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)

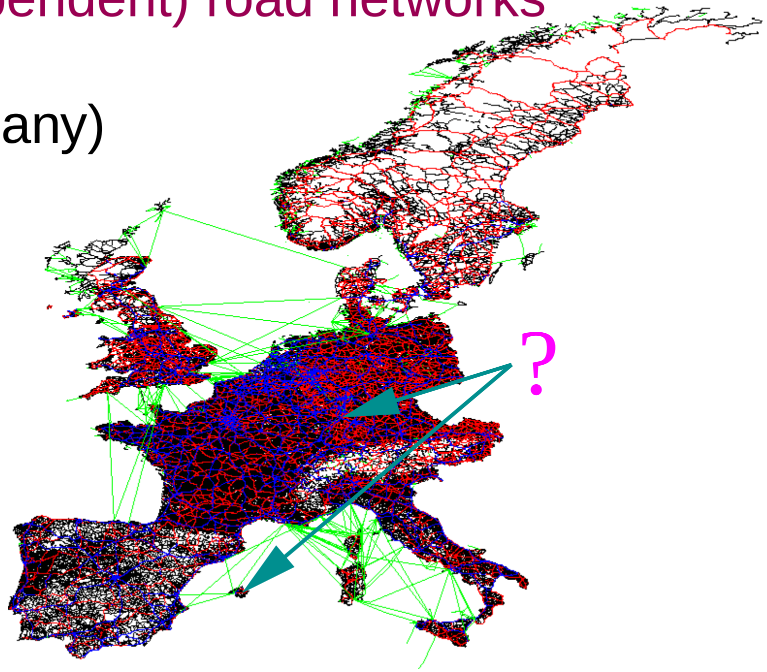
Karlsruhe Institute of Technology

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:

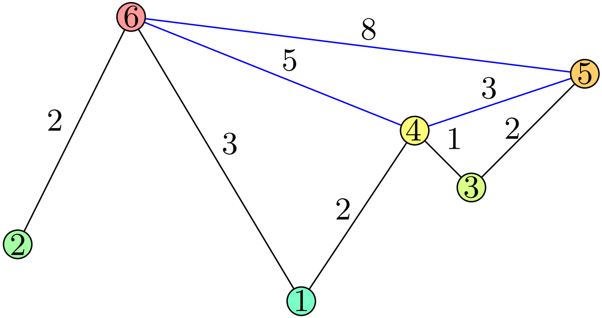
- ☐ plain static routing (very fast)
- ☐ distance tables (even faster)
- ☐ turn penalties
- ☐ mobile implementation
- ☐ time dependent edge weights
- ☐ flexible objective functions
- ☐ traffic jams

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
  $\Rightarrow$ local and cache-efficient operation

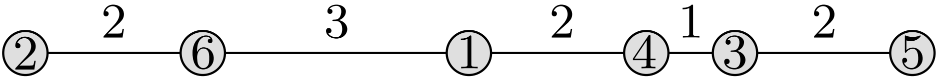
in more detail:

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

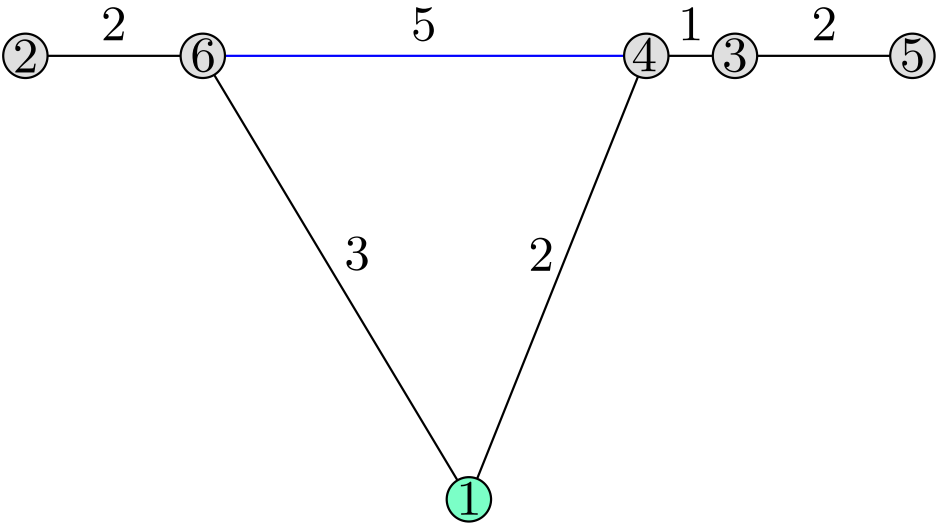
  foreach *pair* (u, v) *and* (v, w) *of edges* **do**
      if $\langle u, v, w \rangle$ *is a unique shortest path* **then**
          add shortcut (u, w) with weight $w(\langle u, v, w \rangle)$

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

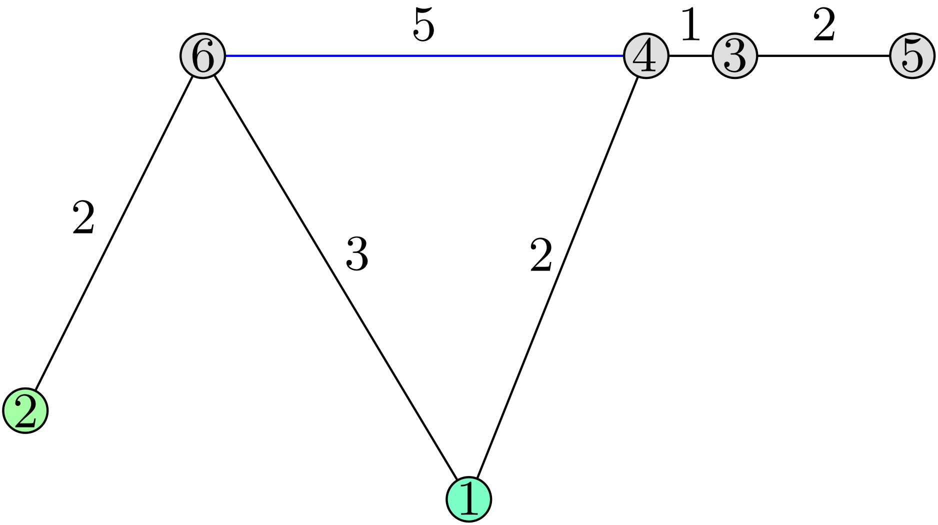
Example: Construction



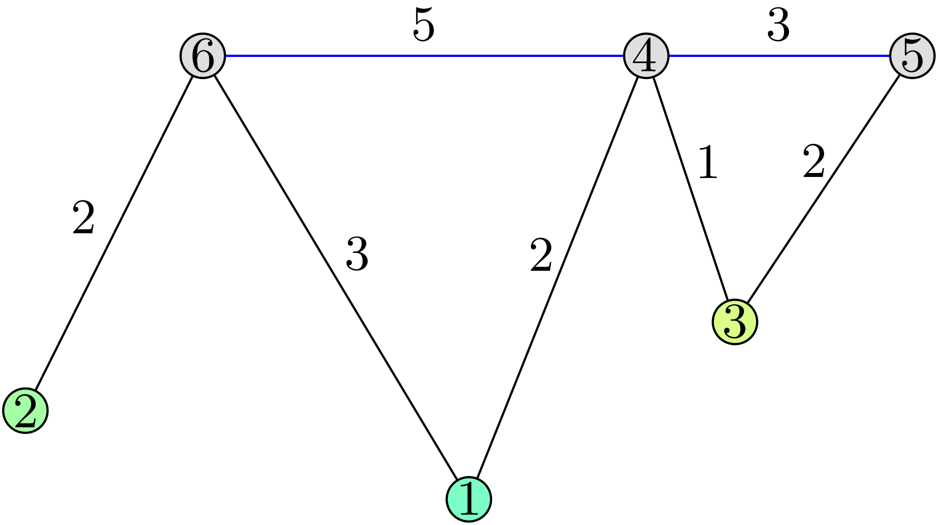
Example: Construction



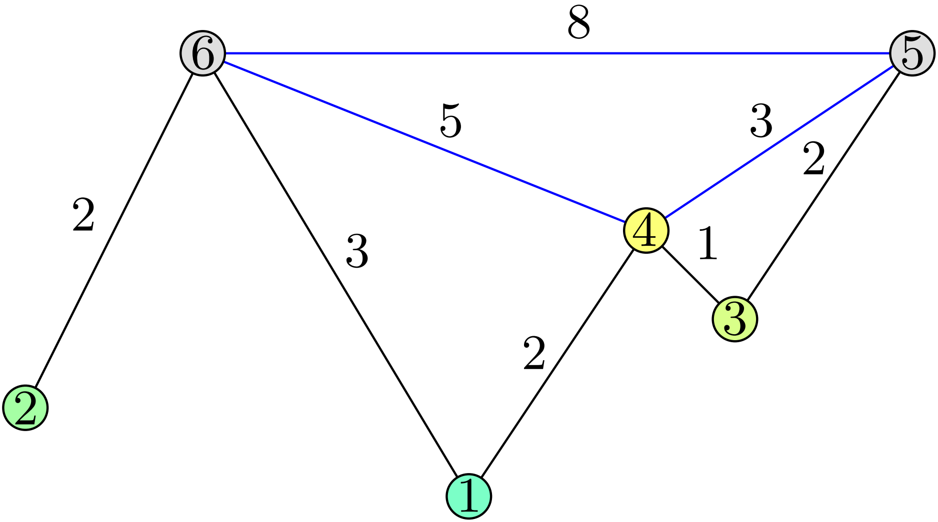
Example: Construction



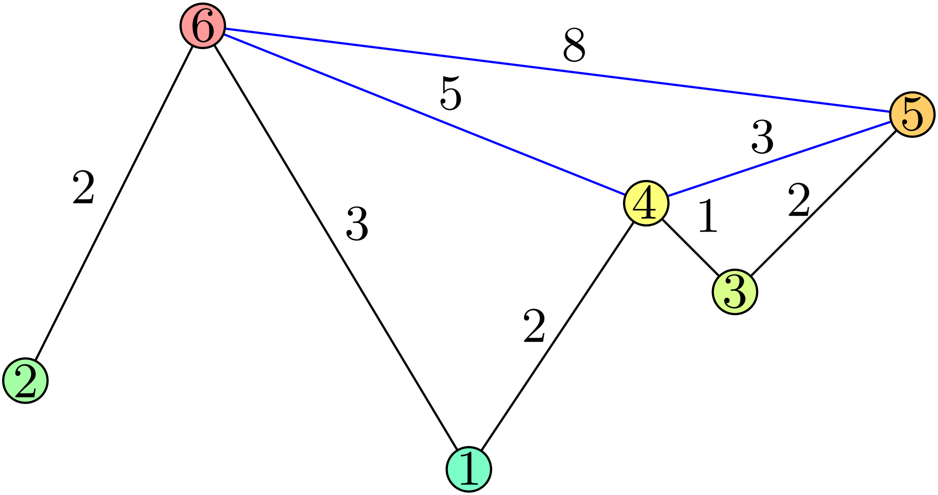
Example: Construction



Example: Construction



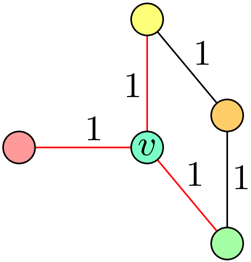
Example: Construction



Construction

to identify necessary shortcuts

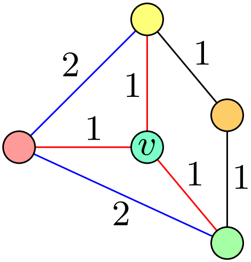
- local searches from all nodes u with incoming edge (u, v)
- ignore node v at search
- add shortcut (u, w) iff found distance $d(u, w) > w(u, v) + w(v, w)$



Construction

to identify necessary shortcuts

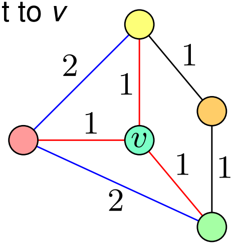
- local searches from all nodes u with incoming edge (u, v)
- ignore node v at search
- add shortcut (u, w) iff found distance $d(u, w) > w(u, v) + w(v, w)$



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
- ...



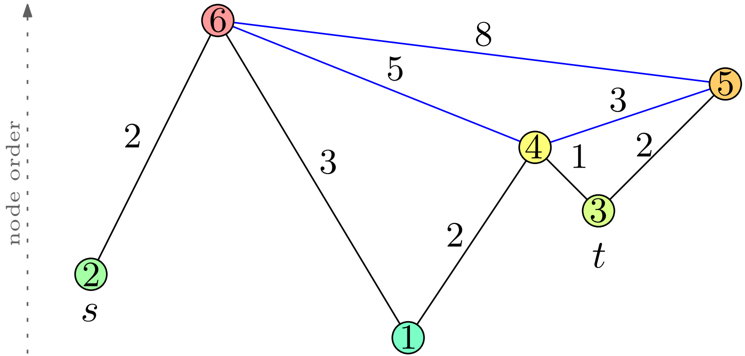
$2 - 3 = -1$

integrated construction and ordering:

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

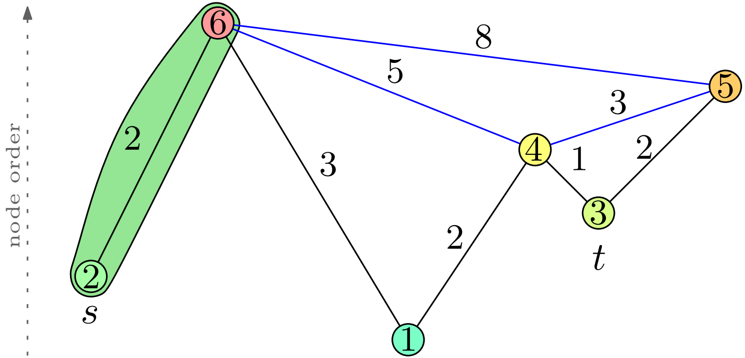
Query

- 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 $G_\downarrow$



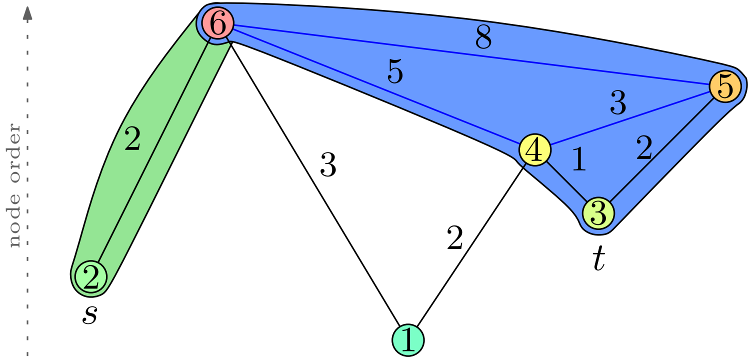
Query

- 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 $G_\downarrow$



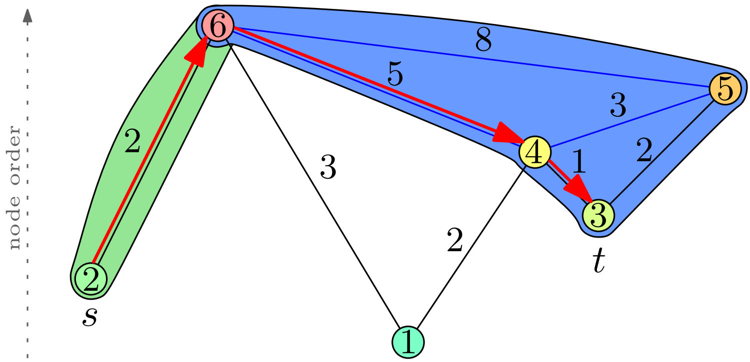
Query

- modified bidirectional Dijkstra algorithm
- upward graph $G_\uparrow := (V, E_\uparrow)$ 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 $G_\downarrow$



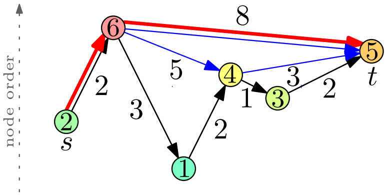
Query

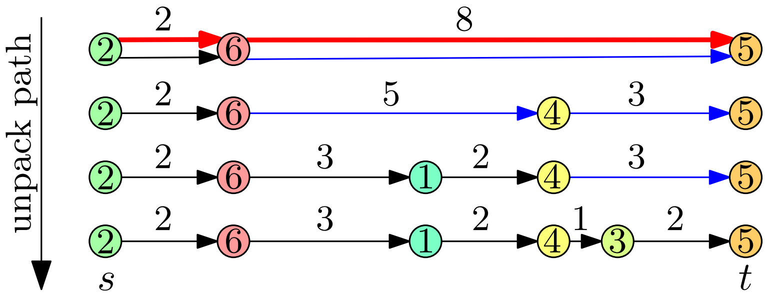
- modified bidirectional Dijkstra algorithm
- upward graph $G_\uparrow := (V, E_\uparrow)$ with $E_\uparrow := \{(u,v) \in E : u < v\}$
  downward graph $G_\downarrow := (V, E_\downarrow)$ with $E_\downarrow := \{(u,v) \in E : u > v\}$
- forward search in $G_\uparrow$ and backward search in $G_\downarrow$



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

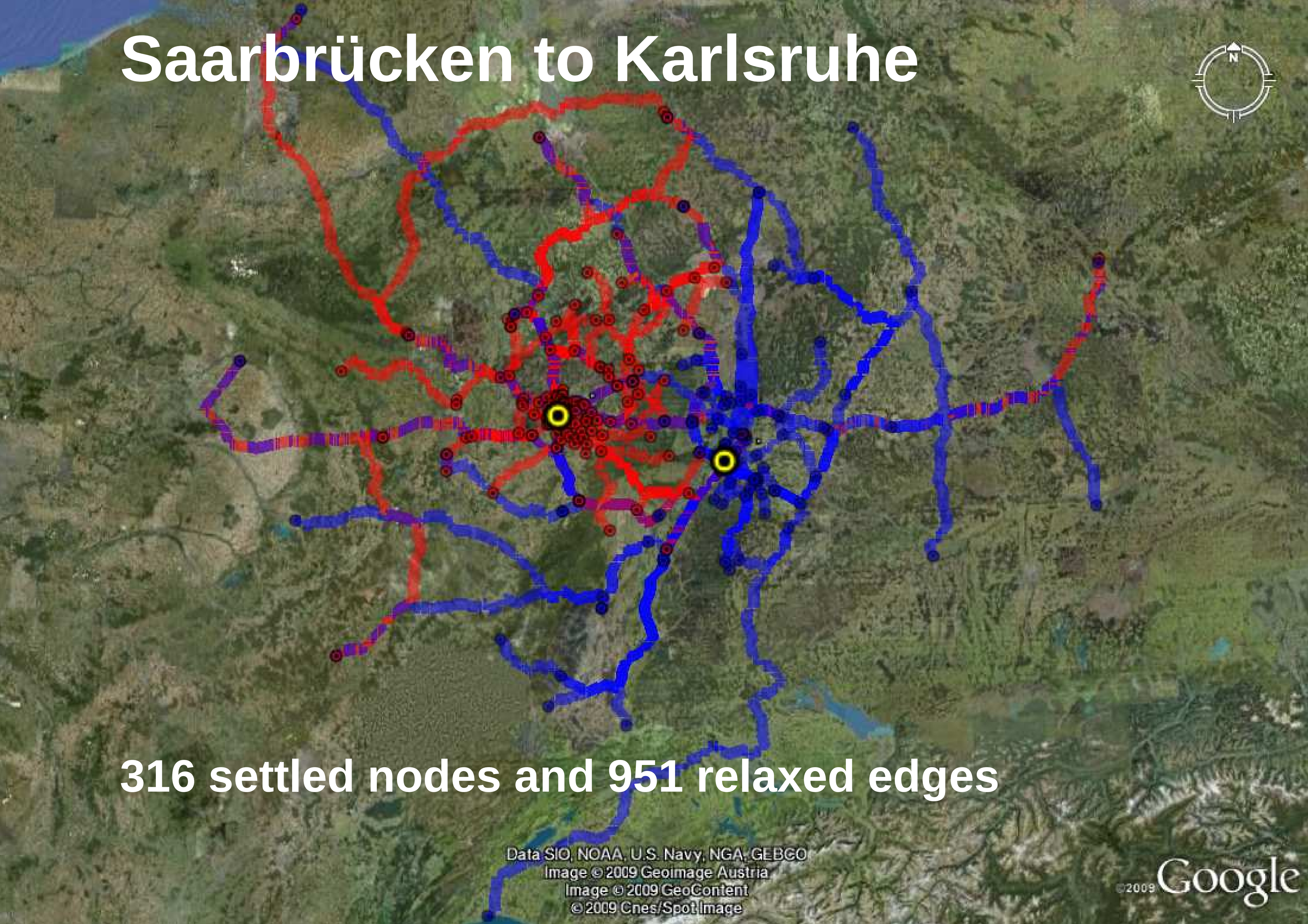




Saarbrücken to Karlsruhe

299 edges compressed to 13 shortcuts.

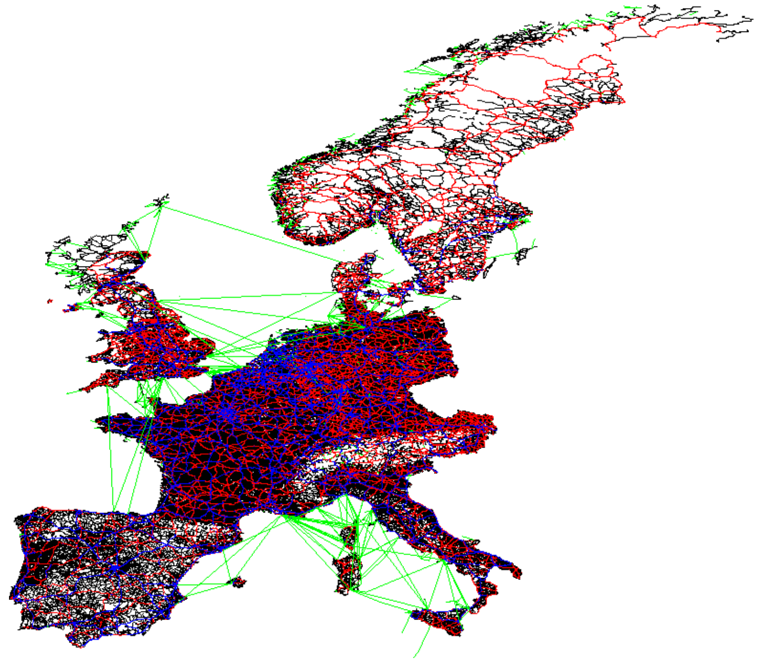
Saarbrücken to Karlsruhe



316 settled nodes and 951 relaxed edges

Contraction Hierarchies

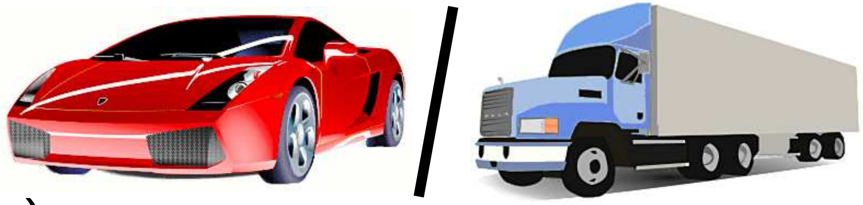
- foundation for our other methods
- conceptually very simple
- handles dynamic scenarios



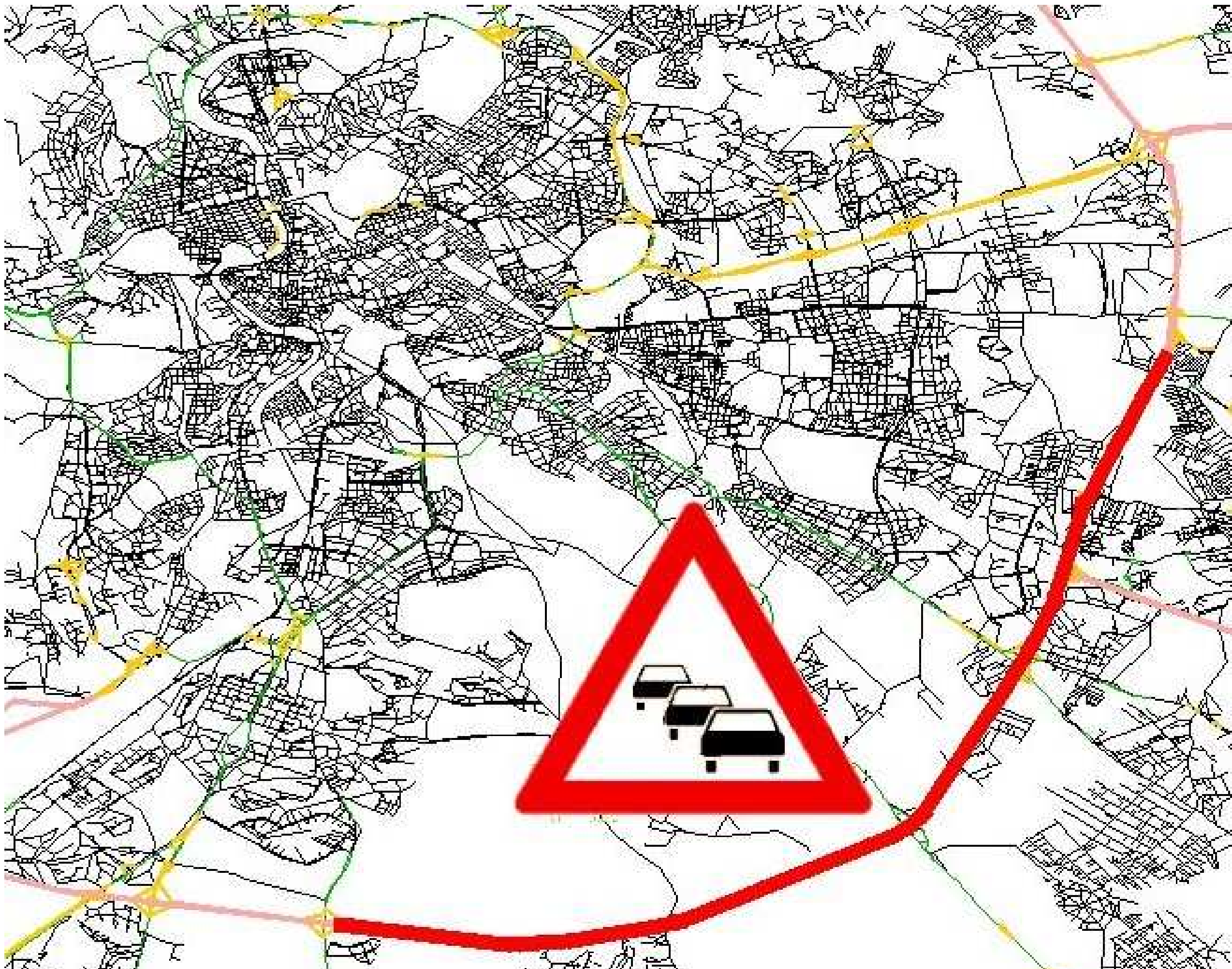
Static scenario:

- 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

[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

Even Faster – Transit-Node Routing

[DIMACS Challenge 06, ALENEX 07, Science 07]

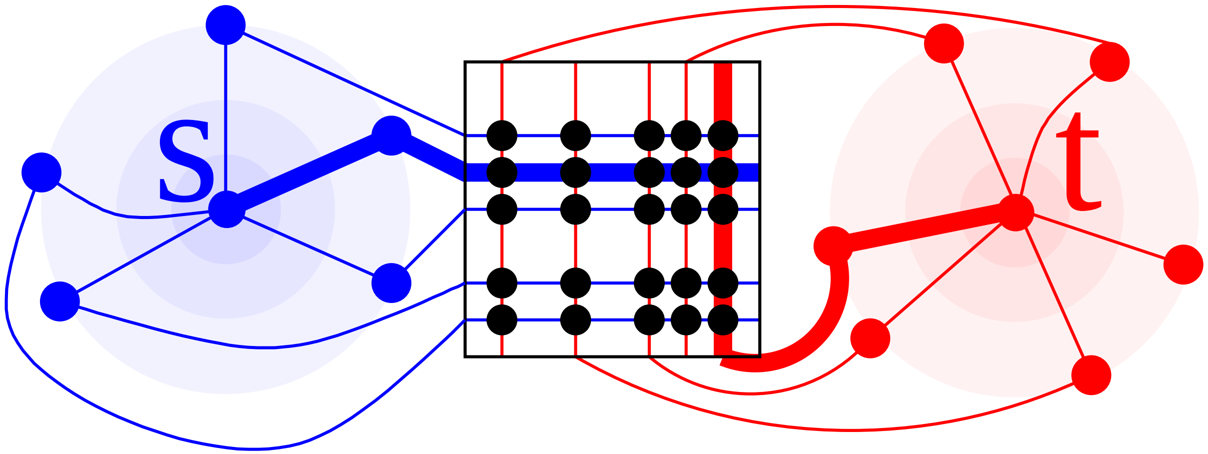
joint work with H. Bast, S. Funke, D. Matijevic

- *very* fast queries

  (down to 1.7 μs, 3 000 000 times faster than DIJKSTRA)

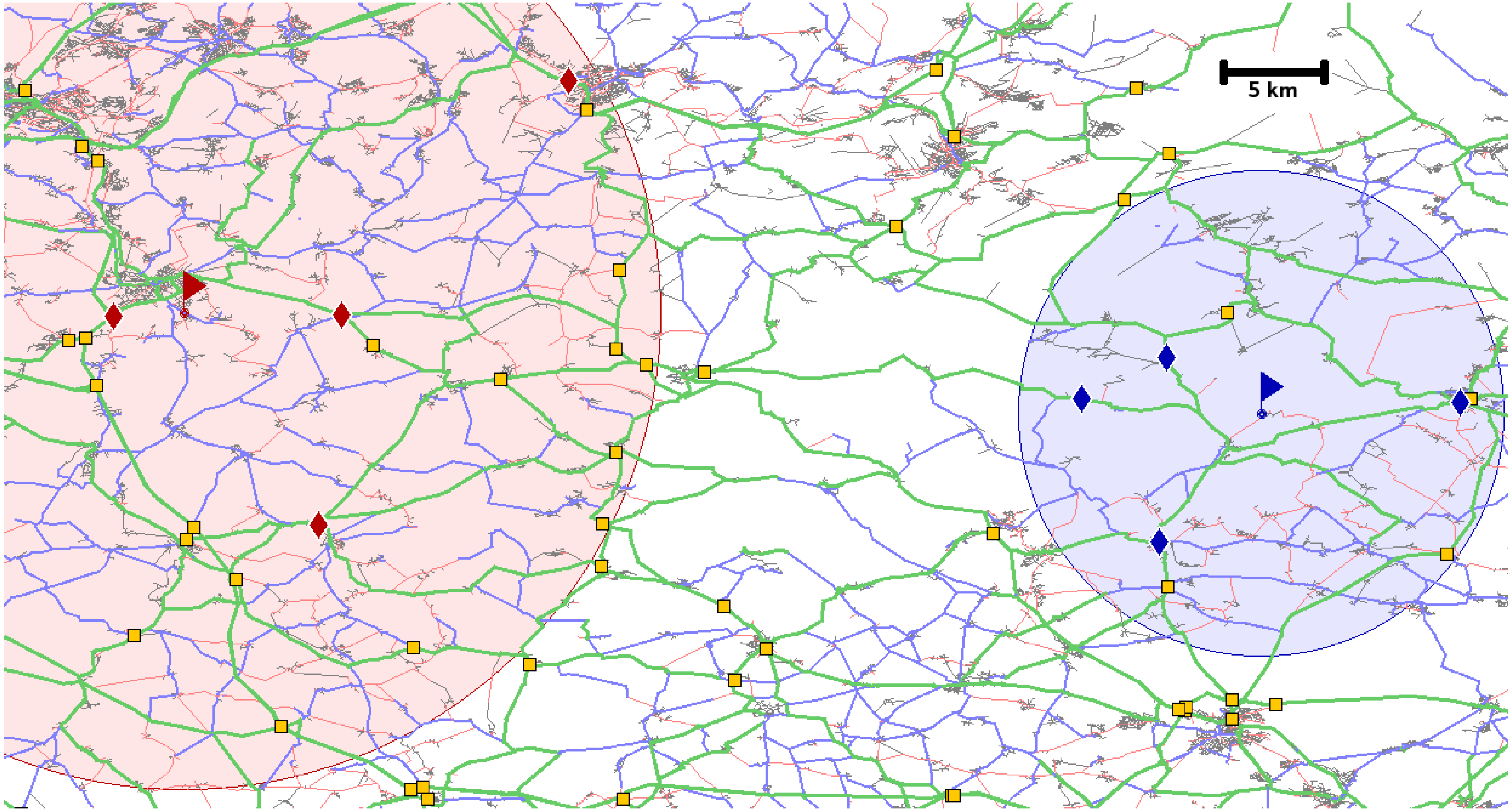
- winner of the 9th DIMACS Implementation Challenge

- more preprocessing time (2:37 h) and space (263 bytes/node) needed



SciAm50 Award

Example

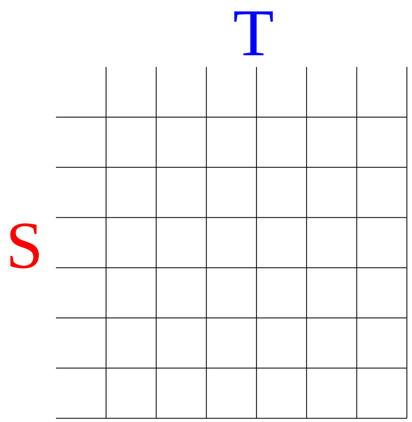


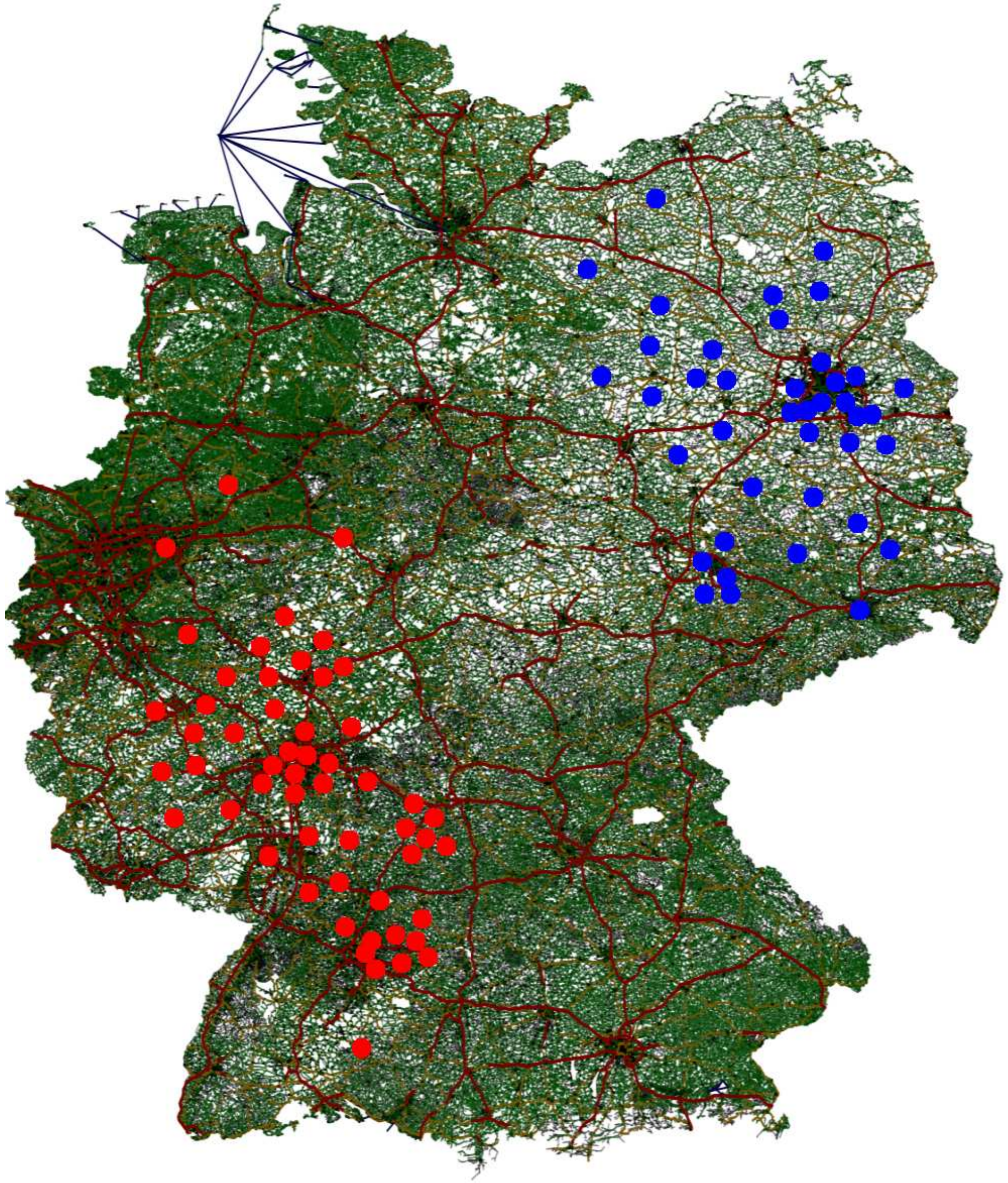
Many-to-Many Shortest Paths

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

[ALENEX 07]

- efficient many-to-many variant of hierarchical bidirectional algorithms
- $10\,000 \times 10\,000$ 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)

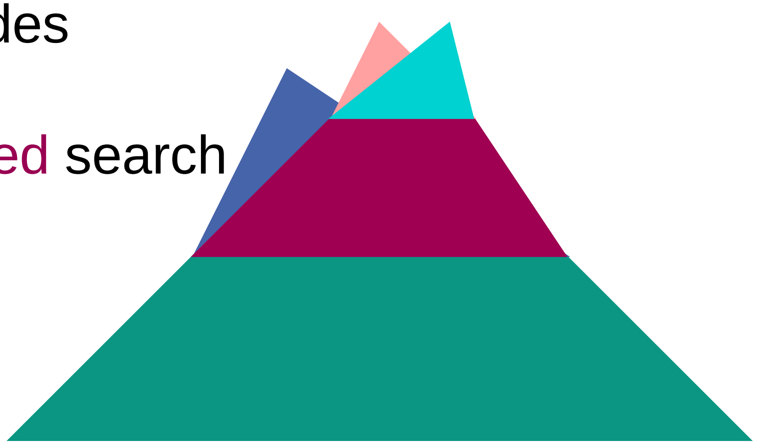
Flexible Objective Functions

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

Cost function: arbitrary linear combination

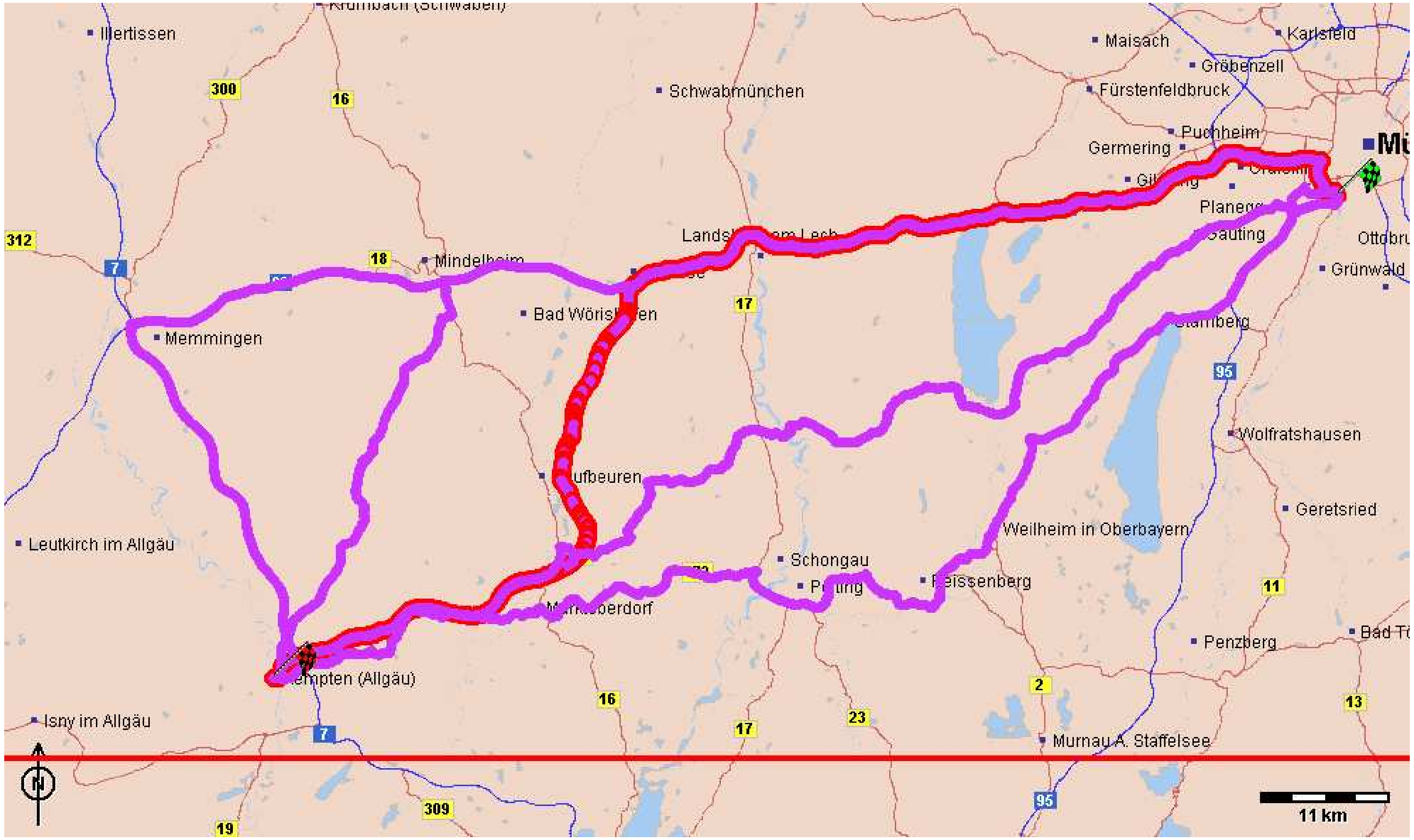
Ideas:

- CHs with valid parameter ranges at each shortcut
- Different node orderings for important nodes
- combine with landmark based goal directed search

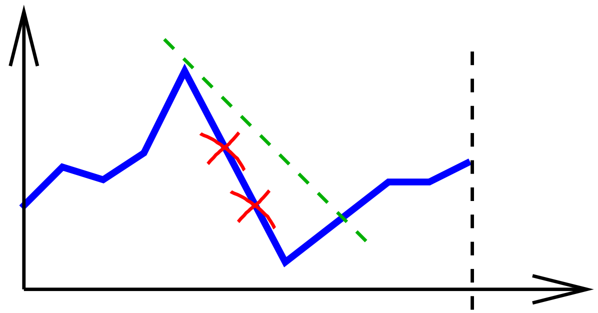


Alternative Routes DA Jonathan Dees, BMW

- 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, τ_0)
  $\longrightarrow$ a fastest s–t-route departing at τ_0
- Profile Query:$(s, t, [\tau, \tau'])$
  $\longrightarrow$ fastest travel times departing between τ and tau'.

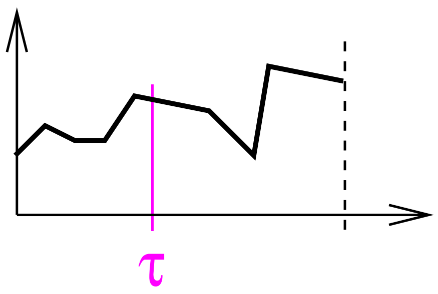
Travel Time Functions

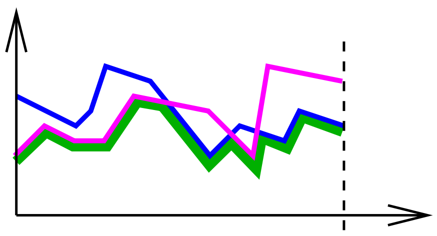
we need three operations

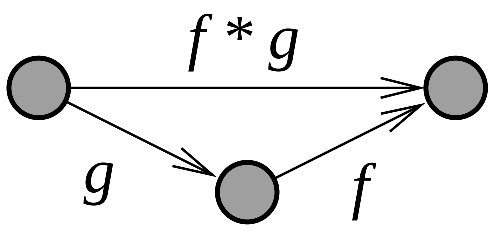
- evaluation: $f(\tau)$ — "$\mathcal{O}(1)$" time
- merging: $\min(f, g)$ — $\mathcal{O}(|f| + |g|)$ time
- chaining: $f * g$ (f "after" g) — $\mathcal{O}(|f| + |g|)$ time

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

$\Rightarrow$ increase of complexity



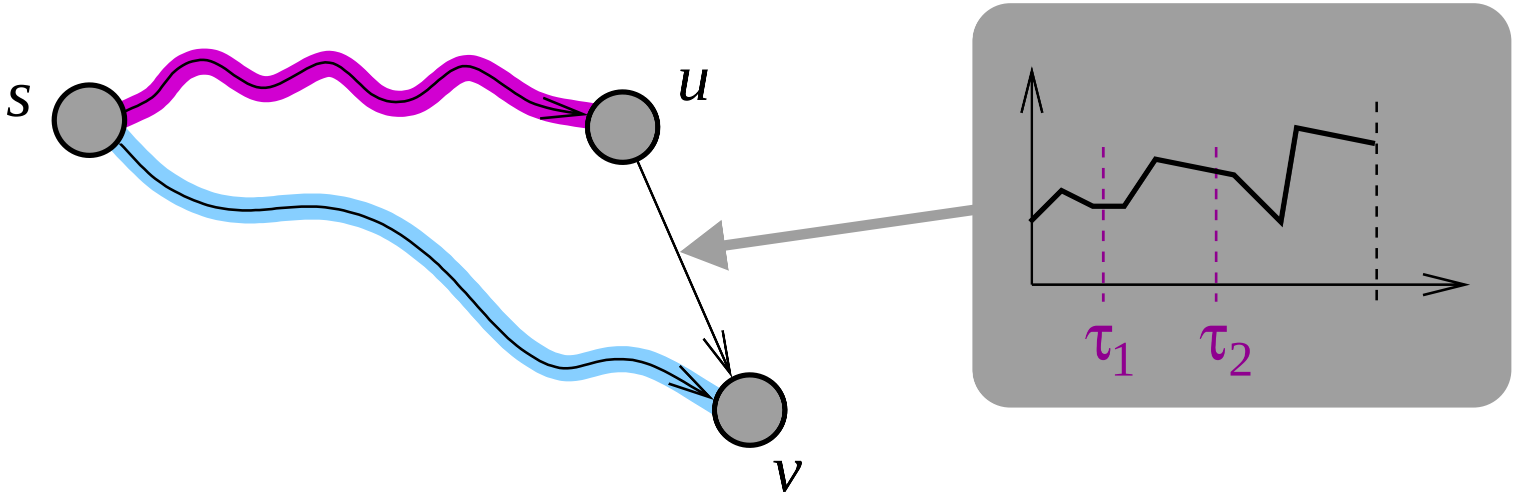




Time-Dependent Dijkstra

Only one difference to standard Dijkstra:

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

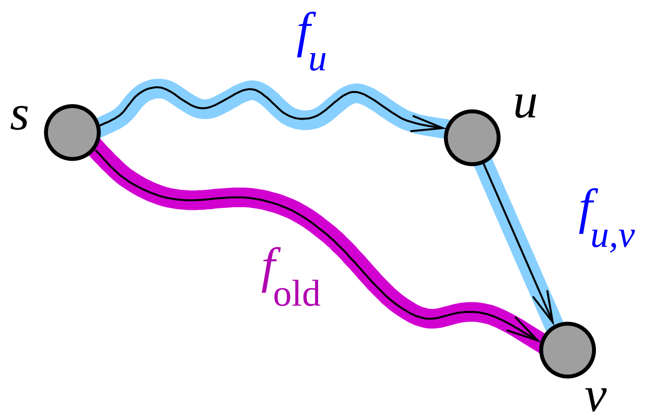


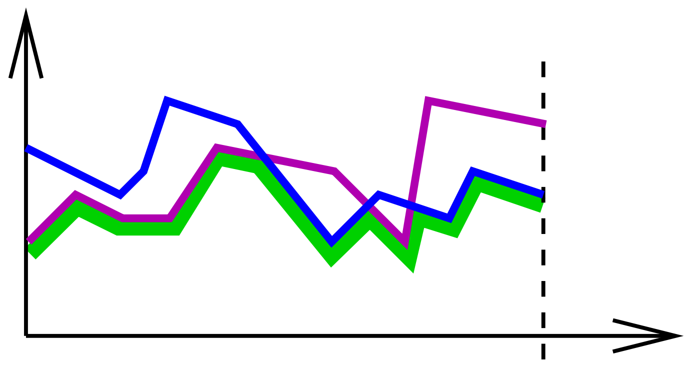
Profile Search

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

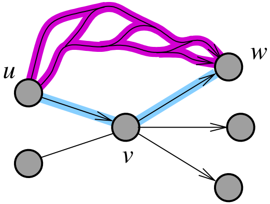


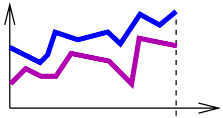


Avoiding Shortcuts

in the time-dependent case

How to know that a shortcut is not needed?





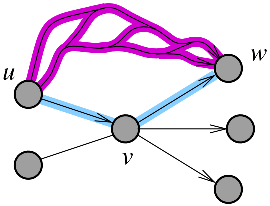
$\Rightarrow$ No shortest path leeds ever over $\langle u, v, w \rangle$

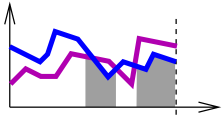
$\Rightarrow$ **Don't insert a shortcut!**

Avoiding Shortcuts

in the time-dependent case

How to know that a shortcut is not needed?





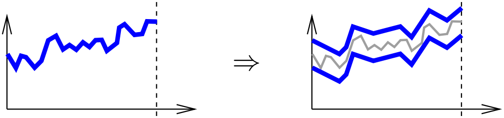
$\Rightarrow$ If a shortest path leeds over $\langle u, v, w\rangle$ for at least one departure time

$\Rightarrow$ **Insert a shortcut!**

ATCH = 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

graph	method	ε [%]	space [B/n] ABS	space [B/n] OVH	query [ms]	query SPD	error [%] MAX	error [%] AVG
Earliest Arrival Query								
Germany	TCH	–	994	899	0.72	1 440	0.00	0.00
	ATCH	1	239	144	1.27	816	0.00	0.00
	ATCH	∞	118	23	1.45	714	0.00	0.00
Europe	TCH	–	589	513	1.89	1 807	0.00	0.00
	ATCH	1	207	131	2.47	1 396	0.00	0.00
	ATCH	∞	99	23	15.43	221	0.00	0.00

Profile Queries on ATCHs with Corridor Contraction

Performance

graph	method	ε [%]	space [B/n] ABS	space [B/n] OVH	query [ms]	error [%] MAX	error [%] AVG
Earliest Arrival Query							
Germany	TCH	–	994	899	1 112.04	0.00	0.00
	ATCH	1	239	144	39.23	0.00	0.00
	ATCH	∞	118	23	81.07	0.00	0.00
Europe	TCH	–	589	513	4 308.35	0.00	0.00
	ATCH	1	207	131	468.43	0.00	0.00
	ATCH	∞	99	23	–	–	–

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]
  $\sim$ transit node routing
- Station-Based CHs [R. Geisberger]
  $\rightsquigarrow$ 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

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

"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

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