Advanced Route Planning

G. V. Batz, D. Delling, R. Geisberger, M. Kobitzsch, D. Luxen T. Pajor, P. Sanders, D. Schultes, C. Vetter, D. Wagner

Universität Karlsruhe (TH)

University + Research Center ≈ largest research inst. in Germany
Route Planning

Goals:

- exact shortest paths in large (time-dependent) 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
Advanced Route Planning

What we can do:

☐ plain static routing (very fast)
☐ distance tables (even faster)
☐ turn penalties
☐ mobile implementation
☐ time dependent edge weights
☐ flexible objective functions
☐ traffic jams
Advanced Route Planning

What we are working on:

- energy efficient routes
- modelling alternative routes
- detouring traffic jams realistically
- integration with public transportation
- novel applications
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
  \[
  \text{foreach pair } (u, v) \text{ and } (v, w) \text{ of edges do}
  \]
  \[
  \text{if } \langle u, v, w \rangle \text{ is a unique shortest path then}
  \]
  \[
  \text{add shortcut } (u, w) \text{ with weight } w(\langle u, v, w \rangle)
  \]
- query relaxes only edges to more “important” nodes
  ⇒ valid due to shortcuts

R. Geisberger, P. Sanders, D. Schultes, D. Delling
Example: Construction
Example: Construction

R. Geisberger, P. Sanders, D. Schultes, D. Delling
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** \#shortcuts – \#edges incident to $v$
- **uniformity** e.g. \#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
Query

- 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 \( G^\downarrow \)
**Query**

- 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 $G^\downarrow$
Query

- 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 $G_{↓}$
Outputting Paths

- 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
Saarbrücken to Karlsruhe
299 edges compressed to 13 shortcuts.
Saarbrücken to Karlsruhe

316 settled nodes and 951 relaxed edges
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)
Dynamic Scenarios

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

- change a few edge weights
  (e.g., due to a traffic jam)
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
- query after 1 000 edge-weight changes, e.g. traffic jams 699 ms
Even Faster – 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 $\mu$s, 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

SciAm50 Award
Example
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 × 10 000 table in 10s
Energy Efficient Routes

Project MeRegioMobil
Moritz Kobitzsch
+DA Sabine Neubauer, PTV

Even more detailed model
(cost-time tradoff
controlled via hourly wage)
Flexible Objective Functions

Two labels at each edge, e.g., travel time and cost
(mostly $\sim$ energy consumption)

Cost function: arbitrary linear combination

Ideas:

- CHs with valid parameter ranges at each shortcut
- Different node orderings for important nodes
- combine with landmark based goal directed search
Alternative Routes DA Jonathan Dees, BMW

- What are good alternative route graphs
- Evaluate heuristics for finding them


**Time-Dependent Route Planning**

- edge weights are travel time functions:
  - \{time of day $\mapsto$ travel time\}
  - piecewise linear
  - FIFO-property $\Rightarrow$ waiting does not help

- Earliest Arrival Query: $(s, t, \tau_0)$
  $\rightarrow$ a fastest $s$–$t$-route departing at $\tau_0$

- Profile Query: $(s, t, [\tau, \tau'])$
  $\rightarrow$ fastest travel times departing between $\tau$ and $\tau'$. 

Travel Time Functions

we need three operations

- evaluation: $f(\tau)$  
  \[ \mathcal{O}(1) \] time

- merging: $\min(f, g)$  
  \[ \mathcal{O}(|f| + |g|) \] time

- chaining: $f \ast g$ ($f$ “after” $g$)  
  \[ \mathcal{O}(|f| + |g|) \] time

note: $\min(f, g)$ and $f \ast g$ have $\mathcal{O}(|f| + |g|)$ points each.

⇒ increase of complexity
Time-Dependent Dijkstra

Only one difference to standard Dijkstra:

- Cost of relaxed edge \((u, v)\) depends...
- ...on shortest path to \(u\).
Profile Search

Modified Dijkstra:

- Node labels are **travel time functions**
- Edge relaxation: \( f_{\text{new}} := \min(f_{\text{old}}, f_{u,v} \cdot f_u) \)
- PQ key is \( \min f_u \)

\( \Rightarrow \) A **label correcting** algorithm
Avoiding Shortcuts
in the time-dependent case

How to know that a shortcut is not needed?

⇒ No shortest path leads ever over $\langle u, v, w \rangle$
⇒ Don’t insert a shortcut!
Avoiding Shortcuts
in the time-dependent case

How to know that a shortcut is not needed?

⇒ If a shortest path leads over \langle u, v, w \rangle for at least one departure time
⇒ Insert a shortcut!
**ATIC = Approximated TCH**

A **Space Efficient** Data Structure

- **For each edge of the TCH do**
  - Replace weights of shortcuts by two approximated functions...
  - ...an upper bound
  - ...a lower bound
  - ...both with much less points
  - ...lower bound given implicitly by upper bound

⇒ Needs much less space (10 vs. 23 points).
## Earliest Arrival Queries on ATCHs

### Performance

<table>
<thead>
<tr>
<th>graph</th>
<th>method</th>
<th>$\varepsilon$ [%]</th>
<th>space [B/n]</th>
<th>query [ms]</th>
<th>error [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>ABS</td>
<td>OVH</td>
<td>SPD</td>
<td>MAX</td>
</tr>
<tr>
<td>Germany</td>
<td>TCH</td>
<td>–</td>
<td>994</td>
<td>899</td>
<td>0.72</td>
</tr>
<tr>
<td></td>
<td>ATCH</td>
<td>1</td>
<td>239</td>
<td>144</td>
<td>1.27</td>
</tr>
<tr>
<td></td>
<td>ATCH</td>
<td>$\infty$</td>
<td>118</td>
<td>23</td>
<td>1.45</td>
</tr>
<tr>
<td>Europe</td>
<td>TCH</td>
<td>–</td>
<td>589</td>
<td>513</td>
<td>1.89</td>
</tr>
<tr>
<td></td>
<td>ATCH</td>
<td>1</td>
<td>207</td>
<td>131</td>
<td>2.47</td>
</tr>
<tr>
<td></td>
<td>ATCH</td>
<td>$\infty$</td>
<td>99</td>
<td>23</td>
<td>15.43</td>
</tr>
</tbody>
</table>
Profile Queries on ATCHs with Corridor Contraction

Performance

<table>
<thead>
<tr>
<th>graph</th>
<th>method</th>
<th>( \varepsilon ) [%]</th>
<th>space [B/n] ABS</th>
<th>space [B/n] OVH</th>
<th>query [ms]</th>
<th>error [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Earliest Arrival Query</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Germany</td>
<td>TCH</td>
<td>–</td>
<td>994</td>
<td>899</td>
<td>1 112.04</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>ATCH</td>
<td>1</td>
<td>239</td>
<td>144</td>
<td>39.23</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>ATCH</td>
<td>( \infty )</td>
<td>118</td>
<td>23</td>
<td>81.07</td>
<td>0.00</td>
</tr>
<tr>
<td>Europe</td>
<td>TCH</td>
<td>–</td>
<td>589</td>
<td>513</td>
<td>4 308.35</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>ATCH</td>
<td>1</td>
<td>207</td>
<td>131</td>
<td>468.43</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>ATCH</td>
<td>( \infty )</td>
<td>99</td>
<td>23</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>
Public Transportation and CHs

Problems:

☐ Less hierarchy
☐ Multicriteria a MUST
☐ complex modelling (walking, changeover delays, . . .)
☐ prices are not edge based

Approaches:

☐ SHARC: Contraction + arc flags [Delling et al.]
☐ Transfer Patterns [Google Zürich]
  ~ transit node routing
☐ Station-Based CHs [R. Geisberger]
  ~⇒ more complex edge information
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

⇒ matches a request to 100 000 offers in ≈ 25 ms
“Ultimate” Routing in Road Networks?

Massive floating car data $\leadsto$ accurate current situation
Past data $+$ traffic model $+$ real time simulation
$\leadsto$ Nash equilibrium predicting near future

time dependent routing in Nash equilibrium
$\leadsto$ realistic traffic-adaptive routing

Yet another step further

traffic steering towards a social optimum
Summary

static routing in road networks is easy

⇒ applications that require massive amount or routing

⇒ instantaneous mobile routing

⇒ techniques for advanced models

time-dependent routing is fast

⇒ bidirectional time-dependent search

⇒ fast queries

⇒ fast (parallel) precomputation
More Future Work

- Multiple objective functions and restrictions (bridge height, ...)
- Other objectives for time-dependent travel