Algorithm Engineering for Route Planning: An Update

International Symposium on Algorithms and Computation 2011
Dorothea Wagner | December 6, 2011
Problem

request:
- find the best connection in a transportation network

idea:
- network as graph $G = (V, E)$
- edge weights are travel times
- shortest paths in $G$ equal quickest connections
- classic problem (Dijkstra)

problems:
- transport networks are huge
- Dijkstra too slow (> 1 second)
Problem

request:
- find the best connection in a transportation network

idea:
- network as graph $G = (V, E)$
- edge weights are travel times
- shortest paths in $G$ equal quickest connections
- classic problem (Dijkstra)

problems:
- transport networks are huge
- Dijkstra too slow (> 1 second)
Speed-Up Techniques

observations:
- Dijkstra visits all nodes closer than the target
- unnecessary computations
- many requests in a hardly changing network

idea:
- two-phase algorithm:
  - offline: compute additional data during preprocessing
  - online: speed-up query with this data
- 3 criteria: preprocessing time and space, speed-up over Dijkstra
Speed-Up Techniques

observations:
- Dijkstra visits all nodes closer than the target
- unnecessary computations
- many requests in a hardly changing network

idea:
- two-phase algorithm:
  - offline: compute additional data during preprocessing
  - online: speed-up query with this data
- 3 criteria: preprocessing time and space, speed-up over Dijkstra
Showpiece of Algorithm Engineering
Showpiece of Algorithm Engineering

Falsifiable Hypotheses

Design
Experiment
Implement
Analyze
Showpiece of Algorithm Engineering

Realistic machine models

Real-world Data

Falsifiable Hypotheses

Design

Experiment

Implement

Analyze

Dorothea Wagner – Algorithm Engineering for Route Planning: An Update

December 6, 2011
Showpiece of Algorithm Engineering

- Performance guarantees & algorithm dependability
- Falsifiable Hypotheses
- Realistic machine models
- Real-world Data

Dorothea Wagner – Algorithm Engineering for Route Planning: An Update December 6, 2011
History I

phase I theory (1959 - 1999):
- improve theoretical worst-case running time
- by introduction of better data structures
- bidirectional search
- A*-search (goal-directed)

phase II speed-up techniques (1999 - 2005):
- started in 1999 with [Schulz et al. 99]
- two approaches: goal-directed and hierarchical approach
- improvement on this for several inputs
- public availability of continental-sized road networks
- focus on these inputs
- DIMACS challenge in 2006
- fastest techniques yield speed-ups of > 3 million over Dijkstra

phase IV: new challenges (since 2008):
- augmented scenarios: time-dependency, multi-criteria, multi-modal...
- revisit timetable information
- back to theory: why do things work?
Landmark-Dijkstra (ALT)

[Goldberg, Harrelson 05]

idea:
- preprocessing:
  - choose some landmarks from the graph (≈ 16)
  - compute distances from and to all landmarks
- search
  - use landmarks and triangle inequality to compute a lower bound to target
  - pushes search towards target
⇒ goal-directed search

remarks:
- known as ALT
- works in dynamic scenarios
Arc-Flags \cite{Lauther04},\cite{Mohring et al. 05,06}

idea:
- partition the graph in $k$ cells
- attach a label of $k$ bits to each edge
- indicates whether $e$ is important for target’s cell
- modified Dijkstra prunes unimportant edges

discussion:
+ simple query algorithm
+ fast
  - time-consuming preprocessing
  - no pruning in target’s cell
Arc-Flags [La04],[MÖ05,06]

idea:
- **partition** the graph in $k$ cells
- attach a **label** of $k$ bits to each edge
- indicates whether $e$ is important for target’s cell
- modified Dijkstra prunes unimportant edges

discussion:
- simple query algorithm
- fast
  - time-consuming preprocessing
- no pruning in target’s cell
Arc-Flags [Lauther04],[Möhring et al. 05,06]

idea:
- **partition** the graph in \( k \) cells
- **attach a label** of \( k \) bits to each edge
- indicates whether \( e \) is important for target’s cell
- **modified Dijkstra** prunes unimportant edges

discussion:
+ simple query algorithm
+ fast
  - time-consuming preprocessing
  - no pruning in target’s cell
Arc-Flags \cite{Lauther04},\cite{Möhring et al. 05,06}

**idea:**
- partition the graph in $k$ cells
- attach a label of $k$ bits to each edge
- indicates whether $e$ is important for target’s cell
- modified Dijkstra prunes unimportant edges

**discussion:**
- simple query algorithm
- fast
  - time-consuming preprocessing
- no pruning in target’s cell
Shortcuts

[Schulz et al 99, Sanders, Schultes 05, Geisberger et al 08]

observation:
- nodes with low degree are **not** important

contract graph
- iteratively remove such nodes
- add shortcuts to preserve distances between non-removed nodes

query:
- bidirectional
- prune edges heading **less** important nodes
 #### Shortcuts

[Schulz et al 99, Sanders, Schultes 05, Geisberger et al 08]

**observation:**
- nodes with low degree are not important

**contract graph**
- iteratively remove such nodes
- add shortcuts to preserve distances between non-removed nodes

**query:**
- bidirectional
- prune edges heading less important nodes
idea:
combine shortcuts (contraction) and arc-flags

results:
- fast query times
- exploits hierarchy
- unidirectional and goal-directed
- can be augmented to time-dependency [Delling 08]
- adaptable to Pareto multi-criteria paths [Delling, Wagner 09]
Space-Efficient SHARC [Brunel et al 10]

problem:
added shortcuts and arc-flags consume too much space
Space-Efficient SHARC [Brunel et al 10]

problem:
added shortcuts and arc-flags consume too much space

1. arc-flag compression
- store unique arc-flags in a separate table
- reduce number of unique arc-flags through bit-flipping
Space-Efficient SHARC [Brunel et al 10]

problem:
added shortcuts and arc-flags consume too much space

2. shortcut removal

- identify unimportant shortcuts in the graph
- remove shortcuts and update arc-flags to maintain correctness
idea: solely use contraction

approach:

- heuristically order nodes by “importance”
- contract nodes in that order
- node $v$ contracted by

1. **forall the** edges $(u, v)$ and $(v, w)$ **do**
2.  
   ```plaintext
   if $(u, v, w)$ unique shortest path then
   3. add shortcut $(u, w)$ with weight $\text{len}(u, v) + \text{len}(v, w)$;
   ```

- query only looks at edges to more important nodes
Example: CH Preprocessing
Example: CH Preprocessing
Example: CH Preprocessing
Example: CH Preprocessing
Example: CH Preprocessing
Example: CH Preprocessing
CH Query

- modified bidirectional Dijkstra
- 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 \( G_{\downarrow} \)
CH Query

- modified bidirectional Dijkstra
- 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 \( G_{↓} \)
CH Query

- modified bidirectional Dijkstra
- 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 \( G_{\downarrow} \)
CH Query

- modified bidirectional Dijkstra
- 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 \( G_{↓} \)
New Challenges

realistic customizable routes:
- consider turn costs
- user customizable metrics
e.g., height restrictions, avoid freeways, eco-friendliness, . . .
- fast customization time per metric
- very small space overhead

timetable information:
- consider public transportation networks
- develop new techniques
- robustness towards the input?
New Challenges II

multi-modal routes:
- change the type of transportation during the journey
- allow only “reasonable” transfers
- several constraints to the shortest path

theoretical analysis:
- explain behavior of speed-up techniques
- complexity of preprocessing
- asymptotic bounds for query time
- graph properties of road networks
Custom Metrics

problem
- preprocessing is metric-dependent
- state-of-the-art algorithms tailored to travel time
  - heavily exploit ‘hierarchy’ of road categories

naive solution
- compute preprocessing for each metric, e.g.
  - distance
  - pedestrian
  - travel time, but don’t use toll roads
  - travel time, avoid left turns, height restrictions, avoid tolls, ...
- preprocessing and query time increase significantly
- higher space overhead
Custom Metrics

**Problem**
- Preprocessing is metric-dependent
- State-of-the-art algorithms tailored to travel time
  heavily exploit ‘hierarchy’ of road categories

**Naive Solution**
- Compute preprocessing for each metric, e.g.
  - Distance
  - Pedestrian
  - Travel time, but don’t use toll roads
  - Travel time, avoid left turns, height restrictions, avoid tolls, . . .
- Preprocessing and query time increase significantly
- Higher space overhead
Turns and Real-Time Traffic

more realism: augment intersections with turn costs

Expanded Model

- 300% space overhead

Compact Model

- 10% space overhead

real time traffic

- new traffic data every minute
- invalidates preprocessing ⇒ infeasible
Turns and Real-Time Traffic

more realism: augment intersections with turn costs

- Expanded Model
  - 300% space overhead

- Compact Model
  - 10% space overhead

real time traffic
- new traffic data every minute
- invalidates preprocessing \( \Rightarrow \) infeasible
more realism: augment intersections with turn costs

**Expanded Model**

- 300% space overhead

**Compact Model**

- 10% space overhead

**Real time traffic**

- new traffic data every minute
- invalidates preprocessing ⇒ infeasible
Turns and Real-Time Traffic

more realism: augment intersections with turn costs

**Expanded Model**

- 300% space overhead

**Compact Model**

- 10% space overhead

real time traffic

- new traffic data every minute
- invalidates preprocessing ⇒ infeasible
Turns and Real-Time Traffic

more realism: augment intersections with turn costs

Expanded Model
- 300% space overhead

Compact Model
- 10% space overhead

real time traffic
- new traffic data every minute
- invalidates preprocessing ⇒ infeasible
Turns and Real-Time Traffic

more realism: augment intersections with turn costs

- Expanded Model
  - 300% space overhead

- Compact Model
  - 10% space overhead

real time traffic

- new traffic data every minute
- invalidates preprocessing ⇒ infeasible
Turns and Real-Time Traffic

more realism: augment intersections with turn costs

Expanded Model
- 300 % space overhead

Compact Model
- 10 % space overhead

real time traffic
- new traffic data every minute
- invalidates preprocessing ⇒ infeasible
Approach: 3 Phases \cite{DGPW11}

1. **metric-independent preprocessing**
   - exploit topology of the network
   - may take a few minutes, linear space overhead

2. **metric customization**
   - compute overlay graphs between boundary nodes in each cell
   - fast: < 30 sec per metric
   - space overhead: small fraction of input

3. **queries**
   - modified bidirectional Dijkstra
   - reasonably fast for each metric ($\approx 1 \text{ ms}$)
   - worst-case guarantees
Phase 1: Partitioning [DGRW11]

idea: exploit only topology of the road network via partitioning

- every path crossing a cell enters/exits via a boundary node
  \[ \Rightarrow \text{minimize } \# \text{ cut edges while cell sizes } \leq U \text{ (input parameter)} \]
Phase 2 and 3: Exploiting Partition

**idea**: precompute distances between boundary nodes

**overlay graph** [JP02, SWZ02]:
- nodes \( \hat{=} \) boundary nodes
- edges between boundary nodes
- + cut edges, – unneeded edges

**search graph**:
- source and target cell
- plus overlay graph
- use bidirectional Dijkstra
Phase 2 and 3: Exploiting Partition

**idea:** precompute distances between boundary nodes

**overlay graph** [JP02, SWZ02]:
- nodes $\hat{=} \text{boundary nodes}$
- edges between boundary nodes
- $+ \text{ cut edges, } - \text{ unneeded edges}$

**search graph:**
- source and target cell
- plus overlay graph
- use bidirectional Dijkstra
Phase 2 and 3: Exploiting Partition

**idea:** precompute distances between boundary nodes

**overlay graph** [JP02, SWZ02]:
- nodes $\hat{\cap}$ boundary nodes
- edges between boundary nodes
- + cut edges, – unneeded edges

**search graph:**
- source and target cell
- plus overlay graph
- use bidirectional Dijkstra
Phase 2 and 3: Exploiting Partition

**idea:** precompute distances between boundary nodes

**overlay graph** [JP02, SWZ02]:
- nodes $\hat{=} \text{boundary nodes}$
- edges between boundary nodes
- + cut edges, − unneeded edges

**search graph:**
- source and target cell
- plus overlay graph
- use bidirectional Dijkstra
Phase 2 and 3: Exploiting Partition

**idea:** precompute distances between boundary nodes

**overlay graph** [JP02, SWZ02]:
- nodes $\hat{=}$ boundary nodes
- edges between boundary nodes
- + cut edges, – unneeded edges

**search graph:**
- source and target cell
- plus overlay graph
- use bidirectional Dijkstra
Phase 2 and 3: Exploiting Partition

**idea:** precompute distances between boundary nodes

**overlay graph** [JP02, SWZ02]:
- nodes $\triangleright$ boundary nodes
- edges between boundary nodes
- + cut edges, – unneeded edges

**search graph:**
- source and target cell
- plus overlay graph
- use bidirectional Dijkstra
Phase 2 and 3: Exploiting Partition

**idea:** precompute distances between boundary nodes

**overlay graph [JP02, SWZ02]:**
- nodes $\hat{=} \text{boundary nodes}$
- edges between boundary nodes
- $+$ cut edges, $-$ unneeded edges

**search graph:**
- source and target cell
- plus overlay graph
- use bidirectional Dijkstra
Phase 2 and 3: Exploiting Partition

**idea:** precompute distances between boundary nodes

**overlay graph** [JP02, SWZ02]:
- nodes \( \hat{=} \) boundary nodes
- edges between boundary nodes
- + cut edges, – unneeded edges

**search graph:**
- source and target cell
- plus overlay graph
- use bidirectional Dijkstra
Phase 2 and 3: Exploiting Partition

**idea:** precompute distances between boundary nodes

**overlay graph** [JP02, SWZ02]:
- nodes $\Rightarrow$ boundary nodes
- edges between boundary nodes
- $+$ cut edges, $-$ unneeded edges

**search graph:**
- source and target cell
- plus overlay graph
- use bidirectional Dijkstra
Phase 2 and 3: Exploiting Partition

**idea:** precompute distances between boundary nodes

**overlay graph** [JP02, SWZ02]:
- nodes $\hat{=} \text{boundary nodes}$
- edges between boundary nodes
- $+$ cut edges, $-$ unneeded edges

**search graph:**
- source and target cell
- plus overlay graph
- use bidirectional Dijkstra
## Experimental Results

<table>
<thead>
<tr>
<th>algorithm</th>
<th>Preprocessing</th>
<th>Queries</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time [h:m]</td>
<td>space [MiB]</td>
</tr>
<tr>
<td>Dijkstra</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Arc-Flags</td>
<td>17:08</td>
<td>326</td>
</tr>
<tr>
<td>SHARC</td>
<td>0:38</td>
<td>240</td>
</tr>
<tr>
<td>CH</td>
<td>0:10</td>
<td>200</td>
</tr>
<tr>
<td>(TNR+AF)</td>
<td>3:49</td>
<td>312</td>
</tr>
</tbody>
</table>

- **no turns**

- **turns**

(Graph: Europe, ≈ 18 M nodes, ≈ 42 M arcs; [BDS+09,GV11])

Fully realistic queries, quick customization and practical query times.
Multi-Modal Routing

up to now, research mostly on uni-modal routing
- restricted to **one** transportation network
- time-independent and time-dependent (separately)

what we **really** want is planning a journey by

- choosing **source** and **destination**
- desired **means** of transportation (car, train, flight, . . .)
- . . . in a **mixed network**
“Classic” Shortest Paths

main challenge for multi-modal route planning:

shortest path

a shortest s-t-path could require too many transfers
Definition (Label Constrained Shortest Path Problem)

Given a $\Sigma$-labeled graph $G = (V, E)$, source/target nodes $s$ and $t$, and a departure time $\tau$, find the shortest $s$-$t$-path where the accumulated edge-labels along the path form a word w.r.t. some language $L \subseteq \Sigma^*$.

Theorem (Barrett et al., 2000)

The Label Constrained Shortest Path Problem (LCSPP) is solvable in polynomial time, if $L$ is a regular language.
Adapting Speed-Up Techniques

[P09,KLPC11]

- bidirectional search
easily adaptable (time-dependency is hard)

- goal-directed search
ALT adaptable but low speed-ups,
Arc-Flags turns out difficult

- contraction
adaptable with some restrictions
  - contracted graph is called the core

two promising approaches:

- access-node routing (ANR)
adapting ideas from transit-node routing (table lookups)

- user-constrained CH (UCCH)
augmenting contraction hierarchies
Access-Node Routing: Idea [DPW09]

**assumption:** road network only used in the beginning and end

**observation:** number of “relevant” entry points in the public transportation network is small

**idea:** compute for each road node its access-nodes and their distances
ANR: Preprocessing

two approaches for computing access-nodes:

forward approach
- **exact** access-nodes
- requires **full profile query** per road node

inverse approach
- **over-approximation**
- requires profile query only on PTN and only per *access-node*

currently only the inverse approach is feasible
Access-Node Routing: Query

given: nodes $s$ and $t$ in the road network

query includes two phases:
- jump into public transit network through access-nodes of $s$ and $t$
- compute distance between the access-nodes

query algorithm on the public transit network can be chosen independently
User-constrained Shortest Paths

- optimality of multi-modal paths depends on user-choice
  desired modes of transport, constraints on the sequence of modes
- user-constraints are an additional input to the query
- preprocessing should respect these query-time constraints
  ANR predetermines the constraint automaton during preprocessing

question: can CH be adapted to this setting?
User-constrained Shortest Paths

- optimality of multi-modal paths depends on user-choice of desired modes of transport, constraints on the sequence of modes
- user-constraints are an additional input to the query
- preprocessing should respect these query-time constraints

ANR predetermines the constraint automaton during preprocessing

**question:** can CH be adapted to this setting?
User-constrained Shortest Paths

- Optimality of multi-modal paths depends on user-choice of desired modes of transport, constraints on the sequence of modes.
- User-constraints are an additional input to the query.
- Preprocessing should respect these query-time constraints. ANR predetermines the constraint automaton during preprocessing.

**Question:** can CH be adapted to this setting?
User-constrained Shortest Paths

- Optimality of multi-modal paths depends on user-choice desired modes of transport, constraints on the sequence of modes
- User-constraints are an additional input to the query
- Preprocessing should respect these query-time constraints

ANR predetermines the constraint automaton during preprocessing

**Question:** Can CH be adapted to this setting?
User-constrained Shortest Paths

- optimality of multi-modal paths depends on user-choice desired modes of transport, constraints on the sequence of modes
- user-constraints are an additional input to the query
- preprocessing should respect these query-time constraints
  ANR predetermines the constraint automaton during preprocessing

**question:** can CH be adapted to this setting?
User-constrained CH (UCCH) [DPW 2012]

multi-modal CH:
- contraction introduces shortcuts with label sequences
- witness search depends on constraints
  requires a-priori knowledge of the constraint automata

idea: do not contract nodes with incident link-edges.

- contraction and witness search are limited to each modality
  ⇒ preprocessing independent of mode sequence constraints
Example: UCCH Preprocessing
UCCH: Details

preprocessing
- linked nodes are not contracted thus contained in the core
- shortcuts between core nodes preserve distances allows using the road network between rail stations

query
- CH search on the component
- label constrained search on the core
- engineering yields further improvement
## Experimental Results

<table>
<thead>
<tr>
<th></th>
<th>Preprocessing</th>
<th>Queries</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time [h:m]</td>
<td>space [MiB]</td>
</tr>
<tr>
<td>flight</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dijkstra</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>ANR</td>
<td>2:41</td>
<td>10 681</td>
</tr>
<tr>
<td>UCCH</td>
<td>1:29</td>
<td>542</td>
</tr>
<tr>
<td>rail+flight</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dijkstra</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>ANR</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>UCCH</td>
<td>2:42</td>
<td>557</td>
</tr>
</tbody>
</table>

Europe and North America road, ≈ 50 M nodes, ≈ 125 M arcs

Reasonable preprocessing and fast query times for user-constrained multi-modal paths.
Shortcut Problem

[Bauer, D’Angelo, Delling, Wagner 09]

theoretical question originating from the technique of inserting shortcuts

**problem description:**
- How to add a given number of shortcuts to a graph such that the number of edges on an edge-minimal shortest path between two random nodes becomes minimal?

**results**
- constant-factor approximation is hard
- stochastical approach to evaluate objective on big networks
- approximation strategies
Preprocessing Speed-Up Techniques is Hard [Bauer et al 10]

situation:
- preprocessing phase of a speed-up technique for static routing
- usually some degree of freedom exists
- example: how to partition the graph (arc-flags)

problem:
- perform the preprocessing optimally

recent outcome:
- NP-hard for most techniques (ALT, Highway-Node routing/Multilevel-overlay graph, Arc-Flags, Sharc, Contraction Hierarchies)
Shortest Path Indices

motivation
- experiments show incredible speed-ups over Dijkstra’s algorithm
- explain why speed-up techniques work so well

shortest path indices
- shortest path indices are *simple indices* that predict how well *complex algorithms* perform on specific datasets

aims
- gain deeper insights on the structure of huge networks
- predict the performance of speed-up techniques on different kinds of graphs
The highway dimension of $G$ is the smallest integer $h$ such that
- for every ball $B$ in $G$ of radius $4r$
- exists a node-set $S$ with $|S| \leq h$ such that
- every shortest path in $B$ of length $\geq r$ contains a node from $S$. 
main result
Given a graph with highway dimension $h$, maximum degree $\Delta$ and diameter $D$

- CH runs in $O((\Delta + h \log D)(h \log D))$ time
- SHARC runs in $O(nh \log n \log D)$ time

... with preprocessing space of no more than $O(nh \log D)$

comparison: Dijkstra runs in $O(m + n \log n)$ time in sparse graphs
Shortest-Path Cover (SPC)

idea:
all shortest paths of a certain length can be covered by a small node set

given:
undirected graph $G = (V, E, \text{len})$

Definition

A set $C$ is called $(r, k)$-SPC of $G$ if and only if

- every ball $B$ of radius $2r$ contains at most $k$ nodes from $C$ and
- every shortest path $P$ of length $r \leq |P| \leq 2r$ contains a node from $C$. 
Highway Dimension and SPCs

**Theorem**

*If G has highway dimension h then \( \forall r \exists a (r, h)\)-SPC.*

**Problem:**

How can we construct a minimum SPC efficiently?

Computing a minimal \((r, h)\)-SPC is NP-hard.

**Theorem**

*If G has highway dimension h, we can construct an \((r, O(h \log n))\)-SPC in polynomial time (for all r).*

**Idea:** Greedy set cover algorithm yields an \(O(\log n)\) approximation.
Conjecture:
Road networks have small (constant) highway dimension.
Labeling Algorithms

generic preprocessing:

- $\forall v \in V : \text{compute label } L(v) \subseteq V$.
- $\forall w \in L(v) : \text{compute dist}(v, w)$.

labeling property:
$\forall s, t \in V$ exists a shortest path $P(s, t)$ with $L(s) \cap L(t) \cap P(s, t) \neq \emptyset$.

Query

Compute $\min_{u \in L(s) \cap L(t)} (\text{dist}(s, u) + \text{dist}(u, t))$

- extremely simple algorithm: Set intersection test (linear time)
- fast if labels are small
Labeling Algorithms

generic preprocessing:
- $\forall v \in V : \text{compute label } L(v) \subseteq V$.
- $\forall w \in L(v) : \text{compute dist}(v, w)$.

labeling property:
$\forall s, t \in V$ exists a shortest path $P(s, t)$ with $L(s) \cap L(t) \cap P(s, t) \neq \emptyset$.

Query

\[
\text{Compute } \min_{u \in L(s) \cap L(t)} (\text{dist}(s, u) + \text{dist}(u, t))
\]

- extremely simple algorithm: Set intersection test (linear time)
- fast if labels are small
Hub Labels (HL) [ADGW11]

- shortest path covers can be used to obtain labels
- but their computation is expensive

Observation
For every node $v \in V$ the upward search space of a CH query yields a valid label.

- use CH search space as labels...
- ... with additional engineering to shrink labels further

fastest known algorithm for point-to-point queries in road networks
Hub Labels (HL) [ADGW11]

- shortest path covers can be used to obtain labels
- but their computation is expensive

Observation
For every node $v \in V$ the upward search space of a CH query yields a valid label.

- use CH search space as labels...
- ...with additional engineering to shrink labels further

fastest known algorithm for point-to-point queries in road networks
## Experimental Results

<table>
<thead>
<tr>
<th>no turns</th>
<th>Preprocessing</th>
<th>Queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>algorithm</td>
<td>time [h:m]</td>
<td>space [MiB]</td>
</tr>
<tr>
<td>Dijkstra</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Arc-Flags</td>
<td>17:08</td>
<td>326</td>
</tr>
<tr>
<td>SHARC</td>
<td>0:38</td>
<td>240</td>
</tr>
<tr>
<td>CH</td>
<td>0:10</td>
<td>200</td>
</tr>
<tr>
<td>(TNR+AF)</td>
<td>3:49</td>
<td>312</td>
</tr>
<tr>
<td>Hub Labels</td>
<td>≈ 4:30</td>
<td>21 300</td>
</tr>
</tbody>
</table>

(Graph: Europe, ≈ 18 M nodes, ≈ 42 M arcs; [BDS+09])
Conclusion

outlook:
- multi-criteria timetable information
- enhance multi-modal route planning
- energy-efficient route planning

on the long run:
- traffic simulation
- variable optimization function

theory:
- what are the reasons for speed-ups?
- predict speed-ups