Algorithm Engineering for Large Graphs

Fast Route Planning

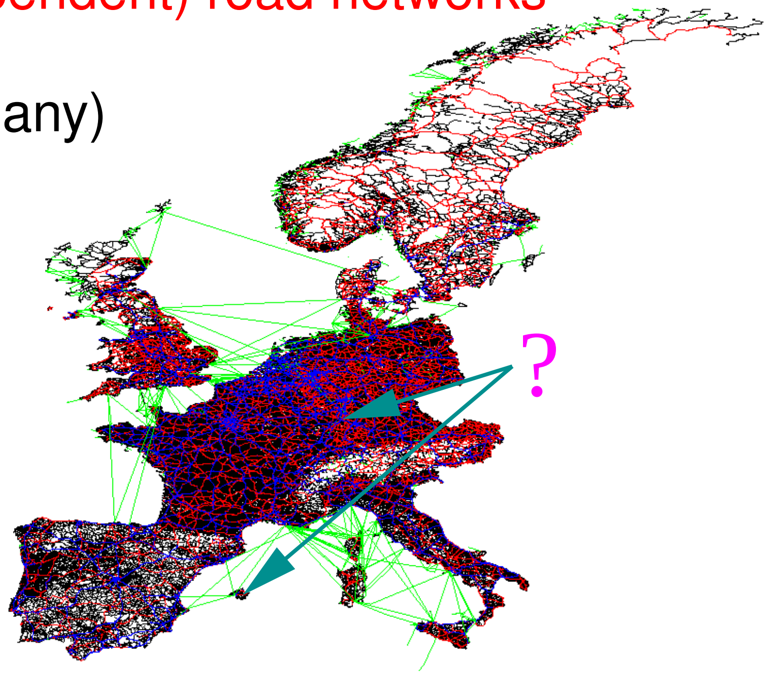
Veit Batz, **Robert Geisberger**, Dennis Luxen,
Peter Sanders, Christian Vetter

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:

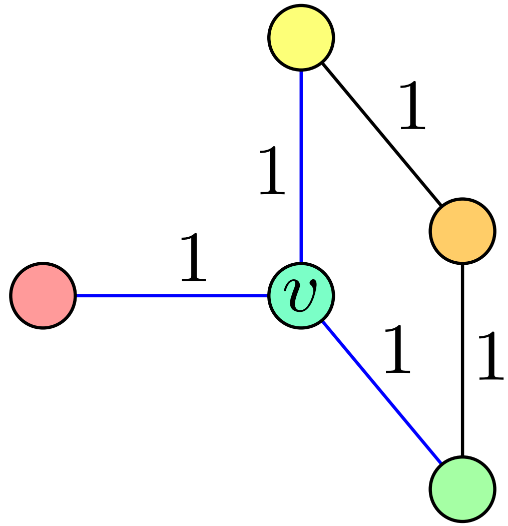
- route planning systems in the internet, car navigation systems,
- ride sharing, traffic simulation, logistics optimisation

Contraction Hierarchies

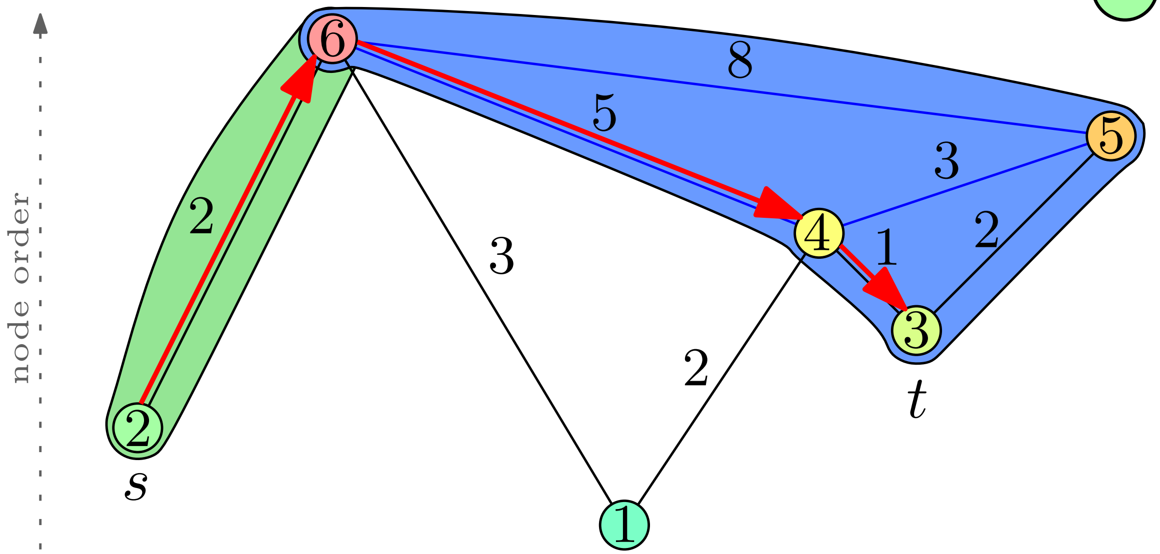
[WEA 08]

- 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 $\langle u, v, w\rangle$ *is a unique shortest path* **then**
          add shortcut (u, w) with weight $w(\langle u, v, w\rangle)$



- query relaxes only edges
  to more “important” nodes
  $\Rightarrow$ valid due to shortcuts



Contraction Hierarchies

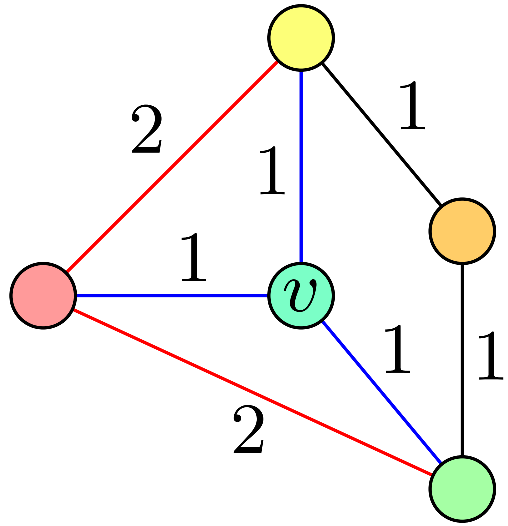
[WEA 08]

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

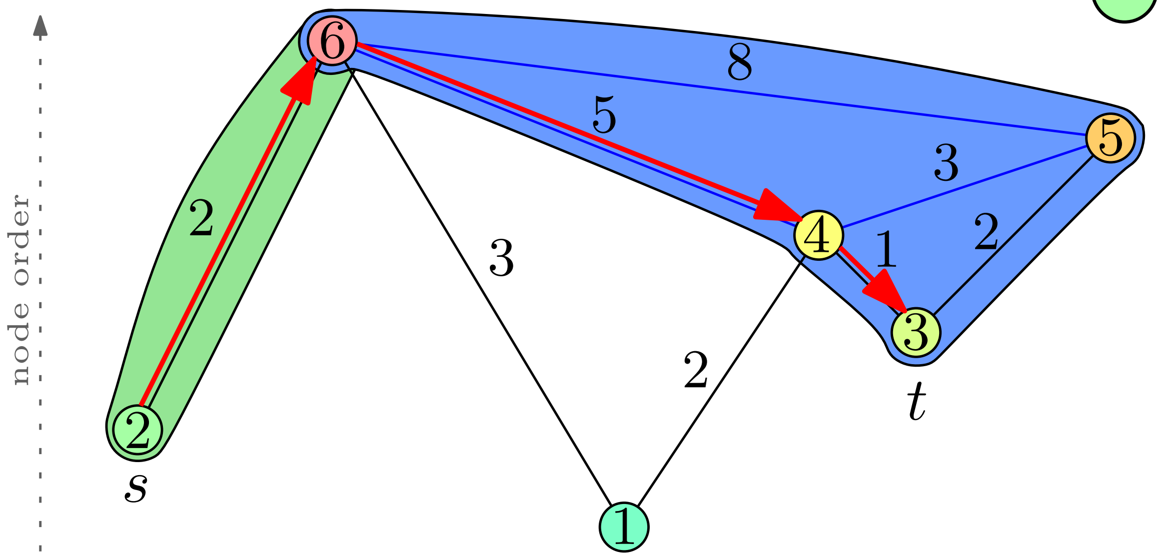
  foreach *pair* (u, v) *and* (v, w) *of edges* **do**

  if $\langle u, v, w \rangle$ *is a unique shortest path* **then**

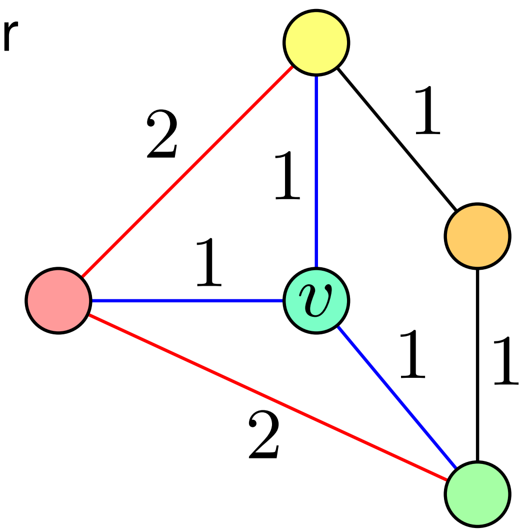
  add shortcut (u, w) with weight $w(\langle u, v, w \rangle)$



- 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:** #shortcuts – #edges incident to v
- **uniformity:** e.g. #deleted neighbors
- ...

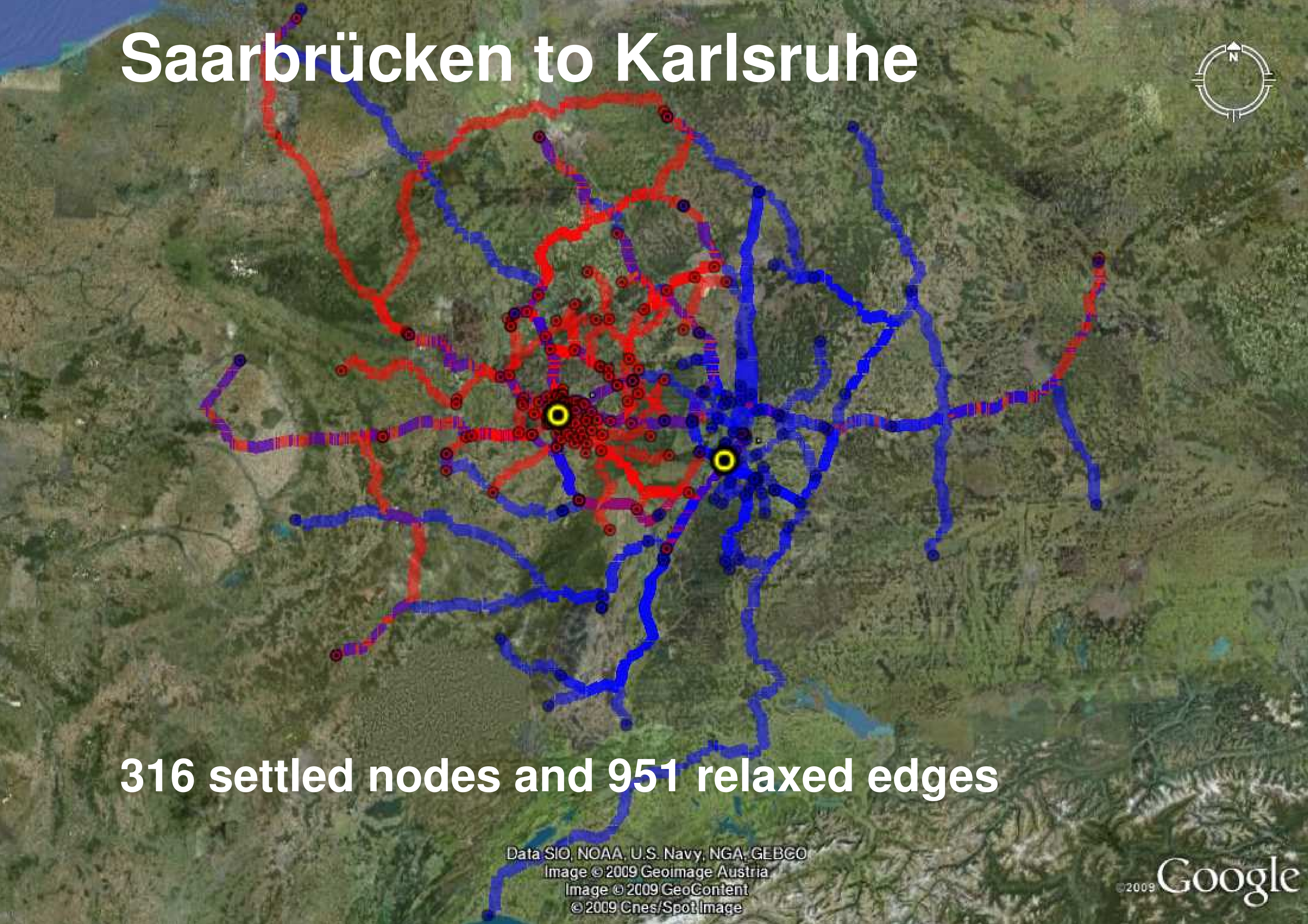
integrated construction and ordering:

1. pop node v on top of the priority queue
2. contract node v
3. update weights of remaining nodes

Saarbrücken to Karlsruhe

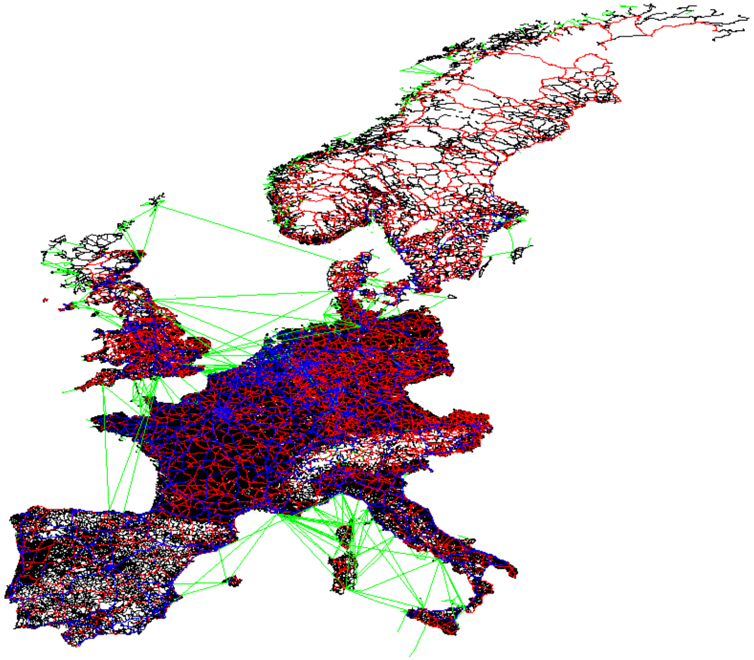
299 edges compressed to 13 shortcuts.

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]

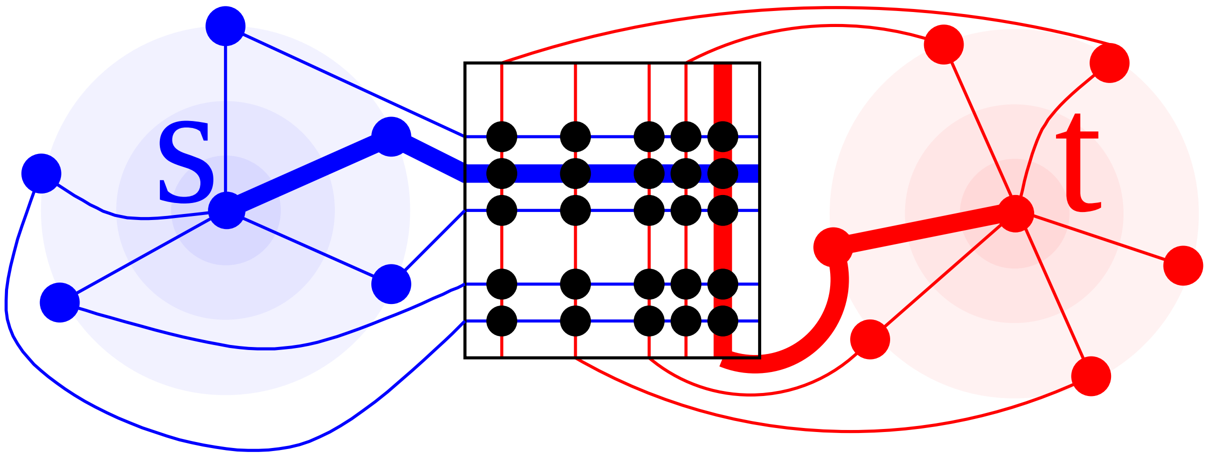
joint work with H. Bast, S. Funke, D. Matijevic

- *very* fast queries

  (down to 1.7 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



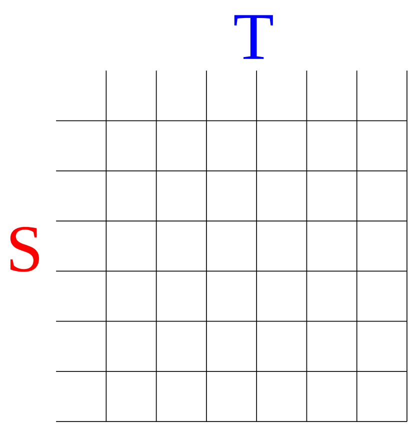
SciAm50 Award

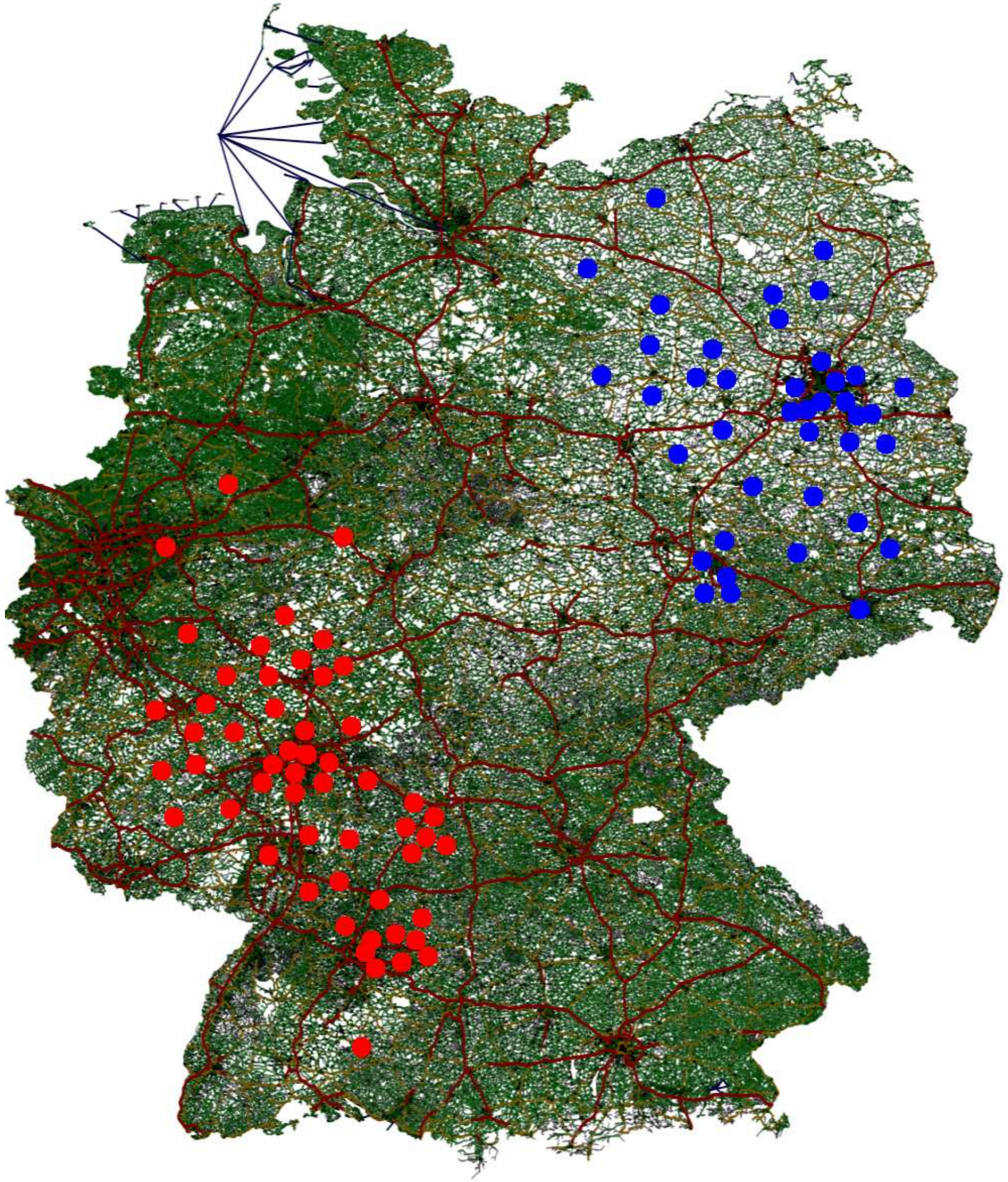
Many-to-Many Shortest Paths

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

[ALENEX 07]

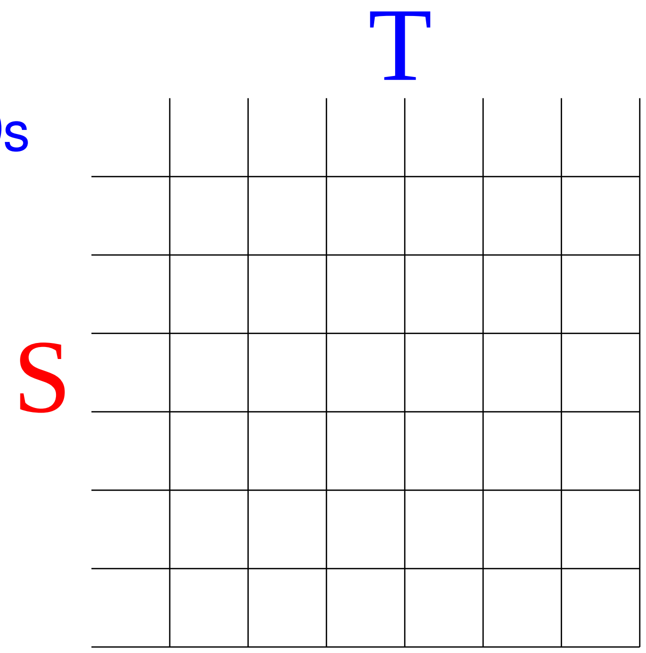
- efficient many-to-many variant of hierarchical bidirectional algorithms
- $10\,000 \times 10\,000$ table in 10s





Many-to-Many Shortest Paths

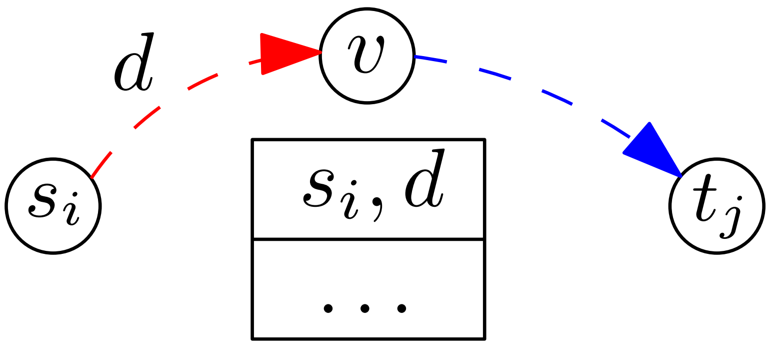
- input: sources $S = \{s_1, \ldots, s_n\}$ and targets $T = \{t_1, \ldots, t_m\}$
- naive algorithm a: perform $\min(n, m)$ Dijkstra one-to-many searches
  $n = m = 10\,000$: $10\,000 \cdot 5$s ≈ 13.9h
- naive algorithm b: perform $n \cdot m$ TNR-queries
  $n = m = 10\,000$: $10\,000 \cdot 10\,000 \cdot 1.7\mu$s $= 170$s
- 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

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



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)

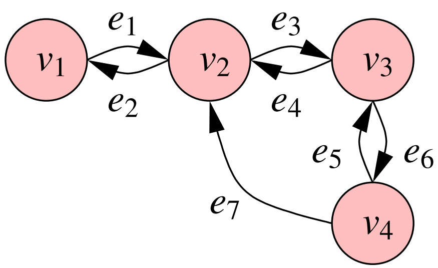
Efficient algorithm:

- adaption of the many-to-many algorithm

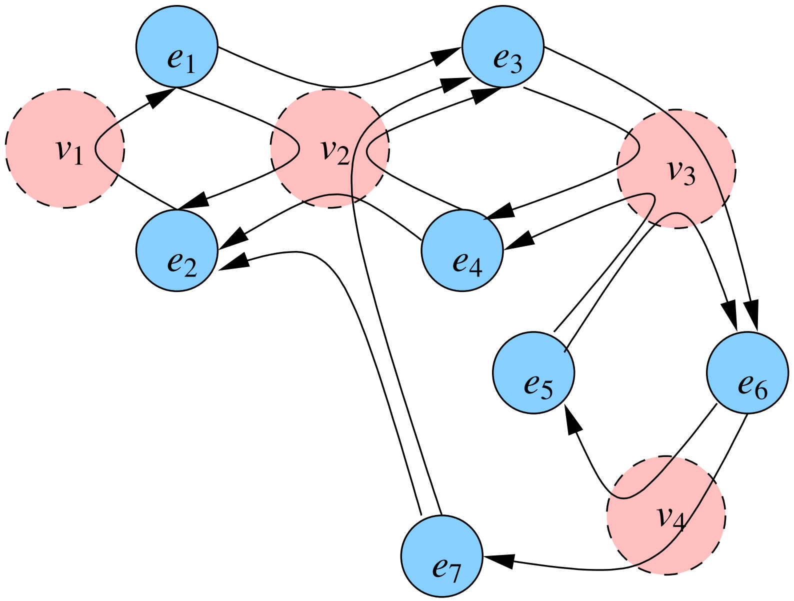
$\Rightarrow$ matches a request to 100 000 offers in $\approx$ 25 ms

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



node-based graph

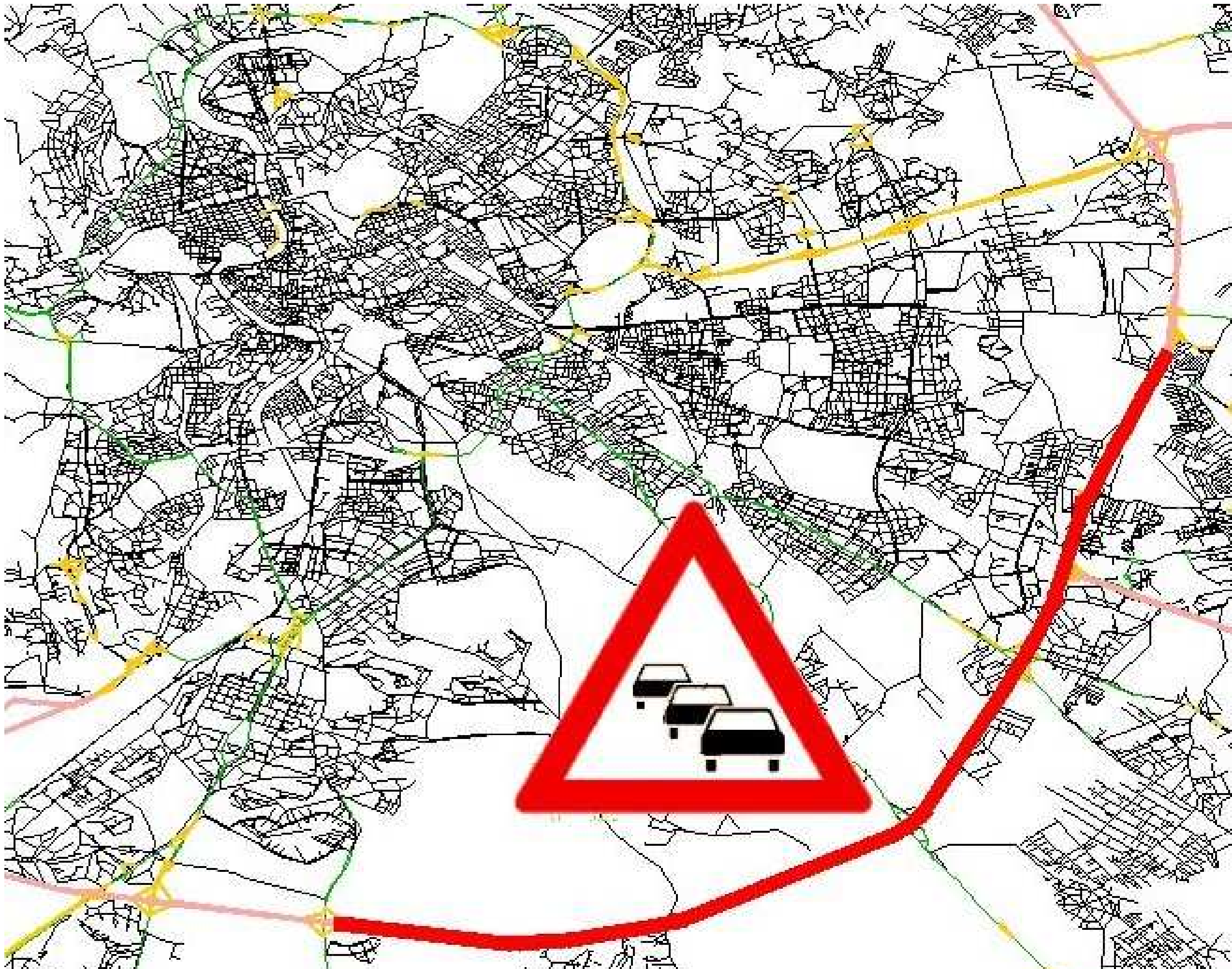


edge-based graph

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

[ESA 08]

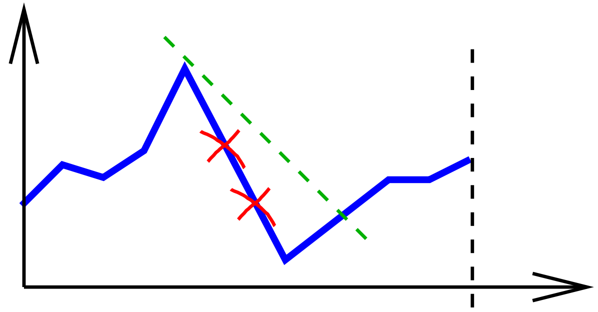
- preprocess data on a personal computer
- highly compressed blocked graph representation — 8 bytes/node
- compact route reconstruction data structure — + 8 bytes/node

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

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
- query (s, t, τ_0) — start, target, departure time
- looking for:

  a fastest route from s to t depending on τ_0

  $\Rightarrow$ Earliest Arrival Problem

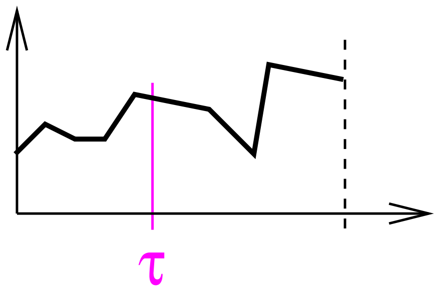
Travel Time Functions

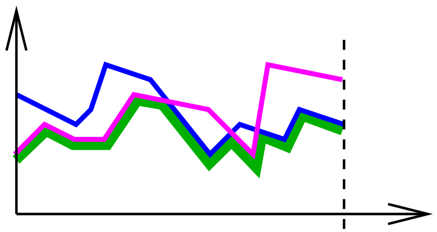
we need three operations

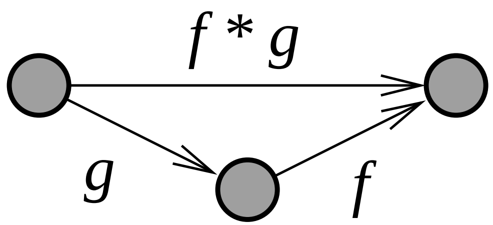
- evaluation: $f(\tau)$ — "$O(1)$" time
- merging: $\min(f,g)$ — $O(|f|+|g|)$ time
- chaining: $f*g$ (f "after" g) — $O(|f|+|g|)$ time

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

$\Rightarrow$ increase of complexity



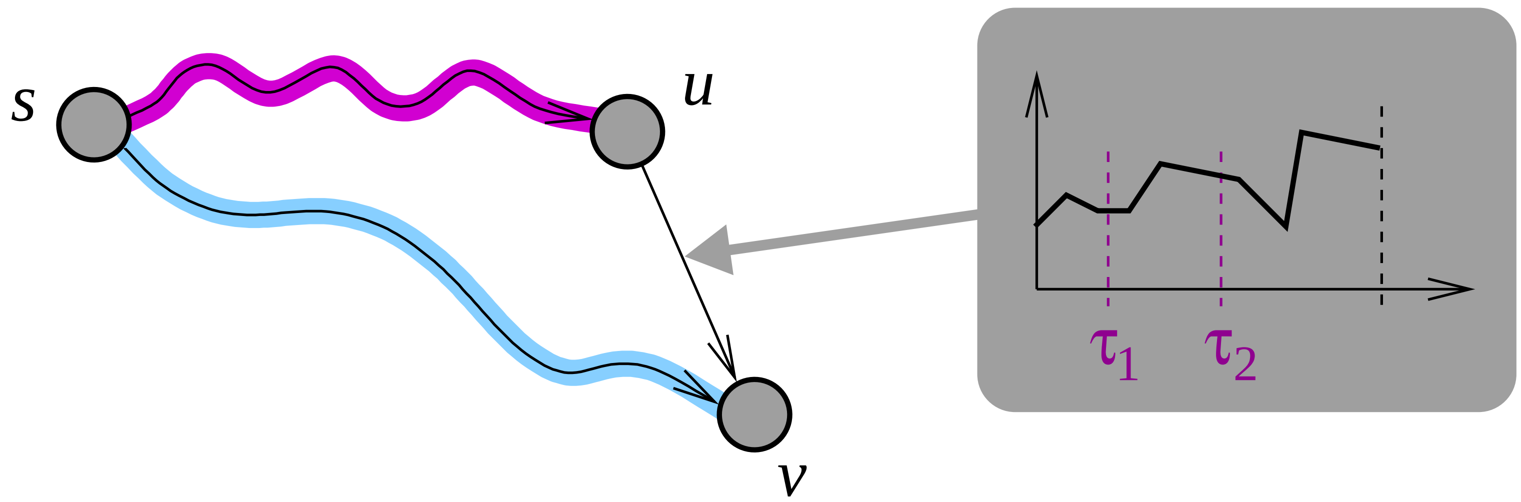




Time-Dependent Dijkstra

Only one difference to standard Dijkstra:

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

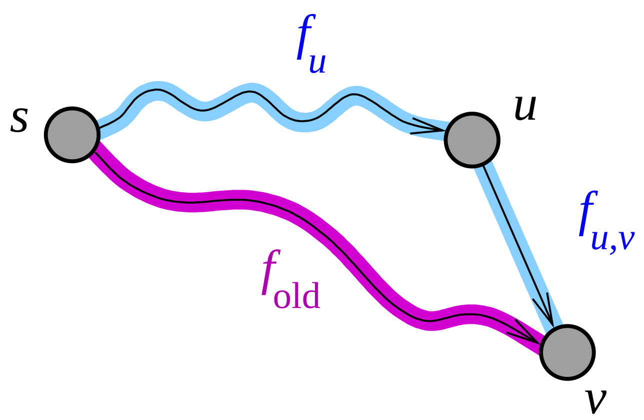


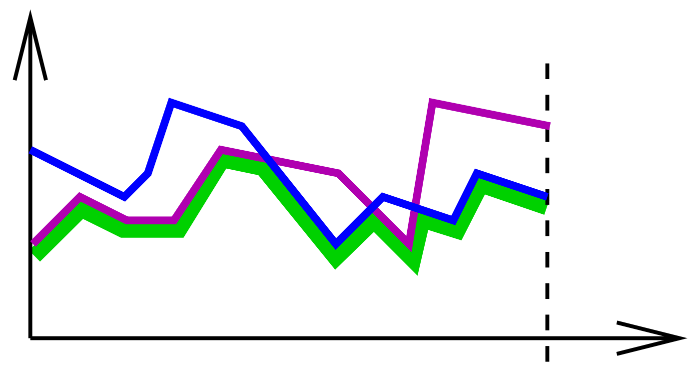
Profile Search

Modified Dijkstra:

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

$\Rightarrow$ A **label correcting** algorithm





Min-Max-Label Search

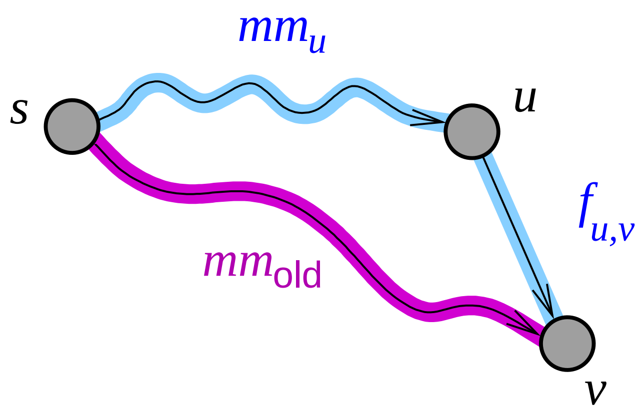
Approximate version of profile search:

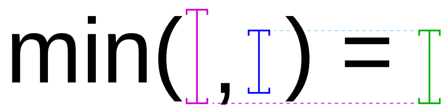
- Computes **upper** and **lower bounds**
- Node labels are pairs $mm_u := (\min f_u, \max f_u)$
- Edge relaxation:

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

- PQ key is the lower bound

$\Rightarrow$ A **label correcting** algorithm





Time-Dependent Contraction Hierarchies

two major challenges:

1. contraction during precomputation

   witnesses can be found by profile search

   ...which is straightforward

   ...but incredibly slow!

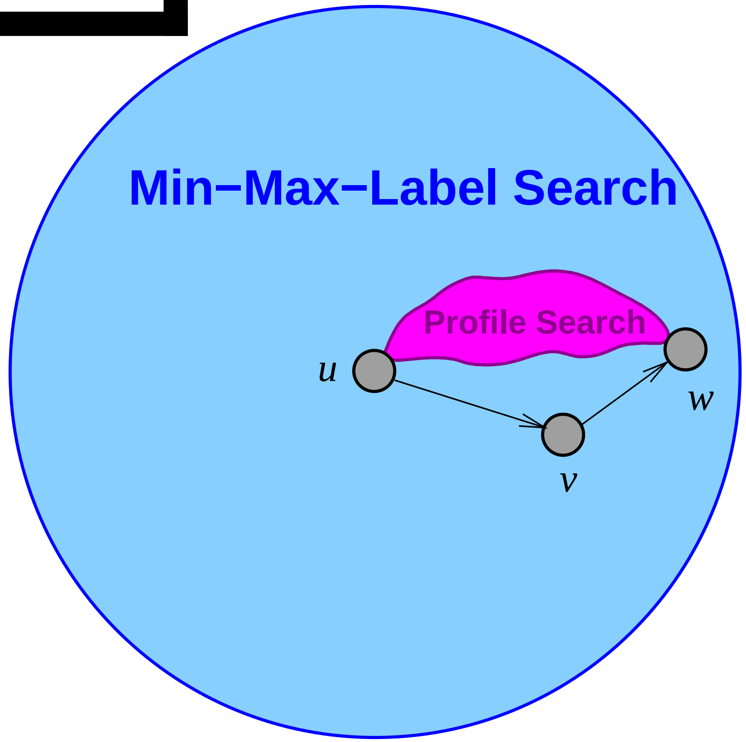
   ⇒ do something more intelligent!

2. bidirectional search

   ⇒ problem: arrival time not known

   ...but can be solved

Restricted Profile Search

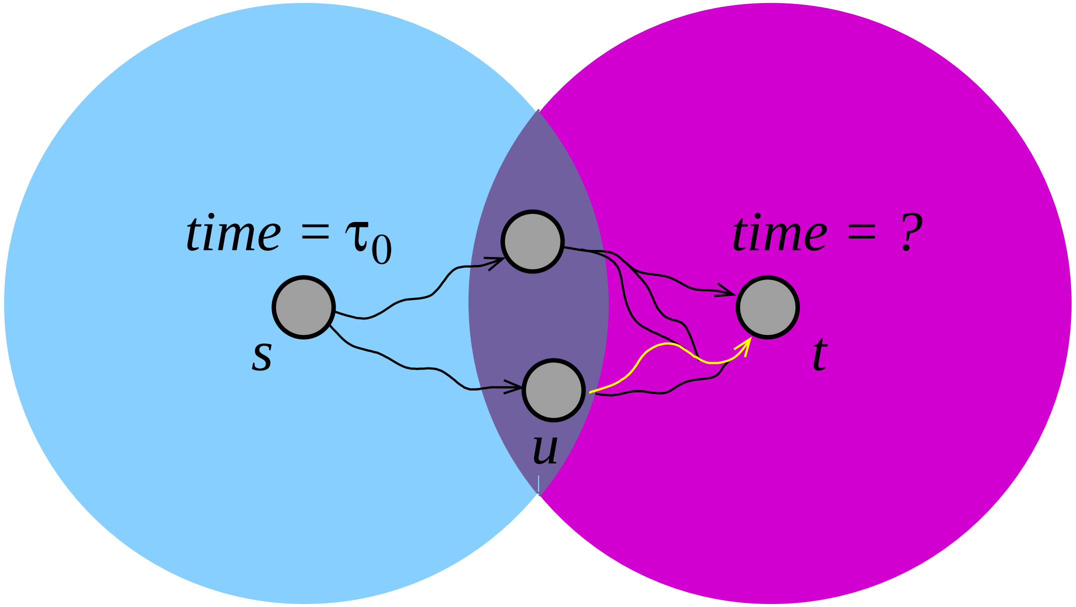


phase 1: restricts the search space

- 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

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

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

$\Rightarrow$ 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 $\rightsquigarrow$ accurate current situation

Past data $+$ traffic model $+$ real time simulation

$\rightsquigarrow$ Nash euqilibrium predicting near future

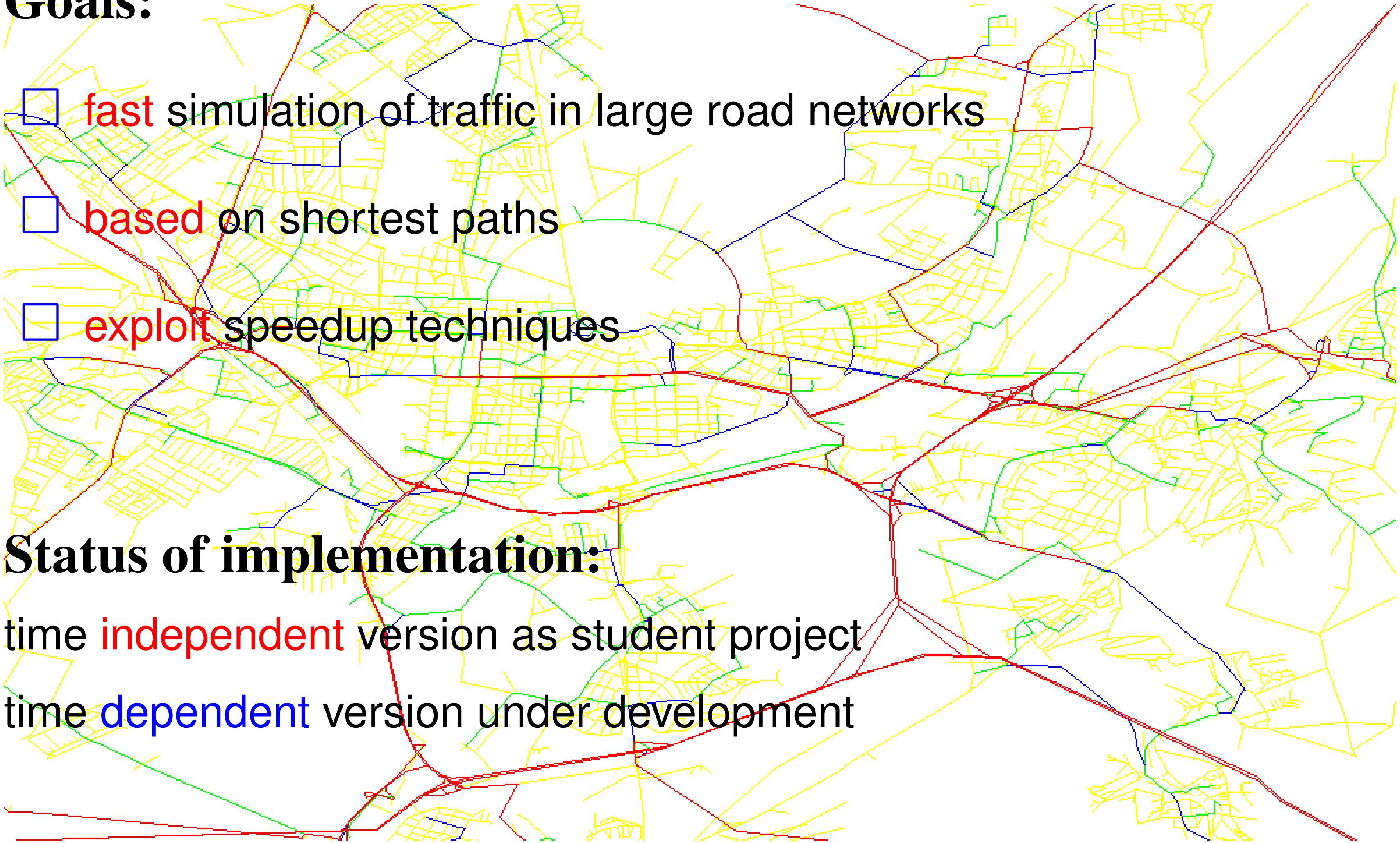
time dependent routing in Nashequilibrium

$\rightsquigarrow$ 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):

- Permute 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:

- modifiy 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 timedependent CH)
  - linear combination of more: $a_1 x_1 + \cdots + a_n x_n$ with $a_i \in [l_i, u_i]$
  - Pareto-optimal (may add too many shortcuts)

$\Rightarrow$ too many shortcuts needed when done naive

Challenges

- 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

$\Rightarrow$ new algorithmic ideas necessary