Faster Transit Routing by Hyper Partitioning

Daniel Delling
Thomas Pajor

Julian Dibbelt
Tobias Zündorf

ATMOS · September 7th, 2017
Public Transit Routing

Given:
- Public transportation network
  - Stops
  - Routes / Trips
  - Footpaths

Goal:
- Find optimal s-t-journeys
  - Travel time
  - Number of transfers
Public Transit Routing

**Given:**
- Public transportation network
  - Stops
  - Routes / Trips
  - Footpaths

**Goal:**
- Find optimal s-t-journeys
  - Travel time
  - Number of transfers
Public Transit Routing

Given:
- Public transportation network
- Stops
- Routes / Trips
- Footpaths

Goal:
- Find optimal s-t-journeys
- Travel time
- Number of transfers
Public Transit Routing

Given:
- Public transportation network
- Stops
- Routes / Trips
- Footpaths

Goal:
- Find optimal s-t-journeys
- Travel time
- Number of transfers
Public Transit Routing

Given:
- Public transportation network
- Stops
- Routes / Trips
- Footpaths

Goal:
- Find optimal s-t-journeys
- Travel time
- Number of transfers
Public Transit Routing

**Given:**
- Public transportation network
- Stops
- Routes / Trips
- Footpaths

**Goal:**
- Find optimal s-t-journeys
- Travel time
- Number of transfers
Related Work

Partitioning-based Approaches:
- Very successful on road networks
- Have already been adapted for common public transit algorithms

Connection Scan Accelerated [Strasser et al. ’14]
- Partition of the stops
- Has not been evaluated for full multi-criteria optimization

Scalable Transfer Patterns [Bast et al. ’16]
- Partition of the stops
- Preprocessing takes several hours
Related Work – RAPTOR [Delling et al. ’12/’14]

Overview:
- Round based algorithm
- Operates on routes as fundamental object
- One round \( \hat{=} \) Using one vehicle

rRAPTOR: (for profile queries)
- Collect all possible departures at the source
- Run RAPTOR once for each departure

Properties:
- Easily adapted to additional criteria
- Has not yet been accelerated through preprocessing
Related Work – RAPTOR [Delling et al. ’12/’14]

Overview:
- Round based algorithm
- Operates on routes as fundamental object
- One round = Using one vehicle

rRAPTOR: (for profile queries)
- Collect all possible departures at the source
- Run RAPTOR once for each departure

Properties:
- Easily adapted to additional criteria
- Does not require transitively closed footpath graph
- Has not yet been accelerated through preprocessing
Our Approach

Basic Idea:
- Restrict RAPTOR to a subset of the routes
- Therefore, use a partition of the routes
- For every cell of the partition:
  - Identify routes required for traversing the cell (fill-in)

Required Steps:
- Construct the route graph
- Partition the graph
- Compute the fill-in
- Use partition + fill-in to accelerate query
Route graph and Partitioning

Construction:
- Create a Vertex for every Route in the network
Route graph and Partitioning

Construction:
- Create a Vertex for every Route in the network
Route graph and Partitioning

Construction:
- Vertices are connected by an edge, if they share a stop.
Route graph and Partitioning

Construction:
- Vertices are connected by an edge, if they share a stop
Route graph and Partitioning

Construction:

- Stops with more than two routes result in hyperedges
Route graph and Partitioning

Construction:
- Stops with more than two routes result in hyperedges
Route graph and Partitioning

Construction:
- Footpaths are treated like routes (and become vertices)
Route graph and Partitioning

Construction:
- Vertices are connected by an edge, if they share a stop
Route graph and Partitioning

Construction:
- Vertices are connected by an edge, if they share a stop
Route graph and Partitioning

Construction:
- Finally, multi-edges can be replaced by weighted edges
Route graph and Partitioning

Partitioning:
- Find a minimal edge cut with balanced cells
Route graph and Partitioning

Partitioning:
- Find a minimal edge cut with balanced cells
Route graph and Partitioning

Partitioning:
- Find a minimal edge cut with balanced cells
Route graph and Partitioning

Partitioning:
- Cells of the partition correspond to sets of routes
Route graph and Partitioning

Partitioning:
- Cut edges correspond to cut stops
Query Algorithm

Idea:
- **RAPTOR** restricted to:
  - Source cell
  - Target cell
Query Algorithm

Idea:
- RAPTOR restricted to:
  - Source cell
  - Target cell

Problem:
- Other cells have to be traversed
Fill-In Computation

Goal:
- Find routes required for traveling between cut stops
Fill-In Computation

Goal:
- Find routes required for traveling between cut stops
Fill-In Computation

Goal:
- Find routes required for traveling between cut stops

Approaches:
- Trade off between preprocessing time and fill-in size:
  1. Run rRAPTOR once for every cut stop
  2. Run rRAPTOR for every cut stop, restricted to adjacent cells
  3. Run rRAPTOR for every pair of cell and cut stop
Fill-In Representation

Problem:
- Not all trips of a route have to be part of the fill-in
- Not all stops of a route have to be part of the fill-in
- How can the essential parts of the route be represented?
Fill-In Representation

Problem:
■ Not all trips of a route have to be part of the fill-in
■ Not all stops of a route have to be part of the fill-in
■ How can the essential parts of the route be represented?

Approaches:
■ Mark important events with flags
■ Precompute offsets between important events
■ Create compressed fill-in routes
Query Algorithm

Idea:
- RAPTOR restricted to:
  - Source cell
  - Target cell

Diagram: [Diagram showing a network with nodes and edges, with nodes labeled S and t.]
Query Algorithm

Idea:
- RAPTOR restricted to:
  - Source cell
  - Target cell
Query Algorithm

Idea:
- RAPTOR restricted to:
  - Source cell
  - Target cell
  - Fill-in
Query Algorithm

**Idea:**
- RAPTOR restricted to:
  - Source cell
  - Target cell
  - Fill-in

**Special Case:**
- Source or target is a cut stop
Query Algorithm

Idea:
- RAPTOR restricted to:
  - Source cell
  - Target cell
  - Fill-in

Special Case:
- Source or target is a cut stop
- Cut stops are not part of any cell
Query Algorithm

Idea:
- RAPTOR restricted to:
  - Source cell
  - Target cell
  - Fill-in

Special Case:
- Source or target is a cut stop
- Cut stops are not part of any cell
- Fill-in is sufficient to reach cut stops
Experiments – Instances

Networks:
- Netherlands and Switzerland  
  [datahub.io/dataset/gtfs-nl]  
  [gtfs.geops.ch]
- Footpaths up to 200 meters

Structure:

<table>
<thead>
<tr>
<th>Instance</th>
<th>Netherlands</th>
<th>Switzerland</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stops</td>
<td>54,500</td>
<td>25,607</td>
</tr>
<tr>
<td>Routes</td>
<td>12,989</td>
<td>16,122</td>
</tr>
<tr>
<td>Trips</td>
<td>618,961</td>
<td>1,076,662</td>
</tr>
<tr>
<td>Stop events</td>
<td>13,231,954</td>
<td>12,733,856</td>
</tr>
<tr>
<td>Footpaths</td>
<td>65,018</td>
<td>14,717</td>
</tr>
</tbody>
</table>
## Experiments – Preprocessing

### Preprocessing (partition by hmetis):

<table>
<thead>
<tr>
<th># cells</th>
<th># cut</th>
<th>% fn. rts</th>
<th>% fn. se</th>
<th>[m:s]</th>
<th># cut</th>
<th>% fn. rts</th>
<th>% fn. se</th>
<th>[m:s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>365</td>
<td>31.5</td>
<td>5.3</td>
<td>67:32</td>
<td>155</td>
<td>19.1</td>
<td>1.5</td>
<td>13:02</td>
</tr>
<tr>
<td>4</td>
<td>589</td>
<td>40.7</td>
<td>7.3</td>
<td>82:53</td>
<td>345</td>
<td>32.0</td>
<td>3.5</td>
<td>20:58</td>
</tr>
<tr>
<td>8</td>
<td>1,072</td>
<td>54.7</td>
<td>13.0</td>
<td>113:45</td>
<td>545</td>
<td>42.6</td>
<td>6.1</td>
<td>27:19</td>
</tr>
<tr>
<td>16</td>
<td>1,980</td>
<td>68.2</td>
<td>22.1</td>
<td>203:34</td>
<td>907</td>
<td>52.5</td>
<td>14.4</td>
<td>36:51</td>
</tr>
</tbody>
</table>

### Partition with 8 cells:

[Maps of Netherlands and Switzerland with partitions]
Experiments – Queries

**Query Performance:**
- Average over 10,000 random queries
- Number of rounds (#rnd)
- Number of scanned routes (#rts)
- Percentage of scanned fill-in routes (#fn.rts)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th># cells</th>
<th># rnd</th>
<th># rts</th>
<th>% fn. rts</th>
<th>[ms]</th>
<th># rnd</th>
<th># rts</th>
<th># fn. rts</th>
<th>[ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAPTOR</td>
<td>—</td>
<td>10.0</td>
<td>28,021</td>
<td>—</td>
<td>29.3</td>
<td>9.1</td>
<td>29,090</td>
<td>—</td>
<td>19.3</td>
</tr>
<tr>
<td>HypRAPTOR 2</td>
<td>2</td>
<td>9.8</td>
<td>24,666</td>
<td>7.8</td>
<td>25.0</td>
<td>9.1</td>
<td>25,306</td>
<td>4.4</td>
<td>16.8</td>
</tr>
<tr>
<td>HypRAPTOR 4</td>
<td>4</td>
<td>9.6</td>
<td>21,313</td>
<td>30.4</td>
<td>19.3</td>
<td>8.9</td>
<td>19,654</td>
<td>24.1</td>
<td>11.8</td>
</tr>
<tr>
<td>HypRAPTOR 8</td>
<td>8</td>
<td>9.7</td>
<td>20,278</td>
<td>57.3</td>
<td>17.5</td>
<td>8.8</td>
<td>17,405</td>
<td>49.1</td>
<td>9.3</td>
</tr>
<tr>
<td>HypRAPTOR 16</td>
<td>16</td>
<td>9.9</td>
<td>21,085</td>
<td>77.3</td>
<td>18.2</td>
<td>8.7</td>
<td>17,799</td>
<td>73.0</td>
<td>10.1</td>
</tr>
</tbody>
</table>

[C++ using LLVM 8.1, on a 2015 15-inch MacBook Pro, Core i7, 16 GiB of 1600 MHz DDR-3 RAM]
Conclusion

Our Algorithm:
- First partition based speedup technique for RAPTOR
- Based on route-partition instead of stop-partition
- Based on route-partition instead of stop-partition

Future work:
- Find better partitions
- Use multi-level partitions
- Optimize more criteria
- Evaluate for unrestricted walking
Conclusion

Our Algorithm:
- First partition based speedup technique for RAPTOR
- Based on route-partition instead of stop-partition
- Based on route-partition instead of stop-partition

Future work:
- Find better partitions
- Use multi-level partitions
- Optimize more criteria
- Evaluate for unrestricted walking

Thank you for your attention!