Route Planning in Transportation Networks – When the Metric Matters

EuroCG’16, Lugano, Switzerland
Prof. Dr. Dorothea Wagner | March 31, 2016
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
  ⇒ 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
⇒ 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
**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 \cite{GSSD08, GSSV12}

Preprocessing example: Iteratively contract nodes

![Graph with nodes and edges](image-url)
Contraction Hierarchies [GSSD08, GSSV12]

Preprocessing example: Iteratively contract nodes
Contraction Hierarchies [GSSD08, GSSV12]

Preprocessing example: Iteratively contract nodes

Dorothea Wagner – Route Planning in Transportation Networks
March 31, 2016
Contraction Hierarchies \[\text{[GSSD08, GSSV12]}\]

Preprocessing example: Iteratively contract nodes

```
6 4 5
1
3 2
5
2
2
6
3
1 5
```
Contraction Hierarchies [GSSD08, GSSV12]

Preprocessing example: Iteratively contract nodes
Contraction Hierarchies \cite{GSSD08, GSSV12}

Preprocessing example: Iteratively contract nodes
Contraction Hierarchies [GSSD08, GSSV12]

Preprocessing example: Iteratively contract nodes

Dorothea Wagner – Route Planning in Transportation Networks
March 31, 2016
Contraction Hierarchies \cite{GSSD08, GSSV12}

Preprocessing example: Iteratively contract nodes

Dorothea Wagner – Route Planning in Transportation Networks
March 31, 2016
Contraction Hierarchies [GSSD08, GSSV12]

Preprocessing example: Iteratively contract nodes

Dorothea Wagner – Route Planning in Transportation Networks
March 31, 2016
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
Contraction Hierarchies [GSSD08, GSSV12]

Query example: Bidirectional upward search

shortest $st$-path
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

Electric vehicles:
- Future means of transportation
- Run on regenerative energy sources

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

⇒ 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):

![Graph showing energy consumption and state-of-charge](image)
Energy-Optimal Routes

Challenges:

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

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

![Graph showing energy consumption vs. state-of-charge]

Dorothea Wagner – Route Planning in Transportation Networks
March 31, 2016
Energy-Optimal Routes

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

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

![Diagram](image-url)
Energy-Optimal Routes

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

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

![Graph showing energy consumption and state-of-charge](image-url)
Energy-Optimal Routes [BDPW13]

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

```
<table>
<thead>
<tr>
<th>consumption</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 kWh</td>
<td>20 min</td>
</tr>
<tr>
<td>3 kWh</td>
<td>35 min</td>
</tr>
</tbody>
</table>
```

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

⇒ Consider tradeoff between speed and energy consumption

Find the **fastest** path such that the battery does not run out: \( \mathcal{NP}\text{-hard} \)
Constrained Shortest Paths [BDHS+14]

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

\[ \Rightarrow \text{Bicriteria Dijkstra on multigraph.} \]

\[
\begin{array}{c@{}c@{}c}
(0, 0) & (360, 0.5) & (510, 1.9) \\
(450, 0.4) & (550, 1.6) & (640, 1.5) \\
\end{array}
\]

\[
\begin{array}{c@{}c@{}c}
\text{city, 5 km} & \text{motorway, 5 km} & \text{rural, 10 km} \\
360 \text{ s} & 0.5 \text{ kWh} & 150 \text{ s} & 1.4 \text{ kWh} & 450 \text{ s} & 1.4 \text{ kWh} \\
450 \text{ s} & 0.4 \text{ kWh} & 190 \text{ s} & 1.1 \text{ kWh} & 600 \text{ s} & 0.9 \text{ kWh} \\
\end{array}
\]

**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

\( s \) \rightarrow \text{?} \rightarrow \( t \)
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

$S$  $?$  $T$
Observations

Find the fastest route from $s$ to $t$:

- Next charging station might be positioned in wrong direction

Reachable area
Charging station

Dorothea Wagner – Route Planning in Transportation Networks
March 31, 2016
Observations

Find the fastest route from $s$ to $t$:

- Reachable area
- Charging station

Dorothea Wagner – Route Planning in Transportation Networks
March 31, 2016
Observations

Find the fastest route from \( s \) to \( t \):

- Reachable area
- Charging station

Dorothea Wagner – Route Planning in Transportation Networks
March 31, 2016
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

Dorothea Wagner – Route Planning in Transportation Networks
March 31, 2016
Find the fastest route from $s$ to $t$:

Reachable area
Charging station
Fast charging station / swapping station
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
Find the fastest route from $s$ to $t$:

Reachable area
Charging station
Fast charging station / swapping station

Dorothea Wagner – Route Planning in Transportation Networks
March 31, 2016
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?
Find the fastest route from \( s \) to \( t \):

- Larger battery \( \Rightarrow \) simpler problem?

---

Dorothea Wagner – Route Planning in Transportation Networks
March 31, 2016
Find the fastest route from $s$ to $t$:

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

Reachable area
Charging station

Dorothea Wagner – Route Planning in Transportation Networks
March 31, 2016
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

![Graph showing SoC over time]

Dorothea Wagner – Route Planning in Transportation Networks
March 31, 2016
CHArge

CHArge = CH & A* & CFP:
- Combines CFP with speedup techniques
- Can handle arbitrary charging station types

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

⇒ Metric customization
Shortest Path Computation

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

Dorothea Wagner – Route Planning in Transportation Networks
March 31, 2016
Contraction Hierarchies
Contraction Hierarchies

```
6 2 12
5 10
2 5
3 3
7 5
1 13
7 5
3 1
2 1
```
Contraction Hierarchies
Contraction Hierarchies
Contraction Hierarchies
Contraction Hierarchies
Graph Fill-In
Graph Fill-In

Dorothea Wagner – Route Planning in Transportation Networks
March 31, 2016
Graph Fill-In
Graph Fill-In

![Graph Diagram](image_url)
Graph Fill-In
Graph Fill-In

1
3
7
5
2
4
6
elimination order

1
3
7
5
2
4
6
elimination order

elimination tree

Dorothea Wagner – Route Planning in Transportation Networks
March 31, 2016
Graph Fill-In

Dorothea Wagner – Route Planning in Transportation Networks
March 31, 2016
Graph Fill-In
Graph Fill-In

elimination order

elimination tree
Nested Dissection (ND)

Institute for Theoretical Informatics
Chair Algorithmics
Nested Dissection (ND)

![Nested Dissection Diagram]

Institute for Theoretical Informatics
Chair Algorithmics
Nested Dissection (ND)

The diagram illustrates the concept of Nested Dissection. It shows how a graph is divided into smaller subgraphs, with each level of division (or iteration) leading to a tree-like structure. The nodes represent parts of the graph, and the edges represent connections between them. The coloring (red and blue) indicates a specific order or partitioning that is crucial for efficient computation in algorithms like Nested Dissection.
Nested Dissection (ND)

Dorothea Wagner – Route Planning in Transportation Networks
March 31, 2016
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
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 \( \approx 50 \) 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
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 \(\infty\)
- Customization: Adjust weights on shortcuts to fulfill lower triangle inequality
Customization

Basic Customization

Shortcut edge
Lower triangle inequality must hold for every lower triangle
Basic Customization

Decrease weight until the inequality holds
Customization

Basic Customization

Do this for every lower triangle
Basic Customization

In which order do we process the edges?
Iterate over edges by increasing level
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
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:

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

**Theorem**: Ancestors of \( x \) = Nodes in search space of \( x \)
Elimination-Tree

The elimination tree is defined as following:

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

**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:

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

**Theorem**: Ancestors of \( x \) = Nodes in search space of \( x \)

The search space of \( x \)
Elimination-Tree

The elimination tree is defined as following:

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

**Theorem:** Ancestors of \( x \) = Nodes in search space of \( x \)

Let \( \text{parent}(x) = y \).

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:

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

**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:

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

**Theorem:** Ancestors of \( x \) = Nodes in search space of \( x \)

\[ z \text{ is in the search space of } y. \]

Contraction
Experimental Evaluation

Instance:
- Standard DIMACS Europe benchmark, travel time metric
- \( \approx 18 \text{M nodes, } \approx 42 \text{M directed edges} \)
- 26.5\% degree 1, 18.7\% degree 2
Experimental Evaluation

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

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

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:

- 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)

**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:
Modeling Plazas

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

Routing:
- When origin on plaza:

\[ \ell \]
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 $\lambda$
- Compute fastest route based on these speeds

slow $\lambda$  medium $\lambda$  $\lambda = 1$
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

Dorothea Wagner – Route Planning in Transportation Networks
March 31, 2016
Case Study

Google Maps  
Nokia HERE  
Our approach
Rush hours
Conclusion & Outlook

Success story for algorithm engineering [BDG\textsuperscript{+}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!
Simeon Danailov Andreev, Julian Dibbelt, Martin Nöllenburg, Thomas Pajor, and Dorothea Wagner.
Towards realistic pedestrian route planning.

Reinhard Bauer, Tobias Columbus, Ignaz Rutter, and Dorothea Wagner.
Search-space size in contraction hierarchies.

Reinhard Bauer and Daniel Delling.
SHARC: Fast and robust unidirectional routing.
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.

Moritz Baum, Julian Dibbelt, Andreas Gemsa, Dorothea Wagner, and Tobias Zündorf.
Shortest feasible paths with charging stops for battery electric vehicles.

Moritz Baum, Julian Dibbelt, Lorenz Hübschle-Schneider, Thomas Pajor, and Dorothea Wagner.
Speed-consumption tradeoff for electric vehicle route planning.
Moritz Baum, Julian Dibbelt, Thomas Pajor, and Dorothea Wagner.
Energy-optimal routes for electric vehicles.

Francisc Bungiu, Michael Hemmer, John Hershberger, Kan Huang, and Alexander Kröller.
Efficient Computation of Visibility Polygons.

The CGAL Project.

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.

Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter.
Exact routing in large road networks using contraction hierarchies.
Martin Holzer, Frank Schulz, and Dorothea Wagner.
Engineering multilevel overlay graphs for shortest-path queries.

Ulrich Lauther.
An extremely fast, exact algorithm for finding shortest paths in static networks with geographical background.

Frank Schulz, Dorothea Wagner, and Karsten Weihe.
Dijkstra’s algorithm on-line: An empirical case study from public railroad transport.