OS-Matcher

Fabian Sandoval Saldias, Julien Gout, Markus Abel

Jul 28, 2021

USAGE

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

CHAPTER

ONE

INSTALLATION

The *OS-Matcher* uses Conan for installing third-party libraries. 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 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 remove --force "$package/*" --src --builds --packages
    conan remove --force "$package/*" --system-reqs
    conan remove --force "$package/*" --remote ambrosys
    conan create conan/repositories/$package amb/stable --build $package --build␣
↪missing
    conan upload "$package/*" --confirm --all --remote ambrosys
done
```

1.4 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`.

CHAPTER

TWO

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
```

CHAPTER

THREE

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):

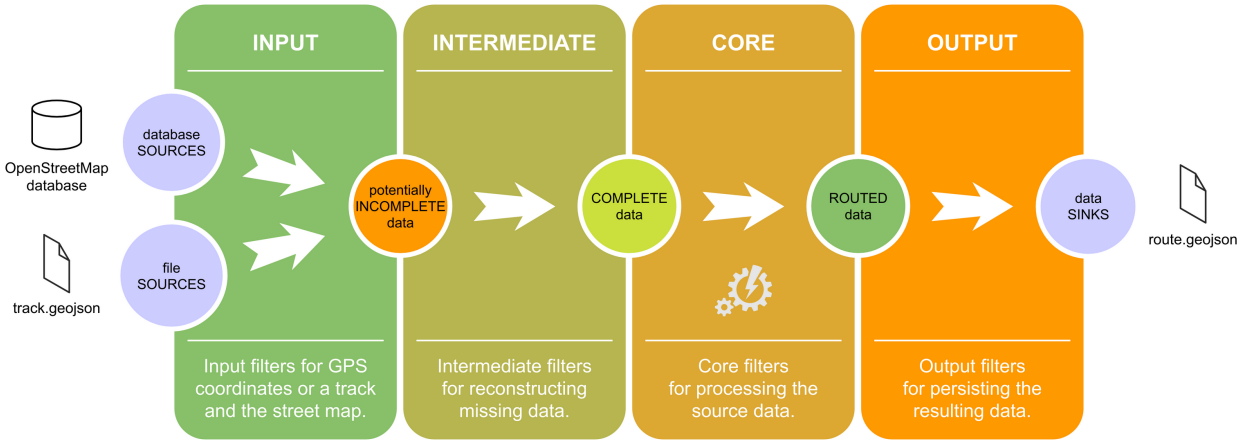


Fig. 3.1: Filter dataflow

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:

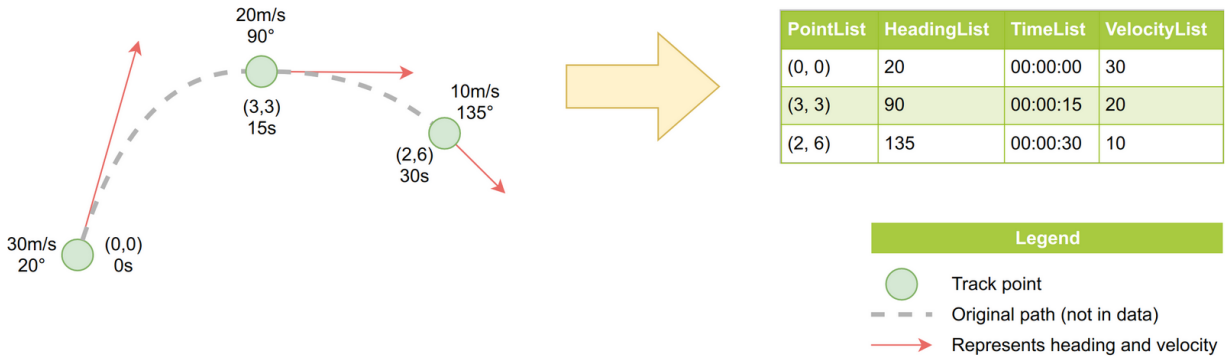


Fig. 3.2: Track data

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

`TimeList` (optional)

- Data type: *`TimeList`*

`VelocityList` (optional)

- Data type: *`VelocityList`*

Example skeleton:

```
class MyTrackReader : public AppComponents::Common::Filter::Filter
{
public:
    MyTrackReader( std::istream & input );
    bool operator()(
        AppComponents::Common::Types::Track::TimeList &,
        AppComponents::Common::Types::Track::PointList &,
        AppComponents::Common::Types::Track::HeadingList &,
        AppComponents::Common::Types::Track::VelocityList & );

private:
    std::istream & input_;
};


MyTrackReader::MyTrackReader( std::istream & input )
: Filter( "MyTrackReader" ), input_( input )
{
    setRequirements( {} );
    setOptionals( {} );
    setFulfillments( { "TimeList", "PointList", "HeadingList", "VelocityList" } );
}

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

(continues on next page)

(continued from previous page)

```
    APP_LOG_TAG( noise, "I/O" ) << "Reading track";

    // read `input_` filling `timeList`, `pointList`, `headingList` and `velocityList`

    return true;
}
```

Street Map

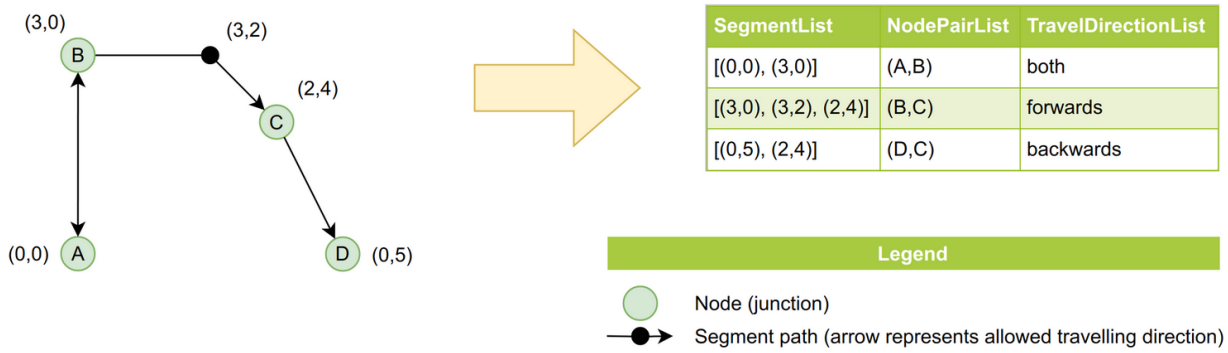


Fig. 3.3: Street map data

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:

```
class MyStreetMapReader : public AppComponents::Common::Filter::Filter
{
public:
    MyStreetMapReader( std::istream & input );
    bool operator()(
        Types::Street::SegmentList &,
        Types::Street::NodePairList &,
        Types::Street::TravelDirectionList & );

private:
    std::istream & input_;
};

```

(continues on next page)

(continued from previous page)

```
14
15  MyStreetMapReader::MyStreetMapReader( std::istream & input )
16  : Filter( "MyStreetMapReader" ), input_( input )
17  {
18      setRequirements( {} );
19      setOptionals( {} );
20      setFulfillments( { "SegmentList", "NodePairList", "TravelDirectionList" } );
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  }
```

CHAPTER

FOUR

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 end points 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.

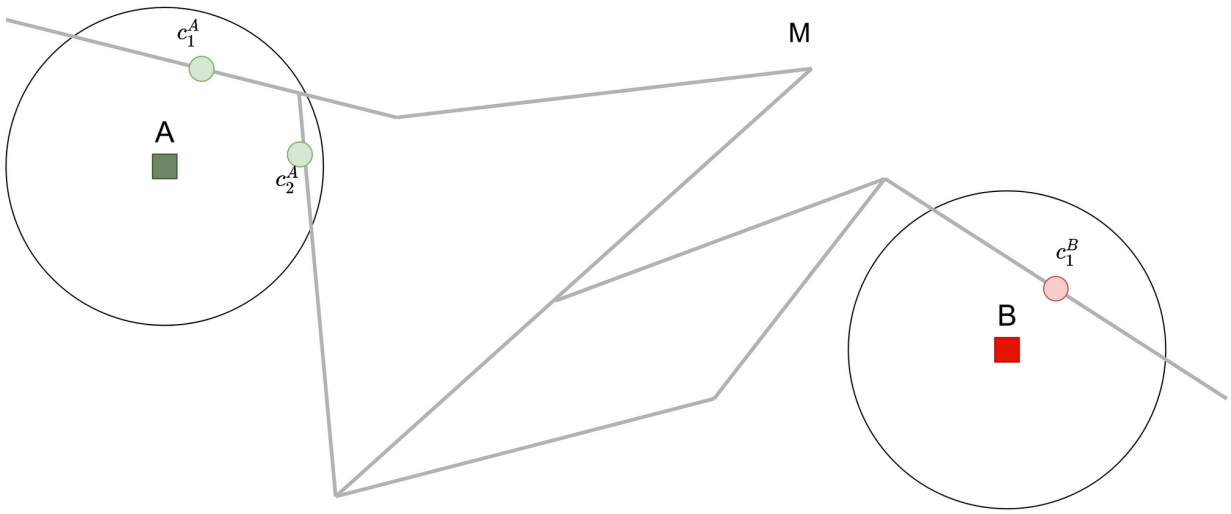


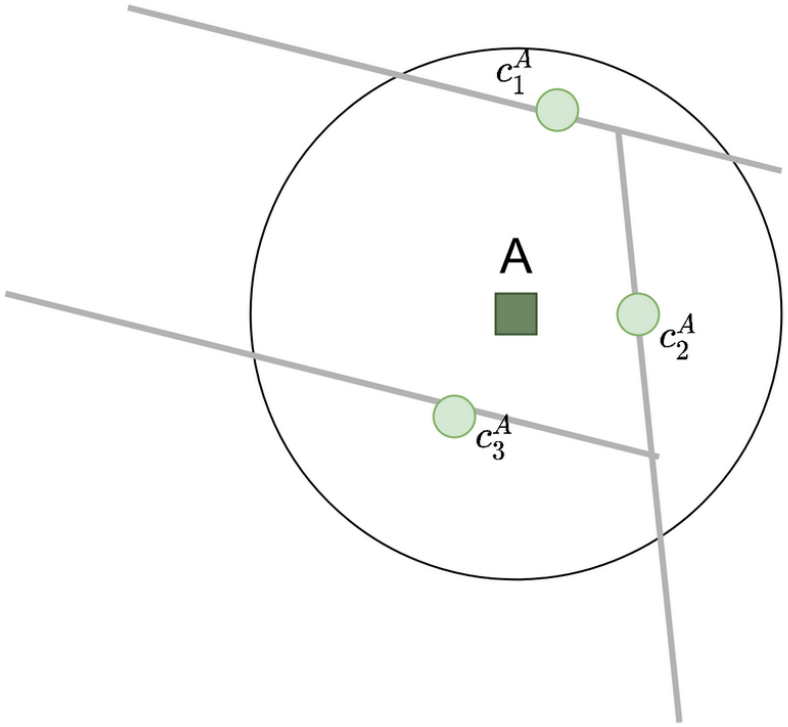
Fig. 4.1: Start and Endpoint with candidates

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.

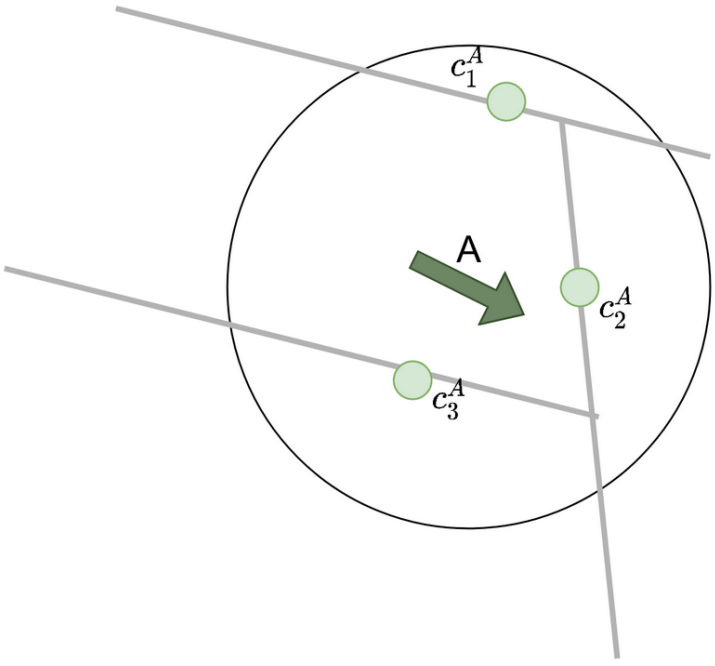
The track data may contain direction data as well, which is considered as the next decision criteria (using `function headingDifference()`). This would result in a distinct decision in our example.

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



Candidate	Dist to A
c_1^A	10
c_2^A	5
c_3^A	5
Order Undecided: c_3^A, c_2^A, c_1^A or c_2^A, c_3^A, c_1^A	

Fig. 4.2: Example for ambiguous candidates



Candidate	Dist to A	Heading Difference
c_1^A	10	15°
c_2^A	5	60°
c_3^A	5	15°
Order: c_3^A, c_2^A, c_1^A		

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)

CHAPTER

FIVE

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 may 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:

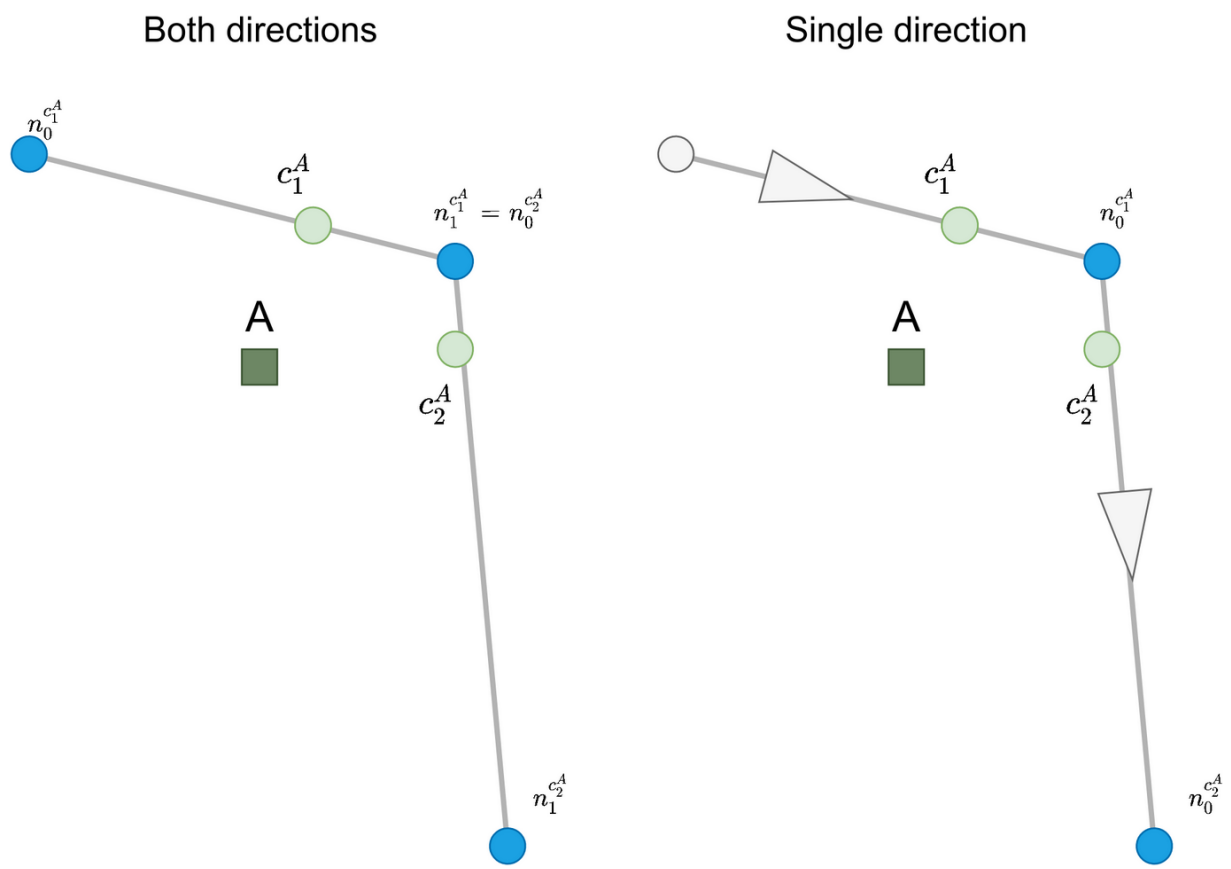


Fig. 5.1: Candidates on one way streets

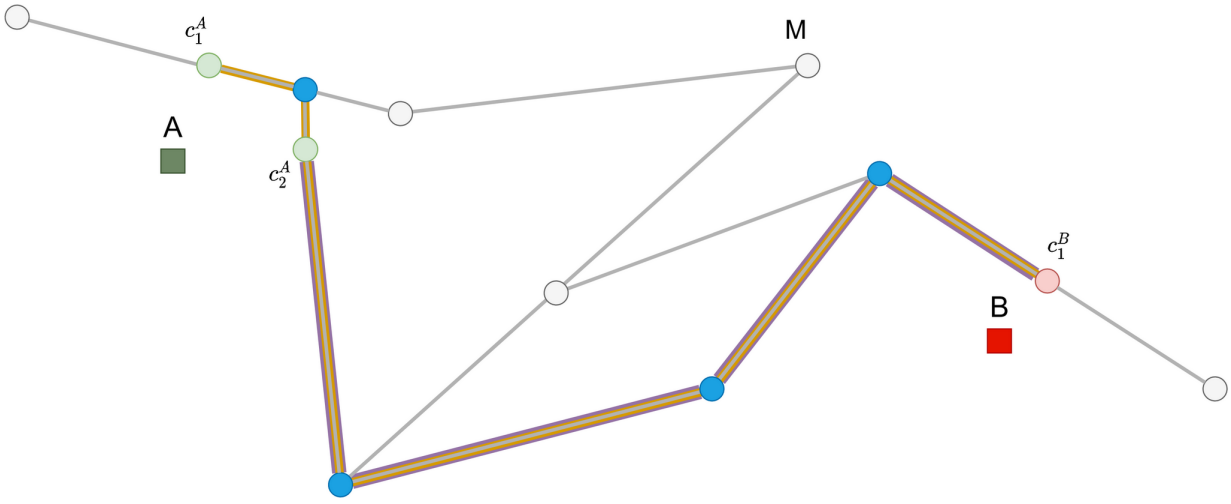


Fig. 5.2: Basically similar routes

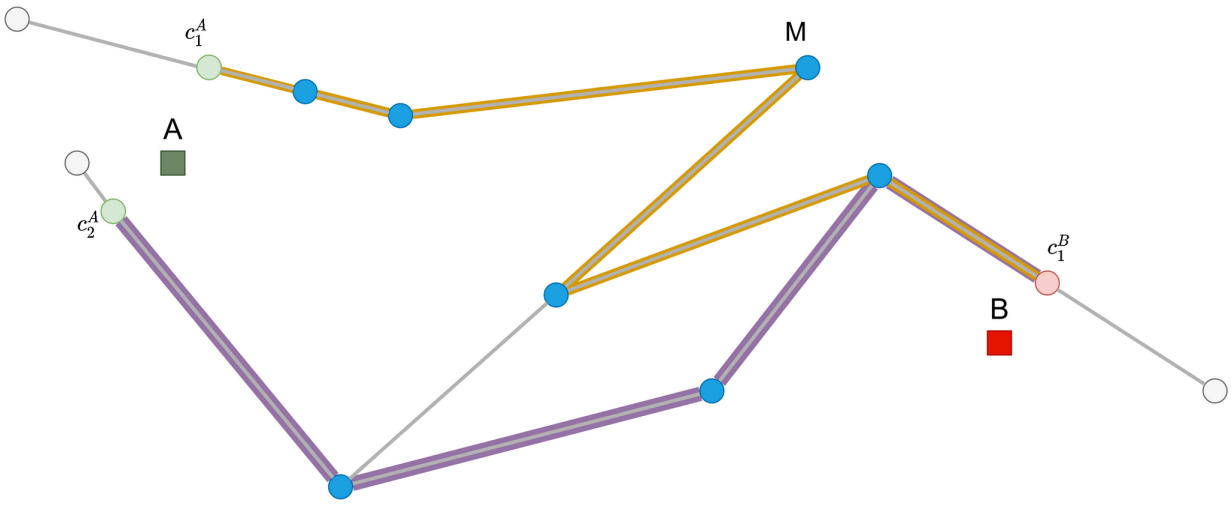


Fig. 5.3: Basically different routes.

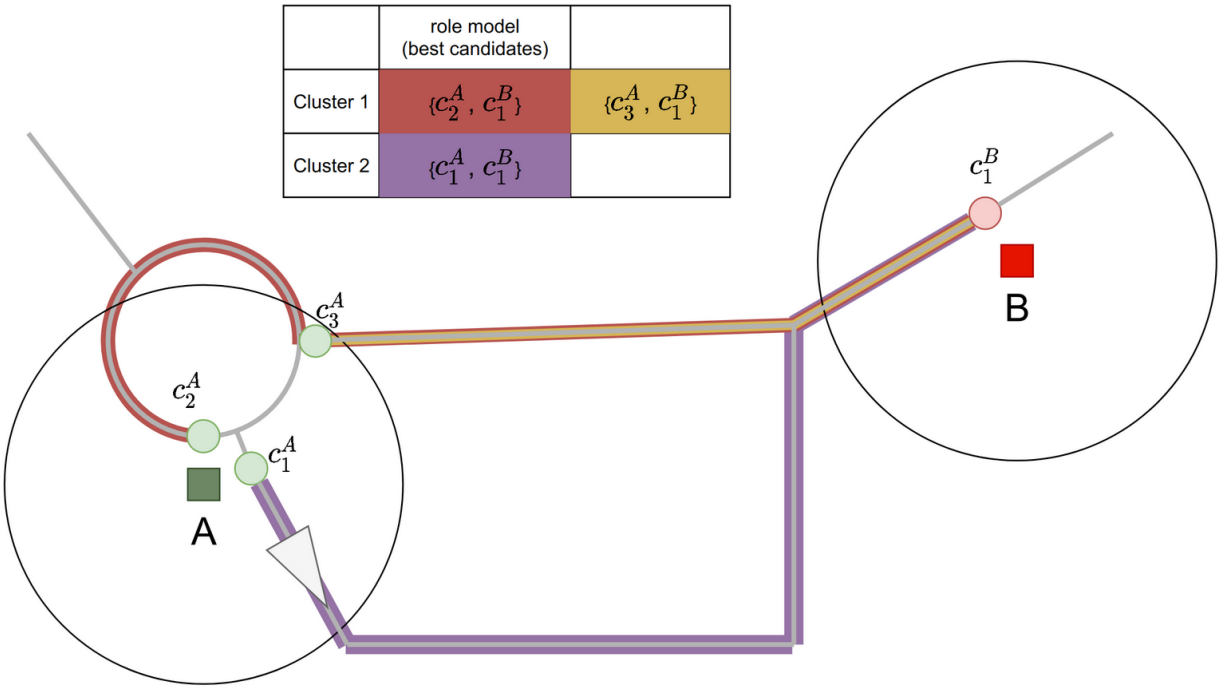


Fig. 5.4: Finding the most realistic route

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 $4 \cdot candidate\ search\ radius$)
- **the source node of one is contained by the other**, $n^s_{r_1} \in r_0$ or $n^s_{r_0} \in r_1$
- **the target node of one is contained by the other**, $n^t_{r_1} \in r_0$ or $n^t_{r_0} \in r_1$
- **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.

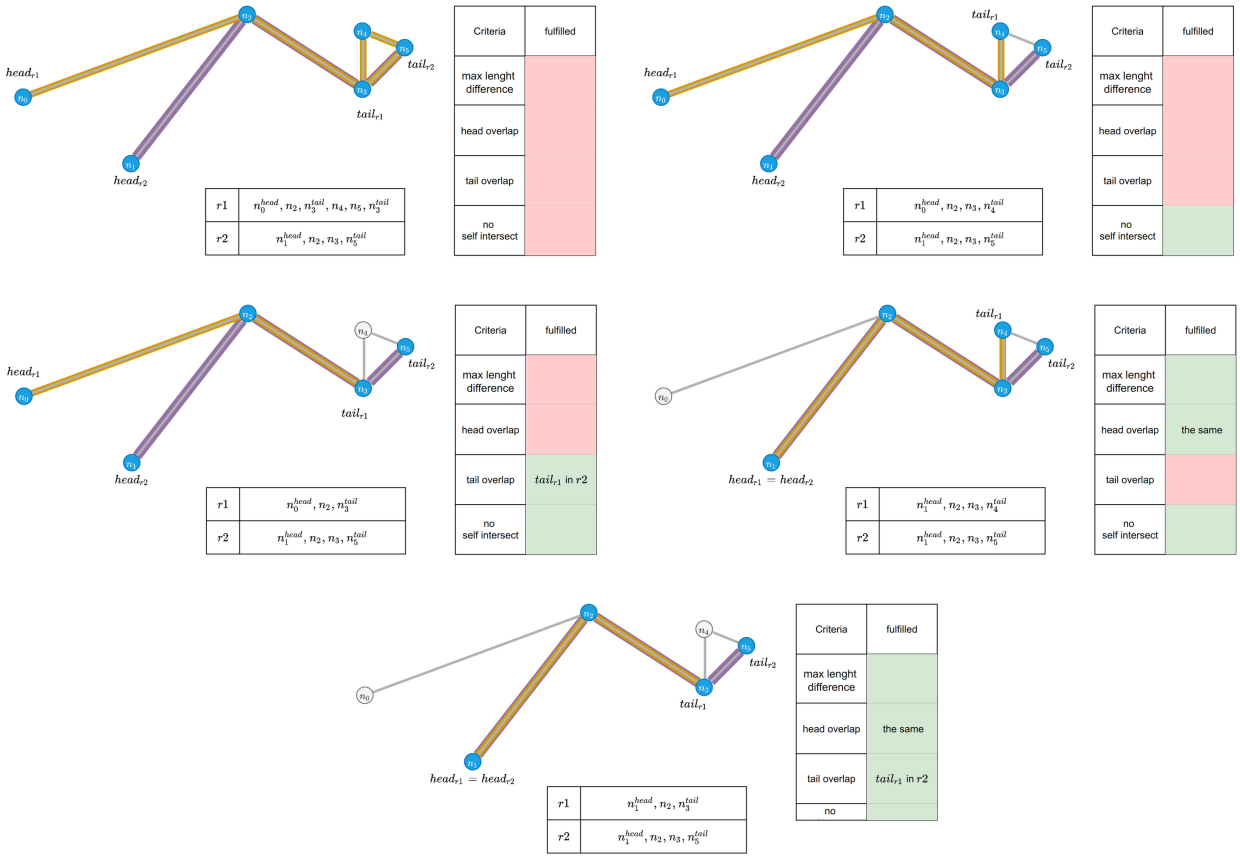


Fig. 5.5: Similarity criteria

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.

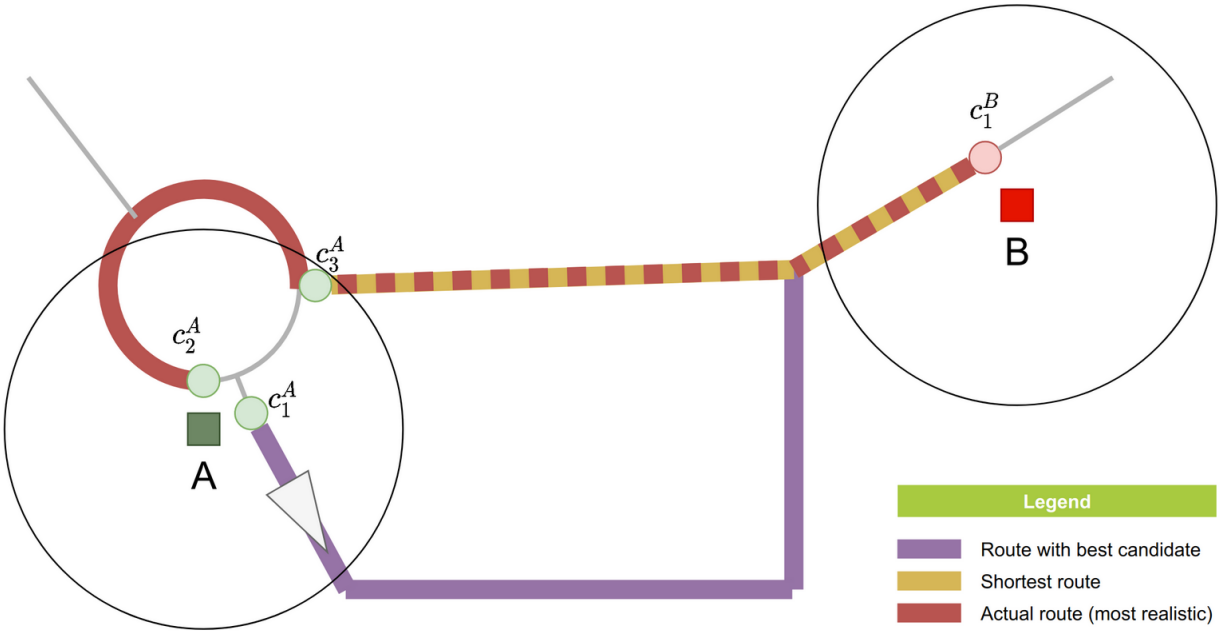


Fig. 5.6: Clustering

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.

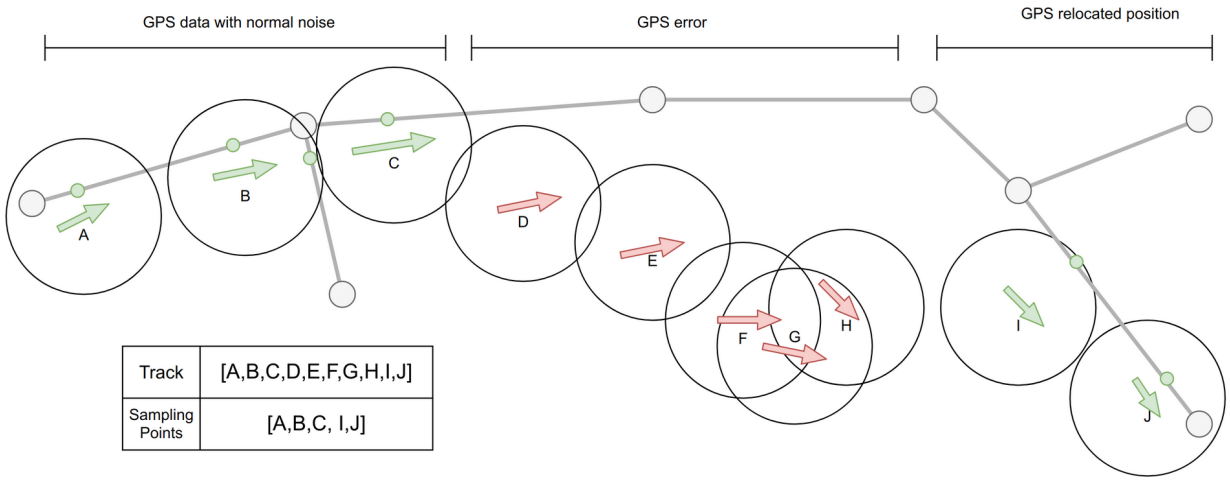


Fig. 5.7: Track to sample points

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.

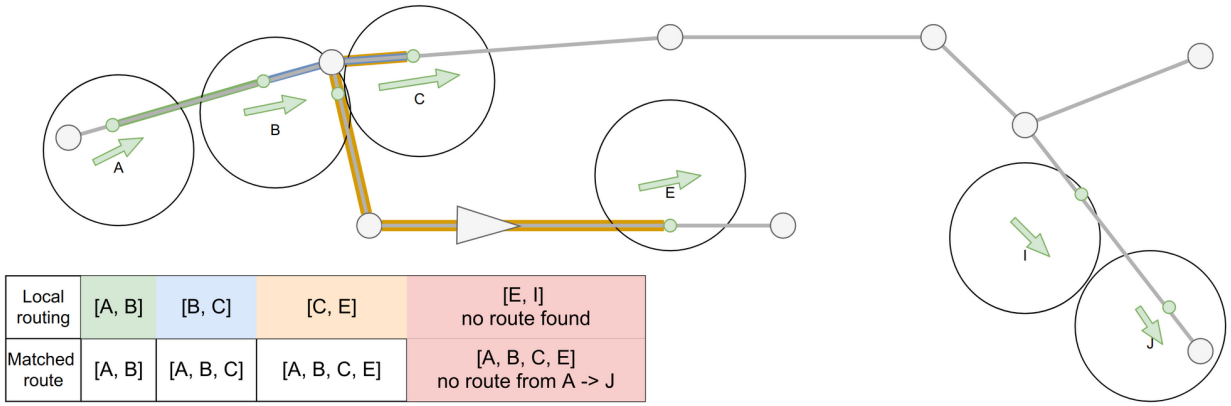


Fig. 5.8: Matching leads into dead end

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.

CHAPTER

SIX

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.

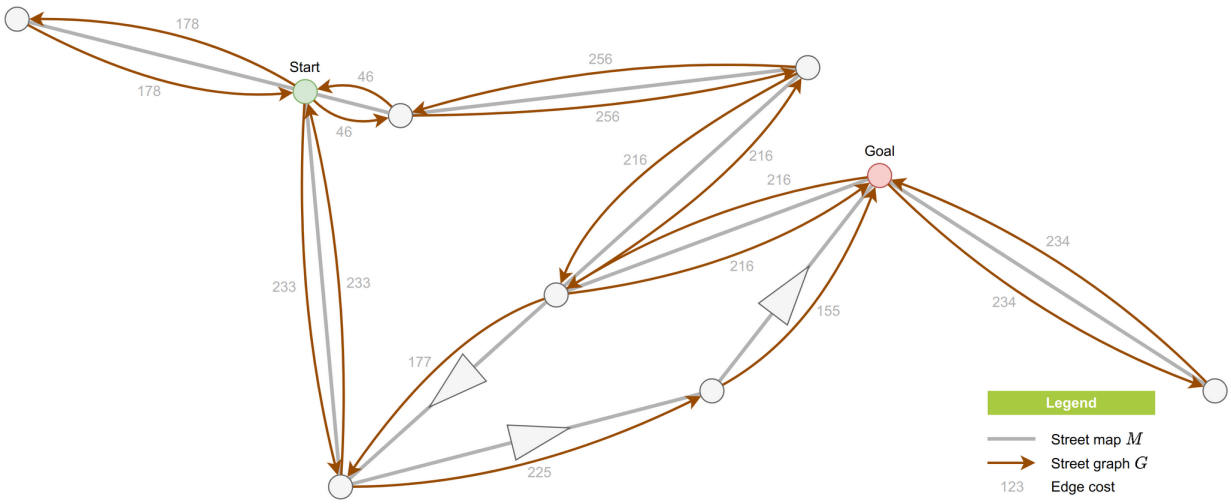


Fig. 6.1: Street map and street graph

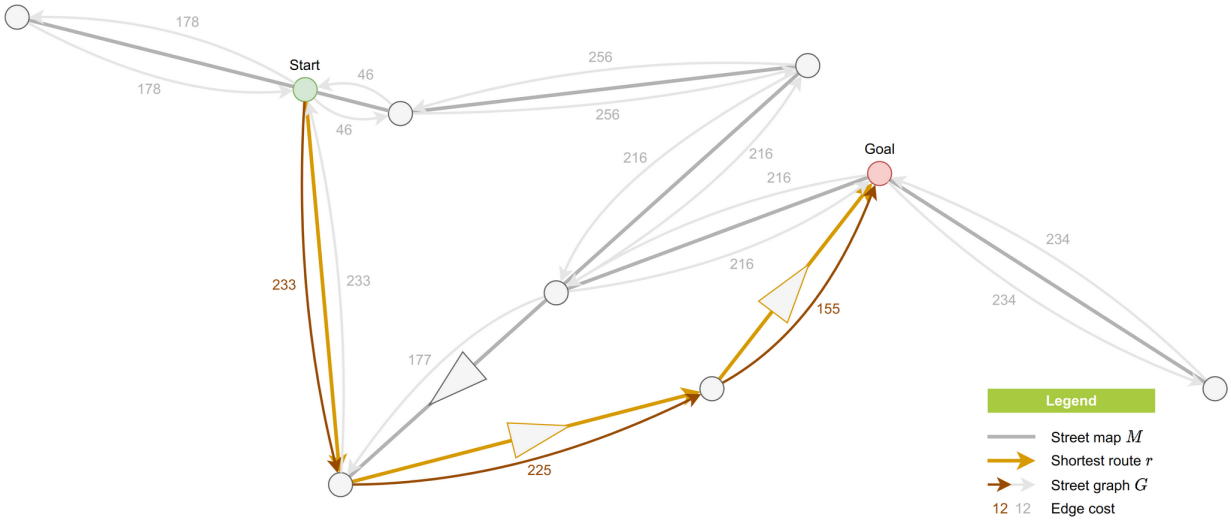


Fig. 6.2: Shortest route

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 choosen 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.

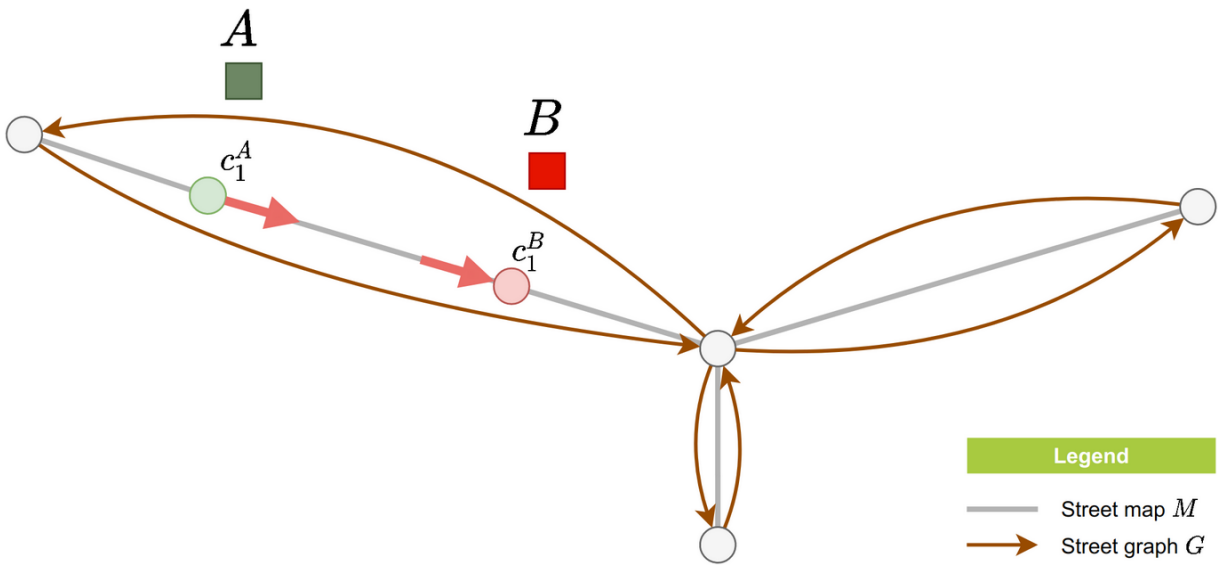


Fig. 6.3: Single edge

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.

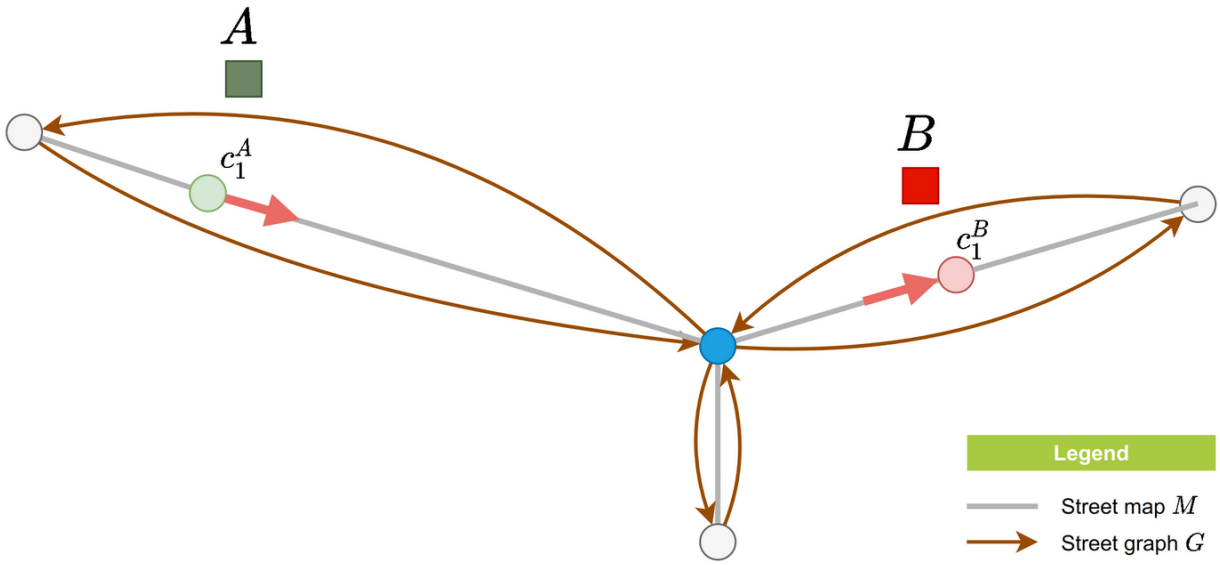


Fig. 6.4: Neighboring edge

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*.

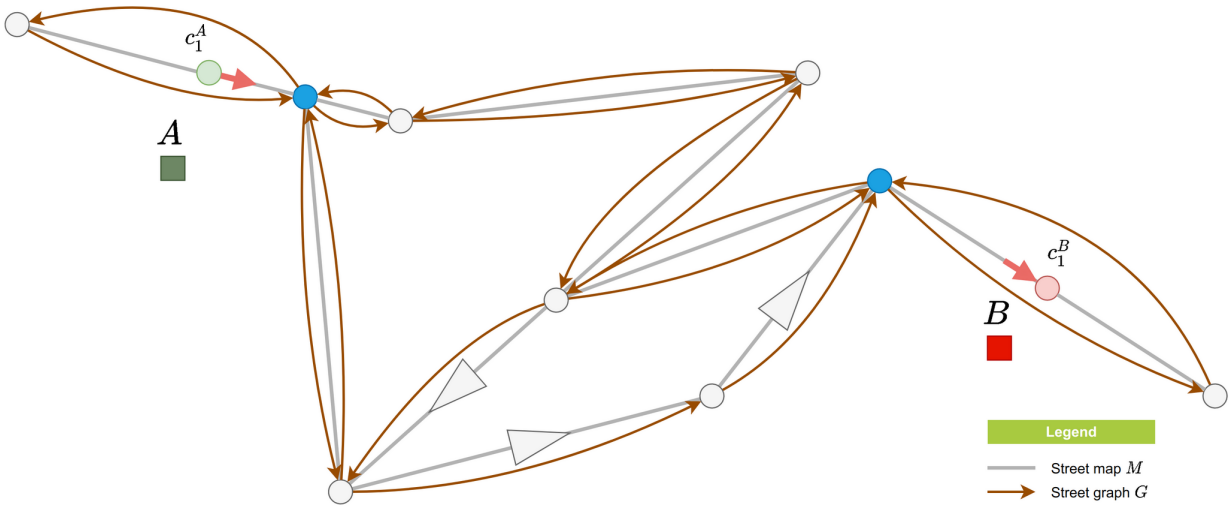


Fig. 6.5: Far edge

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.

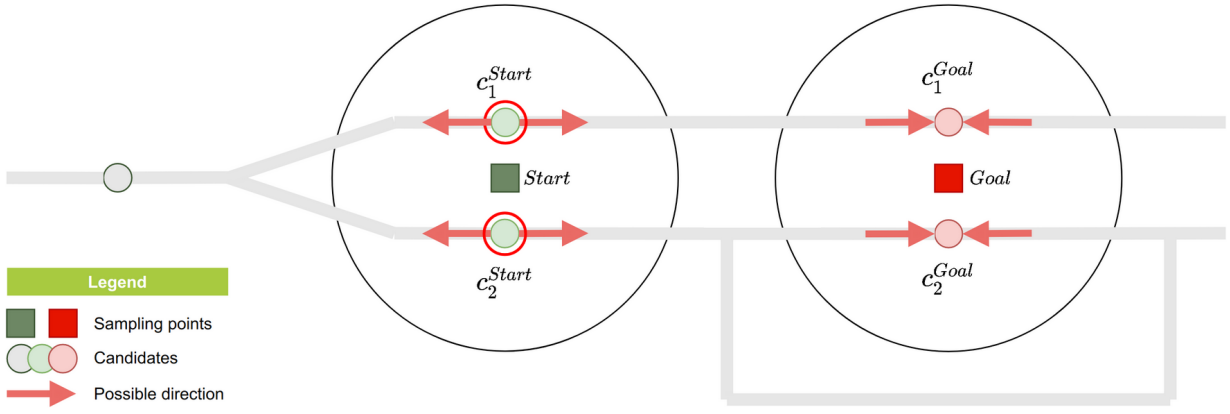


Fig. 6.6: Motivation

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:

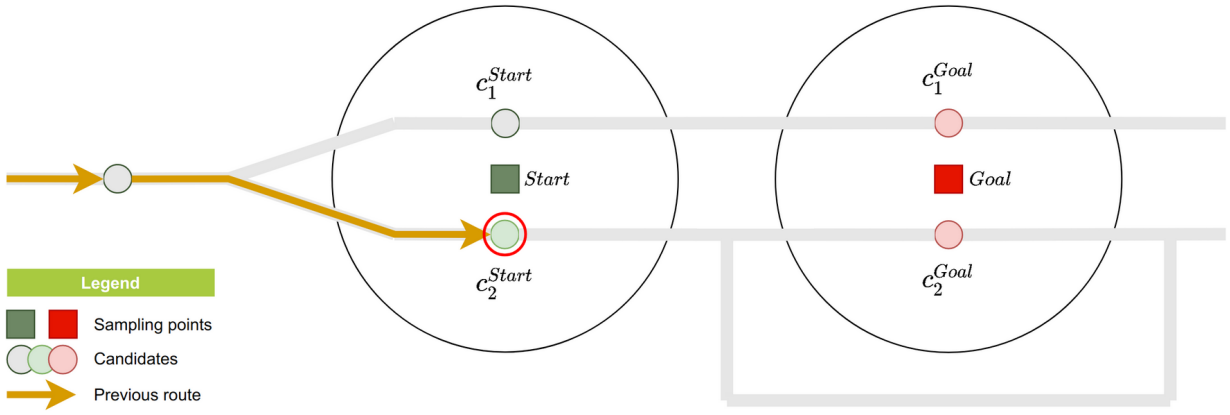


Fig. 6.7: Overview

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}.

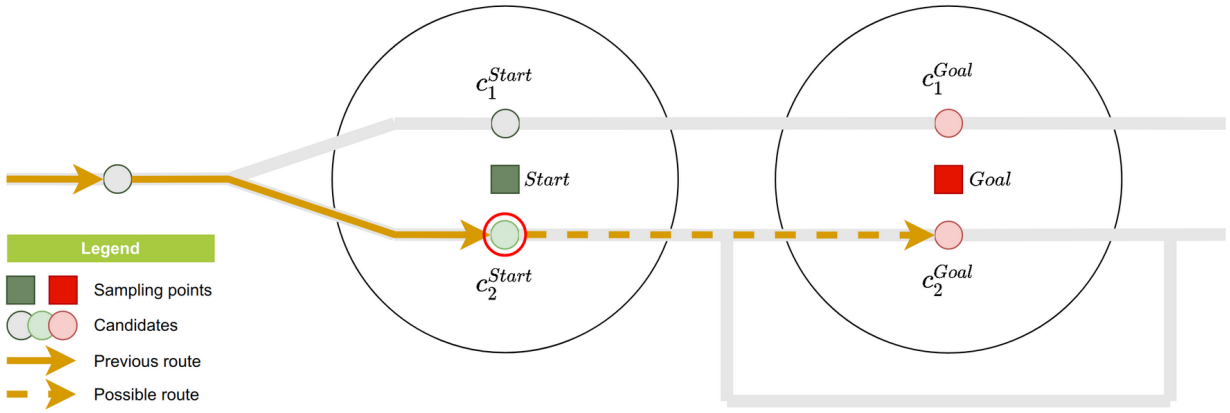


Fig. 6.8: Possible route 1

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

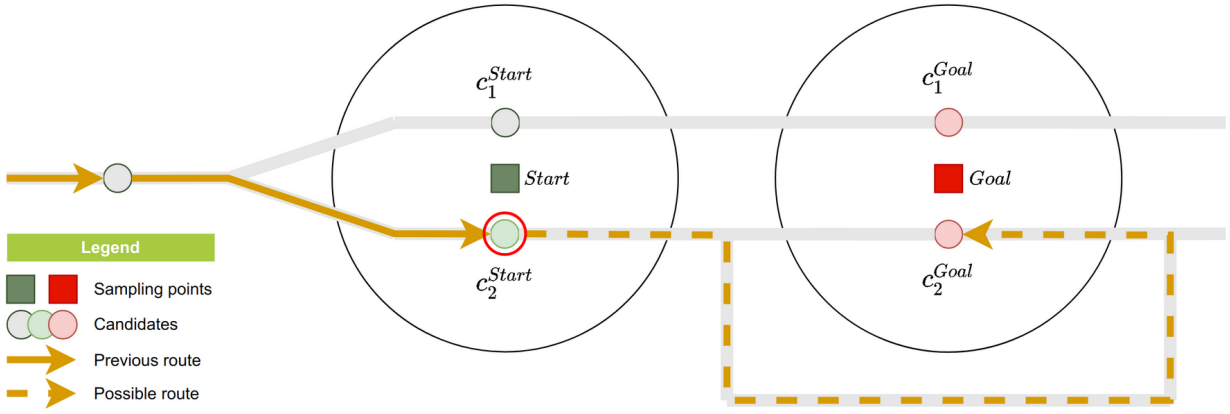


Fig. 6.9: Possible route 2

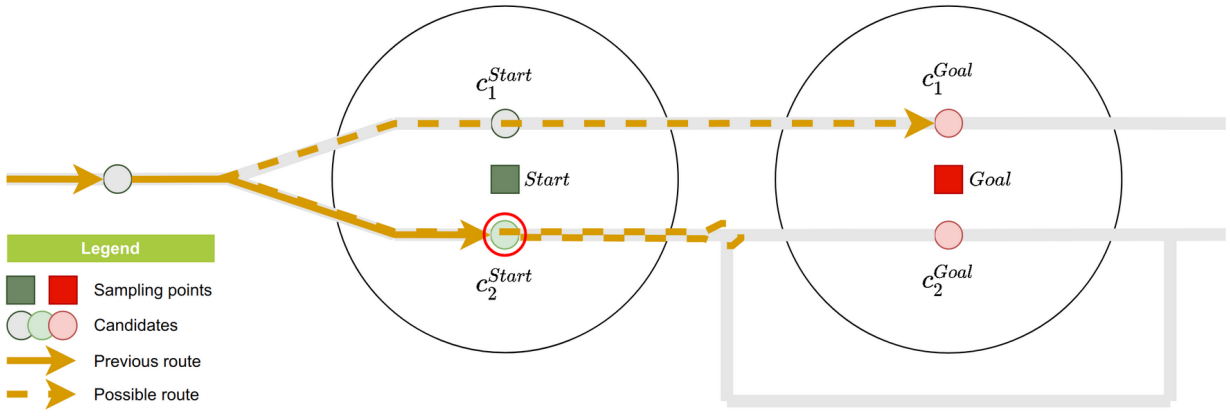


Fig. 6.10: Possible route 3

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 successful and it continues as usual. It then may start a new backtracking session, if necessary.

When backtracking is not successful, 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*).

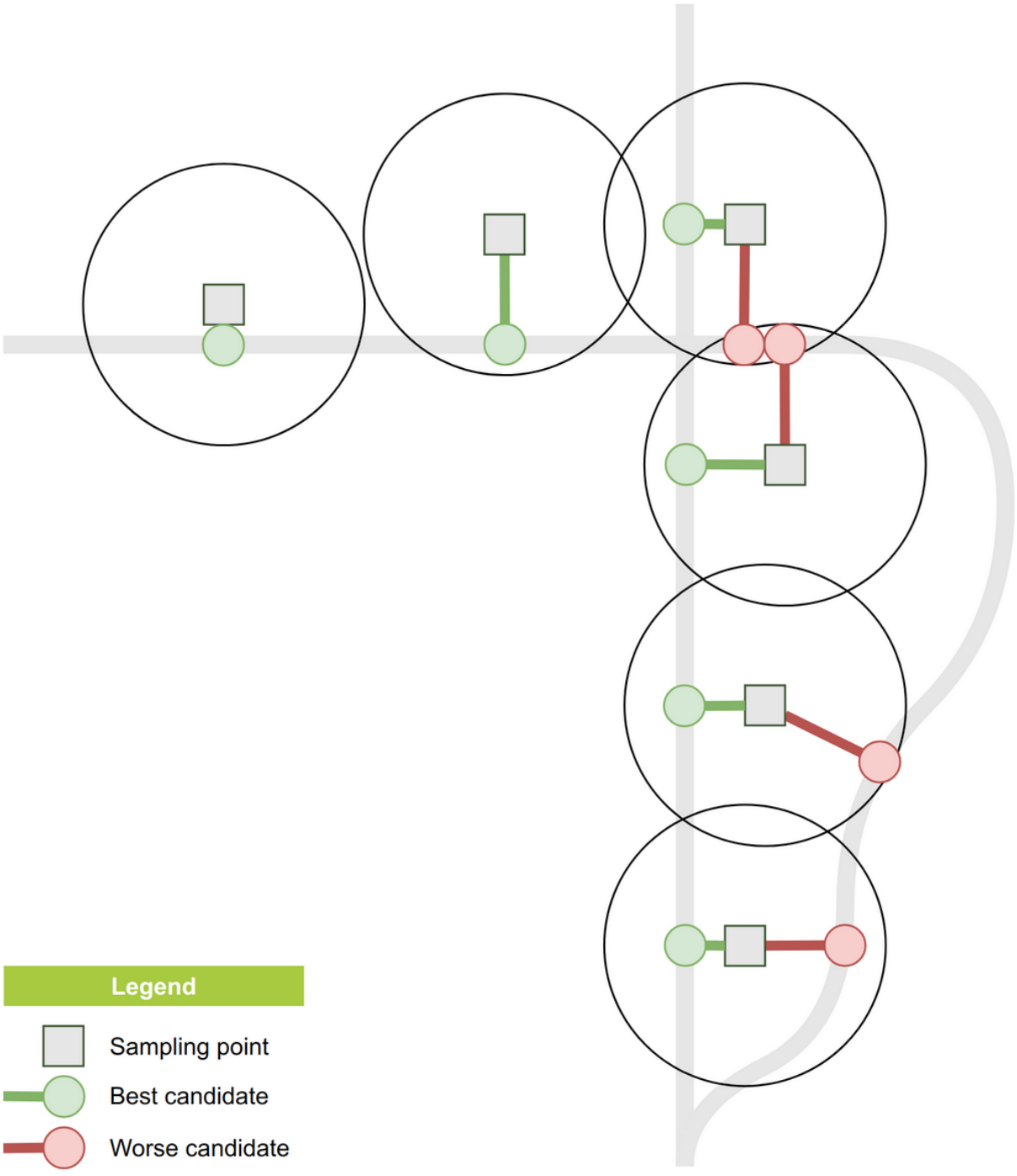


Fig. 6.11: Freeway exit ramp example

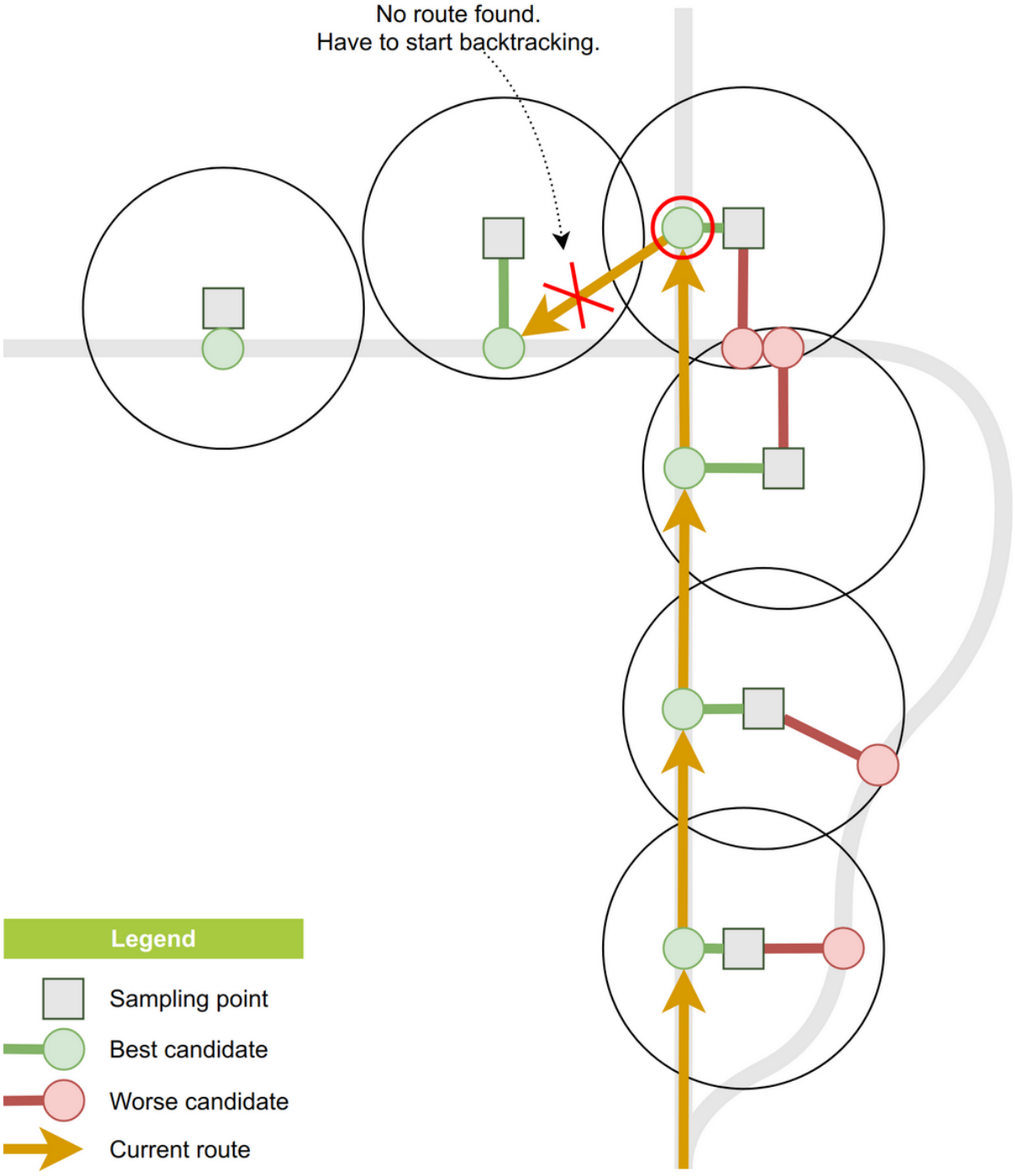


Fig. 6.12: Needs backtracking

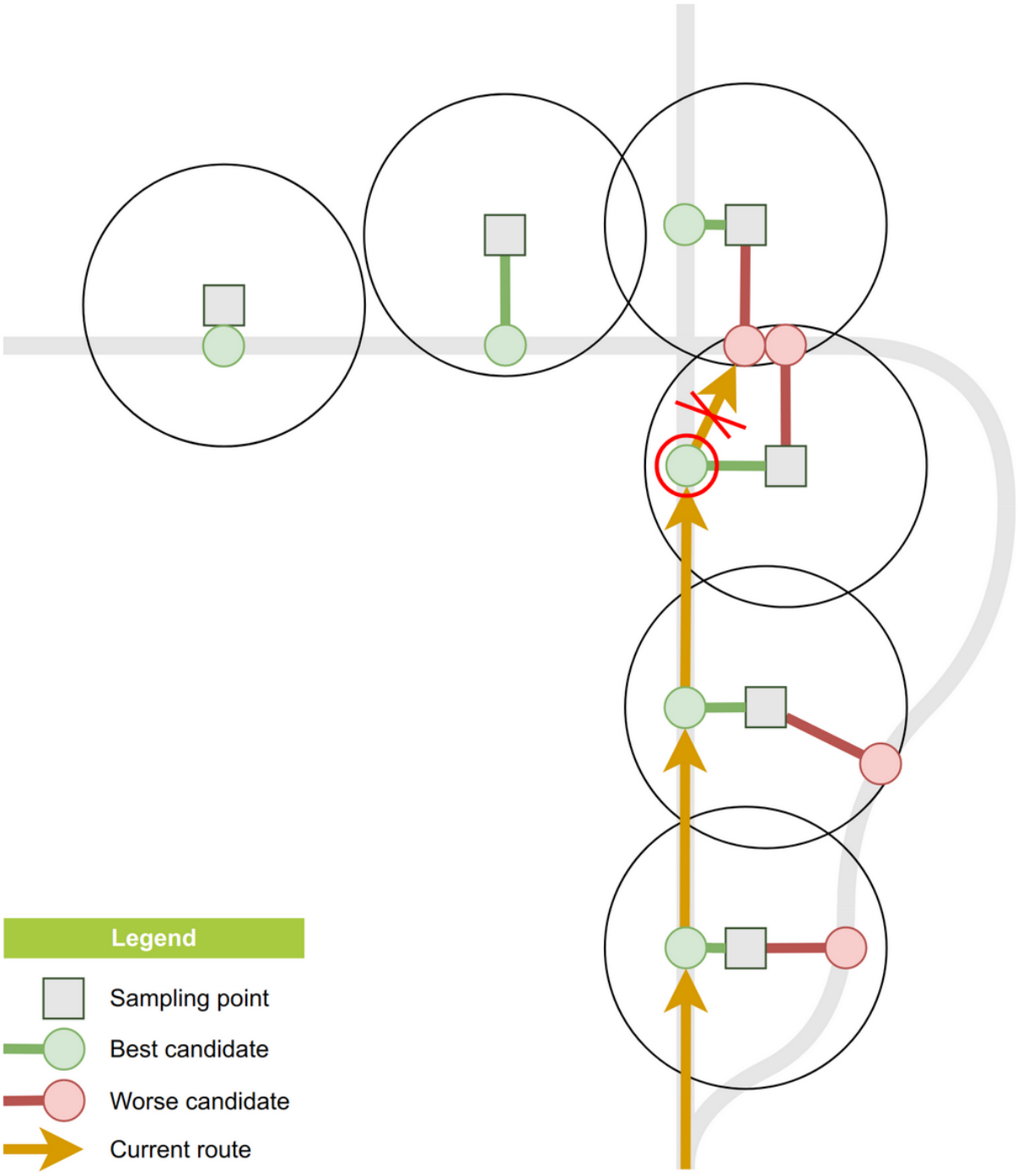


Fig. 6.13: Backtracking trial 1 failed

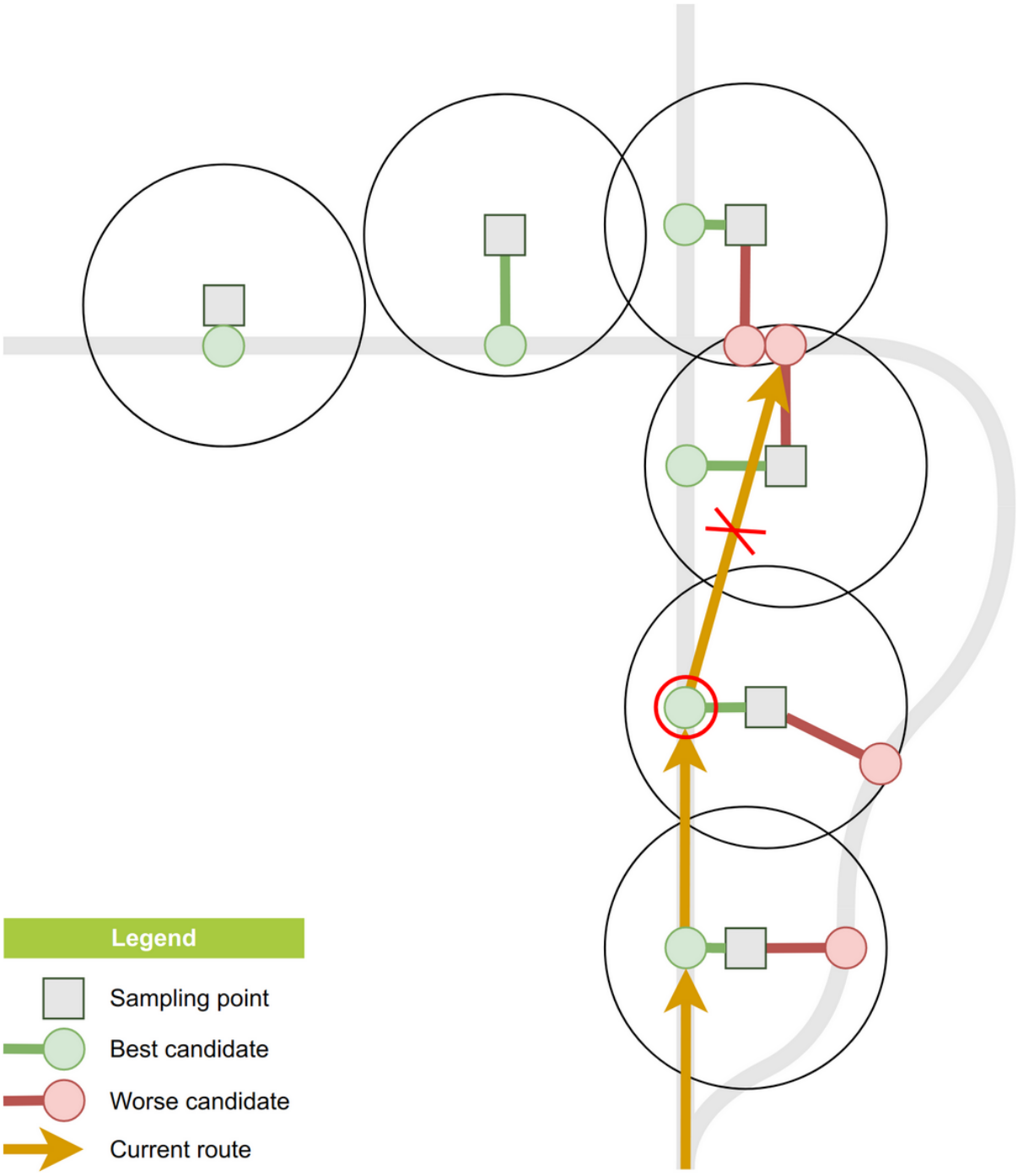


Fig. 6.14: Backtracking trial 2 failed

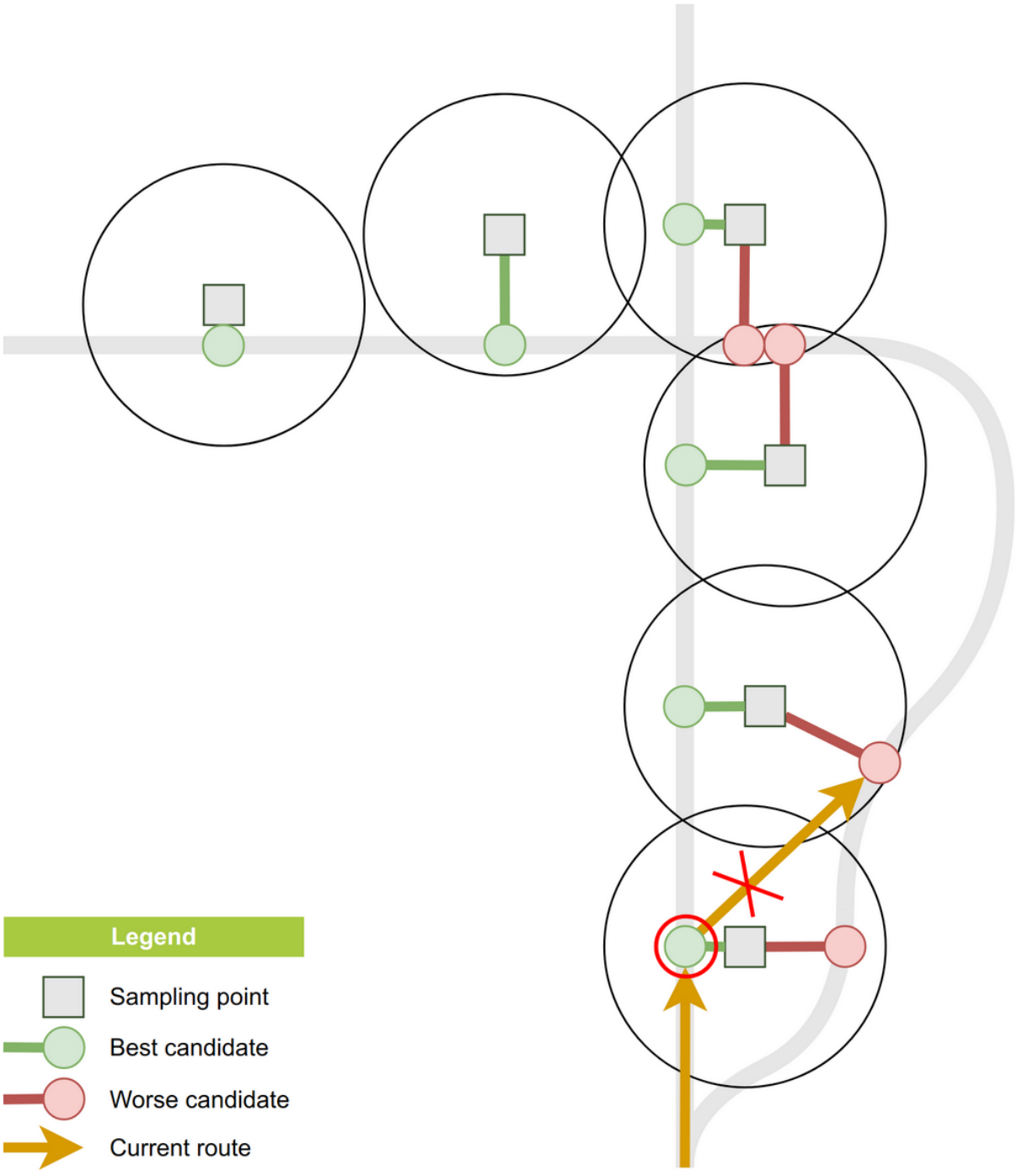


Fig. 6.15: Backtracking trial 3 failed

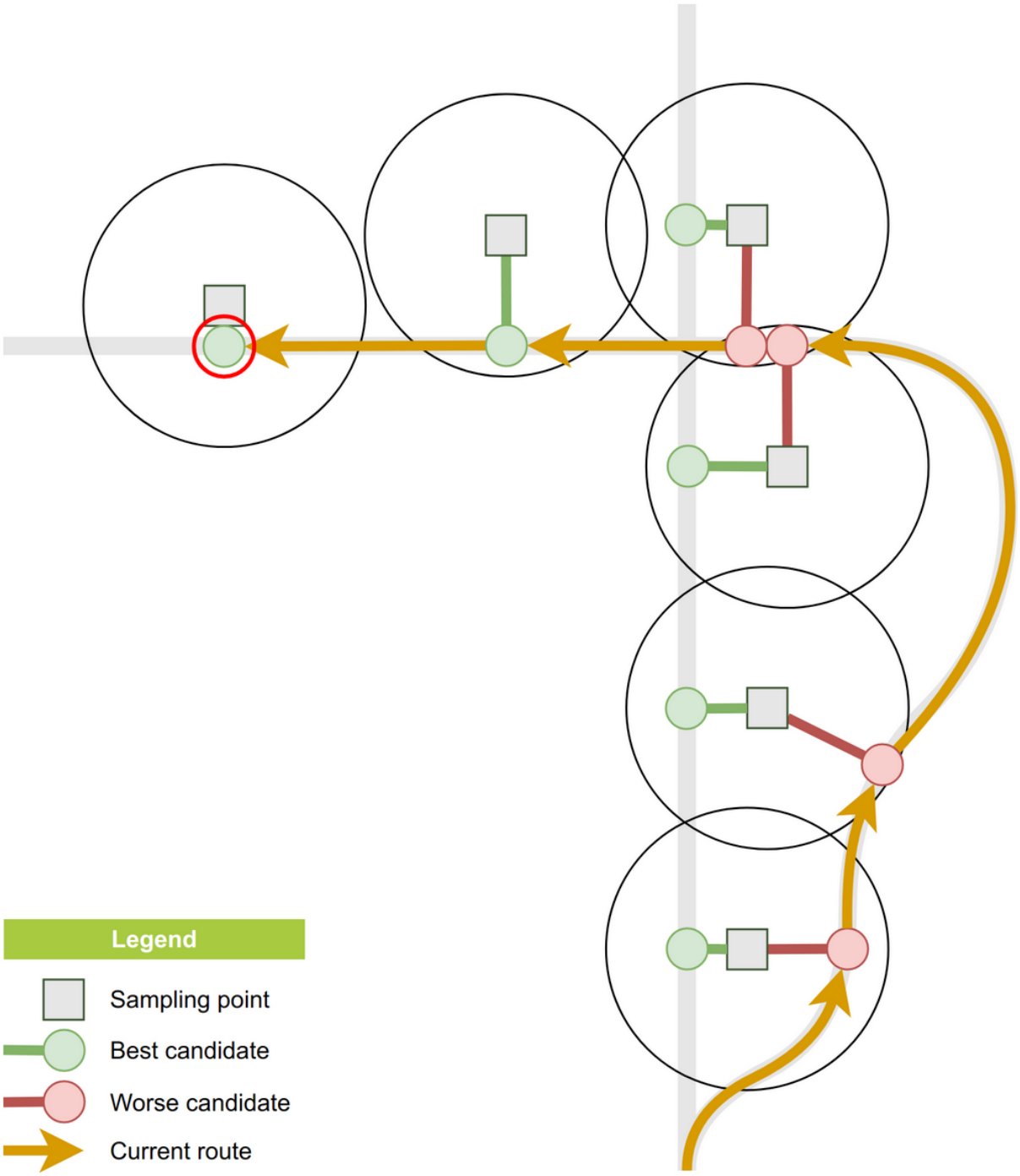


Fig. 6.16: Backtracking successful

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 successful 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.

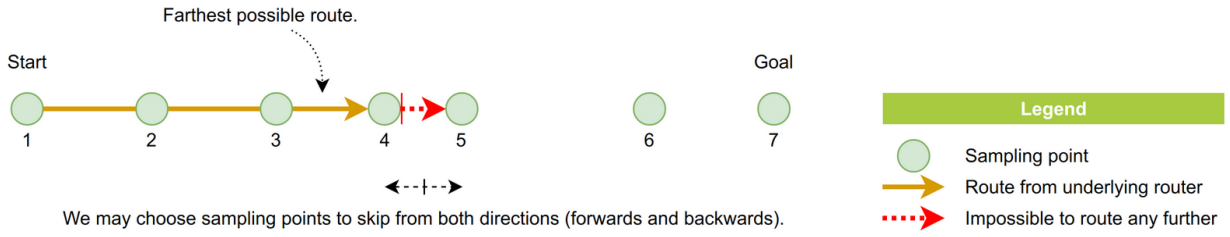


Fig. 6.17: Farthest route does not reach Goal

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 preceeding*.
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.

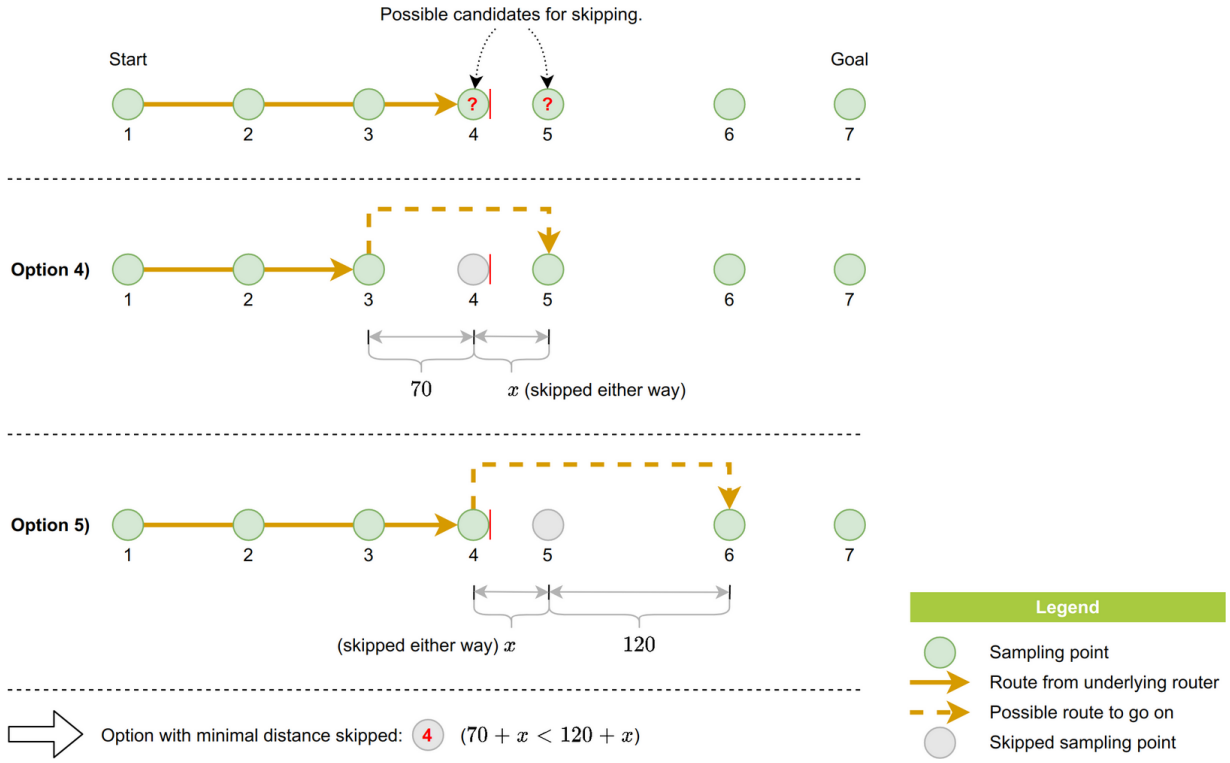


Fig. 6.18: Choosing sampling point to skip

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.

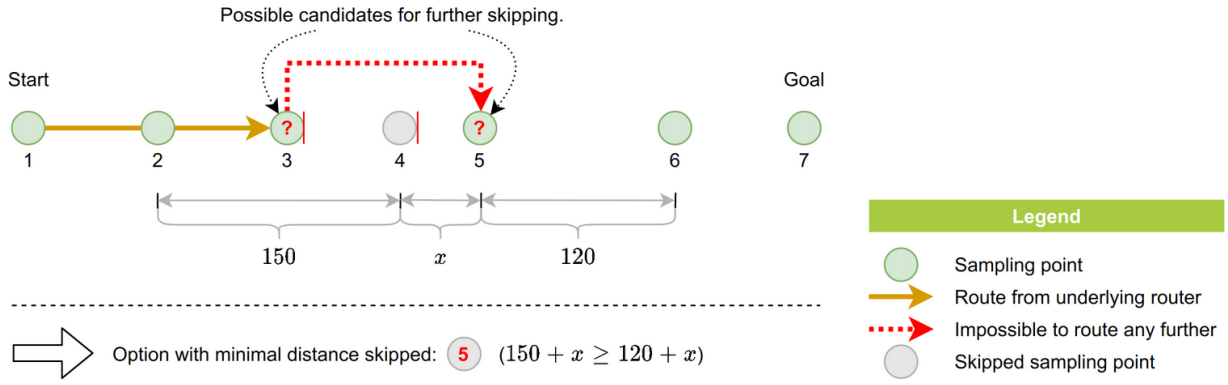


Fig. 6.19: Skipped sampling point 4

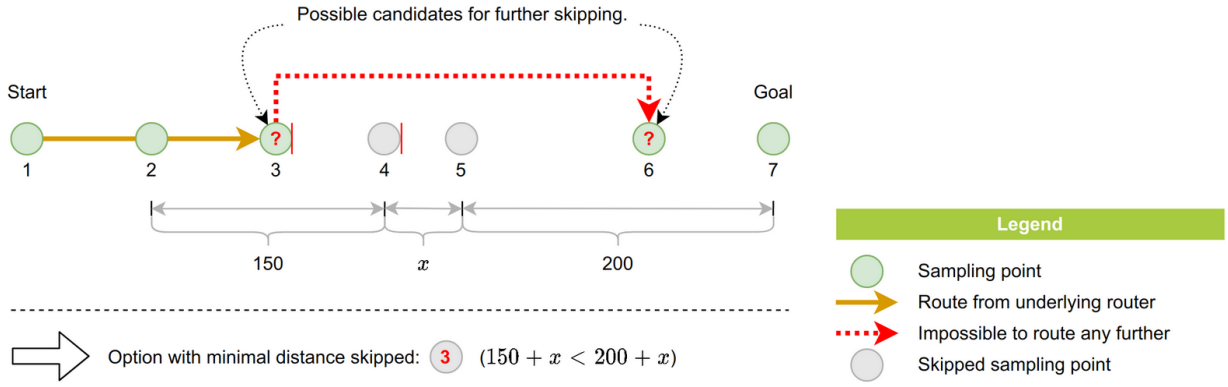


Fig. 6.20: Skipped sampling points 4 and 5

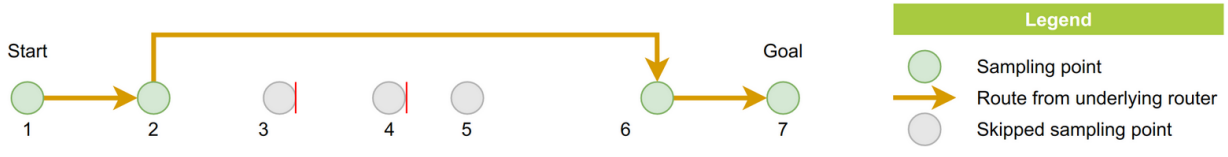


Fig. 6.21: Routed farther by skipping 3 sampling points

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 been 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**.

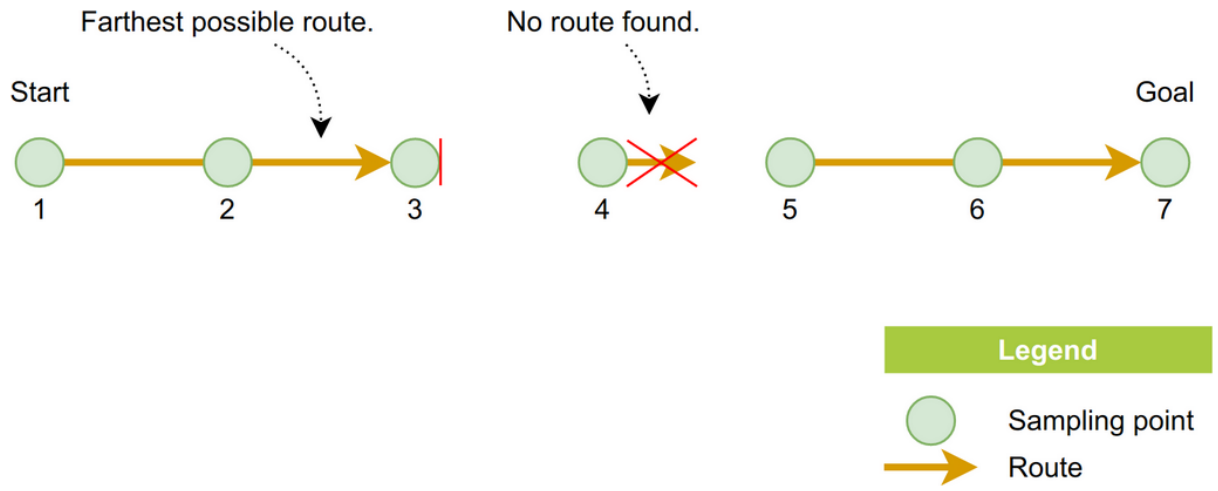


Fig. 6.22: Router functionality

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**.

CHAPTER

SEVEN

FILTERS

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

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

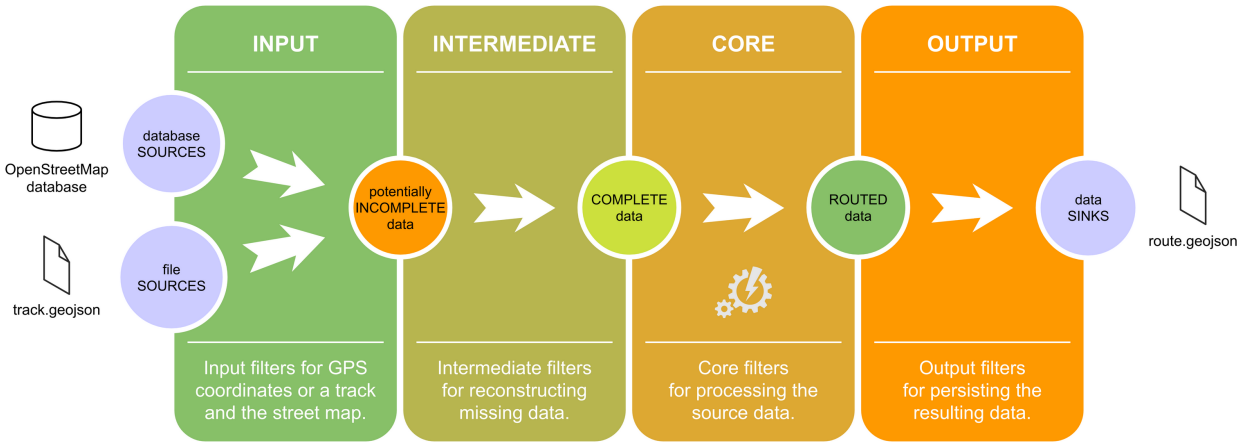


Fig. 7.1: Filter dataflow

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",
  ↪"HighwayList" } );
6 }
7
8 bool OsmMapReader::operator()(
```

(continues on next page)

(continued from previous page)

```
    Types::Track::PointList const & pointList,
    Types::Street::SegmentList & segmentList,
    Types::Street::NodePairList & nodePairList,
    Types::Street::TravelDirectionList & travelDirectionList,
    Types::Street::HighwayList & highwayList )
{
    /* ... */
}
```

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 (*distance of this two points*) / 2 + *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 ot 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_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)
...	...	...	...	...	...	...	...	...	...	...	...	...	...

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 53.2716969,10.4103007 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
...	...	...	...	...	...	...	...	...	...	...	...

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 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*

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,
               "distance": 2.73,
            },
            "candidates-count": 2,
         },
```

(continues on next page)

(continued from previous page)


```
            "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

Namespace anonymous_namespace_MakeHashable.h

Contents

Functions

- *Function anonymous_namespace_MakeHashable.h::hash_combine(std::size_t&)*
- *Template Function anonymous_namespace_MakeHashable.h::hash_combine(std::size_t&, const T&, Rest...)*

Namespace anonymous_namespace_SamplingPointFinder.cpp

Contents

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

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

Namespace AppComponents::Common

Contents

Namespaces

Namespace AppComponents::Common::Filter

Contents

Namespaces

Classes

Namespace AppComponents::Common::Filter::anonymous_namespace_GeoJsonMapReader.cpp

Contents

Functions

Namespace AppComponents::Common::Filter::anonymous_namespace_GeoJsonMapWriter.cpp

Contents

Functions

Namespace AppComponents::Common::Filter::anonymous_namespace_JsonRouteStatisticWriter.cpp

Contents

Functions

Namespace AppComponents::Common::Filter::anonymous_namespace_OsmMapReader.cpp

Contents

Classes

Functions

Typedefs

Namespace AppComponents::Common::Filter::Osm

Contents

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

- *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::SamplingPointCandida*

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

Contents

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

- *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

- *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

- *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

- *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

- *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

Namespace AppComponents::ExampleMatcher::Filter

Contents

Classes

Namespace Core

Contents

Namespaces

Namespace Core::Common

Contents

Namespaces

Namespace Core::Common::Geometry

Contents

Classes

Functions

- *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(nlohmann::json const&)*
- *Function Core::Common::Geometry::toLineString(std::string 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::piover180*

Namespace Core::Common::Postgres

Contents

Classes

Functions

Namespace Core::Common::Time

Contents

Functions

Namespace Core::Graph

Contents

Namespaces

- *Namespace Core::Graph::Routing*

Classes

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

Namespace Core::Graph::Routing

Contents

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

- *Namespace Generic::String*

Classes

- *Class Progress*

Functions

- *Template Function Generic::flipMap*

Namespace Generic::String

Contents

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.cp`

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_OsmM

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::OsmPointCa

Core::Common::Geometry::Point AppComponents::Common::Filter::anonymous_namespace_OsmMa

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::PointLocat

size_t AppComponents::Common::Filter::anonymous_namespace_OsmMapReader.cpp::PointLocat

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** *targetSamplingPointIndexGoal_*, 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*> **const** &*a*,
std::shared_ptr<Types::Routing::*Route*> **const** &*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*> **const** &*a*, std::shared_ptr<Types::Routing::*Route*> **const** &*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** &*subRoutes*)

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** *targetSamplingPointIndexGoal_*, 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> **skippers_**

Skipper ***selectedSkipper_** = {&*skippers_*[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*, *VisitedRouteSet* &*visitedRouteSet*, *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* &*routingStatistic*) **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*, AppComponents::Common::Types::Track::*PointList* *&pointList*, AppComponents::Common::Types::Track::*HeadingList* *&headingList*, AppComponents::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::Routing::*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 & geo

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::getStr

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_namespac

{ 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

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_GeoJsonMa

Function AppComponents::Common::Filter::anonymous_namespace_GeoJsonMapWriter.cpp::toString(Types::Str 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::Str 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>, PointGeo

{ candidates, osmPointMap, 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_OsmMapReade

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::TravelD

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::*Hig*
const &*highways*, std::string **const** &*tableName*)

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::isContain
```

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_nar
```

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::routeCa

- 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::ar
```

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 *sourceSamplingPointIndex*)

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** *maxAngularDeviation*)

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 *samplingPoint*, size_t *max*, std::unordered_s **const** &*skippedSamplingPoints*)

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** &*skippedSamplingPoints*)

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::Routing::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::Commo
`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*, double **`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`** &*geometry2*)

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** &*format*)

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** = 3.14159265358979323846264338327 95

Variable Core::Common::Geometry::piover180

- Defined in file_src_Core_Common_Geometry_NumericConstants.h

Variable Documentation

constexpr double Core::Common::Geometry::**piover180** = *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::geo
```

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<Str
```

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::PointGeoindexAlgorith

- Defined in file_src_AppComponents_Common_Filter_OsmMapReader.cpp

Typedef Documentation

```
using AppComponents::Common::Filter::anonymous_namespace_OsmMapReader.cpp::PointGeoindexAlg
```

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::PointGeoindexVal
```

Typedef AppComponents::Common::Filter::Routing::anonymous_namespace_SamplingPointRouter.cpp::Clustere

- 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::Samplin

- 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::SamplingPointSkipStrategy

- Defined in file_src_AppComponents_Common_Filter_Routing_Types.h

Typedef Documentation

using AppComponents::Common::Filter::Routing::**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*, C

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*>
Boxtype. 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::deg
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**&) >

CHAPTER

NINE

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 consist 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 point. 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`).

CHAPTER

TEN

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.

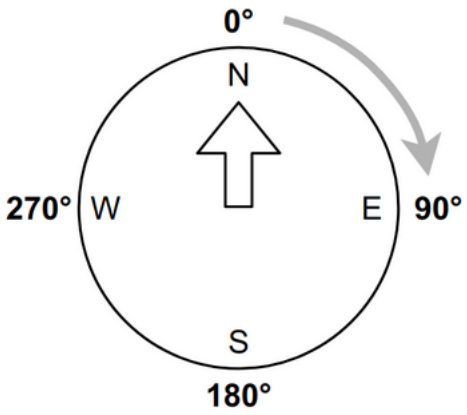


Fig. 10.1: Heading

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.

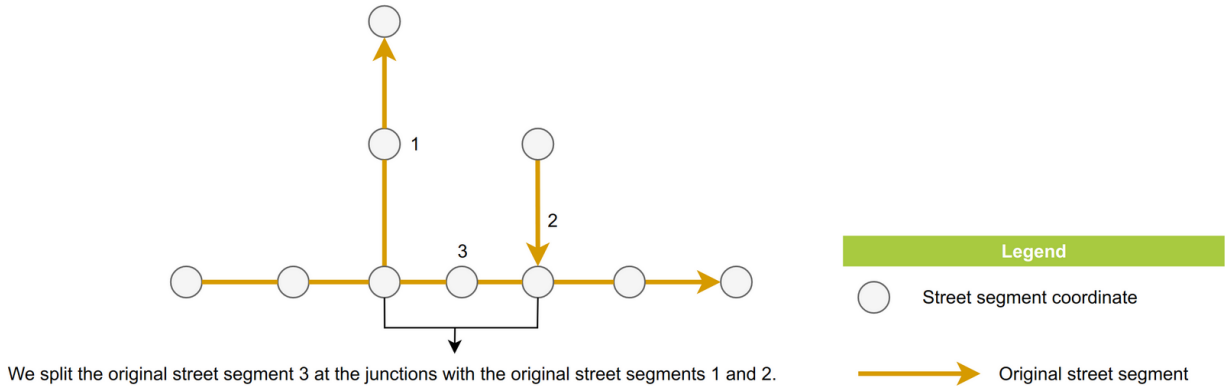


Fig. 10.2: Original street map

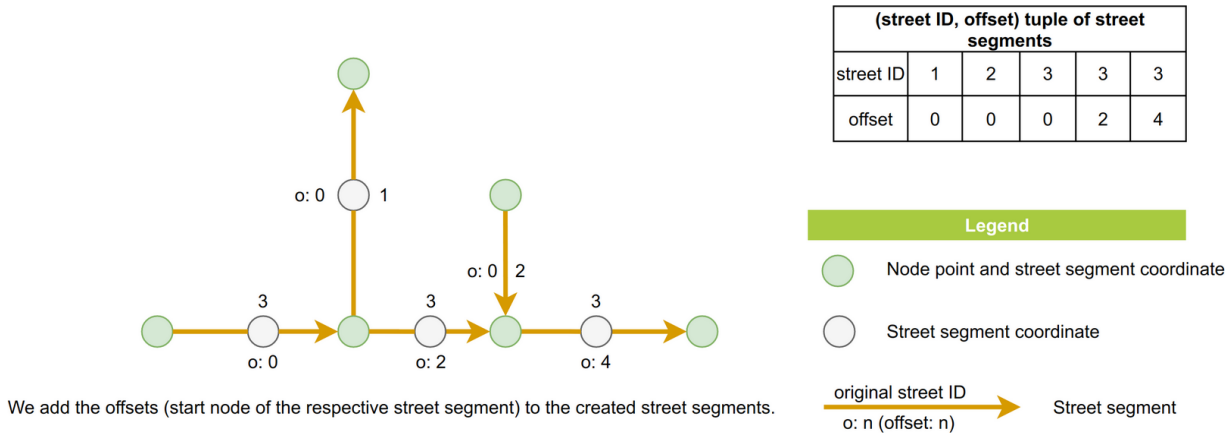


Fig. 10.3: Created street segments

INDEX

A

C

F

G

H

M

R

S

T

V