Route Planning in Road Networks

– simple, flexible, efficient –

Peter Sanders **Dominik Schultes**

Institut für Theoretische Informatik – Algorithmik II

Universität Karlsruhe (TH)

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

Utrecht, May 21, 2008

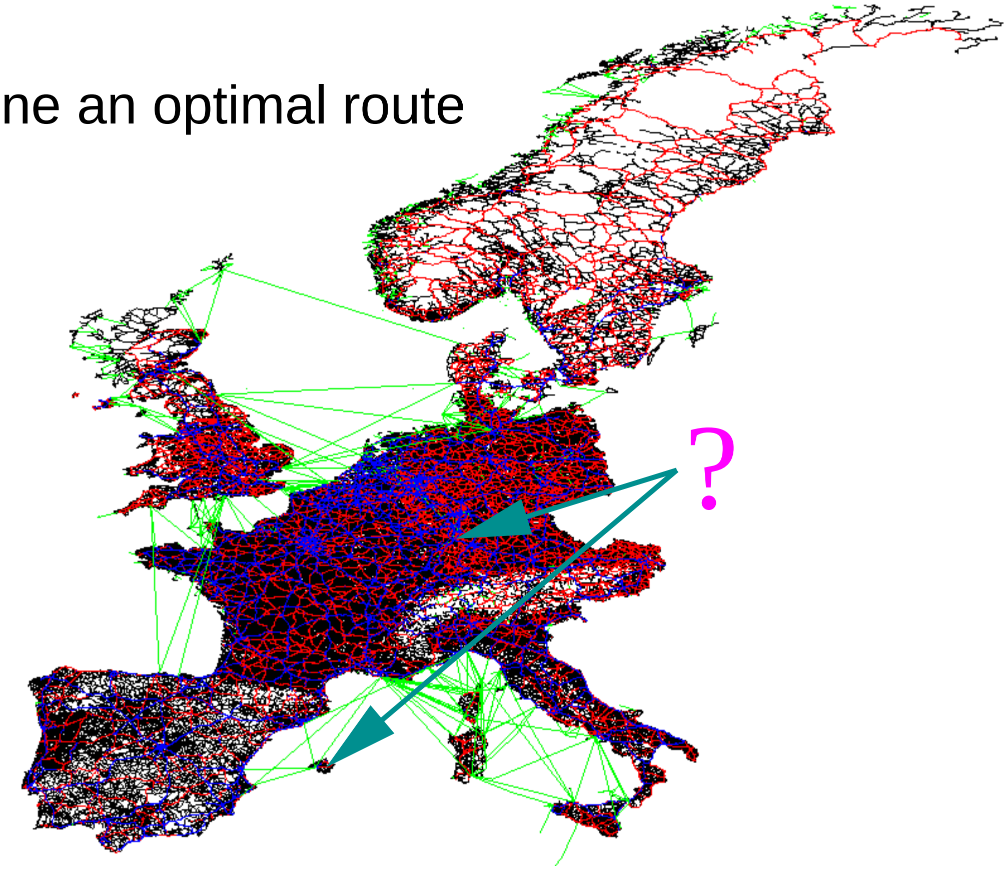
Route Planning

Task:

In a given road network, determine an optimal route

from a given source

to a given target



Applications:

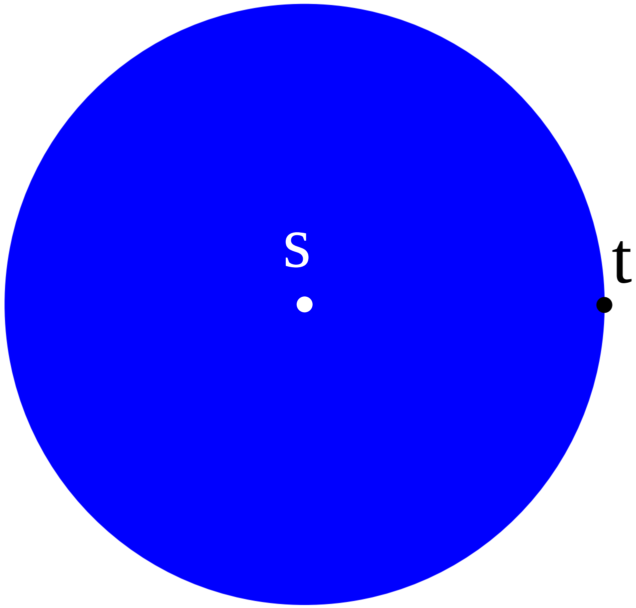
- route planning systems in the internet, car navigation systems,
- traffic simulation, logistics optimisation

DIJKSTRA's Algorithm

the classic solution [1959]

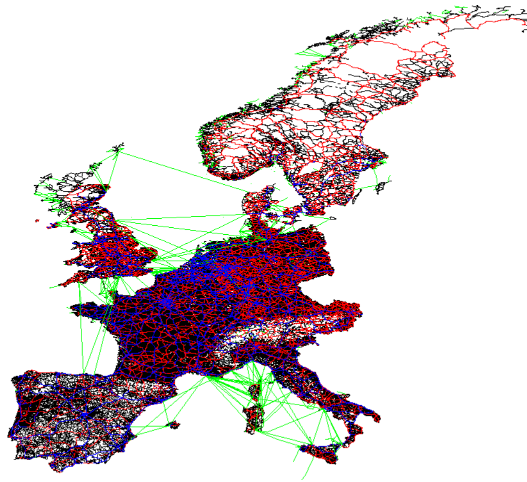
$O(n \log n + m)$ (with Fibonacci heaps)

Dijkstra



not practicable

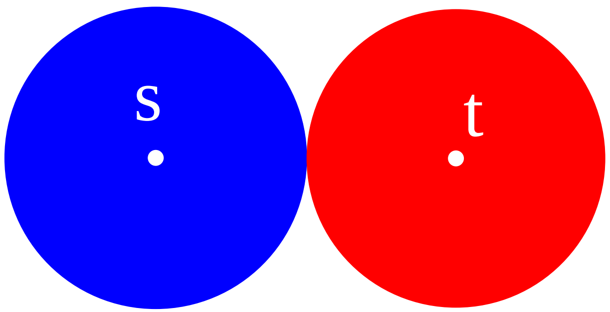
for large graphs



(e.g. European road network:

≈ 18 000 000 nodes)

bidirectional Dijkstra



improves the running time,

but still too slow

Speedup Techniques

⇝ general solution slow — Dijkstra: $\Omega(n+m)$

but:

for special cases there is still **hope** — e.g., for road networks

- additional data — e.g., node coordinates
- preprocessing ⇝ auxiliary data — e.g., ‘signposts’
- special properties of the graph — e.g., planar, hierarchical

Goals

Primary Goals:

- fast query times
- provably optimal results

Secondary Goals:

- fast preprocessing / deal with large networks
- low space consumption
- fast update operations
- simple

Highway Hierarchies

Transit−Node Routing
very fast queries
[DIMACS 06, ALENEX 07, Science 07]

HH Star
goal−directed
[DIMACS 06]

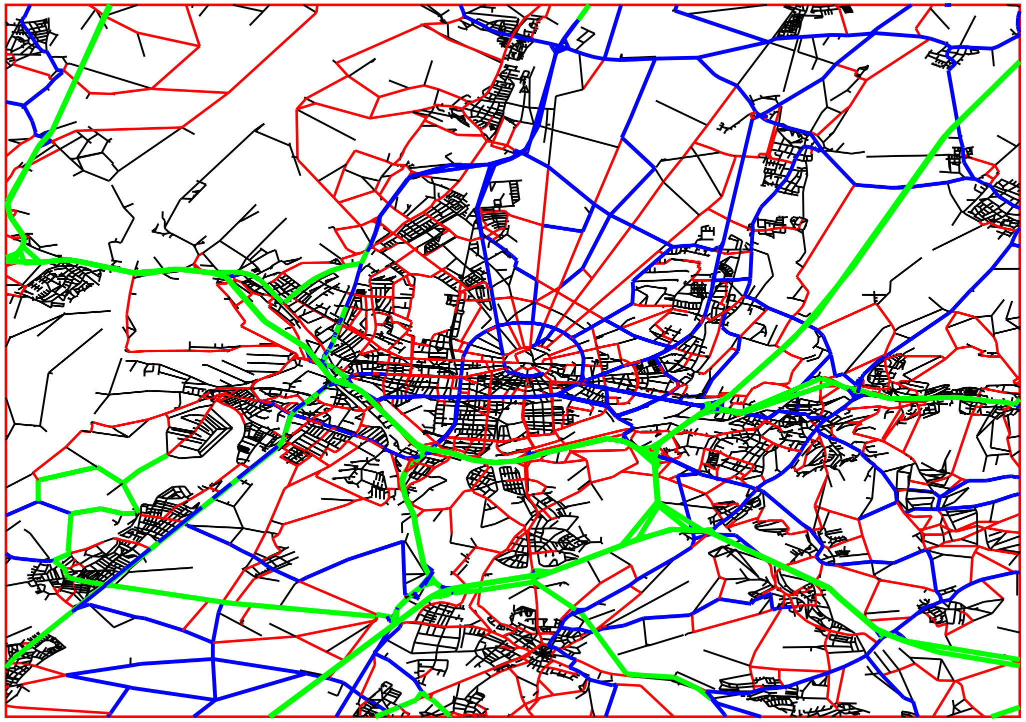
Highway Hierarchies
foundation
[ESA 05, ESA 06]

Many−to−Many
compute distance tables
[ALENEX 07]

Hwy−Node Routing
allow edge weight changes
[WEA 07]

Highway Hierarchies

- determine a hierarchy of highway networks /
- classify roads by 'importance'



bidirectional query algorithm:

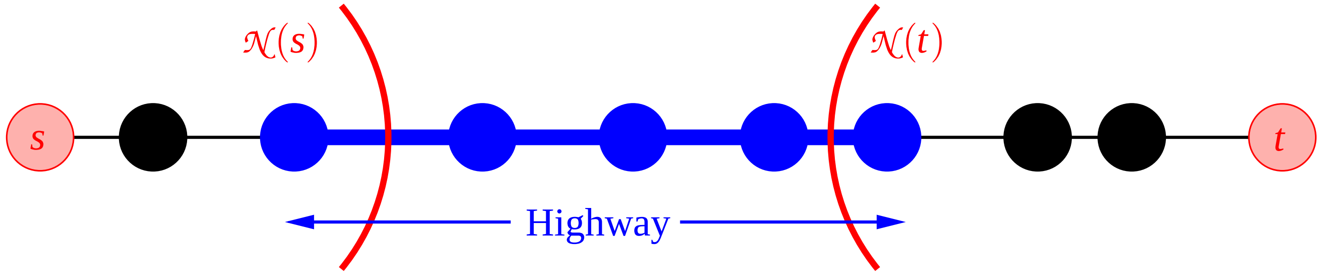
with increasing distance from source/target:

consider only 'more important' roads

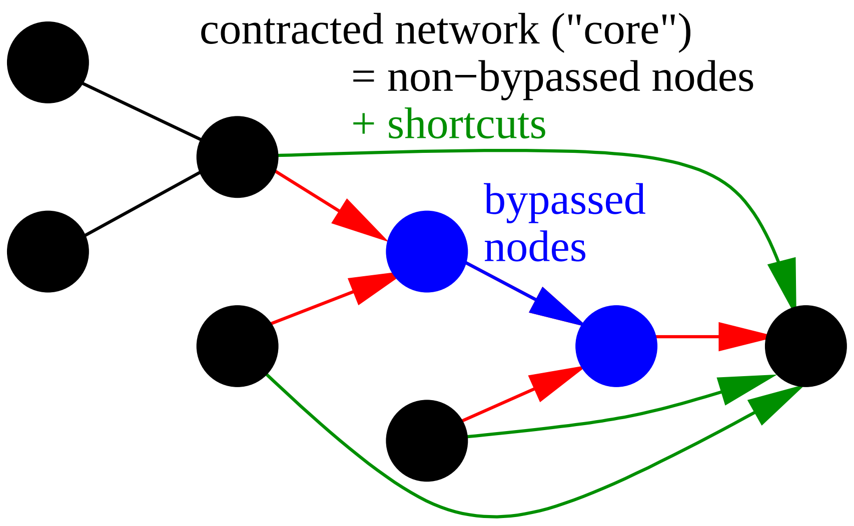
Highway Hierarchies

Construction: iteratively alternate between

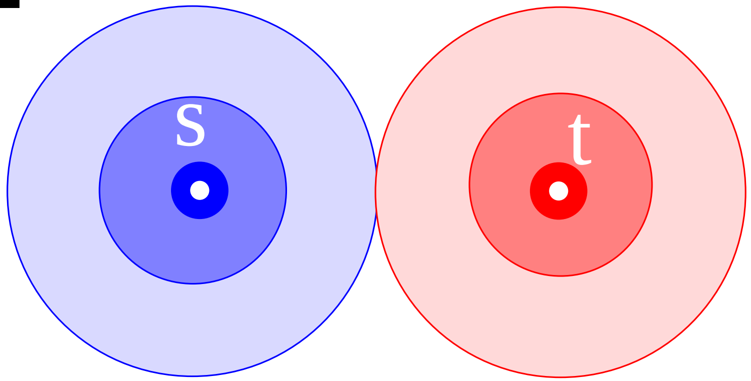
- removal of edges that only appear on shortest paths close to source or target



- removal of low degree nodes



Highway Hierarchies



- foundation for our other methods
- directly allows point-to-point queries
- 13 min preprocessing
- 0.61 ms to determine the path length
- (0.80 ms to determine path description)
- reasonable space consumption (48 bytes/node)

  can be reduced to 17 bytes/node

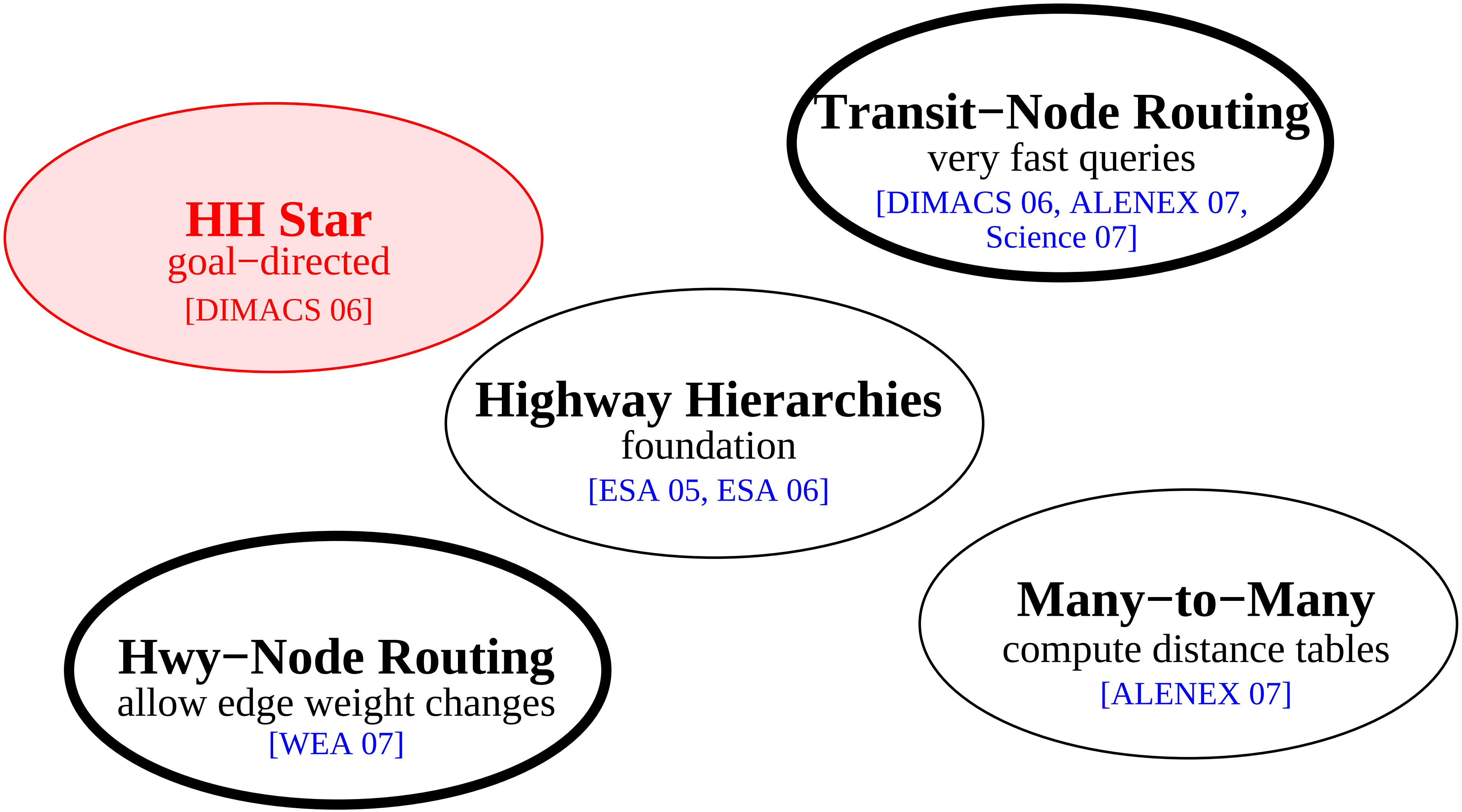
} Europe

≈ 18 000 000 nodes

AMD Opteron 2.0 GHz

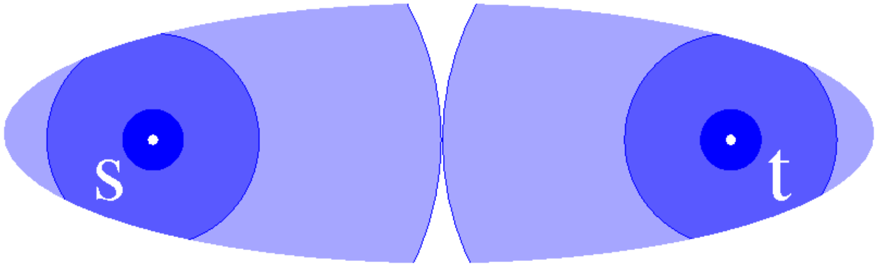
Sanders, Schultes. ESA 2005, 2006.

Highway Hierarchies Star



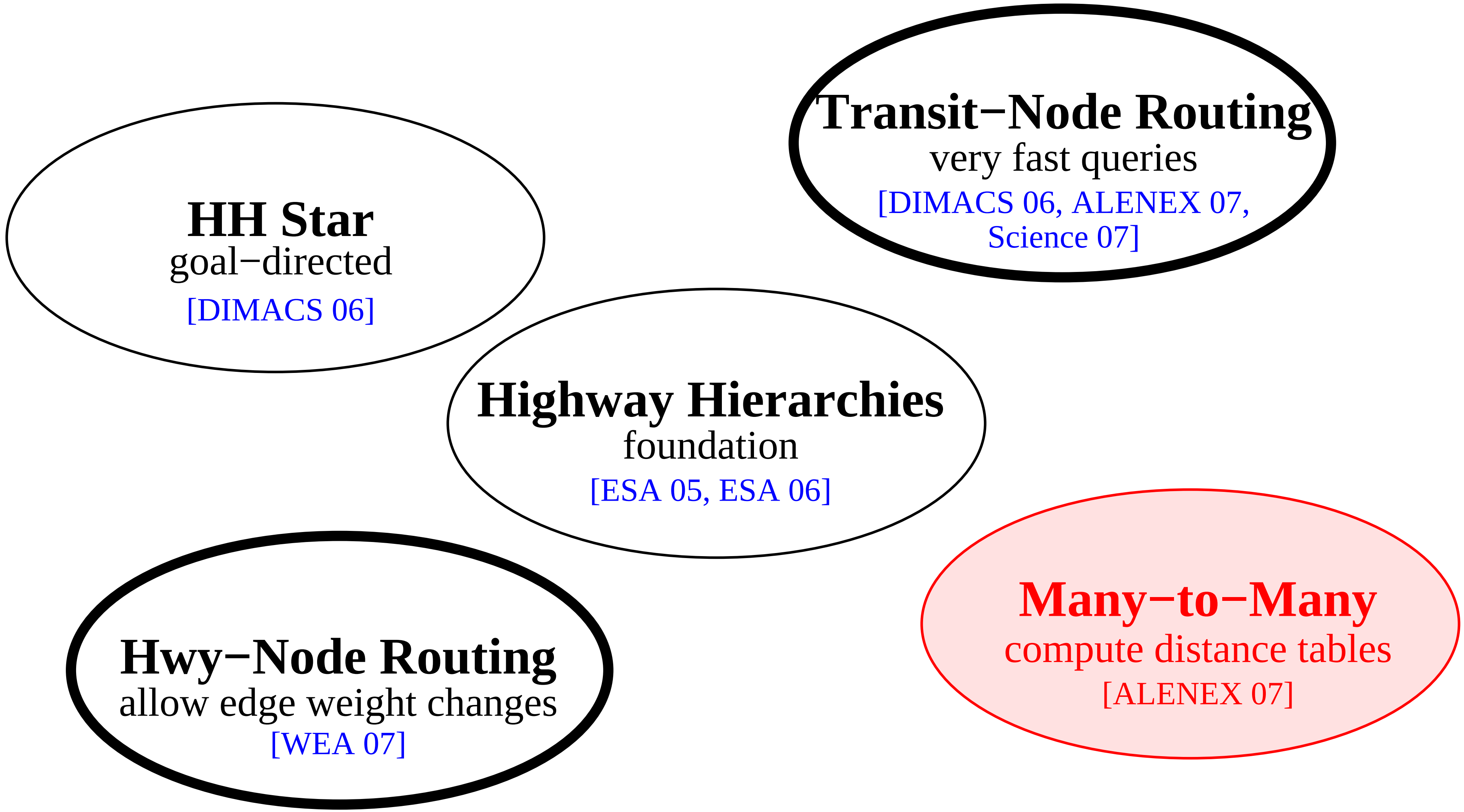
Highway Hierarchies Star

- combination of highway hierarchies with goal-directed search
- slightly reduced query times (0.49 ms)
- more effective
  - for approximate queries or
  - when a distance metric instead of a travel time metric is used

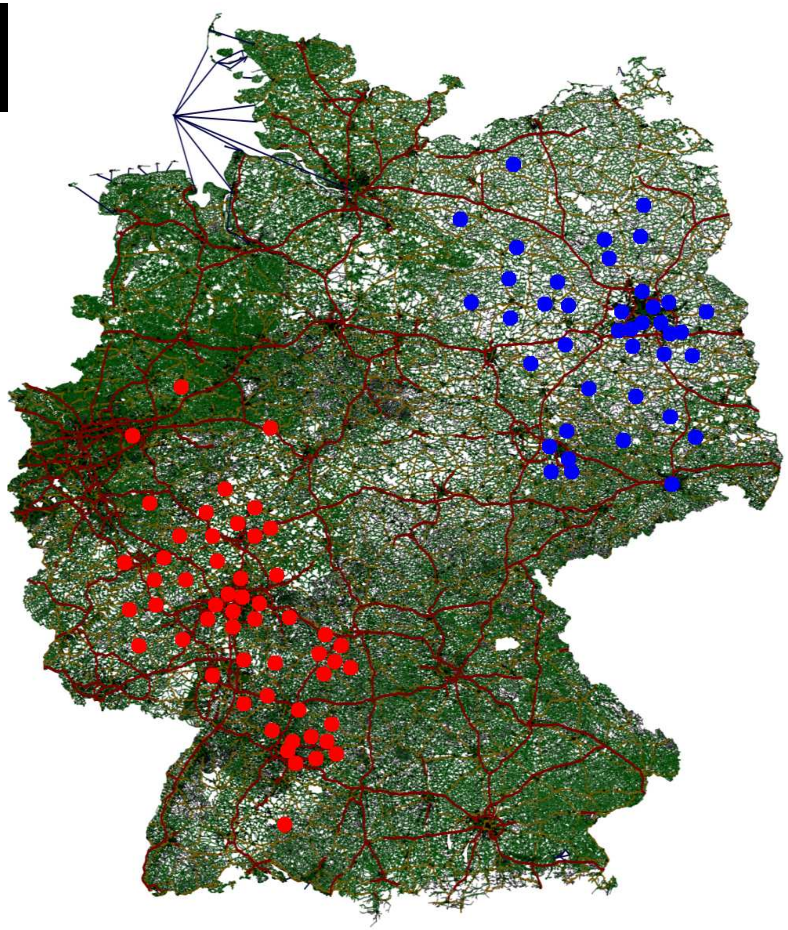


Delling, Sanders, Schultes, Wagner. DIMACS Challenge 2006.

Many-to-Many


Transit−Node Routing
very fast queries
[DIMACS 06, ALENEX 07, Science 07]
HH Star
goal−directed
[DIMACS 06]
Highway Hierarchies
foundation
[ESA 05, ESA 06]
Hwy−Node Routing
allow edge weight changes
[WEA 07]
Many−to−Many
compute distance tables
[ALENEX 07]

Many-to-Many



Given:

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

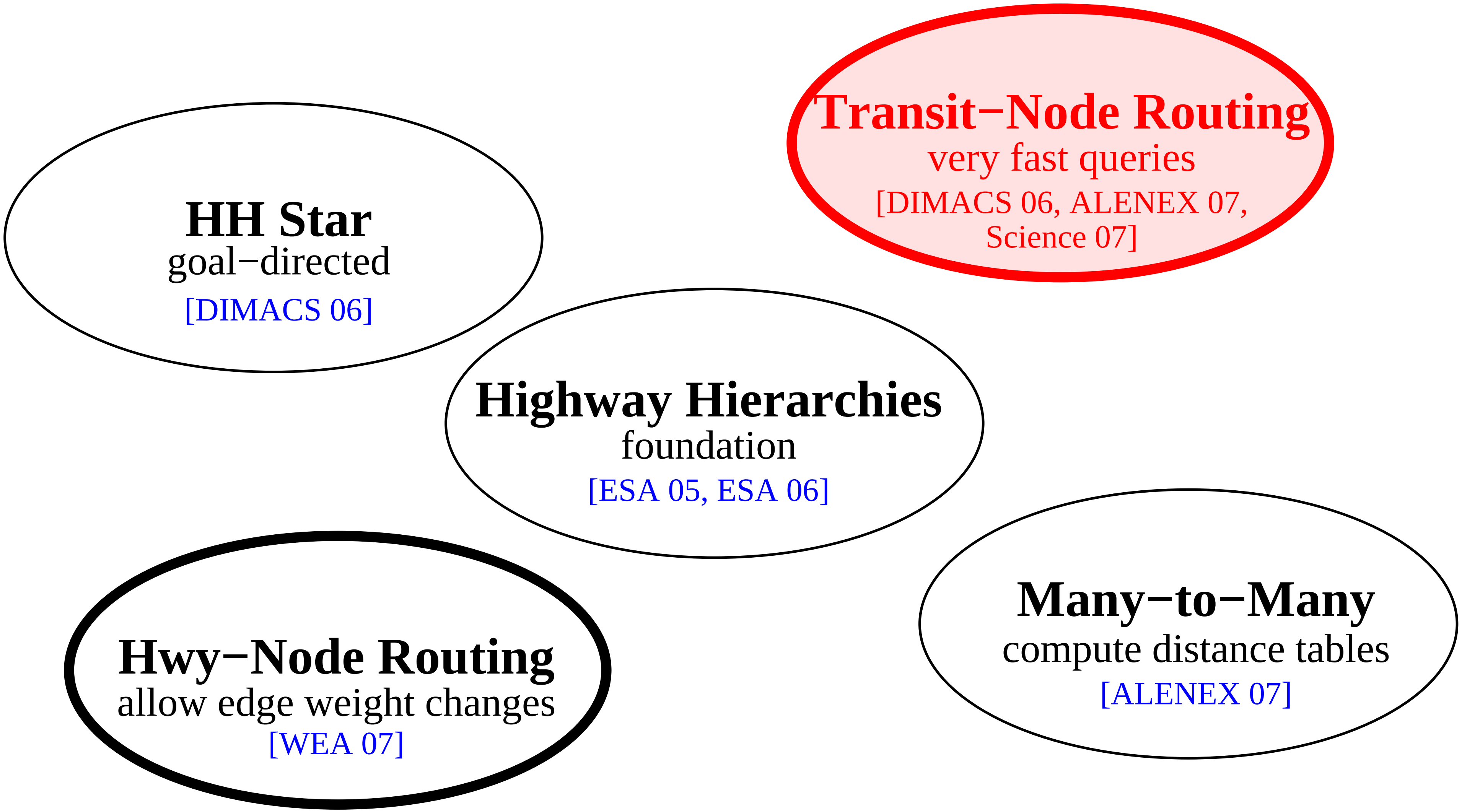
Task: compute $|S| \times |T|$ distance table

containing the shortest path distances

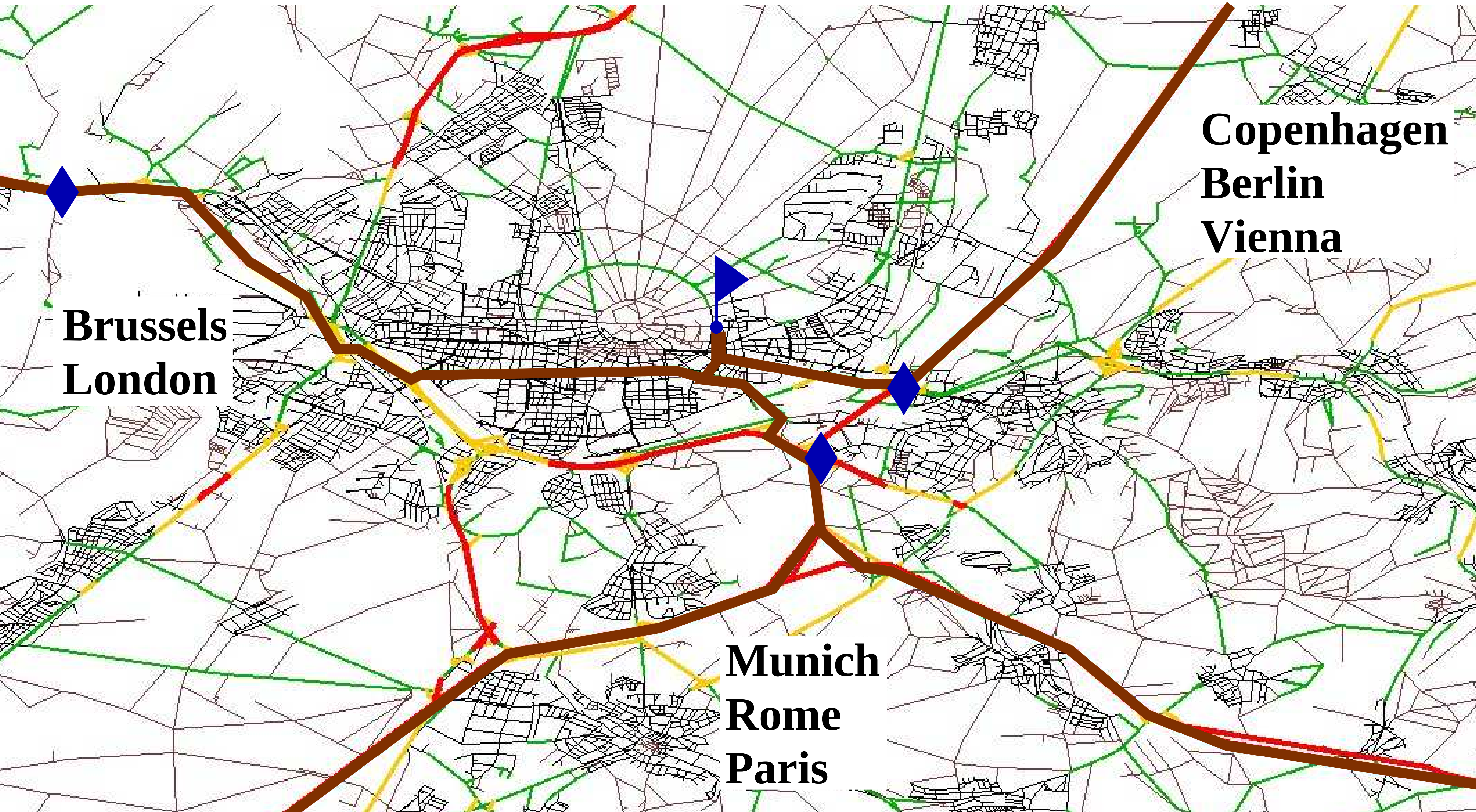
- e.g., $10\,000 \times 10\,000$ table in 23 seconds

Knopp, Sanders, Schultes, Schulz, Wagner. ALENEX 2007.

Transit-Node Routing



Motivation


Copenhagen
Berlin
Vienna
Brussels
London
Munich
Rome
Paris

Observations

1. For long-distance travel: leave current location

via one of only a few ‘important’ traffic junctions,

called **access points**

($\leadsto$ store all access points for each node) [$\approx$ 10 per node]

---

2. Each access point is relevant for several nodes. $\leadsto$

union of the access points of all nodes is small,

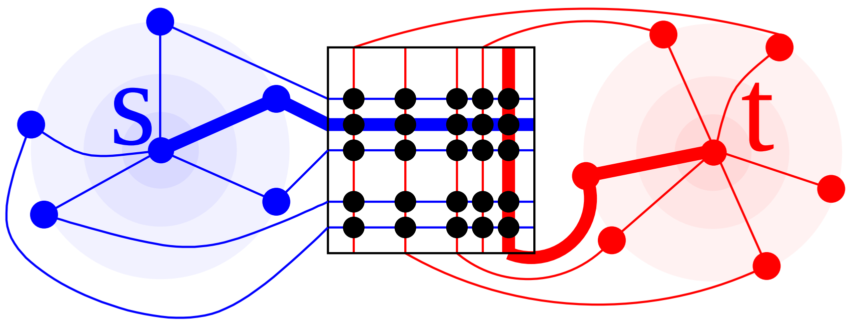
called **transit-node set**

($\leadsto$ store the distances between all transit-node pairs) [$\approx 10\,000^2$ distances]

Transit-Node Routing

Preprocessing:

- identify transit-node set $\mathcal{T} \subseteq V$
- compute complete $|\mathcal{T}| \times |\mathcal{T}|$ distance table
- for each node: identify its access points (mapping $A : V \to 2^{\mathcal{T}}$), store the distances



Query (source s and target t given): compute

$$d_{\text{top}}(s,t) := \min\{d(s,u) + d(u,v) + d(v,t) : u \in A(s), v \in A(t)\}$$

Transit-Node Routing

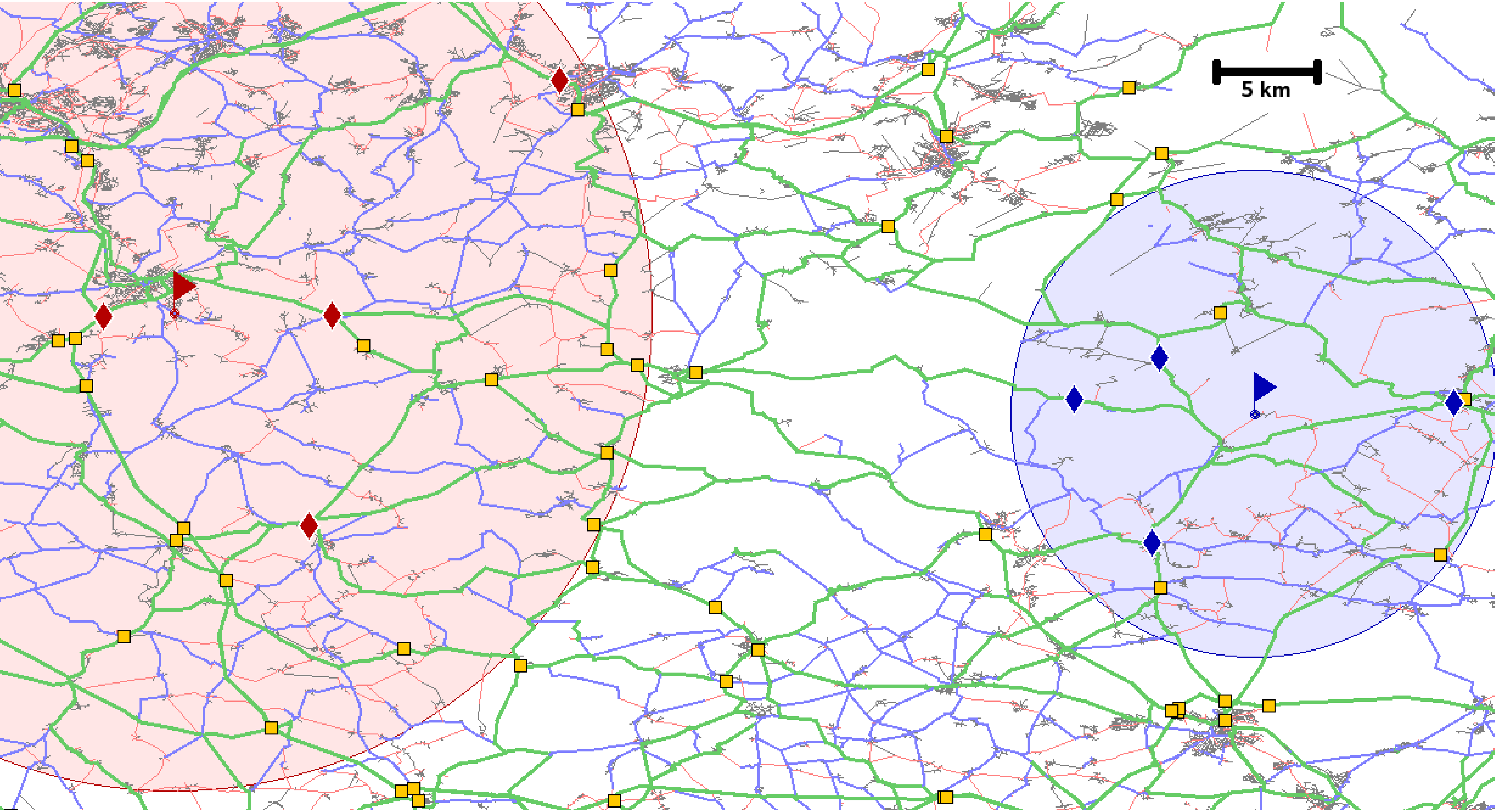
Locality Filter:

local cases must be filtered ($\rightsquigarrow$ special treatment)

$L : V \times V \rightarrow \{\mathsf{true}, \mathsf{false}\}$

$\neg L(s,t)$ implies $d(s,t) = d_{\text{top}}(s,t)$

Example



Experimental Results

- *very* fast queries

  (down to 4 μs, > 1 000 000 times faster than DIJKSTRA)

- more preprocessing time (1:15 h) and space (247 bytes/node)

- winner of the 9th DIMACS Implementation Challenge 2006

- Scientific American 50 Award 2007

Sanders, Schultes. DIMACS Challenge 2006.

Bast, Funke, Sanders, Schultes. Science, 2007.

Bast, Funke, Matijevic, Sanders, Schultes. ALENEX 2007.

c't

DIE WELT

Süddeutsche Zeitung

COMPUTER ZEITUNG

Open Questions

- How to determine the transit nodes?
- How to determine the access points efficiently?
- How to determine the locality filter?
- How to handle local queries?

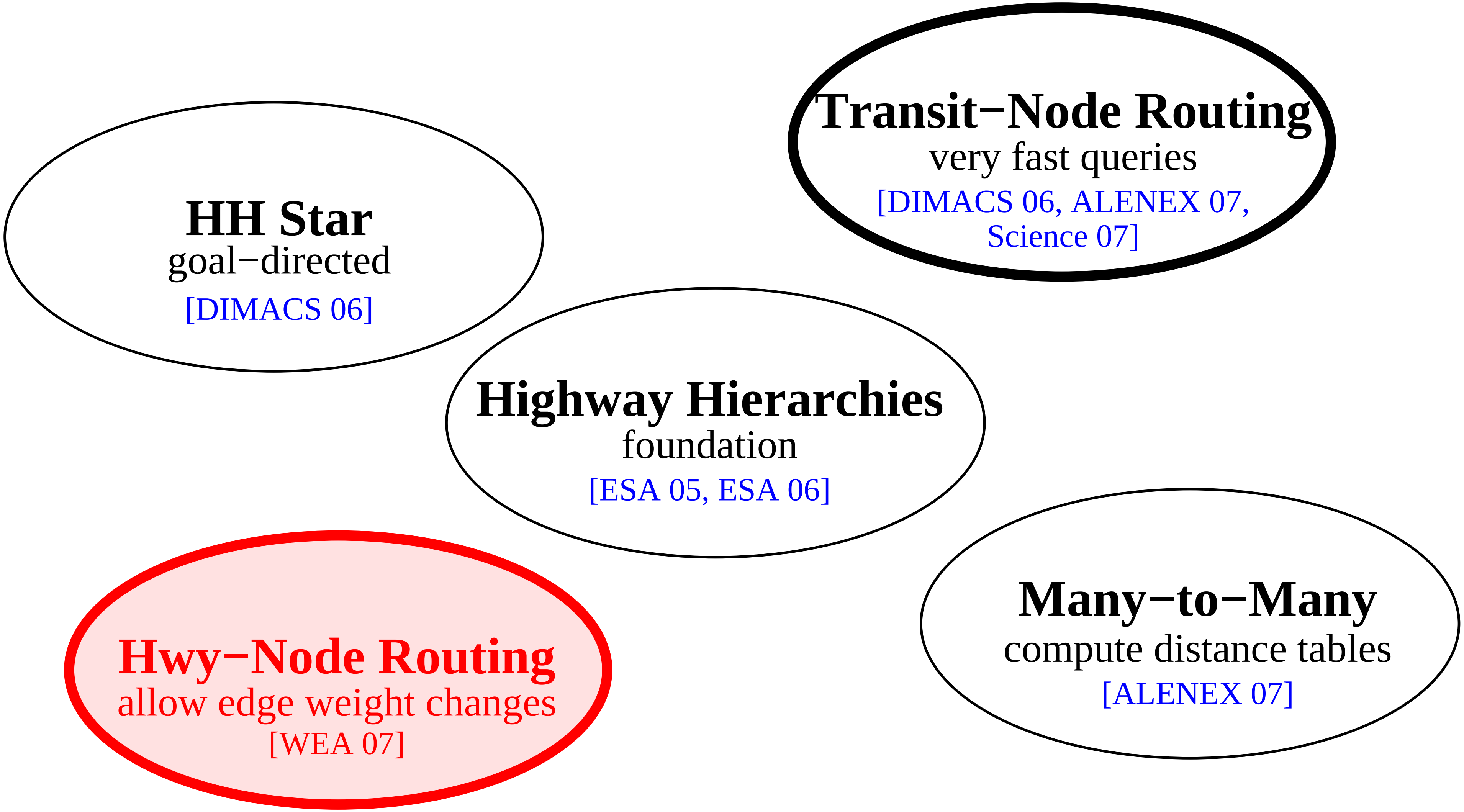
Open Questions

- How to determine the transit nodes?
- How to determine the access points efficiently?
- How to determine the locality filter?
- How to handle local queries?

Answer:

- Use other route planning techniques!

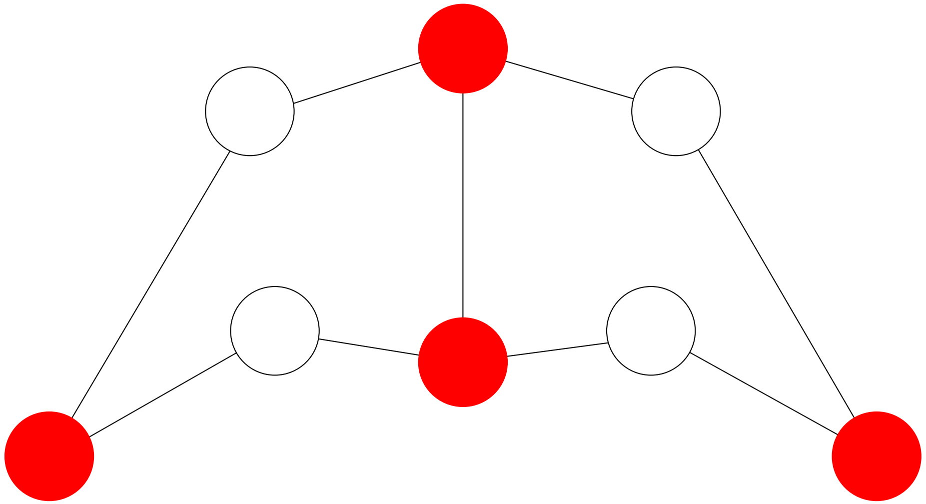
Highway-Node Routing



Overlay Graph: Definition

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

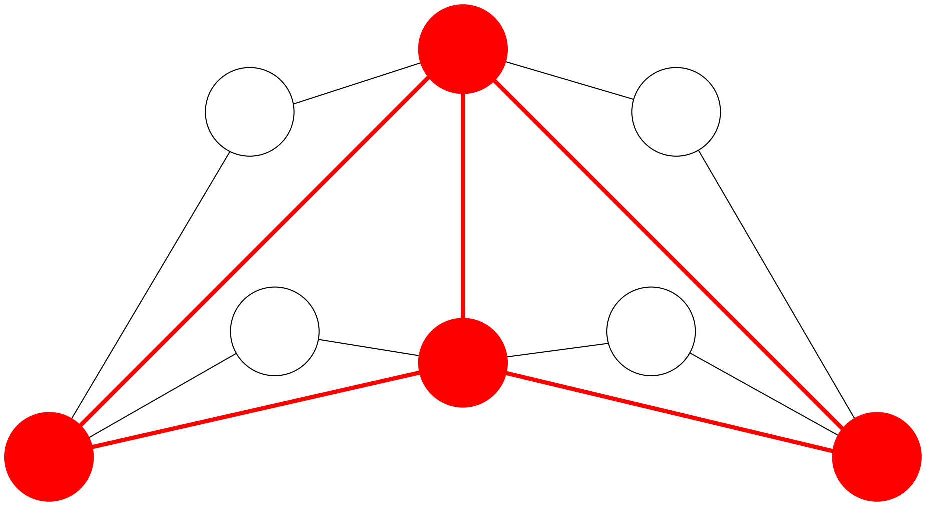
- graph $G = (V, E)$ is given
- select node subset $S \subseteq V$



Overlay Graph: Definition

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

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

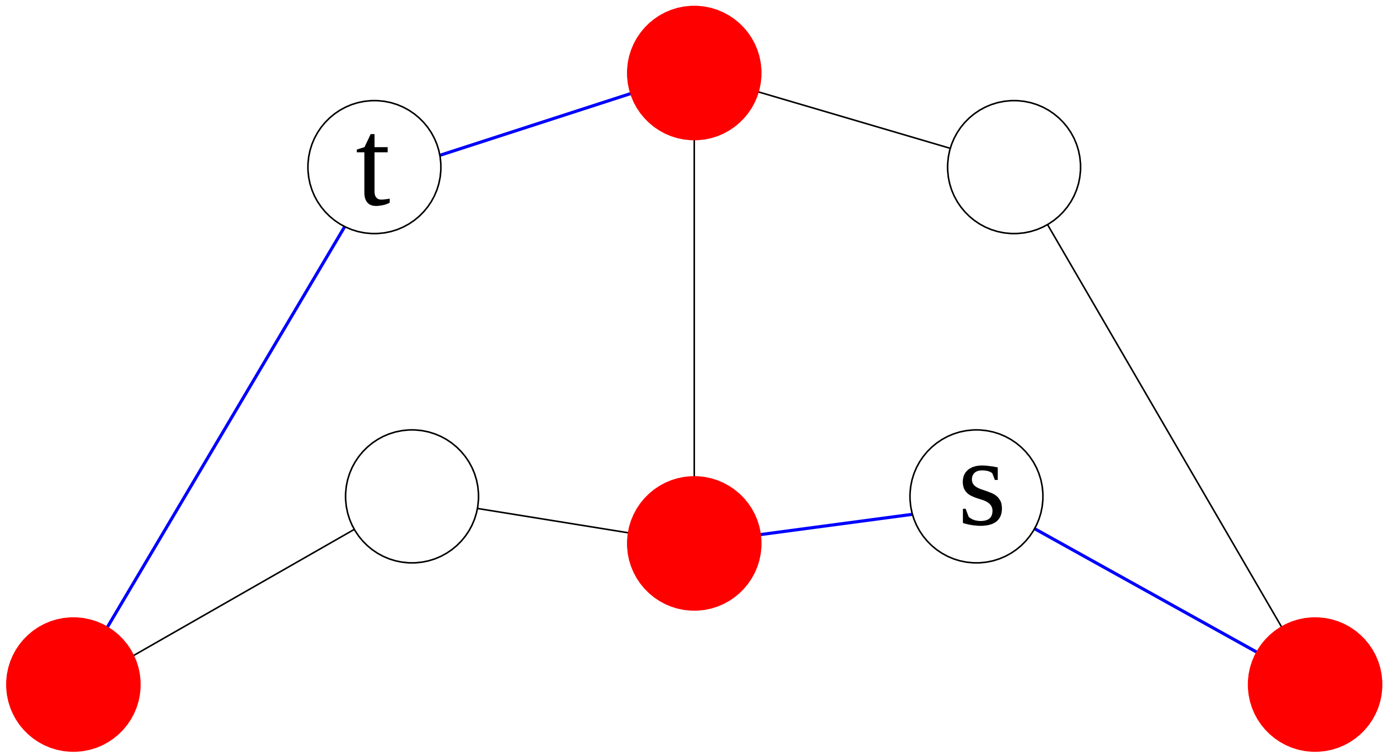


- overlay graph $G' := (S, E')$

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

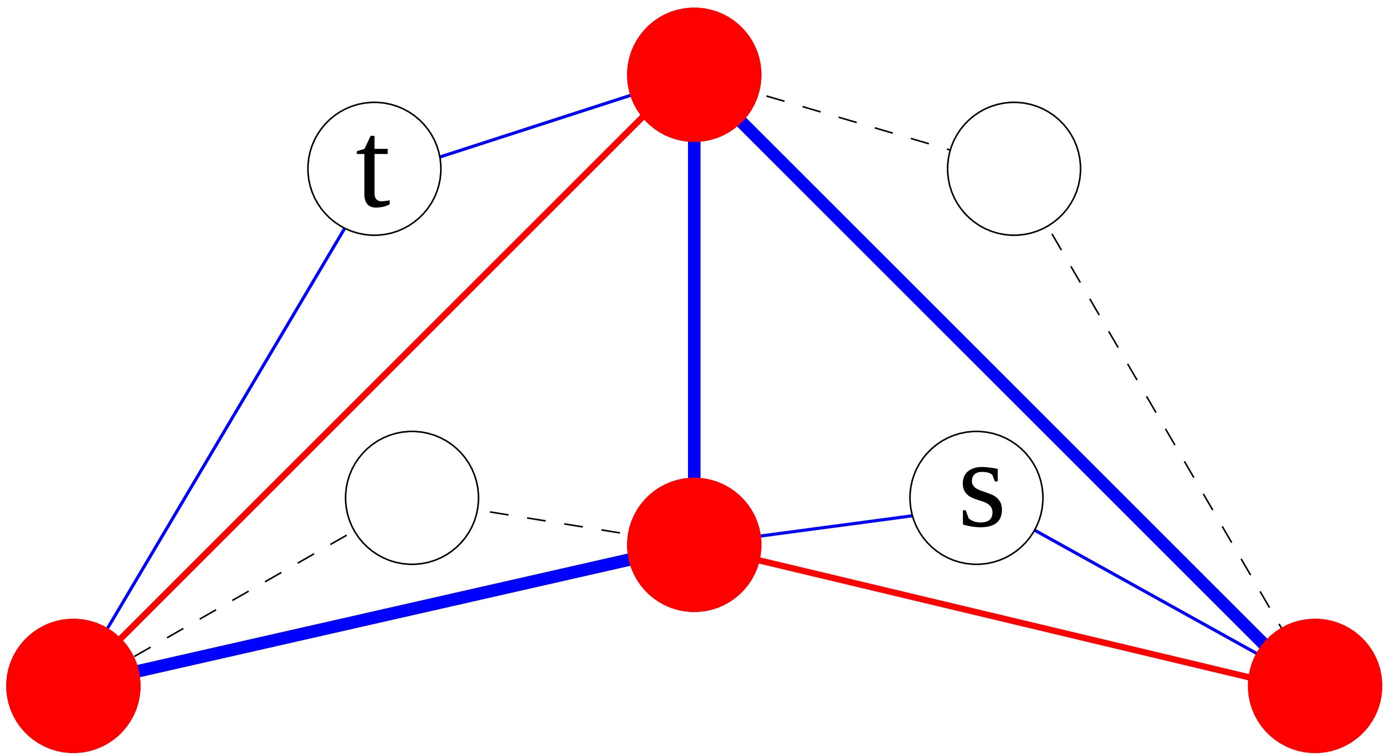
Query: Intuition

- 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'



Highway-Node Routing

- use overlay graph concept iteratively
- classify nodes by ‘importance’ using highway hierarchies

  i.e., determine node sets $V =: S_0 \supseteq S_1 \supseteq S_2 \supseteq S_3 \ldots \supseteq S_L$ 13 min

  (crucial distinction from [Holzer, Schulz, Wagner, Weihe, Zaroliagis])
- construct multi-level overlay graph 2 min

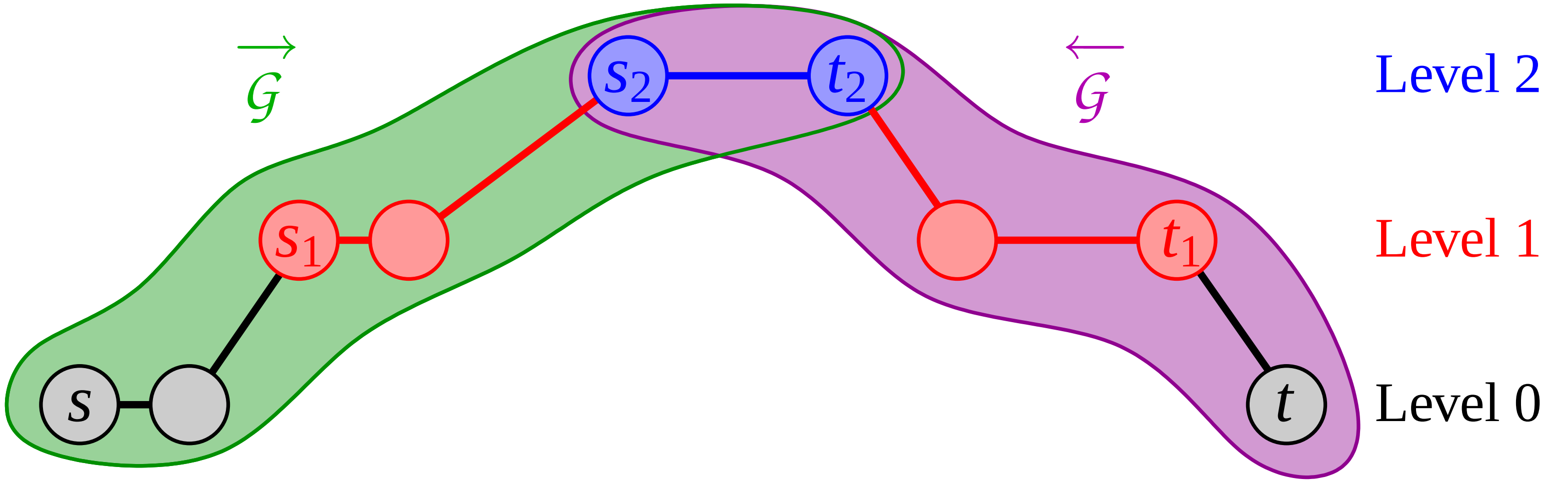
  $G_0 = G = (V, E), G_1 = (S_1, E_1), G_2 = (S_2, E_2), \ldots, G_L = (S_L, E_L)$

  (advanced techniques needed)

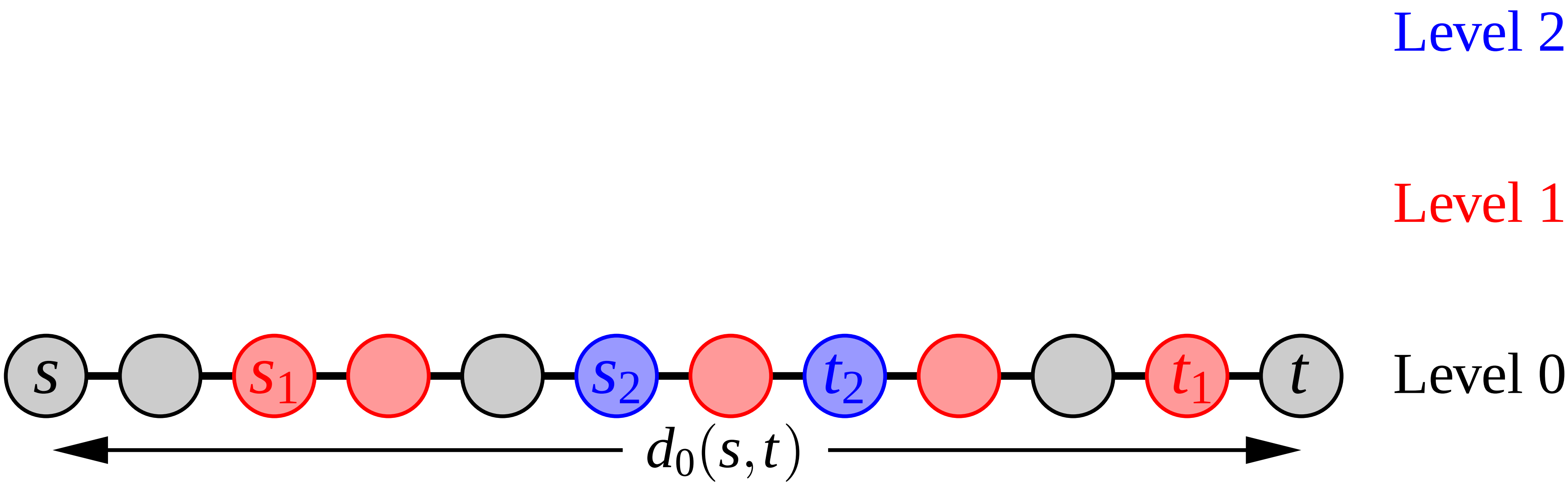
Schultes, Sanders. WEA 2007.

Query Algorithm

- 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 $\overleftarrow{\mathcal{G}}$

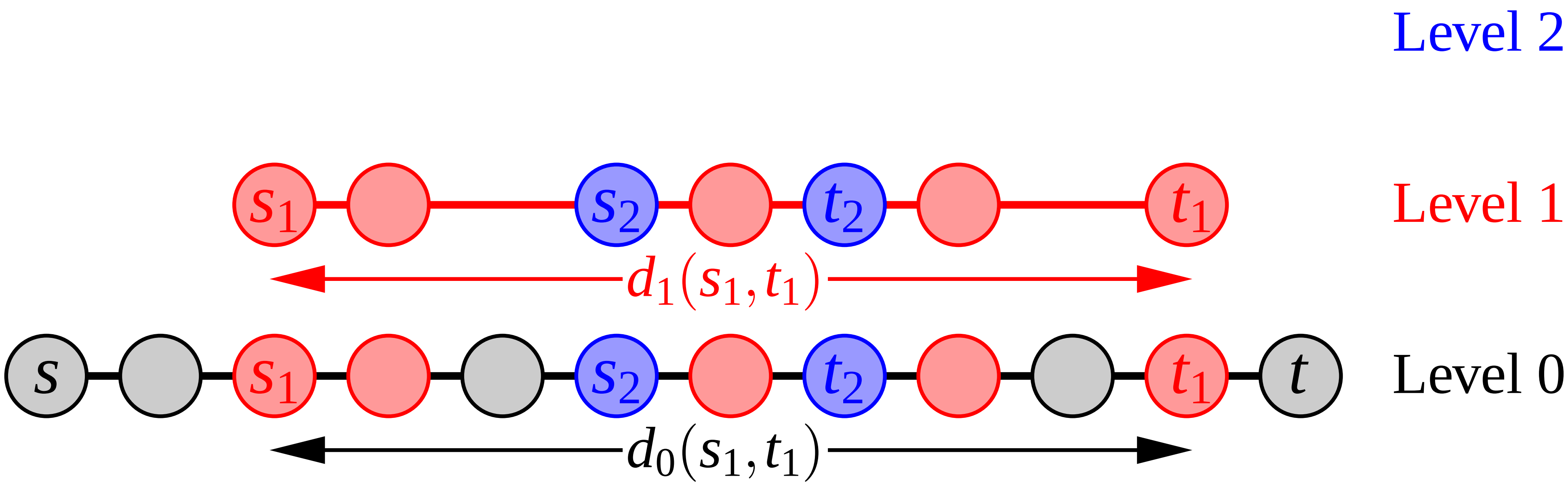


Proof of Correctness



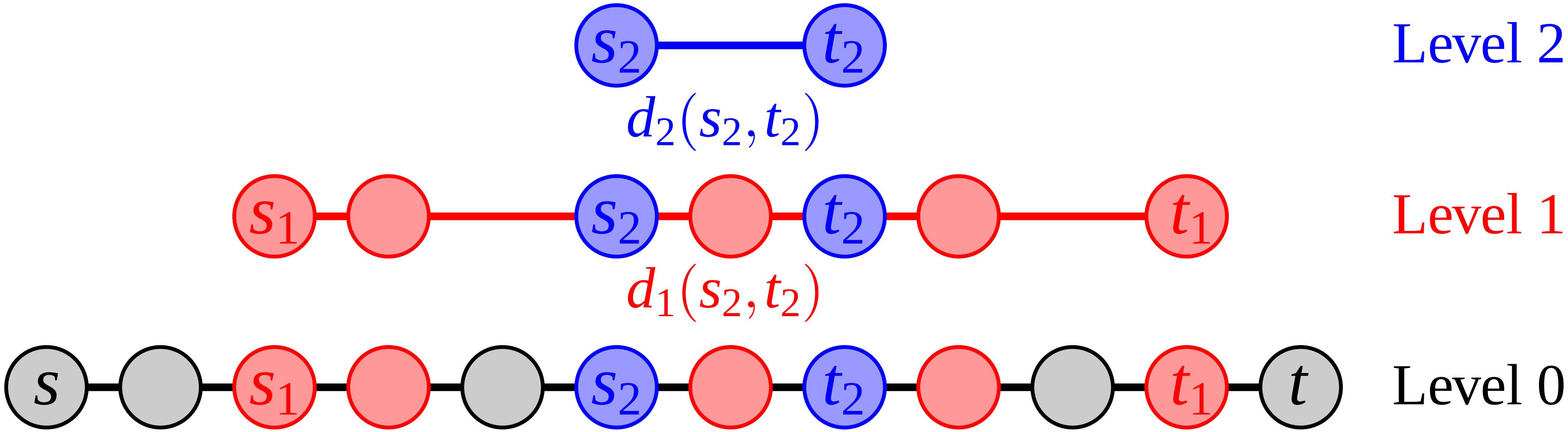
shortest path from s to t in $G = G_0$

Proof of Correctness



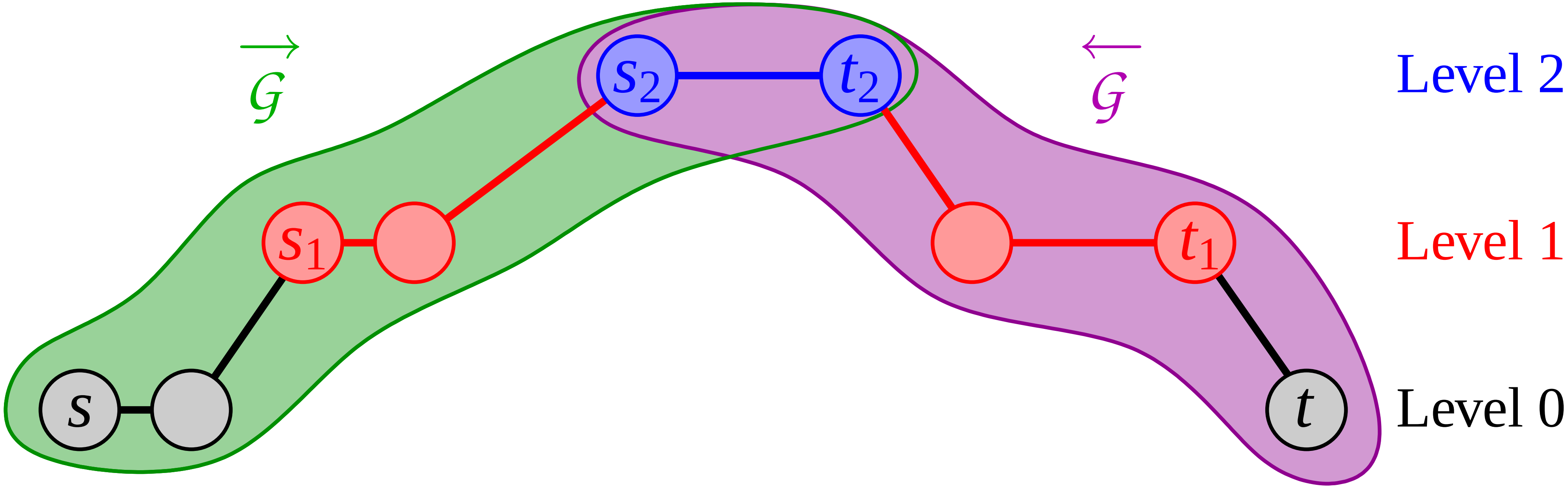
overlay graph G_1 preserves distance from $s_1 \in S_1$ to $t_1 \in S_1$

Proof of Correctness



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

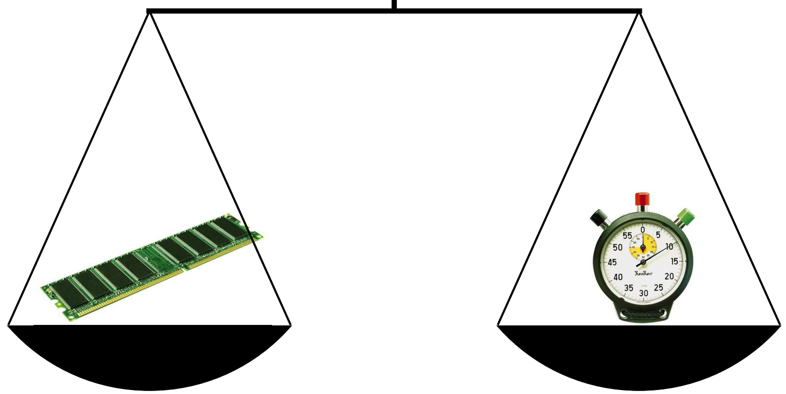
Proof of Correctness



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

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

Memory Consumption / Query Time



different trade-offs

for example:

- 9.5 bytes per node overhead $\longrightarrow$ 0.89 ms

  store complete multi-level overlay graph

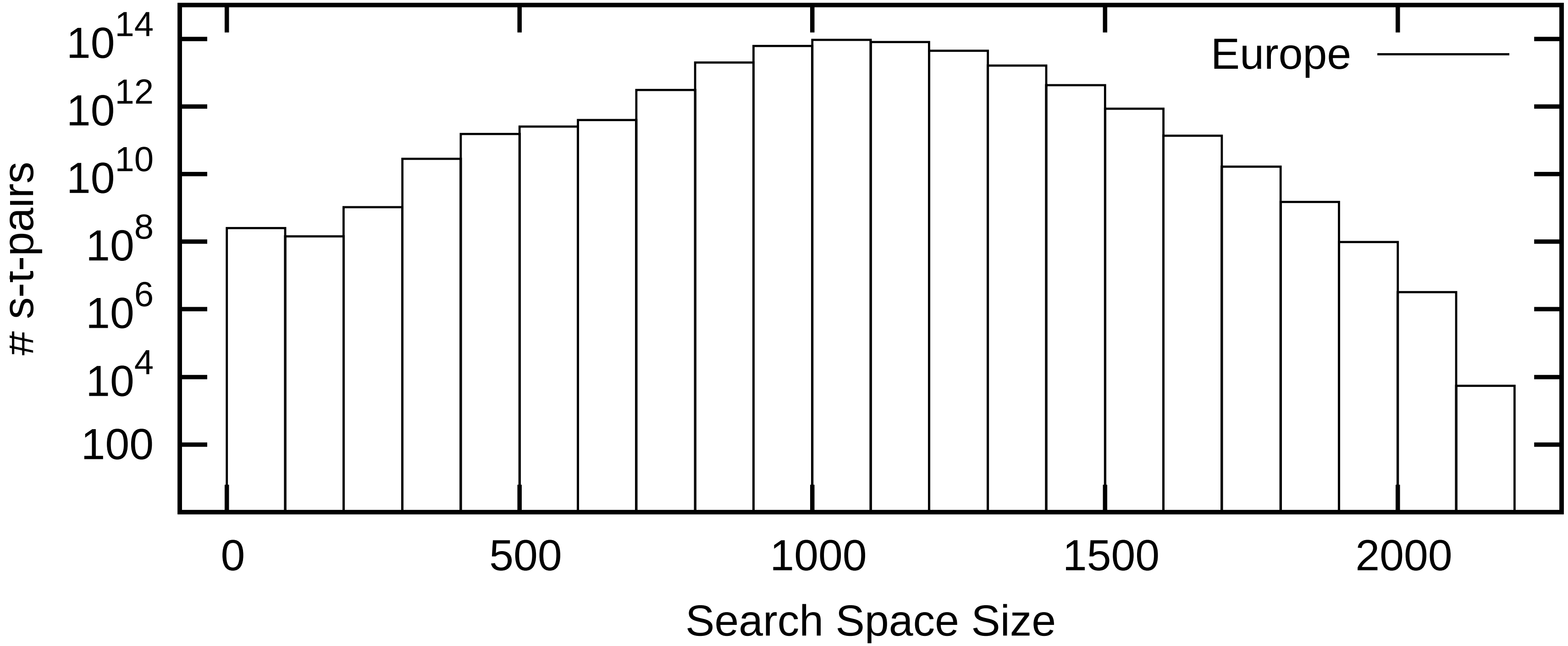
- 0.7 bytes per node overhead $\longrightarrow$ 1.44 ms

  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)

---

query times using the so-called 'stall-on-demand' technique

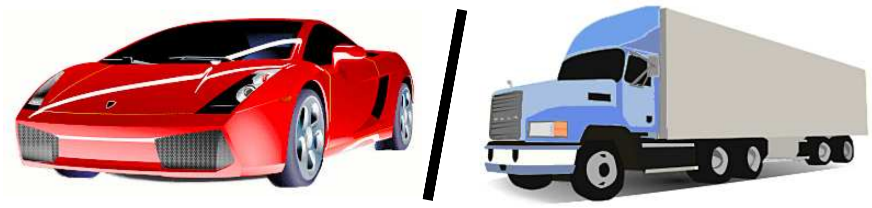
Per-Instance Worst-Case Guarantee



guarantee for Europe: maximum search space size = 2 148 nodes

Dynamic Szenarios

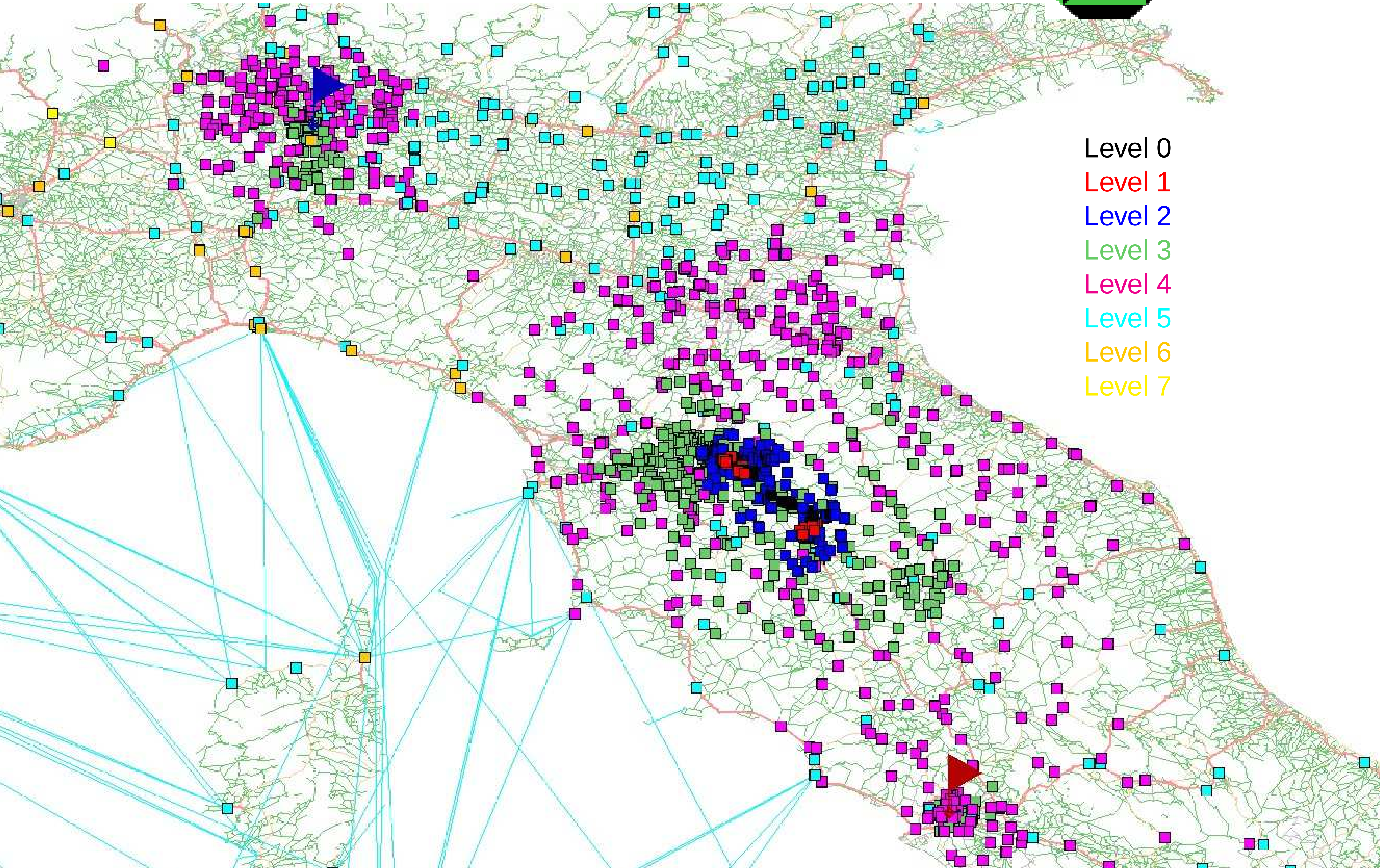
- exchange cost function — typically < 2 min



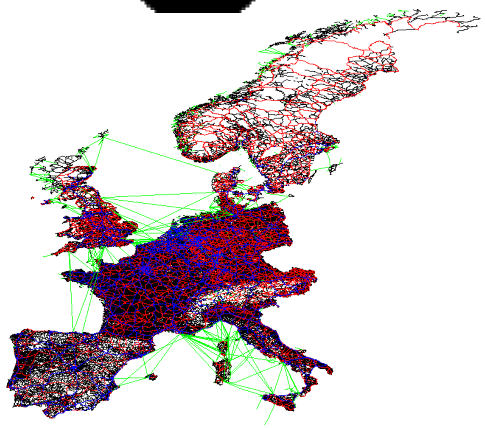
- change a few edge weights
  - update data structures — 2–40 ms per changed edge

  OR

  - bypass the traffic jams — e.g., 3.6 ms in case of 100 traffic jams


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

Summary



- **deal with very large road networks**
- **static point-to-point routing**
  - fastest query times — transit-node routing
  - fastest preprocessing — highway hierarchies
  - lowest memory consumption — highway-node routing
- **dynamic point-to-point routing**
  - exchange cost function
  - change a few edge weights

  } highway-node routing
- **compute distance tables** — many-to-many

Recent Work

concerning **highway-node routing**

- find simpler / better ways to determine the node sets

  $S_1 \supseteq S_2 \supseteq S_3 \ldots$

  [contraction hierarchies]

- parallelise the preprocessing

- implementation for a mobile device

  275 MB to store Europe, < 100 ms query time

Future Work

- handle a massive amount of updates

- deal with time-dependent scenarios

  (where edge weights depend on the time of day)

- allow multi-criteria optimisations