Route Planning in Road Networks

– simple, flexible, efficient –

Dominik Schultes **Peter Sanders**

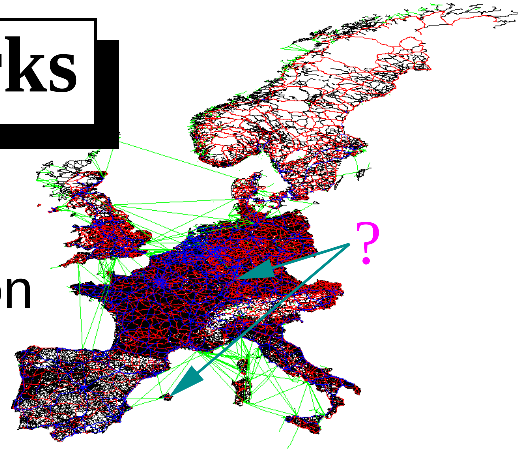
Institut für Theoretische Informatik – Algorithmik II

Universität Karlsruhe (TH)

`http://algo2.iti.uka.de/schultes/hwy/`

Bertinoro, October 1, 2007

Static Route Planning in Road Networks



Task: determine quickest route from source to target location

Problem: for large networks, simple algorithms are too slow

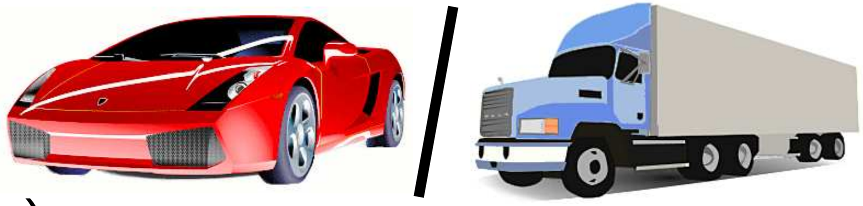
Assumption: road network does not change

Conclusion: use preprocessed data to accelerate source-target-queries

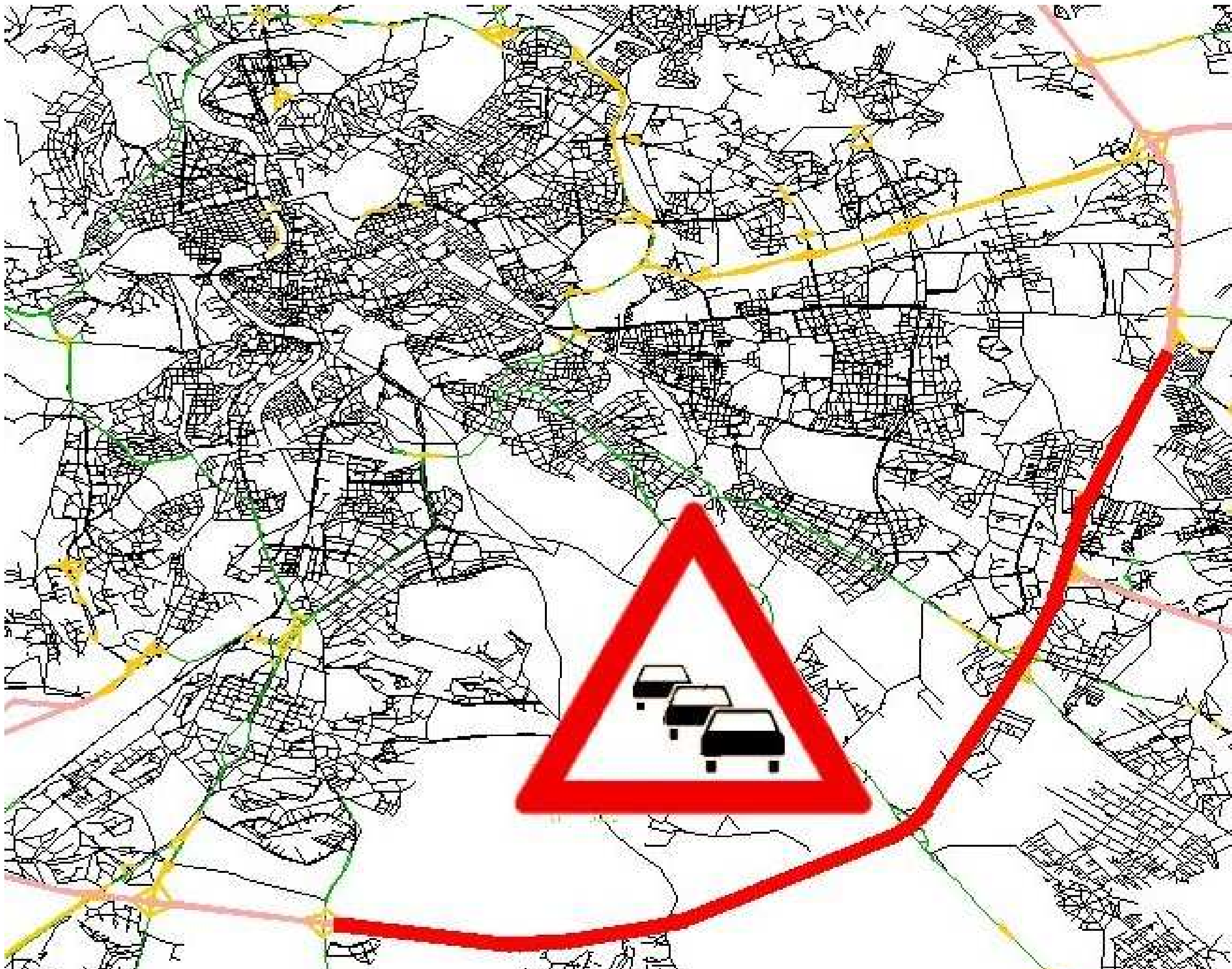
(research focus during the last years [$\longrightarrow$ e.g., 9th DIMACS Challenge])

$\rightsquigarrow$ correctness relies on the above assumption

Dynamic Scenarios



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



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

Constancy of Structure

Weaker Assumption:

- structure of road network does not change

  (no new roads, road removal = set weight to ∞)

  $\leadsto$ not a significant restriction

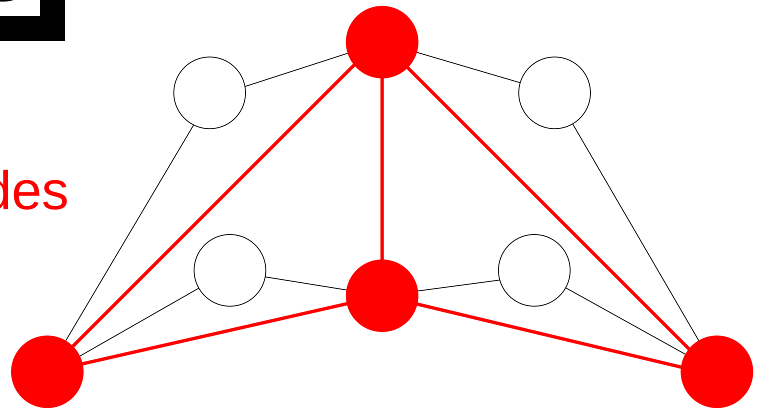
- classification of nodes by 'importance' might be slightly perturbed, but not completely changed

  (e.g., a sports car and a truck both prefer motorways)

  $\leadsto$ performance of our approach relies on that

  (not the correctness)

Highway-Node Routing

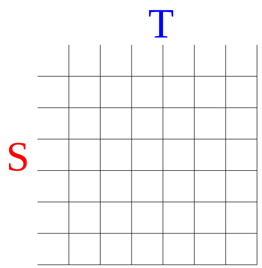


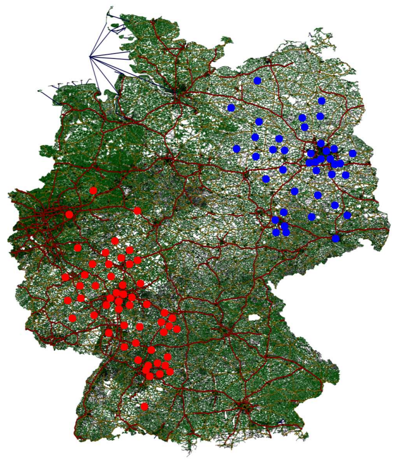
1. **basic concepts:** overlay graphs, covering nodes

2. lightweight, efficient **static approach**

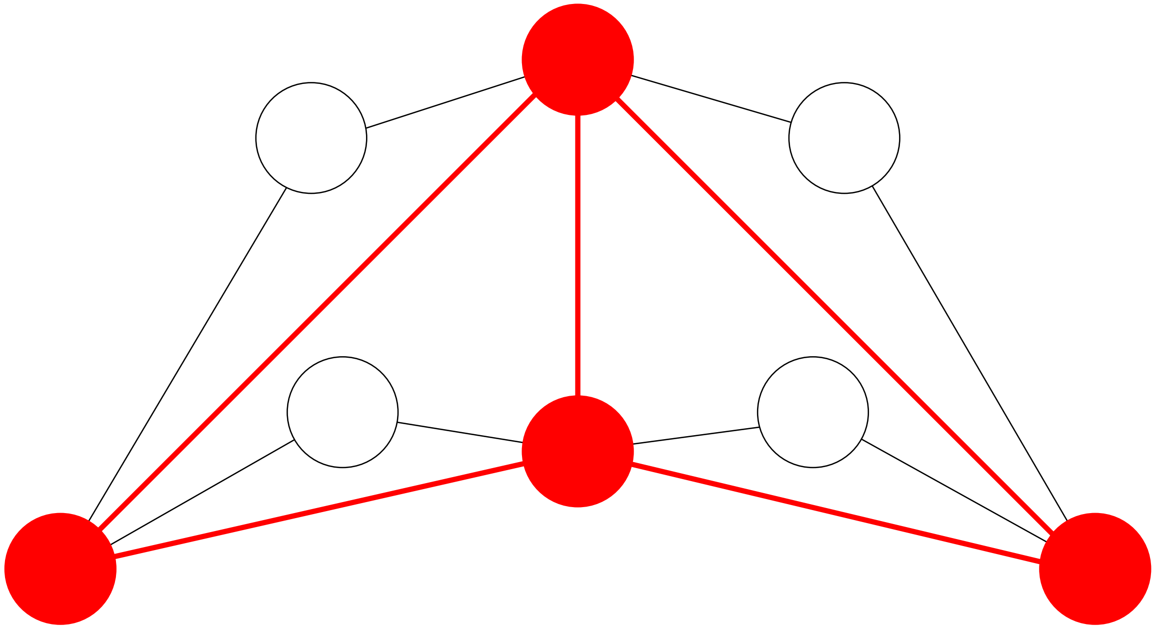
3. **dynamic version**

4. **many-to-many extension**





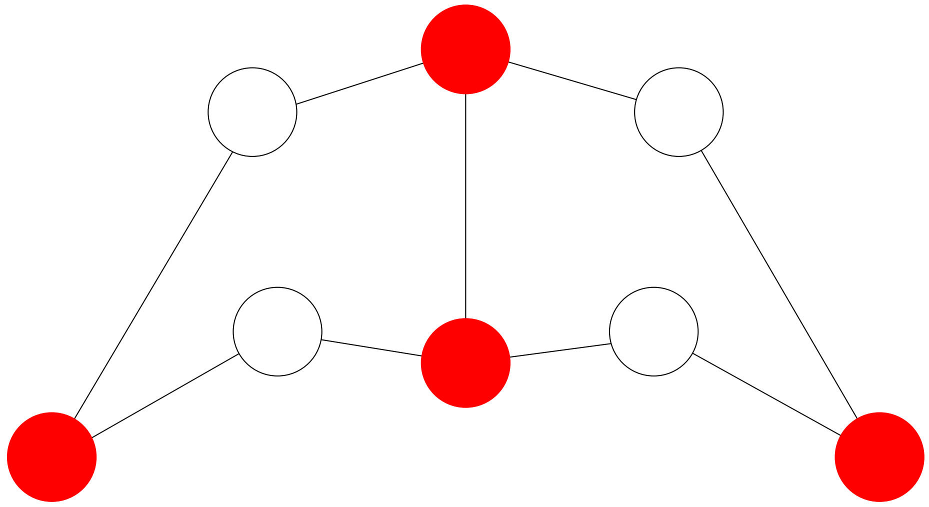
1. Basic Concepts



Overlay Graph: Definition

[Holzer, Schulz, Wagner, Weihe, Zaroliagis 2000–2007]

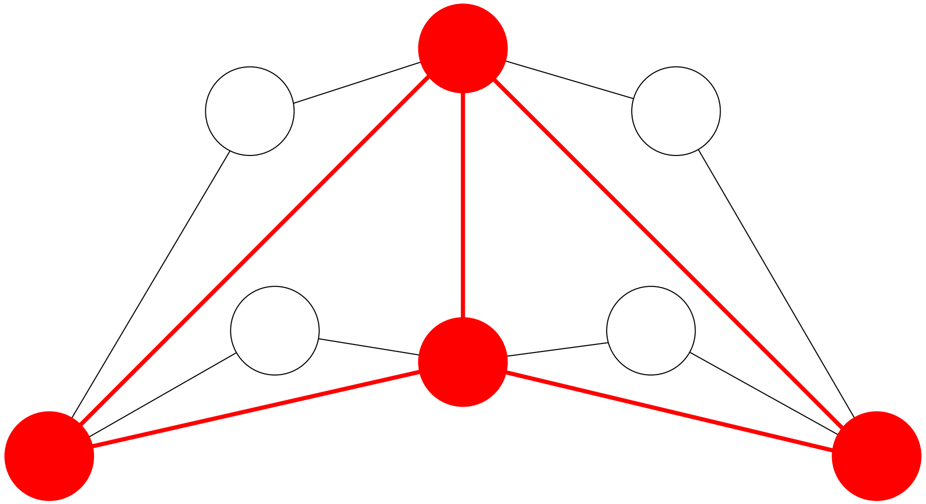
- graph $G = (V, E)$ is given
- select node subset $S \subseteq V$



Overlay Graph: Definition

[Holzer, Schulz, Wagner, Weihe, Zaroliagis 2000–2007]

- graph $G = (V, E)$ is given
- select node subset $S \subseteq V$



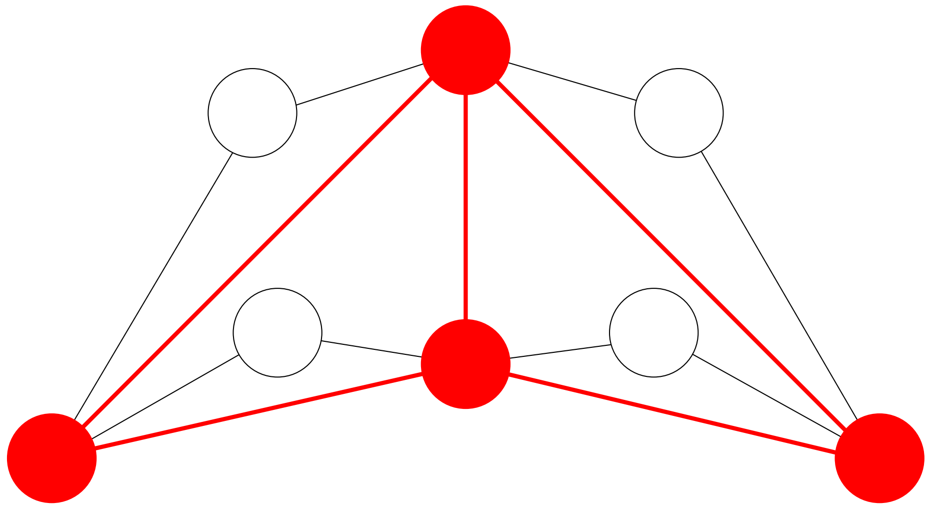
- overlay graph $G' := (S, E')$

determine edge set E' s.t. shortest path distances are preserved

Minimal Overlay Graph

[Holzer, Schulz, Wagner, Weihe, Zaroliagis 2000–2007]

- graph $G = (V, E)$ is given
- select node subset $S \subseteq V$



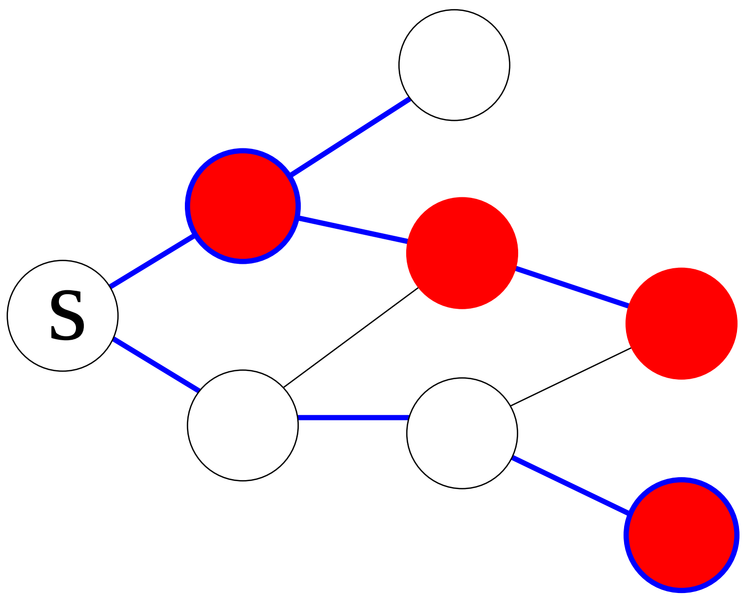
- minimal overlay graph $G' := (S, E')$ where

$E' := \{(s,t) \in S \times S \mid$ no inner node of the shortest s-t-path belongs to $S\}$

Covering Nodes

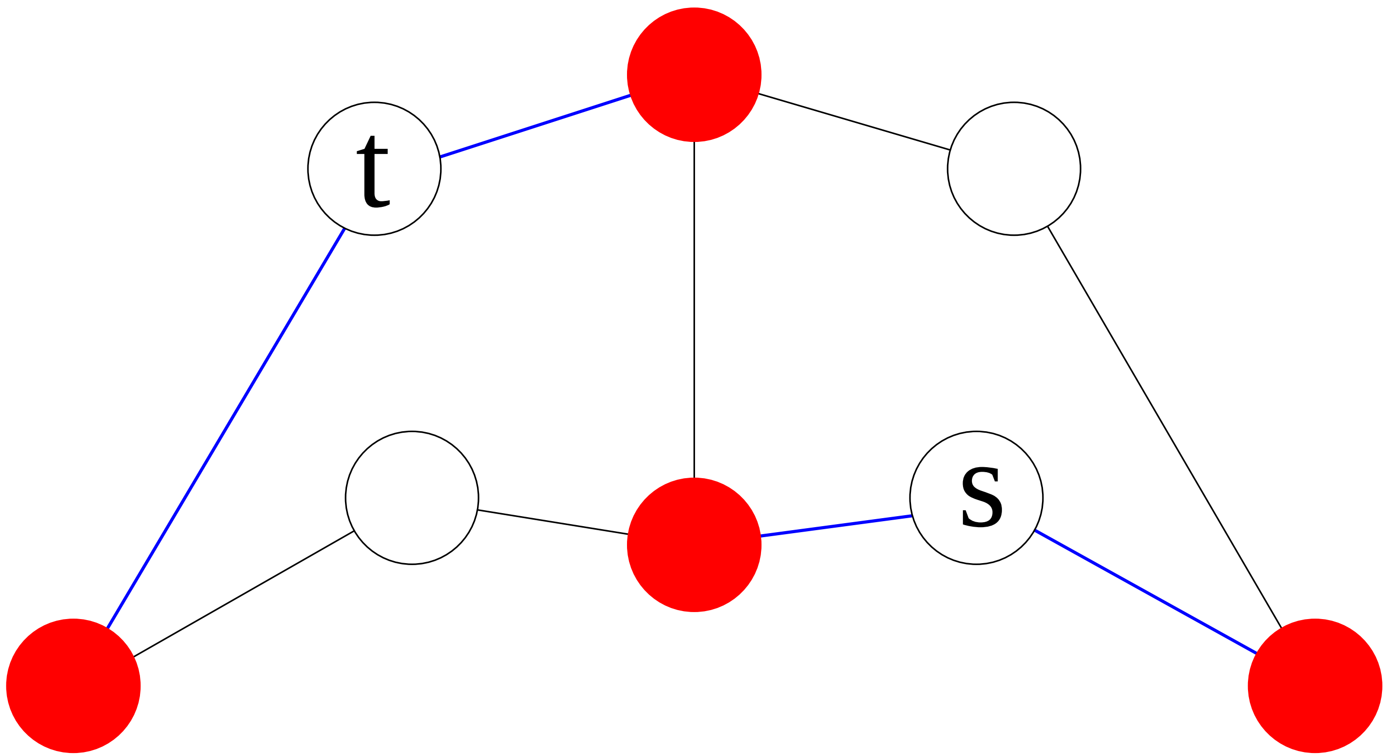
Definitions:

- covered branch: contains a node from S
- covered tree: all branches covered
- covering nodes: on each branch, the node $u \in S$ closest to the root s



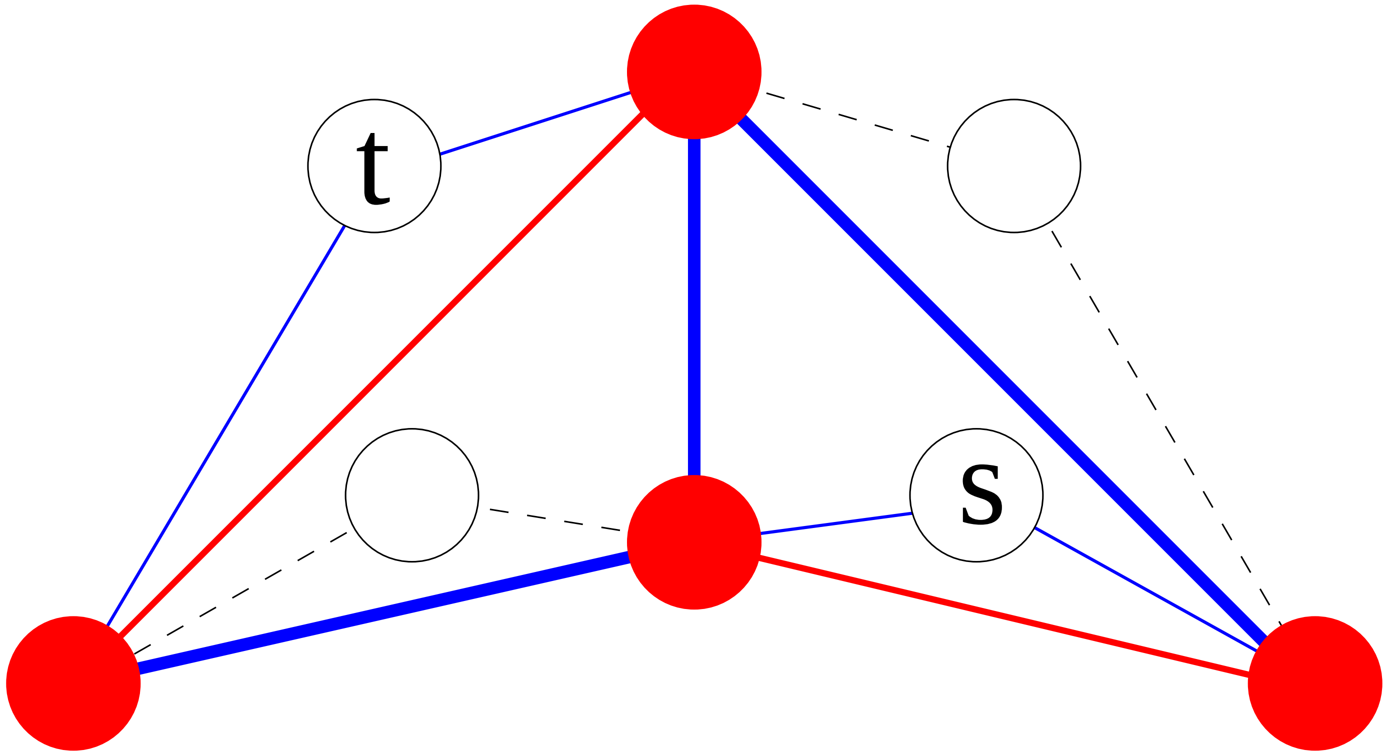
Query: Intuition

- bidirectional
- perform search in G till search trees are covered by nodes in S



Query: Intuition

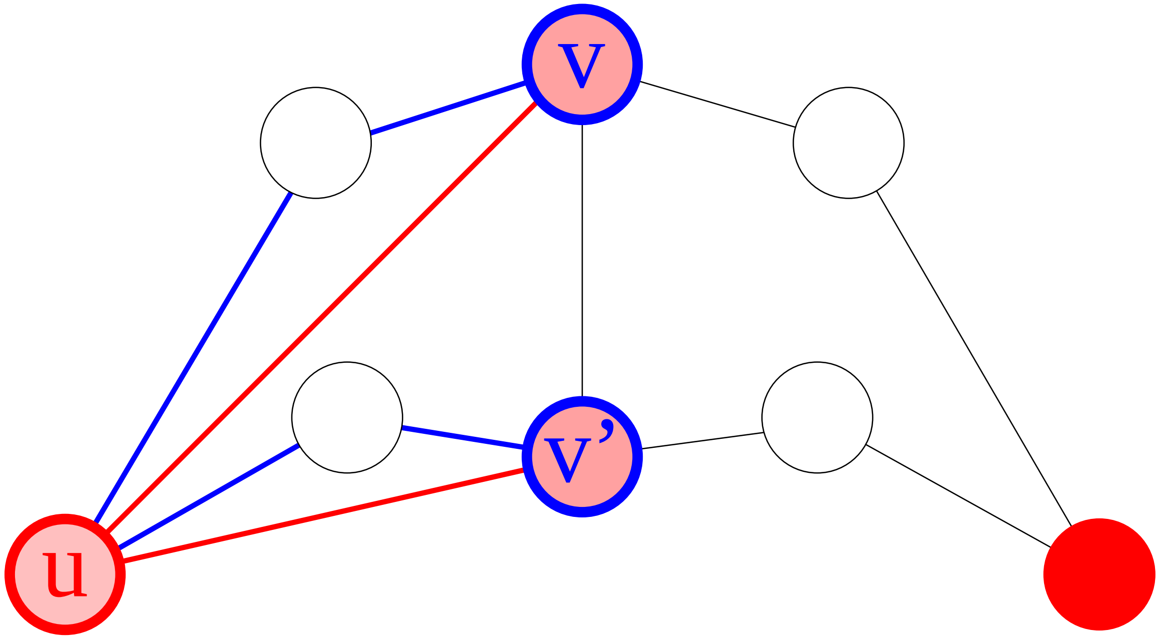
- bidirectional
- perform search in G till search trees are covered by nodes in S
- continue search only in G'



Overlay Graph: Construction

for each node $u \in S$

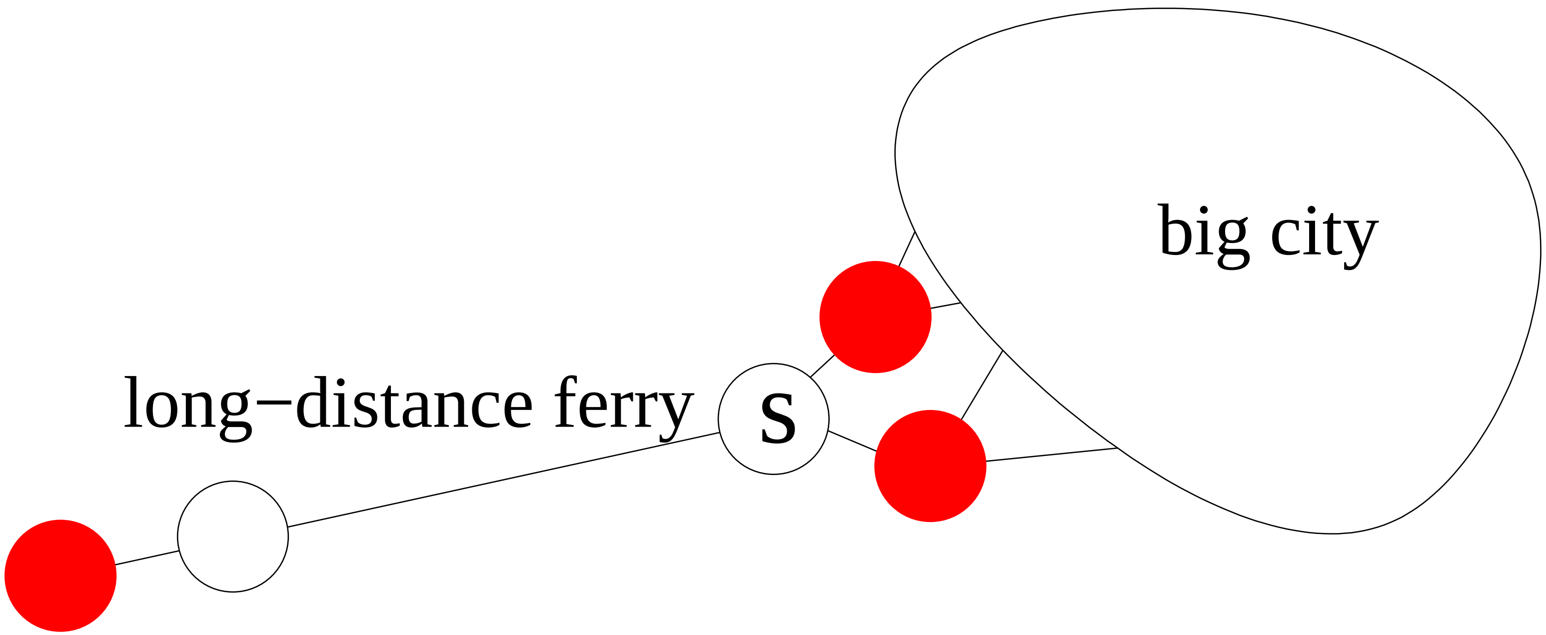
- perform a local search from u in G
- determine the covering nodes
- add an edge (u,v) to E' for each covering node v



Covering Nodes

Conservative Approach:

- stop searching in G when all branches are covered

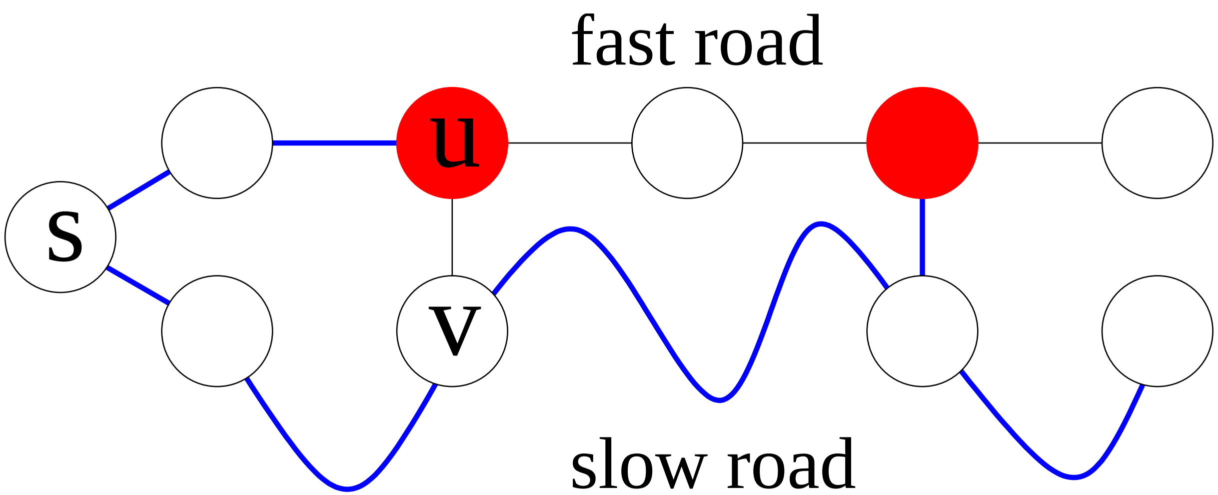


- can be very inefficient

Covering Nodes

Aggressive Approach:

- do not continue the search in G on covered branches



- can be very inefficient

Covering Nodes

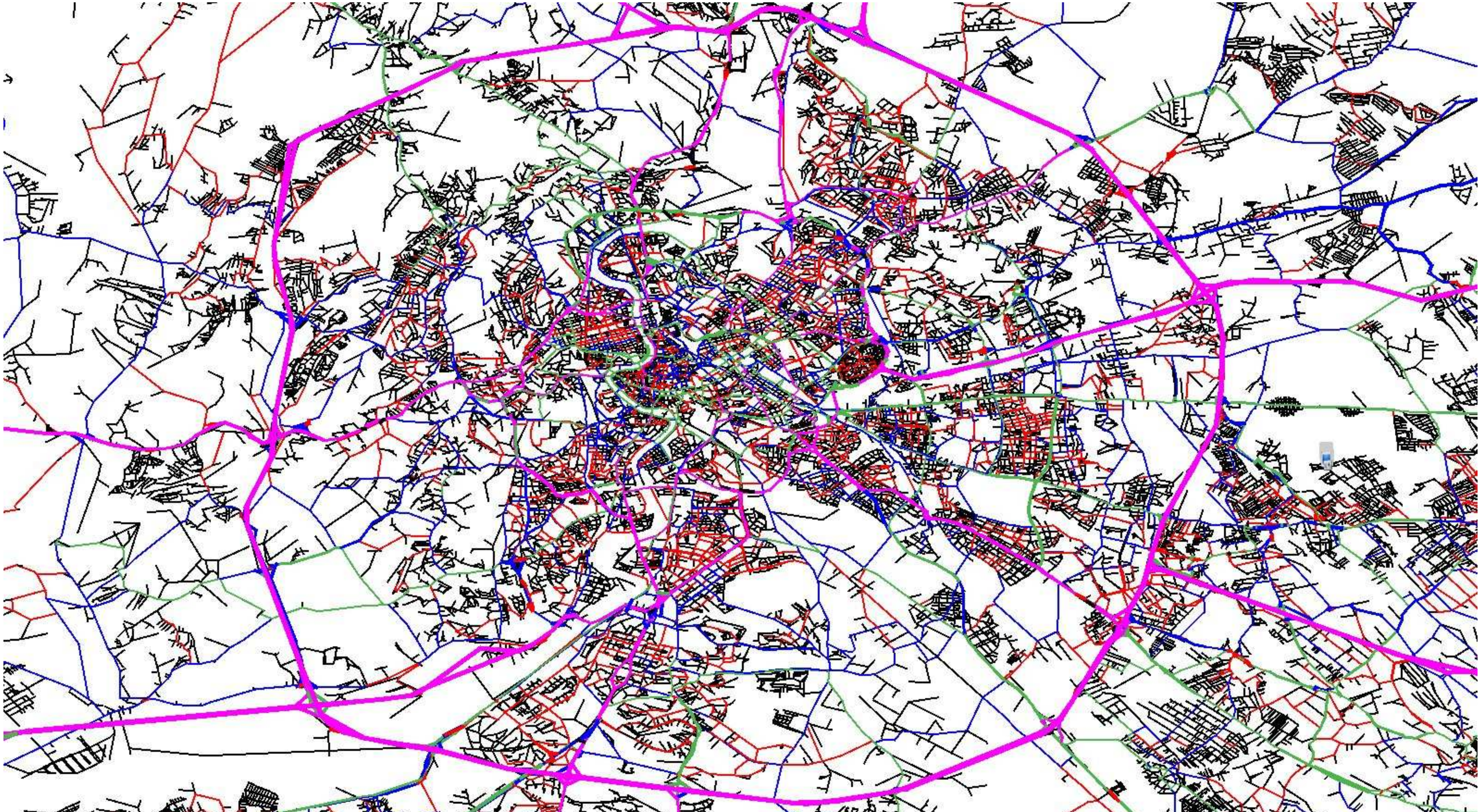
Compromise:

- introduce parameter p
- do not continue the search in G on branches that already contain p nodes from S
- in addition: stop when all branches are covered
- $p = 1 \longrightarrow$ aggressive
- $p = \infty \longrightarrow$ conservative

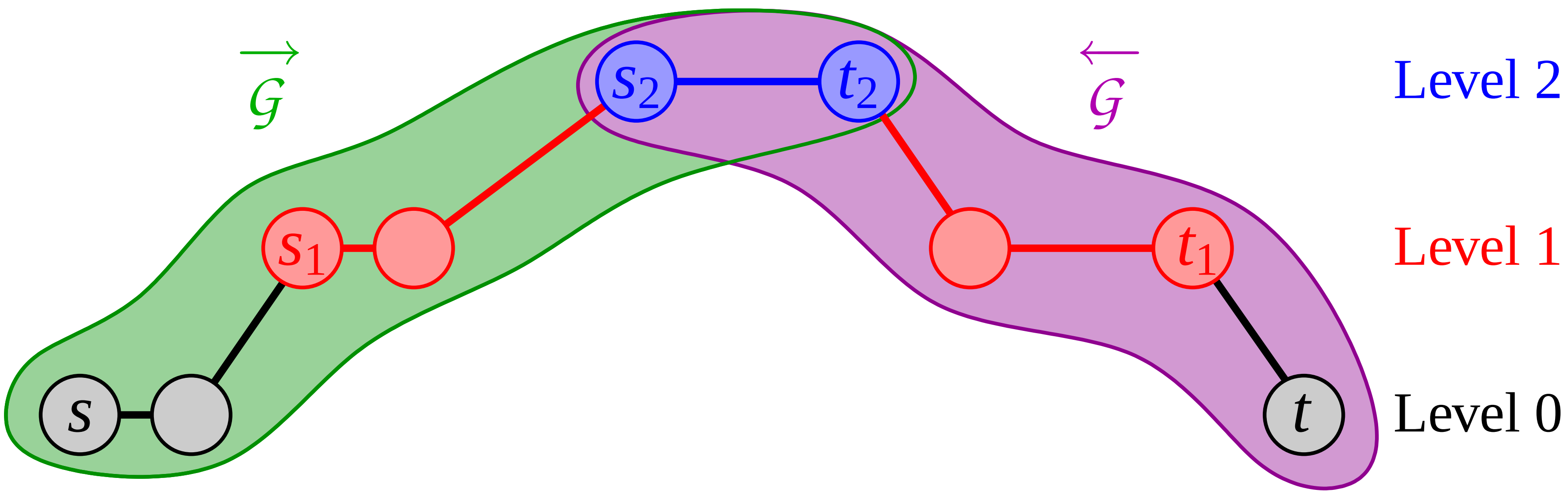
- works very well in practice

Highway Hierarchies

- previous static route-planning approach [SS05–06]
- determines a hierarchical representation of nodes and edges



2. Static Highway-Node Routing



Static Highway-Node Routing

- extend ideas from
  - multi-level overlay graphs [HolzerSchulzWagnerWeiheZaroliagis00–07]
  - highway hierarchies [SS05–06]
  - transit node routing [BastFunkeMatijevicSS06–07]
- use highway hierarchies to classify nodes by 'importance'

  i.e., select node sets $S_1 \supseteq S_2 \supseteq S_3 \ldots \supseteq S_L$

  (crucial distinction from previous separator-based approach)
- construct multi-level overlay graph

  $G_0 = G = (V,E), G_1 = (S_1,E_1), G_2 = (S_2,E_2), \ldots, G_L = (S_L,E_L)$

  (just iteratively construct overlay graphs)

Static Highway-Node Routing

- extend ideas from
  - multi-level overlay graphs [HolzerSchulzWagnerWeiheZaroliagis00–07]
  - highway hierarchies [SS05–06]
  - transit node routing [BastFunkeMatijevicSS06–07]

- use highway hierarchies to classify nodes by 'importance'

  i.e., select node sets $S_1 \supseteq S_2 \supseteq S_3 \ldots \supseteq S_L$ 13 min

  (crucial distinction from previous separator-based approach)

- construct multi-level overlay graph 2 min

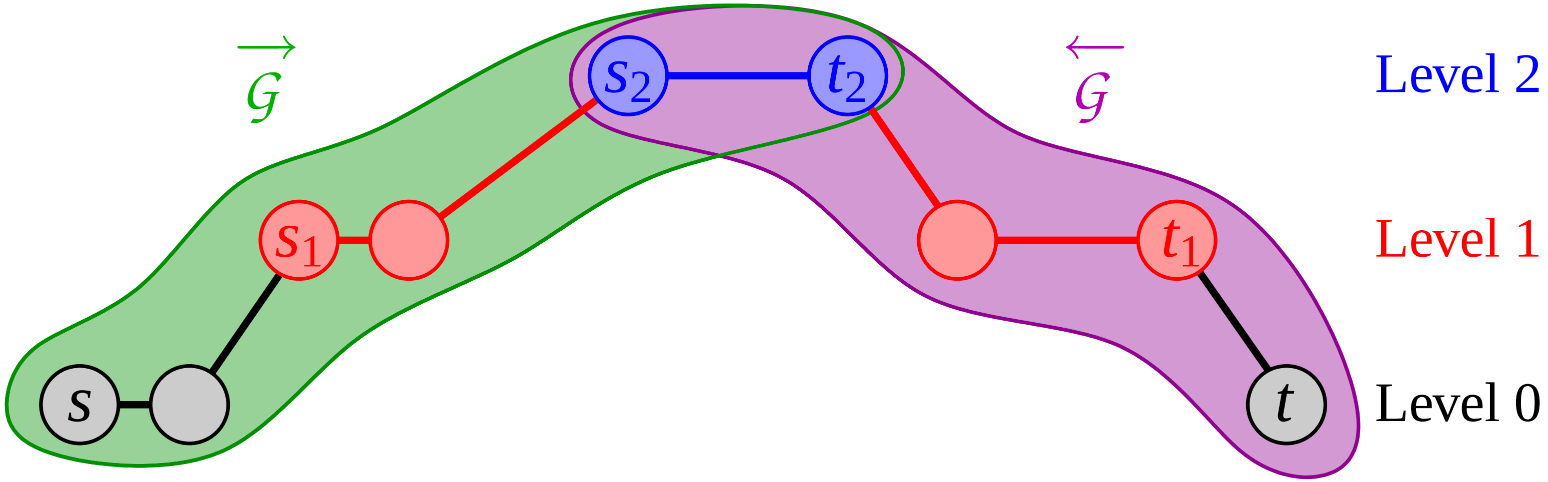
  $G_0 = G = (V,E), G_1 = (S_1,E_1), G_2 = (S_2,E_2), \ldots, G_L = (S_L,E_L)$

  (just iteratively construct overlay graphs)

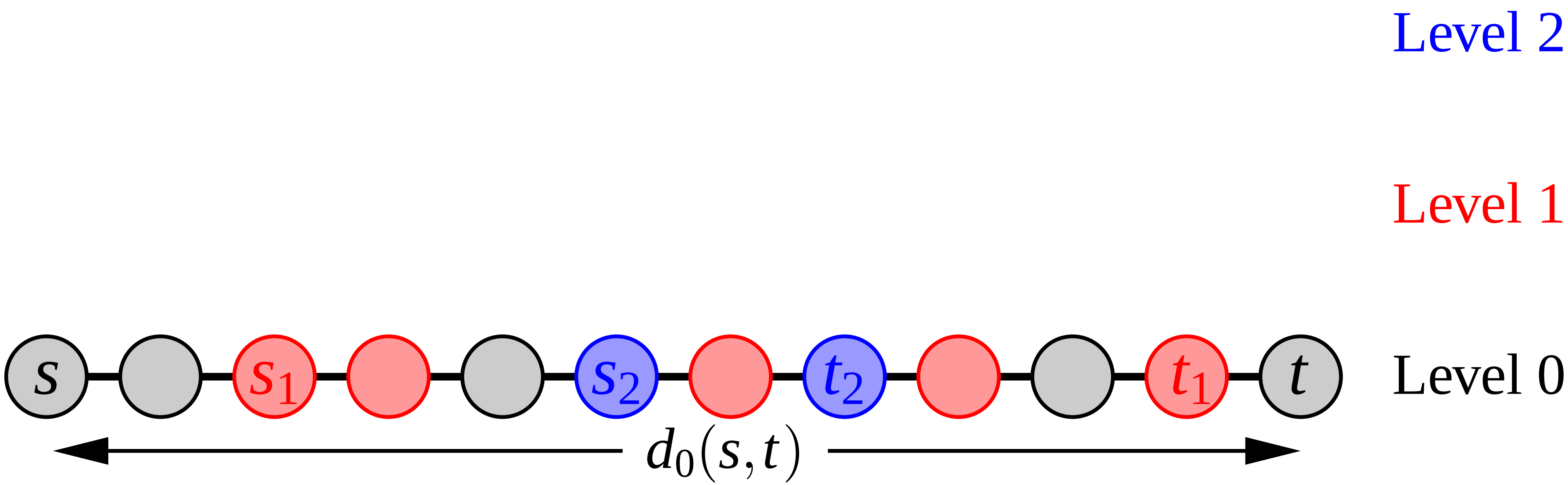
(experiments with a European road network with $\approx$ 18 million nodes)

Query: Aggressive Variant

- node level $\ell(u) := \max\{\ell \mid u \in S_\ell\}$
- forward search graph $\overrightarrow{\mathcal{G}} := \left(V, \left\{(u,v) \mid (u,v) \in \bigcup_{i=\ell(u)}^{L} E_i\right\}\right)$
- backward search graph $\overleftarrow{\mathcal{G}} := \left(V, \left\{(u,v) \mid (v,u) \in \bigcup_{i=\ell(u)}^{L} E_i\right\}\right)$
- perform one plain Dijkstra search in $\overrightarrow{\mathcal{G}}$ and one in $\overleftarrow{\mathcal{G}}$

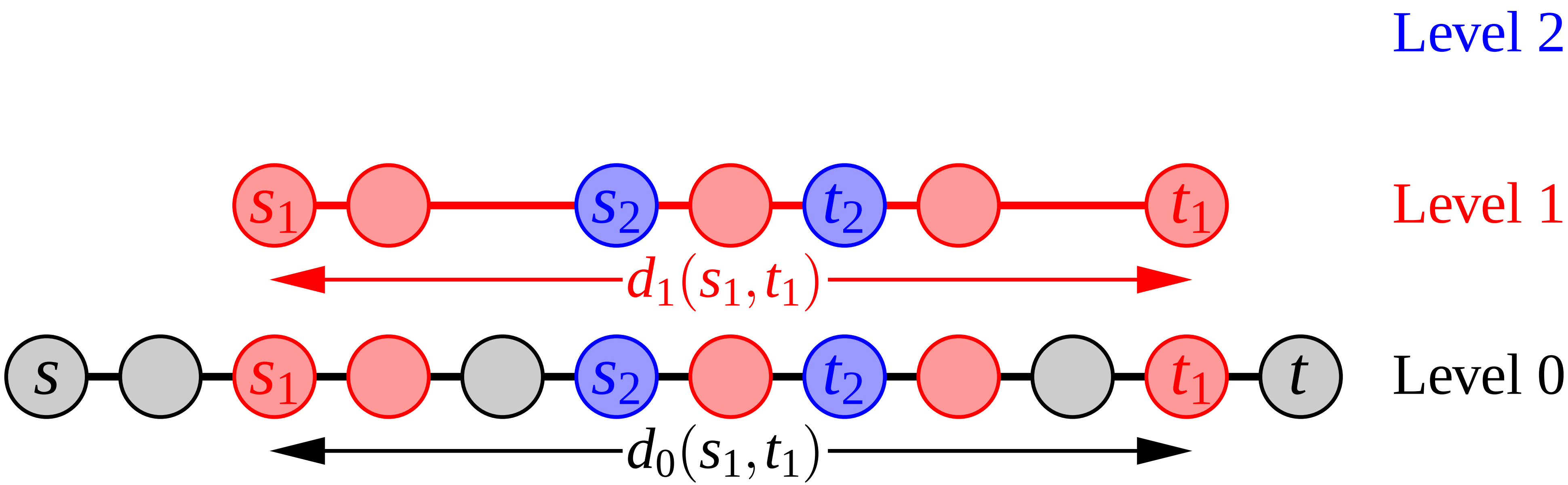


Proof of Correctness



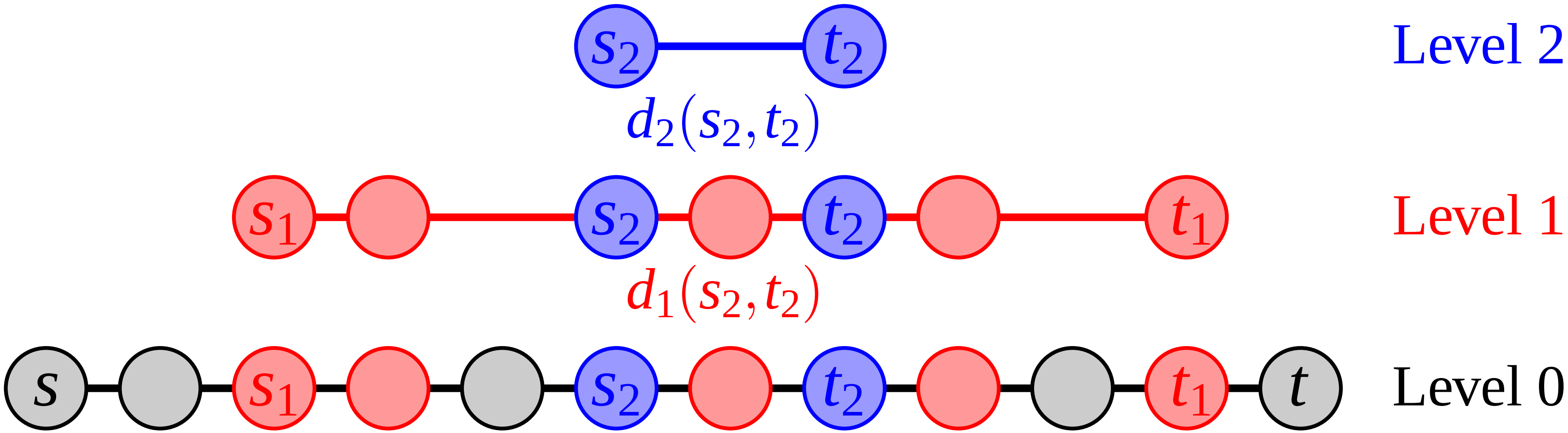
shortest path from s to t in $G = G_0$

Proof of Correctness



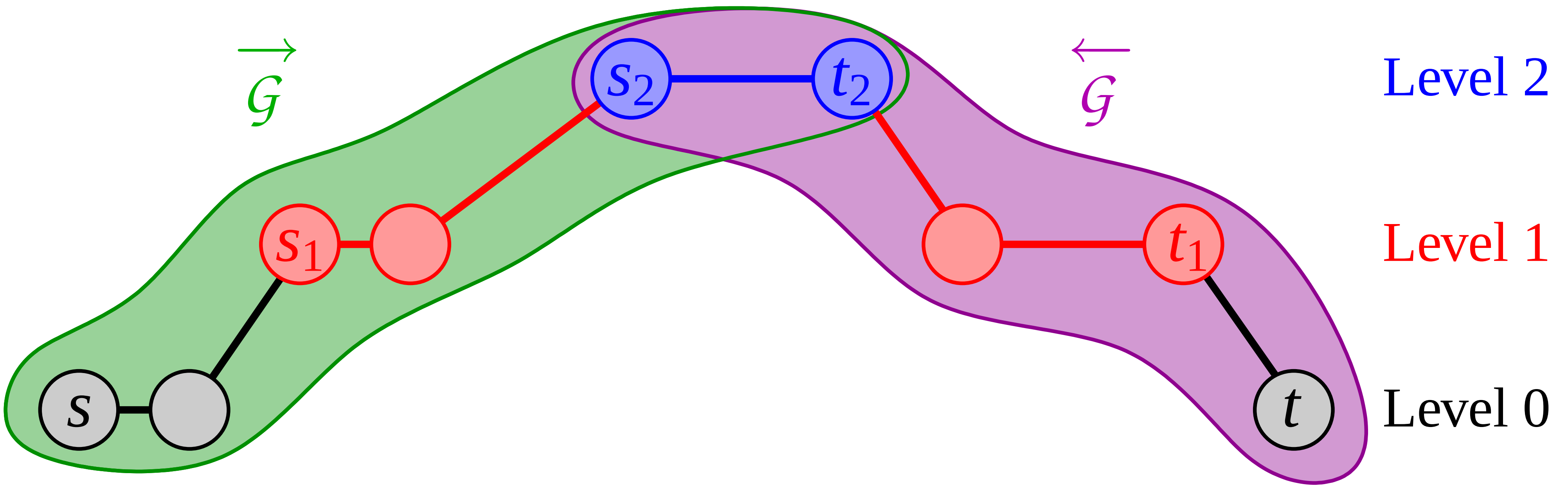
overlay graph G_1 preserves distance from $s_1 \in S_1$ to $t_1 \in S_1$

Proof of Correctness



overlay graph G_2 preserves distance from $s_2 \in S_2$ to $t_2 \in S_2$

Proof of Correctness



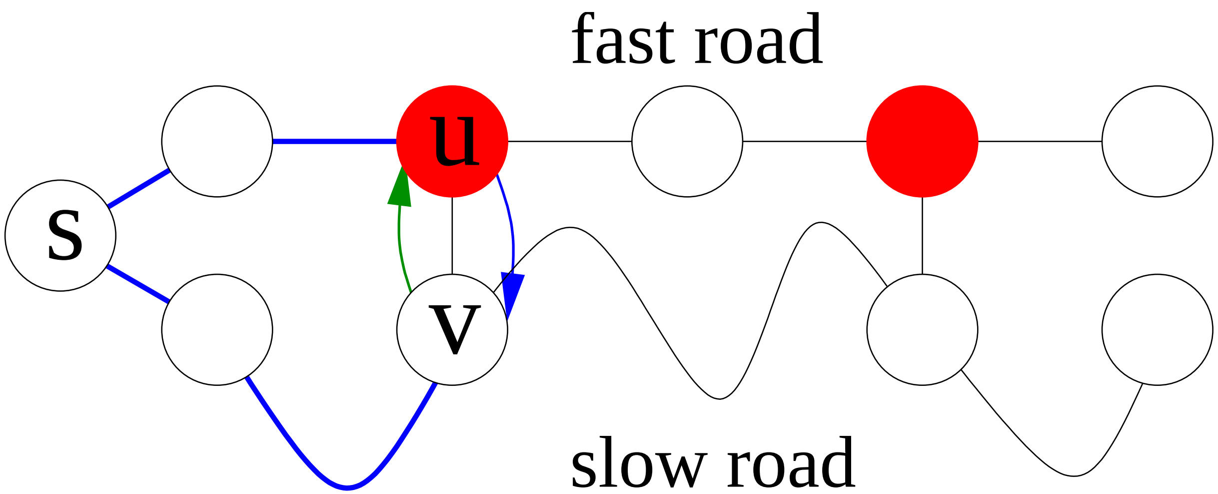
$$\overrightarrow{\mathcal{G}} := \left(V, \left\{(u,v) \mid (u,v) \in \bigcup_{i=\ell(u)}^{L} E_i\right\}\right)$$

$$\overleftarrow{\mathcal{G}} := \left(V, \left\{(u,v) \mid (v,u) \in \bigcup_{i=\ell(u)}^{L} E_i\right\}\right)$$

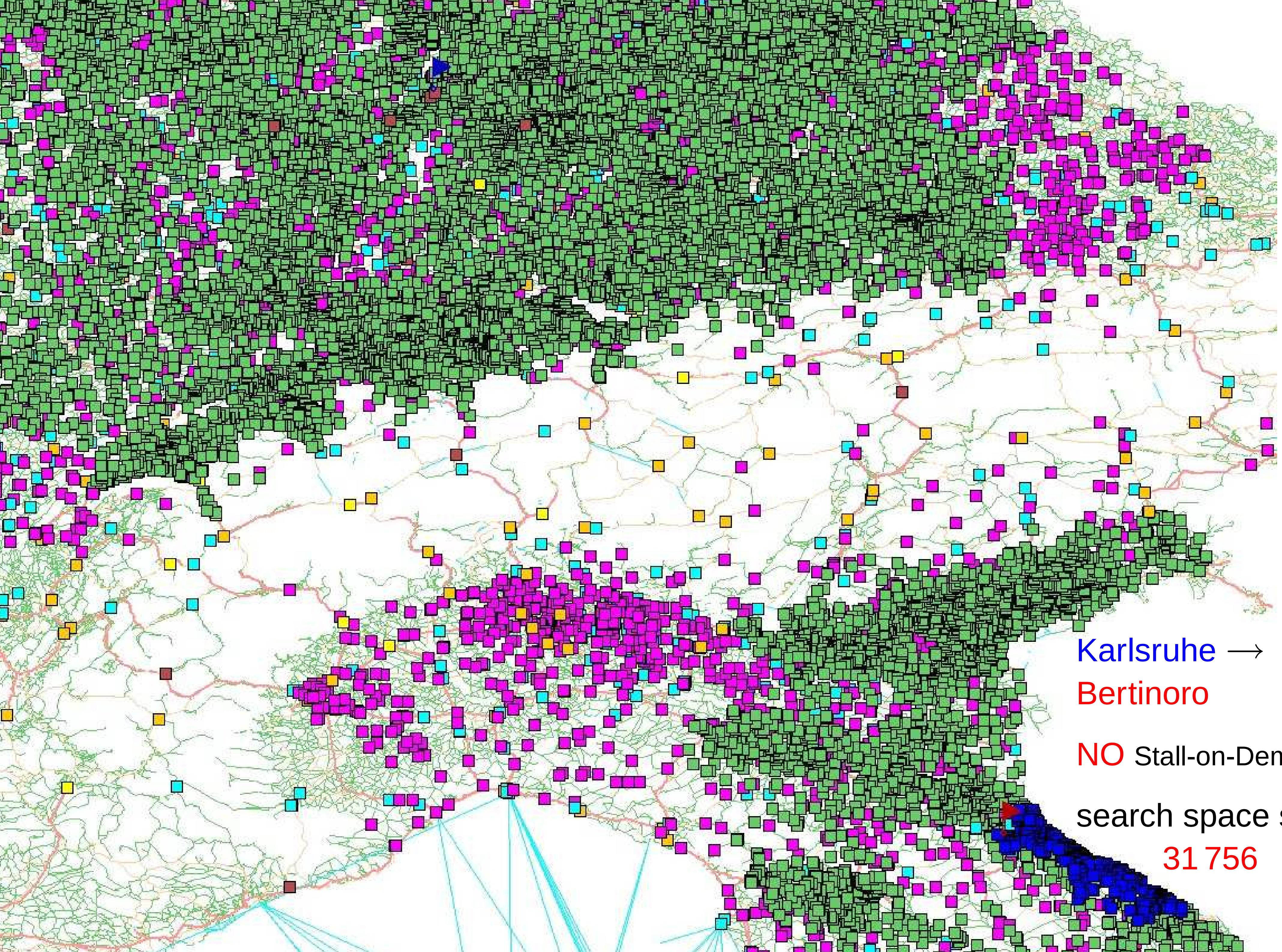
Stall-on-Demand

- a node v can 'wake' a node u if $\ell(u) > \ell(v)$
- u can 'stall' v (if $\delta(u) + w(u,v) < \delta(v)$)

  i.e., search is not contined from v



- stalling can propagate to adjacent nodes
- does not invalidate correctness (only suboptimal paths are stalled)



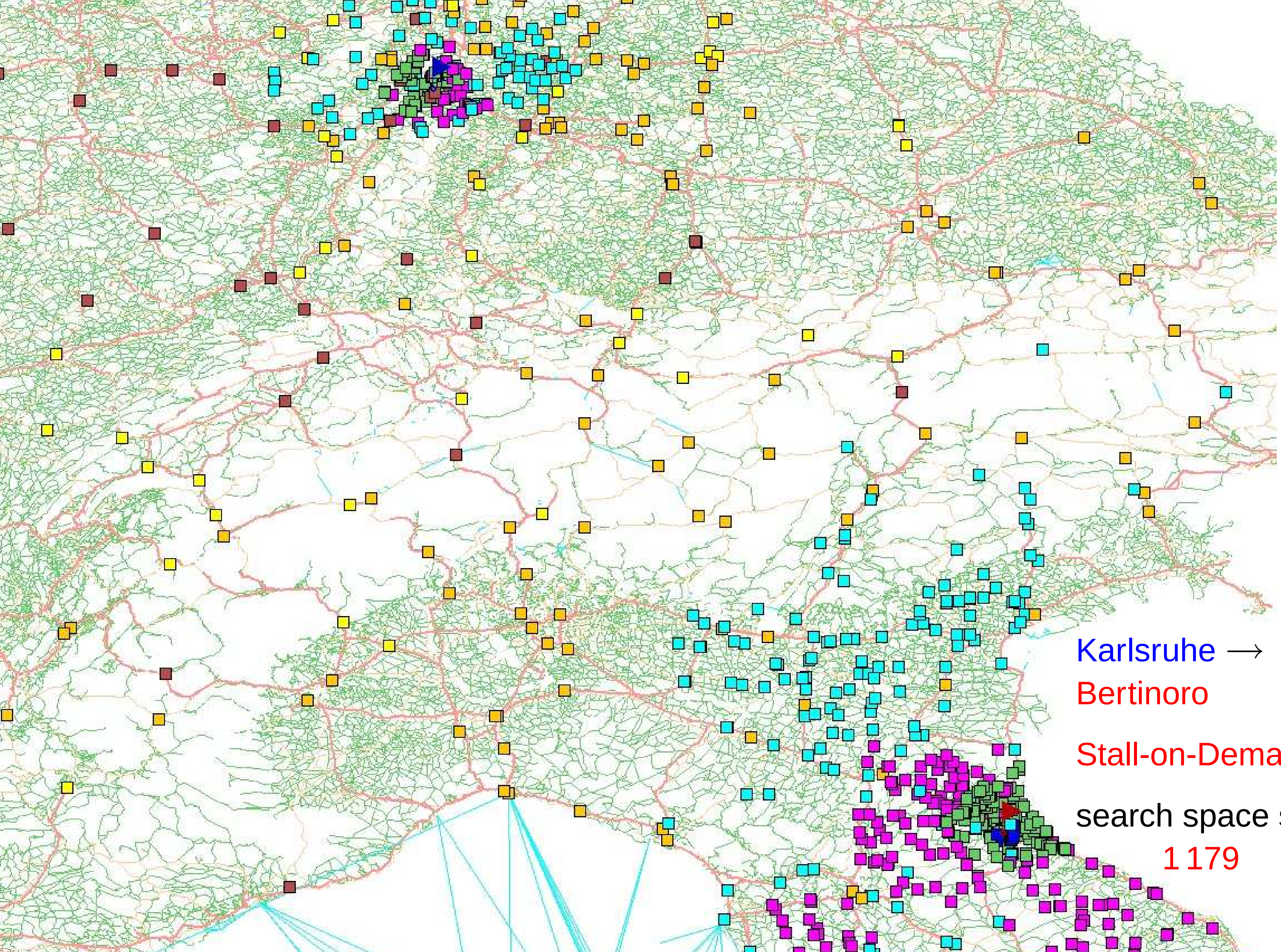
Karlsruhe →

Bertinoro

NO Stall-on-Demand

search space size:

31 756



Karlsruhe →
Bertinoro

Stall-on-Demand

search space size:
1 179

Memory Consumption / Query Time

different trade-offs between memory consumption and query time

for example:

- 9 bytes per node overhead $\longrightarrow$ 0.88 ms

  store complete multi-level overlay graph

- 0.7 bytes per node overhead $\longrightarrow$ 1.44 ms

  store only forward and backward search graph $\overrightarrow{\mathcal{G}}$ and $\overleftarrow{\mathcal{G}}$
  ($\overrightarrow{\mathcal{G}}$ and $\overleftarrow{\mathcal{G}}$ are independent of s and t)

---

numbers refer to the Western European road network with 18 million nodes

3. Dynamic Highway-Node Routing

Dynamic Highway-Node Routing

change entire cost function

- keep the node sets $S_1 \supseteq S_2 \supseteq S_3 \ldots$
- recompute the overlay graphs

speed profile	default	fast car	slow car	slow truck	distance
constr. [min]	1:40	1:41	1:39	1:36	3:56
query [ms]	1.17	1.20	1.28	1.50	35.62
#settled nodes	1 414	1 444	1 507	1 667	7 057

Dynamic Highway-Node Routing

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

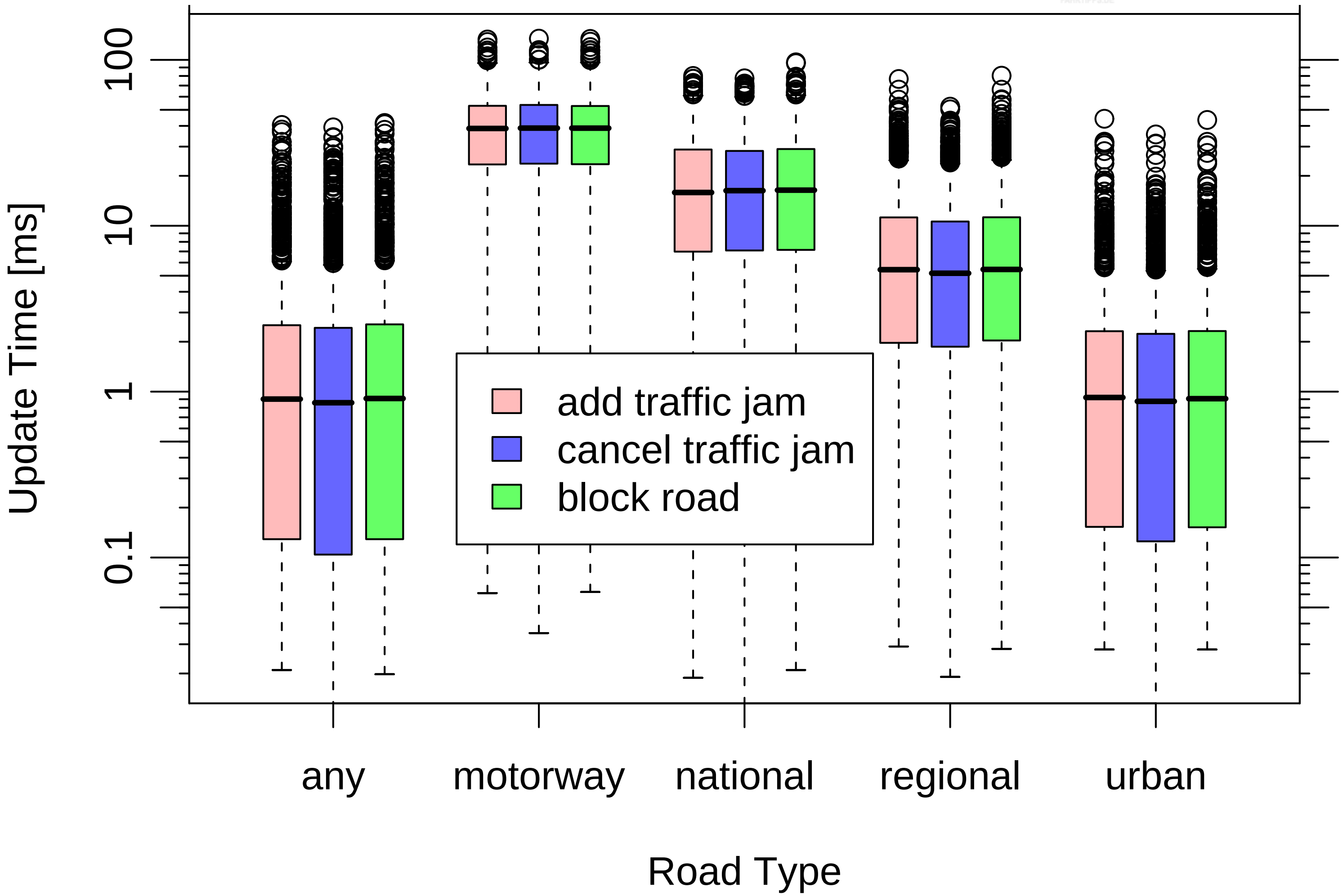
Dynamic Highway-Node Routing

change a few edge weights, server scenario

- keep the node sets $S_1 \supseteq S_2 \supseteq S_3 \ldots$
- recompute only possibly affected parts of the overlay graphs
  - the computation of the level-ℓ overlay graph consists of $|S_\ell|$ local searches to determine the respective covering nodes
  - if the initial local search from $v \in S_\ell$ has not touched a now modified edge (u,x), that local search need not be repeated
  - we manage sets $A_u^\ell = \{v \in S_\ell \mid v$'s level-$\ell$ preprocessing might be affected when an edge (u,x) changes$\}$

Dynamic Highway-Node Routing

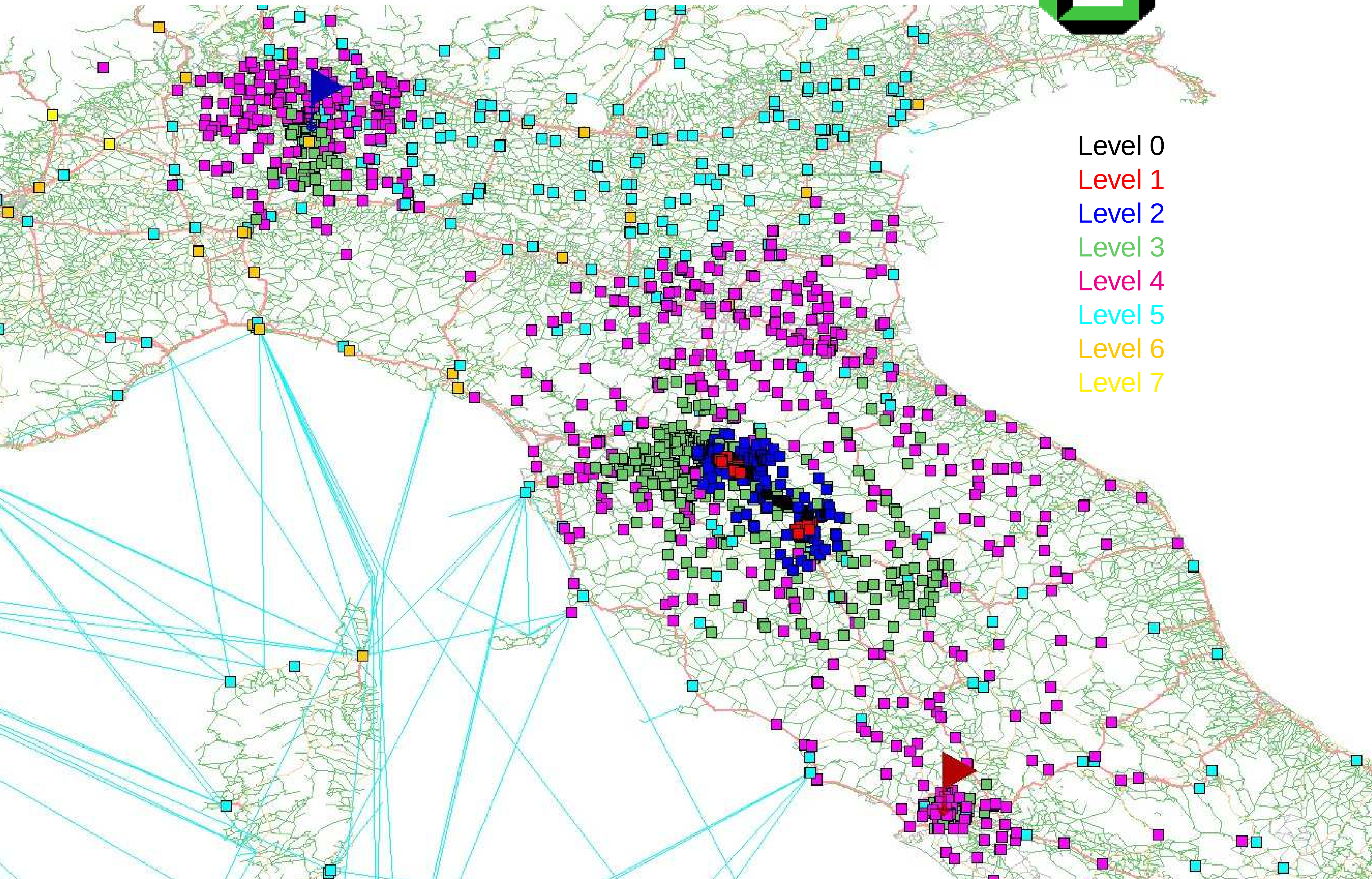
change a few edge weights, server scenario



Dynamic Highway-Node Routing

change a few edge weights, mobile scenario

1. keep the node sets $S_1 \supseteq S_2 \supseteq S_3 \ldots$
2. keep the overlay graphs
3. $C :=$ all changed edges
4. use the sets A_u^ℓ (considering edges in C) to determine for each node v a reliable level $r(v)$
5. during a query, at node v
   - do not use edges that have been created in some level $> r(v)$
   - instead, downgrade the search to level $r(v)$


Level 0
Level 1
Level 2
Level 3
Level 4
Level 5
Level 6
Level 7

Dynamic Highway-Node Routing

change a few edge weights, mobile scenario

iterative variant (provided that only edge weight increases allowed)

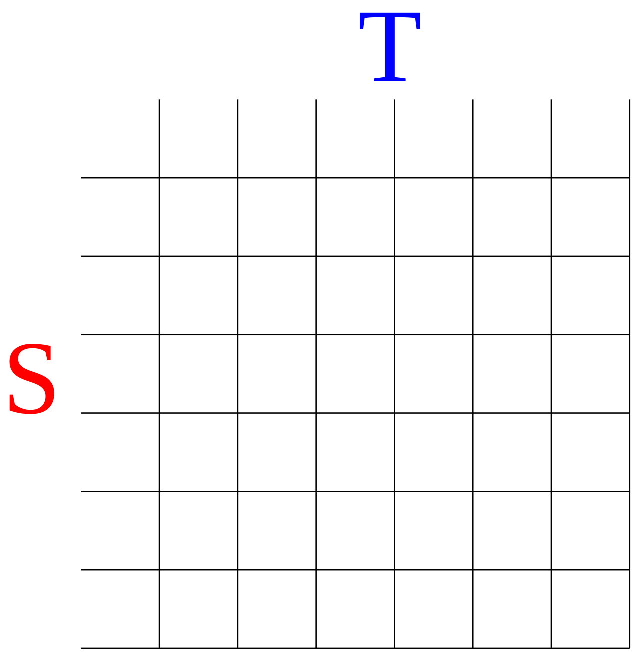
1. keep everything (as before)
2. $C := \emptyset$
3. use the sets A_u^ℓ (considering edges in C) to determine for each node v a reliable level $r(v)$ (as before)
4. ‘prudent’ query (as before)
5. if shortest path P does not contain a changed edge, we are done
6. otherwise: add changed edges on P to C, **repeat** from 3.

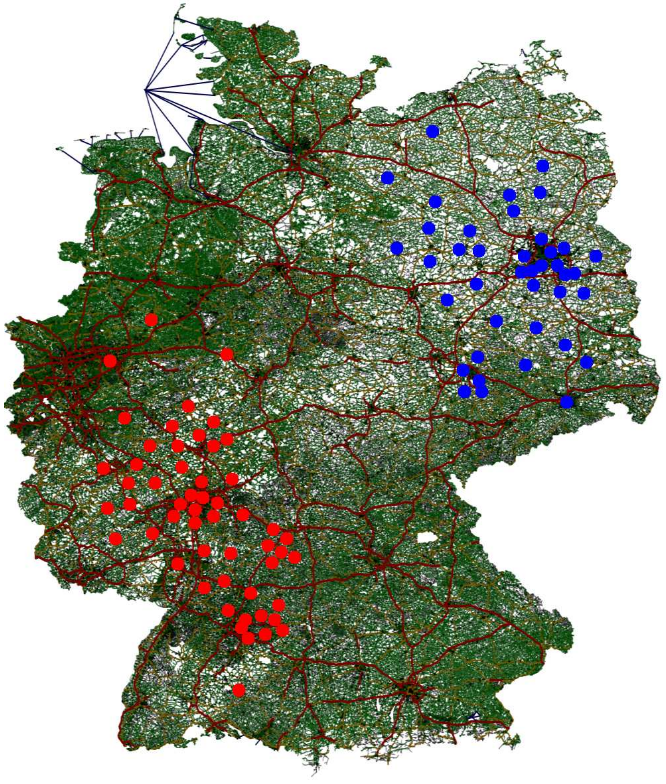
Dynamic Highway-Node Routing

change a few edge weights, mobile scenario

|change set| (motorway edges)	affected queries	single pass: query time [ms]	iterative: query time [ms]	iterative: #iterations avg	iterative: #iterations max
1	0.4 %	2.3	1.5	1.0	2
10	5.8 %	8.5	1.7	1.1	3
100	40.0 %	47.1	3.6	1.4	5
1 000	83.7 %	246.3	25.3	2.7	9

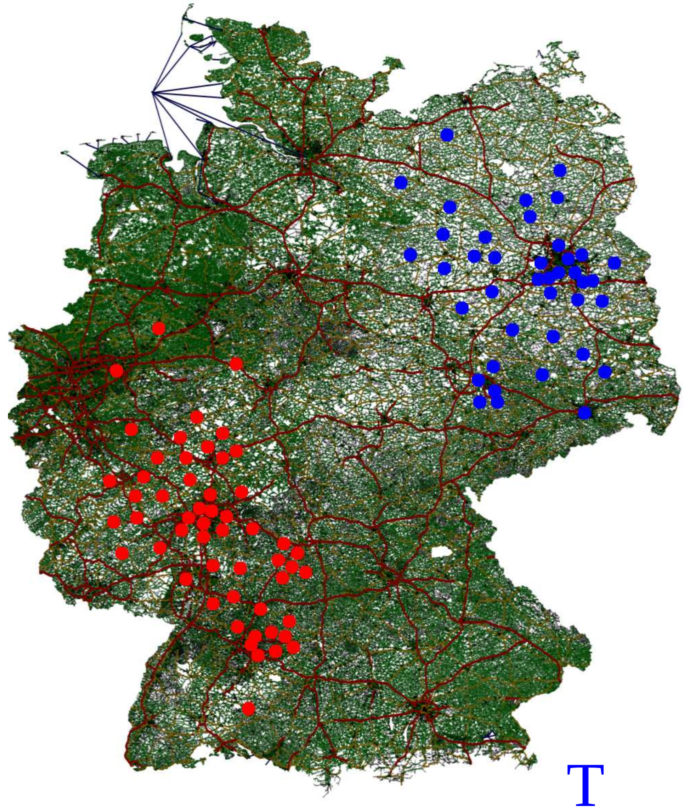
4. Many-to-Many Extension





Many-to-Many Routing

[with S. Knopp, F. Schulz (PTV AG), D. Wagner]

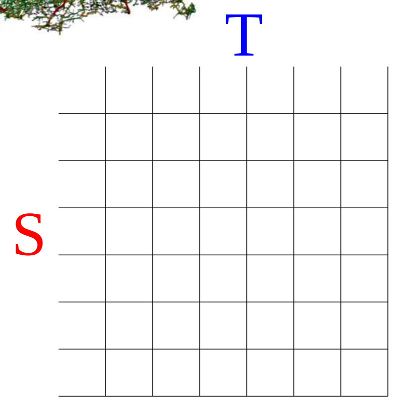


Given:

- graph $G = (V, E)$
- set of source nodes $S \subseteq V$
- set of target nodes $T \subseteq V$

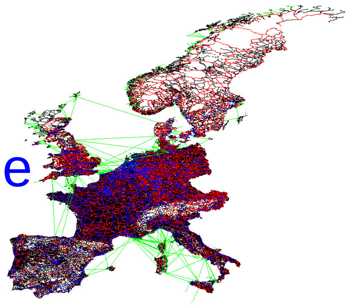
Task: compute $|S| \times |T|$ distance table

containing the shortest path distances



Simple Solutions

Example: 10 000 × 10 000 table in Western Europe



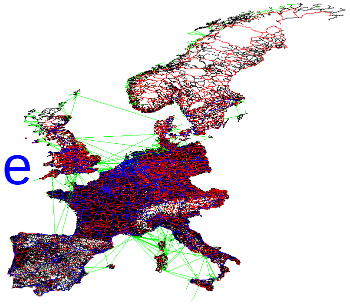
- apply $\underbrace{\text{SSSP algorithm}}_{\text{(e.g. Dijkstra)}}$ $|S|$ times $\approx 10\,000 \times 10\,\text{s} \approx$ one day

- apply $\underbrace{\text{P2P algorithm}}_{\text{(e.g. highway-node routing}^1\text{)}}$ $|S| \times |T|$ times $\approx 10\,000^2 \times 1\,\text{ms} \approx$ one day

---

[1] requires about 15 minutes preprocessing time

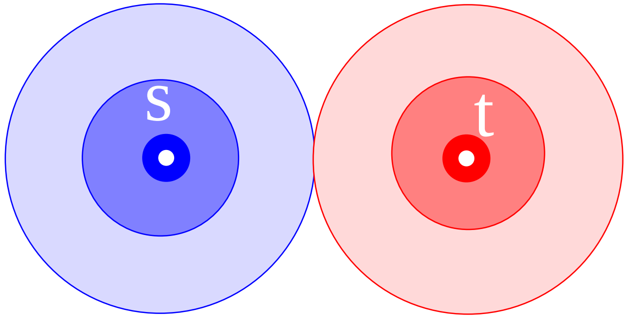
Our Solution



Example: 10 000 × 10 000 table
in Western Europe

- many-to-many algorithm
  based on highway-node routing[1]

23 seconds



[1] requires about 15 minutes preprocessing time

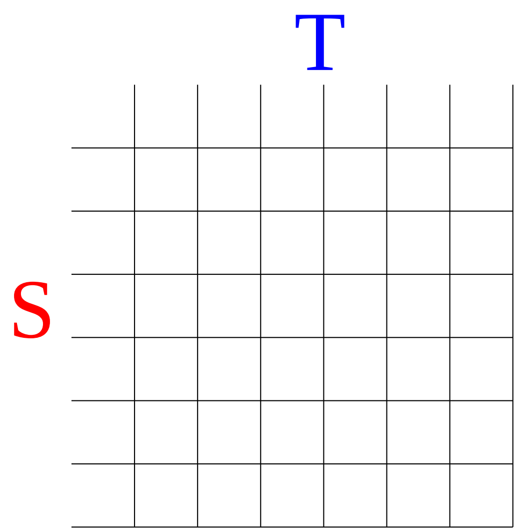
Main Idea

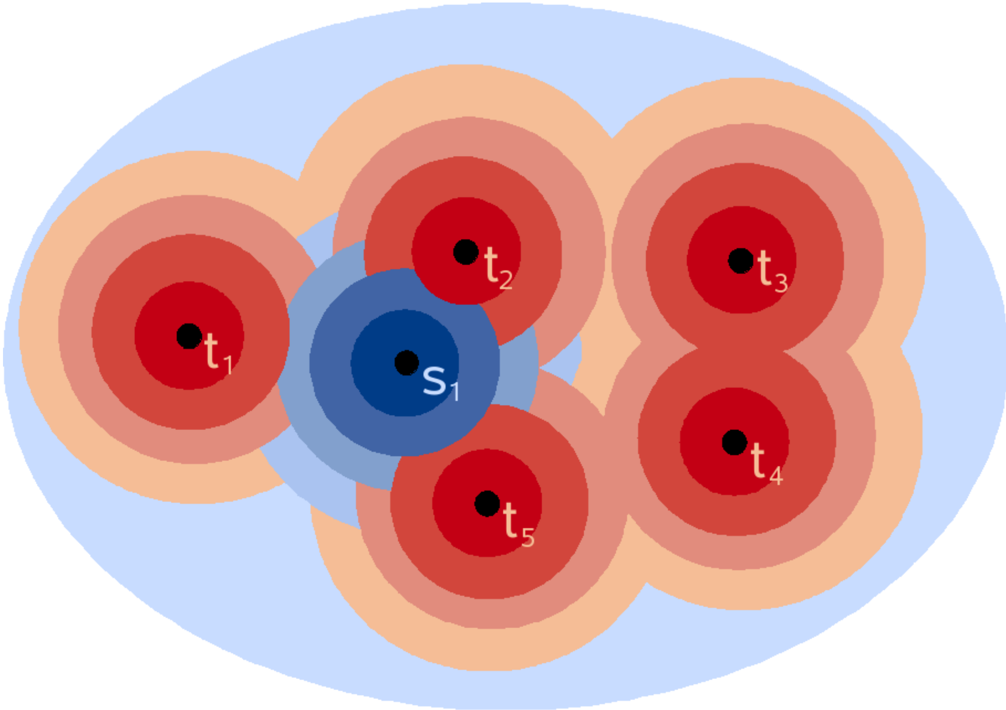
- instead of $|S| \times |T|$ bidirectional highway-node queries
- perform $|S| + |T|$ unidirectional highway-node queries

Algorithm

- maintain an $|S| \times |T|$ table D of tentative distances

  (initialize all entries to ∞)



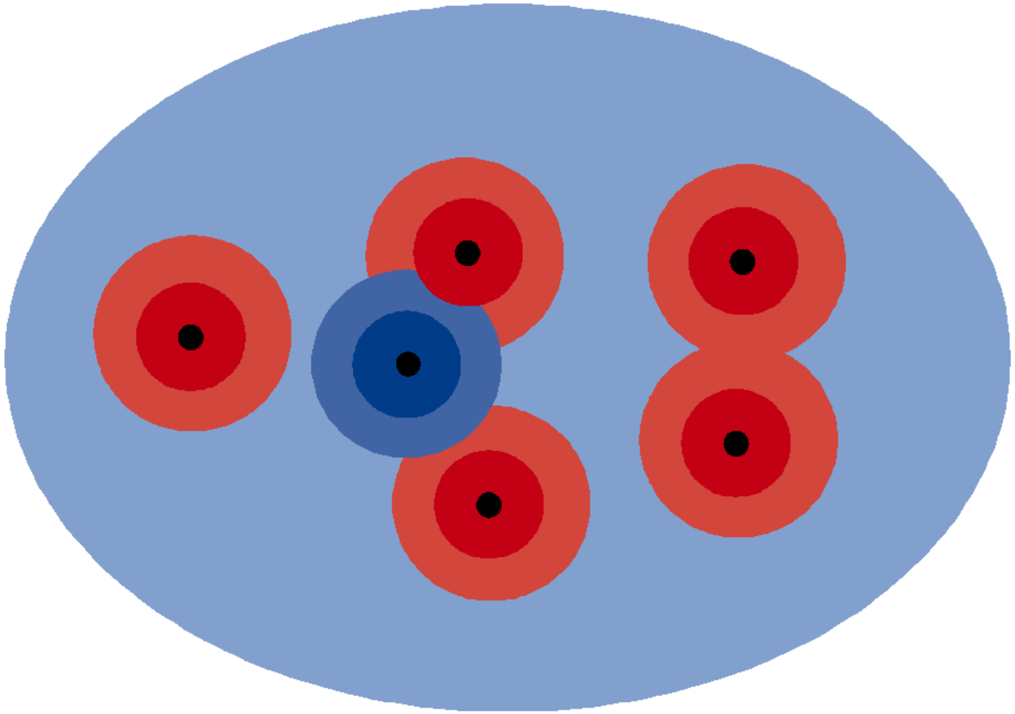


- for each $t \in T$, perform backward search up to the top level,
  store search space entries $(t, u, d(u,t))$
- arrange search spaces: create a bucket for each u
- for each $s \in S$, perform forward search up to and including the top level,
  at each node u, scan all entries $(t, u, d(u,t))$ and
  compute $d(s,u) + d(u,t)$, update $D[s,t]$

Asymmetry

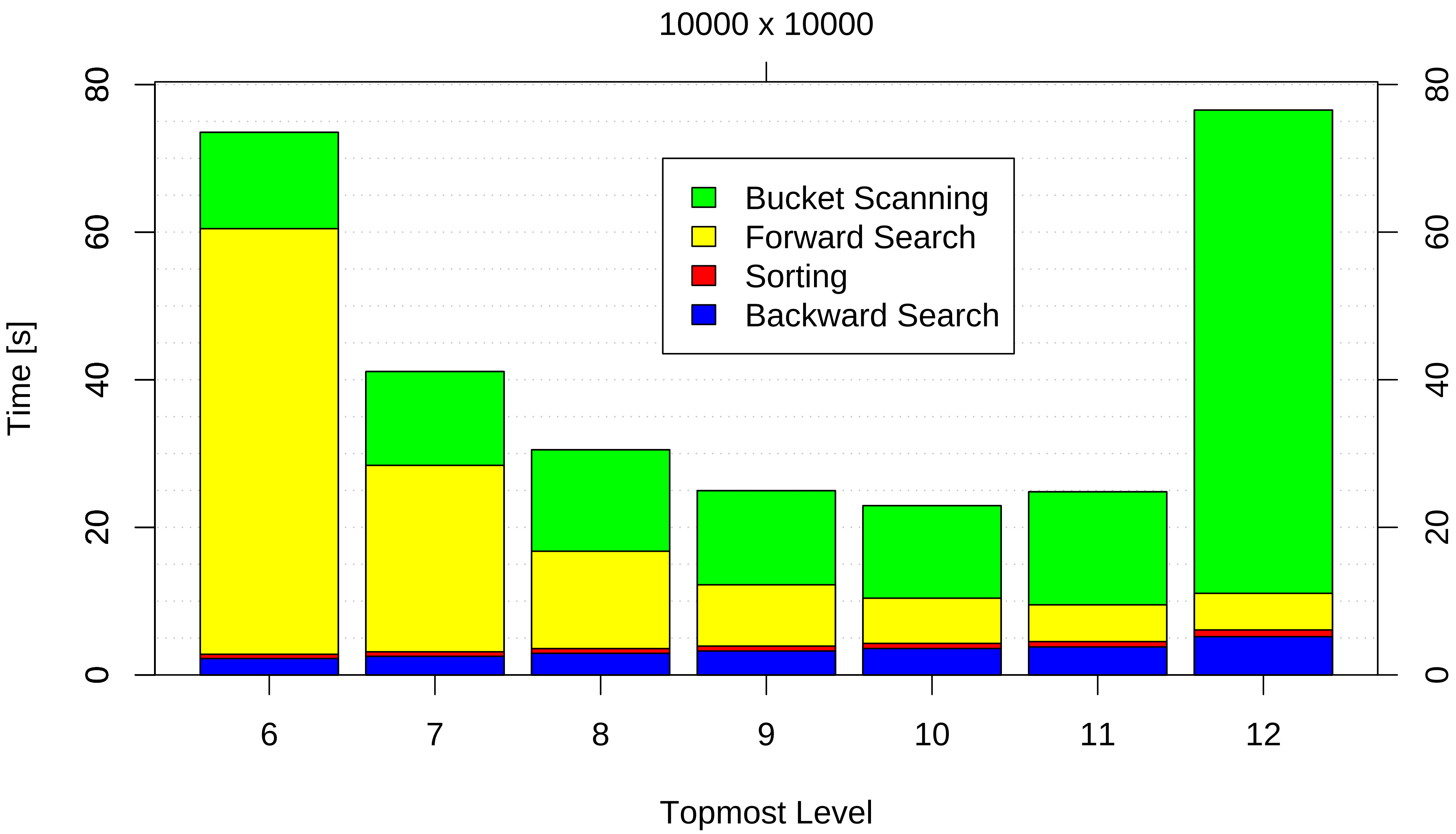
for large distance tables, most time spent on bucket scanning

Solution: use less levels ⇝ strengthen the asymmetry



- backward search spaces get smaller ⇝ less bucket entries
- forward search spaces get bigger

Experiments



Summary

- **efficient static approach**
  - fast preprocessing / fast queries — 15 min / 0.9 ms
  - outstandingly low memory requirements — 0.7 bytes/node $\rightsquigarrow$ 1.4 ms
- **can handle practically relevant dynamic scenarios**
  - change entire cost function — typically < 2 minutes
  - change a few edge weights
    - ∗ update data structures — 2 – 40 ms per changed edge

    OR
    - ∗ iteratively bypass traffic jams — e.g., 3.6 ms in case of 100 traffic jams
- **extensible to many-to-many** — 23 s for 10 000 $\times$ 10 000 table

---

numbers refer to the Western European road network with 18 million nodes

Future Work

- find simpler / better ways to determine the node sets $S_1 \supseteq S_2 \supseteq S_3 \ldots$ (work in progress)
- handle a massive amount of updates

- deal with time-dependent scenarios

  (where edge weights depend on the time of day)
- allow multi-criteria optimisations