Route Planning Algorithms in Transportation Networks

7th International Network Optimization Conference
Dorothea Wagner | May 18, 2015, Warsaw, Poland
Motivation

Important application, e.g.,
- Navigation systems for cars
- Google Maps, Bing Maps, . . .
- Timetable information

Many commercial systems
- Use heuristic methods
- Consider “reasonable” part of the network
- Have no quality guarantees

Find methods for route planning in transportation networks with provably optimal solutions regarding the quality of the routes.
Problem

Request:
- Find the best connection in a transportation network

Idea:
- Network as graph $G = (V, E)$
- Edge weights are travel times
- Shortest paths in $G$ equal quickest connections
- Classic problem (Dijkstra)

Problems:
- Transport networks are huge
- Dijkstra too slow (> 1 second)
Speed-Up Techniques

Observations:
- Dijkstra visits all nodes closer than the target
- Unnecessary computations
- Many requests in a hardly changing network

Idea:
- Two-phase algorithm:
  - Offline: compute additional data during preprocessing
  - Online: speed-up query with this data
- 3 criteria: preprocessing time and space, speed-up over Dijkstra
Showpiece of Algorithm Engineering

Falsifiable Hypotheses

Design
Experiment
Analyze
Implement
Showpiece of Algorithm Engineering

Falsifiable Hypotheses

Design

Experiment

Analyze

Implement

Realistic scenarios

Real-world data

Performance guarantees & practical algorithms
History I

Phase I: Theory (1959 - 1999):
- Improve theoretical worst-case running time
- By introduction of better data structures
- Bidirectional search, A*-search (goal-directed)

Phase II: Speed-up techniques (1999 - 2005):
- Two approaches: goal-directed and hierarchical approach
- Improvement on this for several inputs

Phase III: Road networks (2005 - 2008):
- Focus on continent-sized road networks
- DIMACS challenge in 2006
- Speed-up factors in range of several millions over Dijkstra
Phase IV: Towards more realistic scenarios (2008-2012):
- Time-dependency, multicriteria, alternative routes, . . .
- Timetable information
- Back to theory: why do things work?

Now: New challenges (since 2012):
- Other metrics, e.g., energy consumption
- Customizability (supporting user-centric route planning)
- Multimodal
Many techniques:

- Arc-Flags [Lau04]
- Multi-Level Dijkstra [SWW00, HSW08]
  - Customizable Route Planning (CRP) [DGPW11]
- ALT: A*, Landmarks, Triangle Inequality [GH05, GW05]
- Reach [GKW07]
- Contraction Hierarchies (CH) [GSSD08]
- Transit Node Routing (TNR) [ALS13]
- Hub Labeling (HL) [ADGW12]
- ...
**Shortcuts**

**Observation:**
- Nodes with low degree are **not** important

**Contract graph**
- Iteratively remove such nodes
- Add **shortcuts** to preserve distances between non-removed nodes

**Query:**
- Bidirectional
- Prune edges heading to **less** important nodes
Contraction Hierarchies [GSSD08]

Idea: solely use contraction

Approach:
- Heuristically order nodes by “importance”
- Contract nodes in that order
- Node \( v \) contracted by

1. **forall the edges** \((u, v)\) and \((v, w)\) **do**
2.  
3. **if** \((u, v, w)\) unique shortest path **then**
4. **add shortcut** \((u, w)\) with weight \( \text{len}(u, v) + \text{len}(v, w) \);

- Query only looks at edges to more important nodes
Example: CH Preprocessing

2 2 3 1 2 6 1 3 5 4
Example: CH Preprocessing
Example: CH Preprocessing
Example: CH Preprocessing
Example: CH Preprocessing
Example: CH Preprocessing
CH Query

- Modified bidirectional Dijkstra
- 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 \( G_{\downarrow} \)
CH Query

- Modified bidirectional Dijkstra
- Upward graph \( G_{↑} := (V, E_{↑}) \) with \( E_{↑} := \{(u, v) \in E : u < v\} \)
- Downward graph \( G_{↓} := (V, E_{↓}) \) with \( E_{↓} := \{(u, v) \in E : u > v\} \)
- Forward search in \( G_{↑} \) and backward search in \( G_{↓} \)
CH Query

- Modified bidirectional Dijkstra
- 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 $G^\downarrow$
CH Query

- Modified bidirectional Dijkstra
- 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 $G^\downarrow$
Question: What is a good contraction order?

- No guarantees on search space [GSSD08]
**Question**: What is a good contraction order?
- No guarantees on search space [GSSD08]

**WeakCH** [BCRW13]
- Balanced separator nodes are important
  → resulting CH is called *weak*
- $O(n^\alpha)$ separators → $O(n^\alpha)$ nodes in the search space
- Order is independent of metric
(Multi-Level) Overlays \([SWW00, HSW08]\)

**Observation:** many (long-distance) paths share large subpaths

**Idea:** precompute partial solutions

**Overlay graph:**
- Select important nodes (separators, path coverage, heuristic)
- Compute shortcut-edges:
  - Skip unimportant nodes
  - Conserve distances to important nodes

**Queries:**
- Multi-level Dijkstra variant
- Ignore edges towards less important nodes

analogous: hierarchies with several levels of nodes of varying importances
Hub Labeling

Preprocessing:
- For each node $u$, compute label $L(u)$
- A set of hub nodes $v$ and their distance $\text{dist}(u, v)$ to $u$

Labels must fulfill cover property:
for every $s$, $t$-pair, the shortest path goes through
the intersection of $L(s) \cap L(t)$

$s$–$t$ query:
Find node $v \in L(s) \cap L(t)$
that minimizes $\text{dist}(s, v) + \text{dist}(v, t)$

Observations:
Very simple query (can even be implemented in SQL)
Query performance depends only on label sizes
The "magic" lies in computing a small labeling efficiently
Hub Labeling

Preprocessing:
- For each node $u$, compute label $L(u)$
- A set of hub nodes $v$ and their distance $\text{dist}(u, v)$ to $u$
- Labels must fulfill cover property: for every $s$, $t$-pair, the shortest path goes through the intersection of $L(s) \cap L(t)$

Observations:
- Very simple query (can even be implemented in SQL)
- Query performance depends only on label sizes
- The "magic" lies in computing a small labeling efficiently
Hub Labeling

Preprocessing:
- For each node $u$, compute label $L(u)$
- A set of hub nodes $v$ and their distance $\text{dist}(u, v)$ to $u$
- Labels must fulfill cover property: for every $s$, $t$-pair, the shortest path goes through the intersection of $L(s) \cap L(t)$

$s$–$t$ query:
Find node $v \in L(s) \cap L(t)$...
Hub Labeling

Preprocessing:
- For each node $u$, compute label $L(u)$
  - A set of hub nodes $v$ and their distance $\text{dist}(u, v)$ to $u$
- Labels must fulfill cover property:
  for every $s, t$-pair, the shortest path goes through the intersection of $L(s) \cap L(t)$

$s-t$ query:
- Find node $v \in L(s) \cap L(t)$ . . .

Dorothea Wagner – Route Planning Algorithms in Transportation Networks
May 18, 2015, Warsaw, Poland
Hub Labeling

Preprocessing:
- For each node $u$, compute label $L(u)$
  - A set of hub nodes $v$ and their distance $\text{dist}(u, v)$ to $u$
- Labels must fulfill cover property:
  for every $s, t$-pair, the shortest path goes through the intersection of $L(s) \cap L(t)$

$s-t$ query:
- Find node $v \in L(s) \cap L(t)$ . . .
- . . . that minimizes $\text{dist}(s, v) + \text{dist}(v, t)$
Hub Labeling

Preprocessing:

- For each node $u$, compute label $L(u)$
  - A set of hub nodes $v$ and their distance $\text{dist}(u, v)$ to $u$
- Labels must fulfill cover property:
  for every $s, t$-pair, the shortest path goes through the intersection of $L(s) \cap L(t)$

$s-t$ query:

- Find node $v \in L(s) \cap L(t)$ . . .
- . . . that minimizes $\text{dist}(s, v) + \text{dist}(v, t)$
Hub Labeling

Preprocessing:
- For each node $u$, compute label $L(u)$
  - A set of hub nodes $v$ and their distance $\text{dist}(u, v)$ to $u$
- Labels must fulfill cover property:
  for every $s, t$-pair, the shortest path goes through
  the intersection of $L(s) \cap L(t)$

$s-t$ query:
- Find node $v \in L(s) \cap L(t)$ . . .
- . . . that minimizes $\text{dist}(s, v) + \text{dist}(v, t)$

Observations:
- Very simple query (can even be implemented in SQL)
- Query performance depends only on label sizes
- The “magic” lies in computing a small labeling efficiently
Hub Labeling

Preprocessing:
- For each node \( u \), compute label \( L(u) \)
- A set of hub nodes \( v \) and their distance \( \text{dist}(u, v) \) to \( u \)
- Labels must fulfill cover property:
  for every \( s, t \)-pair, the shortest path goes through the intersection of \( L(s) \cap L(t) \)

\( s-t \) query:
- Find node \( v \in L(s) \cap L(t) \) ...
- ... that minimizes \( \text{dist}(s, v) + \text{dist}(v, t) \)
Hub Labeling

Preprocessing:
- For each node $u$, compute label $L(u)$
  - A set of hub nodes $v$ and their distance $\text{dist}(u, v)$ to $u$
- Labels must fulfill cover property:
  for every $s, t$-pair, the shortest path goes through the intersection of $L(s) \cap L(t)$

$s$–$t$ query:
- Find node $v \in L(s) \cap L(t)$ ...
- ... that minimizes $\text{dist}(s, v) + \text{dist}(v, t)$

Observations:
- Very simple query (can even be implemented in SQL)
- Query performance depends only on label sizes
- The “magic” lies in computing a small labeling efficiently
Experimental Evaluation

**Input:** Road network of Europe
- Approx. 18M nodes
- Approx. 42M edges

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Dijkstra [Dij59]</td>
<td>—</td>
<td>—</td>
<td>2 550 000</td>
<td>—</td>
</tr>
<tr>
<td>ALT [GH05, GW05]</td>
<td>0:42</td>
<td>2.2</td>
<td>24 521</td>
<td>104</td>
</tr>
<tr>
<td>CRP [DGPW11]</td>
<td>1:00</td>
<td>0.5</td>
<td>1 650</td>
<td>1 545</td>
</tr>
<tr>
<td>Arc-Flags [Lau04]</td>
<td>0:20</td>
<td>0.3</td>
<td>408</td>
<td>6 250</td>
</tr>
<tr>
<td>CH [GSSD08]</td>
<td>0:05</td>
<td>0.2</td>
<td>110</td>
<td>23 181</td>
</tr>
<tr>
<td>TNR [ALS13]</td>
<td>0:20</td>
<td>2.1</td>
<td>1.25</td>
<td>2 040 000</td>
</tr>
<tr>
<td>HL [ADGW12]</td>
<td>0:37</td>
<td>18.8</td>
<td>0.56</td>
<td>4 553 571</td>
</tr>
</tbody>
</table>

In use at Bing, Google, TomTom, ...
Experimental Evaluation

**Input:** Road network of Europe
- Approx. 18M nodes
- Approx. 42M edges

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Preprocessing</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dijkstra</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>ALT</td>
<td>0:42</td>
<td>2.2</td>
</tr>
<tr>
<td>CRP</td>
<td>1:00</td>
<td>0.5</td>
</tr>
<tr>
<td>Arc-Flags</td>
<td>0:20</td>
<td>0.3</td>
</tr>
<tr>
<td>CH</td>
<td>0:05</td>
<td>0.2</td>
</tr>
<tr>
<td>TNR</td>
<td>0:20</td>
<td>2.1</td>
</tr>
<tr>
<td>HL</td>
<td>0:37</td>
<td>18.8</td>
</tr>
</tbody>
</table>

In use at Bing, Google, TomTom, ...
New Challenges

More realistic metrics:
- Turn costs, electro mobility
- Points of interests (nearest POIs, shortest via-POIs)
- User customizable metrics e.g., height restrictions, avoid freeways, eco-friendliness, ...
- Fast customization time per metric
- Very small space overhead

Multimodal networks:
- Change the type of transportation during the journey
- Allow only “reasonable” transfers
- Several constraints to the shortest path
- Multicriteria
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):

\[
\text{Consumption} = \begin{cases} 
-2 & \text{if state-of-charge} < 0 \\
4 & \text{if state-of-charge} > M - 2 \\
0 & \text{otherwise}
\end{cases}
\]
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 relationship]
Energy-Optimal Routes

Challenges:

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

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

\[
\begin{align*}
\text{State-of-charge} & \quad \text{Consumption} \\
-2 & \quad \infty \\
4 & \quad M \\
-6 & \quad 2 \\
9 & \quad 4 \\
-2 & \quad 0
\end{align*}
\]
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

Requirements for speedup techniques:
- Shortcuts are functions, not scalar values
- User-dependent consumption profiles (⇒ custom metrics)

Experiments:
- Energy-optimal paths: 63% extra time
- Fastest paths: 62% extra energy
⇒ Energy-optimal routes: follow slow roads

Trading travel time for energy consumption:
- Consider constrained paths
  - E.g., find the fastest path such that the battery does not run out
  - \( \mathcal{NP} \)-hard
- Energy can be saved driving below speed limit
- Additional instructions to the driver
Including Charging Stops

**Task:** Find the fastest path such that the battery does not run out.
- Recharging allowed at some nodes (but requires charging time).
- Realistic models of charging stations:
  - Charging power varies
  - Super chargers
  - Battery swapping stations

**Approach:**
- Extension of bicriteria search
- Propagates charging functions
- CHArge: Combination with CH and A*
  - Optimal routes in seconds / minutes
- Heuristic approaches (based on CHArge)
  - Near-optimal solutions in well below a second
Custom Metrics

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, e.g.
  - Distance
  - Pedestrian
  - Travel time, but don’t use toll roads
  - Travel time, avoid left turns, height restrictions, avoid tolls, ...
- Preprocessing and query time increase significantly
- Higher space overhead
From Theory to Practice: Customizable Contraction Hierarchies [DSW14]

Idea:
- CH topology is the same regardless of metric
- Quickly introduce new metric
From Theory to Practice: Customizable Contraction Hierarchies [DSW14]

Idea:
- CH topology is the same regardless of metric
- Quickly introduce new metric

∞

an edge in the CH
From Theory to Practice: Customizable Contraction Hierarchies [DSW14]

Idea:
- CH topology is the same regardless of metric
- Quickly introduce new metric

establish lower triangle inequality
From Theory to Practice: Customizable Contraction Hierarchies [DSW14]

Idea:
- CH topology is the same regardless of metric
- Quickly introduce new metric

establish lower triangle inequality
Idea:

- CH topology is the same regardless of metric
- Quickly introduce new metric

![Diagram](image)

do this for all lower triangles
What is a Timetable?

Karlsruhe / 10 min

8:00 → 8:31

Frankfurt / 12 min

8:31 → 9:08

Rome / 10 min

Milan / 12 min

8:31 → 11:00
What is a Timetable?

Karlsruhe / 10 min

8:00 → 8:31

Mannheim / 9 min
What is a Timetable?

Karlsruhe / 10 min
8:00 → 8:31
Mannheim / 9 min

Rome / 10 min
8:31 → 11:00
Milan / 12 min
What is a Timetable?

Karlsruhe / 10 min

8:00 → 8:31

Mannheim / 9 min

8:31 → 9:08

Rome / 10 min

8:31 → 11:00

Milan / 12 min

Frankfurt / 12 min
Existing Approaches

list of connections and stops

query
Existing Approaches

list of connections and stops

Time Expanded

[PSWZ08]

query
Existing Approaches

- List of connections and stops
- Time Expanded [PSWZ08]
- Time Dependent [PSWZ08]
- Complex arc weights
- Query
Existing Approaches

- Time Expanded
  - [PSWZ08]

- Time Dependent
  - [PSWZ08]

- list of connections and stops

- query
Existing Approaches

list of connections and stops

Time Expanded
[PSWZ08]

Time Dependent
[PSWZ08]

RAPTOR
[DPW12a]

query
Timetable Queries

- **Inherently time-dependent:** discrete departure times
- **More query scenarios:**
  - Depart now: earliest arrival time?
  - Depart later: shortest travel time?
  - Profile queries: set of journeys with varying departure times
  - Multicriteria: number of transfers, price, ...
- **Different network structure:** less hierarchical, less well-separated, very different schedules at night, ...

![Diagram of travel time vs. departure time]
**Connection Scan (CSA)** [DPSW13]

**Output:** earliest arrival time

**Input:** timetable, source stop, source time, target stop

*missing in the example: footpaths and minimum change times

<table>
<thead>
<tr>
<th>stop ID</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>earliest arrival time</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>elementary connections ordered by departure time</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>dep.stop</th>
<th>arr.stop</th>
<th>dep.time</th>
<th>arr.time</th>
<th>dep.stop</th>
<th>arr.stop</th>
<th>dep.time</th>
<th>arr.time</th>
<th>dep.stop</th>
<th>arr.stop</th>
<th>dep.time</th>
<th>arr.time</th>
</tr>
</thead>
</table>

*missing in the example: footpaths and minimum change times*
### Connection Scan (CSA) [DPSW13]

**Output:** earliest arrival time  
**Input:** timetable, source stop, source time, target stop

<table>
<thead>
<tr>
<th>stop ID</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>earliest arrival time</td>
<td>$+\infty$</td>
<td>$+\infty$</td>
<td>$+\infty$</td>
<td>$+\infty$</td>
<td>$+\infty$</td>
<td>$+\infty$</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>elementary connections ordered by departure time</th>
<th>dep: 1</th>
<th>arr: 3</th>
<th>dep: 3</th>
<th>arr: 4</th>
<th>dep: 3</th>
<th>arr: 4</th>
<th>dep: 3</th>
<th>arr: 4</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>dep</td>
<td>9:00</td>
<td>9:25</td>
<td>9:15</td>
<td>9:45</td>
<td>9:40</td>
<td>9:55</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

*missing in the example: footpaths and minimum change times*
Connection Scan (CSA) [DPSW13]

**Output:** earliest arrival time  
**Input:** timetable, source stop, source time, target stop

<table>
<thead>
<tr>
<th>stop ID</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>earliest arrival time</td>
<td>⋮</td>
<td>+∞</td>
<td>8:00</td>
<td>+∞</td>
<td>+∞</td>
<td>+∞</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>elementary connections ordered by departure time</th>
<th>dep: 1</th>
<th>arr: 3</th>
<th>9:00</th>
<th>9:25</th>
<th>dep: 3</th>
<th>arr: 4</th>
<th>9:15</th>
<th>9:45</th>
<th>dep: 3</th>
<th>arr: 4</th>
<th>9:40</th>
<th>9:55</th>
<th>⋮</th>
</tr>
</thead>
</table>

*missing in the example: footpaths and minimum change times*
**Output:** earliest arrival time  
**Input:** timetable, source stop, source time, target stop

<table>
<thead>
<tr>
<th>stop ID</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>earliest arrival time</td>
<td>+∞</td>
<td>8:00</td>
<td>+∞</td>
<td>+∞</td>
<td>+∞</td>
<td>...</td>
</tr>
</tbody>
</table>

**elementary connections ordered by departure time**

<table>
<thead>
<tr>
<th></th>
<th>dep: 1</th>
<th>dep: 3</th>
<th>dep: 3</th>
<th>dep: 3</th>
<th>dep: 4</th>
<th>dep: 4</th>
<th>dep: 4</th>
<th>dep: 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>arr: 3</td>
<td>9:00</td>
<td>9:25</td>
<td>9:15</td>
<td>9:45</td>
<td>9:40</td>
<td>9:55</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

*missing in the example: footpaths and minimum change times*
Output: earliest arrival time
Input: timetable, source stop, source time, target stop

<table>
<thead>
<tr>
<th>stop ID</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>earliest arrival time</td>
<td>$+\infty$</td>
<td>8:00</td>
<td>$+\infty$</td>
<td>9:25</td>
<td>$+\infty$</td>
</tr>
</tbody>
</table>

elementary connections ordered by departure time

<table>
<thead>
<tr>
<th>dep: 1</th>
<th>arr: 3</th>
<th>9:00</th>
<th>9:25</th>
<th>dep: 3</th>
<th>9:15</th>
<th>9:45</th>
<th>dep: 3</th>
<th>9:40</th>
<th>9:55</th>
<th>$\cdots$</th>
</tr>
</thead>
</table>

*missing in the example: footpaths and minimum change times*
Connection Scan (CSA) \([\text{DPSW13}]\)

**Output:** earliest arrival time  
**Input:** timetable, source stop, source time, target stop

<table>
<thead>
<tr>
<th>stop ID</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>earliest arrival time</td>
<td>(\infty)</td>
<td>8:00</td>
<td>(\infty)</td>
<td>9:25</td>
<td>(\infty)</td>
</tr>
</tbody>
</table>

*missing in the example: footpaths and minimum change times*
**Connection Scan (CSA)** [DPSW13]

**Output:** earliest arrival time

**Input:** timetable, source stop, source time, target stop

<table>
<thead>
<tr>
<th>stop ID</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>earliest arrival time</td>
<td>...</td>
<td>$+\infty$</td>
<td>8:00</td>
<td>$+\infty$</td>
<td>9:25</td>
</tr>
</tbody>
</table>

**elementary connections ordered by departure time**

<table>
<thead>
<tr>
<th>dep</th>
<th>arr</th>
<th>9:00</th>
<th>9:25</th>
<th>dep</th>
<th>arr</th>
<th>9:15</th>
<th>9:45</th>
<th>dep</th>
<th>arr</th>
<th>9:40</th>
<th>9:55</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>dep: 1</td>
<td>arr: 3</td>
<td>9:00</td>
<td>9:25</td>
<td>dep: 3</td>
<td>arr: 4</td>
<td>9:15</td>
<td>9:45</td>
<td>dep: 3</td>
<td>arr: 4</td>
<td>9:40</td>
<td>9:55</td>
<td>...</td>
</tr>
</tbody>
</table>

*missing in the example: footpaths and minimum change times*
### Connection Scan (CSA)  [DPSW13]

**Output:** earliest arrival time  
**Input:** timetable, source stop, source time, target stop  

<table>
<thead>
<tr>
<th>stop ID</th>
<th>earliest arrival time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$+\infty$</td>
<td>8:00</td>
<td>$+\infty$</td>
<td>9:25</td>
<td>9:55</td>
<td>$+\infty$</td>
</tr>
</tbody>
</table>

**Elementary connections ordered by departure time**


*missing in the example: footpaths and minimum change times*
Connection Scan (CSA) [DPSW13]

**Output:** earliest arrival time

**Input:** timetable, source stop, source time, target stop

<table>
<thead>
<tr>
<th>stop ID</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>earliest arrival time</td>
<td>$+\infty$</td>
<td>8:00</td>
<td>$+\infty$</td>
<td>9:25</td>
<td>9:55</td>
</tr>
</tbody>
</table>

*missing in the example: footpaths and minimum change times*

**time table graph is a DAG**

faster than Dijkstra, better use of modern processor architectures
## Experimental Evaluation

**Input: timetable**
- London: 5 M connections, 21 k stops
- Germany: 46 M connections, 252 k stops

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time [ms]</th>
<th>speed-up.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>London</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TE Dijkstra</td>
<td>44.8</td>
<td>—</td>
</tr>
<tr>
<td>TD Dijkstra</td>
<td>10.9</td>
<td>4.1</td>
</tr>
<tr>
<td>CSA</td>
<td>1.8</td>
<td>24.9</td>
</tr>
<tr>
<td><strong>DE</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TE Dijkstra</td>
<td>2960.2</td>
<td>—</td>
</tr>
<tr>
<td>CSA</td>
<td>298.6</td>
<td>9.9</td>
</tr>
<tr>
<td>CSAccel</td>
<td>8.7*</td>
<td>340.2</td>
</tr>
</tbody>
</table>

Intel Xeon E5-2670, 2.6 GHz, 64 GiB DDR3-1600 RAM, 20 MiB L2 cache

*preprocessing: 30 min, 256.4 MiB*
Worldwide network composed of car, rail, flight, ...
Multimodal Routing

Up to now:
- Restricted to one transportation network
- Time-independent and time-dependent (separately)

What we really want is planning a journey by

- Choosing source and destination
- Desired means of transportation (car, train, flight, . . .)
- . . . in a mixed network
Adapting Speed-Up Techniques

Bidirectional search
easily adaptable (time-dependency is hard)

Goal-directed search
ALT adaptable but low speed-ups,
Arc-Flags turns out difficult

Contraction
adaptable with some restrictions
  ‣ Contracted graph is called the Core

two promising approaches:
  ‣ Access-node routing (ANR)
    adapting ideas from transit-node routing (table lookups)
  ‣ User-constrained CH (UCCH)
    augmenting contraction hierarchies
**Problem:** Unrestricted journeys allow arbitrary transfers

- Subway line
- Private car
- Subway line

Dorothea Wagner – Route Planning Algorithms in Transportation Networks
May 18, 2015, Warsaw, Poland
Problem: Unrestricted journeys allow arbitrary transfers

- Not all sequences of transportation modes are reasonable
Multiple Transportation Modes

**Problem:** Unrestricted journeys allow arbitrary transfers

- Not all sequences of transportation modes are reasonable
- Preferred mode of transport varies between users
Solution

“Label Constrained Shortest Path Problem” (LCSPP)

- Define alphabet of transportation mode
- Finite-state automaton describes sequences of vehicles
- Every path must fulfill the requirements imposed by the automaton
Solution

“Label Constrained Shortest Path Problem” (LCSPP)

- Define alphabet of transportation mode
- Finite-state automaton describes sequences of vehicles
- Every path must fulfill the requirements imposed by the automaton

Algorithms for LCSPP

- Dijkstra on the product graph with the automaton works but is slow [BJM00]
- Speed-up techniques: ANR [DPW09], SDALT [KLPC11]
- Automaton as input during the query: UCCH [DPW12b]
User-constrained CH (UCCH) [DPW12b]

Multimodal CH:

- Contraction introduces shortcuts with label sequences
- Witness search depends on constraints
  requires a-priori knowledge of the constraint automata

**Idea:** do not contract nodes with incident link-edges.

- Contraction and witness search are limited to each modality
  ⇒ Preprocessing independent of mode sequence constraints
Example: UCCH Preprocessing
Preprocessing
- Linked nodes are not contracted thus contained in the core
- Shortcuts between core nodes preserve distances
  allows using the road network between rail stations

Query
- CH search on the component
- Label constrained search on the core
- Engineering yields further improvement
Experimental Evaluation

Networks:
road: europe & north america (50 M nodes, 125 M edges)
train: europe (31 k stops, 1.6 M connections)
flight: Star Alliance (1 172 airports, 28 k connections)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Preprocessing</th>
<th></th>
<th></th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time [h:m]</td>
<td>Space [MiB]</td>
<td>Time [ms]</td>
<td>Speedup</td>
</tr>
<tr>
<td>Dijkstra</td>
<td>—</td>
<td>—</td>
<td>33 862</td>
<td>1</td>
</tr>
<tr>
<td>ANR [DPW09]</td>
<td>3:04</td>
<td>14 050</td>
<td>1.07</td>
<td>31 551</td>
</tr>
<tr>
<td>UCCH [DPW12b]</td>
<td>1:18</td>
<td>542</td>
<td>0.67</td>
<td>50 540</td>
</tr>
<tr>
<td>Dijkstra</td>
<td>—</td>
<td>—</td>
<td>35 261</td>
<td>1</td>
</tr>
<tr>
<td>ANR [DPW09]</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>UCCH [DPW12b]</td>
<td>1:27</td>
<td>558</td>
<td>70.52</td>
<td>500</td>
</tr>
</tbody>
</table>

Intel Xeon E5430, 2.66 GHz, 32 GiB RAM, 12 MiB L2 cache
Solution?

Problems of LCSPP

- Restrictions must be known in advance
- User might not know them
- Only a single (best?) journey is computed (no alternatives)

Goal: compute a useful set of multimodal journeys

Dorothea Wagner – Route Planning Algorithms in Transportation Networks
May 18, 2015, Warsaw, Poland
Solution?

Problems of LCSPP

$s$  \(\Rightarrow\)  $t$
Solution?

Problems of LCSPP

- Restrictions must be known in advance
- User might not know them
- Only a single (best?) journey is computed (no alternatives)

Goal: compute a *useful set* of multimodal journeys
Multicriteria Multimodal Routing [DDP+13]

Idea: compute multicriteria, multimodal Pareto sets

- Optimize arrival time plus
- Various (per mode of transport) "convenience criteria" for example # transfers (trains), walking time, taxi costs, etc.
Multicriteria Multimodal Routing [DDP+13]

Idea: compute multicriteria, multimodal Pareto sets

- Optimize arrival time plus
- Various (per mode of transport) „convenience criteria“
  for example # transfers (trains), walking time, taxi costs, etc.

Known problem: Pareto set sizes explode in the number of criteria

Dorothea Wagner – Route Planning Algorithms in Transportation Networks
May 18, 2015, Warsaw, Poland
Relevant Journeys

- 10 min of walking to arrive 10 sec earlier?
- 1 hour of bus drive to walk 10 sec less?
Relevant Journeys

- 10 min of walking to arrive 10 sec earlier?
- 1 hour of bus drive to walk 10 sec less?
- Rate the journeys using fuzzy logic [FA04]
- Journeys with a higher rating are more relevant
Relevant Journeys

- 10 min of walking to arrive 10 sec earlier?
- 1 hour of bus drive to walk 10 sec less?
- Rate the journeys using fuzzy logic [FA04]
- Journeys with a higher rating are more relevant
Reducing the Amount of Work

Problem: queries are slow (> 1 s)

many irrelevant journeys ⇒ can we avoid computing them?

Filter already during the algorithm

- MCR-hf: fuzzy filter
- MCR-hb: Pareto filter, but discrete criteria

Restricted walking (arbitrary heuristic)

- MCR-tx-ry: max $x$ minutes of walking between vehicles and max. $y$ at source/target

Reduce the dimension/number of criteria

- MR-$x$: increase for every $x$ minutes of walking the #transfers by +1
Experimental Evaluation

London, multimodal:
- Roads: 260 k nodes, 1.4 M edges
- Subway, bus, tram, ...
  - 21 k stops, 5 M connections
- 564 cycle hire station

Criteria: arrival time, # transfers, walking time

<table>
<thead>
<tr>
<th>Algorithm</th>
<th># Sol.</th>
<th>Time [ms]</th>
<th>Quality-6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Avg.</td>
<td>Sd.</td>
</tr>
<tr>
<td>MCR</td>
<td>29.1</td>
<td>1 438.7</td>
<td>100 % 0 %</td>
</tr>
<tr>
<td>MCR-hf</td>
<td>10.9</td>
<td>699.4</td>
<td>89 % 11 %</td>
</tr>
<tr>
<td>MCR-hb</td>
<td>9.0</td>
<td>456.7</td>
<td>91 % 10 %</td>
</tr>
<tr>
<td>MCR-t10-r15</td>
<td>13.2</td>
<td>885.0</td>
<td>30 % 31 %</td>
</tr>
<tr>
<td>MR-10</td>
<td>4.3</td>
<td>39.4</td>
<td>45 % 29 %</td>
</tr>
</tbody>
</table>

Intel Xeon E5-2670, 2.6 GHz, 64 GiB DDR3-1600 RAM, 20 MiB L2 cache
Experimental Evaluation

London, multimodal:
- Roads: 260 k nodes, 1.4 M edges
- Subway, bus, tram, . . .
  - 21 k stops, 5 M connections
- 564 cycle hire station

Criteria: arrival time, # transfers, walking time

<table>
<thead>
<tr>
<th>Algorithm</th>
<th># Sol.</th>
<th>Time [ms]</th>
<th>Avg.</th>
<th>Sd.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCR</td>
<td>29.1</td>
<td>1 438.7</td>
<td>100 %</td>
<td>0 %</td>
</tr>
<tr>
<td>MCR-hf</td>
<td>10.9</td>
<td>699.4</td>
<td>89 %</td>
<td>11 %</td>
</tr>
<tr>
<td>MCR-hb</td>
<td>9.0</td>
<td>456.7</td>
<td>91 %</td>
<td>10 %</td>
</tr>
<tr>
<td>MCR-t10-r15</td>
<td>13.2</td>
<td>885.0</td>
<td>30 %</td>
<td>31 %</td>
</tr>
<tr>
<td>MR-10</td>
<td>4.3</td>
<td>39.4</td>
<td>45 %</td>
<td>29 %</td>
</tr>
</tbody>
</table>

Intel Xeon E5-2670, 2.6 GHz, 64 GiB DDR3-1600 RAM, 20 MiB L2 cache
Conclusion

Summary

- Algorithm Engineering: combination of theory and practice
- (Very) fast route planning on road and timetable networks
- Considered metric matters
- Multimodal route planning is more expensive
  - Network offers many interesting trade-offs between criteria
  - Multicriteria optimization useful, to allow the user to chose his journey

Outlook

- Formalization of quality for multimodal journeys done?
- Scalability: multimodal multicriteria for worldwide routing?
- Additional questions: delay-robustness, park & ride, ...?
Thank you for your attention!


Chris Barrett, Riko Jacob, and Madhav V. Marathe.  
Formal-language-constrained path problems.  

Daniel Delling, Julian Dibbelt, Thomas Pajor, Dorothea Wagner, and Renato F. Werneck.  
Computing multimodal journeys in practice.  

Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck.  
Customizable route planning.  

Edsger W. Dijkstra.  
A note on two problems in connexion with graphs.  

Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner.  
Intriguingly simple and fast transit routing.  

Daniel Delling, Thomas Pajor, and Dorothea Wagner.  
Accelerating multi-modal route planning by access-nodes.  
Daniel Delling, Thomas Pajor, and Renato F. Werneck.
Round-based public transit routing.

Julian Dibbelt, Thomas Pajor, and Dorothea Wagner.
User-constrained multi-modal route planning.

Julian Dibbelt, Ben Strasser, and Dorothea Wagner.
Customizable contraction hierarchies.

Marco Farina and Paolo Amato.
A fuzzy definition of “optimality” for many-criteria optimization problems.

Andrew V. Goldberg and Chris Harrelson.
Computing the shortest path: A* search meets graph theory.

Andrew V. Goldberg, Haim Kaplan, and Renato F. Werneck.
Better landmarks within reach.
Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling.

Contraction hierarchies: Faster and simpler hierarchical routing in road networks.

Andrew V. Goldberg and Renato F. Werneck.

Computing point-to-point shortest paths from external memory.

Martin Holzer, Frank Schulz, and Dorothea Wagner.

Engineering multilevel overlay graphs for shortest-path queries.

Dominik Kirchler, Leo Liberti, Thomas Pajor, and Roberto Wolfer Calvo.

UniALT for regular language constraint shortest paths on a multi-modal transportation network.

Ulrich Lauther.

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

Thomas Pajor.

Multi-modal route planning.

Evangelia Pyrga, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis.

Efficient models for timetable information in public transportation systems.
Peter Sanders and Dominik Schultes.
Highway hierarchies hasten exact shortest path queries.

Ben Strasser and Dorothea Wagner.
Connection scan accelerated.

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

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