A Hybrid Algorithm for Finding Shortest Paths Between Arbitrary Coordinates using a Combination of Network and Geometric Routing

Hauke Stieler

Universität Hamburg
Faculty of Mathematics, Informatics and Natural Sciences
Department of Informatics
Databases and Information Systems

August 28, 2023

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Outline

Pedestrian routing applications

Realistic routes beneficial for

- real-world end-user applications
  - → indoor & campus navigation apps

Pedestrian routing applications

Realistic routes beneficial for

- real-world end-user applications
  - → indoor & campus navigation apps
- higher level applications
  - → isochrones, walkability analysis

Pedestrian routing applications

Realistic routes beneficial for

- real-world end-user applications
  → indoor & campus navigation apps
- higher level applications
  → isochrones, walkability analysis
- **agent-based simulations**
  → crowd behavior in evacuation scenario

Routing techniques

Graph-based:

- abstraction of real world: edges represent roads / ways
- Dijkstra, A*, speedup techniques
- often detailed attributes in edges

Routing techniques

Graph-based:

- abstraction of real world: edges represent roads / ways
- Dijkstra, A*, speedup techniques
- often detailed attributes in edges
- only locations on graph are reachable $\rightarrow$ unable to traverse open spaces

Routing techniques

Graph-based:

- abstraction of real world: edges represent roads / ways
- Dijkstra, A*, speedup techniques
- often detailed attributes in edges
- only locations on graph are reachable $\rightarrow$ unable to traverse open spaces

Geometric routing:

- shortest path in presence of obstacles (i.e. lines and polygons)
- two strategies:
  - graph generation (e.g. visibility graphs) + graph-based routing
  - continuous Dijkstra paradigm

Routing techniques

Graph-based:

- abstraction of real world: edges represent roads / ways
- Dijkstra, A*, speedup techniques
- often detailed attributes in edges
- only locations on graph are reachable $\rightarrow$ unable to traverse open spaces

Geometric routing:

- shortest path in presence of obstacles (i.e. lines and polygons)
- two strategies:
  - graph generation (e.g. visibility graphs) + graph-based routing
  - continuous Dijkstra paradigm
- not always realistic (high dependence on data quality)

Example: Graph-based and geometric routing

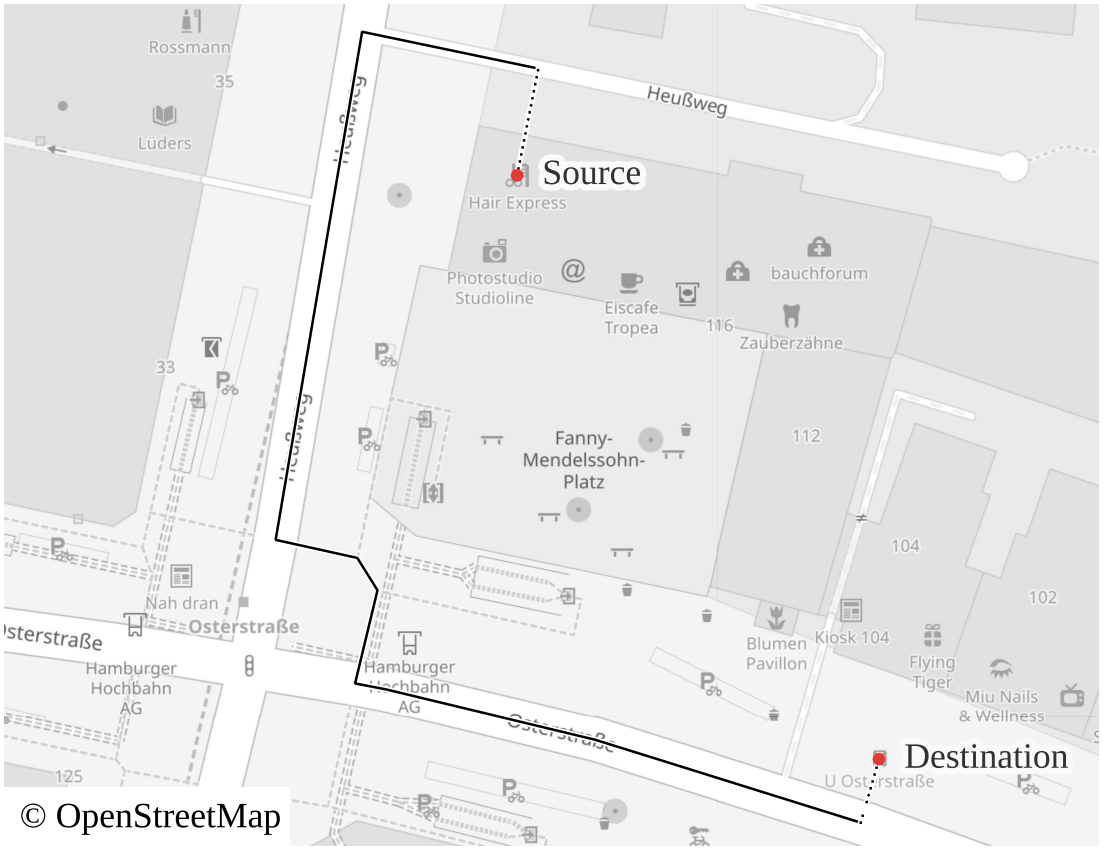


Figure 1: Graph-based routing result using the routing engine *GraphHopper*.

Example: Graph-based and geometric routing

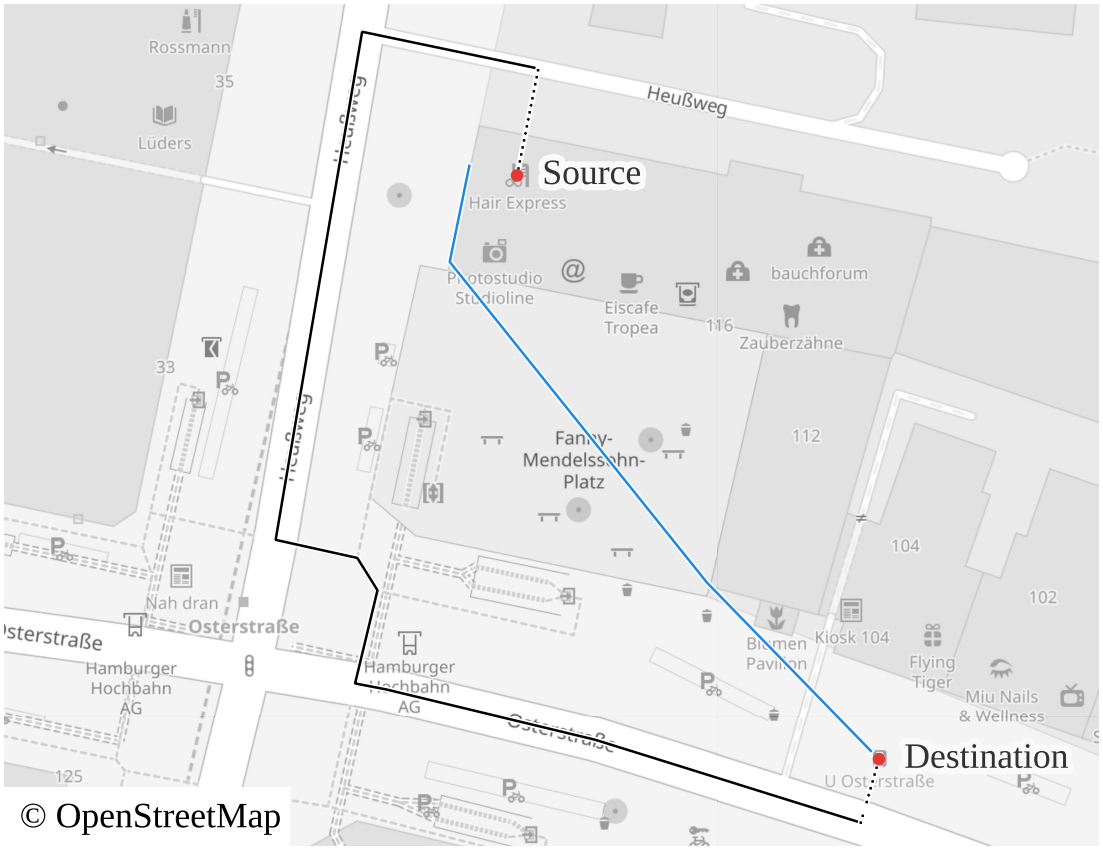


Figure 2: Graph-based route (black) compared to the purely geometric route (blue).

Example: Graph-based and geometric routing

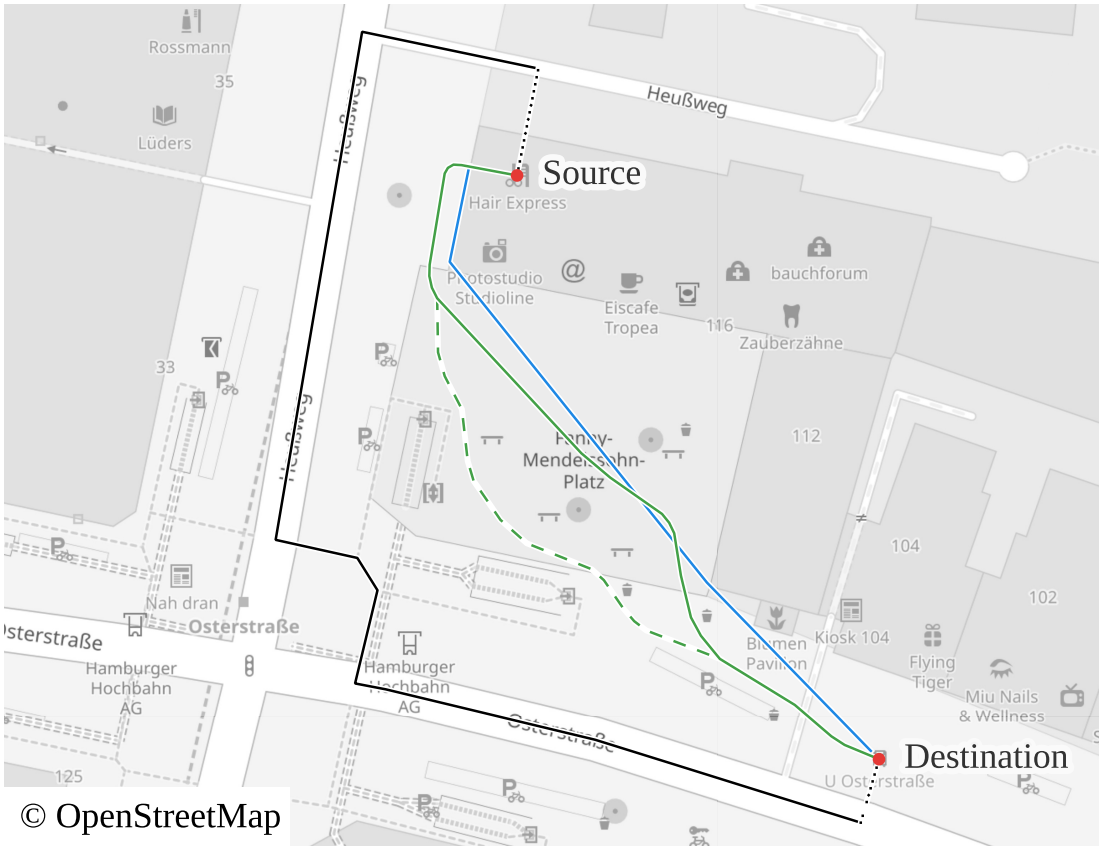


Figure 3: Graph-based (black), purely geometric (blue) and expected real-world route (green).

Value of road data

Road network is still valuable:

- attributes on edges
  - examples: surface conditions, road width, access restrictions
- definitely walkable
  - bridges, tunnels, building passages

Wanted routing algorithm

Should be able to

- use roads and ways
- traverse open spaces $\rightarrow$ reach arbitrary locations
- create realistic routes

Wanted routing algorithm

Should be able to

- use roads and ways
- traverse open spaces $\rightarrow$ reach arbitrary locations
- create realistic routes

Based on C# / .NET, MARS (Multi-Agent Research and Simulation) and NTS (NetTopologySuite)

Design decisions

- visibility graph generation
  - custom algorithm
- merge with existing road network
- connect source / destination with visibility edges
- usage of graph-based routing

Components

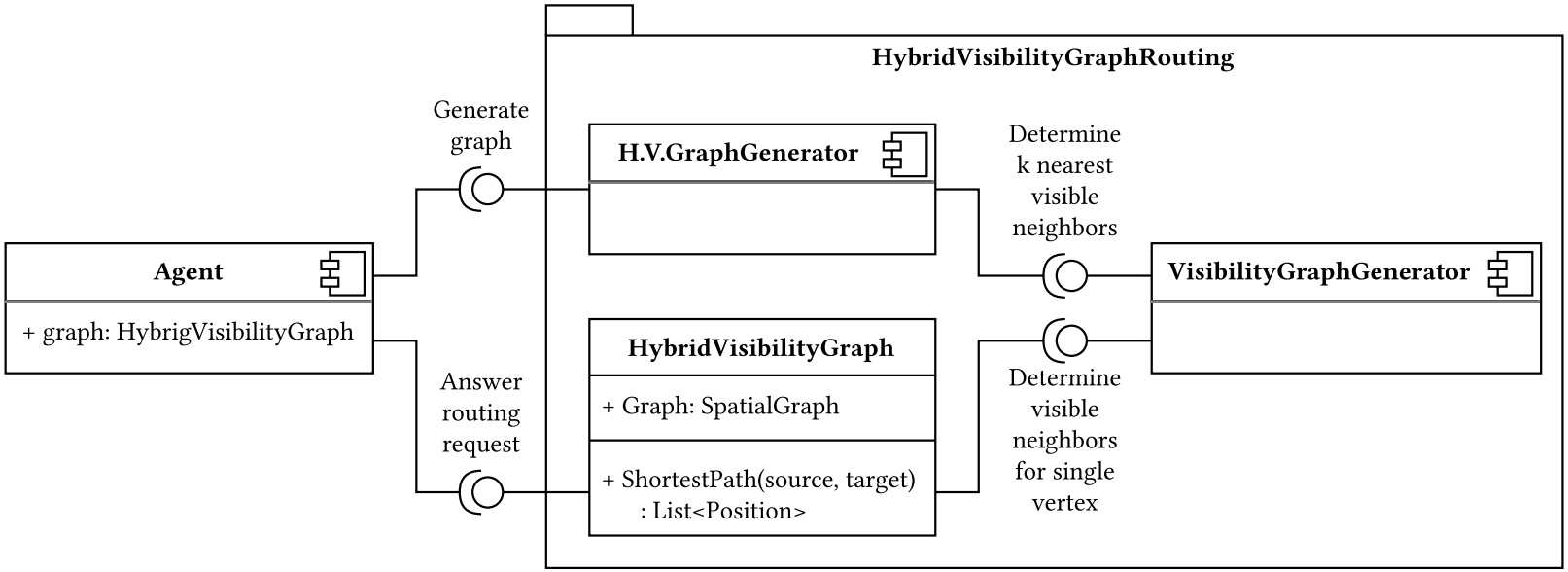


Figure 4: Components of the implemented hybrid routing algorithm.

Considerations and terminology

Key considerations:

- graph generation has to support
  - collinear vertices
  - arbitrary obstacle geometries (especially lines and polygons)
  - intersecting and touching obstacles
  - arbitrary positions and non-unique coordinates

  → custom implementation instead of algorithm from the literature
- generate and connect nodes suitable for routing

Considerations and terminology

Key considerations:

- graph generation has to support
  - collinear vertices
  - arbitrary obstacle geometries (especially lines and polygons)
  - intersecting and touching obstacles
  - arbitrary positions and non-unique coordinates

  → custom implementation instead of algorithm from the literature
- generate and connect nodes suitable for routing

Terminology:

obstacle neighbor	neighbor of v on same or touching obstacle
visibility neighbor	visible neighbor of v across open space
input vertex	vertex / coordinate in the input dataset
output vertex	generated vertex in the visibility graph

Hybrid visibility graph creation – Overview

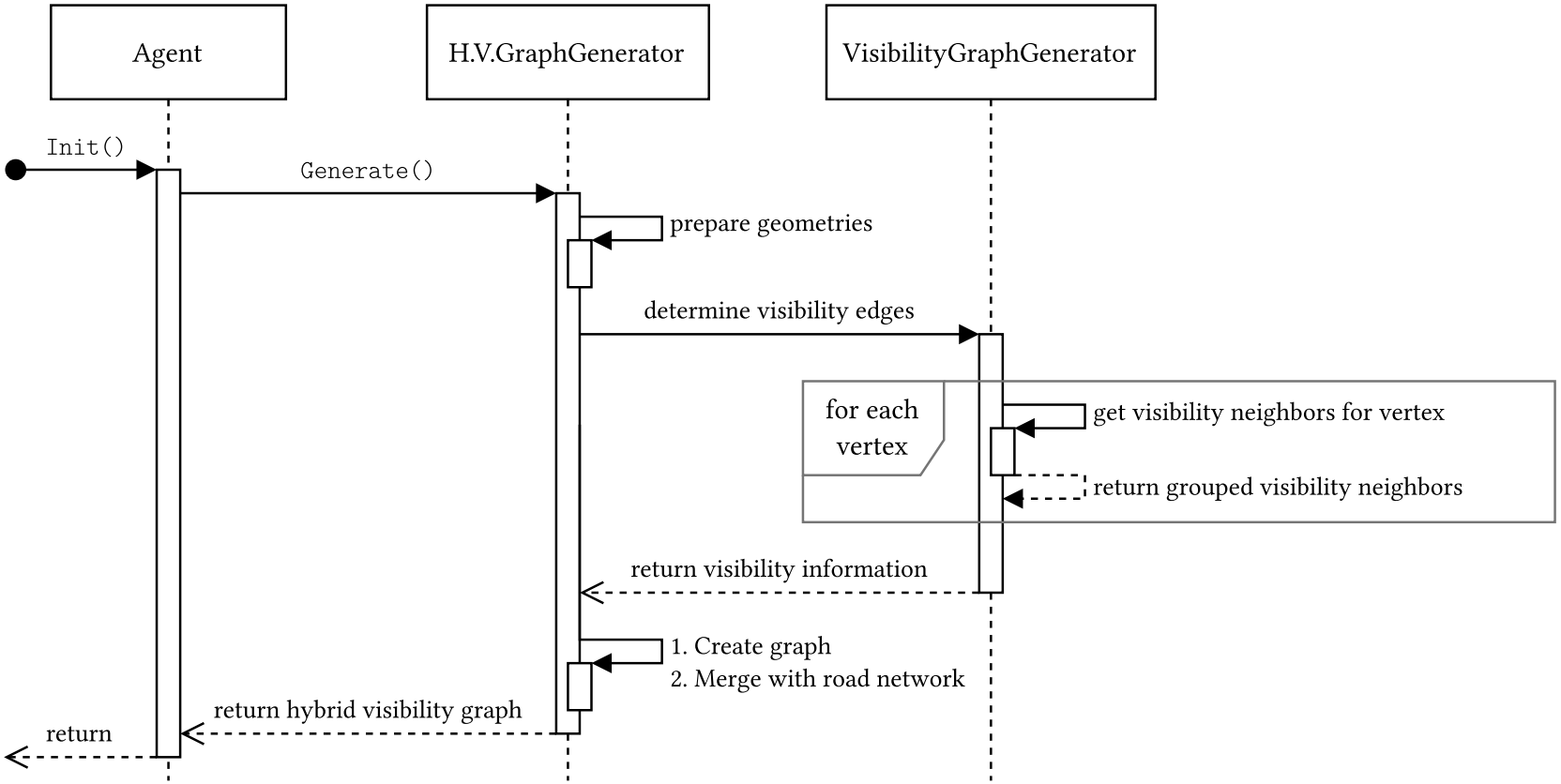


Figure 5: Steps of the graph generation.

Hybrid visibility graph creation – Overview

1. get and prepare obstacle geometries
   - unwrap multi-geometries
   - triangulate polygonal obstacles
     → performance optimization
2. obtain data for visibility graph
   2.1 for each vertex: determine its visibility neighbors
   2.2 group visibility neighbors based on obstacle neighbors
3. generate routable visibility graph
4. merge road edges

Routable visibility graph creation

Naive approach:

- one vertex per input coordinate
- routing through line-based obstacles possible

Routable visibility graph creation

Naive approach:

- one vertex per input coordinate
- routing through line-based obstacles possible

My implementation:

- multiple vertices at same location
- one vertex between adjacent obstacle neighbors
- vertices are not connected
- routing through obstacle not possible

Routable visibility graph creation

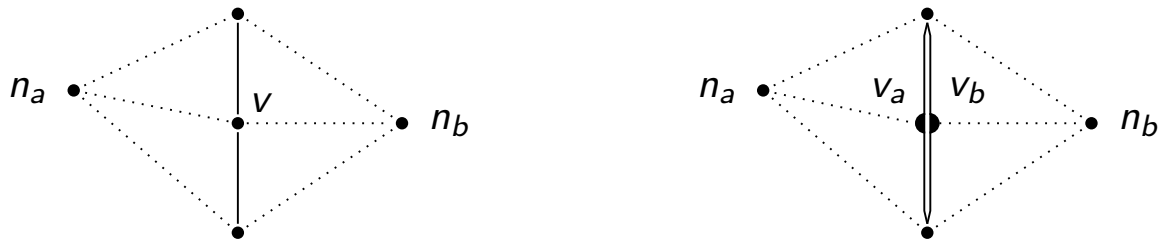


Figure 6: Naive (left) and the implemented (right) vertex creation and connection.

Routable visibility graph creation

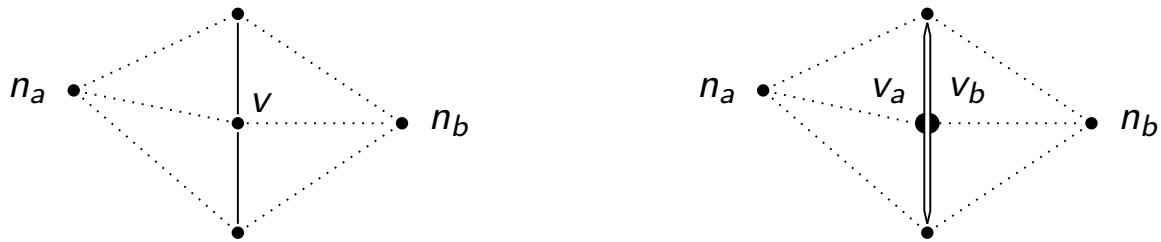


Figure 6: Naive (left) and the implemented (right) vertex creation and connection.

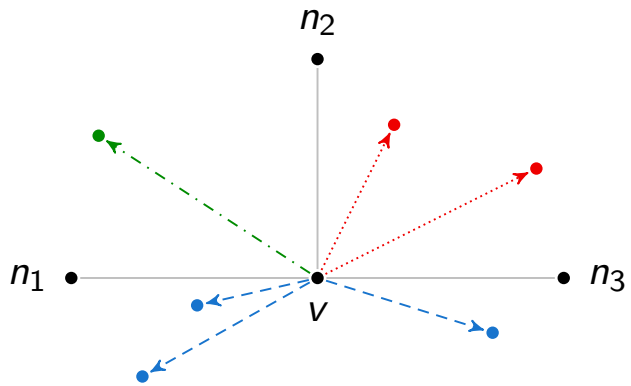


Figure 7: Grouping visibility neighbors into bins, based on the three obstacle neighbors of v.

Merging the road network

1. create vertices at intersection points
2. connect new vertex
3. remove old edges

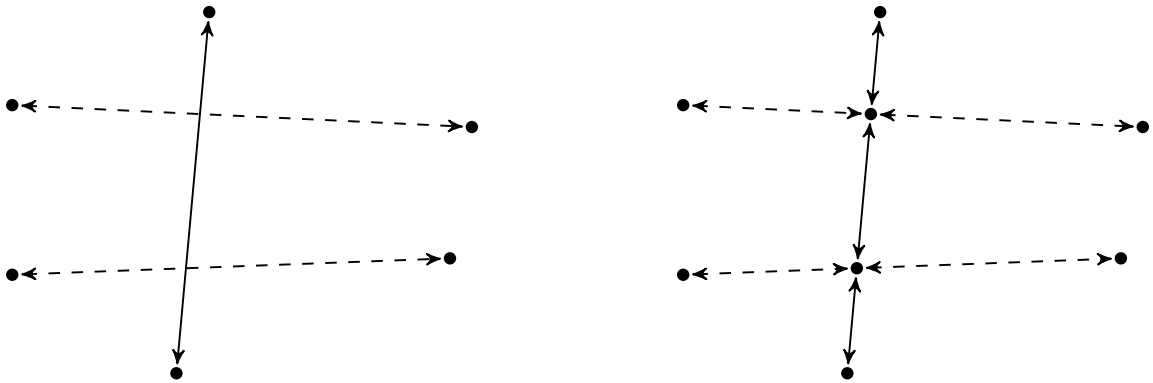


Figure 8: Merge of a road edge (solid line) and two visibility edges (dashed lines).

Answering routing queries

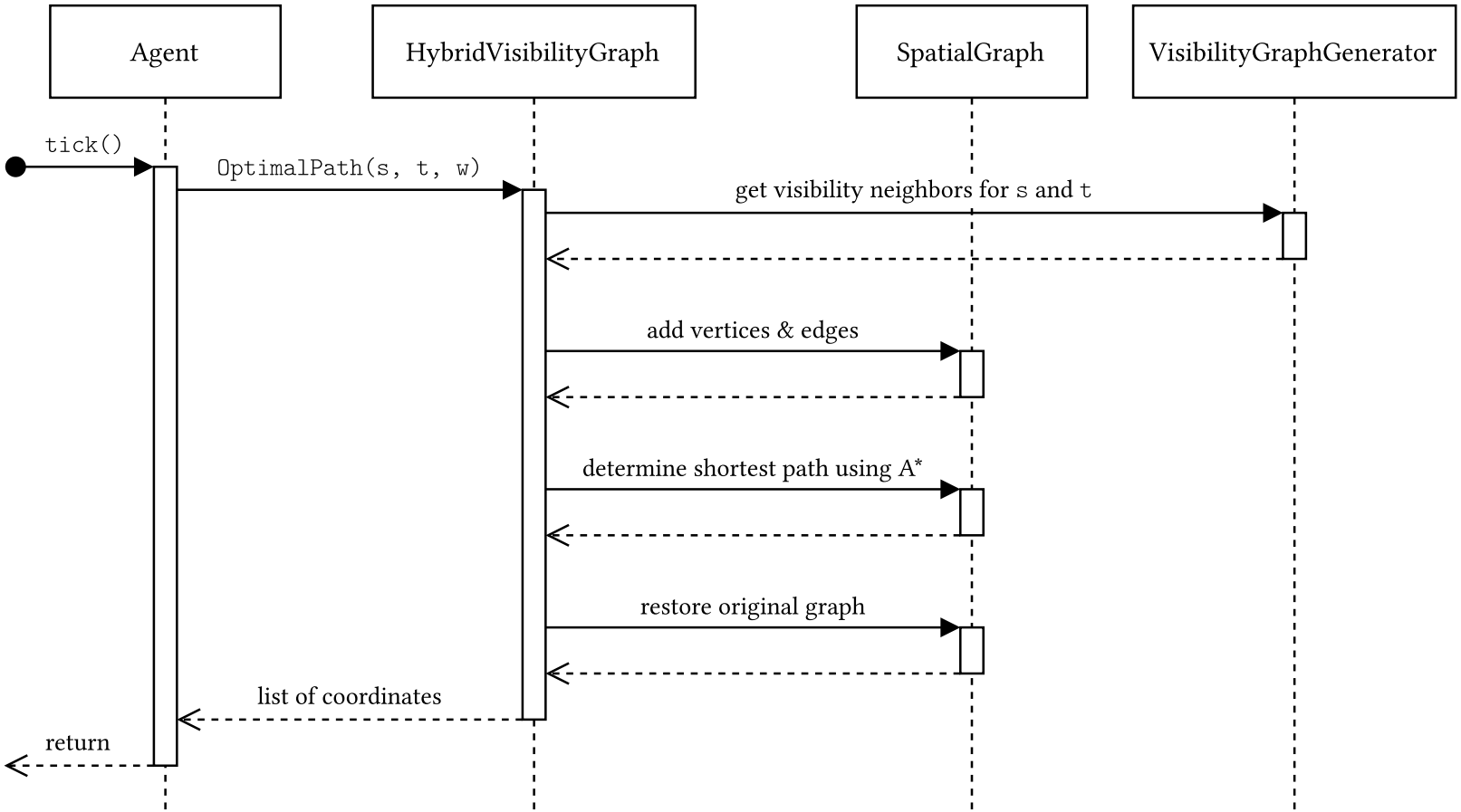


Figure 9: Steps of answering a routing request.

Performance optimizations – Shadow areas

Exclude vertices in "shadow" of obstacles:

- consider vertex v and think of it as light bulb
- obstacles cast shadow outwards, hiding vertices

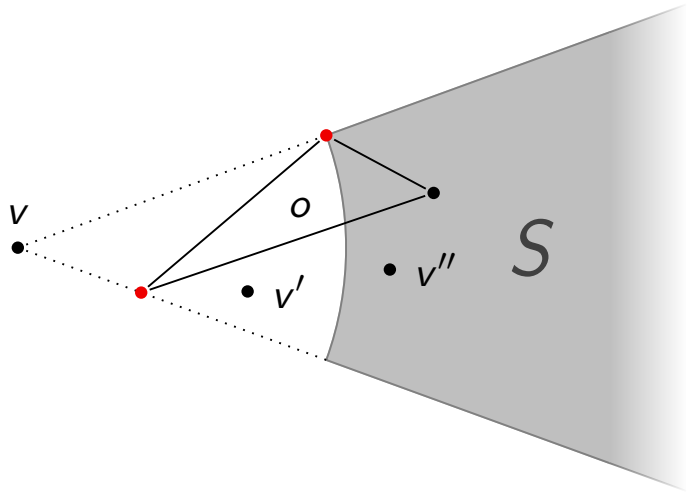


Figure 10: Shadow area S cast by obstacle o for vertex v.

Performance optimizations

Additional optimizations:

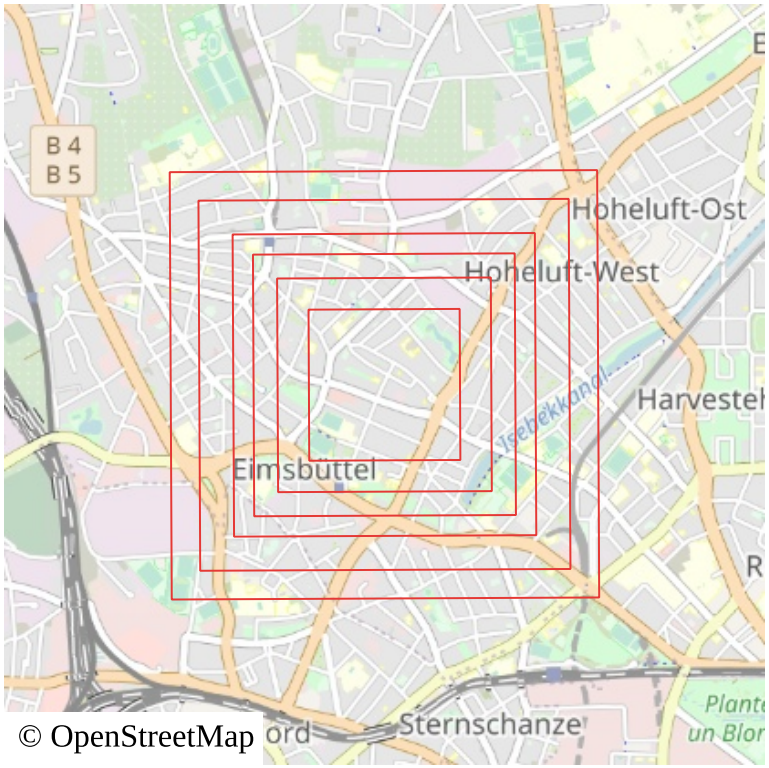
- usage of (spatial) indices
- custom intersection checks
- limiting number of visibility neighbors (kNN search)
- considering only vertices on convex hulls
- vertex filtering by irrelevant angular ranges

Impact of optimizations

Enabled optimization	Time	Speedup
Shadow areas	22.2 s	35.56
kNN filtering	751.7 s	1.05
Vertices on convex hull	287.9 s	2.74
Valid angle areas	322.5 s	2.44
Custom collision detection	769.3 s	1.02
No active optimization	788.5 s	1.00
All optimizations active	11.1 s	71.13

Table 1: Impact of optimizations on the graph generation using the 0.5 km^2 "OSM city" dataset.

Real-world datasets



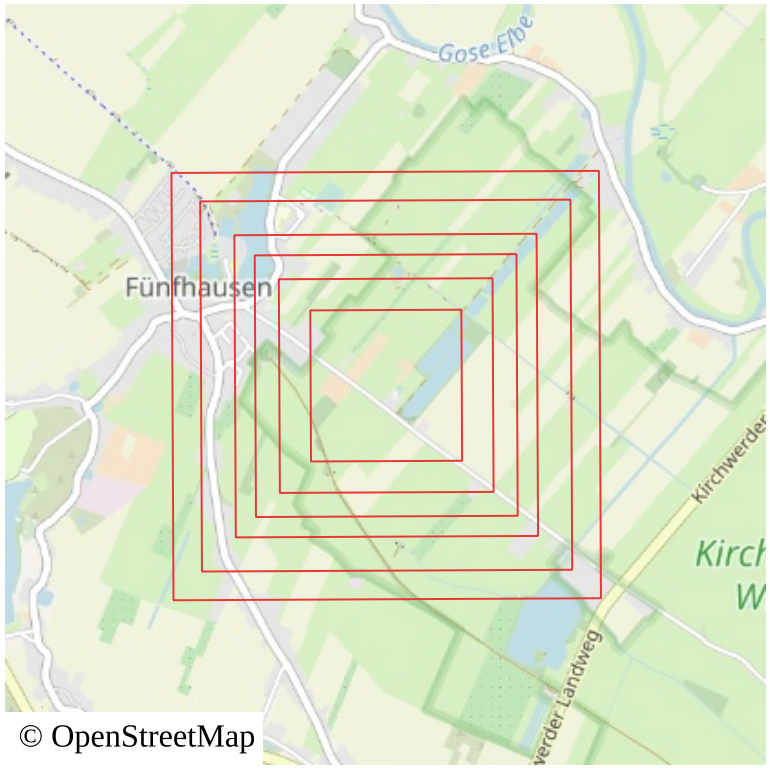


Figure 11: Extents of real-world datasets "OSM city" (left) and "OSM rural" (right) from the OpenStreetMap database.

Graph generation performance

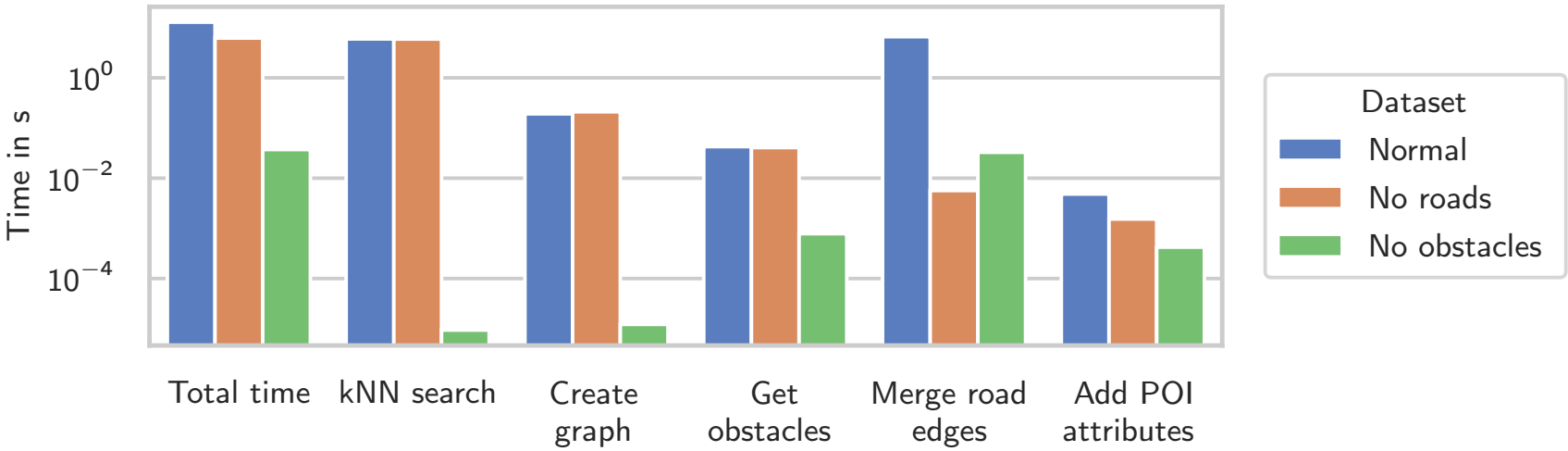


Figure 12: Time of graph generation tasks on different variants of the 4 km^2 "OSM city" dataset.

Graph generation performance

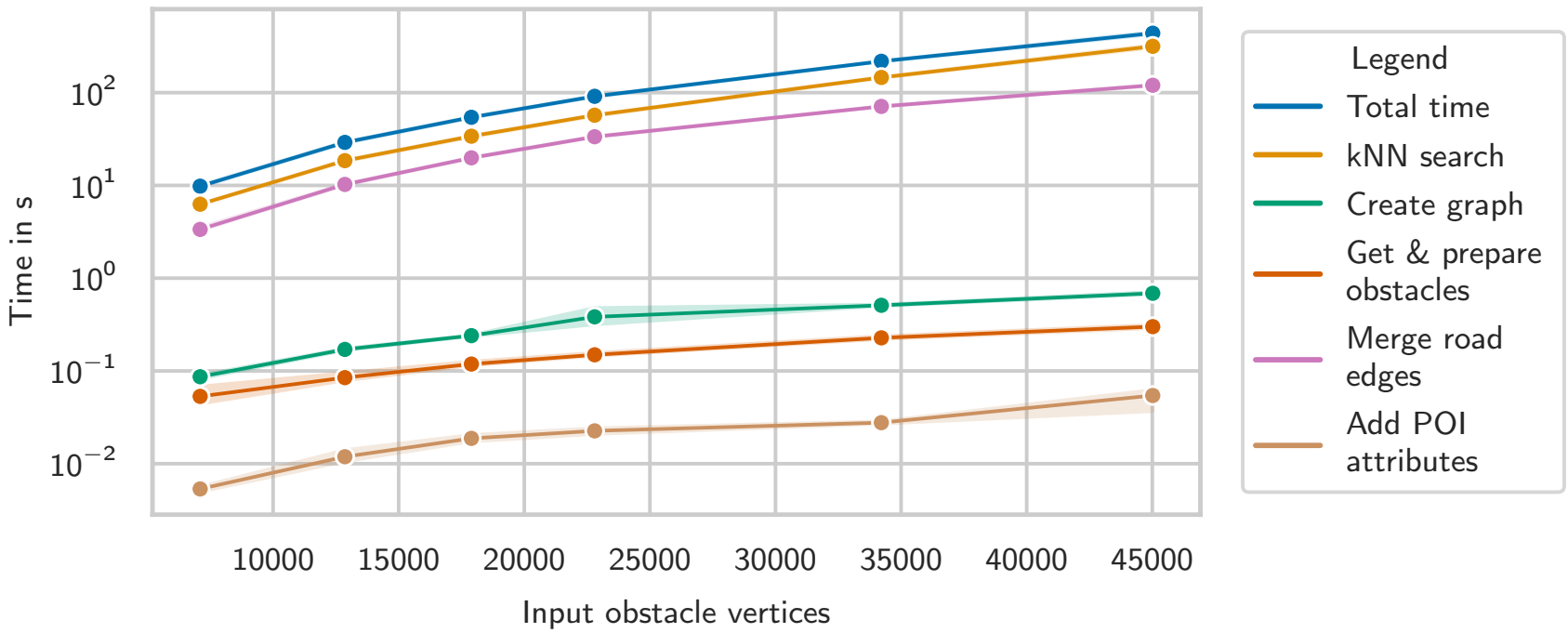


Figure 13: Time of graph generation tasks on all "OSM city" datasets.

Routing performance

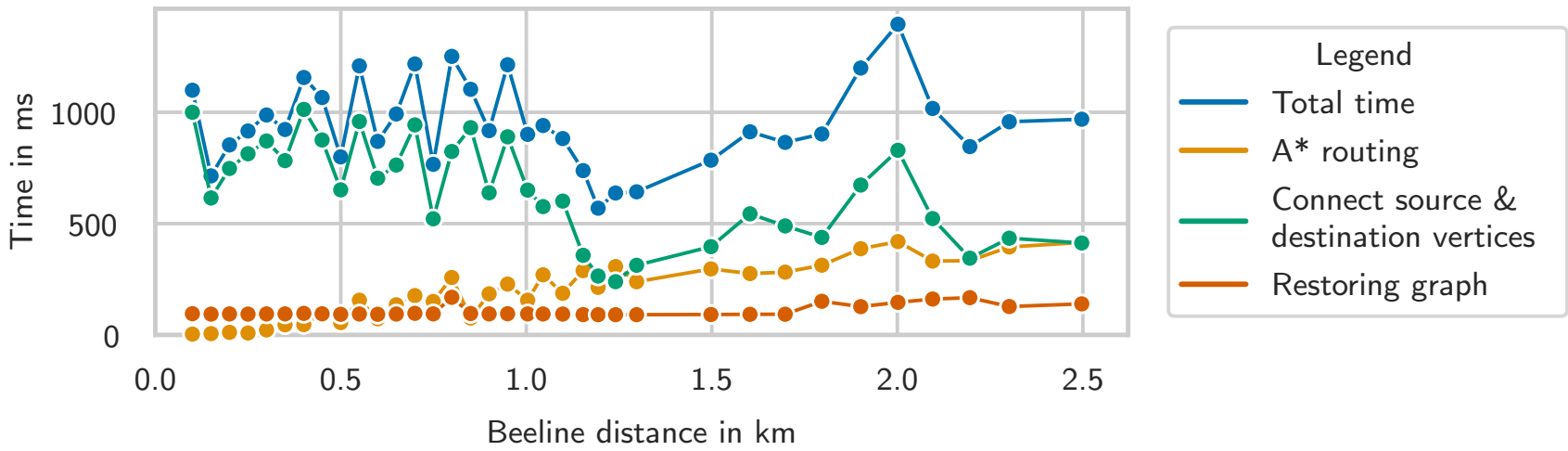


Figure 14: Time needed by routing task for differently distant requests using the 4 km^2 "OSM city" dataset.

Memory usage

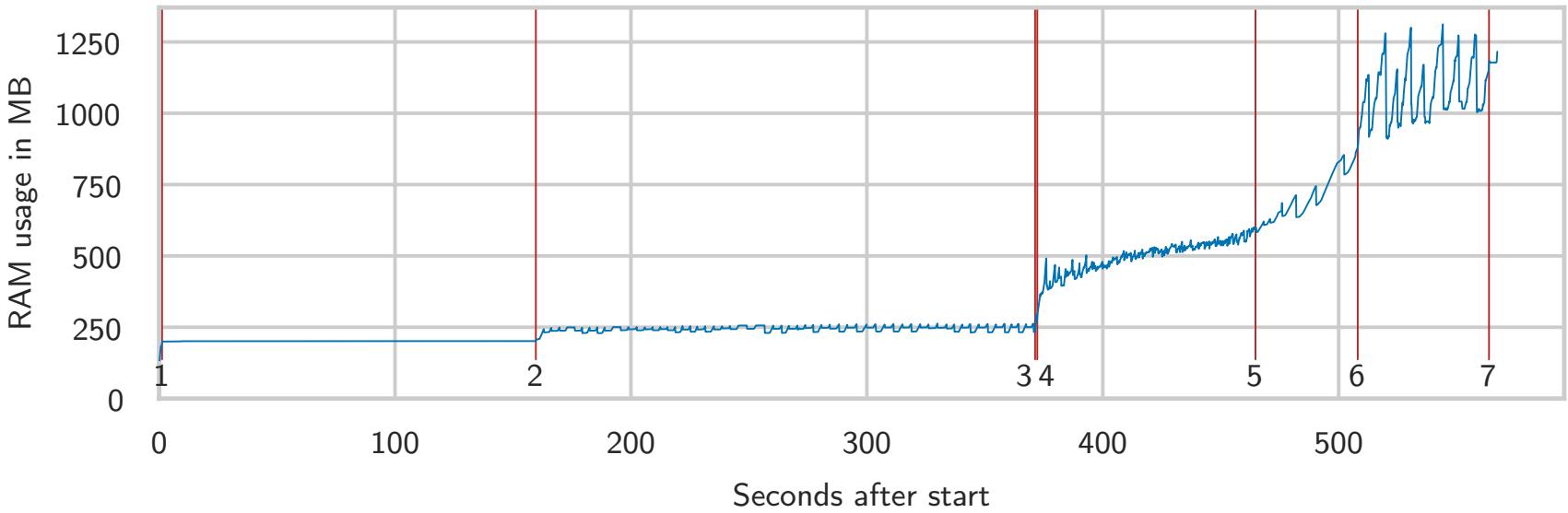


Figure 15: Memory usage of the 4 km^2 "OSM city" dataset during graph generation (1-6) and routing (6-7).

Route quality – Example (urban)

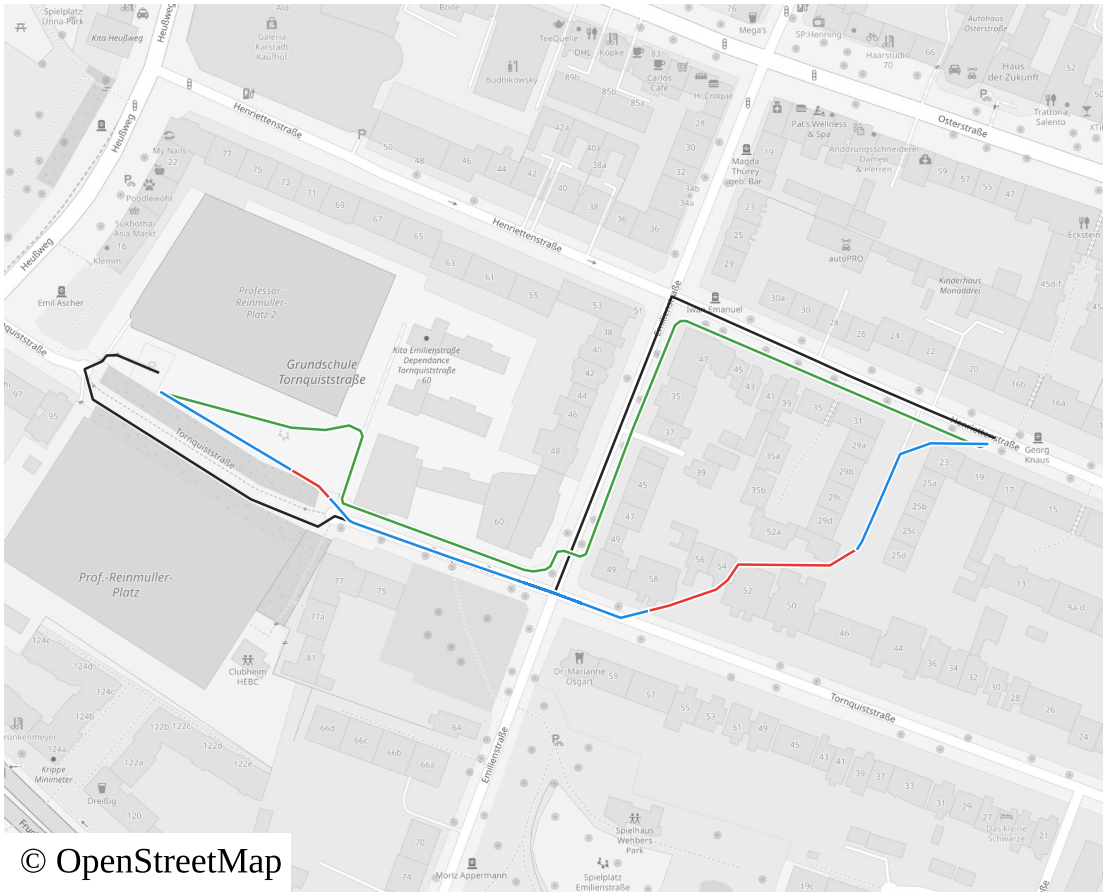


Figure 16: Expected (green), graph-based (black) and actual (blue) routes. Passages not usable in the real world are marked in red.

Route quality – Example (urban)

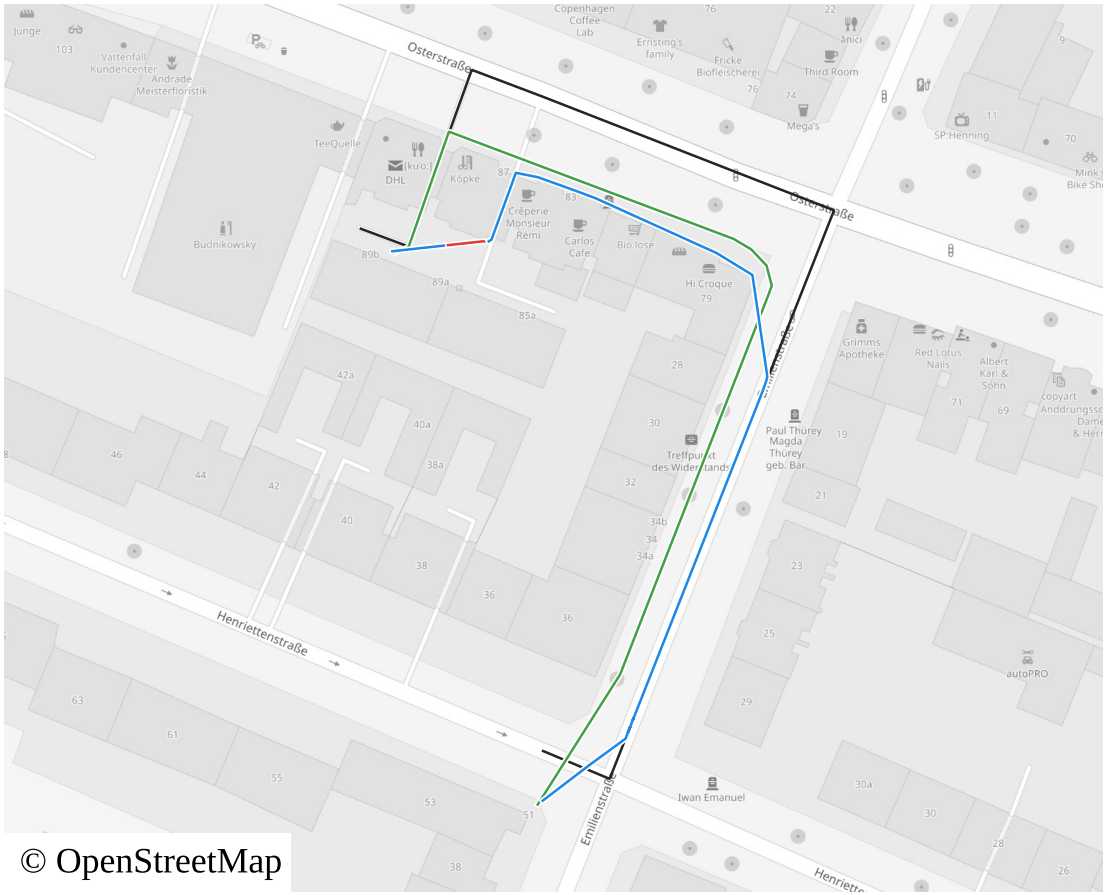


Figure 17: Expected (green), graph-based (black) and actual (blue) routes. Faulty passages marked in red.

Route quality – Graph-based route

Figure 18: Urban graph-based route (red) is already similar to expected route (green).

Route quality – Example (rural)

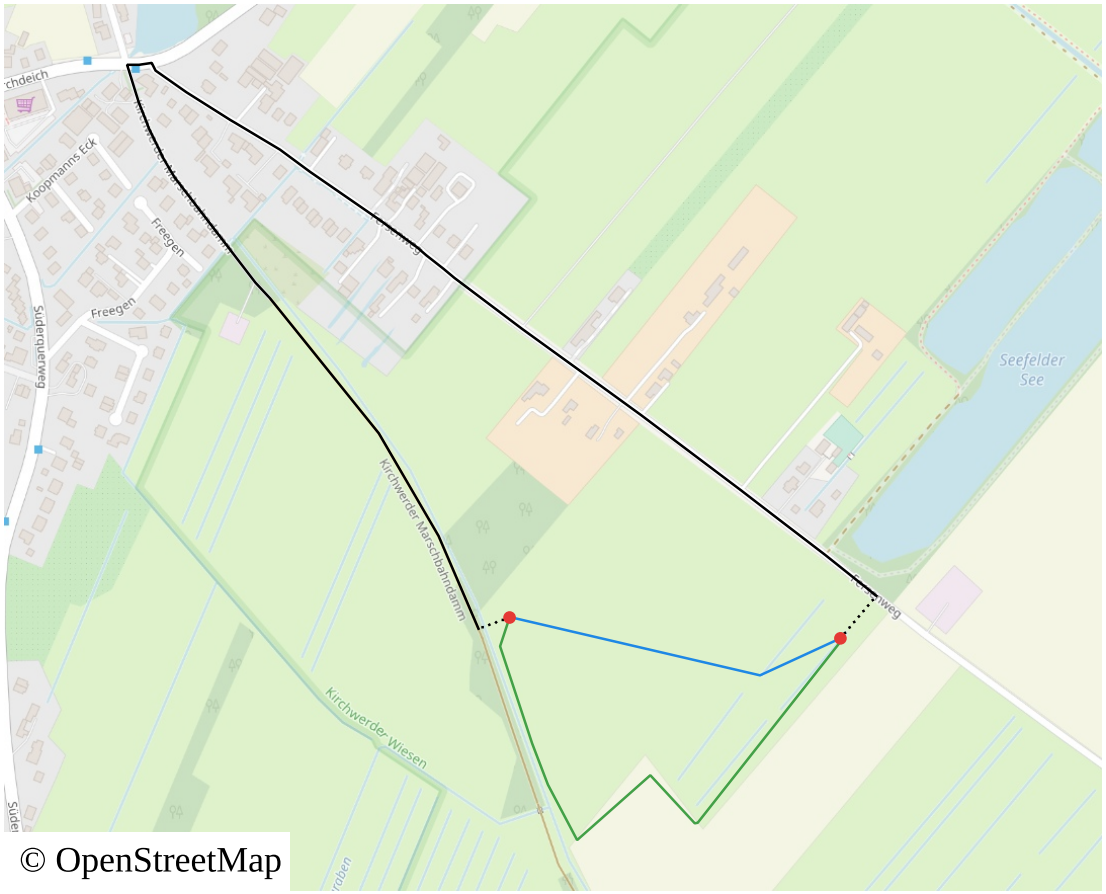


Figure 19: Expected (green), graph-based (black) and actual (blue) routes.

Hausdorff distances of routes

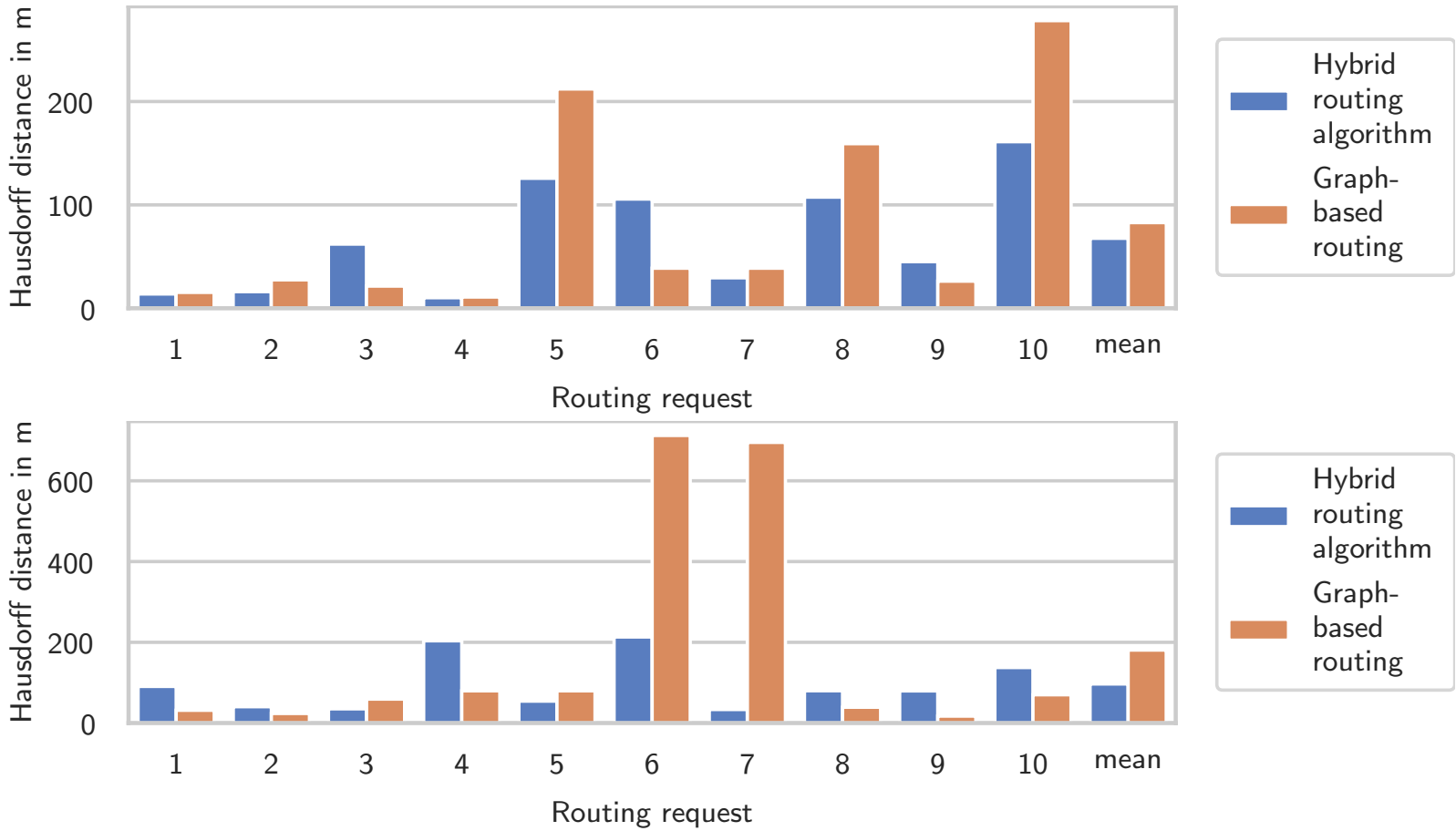


Figure 20: Hausdorff distances between calculated and expected routes using the "OSM city" (above) and "OSM rural" (below) datasets.

Future work

- routing and graph generation enhancements
- performance enhancements
- additional functionalities:
  - respect road attributes and size
  - infer attributes for visibility edges
  - support for third dimension
  - support for parallel routing queries
  - dynamic graph changes

Conclusion

- hybrid routing algorithm able to yield high quality paths
  - visibility edges enable paths through open spaces
  - road edges are walkable and contain useful information
  - arbitrary positions reachable
- urban graph-based routes already close to expected routes
- data quality important for route quality

Alternative approaches

Alternatives to visibility graph generation:

- routing with continuous Dijkstra
- merging multiple routes in postprocessing step
- other graph-generation methods
  - Voronoi diagrams, skeletonization
  - ad-hoc generation (e.g. by using continuous Dijkstra)

Answering routing queries

1. add vertices for source & destination to graph
2. create and add their visibility edges
   → same algorithm used for graph generation
3. use A* to determine shortest path
4. remove previously created vertices and edges

Performance optimizations – kNN search

Only consider $k = k_b \cdot k_n$ many visibility neighbors:

- k_b bins, each covering $360/k_b$ degree
- k_n neighbors per bin
- bins ensure that neighbors exist in all directions

Performance optimizations – Valid angle areas

Only consider potential visibility neighbors at certain angles:

- consider shortest path $p \rightarrow$ no edges can be relaxed
- only vertices at certain angles are valid neighbors $\rightarrow$ valid angle area

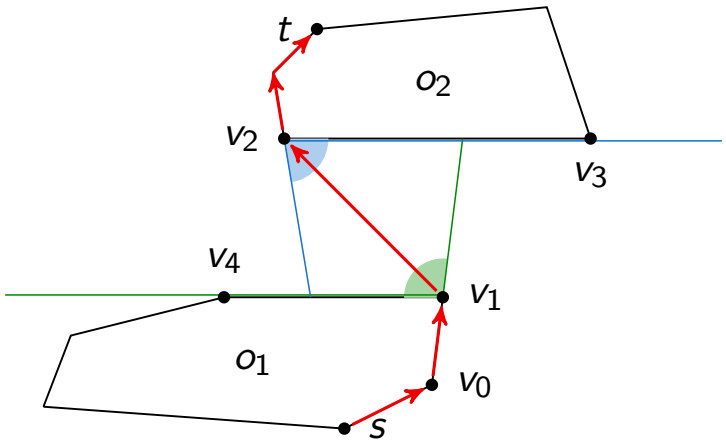


Figure 21: Valid angle areas (blue and green).

Performance optimizations – Convex hull filtering

Only consider vertices on a convex hull:

- convex hull is shortest path around polygon

→ vertex not on any convex hull will never be part of any shortest path

Performance optimizations – BinIndex data structure

Interval data structure for shadow areas:

- bin-based: each bin covers certain angle interval
- constant-time bin access

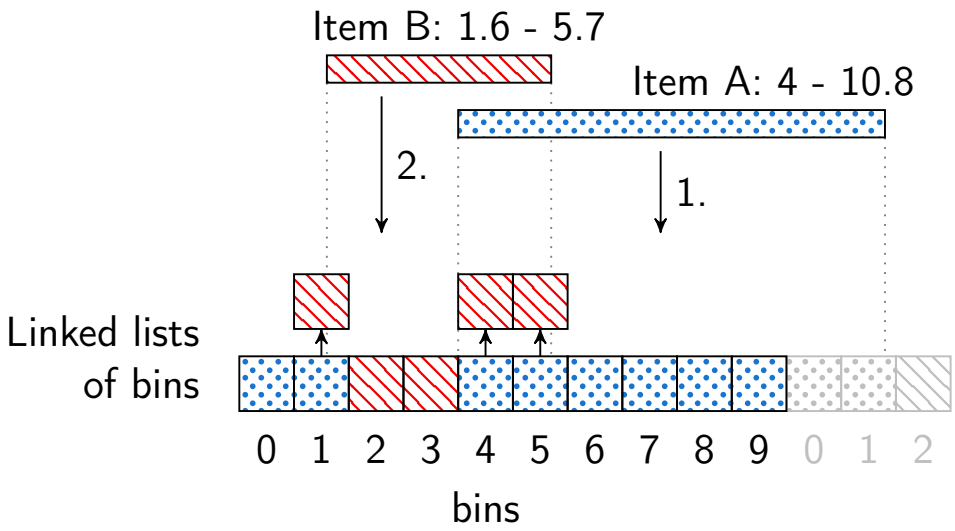


Figure 22: BinIndex data structure storing two overlapping intervals.

Value of graph data: Road edge preference

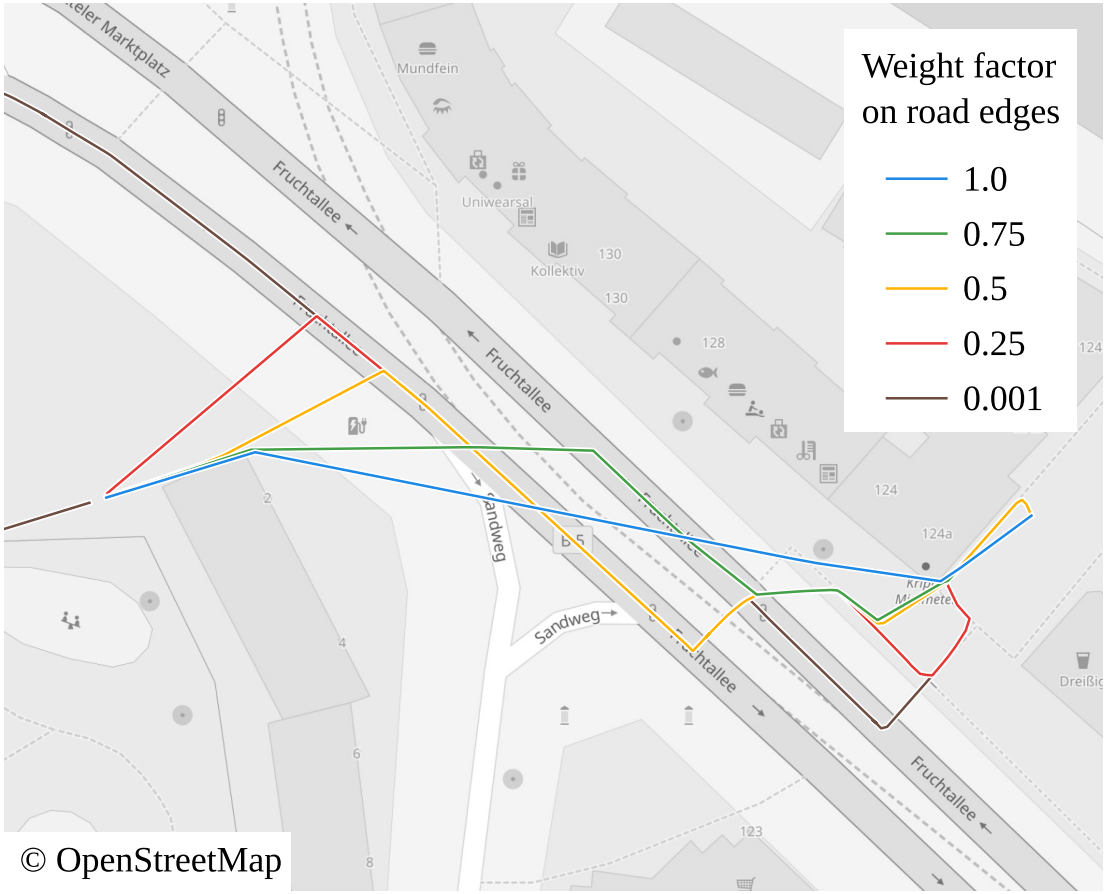


Figure 23: Different preferences on edges. High factor prefers visibility edges, low factor prefers road edges.

Graph generation task performance

Operation	Normal	No roads	No obstacles
kNN search	5,957.97 ms	5,924.27 ms	0.01 ms
Create graph	191.68 ms	210.40 ms	0.01 ms
Get obstacles	42.53 ms	40.76 ms	0.78 ms
Merge road edges	6,599.78 ms	5.67 ms	32.79 ms
Add POI attributes	4.83 ms	1.53 ms	0.42 ms
Total time	12,853.51 ms	6,201.96 ms	37.19 ms

Table 2: Time of graph generation tasks on different variants of the 4 km^2 "OSM city" dataset.

Route quality analysis – Examples (OSM city)

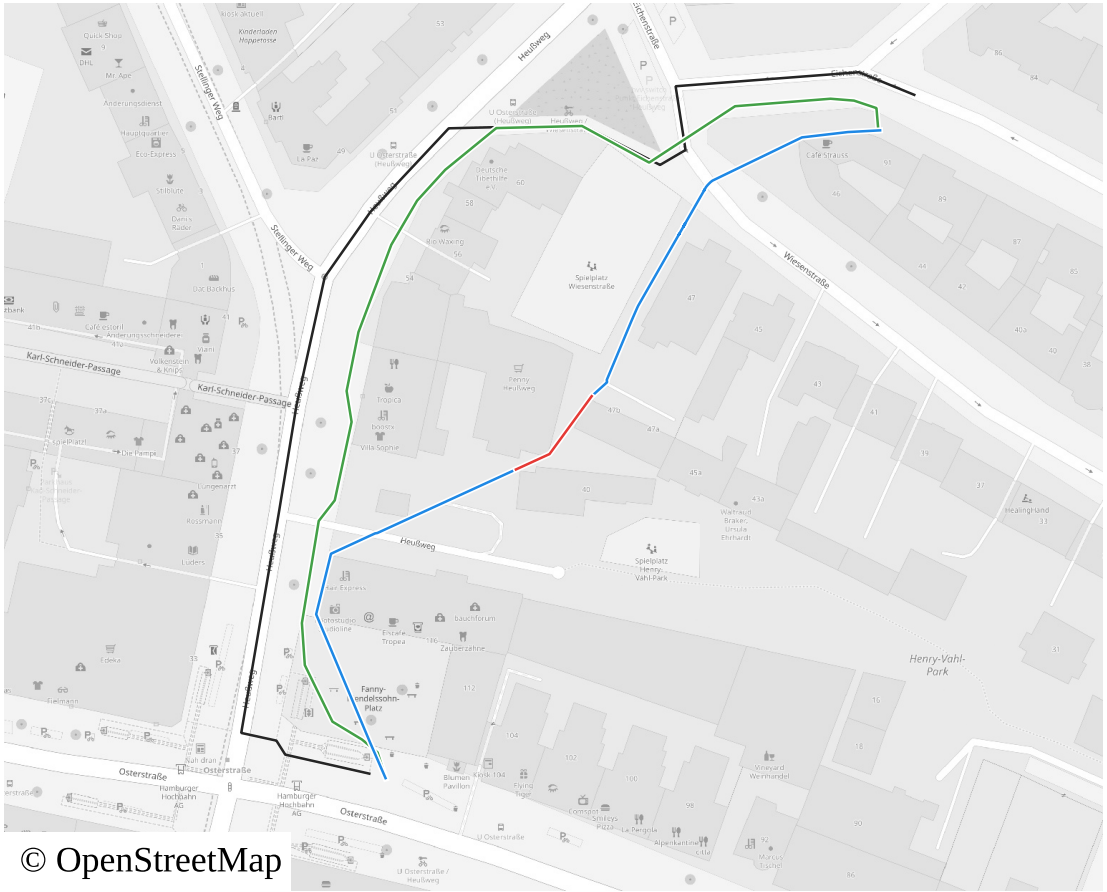


Figure 24: Expected (green), graph-based (black) and actual (blue) routes. Faulty passages marked in red.

Route quality analysis – Examples (OSM rural)

Figure 25: Comparison of routes to aerial imagery.

Route quality analysis – Examples (OSM rural)

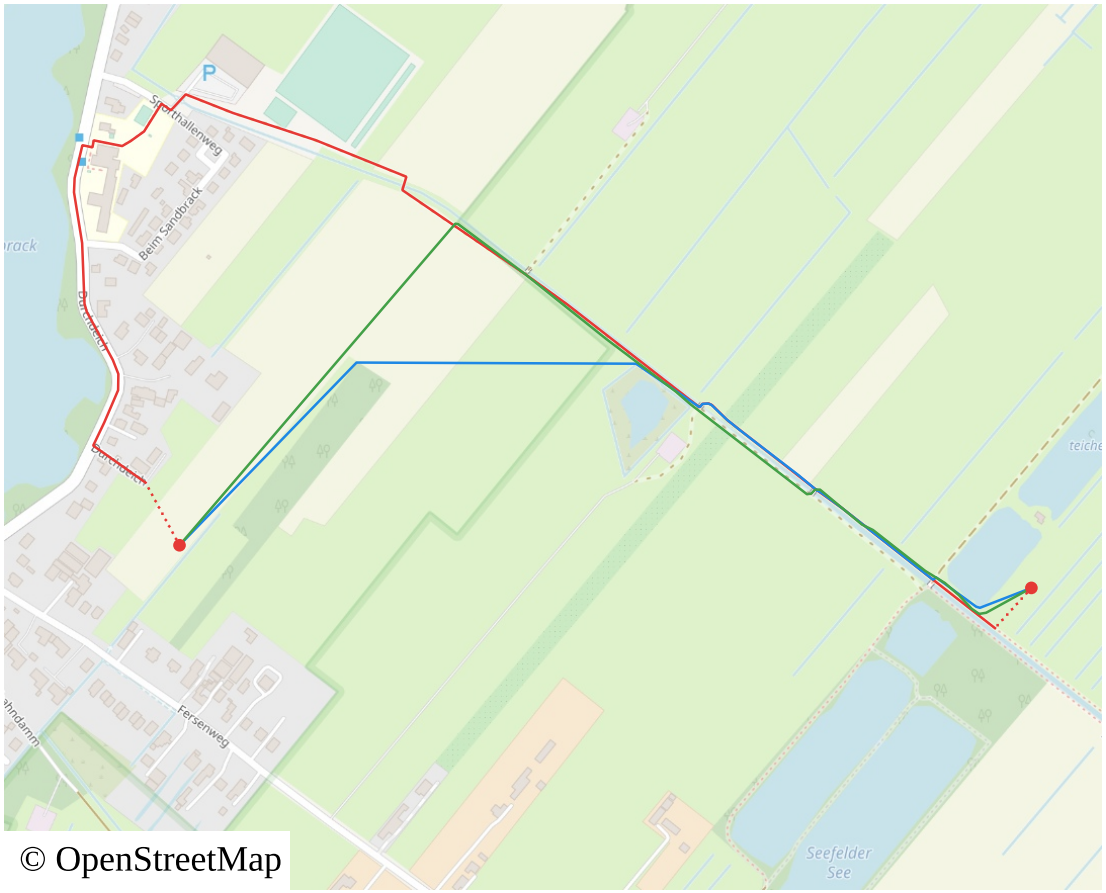


Figure 26: Expected (green), graph-based (red) and actual (blue) routes.

Routing performance

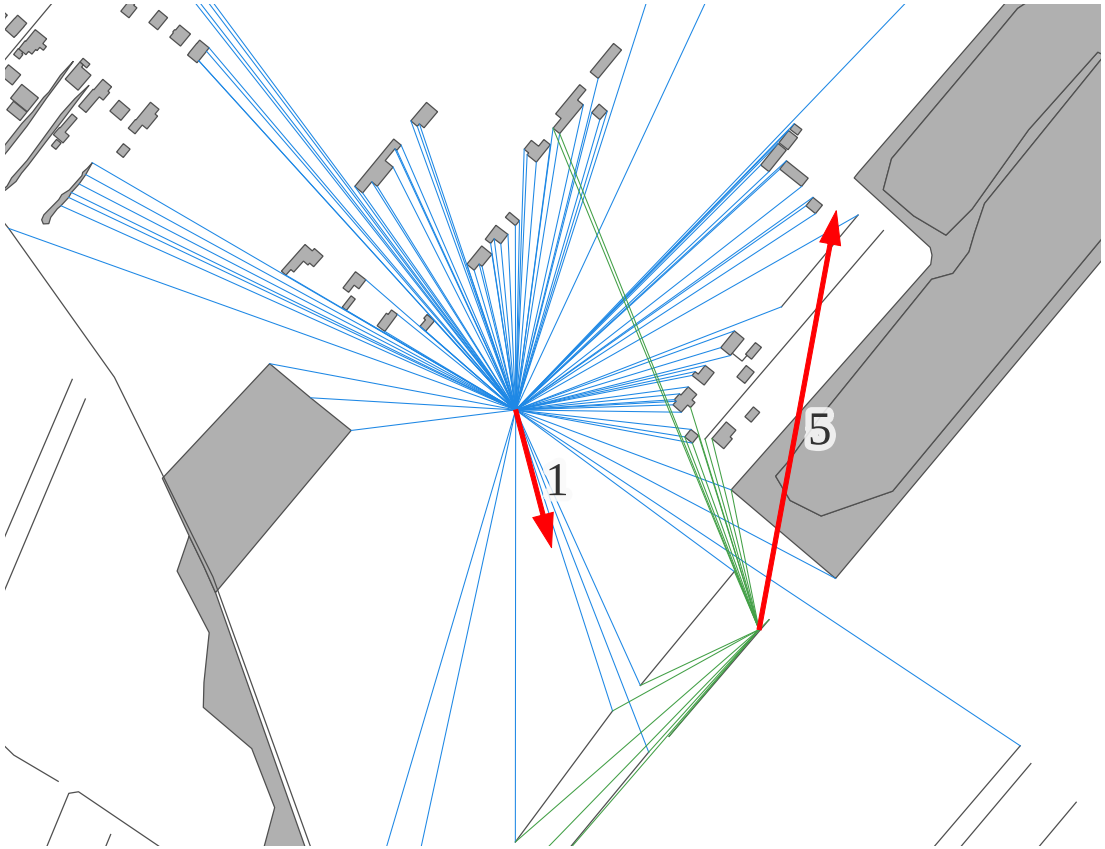


Figure 27: Edges creates for two routing requests in the "OSM rural" dataset.

Graph generation task performance

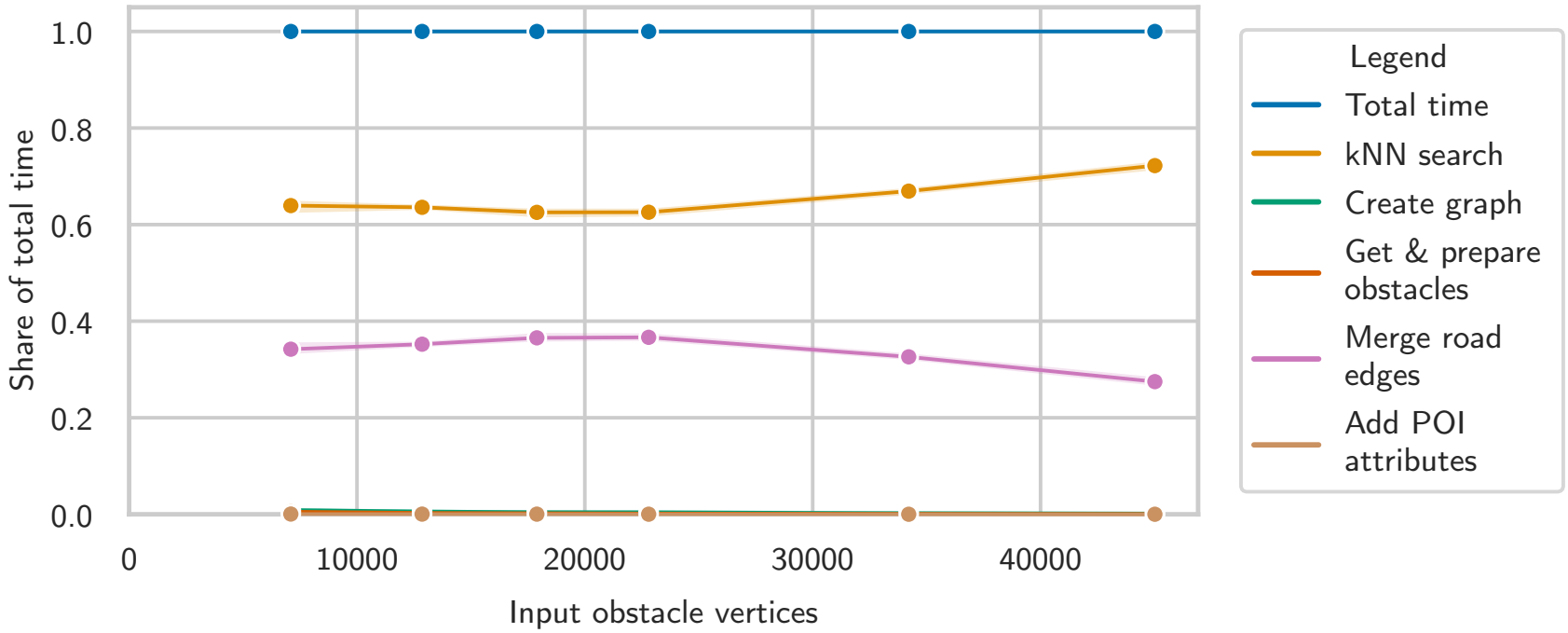


Figure 28: Time of graph generation tasks on all "OSM city" datasets.

Graph generation task performance

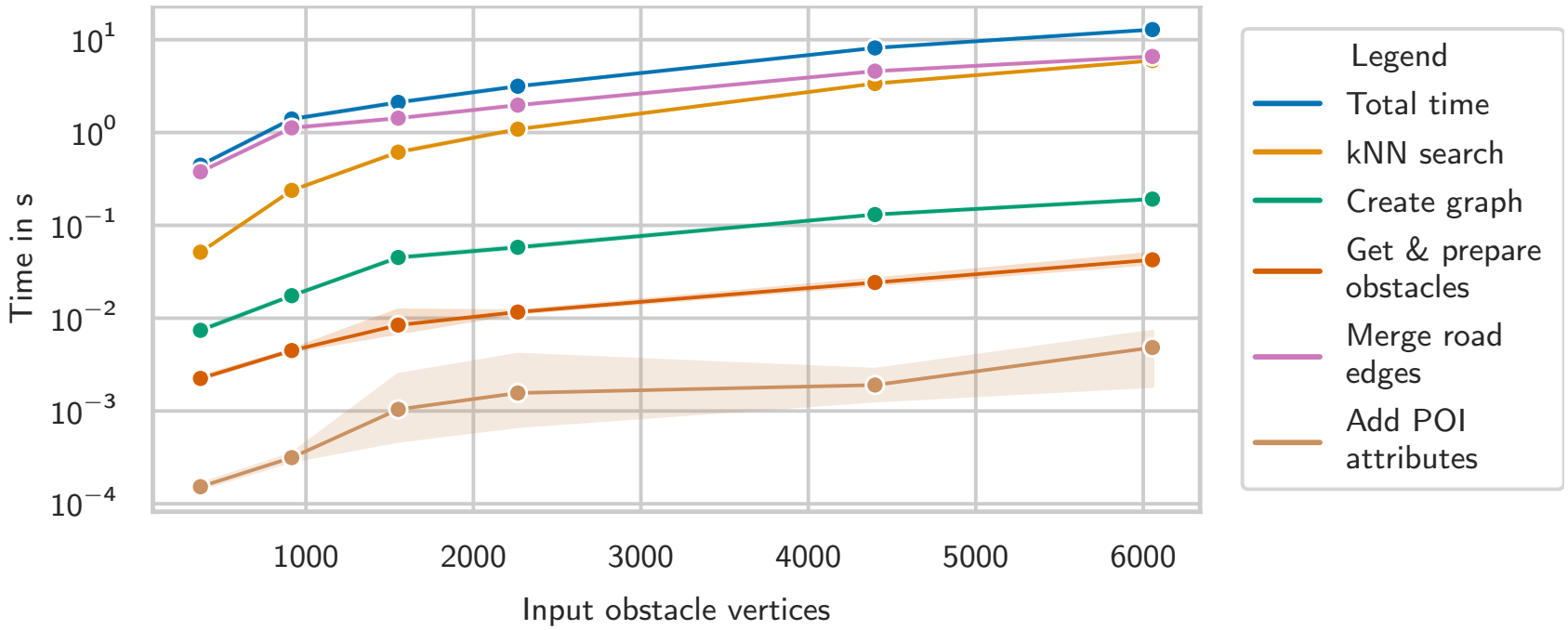


Figure 29: Time of graph generation tasks on all "OSM rural" datasets.

Graph generation task performance

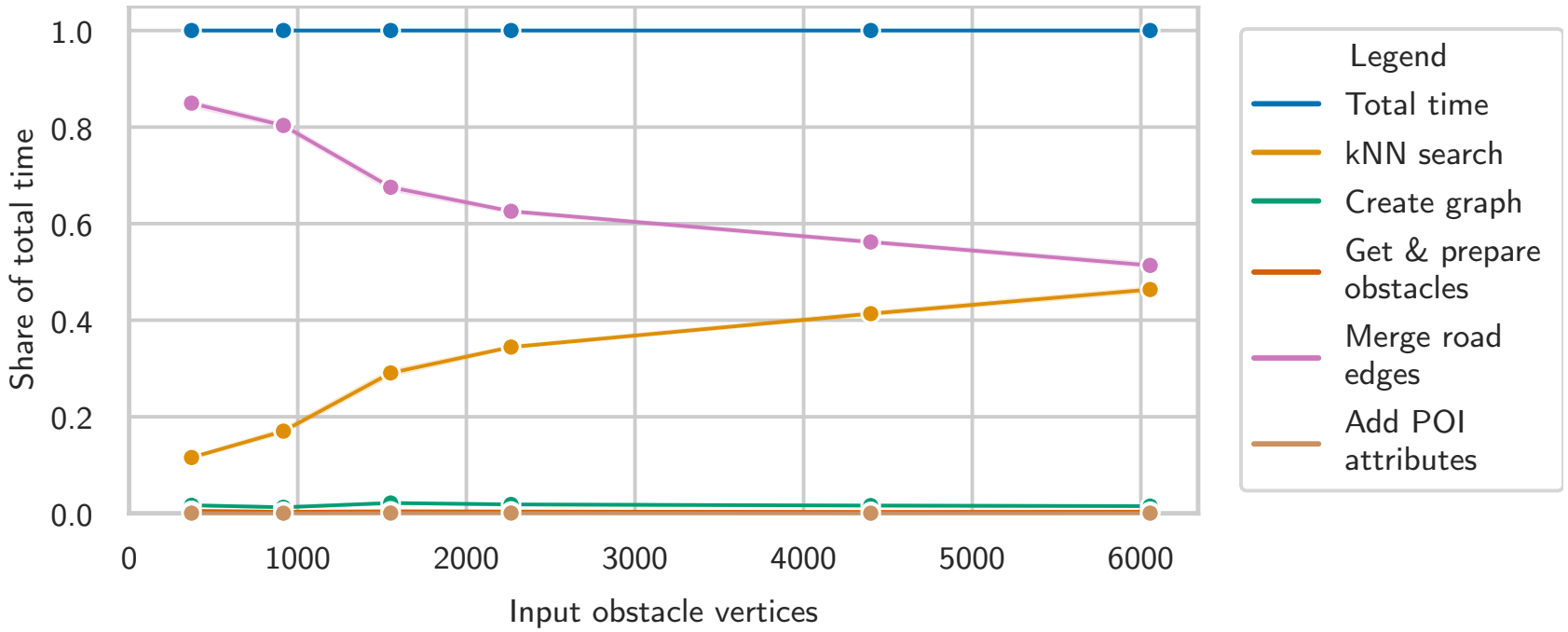


Figure 30: Time of graph generation tasks on all "OSM rural" datasets.