Algorithm Engineering for Large Graphs

Fast Route Planning

Veit Batz, Robert Geisberger, Dennis Luxen, Peter Sanders, Christian Vetter

Universität Karlsruhe (TH)

Zurich, October 22, 2008
Route Planning

Goals:

- exact shortest (i.e. fastest) paths in large road networks
- fast queries (point-to-point, many-to-many)
- fast preprocessing
- low space consumption
- fast update operations

Applications:

- route planning systems in the internet, car navigation systems,
- ride sharing, traffic simulation, logistics optimisation
Overview

- **Exact** Contraction Hierarchies – a very simple approach
- Transit Node Routing – getting **really** fast
- Mobile Contraction Hierarchies
- Many-to-many Routing
- Ride Sharing
- Dynamic Scenario
- Time-dependent Contraction Hierarchies
- Future Work
Contraction Hierarchies (CH)
**Main Idea**

**Contraction Hierarchies (CH)**

- contract only one node at a time
  - ⇒ local and cache-efficient operation

in more detail:

- order nodes by “importance”, $V = \{1, 2, \ldots, n\}$
- contract nodes in this order, node $v$ is contracted by

  ```
  foreach pair $(u, v)$ and $(v, w)$ of edges do
  if $(u, v, w)$ is a unique shortest path then
  add shortcut $(u, w)$ with weight $w((u, v, w))$
  ```

- query relaxes only edges to more “important” nodes
  - ⇒ valid due to shortcuts
Example: Construction
Example: Construction
Example: Construction
Example: Construction
Example: Construction
Example: Construction
to identify necessary shortcuts

- **local searches** from all nodes $u$ with incoming edge $(u, v)$
- ignore node $v$ at search
- add shortcut $(u, w)$ iff found distance $d(u, w) > w(u, v) + w(v, w)$
Construction

to identify necessary shortcuts

- local searches from all nodes $u$ with incoming edge $(u, v)$
- ignore node $v$ at search
- add shortcut $(u, w)$ iff found distance $d(u, w) > w(u, v) + w(v, w)$
Node Order

use priority queue of nodes, node $v$ is weighted with a linear combination of:

- **edge difference** $\#\text{shortcuts} - \#\text{edges incident to } v$
- **uniformity** e.g. $\#\text{deleted neighbors}$
- ... 

integrated construction and ordering:
1. remove node $v$ on top of the priority queue
2. contract node $v$
3. update weights of remaining nodes
- modified bidirectional Dijkstra algorithm
- 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 $G_{↓}$
modified bidirectional Dijkstra algorithm

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 \( G^\downarrow \)
modified bidirectional Dijkstra algorithm

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 \( G^\downarrow \)
- modified bidirectional Dijkstra algorithm
- 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 $G^\downarrow$
for a shortcut \((u, w)\) of a path \(\langle u, v, w \rangle\), store middle node \(v\) with the edge

expand path by recursively replacing a shortcut with its originating edges
Stall-on-Demand

- $v$ can be "stalled" by $u$ (if $d(u) + w(u, v) < d(v)$)
- stalling can propagate to adjacent nodes
- search is not continued from stalled nodes

- does not invalidate correctness (only suboptimal paths are stalled)
Experiments

**environment**
- AMD Opteron Processor 270 at 2.0 GHz
- 8 GB main memory
- GNU C++ compiler 4.2.1

**test instance**
- road network of Western Europe (PTV)
- 18,029,721 nodes
- 42,199,587 directed edges
Performance

- E: edge difference
- D: deleted neighbors
- L: limit search space on weight calculation
- S: search space size
- i: (digits) hop limits for testing shortcuts
- V: Voronoi region size
- Q: upper bound on edges in search path
- W: relative betweenness

HNR: 594 s / 802 µs
Worst Case Costs

% of searches

settled nodes

- Red: CH aggr. (max. 884)
- Green: CH eco. (max. 1012)
- Blue: HNR (max. 2148)
Contraction Hierarchies

- foundation for our other methods
- conceptually very simple
- handles dynamic scenarios

Static scenario:

- 7.5 min preprocessing
- 0.21 ms to determine the path length
- 0.56 ms to determine a complete path description
- little space consumption (23 bytes/node)
Transit-Node Routing

[DIMACS Challenge 06, ALENEX 07, Science 07]

joint work with H. Bast, S. Funke, D. Matijevic

- very fast queries
  (down to 1.7 µs, 3,000,000 times faster than Dijkstra)

- winner of the 9th DIMACS Implementation Challenge

- more preprocessing time (2:37 h) and space (263 bytes/node) needed
Mobile Contraction Hierarchies

- preprocess data on a personal computer
- highly compressed blocked graph representation 8 bytes/node
- compact route reconstruction data structure + 8 bytes/node

Experiments on a Nokia N800 at 400 MHz
- cold query with empty block cache 56 ms
- compute complete path 73 ms
- recomputation, e.g. if driver took the wrong exit 14 ms
Many-to-Many Shortest Paths

joint work with S. Knopp, F. Schulz, D. Wagner

[ALENEX 07]

- efficient many-to-many variant of hierarchical bidirectional algorithms
- $10,000 \times 10,000$ table in 10s
Ride Sharing

Current approaches:

- match only ride offers with *identical* start/destination (perfect fit)
- sometimes radial search around start/destination

Our approach:

- driver picks passenger up and gives him a ride to his destination
- find the driver with the *minimal detour* (reasonable fit)

Efficient algorithm:

- adaption of the many-to-many algorithm
Highway-Node Routing

- generalization of contraction hierarchies
- allow multiple nodes in the same ‘importance’-level
  i.e., select node sets $S_1 \supseteq S_2 \supseteq S_3 \ldots$
- construct multi-level overlay graph
- perform multi-level query
- designed for dynamic scenarios
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

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

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

- change a few edge weights
  (e.g., due to a traffic jam)
Assumption:

- **structure** of road network does not change
  (no new roads, road removal = set weight to $\infty$)
  $\Rightarrow$ 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)
  $\Rightarrow$ performance of our approach relies on that
  (not the correctness)
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>
Dynamic Highway-Node Routing

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
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 \( A_{u}^{\ell} = \{ v \in S_\ell \mid v\text{'s level-}\ell \text{ preprocessing might be affected when an edge } (u, x) \text{ changes}\} \)
Dynamic Highway-Node Routing

change a few edge weights, server scenario

<table>
<thead>
<tr>
<th>Road Type</th>
<th>Update Time [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>any</td>
<td>0.1</td>
</tr>
<tr>
<td>motorway</td>
<td>0.1</td>
</tr>
<tr>
<td>national</td>
<td>1</td>
</tr>
<tr>
<td>regional</td>
<td>10</td>
</tr>
<tr>
<td>urban</td>
<td>100</td>
</tr>
</tbody>
</table>

- add traffic jam
- cancel traffic jam
- block road
Dynamic Highway-Node Routing

change a few edge weights, mobile scenario

1. keep the node sets $S_1 \supseteq S_2 \supseteq S_3 \ldots$

2. keep the overlay graphs

3. $C := \text{all changed edges}$

4. use the sets $A_{u}^{\ell}$ (considering edges in $C$) to determine for each node $v$ a reliable level $r(v)$

5. during a query, at node $v$
   - do not use edges that have been created in some level $> r(v)$
   - instead, downgrade the search to level $r(v)$ (forward search only)
Dynamic Highway-Node Routing

change a few edge weights, mobile scenario

reliable levels: \( r(x) = 0, \quad r(s_2) = r(t_2) = 1 \)
Sanders et al.: Route Planning

Level 0
Level 1
Level 2
Level 3
Level 4
Level 5
Level 6
Level 7
Dynamic Highway-Node Routing

change a few edge weights, mobile scenario

iterative variant (provided that only edge weight increases allowed)

1. keep everything (as before)

2. $C := \emptyset$

3. use the sets $A_{u}^{\ell}$ (considering edges in $C$) to determine for each node $v$ a reliable level $r(v)$ (as before)

4. ‘prudent’ query (as before)

5. if shortest path $P$ does not contain a changed edge, we are done

6. otherwise: add changed edges on $P$ to $C$, repeat from 3.
**Dynamic Highway-Node Routing**

Change a few edge weights, mobile scenario

<table>
<thead>
<tr>
<th>change set</th>
<th>affected queries</th>
<th>single pass</th>
<th>iterative</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(motorway edges)</td>
<td>query time [ms]</td>
<td>query time [ms]</td>
<td>#iterations</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.4 %</td>
<td>2.3</td>
<td>1.5</td>
<td>1.0</td>
</tr>
<tr>
<td>10</td>
<td>5.8 %</td>
<td>8.5</td>
<td>1.7</td>
<td>1.1</td>
</tr>
<tr>
<td>100</td>
<td>40.0 %</td>
<td>47.1</td>
<td>3.6</td>
<td>1.4</td>
</tr>
<tr>
<td>1000</td>
<td>83.7 %</td>
<td>246.3</td>
<td>25.3</td>
<td>2.7</td>
</tr>
</tbody>
</table>
Summary

static routing in road networks is easy

⇝ applications that require massive amount or routing

⇝ instantaneous mobile routing

⇝ techniques for advanced models

⇝ updating a few edge weights is OK
Current / Future Work

- Time-dependent edge weights
  challenge: backward search impossible (?)

- Multiple objective functions and restrictions (bridge height, . . .)

- Multicriteria optimization (cost, time, . . .)

- Integrate individual and public transportation

- Other objectives for time-dependent travel

- Routing driven traffic simulation