On Compact Representation and Robustness of Transfer Patterns in Public Transportation Routing

Master’s Thesis

Jonas Sternisko

Institut für Informatik
Albert-Ludwigs-Universität Freiburg

22 March, 2013
Outline

Transfer Pattern Routing

Compact Representation of Transfer Patterns

Robustness of Transfer Patterns
Outline

Transfer Pattern Routing

Compact Representation of Transfer Patterns

Robustness of Transfer Patterns
Outline

Transfer Pattern Routing

Compact Representation of Transfer Patterns

Robustness of Transfer Patterns
Transfer Pattern Routing

Transfer Patterns

- Example Freiburg → Munich
  - [F, Karlsruhe, M], [F, Titisee, Ulm, M]

State-of-the-art routing algorithm (Hannah Bast et al. [1])

- Optimal transfer patterns
- Efficient direct connection queries

Modeling timetable data

- Time-expanded graph
- Realistic routing adds transfer buffer, walking (Robert Geisberger [2])
- Multi-variate cost model: time of travel, number of transfers
- Pareto-optimal paths by multi-label Dijkstra
Transfer Patterns

Computation and Storage

Perform a full Dijkstra for every station
  ▶ Backtrack optimal paths from each destination
  ▶ Transfers along paths yield patterns

Store patterns as Directed Acyclic Graph
  ▶ Reversed, prefix-free
  ▶ One DAG per station
  ▶ Example for patterns ‘ABC‘, ‘AE‘, ‘ABE‘, ‘ABDE‘, ‘ABCDE‘
Transfer Patterns

Search

Query Graph

- Construction from patterns
- Arcs present direct connections

Efficient search

- Direct connection queries for arc relaxation
- List-intersection based algorithm

Pareto-optimal paths within a few ms
Transfer Pattern Routing

Compact Representation of Transfer Patterns

Robustness of Transfer Patterns
Motivation

Size of the information

- Hardware requirements
- Access speed
- Future: increased number of patterns (multi-modal route planning)
Motivation

Size of the information $\rightarrow$ *Compact representation*

- Hardware requirements
- Access speed
- Future: increased number of patterns (multi-modal route planning)
First Approach
Routing with first transfers

Idea
- Store only the first transfer instead of full patterns
- At search time, recursively construct query graph
- Example Freiburg → Munich

Problems
- Requires same precomputation...
- ...but discards most information
- Less informed, search space twice as large → slower
- Only small advantage in space
First Approach
Routing with first transfers

Idea
- Store only the first transfer instead of full patterns
- At search time, recursively construct query graph
- Example Freiburg → Munich

Problems
- Requires same precomputation...
- ...but discards most information
- Less informed, search space twice as large → slower
- Only small advantage in space
Removing Redundancy (1)

Equal suffixes

- DAG is prefix-free
- Detect and remove equal suffixes
Removing Redundancy (1)

Equal suffixes

- DAG is prefix-free
- Detect and remove equal suffixes
Removing Redundancy (1)

Equal suffixes

- DAG is prefix-free
- Detect and remove equal suffixes
Removing Redundancy (1)

Equal suffixes

- DAG is prefix-free
- Detect and remove equal suffixes
Removing Redundancy (1)

Equal suffixes

- DAG is prefix-free
- Detect and remove equal suffixes
Removing Redundancy (2)

Entry points

Information of destination nodes

- Station id + successors

Destination map determines station id as well

- Merge destination nodes with equal successors
Removing Redundancy (2)

Entry points

Information of destination nodes

- Station id + successors

Destination map determines station id as well

- Merge destination nodes with equal successors

![Diagram showing nodes A, B, C, D, and E with arrows indicating connections. A table labeled TargetMap(A) is shown with entries C: C, D: D, E: E.]
Removing Redundancy (2)

Entry points

Information of destination nodes

- Station id + successors

Destination map determines station id as well

- Merge destination nodes with equal successors
Removing Redundancy (2)

Entry points

Information of destination nodes

- Station id + successors

Destination map determines station id as well

- Merge destination nodes with equal successors

\[
\text{TargetMap}(A) =
\begin{array}{c}
C: * \\
D: D \\
E: * \\
\end{array}
\]
Removing Redundancy (3)

Joint Graph

Observation

- Information of the departure node is redundant
- Context determines station id

Let all patterns share a common root

- Joint DAG resolves redundancy between all DAGs
- Station id is assigned at query time
- Other techniques can be applied on top of that
Removing Redundancy (3)

Joint Graph

Observation

- Information of the departure node is redundant
- Context determines station id

Let all patterns share a common root

- Joint DAG resolves redundancy between all DAGs
- Station id is assigned at query time
- Other techniques can be applied on top of that
Removing Redundancy (3)

Joint Graph

Observation

- Information of the departure node is redundant
- Context determines station id

Let all patterns share a common root

- Joint DAG resolves redundancy between all DAGs
- Station id is assigned at query time
- Other techniques can be applied on top of that
Removing Redundancy (3)

Joint Graph

Observation

- Information of the departure node is redundant
- Context determines station id

Let all patterns share a common root

- Joint DAG resolves redundancy between all DAGs
- Station id is assigned at query time
- Other techniques can be applied on top of that
Removing Redundancy (3)

Joint Graph

Observation

- Information of the departure node is redundant
- Context determines station id

Let all patterns share a common root

- Joint DAG resolves redundancy between all DAGs
- Station id is assigned at query time
- Other techniques can be applied on top of that
Results
Compact Representation

- Computed transfer patterns for Hawaii, Detroit, Toronto, NYC
- Measure size of 4 representations
- Example: Toronto at 1000m walking distance (162M patterns, 5.8 patterns per destination in average) \(^1\)

<table>
<thead>
<tr>
<th></th>
<th>Toronto@1000m: 162.0M, 5.8</th>
</tr>
</thead>
<tbody>
<tr>
<td># internal nodes</td>
<td>24.2M</td>
</tr>
<tr>
<td># destination nodes</td>
<td>27.9M</td>
</tr>
<tr>
<td># arcs</td>
<td>186.2M</td>
</tr>
<tr>
<td>Memory size (Byte)</td>
<td>1.6G</td>
</tr>
<tr>
<td>Byte/pattern</td>
<td>9.8</td>
</tr>
<tr>
<td></td>
<td>TP</td>
</tr>
</tbody>
</table>

\(^1\)TP\(^c\): techniques 1 + 2, jDAG: technique 3, jDAG\(^c\): 1 + 2 + 3
Results
Compact Representation

Evaluation (general)

- Removing equal suffixes saves 10–20% internal nodes
- Merging destination nodes removes 50–80% nodes and a lot of arcs
- Joint DAG shrinks the internal structures by factor $\sim 8$
- Approaches combine very well
- Destination maps become dominant part of the data
Transfer Pattern Routing

Compact Representation of Transfer Patterns

Robustness of Transfer Patterns
Motivation

Transfer patterns vs. real-time updates

Time-consuming precomputation

- Computation of transfer patterns in $O(N^2)$
- Heuristics: important stations, limits
- Still very long

Realistic applications

- Frequent updates: delay
  - Traffic jam, strike, cow on the track, ...
- Can the patterns still guarantee optimal responses?
Motivation
Transfer patterns vs. real-time updates

Time-consuming precomputation $\rightarrow$ Robustness
- Computation of transfer patterns in $O(N^2)$
- Heuristics: important stations, limits
- Still very long

Realistic applications
- Frequent updates: delay
  - Traffic jam, strike, cow on the track, ...
- Can the patterns still guarantee optimal responses?
Delay model

Scenarios

- Delay fixed percentage of trips
- Random, exponentially distributed offset
- After random insertion stop

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Share of trips and average delay</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LOW</strong></td>
<td>25% : 5 min</td>
</tr>
<tr>
<td><strong>MEDIUM</strong></td>
<td>25% : 15 min</td>
</tr>
<tr>
<td><strong>HIGH</strong></td>
<td>25% : 50 min</td>
</tr>
<tr>
<td><strong>SWITZERLAND</strong></td>
<td>10% : 5 min, 3% : 15 min, 1% : 50 min</td>
</tr>
<tr>
<td><strong>GERMANY</strong></td>
<td>20% : 5 min, 10% : 15 min, 5% : 50 min</td>
</tr>
<tr>
<td><strong>INDIA</strong></td>
<td>40% : 5 min, 40% : 15 min, 20% : 50 min</td>
</tr>
</tbody>
</table>
Experimental setup

- Compute transfer patterns
- Update network
- Answer random location queries
  - Original transfer pattern, updated direct connection data
  - Dijkstra on updated network
- Compare and classify responses

<table>
<thead>
<tr>
<th>Class</th>
<th>Difference $d$ to optimal path costs $c^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPTIMAL</td>
<td>$d = 0$</td>
</tr>
<tr>
<td>ALMOST OPTIMAL A</td>
<td>$d \leq 5\text{min} \land \frac{d}{c^*} \leq 5%$</td>
</tr>
<tr>
<td>ALMOST OPTIMAL B</td>
<td>$d \leq 10\text{min} \land \frac{d}{c^*} \leq 10%$</td>
</tr>
<tr>
<td>FAILING</td>
<td>otherwise</td>
</tr>
</tbody>
</table>
## Results

### Robustness

Example: 50,000 successful queries on Toronto

- Classification of responses

<table>
<thead>
<tr>
<th>Country</th>
<th>Optimal</th>
<th>Almost A</th>
<th>Almost B</th>
<th>Failing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Null</td>
<td>99.98%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>Low</td>
<td>99.73%</td>
<td>0.12%</td>
<td>0.04%</td>
<td>0.08%</td>
</tr>
<tr>
<td>Medium</td>
<td>99.59%</td>
<td>0.20%</td>
<td>0.06%</td>
<td>0.13%</td>
</tr>
<tr>
<td>High</td>
<td>99.49%</td>
<td>0.27%</td>
<td>0.07%</td>
<td>0.16%</td>
</tr>
<tr>
<td>Switzerland</td>
<td>99.82%</td>
<td>0.09%</td>
<td>0.02%</td>
<td>0.05%</td>
</tr>
<tr>
<td>Germany</td>
<td>99.54%</td>
<td>0.22%</td>
<td>0.06%</td>
<td>0.16%</td>
</tr>
<tr>
<td>India</td>
<td>97.85%</td>
<td>1.12%</td>
<td>0.31%</td>
<td>0.70%</td>
</tr>
</tbody>
</table>
Results

Robustness

▶ Suboptimal responses: time of travel
Results

Robustness

Evaluation (general)

- Never more than 5% suboptimal queries
- Majority of suboptimal responses is almost optimal
- Even under worst scenario INDIA
- But: A few critical outliers
Summary

Compact Representation

Contribution

- Understanding sources of redundancy
- Several techniques reducing data size, maintain accessibility
- Store twice as many patterns in the same memory

Future work

- Dominant destination maps
  - Joint destination maps
  - Invest in hub selection strategies
- Space-efficient implementation
Summary
Robustness

Contribution
- Indication for robustness
- Even for extreme scenarios
- Quality guarantee for transfer pattern routing

Future work
- Acquire and test with real data
- Dependency from precomputation parameters
- How to improve robustness?