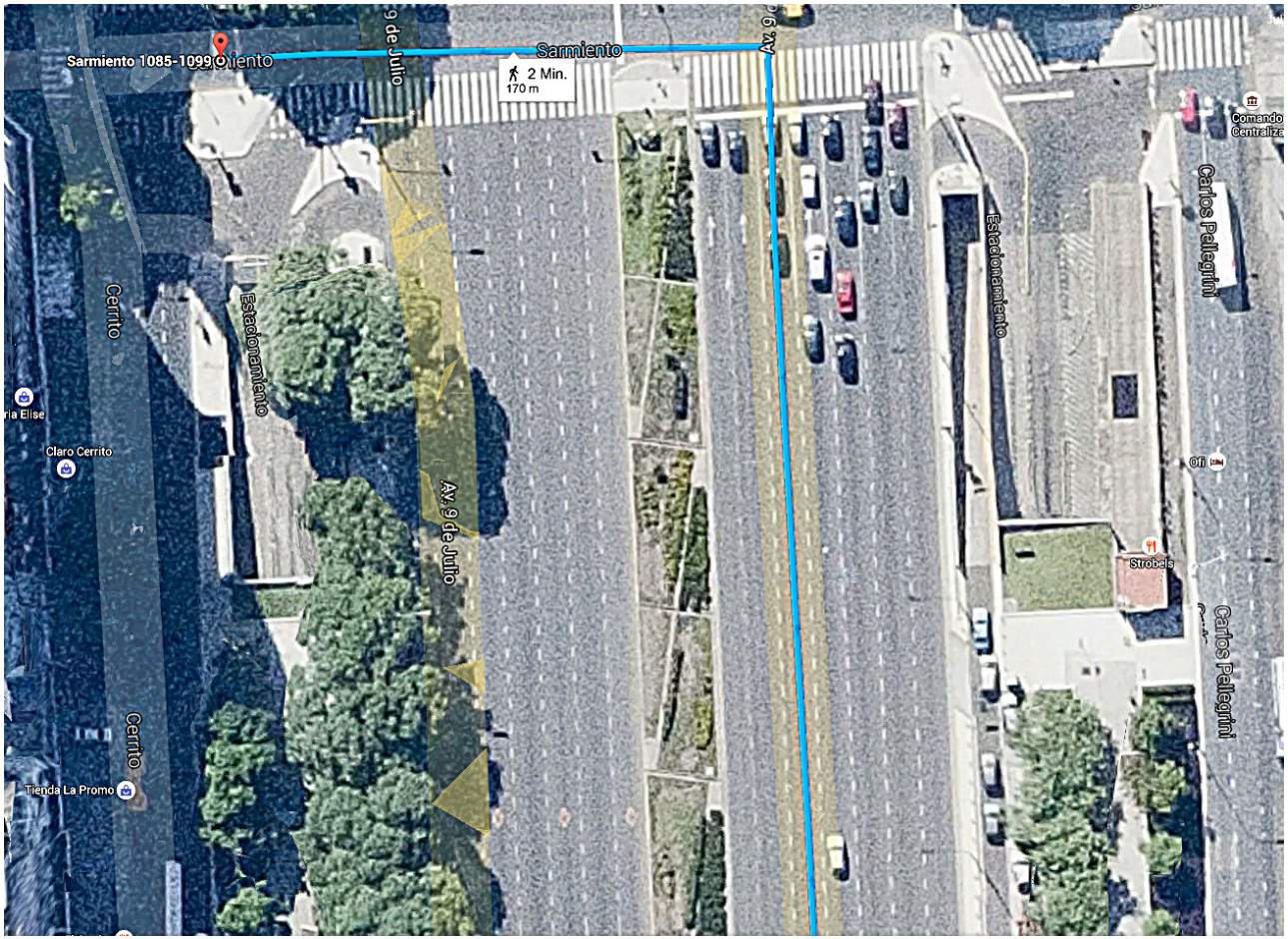
Shortcomings of Existing Solutions

Open spaces not considered

France, Lyon, Place Bellecour (Source: Google Maps)

Shortcomings of Existing Solutions

Routing/Drawing on the middle of the road

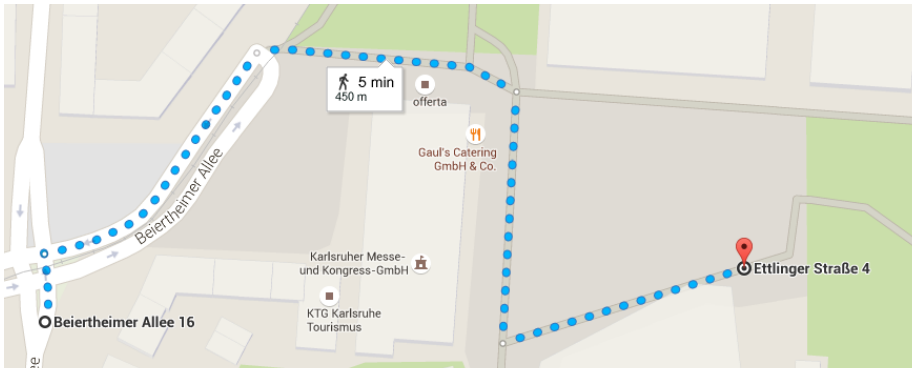


Argentina, Buenos Aires, Avenida 9 de Julio (Source: Google Maps)

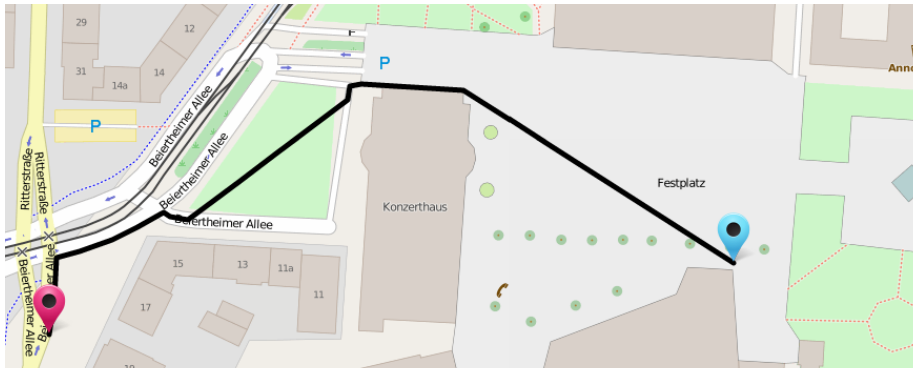
What is Special for Pedestrians?

- Excluded road categories
- Uniform walking speed
- Use sidewalks, if present
  - Choosing the side of the street explicitly may lead to faster route
- Avoid large crossings
  - Nearby bridge or underpass may be faster
- Able to deviate from streets: walk across plazas and parks

⇒ Faster, natural, and more appealing routes



Google Maps



Our Approach

Our Approach

Karlsruhe Institute of Technology

Data challenges

- Sidewalk data often incomplete
- Plaza and park data only available in polygonal form

Augment the street network model

- Automatically generate sidewalks
- Calculate crossing penalties for major roads
- Preprocess plazas and parks for routing
- Use computational geometry toolbox [CGAL 4.6, BHH+14]
  (Visibility, segment intersection, range queries, point-in-polygon, . . .)

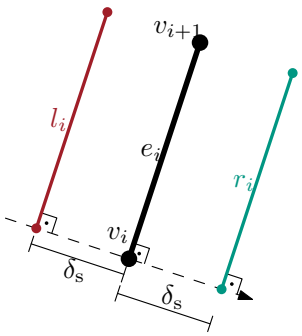
Tailored routing algorithm

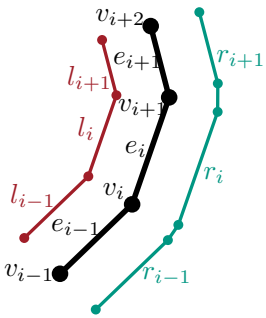
- Seamless query between arbitrary locations (street, plaza, park)
- Adapting a speedup technique for interactive queries

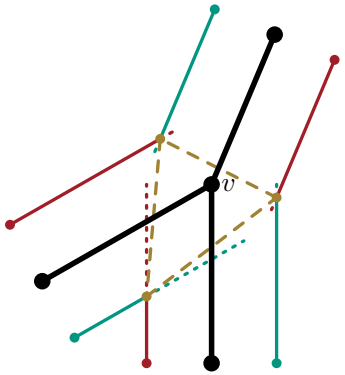
Sidewalks and Street Crossings

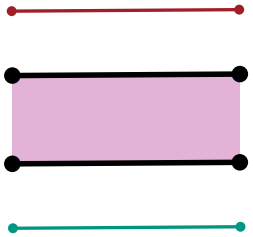
Generate sidewalks for regular streets

- Follow street geometry
  - Left and right with offset, follow curves, trim at intersections
  - Add crossings
- Handle multi-lane streets and dividers





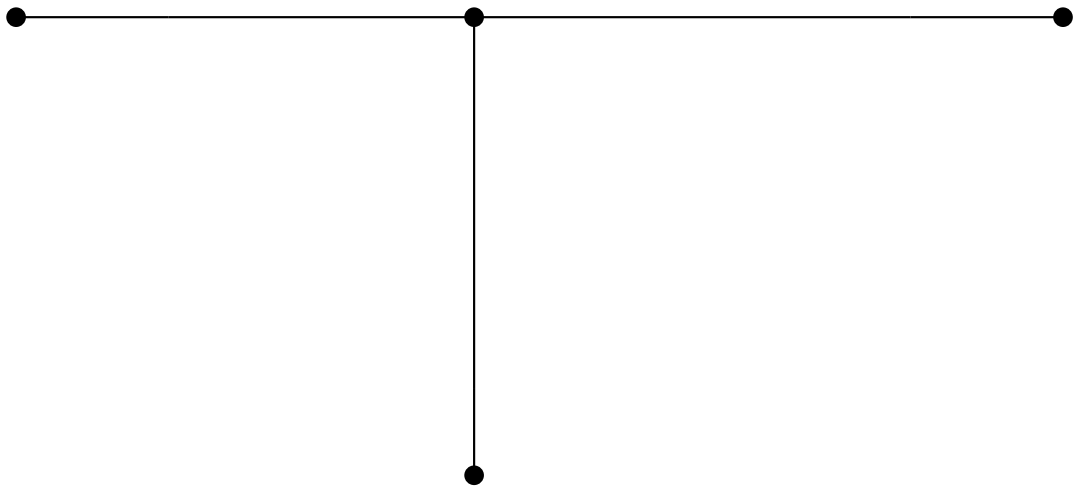




- Add penalties for crossings:
  - For entering the street (waiting for green light or traffic to clear)
  - For distance spent on roadway

Modeling Plazas

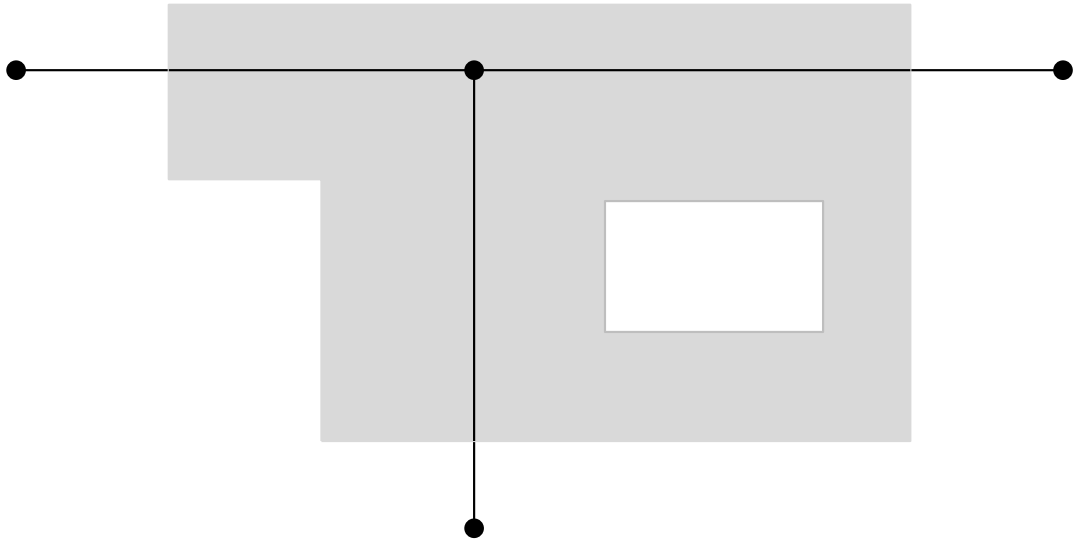
Given: Set of plaza polygons, possibly with holes (obstacles)



Preprocessing:

Modeling Plazas

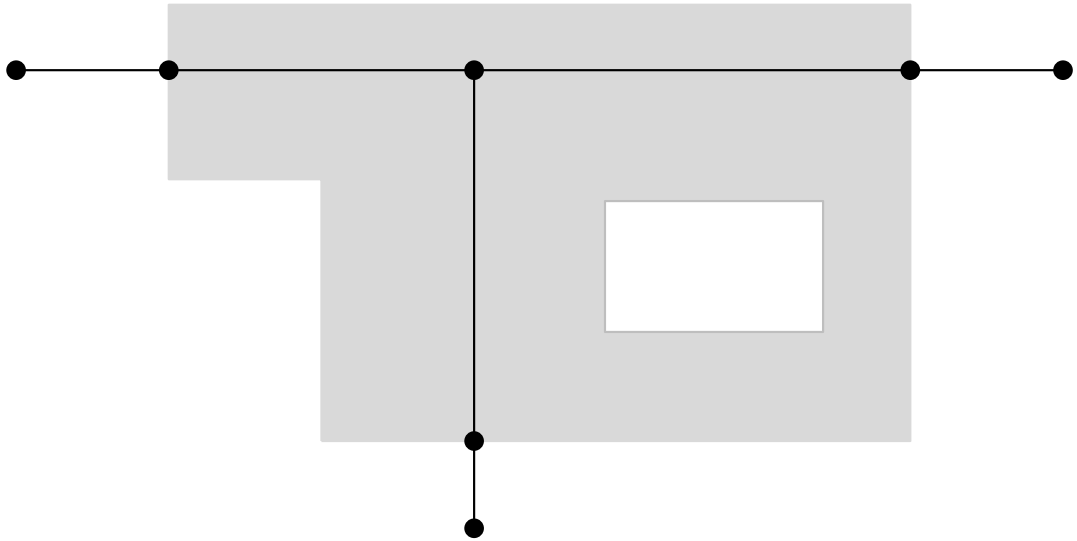
Given: Set of plaza polygons, possibly with holes (obstacles)



Preprocessing:

Modeling Plazas

Given: Set of plaza polygons, possibly with holes (obstacles)

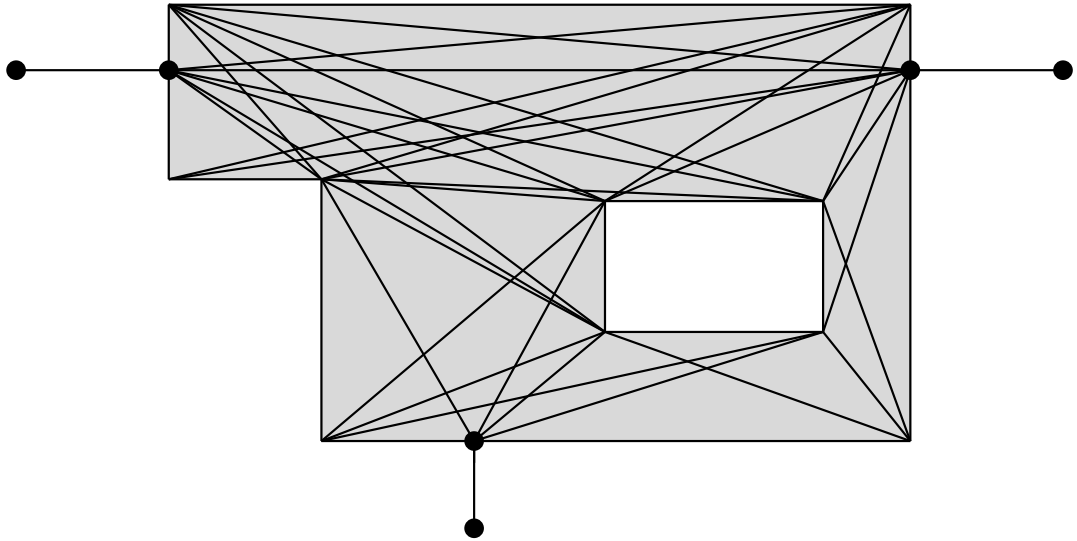


Preprocessing:

- Determine entry nodes of each plaza

Modeling Plazas

Given: Set of plaza polygons, possibly with holes (obstacles)

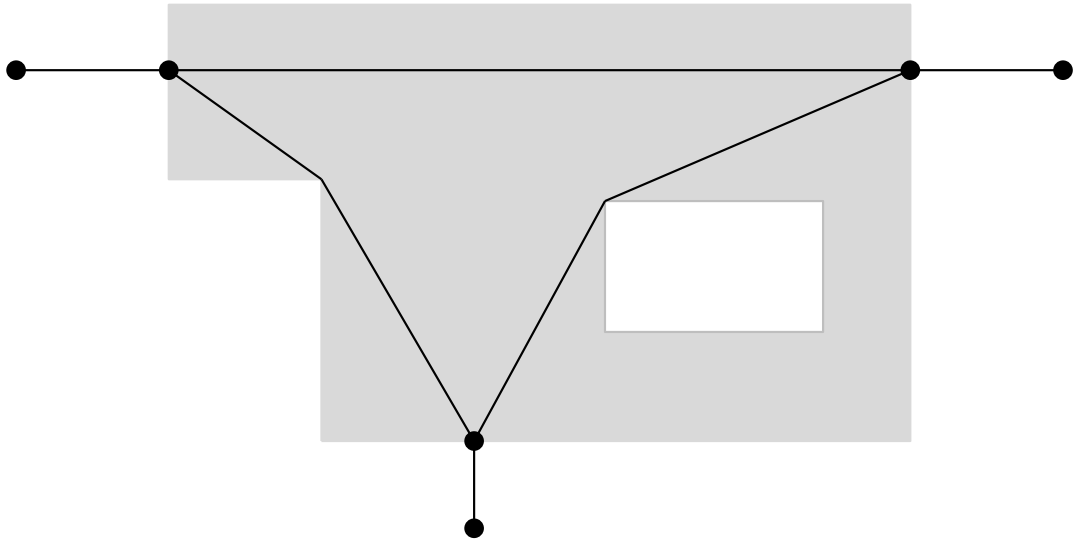


Preprocessing:

- Determine entry nodes of each plaza
- Precompute full visibility graph

Modeling Plazas

Given: Set of plaza polygons, possibly with holes (obstacles)

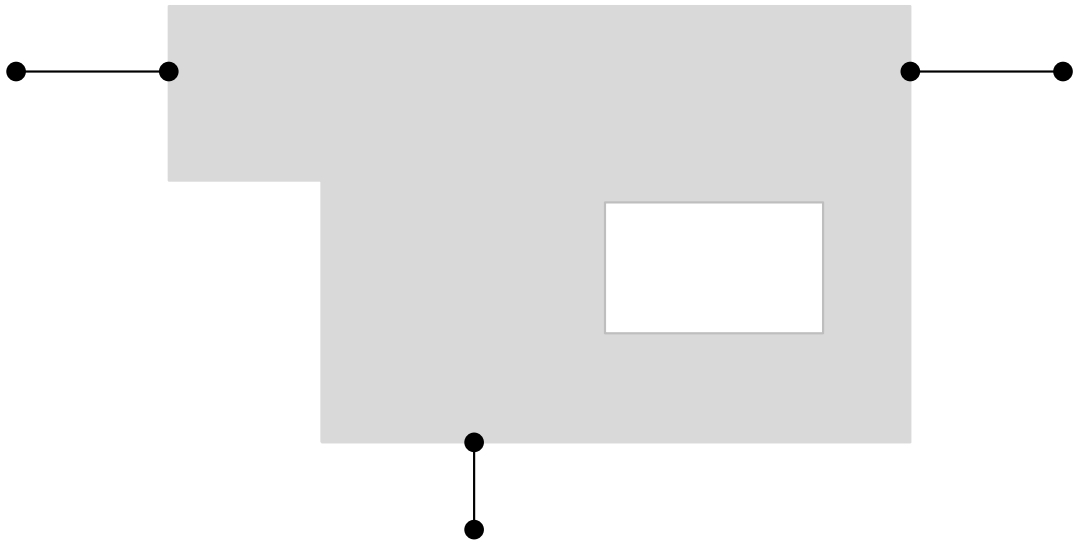


Preprocessing:

- Determine entry nodes of each plaza
- Precompute full visibility graph
- Mark visibility edges on shortest paths
  - Restricted to each plaza between entry nodes

Modeling Plazas

Given: Set of plaza polygons, possibly with holes (obstacles)

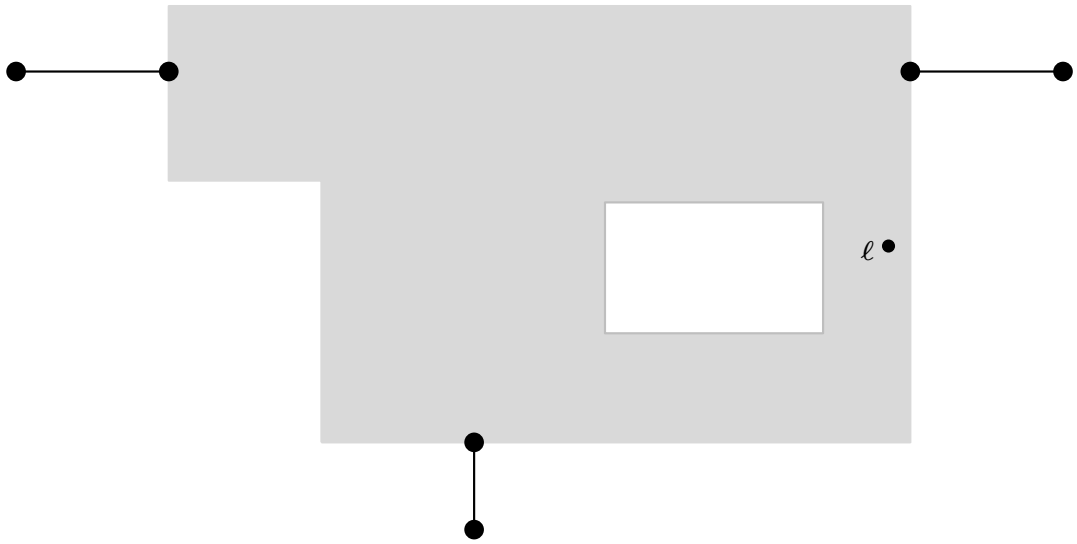


Routing:

Institute for Theoretical Informatics
Chair Algorithmics

Modeling Plazas

Given: Set of plaza polygons, possibly with holes (obstacles)

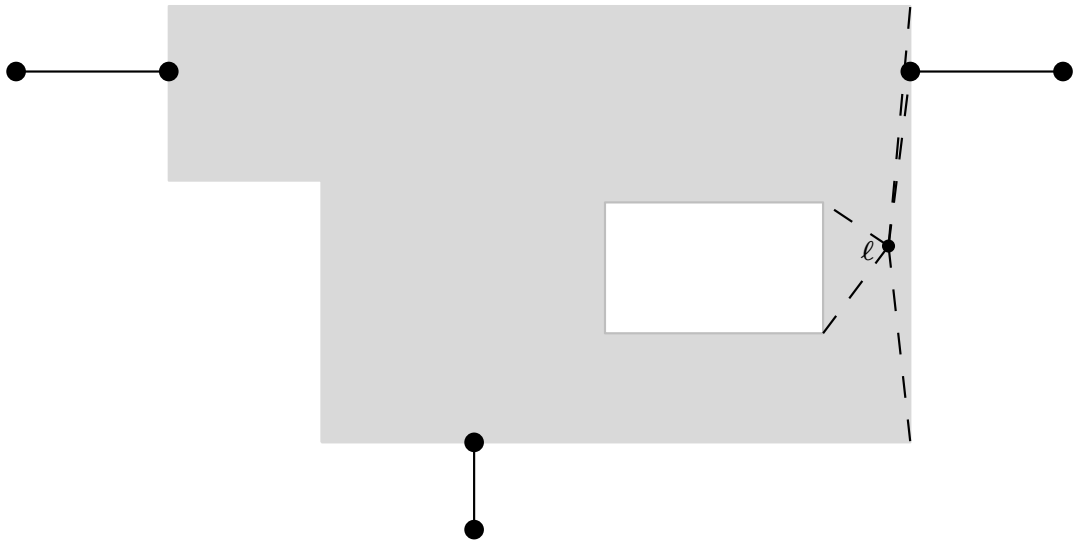


Routing:

- When origin on plaza:

Modeling Plazas

Given: Set of plaza polygons, possibly with holes (obstacles)

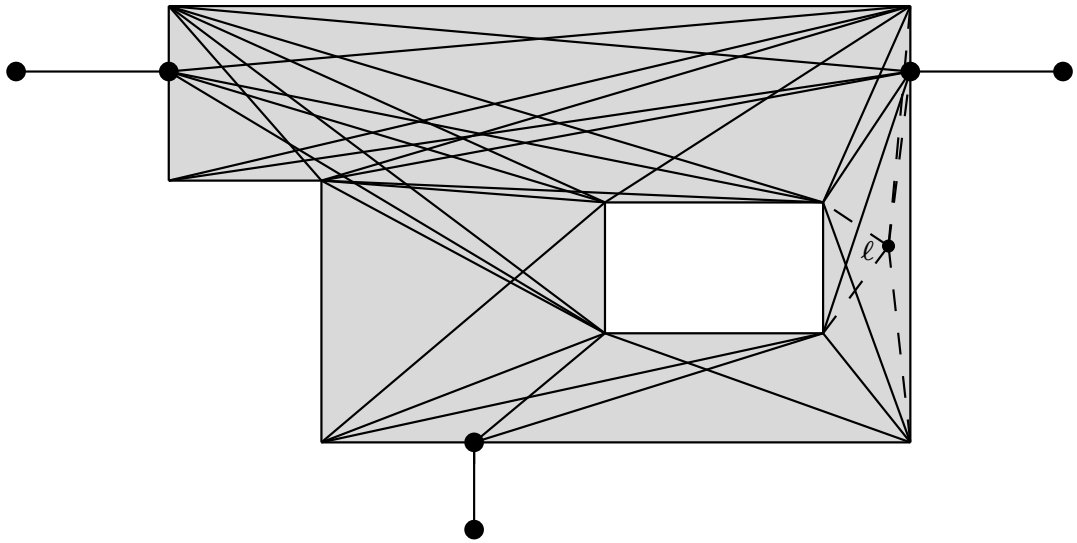


Routing:

- When origin on plaza:
  - Compute visibility edges from origin

Modeling Plazas

Given: Set of plaza polygons, possibly with holes (obstacles)

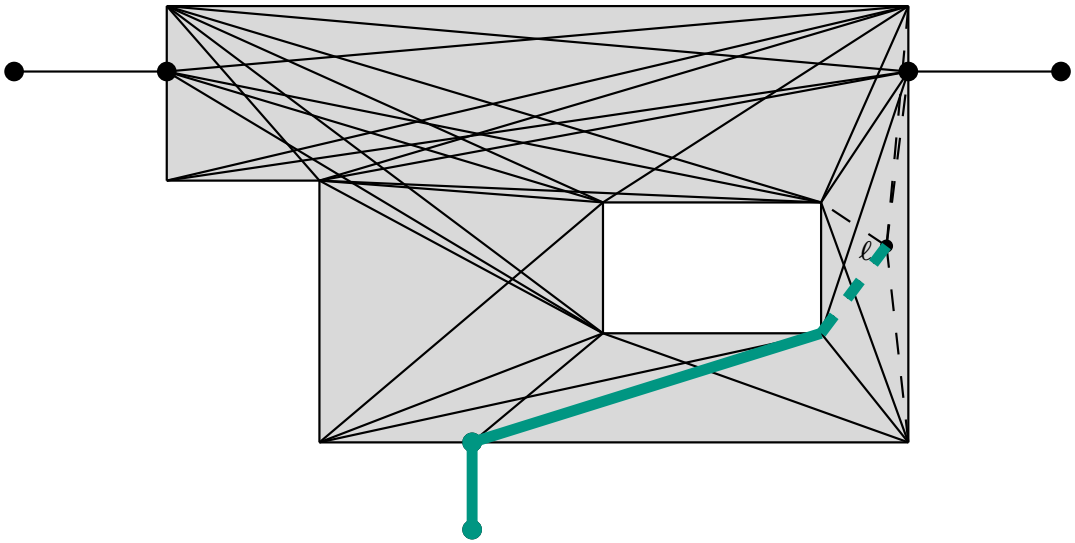


Routing:

- When origin on plaza:
  - Compute visibility edges from origin
  - Use all precomputed visibility edges of polygon for routing

Modeling Plazas

Given: Set of plaza polygons, possibly with holes (obstacles)

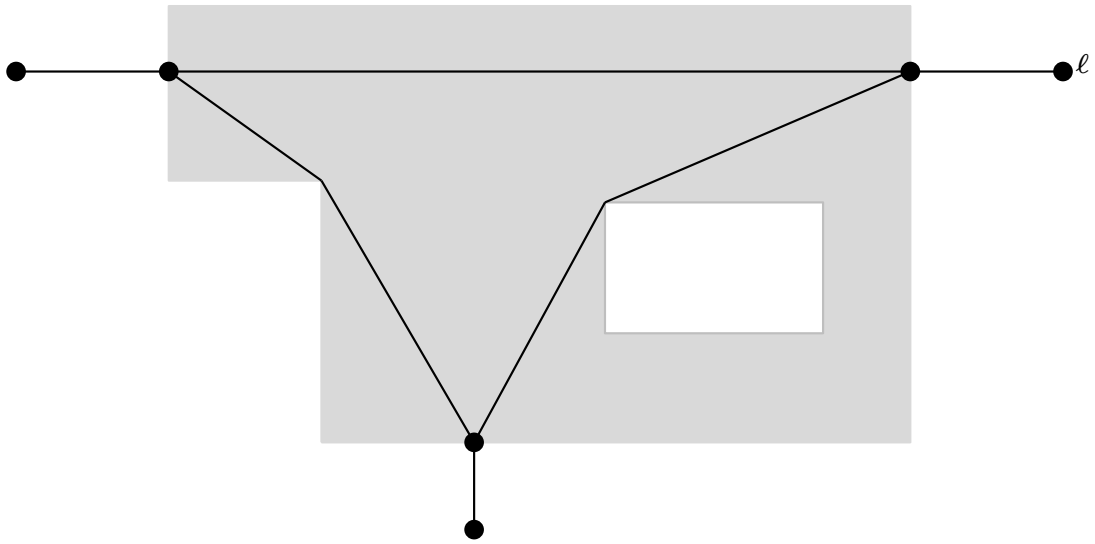


Routing:

- When origin on plaza:
  - Compute visibility edges from origin
  - Use all precomputed visibility edges of polygon for routing

Modeling Plazas

Given: Set of plaza polygons, possibly with holes (obstacles)

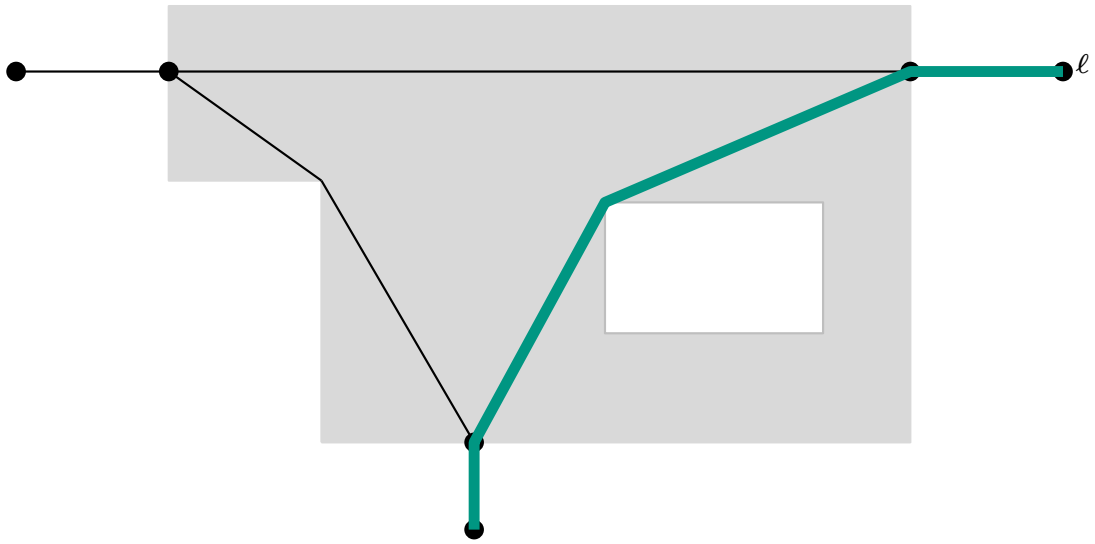


Routing:

- When origin on plaza:
  - Compute visibility edges from origin
  - Use all precomputed visibility edges of polygon for routing
- Otherwise: ignore unmarked edges

Modeling Plazas

Given: Set of plaza polygons, possibly with holes (obstacles)



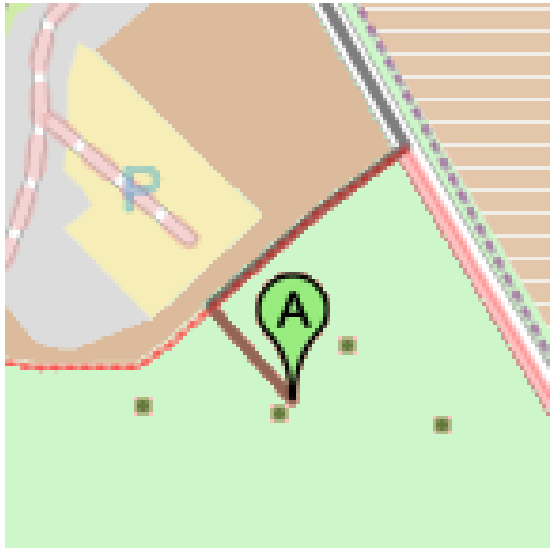
Routing:

- When origin on plaza:
  - Compute visibility edges from origin
  - Use all precomputed visibility edges of polygon for routing
- Otherwise: ignore unmarked edges

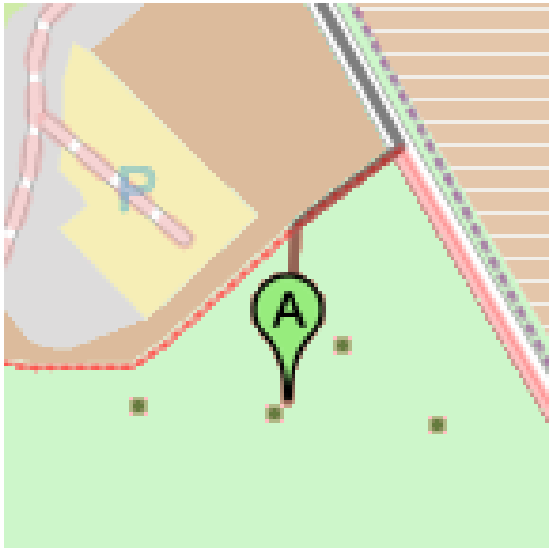
Modeling Parks

Routing model

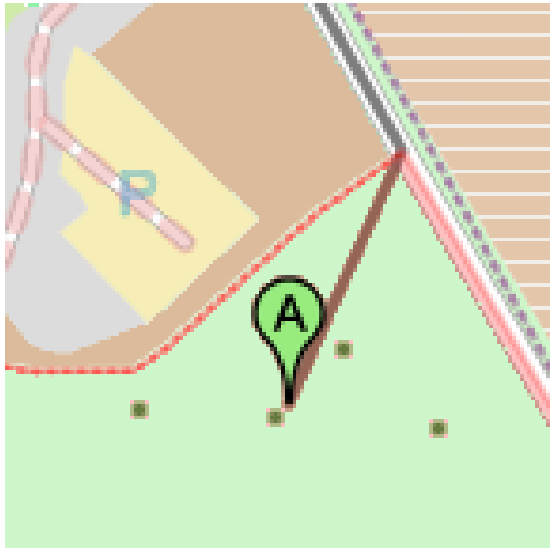
- Parks have designated walkways
- Walk across the lawn only near origin
- Walking speed on lawn is slower by factor λ
- Compute fastest route based on these speeds



slow λ



medium λ



$\lambda = 1$

Routing Algorithm

Localization

- Determine whether origin is in plaza, park, or at a street

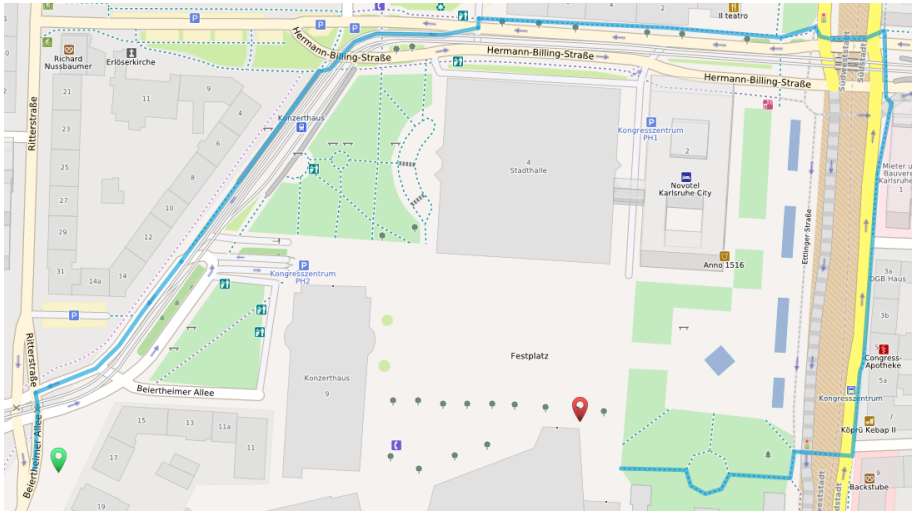
Initialization: augment routing graph

- Plaza: add visibility edges from location
- Park: compute projections to walkways from location

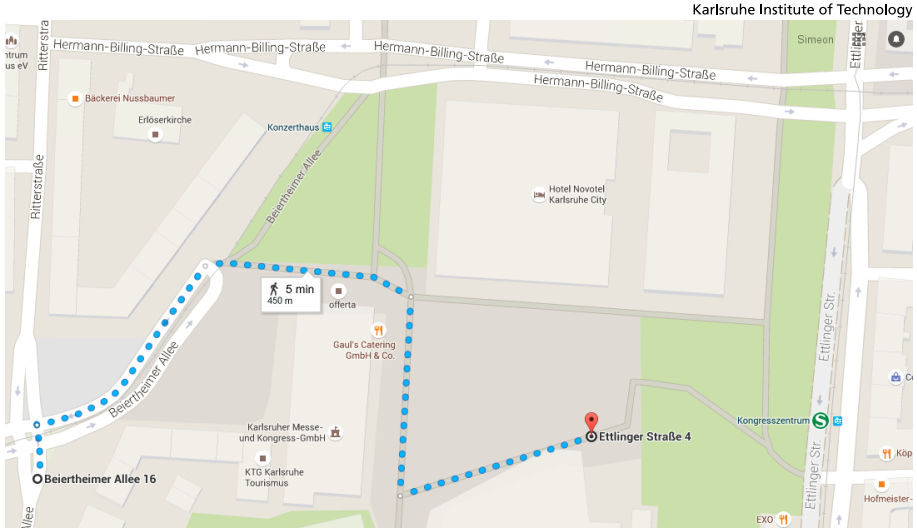
Query

- Shortest path query on augmented routing graph
- Adapt speedup techniques for faster queries

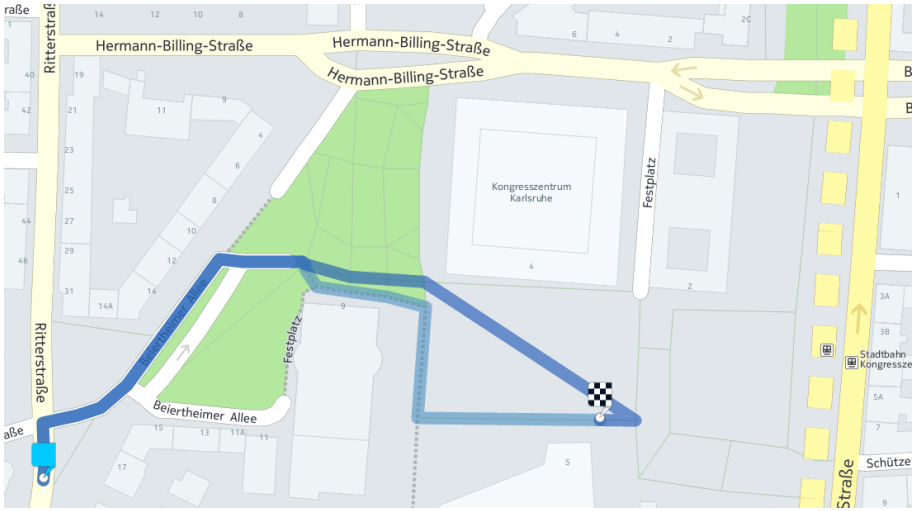
Case Study



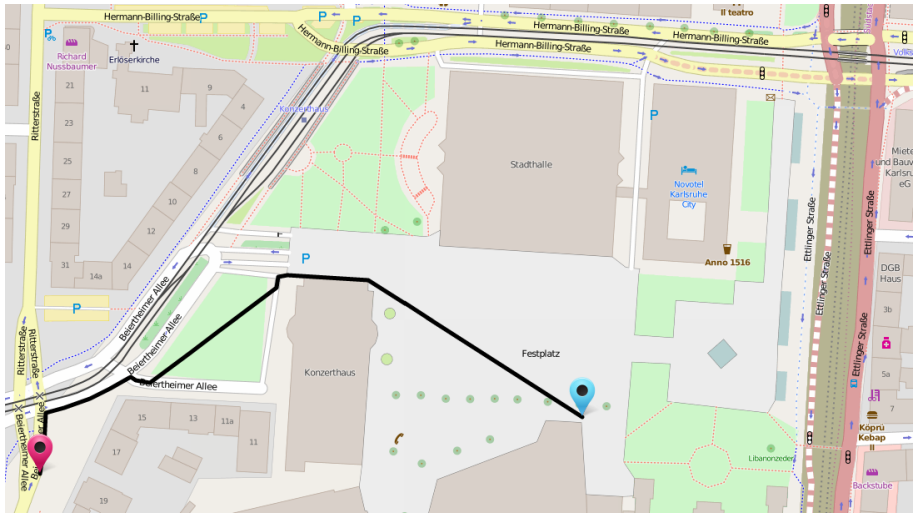
OpenRouteService



Google Maps

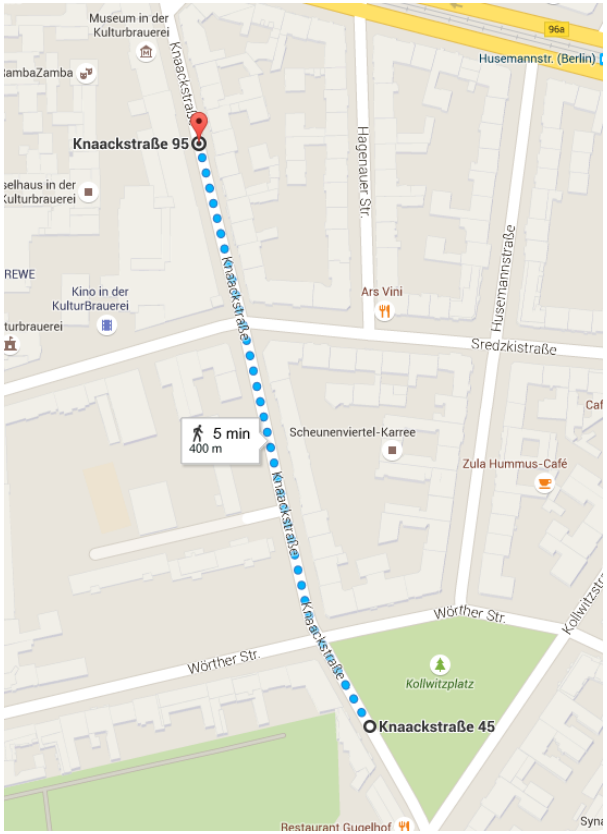


Nokia HERE

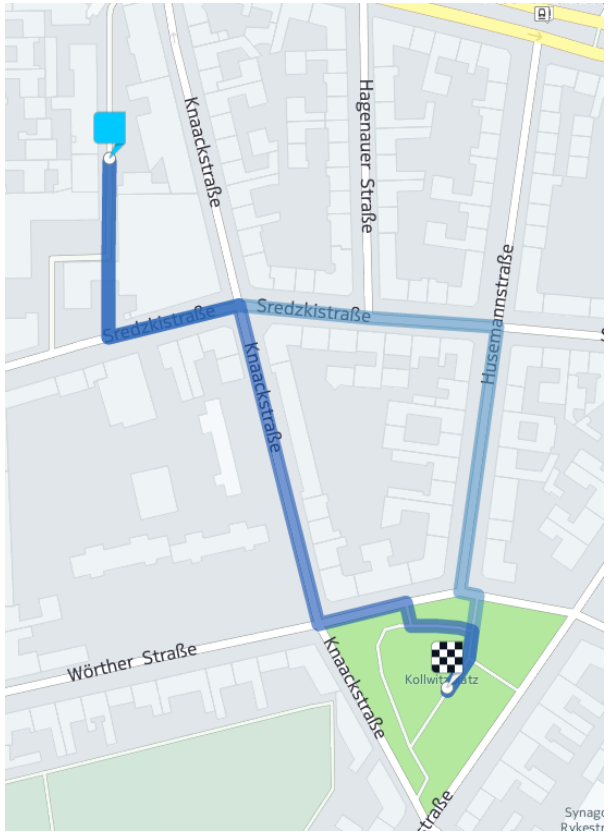


Our approach

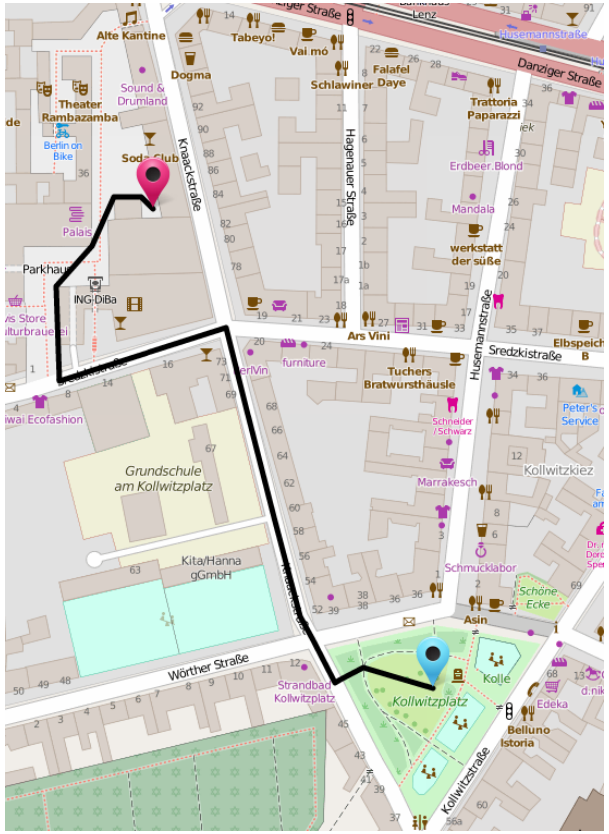
Case Study



Google Maps



Nokia HERE



Our approach

Rush hours

Conclusion & Outlook

Success story for algorithm engineering [BDG+15a]

- Fast route planning on road and timetable networks
- Metric matters

Many new challenges

- Multimodal route planning
- Scalability and quality in multimodal route planning
- Incorporating alternative mobility concepts
- Robustness, adjustable to unforeseen traffic situations
- Personalized route planning
- Eco-friendliness
- Autonomous driving
- Traffic control
- . . .

Thank you for your attention!

Institute for Theoretical Informatics
Chair Algorithmics

Bibliography I

Simeon Danailov Andreev, Julian Dibbelt, Martin Nöllenburg, Thomas Pajor, and Dorothea Wagner.
Towards realistic pedestrian route planning.
In *Proceedings of the 15th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'15)*, volume 48 of *OpenAccess Series in Informatics (OASIcs)*, pages 1–15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, September 2015.

Reinhard Bauer, Tobias Columbus, Ignaz Rutter, and Dorothea Wagner.
Search-space size in contraction hierarchies.
In *Proceedings of the 40th International Colloquium on Automata, Languages, and Programming (ICALP'13)*, volume 7965 of *Lecture Notes in Computer Science*, pages 93–104. Springer, 2013.

Reinhard Bauer and Daniel Delling.
SHARC: Fast and robust unidirectional routing.
ACM Journal of Experimental Algorithmics, 14(2.4):1–29, August 2009.
Special Section on Selected Papers from ALENEX 2008.

Hannah Bast, Daniel Delling, Andrew V. Goldberg, Matthias Müller–Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck.
Route planning in transportation networks.
Technical Report abs/1504.05140, ArXiv e-prints, 2015.

Moritz Baum, Julian Dibbelt, Andreas Gemsa, Dorothea Wagner, and Tobias Zündorf.
Shortest feasible paths with charging stops for battery electric vehicles.
In *Proceedings of the 23rd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM Press, 2015.

Moritz Baum, Julian Dibbelt, Lorenz Hübschle-Schneider, Thomas Pajor, and Dorothea Wagner.
Speed-consumption tradeoff for electric vehicle route planning.
In *Proceedings of the 14th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'14)*, volume 42 of *OpenAccess Series in Informatics (OASIcs)*, pages 138–151. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, September 2014.

Bibliography II

Moritz Baum, Julian Dibbelt, Thomas Pajor, and Dorothea Wagner.
Energy-optimal routes for electric vehicles.
In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 54–63. ACM Press, 2013.

Francisc Bungiu, Michael Hemmer, John Hershberger, Kan Huang, and Alexander Kröller.
Efficient Computation of Visibility Polygons.
CoRR, abs/1403.3905, 2014.

The CGAL Project.
CGAL User and Reference Manual.
CGAL Editorial Board, 4.6 edition, 2015.

Julian Dibbelt, Ben Strasser, and Dorothea Wagner.
Customizable contraction hierarchies.
ACM Journal of Experimental Algorithmics, 2016.
accepted.

Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling.
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.

Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter.
Exact routing in large road networks using contraction hierarchies.
Transportation Science, 46(3):388–404, August 2012.

Bibliography III

Martin Holzer, Frank Schulz, and Dorothea Wagner.
Engineering multilevel overlay graphs for shortest-path queries.
ACM Journal of Experimental Algorithmics, 13(2.5):1–26, December 2008.

Ulrich Lauther.
An extremely fast, exact algorithm for finding shortest paths in static networks with geographical background.
In *Geoinformation und Mobilität - von der Forschung zur praktischen Anwendung*, volume 22, pages 219–230. IfGI prints, 2004.

Frank Schulz, Dorothea Wagner, and Karsten Weihe.
Dijkstra's algorithm on-line: An empirical case study from public railroad transport.
ACM Journal of Experimental Algorithmics, 5(12):1–23, 2000.