Transit Node Routing
based on Highway Hierarchies

Peter Sanders        Dominik Schultes

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

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

New York City, November 16, 2006
Route Planning

Goals:

- exact shortest (i.e. fastest) paths in large road networks
- fast queries
- fast preprocessing
- low space consumption

Applications:

- route planning systems in the internet
- car navigation systems
- …
Motivation

‘Problem’: existing solutions are already ‘too fast’.

Example: perform a query (using hwy. hierarchies): $\approx 1 \text{ ms}$

visualise the path (using our Java application): $\approx 400 \text{ ms}$

Counter-Argument:
applications that require a lot of queries (and only a few paths)

- massive traffic simulations
- optimisations in logistics systems
Example:
Karlsruhe → Copenhagen
Example:
Karlsruhe → Berlin
Example:
Karlsruhe → Vienna
Example:
Karlsruhe → Munich
Example:
Karlsruhe → Rome
Example:
Karlsruhe → Paris
Example:
Karlsruhe → London
Example:
Karlsruhe → Brussels
Example:
Karlsruhe → Copenhagen
Example:
Karlsruhe → Berlin
Example:
Karlsruhe → Vienna
Example:
Karlsruhe → Munich
Example:
Karlsruhe → Rome
Example:
Karlsruhe → Paris
Example:
Karlsruhe $\rightarrow$ London
Example:
Karlsruhe → Brussels
First Observation

For **long-distance travel**: leave current location via one of only a few ‘important’ traffic junctions, called **access points**

(→ we can afford to store all access points for each node)

[in Europe: about 10 access points per node on average]
Example:
Karlsruhe → Berlin
Example:
Karlsruhe → Berlin
Example:
Karlsruhe → Berlin
Second Observation

Each access point is relevant for several nodes. \( \Rightarrow \)

union of the access points of all nodes is small, called **transit node set**

(\( \Rightarrow \) we can afford to store the distances between all transit node pairs)

[in Europe: about 10 000 transit nodes]
Transit Node Routing

Preprocessing:

- Identify transit node set $\mathcal{T} \subseteq V$
- Compute complete $|\mathcal{T}| \times |\mathcal{T}|$ distance table
- For each node: identify its access points (mapping $A : V \rightarrow 2^\mathcal{T}$), store the distances

Query (source $s$ and target $t$ given): compute

$$d_{\text{top}}(s, t) := \min \{d(s, u) + d(u, v) + d(v, t) : u \in A(s), v \in A(t)\}$$
Transit Node Routing

Locality Filter:

local cases must be filtered (\(\leadsto\) special treatment)

\[ L : V \times V \rightarrow \{\text{true}, \text{false}\} \]

\[ \neg L(s,t) \implies d(s,t) = d_{\text{top}}(s,t) \]
Related Work

- **separator-based** implementation [Müller et al. 2006]
  - determine separator nodes (= transit nodes)
  - partition the graph into small components
  - access points of node $u$: border nodes of $u$’s component
  - locality filter: “same component?”

- **grid-based** implementation [Bast, Funke, Matijevic 2006]
  - compute geometric subdivision of the network into cells
  - access points: border nodes needed for ‘long-distance’ travel
  - transit nodes: union of all access points
  - locality filter: “less than a certain number of grid cells apart?”
Our Approach: Highway Hierarchies

- **complete** search within a local area
- **search in a (thinner) highway network**
  - minimal graph that preserves all shortest paths
- **contract network**, e.g.,
- **iterate** → highway hierarchy

---

1 presented at ESA 2005 and ESA 2006
Example: New York City
Local Area

- choose neighbourhood radius $r(s)$
  (by a heuristic)

- define neighbourhood of $s$

\[
\mathcal{N}(s) := \{ v \in V \mid d(s, v) \leq r(s) \} 
\]
Edge \((u, v)\) belongs to highway network \(\text{iff}\) there are nodes \(s\) and \(t\) s.t.

\[\square (u, v) \text{ is on the "canonical" shortest path from } s \text{ to } t\]

and

\[\square (u, v) \text{ is not entirely within } \mathcal{N}(s) \text{ or } \mathcal{N}(t)\]
Contraction

contraction network ("core")
= non–bypassed nodes
+ shortcuts
Distance Table: Search Space Example
Compute an all-pairs distance table for the core of the topmost level $\ell$.  

Abort the search when all entrance points in the core of level $\ell$ have been encountered.

Use the distance table to bridge the gap.
HH-based Transit Node Routing

- Compute an all-pairs distance table for the core of the topmost level $\ell$. 13 465 $\times$ 13 465 entries

- Abort the search when all entrance points in the core of level $\ell$ have been encountered. $\approx$ 55 for each direction do not 'search', just perform look-ups

- Use the distance table to bridge the gap. $\approx$ 55 $\times$ 55 entries
1. **locality filter**: use geometric disks

\[ d(s,t) = d_u(s,t) < d_{\text{top}}(s,t) \]

\[ L(s,t) := \text{“disks of } s \text{ and } t \text{ overlap”} \]
2. too many ‘entrance points’

**solution:** fall back on comparatively few ‘access points’

(motivated by the observations from [Bast, Funke, Matijevic 2006])
3. compute top distance table in the original graph

solution: [Knopp, S, S, Schulz, Wagner 2007]

“Computing Many-to-Many Shortest Paths Using Highway Hierarchies”

(e.g. $10,000 \times 10,000$ table in one minute)
for each $t \in \mathcal{T}$, perform **backward search** up to top level $\ell$,
store search space entries $(t, u, d(u, t))$

arrange search spaces: group entries by $u$

for each $s \in \mathcal{T}$, perform **forward search**, 
at each node $u$, scan all entries $(t, u, d(u, t))$ and compute $d(s, u) + d(u, t)$
Second Layer

(to deal with medium range queries)

- secondary transit node set $\mathcal{T}_2 \supset \mathcal{T}$
- secondary access mapping $A_2 : V \rightarrow 2^{\mathcal{T}_2}$
- secondary dist. table $\{d(u, v) : u, v \in \mathcal{T}_2 \land d(u, v) \neq d_{\text{top}}(u, v)\}$
- secondary locality filter $L_2$

$\neg L_2(s, t)$ implies

$d(s, t) = d_{\text{top}}(s, t)$

OR

$d(s, t) = \min \{d(s, u) + d(u, v) + d(v, t) : u \in A_2(s), v \in A_2(t)\}$
Two Concrete Variants

Level  | Layer  | Level  | Layer
---|---|---|---
5 | economical | 1 | 1
3 | 2 | 2
1 | L1 | 4 | 1
0 | L2 | 1 | (3)
0 | L2 | 1 | (3)
Preprocessing

- for each secondary transit node $t \in T_2$:
  - perform backward highway search
  - stop at primary transit nodes (set of backward access points)
  - store search space entries $(t, u, d(u, t))$

- arrange search spaces
for each secondary transit node $s \in \mathcal{T}_2$:

- perform \textbf{forward highway search}
- stop at primary transit nodes (\(\rightsquigarrow\) set of \textbf{forward access points})
- at each node $u$ and for each search space entry $(t, u, d(u, t))$:
  - compute distance $d_u(s, t) := d(s, u) + d(u, t)$ \textbf{via} $u$
  - compute distance $d_{\text{top}}(s, t)$ \textbf{via top layer}
  - if $d_u(s, t) < d_{\text{top}}(s, t)$ then
    - add entry $d_u(s, t)$ to the secondary distance table
    - ensure that the \textbf{disks} around $s$ and $t$ contain $u$

(use similar procedure for lower layers)
### Experiments

<table>
<thead>
<tr>
<th>W. Europe (PTV)</th>
<th>Our Inputs</th>
<th>USA (TIGER/Line)</th>
</tr>
</thead>
<tbody>
<tr>
<td>18 010 173</td>
<td>#nodes</td>
<td>23 947 347</td>
</tr>
<tr>
<td>42 560 279</td>
<td>#directed edges</td>
<td>58 333 344</td>
</tr>
<tr>
<td>13</td>
<td>#road categories</td>
<td>4</td>
</tr>
<tr>
<td>10–130</td>
<td>speed range [km/h]</td>
<td>40–100</td>
</tr>
</tbody>
</table>

![Map of Europe](image1.png)

![Map of USA](image2.png)
### Preprocessing

<table>
<thead>
<tr>
<th>metric</th>
<th>variant</th>
<th>layer 1</th>
<th>layer 2</th>
<th>space</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$</td>
<td>T</td>
<td>$</td>
<td>$</td>
</tr>
<tr>
<td>time</td>
<td>eco</td>
<td>12 111</td>
<td>6.1</td>
<td>184 379</td>
<td>4.9</td>
</tr>
<tr>
<td>USA</td>
<td>gen</td>
<td>10 674</td>
<td>5.7</td>
<td>485 410</td>
<td>4.2</td>
</tr>
<tr>
<td>dist</td>
<td>eco</td>
<td>15 399</td>
<td>17.0</td>
<td>102 352</td>
<td>10.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8 964</td>
<td>10.1</td>
<td>118 356</td>
<td>5.5</td>
</tr>
<tr>
<td>time</td>
<td>eco</td>
<td>11 293</td>
<td>9.9</td>
<td>323 356</td>
<td>4.1</td>
</tr>
<tr>
<td>EUR</td>
<td>gen</td>
<td>11 610</td>
<td>20.3</td>
<td>69 775</td>
<td>13.1</td>
</tr>
<tr>
<td>metric</td>
<td>variant</td>
<td>layer 1 [%]</td>
<td>layer 2 [%]</td>
<td>time</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>---------</td>
<td>----------------------</td>
<td>----------------------</td>
<td>------------</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>wrong</td>
<td>cont'd</td>
<td>wrong</td>
<td>cont'd</td>
</tr>
<tr>
<td>time</td>
<td>eco</td>
<td>0.14</td>
<td>1.13</td>
<td>0.0064</td>
<td>0.2780</td>
</tr>
<tr>
<td>USA</td>
<td>gen</td>
<td>0.11</td>
<td>0.80</td>
<td>0.0014</td>
<td>0.0138</td>
</tr>
<tr>
<td></td>
<td>dist</td>
<td>1.57</td>
<td>8.10</td>
<td>0.0489</td>
<td>2.2352</td>
</tr>
<tr>
<td>time</td>
<td>eco</td>
<td>0.54</td>
<td>2.87</td>
<td>0.0092</td>
<td>0.5843</td>
</tr>
<tr>
<td>EUR</td>
<td>gen</td>
<td>0.26</td>
<td>1.35</td>
<td>0.0015</td>
<td>0.0190</td>
</tr>
<tr>
<td></td>
<td>dist</td>
<td>4.68</td>
<td>18.32</td>
<td>0.1761</td>
<td>4.2764</td>
</tr>
</tbody>
</table>
Local Queries: USA, travel time metric

Query Time [μs]

Dijkstra Rank

- economical
- generous
**Determine Shortest Paths**

- **From source** \( s \), look iteratively for the next adjacent node \( s' \) that leads to **fwd acc pnt** \( u \)

\[
d(s, s') + d(s', u) = d(s, u)
\]

- Analogously, from target to **bwd acc pnt**

- From **fwd acc pnt** \( u \), look iteratively for the next adjacent node \( u' \) in the **topmost level** that leads to **bwd acc pnt** \( v \)

\[
d(u, u') + d(u', v) = d(u, v)
\]

(if necessary, use a **hidden path** instead)

- **Unpack** shortcuts

- If path not through top layer, **Fall back** on highway query
## Determine Shortest Paths

<table>
<thead>
<tr>
<th></th>
<th>preproc.</th>
<th>space</th>
<th>query</th>
<th># hops</th>
</tr>
</thead>
<tbody>
<tr>
<td>USA</td>
<td>4:04</td>
<td>193</td>
<td>258</td>
<td>4,537</td>
</tr>
<tr>
<td>EUR</td>
<td>7:43</td>
<td>188</td>
<td>155</td>
<td>1,373</td>
</tr>
</tbody>
</table>
Summary

- **insight**: human intuition is applicable in this case
- extremely fast queries (down to $5 \mu s$)
- handles all kinds of queries (local/global) (5–20 $\mu s$)
- moderate preprocessing times (down to 46 min)
- output shortest paths quickly (down to $155 \mu s$)
Negative Example
Future Work

- select better transit node set
  (e.g. some combination with the separator-based approach)

- more flexible implementation (to better deal with the distance metric)

- more space-efficient implementation
  (e.g. store access points only at the core-1 nodes)

- use access points as landmarks to guide local search

- improve locality filter (get rid of geometry?)