DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Vehicle routing with higher-order penalties

Egor Spirin

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Vehicle routing with higher-order penalties

Tourenplanung mit Kosten höherer Ordnung

Author:	Egor Spirin
Supervisor:	Prof. Dr.-Ing. Matthias Althoff
Advisor:	Dr.-Ing. Julius Ziegler
Submission Date:	15.07.2022

I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Garching, 15.07.2022 Egor Spirin

Acknowledgments

With many thanks to my advisor Dr.-Ing. Julius Ziegler for his guidance and encouragement throughout this research and to Atlatec GmbH for supporting the thesis and providing an exciting research topic.

Abstract

Autonomous driving has become a trendy topic in both academia and industry. Many ADAS systems rely on high-definition maps for navigation and localization. The creation process of HD maps is complex and often involves recording large road networks. Atlatec GmbH uses a proprietary navigation system called AtlaRoute to create optimized routes for the company's fleet of recording vehicles.

This thesis is dedicated to the problem of optimizing routes for driver comfort by punishing inharmonious maneuvers, such as premature highway exits. The problem frequently occurs when routes are optimized exclusively for duration or distance with standard Travelling Salesperson Problem solvers.

The work proposes two approaches to solve the problem: adding lane-level topology information to the routing graph to restrict unnecessary lane changes and optimizing routes with higher-order penalties using a higher-order graph representation of the road network, which we call the X-Graph. The latter approach is novel in the field of navigation and shows promising results.

Contents

1. Introduction

1.1. Motivation

Autonomous driving is one of the most attractive topics of today[Mat+16]. There are different approaches to building AV/ADAS systems. Some companies, such as Tesla, generate the map from the sensor data on-the-fly, but many others, including Bosch and Waymo, rely on HD maps for localization, simulation, and other tasks[TB12].

HD map creation is a complex and extensive process, including the recording of road networks, road graph reconstruction, detection of signs and road markings, etc. One of the most time-consuming stages[Pan+20] is on-site road recording, which requires a driver to cover the whole area of a given road network consistently. Often, the recording of each lane is required, making the navigation process even more complex.

Atlatec GmbH[1], a producer of large-scale HD maps, employs their own navigation and progress tracking system called *AtlaRoute*. The system optimizes road network traversals by solving Travelling Salesmen Problem (TSP)[G G02].

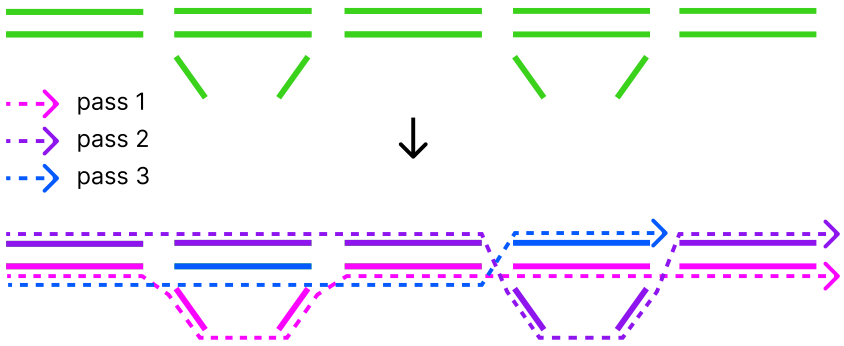


Figure 1.1.: Inharmonious maneuvers

One of the issues the company's drivers face is the problem of *inharmonious maneuvers*. This is a phenomenon in which the optimizer returns a route with many untimely maneuvers, such as premature highway exits, instead of staying on the highway for as long as possible. The consequence of the phenomenon is the inability of a driver to

[1] `https://www.atlatec.de`

follow the route, often because there is not enough time to change lanes between two consecutive maneuvers. Lane changes are also regarded as relatively unsafe maneuvers; hence they should be avoided.

Figure 1.1 shows an example of inharmonious maneuvers. The shown route is an optimal TSP solution of the AtlaRoute. In the first pass, the driver starts recording the rightmost lane but has to immediately take an exit and return to the highway. In the second pass, the driver records the leftmost lane of a highway but needs to take a right exit and then enter the highway and continue recording the left lane again. After two passes, two pieces of the highway are unrecorded (depicted in blue). The third pass covers the missed parts. As one can see from the figure, there are many unnecessary lane changes, some of which are dangerous.

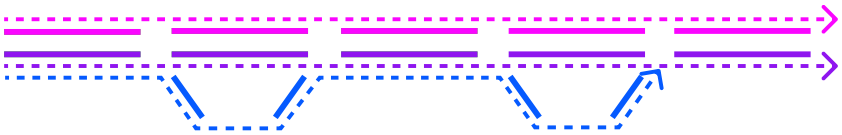


Figure 1.2.: Harmonious maneuvers

Figure 1.2 shows the same road network as figure 1.1 but with better passes. The colors of the passes are the same as in the other figure. The first pass completely covers the leftmost lane of the highway; the second pass covers the rightmost lane. Then the third pass records the remaining exits and entries. As one can see, there are fewer lane changes, with two passes having none. The example traversal has no inharmonious maneuvers. In this thesis, we want to achieve routing results that resemble this one.

The solution to the problem of inharmonious might significantly reduce the recording times because the driver would not need to return and rerecord the missing pieces of the road network. It might also improve the driver's comfort and safety, giving them more time to change lanes before the next maneuver.

1.2. Structure

The thesis is split into three chapters: Introduction, Approaches, and Evaluation. In the Introduction chapter we discuss the basics of Vehicle Routing Problems, introduce terminology used to describe routing graphs, and explain the current routing process. In the second chapter we introduce two approaches that solve the problem of inharmonious maneuvers and explain the graph transformation required by these approaches. In the Evaluation chapter we compare the proposed approaches with the current baseline solution and discuss their strengths and weaknesses and their potential application in a production environment.

1.3. Related work

There are only a few publications related to the topic of the thesis. [Gol08] gives an introduction to vehicle routing problems, which are a generalization of the traveling salesmen problem. We use some of the concepts presented there via the Google Ortools library[Per11]. [Liu+17] suggests to use a graph representation similar to the lane topology graph we present in section 2.1. While they present it only conceptually, we suggest a rule-based system to derive this from freely available data[Ope17].

1.4. Setup and terminology

Atlatec GmbH has developed its own navigation and progress tracking system called AtlaRoute. One of the main functions of the system is route optimization. This thesis primarily focuses on the part of AtlaRoute called Optimizer, which solves the Asymmetric Travelling Salesperson Problem for distance or duration of the journey.

Figure 1.3.: Routing problem

AtlaRoute uses our fork of Open Source Routing Machine[2], an open-source routing system that uses OSM[3] map data. It employs OSRM's concept of segments and global segment IDs and utilizes OSRM's API to get distance matrices, create routes, and download its internal routing graph. OSRM's routing graph has segment-level topological information, such as whether a segment is reachable from another segment and how much a specific maneuver costs.

[2] `http://www.project-osrm.org`
[3] `https://en.wikipedia.org/wiki/OpenStreetMap`

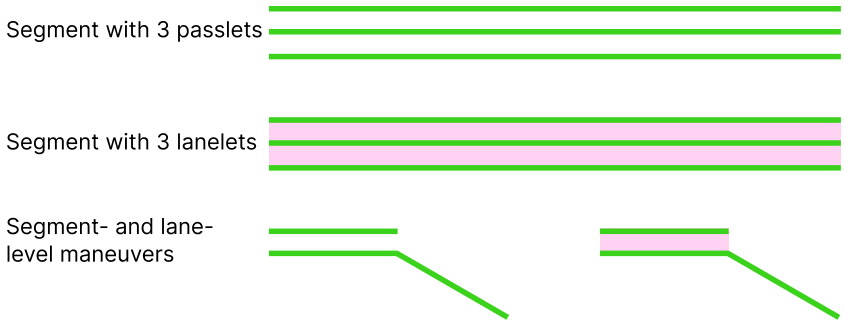


Figure 1.4.: Terminology

1.4.1. Terminology

A part of the terminology used in the thesis comes from AtlaRoute and needs to be introduced. *Routing problem* (RP) is a routing graph that needs to be recorded. A typical routing problem is shown in figure 1.3. Routing problems are defined with the help of OSRM and consist of *segments*, atomic parts of the road network usually spanning between two intersections. Each segment has a globally unique *segment ID*. A lane of a segment is called a *lanelet*, while a segment pass is called a *passlet*. Lanelet is a lane-level term, meaning that it is used only in the context of graphs with lane-level topology. In contrast, passlet is a pass-level term used in the context of a graph without lane-level topology. We will explain the difference between lanelet and passlet routing in detail in the next section. A *maneuver* is a transition from one segment, lanelet or passlet to another. An overview of terminology can be found in figure 1.4.

Routing problem segments are not required to be directly accessible from other segments. It is possible that a routing problem has multiple connected components (islands) if we consider accessibility only over segments present in a given routing problem. Of course, all routing problem segments must be in the same strongly connected component when considering accessibility over the entire road network.

In figure 1.5 we can see a distinct island in the upper part. It is not directly connected to other islands via other segments, but it is in the same strongly connected component in the road network. The island is accessible from the other parts of the routing problem and vice versa.

AtlaRoute is a complex system with much more functionality than is touched upon in this thesis. To give the reader a brief overview of what other important functions AtlaRoute offers other than route optimization, here is a short list of AtlaRoute's other functionality and concepts:

- *Saves* - a concept that allows to mark some parts of a routing problem as done

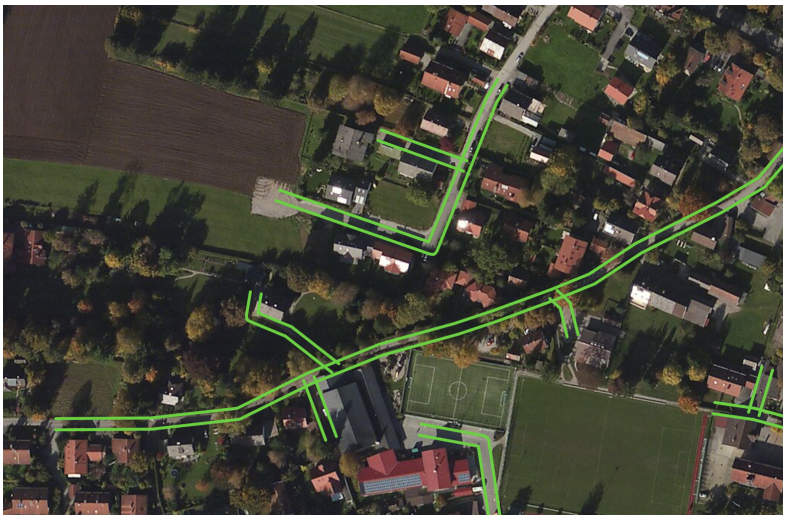


Figure 1.5.: Routing problem with multiple islands

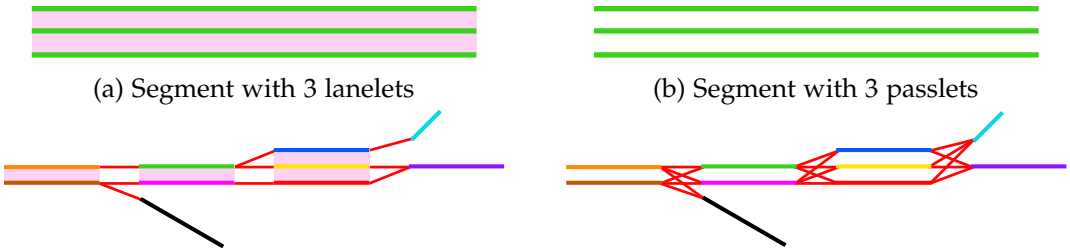
and exclude them from routing. It allows drivers to start recording from a specific checkpoint, like in video games. Saves contain information about which parts of a routing problem are recorded and to which extent. For example, if only a part of a segment is recorded, the router will still include this segment in the route.

- *Zugs* - an abstraction for a list of connected segments that must be traversed strictly in the predefined order and is also a part of a routing problem. It is used to model specific maneuvers that have to be taken to record some parts of the road network. For example, to completely record a large intersection, multiple turning maneuvers are required. They are often modeled as zugs in routing problems.

- *Strings* - an abstraction for a list of consecutive segments that can be bundled together to decrease the optimizer input size without impacting the routing results' optimality. For example, some consecutive segments on an autobahn can be combined into a string if there are no autobahn exits, entries, or other intersections between them. This concept helps speed up optimization and allows us to create large routing problems.

1.4.2. Lane-level vs pass-level routing

Currently, AtlaRoute performs optimization on pass-level, meaning that the nodes of the Traveling Salesperson Problem graph are segments and edge costs are road distances or durations between segments. Because we need to visit the same segment



(a) Segment with 3 lanelets

(b) Segment with 3 passlets

(c) Lane-level graph

(d) Pass-level graph

Figure 1.6.: Lane-level vs pass-level routing

multiple times, we clone nodes in the TSP graph, thus enforcing the optimizer to visit the same segment repeatedly. The name pass-level comes from the fact that some segments in the routing problem are visited multiple times or, in other words, have multiple passes.

OSRM's routing graph does not provide lane-level information. In the current workflow, routing problems are manually augmented with the number of lanes for each segment because every lane of a road network must be recorded to create a complete HD map. However, no lane-level topological information is currently available at the optimization stage.

A lane-level approach assumes a complete lane-level topology is available for a given road network. It should describe the lanelet-to-lanelet connectivity. For example, if the left turn is only allowed from the leftmost turning lane, only an outgoing left-turning edge from the leftmost lanelet should exist in the lane-level graph. Ideally, even the lane change restrictions should be present in the graph, but such detailed information is usually unavailable from mainstream routing data providers.

Figure 1.6 shows the difference between lane-level and pass-level graphs. Lane-level graphs and segments are represented with yellow background, while pass-level graphs and segments do not have a background. It can also be seen that the lane-connectivity information is not present in the pass-level example, as passlets do not represent lanes of a segment and thus have all-to-all connectivity in successive segments.

1.5. Current routing process

1.5.1. Overview

The current routing process gets a routing problem as input and gives an optimized route as output. In the process, the routing segments are expanded to add multiple passes, a distance or durations matrix is created, and an Asymmetric Travelling

Salesperson problem is solved.

A routing problem consists of segments, each of which holds the number of required passes for this segment. This number is added manually during the project preparation step, as OSRM does not provide the number of lanes for each segment. JOSM[4] and a proprietary InputCreator plugin are used to create and modify routing problems.

Figure 1.7.: Two-way street with adjacent two-way streets.

Each segment represents a single road direction, so two segments with opposite directions are required to traverse a typical two-way street. In the case of two-way roads, lane numbers are specified for both segments, as the number of lanes in one direction can differ from the number of lanes in another direction.

Figure 1.7 shows a typical suburban scenario with two-way streets that have one lane in each direction. The small adjacent alleys in the lower part of the picture are also modeled as two-way streets. In some cases, small alleys have physical space only for one vehicle at a time but are still modeled as two-way streets. This is done to show that a vehicle is allowed to move in both directions.

1.5.2. Graph creation and transformation

An *edge-based graph* (EBG) is defined as a directed graph with routing problem passlets as nodes, possible transitions from one passlet to another as edges, and road distances or durations as edge weights. EBG is used to represent routing problems during routing

[4] `https://josm.openstreetmap.de`

Durations matrix				
From/To	ID=1	ID=2	ID=3	ID=4
ID=1	315	0	23	17
ID=2	302	310	5	75
ID=3	284	305	65	189
ID=4	43	225	134	62

Table 1.1.: Step 1: example durations matrix

in the current AtlaRoute. The provided algorithm can be followed to convert a routing problem to EBG.

The first step is the creation of a distance or a durations matrix, depending on the routing settings. In most cases, durations are used instead of distances because the desired optimization goal is the reduction in the traversal time, not in the total route distance. The matrix contains N x N entries representing distances and durations from N segments in the routing problem to themselves. AtlaRoute uses several techniques to decrease that number by smartly combining some segments, which is out of this thesis's scope.

The matrix is requested from OSRM at runtime and is not stored or cached. That allows us to update the underlying routing information and influence routing behavior. For example, the weights can be updated with live traffic information, or a single road can be excluded from routing if it is not traversable because of construction work. The matrix is not modified and is kept in RAM for the duration of routing. It is represented as a dictionary of dictionaries, which allows for O(1) element access. The keys of the outer and inner dictionaries are segment IDs.

When talking about a distance or durations matrix, we talk about segments. Because passlets and lanelets represent passes and lanes of a segment, the distances or durations between them correspond to the distances and durations between their parent segments. We do not request distances or durations between passlets but between their parent segments to avoid unnecessary duplicates.

An example of a durations matrix is presented by the figure 1.1. Durations are specified in seconds. It can be concluded from the matrix that segments with ID=1, ID=2, and ID=3 are successive with a maneuver costing 5s between segments ID=2 and ID=3. Segment ID=4 is probably not directly connected to any of the other segments. It is also clear from the matrix that it was created for a routing problem in an urban environment since the detour durations are relatively small.

The second step is the conversion of the routing problem into the EBG. Routing problem segments become nodes, and possible connections between segments become

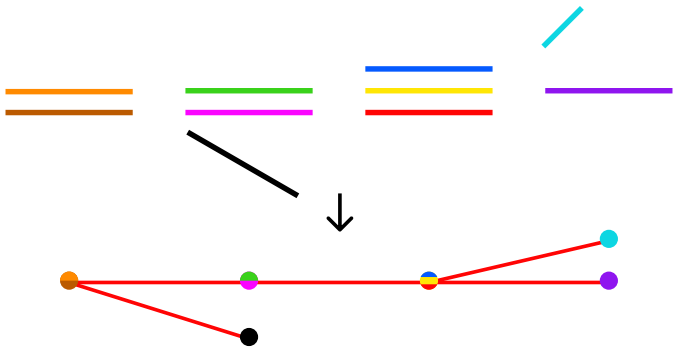


Figure 1.8.: Step 2: Edge-Based graph derived from the road network

edges. The weights of the edges are taken from the distance or durations matrix. The lanes number and segment ID attributes of each segment are saved in the corresponding nodes.

Figure 1.8 illustrates the second step. You can see that RP segments are converted into EBG nodes. After this step, there are fewer nodes than passlets because the step is performed on segments that contain multiple passlets. After the third step, the number of EBG nodes will equal the number of passlets in RP.

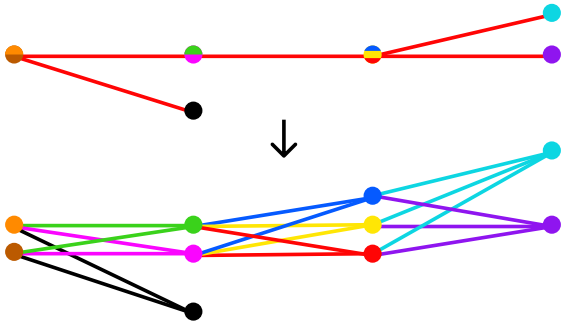


Figure 1.9.: Step 3: Node-to-passlet expansion

The third step is the node expansion. In the pass-level routing, a repeated traversal of a segment is achieved by copying the node and adding the copies to the routing graph with the same inbound and outbound edges and weights as the original node has. For each node with N lanes, N-1 node copies are created and added to the graph. The edge weights between node copies equal the weight from the original node to itself. That cost represents the distance or duration required to reach the node from itself, which requires a significant detour in practice.

In figure 1.9 the stored information about the passlets is used to expand the EBG

(a) Part of routing problem

Durations matrix			
From/To	ID=1	ID=2	ID=3
ID=1	145	8	47
ID=2	98	56	59
ID=3	5	31	78

(b) Durations matrix

Figure 1.10.: Real-world edge weights

nodes. After the expansion, the number of nodes equals the number of passlets in the original routing problem. This property guarantees that all passlets of the RP are present in the routing result and will be traversed.

1.5.3. Edge weights

We use actual road distances or durations as edge weights for optimization. As described above, they are taken from the matrix that comes from OSRM.

In the case of directly connected nodes, there are two options to determine the edge weight. The first option is to return 0, but the nodes have to be not only directly connected, but also there should exist a maneuver that connects them, and the maneuver cost should be negligible. The second option is to take the cost from the OSRM matrix, as many corner cases such as this are accounted for on the OSRM side.

The requirement to use a complete N x N matrix comes from two components. First, the chosen optimizer, Google OR-Tools, requires a complete all-to-all matrix. Second, OSRM's table service does not have an option to return sparse matrices. A sparse matrix can be created by combining results of multiple table service requests, but that is not practically useful, as a single all-to-all request is faster than many one-to-many or many-to-one requests required to create a sparse matrix.

Because of the requirement to always have a complete matrix, we do not depict edges in EBG and later in X-Graph, as it would not be practically possible. EBG, X-Graph, and their intermediate stages are depicted as nodes, and complete connectivity is always assumed.

A real-world example of a durations matrix is shown in figure 1.10. The maneuver required to get from segment ID=1 to segment ID=2 is a left turn at an intersection without a traffic light. The maneuver requires first to let the opposite traffic pass and then turn left, which is why the maneuver cost is 8s. The cost of the edge from segment ID=1 to segment ID=2 amounts to 47s, hinting that the next U-turn opportunity is close, but we do not see it on the image. The smallest cost in the durations matrix is from segment ID=3 to segment ID=1. The only explanation for a cost of 5s between segments with opposite durations is that there is either an intersection allowing U-turns or a roundabout directly after segment ID=3.

A *detour* is a part of a route that goes via segments that are not part of the routing problem. For example, the route from segment ID=1 to segment ID=2, depicted as a pink arrow, does not go via any green segments, which are part of the routing problem. Therefore, it is a detour.

Sometimes the interpretation and verification of distance or durations matrices can be complicated without special tools. To verify the correctness of matrices, we display OSRM routes between pairs of segments for each pair in the matrix and check their correctness. Having such a visual clue eliminates the need for guessing. However, OSRM's internal algorithms for matrix routing and normal routing differ and can produce slightly different results because the matrix routing is optimized for higher performance. This phenomenon is rare and usually does not impact the matrix inspection.

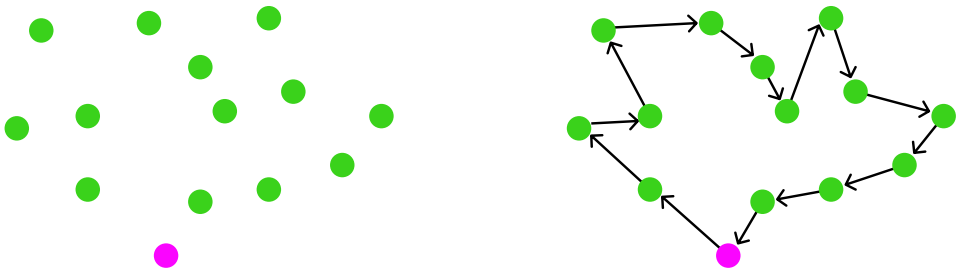
1.6. Vehicle routing problems

As the thesis's main topic is an instance of the general topic of vehicle routing problems, we introduce the essential concepts of this field.

In a Vehicle Routing Problem (VRP)[Gol08] the goal is finding optimal routes for multiple vehicles visiting a set of destinations. Usually, the optimization metric is chosen as the sum of all vehicles' route distances or durations. The aim of a VRP is to minimize this metric. In the following, we will describe only VRPs with one vehicle, as multi-vehicle VRPs are not relevant for the topic of this thesis.

VRPs consist of the following components:

- *Vehicle*. It might be a car, a bicycle, or even a pedestrian.
- *Locations* - a set of locations that need to be visited by the vehicle. All locations have to be in the same strongly connected component for a VRP to have a solution if there is only one vehicle.



(a) TSP (b) TSP solution

Figure 1.11.: Travelling Salesperson Problem with solution

- *Constraints*. There are many different types of constraints, but the most popular are the following.
  - Capacity constraints. The vehicle has a limited carrying capacity, while items have a quantity such as volume or weight. The vehicle cannot carry more items than its capacity allows. Some VRPs allow the vehicle not only to pick up items at some locations but also to unload items at some locations.
  - Time window constraints. Locations have specific time intervals during which they have to be visited.
  - Accumulated constraints. The vehicle has a limited amount of fuel. It limits the length of the route that the vehicle can take.
- *Penalties*. If the vehicle cannot visit all locations, a penalty is paid for each dropped location. A penalty can be defined separately for each location, prioritizing some locations other the others.

A particular instance of VRPs is the Travelling Salesperson Problem. It is formulated as follows: find the shortest route for a salesperson that needs to visit customers at different places and return to the initial point. VRPs in general and TSP in particular can be either symmetric or asymmetric. In an asymmetric VRP, the edges of the graph are directed.

Figure 1.11 shows an example of a Travelling Salesperson Problem. In this TSP, all nodes are connected, and distances between nodes are euclidean and are represented exactly on the image. The starting nodes are shown in pink. As one can see, the solution of the TSP is a tour connecting all nodes and returning to the starting node.

Some VRPs that allow dropping locations use the concept of *disjunctions*. A disjunction is a constraint on the locations of a VRP. It tells the optimizer to pick some locations out of the provided list. The optimizer pays the penalty if it cannot visit enough locations from the list. A disjunction is represented by a 3-tuple:

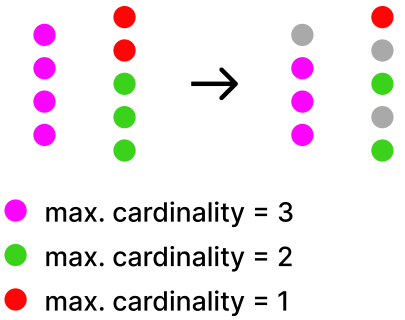


Figure 1.12.: Disjunctions and their resolution

- List of location indices - locations that are part of the disjunction.
- Maximal cardinality - the maximal number of locations that can be active in the disjunction.
- Penalty. If the optimizer picks less than maximal cardinality locations, a penalty is paid per inactive index.

We are particularly interested in disjunctions with maximum cardinality of one. If the maximum cardinality is one and the penalty is high enough compared to other costs in the graph, the disjunction can be described as: pick exactly one location from the list of provided locations. This type of disjunction is required when using the X-Graph representation we will later introduce in section 2.2.

An example of disjunctions and their resolution is depicted in the figure 1.12. The disjunctions in the example have different maximum cardinalities and high penalties for dropping visits. In each disjunction, exactly maximum cardinality nodes are chosen. The choice of which nodes to pick depends on other factors and is irrelevant for this example. Nodes that are not chosen are drawn in gray.

The VRP currently used in AtlaRoute is Asymmetric Travelling Salesperson Problem with segments as locations, road distances or durations between segments as costs, one vehicle, no penalties, no disjunctions, and no constraints. In comparison, the VRP used in X-Graph optimization is a VRP with maneuvers as locations, road distances, or durations between maneuvers with extra penalties as costs, one vehicle, multiple disjunctions, but no constraints.

It is important to note that most VRPs, including TSP, are NP-hard problems. Thus, it is practically impossible to find the optimal solution from some input size. Therefore, when the word solve is used in conjunctions with a VRP, the process of optimization and finding a good enough result is usually meant. In production, AtlaRoute allows the driver to decide when the optimization should be stopped. Usually, this decision is

made when the optimization result stabilizes and does not improve for several seconds. In this thesis, the optimizations are done multiple times with different optimization durations to ensure the optimizer had enough time to come to a nearly optimal solution.

2. Approaches

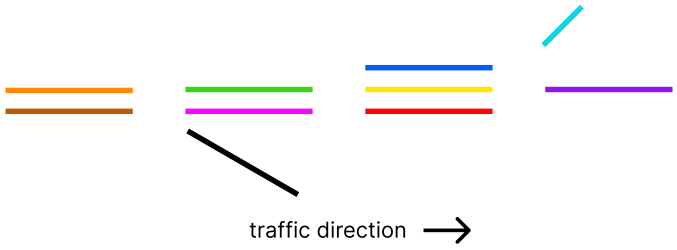


Figure 2.1.: Running example. Routing problem.

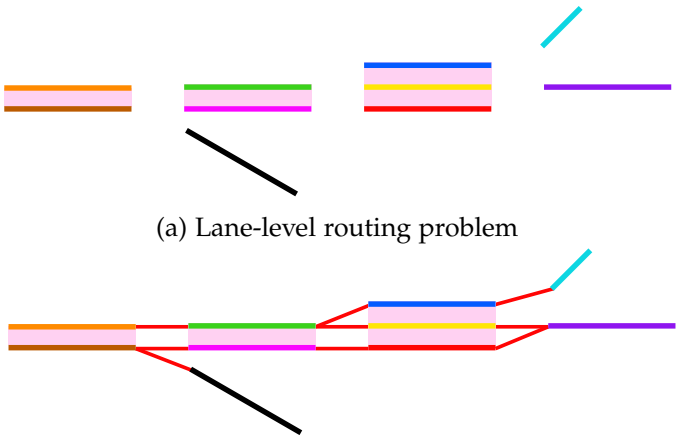
We introduce a running example to explain the idea and the approaches better. As shown in figure 2.1, it is a routing problem with traffic going from left to right. At first, no lane-level topology is available for the routing problem, only pass-level topology.

According to the pass-level topology, the right and the left turns are accessible from all passlets of the corresponding predecessor segments. In the case of the left turn, this constitutes a problem since there is a dedicated turning lane on the predecessor segment. The left turns from the middle or the rightmost lane are not allowed, but the graph does not contain this lane-level information. In the case of the right turn, the option to turn right from the leftmost predecessor passlet is also problematic, as this is likely an illegal maneuver.

The example is used to illustrate two approaches that this thesis proposes. It will be slightly modified in some places to highlight some important details. If a segment has a pink background, we are in the lane-level context, and the segment consists of lanelets. If the background is missing, we are in the pass-level context, and the segment consists of passlets.

2.1. Lane-level topology graph

The approach includes adding lane-level topology information to existing routing problems and solving lanelet-level TSP with special lane change constraints. The lane-level topology is generated with a set of heuristics and includes information about the neighbors of each lanelet.



(a) Lane-level routing problem

(b) Lane-level routing problem with lane-level topology

Figure 2.2.: Process of adding lane-level topology

Figure 2.2 shows the routing problem with lanelets and the result of adding lane-level information using heuristics. As one can see, the left turn is done from a dedicated turning lane. The heuristics detected it and connected the lanelets accordingly. In the pass-based graph, the turn would be allowed from any passlet.

Before applying heuristics, a routing problem is transformed into EBG. The process is similar to transforming a routing problem into EBG in the current AtlaRoute routing process. However, there are some important differences. When converting segments into EBG nodes, the topological information is also copied to the new nodes. That means that the EBG has information about which nodes are directly connected and which maneuver needs to be taken to get from node A to node B. This information is used later in heuristics. The second difference is that the EBG nodes are also provided with information about the common parent. For example, if a segment has three lanes, the three EBG nodes created from this segment will have an attribute parent pointing at the original segment.

The heuristics for adding lane-level information handle the most common cases, such as:

a Segment with N lanes -> segment with N lanes - trivial, connect each lanelet with the corresponding one.

b Left or right turn, but not a merge - start connecting lanelets from the same side as the direction of the turn

c Merge maneuver - start connecting lanelets from the opposite side to the maneuver direction

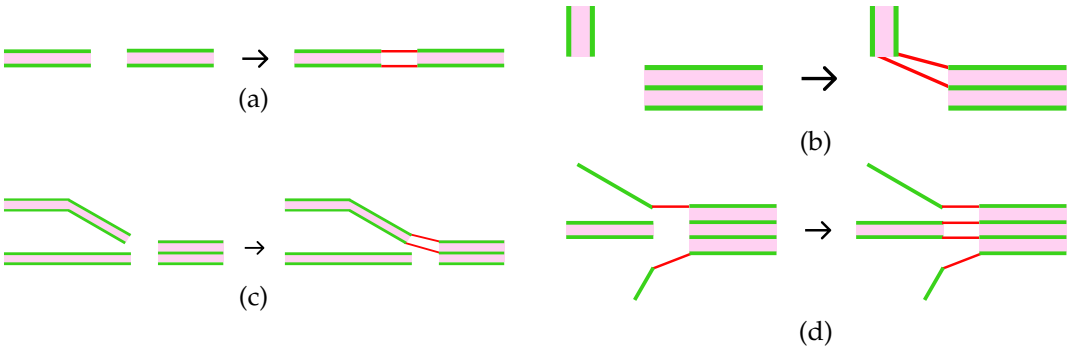


Figure 2.3.: Heuristics

d Segment with N lanes -> segment with N lanelets without incoming connections but with some other incoming connections - connect to the unoccupied lanes of the destination segment

This is not the complete list of heuristics. All heuristics can be found in the project's repository[1].

The cases from the list above are depicted in figure 2.3. On the left side of each image, one can see the original situation, and on the right side, the result of the applied heuristic. The labels in the figure correspond to the labels in the list.

2.1.1. Heuristics application

The heuristics are applied one after another in two passes over all routing problem lanelets. The first pass connects lanelets that are a part of a turn maneuver. It is essential first to create turn connections, as in most cases we can tell how the lanelets in a turn maneuver are connected. For example, almost all highway merge maneuvers merge from the right side if the country has right-hand traffic, while it is practically impossible to correctly guess which straight-going lanelet will be connected to which after a complex intersection. The second pass connects the remaining lanelets. In this pass, the information about the common ancestry of lanelets and already created connections is used instead of the information about maneuvers.

The heuristics application process can be imagined as industrial sieving: a stone falls into the first large enough hole on the conveyor. In each pass first the strongest heuristic is tested. If it does not apply to the pair of lanelets, the next heuristic is tested. When a heuristic matches, no other heuristics in this pass are applied to this pair.

[1] `https://github.com/espirin/vehicle-routing-with-higher-order-penalties`

2.1.2. Cases

Let us take a closer look at the four cases and discuss them with the help of real-world examples. The cases are taken from the example routing problem, which was used for other examples and evaluations in this thesis. It is also used as the primary routing problem in the Jupyter notebook.

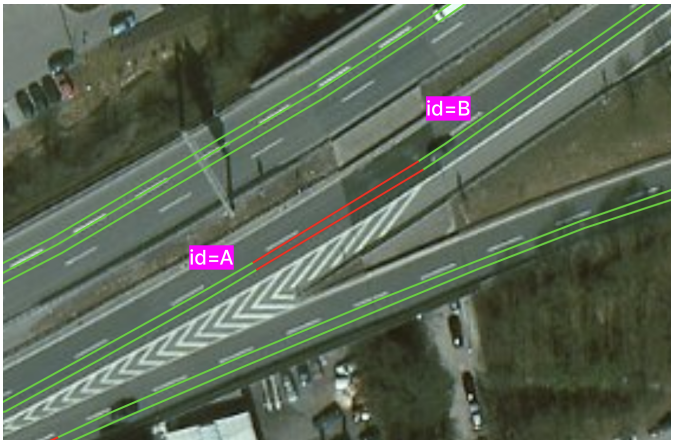


Figure 2.4.: Heuristic A

Heuristic A is a second pass heuristic, as it connects remaining lanelets and does not rely on maneuver information. It can be formulated more strictly: if a segment has exactly one predecessor and the predecessor has precisely the same number of lanes, connect their lanelets in the apparent order. This case is often seen on highways and long streets with few intersections.

Figure 2.4 shows two segments with two lanes each. Heuristic A connects the lanelets of segment id=A with segment id=B. The connections are depicted in red.

Heuristic B is a first pass heuristic. It connects left and right turns, U-turns and other maneuvers to the leftmost lanelets of the destination segment if the maneuver direction is left or to the rightmost lanes of the destination segment if the direction is right.

Figure 2.5 shows a right turn maneuver from segment id=C to segment id=D. The incoming connection from another segment to segment id=D can be ignored, as it is created in the second pass after heuristic B is applied. As this is not a merge maneuver, the single lanelet of segment id=C is connected to the rightmost lanelet of segment=D.

Heuristic C is a first pass heuristic. It is a particular heuristic for merge maneuvers. Merge maneuvers work the opposite way from the standard turn maneuvers. If a segment merges from the right, the maneuver direction will be left, as the driver needs to do a left turn to merge onto the highway. The heuristic connects merges with the left maneuver direction to the rightmost lanelets of the destination segment and vice versa.

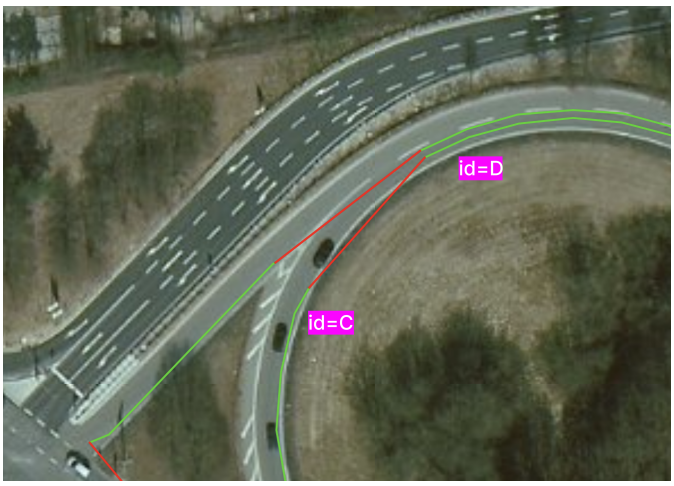


Figure 2.5.: Heuristic B

In figure 2.6 one can see a merge onto a highway with a left maneuver direction. Segment id=E has two lanes; segment id=F has four. The straight incoming connection to segment id=F can be ignored, as it was created in the second pass. Two lanelets of segment id=E are connected to two rightmost lanelets of segment id=F as in merge maneuvers the maneuver direction is the opposite of the direction of standard turn maneuvers.

Heuristic D is a second pass heuristic. It is applied in cases when some of the destination segment's lanelets already have incoming connections, and its number of lanelets without incoming connections is equal to the number of lanelets of the source segment. The lanelets of the source segments are connected to the free lanelets of the destination segment.

Figure 2.7 shows a more complicated case in which heuristic D is applied. In the first pass, the only lanelet of segment id=I was connected to the leftmost lanelet of segment id=G. Then heuristic D is applied in the second pass. It connects two lanelets of segment id=H to the two unoccupied lanelets of segment id=G.

2.1.3. Considerations

The heuristics described above are relatively primitive and do not cover all possible cases. In some scenarios, it is not possible to define a rule that would correctly connect lanelets (EBG nodes); in some other scenarios, the definition of a correct rule can be very complex.

Another problem with heuristics is that they are not universal with regard to countries.

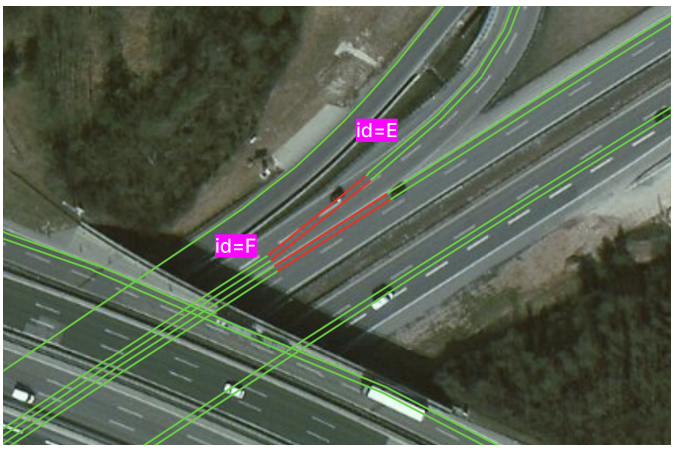


Figure 2.6.: Heuristic C

Figure 2.7.: Heuristic D

Road construction code can differ from country to country, requiring the heuristics to be adapted for each region, as many heuristics rely on road topology assumptions.

The creation of connections in a routing problem using the described heuristics has linear time complexity, as all connections can be created in a single pass over the routing problem's lanelets.

2.2. X-Graph

The X-Graph approach is an instance of a Vehicle Routing Problem with one vehicle, maneuvers as nodes, road distances or durations between maneuvers as edge weights, special higher-order penalties encouraging the desired route characteristics, and dis-

junctions that allow the optimizer to drop some location visits. In our nomenclature, the X-Graph approach is a pass-based method. A maneuver is defined as a pair of two passlets in this section. The name of the X-Graph was chosen arbitrarily and does not have an intrinsic meaning. The term X-graph was chosen early in the conceptual phase of this work, and we use it now for convenience. The more appropriate name would be *Maneuver-Based Graph* (MBG).

The additional penalty defined for particular sequences of maneuvers can incentivize different properties of the resulting route. The desired property for our route is to have few inharmonious maneuvers. We propose a penalty definition that punishes the change of the maneuver direction from straight to non-straight and vice versa. This definition achieves the effect of building strings of straight maneuvers followed by strings of non-straight maneuvers, etc.

2.2.1. VRP graph orders

X-Graph can be viewed as a higher-order graph because it represents a "derivative" (line graph[Edg17]) of the EBG, which is itself a "derivative" of the lower-order Node-Based Graph (NBG), which will be introduced in this section. Let us take a look at the three graph types that can be used in VRPs: 0th-order Node-Based Graph, 1st-order Edge-Based Graph, and 2nd-order X-Graph.

An important note is that the edges and edge weights definitions can vary significantly between applications (in a graph-theoretical sense). We specify how edges are defined where it is relevant, but in some places, that information might be left out. Edge weights can also be defined differently. A common distinction is whether to include the node weight into the incoming or outgoing edge weights. For example, if the distance between city A and city B is 50km, but the visit to city B requires another 20km, this distance might be added to the inbound distance, making the A -> B edge weight 70km. The optimization result is not influenced by that choice, as the sum of the node weights is just a constant added to the cost independent of the routing result. That means that the node cost does not influence the routing result if all nodes need to be visited.

Node-Based Graph is the 0th-order graph with routing problem nodes as nodes. The edges of the graph connect nodes to form passlets. For example, a routing problem passlet with two nodes would be represented as two NBG nodes with one edge connecting them with the direction of the original passlet. The weights in the NBG represent road distances or durations between passlet nodes.

Figure 2.8 shows the running example in the form of the NBG. Red dots are nodes; edges are NBG edges. Two nodes and an edge connecting them form a passlet.

NBG's use in VRPs is limited since the data structure is not well suited for VRP

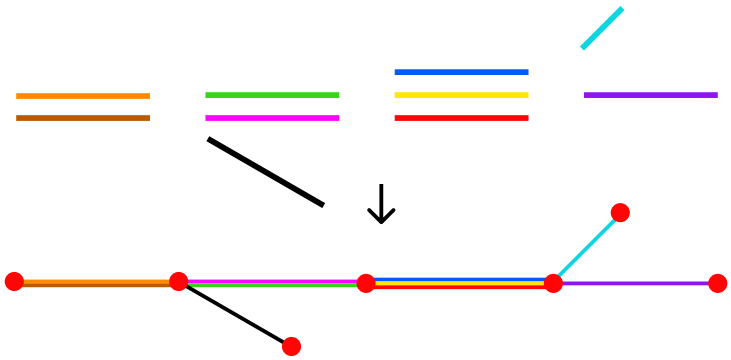


Figure 2.8.: Node-Based Graph derived from routing problem

optimization. The optimization of the NBG does not solve the VRP in which the goal is to visit all passlets because the optimization is done on nodes, not passlets.

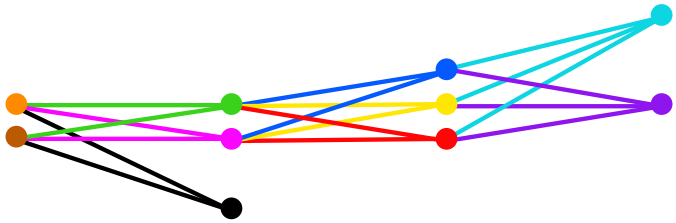


Figure 2.9.: Edge-Based Graph

Edge-Based Graph (EBG) is a widely used road network representation with routing problem passlets as nodes and road distances or durations between passlets as edge weights. The edges can be specified differently. Some applications, as this thesis, require complete all-to-all connectivity, while other applications might require edges only between directly connected nodes.

Figure 2.9 shows the Edge-Based Graph of the running example. Dots are passlets; edges are connections between passlets.

EBG is a modified *line graph* of NBG. Line graph of a graph G is another graph $L(G)$ that represents the adjacencies between edges of G. Other terms used for the line graph are derivative, edge graph or edge-to-vertex dual[Edg17]. Per definition of the line graph, the edges of NBG become nodes of its line graph, and edges are created between all line graph's nodes that share the same node in the NBG representation. If we modify the definition slightly by adding that successive NBG nodes without branching can be combined into passlets (EBG nodes), we get the definition of EBG.

EBG is a convenient representation for routing problems since the routing is done on passlet level, which is the same as segment-level routing other than allowing to

force multiple passes on the same segment. The definition of a segment says that, in most cases, a segment spans between two junctions. It makes the definition granular enough for vehicle routing since there are usually no cases where a relevant maneuver is possible on the road sections between junctions. It is important to note that under a junction we understand any place in a road network where more than one node is reachable from the current node.

All graph representations require a way to specify more precise locations on the map. For example, the EBG definition does not allow to specify a location on the passlet because the passlet is the smallest part of the EBG definition. OSRM uses the concept of *Phantom Nodes* (PN) to precisely specify the position of the start, end, and via nodes on the road network. A Phantom Node is a particular type of node in EBG that does not represent a passlet but an exact position on a passlet. It has an offset attribute that defines the offset from the start of the passlet where the PN's actual location is. The distances or durations to and from other segments can be trivially calculated by considering PN's position on the underlying passlet.

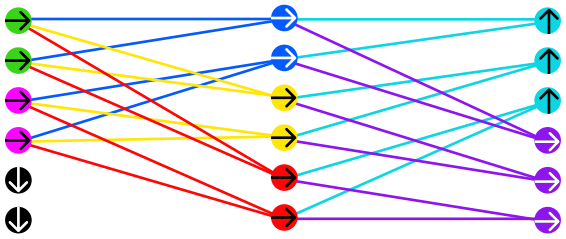


Figure 2.10.: X-Graph

X-Graph is the line graph of EBG, which itself is the modified line graph of NBG. The nodes of the X-Graph are pairs of successive passlets or, in other words, maneuvers. The edges can be defined as all-to-all or only for directly connected maneuvers that share a segment as a common node in EBG.

Figure 2.10 shows the X-Graph representation of the running example. Nodes are maneuvers (pairs of passlets). Edges are direct connections between maneuvers. The construction of the X-Graph is described in detail in the next section.

X-Graph's definition allows us to set maneuver-to-maneuver costs, more than doubling the amount of information that can be encoded in each edge weight compared to EBG. X-Graph edge weight can contain information about two segments in the source maneuver and two segments in the destination maneuver. Furthermore, it can contain the maneuver information of both maneuvers, such as direction, cost, etc. The increased amount of information in each edge allows optimization with more complex optimization goals. In this thesis, the optimization goal of the VRP based on the

X-Graph can be roughly defined as: visit all nodes of the X-Graph, minimizing the total detour distance or duration, and the number of inharmonious maneuvers in the tour.

In practice, the NBG -> EBG transition significantly decreases the number of nodes, while the EBG -> X-Graph transition dramatically expands the number of nodes. In the example from the Jupyter notebook, the number of nodes in NBG equals 549. The line graph transformation further increases that number. However, EBG is a modified line graph of NBG in which NBG nodes forming a passlet are merged together. Because of that fact, the number of nodes in the EBG after the transformation equals 60. The next transformation into X-Graph increases that number to 147.

The decrease in the NBG -> EBG transition can be explained by the fact that we merge successive nodes without branching into passlets, decreasing the number of nodes in EBG in comparison to the pure line graph transformation. The average branching factor[Dav17] is smaller than the average number of nodes in a segment in that particular routing problem; hence the number of nodes decreases during the transformation. That is not surprising since the large portion of the routing problem is an autobahn with many curved turns and curved entries and exits.

The node number increase in the EBG -> X-Graph transition is explained by the average backward branching factor of EBG being larger than 1. The more branching there is in the road network, the more extra nodes will be created during the EBG -> X-Graph transition.

2.2.2. Conversion of Edge-Based Graph into X-Graph

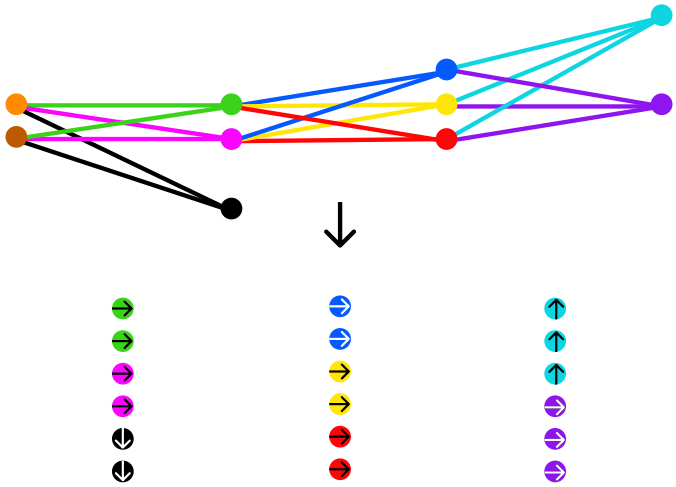


Figure 2.11.: Conversion of EBG into X-Graph, step 1

The first step of the Edge-Based graph to X-Graph conversion is transforming directly connected pairs of passlets into maneuvers. Each pair of directly connected passlets makes up a maneuver and becomes a node in the X-Graph. Passlets are directly connected if one can get from the source passlet to the destination passlet without visiting another passlet and obeying the traffic rules as they are specified in the road graph.

Figure 2.11 shows an example of the first step of EBG to X-Graph conversion. Maneuvers with the same color have the same destination node in EBG. They also form disjunctions. For example, there are two maneuvers from the first segment (orange and brown passlets) to the green passlet. They are depicted in green. These two maneuvers form a disjunction, making sure that the destination node (green) is entered exactly once.

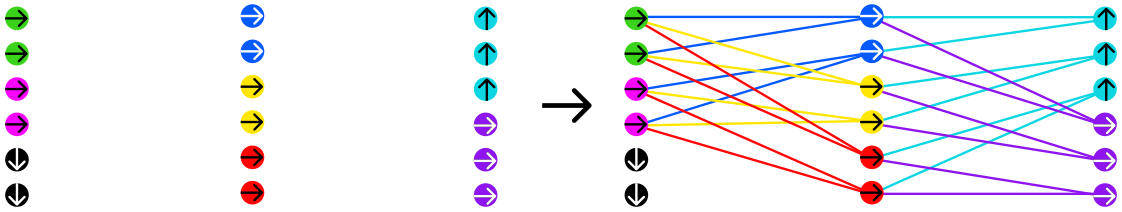


Figure 2.12.: Conversion of EBG into X-Graph, step 2

In the second step, we create edges of the X-Graph. Per definition of the line graph, edges of the original graph sharing a node get connected with an edge in the line graph. In X-Graph, we connect the maneuvers that share the EBG node in the EBG representation.

Figure 2.12 shows how edges are created for the example X-Graph. One can see that the green X-Graph nodes are only connected to the upper yellow node. This can be explained by the fact that only the upper yellow edge in the EBG goes to the green EBG node. On the other hand, the pink X-graph nodes are only connected to the lower yellow node for the same reason. The black nodes are not connected to other nodes because they represent a highway exit maneuver.

As our chosen solver does not support sparse distance matrices, we create edges from every node to every node, or in other words, from every maneuver to every maneuver. We do not depict all edges in the figures, drawing only direct connections between maneuvers for readability reasons.

Practically, the edge objects are not created since all the information required from the edge is its weight. In our implementation, each maneuver has a method that returns the edge cost to the maneuver supplied as an argument.

In the third step, we define disjunctions for each passlet. In the TSP solution of

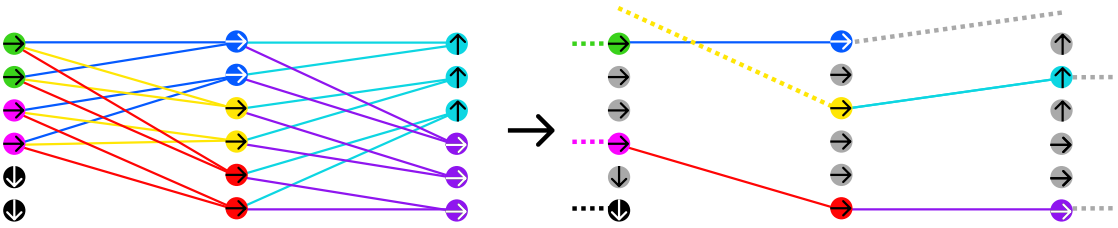


Figure 2.13.: Disjunctions resolution - X-Graph form

the original edge-based graph, each passlet is visited exactly once. We want to have the same property in the TSP result of the X-Graph with the help of disjunctions. A disjunction is a rule for the solver to pick at least one of the nodes defined in the disjunction or pay the penalty if none is selected. A complete definition of disjunction and an example can be found in the introduction. We force the optimizer to pick exactly one node in the disjunction by setting a penalty significantly higher than all other costs in the graph. If we do ordinary TSP without disjunctions, all maneuvers would we visited and that would mean we do some passes multiple times because maneuvers can share the same pass.

Figure 2.13 shows an example of resolved disjunctions. Disjunctions are nodes that have the same color. Out of each disjunction, exactly one node is chosen by the optimizer. The yellow maneuver is visited not from a directly connected previous maneuver, but via a detour from some other maneuver. After visiting the blue maneuver, the optimizer has no choice of directly connected maneuvers as both the cyan and the purple maneuvers are already resolved. The optimizer connects the blue maneuver with another maneuver (not in the picture) via a detour (dashed gray line).

For each passlet, all maneuvers that have that passlet as a destination are a part of the disjunction. That means that exactly one maneuver with that passlet as a destination is guaranteed to be present in the routing result.

Figure 2.14 shows the same disjunction resolution as in figure 2.13, but in routing problem form. The detour leading to the yellow maneuver goes through the green passlet, because the chosen upper yellow maneuver consists of the green and yellow passlets.

In the fourth step, we define the edge weights of our X-Graph. This is the most complex part of the algorithm. The weights are defined in two parts. First, the base cost is calculated, and then an additional penalty is applied.

The base cost is calculated as follows. If the second passlet's segment of the source maneuver is the same as the first passlet's segment of the destination maneuver, the base cost is zero. Otherwise, the base cost equals road distance or duration from the second segment of the source maneuver to the first segment of the destination

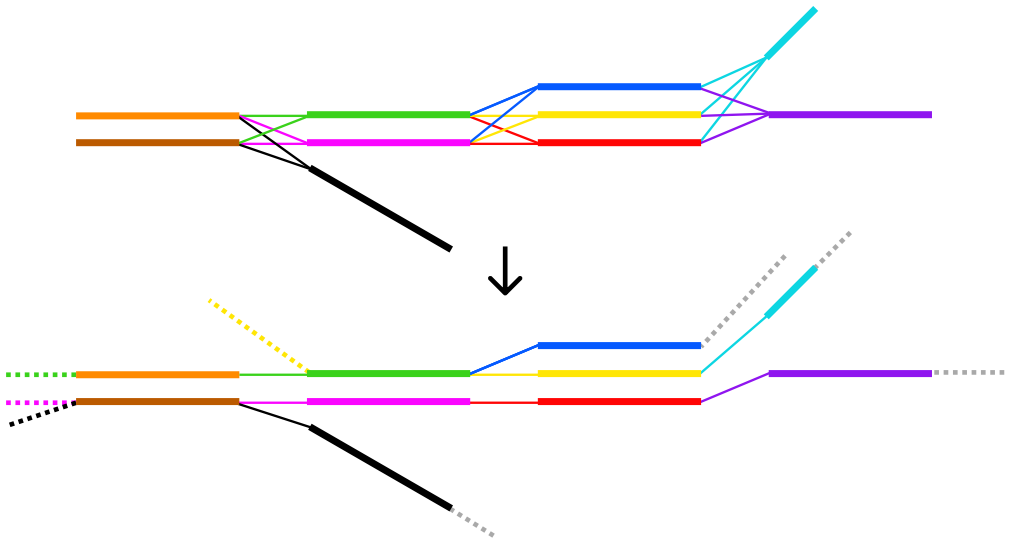


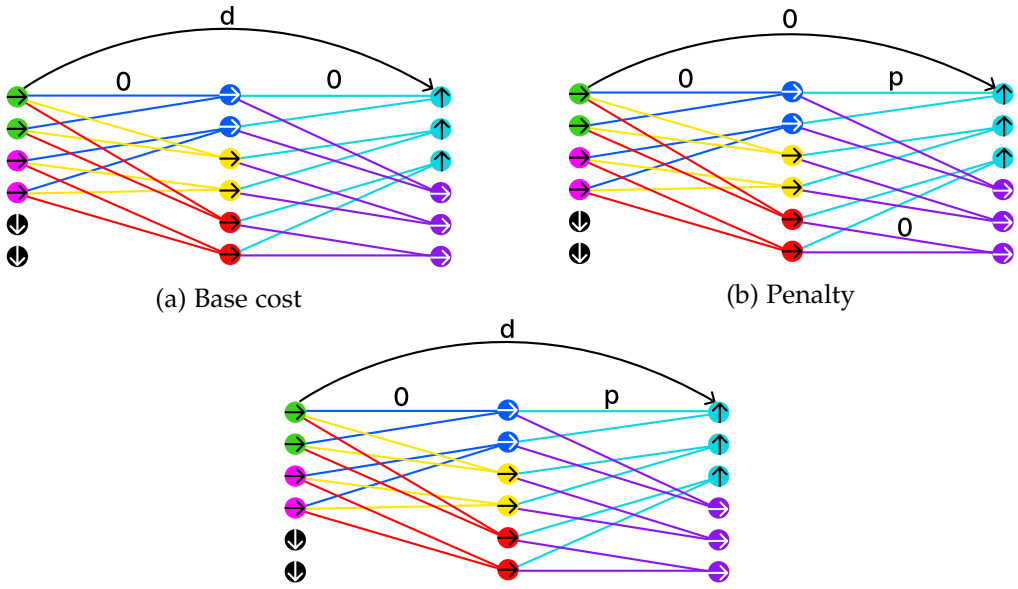
Figure 2.14.: Disjunctions resolution - RP form

maneuver. As explained in the second paragraph of section 2.2.1, the weight of the nodes in VRPs, where a constant set of locations needs to be visited, does not influence the routing results. Therefore we ignore the own distance or duration of maneuvers and only consider distances or durations between them. These can be referred to as detours.

Figure 2.15a gives an example of the base cost calculation in X-Graph. d is the road distance or duration. Successive maneuvers have a base cost of zero.

The distance or duration between maneuvers calculation is illustrated in figure 2.16. For example, the distance between maneuvers orange->green and green->blue is zero since they share a common segment. In contrast, the distance between maneuvers blue->cyan and orange->green equals the distance between the cyan passlet's segment and the orange passlet's segment.

The penalty is computed in the following way. If the source maneuver's direction is straight and the destination maneuver's direction is non-straight (left, right, u-turn, etc.), the penalty is X. If the source maneuver's direction is non-straight and the destination maneuver's direction is straight, the penalty is Y. Otherwise, the penalty is 0. Having two separate penalties allows for more flexibility and more granular optimizer tuning. In general, X should be higher than Y, as cases like turn, straight, turn, turn... are inevitable in many environments, while it is almost always possible to go straight after going straight. The penalty is applied only to directly connected maneuvers (maneuvers sharing a segment).e



(a) Base cost

(b) Penalty

(c) Combined cost

Figure 2.15.: X-Graph edge costs

In figure 2.15b one can see three stages of penalty calculation. Straight to straight transition is worth 0, while straight to left transition results in a penalty p.

The weight of the edge is a sum of the base cost and the penalty and is calculated lazily at runtime, as a complete X-Graph matrix for a modest routing problem can contain tens of millions of entries. Figure 2.15c shows how the base cost and penalty from two previous examples are combined.

2.2.3. Optimization performance considerations

As known from the graph theory, the number of nodes in the line graph equals the number of edges in the original graph. As the average branching factor of routing problems is larger than 1, the number of nodes in the X-Graph is larger than the number of nodes in the edge-based graph.

The *average branching factor* (ABF) of EBG is the average number of nodes directly reachable from a node in the graph. We calculated ABF for different routing problems, including highway, urban and mixed scenarios. Figure 2.17 shows the average branching factors for the scenarios. Higher ABF in highway scenarios can be explained by the higher average number of passlets on highways. It would be reasonable to expect the average branching factor to be higher in urban scenarios, as there are more intersections.

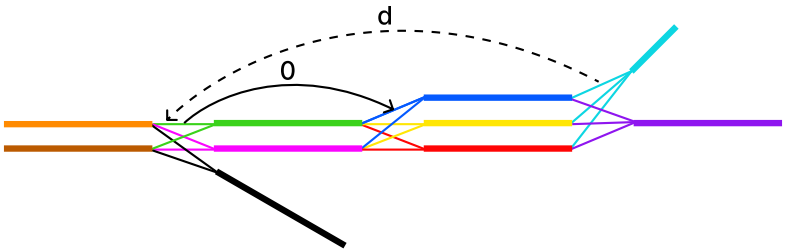


Figure 2.16.: Distances or durations between maneuvers

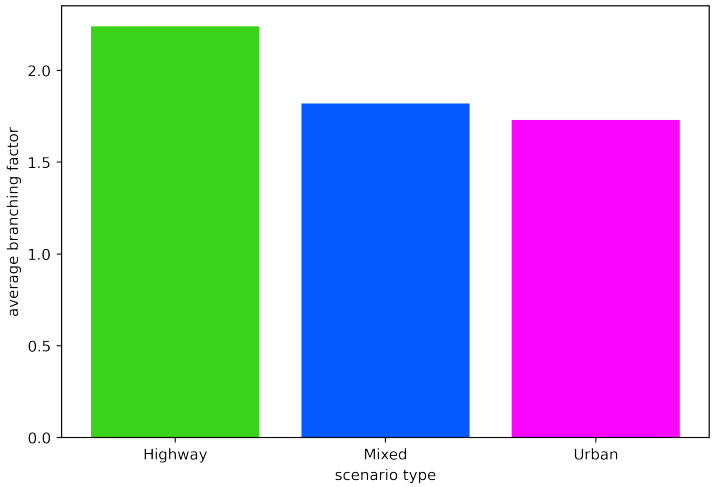


Figure 2.17.: Average branching factor for different scenarios

However, in practice, the higher average number of lanes on highways outweighs the effect of the higher number of intersections in urban scenarios. The ABF in a mixed scenario with both highways and urban roads lays between two other results, as could be expected.

Figures A.1, A.2 and A.3 show the routing problems used for measuring ABF. The routing problems have the correct number of lanes specified for each segment. The routing problems are chosen arbitrarily without any specific criteria.

An important observation about the X-Graph is that the number of disjunctions equals the number of nodes in the edge-based graph and the number of nodes in the VRP solution of the X-Graph. The average size of a disjunction is the average backward branching factor of the X-Graph. Thus, the optimization performance depends on the branching factor of the graph.

2.2.4. Optimizer setup

Google OR-Tools Optimizer was used to solve Vehicle Routing Problems. The optimizer supports different types of VRPs, including VRPs with dropping visits. It provides a powerful concept of disjunctions that allow the optimizer to choose which nodes will be present in the resulting tour.

The input of the optimizer is a list of nodes that need to be visited. In our case, this is a list of maneuvers. The nodes are grouped into sets of disjunctions. Other options include the first solution strategy, local search metaheuristic, and maximal optimization duration. They are described in detail below.

The result of a TSP optimization is usually a round trip, meaning that the vehicle returns to the starting node at the end of the journey. OR-Tools allows users to select the last node. However, this is an undesired behavior for our use case, as we just want to visit all passlets and do not want to specify the last maneuver beforehand. To get around this limitation, we create an extra last node and set it as the destination node in the solver's settings. We also define the distances and durations to this node from all other nodes as 0.

Another restriction of the optimizer is the need to select a starting node in advance. If we chose the starting node out of all routing problem's nodes, it would negatively impact the routing result, as even an intelligent selection of the first node might be suboptimal. It influences the routing result downstream and limits the optimizer's choices. To solve the problem, we define an extra starting node and set it as the source in the solver's settings. We further set to 0 all distances and durations from this node to all other nodes.

OR-Tools optimizer provides the user with a choice of a first solution strategy and a local search strategy. They influence the quality of the end result and the optimization duration. It is important to pick the strategies that are best suited for the optimization problem one is trying to solve.

A local search strategy (LSS) is employed by the optimizer during the local search phase. It is a rule that helps the solver escape local minima. OR-Tools implements several well-known local search options:

- Greedy Descent
- Guided Local Search
- Simulated Annealing
- Tabu Search

Figure 2.18 compares performances of four local search strategies. The optimization result is a sum of all edge weights in the final route. The result consists of base costs

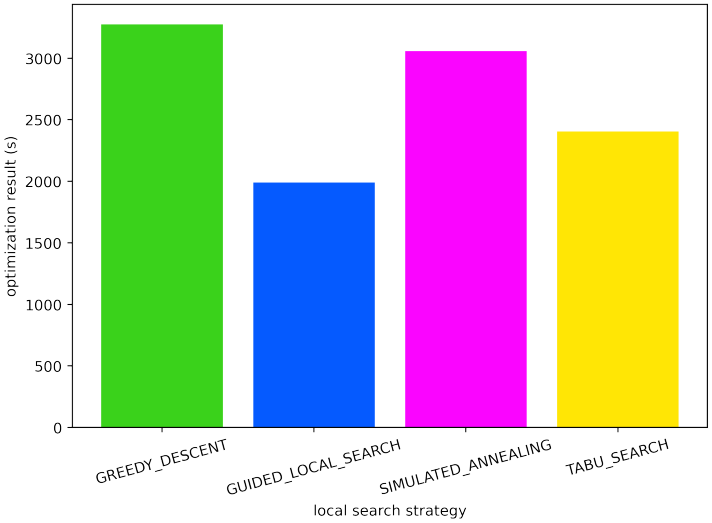


Figure 2.18.: Optimization results of different local search strategies

and penalties between all the tour's maneuvers. The test was performed with a time limit of 10 minutes, which is enough for the optimizer to stabilize, signifying that the result does not improve for several minutes, practically meaning that a further improvement is unlikely. Automatic first solution strategy was used in all tests. Each strategy and its results are briefly described in the following.

The first LSS OR-Tools offers is Greedy Descent (GD). It accepts local search neighbors until a local minimum is reached. This search strategy returns quicker than the other strategies because it does not try to escape local minima. In our tests, Greedy Descent returned after around 3s on average, delivering the worst optimization performance of all LSS.

Guided Local Search (GLS)[VTA10] is an LSS currently used as the default local search strategy in AtlaRoute. The strategy uses penalties to escape local minima and plateaus. It shows the best result out of all 4 LSS, outperforming the next best strategy by 17%.

Simulated Annealing (SA)[HJJ06] is a well-known technique for approximating the global optimum of a given function. It accepts a worse solution with a probability that decreases with time, similar to the metallurgical process of annealing, where the attributes of a material depend on its thermodynamic free energy. The LSS delivers the third-best result, which is 50% higher than the best result.

The last strategy we tested is Tabu Search (TS)[GLM08]. It escapes local minima by accepting worse results and introducing prohibitions (henceforth the term tabu) to stop

the optimizer from returning to the previously visited solutions. The LSS delivers the second-best result, being outperformed only by the Guided Local Search.

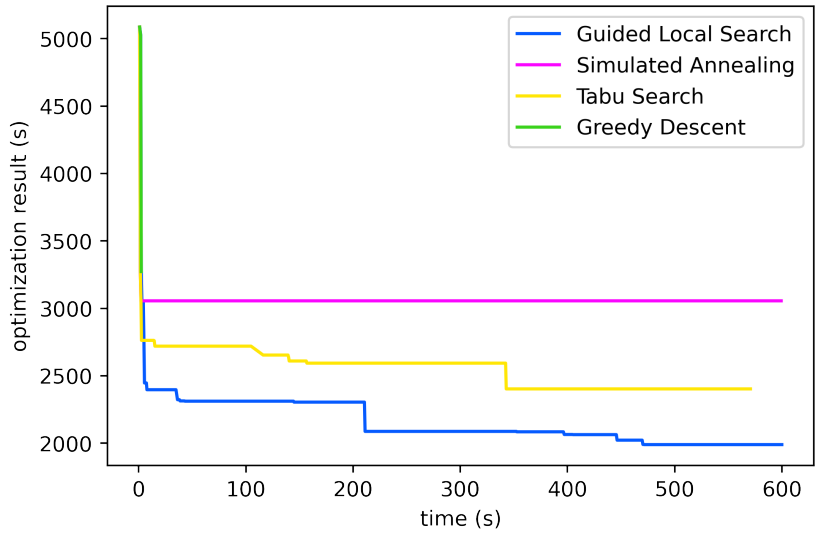


Figure 2.19.: Optimization histories of different LSS

Figure 2.19 shows the optimization histories of the four LSS. The first thing one notices is that Greedy Descent terminated very quickly, achieving only a mediocre result. Another notable feature is the inability of Simulated Annealing to improve on one of the first results. Both Guided Local Search and Tabu Search had a significant drop of around 200s, but GLS found that improvement around 2 minutes earlier. GLS and TS start plateauing in the last several minutes of the search, meaning that the optimizer stabilized and is unlikely to find an improvement.

First solution strategy (FSS) is a strategy used by the optimizer to create the initial solution. OR-Tools provides multiple predefined first solution strategies. Since there are many of them, we are not going to discuss each one in detail but only compare some of them and check whether the automatic option chooses the best strategy for our particular optimization problem. We tested the following first solution strategies:

- Automatic
- Path Cheapest Arc
- Christofides[Kyl15]
- Parallel Cheapest Insertion

- Savings[SAP17]
- Local Cheapest Insertion[RSL09]
- First Unbound Min Value
- Local Cheapest Arc
- All Unperformed

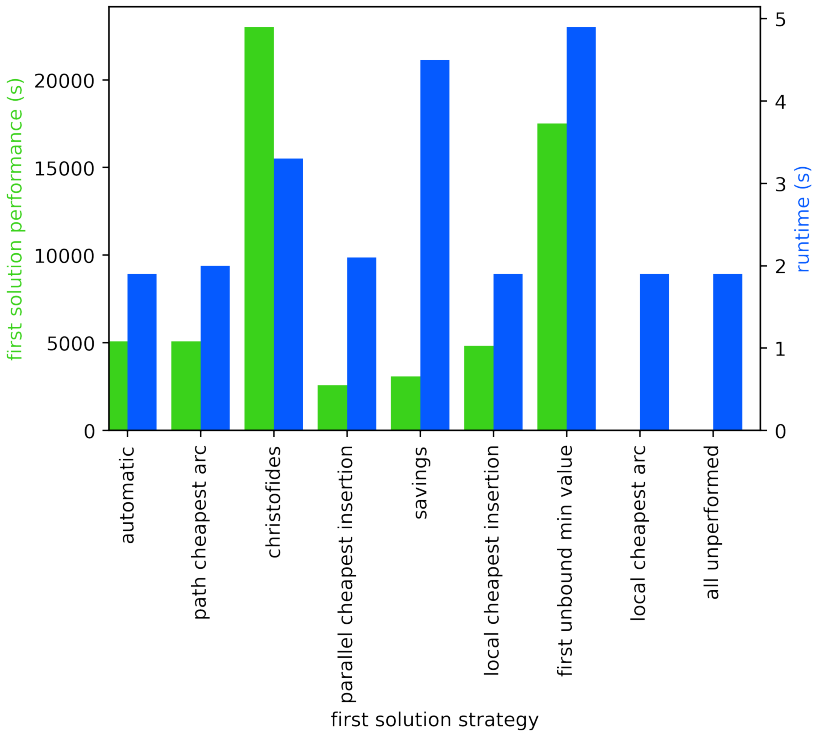


Figure 2.20.: Performances of different FSS

We picked strategies that do not require providing additional callbacks and which found a solution in the first 20 seconds. Local Cheapest Arc and All Unperformed strategies did not resolve disjunctions without getting penalties, so we did not include their first solution performances in comparison.

Figure 2.20 shows the performance comparison of different first solution strategies. The first solution performance is shown in green, and time until the first solution in blue. The test was performed on a MacBook Pro with M1 ARM processor. Parallel

Cheapest Insertion shows the best first solution of 2575s, Savings delivers the second-best solution of 3084s, and Local Cheapest Insertion is the third-best with the result of 4818s.

The Automatic strategy selected Path Cheapest Arc, the fourth-best option, delivering a result 1.9 times larger than the best solution. The same test was conducted for AtlaRoute and optimization of routing problems without disjunctions. In that test, the Automatic strategy always picked the first or the second-best first solution strategy.

The test shows significant differences in the runtime of the strategies. Six fastest strategies finished in less than 2 seconds, while the other three finished in under 5 seconds. The difference is practically insignificant, as the optimization takes much more time.

3. Evaluation

In this chapter, we evaluate the proposed approaches and compare them with the existing optimizer of AtlaRoute. We also check whether the proposed solutions solve the problem of inharmonious maneuvers by introducing a metric that measures the intensity of the problem.

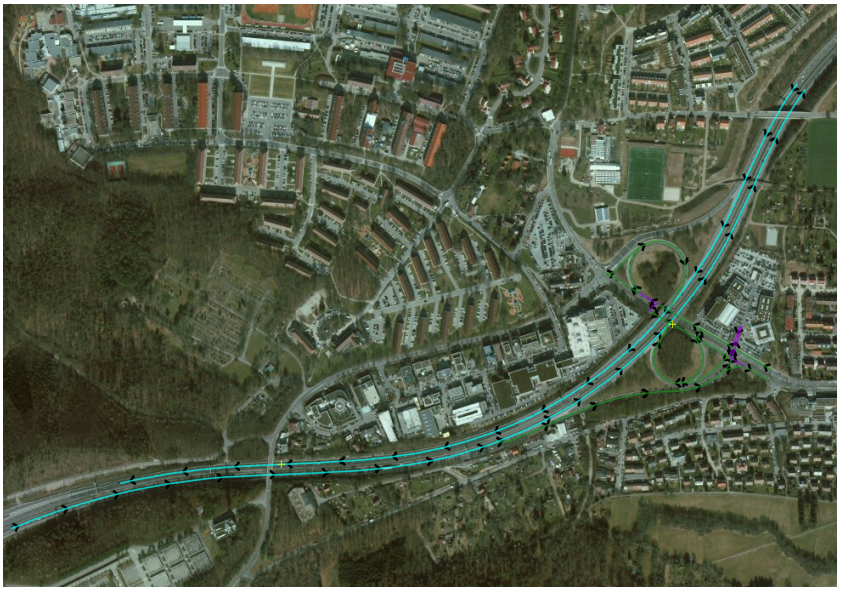


Figure 3.1.: Mixed routing problem scenario

For the evaluation, we use a routing problem with both a highway part and an urban part. The RP is depicted in figure 3.1. It has the correct number of lanes specified for each segment. The routing problem was chosen because it represents a typical routing scenario for Atlatec. The largest part of the routing problem is a highway with entries and exits, but there is also a small urban part in the eastern half of the picture. The current version of AtlaRoute produces a route with severe inharmonious maneuvers for this routing problem, as shown later in the chapter.

3.1. Optimization duration and required resources

3.1.1. Measurement technique

An important metric for a routing system is route creation time. Commercial navigation systems, such as Google Maps[1], produce routes in hundreds of milliseconds, including network communication. However, a comparison between off-the-shelf and our routing system is not reasonable since ordinary navigation suites do not solve Vehicle Routing Problems, which takes the largest portion of the routing time. In this section, we compare route creation times of the current AtlaRoute, lane-level topology approach, and X-Graph approach.

The route creation process includes routing problem preparation, graph transformation, and optimization, which is the longest part of the process. It is, therefore, essential to reasonably define when the optimization should be considered done. Currently, AtlaRoute does not decide when the optimization should be terminated. This job is delegated to the driver who uses the system. As there is no existing criterium for the termination of optimization, we introduce a new one.

We define the optimization as done if the optimization metric does not improve by at least 1% in two minutes. We call this state stable and say that the optimization has stabilized. The definition implies that the stable minutes will be included in the optimization duration. This does not constitute a problem since we can subtract this time from the optimization durations before comparing them. The threshold of two minutes was found empirically. It suits the typical size and complexity of an AtlaRoute routing problem. As the RP chosen for the evaluation represents a typical AtlaRoute routing problem, the threshold of two minutes should fit perfectly.

3.1.2. Testing setup

All tests are performed on the same machine, MacBook Pro with M1[2] ARM processor. No other CPU-intensive tasks are run on the machine during the testing. Even though OR-Tools does not utilize multiple CPU cores, the tests are not run in parallel to avoid unfair scheduling that might happen on big.LITTLE[3] architecture such as of M1 chip. The test is run using the normal Python interpreter instead of in a Jupyter[4] notebook to avoid any performance decreases that the Jupyter notebook and iPython might cause.

We use the Parallel Cheapest Insertion first solution strategy and Guided Local Search local search metaheuristic, as they have shown the best results in our strategies

[1] `https://maps.google.com`
[2] `https://en.wikipedia.org/wiki/Apple_M1`
[3] `https://en.wikipedia.org/wiki/ARM_big.LITTLE`
[4] `https://www.jupyter.org`

comparison in the previous chapter. The maximum optimization duration is picked arbitrarily to guarantee that the optimization stabilizes during the defined time. The penalties for the X-Graph approach are also chosen arbitrarily and lay close to the optimal penalties for this routing problem. The choice of optimal penalties is made in the next chapter.

We also measure the memory used by the Python process to see if any of the approaches require more resources than the other ones. To do that, we start a separate thread that constantly polls the process's memory usage using Python's resource library and saves the readings to a list.

3.1.3. Results and comparison

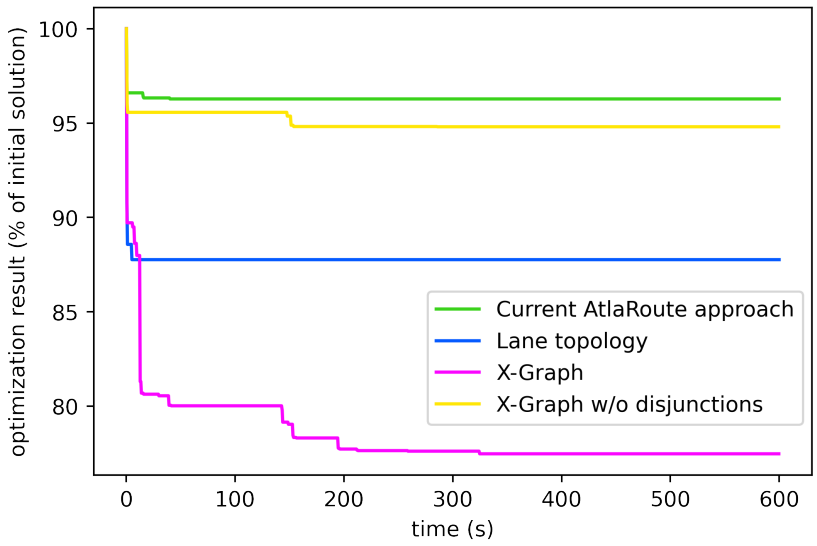


Figure 3.2.: Approaches optimization comparison

Figure 3.2 shows the optimization histories of the three approaches. It is clear from the picture that both the current approach and lane-topology approach reached the stable state in the first seconds, while X-Graph continued improving for three minutes. This means that X-Graph is significantly more complex than the other two graphs.

Current AtlaRoute pass-level optimization reached the result of 1791 after 500 milliseconds and only improved to 1785 after 39 seconds. Therefore, according to the definition, the stable state was reached after 0.5 seconds. The final result is just 3% better than the initial solution, signifying a good choice of the first solution strategy.

The lane-level topology optimization stabilized after 5 seconds and reached the result

of 3687. After that, the optimizer could not find further improvement. The improvement over the initial solution is 12%, which is more than the improvement of the pass-level optimization. It can be explained by the higher complexity of the lane-level topology graph and the higher number of restrictions. The first solution strategy is more effective in simpler, less restricted graphs.

The X-Graph optimization showed significant improvements until the second 155 of the optimization duration. After that, the stable state was entered with the result of 2803 and continued with a series of small decreases of less than 1%, which were not enough to exit the stable state. The improvement over the initial result is 21.7%. It is the largest improvement of all three optimizations. With the high complexity of the X-Graph and the high number of locations, the optimizer requires significantly more time compared to other graphs.

The X-Graph optimization without disjunctions required 1.2 s to stabilize. After 154 s from the optimization start, there was a 0.7% drop in the result. One can see from the figure that the X-Graph optimization requires more time due to disjunctions.

It is important to note that the optimization results are not comparable between approaches because they include approach-specific penalties. The comparison of route distances and duration of the three approaches is made in the next chapter.

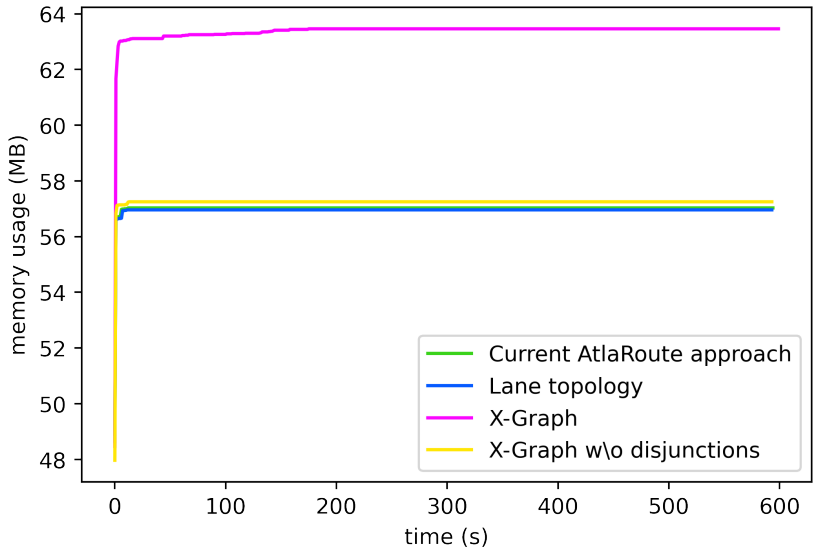


Figure 3.3.: Approaches memory usage

Figure 3.3 shows the comparison of the memory usage of the Python processes during the optimization of the three approaches. The samples were taken at a rate of 1

Metrics/Penalties	600,300	400,200	200,200	200,100	100,50	120,40	120,0
Distance (m)	49903	49902	45218	44856	40166	39753	39664
Duration (s)	3400	3400	3134	3082	2840	2788	2783
ASL (m)	511	448	489	471	542	563	572

Table 3.1.: Penalties search

Hz. All three approaches have a rapid memory consumption increase in the first second, which can be attributed to the loading of the durations matrix and the initialization of the optimizer. Current and lane topology approaches show a similar stable memory usage of around 57 MB after the initial climb, while the X-Graph slightly increases the memory usage in the first 150 seconds after the initial climb and then keeps steady at 63.5 MB. The difference in the memory consumption between the approaches can be explained by the larger number of nodes in the X-Graph and the disjunctions, which are not present in the other two approaches.

3.2. X-Graph penalties choice

The choice of the X-Graph penalties affects many characteristics of the resulting route and is one of the most critical parameters of the optimization. In this section, we compare different combinations of X-Graph penalties and use metrics from the next sections to pick the optimal set of penalties. Before reading this, we kindly ask the reader to familiarize themselves with the metrics definitions from the following sections. The decision to put this section before the others is justified by the fact that the optimal penalties found in this section are used in the later sections.

There are two penalties in the X-Graph that we can adjust: straight -> non-straight penalty and non-straight -> straight penalty. The penalties directly affect the behavior of the optimizer and the properties of the resulting route we want to achieve. For example, if the straight -> non-straight penalty is too low, the optimizer is encouraged to choose a faster or shorter route instead of the route with less inharmonious maneuvers. Also, the parameters are not independent, meaning that the effect of the combination is not entirely predictable from the effects of the single parameter changes. This makes the parameter choice process more complex.

The search for the optimal penalties was performed gradually from the higher sensible bound of both penalties. When an optimal region was reached, we decreased the steps of both penalties and tried slightly different combinations to improve the result further.

Table 3.1 shows several steps from the search for the optimal penalties. As one can

see from the figure, the initial penalties are too high, and the decrease of both penalties leads first to the worsening and then to improving ALS. When the region of straight -> non-straight = 100 s, non-straight -> straight = 50 s is reached, the change in penalties start impacting the metrics less. From that point on, we reduce the size of steps and try many combinations of penalties, further improving the result.

The optimal combination of X-Graph optimization penalties is found to be straight -> non-straight = 120 s, non-straight -> straight = 0 s. This combination delivers a result that is best both in distance and duration and ASL. This combination of penalties is very particular because the second penalty is 0. It means that the best results are achieved when the non-straight -> straight transitions are not penalized. We tried other combinations with the second penalty set to zero but found no improvement.

To check if the chosen penalties are universally optimal, we performed the same search on other routing problems. Table A.1 shows the search results for the smaller version of routing problem depicted in figure A.2. The smaller version of the routing problem has 40% of the original RP to decrease optimization duration. As seen from the table, the chosen penalties also perform best for the other scenario, showing their universality.

3.3. Route length and duration

In this section, we compare the distance and duration of routes produced by the three approaches. The visual comparison of routes is done in the next section but focuses mainly on the phenomenon of inharmonious maneuvers.

The optimization results are taken from the previous chapter. The optimization order includes the special first and last nodes, which we remove before measuring route distance and duration.

There are two possibilities to measure route distances and durations. The first option is to loop over the segments in the optimization result and calculate the total distance and duration of the route manually using the data from the matrix and segments' own distances and traversal durations. The second option is to query the routes from OSRM using segment IDs and get the total distance and duration from the route's summary. As both options are supposed to produce the same results, we choose the second option out of practicality since the same routes can be later reused in the next sections.

Figure 3.4 shows the distances and durations of routes created with the three approaches. It is not surprising that the current approach is the shortest and the fastest route of all because it was optimized exclusively for duration, unlike other approaches. The X-Graph approach shows a result 2.2% larger than the lane-topology approach. This can be explained by the fact that the optimizer often prefers taking a detour to

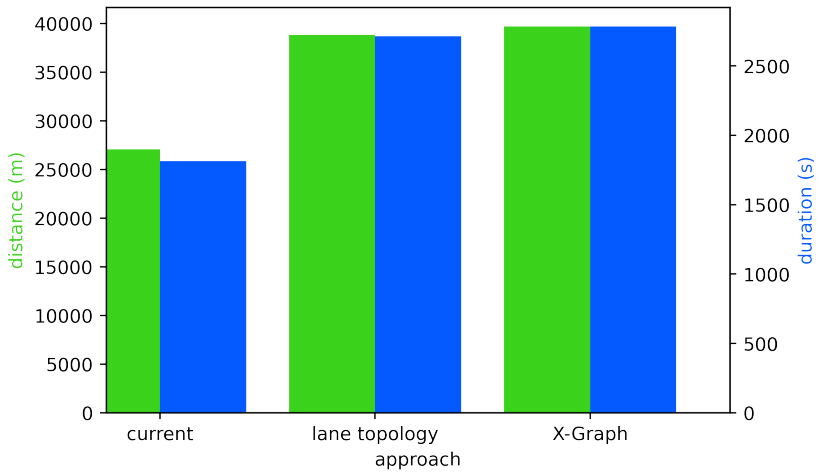


Figure 3.4.: Routes comparison

getting a penalty for a straight -> non-straight transition. The absolute difference of 1 minute and 10 seconds between X-Graph and lane topology approaches is negligible for a one-hour-long recording mission.

The duration difference between the current and X-Graph approaches is 53.5% or 16 minutes in absolute terms. Considering the fact that a missed maneuver caused by the lack of time to change lanes might cause a 10-minute detour, the more comfortable but longer route would be preferred in practice.

3.4. Measuring inharmonious maneuvers

3.4.1. Metric for inharmonious maneuvers

In this section, we introduce a metric that measures the intensity of inharmonious maneuvers in a given routing problem and compare the proposed approaches with the current version of AtlaRoute. We show that the new approaches reduce the intensity of inharmonious maneuvers and, therefore, improve driver's comfort and decrease the probability that some parts of the road network are not recorded.

The proposed metric for measuring the intensity of inharmonious maneuvers is the *average straight length* (ASL). It is defined as the average length of passes that do not contain turn maneuvers. Under a pass we understand a successive sequence of segments. For example, a straight pass might begin after merging onto a highway and end right before the highway exit. Another example is a straight piece of road consisting of multiple segments and ending without successive segments. It is important

to note that the metric is applied to optimizer results (sequences of segments), not to actual routes containing detours because the optimizer is unaware of the straights and maneuvers in the detours.

The metric measures how much time or distance a driver can drive on average without making turns. It directly impacts the comfort of the driver and the quality of the recordings. If the metric is low, a driver needs to take different turns often and, therefore, change lanes. Excessive lane changes are detrimental to the recording quality and put undue pressure on the driver.

The metric is applied to routes resulting from stabilized optimizations, meaning that the maximal optimization duration does not constrain the result of optimization. The routing problem for the comparison is the same as in the last section and is depicted in figure 3.1.

3.4.2. Comparison of approaches

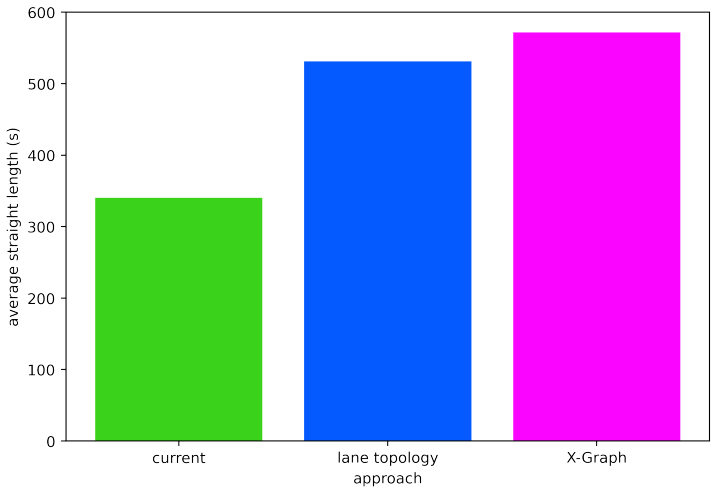
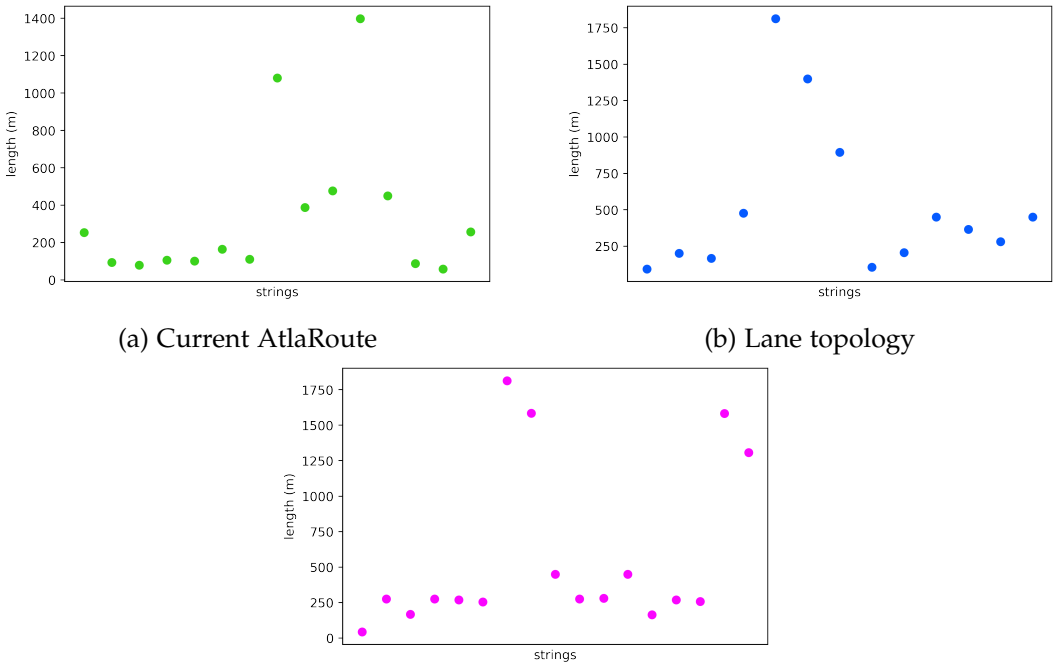


Figure 3.5.: Average straight length

Figure 3.5 shows the average straight lengths of the three approaches. It is clear from the image that the X-Graph outperforms other approaches, with the lane topology approach showing the second-best result. Let us discuss the results in more detail.

The current AtlaRoute approach shows the lowest result of 340 m average straight length. It is not surprising because the standard AtlaRoute optimization does not address the problem of inharmonious maneuvers. Even though it is the smallest result, it is still relatively large. This hints at the nature of the routing problem, where long highway sections prevail.



(a) Current AtlaRoute

(b) Lane topology

(c) X-Graph

Figure 3.6.: Straights lengths

The lane topology approach shows the second-best result of the three approaches. It outperforms the current approach by 56% and is 41 meters smaller than the X-Graph result.

The X-Graph approach shows the best result of 572 meters. It is 68.1% larger than the result of the current AtlaRoute approach, making it a significant improvement over the baseline. The dominance of the X-Graph can be explained by the slightly longer route that allows the X-Graph to increase the ASL.

3.4.3. Visual comparison of approaches

Figure 3.6 shows the lengths of the straights of routes resulting from the three approaches. The dots in the scatter charts represent straights; the order of the dots corresponds to the order of the straights in the routes. The routing problem used for the comparison is the same as in the previous chapter.

As can be seen in the figure, X-Graph has two distinct pairs of successive straights in the upper part of the chart. They represent complete traversals of the highway from south-west to north-west and the other way round. The traversals are highlighted in

Figure 3.7.: X-Graph complete highway traversals

the figure 3.7. It is exactly the behavior we wanted to achieve, as our drivers would normally first try to record the main parts of the highway and then record the exits and entries of the highway.

The routing problem used for the comparison has three lanes in most parts of the highway and two lanes in some other parts. This means that we expect to see three complete passes in each highway direction. However, the X-Graph straights sequence only shows two complete traversals. This can be explained by the optimizer accessing the parts of the highway requiring the third traversal using detours. In fact, three complete traversals of the highway are impossible since there are only two lanes in some parts of the highway. The chart does not show straights in the detours because we only analyze optimization results (sequences of segments), not the complete routes from OSRM that include detours.

The lane topology approach also has a clear complete traversal of the highway in both directions. However, it lacks the second complete traversal. The current AtlaRoute approach only includes two nonconsecutive complete highway traversals, but each includes only one way.

Another thing that can be read from the figure is the number of short straights. In the X-Graph approach, there is only one straight shorter than 150 m, while the lane topology result contains three, and the current approach result contains seven such straights. The lengths of straights in the X-Graph solution are more homogeneous than

in other solutions and can be easily clustered into long straights and medium-length straights, significantly reducing the problem of inharmonious maneuvers.

3.5. Conclusion

In this section, we further evaluate the proposed approaches and assess their applicability in production environments. We also suggest ideas for future work.

3.5.1. X-Graph approach evaluation

X-Graph approach shows the best performance in reducing the intensity of inharmonious maneuvers and only lacks a few percent to lane topology approach in route distance and duration. X-Graph shows a major improvement in route drivability over the baseline.

The main downside of the X-Graph approach is the long optimization duration. For the same middle-sized routing problem, X-Graph requires 31 times more optimization time than the lane topology approach or, in absolute terms, 155 s instead of the 5 seconds. However, as the average route duration at Atlatec is several hours, the optimization time of several minutes is not an obstacle. The improved route drivability might decrease the number of unrecorded route network parts, thus decreasing the extra time the company's drivers spend to record the missed parts. However, the significantly increased optimization time was unexpected to us. We see a potential for improving the optimization duration by picking a better solver in future research.

X-Graph is an innovative approach for route optimization with higher-order penalties. As there is no directly related research, there are many aspects of the approach that can be improved or extended in further work. This thesis focuses on penalties based on the specific sequences of maneuvers. It is, however, possible to use other penalties, such as based on sequences of average speeds on the segments. The choice of penalties affects different properties of the resulting route. Other exciting and practically important properties can be achieved by further researching different penalties in the X-Graph approach.

3.5.2. Lane-level topology approach evaluation

Lane-level topology approach shows good performance in all conducted comparisons. It reduces the problem of inharmonious maneuvers and improves the drivability of the routes compared to the baseline. We consider the approach the best solution to the problem, but at the time of writing it is not practically viable.

The approach shows better route distance and duration than X-Graph while having an ASL only 7% smaller than X-Graph. The optimization duration of the lane-level topology approach is similar to AtlaRoute's optimization duration. It is significantly shorter than the optimization time of the X-Graph.

However, the lane topology approach has a drawback preventing it from industrial application. The problem with the approach is that the lane-level topology graph created with the help of heuristics is not entirely topologically correct and requires a substantial amount of manual editing. Without the correct topology, a good routing solution is not possible.

The heuristics proposed do not cover all possible cases and do not guarantee the correctness of the produced topology. The requirement to manually check and correct the heuristics results considerably prolongs and complicates the process of routing problem creation and makes it practically unviable for application at Atlatec.

We believe the process of lane topology creation can be improved in future work to produce production-quality results with the help of more complex heuristics and graph algorithms. Another possible addition to improving the approach is the automatic topology validation.

A. Appendix

Figure A.1.: Highway routing problem scenario

Metrics/Penalties	600,200	400,100	300,150	100,50	110,40	120,0	150,0
Distance (m)	50658	47395	48740	45711	46972	44195	46084
Duration (s)	5286	4905	5063	4759	4861	4738	4792
ASL (m)	281	310	303	300	291	315	311

Table A.1.: Penalties search - mixed scenario

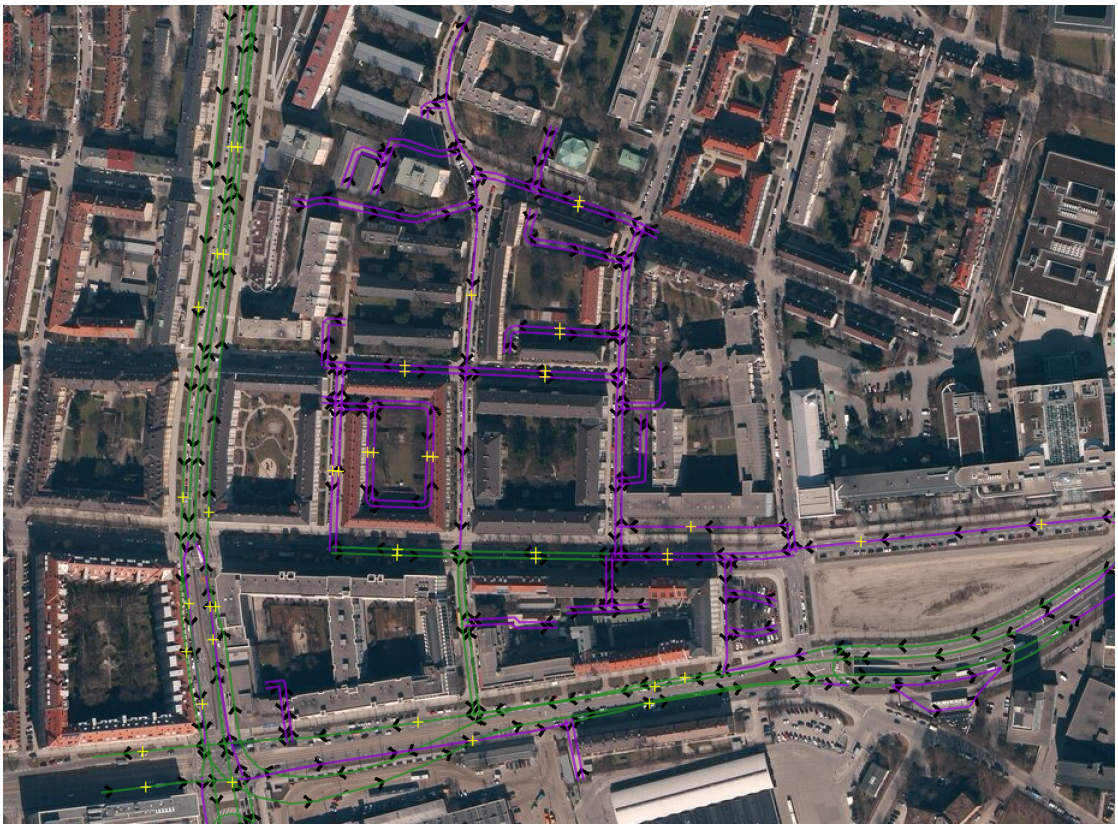


Figure A.2.: Mixed routing problem scenario

Figure A.3.: Urban routing problem scenario

List of Figures

List of Tables

Bibliography

[Dav17] A. M. David Poole. *Artificial Intelligence: Foundations of Computational Agents*. Cambridge University Press, 2017.

[Edg17] M. P. Edgar Goodaire. *Discrete mathematics with graph theory*. Pearson, 2017.

[G G02] A. P. P. G. Gutin. *The Traveling Salesman Problem and Its Variations*. Springer-Verlag GmbH, 2002. ISBN: 0387444599.

[GLM08] F. Glover, M. Laguna, and R. Marti. *Tabu Search*. Vol. 16. July 2008. DOI: 10.1007/978-1-4615-6089-0.

[Gol08] S. R. u. E. A. W. Golden Bruce L. *The Vehicle Routing Problem: Latest Advances and New Challenges*. Springer US, 2008. ISBN: 0387777776.

[HJJ06] D. Henderson, S. Jacobson, and A. Johnson. "The Theory and Practice of Simulated Annealing." In: Apr. 2006, pp. 287–319. DOI: 10.1007/0-306-48056-5_10.

[Kyl15] D. P. W. Kyle Genova. "An Experimental Evaluation of the Best-of-Many Christofides' Algorithm for the Traveling Salesman Problem." In: (June 2015).

[Liu+17] C. Liu, K. Jiang, Z. Xiao, Z. Cao, and D. Yang. "Lane-level route planning based on a multi-layer map model." In: *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*. 2017, pp. 1–7. DOI: 10.1109/ITSC.2017.8317681.

[Mat+16] R. Matthaei, A. Reschka, J. Rieken, F. Dierkes, S. Ulbrich, T. Winkle, and M. Maurer. *Autonomous Driving*. Jan. 2016, pp. 1519–1556. ISBN: 978-3-319-12352-3. DOI: 10.1007/978-3-319-09840-1_61-1.

[Ope17] OpenStreetMap contributors. *Planet dump retrieved from https://planet.osm.org*. https://www.openstreetmap.org. 2017.

[Pan+20] D. Pannen, M. Liebner, W. Hempel, and W. Burgard. "How to Keep HD Maps for Automated Driving Up To Date." In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pp. 2288–2294. DOI: 10.1109/ICRA40945.2020.9197419.

[Per11] L. Perron. "Operations Research and Constraint Programming at Google." In: Jan. 2011, p. 2. ISBN: 978-3-642-23785-0. DOI: 10.1007/978-3-642-23786-7_2.

[RSL09] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis. "An analysis of several heuristics for the traveling salesman problem." In: *Fundamental Problems in Computing: Essays in Honor of Professor Daniel J. Rosenkrantz*. Ed. by S. S. Ravi and S. K. Shukla. Dordrecht: Springer Netherlands, 2009, pp. 45–69. ISBN: 978-1-4020-9688-4. DOI: 10.1007/978-1-4020-9688-4_3.

[SAP17] K. Sörensen, F. Arnold, and D. Palhazi Cuervo. "A critical analysis of the "improved Clarke and Wright savings algorithm": Sörensen et al ." In: *International Transactions in Operational Research* 26 (Aug. 2017). DOI: 10.1111/itor.12443.

[TB12] C. Toth and D. Brzezinska. "MOVING TOWARD REAL-TIME MOBILE MAPPING: AUTONOMOUS VEHICLE NAVIGATION." In: (May 2012).

[VTA10] C. Voudouris, E. Tsang, and A. Alsheddy. "Guided Local Search." In: Sept. 2010, pp. 321–361. DOI: 10.1007/978-1-4419-1665-5_11.