Dynamic Highway-Node Routing

Dominik Schultes       Peter Sanders

Institut für Theoretische Informatik – Algorithmik II
Universität Karlsruhe (TH)

http://algo2.itu.uka.de/schultes/hwy/

Rome, June 6, 2007
Static Route Planning in Road Networks

**Task:** determine quickest route from source to target location

**Problem:** for large networks, simple algorithms are too slow

**Assumption:** road network does not change

**Conclusion:** use preprocessed data to accelerate source-target-queries

(research focus during the last years [→ invited talk])

⇒ correctness relies on the above assumption
Dynamic Scenarios

- change entire **cost function**
  (e.g., use different speed profile)

- change a **few edge weights**
  (e.g., due to a traffic jam)
Constancy of Structure

Weaker Assumption:

- **structure** of road network does not change
  
  (no new roads, road removal = set weight to ∞)

  ▸ not a significant restriction

- classification of nodes by ‘importance’ might be slightly perturbed,
  but not completely changed

  (e.g., a sports car and a truck both prefer motorways)

  ▸ performance of our approach relies on that

  (not the correctness)
**Outline**

- **basic concepts**: overlay graphs, covering nodes
- lightweight, efficient **static** approach
- **dynamic** version
Overlay Graph


- graph $G = (V, E)$ is given
- select node subset $S \subseteq V$
Overlay Graph


□ graph $G = (V, E)$ is given

□ select node subset $S \subseteq V$

□ overlay graph $G' := (S, E')$ where

$$E' := \{(s, t) \in S \times S \mid \text{no inner node of the shortest } s-t\text{-path belongs to } S\}$$
Definitions:

- **covered branch**: contains a node from \( S \)
- **covered tree**: all branches covered
- **covering nodes**: on each branch, the node \( u \in S \) closest to the root \( s \)
Query

- bidirectional

- perform search in $G$ till search trees are covered by nodes in $S$
Query

- bidirectional

- perform search in $G$ till search trees are covered by nodes in $S$

- continue search only in $G'$
Covering Nodes

Conservative Approach:

- stop searching in $G$ when all branches are covered

(can be very inefficient)
Covering Nodes

Aggressive Approach:

- do not continue the search in $G$ on covered branches

- can be very inefficient
Covering Nodes

Stall-on-Demand:

- do not continue the search in $G$ on covered branches
- a node $v$ can 'wake' a node $u$ on a covered branch
- $u$ can 'stall' $v$ (if $\delta(u) + w(u, v) < \delta(v)$)
  i.e., search is not continued from $v$
Highway Hierarchies

previous static route-planning approach

determines a hierarchical representation of nodes and edges
Static Highway-Node Routing

- extend ideas from
  - multi-level overlay graphs [HolzerSchulzWagnerWeiheZaroliagis00–07]
  - highway hierarchies [SS05–06]
  - transit node routing [BastFunkeMatijevicSS06–07]

- use highway hierarchies to classify nodes by ‘importance’
  i.e., select node sets $S_1 \supseteq S_2 \supseteq S_3 \ldots$
  (crucial distinction from previous separator-based approach)

- construct multi-level overlay graph

- perform query with stall-on-demand technique
Static Highway-Node Routing

- extend ideas from
  - multi-level overlay graphs [HolzerSchulzWagnerWeiheZaroliagis00–07]
  - highway hierarchies [SS05–06]
  - transit node routing [BastFunkeMatijevicSS06–07]

- use highway hierarchies to classify nodes by ‘importance’
  i.e., select node sets $S_1 \supseteq S_2 \supseteq S_3 \ldots$
  (crucial distinction from previous separator-based approach)

- construct multi-level overlay graph 3 min, 8 bytes/node

- perform query with stall-on-demand technique 1.1 ms

(experiments with a European road network with $\approx 18$ million nodes)
Dynamic Highway-Node Routing

change entire cost function

keep the node sets \( S_1 \supseteq S_2 \supseteq S_3 \ldots \)

recompute the overlay graphs

<table>
<thead>
<tr>
<th>speed profile</th>
<th>default</th>
<th>fast car</th>
<th>slow car</th>
<th>slow truck</th>
<th>distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>constr. [min]</td>
<td>1:40</td>
<td>1:41</td>
<td>1:39</td>
<td>1:36</td>
<td>3:56</td>
</tr>
<tr>
<td>query [ms]</td>
<td>1.17</td>
<td>1.20</td>
<td>1.28</td>
<td>1.50</td>
<td>35.62</td>
</tr>
<tr>
<td>#settled nodes</td>
<td>1 414</td>
<td>1 444</td>
<td>1 507</td>
<td>1 667</td>
<td>7 057</td>
</tr>
</tbody>
</table>
change a few edge weights

- **server scenario:** if something changes,
  - update the preprocessed data structures
  - answer many subsequent queries very fast

- **mobile scenario:** if something changes,
  - it does not pay to update the data structures
  - perform single ‘prudent’ query that takes changed situation into account
Dynamic Highway-Node Routing

change a few edge weights, server scenario

- keep the node sets $S_1 \supseteq S_2 \supseteq S_3 \ldots$

- recompute only possibly affected parts of the overlay graphs
  - the computation of the level-$\ell$ overlay graph consists of $|S_\ell|$ local searches to determine the respective covering nodes
  - if the initial local search from $v \in S_\ell$ has not touched a now modified edge $(u, x)$, that local search need not be repeated
  - we manage sets $A^\ell_u = \{ v \in S_\ell \mid v$’s level-$\ell$ preprocessing might be affected when an edge $(u, x)$ changes $\}$
change a few edge weights, server scenario
Dynamic Highway-Node Routing

Change a few edge weights, mobile scenario

- Keep the node sets $S_1 \supseteq S_2 \supseteq S_3 \ldots$
- Keep the overlay graphs
- Use the sets $A_u^\ell$ to determine for each node $u$ a reliable level $r(u)$
- During a query, at node $u$
  - Do not use edges that have been created in some level $> r(u)$
  - Instead, downgrade the search to level $r(u)$
change a few edge weights, mobile scenario

<table>
<thead>
<tr>
<th>change set</th>
<th>affected</th>
<th>#settled nodes</th>
<th>query time</th>
</tr>
</thead>
<tbody>
<tr>
<td>(motorway edges)</td>
<td>queries</td>
<td>absolute</td>
<td>relative</td>
</tr>
<tr>
<td>1</td>
<td>0.6 %</td>
<td>2 347</td>
<td>(1.7)</td>
</tr>
<tr>
<td>10</td>
<td>6.3 %</td>
<td>8 294</td>
<td>(5.9)</td>
</tr>
<tr>
<td>100</td>
<td>41.3 %</td>
<td>43 042</td>
<td>(30.4)</td>
</tr>
<tr>
<td>1 000</td>
<td>82.6 %</td>
<td>200 465</td>
<td>(141.8)</td>
</tr>
</tbody>
</table>
Summary

Efficient static approach
- fast preprocessing < 20 min
- fast queries 1 ms
- outstandingly low memory requirements 2 bytes/node ⇝ 1.6 ms

Can handle practically relevant dynamic scenarios
- change entire cost function typically < 2 minutes
- change a few edge weights
  * update data structures 2 – 40 ms per changed edge
  OR
  * perform prudent query e.g., 48 ms if 100 motorway edges changed

Numbers refer to the Western European road network with 18 million nodes
Future Work

- make it even faster / less space-consuming
- find simpler / better ways to determine the node sets $S_1 \supseteq S_2 \supseteq S_3 \ldots$
- adapt to many-to-many queries
- deal with time-dependent scenarios (where edge weights depend on the time of day)