



## OS-Matcher

**Fabian Sandoval Saldias, Julien Gout, Markus Abel**

**May 03, 2021**



# USAGE

<b>1 Installation</b>	<b>3</b>
1.1 Requirements . . . . .	3
1.2 Getting the code . . . . .	3
1.3 Building third party code . . . . .	4
1.4 Provision Conan packages . . . . .	4
1.5 Building <i>OS-Matcher</i> . . . . .	4
<b>2 Execute example matcher</b>	<b>5</b>
2.1 Preparation . . . . .	5
2.2 Execution . . . . .	5
<b>3 Data selection</b>	<b>7</b>
3.1 Data structures . . . . .	7
3.2 Input data . . . . .	9
<b>4 Overview</b>	<b>13</b>
4.1 Candidate search . . . . .	13
<b>5 Overview</b>	<b>17</b>
5.1 Routing . . . . .	17
5.2 Matching . . . . .	22
<b>6 Algorithms</b>	<b>25</b>
6.1 Dijkstra router . . . . .	25
6.2 Directed candidate router . . . . .	27
6.3 Sampling point router . . . . .	29
6.4 Backtrack router . . . . .	32
6.5 Skip router . . . . .	39
6.6 Piecewise router . . . . .	42
<b>7 Filters</b>	<b>43</b>
7.1 CsvTrackReader . . . . .	44
7.2 JsonTrackReader . . . . .	45
7.3 GeoJsonMapReader . . . . .	46
7.4 OsmMapReader . . . . .	47
7.5 SamplingPointFinder . . . . .	48
7.6 GraphBuilder . . . . .	49
7.7 Router . . . . .	50
7.8 CsvRouteWriter . . . . .	52
7.9 CsvSubRouteWriter . . . . .	53
7.10 GeoJsonRouteWriter . . . . .	56

7.11	CsvTrackWriter . . . . .	57
7.12	GeoJsonTrackWriter . . . . .	58
7.13	GeoJsonMapWriter . . . . .	59
7.14	JsonRouteStatisticWriter . . . . .	60
<b>8</b>	<b>API documentation</b>	<b>65</b>
8.1	Class Hierarchy . . . . .	65
8.2	File Hierarchy . . . . .	65
8.3	Full API . . . . .	65
<b>9</b>	<b>Glossary</b>	<b>161</b>
9.1	General . . . . .	161
9.2	Operational data details . . . . .	162
<b>10</b>	<b>Specifications</b>	<b>163</b>
10.1	General . . . . .	163
<b>Index</b>		<b>165</b>

---

**Note:** *OS-Matcher* is an open-source C++ framework for routing applications.

---



## INSTALLATION

The *OS-Matcher* uses some third party repositories in Git submodules as well as Conan repositories. Some Conan packages aren't in any public registry yet and have to be made available in your company's infrastructure or through a local registry. We will use the latter approach in this guide.

### 1.1 Requirements

**Warning:** For the following installation guide we assume you have a running Debian based system (f.ex. Ubuntu) and the following tools installed and configured:

- Git
- GCC 5+
- Python 3

We need additional build tools and the PostgreSQL library.

```
sudo apt install ninja-build cmake iwyu libpq-dev
python3 -m pip install conan --upgrade
```

Furthermore we need to set up a Conan profile used to get packages. We re-create the default profile now, configured for GCC and the new C++11 ABI.

```
# Create default profile configured for GCC and new C++11 ABI
rm -f ~/.conan/profiles/default
conan profile new default --detect
conan profile update settings.compiler.libcxx=libstdc++11 default
```

### 1.2 Getting the code

Change into some workspace directory and then clone the repository.

```
git clone https://github.com/Ambrosys/os-matcher.git
```

## 1.3 Building third party code

We need to build third party libraries which are not available as Conan packages.

```
git submodule update --init  
NJOBS=$(nproc) make -C ThirdParty
```

## 1.4 Provision Conan packages

We run a local Conan registry in a seperate terminal.

```
cd conan  
./run-server.sh
```

Configure Conan to use this registry when refering to ambrosys.

```
remote_url=http://localhost:9300/  
username=demo  
password=demo  
  
conan remote add ambrosys $remote_url  
conan user --remote ambrosys $username -p $password
```

Now we can download, build and publish packages into our local registry.

```
mkdir conan/repositories  
for package in amb-graph amb-log amb-thread amb-pipeline cli-app; do  
    git clone https://github.com/Ambrosys/$package.git conan/repositories/$package  
    conan create conan/repositories/$package ambrosys/stable --build missing  
    conan upload "$package/*" --confirm --all --remote ambrosys  
done
```

## 1.5 Building *OS-Matcher*

Now that all dependencies are at hand, we can finally build the *OS-Matcher* code.

```
mkdir build  
cmake -Bbuild -H. -GNinja \  
      -DCMAKE_BUILD_TYPE=Release \  
      -DCMAKE_INSTALL_PREFIX=install  
ninja -C build test-if-git-HEAD-has-changed  
ninja -C build install
```

You will find the binaries now in build/install.

## EXECUTE EXAMPLE MATCHER

### 2.1 Preparation

To run the example application of the *OS-Matcher*, you need some data.

#### 2.1.1 Database

You need a running OpenStreetMap database. We reference the server and the port as \$DB\_SERVER and \$DB\_PORT in the following.

#### 2.1.2 Track

You need a CSV file containing a track. The track file needs to have the following format:

```
2020-12-31T14:00:00.000;52.0000000000000;13.0000000000000;180.00;24.00
```

The fields are:

- date/time
- latitude
- longitude
- heading
- velocity

We reference the file as `in/track.csv` in the following.

### 2.2 Execution

```
export LD_LIBRARY_PATH=build/lib
build/bin/ExampleMatcher \
  --fsp-in in/track.csv \
  --map-out out/map.geojson \
  --route out/route.csv \
  --sub-route out/sub-route.csv \
  --route-geojson out/route.geojson \
  --route-statistic out/route-statistic.json \
  --pipeline out/pipeline.dot \
```

(continues on next page)

(continued from previous page)

```
--host $DB_SERVER \
--port $DB_PORT \
--log-level noise
```

## DATA SELECTION

The *OS-Matcher* supports various input and output formats and can easily be extended to support your own formats.

The core algorithms work on well-defined data structures which are processed by various filters. The source data is either completely provided by the input filters (coordinate/track and street map readers) or it can partially be reconstructed by intermediate filters. The core filters are performing the sampling point candidate search, street graph construction and the actual matching and routing of the track on the street map. The resulting data gets passed to registered output filters to save, for example, the calculated route.

Basic data flow (simplified filter pipeline):

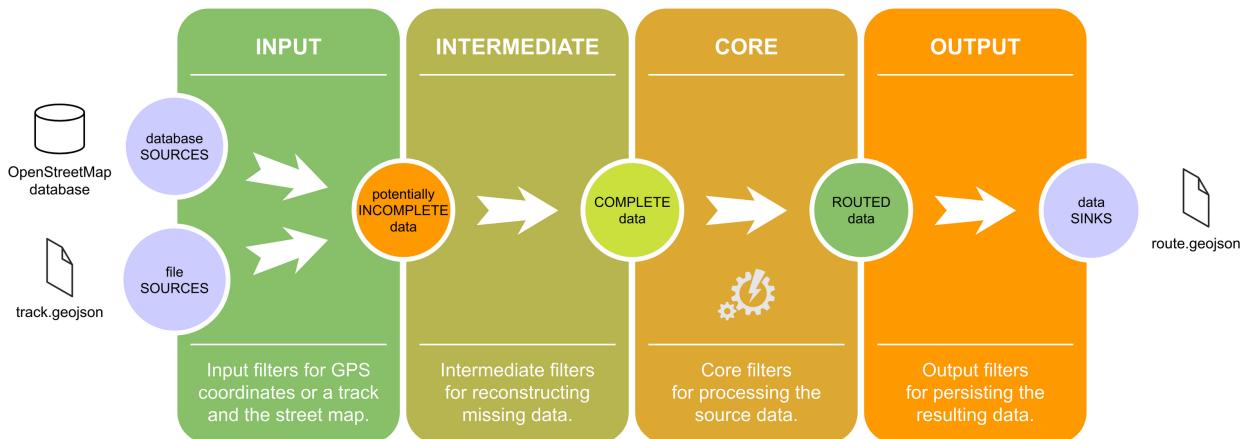


Fig. 3.1: Filter dataflow

### 3.1 Data structures

The core processing component of the *OS-Matcher* is, of course, the router. To perform the routing it needs data which it gets from input filters and intermediate filters. The input filters may also depend on each other. An example: to be able to load the map from a database, the track is needed to define the boundaries.

### 3.1.1 Processed data

Each processing component is designed as a filter with a well defined set of input data (requirements and optionals) as well as output data (fulfillments).

**See also:**

See filter to learn how to implement and plug in a filter.

#### Example: Router

The router mandatorily needs data about the street topology and the *track* point projections onto street segments. For each track point, there might exist more than one *candidate* (see *Candidate search* or *Sampling point router*).

It may optionally process the track point's timestamp and velocity, if present. In contrast, it does not need the track point's coordinates.

The output is the route along with additional data to help reason about the found route.

### 3.1.2 Type listing

It follows an overview about all the data types that are passed along the filters.

**For each track point:**

- coordinate pair, as *PointList*
- heading, as *HeadingList*
- timestamp, as *TimeList*
- velocity, as *VelocityList*

**For each street segment:**

- coordinates, as *SegmentList*
- junctions, as *NodePairList*
- travel direction, as *TravelDirectionList*

**Street graph:**

- street graph, as *Graph*
- mapping from graph edge to street segment, as *GraphEdgeMap*
- mapping from graph node to street junction, as *NodeMap*
- mapping from street segment to graph edge, as *StreetIndexMap*

**Routing:**

- resulting route data, as *RouteList*
- additional routing data per sampling point route data, as *RoutingStatistic*
- projected track points, as *SamplingPointList*

## 3.2 Input data

To be able to perform the routing, we need a minimum set of data. When the input filters are not able to provide all needed data intermediate filters could be used to reconstruct the missing entries if possible. Additional data can be provided to increase the accuracy.

### 3.2.1 Data overview

#### Track

To perform the routing or matching, the filters have to provide at least two coordinate pairs (latitude and longitude; *PointList*).

If possible, the following data should be provided for each point, too:

- heading (does not have to be complete, points without a heading are allowed) (*HeadingList*)
- timestamp (*TimeList*)
- velocity (*VelocityList*)

#### Street Map

To built the routable street graph, the filters have to provide the following data **\*for each street item\***:

- coordinates (at least two coordinate pairs) (*SegmentList*)
- identifiers for the two endpoints (junctions) (*NodePairList*)
- allowed travel direction (forwards, backwards, both) (*TravelDirectionList*)

### 3.2.2 Input filter

The data must be provided by one or more filters.

#### See also:

See filter to learn how to implement and plug in a filter.

#### Track

In Fig. 3.2 Track data you see an example track and the representing data lists which have to be filled by the filters.

For a track to be complete, the following requirements have to be fulfilled by one or more filters:

#### PointList

- Data type: *PointList*

#### HeadingList or PartialHeadingList (both optional)

- Use the latter if the listing may contain *Nan* items. Using *PartialHeadingList*, a filter that can reconstruct the missing heading data (fulfilling *HeadingList*) could easily be plugged in.
- Data type: *HeadingList*

---

**Note:** For *Nan* items, use:

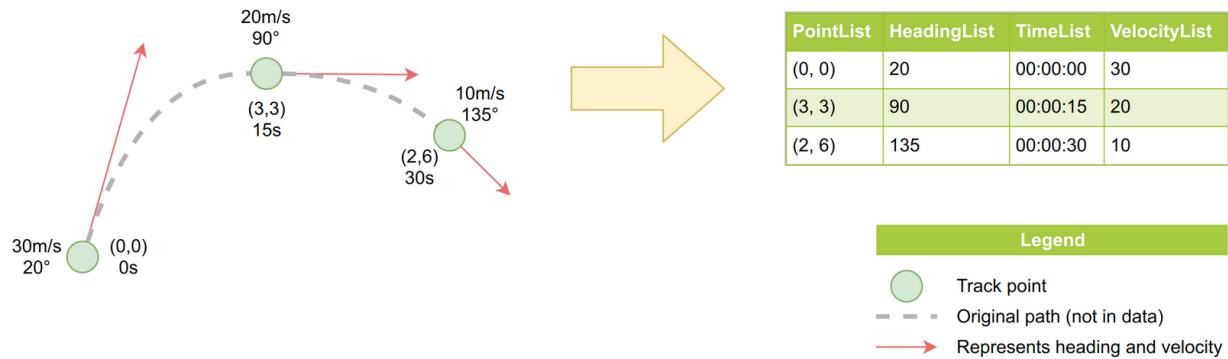


Fig. 3.2: Track data

```
std::numeric_limits<AppComponents::Common::Types::Track::Heading>::quiet_NaN()
```

### TimeList (optional)

- Data type: *TimeList*

### VelocityList (optional)

- Data type: *VelocityList*

Example skeleton:

```

1 class MyTrackReader : public AppComponents::Common::Filter::Filter
2 {
3     public:
4         MyTrackReader( std::istream & input );
5         bool operator()( 
6             AppComponents::Common::Types::Track::TimeList &,
7             AppComponents::Common::Types::Track::PointList &,
8             AppComponents::Common::Types::Track::HeadingList &,
9             AppComponents::Common::Types::Track::VelocityList & );
10
11     private:
12         std::istream & input_;
13     };
14
15
16     MyTrackReader::MyTrackReader( std::istream & input )
17     : Filter( "MyTrackReader" ), input_( input )
18     {
19         setRequirements( {} );
20         setOptionals( {} );
21         setFulfillments( { "TimeList", "PointList", "HeadingList", "VelocityList" } );
22     }
23
24     bool MyTrackReader::operator()( 
25         Common::Types::Track::TimeList & timeList,
26         Common::Types::Track::PointList & pointList,
27         Common::Types::Track::HeadingList & headingList,
28         Common::Types::Track::VelocityList & velocityList )
29     {

```

(continues on next page)

(continued from previous page)

```

30     APP_LOG_TAG( noise, "I/O" ) << "Reading track";
31
32     // read `input_` filling `timeList`, `pointList`, `headingList` and `velocityList`
33
34     return true;
35 }
```

## Street Map

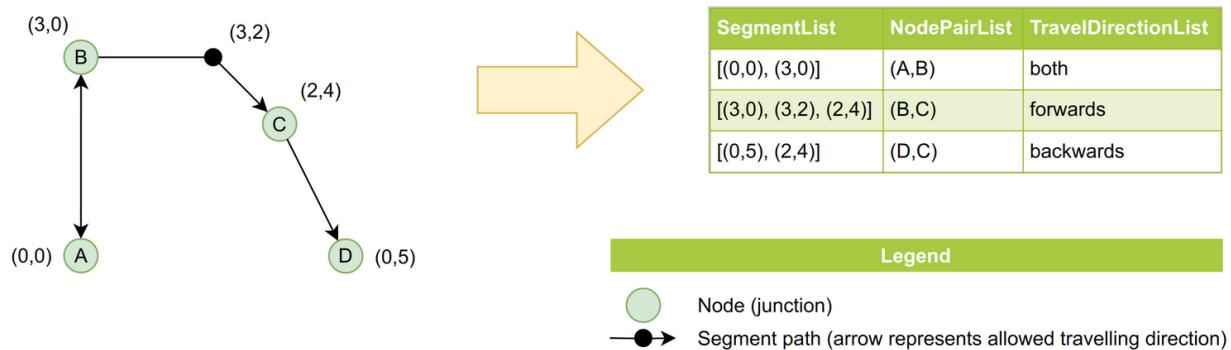


Fig. 3.3: Street map data

In Fig. 3.3 Street map data you see an example street map and the representing data lists which have to be filled by the filters.

For a street map to be complete, the following requirements have to be fulfilled by one or more filters:

### SegmentList

- Data type: *SegmentList*

### NodePairList

- Data type: *NodePairList*

### TravelDirectionList

- Data type: *TravelDirectionList*

Example skeleton:

```

1 class MyStreetMapReader : public AppComponents::Common::Filter::Filter
2 {
3 public:
4     MyStreetMapReader( std::istream & input );
5     bool operator()( 
6         Types::Street::SegmentList &,
7         Types::Street::NodePairList &,
8         Types::Street::TravelDirectionList & );
9
10 private:
11     std::istream & input_;
12 }
```

(continues on next page)

(continued from previous page)

```
14 MyStreetMapReader::MyStreetMapReader( std::istream & input )
15 : Filter( "MyStreetMapReader" ), input_( input )
16 {
17     setRequirements( {} );
18     setOptionals( {} );
19     setFulfillments( { "SegmentList", "NodePairList", "TravelDirectionList" } );
20 }
21
22
23 bool MyStreetMapReader::operator()( 
24     Types::Street::SegmentList & segmentList,
25     Types::Street::NodePairList & nodePairList,
26     Types::Street::TravelDirectionList & travelDirectionList )
27 {
28     APP_LOG_TAG( noise, "I/O" ) << "Reading street map";
29
30     // read `input_` filling `segmentList`, `nodePairList` and `travelDirectionList`
31
32     return true;
33 }
```

## OVERVIEW

Let  $M$  be the street map and  $A$  and  $B$  two track points.

The street map  $M$  in use is the [OSM map](#). Start and endpoint  $A$  and  $B$  are GPS locations, which not necessarily lay on the street map.

### 4.1 Candidate search

Since the start and end points are potentially unbound GPS locations, the algorithm starts with a candidate search (using `class SamplingPointFinder` with a definable search radius) to find nearby street segments from  $M$  resulting in a set of street segments for both points,  $S_A$  and  $S_B$ .

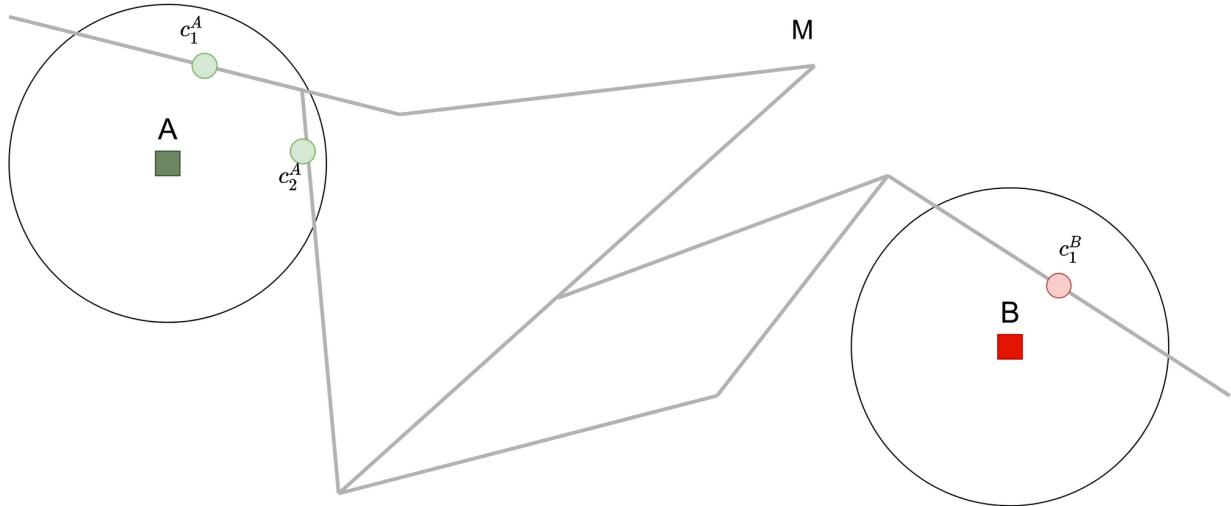


Fig. 4.1: Start and Endpoint with candidates

The point is than projected onto each segment of the set resulting in a set of candidate points  $C^A$  and  $C^B$  with the candidates  $c_i^A$  and  $c_j^B$ .

The candidates are ordered according to their distance. However this might not be unique as you see in the following image.

The track data may contain direction data as well, which is considered as the next decision criteria (using function `headingDifference()`). This would result in a distinct decision in our example.

If even that is not enough to determine a priority ordering of candidates, there are three more optional categories to deal with that issue:

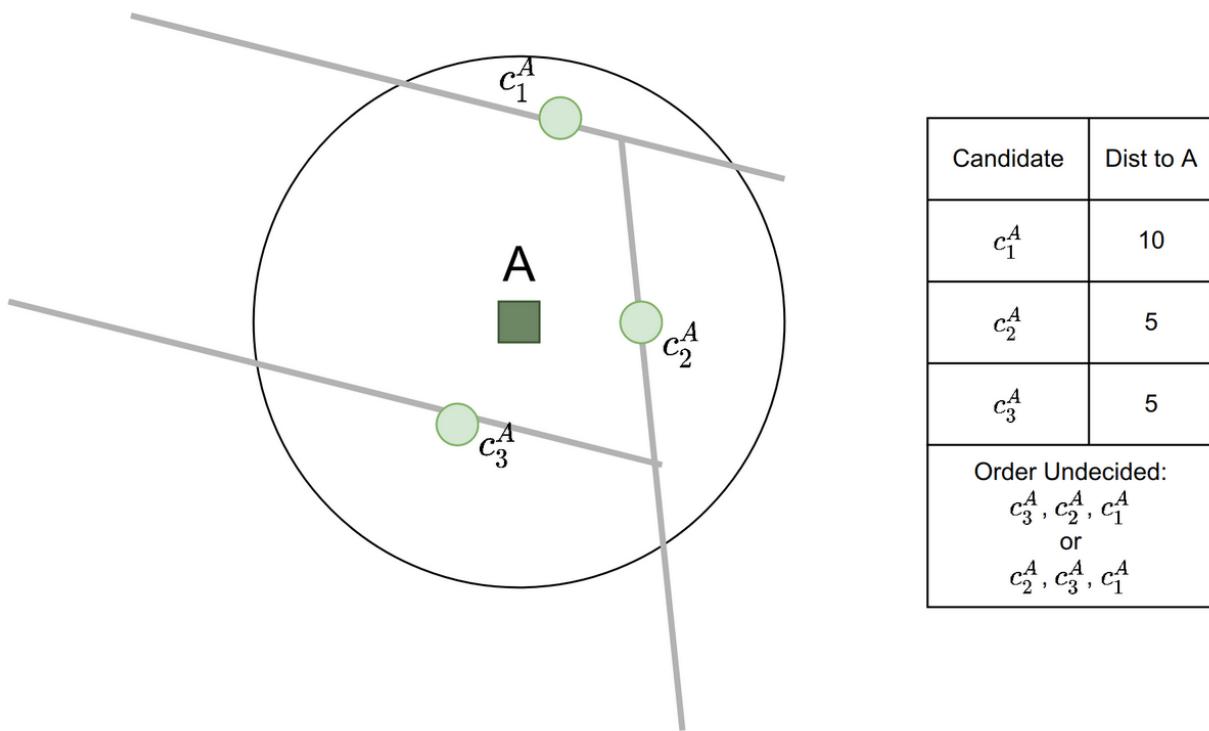


Fig. 4.2: Example for ambiguous candidates

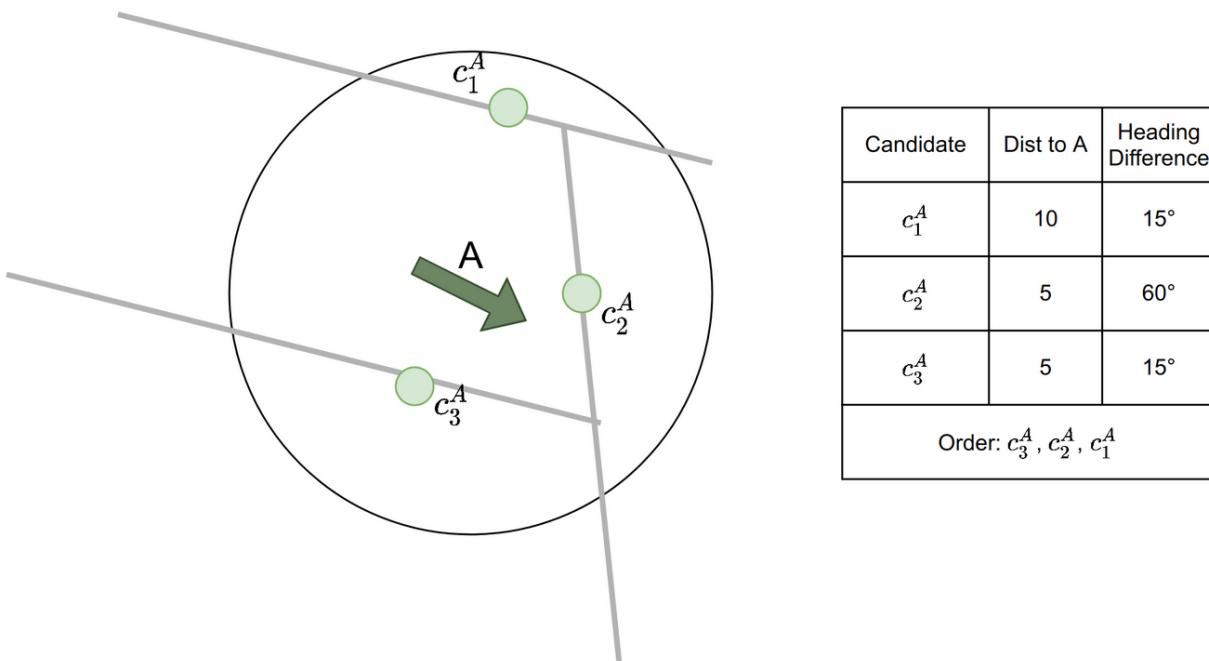


Fig. 4.3: Candidate rank with distance and heading

- **originId** of the street segment from the input street map data the candidate is laying on (for example the ID a street segment has in the osm database)
- **originOffset + streetSegmentIndex** (because it might arise that a street segment is divided in sub street segments)
- **streetIndex** of the street segment as it was placed in the internal data structure (as a last decision criteria which is guaranteed to be unique, for the rare cases when the input street map data has duplicate IDs)



## OVERVIEW

Routing and Matching seem sometimes interchangeable, but they are not.

### 5.1 Routing

Routing is defined as a threefold-function  $R(A, B, M) = r$  where  $M$  is the street map and  $A$  and  $B$  are start and endpoint of the street map.  $R$  returns a *valid navigation route* from  $A$  to  $B$  on the street map  $M$ .

The street map  $M$  in use is the [OSM map](#). Start and endpoint  $A$  and  $B$  are GPS locations, which not necessarily lay on the street map.

The routing algorithm falls apart into three parts:

- *Routing*
- *Clustering*
- *Final Evaluation*

#### 5.1.1 Routing

After the candidates are found (see [Candidate search](#)), we have a list of pairs  $P = \{(s, t) | s \in C^A, t \in C^B\}$ .

The street map  $M$  is in its core a graph of nodes and edges ( $G(V, N)$ ), the candidates are points on the edges and therefore not part of the graph. To overcome that, the nodes of the edge a candidate is placed on, become the first selected nodes in the routing.

Notice, if the used street map supports information on travel directions, here already a filtering step is done by only considering one node of the edge aka the street segment.

For each combination of the one or two possible first nodes of  $A$  and  $B$  now a routing is done on the graph of the street map using a routing algorithm. The default algorithm in *OS-Matcher* is Dijkstra's algorithm with the geographical length of an edge as its cost function.

So we end up with a list of routes, one for each pair  $(s, t)$  and for each combination of possible nodes ( $n_i^s$  and  $n_j^t$ ) for the edges ( $e_s$  and  $e_t$ )  $s$  and  $t$  are placed on.

Since all the routes are starting from candidates of the same track point  $A$  and likewise are ending on candidates of the same track point  $B$ , often the resulting routes are very similar.

In [Fig. 5.2](#) we see two *basically similar* routes like that.

Contrary, in [Fig. 5.3](#) we see two *basically different* routes.

When very similar and very different routes are likewise found for the same start and end points, as we see in the following example, mathematically justifying how to choose the most realistic route is very difficult:

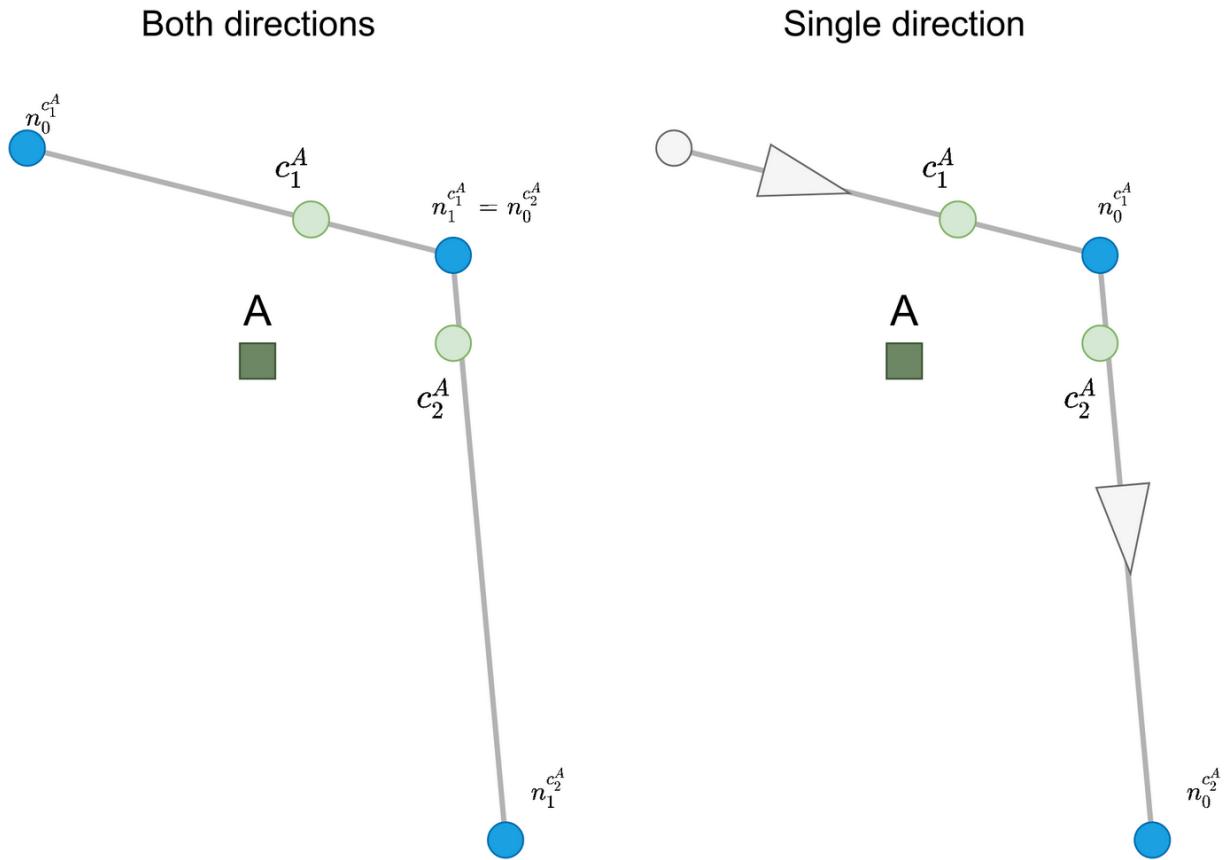


Fig. 5.1: Candidates on one way streets

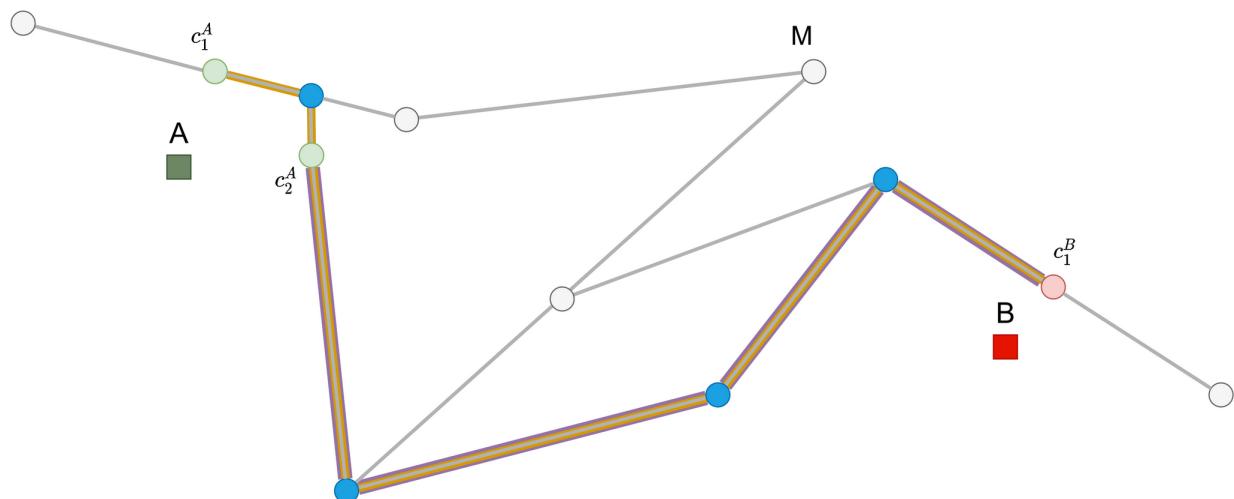


Fig. 5.2: Basically similar routes

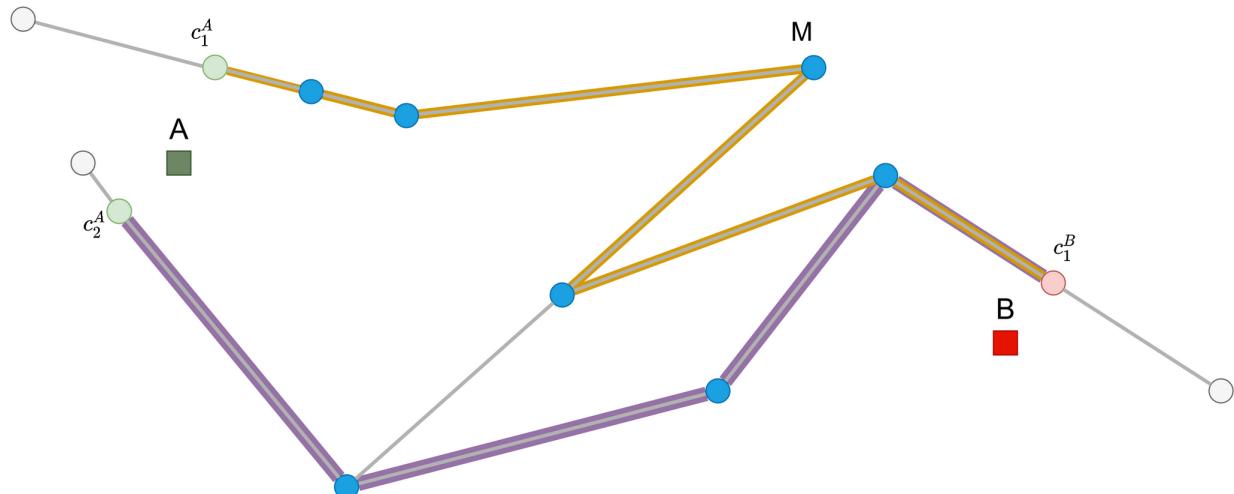


Fig. 5.3: Basically different routes.

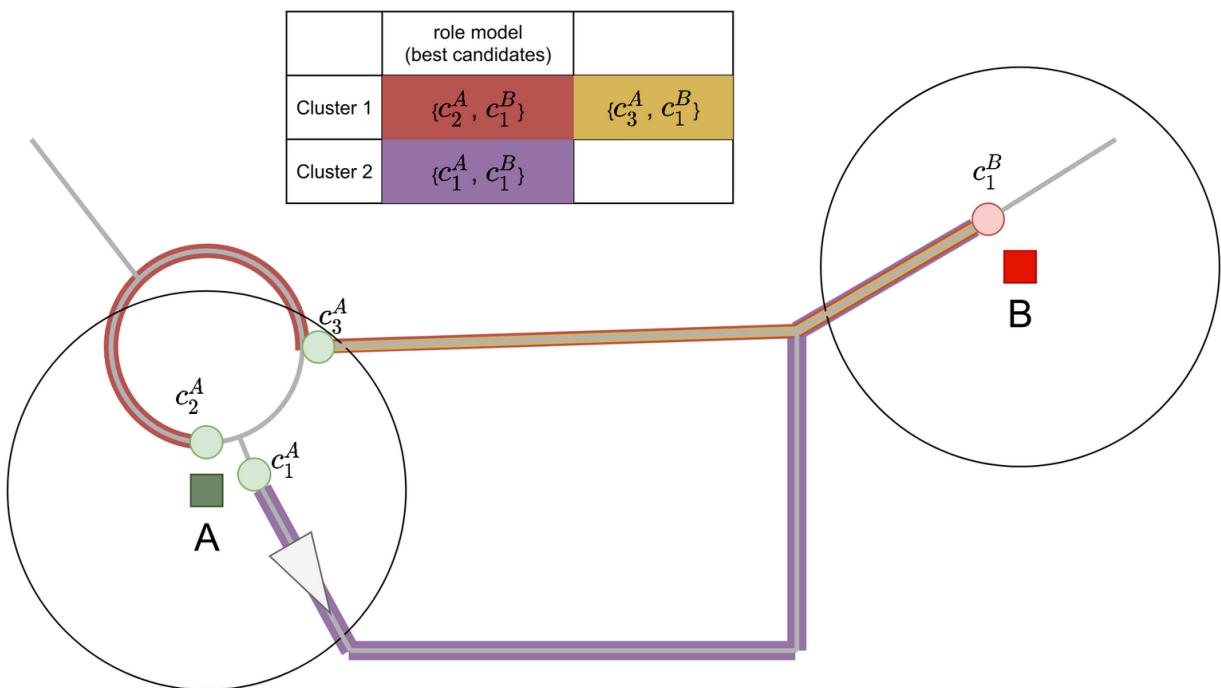


Fig. 5.4: Finding the most realistic route

In Fig. 5.4,  $C_1^A$  is the best candidate and  $C_3^A$  produces the shortest route, but both are not correct -  $C_2^A$  leads to the actual route.

To solve the problem, the routes are getting clustered, where each cluster represents a set of *basically similar* routes.

### 5.1.2 Clustering

As motivated above, a cluster is a set of routes which are representing *basically similar* routes. All routes in a cluster are ranked by a comparison class `BestSimilarRouteComparator`, which compares the candidate's rank according to the system described in [Candidate search](#).

Any new route which shall be placed into a cluster is compared to the highest ranked member of that cluster, the *role model* of that cluster, using a similarity function. The function `isSimilar()` compares two routes  $r_0$  and  $r_1$  against several criteria. Only when all of them are met, the route will be added to the cluster.

The criteria are:

- **max length difference**, the outermost two routes may differ in length (it is recommended to set this value to  $4 \cdot \text{candidate search radius}$ )
- **the source node of one is contained by the other**,  $n_{r_1}^s \in r_0$  or  $n_{r_0}^s \in r_1$
- **the target node of one is contained by the other**,  $n_{r_1}^t \in r_0$  or  $n_{r_0}^t \in r_1$
- **source and target node are not visited twice**

Note that the second and third criteria do not need to be fulfilled by both routes, only by one.

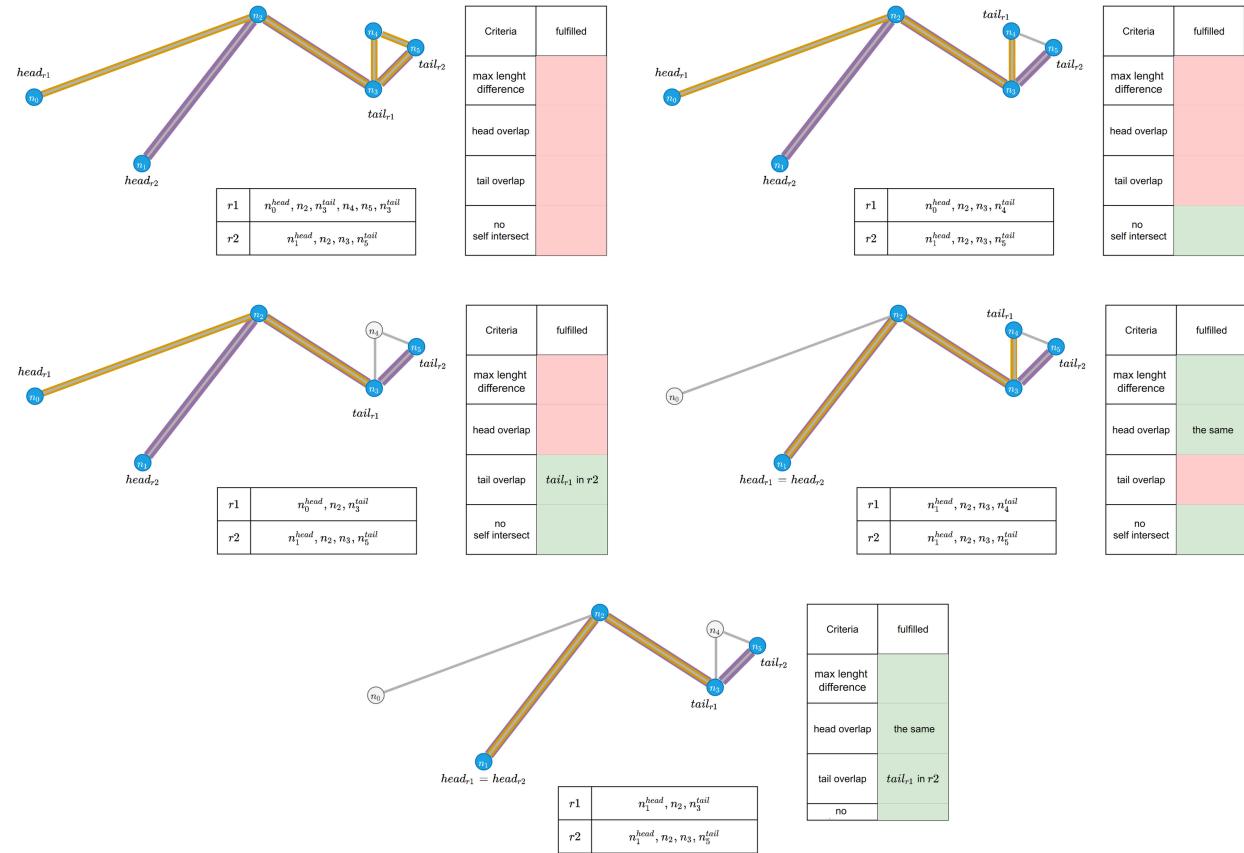


Fig. 5.5: Similarity criteria

Let's recap the example illustrated by Fig. 5.4. The track point  $A$  originates indeed from the roundabout. But due to the drift of  $A$  (may be induced due to data noise or inaccurate map data), the best candidate sits on a one-way street to the south ( $C_1^A$ ). The best (shortest) route however starts at the most unlikely candidate ( $C_3^A$ ), while the actual route starts at ( $C_2^A$ ).

Clustering is a way to overcome those and similar situations by filtering out unlikely routes which are just considered because of its candidate rank. Within a cluster, however, the candidate's rank assures that we get the most accurate starting point from all the *basically similar* routes.

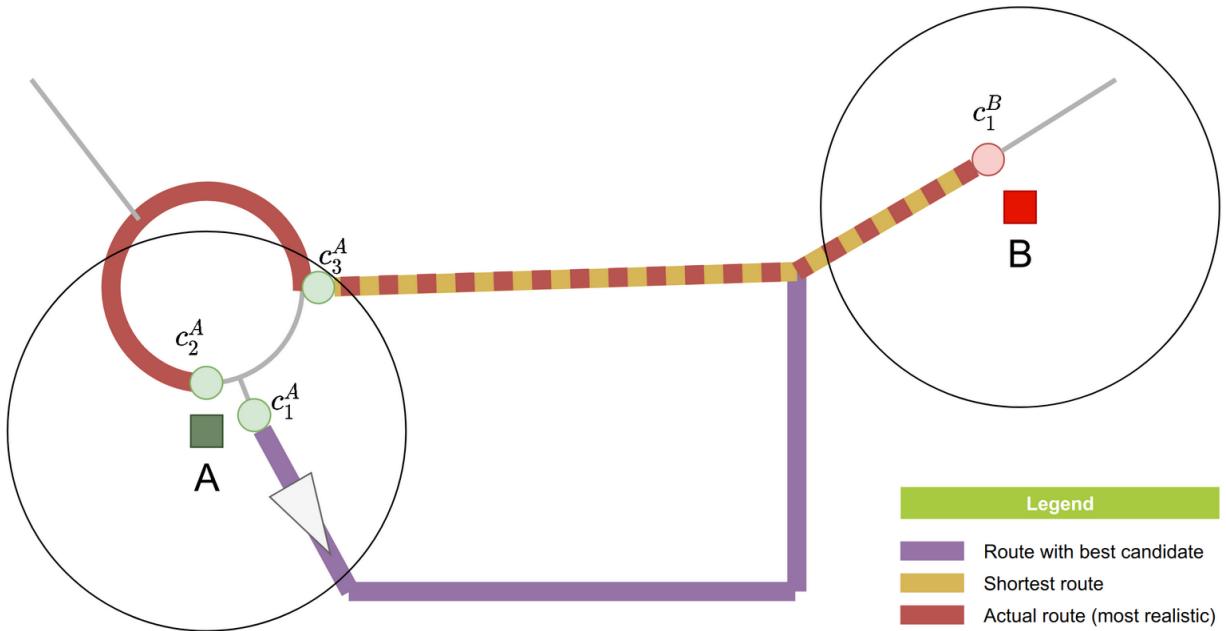


Fig. 5.6: Clustering

Now we have a set of clusters, each having a *role model*.

### 5.1.3 Final Evaluation

From each cluster the role model is chosen and all of those role models are compared using the class `BestRouteComparator`. The Comparator has three criteria:

- **length**, which selects the shortest route
- **cost**, which selects the route with the lowest routing costs
- **number of points**, which selects the route with the least number of nodes

The comparator comes in two flavors of criteria preference:

- **cheapest**, criteria order  $cost > length > number\ of\ points$
- **shortest**, criteria order  $length > cost > number\ of\ points$

The best route according to this comparison is then the result of our routing  $R(A, B, M) = r$ .

## 5.2 Matching

While *routing* is the solution to find a route in a graph for a given start and end point, matching can be understood as the generalization of that approach towards a list of track points. In *OS-Matcher* this list consists of geo points and the graph is a street map. In that way routing becomes applicable to the street matching problem which is common today especially in the tolling industry.

### Note: Street Matching Problem

Given a time bound track and a street map we want to identify the most likely route over the map that was taken by the track provider.

#### See also:

See *Data structures* for information of the track and map data.

The *OS-Matcher* is challenging this problem by solving stepwise rather small routing steps of the matching and combines them to a routing solution. And the algorithm is an extension of the steps described in *Routing*.

First the track is processed to become a List of sampling points. That means for each track point a candidate search is performed, identifying segments in the street map from which the data point might had originated. Only those track points with at least one candidate are considered later on since, without a candidate on the street map, the algorithm has no useful hints from the track point where to route to on the street map.

**Note:** Some implemented optimizations are dependent on additional meta data for the track, like heading and velocity. Not providing those can lead to poor matchings.

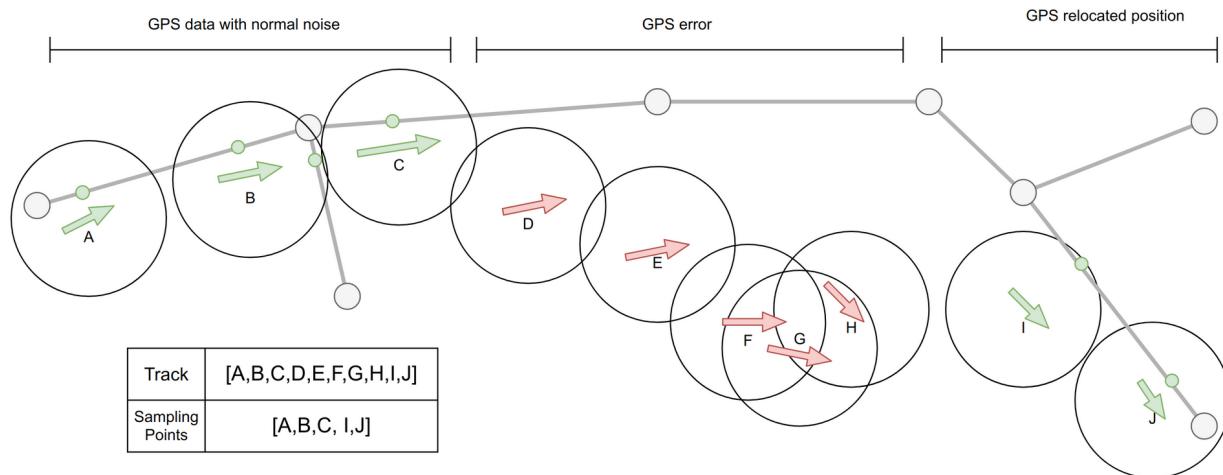


Fig. 5.7: Track to sample points

Beginning with the first sampling point, a routing is performed to the subsequent one. After a successful routing we search from the end of our found routing to the next sampling point and perform the next routing. Iteratively like that we get a matching of our whole track.

However an iterative matching approach like that can lead into situations where no routing can be performed even thou the real world track might would have been plausible. Let us take our example from *Track to sample points* and add a one-way street at an unfortunate position, which makes the track point *E* a sampling point with a candidate. Now our iterative approach will find a route connecting the candidates of sampling point *A* until *E* but cannot get further.

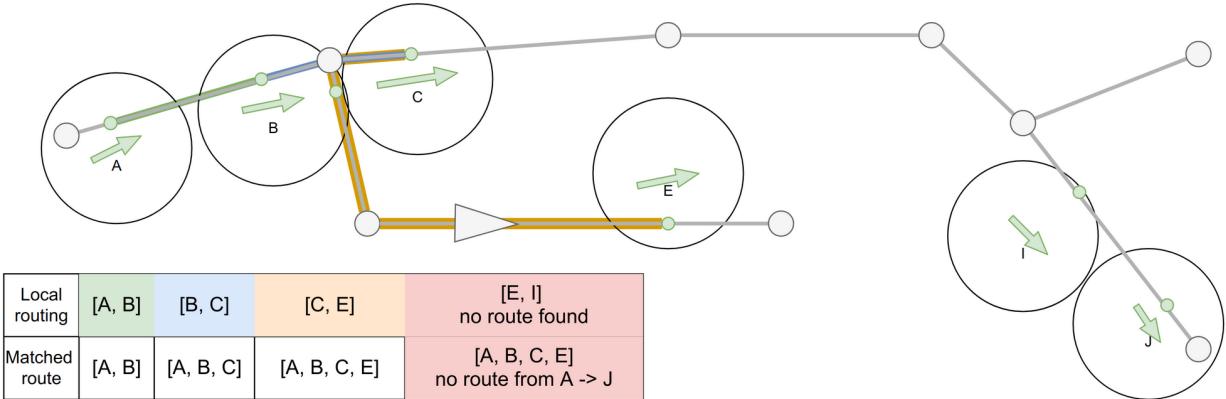


Fig. 5.8: Matching leads into dead end

The *OS-Matcher* idea is the usage of backtracking (see [backtracking](#)).

**Note:** One of the major assumptions is that the *OS-Matcher* is only routing between two points which are relatively near to each other. Like that routing for each possible candidates pair as described in [Routing](#) remains cheap since the routes are short considering only some street segments sometimes only one. However, if a track has many big holes (e.g. due to connection loss for an GPS vehicle track) this can lead to an increase in computing complexity if the edge points just before and after the hole have several to many candidates.



## ALGORITHMS

The *OS-Matcher* uses a hierarchy of routers, each addressing a specific domain of routing targets at a different scale. The innermost router is responsible for routing on the street graph while the outermost tries to find routes as continuously as possible with the help of its underlying routers.

---

**Note:** As described in the [Overview](#), we have a *street map*  $M$  and *sampling points*  $S_1$  to  $S_n$  which are to be mapped onto  $M$ .

To prepare the algorithms, a *simple directed graph*  $G$  gets built from the street map  $M$ , while interconnecting them with each other.

---

In the following sections, we describe and illustrate all the routers, beginning with the innermost one.

### 6.1 Dijkstra router

While building the route, it is essential to find the shortest path from one node on the *street graph* (which corresponds to a node on the *street map*) to another.

To find the path, the [Dijkstra's algorithm](#) is used.

---

**Note:** The *cost function* is set to the length of the geometry representing the edge in the *street graph*, calculated with the [haversine formula](#).

---

#### 6.1.1 Example

In the following example (see Fig. 6.1), we want to find a path from node **Start** to node **Goal**.

After executing the Dijkstra's algorithm, we get the shortest route  $r$  as illustrated in Fig. 6.2.

---

**Note:** As noted in [Matching](#), the routes are in general very short. Between two *sampling points*, many different routes may get calculated if the *sampling point candidates* were projected to different street segments.

---

In the following chapters we learn how the small routes between the street map nodes are used to build the final route.

---

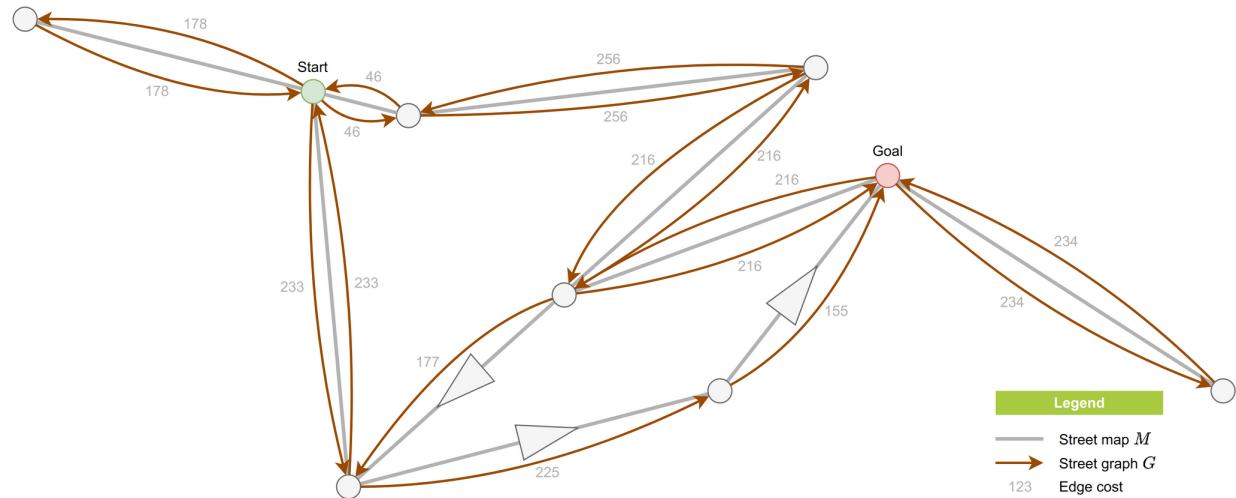


Fig. 6.1: Street map and street graph

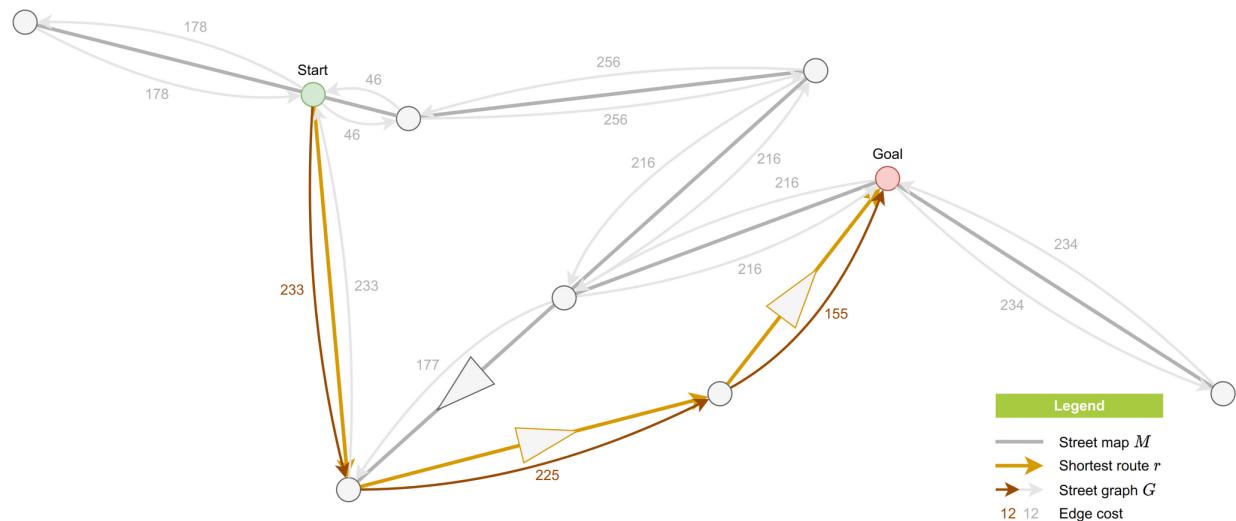


Fig. 6.2: Shortest route

## 6.2 Directed candidate router

The *OS-Matcher* needs to find *valid navigation routes* between *sampling point candidates*, which is done by this router.

**Note:** The sampling point candidates are mapped onto the street map geometry and have no corresponding nodes on the street map/graph (see *Track point projection*).

To be able to calculate a single route and to simplify this router, direction information is required, additionally to the two sampling point candidates, as an input - the *sampling points selection*.

The router then tries to find the shortest route between these two points. Depending on the topology, it may use the underlying *Dijkstra router*.

Finally, the configuration options given in *routeRestrictions* are evaluated and may discard the found route. Another route may then be chosen by the next-outer routers.

### 6.2.1 Examples

In the following examples, we want to find a route from sampling point candidate  $C_1^A$  to  $C_1^B$ .

Here, we focus on route construction; the *routeRestrictions* evaluation is not further described.

#### Single edge, simple case

In the case that both candidates lay on the same street map/graph edge and  $C_1^A$  is headed towards  $C_1^B$  while both having the same orientation, no street map/graph node needs to be traversed, as shown in Fig. 6.3.

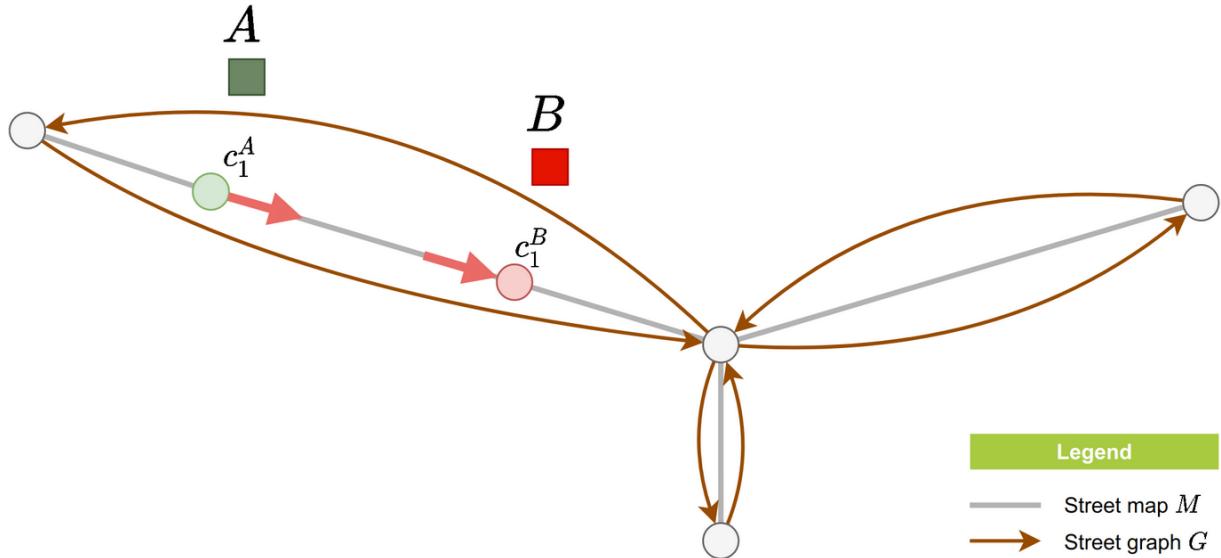


Fig. 6.3: Single edge

The route can easily be constructed by following the geometry of the street map edge and inserting the coordinates of  $C_1^A$  and  $C_1^B$  at the respective ends.

### Other cases

If  $C_1^B$  does not lay on the same edge as  $C_1^A$  or the orientation does not match as in the previous example, this router has to seek to the next nodes first.

This is needed, because the next-inner router is only able to route from street map/graph nodes.

The following examples show the two possible outcomes.

### Identical nodes

It may happen, that the nodes are identical, as illustrated by the blue-ish circle in Fig. 6.4.

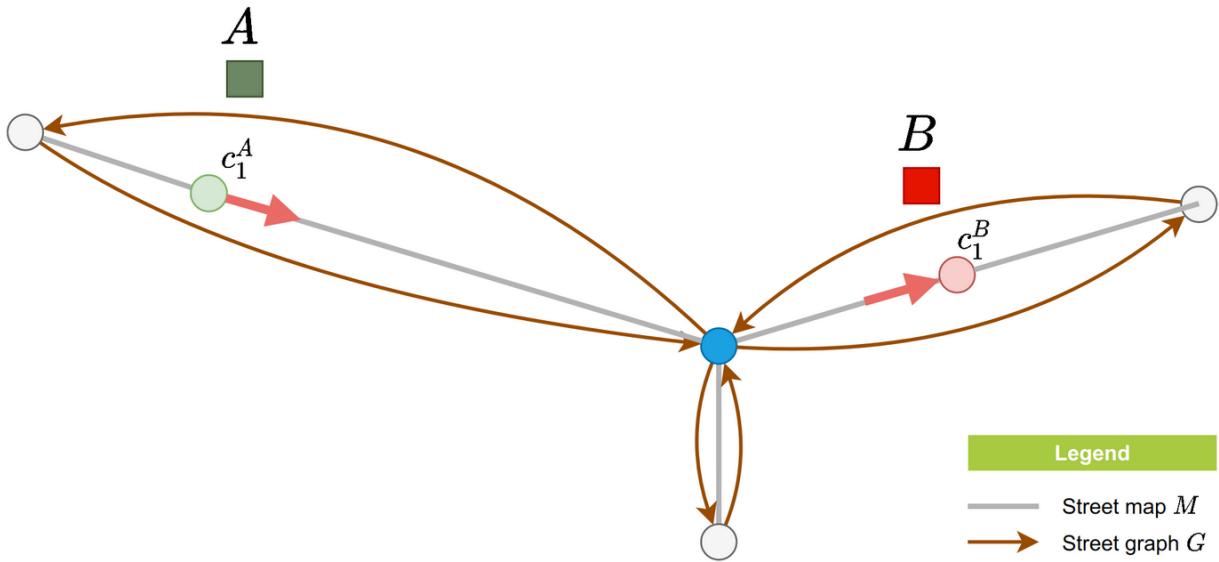


Fig. 6.4: Neighboring edge

No further routing is necessary to construct the route, similar to the first example.

### Different nodes

In the last example, the nodes are not identical (blue-ish circles in Fig. 6.5).

---

**Note:** This case with different nodes would also apply in our first example, if at least one of the sampling point candidates were oriented differently. Note that this orientation is not derived from the heading of the track but rather an experiment of the next-outer router (all four possibilities may be tried, but in general the orientation of the first sampling point is fixed due to the preceding route).

---

To complete the route from and to the blue-ish nodes, this scenario requires routing over street map/graph edges via the *Dijkstra router*.

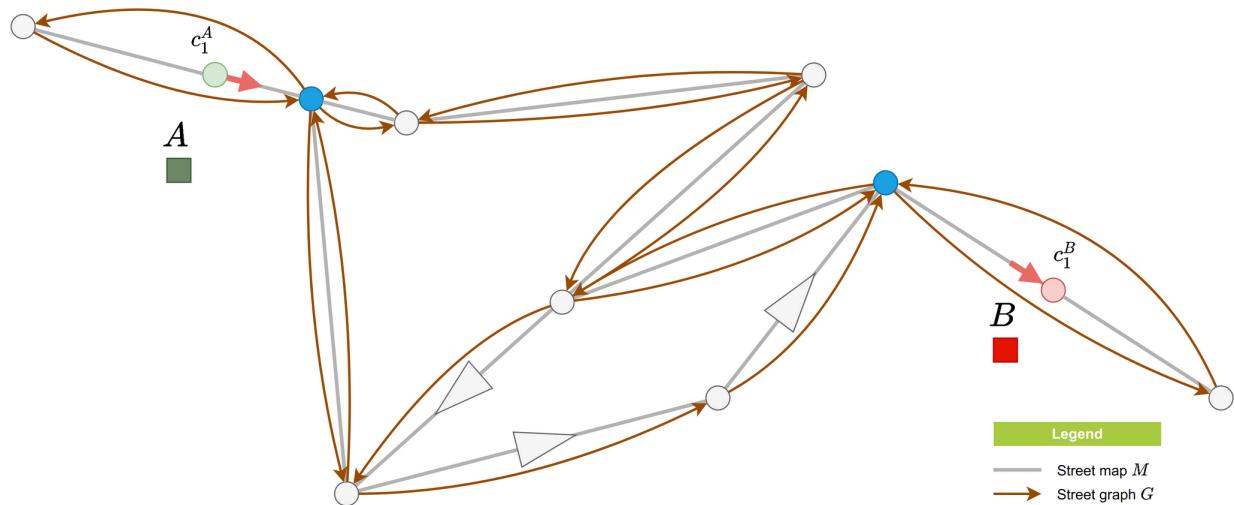


Fig. 6.5: Far edge

## 6.3 Sampling point router

This router is responsible for finding the best route between two given *sampling points* *Start* and *Goal*. It is composed of the following three operations.

### 6.3.1 1. Possible routes calculation

Because a sampling point may have more than one candidate and a driver can cross the projected point in two directions, there are many possible options how to begin the routing between the two sampling points, as we see in Fig. 6.6.

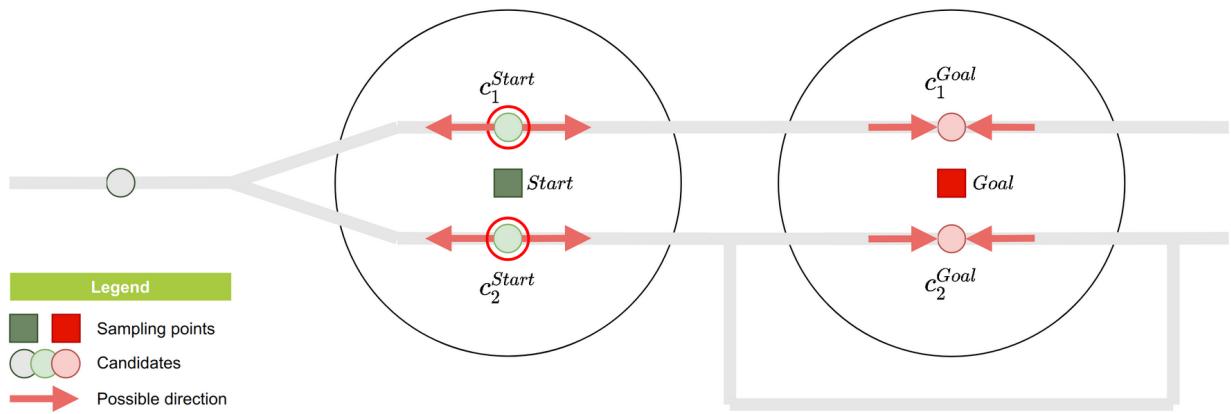


Fig. 6.6: Motivation

The underlying *Directed candidate router* needs a single *sampling points selection* which consists of the following data:

- The two *sampling point candidates* *Start* and *Goal*.
- Direction information for each sampling point candidate.

The set of possible *sampling points selections* is reduced by two givens (see also *Routing*):

- The initial direction of the *Start* sampling point, induced by the preceding route (if it's not the first sampling point of a new route).
- The allowed travel directions.

### Example

In the following example, the current route ends at sampling point candidate  $C_2^{Start}$ , that's why routes from other candidates (here  $C_1^{Start}$ ) are not considered:

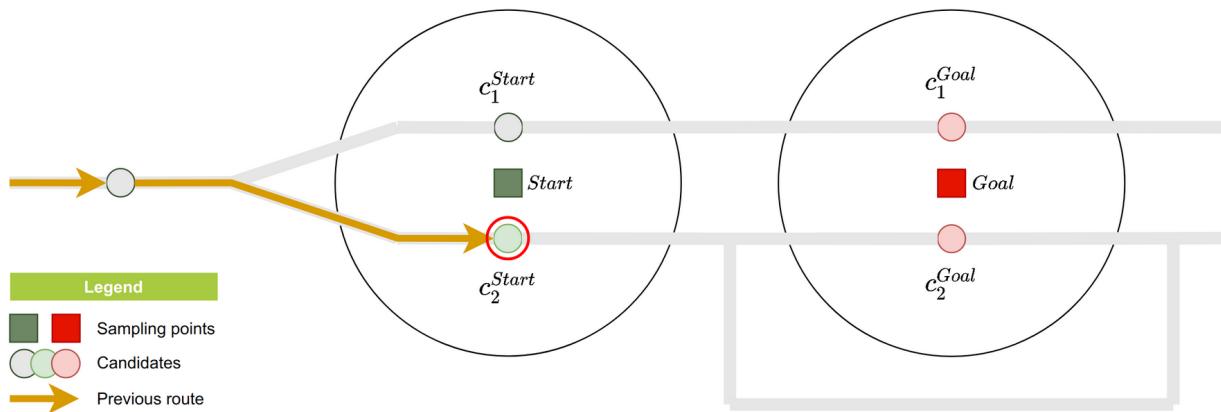


Fig. 6.7: Overview

In Fig. 6.8 and Fig. 6.9 two possible routes have been calculated by the underlying router for two possible directions of  $C_2^{Goal}$ .

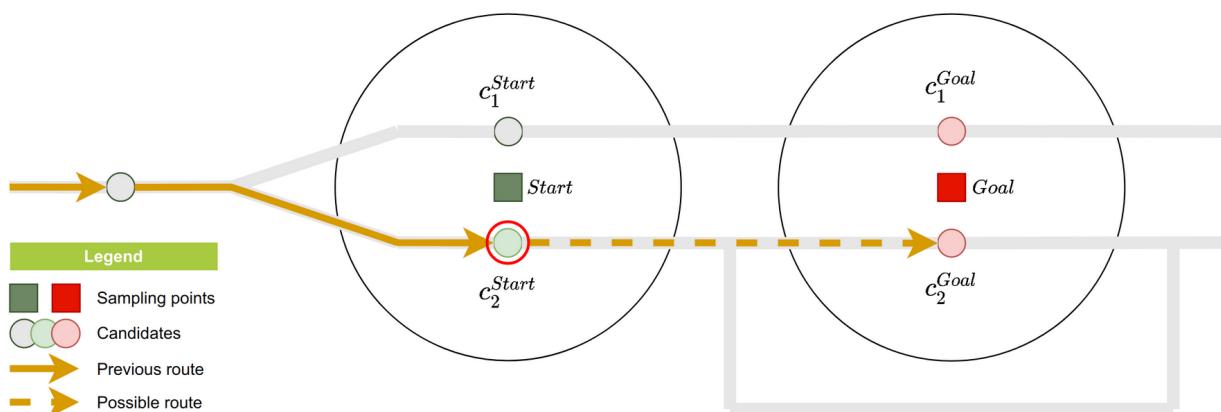


Fig. 6.8: Possible route 1

If *Router filter's allowSelfIntersection* is set to true, also the following route would be possible:

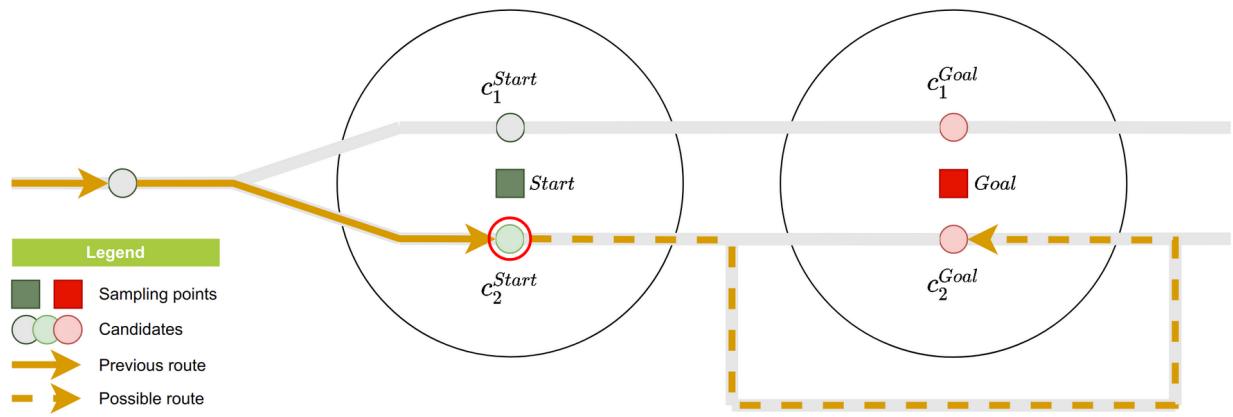


Fig. 6.9: Possible route 2

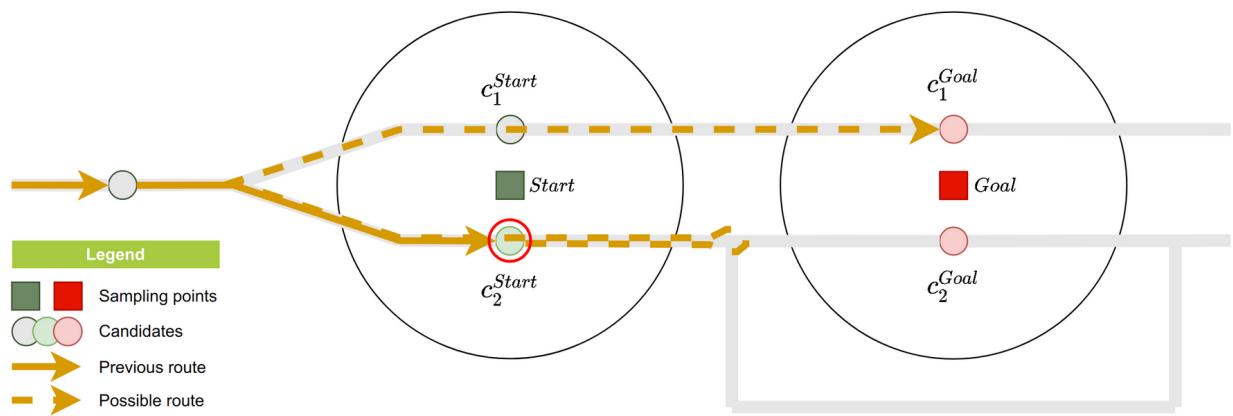


Fig. 6.10: Possible route 3

### 6.3.2 2. Routes clustering

All found routes are now clustered as described in [Clustering](#).

### 6.3.3 3. Route selection

The role model of all clusters are then sorted as described in [Final Evaluation](#) determining the final, most realistic route.

## 6.4 Backtrack router

This router shall find the *farthest route possible* beginning at *sampling point Start*.

Basically, it does so by successively routing from one sampling point to the next with its underlying candidate\_router.

But if it at any point is not able to find a valid navigation route, it is able to go back to previous sampling points in history (up to the configured maximum, *maxCandidateBacktrackingDistance*) and re-route from other candidates. That means it tries out other routes that were not considered optimal in the first place.

If it is able to route farther, the backtracking is considered successfull and it continues as usual. It then may start a new backtracking session, if necessary.

When backtracking is not successfull, the router returns the route before the last backtracking session has begun.

### 6.4.1 Example

In this example the driver takes a freeway exit ramp, but the GPS tracker has poor signal and does not catch the turn fast enough. As a consequence the best candidates are sitting next to the main freeway lane, as illustrated below.

As a consequence the routers will route on the main freeway lane and will not be able to continue at some point. Then backtracking starts.

While backtracking, already passed sampling points are considered again while choosing different candidates. In the following three images we see the algorithm going back and trying out other candidates.

As you see in the next image, the router finally finds a valid navigation route by choosing a different candidate for the southernmost sampling point.

In this simple example, the router has found a route by just trying out the next best candidate for the previous sampling points.

Here are some not yet mentioned features:

- While backtracking, the router may go back beyond already backtracked passages, but will not try out already visited routes.
- While backtracking, the router may go back beyond the **Start** sampling point.
- It may go beyond skipped sampling points (we will discuss this later in [Skip router](#)).
- It may not go beyond the current route (which could be in the middle of the track, we will discuss this later in [Piecewise router](#)).

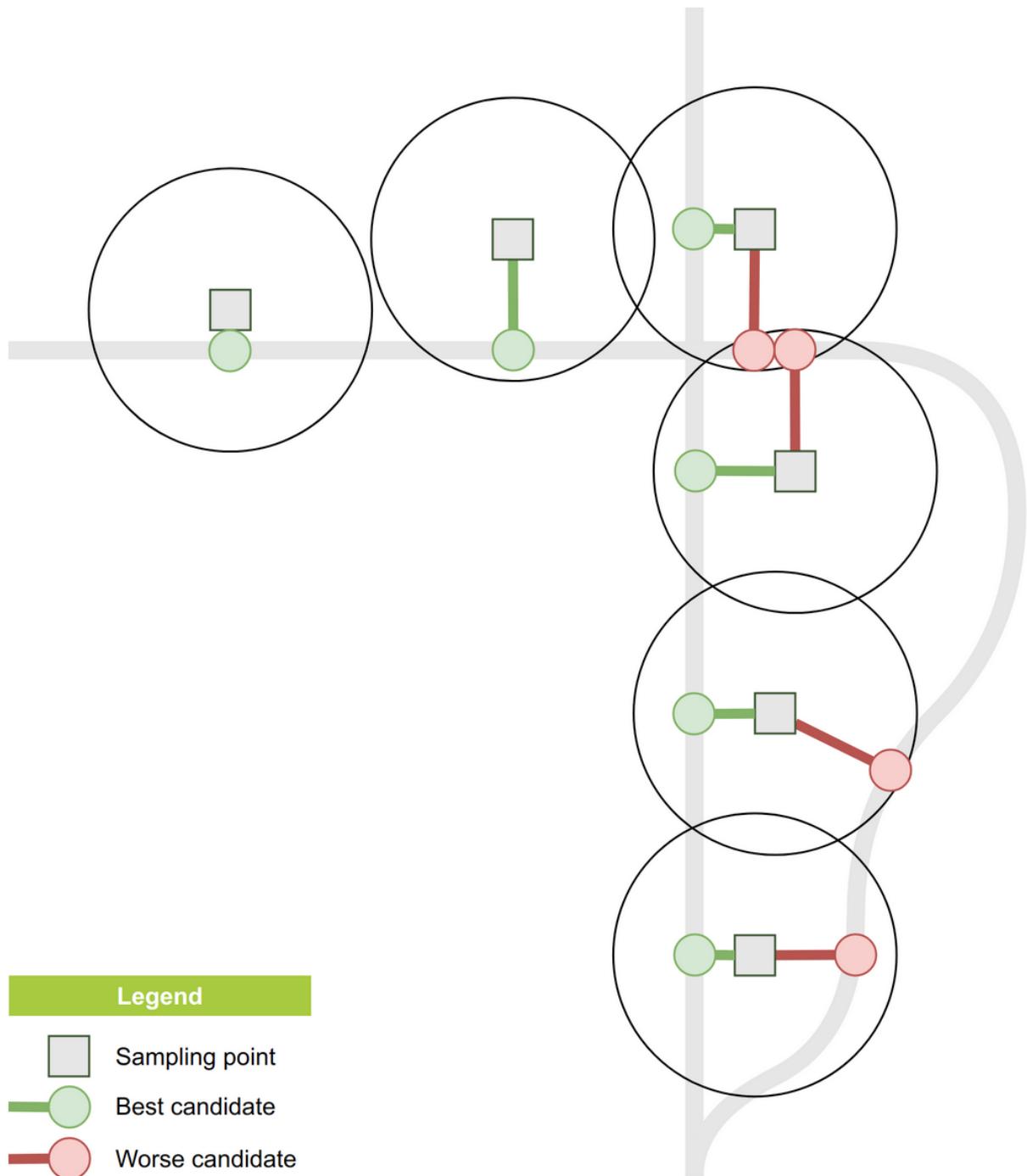


Fig. 6.11: Freeway exit ramp example

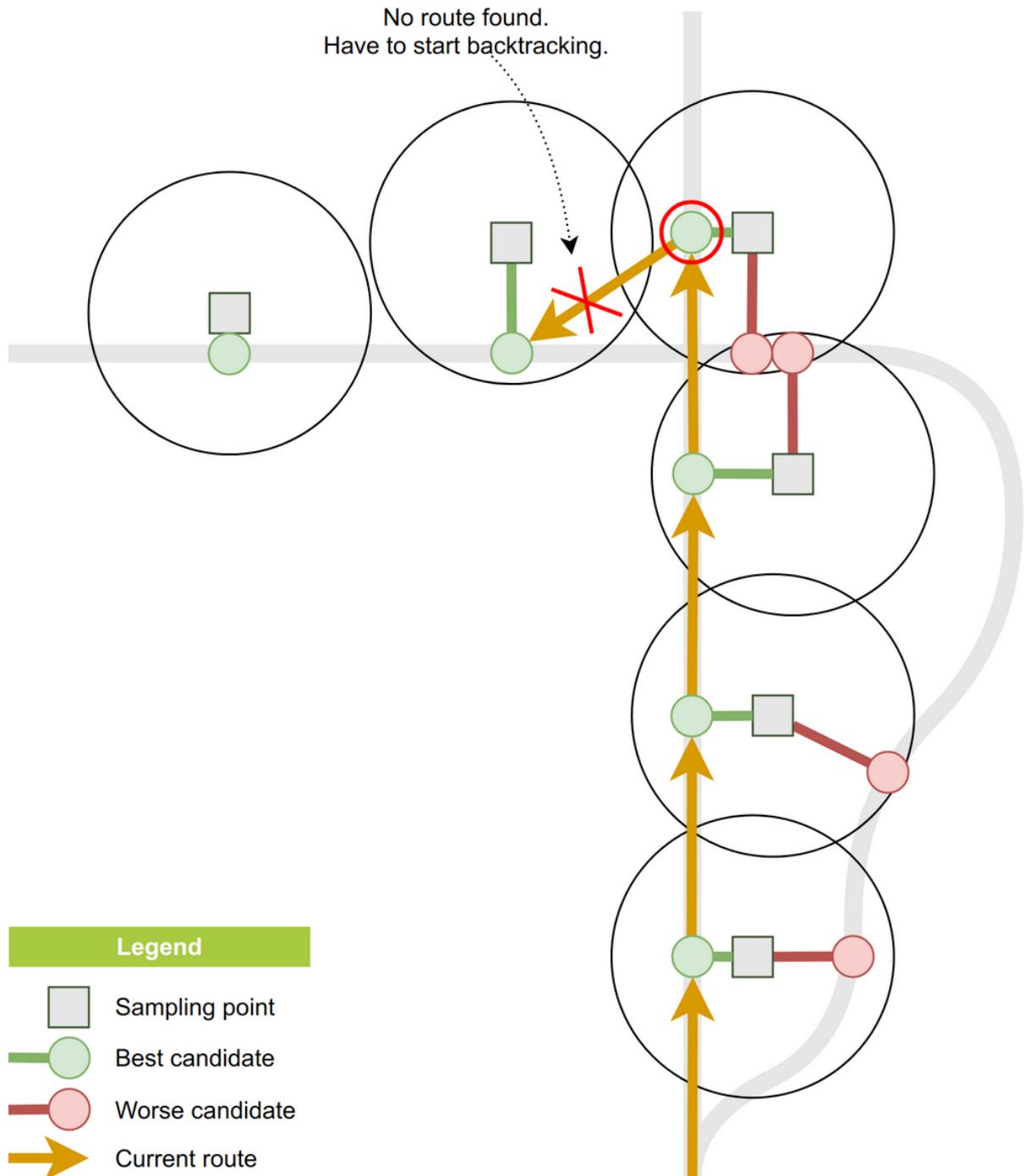


Fig. 6.12: Needs backtracking

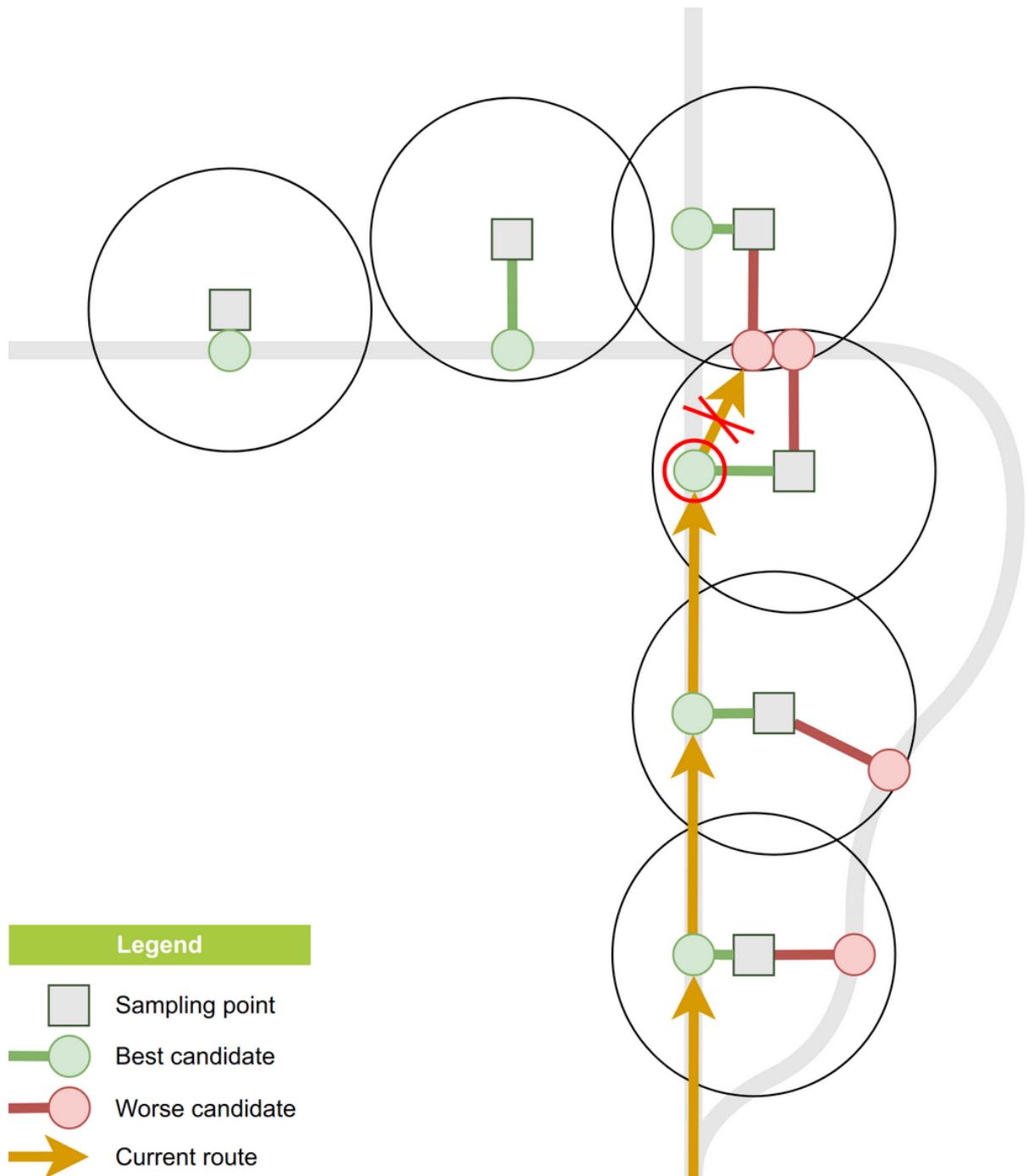


Fig. 6.13: Backtracking trial 1 failed

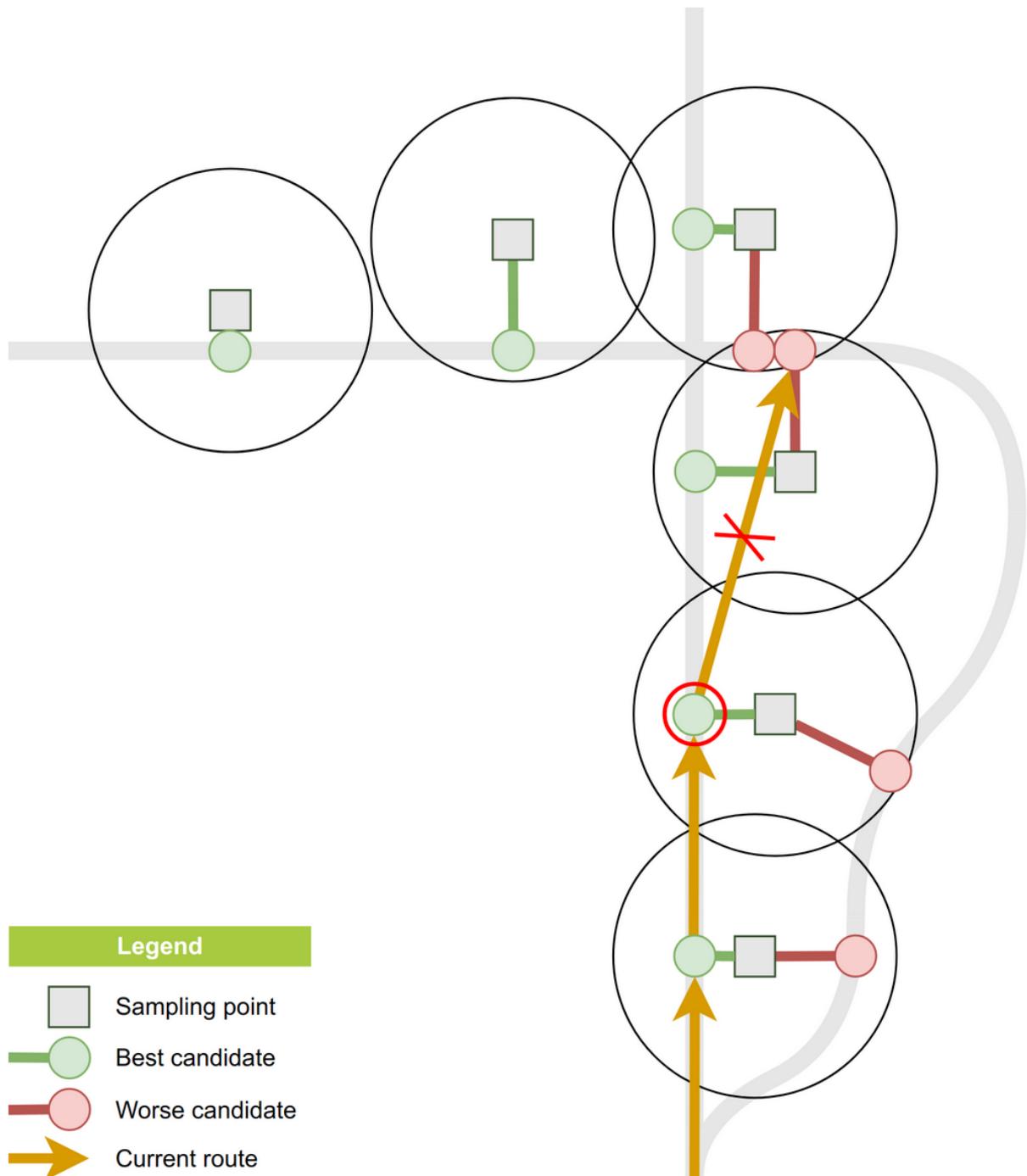


Fig. 6.14: Backtracking trial 2 failed

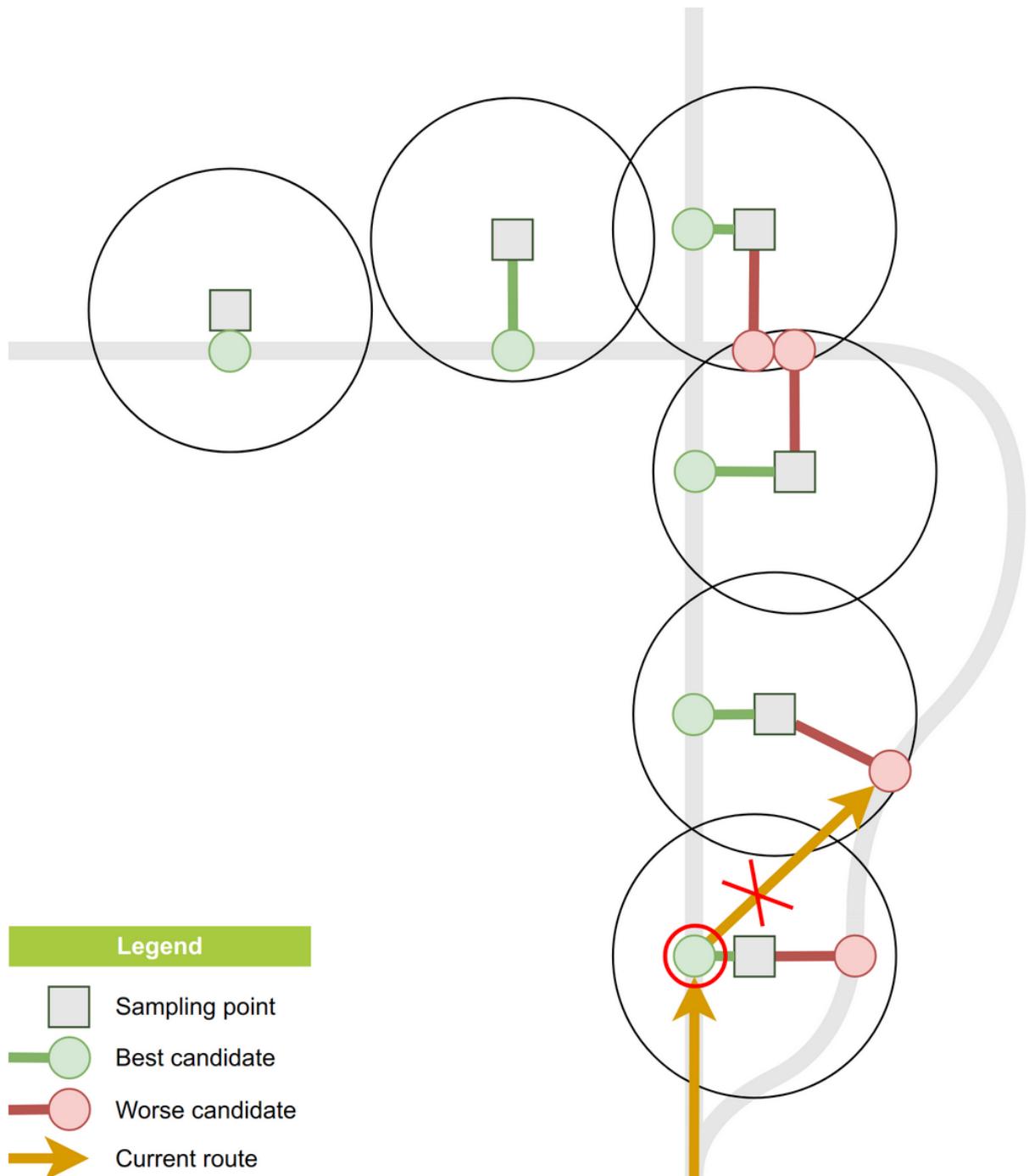


Fig. 6.15: Backtracking trial 3 failed

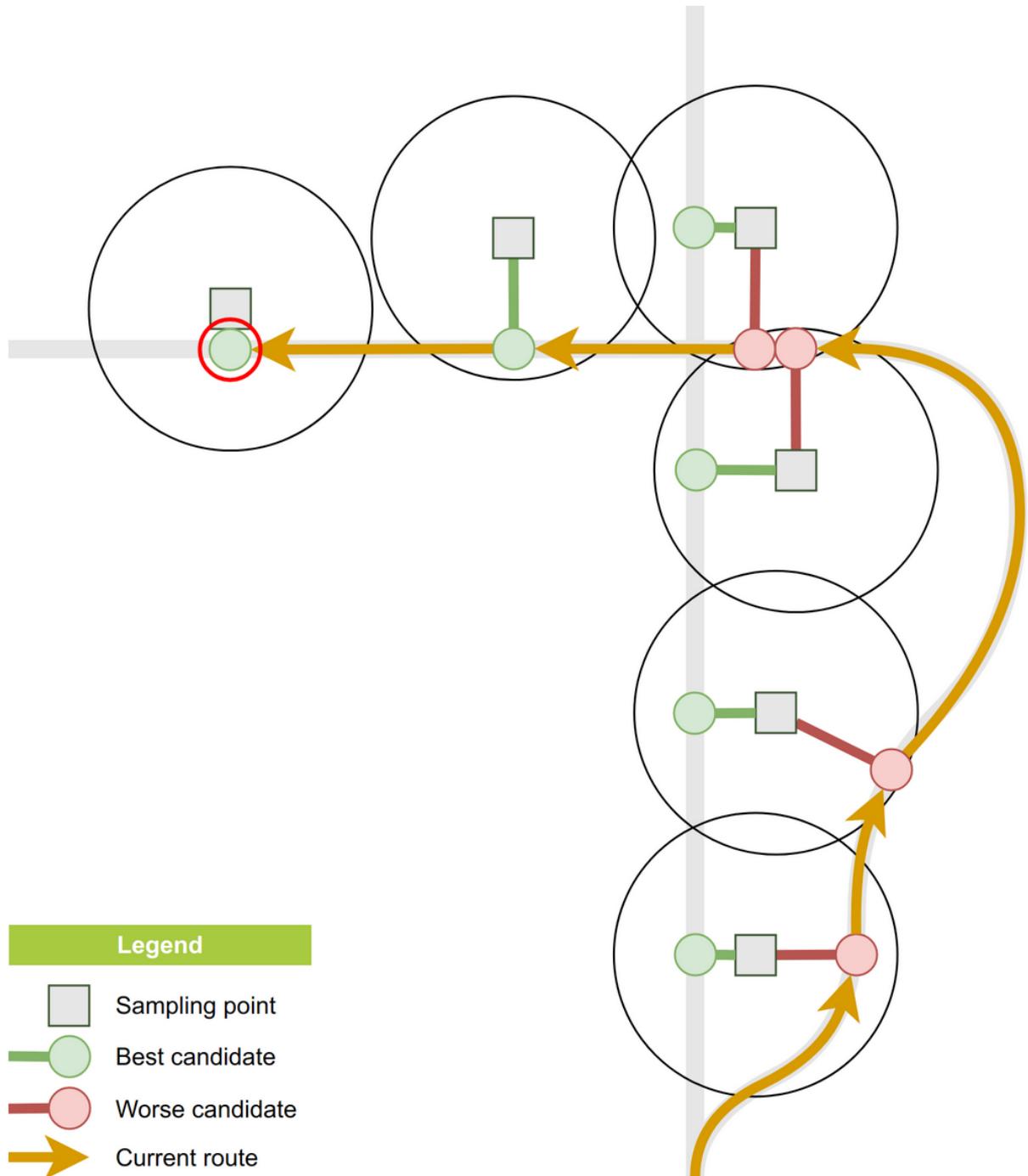


Fig. 6.16: Backtracking successful

## 6.5 Skip router

This router tries to find the *farthest route* with the underlying *Backtrack router*. If beneficial, specific *sampling points* are requested to be ignored by the underlying routers.

Each time the underlying router has found its farthest route and this route does not reach the sampling point **Goal**, the probability is high that some of the last sampling points's track positions were recorded with an inaccurate GPS source or due to street map errors. The router therefore starts to skip consecutive sampling points in a specific, skip-distance-dependant order, starting with the last sampling point of the farthest route or the following. The underlying routers then try to route farther by ignoring these sampling points.

If it is able to route farther, the skipping is considered successfull and it continues as usual. It then may start a new skipping session, if necessary.

If the distance to skip reaches some threshold before finding a route, the router returns the route before the last skipping session has begun.

### 6.5.1 Example

In this example, the underlying router failed to route beyond sampling point 4, as shown below.

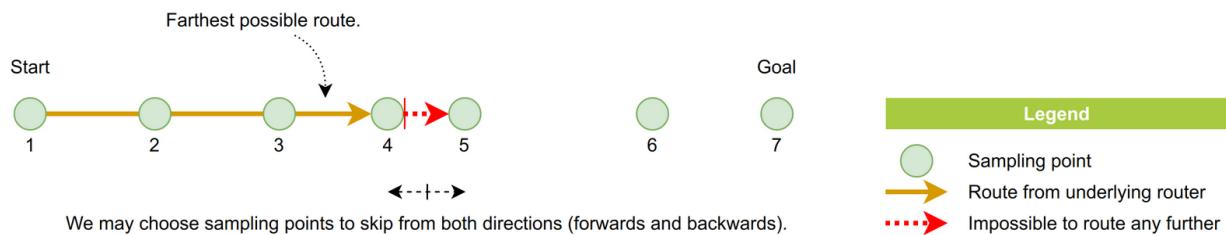


Fig. 6.17: Farthest route does not reach Goal

The router now starts skipping sampling points consecutively from both directions until it is able to find a route.

To decide whether it should skip over a sampling point from behind or from ahead, it calculates the linear distances that need to be skipped (from sampling point to sampling point) and chooses the option where this is minimal.

---

**Note:** Depending on the chosen strategy (*samplingPointSkipStrategy*), the distances are measured from the sampling points before/after the to-be-skipped sampling points (*includeEdgeCosts*) or from the outermost sampling points that are to be skipped (*excludeEdgeCosts*). In the latter case the costs are zero for the first trial; as a rule the sampling point ahead is choosen to be skipped first.

---

**Note:** The selection process tries out three different consecutively growing sampling point selections in parallel and always chooses the next with the minimal distance to be skipped:

1. The first growing selection consists of the sampling point *ahead and the following*.
  2. The second growing selection consists of the sampling point *before and the preceding*.
  3. The third selection can grow in *both* directions, depending on which next sampling point to skip has the minimal distance to the one before.
- 

For simplicity, we now focus on the strategy *includeEdgeCosts* and assume that only the third growing selection process is used (growing to *both* directions).

This selection process is illustrated in Fig. 6.18.

**Note:** The distance is always measured as a linear distance, beginning at one of the first two sampling points causing the router to start the skipping process (in our example **4** and **5**). It is calculated using the [haversine formula](#).

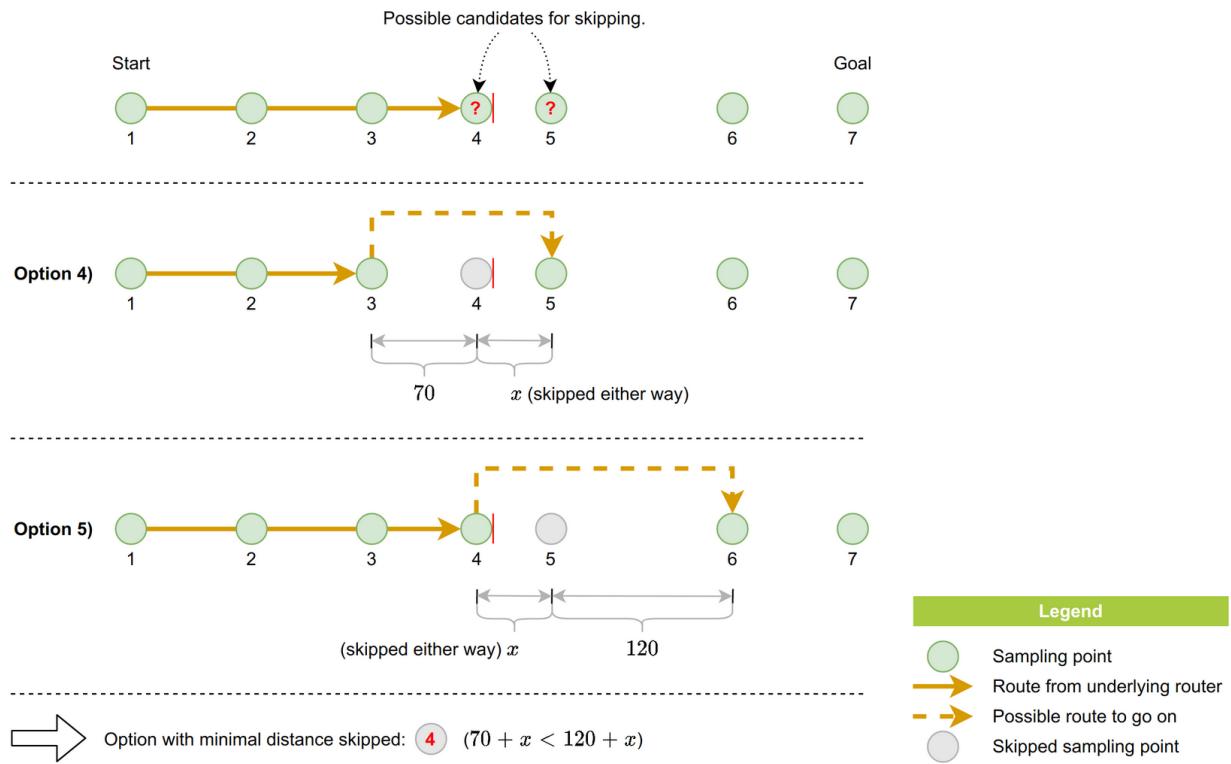


Fig. 6.18: Choosing sampling point to skip

In our example the router choosed to skip sampling point **4**.

Now the routing continues, but the underlying router still does not find a route to go on. So the selection process is repeated, as shown in Fig. 6.19.

Now sampling points **4** and **5** are requested to be skipped.

Again, the underlying router failes to find a route and selection process is repeated, as shown in Fig. 6.20.

Now sampling points **3** to **5** are requested to be skipped.

Finally, the routing proceeds, illustrated in Fig. 6.21.

Sampling points **3**, **4** and **5** are ignored now. However, the route is still consecutive.

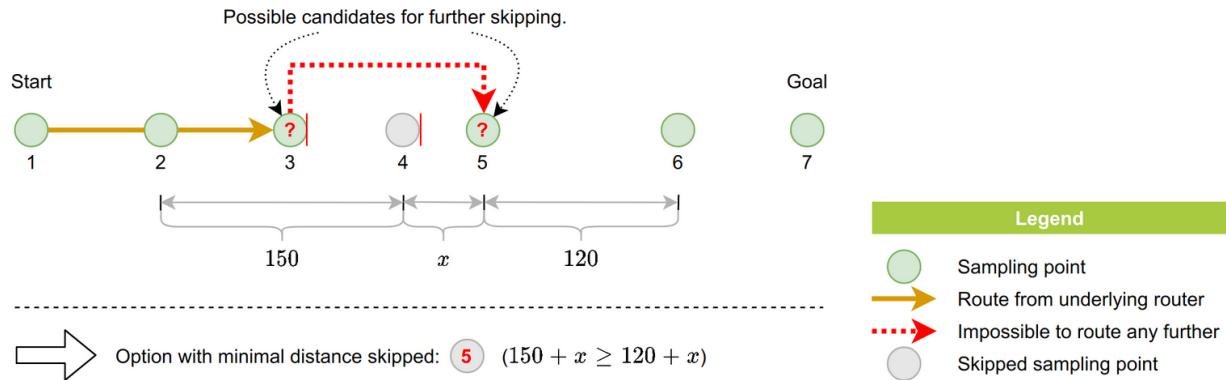


Fig. 6.19: Skipped sampling point 4

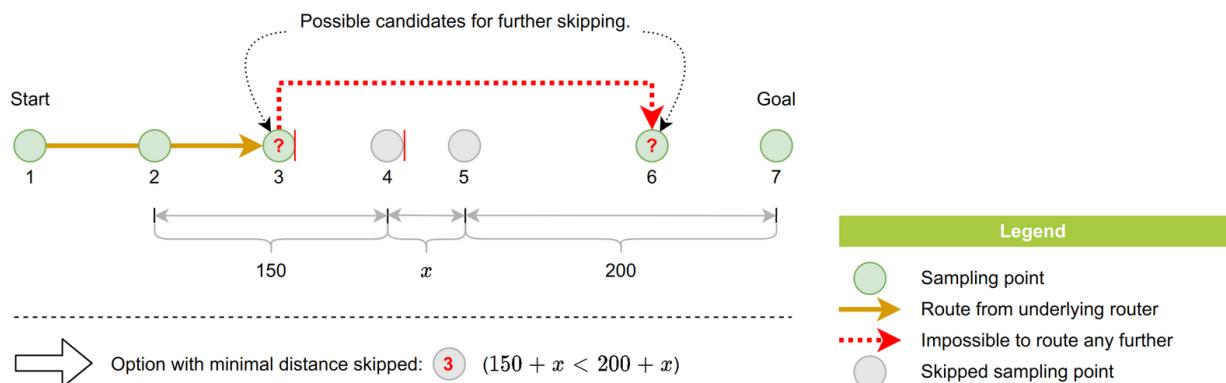


Fig. 6.20: Skipped sampling points 4 and 5

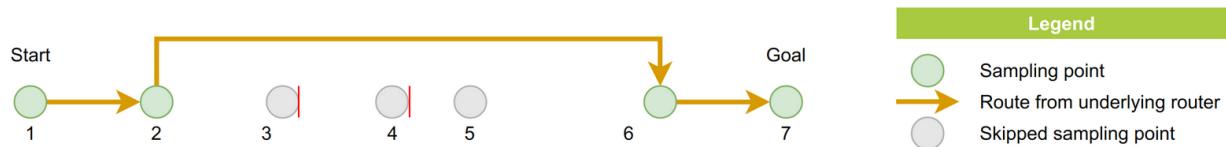


Fig. 6.21: Routed farther by skipping 3 sampling points

## 6.6 Piecewise router

This router is responsible for finding consecutive *routes as far as possible* until reaching the *sampling point Goal*.

Starting at the sampling point **Start**, it uses the underlying *Skip router* to find the farthest consecutive route. If the target sampling point could not be reached, it starts a new route and repeats the process, until the route is complete.

---

**Note:** As this is the outermost router, gaps produced here will result in actual gaps in the final route. However, this will rarely happen, as the underlying routers are capable of resolving most of the routing problems that could arise.

---

### 6.6.1 Example

In this example we try to find a route from the sampling point **1** to the sampling point **7**.

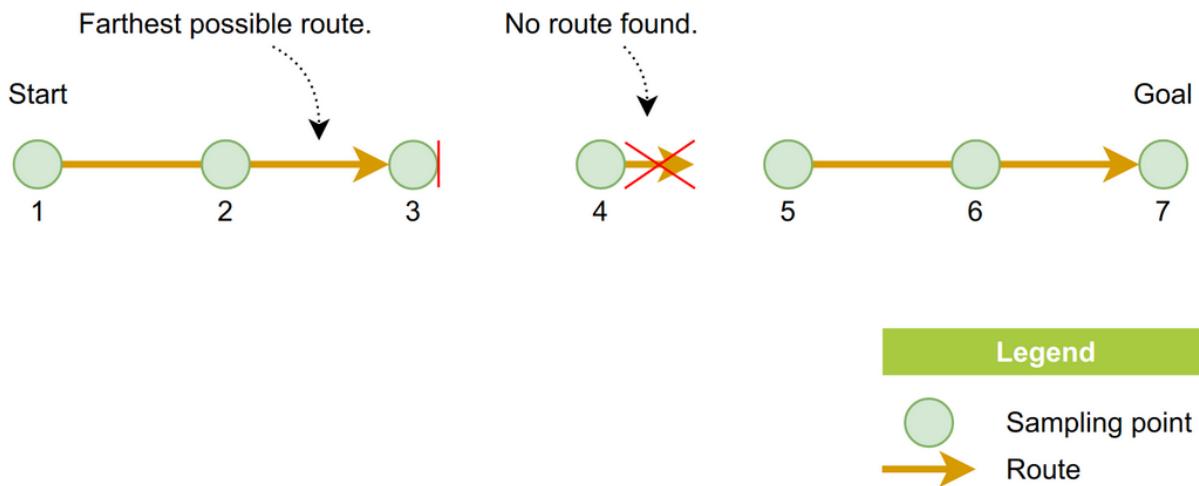


Fig. 6.22: Router functionality

Starting at sampling point **1**, the underlying router finds its “longest” route no farther than to sampling point **3**.

A new route is then searched, starting at sampling point **4**.

---

**Note:** Note that starting a new route from the next sampling point is different from skipping a sampling point. The latter actually results in a consecutive route, only that we would have ignored a sampling point (by routing around). In the case we have in our example, it is indeed not possible for the subsequent routers to route any further without having to skip too many sampling points and reaching some threshold.

---

As no consecutive route could be found here, the next route is then searched from sampling point **5** and finally reaches its goal at sampling point **7**.

## FILTERS

Each processing step of an application is implemented as a *filter*.

Here you can find the basic data flow (simplified filter pipeline):

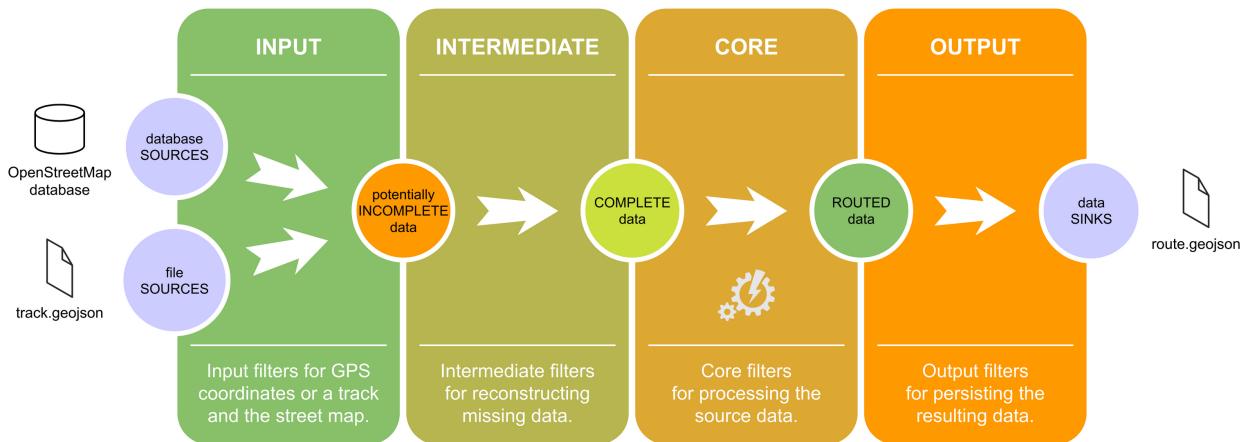


Fig. 7.1: Filter dataflow

A filter is a functor which can read, process and write specific data.

An application holds all relevant data structures, which are to be passed to filters, in its *context*.

A filter may have obligatory preconditions (*requirements*) and/or optional preconditions (*optionals*). These can either be other filters or *features*, which must be satisfied before execution. After an execution of a filter, it satisfies specific features (*fulfillments*).

The *OsmMapReader class* for example has the obligatory precondition *PointList* and satisfies (after successful execution) the features *SegmentList*, *NodePairList*, *TravelDirectionList* and *HighwayList*. This is reflected by the parameters of its *operator()*, where the preconditions are constant references and the satisfiable variables are writable references:

```

1 OsmMapReader::OsmMapReader( /* ... */ ) : Filter( "OsmMapReader" ), /* ... */
2 {
3     setRequirements( { "PointList" } );
4     setOptionals( {} );
5     setFulfillments( { "SegmentList", "NodePairList", "TravelDirectionList",
6     ↳ "HighwayList" } );
7 }
8
9 bool OsmMapReader::operator() (

```

(continues on next page)

(continued from previous page)

```

9   Types::Track::PointList const & pointList,
10  Types::Street::SegmentList & segmentList,
11  Types::Street::NodePairList & nodePairList,
12  Types::Street::TravelDirectionList & travelDirectionList,
13  Types::Street::HighwayList & highwayList )
14 {
15  /* ... */
16 }
```

Based on these definitions (*requirements*, *optionals* and *fulfillments*) a multithreaded execution pipeline can be created automatically by adding the filters to the *pipeline* (see below).

You may adapt the definition of OsmMapReader to your needs or write your own filter and apply the above concept analogously. In this chapter we are providing descriptions for input, output and the available configuration parameters of all implemented filters of the *OS-Matcher*.

### List of filters

## 7.1 CsvTrackReader

This filter receives a track in form of a CSV stream and produces an output which can be further processed by other filters, e.g. the routing filters.

### 7.1.1 Input

- CSV stream with the following content of a row: *time*, *latitude*, *longitude*, *heading*, *velocity*
- where the columns will need to have the following format:
  - **time**
    - \* date/time in the format “yyyy-MM-ddThh:mm:ss” [ISO-8601] (example: 2020-01-01T12:00:00)
  - **latitude**
    - \* floating-point (interpretable by `std::stod`)
  - **longitude**
    - \* floating-point (interpretable by `std::stod`)
  - **heading**
    - \* floating-point (interpretable by `std::stod`)
  - **velocity [m/s]**
    - \* floating-point (interpretable by `std::stod`)

### 7.1.2 Output

- TimeList
- PointList
- HeadingList
- VelocityList

### 7.1.3 Configuration

- None

## 7.2 JsonTrackReader

This filter receives a track in form of a JSON stream and produces an output which can be further processed by other filters, e.g. the routing filters.

### 7.2.1 Input

A JSON stream which contains a list with JSON objects. The JSON objects are having the following format:

- **pos\_time**
  - text in the format “YYYY-MM-DDTHH:MM:SSZ”
- **latitude**
  - decimal number
- **longitude**
  - decimal number
- **course [degrees]**
  - decimal number
  - describes the heading
  - if available
- **speed [km/h]**
  - decimal number
  - describes the velocity
  - will be converted to [m/s]
- Example of the JSON stream:

```
[  
  {  
    "pos_time": "2018-07-03T04:22:53Z",  
    "latitude": 50.050303,  
    "longitude": 8.6567,  
    "course": 291,
```

(continues on next page)

(continued from previous page)

```
        "speed": 16,  
    },  
    ...  
]
```

## 7.2.2 Output

- TimeList
- PointList
- PartialHeadingList
- VelocityList

## 7.2.3 Configuration

- None

## 7.3 GeoJsonMapReader

This filter receives a *street map* in form of a GeoJSON stream and produces an output which can be further processed by other filters, e.g. the routing filters.

### 7.3.1 Input

- **feature:**
  - **LineString** [[longitude, latitude], [longitude, latitude], ... ]
- **properties per feature:**
  - **Id:** Original ID of the street from which this segment originates
  - **Offset:** Node index at which this segment starts in the original street (see *street segment origin offset*)
  - **SourceNode:** Node identifier for the first point of the segment
  - **TargetNode:** Node identifier for the last point of the segment
  - **TravelDirection:** Allowed travel direction; one of [Both, Forwards, Backwards]
  - **Highway:** Highway type; one of [Motorway, Trunk, Primary, Secondary, Tertiary, MotorwayLink, TrunkLink, PrimaryLink, SecondaryLink, TertiaryLink, Unknown]
- **Example:**

```
{  
    "type": "FeatureCollection",  
    "features": [  
        {  
            "type": "Feature",  
            "geometry": {  
                "coordinates": [  
                    [
```

(continues on next page)

(continued from previous page)

```

        [
            [
                12.6079441,
                52.9935395
            ],
            [
                [
                    12.6103468,
                    52.9915834
                ],
                [
                    [
                        12.6110675,
                        52.9909935
                    ]
                ],
                "type": "LineString"
            ],
            "properties": {
                "Id": 265483627,
                "Offset": 0,
                "SourceNode": 16,
                "TargetNode": 9,
                "TravelDirection": "Forwards",
                "Highway": "Motorway"
            }
        },
        ...
    ]
}

```

### 7.3.2 Output

- SegmentList
- NodePairList
- TravelDirectionList
- HighwayList

### 7.3.3 Configuration

- None

## 7.4 OsmMapReader

This filter receives a *street map* from OpenStreetMap by connecting to a PostGIS database and produces an output which can be further processed by other filters, e.g. the routing filter.

### 7.4.1 Input

- **Connection**
  - Provides the connection to the PostGIS database with the street map from OpenStreetMap.
- **PointList**
  - This PointList is an obligatory precondition for the OsmMapReader providing the track points. These track points are the basis to build a spatially limited street map to avoid the delivering of the complete street map.

### 7.4.2 Output

- *SegmentList*
- *NodePairList*
- *TravelDirectionList*
- *HighwayList*

### 7.4.3 Configuration

- **highwaySelection: `unordered_set<HighwayType>`** Specifies a filter for the *highway type*, based on OpenStreetMap Key:highway.
- **fetchCorridor: `double`** Specifies a buffer to extend the *track* before searching. See *useSingleSearchCircle* below for details.
- **useSingleSearchCircle: `bool`**
  - **false:** Search within a tube around the *track*. The search will be performed with the PostGIS function `ST_DWithin` as follows: `ST_DWithin( <track linestring>, planet_osm_line.way, <fetchCorridor> )`. If the distance between two consecutive points is too long, this can result in gaps (missing *street segments*).
  - **true:** Search within a big circle around the track. Its center is the midpoint of the line between the first and the last point of the *track*, its radius is the  $(\text{distance of this two points}) / 2 + \text{fetchCorridor}$ . If the distance between startpoint and endpoint is too small this can result in gaps (missing *street segments*). Imagine the route between these two points would proceed partially out of the circle. Then some necessary parts of the map (out of the circle) would be missing and the matching cannot be complete.

## 7.5 SamplingPointFinder

This filter projects the *track* onto the *street map* by creating a sampling point list for each track point. For more information we recommend reading the chapter *Track point projection*.

### 7.5.1 Input

- **mandatory**
  - *PointList*
  - *SegmentList*
  - *TravelDirectionList*
- **optional (mutually exclusive - only one of the following may be given) :**
  - *HeadingList*
  - *PartialHeadingList*

### 7.5.2 Output

- *SamplingPointList*

### 7.5.3 Configuration

- **selectionStrategy: SelectionStrategy**
  - one of the values in [all, best, single]
  - **all:** all candidates are added to the sampling point
  - **best:** only the best candidate is added to the sampling point (see *Candidate search*)
  - **single:** the sampling point is only considered when exactly one candidate is found, which is added
- **searchRadius: [m] floating-point**
  - Value is implicitly used to determine the maximum distance a track point can have from a *street segment*, it is however not used as a ‘radius’.
- **maxHeadingDifference: [degree] floating-point**
  - allowed *heading* difference between a track point and the corresponding candidate (the heading of a candidate depends on the street segment it is projected onto and the driving orientation)
  - only used if heading data is available

## 7.6 GraphBuilder

This filter creates the *street graph* representation of the *street map* which can be further processed by other filters, e.g. the *Router*.

### 7.6.1 Input

- *NodePairList*
- *TravelDirectionList*

### 7.6.2 Output

- *Graph*, bidirectional graph representation
- *GraphEdgeMap*, maps from the graph edges to the *street map* segments
- *StreetIndexMap*, maps *street map* segments to the corresponding edges and node in the *Graph*
- *NodeMap*, maps the street nodes from *NodePairList* to the graph nodes in the *Graph*.

### 7.6.3 Configuration

- None

## 7.7 Router

This filter creates the *route* representation from the *track*, *street map* and its *street graph* representation. It further creates statistical data about the routing. It is therefore the core filter of the *OS-Matcher*.

For further explanation of its functionality, please refer to the *Algorithms* chapter.

### 7.7.1 Input

- **mandatory**
  - *SamplingPointList*
  - *SegmentList*
  - *Graph*
  - *GraphEdgeMap*
  - *StreetIndexMap*
- **optional**
  - *TimeList*
  - *VelocityList*

## 7.7.2 Output

- *RouteList*
- *RoutingStatistic*

## 7.7.3 Configuration

- **maxVelocityDifference: [m/s] double** Based on the time stamps in the *track* data and the length of the route, it gets calculated how fast the vehicle must have been driven. This then gets compared with the mean velocity of the start and end points (ignoring the points inbetween). If the difference is greater than the one given, the route gets discarded.
- **allowSelfIntersection: bool**
  - `false`: Discards routes, whose start or end point occurs one more time within the route.
- **maxAngularDeviation: [degree] double**
  - 360: Deactivates this restriction.
  - Other values: Discards routes which have a greater *heading* difference between a used candidate and its corresponding sampling point. The algorithm to determine this is rather complex, see *function checkMaxAngularDeviation* for the implementation.
- **accountTurningCircleLength: [m] double**
  - add this turning circle length to the route length when routing back the exact same coordinates (when the second-last coordinate equals the current coordinate)
- **maxSamplingPointSkippingDistance: [m] double**
  - maximal allowed distance to jump forwards or backwards
- **samplingPointSkipStrategy: enum SamplingPointSkipStrategy (see Skip router)**
  - `includeEdgeCosts`: The distances are measured from the sampling points before/after the to-be-skipped sampling points.
  - `excludeEdgeCosts`: The distances are measured from the outermost sampling points that are to be skipped.
- **maxCandidateBacktrackingDistance: [m] double**
  - maximal allowed distance to go back to search for other ways
- **maxClusteredRoutesLengthDifference: [m] double**
  - length difference that is allowed for a route to become part of a route cluster (see *Clustering*)
  - should be 4 times *Sampling Point Finder's* `searchRadius`
- **routeClusterPreference: enum RouteClusterPreference (see Clustering)**
  - `cheapest`: Chooses from all best routes of the clusters the route with the lowest routing costs.
  - `shortest`: Chooses from all best routes of the clusters the shortest route.

## 7.8 CsvRouteWriter

This filter writes the internal representation of the *routes* to a stream in **CSV** format.

### 7.8.1 Input

- *RouteList*
- *GraphEdgeMap*
- *NodeMap*
- *TimeList*
- *SegmentList*
- *SamplingPointList*

### 7.8.2 Output

A CSV stream with the following format:

- **CSV columns and corresponding types:**
  - **osm\_ids**: listing of the corresponding OSM IDs (see origin IDs)
  - **route**: List of geo points, LINESTRING [[longitude, latitude], [longitude, latitude], ... ]
  - **length**: [m] floating point
  - **cost**: floating point
  - **sourceNode**: Node identifier for the first point of the segment
  - **targetNode**: Node identifier for the last point of the segment
  - **sourceSamplingPointTime**: date/time in the format “yyyy-MM-ddThh:mm:ss” [ISO-8601] (example: 2020-01-01T12:00:00)
  - **targetSamplingPointTime**: date/time in the format “yyyy-MM-ddThh:mm:ss” [ISO-8601] (example: 2020-01-01T12:01:00)
  - **sourceSamplingPointCandidateIndex**: Node index for the candidate of the source sampling point (see *sampling point*)
  - **targetSamplingPointCandidateIndex**: Node index for the candidate of the target sampling point (see *sampling point*)
  - **sourceSamplingPointCandidateConsideredForwards**: Direction for the candidate of the source sampling point (relative to the order in the street segment)
  - **targetSamplingPointCandidateConsideredForwards**: Direction for the candidate of the target sampling point (relative to the order in the street segment)
  - **routeStartPoint**: geo point of the startpoint, POINT [longitude, latitude]
  - **routeEndPoint**: geo point of the endpoint, POINT [longitude, latitude]
- **CSV properties:**
  - **Separator**: semicolon ( ; )
  - **Floating point precision**: 14

- Column headers in first row: Yes
- Example:

```

osm_ids;route;length;cost;sourceNode;targetNode;
→sourceSamplingPointTime;targetSamplingPointTime;
→sourceSamplingPointCandidateIndex;targetSamplingPointCandidateIndex;
→sourceSamplingPointCandidateConsideredForwards;
→targetSamplingPointCandidateConsideredForwards;routeStartPoint;
→routeEndPoint
23090008;317179950;LINESTRING(10.409835677053 53.271690899804,10.
→4098579 53.2716969,10.4103007 53.2718317,10.410396898775 53.
→271860113945);41.844870470798;0;2862;3108;2018-07-01T19:52:29;2018-
→07-01T19:52:32;0;0;1;1;POINT(10.409835677053 53.271690899804);
→POINT(10.410396898775 53.271860113945)
...

```

representing:

osm_id	route	length	cost	sourceNode	sourceNode	sourceSamplingPointTime	targetSamplingPointTime	sourceSamplingPointCandidateIndex	targetSamplingPointCandidateIndex	sourceSamplingPointCandidateConsideredForwards	targetSamplingPointCandidateConsideredForwards	routeStartPoint	routeEndPoint
23090008	LINESTRING(10.409835677053 53.271690899804,10.4098579 53.2716969,10.4103007 53.2718317,10.410396898775 53.271860113945)			3108		2018-07-01T19:52:29	2018-07-01T19:52:32	0	0	1	1	POINT(10.409835677053 53.271690899804)	POINT(10.410396898775 53.271860113945)
...	...	...	...	...	...	...	...	...	...	...	...	...	...

### 7.8.3 Configuration

- None

## 7.9 CsvSubRouteWriter

This filter writes the internal representation of the calculated *route* with some additional data to a stream in CSV format.

### 7.9.1 Input

- *RouteList*
- *GraphEdgeMap*
- *NodeMap*
- *TimeList*
- *SegmentList*
- *SamplingPointList*

### 7.9.2 Output

A CSV stream with the following format:

- **CSV columns and corresponding types:**
  - **osm\_id**: OSM ID (see origin ID)
  - **route**: List of geo points, LINESTRING [[longitude, latitude], [longitude, latitude], ... ]
  - **length**: [m] floating point
  - **cost**: floating point
  - **sourceNode**: Node identifier for the first point of the segment
  - **targetNode**: Node identifier for the last point of the segment
  - **sourceSamplingPointTime**: date/time in the format “yyyy-MM-ddThh:mm:ss” [ISO-8601] (example: 2020-01-01T12:00:00)
  - **targetSamplingPointTime**: date/time in the format “yyyy-MM-ddThh:mm:ss” [ISO-8601] (example: 2020-01-01T12:01:00)
  - **sourceSamplingPointCandidateIndex**: Node index for the candidate of the source sampling point (see *sampling point*)
  - **targetSamplingPointCandidateIndex**: Node index for the candidate of the target sampling point (see *sampling point*)
  - **sourceSamplingPointCandidateConsideredForwards**: Direction for the candidate of the source sampling point (relative to the order in the street segment)
  - **targetSamplingPointCandidateConsideredForwards**: Direction for the candidate of the target sampling point (relative to the order in the street segment)
- **CSV properties:**
  - **Separator**: semicolon ( ; )
  - **Floating point precision**: 14
  - **Column headers in first row**: Yes
- **Example:**

```

osm_id;route;length;cost;sourceNode;targetNode;sourceSamplingPointTime;
→targetSamplingPointTime;sourceSamplingPointCandidateIndex;
→targetSamplingPointCandidateIndex;
→sourceSamplingPointCandidateConsideredForwards;
→targetSamplingPointCandidateConsideredForwards
23090008;LINESTRING(10.409835677053 53.271690899804,10.4098579 53.
→2716969,10.4103007 53.2718317);34.701985551806;0;2862;3108;2018-07-
→01T19:52:29;2018-07-01T19:52:32;0;0;1;1
317179950;LINESTRING(10.4103007 53.2718317,10.410396898775 53.
→271860113945);7.1428849189925;0;2862;3108;2018-07-01T19:52:29;2018-
→07-01T19:52:32;0;0;1;1
...

```

representing:

osm_id	route	length	cost	sourceNode	targetNode	sourceSamplingPointTime	targetSamplingPointTime	sourceSamplingPointCandidateIndex	targetSamplingPointCandidateIndex	sourceSamplingPointCandidateConsideredForwards	targetSamplingPointCandidateConsideredForwards
23090008	LINESTRING(10.409835677053 53.271690899804,10.4098579 53.			3108		2018-07-01T19:52:29	2018-07-01T19:52:32	0	0	1	1
317179950	LINESTRING(10.4103007 53.2718317,10.410396898775 53.			3108		2018-07-01T19:52:29	2018-07-01T19:52:32	0	0	1	1
...	...	...	...	...	...	...	...	...	...	...	...

### 7.9.3 Configuration

- None

## 7.10 GeoJsonRouteWriter

This filter writes the internal representation of the *route* to a stream in **GeoJson** format.

### 7.10.1 Input

- *RouteList*
- *GraphEdgeMap*
- *NodeMap*
- *TimeList*
- *SegmentList*
- *SamplingPointList*

### 7.10.2 Output

A GeoJson stream with the following content

- **feature:**
  - **LineString:** [[longitude, latitude], [longitude, latitude], ... ]
- **properties per feature:**
  - **Id:** Original ID of the street from which this segment originates
  - **entry\_time:** [ISO-8601] (example: 2020-01-01T12:00:00)
  - **exit\_time:** [ISO-8601] (example: 2020-01-01T12:01:00)
- **Example:**

```
{
    "type": "FeatureCollection",
    "features": [
        {
            "type": "Feature",
            "geometry": {
                "type": "LineString",
                "coordinates": [[52.52, 13.40], [52.405, 13.40], [52.30, 13.407], ↵
                ↵[52.562, 13.405]]
            },
            "properties": {
                "Id": "1234567",
                "entry_time": "2020-07-20T11:11:11",
                "exit_time": "2020-07-20T11:12:12"
            }
        },
        ...
    ]
}
```

### 7.10.3 Configuration

- None

## 7.11 CsvTrackWriter

---

### Disclaimer: Not part of the open source repo

The *CsvTrackWriter* was designed for a specific use case with a specific file layout and may therefore not be usable for the general public, so it is not part of the open source material. Feel free to [contact us](#) if you need support for a CSV writer for tracks.

---

This filter writes the internal representation of the *track* to a stream in **CSV** format.

### 7.11.1 Input

- **mandatory:**
  - *TimeList*
  - *PointList*
  - *VelocityList*
- **optional (mutually exclusive - only one of the following may be given) :**
  - *HeadingList*
  - *PartialHeadingList*

### 7.11.2 Output

A CSV stream in the following format:

- **CSV columns and corresponding types:**
  - **time:** date/time in the format “yyyy-MM-ddThh:mm:ss” [ISO-8601] (example: 2020-01-01T12:00:00)
  - **lat:** floating-point
  - **lon:** floating-point
  - **heading:** [degree] floating-point
  - **velocity:** [m/s] floating-point
- **CSV properties:**
  - **Separator:** comma (,)
  - **Floating point precision:** 14
  - **Column headers in first row:** Yes
- Example Table

```
time,lat,lon,heading,velocity
"2020-01-01T12:00:00",52.52000,13.40500,315.0,12.5
"2020-01-01T12:00:05",52.52001,13.40499,320.0,13.4
...
```

representing:

time	lat	lon	heading	velocity
"2020-01-01T12:00:00"	52.52000	13.40500	315.0	12.5
"2020-01-01T12:00:05"	52.52001	13.40499	320.0	13.4
...	...	...	...	...

### 7.11.3 Configuration

- None

## 7.12 GeoJsonTrackWriter

This filter writes the internal representation of the *track* to a stream in GeoJson format.

### 7.12.1 Input

- **mandatory:**
  - TimeList
  - PointList
  - VelocityList
- **optional (mutually exclusive - only one of the following may be given) :**
  - HeadingList
  - PartialHeadingList

### 7.12.2 Output

A GeoJson stream with the following contents

- **feature:**
  - Point [longitude, latitude]
- **properties per feature:**
  - **time** [ISO-8601] (example: 2020-01-01T12:00:00)
  - **velocity** [m/s]
  - **heading**, if available [degrees]
- **Example:**

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "Point",
        "coordinates": [52.52, 13.40]
      },
      "properties": {
        "time": "2020-07-20T11:11:11",
        "velocity": 50.01,
        "heading": 54.5
      }
    },
    ...
  ]
}
```

### 7.12.3 Configuration

- None

## 7.13 GeoJsonMapWriter

This filter writes the internal representation of the *street map* to a stream in GeoJson format.

### 7.13.1 Input

- *SegmentList*
- *NodePairList*
- *TravelDirectionList*
- *HighwayList*

### 7.13.2 Output

A GeoJson stream with the following contents

- **feature:**
  - **LineString** [[longitude, latitude], [longitude, latitude], ... ]
- **properties per feature:**
  - **Id:** Original ID of the street from which this segment originates
  - **Offset:** Node index at which this segment starts in the original street (see *street segment origin offset*)
  - **SourceNode:** Node identifier for the first point of the segment
  - **TargetNode:** Node identifier for the last point of the segment

- **TravelDirection:** Allowed travel direction; one of [Both, Forwards, Backwards] (*TravelDirection*)
  - **Highway:** Highway type; one of [Motorway, Trunk, Primary, Secondary, Tertiary, MotorwayLink, TrunkLink, PrimaryLink, SecondaryLink, TertiaryLink, Unknown] (*HighwayType*, based on OSM-Highway type)
- **Example:**

```
{  
    "type": "FeatureCollection",  
    "features": [  
        {  
            "type": "Feature",  
            "geometry": {  
                "type": "LineString",  
                "coordinates": [[52.52, 13.40], [52.405, 13.40], [52.30, 13.407],  
                ↪[52.562, 13.405]]  
            },  
            "properties": {  
                "Id": 23315,  
                "Offset": 0,  
                "SourceNode": 4,  
                "TargetNode": 85,  
                "TravelDirection": "Backwards",  
                "Highway": "Primary"  
            }  
        },  
        ...  
    ]  
}
```

### 7.13.3 Configuration

- None

## 7.14 JsonRouteStatisticWriter

Writes statistical data obtained from the *Router* and the internal *track* presentation to a JSON stream.

### 7.14.1 Input

- **mandatory**
  - *RoutingStatistic*
  - *SamplingPointList*
- **optional**
  - *TimeList*

## 7.14.2 Output

A JSON stream which contains a JSON object. The JSON object has the following format:

- **calculated:** Array of JSON objects with the following elements:
  - **source:** *SamplingPoint-JSON*
  - **target:** *SamplingPoint-JSON*
  - **cost:** decimal number, cost of the edge
  - **length:** decimal number, length of the geometry
  - **sub-routes-count:** integer number, amount of subroutes to fulfil the routing
- **visited:** Array of JSON objects with the following elements:
  - **successfull:** boolean value; true, if the routing was successful
  - **source:** *SamplingPoint-JSON*
  - **target:** *SamplingPoint-JSON*

---

**Todo:** typo in successfull -> successful

---

### Def: SamplingPoint-JSON

- **SamplingPoint-JSON:** JSON object representing a sampling point with the following elements:
  - **index:** index of the representing *track* point
  - **time:** [ISO-8601] time value of the corresponding *track* point (example: 2020-01-01T12:00:00)
  - **candidate:** *SamplingPointCandidate-JSON*
  - **candidates-count:** integer number

### Def: SamplingPointCandidate-JSON

- **candidate:** JSON object representing the used candidate with the following elements:
  - **index:** index in the list of candidates of the corresponding sampling point
  - **considered-forwards:** [boolean], determines whether the candidate was, for this route, assumed to be orientated equally to the street map edge.
  - **distance:** [m], distance of the candidate to its sampling point
- Example of the JSON output stream:

```
{
  "calculated": [
    {
      "source": {
        "index": 0,
        "time": "2020-01-01T12:00:00",
        "candidate": {
          "index": 1,
          "considered-forwards": true,
        }
      }
    }
  ]
}
```

(continues on next page)

(continued from previous page)

```
        "distance": 2.73,
    },
    "candidates-count": 2,
},
"target": {
    "index": 249,
    "time": "2020-01-01T12:34:56",
    "candidate": {
        "index": 0,
        "considered-forwards": false,
        "distance": 0.98,
    },
    "candidates-count": 1,
},
"cost": 756.33,
"length": 512.65,
"sub-routes-count": 3,
},
...
],
"visited": [
{
    "successfull": true,
    "source": {
        "index": 0,
        "time": "2020-01-01T12:00:00",
        "candidate": {
            "index": 1,
            "considered-forwards": true,
            "distance": 2.73,
        },
        "candidates-count": 2,
    },
    "target": {
        "index": 249,
        "time": "2020-01-01T12:34:56",
        "candidate": {
            "index": 0,
            "considered-forwards": false,
            "distance": 0.98,
        },
        "candidates-count": 1,
    }
},
...
]
```

### 7.14.3 Configuration

- None



---

CHAPTER  
EIGHT

---

## API DOCUMENTATION

### 8.1 Class Hierarchy

### 8.2 File Hierarchy

### 8.3 Full API

#### 8.3.1 Namespaces

Namespace `anonymous_namespace_FlipMap.h`

Contents

- *Functions*

#### Functions

- *Template Function anonymous\_namespace\_FlipMap.h::flipPair*

Namespace `anonymous_namespace_MakeHashable.h`

Contents

- *Functions*

### Functions

- *Template Function anonymous\_namespace\_MakeHashable.h::hash\_combine(std::size\_t&, const T&, Rest...)*
- *Function anonymous\_namespace\_MakeHashable.h::hash\_combine(std::size\_t&)*

## Namespace anonymous\_namespace\_SamplingPointFinder.cpp

### Contents

- *Functions*
- *Typedefs*

### Functions

- *Function anonymous\_namespace\_SamplingPointFinder.cpp::addStreetIndex*
- *Function anonymous\_namespace\_SamplingPointFinder.cpp::getStreetIndices*
- *Function anonymous\_namespace\_SamplingPointFinder.cpp::headingDifference*

### Typedefs

- *Typedef anonymous\_namespace\_SamplingPointFinder.cpp::StreetIndexGeoindex*
- *Typedef anonymous\_namespace\_SamplingPointFinder.cpp::StreetIndexGeoindexAlgorithm*
- *Typedef anonymous\_namespace\_SamplingPointFinder.cpp::StreetIndexGeoindexGeometry*
- *Typedef anonymous\_namespace\_SamplingPointFinder.cpp::StreetIndexGeoindexValue*

## Namespace AppComponents

### Contents

- *Namespaces*

### Namespaces

- *Namespace AppComponents::Common*
- *Namespace AppComponents::ExampleMatcher*

## Namespace AppComponents::Common

### Contents

- *Namespaces*

### Namespaces

- *Namespace AppComponents::Common::Filter*
- *Namespace AppComponents::Common::Types*

## Namespace AppComponents::Common::Filter

### Contents

- *Namespaces*
- *Classes*

### Namespaces

- *Namespace AppComponents::Common::Filter::anonymous\_namespace\_GeoJsonMapReader.cpp*
- *Namespace AppComponents::Common::Filter::anonymous\_namespace\_GeoJsonMapWriter.cpp*
- *Namespace AppComponents::Common::Filter::anonymous\_namespace\_JsonRouteStatisticWriter.cpp*
- *Namespace AppComponents::Common::Filter::anonymous\_namespace\_OsmMapReader.cpp*
- *Namespace AppComponents::Common::Filter::Osm*
- *Namespace AppComponents::Common::Filter::Routing*

### Classes

- *Class GeoJsonMapReader*
- *Class GeoJsonMapWriter*
- *Class GeoJsonTrackWriter*
- *Class GraphBuilder*
- *Class JsonRouteStatisticWriter*
- *Class OsmMapReader*
- *Class Router*
- *Class SamplingPointFinder*

### Namespace AppComponents::Common::Filter::anonymous\_namespace\_GeoJsonMapReader.cpp

#### Contents

- *Functions*

#### Functions

- *Function AppComponents::Common::Filter::anonymous\_namespace\_GeoJsonMapReader.cpp::toHighway*
- *Function AppComponents::Common::Filter::anonymous\_namespace\_GeoJsonMapReader.cpp::toTravelDirection*

### Namespace AppComponents::Common::Filter::anonymous\_namespace\_GeoJsonMapWriter.cpp

#### Contents

- *Functions*

#### Functions

- *Function AppComponents::Common::Filter::anonymous\_namespace\_GeoJsonMapWriter.cpp::toString(Types::Street::TravelDir const)*
- *Function AppComponents::Common::Filter::anonymous\_namespace\_GeoJsonMapWriter.cpp::toString(Types::Street::Highway const)*

### Namespace AppComponents::Common::Filter::anonymous\_namespace\_JsonRouteStatisticWriter.cpp

#### Contents

- *Functions*

#### Functions

- *Function AppComponents::Common::Filter::anonymous\_namespace\_JsonRouteStatisticWriter.cpp::toJson*

## Namespace AppComponents::Common::Filter::anonymous\_namespace\_OsmMapReader.cpp

### Contents

- *Classes*
- *Functions*
- *Typedefs*

### Classes

- *Struct Candidate*
- *Struct OsmLineCandidate*
- *Struct OsmPointCandidate*
- *Struct PointLocation*
- *Struct UniquePoint*

### Functions

- *Function AppComponents::Common::Filter::anonymous\_namespace\_OsmMapReader.cpp::getCandidates*
- *Function AppComponents::Common::Filter::anonymous\_namespace\_OsmMapReader.cpp::getUniquePoint*
- *Function AppComponents::Common::Filter::anonymous\_namespace\_OsmMapReader.cpp::processCandidates*

### Typedefs

- *Typedef AppComponents::Common::Filter::anonymous\_namespace\_OsmMapReader.cpp::PointGeoindex*
- *Typedef AppComponents::Common::Filter::anonymous\_namespace\_OsmMapReader.cpp::PointGeoindexAlgorithm*
- *Typedef AppComponents::Common::Filter::anonymous\_namespace\_OsmMapReader.cpp::PointGeoindexValue*

## Namespace AppComponents::Common::Filter::Osm

### Contents

- *Functions*

## Functions

- *Function AppComponents::Common::Filter::Osm::toHighway*
- *Function AppComponents::Common::Filter::Osm::toHighwaySelectionSql*
- *Function AppComponents::Common::Filter::Osm::toOsmString*
- *Function AppComponents::Common::Filter::Osm::toTravelDirection*

## Namespace AppComponents::Common::Filter::Routing

### Contents

- *Namespaces*
- *Classes*
- *Enums*
- *Functions*
- *Typedefs*

### Namespaces

- *Namespace AppComponents::Common::Filter::Routing::anonymous\_namespace\_Comparators.cpp*
- *Namespace AppComponents::Common::Filter::Routing::anonymous\_namespace\_SamplingPointRouter.cpp*
- *Namespace AppComponents::Common::Filter::Routing::Generic*

### Classes

- *Struct BacktrackRouter::Configuration*
- *Struct BacktrackRouter::Session*
- *Struct BestRouteComparator*
- *Struct BestSimilarRouteComparator*
- *Struct DirectedCandidateRouter::Configuration*
- *Struct SamplingPointRouter::Configuration*
- *Struct SamplingPointsSelection*
- *Struct SkipRouter::Configuration*
- *Struct SkipRouter::Session*
- *Class BacktrackRouter*
- *Class DirectedCandidateRouter*
- *Class PiecewiseRouter*
- *Class SamplingPointRouter*

- *Class SkipRouter*

## Enums

- *Enum RouteClusterPreference*
- *Enum RouteResult*

## Functions

- *Function AppComponents::Common::Filter::Routing::attachToPreviousRoute*
- *Function AppComponents::Common::Filter::Routing::calcApproximateDistanceBetweenSamplingPoints*
- *Function AppComponents::Common::Filter::Routing::checkMaxAngularDeviation*
- *Function AppComponents::Common::Filter::Routing::findPreviousConnectedRoute*
- *Function AppComponents::Common::Filter::Routing::geoDistance*
- *Function AppComponents::Common::Filter::Routing::isSelfIntersectingRoute*
- *Function AppComponents::Common::Filter::Routing::isSimilar*
- *Function AppComponents::Common::Filter::Routing::routeResultToString*

## Typedefs

- *Typedef AppComponents::Common::Filter::Routing::SamplingPointSkipStrategy*

## Namespace AppComponents::Common::Filter::Routing::anonymous\_namespace\_Comparators.cpp

### Contents

- *Functions*

## Functions

- *Function AppComponents::Common::Filter::Routing::anonymous\_namespace\_Comparators.cpp::isContained*

## Namespace AppComponents::Common::Filter::Routing::anonymous\_namespace\_SamplingPointRouter.cpp

### Contents

- *Functions*
- *Typedefs*

## Functions

- *Function AppComponents::Common::Filter::Routing::anonymous\_namespace\_SamplingPointRouter.cpp::cluster*
- *Function AppComponents::Common::Filter::Routing::anonymous\_namespace\_SamplingPointRouter.cpp::getBestRoute*
- *Function AppComponents::Common::Filter::Routing::anonymous\_namespace\_SamplingPointRouter.cpp::routeCached*
- *Function AppComponents::Common::Filter::Routing::anonymous\_namespace\_SamplingPointRouter.cpp::selectCandidates*

## Typedefs

- *Typedef AppComponents::Common::Filter::Routing::anonymous\_namespace\_SamplingPointRouter.cpp::ClusteredRouteMatrix*
- *Typedef AppComponents::Common::Filter::Routing::anonymous\_namespace\_SamplingPointRouter.cpp::SamplingPointCandidate*

## Namespace AppComponents::Common::Filter::Routing::Generic

### Contents

- *Classes*
- *Functions*

## Classes

- *Class SelectiveSkipper*
- *Class Skipper*

## Functions

- *Function AppComponents::Common::Filter::Routing::Generic::findNextAllowed*
- *Function AppComponents::Common::Filter::Routing::Generic::findPreviousAllowed*

## Namespace AppComponents::Common::Types

### Contents

- *Namespaces*

## Namespaces

- *Namespace AppComponents::Common::Types::Graph*
- *Namespace AppComponents::Common::Types::Routing*
- *Namespace AppComponents::Common::Types::Street*
- *Namespace AppComponents::Common::Types::Track*

### Namespace AppComponents::Common::Types::Graph

#### Contents

- *Classes*
- *Typedefs*

#### Classes

- *Struct GraphTriplePair*
- *Struct StreetEdge*

#### Typedefs

- *Typedef AppComponents::Common::Types::Graph::Graph*
- *Typedef AppComponents::Common::Types::Graph::GraphEdgeMap*
- *Typedef AppComponents::Common::Types::Graph::GraphTriple*
- *Typedef AppComponents::Common::Types::Graph::LemonDigraph*
- *Typedef AppComponents::Common::Types::Graph::NodeMap*
- *Typedef AppComponents::Common::Types::Graph::StreetIndexMap*

### Namespace AppComponents::Common::Types::Routing

#### Contents

- *Classes*
- *Typedefs*

## Classes

- *Struct CalculatedRouteStatistic*
- *Struct Route*
- *Struct RouteNode*
- *Struct RoutingStatistic*
- *Struct SamplingPoint*
- *Struct SamplingPointCandidate*
- *Struct SamplingPointCandidateSelection*
- *Struct SamplingPointSelection*
- *Struct SubRoute*

## Typedefs

- *Typedef AppComponents::Common::Types::Routing::Edge*
- *Typedef AppComponents::Common::Types::Routing::Node*
- *Typedef AppComponents::Common::Types::Routing::RouteList*
- *Typedef AppComponents::Common::Types::Routing::SamplingPointList*

## Namespace AppComponents::Common::Types::Street

### Contents

- *Classes*
- *Enums*
- *Typedefs*

## Classes

- *Struct NodePairList*
- *Struct Segment*

## Enums

- *Enum HighwayType*
- *Enum TravelDirection*

## Typedefs

- *Typedef AppComponents::Common::Types::Street::Highway*
- *Typedef AppComponents::Common::Types::Street::HighwayList*
- *Typedef AppComponents::Common::Types::Street::NodePair*
- *Typedef AppComponents::Common::Types::Street::SegmentList*
- *Typedef AppComponents::Common::Types::Street::TravelDirectionList*

## Namespace AppComponents::Common::Types::Track

### Contents

- *Classes*
- *Typedefs*

## Classes

- *Struct AltitudeList*
- *Struct HeadingList*
- *Struct PointList*
- *Struct TimeList*
- *Struct VelocityList*

## Typedefs

- *Typedef AppComponents::Common::Types::Track::Altitude*
- *Typedef AppComponents::Common::Types::Track::Heading*
- *Typedef AppComponents::Common::Types::Track::Point*
- *Typedef AppComponents::Common::Types::Track::Time*
- *Typedef AppComponents::Common::Types::Track::Velocity*

**Namespace AppComponents::ExampleMatcher****Contents**

- *Namespaces*

**Namespaces**

- *Namespace AppComponents::ExampleMatcher::Filter*

**Namespace AppComponents::ExampleMatcher::Filter****Contents**

- *Classes*

**Classes**

- *Class CsvTrackReader*
- *Class JsonTrackReader*

**Namespace Core****Contents**

- *Namespaces*

**Namespaces**

- *Namespace Core::Common*
- *Namespace Core::Graph*

**Namespace Core::Common****Contents**

- *Namespaces*

## Namespaces

- *Namespace Core::Common::Geometry*
- *Namespace Core::Common::Postgres*
- *Namespace Core::Common::Time*

## Namespace Core::Common::Geometry

### Contents

- *Classes*
- *Functions*
- *Typedefs*
- *Variables*

### Classes

- *Struct LineStringProjectionResult*
- *Struct Point*
- *Struct Point::Latitude*
- *Struct Point::Longitude*

### Functions

- *Function Core::Common::Geometry::absHeadingDiff*
- *Function Core::Common::Geometry::angleBetweenSegments*
- *Function Core::Common::Geometry::buffer*
- *Template Function Core::Common::Geometry::degree*
- *Function Core::Common::Geometry::flattened\_simple*
- *Function Core::Common::Geometry::geoDistance(Point const&, Point const&)*
- *Function Core::Common::Geometry::geoDistance(Point const&, Segment const&)*
- *Template Function Core::Common::Geometry::geoDistance(G1 const&, G2 const&)*
- *Template Function Core::Common::Geometry::geoLength*
- *Function Core::Common::Geometry::heading(Point const&, Point const&)*
- *Function Core::Common::Geometry::heading(Segment const&)*
- *Function Core::Common::Geometry::headingDiff*
- *Function Core::Common::Geometry::normalizeAngle*
- *Function Core::Common::Geometry::operator""\_lat*

- *Function Core::Common::Geometry::operator""\_lon*
- *Function Core::Common::Geometry::operator==*
- *Function Core::Common::Geometry::project*
- *Function Core::Common::Geometry::projectOntoLineString*
- *Template Function Core::Common::Geometry::rad*
- *Function Core::Common::Geometry::relativeDistanceAlongLineString*
- *Function Core::Common::Geometry::reversedHeading*
- *Function Core::Common::Geometry::reverseHaversine*
- *Function Core::Common::Geometry::snap*
- *Function Core::Common::Geometry::toGeoJson(LineString const&)*
- *Function Core::Common::Geometry::toGeoJson(Point const&)*
- *Function Core::Common::Geometry::toLineString(std::string const&)*
- *Function Core::Common::Geometry::toLineString(nlohmann::json const&)*
- *Function Core::Common::Geometry::toPoint*
- *Function Core::Common::Geometry::toWkt(Point const&)*
- *Function Core::Common::Geometry::toWkt(std::vector<Point> const&)*
- *Function Core::Common::Geometry::toWkt(std::vector<std::vector<Point>> const&)*
- *Function Core::Common::Geometry::trimmed*
- *Template Function Core::Common::Geometry::within*

## Typedefs

- *Typedef Core::Common::Geometry::Box*
- *Typedef Core::Common::Geometry::CoordinateSystem*
- *Typedef Core::Common::Geometry::LineString*
- *Typedef Core::Common::Geometry::Polygon*
- *Typedef Core::Common::Geometry::Segment*
- *Typedef Core::Common::Geometry::ValueType*

## Variables

- *Variable Core::Common::Geometry::equatorRadiusKiloMeter*
- *Variable Core::Common::Geometry::equatorRadiusMeter*
- *Variable Core::Common::Geometry::pi*
- *Variable Core::Common::Geometry::pover180*

## Namespace Core::Common::Postgres

### Contents

- *Classes*
- *Functions*

### Classes

- *Class Connection*

### Functions

- *Template Function Core::Common::Postgres::getOptional(C const&)*
- *Template Function Core::Common::Postgres::getOptional(C const&, T)*

## Namespace Core::Common::Time

### Contents

- *Functions*

### Functions

- *Function Core::Common::Time::fromIsoString*
- *Function Core::Common::Time::fromIsoZString*
- *Function Core::Common::Time::fromString*
- *Function Core::Common::Time::toIsoString*
- *Function Core::Common::Time::toIsoZString*
- *Function Core::Common::Time::toString*

## Namespace Core::Graph

### Contents

- *Namespaces*
- *Classes*

## Namespaces

- *Namespace Core::Graph::Routing*

## Classes

- *Struct Edge*
- *Struct Node*
- *Class Graph*
- *Class LemonDigraph*

## Namespace Core::Graph::Routing

### Contents

- *Classes*
- *Functions*
- *Typedefs*

## Classes

- *Class Dijkstra*
- *Class Dijkstra::PathNode*
- *Class PathView*
- *Class PathViewImpl*
- *Class PathViewIterator*
- *Class RoutingAlgorithm*

## Functions

- *Function Core::Graph::Routing::operator<<*
- *Function Core::Graph::Routing::operator>*

## TypeDefs

- *Typedef Core::Graph::Routing::CostFunction*
- *Typedef Core::Graph::Routing::FilterFunction*

## Namespace Generic

### Contents

- *Namespaces*
- *Classes*
- *Functions*

### Namespaces

- *Namespace Generic::String*

### Classes

- *Class Progress*

### Functions

- *Template Function Generic::flipMap*

## Namespace Generic::String

### Contents

- *Functions*

### Functions

- *Function Generic::String::ltrim*
- *Function Generic::String::ltrimmed*
- *Function Generic::String::readNextRow*
- *Function Generic::String::rtrim*
- *Function Generic::String::rtrimmed*
- *Template Function Generic::String::split*
- *Function Generic::String::trim*

- *Function Generic::String::trimmed*

**Namespace std****Contents**

- *Classes*

**Classes**

- *Template Struct hash< Core::Graph::Edge >*
- *Template Struct hash< Core::Graph::Node >*

### 8.3.2 Classes and Structs

**Struct Candidate**

- Defined in file\_src\_AppComponents\_Common\_Filter\_OsmMapReader.cpp

**Struct Documentation****struct****Public Members**

```
OsmLineCandidate AppComponents::Common::Filter::anonymous_namespace_OsmMapReader.cpp::  
std::vector<size_t> AppComponents::Common::Filter::anonymous_namespace_OsmMapReader.cpp::
```

**Struct OsmLineCandidate**

- Defined in file\_src\_AppComponents\_Common\_Filter\_OsmMapReader.cpp

**Struct Documentation****struct**

### Public Members

```
Types::Street::Segment AppComponents::Common::Filter::anonymous_namespace_OsmMapReader
Types::Street::TravelDirection AppComponents::Common::Filter::anonymous_namespace_OsmMapReader
Types::Street::Highway AppComponents::Common::Filter::anonymous_namespace_OsmMapReader
```

### Struct OsmPointCandidate

- Defined in file\_src\_AppComponents\_Common\_Filter\_OsmMapReader.cpp

### Struct Documentation

**struct**

### Public Members

```
size_t AppComponents::Common::Filter::anonymous_namespace_OsmMapReader.cpp::OsmPointCandidate
Core::Common::Geometry::Point AppComponents::Common::Filter::anonymous_namespace_OsmMapReader
```

### Struct PointLocation

- Defined in file\_src\_AppComponents\_Common\_Filter\_OsmMapReader.cpp

### Struct Documentation

**struct**

### Public Members

```
size_t AppComponents::Common::Filter::anonymous_namespace_OsmMapReader.cpp::PointLocation
size_t AppComponents::Common::Filter::anonymous_namespace_OsmMapReader.cpp::PointLocation
```

### Struct UniquePoint

- Defined in file\_src\_AppComponents\_Common\_Filter\_OsmMapReader.cpp

### Struct Documentation

**struct**

**Public Members**

```
size_t AppComponents::Common::Filter::anonymous_namespace_OsmMapReader.cpp::UniquePoint  
std::vector<PointLocation> AppComponents::Common::Filter::anonymous_namespace_OsmMapRe
```

**Struct BacktrackRouter::Configuration**

- Defined in file\_src\_AppComponents\_Common\_Filter\_Routing\_BacktrackRouter.h

**Nested Relationships**

This struct is a nested type of [Class BacktrackRouter](#).

**Struct Documentation**

```
struct Configuration
```

**Public Members**

```
double maxBacktrackingDistance
```

**Struct BacktrackRouter::Session**

- Defined in file\_src\_AppComponents\_Common\_Filter\_Routing\_BacktrackRouter.cpp

**Nested Relationships**

This struct is a nested type of [Class BacktrackRouter](#).

**Struct Documentation**

```
struct Session
```

**Public Functions**

```
Session(size_t const sourceSamplingPointIndexMinimum_, size_t const targetSampling-  
PointIndexGoal_, std::unordered_set<size_t> const &skippedSamplingPoints_,  
Types::Routing::RouteList &routeList_, SamplingPointRouter::RouteMap &routeMap_,  
Types::Routing::RoutingStatistic &routingStatistic_)
```

## Public Members

```
size_t const sourceSamplingPointIndexMinimum
size_t const targetSamplingPointIndexGoal
std::unordered_set<size_t> const &skippedSamplingPoints
Types::Routing::RouteList &routeList
Types::Routing::RoutingStatistic &routingStatistic
SamplingPointRouter::RouteMap &routeMap
SamplingPointRouter::VisitedRouteSet visitedRouteSet
```

## Struct BestRouteComparator

- Defined in file\_src\_AppComponents\_Common\_Filter\_Routing\_Comparators.h

### Struct Documentation

```
struct BestRouteComparator
```

#### Public Functions

```
BestRouteComparator (RouteClusterPreference routeClusterPreference)
```

```
bool operator () (std::shared_ptr<Types::Routing::Route> a, std::shared_ptr<Types::Routing::Route> b) const
```

#### Private Members

```
RouteClusterPreference routeClusterPreference_
```

## Struct BestSimilarRouteComparator

- Defined in file\_src\_AppComponents\_Common\_Filter\_Routing\_Comparators.h

### Struct Documentation

```
struct BestSimilarRouteComparator
```

## Public Functions

```
bool operator() (std::shared_ptr<Types::Routing::Route> a, const std::shared_ptr<Types::Routing::Route> b) const
```

## Struct DirectedCandidateRouter::Configuration

- Defined in file\_src\_AppComponents\_Common\_Filter\_Routing\_DirectedCandidateRouter.h

### Nested Relationships

This struct is a nested type of [Class DirectedCandidateRouter](#).

### Struct Documentation

```
struct Configuration
```

#### Public Members

```
double maxVelocityDifference  
bool allowSelfIntersection  
double maxAngularDeviation  
double accountTurningCircleLength
```

## Struct SamplingPointRouter::Configuration

- Defined in file\_src\_AppComponents\_Common\_Filter\_Routing\_SamplingPointRouter.h

### Nested Relationships

This struct is a nested type of [Class SamplingPointRouter](#).

### Struct Documentation

```
struct Configuration
```

#### Public Members

```
double maxClusteredRoutesLengthDifference  
RouteClusterPreference routeClusterPreference
```

## Struct SamplingPointsSelection

- Defined in file\_src\_AppComponents\_Common\_Filter\_Routing\_Types.h

### Struct Documentation

```
struct SamplingPointsSelection
```

#### Public Functions

```
bool operator==(SamplingPointsSelection const &other) const
```

#### Public Members

```
Types::Routing::SamplingPointSelection source
```

```
Types::Routing::SamplingPointSelection target
```

## Struct SkipRouter::Configuration

- Defined in file\_src\_AppComponents\_Common\_Filter\_Routing\_SkipRouter.h

### Nested Relationships

This struct is a nested type of [Class SkipRouter](#).

### Struct Documentation

```
struct Configuration
```

#### Public Members

```
double maxBacktrackingDistance
```

```
Generic::Skipper::Strategy skipStrategy
```

## Struct SkipRouter::Session

- Defined in file\_src\_AppComponents\_Common\_Filter\_Routing\_SkipRouter.cpp

## Nested Relationships

This struct is a nested type of [Class SkipRouter](#).

### Struct Documentation

```
struct Session
```

#### Public Functions

```
Session(size_t const sourceSamplingPointIndexStart_, size_t const targetSamplingPointIndexGoal_, Types::Routing::RouteList &routeList_, Types::Routing::RoutingStatistic &routingStatistic_)
```

#### Public Members

```
size_t const sourceSamplingPointIndexStart  
size_t const targetSamplingPointIndexGoal  
Types::Routing::RouteList &routeList  
Types::Routing::RoutingStatistic &routingStatistic  
SamplingPointRouter::RouteMap routeMap  
std::unordered_set<size_t> skippedSamplingPoints
```

## Struct GraphTriplePair

- Defined in file\_src\_AppComponents\_Common\_Types\_Graph\_EdgeMap.h

### Struct Documentation

```
struct GraphTriplePair
```

#### Public Members

```
std::optional<GraphTriple> forwards  
std::optional<GraphTriple> backwards
```

## Struct StreetEdge

- Defined in file\_src\_AppComponents\_Common\_Types\_Graph\_EdgeMap.h

### Struct Documentation

```
struct StreetEdge
```

#### Public Members

```
size_t streetIndex  
bool forwards
```

## Struct CalculatedRouteStatistic

- Defined in file\_src\_AppComponents\_Common\_Types\_Routing\_Statistic.h

### Struct Documentation

```
struct CalculatedRouteStatistic
```

#### Public Members

```
double cost  
double length  
size_t subRoutesCount
```

## Struct Route

- Defined in file\_src\_AppComponents\_Common\_Types\_Routing\_Edge.h

### Struct Documentation

```
struct Route
```

#### Public Functions

```
Route (RouteNode const &source, RouteNode const &target, std::vector<SubRoute> const &sub-  
Routes)  
double cost () const  
double length () const  
size_t numPoints () const
```

**Public Members**

*RouteNode* **source**  
*RouteNode* **target**  
std::vector<*SubRoute*> **subRoutes**

**Private Members**

std::optional<double> **cost\_**  
std::optional<double> **length\_**  
std::optional<size\_t> **numPoints\_**

**Struct RouteNode**

- Defined in file\_src\_AppComponents\_Common\_Types\_Routing\_Edge.h

**Struct Documentation**

```
struct RouteNode
```

**Public Members**

*Node* **node**  
*SamplingPointSelection* **samplingPoint**

**Struct RoutingStatistic**

- Defined in file\_src\_AppComponents\_Common\_Types\_Routing\_Statistic.h

**Struct Documentation**

```
struct RoutingStatistic
```

Holds additional routing data per sampling point route data.

**Public Members**

std::unordered\_map<Filter::Routing::*SamplingPointsSelection*, *CalculatedRouteStatistic*> **calculated**  
std::vector<std::pair<Filter::Routing::*SamplingPointsSelection*, bool>> **visited**

## Struct SamplingPoint

- Defined in file\_src\_AppComponents\_Common\_Types\_Routing\_SamplingPoint.h

### Struct Documentation

```
struct SamplingPoint
```

Represents a projected track point.

#### Public Members

```
size_t trackIndex
```

```
std::vector<SamplingPointCandidate> candidates
```

## Struct SamplingPointCandidate

- Defined in file\_src\_AppComponents\_Common\_Types\_Routing\_SamplingPoint.h

### Struct Documentation

```
struct SamplingPointCandidate
```

#### Public Types

```
using Point = Core::Common::Geometry::Point
```

#### Public Members

```
size_t streetIndex
```

```
size_t streetSegmentIndex
```

```
Point streetSegmentProjectedPoint
```

```
double streetSegmentProjectedPointNormLength
```

```
double streetSegmentDistance
```

```
double streetSegmentHeading
```

```
double streetSegmentHeadingDifference
```

```
Street::TravelDirection streetSegmentTravelDirection
```

**Struct SamplingPointCandidateSelection**

- Defined in file\_src\_AppComponents\_Common\_Types\_Routing\_Edge.h

**Struct Documentation**

```
struct SamplingPointCandidateSelection
```

**Public Functions**

```
bool operator==(SamplingPointCandidateSelection const &other) const
```

**Public Members**

```
size_t index  
bool consideredForwards
```

**Struct SamplingPointSelection**

- Defined in file\_src\_AppComponents\_Common\_Types\_Routing\_Edge.h

**Struct Documentation**

```
struct SamplingPointSelection
```

**Public Functions**

```
bool operator==(SamplingPointSelection const &other) const
```

**Public Members**

```
size_t index  
SamplingPointCandidateSelection candidate
```

**Struct SubRoute**

- Defined in file\_src\_AppComponents\_Common\_Types\_Routing\_Edge.h

## Struct Documentation

```
struct SubRoute
```

### Public Members

```
Edge edge  
double cost  
Core::Common::Geometry::LineString route  
double length
```

## Struct NodePairList

- Defined in file\_src\_AppComponents\_Common\_Types\_Street\_NodePair.h

### Inheritance Relationships

#### Base Type

- public std::vector< NodePair >

## Struct Documentation

```
struct NodePairList : public std::vector<NodePair>
```

## Struct Segment

- Defined in file\_src\_AppComponents\_Common\_Types\_Street\_Segment.h

## Struct Documentation

```
struct Segment
```

Represents a street segment.

### Public Members

```
size_t originId  
size_t originOffset = {0}  
In case the original segment was splitted.  
Core::Common::Geometry::LineString geometry
```

### Struct AltitudeList

- Defined in file\_src\_AppComponents\_Common\_Types\_Track\_Altitude.h

### Inheritance Relationships

#### Base Type

- public std::vector< Altitude >

### Struct Documentation

```
struct AltitudeList : public std::vector<Altitude>
```

### Struct HeadingList

- Defined in file\_src\_AppComponents\_Common\_Types\_Track\_Heading.h

### Inheritance Relationships

#### Base Type

- public std::vector< Heading >

### Struct Documentation

```
struct HeadingList : public std::vector<Heading>
```

### Struct PointList

- Defined in file\_src\_AppComponents\_Common\_Types\_Track\_Point.h

### Inheritance Relationships

#### Base Type

- public std::vector< Point >

## Struct Documentation

```
struct PointList : public std::vector<Point>
```

## Struct TimeList

- Defined in file\_src\_AppComponents\_Common\_Types\_Track\_Time.h

## Inheritance Relationships

### Base Type

- public std::vector< Time >

## Struct Documentation

```
struct TimeList : public std::vector<Time>
```

## Struct VelocityList

- Defined in file\_src\_AppComponents\_Common\_Types\_Track\_Velocity.h

## Inheritance Relationships

### Base Type

- public std::vector< Velocity >

## Struct Documentation

```
struct VelocityList : public std::vector<Velocity>
```

## Struct LineStringProjectionResult

- Defined in file\_src\_Core\_Common\_Geometry\_Helper.h

## Struct Documentation

```
struct LineStringProjectionResult
```

## Public Members

`Point projectionPoint`  
double `distanceToSegment`  
in meters.  
double `distanceAlongLineString`  
in meters.  
double `segmentHeading`  
azimuth in degrees.  
bool `offLineString`  
has no normal projection onto segment.

## Struct Point

- Defined in file\_src\_Core\_Common\_Geometry\_Types.h

## Nested Relationships

### Nested Types

- Struct Point::Latitude*
- Struct Point::Longitude*

## Struct Documentation

`struct Point`

### Public Functions

`Point()`  
`Point (Longitude lon, Latitude lat)`  
`Point (Latitude lat, Longitude lon)`  
`Point (Point const&)`  
`Point (Point&&)`  
`Point &operator= (Point const&)`  
`Point &operator= (Point&&)`  
`~Point()`  
`ValueType lat () const`  
`ValueType lon () const`  
`Point &setLat (ValueType latitude)`

```
Point &setLon (ValueType longitude)
std::string to_string () const
```

### Private Members

```
ValueType longitude_
ValueType latitude_
struct Latitude
```

### Public Functions

```
Latitude (ValueType value)
Point::Latitude operator- () const
operator ValueType () const
```

### Private Members

```
ValueType value
struct Longitude
```

### Public Functions

```
Longitude (ValueType value)
Point::Longitude operator- () const
operator ValueType () const
```

### Private Members

```
ValueType value
```

## Struct Point::Latitude

- Defined in file\_src\_Core\_Common\_Geometry\_Types.h

## Nested Relationships

This struct is a nested type of [Struct Point](#).

### Struct Documentation

```
struct Latitude
```

#### Public Functions

```
Latitude (ValueType value)
```

```
Point::Latitude operator- () const
```

```
operator ValueType () const
```

#### Private Members

```
ValueType value
```

## Struct Point::Longitude

- Defined in file\_src\_Core\_Common\_Geometry\_Types.h

## Nested Relationships

This struct is a nested type of [Struct Point](#).

### Struct Documentation

```
struct Longitude
```

#### Public Functions

```
Longitude (ValueType value)
```

```
Point::Longitude operator- () const
```

```
operator ValueType () const
```

### Private Members

*ValueType* **value**

### Struct Edge

- Defined in file\_src\_Core\_Graph\_Graph.h

### Struct Documentation

**struct Edge**

#### Public Functions

```
bool operator== (Edge const &other) const
size_t id () const
```

#### Private Functions

**Edge** (size\_t *id*)

### Private Members

size\_t **id\_**

### Friends

```
friend Core::Graph::Edge::Graph
Represents the street graph.
```

### Struct Node

- Defined in file\_src\_Core\_Graph\_Graph.h

### Struct Documentation

**struct Node**

**Public Functions**

```
bool operator==(Node const &other) const  
size_t id() const
```

**Private Functions**

```
Node(size_t id)
```

**Private Members**

```
size_t id_
```

**Friends**

```
friend Core::Graph::Node::Graph  
Represents the street graph.
```

**Template Struct hash< Core::Graph::Edge >**

- Defined in file\_src\_Core\_Graph\_Graph.h

**Struct Documentation**

```
template<>  
struct hash<Core::Graph::Edge>
```

**Public Functions**

```
size_t operator()(Core::Graph::Edge const &edge) const
```

**Template Struct hash< Core::Graph::Node >**

- Defined in file\_src\_Core\_Graph\_Graph.h

**Struct Documentation**

```
template<>  
struct hash<Core::Graph::Node>
```

## Public Functions

```
size_t operator () (Core::Graph::Node const &node) const
```

## Class GeoJsonMapReader

- Defined in file\_src\_AppComponents\_Common\_Filter\_GeoJsonMapReader.h

## Inheritance Relationships

### Base Type

- public Filter

## Class Documentation

```
class GeoJsonMapReader : public Filter
```

## Public Functions

```
GeoJsonMapReader (std::istream &input)
```

```
bool operator () (Types::Street::SegmentList      &segmentList,          Types::Street::NodePairList
                  &nodePairList,           Types::Street::TravelDirectionList &travelDirectionList,
                  Types::Street::HighwayList &highwayList)
```

## Private Members

```
std::istream &input_
```

## Class GeoJsonMapWriter

- Defined in file\_src\_AppComponents\_Common\_Filter\_GeoJsonMapWriter.h

## Inheritance Relationships

### Base Type

- public Filter

## Class Documentation

```
class GeoJsonMapWriter : public Filter
```

### Public Functions

```
GeoJsonMapWriter (std::ostream &output)
```

```
bool operator () (Types::Street::SegmentList const &segmentList, Types::Street::NodePairList  
const &nodePairList, Types::Street::TravelDirectionList const &travelDirectionList,  
Types::Street::HighwayList const &highwayList)
```

### Private Members

```
std::ostream &output_
```

## Class GeoJsonTrackWriter

- Defined in file\_src\_AppComponents\_Common\_Filter\_GeoJsonTrackWriter.h

### Inheritance Relationships

#### Base Type

- public Filter

## Class Documentation

```
class GeoJsonTrackWriter : public Filter
```

### Public Functions

```
GeoJsonTrackWriter (std::ostream &output)
```

```
bool operator () (Types::Track::TimeList const &timeList, Types::Track::PointList  
const &pointList, Types::Track::HeadingList const &headingList,  
Types::Track::VelocityList const &velocityList)
```

### Private Members

```
std::ostream &output_
```

## Class GraphBuilder

- Defined in file\_src\_AppComponents\_Common\_Filter\_GraphBuilder.h

### Inheritance Relationships

#### Base Type

- public Filter

### Class Documentation

```
class GraphBuilder : public Filter
```

#### Public Functions

```
GraphBuilder()

bool operator() (Types::Street::NodePairList           const      &nodePairList,
                  Types::Street::TravelDirectionList    const      &travelDirectionList,
                  Types::Graph::Graph &graph,          Types::Graph::GraphEdgeMap &graphEdgeMap,
                  Types::Graph::StreetIndexMap &streetIndexMap,   Types::Graph::NodeMap
&nodeMap)
```

## Class JsonRouteStatisticWriter

- Defined in file\_src\_AppComponents\_Common\_Filter\_JsonRouteStatisticWriter.h

### Inheritance Relationships

#### Base Type

- public Filter

### Class Documentation

```
class JsonRouteStatisticWriter : public Filter
```

## Public Functions

```
JsonRouteStatisticWriter (std::ostream &output)
bool operator () (Types::Routing::RoutingStatistic      const      &routingStatistic,
                  Types::Routing::SamplingPointList const      &samplingPointList,
                  Types::Track::TimeList const &timeList)
```

## Private Members

```
std::ostream &output_
```

## Class OsmMapReader

- Defined in file\_src\_AppComponents\_Common\_Filter\_OsmMapReader.h

## Inheritance Relationships

### Base Type

- public Filter

## Class Documentation

```
class OsmMapReader : public Filter
Reads OpenStreetMap data from a PostGIS database.
```

## Public Functions

```
OsmMapReader (Core::Common::Postgres::Connection      &connection,
               std::unordered_set<Types::Street::HighwayType> const      &highwaySelection,
               double fetchCorridor, bool useSingleSearchCircle)

bool operator () (Types::Track::PointList      const      &pointList,      Types::Street::SegmentList
                  &segmentList,          Types::Street::NodePairList      &nodePairList,
                  Types::Street::TravelDirectionList &travelDirectionList, Types::Street::HighwayList
                  &highwayList)
```

## Private Members

```
Core::Common::Postgres::Connection &connection_
std::unordered_set<Types::Street::HighwayType> const highwaySelection_
double const fetchCorridor_
bool const useSingleSearchCircle_
```

## Class Router

- Defined in file\_src\_AppComponents\_Common\_Filter\_Router.h

## Inheritance Relationships

### Base Type

- public Filter

## Class Documentation

```
class Router : public Filter
```

### Public Functions

```
Router(double maxVelocityDifference, bool allowSelfIntersection, double maxAngularDeviation,
       double accountTurningCircleLength, double maxSamplingPointSkippingDistance, Routing::SamplingPointSkipStrategy samplingPointSkipStrategy, double maxCandidateBacktrackingDistance, double maxClusteredRoutesLengthDifference, Routing::RouteClusterPreference routeClusterPreference)

bool operator()(Types::Routing::SamplingPointList const &samplingPointList,
                 Types::Track::TimeList const &timeList, Types::Track::VelocityList
                 const &velocityList, Types::Street::SegmentList const &segmentList,
                 Types::Graph::Graph const &graph, Types::Graph::GraphEdgeMap const
                 &graphEdgeMap, Types::Graph::StreetIndexMap const &streetIndexMap,
                 Types::Routing::RouteList &routeList, Types::Routing::RoutingStatistic &routingStatistic)
```

### Private Members

```
double const maxVelocityDifference_
bool const allowSelfIntersection_
double const maxAngularDeviation_
double const accountTurningCircleLength_
double const maxSamplingPointSkippingDistance_
Routing::SamplingPointSkipStrategy const samplingPointSkipStrategy_
double const maxCandidateBacktrackingDistance_
double const maxClusteredRoutesLengthDifference_
Routing::RouteClusterPreference const routeClusterPreference_
```

## Class BacktrackRouter

- Defined in file\_src\_AppComponents\_Common\_Filter\_Routing\_BacktrackRouter.h

### Nested Relationships

#### Nested Types

- Struct BacktrackRouter::Configuration*
- Struct BacktrackRouter::Session*

### Class Documentation

```
class BacktrackRouter
```

#### Public Functions

```
BacktrackRouter(SamplingPointRouter const &router, Configuration const configuration, Types::Routing::SamplingPointList const &samplingPointList, Types::Track::TimeList const &timeList)  
void operator()(size_t sourceSamplingPointIndexMinimum, size_t sourceSamplingPointIndexStart, size_t targetSamplingPointIndexGoal, std::unordered_set<size_t> const &skippedSamplingPoints, SamplingPointRouter::RouteMap &routeMap, Types::Routing::RouteList &routeList, Types::Routing::RoutingStatistic &routingStatistic) const
```

#### Private Functions

```
std::tuple<RouteResult, size_t> routeProcess(size_t sourceSamplingPoint, Session &session)  
const
```

Routes in normal mode.

**Return** The route if the goal is not yet reached and still reachable.

```
std::tuple<RouteResult, size_t> backtrackProcess(size_t sourceSamplingPoint, Session &session)  
const
```

Routes in skip mode.

**Return** The route if the goal is not yet reached and still reachable.

#### Private Members

```
SamplingPointRouter const &router_  
Configuration const configuration_  
Types::Routing::SamplingPointList const &samplingPointList_  
Types::Track::TimeList const &timeList_  
struct Configuration
```

**Public Members**

```
double maxBacktrackingDistance
struct Session
```

**Public Functions**

```
Session(size_t const sourceSamplingPointIndexMinimum_, size_t const targetSampling-
        PointIndexGoal_, std::unordered_set<size_t> const &skippedSamplingPoints_,
        Types::Routing::RouteList &routeList_, SamplingPointRouter::RouteMap &routeMap_,
        Types::Routing::RoutingStatistic &routingStatistic_)
```

**Public Members**

```
size_t const sourceSamplingPointIndexMinimum
size_t const targetSamplingPointIndexGoal
std::unordered_set<size_t> const &skippedSamplingPoints
Types::Routing::RouteList &routeList
Types::Routing::RoutingStatistic &routingStatistic
SamplingPointRouter::RouteMap &routeMap
SamplingPointRouter::VisitedRouteSet visitedRouteSet
```

**Class DirectedCandidateRouter**

- Defined in file\_src\_AppComponents\_Common\_Filter\_Routing\_DirectedCandidateRouter.h

**Nested Relationships****Nested Types**

- Struct DirectedCandidateRouter::Configuration*

**Class Documentation**

```
class DirectedCandidateRouter
```

## Public Functions

```
DirectedCandidateRouter(Core::Graph::Routing::RoutingAlgorithm &algorithm, Configuration const configuration, Types::Routing::SamplingPointList const &samplingPointList, Types::Graph::GraphEdgeMap const &graphEdgeMap, Types::Graph::StreetIndexMap const &streetIndexMap, Types::Track::TimeList const &timeList, Types::Track::VelocityList const &velocityList, Types::Street::SegmentList const &segmentList)

std::shared_ptr<Types::Routing::Route> operator()(SamplingPointsSelection samplingPointsSelection) const
```

## Private Members

```
Core::Graph::Routing::RoutingAlgorithm &algorithm_
Configuration const configuration_
Types::Routing::SamplingPointList const &samplingPointList_
Types::Graph::GraphEdgeMap const &graphEdgeMap_
Types::Graph::StreetIndexMap const &streetIndexMap_
Types::Track::TimeList const &timeList_
Types::Track::VelocityList const &velocityList_
Types::Street::SegmentList const &segmentList_

struct Configuration
```

## Public Members

```
double maxVelocityDifference
bool allowSelfIntersection
double maxAngularDeviation
double accountTurningCircleLength
```

## Class SelectiveSkipper

- Defined in file\_src\_AppComponents\_Common\_Filter\_Routing\_Generic\_Skipper.h

## Class Documentation

```
class SelectiveSkipper
```

## Public Functions

```
SelectiveSkipper(size_t source, size_t target, size_t lowerBound, size_t upperBound,
                    std::unordered_set<size_t> const &blocklist, Skipper::CostFunction const
                    &costFunction, double costLimit, Skipper::Strategy strategy)

bool isValid() const
size_t source() const
size_t target() const
std::unordered_set<size_t> skipped() const
bool next()
```

## Private Members

```
std::array<Skipper, 3> skipppers_
Skipper *selectedSkipper_ = {&skipppers_[0]}
```

## Class Skipper

- Defined in file\_src\_AppComponents\_Common\_Filter\_Routing\_Generic\_Skipper.h

## Class Documentation

```
class Skipper
```

### Public Types

```
enum Strategy
Values:
includeEdgeCosts
excludeEdgeCosts

using CostFunction = std::function<double (size_t, size_t) >
```

### Public Functions

```
Skipper(size_t source, size_t target, size_t lowerBound, size_t upperBound,
          std::unordered_set<size_t> const &blocklist, CostFunction const &costFunction,
          double costLimit, Strategy strategy)

bool isValid() const
size_t source() const
size_t target() const
double cost() const
```

```
std::unordered_set<size_t> skipped() const
bool next()
```

### Private Members

```
size_t const lowerBound_
size_t const upperBound_
std::unordered_set<size_t> const &blocklist_
CostFunction const &costFunction_
double const costLimit_
Strategy const strategy_
bool isValid_ = {true}
size_t lastSource_
size_t lastTarget_
size_t currentSource_
size_t currentTarget_
double currentCost_ = {0}
std::unordered_set<size_t> skipped_
```

## Class PiecewiseRouter

- Defined in file\_src\_AppComponents\_Common\_Filter\_Routing\_PiecewiseRouter.h

### Class Documentation

```
class PiecewiseRouter
```

### Public Functions

```
PiecewiseRouter (SkipRouter const &router, Types::Routing::SamplingPointList const &samplingPointList, Types::Track::TimeList const &timeList)
bool operator() (Types::Routing::RouteList &routeList, Types::Routing::RoutingStatistic &routingStatistic)
```

## Private Members

```
SkipRouter const &router_
Types::Routing::SamplingPointList const &samplingPointList_
Types::Track::TimeList const &timeList_
```

## Class SamplingPointRouter

- Defined in file\_src\_AppComponents\_Common\_Filter\_Routing\_SamplingPointRouter.h

## Nested Relationships

### Nested Types

- Struct SamplingPointRouter::Configuration*

## Class Documentation

```
class SamplingPointRouter
```

### Public Types

```
using VisitedRouteSet = std::unordered_set<SamplingPointsSelection>
using RouteMap = std::unordered_map<SamplingPointsSelection, std::shared_ptr<Types::Routing::Route>>
```

### Public Functions

```
SamplingPointRouter (DirectedCandidateRouter const &router, Configuration const configuration,
                     Types::Routing::SamplingPointList const &samplingPointList,
                     Types::Graph::GraphEdgeMap const &graphEdgeMap)
```

#### Parameters

- maxClusteredRoutesLengthDifference: Should be 4 times sampling point candidate search radius.

```
std::shared_ptr<Types::Routing::Route> operator() (size_t sourceSamplingPointIndex, size_t targetSamplingPointIndex, Types::Routing::RouteList const &routeList, VisitedRouteSet &visitedRouteSet, RouteMap &routeMap, Types::Routing::RoutingStatistic &routingStatistic) const
```

### Private Functions

```
std::vector<std::shared_ptr<Types::Routing::Route>> calcPossibleRoutes (size_t      sourceSamplingPointIndex,
                                                               size_t      targetSamplingPointIndex,
                                                               VisitatedRouteSet &visitatedRouteSet, RouteMap &routeMap,
                                                               Types::Routing::RoutingStatistic &routingStatistic,
                                                               Types::Routing::RouteList const &routeList)
                                                               const
```

### Private Members

```
DirectedCandidateRouter const &router_
Configuration const configuration_
Types::Routing::SamplingPointList const &samplingPointList_
Types::Graph::GraphEdgeMap const &graphEdgeMap_
struct Configuration
```

### Public Members

```
double maxClusteredRoutesLengthDifference
RouteClusterPreference routeClusterPreference
```

## Class SkipRouter

- Defined in file\_src\_AppComponents\_Common\_Filter\_Routing\_SkipRouter.h

### Nested Relationships

#### Nested Types

- Struct SkipRouter::Configuration*
- Struct SkipRouter::Session*

## Class Documentation

```
class SkipRouter
```

### Public Functions

```
SkipRouter (BacktrackRouter const &router, Configuration const configuration,
Types::Routing::SamplingPointList const &samplingPointList, Types::Track::TimeList
const &timeList)
```

```
void operator() (size_t sourceSamplingPointIndexStart, size_t targetSamplingPointIndexGoal,
Types::Routing::RouteList &routeList, Types::Routing::RoutingStatistic &rout-
ingStatistic) const
```

### Private Functions

```
std::tuple<RouteResult, size_t> routeProcess (size_t sourceSamplingPoint, Session &session)
const
```

Routes in normal mode.

**Return** The RouteResult and reached sampling point index.

```
std::tuple<RouteResult, size_t> skipProcess (size_t sourceSamplingPoint, Session &session) const
```

Routes in skip mode.

**Return** The RouteResult and reached sampling point index.

### Private Members

```
BacktrackRouter const &router_
Configuration const configuration_
Types::Routing::SamplingPointList const &samplingPointList_
Types::Track::TimeList const &timeList_
```

```
struct Configuration
```

### Public Members

```
double maxBacktrackingDistance
Generic::Skipper::Strategy skipStrategy
```

```
struct Session
```

## Public Functions

```
Session(size_t const sourceSamplingPointIndexStart_, size_t const targetSamplingPointIndexGoal_, Types::Routing::RouteList &routeList_, Types::Routing::RoutingStatistic &routingStatistic_)
```

## Public Members

```
size_t const sourceSamplingPointIndexStart  
size_t const targetSamplingPointIndexGoal  
Types::Routing::RouteList &routeList  
Types::Routing::RoutingStatistic &routingStatistic  
SamplingPointRouter::RouteMap routeMap  
std::unordered_set<size_t> skippedSamplingPoints
```

## Class SamplingPointFinder

- Defined in file\_src\_AppComponents\_Common\_Filter\_SamplingPointFinder.h

### Inheritance Relationships

#### Base Type

- public Filter

### Class Documentation

```
class SamplingPointFinder : public Filter
```

#### Public Types

```
enum SelectionStrategy  
Values:  
    all  
    best  
    singles
```

## Public Functions

```
SamplingPointFinder (SelectionStrategy selectionStrategy, double searchRadius, double maxHeadingDifference)

bool operator () (Types::Track::PointList const &pointList, Types::Track::HeadingList
                  const &headingList, Types::Street::SegmentList const &segmentList,
                  Types::Street::TravelDirectionList const &travelDirectionList,
                  Types::Routing::SamplingPointList &samplingPointList)
```

## Private Members

```
SelectionStrategy const selectionStrategy_
double const searchRadius_
double const maxHeadingDifference_
```

## Class CsvTrackReader

- Defined in file\_src\_AppComponents\_ExampleMatcher\_Filter\_CsvTrackReader.h

## Inheritance Relationships

### Base Type

- public Filter

## Class Documentation

```
class CsvTrackReader : public Filter
```

## Public Functions

```
CsvTrackReader (std::istream &input)

bool operator () (AppComponents::Common::Types::Track::TimeList &timeList,
                  AppComponents::Common::Types::Track::PointList &pointList,
                  AppComponents::Common::Types::Track::HeadingList &headingList,
                  AppComponents::Common::Types::Track::VelocityList &velocityList)
```

### Private Members

std::istream &**input\_**

### Class JsonTrackReader

- Defined in file\_src\_AppComponents\_ExampleMatcher\_Filter\_JsonTrackReader.h

### Inheritance Relationships

#### Base Type

- public Filter

### Class Documentation

```
class JsonTrackReader : public Filter
```

#### Public Functions

**JsonTrackReader** (std::istream &*input*)

```
bool operator() (AppComponents::Common::Types::Track::TimeList &timeList, App-
Components::Common::Types::Track::PointList &pointList, AppCompo-
nents::Common::Types::Track::HeadingList &headingList, AppCompo-
nents::Common::Types::Track::VelocityList &velocityList)
```

### Private Members

std::istream &**input\_**

### Class Connection

- Defined in file\_src\_Core\_Common\_Postgres\_Connection.h

### Class Documentation

#### class Connection

This class holds connection credentials and the strategy on how to deal with multiple connections. A PostgreSQL connection can then be opened using those credentials with *getConnection()*.

## Public Types

```
enum Strategy
    Values:
        globalLocked
            open a global connection (next connection request will block until the current connection is no longer used)
        globalUnlocked
            open a global connection (without locking) (should be preferred if you only need a single connection)
        local
            open a new local connection
```

## Public Functions

```
Connection (Strategy strategy, std::string const host, unsigned short const port, std::string const
           dbName, std::string const dbUser, std::string const dbPass)

std::shared_ptr<pqxx::connection> getConnection()
```

## Private Functions

```
void ensureConnection()
```

## Private Members

```
Strategy strategy_
std::shared_ptr<pqxx::connection> connection_
std::mutex mutex_
std::string const host_
unsigned short const port_
std::string const dbName_
std::string const dbUser_
std::string const dbPass_
```

## Class Graph

- Defined in file\_src\_Core\_Graph\_Graph.h

## Inheritance Relationships

### Derived Type

- public Core::Graph::LemonDigraph (*Class LemonDigraph*)

## Class Documentation

### **class Graph**

Subclassed by *Core::Graph::LemonDigraph*

#### Public Functions

```
virtual ~Graph()

virtual Node createNode() = 0

virtual void remove (Node node) = 0

virtual Edge addEdge (Node source, Node target) = 0

virtual void remove (Edge edge) = 0

virtual bool has (Node node) const = 0

virtual bool has (Edge edge) const = 0

virtual Node source (Edge edge) const = 0

virtual Node target (Edge edge) const = 0

virtual std::vector<Edge> outEdges (Node node) const = 0

virtual std::vector<Edge> inEdges (Node node) const = 0
```

#### Protected Functions

```
Node newNode (size_t id) const

Edge newEdge (size_t id) const
```

## Class LemonDigraph

- Defined in file\_src\_Core\_Graph\_LemonDigraph.h

## Inheritance Relationships

### Base Type

- public Core::Graph::Graph (*Class Graph*)

### Class Documentation

```
class LemonDigraph : public Core::Graph::Graph
```

#### Public Functions

```
Node createNode ()  

void remove (Node node)  

Edge addEdge (Node source, Node target)  

void remove (Edge edge)  

bool has (Node node) const  

bool has (Edge edge) const  

Node source (Edge edge) const  

Node target (Edge edge) const  

std::vector<Edge> outEdges (Node node) const  

std::vector<Edge> inEdges (Node node) const  

Node fromLemonNode (lemon::ListDigraph::Node lemonNode) const  

lemon::ListDigraph::Node toLemonNode (Node node) const  

Edge fromLemonArc (lemon::ListDigraph::Arc lemonArc) const  

lemon::ListDigraph::Arc toLemonArc (Edge edge) const
```

#### Private Members

```
lemon::ListDigraph graph_
```

### Class Dijkstra

- Defined in file\_src\_Core\_Graph\_Routing\_Dijkstra.h

## Nested Relationships

### Nested Types

- *Class Dijkstra::PathNode*

## Inheritance Relationships

### Base Type

- public Core::Graph::Routing::RoutingAlgorithm (*Class RoutingAlgorithm*)

## Class Documentation

```
class Dijkstra : public Core::Graph::RoutingAlgorithm
```

### Public Functions

```
Dijkstra (Core::Graph::Graph const &graph)
```

```
PathView operator () (Core::Graph::Node source, Core::Graph::Node destination)
```

### Private Types

```
using Container = std::deque<std::shared_ptr<PathNode>>
using GreaterFunction = std::greater<std::shared_ptr<Dijkstra::PathNode>>
using Frontier = std::priority_queue<std::shared_ptr<PathNode>, Container, GreaterFunction>
using Visited = std::unordered_set<Core::Graph::Edge>
```

### Private Functions

```
void init (Core::Graph::Node source)
```

```
std::vector<std::shared_ptr<Dijkstra::PathNode>> explore (std::shared_ptr<PathNode> const &parent)
```

```
bool reached (std::shared_ptr<PathNode> const &pathNode, Core::Graph::Node graphNode)
```

## Private Members

```
Core::Graph::Graph const &graph_  
Frontier frontier_  
Visited visited_  
class PathNode : public Core::Graph::Routing::PathViewImpl
```

## Public Functions

```
PathNode (Core::Graph::Edge edge, double cost, std::shared_ptr<PathNode> const &previous)  
Core::Graph::Edge edge () const  
double cost () const  
PathViewImpl *previous () const
```

## Public Members

```
Core::Graph::Edge edge_  
double cost_  
std::shared_ptr<PathNode> previous_
```

## Class Dijkstra::PathNode

- Defined in file\_src\_Core\_Graph\_Routing\_Dijkstra.h

## Nested Relationships

This class is a nested type of *Class Dijkstra*.

## Inheritance Relationships

### Base Type

- public Core::Graph::Routing::PathViewImpl (*Class PathViewImpl*)

## Class Documentation

```
class PathNode : public Core::Graph::Routing::PathViewImpl
```

## Public Functions

```
PathNode (Core::Graph::Edge edge, double cost, std::shared_ptr<PathNode> const &previous)  
Core::Graph::Edge edge () const  
double cost () const  
PathViewImpl *previous () const
```

## Public Members

```
Core::Graph::Edge edge_  
double cost_  
std::shared_ptr<PathNode> previous_
```

## Class PathView

- Defined in file\_src\_Core\_Graph\_Routing\_PathView.h

## Class Documentation

```
class PathView
```

### Public Types

```
using Iterator = PathViewIterator
```

### Public Functions

```
PathView (PathViewImpl *impl)  
Core::Graph::Edge edge () const  
double cost () const  
PathView back () const  
PathView front () const  
PathView previous () const  
PathView::Iterator begin () const  
PathView::Iterator end () const  
bool empty () const  
size_t size () const
```

## Private Members

`PathViewImpl *impl_`

## Class PathViewImpl

- Defined in file\_src\_Core\_Graph\_Routing\_PathView.h

## Inheritance Relationships

### Derived Type

- public Core::Graph::Routing::Dijkstra::PathNode (*Class Dijkstra::PathNode*)

## Class Documentation

```
class PathViewImpl
Subclassed by Core::Graph::Routing::Dijkstra::PathNode
```

## Public Functions

```
virtual ~PathViewImpl()
virtual Core::Graph::Edge edge() const = 0
virtual double cost() const = 0
virtual PathViewImpl *previous() const = 0
```

## Class PathViewIterator

- Defined in file\_src\_Core\_Graph\_Routing\_PathView.h

## Inheritance Relationships

### Base Type

- public boost::iterator\_facade< PathViewIterator, PathView,
boost::forward\_traversal\_tag, PathView >

## Class Documentation

```
class PathViewIterator : public boost::iterator_facade<PathViewIterator, PathView, boost::forward_traversal_tag, PathView>
```

### Public Functions

```
PathViewIterator (PathViewImpl *path)
```

### Private Functions

```
void increment ()  
bool equal (PathViewIterator const &other) const  
PathView dereference () const
```

### Private Members

```
PathViewImpl *path_
```

### Friends

```
friend Core::Graph::Routing::PathViewIterator::boost::iterator_core_access
```

## Class RoutingAlgorithm

- Defined in file\_src\_Core\_Graph\_Routing\_Algorithm.h

## Inheritance Relationships

### Derived Type

- public Core::Graph::Routing::Dijkstra (*Class Dijkstra*)

## Class Documentation

```
class RoutingAlgorithm
```

Subclassed by *Core::Graph::Routing::Dijkstra*

## Public Functions

```
RoutingAlgorithm()  
  
virtual ~RoutingAlgorithm()  
  
virtual PathView operator() (Core::Graph::Node source, Core::Graph::Node destination) = 0  
  
virtual PathView run (Core::Graph::Node source, Core::Graph::Node destination)  
  
virtual RoutingAlgorithm & setCost (CostFunction costFuncton)  
  
virtual RoutingAlgorithm & setFilter (FilterFunction filterFunction)
```

## Protected Attributes

```
CostFunction costFunction_  
FilterFunction filterFunction_
```

## Class Progress

- Defined in file\_src\_Generic\_Progress\_Progress.h

## Class Documentation

```
class Progress
```

### Public Functions

```
Progress (std::string const &text, uint64_t ticks)  
void tick()  
void updateTotal (uint64_t ticks)
```

### Private Types

```
using Clock = std::chrono::steady_clock  
using Duration = std::chrono::duration<float, Clock::period>
```

### Private Functions

```
void output (std::ostream &os, Duration duration)
```

**Private Members**

```
std::string text_  
std::uint64_t totalTicks_  
std::uint64_t ticksOccured_  
Clock::time_point const beginTime_
```

### 8.3.3 Enums

**Enum RouteClusterPreference**

- Defined in file\_src\_AppComponents\_Common\_Filter\_Routing\_Types.h

**Enum Documentation**

```
enum AppComponents::Common::Filter::Routing::RouteClusterPreference  
Values:  
cheapest  
shortest
```

**Enum RouteResult**

- Defined in file\_src\_AppComponents\_Common\_Filter\_Routing\_Helper.h

**Enum Documentation**

```
enum AppComponents::Common::Filter::Routing::RouteResult  
Values:  
goalReached  
goalNotReachedButReachable  
goalNotReachable
```

**Enum HighwayType**

- Defined in file\_src\_AppComponents\_Common\_Types\_Street\_Highway.h

## Enum Documentation

**enum** AppComponents::Common::Types::Street::**HighwayType**  
See <https://wiki.openstreetmap.org/wiki/Key:highway>

*Values:*

**motorway**  
**trunk**  
**primary**  
**secondary**  
**tertiary**  
**motorway\_link**  
**trunk\_link**  
**primary\_link**  
**secondary\_link**  
**tertiary\_link**

## Enum TravelDirection

- Defined in file\_src\_AppComponents\_Common\_Types\_Street\_TravelDirection.h

## Enum Documentation

**enum** AppComponents::Common::Types::Street::**TravelDirection**  
*Values:*

**both**  
Travelling allowed in both directions.  
**forwards**  
Travelling allowed in the direction the coordinates are defined.  
**backwards**  
Travelling allowed in reverse of the direction the coordinates are defined.

## 8.3.4 Functions

### Template Function anonymous\_namespace\_FlipMap.h::flipPair

- Defined in file\_src\_Generic\_Map\_FlipMap.h

**Function Documentation**

```
template<typename A, typename B>std::pair<B, A> anonymous_namespace_FlipMap.h::flipPair (co
```

**Function anonymous\_namespace\_MakeHashable.h::hash\_combine(std::size\_t&)**

- Defined in file\_src\_Generic\_Hash\_MakeHashable.h

**Function Documentation**

**Warning:** doxygenfunction: Unable to resolve multiple matches for function “anonymous\_namespace\_MakeHashable.h::hash\_combine” with arguments (std::size\_t&) in doxygen xml output for project “OS-Matcher” from directory: /builds/os-matcher/os-matcher-on-github/build/docs/doxygen/xml. Potential matches:

```
- template<typename T, typename... Rest>void anonymous_namespace_MakeHashable.  
→h::hash_combine(std::size_t &, const T &, Rest...)  
- void anonymous_namespace_MakeHashable.h::hash_combine(std::size_t &)
```

**Template Function anonymous\_namespace\_MakeHashable.h::hash\_combine(std::size\_t&, const T&, Rest...)**

- Defined in file\_src\_Generic\_Hash\_MakeHashable.h

**Function Documentation**

**Warning:** doxygenfunction: Unable to resolve multiple matches for function “anonymous\_namespace\_MakeHashable.h::hash\_combine” with arguments (std::size\_t&, const T&, Rest...) in doxygen xml output for project “OS-Matcher” from directory: /builds/os-matcher/os-matcher-on-github/build/docs/doxygen/xml. Potential matches:

```
- template<typename T, typename... Rest>void anonymous_namespace_MakeHashable.  
→h::hash_combine(std::size_t &, const T &, Rest...)  
- void anonymous_namespace_MakeHashable.h::hash_combine(std::size_t &)
```

**Function anonymous\_namespace\_SamplingPointFinder.cpp::addStreetIndex**

- Defined in file\_src\_AppComponents\_Common\_Filter\_SamplingPointFinder.cpp

## Function Documentation

```
void anonymous_namespace_SamplingPointFinder.cpp::addStreetIndex(StreetIndexGeoindex & geoindex)
```

Add street index and all segment indices to spatial index.

### Function anonymous\_namespace\_SamplingPointFinder.cpp::getStreetIndices

- Defined in file\_src\_AppComponents\_Common\_Filter\_SamplingPointFinder.cpp

## Function Documentation

```
std::vector<std::pair<size_t, size_t> > anonymous_namespace_SamplingPointFinder.cpp::getStreetIndices()
```

vector of { streetIndex, streetSegmentIndex }

### Function anonymous\_namespace\_SamplingPointFinder.cpp::headingDifference

- Defined in file\_src\_AppComponents\_Common\_Filter\_SamplingPointFinder.cpp

## Function Documentation

```
std::pair<double, AppComponents::Common::Types::Street::TravelDirection> anonymous_namespace_SamplingPointFinder.cpp::headingDifference(double trackHeading, TravelDirection trackTravelDirection, double segmentHeading, TravelDirection segmentTravelDirection)
```

headingDifference, trackTravelDirection } trackTravelDirection is TravelDirection::both if travelDirection is also TravelDirection::both and the heading difference is nearly 90 degrees.

### Parameters

- trackHeading: Heading of the track point.
- segmentHeading: Heading in forwards-direction of the segment. The reversed segmentHeading may be considered for the difference-calculation.
- travelDirection: If TravelDirection::both, the direction nearest to the trackHeading is considered. If TravelDirection::backwards the segmentHeading is considered reversed.

### Function AppComponents::Common::Filter::anonymous\_namespace\_GeoJsonMapReader.cpp::toHighway

- Defined in file\_src\_AppComponents\_Common\_Filter\_GeoJsonMapReader.cpp

## Function Documentation

```
Types::Street::Highway AppComponents::Common::Filter::anonymous_namespace_GeoJsonMapReader::toHighway(const GeoJsonMap & map, const GeoJsonLine & line, const GeoJsonPoint & point)
```

### Function AppComponents::Common::Filter::anonymous\_namespace\_GeoJsonMapReader.cpp::toTravelDirection

- Defined in file\_src\_AppComponents\_Common\_Filter\_GeoJsonMapReader.cpp

#### Function Documentation

`Types::Street::TravelDirection AppComponents::Common::Filter::anonymous_namespace_GeoJsonMapReader::toTravelDirection(const Types::Street::TravelDirection&)`

### Function AppComponents::Common::Filter::anonymous\_namespace\_GeoJsonMapWriter.cpp::toString(Types::Street::TravelDirection const)

- Defined in file\_src\_AppComponents\_Common\_Filter\_GeoJsonMapWriter.cpp

#### Function Documentation

**Warning:** doxygenfunction: Unable to resolve multiple matches for function “AppComponents::Common::Filter::anonymous\_namespace\_GeoJsonMapWriter.cpp::toString” with arguments (Types::Street::TravelDirection const) in doxygen xml output for project “OS-Matcher” from directory: /builds/os-matcher/os-matcher-on-github/build/docs/doxygen/xml. Potential matches:

```
- std::string AppComponents::Common::Filter::anonymous_namespace_GeoJsonMapWriter.  
  ~cpp::toString(Types::Street::Highway const)  
- std::string AppComponents::Common::Filter::anonymous_namespace_GeoJsonMapWriter.  
  ~cpp::toString(Types::Street::TravelDirection const)
```

### Function AppComponents::Common::Filter::anonymous\_namespace\_GeoJsonMapWriter.cpp::toString(Types::Street::TravelDirection const)

- Defined in file\_src\_AppComponents\_Common\_Filter\_GeoJsonMapWriter.cpp

#### Function Documentation

**Warning:** doxygenfunction: Unable to resolve multiple matches for function “AppComponents::Common::Filter::anonymous\_namespace\_GeoJsonMapWriter.cpp::toString” with arguments (Types::Street::Highway const) in doxygen xml output for project “OS-Matcher” from directory: /builds/os-matcher/os-matcher-on-github/build/docs/doxygen/xml. Potential matches:

```
- std::string AppComponents::Common::Filter::anonymous_namespace_GeoJsonMapWriter.  
  ~cpp::toString(Types::Street::Highway const)  
- std::string AppComponents::Common::Filter::anonymous_namespace_GeoJsonMapWriter.  
  ~cpp::toString(Types::Street::TravelDirection const)
```

**Function AppComponents::Common::Filter::anonymous\_namespace\_JsonRouteStatisticWriter.cpp::toJson**

- Defined in file\_src\_AppComponents\_Common\_Filter\_JsonRouteStatisticWriter.cpp

**Function Documentation**

```
nlohmann::json AppComponents::Common::Filter::anonymous_namespace_JsonRouteStatisticWriter
```

**Function AppComponents::Common::Filter::anonymous\_namespace\_OsmMapReader.cpp::getCandidates**

- Defined in file\_src\_AppComponents\_Common\_Filter\_OsmMapReader.cpp

**Function Documentation**

```
std::tuple<std::vector<Candidate>, std::unordered_map<size_t, OsmPointCandidate>> AppComponents::Common::Filter::anonymous_namespace_OsmMapReader::getCandidates(const std::vector<LineString> &records, const std::map<size_t, std::vector<Point>> &osmPointMap, const GeoIndex &geoindex)
```

**Parameters**

- records: The query should have returned lines and points pairwise sequentially (multiply line data), like { { line1, point1 }, { line1, point2 }, { line2, point1 }, ... }.

**Function AppComponents::Common::Filter::anonymous\_namespace\_OsmMapReader.cpp::getUniquePoint**

- Defined in file\_src\_AppComponents\_Common\_Filter\_OsmMapReader.cpp

**Function Documentation**

```
std::shared_ptr<UniquePoint> AppComponents::Common::Filter::anonymous_namespace_OsmMapReader::getUniquePoint(const LineString &line, const std::map<size_t, std::vector<Point>> &osmPointMap, const GeoIndex &geoindex)
```

Points are considered equal if the distance is 10cm or less.

**Function AppComponents::Common::Filter::anonymous\_namespace\_OsmMapReader.cpp::processCandidates**

- Defined in file\_src\_AppComponents\_Common\_Filter\_OsmMapReader.cpp

**Function Documentation**

```
std::tuple<Types::Street::SegmentList, Types::Street::NodePairList, Types::Street::TravelDirections> AppComponents::Common::Filter::anonymous_namespace_OsmMapReader::processCandidates(const LineString &line, const std::map<size_t, std::vector<Point>> &osmPointMap, const GeoIndex &geoindex)
```

Note: Candidates are split on shared point intersections.

### Function AppComponents::Common::Filter::Osm::toHighway

- Defined in file\_src\_AppComponents\_Common\_Filter\_Osm\_Conversion.cpp

#### Function Documentation

```
Types::Street::Highway AppComponents::Common::Filter::Osm::toHighway (std::string const
&highway)
```

### Function AppComponents::Common::Filter::Osm::toHighwaySelectionSql

- Defined in file\_src\_AppComponents\_Common\_Filter\_Osm\_Conversion.cpp

#### Function Documentation

```
std::string AppComponents::Common::Filter::Osm::toHighwaySelectionSql (std::unordered_set<Types::Street::Highway>
const
&highways,
std::string
const
&table-
Name)
```

### Function AppComponents::Common::Filter::Osm::toOsmString

- Defined in file\_src\_AppComponents\_Common\_Filter\_Osm\_Conversion.cpp

#### Function Documentation

```
std::string AppComponents::Common::Filter::Osm::toOsmString (Types::Street::HighwayType
const highway)
```

### Function AppComponents::Common::Filter::Osm::toTravelDirection

- Defined in file\_src\_AppComponents\_Common\_Filter\_Osm\_Conversion.cpp

#### Function Documentation

```
Types::Street::TravelDirection AppComponents::Common::Filter::Osm::toTravelDirection (std::string
const
&oneway)
```

**Function AppComponents::Common::Filter::Routing::anonymous\_namespace\_Comparators.cpp::isContained**

- Defined in file\_src\_AppComponents\_Common\_Filter\_Routing\_Comparators.cpp

**Function Documentation**

```
bool AppComponents::Common::Filter::Routing::anonymous_namespace_Comparators.cpp::isContained
```

**Function AppComponents::Common::Filter::Routing::anonymous\_namespace\_SamplingPointRouter.cpp::cluster**

- Defined in file\_src\_AppComponents\_Common\_Filter\_Routing\_SamplingPointRouter.cpp

**Function Documentation**

```
std::shared_ptr<ClusteredRouteMatrix> AppComponents::Common::Filter::Routing::anonymous_na
```

**Function AppComponents::Common::Filter::Routing::anonymous\_namespace\_SamplingPointRouter.cpp::getBest**

- Defined in file\_src\_AppComponents\_Common\_Filter\_Routing\_SamplingPointRouter.cpp

**Function Documentation**

```
std::shared_ptr<Types::Routing::Route> AppComponents::Common::Filter::Routing::anonymous_na
```

**Function AppComponents::Common::Filter::Routing::anonymous\_namespace\_SamplingPointRouter.cpp::routeC**

- Defined in file\_src\_AppComponents\_Common\_Filter\_Routing\_SamplingPointRouter.cpp

**Function Documentation**

```
std::shared_ptr<Types::Routing::Route> AppComponents::Common::Filter::Routing::anonymous_na
```

**Function AppComponents::Common::Filter::Routing::anonymous\_namespace\_SamplingPointRouter.cpp::selectC**

- Defined in file\_src\_AppComponents\_Common\_Filter\_Routing\_SamplingPointRouter.cpp

**Function Documentation**

```
std::vector<SamplingPointCandidateSelectionPair> AppComponents::Common::Filter::Routing::an
```

**Function AppComponents::Common::Filter::Routing::attachToPreviousRoute**

- Defined in file\_src\_AppComponents\_Common\_Filter\_Routing\_Helper.cpp

**Function Documentation**

```
size_t AppComponents::Common::Filter::Routing::attachToPreviousRoute (Types::Routing::RouteList& routeList,  
size_t source-  
Sampling-  
PointIn-  
dex)
```

**Function AppComponents::Common::Filter::Routing::calcApproximateDistanceBetweenSamplingPoints**

- Defined in file\_src\_AppComponents\_Common\_Filter\_Routing\_Helper.cpp

**Function Documentation**

```
double AppComponents::Common::Filter::Routing::calcApproximateDistanceBetweenSamplingPoints (s
```

**Function AppComponents::Common::Filter::Routing::checkMaxAngularDeviation**

- Defined in file\_src\_AppComponents\_Common\_Filter\_Routing\_Helper.cpp

## Function Documentation

```
bool AppComponents::Common::Filter::Routing::checkMaxAngularDeviation(Types::Routing::Route
    const
    &route,
    double
    const
    maxAn-
    gu-
    larDevi-
    ation)
```

### Function AppComponents::Common::Filter::Routing::findPreviousConnectedRoute

- Defined in file\_src\_AppComponents\_Common\_Filter\_Routing\_Helper.cpp

## Function Documentation

```
std::optional<std::shared_ptr<Types::Routing::Route>> AppComponents::Common::Filter::Routing::findPreviousC
```

### Function AppComponents::Common::Filter::Routing::Generic::findNextAllowed

- Defined in file\_src\_AppComponents\_Common\_Filter\_Routing\_Generic\_Helper.cpp

## Function Documentation

```
std::optional<size_t> AppComponents::Common::Filter::Routing::Generic::findNextAllowed(size_t
    sam-
    pling-
    Point,
    size_t
    max,
    std::unordered_s
    const
    &skipped-
    Sam-
    pling-
    Points)
```

Find next sampling point (starting at samplingPoint) samplingPoint that was not skipped.

**Function AppComponents::Common::Filter::Routing::Generic::findPreviousAllowed**

- Defined in file\_src\_AppComponents\_Common\_Filter\_Routing\_Generic\_Helper.cpp

**Function Documentation**

```
std::optional<size_t> AppComponents::Common::Filter::Routing::Generic::findPreviousAllowed(size_t  
    samplingPoint,  
    size_t min,  
    std::unord-  
    const  
    &skipped-  
    Sampling-  
    Points)
```

Find previous sampling point (starting at samplingPoint) that was not skipped.

**Function AppComponents::Common::Filter::Routing::geoDistance**

- Defined in file\_src\_AppComponents\_Common\_Filter\_Routing\_Helper.cpp

**Function Documentation**

```
double AppComponents::Common::Filter::Routing::geoDistance(Core::Common::Geometry::LineString  
    lineString)
```

**Function AppComponents::Common::Filter::Routing::isSelfIntersectingRoute**

- Defined in file\_src\_AppComponents\_Common\_Filter\_Routing\_Helper.cpp

**Function Documentation**

```
bool AppComponents::Common::Filter::Routing::isSelfIntersectingRoute(Types::Routing::Route  
    const  
    &route)
```

**Function AppComponents::Common::Filter::Routing::isSimilar**

- Defined in file\_src\_AppComponents\_Common\_Filter\_Routing\_Comparators.cpp

## Function Documentation

```
bool AppComponents::Common::Filter::Routing::isSimilar(Types::Routing::Route const
&a, Types::Routing::Route const &b, double const
maxLengthDifference,
Types::Graph::GraphEdgeMap
const &graphEdgeMap)
```

## Function AppComponents::Common::Filter::routeResultToString

- Defined in file\_src\_AppComponents\_Common\_Filter\_Routing\_Helper.h

## Function Documentation

```
std::string AppComponents::Common::Filter::Routing::routeResultToString(RouteResult
const
result)
```

## Function Core::Common::Geometry::absHeadingDiff

- Defined in file\_src\_Core\_Common\_Geometry\_Helper.cpp

## Function Documentation

```
double Core::Common::Geometry::absHeadingDiff(double a, double b)
Calculate difference in heading normalized to the range [0°, +180°].
```

**Return** angle difference in degree.

## Function Core::Common::Geometry::angleBetweenSegments

- Defined in file\_src\_Core\_Common\_Geometry\_Helper.cpp

## Function Documentation

```
double Core::Common::Geometry::angleBetweenSegments(Segment const &segment1, Segment
const &segment2)
Returns the angle between segment1 and segment2 in range [180°, -180°].
```

**Function Core::Common::Geometry::buffer**

- Defined in file\_src\_Core\_Common\_Geometry\_Helper.cpp

**Function Documentation**

*Box* Core::Common::Geometry::**buffer** (*Box const &box, double distanceMeter*)  
Enlarge a box by *distanceMeter* on all sides.

**Template Function Core::Common::Geometry::degree**

- Defined in file\_src\_Core\_Common\_Geometry\_Helper.h

**Function Documentation**

```
template<typename T>
constexpr auto Core::Common::Geometry::degree (T rad)
    Convert from radians to degree.

Return angle in deg.
```

**Function Core::Common::Geometry::flattened\_simple**

- Defined in file\_src\_Core\_Common\_Geometry\_Helper.cpp

**Function Documentation**

```
std::vector<Core::Common::Geometry::Point> Core::Common::Geometry::flattened_simple (std::vector<Core::Common::Geometry::Point const &points, size_t const keep-Each-N-th-Point)
```

**Function Core::Common::Geometry::geoDistance(Point const&, Point const&)**

- Defined in file\_src\_Core\_Common\_Geometry\_Helper.cpp

## Function Documentation

*ValueType* Core::Common::Geometry::geoDistance (*Point const &g1*, *Point const &g2*)

### Function Core::Common::Geometry::geoDistance(Point const&, Segment const&)

- Defined in file\_src\_Core\_Common\_Geometry\_Helper.cpp

## Function Documentation

std::tuple<*ValueType*, *Point*, double> Core::Common::Geometry::geoDistance (*Point const &g1*,  
*Segment const &g2*)

**Return** { distance, projectedPointOnSegment, normLengthOnSegment }

### Template Function Core::Common::Geometry::geoDistance(G1 const&, G2 const&)

- Defined in file\_src\_Core\_Common\_Geometry\_Helper.h

## Function Documentation

template<typename **G1**, typename **G2**>

*ValueType* Core::Common::Geometry::geoDistance (*G1 const &g1*, *G2 const &g2*)  
Calculate distance between geometries g1 and g2 on earth surface.

**Return** distance in meters.

### Template Function Core::Common::Geometry::geoLength

- Defined in file\_src\_Core\_Common\_Geometry\_Helper.h

## Function Documentation

template<typename **G1**>

auto Core::Common::Geometry::geoLength (*G1 const &g1*)  
Calculate length of geometry g1 on earth surface.

**Return** length in meters.

**Function Core::Common::Geometry::heading(Point const&, Point const&)**

- Defined in file\_src\_Core\_Common\_Geometry\_Helper.cpp

**Function Documentation**

```
double Core::Common::Geometry::heading (Point const &p1, Point const &p2)
```

Calculate the initial heading for a great-circle route from point p1 to point p2.

**Return** heading in degrees.

**Function Core::Common::Geometry::heading(Segment const&)**

- Defined in file\_src\_Core\_Common\_Geometry\_Helper.cpp

**Function Documentation**

```
double Core::Common::Geometry::heading (Segment const &segment)
```

Calculate the heading between segment start and end point.

**Return** heading in degrees.

**Function Core::Common::Geometry::headingDiff**

- Defined in file\_src\_Core\_Common\_Geometry\_Helper.cpp

**Function Documentation**

```
double Core::Common::Geometry::headingDiff (double a, double b)
```

Calculate difference in heading normalized to the range (-180°, +180°].

**Return** angle difference in degree.

**Function Core::Common::Geometry::normalizeAngle**

- Defined in file\_src\_Core\_Common\_Geometry\_Helper.cpp

**Function Documentation**

```
double Core::Common::Geometry::normalizeAngle (double angle)
```

Normalize angle, given in degrees, to the range [0°, 360°].

**Return** angle in degrees.

### Function Core::Common::Geometry::operator""\_lat

- Defined in file\_src\_Core\_Common\_Geometry\_Types.cpp

#### Function Documentation

*Point::Latitude* Core::Common::Geometry::**operator""\_lat** (long double *a*)

### Function Core::Common::Geometry::operator""\_lon

- Defined in file\_src\_Core\_Common\_Geometry\_Types.cpp

#### Function Documentation

*Point::Longitude* Core::Common::Geometry::**operator""\_lon** (long double *a*)

### Function Core::Common::Geometry::operator==

- Defined in file\_src\_Core\_Common\_Geometry\_Types.h

#### Function Documentation

bool Core::Common::Geometry::**operator==(Point const &*a*, Point const &*b*)**

### Function Core::Common::Geometry::project

- Defined in file\_src\_Core\_Common\_Geometry\_Helper.cpp

#### Function Documentation

std::pair<Point, double> Core::Common::Geometry::**project** (Point const &*p*, Segment const &*segment*)

Project *Point* onto Segment.

**Return** Projection point and distance along segment normalized to segment length.

### Function Core::Common::Geometry::projectOntoLineString

- Defined in file\_src\_Core\_Common\_Geometry\_Helper.cpp

## Function Documentation

```
LineStringProjectionResult Core::Common::Geometry::projectOntoLineString(Point const
&point,
LineString
const
&lineString)
```

Search for minimum distance of point to line in the lineString.

- A) p ^ | <~~ distance to segment | p1<=====x----->p2 ^~~~projected point/distance along segment
- B) p / / <~~ distance to segment / p1<=====x>p2 ^~~~projected point/distance along segment

## Template Function Core::Common::Geometry::rad

- Defined in file\_src\_Core\_Common\_Geometry\_Helper.h

## Function Documentation

```
template<typename T>
constexpr auto Core::Common::Geometry::rad(T degree)
Convert from degree to radians.
```

**Return** angle in rad.

## Function Core::Common::Geometry::relativeDistanceAlongLineString

- Defined in file\_src\_Core\_Common\_Geometry\_Helper.cpp

## Function Documentation

```
double Core::Common::Geometry::relativeDistanceAlongLineString(LineString const
&lineString, Point
const &point)
```

## Function Core::Common::Geometry::reversedHeading

- Defined in file\_src\_Core\_Common\_Geometry\_Helper.cpp

## Function Documentation

```
double Core::Common::Geometry::reversedHeading(double heading)
Reverses the given heading.
```

**Return** Reversed heading in the range [0°, 360°].

### Parameters

- heading: Has to be in the range [-180°, 540°).

### Function Core::Common::Geometry::reverseHaversine

- Defined in file\_src\_Core\_Common\_Geometry\_Helper.cpp

#### Function Documentation

`Point Core::Common::Geometry::reverseHaversine (Point const &p, double bearing, double distanceMeter)`

Implementation of the reverse of Haversine formula.

**Return** `Point` with bearing and distanceMeter away from point p.

### Function Core::Common::Geometry::snap

- Defined in file\_src\_Core\_Common\_Geometry\_Helper.cpp

#### Function Documentation

`std::pair<Point, double> Core::Common::Geometry::snap (Point const &p, Segment segment)`  
Clip point p to segment.

**Return** first: closest point to p on segment, second: distance along segment as returned by `project()`.

### Function Core::Common::Geometry::toGeoJson(LineString const&)

- Defined in file\_src\_Core\_Common\_Geometry\_Conversion.cpp

#### Function Documentation

`nlohmann::json Core::Common::Geometry::toGeoJson (LineString const &lineString)`

### Function Core::Common::Geometry::toGeoJson(Point const&)

- Defined in file\_src\_Core\_Common\_Geometry\_Conversion.cpp

#### Function Documentation

`nlohmann::json Core::Common::Geometry::toGeoJson (Point const &point)`

### Function Core::Common::Geometry::toLineString(nlohmann::json const&)

- Defined in file\_src\_Core\_Common\_Geometry\_Conversion.cpp

#### Function Documentation

*LineString* Core::Common::Geometry::**toLineString** (nlohmann::json **const** &*geoJson*)

### Function Core::Common::Geometry::toLineString(std::string const&)

- Defined in file\_src\_Core\_Common\_Geometry\_Conversion.cpp

#### Function Documentation

*LineString* Core::Common::Geometry::**toLineString** (std::string **const** &*wkt*)

### Function Core::Common::Geometry::toPoint

- Defined in file\_src\_Core\_Common\_Geometry\_Conversion.cpp

#### Function Documentation

*Point* Core::Common::Geometry::**toPoint** (std::string **const** &*wkt*)

### Function Core::Common::Geometry::toWkt(Point const&)

- Defined in file\_src\_Core\_Common\_Geometry\_Conversion.cpp

#### Function Documentation

std::string Core::Common::Geometry::**toWkt** (*Point* **const** &*point*)

### Function Core::Common::Geometry::toWkt(std::vector<Point> const&)

- Defined in file\_src\_Core\_Common\_Geometry\_Conversion.cpp

#### Function Documentation

std::string Core::Common::Geometry::**toWkt** (std::vector<*Point*> **const** &*points*)

**Function Core::Common::Geometry::toWkt(std::vector<std::vector<Point>> const&)**

- Defined in file\_src\_Core\_Common\_Geometry\_Conversion.cpp

**Function Documentation**

```
std::string Core::Common::Geometry::toWkt (std::vector<std::vector<Point>> const &points)
```

**Function Core::Common::Geometry::trimmed**

- Defined in file\_src\_Core\_Common\_Geometry\_Helper.cpp

**Function Documentation**

```
std::vector<Core::Common::Geometry::Point> Core::Common::Geometry::trimmed (std::vector<Core::Common::Geometry::  
    const  
    &points,  
    double const  
    trimLeft, dou-  
    ble const  
    trimRight)
```

**Template Function Core::Common::Geometry::within**

- Defined in file\_src\_Core\_Common\_Geometry\_Helper.h

**Function Documentation**

```
template<typename Geometry1, typename Geometry2>  
bool Core::Common::Geometry::within (Geometry1 const &geometry1, Geometry2 const &ge-  
ometry2)
```

**Template Function Core::Common::Postgres::getOptional(C const&)**

- Defined in file\_src\_Core\_Common\_Postgres\_Helper.h

**Function Documentation**

```
template<typename T, typename C>  
std::optional<T> Core::Common::Postgres::getOptional (C const &cell)
```

**Template Function Core::Common::Postgres::getOptional(C const&, T)**

- Defined in file\_src\_Core\_Common\_Postgres\_Helper.h

**Function Documentation**

```
template<typename T, typename C>
T Core::Common::Postgres::getOptional(C const &cell, T defaultValue)
```

**Function Core::Common::Time::fromIsoString**

- Defined in file\_src\_Core\_Common\_Time\_Helper.h

**Function Documentation**

```
std::pair<std::chrono::system_clock::time_point, bool> Core::Common::Time::fromIsoString(std::string
                                         const
                                         &text)
```

**Parameters**

- text: Sth. like 2018-06-22T03:14:10 [.000].

**Function Core::Common::Time::fromIsoZString**

- Defined in file\_src\_Core\_Common\_Time\_Helper.h

**Function Documentation**

```
std::pair<std::chrono::system_clock::time_point, bool> Core::Common::Time::fromIsoZString(std::string
                                         const
                                         &text)
```

**Parameters**

- text: Sth. like 2018-06-22T03:14:10 [.000] Z.

**Function Core::Common::Time::fromString**

- Defined in file\_src\_Core\_Common\_Time\_Helper.cpp

**Function Documentation**

```
std::pair<std::chrono::system_clock::time_point, bool> Core::Common::Time::fromString(std::string
                                         const
                                         &text,
                                         std::string
                                         const
                                         &for-
                                         mat)
```

### Function Core::Common::Time::toIsoString

- Defined in file\_src\_Core\_Common\_Time\_Helper.h

#### Function Documentation

```
std::string Core::Common::Time::toIsoString(std::chrono::system_clock::time_point time)
```

**Return** Sth. like 2018-06-22T03:14:10[.000].

### Function Core::Common::Time::toIsoZString

- Defined in file\_src\_Core\_Common\_Time\_Helper.h

#### Function Documentation

```
std::string Core::Common::Time::toIsoZString(std::chrono::system_clock::time_point time)
```

**Return** Sth. like 2018-06-22T03:14:10[.000]Z.

### Function Core::Common::Time::toString

- Defined in file\_src\_Core\_Common\_Time\_Helper.cpp

#### Function Documentation

```
std::string Core::Common::Time::toString(std::chrono::system_clock::time_point time, std::string  
const &format)
```

### Function Core::Graph::Routing::operator<<

- Defined in file\_src\_Core\_Graph\_Routing\_PathView.cpp

#### Function Documentation

```
std::ostream &Core::Graph::Routing::operator<<(std::ostream &os, PathView pathView)
```

### Function Core::Graph::Routing::operator>

- Defined in file\_src\_Core\_Graph\_Routing\_Dijkstra.cpp

**Function Documentation**

```
bool Core::Graph::Routing::operator> (std::shared_ptr<Dijkstra::PathNode> const &lhs,  
                                     std::shared_ptr<Dijkstra::PathNode> const &rhs)
```

**Template Function Generic::flipMap**

- Defined in file\_src\_Generic\_Map\_FlipMap.h

**Function Documentation**

```
template<typename A, typename B, typename C = std::less<B>, template<class, class, class...> class M, class ...Args>  
std::multimap<B, A, C> Generic::flipMap (const M<A, B, Args...> &src, C const &cmp = C())
```

**Function Generic::String::ltrim**

- Defined in file\_src\_Generic\_String\_Trim.h

**Function Documentation**

```
static void Generic::String::ltrim (std::string &s)
```

**Function Generic::String::ltrimmed**

- Defined in file\_src\_Generic\_String\_Trim.h

**Function Documentation**

```
std::string Generic::String::ltrimmed (std::string s_)
```

**Function Generic::String::readNextRow**

- Defined in file\_src\_Generic\_String\_Csv.h

**Function Documentation**

```
std::vector<std::string> Generic::String::readNextRow (std::istream &str, char const separator)
```

### Function Generic::String::rtrim

- Defined in file\_src\_Generic\_String\_Trim.h

#### Function Documentation

```
static void Generic::String::rtrim(std::string &s)
```

### Function Generic::String::rtrimmed

- Defined in file\_src\_Generic\_String\_Trim.h

#### Function Documentation

```
std::string Generic::String::rtrimmed(std::string s_)
```

### Template Function Generic::String::split

- Defined in file\_src\_Generic\_String\_Split.h

#### Function Documentation

```
template<typename Out>
void Generic::String::split(std::string const &s, char delim, Out result)
```

### Function Generic::String::trim

- Defined in file\_src\_Generic\_String\_Trim.h

#### Function Documentation

```
void Generic::String::trim(std::string &io_s_)
```

### Function Generic::String::trimmed

- Defined in file\_src\_Generic\_String\_Trim.h

**Function Documentation**

```
std::string Generic::String::trimmed(std::string s_)
```

## 8.3.5 Variables

**Variable Core::Common::Geometry::equatorRadiusKiloMeter**

- Defined in file\_src\_Core\_Common\_Geometry\_NumericConstants.h

**Variable Documentation**

```
constexpr double Core::Common::Geometry::equatorRadiusKiloMeter = 6378.388
```

**Variable Core::Common::Geometry::equatorRadiusMeter**

- Defined in file\_src\_Core\_Common\_Geometry\_NumericConstants.h

**Variable Documentation**

```
constexpr double Core::Common::Geometry::equatorRadiusMeter = equatorRadiusKiloMeter * 1000.0
```

**Variable Core::Common::Geometry::pi**

- Defined in file\_src\_Core\_Common\_Geometry\_NumericConstants.h

**Variable Documentation**

```
constexpr double Core::Common::Geometry::pi = 3.1415926535897932384626433832795
```

**Variable Core::Common::Geometry::pover180**

- Defined in file\_src\_Core\_Common\_Geometry\_NumericConstants.h

**Variable Documentation**

```
constexpr double Core::Common::Geometry::pover180 = pi / 180.0
```

### 8.3.6 Defines

#### Define `MAKE_HASHABLE`

- Defined in file\_src\_Generic\_Hash\_MakeHashable.h

#### Define Documentation

`MAKE_HASHABLE` (type, ...)

### 8.3.7 Typedefs

#### TypeDef anonymous\_namespace\_SamplingPointFinder.cpp::StreetIndexGeoindex

- Defined in file\_src\_AppComponents\_Common\_Filter\_SamplingPointFinder.cpp

#### TypeDef Documentation

`using anonymous_namespace_SamplingPointFinder.cpp::StreetIndexGeoindex = boost::geometry::`

#### TypeDef anonymous\_namespace\_SamplingPointFinder.cpp::StreetIndexGeoindexAlgorithm

- Defined in file\_src\_AppComponents\_Common\_Filter\_SamplingPointFinder.cpp

#### TypeDef Documentation

`using anonymous_namespace_SamplingPointFinder.cpp::StreetIndexGeoindexAlgorithm = boost::ge`

#### TypeDef anonymous\_namespace\_SamplingPointFinder.cpp::StreetIndexGeoindexGeometry

- Defined in file\_src\_AppComponents\_Common\_Filter\_SamplingPointFinder.cpp

#### TypeDef Documentation

`using anonymous_namespace_SamplingPointFinder.cpp::StreetIndexGeoindexGeometry = boost::ge`

#### TypeDef anonymous\_namespace\_SamplingPointFinder.cpp::StreetIndexGeoindexValue

- Defined in file\_src\_AppComponents\_Common\_Filter\_SamplingPointFinder.cpp

**Typedef Documentation**

```
using anonymous_namespace_SamplingPointFinder.cpp::StreetIndexGeoindexValue = std::pair<St
```

**Typedef AppComponents::Common::Filter::anonymous\_namespace\_OsmMapReader.cpp::PointGeoindex**

- Defined in file\_src\_AppComponents\_Common\_Filter\_OsmMapReader.cpp

**Typedef Documentation**

```
using AppComponents::Common::Filter::anonymous_namespace_OsmMapReader.cpp::PointGeoindex =
```

**Typedef AppComponents::Common::Filter::anonymous\_namespace\_OsmMapReader.cpp::PointGeoindexAlgorithm**

- Defined in file\_src\_AppComponents\_Common\_Filter\_OsmMapReader.cpp

**Typedef Documentation**

```
using AppComponents::Common::Filter::anonymous_namespace_OsmMapReader.cpp::PointGeoindexAlgorithm =
```

**Typedef AppComponents::Common::Filter::anonymous\_namespace\_OsmMapReader.cpp::PointGeoindexValue**

- Defined in file\_src\_AppComponents\_Common\_Filter\_OsmMapReader.cpp

**Typedef Documentation**

```
using AppComponents::Common::Filter::anonymous_namespace_OsmMapReader.cpp::PointGeoindexValue =
```

**Typedef AppComponents::Common::Filter::Routing::anonymous\_namespace\_SamplingPointRouter.cpp::ClusteredSamplingPoint**

- Defined in file\_src\_AppComponents\_Common\_Filter\_Routing\_SamplingPointRouter.cpp

**Typedef Documentation**

```
using AppComponents::Common::Filter::Routing::anonymous_namespace_SamplingPointRouter.cpp::
```

**Typedef AppComponents::Common::Filter::Routing::anonymous\_namespace\_SamplingPointRouter.cpp::SamplingPoint**

- Defined in file\_src\_AppComponents\_Common\_Filter\_Routing\_SamplingPointRouter.cpp

**Typedef Documentation**

```
using AppComponents::Common::Filter::Routing::anonymous_namespace_SamplingPointRouter.cpp:
```

**Typedef AppComponents::Common::Filter::SamplingPointSkipStrategy**

- Defined in file\_src\_AppComponents\_Common\_Filter\_Routing\_Types.h

**Typedef Documentation**

```
using AppComponents::Common::Filter::SamplingPointSkipStrategy = Generic::Skipper::Strategy
```

**Typedef AppComponents::Common::Types::Graph::Graph**

- Defined in file\_src\_AppComponents\_Common\_Types\_Graph\_Graph.h

**Typedef Documentation**

```
using AppComponents::Common::Types::Graph::Graph = Core::Graph::Graph
```

Represents the street graph.

**Typedef AppComponents::Common::Types::Graph::GraphEdgeMap**

- Defined in file\_src\_AppComponents\_Common\_Types\_Graph\_EdgeMap.h

**Typedef Documentation**

```
using AppComponents::Common::Types::Graph::GraphEdgeMap = std::unordered_map<Core::Graph::Edge, StreetEd
```

Maps graph edge to street segment.

**Typedef AppComponents::Common::Types::Graph::GraphTriple**

- Defined in file\_src\_AppComponents\_Common\_Types\_Graph\_EdgeMap.h

**Typedef Documentation**

```
using AppComponents::Common::Types::Graph::GraphTriple = std::tuple<Core::Graph::Node, Core::Graph::Edge,
```

**Typedef AppComponents::Common::Types::Graph::LemonDigraph**

- Defined in file\_src\_AppComponents\_Common\_Types\_Graph\_LemonDigraph.h

**Typedef Documentation**

**using** AppComponents::Common::Types::Graph::**LemonDigraph** = Core::Graph::*LemonDigraph*

**Typedef AppComponents::Common::Types::Graph::NodeMap**

- Defined in file\_src\_AppComponents\_Common\_Types\_Graph\_EdgeMap.h

**Typedef Documentation**

**using** AppComponents::Common::Types::Graph::**NodeMap** = std::unordered\_map<Core::Graph::*Node*, size\_t>  
Maps graph node to street junction.

**Typedef AppComponents::Common::Types::Graph::StreetIndexMap**

- Defined in file\_src\_AppComponents\_Common\_Types\_Graph\_EdgeMap.h

**Typedef Documentation**

**using** AppComponents::Common::Types::Graph::**StreetIndexMap** = std::unordered\_map<size\_t, *GraphTriplePair*>  
Maps street segment to graph edge.

**Typedef AppComponents::Common::Types::Routing::Edge**

- Defined in file\_src\_AppComponents\_Common\_Types\_Routing\_Edge.h

**Typedef Documentation**

**using** AppComponents::Common::Types::Routing::**Edge** = Core::Graph::*Edge*

**Typedef AppComponents::Common::Types::Routing::Node**

- Defined in file\_src\_AppComponents\_Common\_Types\_Routing\_Edge.h

## Typedef Documentation

```
using AppComponents::Common::Types::Routing::Node = Core::Graph::Node
```

## Typedef AppComponents::Common::Types::Routing::RouteList

- Defined in file\_src\_AppComponents\_Common\_Types\_Routing\_Edge.h

## Typedef Documentation

```
using AppComponents::Common::Types::Routing::RouteList = std::vector<std::shared_ptr<Route>>
```

## Typedef AppComponents::Common::Types::Routing::SamplingPointList

- Defined in file\_src\_AppComponents\_Common\_Types\_Routing\_SamplingPoint.h

## Typedef Documentation

```
using AppComponents::Common::Types::Routing::SamplingPointList = std::vector<SamplingPoint>
```

## Typedef AppComponents::Common::Types::Street::Highway

- Defined in file\_src\_AppComponents\_Common\_Types\_Street\_Highway.h

## Typedef Documentation

```
using AppComponents::Common::Types::Street::Highway = std::optional<HighwayType>
```

## Typedef AppComponents::Common::Types::Street::HighwayList

- Defined in file\_src\_AppComponents\_Common\_Types\_Street\_Highway.h

## Typedef Documentation

```
using AppComponents::Common::Types::Street::HighwayList = std::vector<Highway>
```

## Typedef AppComponents::Common::Types::Street::NodePair

- Defined in file\_src\_AppComponents\_Common\_Types\_Street\_NodePair.h

**Typedef Documentation**

**using** AppComponents::Common::Types::Street::**NodePair** = std::pair<size\_t, size\_t>  
Represents a junction between two street segments.

**Typedef AppComponents::Common::Types::Street::SegmentList**

- Defined in file\_src\_AppComponents\_Common\_Types\_Street\_Segment.h

**Typedef Documentation**

**using** AppComponents::Common::Types::Street::**SegmentList** = std::vector<*Segment*>

**Typedef AppComponents::Common::Types::Street::TravelDirectionList**

- Defined in file\_src\_AppComponents\_Common\_Types\_Street\_TravelDirection.h

**Typedef Documentation**

**using** AppComponents::Common::Types::Street::**TravelDirectionList** = std::vector<*TravelDirection*>

**Typedef AppComponents::Common::Types::Track::Altitude**

- Defined in file\_src\_AppComponents\_Common\_Types\_Track\_Altitude.h

**Typedef Documentation**

**using** AppComponents::Common::Types::Track::**Altitude** = double  
Represents the height above ground.

**Typedef AppComponents::Common::Types::Track::Heading**

- Defined in file\_src\_AppComponents\_Common\_Types\_Track\_Heading.h

**Typedef Documentation**

**using** AppComponents::Common::Types::Track::**Heading** = double  
Represents the heading (in degrees, 0 is north, clockwise direction).

### Typedef AppComponents::Common::Types::Track::Point

- Defined in file\_src\_AppComponents\_Common\_Types\_Track\_Point.h

#### Typedef Documentation

**using** AppComponents::Common::Types::Track::Point = Core::Common::Geometry::Point  
Represents a coordinate pair.

### Typedef AppComponents::Common::Types::Track::Time

- Defined in file\_src\_AppComponents\_Common\_Types\_Track\_Time.h

#### Typedef Documentation

**using** AppComponents::Common::Types::Track::Time = std::chrono::system\_clock::time\_point  
Represents the timestamp.

### Typedef AppComponents::Common::Types::Track::Velocity

- Defined in file\_src\_AppComponents\_Common\_Types\_Track\_Velocity.h

#### Typedef Documentation

**using** AppComponents::Common::Types::Track::Velocity = double  
Represents the velocity [m/s].

### Typedef Core::Common::Geometry::Box

- Defined in file\_src\_Core\_Common\_Geometry\_Types.h

#### Typedef Documentation

**using** Core::Common::Geometry::Box = boost::geometry::model::box<Point>  
Boxtyle. The sides of the box are parallel to the coordinate system axis.

### Typedef Core::Common::Geometry::CoordinateSystem

- Defined in file\_src\_Core\_Common\_Geometry\_Types.h

**Typedef Documentation**

```
using Core::Common::Geometry::CoordinateSystem = boost::geometry::cs::spherical_equatorial<boost::geometry::degree>;  
WGS84 in degrees.
```

**Typedef Core::Common::Geometry::LineString**

- Defined in file\_src\_Core\_Common\_Geometry\_Types.h

**Typedef Documentation**

```
using Core::Common::Geometry::LineString = boost::geometry::model::linestring<Point>;  
Line string type.
```

**Typedef Core::Common::Geometry::Polygon**

- Defined in file\_src\_Core\_Common\_Geometry\_Types.h

**Typedef Documentation**

```
using Core::Common::Geometry::Polygon = boost::geometry::model::ring<Point>;  
Polygontyp. A polygon cannot have holes.
```

**Typedef Core::Common::Geometry::Segment**

- Defined in file\_src\_Core\_Common\_Geometry\_Types.h

**Typedef Documentation**

```
using Core::Common::Geometry::Segment = boost::geometry::model::segment<Point>;  
Segment type.
```

**Typedef Core::Common::Geometry::ValueType**

- Defined in file\_src\_Core\_Common\_Geometry\_Types.h

**Typedef Documentation**

```
using Core::Common::Geometry::ValueType = double;  
Numeric basis.
```

**Typedef Core::Graph::Routing::CostFunction**

- Defined in file\_src\_Core\_Graph\_Routing\_Algorithm.h

**Typedef Documentation**

```
using Core::Graph::Routing::CostFunction = std::function<double (Core::Graph::Edge)>
```

**Typedef Core::Graph::Routing::FilterFunction**

- Defined in file\_src\_Core\_Graph\_Routing\_Algorithm.h

**Typedef Documentation**

```
using Core::Graph::Routing::FilterFunction = std::function<bool (PathView const&)>
```



## GLOSSARY

### 9.1 General

**farthest navigation route** The farthest route is a *valid navigation route* with its last *sampling point* index as high as possible.

**route** A route usually is the result of a map matching process and is snapped to the geometry of the streets on a *street map*.

**sampling point** A track point, which at least has one *candidate* on a *street segment*.

**sampling point candidate** A track point projected on a *street segment*. The projection has to meet specific requirements in order to be considered a candidate.

Candidates are ordered according to several criteria (see *Candidate search*).

**street graph** A *simple directed graph* representing the *street map*. The nodes and edges of the street graph are tightly coupled to the street map data. Makes the street map applicable to routing algorithms.

**street map** Refers to a data source with street map data or to the map itself as described by the data. A street map consists at least of a set of street segment geometries, connected by node points.

**street segment** A minimal street segment without intersections (nodes) inbetween. May consist of more than two points. The corresponding linestring of the *street map* may have been cutted to satisfy this requirement.

**track** Refers to a data source with track data or to the track itself as described by the data. A track consists at least of an ordered list of geo points (track points). The track might consist of additional data like velocity, heading, timestamp etc.

**valid navigation route** A valid navigation route is a consecutive *route* which you could navigate.

**Currently the following constraints are considered:**

- Allowed travel direction
- Existing street intersections
- Self intersections (optional)
- Velocity difference (optional; for street matching)
- Angular deviation (optional; for street matching)

## 9.2 Operational data details

See also:

*Type listing*

**sampling point candidate index** Index for *sampling point candidates*.

**sampling point index** Index for *sampling points*.

**street index** Shared index for *street map* data (f.ex. for SegmentList, NodePairList, TravelDirectionList).

**street segment index** Index of a sub segment (consisting of two points) of a *street segment*.

**track index** Shared index for *track* data (f.ex. for TimeList, PointList, HeadingList).

## SPECIFICATIONS

### 10.1 General

#### heading

**track point heading** The heading of a track point is the nautical or geographical azimuth, the clockwise in degrees measured angle between the north and the heading direction.

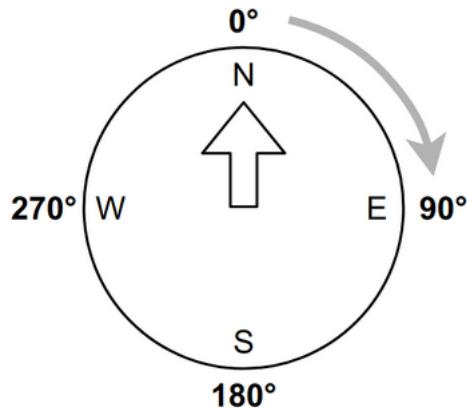


Fig. 10.1: Heading

**street segment origin ID** To a *street segment* corresponding ID from the original data source.

**street segment origin offset** It can be necessary to split a street given by a data source into multiple street segments, e.g. because of junctions. We explain the idea of the offset by the following example.

By splitting the original street segment 3 at the junctions with the original street segments 1 and 2 we created three street segments corresponding to the original street segment 3, all starting at a different offset.

In the end we have (street ID, offset) tuples. Such a tuple defines the beginning of a street segment. While the end of the street segment can be reconstructed implicit by adding the coordinate count of the street segment to the offset.

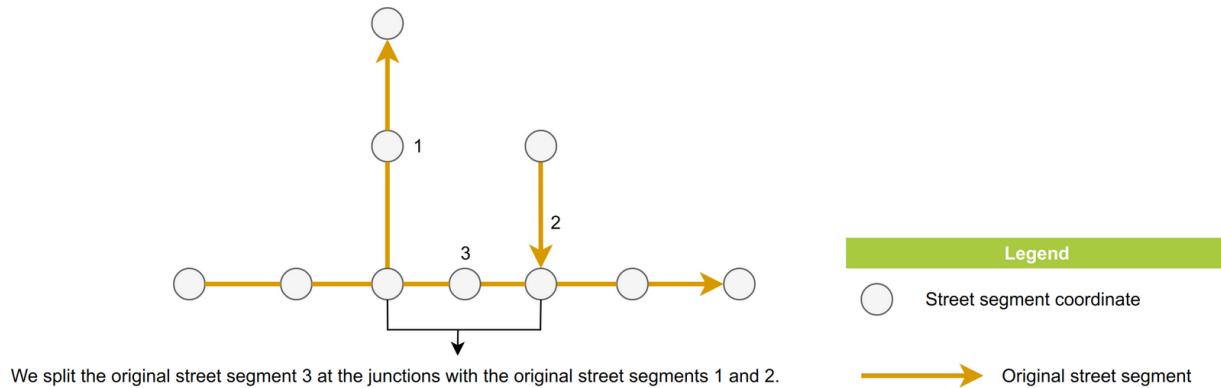


Fig. 10.2: Original street map

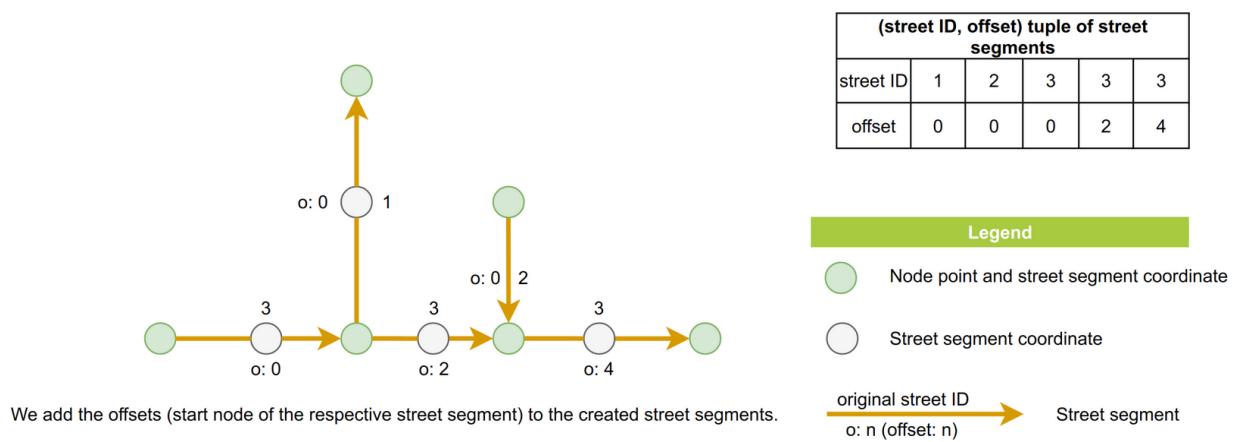


Fig. 10.3: Created street segments

## INDEX

A





(C++ member), 112  
AppComponents::Common::Filter::Routing::AppBpmgRmtsRdmonSamplingPointList::SkipRouter  
(C++ member), 112  
(C++ function), 113  
AppComponents::Common::Filter::Routing::AppBpmgRmtsRdmonSamplingPointRouterSkipRouter  
(C++ function), 111  
(C++ type), 111  
(C++ class), 114  
AppComponents::Common::Filter::Routing::AppBpmgRmtsSkpmtateFilter::SamplingPointFinder  
(C++ type), 153  
(C++ enumerator), 114  
AppComponents::Common::Filter::Routing::AppBpmgRmtsSdmnFilter::SamplingPointFinder  
(C++ class), 87  
(C++ enumerator), 114  
AppComponents::Common::Filter::Routing::AppBpmgRmtsSdmnFidperatSamplingPointFinder  
(C++ function), 87  
(C++ member), 115  
AppComponents::Common::Filter::Routing::AppBpmgRmtsSdmnFisrceSamplingPointFinder  
(C++ member), 87  
(C++ function), 115  
AppComponents::Common::Filter::Routing::AppBpmgRmtsSdmnFitagetSamplingPointFinder  
(C++ member), 87  
(C++ function), 115  
AppComponents::Common::Filter::Routing::AppComponents::Common::Filter::SamplingPointFinder  
(C++ enumerator), 126  
(C++ member), 115  
AppComponents::Common::Filter::Routing::AppComponents::Common::Filter::SamplingPointFinder  
(C++ class), 113  
(C++ enum), 114  
AppComponents::Common::Filter::Routing::AppComponents::Common::FilterSamplingPointFinder  
(C++ class), 87, 113  
(C++ member), 115  
AppComponents::Common::Filter::Routing::AppComponents::Common::FilterSamplingPointFinder  
(C++ member), 87, 113  
(C++ enumerator), 114  
AppComponents::Common::Filter::Routing::AppComponents::Common::FilterSamplingPointFinder  
(C++ member), 87, 113  
(C++ type), 153  
AppComponents::Common::Filter::Routing::AppComponents::Common::FilterTypes::Graph::GraphEdgeMap  
(C++ member), 113  
(C++ type), 153  
AppComponents::Common::Filter::Routing::AppComponents::Common::Types::Graph::GraphTriple  
(C++ function), 113  
(C++ type), 153  
AppComponents::Common::Filter::Routing::AppComponents::Common::Types::Graph::GraphTriplePa  
(C++ function), 113  
(C++ class), 88  
AppComponents::Common::Filter::Routing::AppComponents::Common::Types::Graph::GraphTriplePa  
(C++ member), 113  
(C++ member), 88  
AppComponents::Common::Filter::Routing::AppComponents::Common::SamplingRoifyType::Graph::GraphTriplePa  
(C++ member), 113  
(C++ member), 88  
AppComponents::Common::Filter::Routing::AppComponents::Common::Types::Graph::LemonDigraph  
(C++ class), 88, 113  
(C++ type), 154  
AppComponents::Common::Filter::Routing::AppComponents::Common::Types::Graph::NodeMap  
(C++ member), 88, 114  
(C++ type), 154  
AppComponents::Common::Filter::Routing::AppComponents::Common::Types::Graph::StreetEdge  
(C++ member), 88, 114  
(C++ class), 89  
AppComponents::Common::Filter::Routing::AppComponents::Common::Types::Graph::StreetEdge::fa  
(C++ member), 88, 114  
(C++ member), 89  
AppComponents::Common::Filter::Routing::AppComponents::Common::Types::Graph::StreetEdge::st  
(C++ function), 88, 114  
(C++ member), 89  
AppComponents::Common::Filter::Routing::AppComponents::Common::Types::Graph::StreetEdge::st  
(C++ member), 88, 114  
(C++ type), 154  
AppComponents::Common::Filter::Routing::AppComponents::Common::Types::Graph::SamBmngRginCahdekaedR  
(C++ member), 88, 114  
(C++ class), 89  
AppComponents::Common::Filter::Routing::AppComponents::Common::Types::Graph::SamBmngRginCahdekaedR  
(C++ member), 88, 114  
(C++ member), 89  
AppComponents::Common::Filter::Routing::AppComponents::Common::Types::Graph::CalculatedR



(C++ enum), 127  
AppComponents::Common::Types::Street::moAppComponents::Common::Types::Track::Velocity  
(C++ enumerator), 127  
AppComponents::Common::Types::Street::moAppComponents::Common::Types::Track::VelocityList  
(C++ enumerator), 127  
AppComponents::Common::Types::Street::NoAppComponents::ExampleMatcher::Filter::CsvTrackReader  
(C++ type), 156  
AppComponents::Common::Types::Street::NoAppComponents::ExampleMatcher::Filter::CsvTrackReader  
(C++ class), 93  
AppComponents::Common::Types::Street::NoAppComponents::ExampleMatcher::Filter::CsvTrackReader  
(C++ function), 115  
AppComponents::Common::Types::Street::NoAppComponents::ExampleMatcher::Filter::CsvTrackReader  
(C++ member), 116  
AppComponents::Common::Types::Street::NoAppComponents::ExampleMatcher::Filter::CsvTrackReader  
(C++ enumerator), 127  
AppComponents::Common::Types::Street::NoAppComponents::ExampleMatcher::Filter::CsvTrackReader  
(C++ function), 115  
AppComponents::Common::Types::Street::NoAppComponents::ExampleMatcher::Filter::JsonTrackReader  
(C++ enumerator), 127  
AppComponents::Common::Types::Street::NoAppComponents::ExampleMatcher::Filter::JsonTrackReader  
(C++ member), 116  
AppComponents::Common::Types::Street::NoAppComponents::ExampleMatcher::Filter::JsonTrackReader  
(C++ enumerator), 127  
AppComponents::Common::Types::Street::NoAppComponents::ExampleMatcher::Filter::JsonTrackReader  
(C++ function), 116  
AppComponents::Common::Types::Street::SegmentGeometryExampleMatcher::Filter::JsonTrackReader  
(C++ member), 93  
AppComponents::Common::Types::Street::Segment::originId  
(C++ member), 93

**C**

AppComponents::Common::Types::Street::Segment::commonOffset  
(C++ member), 93  
AppComponents::Common::Types::Street::Segment::angleBetweenSegments  
(C++ type), 156  
AppComponents::Common::Types::Street::Segment::angleBetweenSegments  
(C++ function), 137  
AppComponents::Common::Types::Street::tertiaryCommon::Geometry::Box (C++ type), 157  
(C++ enumerator), 127  
Core::Common::Geometry::buffer (C++ func-  
AppComponents::Common::Types::Street::tertiaryLink::link  
(C++ enumerator), 127  
Core::Common::Geometry::CoordinateSystem  
AppComponents::Common::Types::Street::TravelDirection  
(C++ enum), 127  
Core::Common::Geometry::degree (C++ func-  
AppComponents::Common::Types::Street::TravelDirectionList  
(C++ type), 158  
Core::Common::Geometry::equatorRadiusKiloMeter  
AppComponents::Common::Types::Street::trunk  
(C++ member), 150  
Core::Common::Geometry::equatorRadiusMeter  
AppComponents::Common::Types::Street::trunk\_link  
(C++ member), 150  
Core::Common::Geometry::flattened\_simple  
AppComponents::Common::Types::Track::Altitude  
(C++ type), 156  
Core::Common::Geometry::geoDistance  
AppComponents::Common::Types::Track::AltitudeList  
(C++ function), 139  
(C++ class), 94  
Core::Common::Geometry::geoLength (C++  
AppComponents::Common::Types::Track::Heading  
(C++ type), 156  
Core::Common::Geometry::heading (C++  
AppComponents::Common::Types::Track::HeadingList  
(C++ function), 140  
(C++ class), 94  
Core::Common::Geometry::headingDiff  
AppComponents::Common::Types::Track::Point  
(C++ type), 157  
Core::Common::Geometry::LineString (C++  
AppComponents::Common::Types::Track::PointList  
(C++ class), 95  
Core::Common::Geometry::LineStringProjectionResult  
AppComponents::Common::Types::Track::Time  
(C++ type), 157  
Core::Common::Geometry::LineStringProjectionResult  
AppComponents::Common::Types::Track::TimeList  
(C++ member), 96

Core::Common::Geometry::LineStringProjectCommon::GeometrySegment::Point  
     (C++ member), 96  
     (C++ function), 96  
 Core::Common::Geometry::LineStringProjectCommon::GeometryPoint::setLat  
     (C++ member), 96  
     (C++ function), 96  
 Core::Common::Geometry::LineStringProjectCommon::GeometryPoint::setLon  
     (C++ member), 96  
     (C++ function), 96  
 Core::Common::Geometry::LineStringProjectCommon::GeometryPoint::to\_string  
     (C++ member), 96  
     (C++ function), 97  
 Core::Common::Geometry::normalizeAngle   Core::Common::Geometry::Polygon   (C++  
     (C++ function), 140                                 type), 158  
 Core::Common::Geometry::operator""\_lat   Core::Common::Geometry::project   (C++  
     (C++ function), 141                                 function), 141  
 Core::Common::Geometry::operator""\_lon   Core::Common::Geometry::projectOntoLineString  
     (C++ function), 141                                 (C++ function), 142  
 Core::Common::Geometry::operator==(C++   Core::Common::Geometry::rad (C++ function),  
     function), 141   142  
 Core::Common::Geometry::pi (C++ member),   Core::Common::Geometry::relativeDistanceAlongLineSt  
     150   (C++ function), 142  
 Core::Common::Geometry::pi\_over180 (C++   Core::Common::Geometry::reversedHeading  
     member), 150   (C++ function), 142  
 Core::Common::Geometry::Point (C++ class),   Core::Common::Geometry::reverseHaversine  
     96   (C++ function), 143  
 Core::Common::Geometry::Point::~Point   Core::Common::Geometry::Segment   (C++  
     (C++ function), 96                                     type), 158  
 Core::Common::Geometry::Point::lat (C++   Core::Common::Geometry::snap (C++ func-  
     function), 96   tion), 143  
 Core::Common::Geometry::Point::Latitude   Core::Common::Geometry::toGeoJson (C++  
     (C++ class), 97, 98                                 function), 143  
 Core::Common::Geometry::Point::Latitude::operator<< (C++ function), 144  
     (C++ function), 97, 98  
 Core::Common::Geometry::Point::Latitude::operator>> (C++ function), 144  
     (C++ function), 97, 98  
 Core::Common::Geometry::Point::Latitude::operator<< (C++ function), 144, 145  
     (C++ function), 97, 98  
 Core::Common::Geometry::Point::Latitude::operator>> (C++ function), 144, 145  
     (C++ function), 97, 98  
 Core::Common::Geometry::Point::latitude   Core::Common::Geometry::ValueType (C++  
     (C++ member), 97                                     type), 158  
 Core::Common::Geometry::Point::lon (C++   Core::Common::Geometry::within (C++ func-  
     function), 96   tion), 145  
 Core::Common::Geometry::Point::Longitude   Core::Common::Postgres::Connection (C++  
     (C++ class), 97, 98                                 class), 116  
 Core::Common::Geometry::Point::Longitude::operator<< (C++ function), 117  
     (C++ function), 97, 98  
 Core::Common::Geometry::Point::Longitude::operator>> (C++ function), 117  
     (C++ function), 97, 98  
 Core::Common::Geometry::Point::Longitude::operator<< (C++ member), 117  
     (C++ function), 97, 98  
 Core::Common::Geometry::Point::Longitude::operator>> (C++ member), 117  
     (C++ function), 97, 99  
 Core::Common::Geometry::Point::longitude   Core::Common::Postgres::Connection::dbUser\_  
     (C++ member), 97                                     (C++ member), 117  
 Core::Common::Geometry::Point::operator=(C++   Core::Common::Postgres::Connection::ensureConnection  
     function), 96   (C++ function), 117

Core::Common::Postgres::Connection::getConnectGraph::Graph::remove (*C++ function*, 118)  
Core::Common::Postgres::Connection::globaleckGraph::Graph::source (*C++ function*, 118)  
Core::Common::Postgres::Connection::globaleckGraph::Graph::target (*C++ function*, 118)  
Core::Common::Postgres::Connection::hostCore::Graph::LemonDigraph (*C++ class*, 119)  
Core::Common::Postgres::Connection::hostCore::Graph::LemonDigraph::addEdge (*C++ member*, 119)  
Core::Common::Postgres::Connection::local (*C++ enumerator*, 117)  
Core::Common::Postgres::Connection::local (*C++ function*, 119)  
Core::Common::Postgres::Connection::mutex\_ (*C++ member*, 117)  
Core::Common::Postgres::Connection::mutex\_ (*C++ function*, 119)  
Core::Common::Postgres::Connection::port\_ (*C++ member*, 117)  
Core::Common::Postgres::Connection::port\_ (*C++ function*, 119)  
Core::Common::Postgres::Connection::Strategy (*C++ enum*, 117)  
Core::Common::Postgres::Connection::Strategy (*C++ function*, 119)  
Core::Common::Postgres::Connection::strategy\_ (*C++ member*, 117)  
Core::Common::Postgres::Connection::strategy\_ (*C++ function*, 119)  
Core::Common::Postgres::getOptional (*C++ function*, 145, 146)  
Core::Common::Time::fromIsoString (*C++ function*, 146)  
Core::Common::Time::fromIsoZString (*C++ function*, 146)  
Core::Common::Time::fromString (*C++ function*, 146)  
Core::Common::Time::toIsoString (*C++ function*, 147)  
Core::Common::Time::toIsoZString (*C++ function*, 147)  
Core::Common::Time::toString (*C++ function*, 147)  
Core::Graph::Edge (*C++ class*, 99)  
Core::Graph::Edge::Edge (*C++ function*, 99)  
Core::Graph::Edge::id (*C++ function*, 99)  
Core::Graph::Edge::id\_ (*C++ member*, 99)  
Core::Graph::Edge::operator== (*C++ function*, 99)  
Core::Graph::Graph (*C++ class*, 118)  
Core::Graph::Graph::~Graph (*C++ function*, 118)  
Core::Graph::Graph::addEdge (*C++ function*, 118)  
Core::Graph::Graph::createNode (*C++ function*, 118)  
Core::Graph::Graph::has (*C++ function*, 118)  
Core::Graph::Graph::inEdges (*C++ function*, 118)  
Core::Graph::Graph::newEdge (*C++ function*, 118)  
Core::Graph::Graph::newNode (*C++ function*, 118)  
Core::Graph::Graph::outEdges (*C++ function*, 118)  
Core::Graph::Graph::remove (*C++ function*, 118)  
Core::Graph::Graph::source (*C++ function*, 118)  
Core::Graph::Graph::target (*C++ function*, 118)  
Core::Graph::Graph::graph\_ (*C++ member*, 119)  
Core::Graph::Graph::graph\_ (*C++ function*, 119)  
Core::Graph::Graph::has (*C++ function*, 119)  
Core::Graph::Graph::inEdges (*C++ function*, 119)  
Core::Graph::Graph::outEdges (*C++ function*, 119)  
Core::Graph::Graph::remove (*C++ function*, 119)  
Core::Graph::LemonDigraph::source (*C++ function*, 119)  
Core::Graph::LemonDigraph::target (*C++ function*, 119)  
Core::Graph::Node (*C++ class*, 100)  
Core::Graph::Node::id (*C++ function*, 100)  
Core::Graph::Node::id\_ (*C++ member*, 100)  
Core::Graph::Node::Node (*C++ function*, 100)  
Core::Graph::Node::operator== (*C++ function*, 100)  
Core::Graph::Routing::CostFunction (*C++ type*, 159)  
Core::Graph::Routing::Dijkstra (*C++ class*, 120)  
Core::Graph::Routing::Dijkstra::Container (*C++ type*, 120)  
Core::Graph::Routing::Dijkstra::Dijkstra (*C++ function*, 120)  
Core::Graph::Routing::Dijkstra::explore (*C++ function*, 120)  
Core::Graph::Routing::Dijkstra::Frontier (*C++ type*, 120)  
Core::Graph::Routing::Dijkstra::frontier\_ (*C++ member*, 121)  
Core::Graph::Routing::Dijkstra::graph\_ (*C++ member*, 121)

Core::Graph::Routing::Dijkstra::GreaterFunction (C++ member), 121  
 Core::Graph::Routing::Dijkstra::GreaterFunction (C++ type), 120  
 Core::Graph::Routing::Dijkstra::init (C++ function), 120  
 Core::Graph::Routing::Dijkstra::operator< (C++ function), 120  
 Core::Graph::Routing::Dijkstra::PathNodeCore::Graph::Routing::PathViewImpl (C++ class), 121  
 Core::Graph::Routing::Dijkstra::PathNodeCore::Graph::Routing::PathViewImpl (C++ function), 121, 122  
 Core::Graph::Routing::Dijkstra::PathNodeCore::Graph::Routing::PathViewImpl::cost (C++ member), 121, 122  
 Core::Graph::Routing::Dijkstra::PathNodeCore::Graph::Routing::PathViewImpl::edge (C++ function), 121, 122  
 Core::Graph::Routing::Dijkstra::PathNodeCore::Graph::Routing::PathViewImpl::previous (C++ member), 121, 122  
 Core::Graph::Routing::Dijkstra::PathNodeCore::PathNodeIterator (C++ function), 121, 122  
 Core::Graph::Routing::Dijkstra::PathNodeCore::PathViewIterator (C++ class), 124  
 Core::Graph::Routing::Dijkstra::PathNodeCore::PathViewIterator::dereference (C++ function), 121, 122  
 Core::Graph::Routing::Dijkstra::PathNodeCore::PathViewIterator::equal (C++ member), 121, 122  
 Core::Graph::Routing::Dijkstra::reached (C++ function), 120  
 Core::Graph::Routing::Dijkstra::Visited (C++ type), 120  
 Core::Graph::Routing::Dijkstra::visited (C++ member), 121  
 Core::Graph::Routing::FilterFunction (C++ type), 159  
 Core::Graph::Routing::operator> (C++ function), 148  
 Core::Graph::Routing::operator<< (C++ function), 147  
 Core::Graph::Routing::PathView (C++ class), 122  
 Core::Graph::Routing::PathView::back (C++ function), 122  
 Core::Graph::Routing::PathView::begin (C++ function), 122  
 Core::Graph::Routing::PathView::cost (C++ function), 122  
 Core::Graph::Routing::PathView::edge (C++ function), 122  
 Core::Graph::Routing::PathView::empty (C++ function), 122  
 Core::Graph::Routing::PathView::end (C++ function), 122  
 Core::Graph::Routing::PathView::front (C++ function), 122  
 Core::Graph::Routing::PathView::impl\_ (C++ member), 123  
 Core::Graph::Routing::PathView::Iterator (C++ type), 122  
 Core::Graph::Routing::PathView::PathView (C++ class), 122  
 Core::Graph::Routing::PathView::PathView (C++ function), 122  
 Core::Graph::Routing::PathView::previous (C++ function), 122  
 Core::Graph::Routing::PathView::size (C++ function), 122  
 Core::Graph::Routing::PathViewImpl (C++ class), 123  
 Core::Graph::Routing::PathViewImpl (C++ function), 123  
 Core::Graph::Routing::PathViewImpl::cost (C++ function), 123  
 Core::Graph::Routing::PathViewImpl::edge (C++ function), 123  
 Core::Graph::Routing::PathViewImpl::previous (C++ function), 123  
 Core::Graph::Routing::PathViewIterator (C++ class), 124  
 Core::Graph::Routing::PathViewIterator::dereference (C++ function), 124  
 Core::Graph::Routing::PathViewIterator::equal (C++ function), 124  
 Core::Graph::Routing::PathViewIterator::increment (C++ function), 124  
 Core::Graph::Routing::PathViewIterator::path\_ (C++ member), 124  
 Core::Graph::Routing::PathViewIterator::PathViewIterator (C++ function), 124  
 Core::Graph::Routing::RoutingAlgorithm (C++ class), 124  
 Core::Graph::Routing::RoutingAlgorithm (C++ function), 125  
 Core::Graph::Routing::RoutingAlgorithm::~RoutingAlgorithm (C++ function), 125  
 Core::Graph::Routing::RoutingAlgorithm::costFunction (C++ member), 125  
 Core::Graph::Routing::RoutingAlgorithm::filterFunction (C++ member), 125  
 Core::Graph::Routing::RoutingAlgorithm::operator() (C++ function), 125  
 Core::Graph::Routing::RoutingAlgorithm::RoutingAlgorithm (C++ function), 125  
 Core::Graph::Routing::RoutingAlgorithm::run (C++ function), 125  
 Core::Graph::Routing::RoutingAlgorithm::setCost (C++ function), 125  
 Core::Graph::Routing::RoutingAlgorithm::setFilter (C++ function), 125

**F**

farthest navigation route, 161

**G**

Generic::flipMap (C++ function), 148  
 Generic::Progress (C++ class), 125

Generic::Progress::beginTime\_ (*C++ member*), 126  
Generic::Progress::Clock (*C++ type*), 125  
Generic::Progress::Duration (*C++ type*), 125  
Generic::Progress::output (*C++ function*), 125  
Generic::Progress::Progress (*C++ function*), 125  
Generic::Progress::text\_ (*C++ member*), 126  
Generic::Progress::tick (*C++ function*), 125  
Generic::Progress::ticksOccured\_ (*C++ member*), 126  
Generic::Progress::totalTicks\_ (*C++ member*), 126  
Generic::Progress::updateTotal (*C++ function*), 125  
Generic::String::ltrim (*C++ function*), 148  
Generic::String::ltrimmed (*C++ function*), 148  
Generic::String::readNextRow (*C++ function*), 148  
Generic::String::rtrim (*C++ function*), 149  
Generic::String::rtrimmed (*C++ function*), 149  
Generic::String::split (*C++ function*), 149  
Generic::String::trim (*C++ function*), 149  
Generic::String::trimmed (*C++ function*), 150

## H

heading, 163

## M

MAKE\_HASHABLE (*C macro*), 151

## R

route, 161

## S

sampling point, 161  
sampling point candidate, 161  
sampling point candidate index, 162  
sampling point index, 162  
std::hash::operator() (*C++ function*), 100, 101  
std::hash<Core::Graph::Edge> (*C++ class*), 100  
std::hash<Core::Graph::Node> (*C++ class*), 100  
street graph, 161  
street index, 162  
street map, 161  
street segment, 161  
street segment index, 162  
street segment origin ID, 163  
street segment origin offset, 163

## T

track, 161  
track index, 162  
track point heading, 163

## V

valid navigation route, 161