Routing in Multimodal Networks With Bicycles

Student: Mykhailo Zima

Supervisors: David Coudert, Nicolas Nisse
Motivation

- Bicycles - An increasingly popular means of transport.
- Need to develop an algorithm that finds an A to B optimal path.
- Cyclists’ paths preferences depend not only on distance but on a lot of other path features (slopes, traffic etc)
Personalized route planner for bicycles

- Allows users to navigate road networks optimally.
- Based on individual driving styles as well as personal preferences.
- Takes as input
  1) A road network $G = (V, E)$ and a set of cost functions $c_1, c_2, \ldots, c_r$ with $c_i : E \rightarrow [0,\infty)$
  for every metric $i$.
  2) A starting point
  3) A destination
  4) A set of weights $(w_1, w_2, \ldots, w_r)$ that determine which metrics the optimal path should be computed based upon.
  5) A set of parameters that determine some cyclist’s individual riding features.
- Produces as output:
  A set of vertices that corresponds to the path with minimal weighted cost
State of the art - Graph Compression and Dijkstra Variations

1. Direct continuation of N. Vadakke-Palangatt and M. Zima PFE work

2. Graph compression during preprocessing: A very popular approach (18.0 million vertices and 42.5 million edges: Memory, preprocessing time & Query time). Dijkstra - 0.4 Gb, -, 2.2 s.
   a. Hub-labeling - 18.8 Gb, 0:37 h, 0.56 µs [D. Delling, A. Goldberg, R. Werneck, 2013]
   b. Contraction Hierarchy - 0.4 Gb, 0:05 h, 110 µs [Robert Geisberger, Peter Sanders, 2012]
   c. Customizable Route Planning - 0.9 Gb, 1:00 h, 1650 µs [D. Delling, A. Goldberg, T. Pajor, R. Werneck, 2014]
   d. Pruned Landmark Labeling [T. Akiba, Y. Iwata, Y. Yoshida, 2013]

3. Dijkstra speedups: Bidirectional Dijkstra, Heuristic Based Dijkstra (A-star)

4. Tools used: a) OpenStreetMap b) QGIS c) Sagemath
Contribution during the internship

Objectives:

● Find solution for optimal path search that takes into account features specific for cyclists;
● Create a real-world graph that contains these features;
● Make these features balanced in comparison to each other;
● Find different optimal paths for different users according to their preferences.

End result:

Working implementation of the algorithm on Nice’s graph.
The approach to Graph Compression: K-path Covers (Funke et al, 2014) & Cover Hierarchy (Akiba et al, 2016)

**k-Path Cover**: In Graph $G(V, E)$, a set $C \subseteq V$ such that $C \cap P \neq \emptyset$ for any path $P$ of length $k$.

- Minimum $k$-path Cover: A NP-Hard problem
- **k-all-path-cover hierarchy**: Based of vertex-covers (Akiba et al, 2016)
- **Idea**: Nth layers of vertex cover is the $2^N$-path cover of the original graph.

Fig: A 2-KPC Example (K-path cover in blue)
Algorithm modification during internship

- **Reason**: to make the algorithm less dependent on graph topology;
- Before creating overlay layers leave as access nodes only those vertices which have **more than 2 neighbours**.
- Proved to be efficient.
The approach to Graph Compression: Overlay Graphs

- Maintain all the routes possible among the compressed vertices.
- Route info: To relate edge to corresponding route in original graph.
- Cost info (dist, time etc): Single lookup retrieval of the route cost.
Client - server socket system

**Server:**
- operates in **sage**;
- contains precomputed overlay graph;
- **receives queries** from clients, processes them and **sends responses** back.
- responses contain lists of vertices that correspond to the optimal path and cost of paths

**Client:**
- operates in **QGIS**;
- contains full graph;
- **sends queries** which contain:
  - source node;
  - destination node;
  - user weights;
  - parameters.
- **receives responses** and presents them to users.

Client and server communicate with each other using socket system
Client part

- User interface for simple source and destination point, user weights selection;
- The result is presented as a line with highlighted nodes.
Retrieving the graph

- Data downloaded from OpenStreetMap.
- Presented as Shapefile (.shp)
- Afterwards converted to Sage object (.sobj)
- **Steps to retrieve a working graph:**
  1) Convert **multilines** (lines with intermediate points) to edges;
  2) Make the graph **directed** - according to ‘oneway’ tag;
  3) Make the graph **strongly connected**;
  4) Add **cliques** to squares - they are denoted with ‘place=square’ tag.

Example of OpenStreetMap tags in edges represented in QGIS
Retrieving the graph

The initial graph

The processed graph
User’s input during query

The user has to choose values for these **metrics**:

- **Travel time** [0..1] - how fast a user can reach destination;
- **Comfort** [0..1] - how comfortable is user’s ride;
- **Flatness** [0..1] - how many slopes will the route contain.

and these **parameters**:

- **Speed** (m/s);
- **Uphill penalty** - how much uphill ride slows down the user;
- **Downhill speed multiplier** - maximum value of downhill speed;
- **Critical downhill grade** - value when maximum downhill speed is achieved.

More detailed information in appendix.
## Path features which affect cyclist’s choice

<table>
<thead>
<tr>
<th>Feature</th>
<th>Affects</th>
<th>Description</th>
<th>Tags in OpenStreetMap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distance</td>
<td>time, comfort, flatness</td>
<td>length of the edge in m</td>
<td></td>
</tr>
<tr>
<td>Slope</td>
<td>time, flatness</td>
<td>relation of vertices height difference and distance</td>
<td></td>
</tr>
<tr>
<td>Surface</td>
<td>time, comfort</td>
<td>type of surface which affects speed and comfort (asphalt, cobblestone, gravel etc)</td>
<td>smoothness, surface, tracktype</td>
</tr>
<tr>
<td>Highway</td>
<td>comfort</td>
<td>type of road (cycleway, primary, residential, pedestrian etc)</td>
<td>highway, bicycle, cycleway</td>
</tr>
<tr>
<td>Slowdown</td>
<td>time</td>
<td>obstacles that make the cyclist stop (crossings, traffic signals, steps etc)</td>
<td>crossing, highway</td>
</tr>
</tbody>
</table>
Elevation data

- Elevation data **absent** from OpenStreetMap;
- Used **SRTM 1 Arc-Second Global** from EarthExplorer;
- Every edge given slope value by this formula:
  
  \[
  \text{slope}(u, v) = \frac{\text{height}(v) - \text{height}(u)}{\text{distance}(u, v)}
  \]

- **Positive** if uphill, **negative** if downhill.
Slopes metric problems

**Gentle** slopes are easier for cyclists even if they are longer.

**Solution**: Flatness coefficient which polynomially depends on gradient value. (Crispin H.V. Cooper, 2016)

**Less continuous slopes** are easier for cyclists.

**Solution**: Currently an open question.
### Analysis of the algorithm’s performance on the map of Nice

<table>
<thead>
<tr>
<th>Overlay layer</th>
<th>k</th>
<th>Constr. time (s)</th>
<th># vertices</th>
<th># edges</th>
<th>D avg</th>
<th>D max</th>
<th>Dijkstra (ms)</th>
<th>Search (ms)</th>
<th>Speed up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>-</td>
<td>-</td>
<td>100768</td>
<td>200155</td>
<td>3.97</td>
<td>56</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>12.9</td>
<td>17513</td>
<td>43276</td>
<td>4.94</td>
<td>56</td>
<td>873</td>
<td>421</td>
<td>2.07</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>16.8</td>
<td>9356</td>
<td>35529</td>
<td>7.59</td>
<td>146</td>
<td>873</td>
<td>322</td>
<td>2.71</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>20.4</td>
<td>5800</td>
<td>36819</td>
<td>12.7</td>
<td>304</td>
<td>873</td>
<td>276</td>
<td>3.16</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>24.2</td>
<td>3899</td>
<td>51293</td>
<td>26.31</td>
<td>1183</td>
<td>873</td>
<td><strong>240</strong></td>
<td>3.64</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>38.3</td>
<td>2830</td>
<td>137394</td>
<td>97.1</td>
<td>9081</td>
<td>873</td>
<td>280</td>
<td>3.12</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>287</td>
<td>2190</td>
<td>694848</td>
<td>634.56</td>
<td>49508</td>
<td>873</td>
<td>614</td>
<td>1.42</td>
</tr>
</tbody>
</table>

- Overlay layer #3 has the best performance for the graph of Nice with **240 ms** of search;
- Compare it to **358 ms** by the previous version of the algorithm.

Test Machine: Linux machine with 2.10 GHz Intel(R) Core(TM) i3-2310M CPU and 4GB of memory.
## Analysis of the algorithm’s performance on the map of New York City

<table>
<thead>
<tr>
<th>Overlay layer</th>
<th>k</th>
<th>Constr. time (s)</th>
<th># vertices</th>
<th># edges</th>
<th>D avg</th>
<th>D max</th>
<th>Dijkstra (ms)</th>
<th>Search (ms)</th>
<th>Speed up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>-</td>
<td>-</td>
<td>240474</td>
<td>431371</td>
<td>3.59</td>
<td>12</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>27.9</td>
<td>66435</td>
<td>167805</td>
<td>5.05</td>
<td>12</td>
<td>2226</td>
<td>1464</td>
<td>1.52</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>38.4</td>
<td>38874</td>
<td>153786</td>
<td>7.91</td>
<td>24</td>
<td>2226</td>
<td>1206</td>
<td>1.85</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>50.3</td>
<td>26105</td>
<td>171377</td>
<td>13.13</td>
<td>72</td>
<td>2226</td>
<td>947</td>
<td>2.35</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>61</td>
<td>19110</td>
<td>231061</td>
<td>24.18</td>
<td>172</td>
<td>2226</td>
<td>994</td>
<td>2.24</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>81</td>
<td>15100</td>
<td>375574</td>
<td>49.74</td>
<td>1129</td>
<td>2226</td>
<td>1061</td>
<td>2.1</td>
</tr>
</tbody>
</table>

- About **2.2 times larger** than graph of Nice;
- Overlay layer #2 has the best performance for the graph of New York with **947 ms** of search;
- The speed-up is worse than the graph of Nice had.
Conclusions

Achievements

- A shortest path algorithm that takes into account cyclist’s needs is designed and successfully tested;
- Its performance was made more independent on graph topology;
- A real-world graph with data important for cyclists was created;
- Implemented the diameter search DiFUB algorithm.
Further development

- Find a good default ratio between metrics;
- Develop a continuous slopes metric;
- Consider individual user’s preferences;
- Expand the graph to the whole PACA region;
- Implement less curves metric;
- Implement several optimal paths search, not only one.
Thank you!
Appendices
Main Steps in my compression implementation:

1. Create an overlay graph where crossroads are access points
2. Find Vertex cover of the previous layer
3. Create overlay graph for the vertex cover.

My Solution: Custom implement a vertex cover heuristic (LR-deg).

**LR-deg**: Initialize Vertex Cover VC to an empty Set. For each $v \in V$ ($v$ picked in increasing order of degree), add Neighbor($v$) to VC if $v$ not already in VC.

Real time Querying: Funke’s algorithm

A fast bidirectional dijkstra using access points (H. Bast et al, 2007)
Formulas used in the algorithm

Travel time value:

\[ c_1(u, v) = \begin{cases} 
\frac{l(u, v) + a(u, v) \cdot l(u, v)}{r_1(u, v)} + q(u, v) \cdot s & \text{if } a_l > 0, \\
\frac{s_d(u, v, s_{d_{max}})}{r_1(u, v)} + q(u, v) \cdot s & \text{otherwise,} 
\end{cases} \]

where \( l(u, v) \) - edge distance, \( a(u, v) \) - edge slope, \( r_1 \) - r_time coefficient, \( q(u, v) \) - r_slowdown value, \( s \) - speed (m/s), \( sd \) - downhill speed multiplier

where \( s_d(u, v, s_{d_{max}}) := \begin{cases} 
\frac{s_{d_{max}} \cdot d'(u, v)}{d_c} + 1 & \text{if } d'(u, v) > d_c, \\
\text{otherwise,} 
\end{cases} \)

where \( s_{d_{max}} \) - maximum downhill speed multiplier, \( d'(u, v) \) - edge slope, \( d_c \) - critical \( d' \) value when speed equals \( s_{d_{max}} \)

Comfort value:

\[ c_2(u, v) = l(u, v) \cdot \max\{r_s(u, v), r_t(u, v)\} \]

where \( r_s \) - r_surface, \( r_t \) - r_traffic

Flatness value:

\[ c_3(u, v) = \begin{cases} 
10128.974 \cdot a(u, v)^3 - 140.785 \cdot a(u, v)^2 + 6.693 \cdot a(u, v) + 1 & \text{if } a(u, v) > 0, \\
l(u, v) & \text{otherwise.} 
\end{cases} \]
## Initial version of the algorithm performance

Analysis of the initial algorithm's performance on the map of Nice

<table>
<thead>
<tr>
<th>Overlay layer</th>
<th>k</th>
<th>Construction time (s)</th>
<th># vertices</th>
<th># edges</th>
<th>D avg</th>
<th>D max</th>
<th>Dijkstra (ms)</th>
<th>Search (ms)</th>
<th>Speed up</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>7.67</td>
<td>100768</td>
<td>200155</td>
<td>3.97</td>
<td>56</td>
<td>873</td>
<td>1898</td>
<td>0.46</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>23.1</td>
<td>55552</td>
<td>119675</td>
<td>4.31</td>
<td>146</td>
<td>873</td>
<td>975</td>
<td>0.9</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>32.6</td>
<td>30646</td>
<td>79887</td>
<td>5.21</td>
<td>322</td>
<td>873</td>
<td>561</td>
<td>1.56</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>38.7</td>
<td>17169</td>
<td>71627</td>
<td>8.34</td>
<td>1034</td>
<td>873</td>
<td>458</td>
<td>1.91</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>49.2</td>
<td>9923</td>
<td>114459</td>
<td>23.07</td>
<td>4263</td>
<td>873</td>
<td>358</td>
<td>2.44</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>138</td>
<td>6070</td>
<td>365236</td>
<td>120.34</td>
<td>28338</td>
<td>873</td>
<td>466</td>
<td>1.87</td>
</tr>
</tbody>
</table>
Example of optimal path

- The optimal path avoids:
  - hills;
  - major roads;
Diameters of the graph

The paths which are:

- the most time-consuming;
- the least comfortable;
- the most hilly.