Algorithm Engineering for Large Graphs

Fast Route Planning

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

Universität Karlsruhe (TH)
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
Contraction Hierarchies

- 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
  $\Rightarrow$ valid due to shortcuts
Contraction Hierarchies

- 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
  $\Rightarrow$ valid due to shortcuts
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. **pop** node $v$ on top of the priority queue

2. **contract** node $v$

3. **update weights** of remaining nodes
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)
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
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
Many-to-Many Shortest Paths

- input: sources $S = \{s_1, \ldots, s_n\}$ and targets $T = \{t_1, \ldots, t_m\}$
- naive algorithm a: perform $\min(n, m)$ Dijkstra one-to-many searches
  $n = m = 10000$: $10000 \cdot 5s \approx 13.9h$
- naive algorithm b: perform $n \cdot m$ TNR-queries
  $n = m = 10000$: $10000 \cdot 10000 \cdot 1.7\mu s = 170s$
- better algorithm: exploit hierarchical nature of CH
Many-to-Many Shortest Paths

- perform $n$ forward-upward searches from each $s_i$
- store the distance $d = \delta(s_i, v)$ of each reached node $v$ in buckets
- then perform $m$ backward-upward searches from each $t_j$
- scan buckets at each reached node
- correctness of CH ensures that

$$d(s_i, t_j) = \min_{v \text{ reached}} \left( \delta(s_i, v) + \delta(v, t_j) \right)$$
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:

☐ adaptation of the many-to-many algorithm

⇒ matches a request to 100,000 offers in ≈ 25 ms
Turn Penalties

- Convert node-based graph to edge-based graph
- Apply speedup technique, e.g. CH
- Germany: 1.8 → 12 min preprocessing, 200 → 422 µs query
Dynamic Scenarios

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

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

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
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 1000 edge-weight changes, e.g. traffic jams 699 ms
Time-Dependent Route Planning

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

- query \( (s, t, \tau_0) \) \( \leftarrow \) start, target, departure time

- looking for:
  a fastest route from \( s \) to \( t \) depending on \( \tau_0 \)

\( \Rightarrow \) Earliest Arrival Problem
we need three operations

- **evaluation:** \( f(\tau) \) — “\( O(1) \)” time
- **merging:** \( \min(f, g) \) — \( O(|f| + |g|) \) time
- **chaining:** \( f \star g \) (\( f \) “after” \( g \)) — \( O(|f| + |g|) \) time

**note:** \( \min(f, g) \) and \( f \star g \) have \( O(|f| + |g|) \) points each.

⇒ increase of complexity
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
Min-Max-Label Search

Approximate version of profile search:

- Computes **upper** and **lower bounds**

- Node labels are pairs $mm_u := (\min f_u, \max f_u)$

- Edge relaxation:
  
  $mm_{new} := \min(mm_{old}, mm_u + (\min f_{u,v}, \max f_{u,v}))$

- PQ key is the lower bound

$\Rightarrow$ A **label correcting** algorithm
two major challenges:

1. contraction during precomputation
   
   witnesses can be found by profile search
   
   ...which is straightforward
   
   ...but incredibly slow!
   
   ⇒ do something more intelligent!

2. bidirectional search
   
   ⇒ problem: arrival time not known
   
   ...but can be solved
Restricted Profile Search

**phase 1:** restricts the search space

- min-max-label search
- might already find a witness
- if not: mark a corridor of nodes:
  - initially mark node \( w \)
  - for each node \( v' \) mark only those two predecessors corresponding to the upper / lower bound

**phase 2:** profile search only using marked nodes
**Bidirectional Time-Dependent Search**

**phase 1:** two alternating searches:
- **forward:** time-dependent Dijkstra
- **backward:** min-max-label search
- meeting points are **candidates**

**phase 2:** from all candidates...

...do time-dependent **many-to-one forward Dijkstra**

...only using **visited edges**

...using min/max distances to **prune** search
## Experimental Comparision

<table>
<thead>
<tr>
<th>input</th>
<th>algorithm</th>
<th>PREPROCESSING</th>
<th>QUERIES</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>time</td>
<td>speed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[h:m]</td>
<td>[B/n]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[ms]</td>
<td>up</td>
</tr>
<tr>
<td>Germany</td>
<td>TCH timed ord</td>
<td>1:48 + 0:14</td>
<td>743</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1.19</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 242</td>
</tr>
<tr>
<td>midweek</td>
<td>TCH min ord</td>
<td>0:05 + 0:20</td>
<td>1 029</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1.22</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 212</td>
</tr>
<tr>
<td>Germany</td>
<td>TCH timed ord.</td>
<td>0:38 + 0:07</td>
<td>177</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1.07</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 321</td>
</tr>
<tr>
<td>Sunday</td>
<td>TCH min ord.</td>
<td>0:05 + 0:06</td>
<td>248</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.71</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 980</td>
</tr>
</tbody>
</table>
Parallel Precomputation

contraction:

- contract maximum independent sets of nodes, i.e. nodes that are least important in their 1 hop neighborhood, in parallel
- add shortcuts even in case of equality

node order:

- use the current priority terms in the priority queue
- use 2-3 hop neighborhood for good results
- use priority terms that rarely decrease on update

⇒ 6.5x speedup on 8 cores
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
Current / Future Work

- 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
- Real-time traffic processing for optimal global routing
“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
Macroscopic Traffic Simulation

Goals:
- fast simulation of traffic in large road networks
- based on shortest paths
- exploit speedup techniques

Status of implementation:
- time independent version as student project
- time dependent version under development

Basis for equilibria computation
Nash Equilibria in Road Networks

**Computation:** Iterative simulation with adapted edge weights

Basic approach (simplified):

- Permit set of \( s-t \)-pairs
- For each \( s-t \)-pair (until equilibrium is reached)
  - Compute path and update weights on its edges

**Goals and applications:**

- Develop model for near future predictions of road traffic
- Provide realistic traffic-adaptive routing
- Traffic steering towards social optimum
Multi-Criteria Routing

- multiple optimization criterias
  e.g. distance, time, costs

- flexibility at route calculation time
  e.g. individual vehicle speeds

- diversity of results
  e.g. calculate Pareto-optimal results

- roundtrips with scenic value
  e.g. for tourists
Current State

adopt contraction hierarchies to multi-criteria:

- modify the contraction so the query stays simple
- add all necessary shortcuts during contraction
- do this by modifying the local search
  - linear combination of two: $x + ay$ with $a \in [l, u]$  
    label is now a function of $x$ (see time-dependent CH)
  - linear combination of more: $a_1x_1 + \cdots + a_nx_n$ with $a_i \in [l_i, u_i]$  
  - Pareto-optimal (may add too many shortcuts)

⇒ too many shortcuts needed when done naive
current speedup-techniques largely rely on hierarchy

every optimization criterion has a specific influence on the hierarchy of a road network
  e.g. finding the fastest route contains more hierarchy than finding the shortest route

however multiple criteria interfere with hierarchy, but the algorithm should work fast on large graphs
  e.g. motorways drop in the hierarchy because of road tolls

⇒ new algorithmic ideas necessary