Accurate High-Performance
Route Planning

Peter Sanders **Dominik Schultes**

Institut für Theoretische Informatik – Algorithmik II

Universität Karlsruhe (TH)

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

Gouda, July 11, 2006

How do I get there from here ?

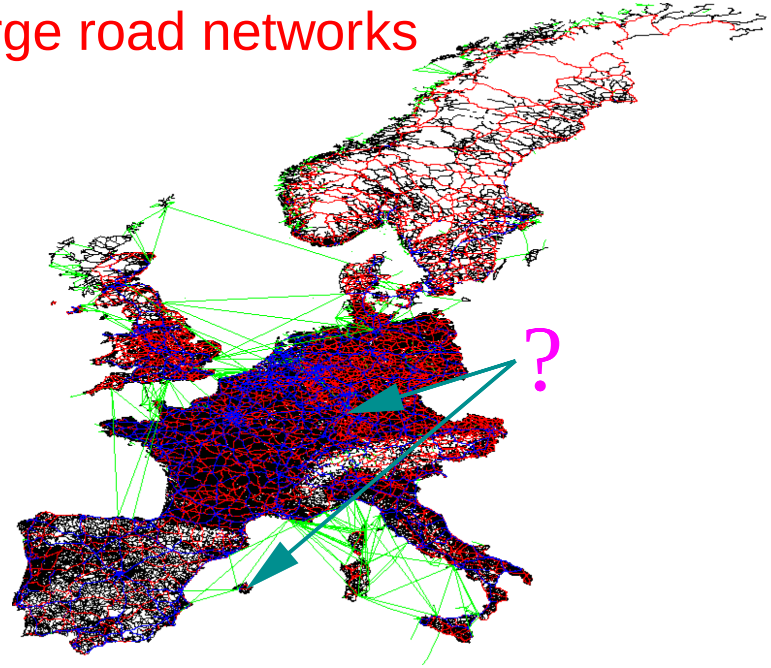
Applications

- route planning systems in the internet (e.g. `www.map24.de`)
- car navigation systems
- . . .

Goals

- exact shortest (i.e. fastest) paths in large road networks
- fast queries
- fast preprocessing
- low space consumption
- scale-invariant,

  i.e., optimised not only for long paths



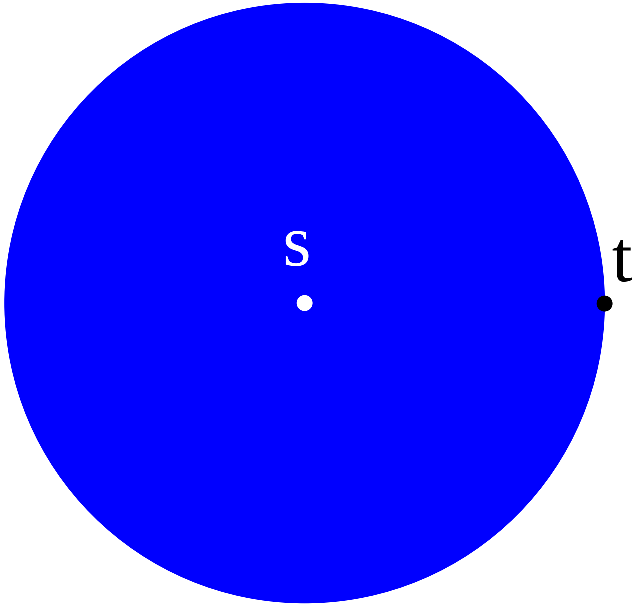
Related Work

	method	query	prepr.	space	scale	source
$\rightarrow$	basic A^*	$-$	$++$	$++$	$+$	[Hart et al. 68]
	bidirected	$-$	$++$	$++$	$+$	[Pohl 71]
$\triangle$	heuristic hwy hier.	$+$	$++$	$+$	$+$	[commercial]
$\triangle$	separator hierarchies	o	?	$-$	$-$	[several groups 02]
$\triangle\!\!\!\rightarrow$	geometric containers	$++$	$--$	$+$	$+$	[Wagner et al. 03]
$\triangle\!\!\!\rightarrow$	bitvectors	$++$	$-$	o	$-$	[Lauther. . . 04]
$\rightarrow$	landmarks	$+$	$++$	$-$	$-$	[Goldberg et al. 04]
$\triangle\!\!\!\rightarrow$	landmarks $+$ reaches	$++$	o	o	o	[Goldberg et al. 06]
$\triangle$	highway hierarchies	$++$	$+$	$+$	$+$	here

$\rightarrow$ direct towards target $\triangle$ exploit hierarchy

DIJKSTRA's Algorithm

Dijkstra



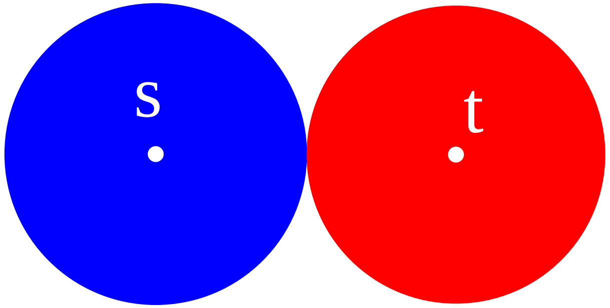
not practicable

for large road networks

(e.g. Western Europe:

≈ 18 000 000 nodes)

bidirectional Dijkstra



improves the running time,

but still too slow

Naive Route Planning

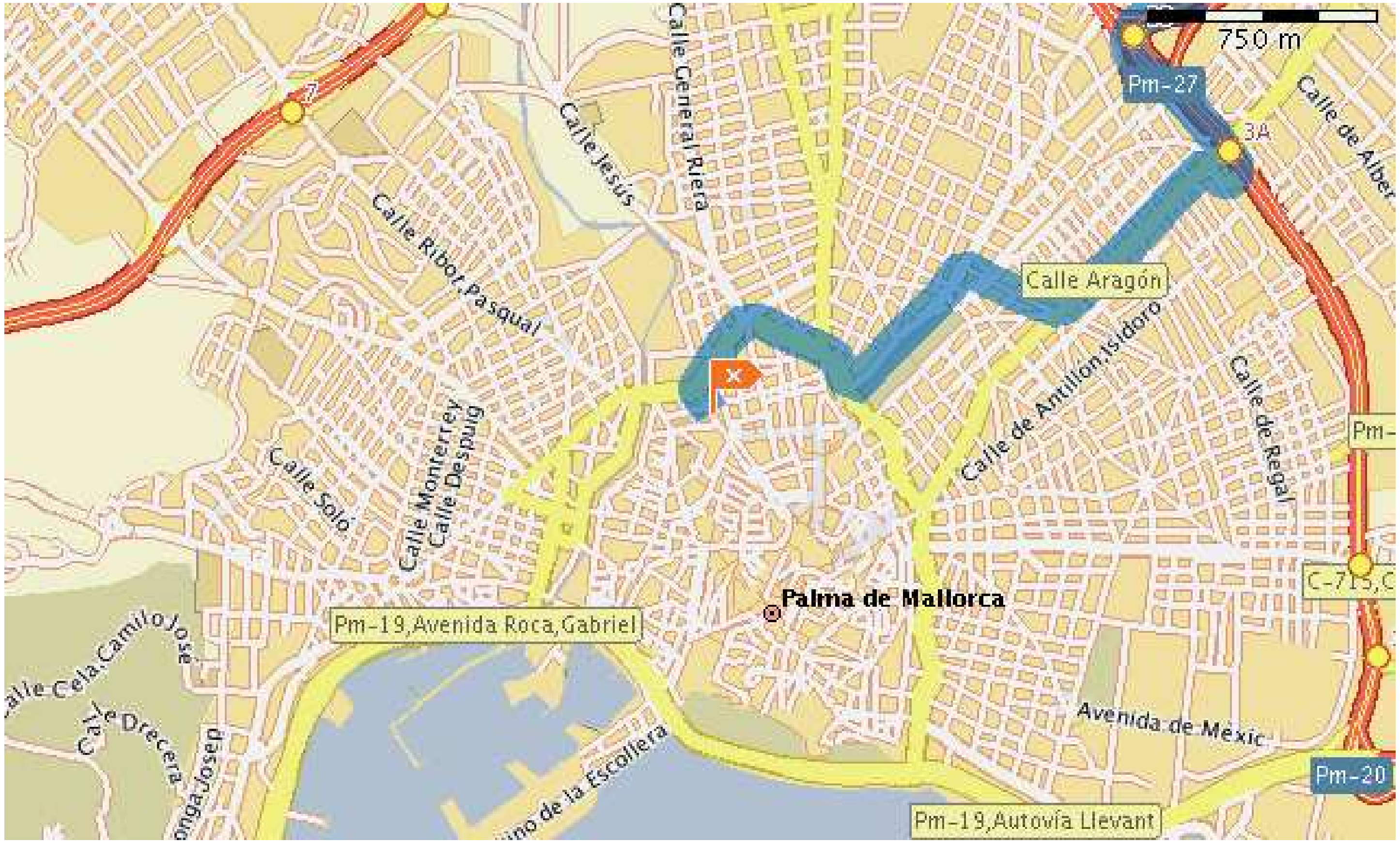
1. Look for the next reasonable motorway

Naive Route Planning

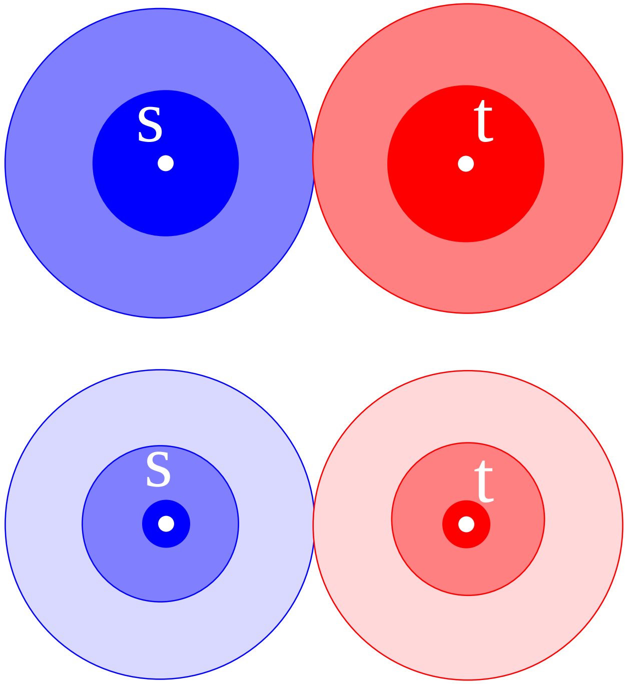
1. Look for the next reasonable motorway
2. Drive on motorways to a location close to the target

Naive Route Planning

1. Look for the next reasonable motorway
2. Drive on motorways to a location close to the target
3. Search the target starting from the motorway exit



Commercial Approach



Heuristic Highway Hierarchy

- complete search in local area
- search in (sparser) highway network
- iterate ⇝ highway hierarchy

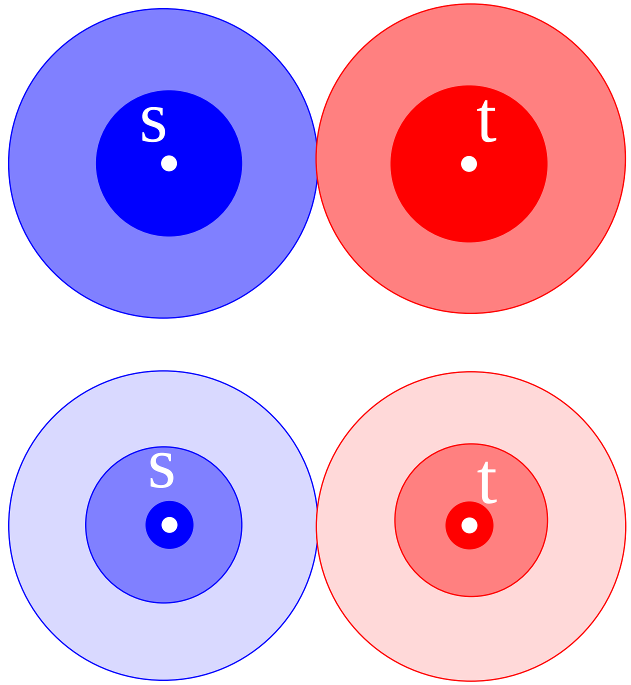
Defining the highway network:

use road category (highway, federal highway, motorway,. . .)

+ manual rectifications

- delicate compromise
- **speed ⇔ accuracy**

Our Approach



Exact Highway Hierarchy

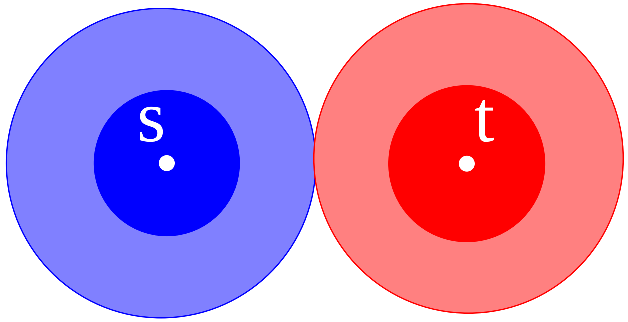
- complete search in local area
- search in (sparser) highway network
- iterate ⇝ highway hierarchy

Defining the highway network:

minimal network that preserves all shortest paths

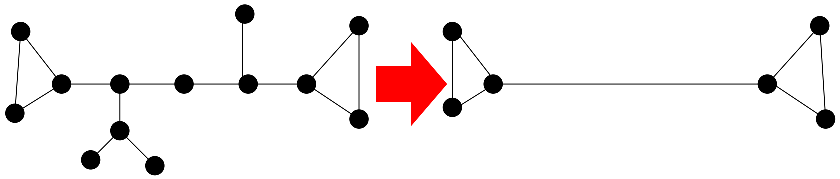
- fully automatic (just fix neighborhood size)
- uncompromisingly **fast**

Our Approach

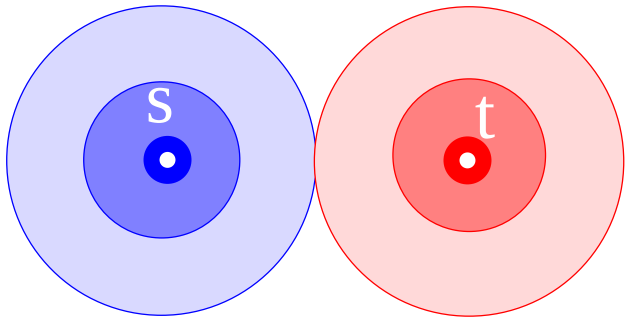


Exact Highway Hierarchy

- complete search in local area
- search in (sparser) highway network
- **<u>contract network</u>**, e.g.,



- iterate ⇝ highway hierarchy



A Meaning of "Local"

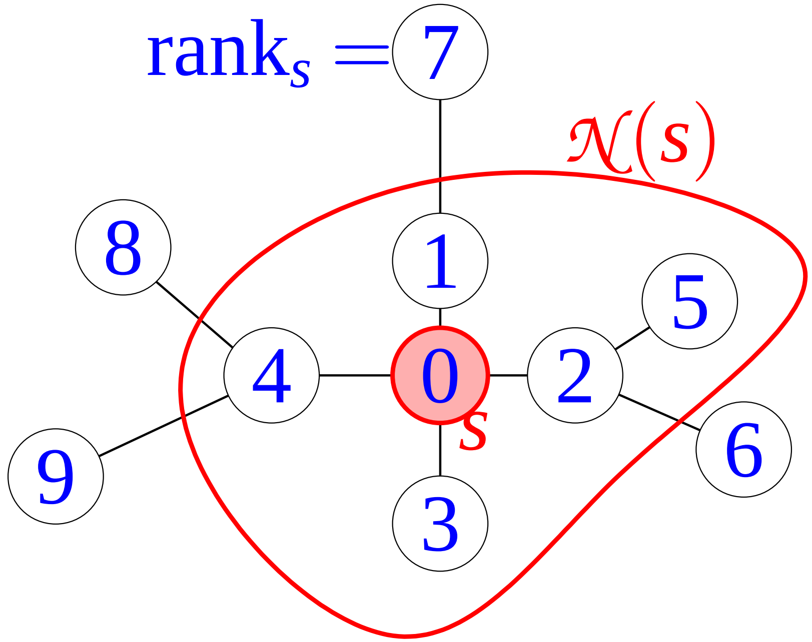
- choose neighbourhood radius $r(s)$

  e.g. distance to the H-closest node for a fixed parameter H

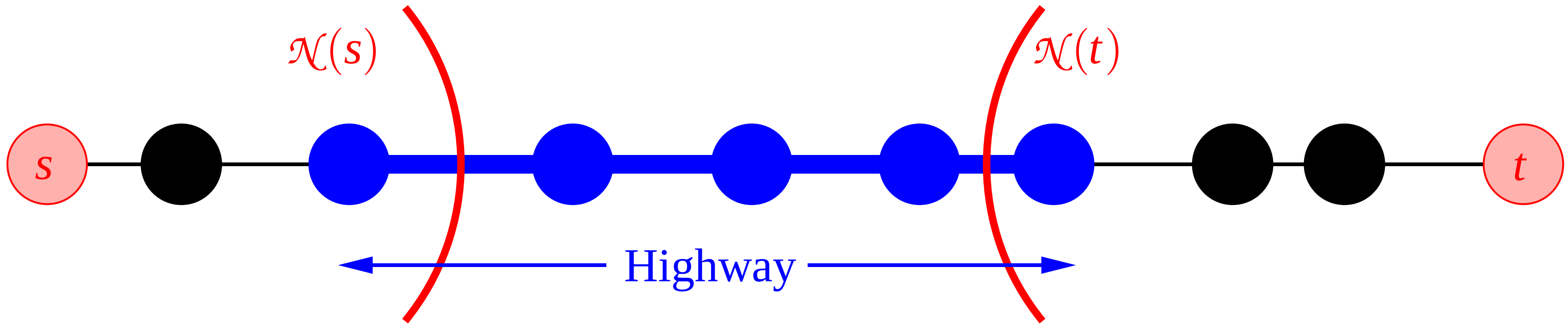
- define neighbourhood of s:

  $\mathcal{N}(s) := \{v \in V \mid d(s,v) \le r(s)\}$

- example for $H = 5$



Highway Network



Edge (u,v) belongs to highway network *iff* there are nodes s and t s.t.

- (u,v) is on the "*canonical*" shortest path from s to t

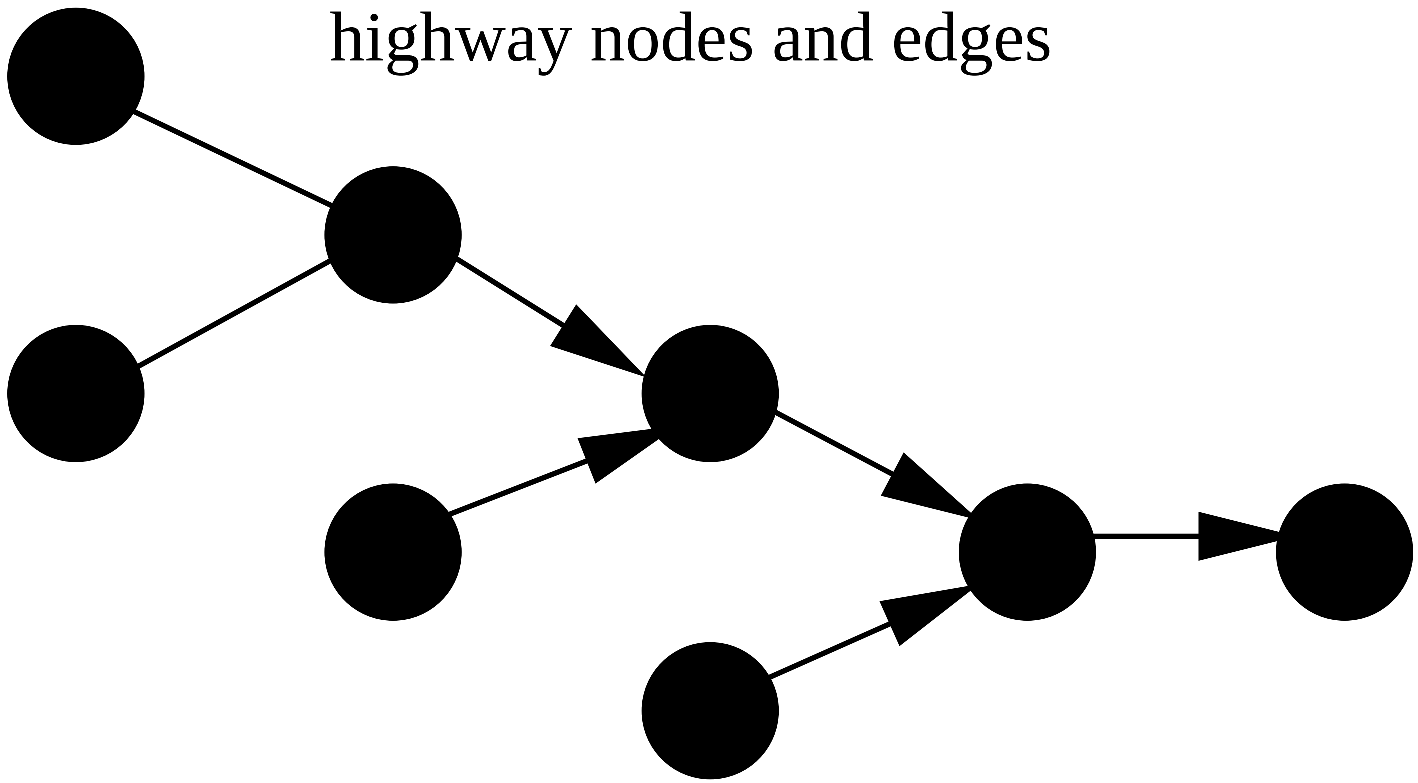
and

- $v \notin \mathcal{N}(s)$

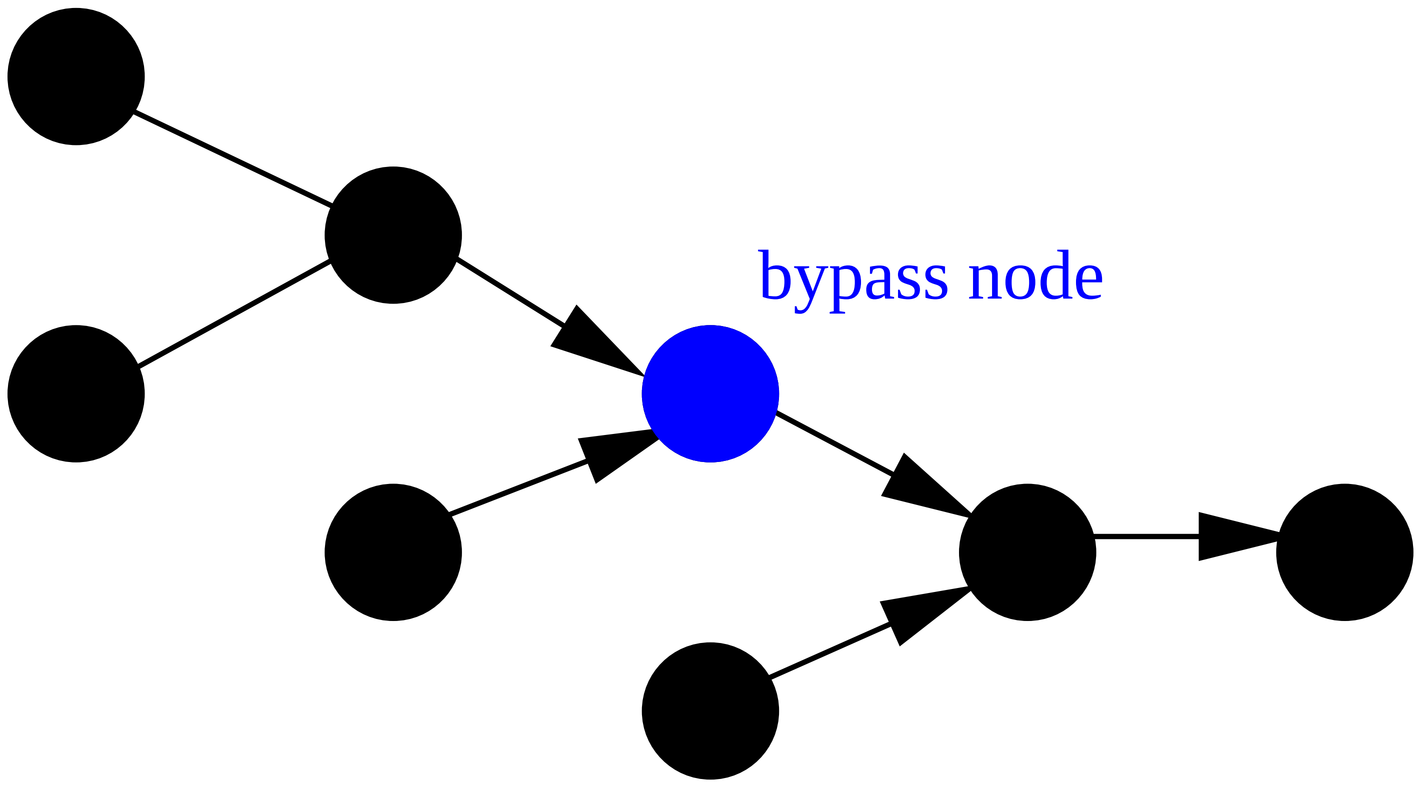
and

- $u \notin \mathcal{N}(t)$

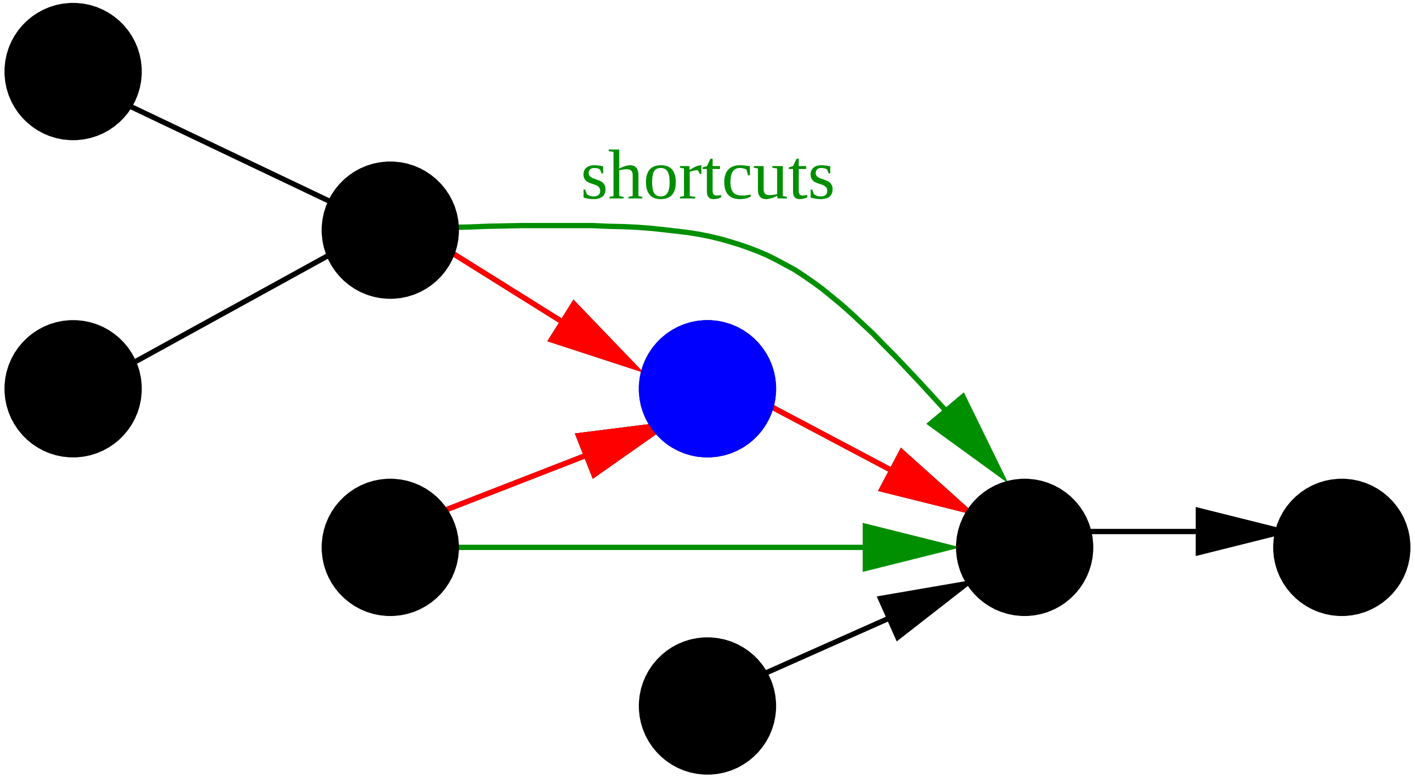
Contraction



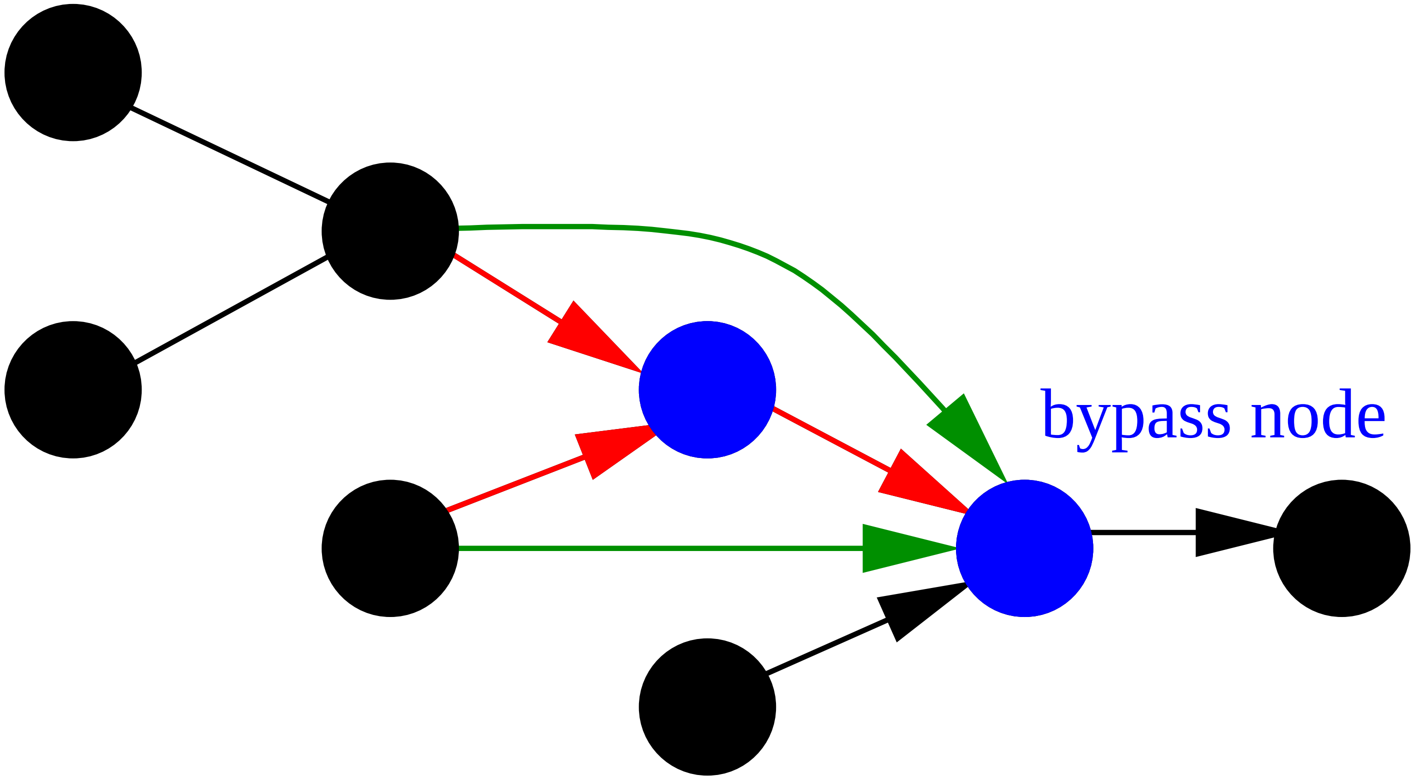
Contraction



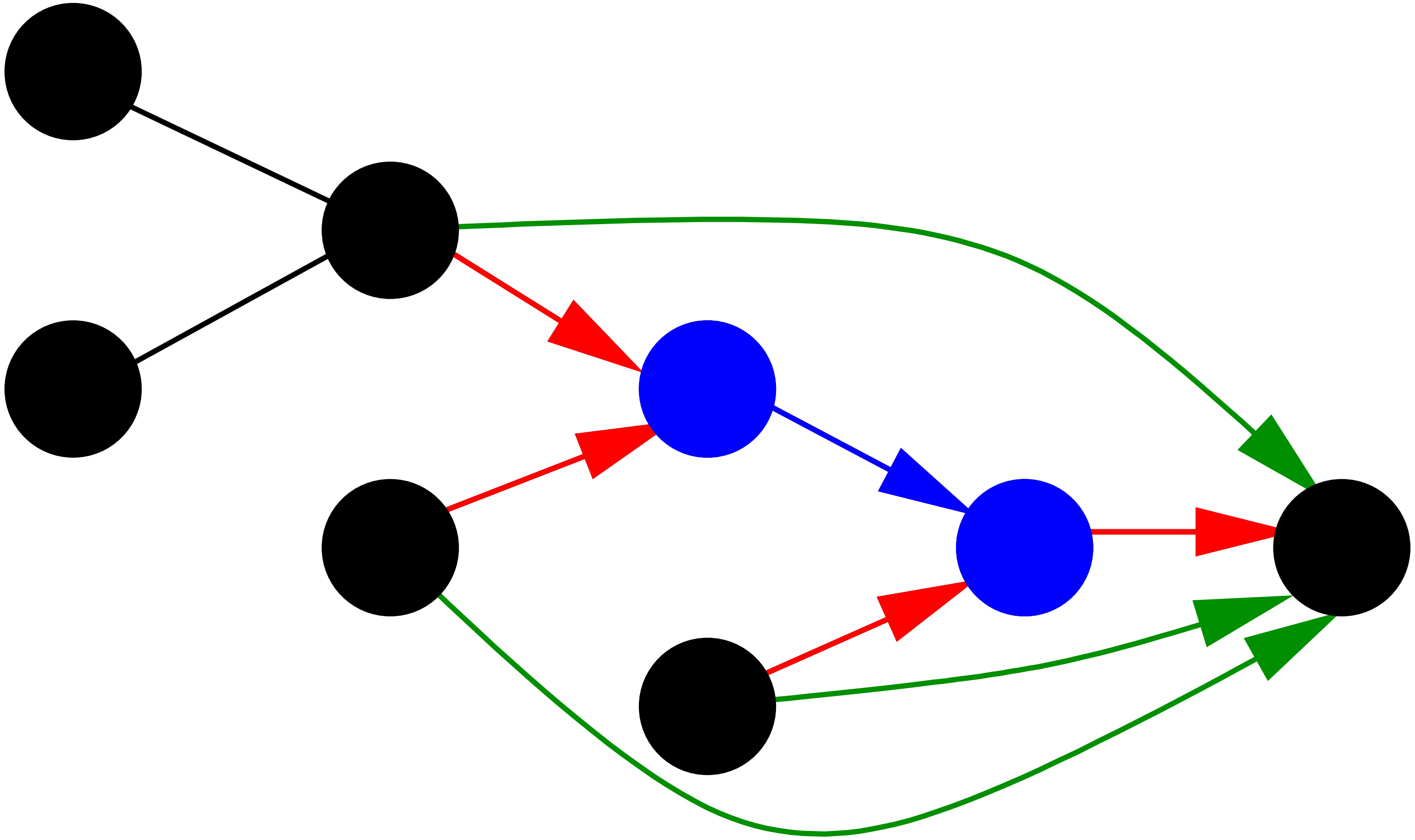
Contraction



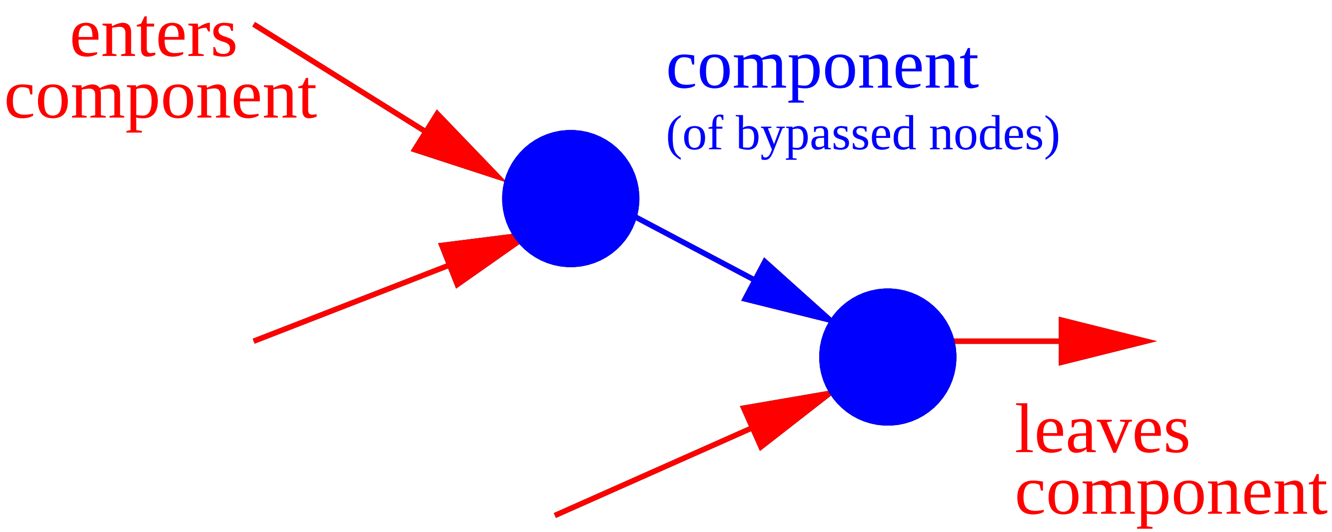
Contraction



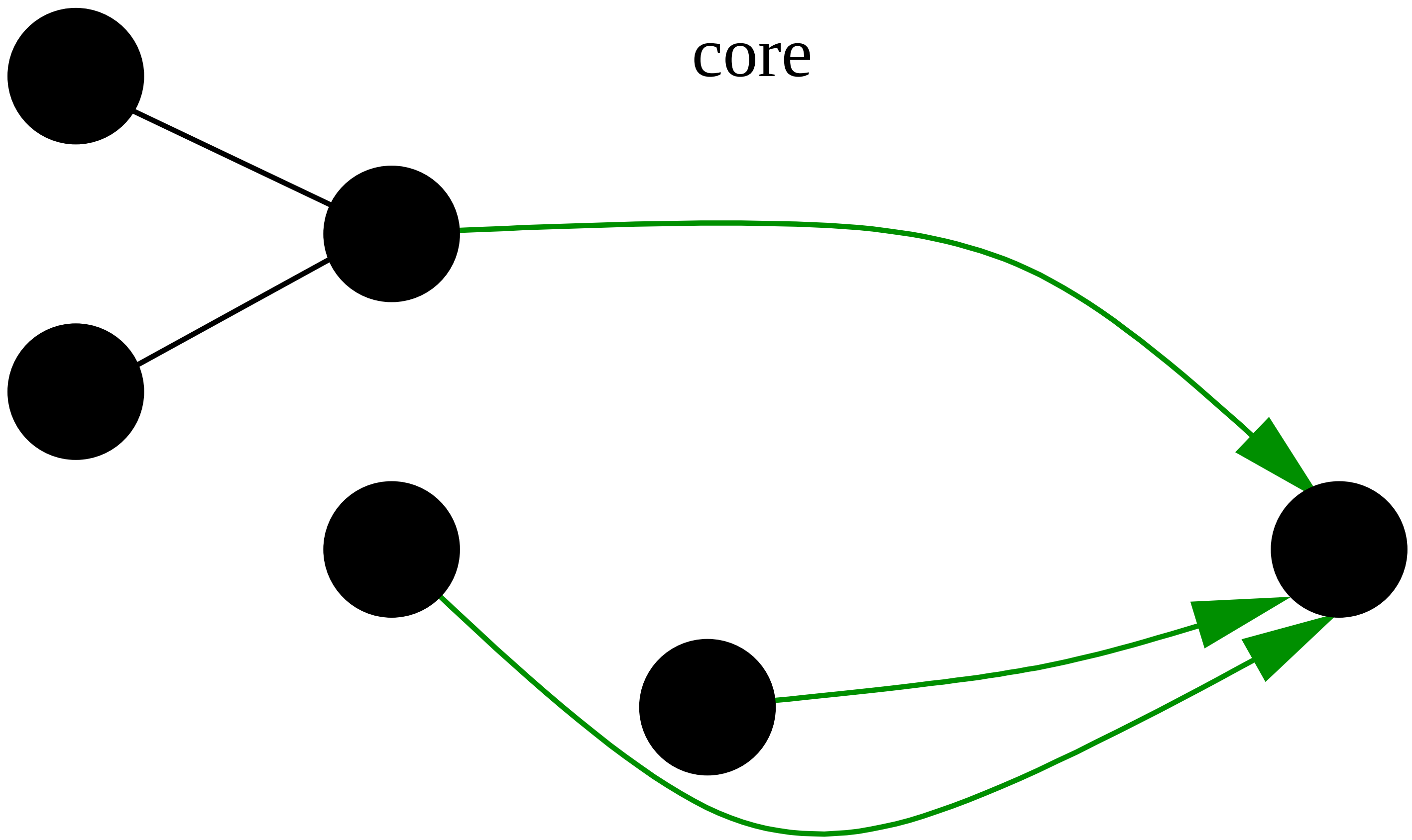
Contraction



Contraction



Contraction



Contraction

Which nodes should be bypassed?

Use some heuristic taking into account

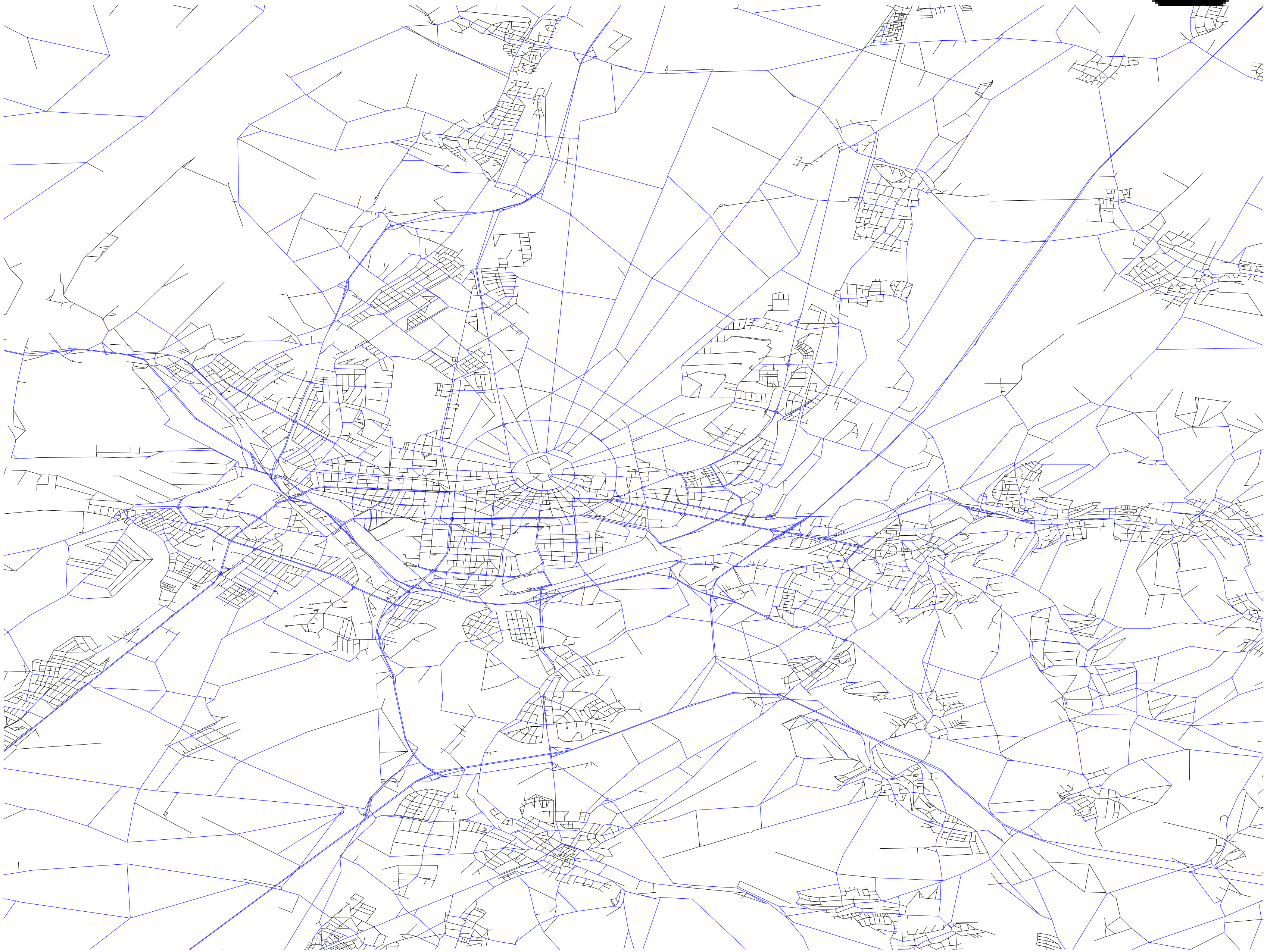
- the number of shortcuts that would be created and
- the degree of the node.

Construction

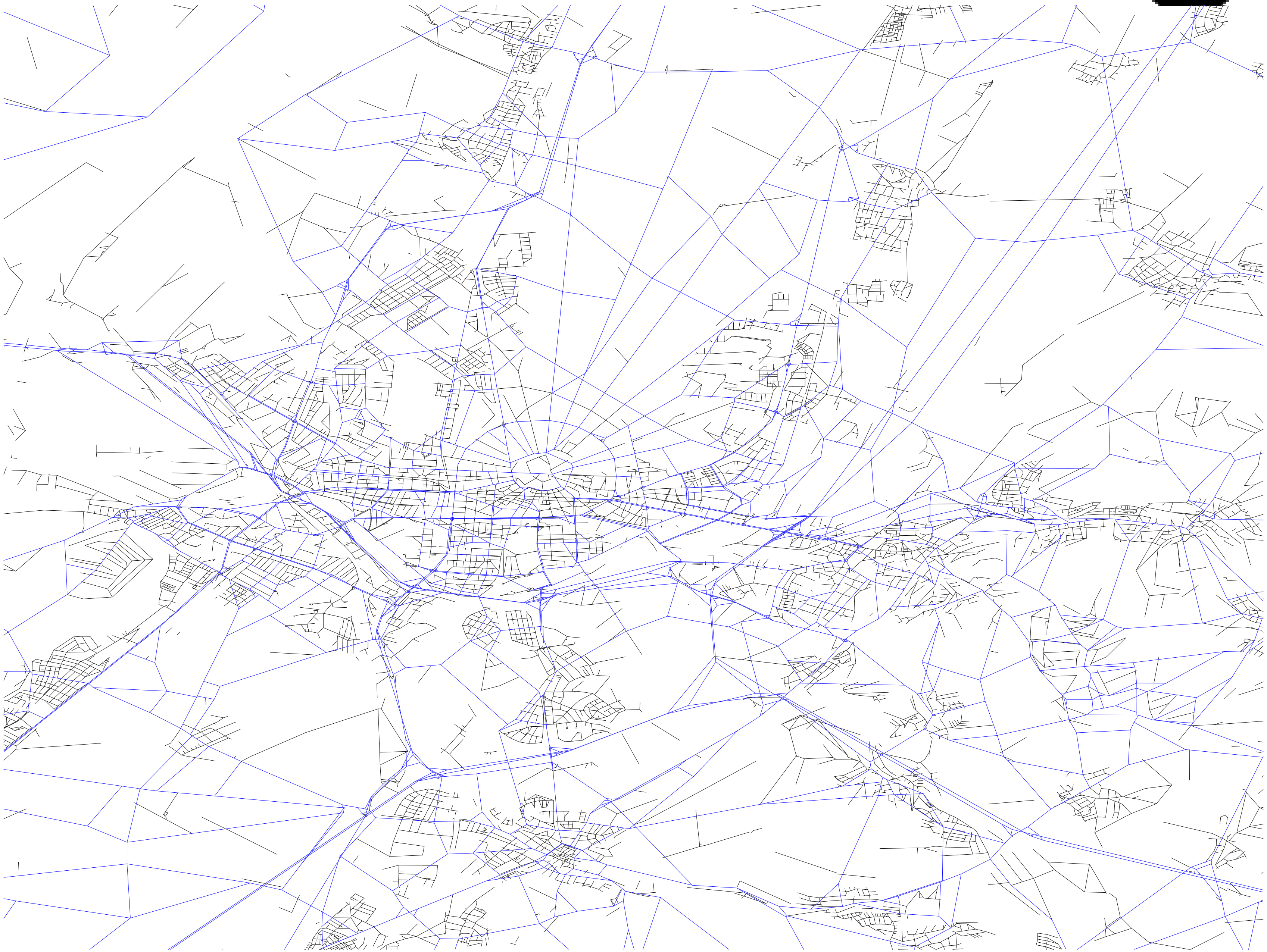
Example: Western Europe, bounding box around Karlsruhe

Complete Road Network

Highway Network



Contracted Network (Core)

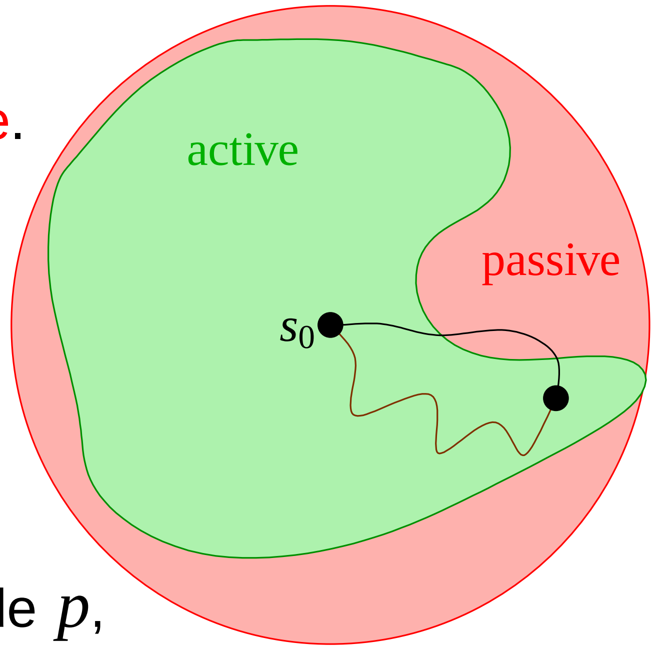


Highway Hierarchy: Level 1 and Level 2

Fast Construction

Phase 1: *Construction of Partial Shortest Path Trees*

For each node s_0, perform an SSSP search from s_0.



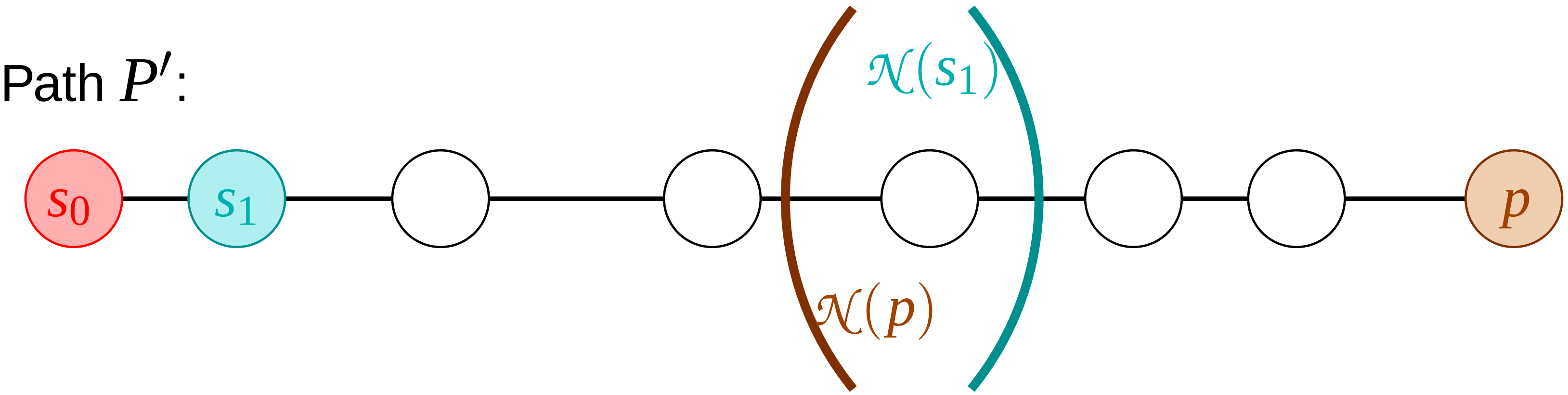
- A node's state is either active or passive.
- s_0 is active.
- A node inherits the state of its parent in the shortest path tree.
- If the abort condition is fulfilled for a node p, p's state is set to passive.

The search is aborted when all queued nodes are passive.

Fast Construction

Abort Condition:

Path P':



p is set to passive *iff*

$$s_1 \prec p \wedge p \notin \mathcal{N}(s_1) \wedge s_0 \notin \mathcal{N}(p) \wedge$$

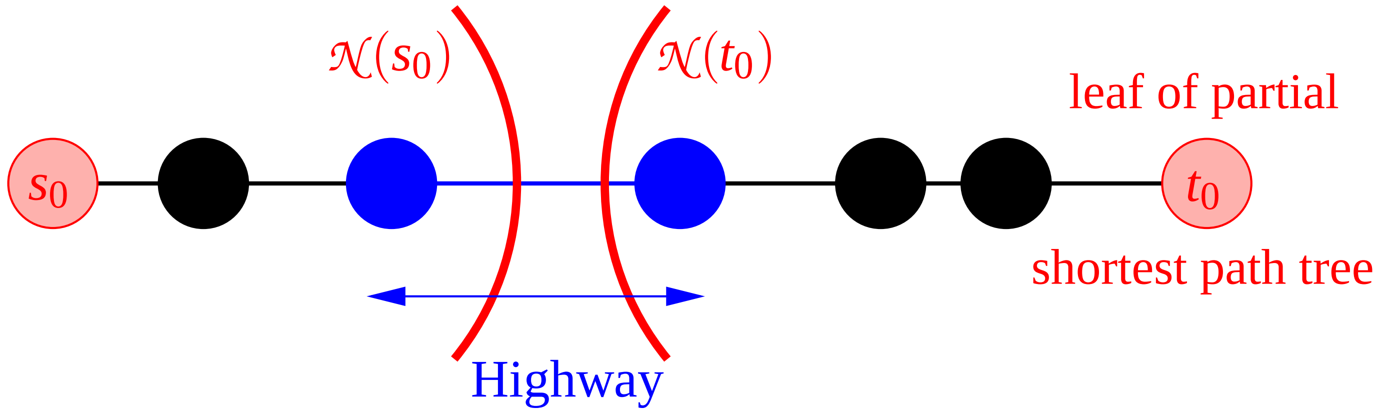
$$|P' \cap \mathcal{N}(s_1) \cap \mathcal{N}(p)| \leq 1$$

Fast Construction, Phase 2

Theorem:

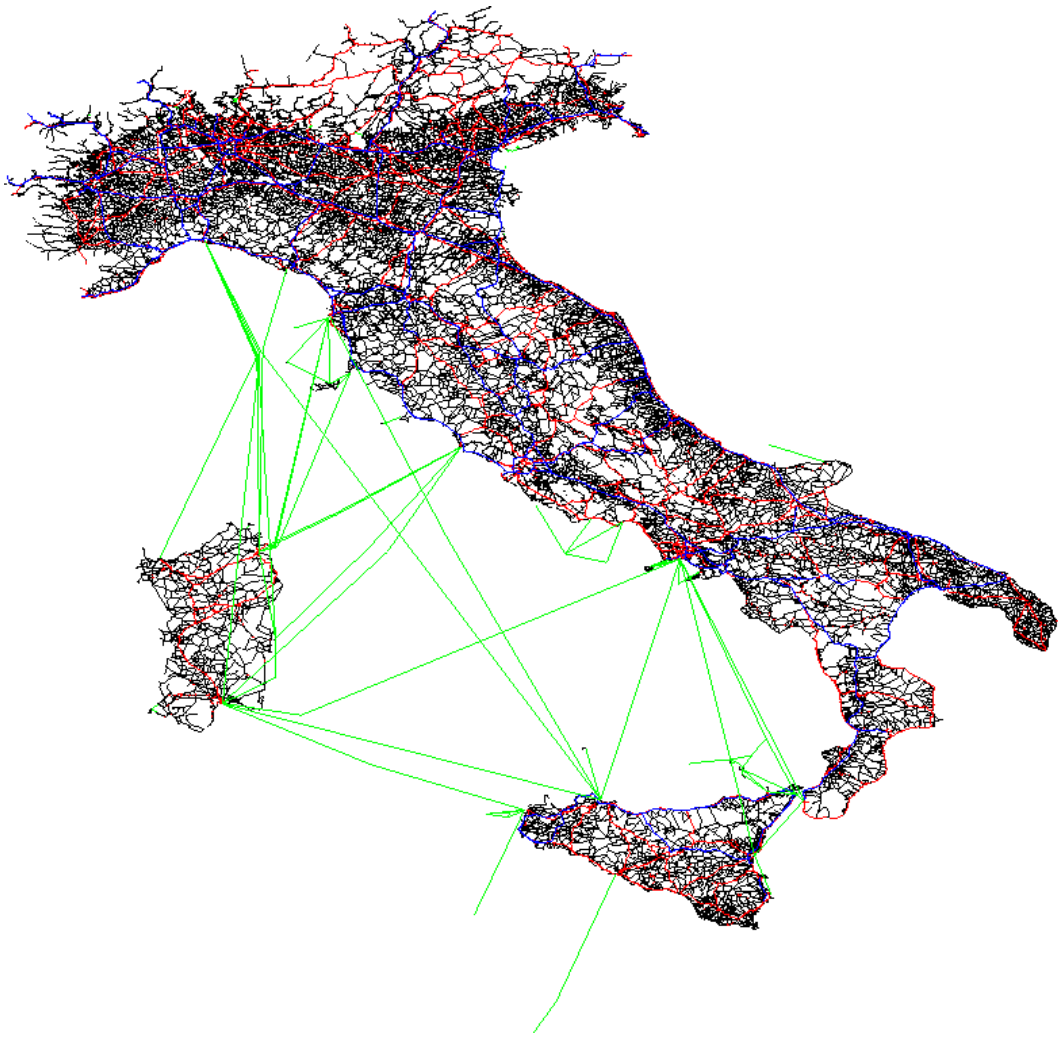
The tree roots and leaves encountered in Phase 1 witness all highway edges.

The highway edges can be found in time linear in the tree sizes.



Fast Construction

Problem: very long edges, e.g. ferries



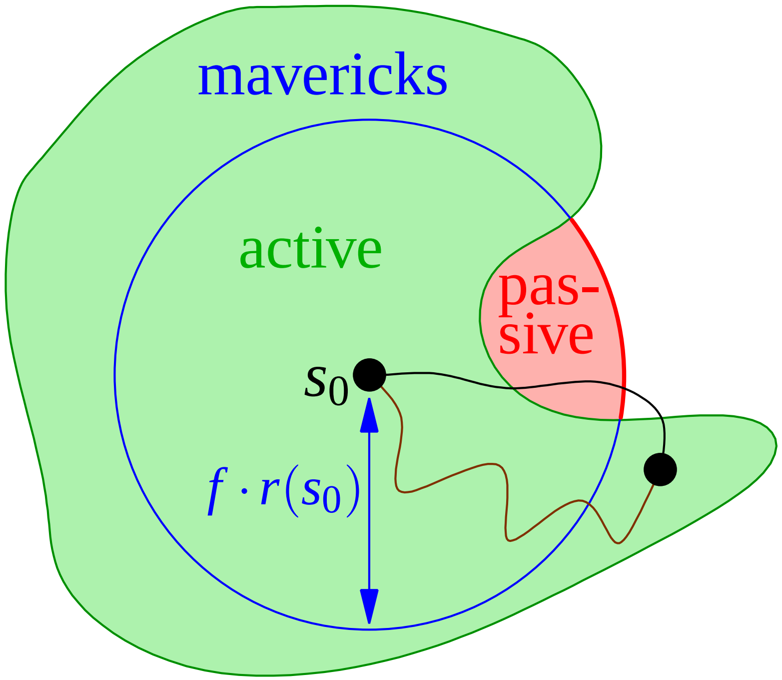
Faster Construction

Solution: An active node v is declared to be a maverick if

$$d(s_0, v) > f \cdot r(s_0).$$

When all active nodes are mavericks, the search from passive nodes is no longer continued.

$\rightsquigarrow$ superset of the highway network



Space Consumption

Choose neighborhood sizes such that levels shrink geometrically

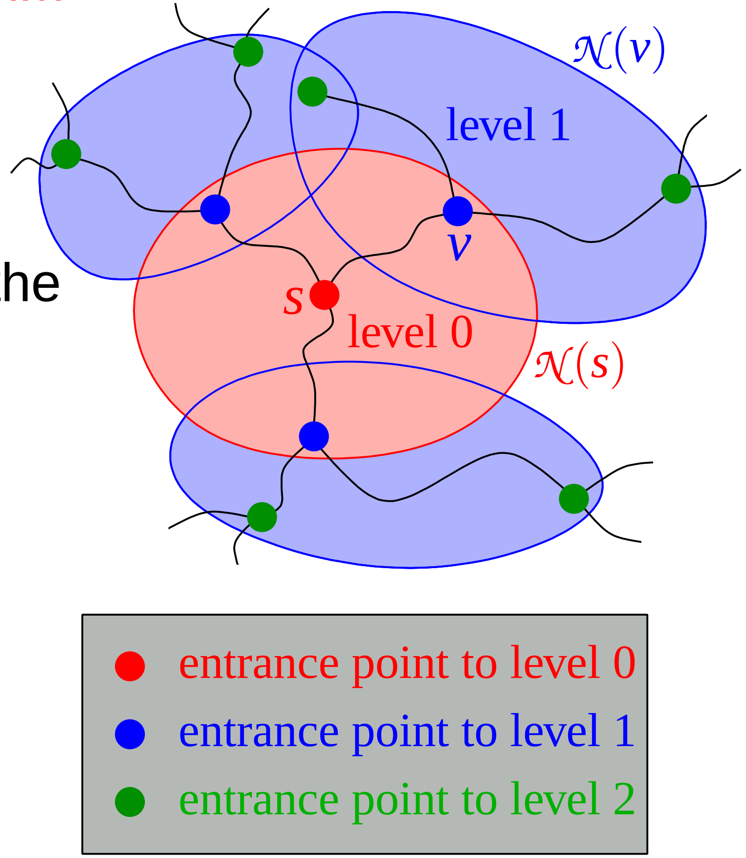
$\rightsquigarrow$ linear space consumption

Arbitrarily Small Constant Factor (not implemented):

- Large H_0 $\rightsquigarrow$ large level-0 radius $\rightsquigarrow$ small higher levels
- No $r(\cdot)$ needed for level-0 search (under certain assumptions)
- Mapping to next level by hash table

Query

Bidirectional version of Dijkstra's Algorithm



Restrictions:

- Do not leave the neighbourhood of the entrance point to the current level.

  Instead: switch to the next level.

- Do not enter a component of bypassed nodes.

Query (bidir. Dijkstra I)

Operations on two priority queues $\overrightarrow{Q}$ and $\overleftarrow{Q}$:

- void **insert**(nodeID, key)
- void **decreaseKey**(nodeID, key)
- nodeID **deleteMin**()

node u has key $\delta(u)$

- (tentative) distance from the respective source node

Query (bidir. Dijkstra II)

query(s,t) {

  $\overrightarrow{Q}$.insert$(s,0)$; $\overleftarrow{Q}$.insert$(t,0)$;

  while $(\overrightarrow{Q} \cup \overleftarrow{Q} \neq \emptyset)$ **do** {

    $\leftrightharpoons\, \in \{\rightarrow, \leftarrow\}$;   *//select direction*

    u := $\overset{\leftrightharpoons}{Q}$.deleteMin$()$;

    relaxEdges$(\leftrightharpoons, u)$;

  }

}

Query (bidir. Dijkstra III)

relaxEdges$(\leftrightharpoons, u)$ {

  foreach $e = (u,v) \in \overset{\leftrightharpoons}{E}$ **do** {

    $k := \delta(u) + w(e)$;

    if $v \in \overset{\leftrightharpoons}{Q}$ **then** $\overset{\leftrightharpoons}{Q}$.decreaseKey$(v,k)$; **else** $\overset{\leftrightharpoons}{Q}$.insert$(v,k)$;

  }

}

Query (Hwy I)

Operations on two priority queues $\overrightarrow{Q}$ and $\overleftarrow{Q}$:

- void **insert**(nodeID, key)
- void **decreaseKey**(nodeID, key)
- nodeID **deleteMin**()

node u has key $(\delta(u), \ell(u), \text{gap}(u))$

- (tentative) distance from the respective source node
- search level
- gap to the next neighbourhood border

lexicographical order: $<, >, <$

Query (Hwy II)

query(s,t) {

$\overrightarrow{Q}$.insert$(s, (0,0,r_0^{\rightarrow}(s)))$; $\overleftarrow{Q}$.insert$(t, (0,0,r_0^{\leftarrow}(t)))$;

while $(\overrightarrow{Q} \cup \overleftarrow{Q} \neq \emptyset)$ **do** {

$\leftrightharpoons \in \{\rightarrow, \leftarrow\}$; *//select direction*

u := $\overset{\leftrightharpoons}{Q}$.deleteMin$()$;

relaxEdges$(\leftrightharpoons, u)$;

}

}

Query (Hwy III)

relaxEdges$(\leftrightharpoons, u)$ {

foreach $e = (u,v) \in \overleftrightharpoons{E}$ **do** {

gap := gap(u);

if gap = ∞ **then** gap $:= r_{\overleftrightharpoons{\ell(u)}}(u)$; // leave component

for $(\ell := \ell(u)$; $w(e) >$ gap; ℓ++, gap := $r_{\overleftrightharpoons{\ell}}(u))$; // go "upwards"

if $\ell(e) < \ell$ **then continue**; // Restriction 1

if e "enters a component" **then continue**; // Restriction 2

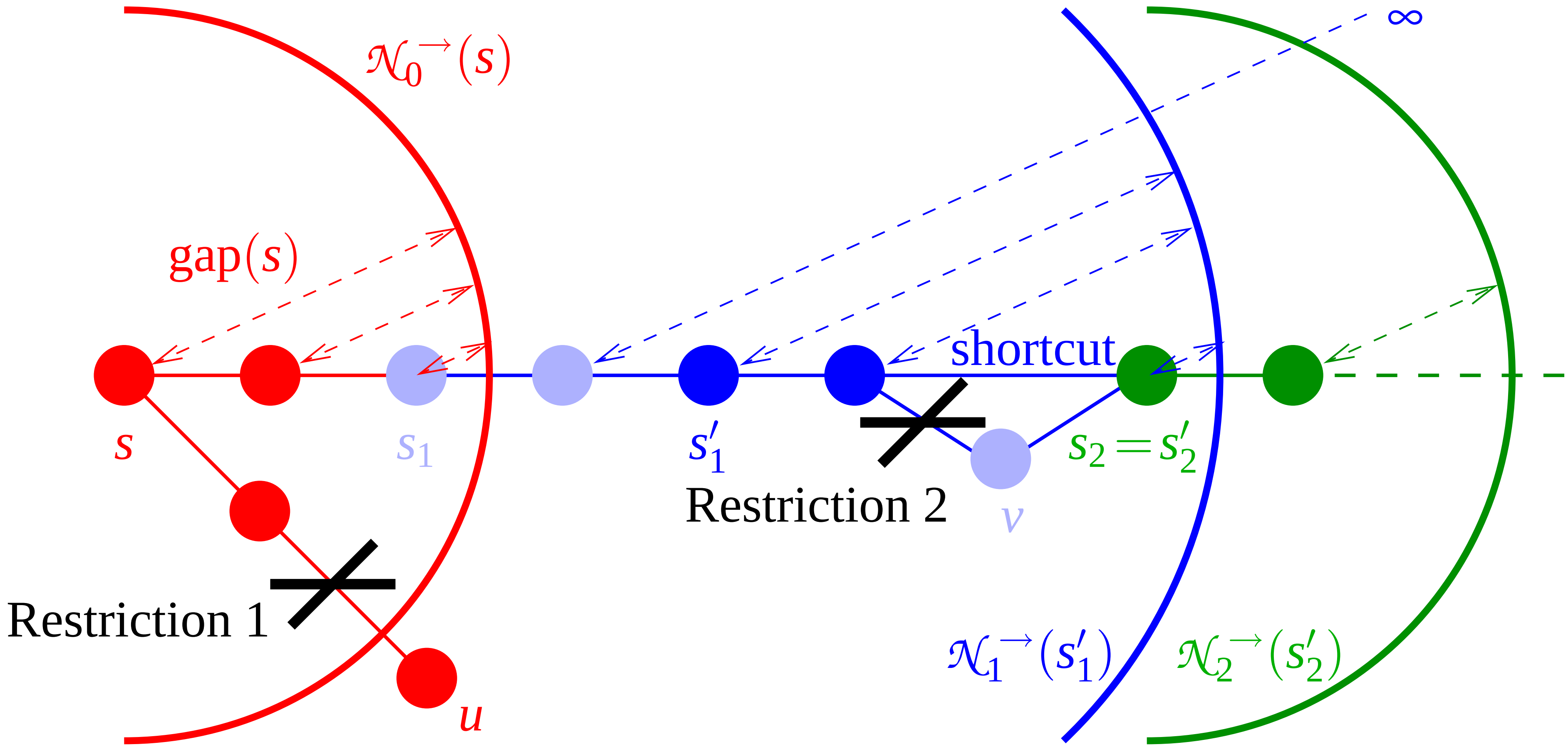
k := $(\delta(u) + w(e)$, ℓ, gap$-w(e))$;

if $v \in \overleftrightharpoons{Q}$ **then** $\overleftrightharpoons{Q}$.decreaseKey$(v,k)$; **else** $\overleftrightharpoons{Q}$.insert$(v,k)$;

}

}

Query


$\mathcal{N}_0^{\rightarrow}(s)$
gap(s)
∞
shortcut
s
s_1
s'_1
$s_2 = s'_2$
v
u
Restriction 1
Restriction 2
$\mathcal{N}_1^{\rightarrow}(s'_1)$
$\mathcal{N}_2^{\rightarrow}(s'_2)$

Query

Theorem:

We still find the shortest path.

Query

Example: from Karlsruhe, Am Fasanengarten 5

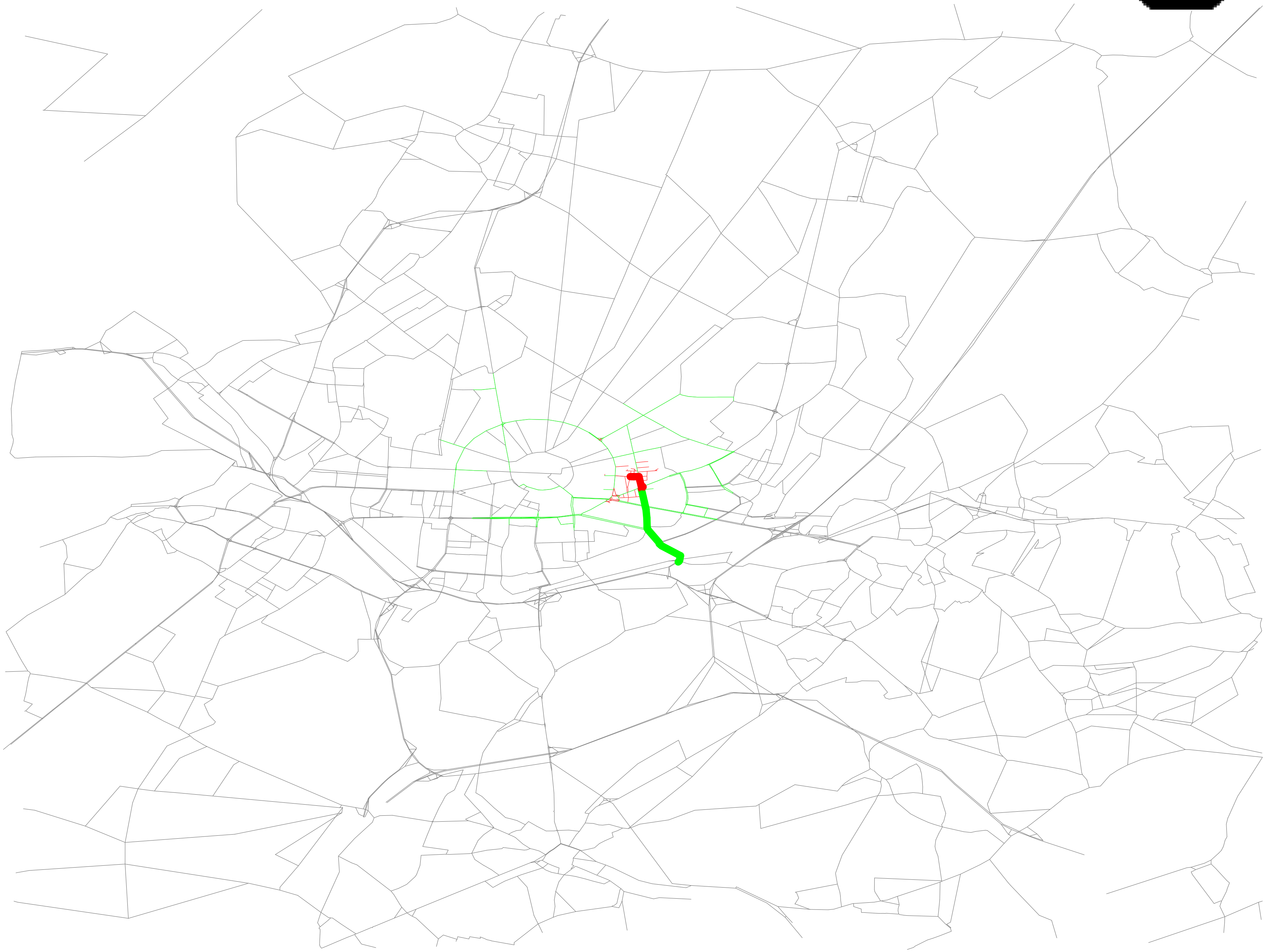
to Palma de Mallorca

Bounding Box: 20 km Level 0

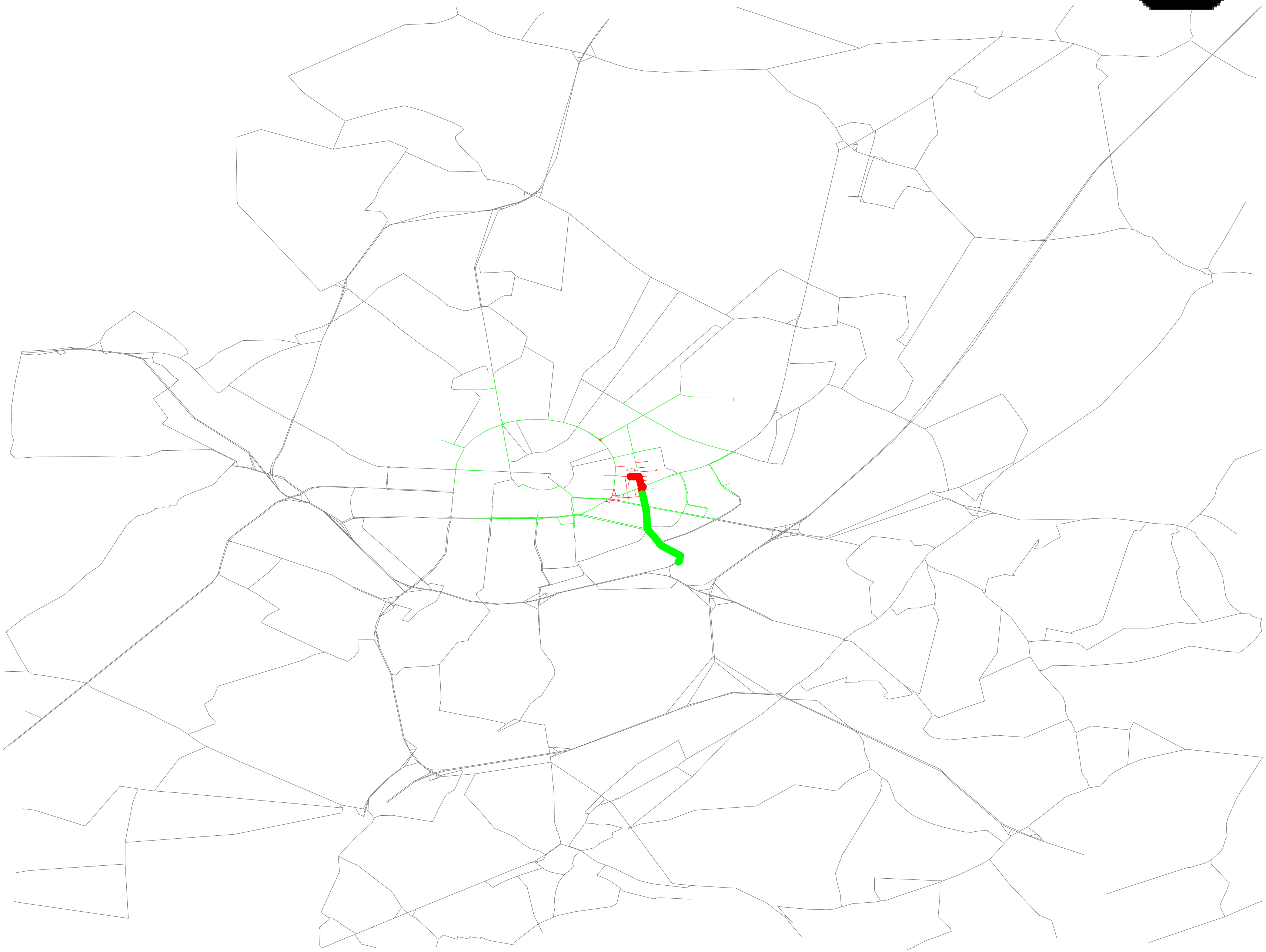
Bounding Box: 20 km Level 0 Search Space

Bounding Box: 20 km Level 1

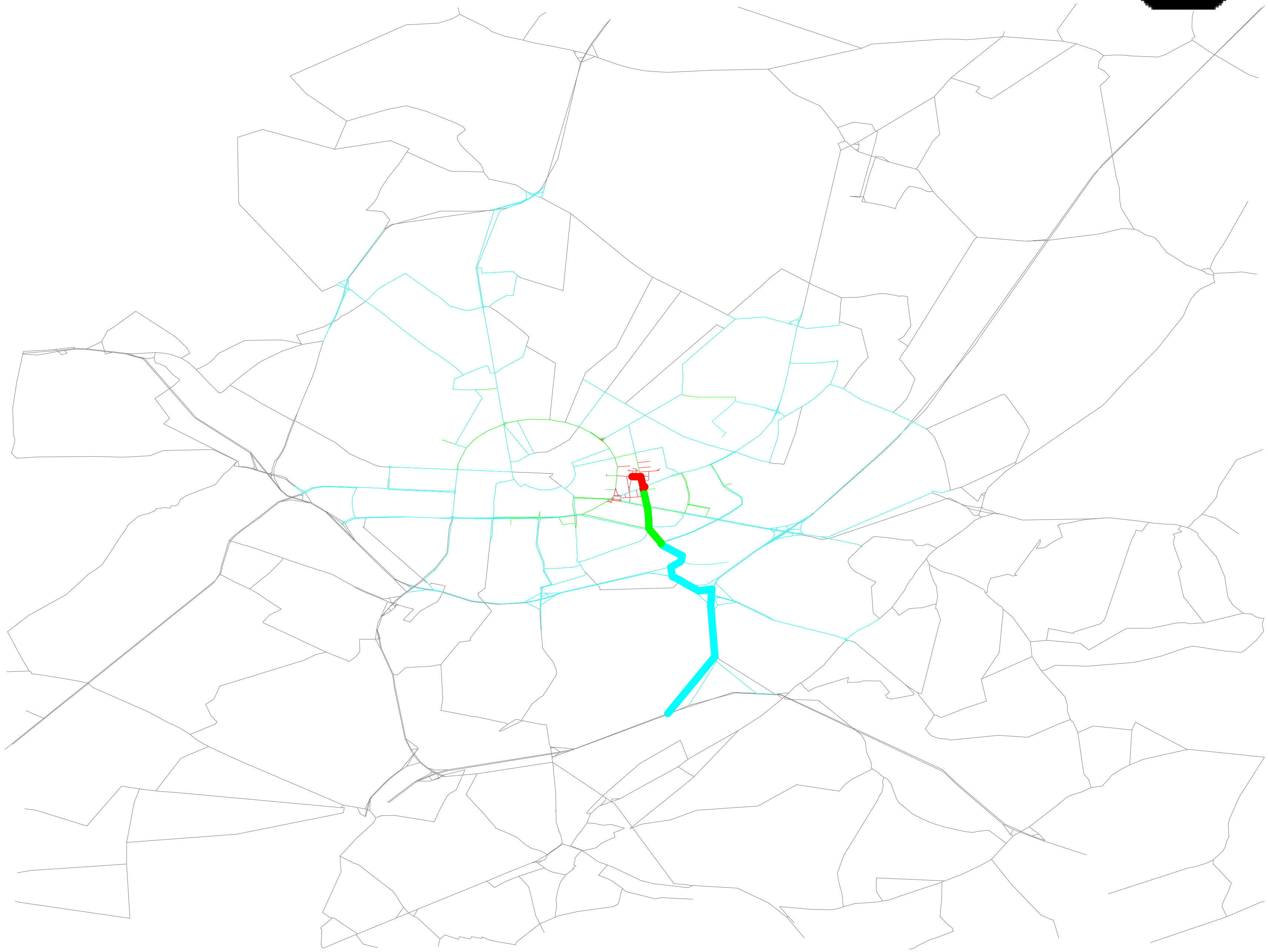
Bounding Box: 20 km Level 1 Search Space



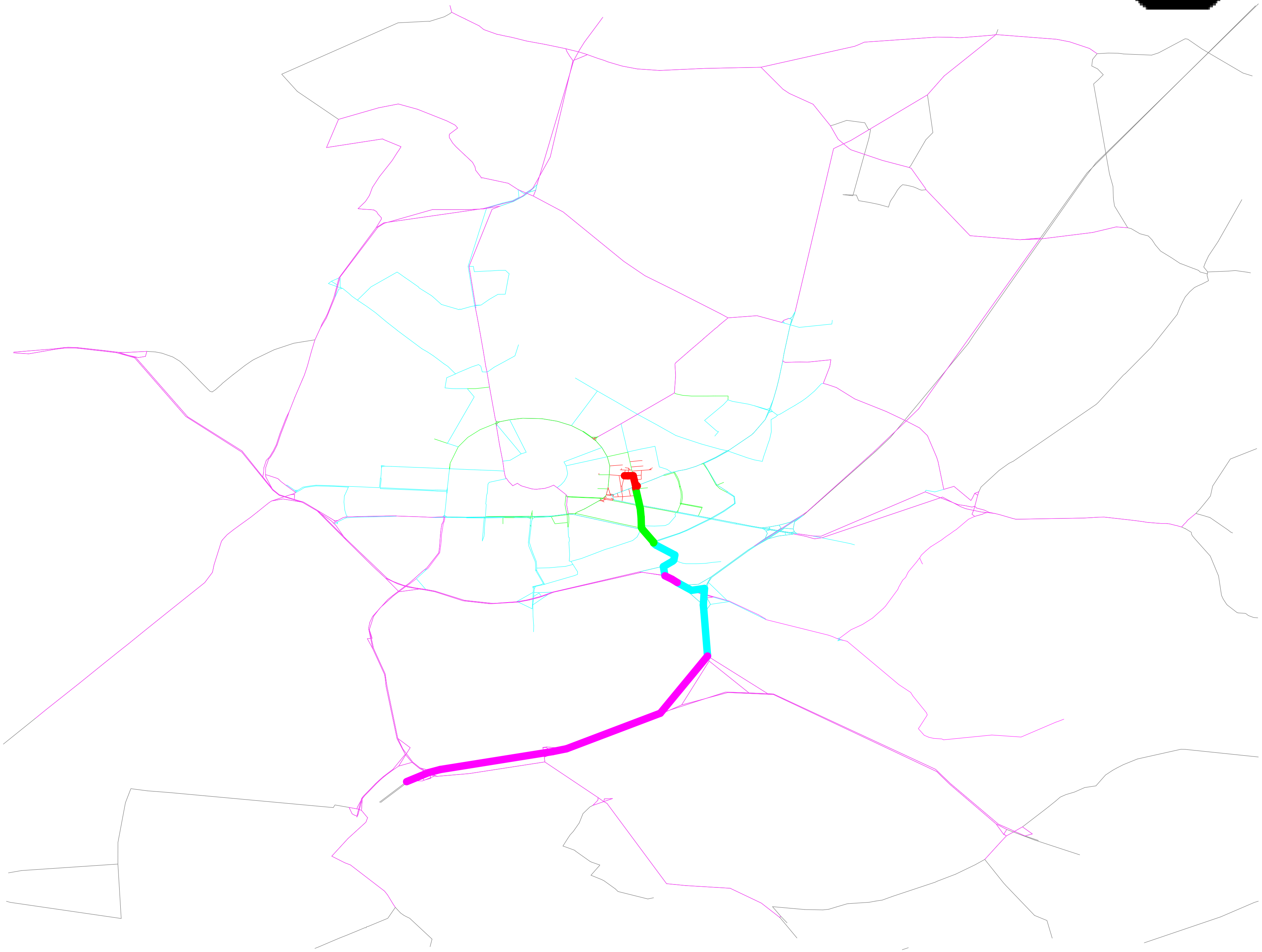
Bounding Box: 20 km Level 2



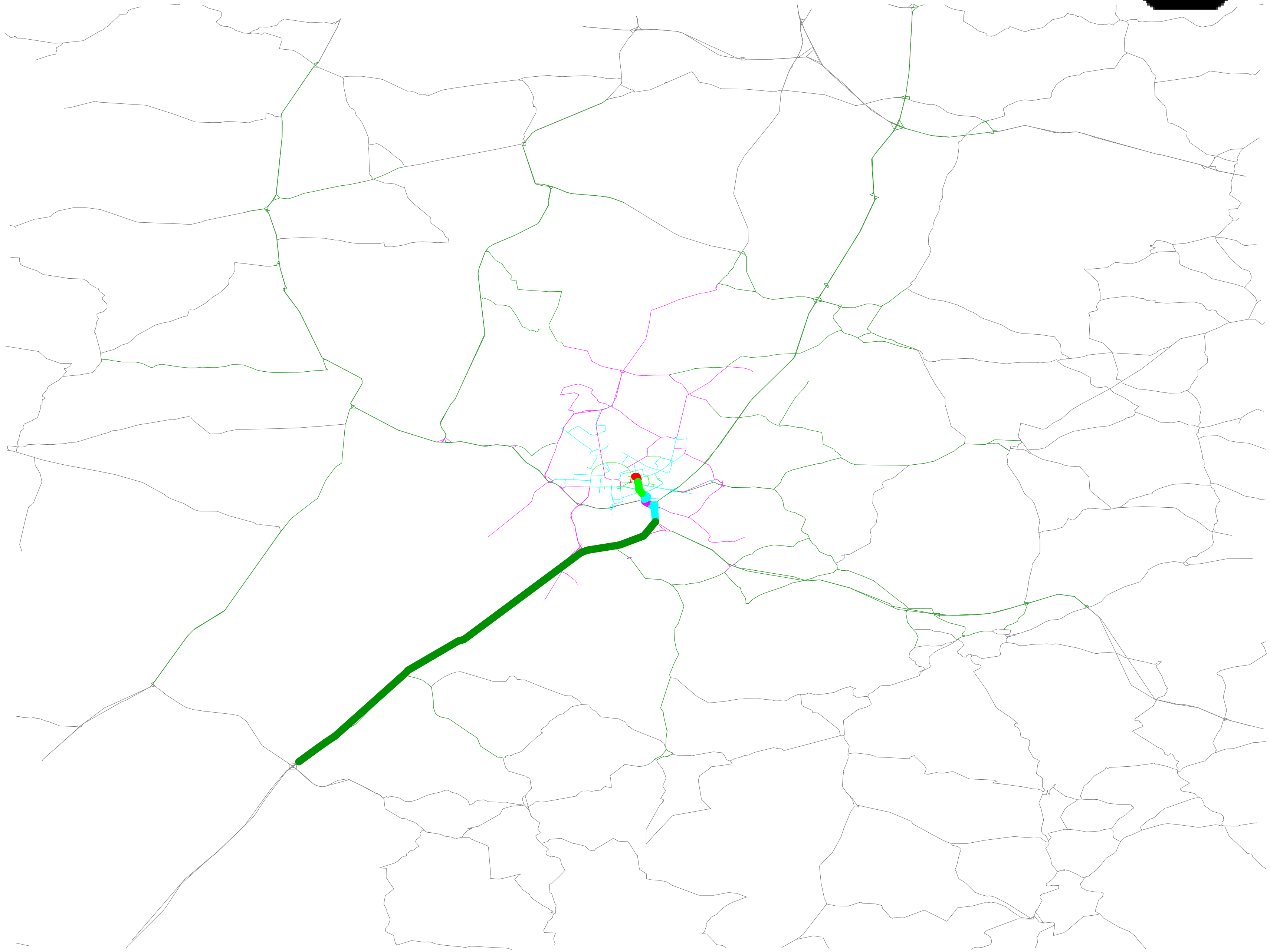
Bounding Box: 20 km Level 2 Search Space



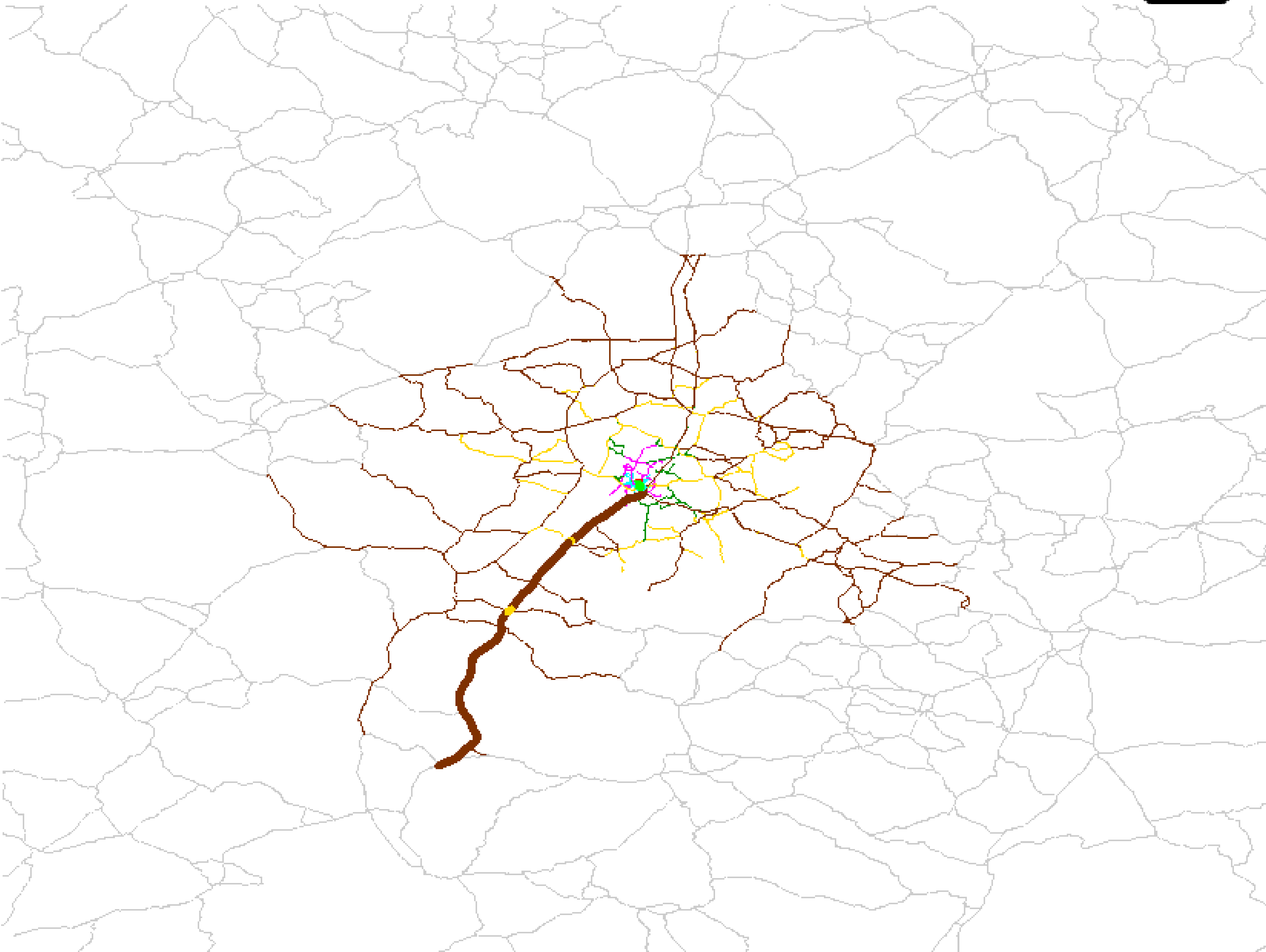
Bounding Box: 20 km Level 3 Search Space

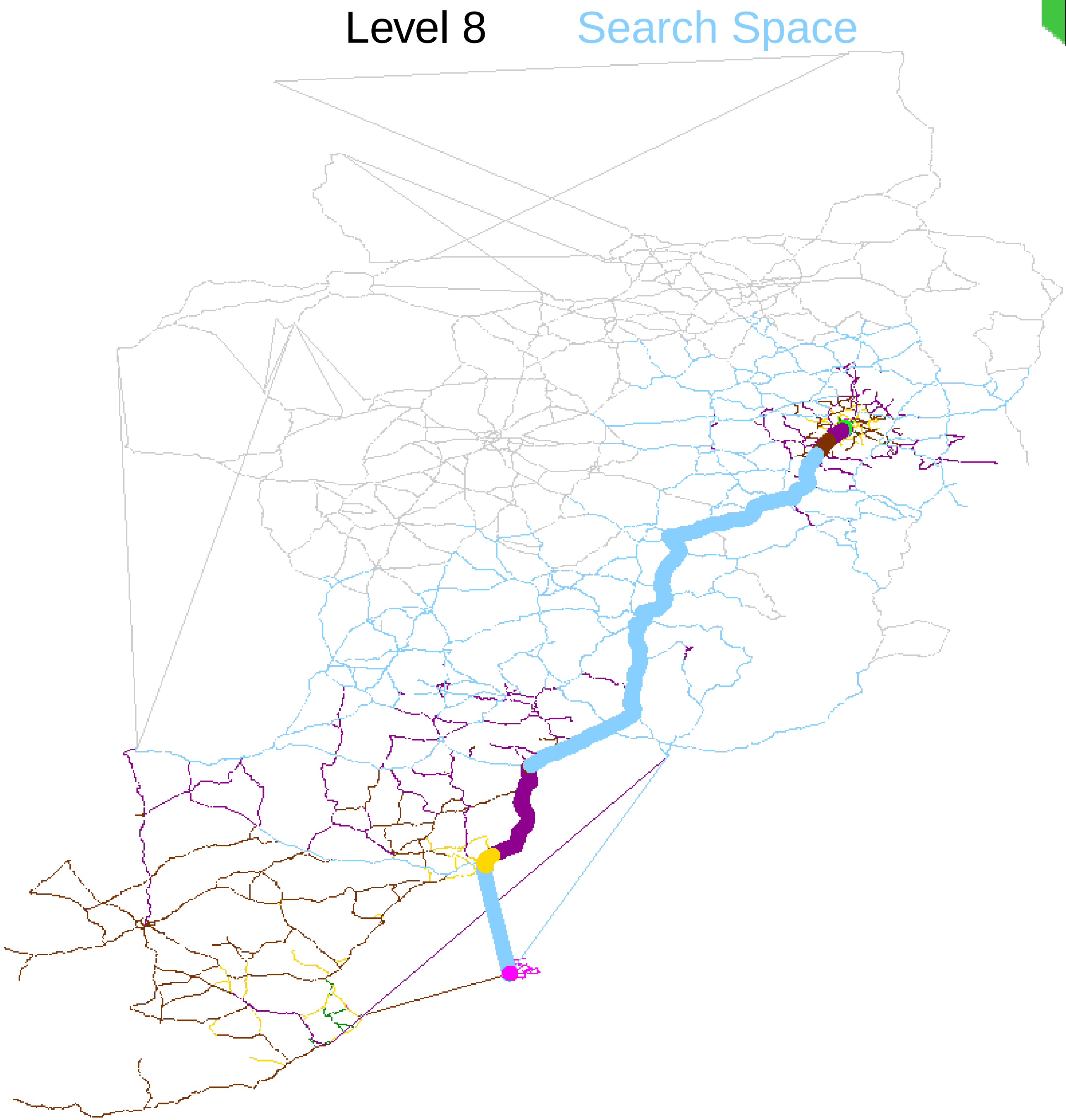


Bounding Box: 80 km Level 4 Search Space

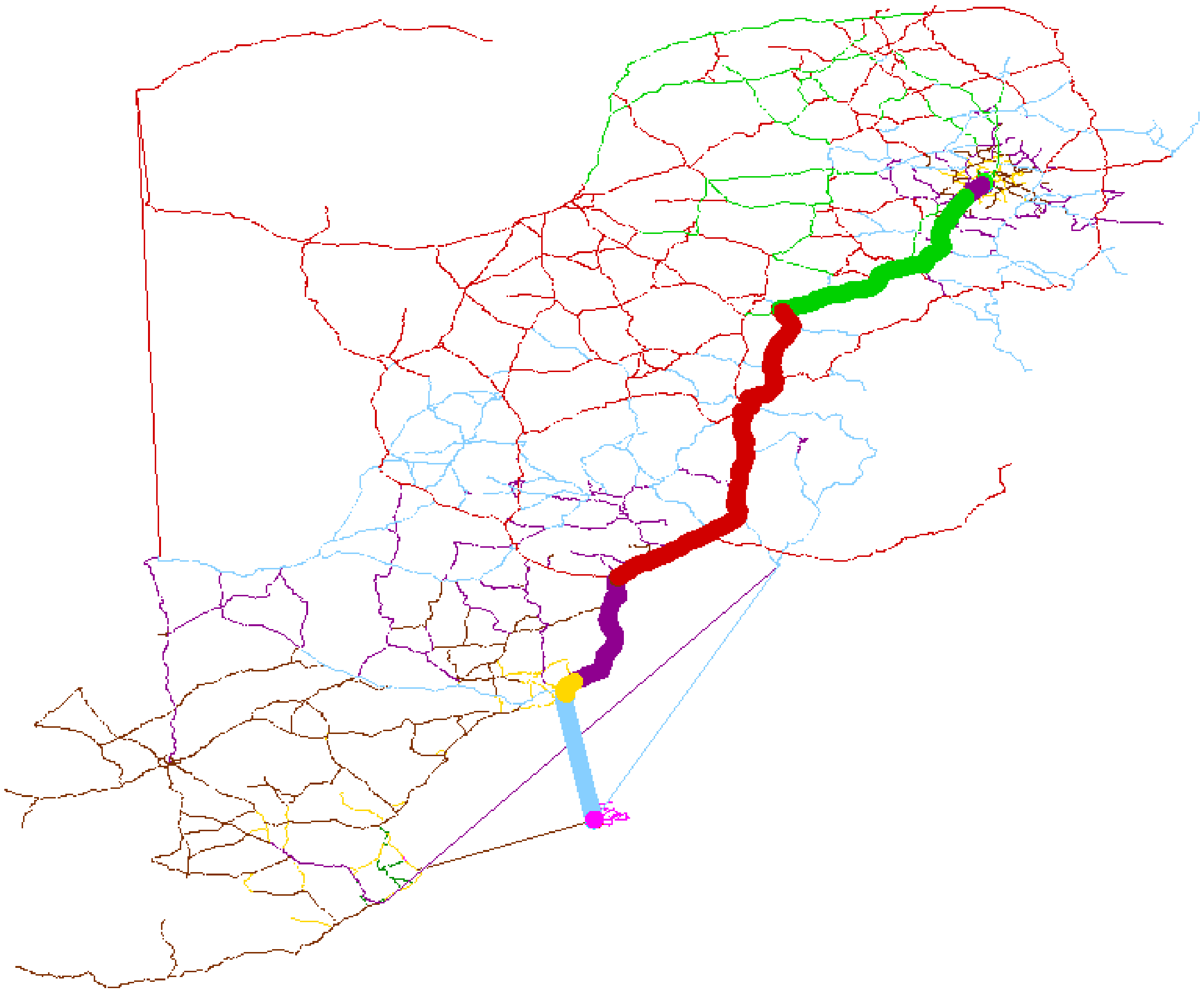


Bounding Box: 400 km Level 6 Search Space




Level 8
Search Space

Level 10 Search Space

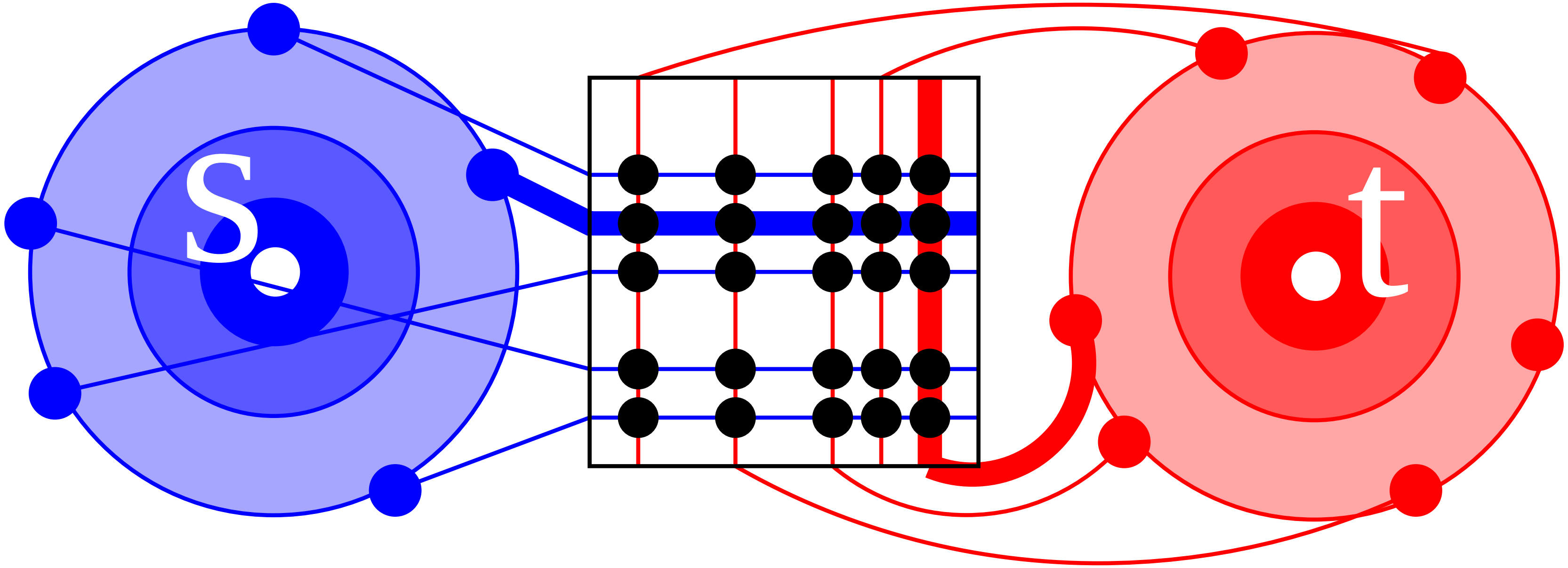


Optimisation: Distance Table

Construction:

- Construct fewer levels. e.g. 4 instead of 9
- Compute an all-pairs distance table for the topmost level L. 13 465 $\times$ 13 465 entries

Distance Table Query:



- Abort the search when all entrance points in the core of level L have been encountered. $\approx$ 55 for each direction
- Use the distance table to bridge the gap. $\approx 55 \times 55$ entries

Implementation

- C++
- heavy use of templates
- reused binary heap priority queue
- reused DIJKSTRA
- 5 750 lines of code

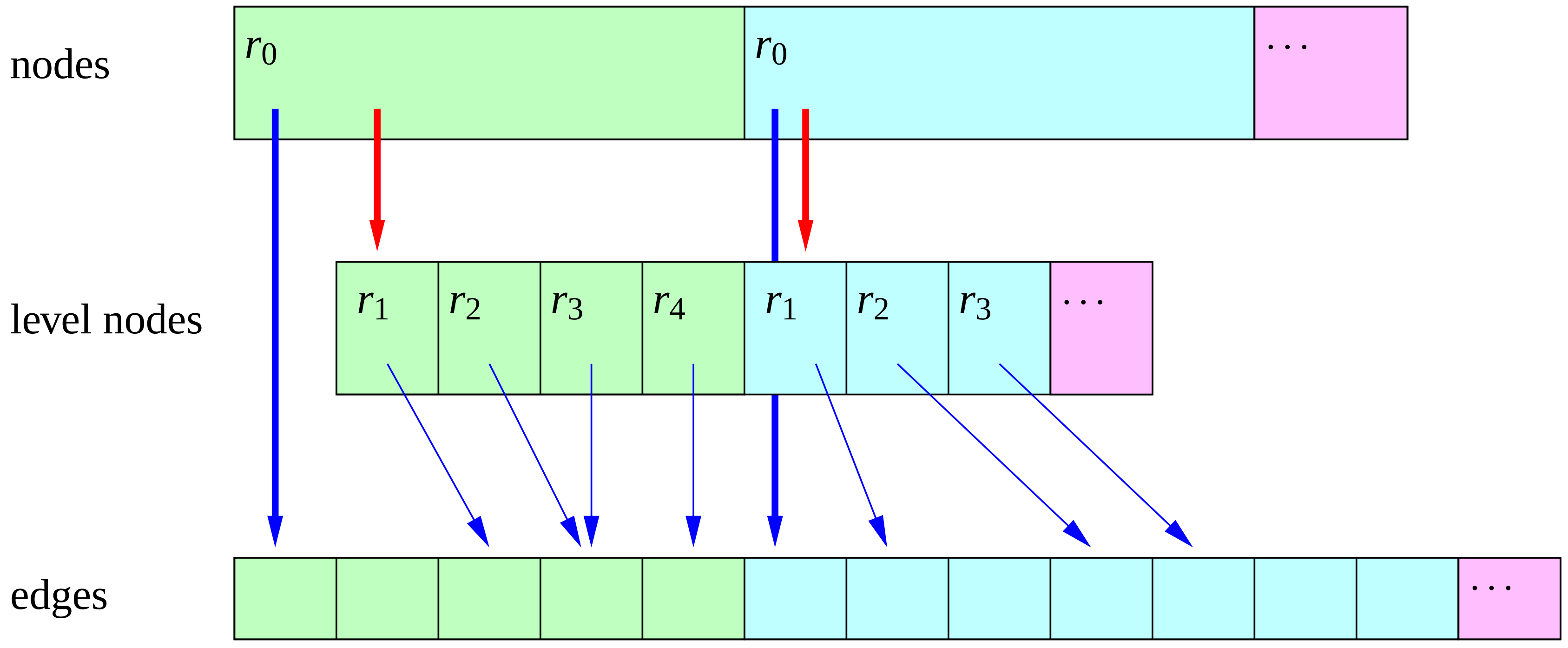
  incl. comments, excl. infrastructure (I/O, logging, . . .)

Implementation

- adjacency array graph representation

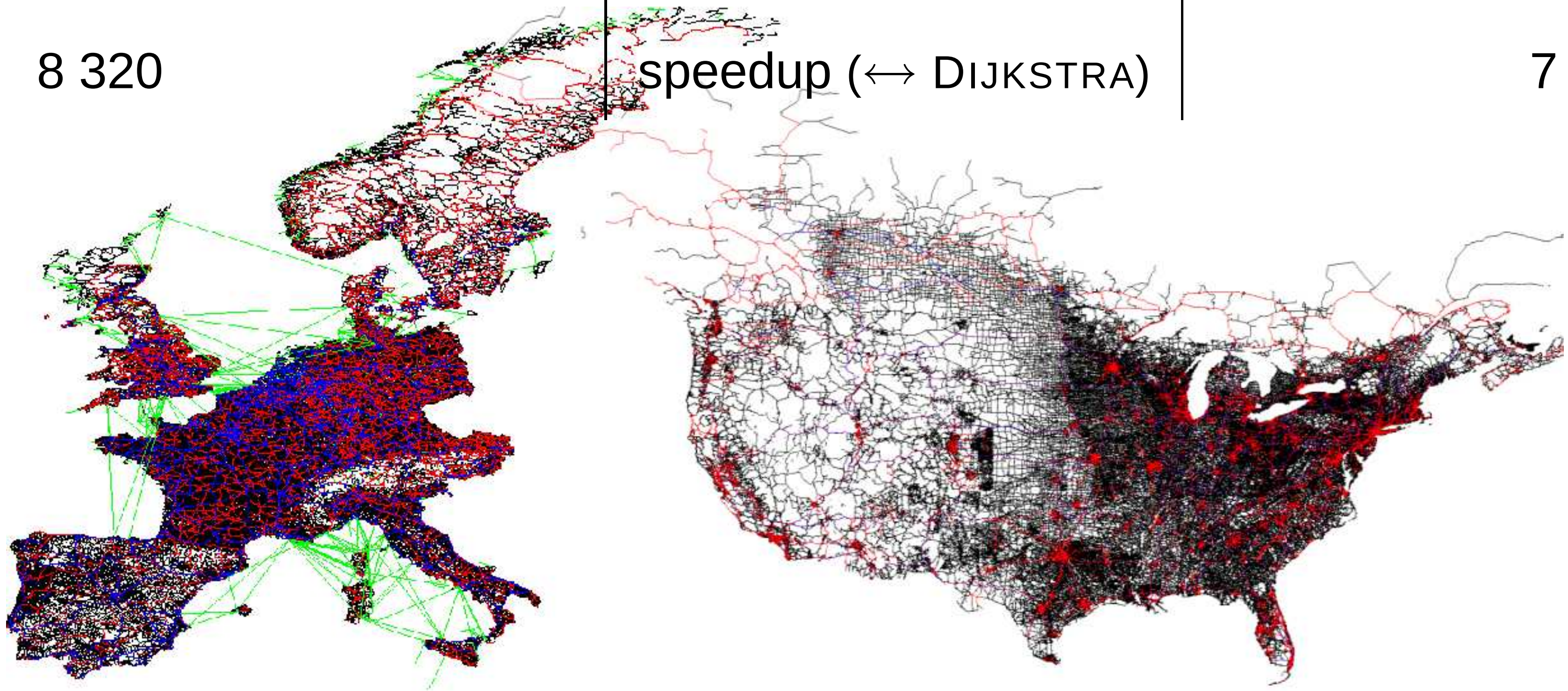
  forward + backward directed edges

  separate array for level specific data

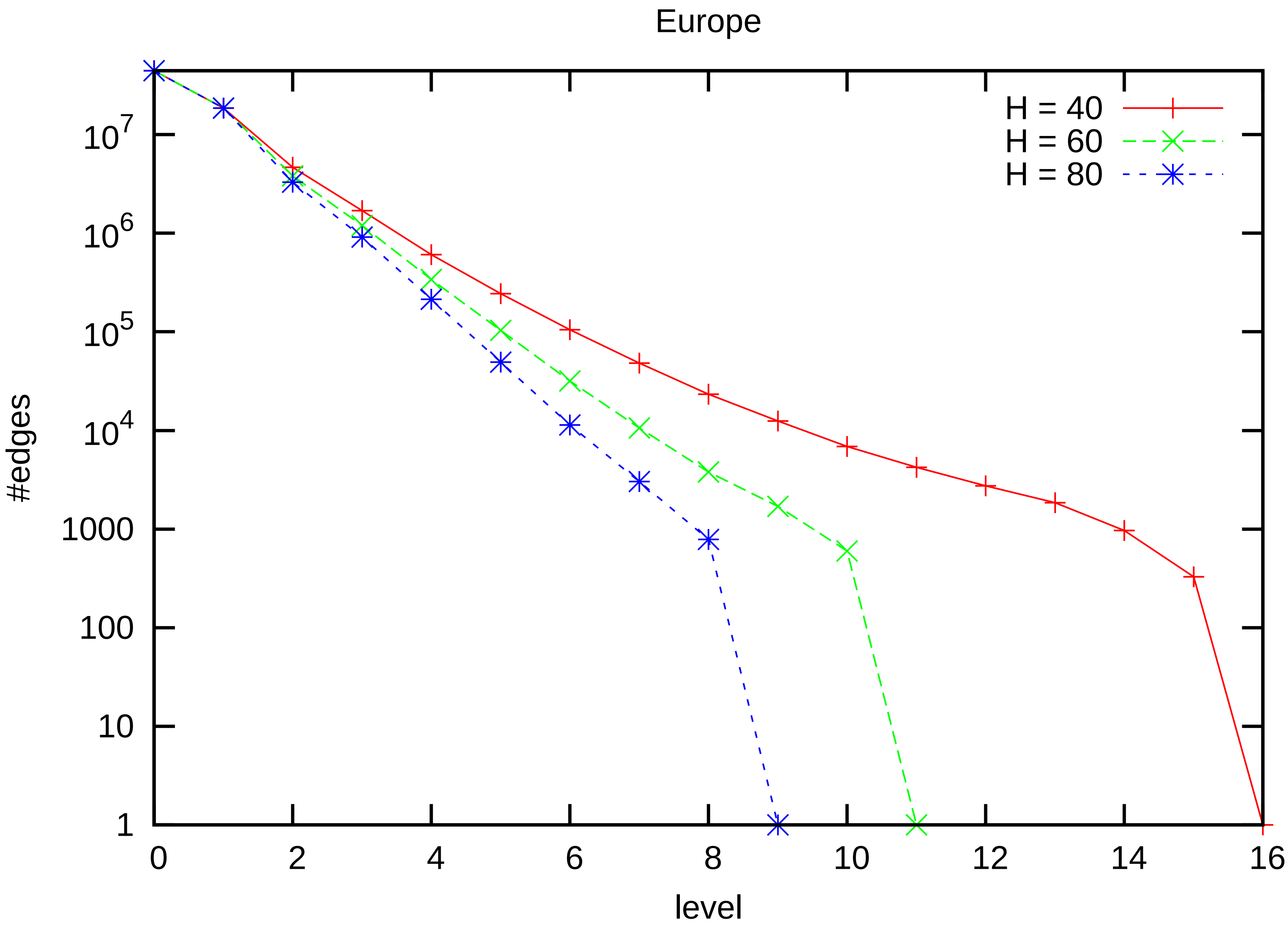


Experiments

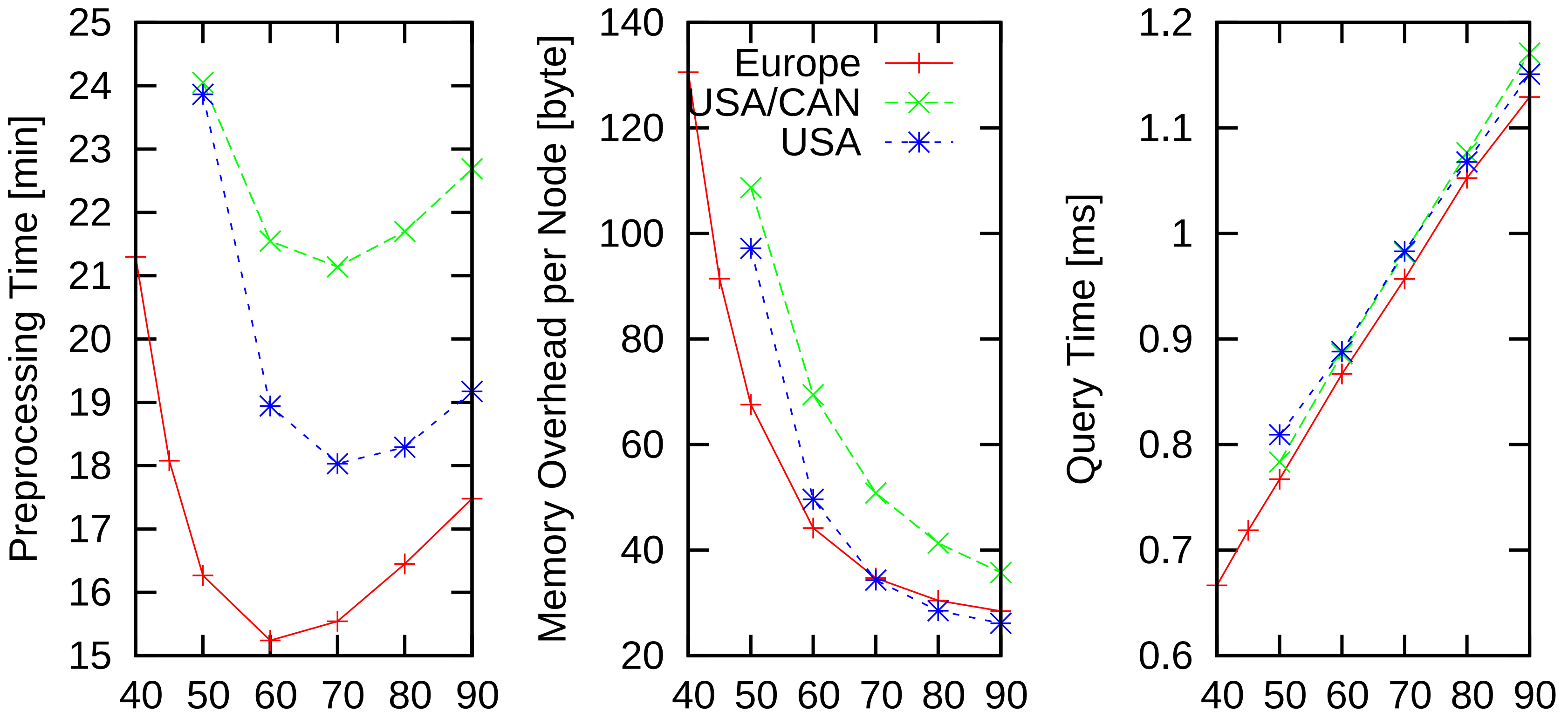
W. Europe (PTV)		USA/CAN (PTV)
18 029 721	#nodes	18 741 705
42 199 587	#directed edges	47 244 849
15	construction [min]	20
0.76	search time [ms]	0.90
8 320	speedup ($\leftrightarrow$ DIJKSTRA)	7 232



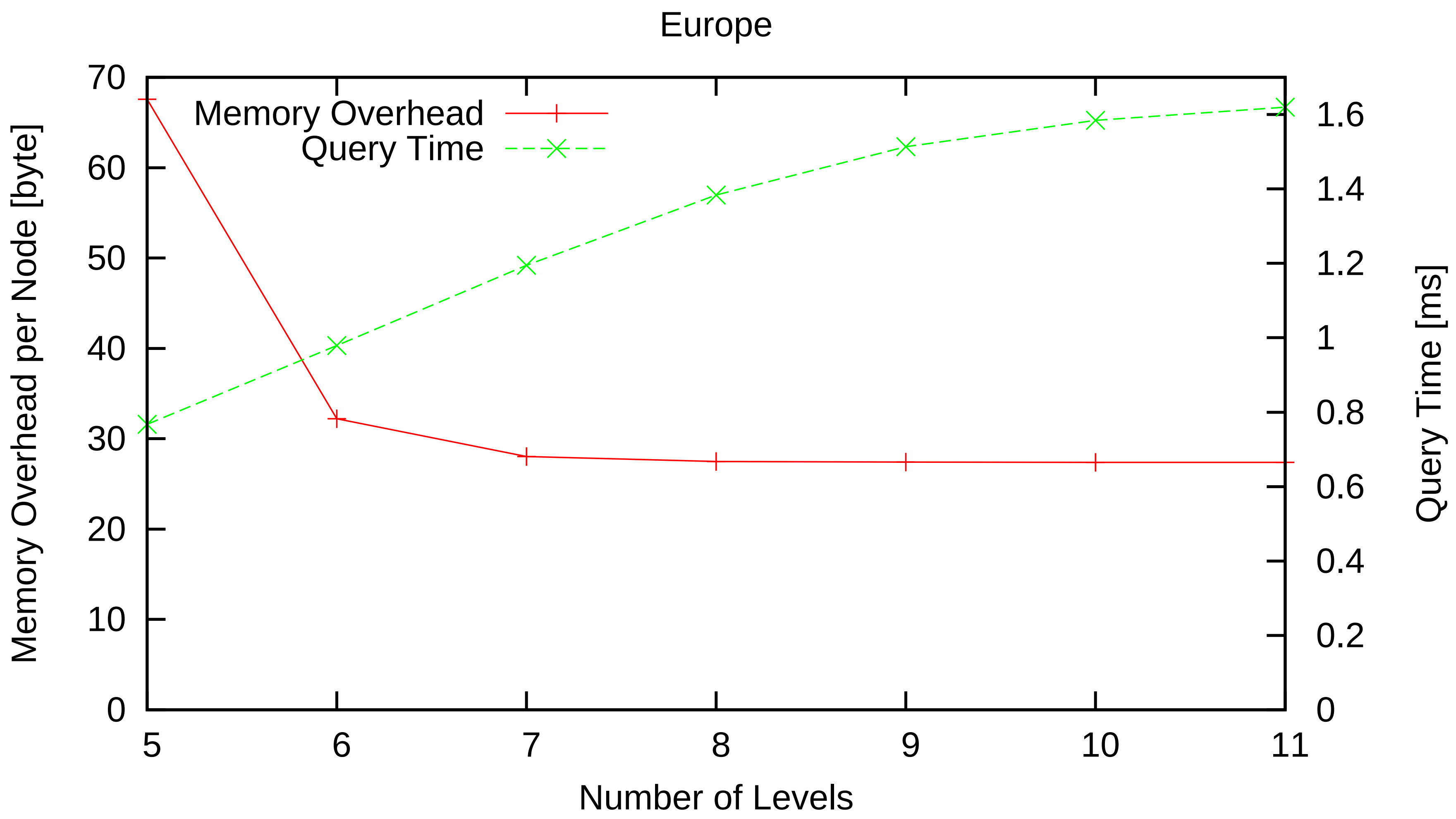
Shrinking of the Highway Networks



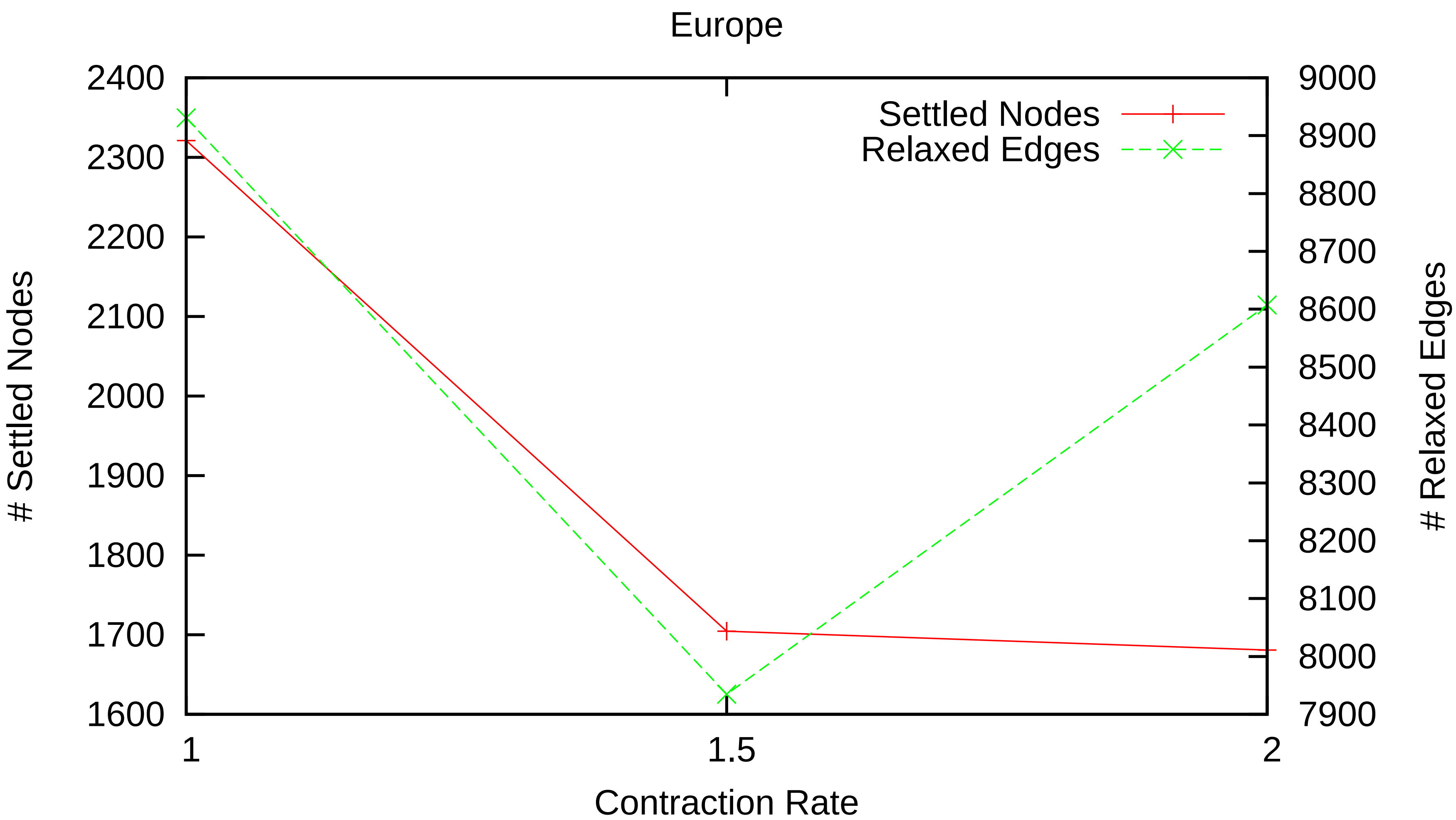
Neighbourhood Size



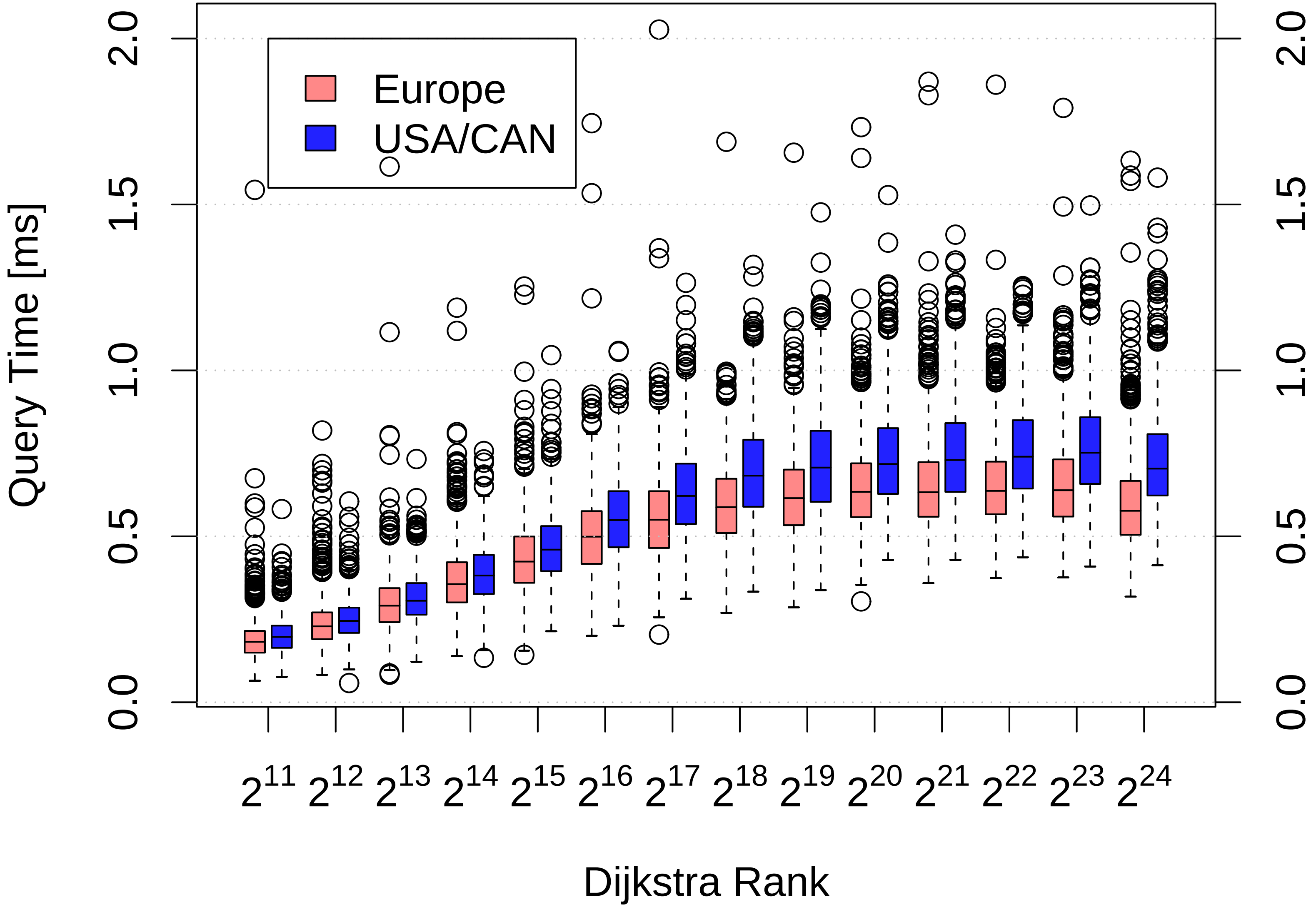
Number of Levels



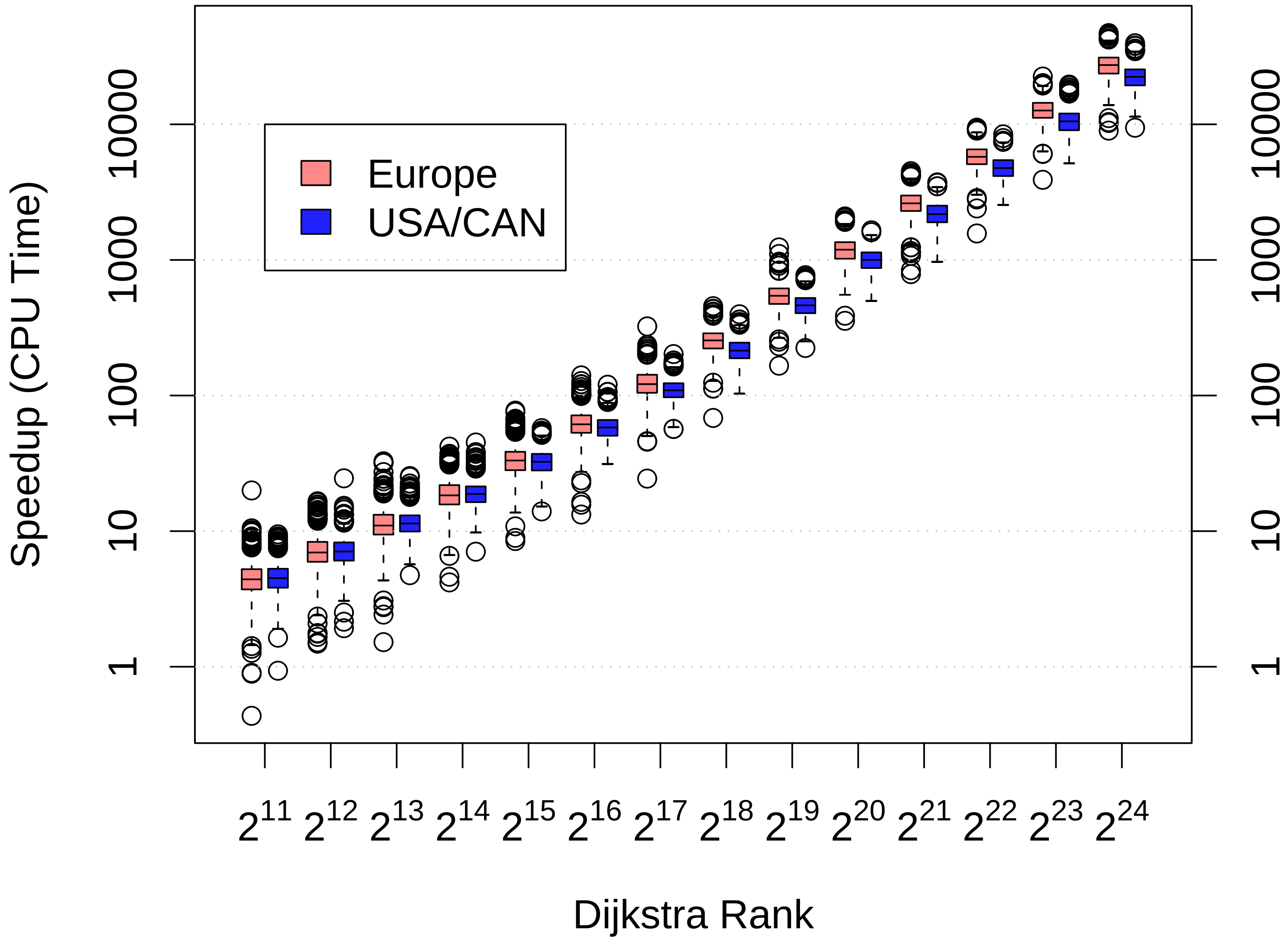
Contraction Rate



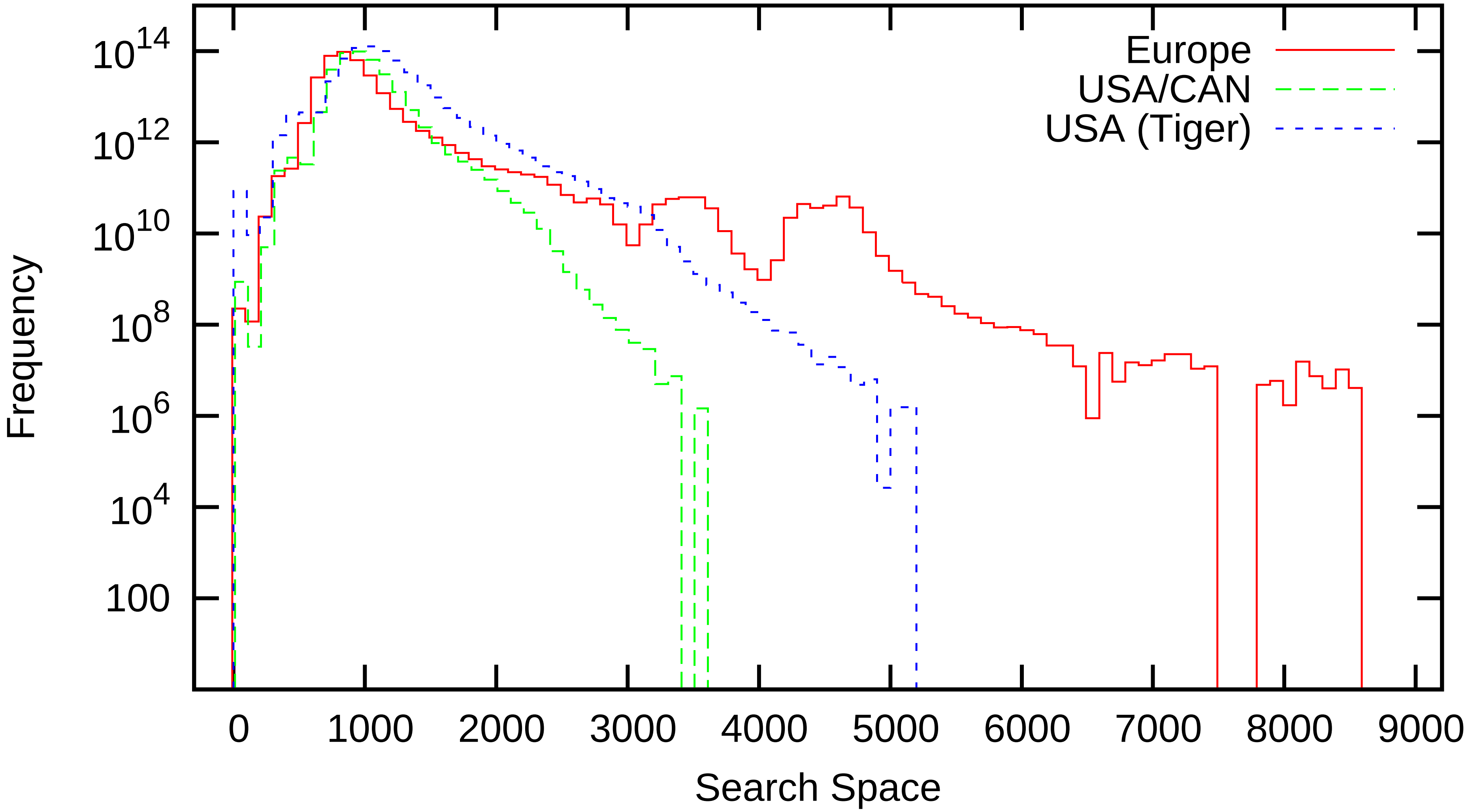
Queries – Time



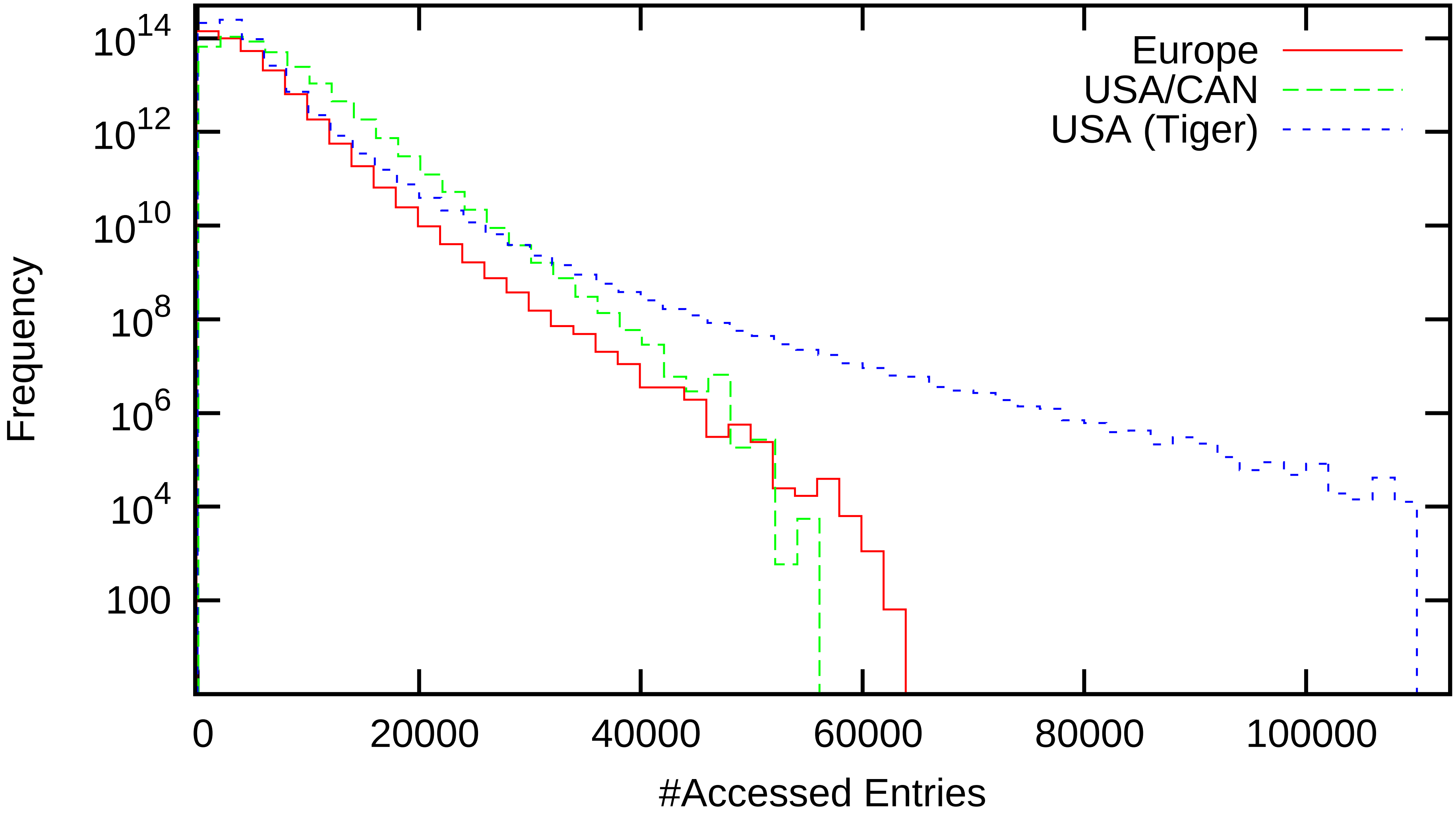
Queries – Speedup



Search Space



Distance Table Accesses



Summary

- exact routes in large street networks — e.g. ≈ 18 million nodes
  - ⇝ quality advantage, advertisement argument
- fast search — < 1 ms
  - ⇝ cheap, energy efficient processors in mobile devices
  - ⇝ low server load
  - ⇝ lots of room for additional functionality
- fast preprocessing — ≈ 20 min
- low space consumption — ≪ data base
- no manual postprocessing of data
  - ⇝ less dependence on data sources
- organic enhancement of existing commercial solutions

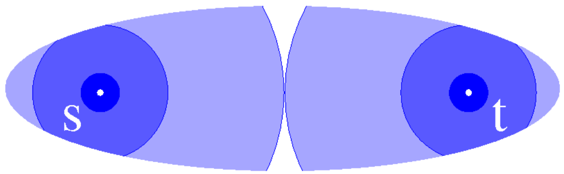
Work in Progress

- computation of $M \times N$ distance tables

  joint work with [Knopp, Schulz][1,2]

- combination with a goal directed approach (landmarks)

  joint work with [Delling, Holzer][1]



[1] Universität Karlsruhe, Algorithmics I Group

[2] PTV AG

Future Work

- fast, local updates on the highway network

  (e.g. for traffic jams)

- implementation for mobile devices

  (flash access . . .)

- multi-criteria shortest paths

  joint work with [Müller-Hannemann, Schnee][3]

- flexible objective functions

---

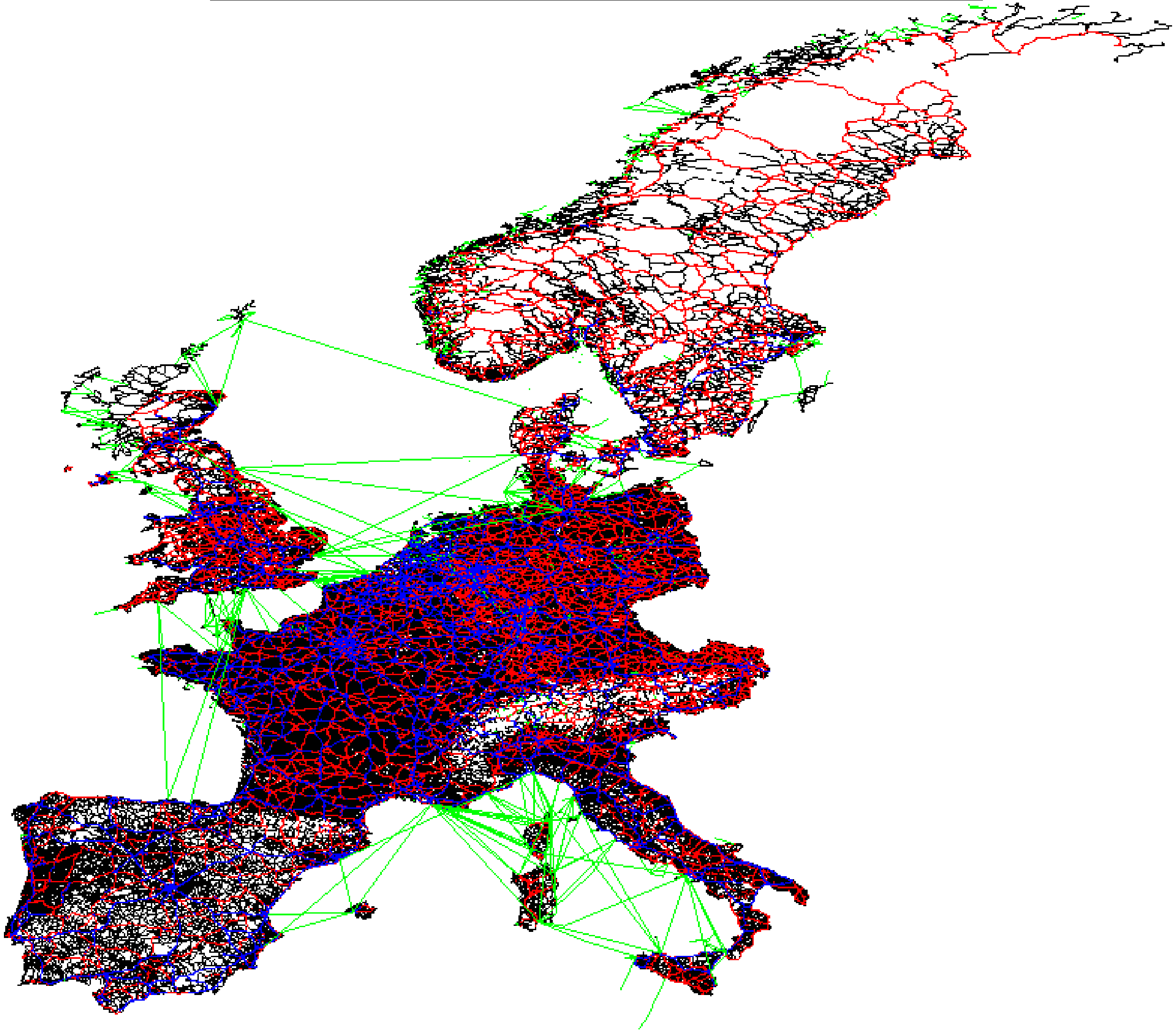
[3] Technische Universität Darmstadt, Algorithmics Group

Industrial Cooperations

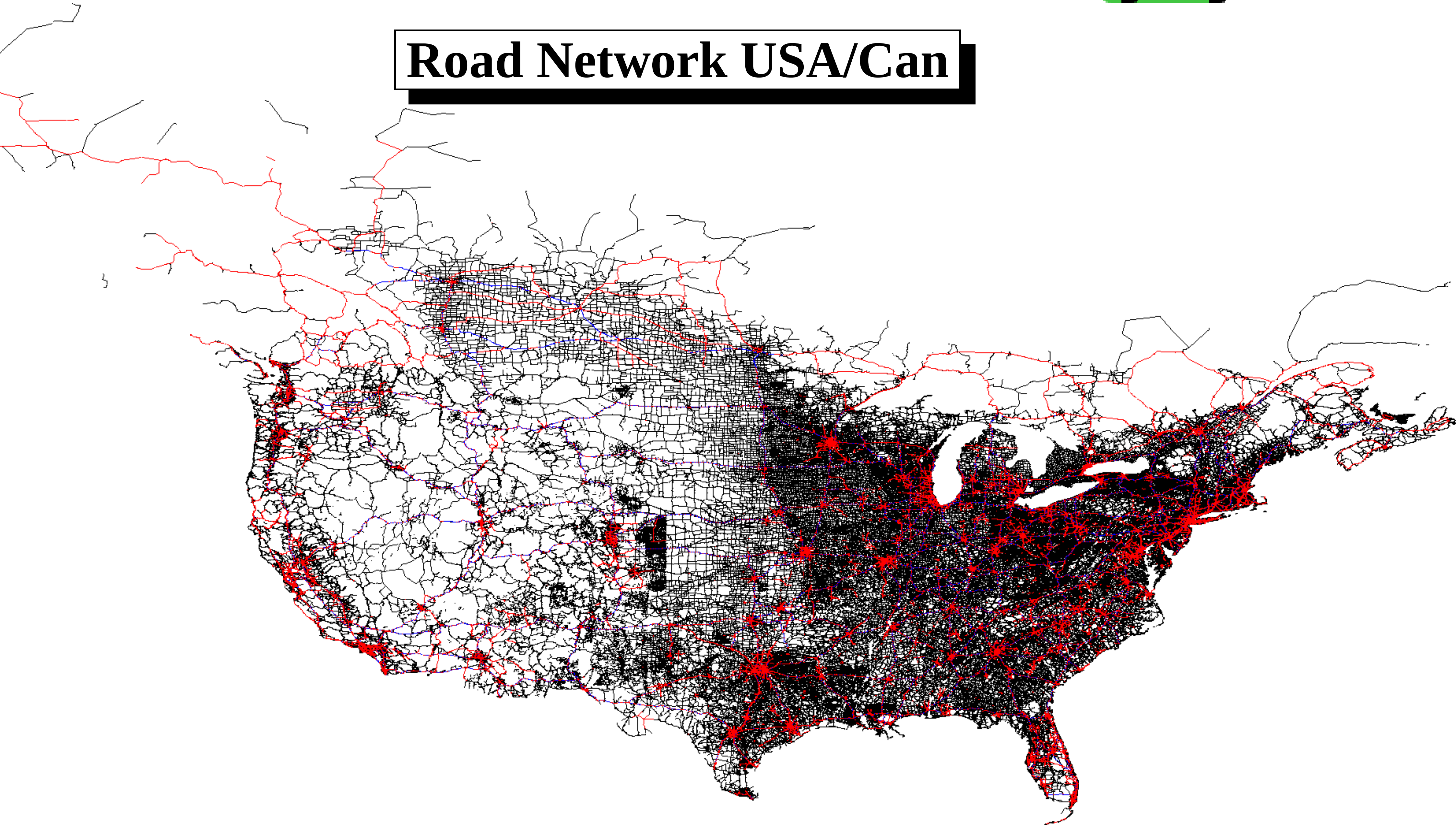
- We help transforming technology into products: consulting . . .
- Joint projects for further features

. . .

Road Network of Europe



Road Network USA/Can



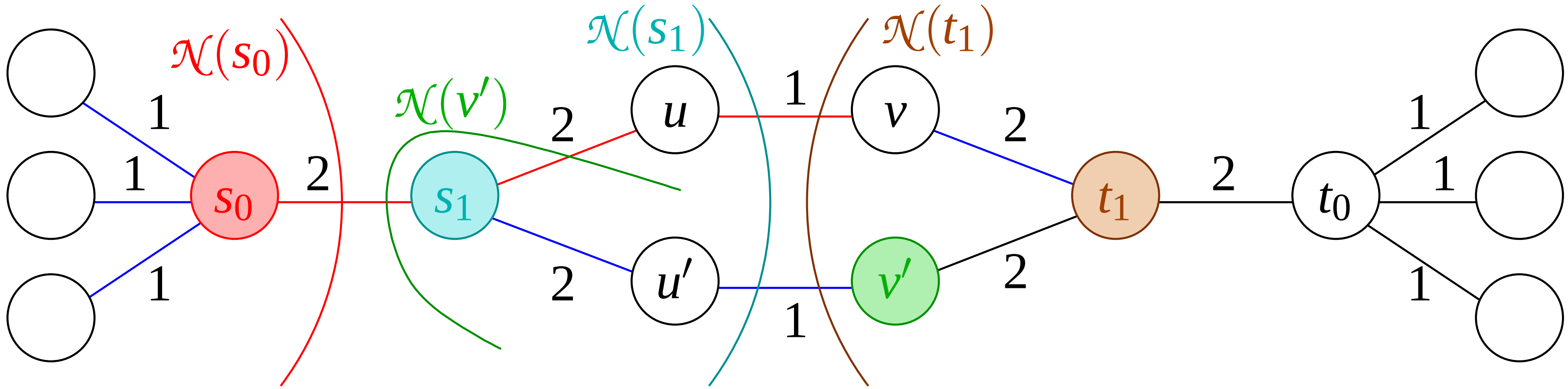
Canonical Shortest Paths

$\mathcal{SP}$: Set of shortest paths

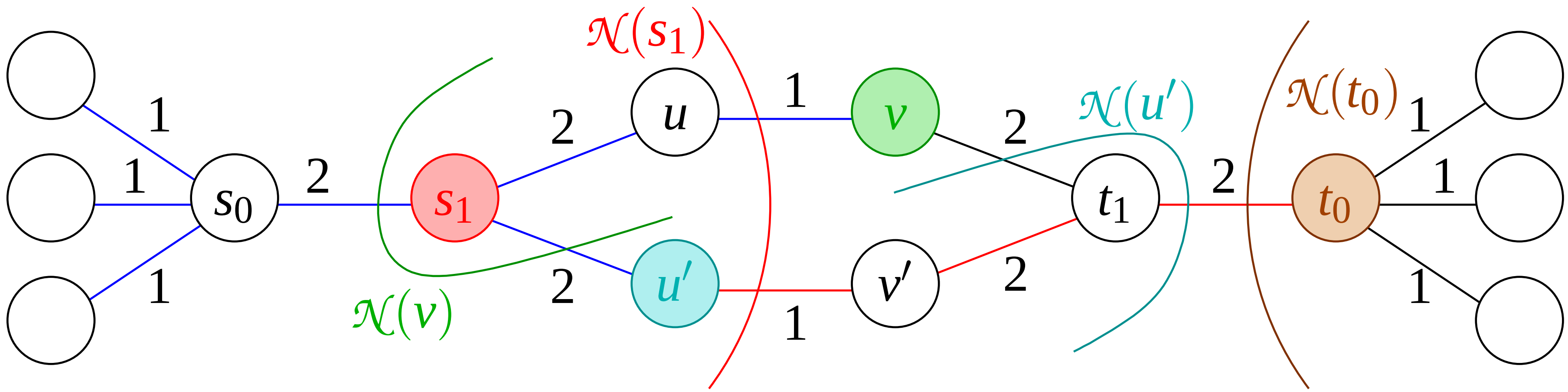
$\mathcal{SP}$ canonical $\Leftrightarrow$

$$\forall P = \langle s, \ldots, s', \ldots, t', \ldots, t \rangle \in \mathcal{SP} : \langle s' \rightarrow t' \rangle \in \mathcal{SP}$$

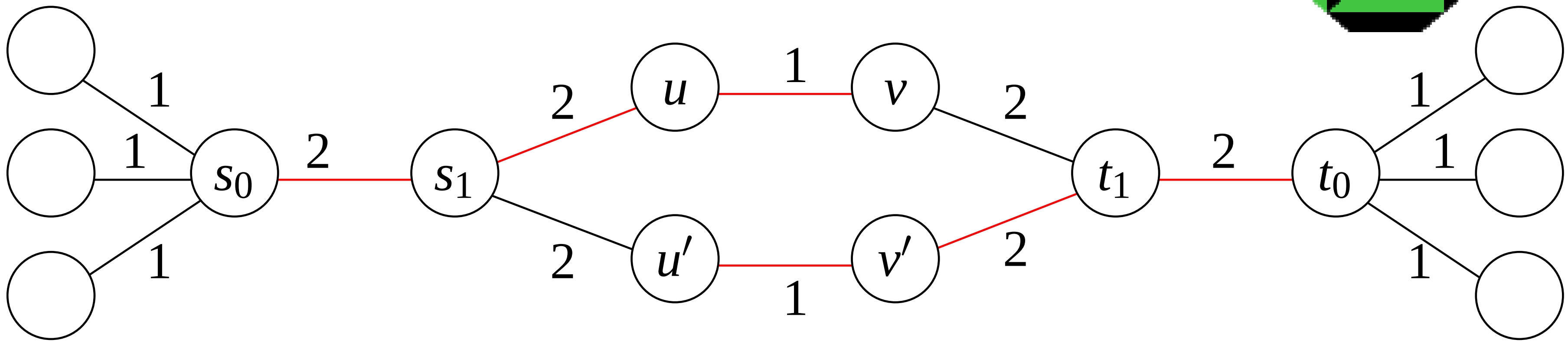
Canonical Shortest Paths



(a) Construction, started from s_0.



(b) Construction, started from s_1.

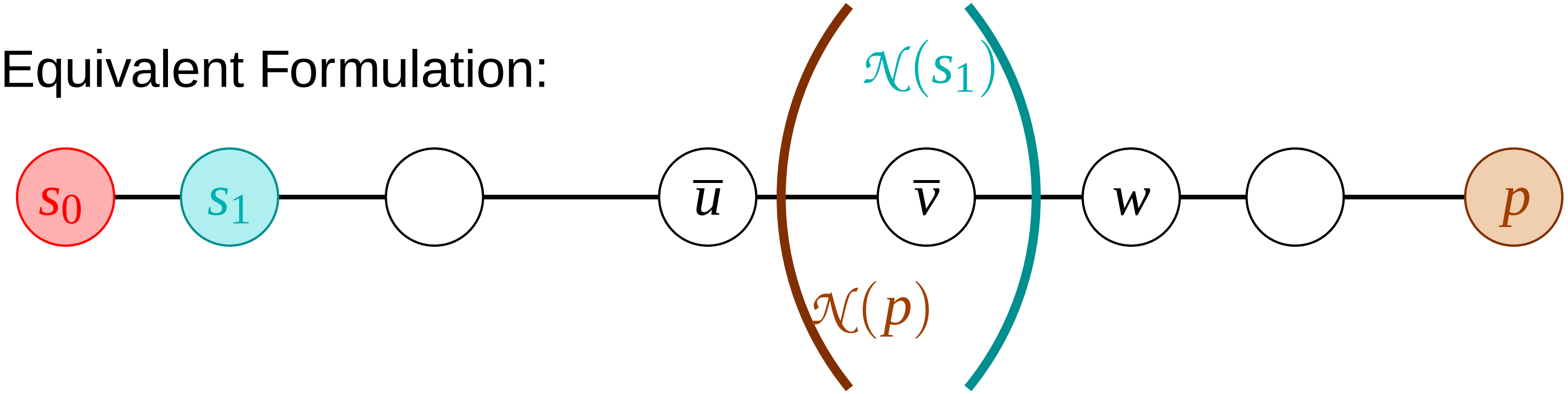


(c) Result of the construction.

Fast Construction

Abort Condition: Efficient Testing.

Equivalent Formulation:



p is set to passive *iff*

$$d(s_1,\overline{v}) \le r(s_1) < d(s_1,w) \wedge$$

$$\overline{v} \prec p \wedge d(\overline{u},p) > r(p)$$