Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät
Institut für Informatik

Design and Implementation of an Optimization System for Networked Rescue Mobility in Urban Traffic

Masterarbeit

zur Erlangung des akademischen Grades
Master of Science (M. Sc.)

eingereicht von: Jonas Trappe
geboren am: 16.5.1997
geboren in: Berlin

Gutachter/innen: Prof. Dr. Henning Meyerhenke
Sten Ruppe, M. Eng.

eingereicht am: ..... verteidigt am: .....

Abstract. Efficient and safe emergency response is a critical requirement in urban mobility systems. Emergency response drivers have to strike a balance between arriving at the emergency site quickly and avoiding dangerous traffic situations. This thesis addresses the challenge of forming timely and secure emergency corridors in urban environments by using Vehicle-to-Everything (V2X) technology. As part of the Gaia-X4 Advanced Mobility Services initiative, the RGS (Rettungsgassenservice, Emergency Corridor Service) controller computes optimal paths for emergency vehicles and communicates them to other road users. The thesis formulates the corridor generation as a dynamic simulation optimization problem, where the objective is to minimize emergency vehicle response time while ensuring safety and stability of traffic behavior. The controller uses the SUMO traffic simulator to predict the outcomes of traffic situations. It evaluates three optimization strategies: the Downhill-Simplex algorithm, a memetic algorithm, and a deep reinforcement learning agent, in both synthetic and real-world traffic scenarios. Additionally, the influence of non-cooperative vehicles on optimization performance is studied. Results demonstrate the ability of the controller to support emergency corridor formation in various traffic situations. While the reinforcement learning agent most effectively solves simple scenarios, the memetic algorithm excels at the most complex real-world scenario in the evaluation suite.

Contents

1 Introduction **1**

2 Preliminaries **4**
2.1 Optimization Problems 4
2.2 Connected Vehicles 5
2.3 Related Work 6

3 System Design **9**
3.1 RGS Pipeline architecture 9
3.2 Problem Definition 12
3.3 Controller Architecture 14
3.4 Simulation Adaptations 20

4 Optimization Algorithms **23**
4.1 Downhill-Simplex 23
4.2 Memetic Algorithm 26
4.3 Reinforcement Learning 30

5 Experimental Setup **35**
5.1 Evaluation Scenarios 35
5.2 Parameters for Controller and Simulation 40
5.3 Definition of Experiments 41

6 Experimental Results **43**
6.1 Relative Performance of Optimization Strategies 43
6.2 Impact of non-cooperative Vehicles on Optimization 52

7 Discussion **56**

8 Conclusion **59**

Appendix **v**
Figures vi

1 Introduction

Rescue mobility is a crucial aspect of any modern healthcare system. In the event of a medical emergency, treatment must often be provided on-site within a limited time frame. The total response time to an emergency is composed of the time between the emergency and the first report, the time taken by the operator to receive the report, the time taken to prepare the response, and finally, the time the emergency vehicles need to arrive at the scene. Rescue services typically calculate between 8 and 15 minutes for an emergency response in Germany [1], depending on the state and level of urbanization. German federal laws define a goal that 95% of all emergency responses shall be completed within the locally defined response times. However, this goal is not always met in reality: A 2017 analysis of day-time emergency response trips in Brunswick showed that only 86% of them would arrive within North Rhine-Westphalia's 13-minute response time [2]. Other reports show similar or even worse results, with up to 40% of all emergency responses not being completed in time in Berlin [1].

To achieve better response times, emergency drivers are permitted to ignore certain traffic rules to reach their destinations swiftly. These include driving at speeds beyond the speed limit or crossing red traffic lights. However, these special permissions make emergency responses challenging and also dangerous: an emergency vehicle making use of special permissions in Germany is eight times as likely to be involved in an accident [2].

Both the achieved response times and the likelihood of an accident are not only influenced by the emergency vehicles, but also by the reactions of other drivers. While German law defines a clear procedure on how to form rescue corridors outside of settlements[1], complex situations in urban traffic often require drivers to react more dynamically. Mistakes made in such situations negatively affect response times and lead to a higher risk of accidents. Emergency vehicles will lose one minute on average per response due to other drivers' wrong reactions [2].

As part of the Gaia-X initiative [3] for a joint European data infrastructure, the Gaia-X4 AMS (Advanced Mobility Services) aims to improve mobility-related topics through technological means. One use case of this project is the networked and secure emergency corridor, which shall mitigate the aforementioned problems through communication between vehicles and infrastructure. As autonomous and connected vehicles become more prevalent in traffic, new challenges and opportunities arise. Connected vehicles can share data and receive information from others to enhance situational awareness. Autonomous vehicles can increase safety by performing predictable maneuvers, but they need data to base their decisions on. Gaia-X4 AMS' emergency corridor use case is designed to inform vehicles ahead of time that an emergency vehicle is approaching and an emergency corridor needs to be formed. By doing so, the emergency vehicle can pass through the corridor swiftly when it arrives and faces a lower risk of accidents, as other vehicles have

[1] StVO §11 Section 2: "As soon as vehicles on motorways and on rural roads with at least two lanes in one direction are traveling at walking pace or when the vehicles are stationary, these vehicles must form a clear corridor between the leftmost lane and the lane immediately to the right of it for the passage of police and emergency vehicles.", translated from German from: `https://www.gesetze-im-internet.de/stvo_2013/__11.html`

already performed necessary evasive maneuvers.

Multiple components are required to address this use case: traffic information needs to be gathered, a rescue corridor has to be calculated based on that information, and the corridor must ultimately be communicated to vehicles that shall clear it. The German Aerospace Center (German: Deutsches Zentrum für Luft- und Raumfahrt, DLR) has developed a system for this task. It is called the **rescue corridor service** (German: Rettungsgassenservice, RGS). This thesis is focused on the **RGS controller**, the component of the RGS that is responsible for calculating an optimal emergency corridor.

The RGS supports urban rescue mobility by providing a recommended trajectory to emergency vehicles. Other vehicles are to clear a corridor around this trajectory, allowing safe passage for emergency vehicles. Finding such a trajectory is considered an optimization problem: the controller has to select one out of many possible trajectories based on the current traffic situation. Finding such a trajectory requires the RGS controller to make predictions about the behavior of vehicles. Should these predictions differ from reality, the output of the controller needs to be adjusted. The optimization needs to be performed in a limited time frame based on a dynamic traffic situation, making it a challenging problem.

Another issue is that some vehicles do not behave correctly in emergency situations. These are referred to as non-cooperative vehicles. Predicting their behavior is more difficult than for vehicles that comply with rules. Non-cooperative vehicles exist today, as not all drivers form rescue corridors correctly. In the RGS use case, non-cooperative vehicles can also be those incapable of receiving the messages sent by the RGS, possibly due to not having relevant technology installed. Their behavior is unpredictable to the system, leading to potentially worse optimization performance.

The scientific contribution of this thesis lies in the evaluation of different optimization techniques for the rescue corridor use case. Three different optimization techniques to solve the problem were selected in the study project preceding this thesis. The performance of evaluation techniques is evaluated on two types of traffic scenarios: synthetic and real-world. Synthetic scenarios are modeled after generic traffic situations, but are not explicitly designed after real locations, whereas real-world scenarios are based on existing locations and use real traffic data. The following research questions were formulated for this work:

- **RQ1:** How do selected optimization algorithms perform at guiding rescue corridor formation in synthetic traffic scenarios compared to each other?
- **RQ2:** How do selected optimization algorithms perform at guiding rescue corridor formation in a real-world traffic scenario compared to each other?
- **RQ3:** How do non-cooperative vehicles influence optimization algorithm performance?

This thesis provides insights into the viability of different optimization strategies for rescue corridor formation. It also provides an extensible architecture in which optimization techniques can be integrated. Compared to other approaches, which typically involve

traffic light sequence control, the proposed RGS controller assists rescue corridor formation on a microscopic level. It can be deployed in a decentralized manner and is designed as a general optimizer for any given urban traffic scenario.

Before detailing the specifics of the RGS controller, Section 2 gives relevant background information on optimization and connected vehicles. It also showcases the current state of the art in connected emergency mobility. Section 3 then describes how the RGS controller fits into the larger RGS pipeline and which implications the pipeline had on the controllers' design. It also defines the optimization problem to be solved by the controller. Finally, the section explains the classes of the controller in detail and provides a technical understanding of how input is processed and output is generated. The optimization algorithms are discussed in Section 4. For each algorithm, the functionality and any special considerations made during the implementation for the RGS controller are described. The following Section 5 describes the evaluation scenarios and system parameters chosen for the evaluation. The evaluation results are then shown and discussed in the next two sections, followed by a conclusion in Section 8.

2 Preliminaries

This section covers topics relevant to this thesis. First, relevant terms regarding optimization problems, such as dynamic optimization and simulation optimization, are defined. Then, an overview of connected vehicles and V2X messages used in this work is given. The state of related research on improving rescue mobility is detailed in the final subsection.

2.1 Optimization Problems

The RGS controller solves an optimization problem to find a rescue corridor that best satisfies a set of conditions. This subsection contains relevant common definitions for optimization problems and introduces terms used in later sections.

Optimization Basics The core of any optimization problem is a **target function** f, which maps all elements u of a solution space U to real numbers $\mathbb{R}$ [4]. The values of $f(u)$ are referred to as target function values, fitness values, evaluations, or scores. U thus contains all elements u for which f returns a valid target function value $f(u)$. Elements u of the solution space U are called decision variables, candidates, parameters, or solutions. An optimization problem P aims to find an $u \in U$ that minimizes the value of f, so that $f(u) \leq f(u') \quad \forall u' \in U, u' \neq u$. Maximization problems also exist, but optimization problems are generally assumed to be minimization problems in this thesis. An optimization algorithm is thus a procedure that aims to find the best possible solution u for P. Depending on the nature of U, P can be further categorized. If U is an infinitely large set, for example $U = \mathbb{R}^n$, P is a continuous optimization problem. If U is a finite set of objects that have to be combined to generate a solution, for example, in a permutation or sequence, P is a combinatorial optimization problem. While it is theoretically possible to calculate all possible solutions to a combinatorial optimization problem, it is often unfeasible in practice, as the solution space is too large.

Dynamic Optimization When assuming that $f(u)$ has the same value for a given u under all circumstances, the problem is called an Offline or **Static Optimization Problem** (SOP). If f is, however, influenced by other factors that might change over time, P becomes a **Dynamic Optimization Problem** (DOP). Following the definition of Fu et al. [4], a DOP is a series of SOPs that are solved sequentially over time, either at fixed time intervals or in reaction to events that can occur. Let all SOPs solved in a DOP be enumerated as $i \in [0, .., n]$, where n is the last SOP that has to be solved. Then each SOP i requires a solution u_i and is based on a state s_i. The state s_i contains all information that influences f outside of u_i. It is assumed that s_i can change over time; thus, $f(u, s_i) = f(u, s_j)$ cannot be assumed if $i \neq j$. Oftentimes, the state s_i also depends on previous states and decisions. In the dynamic case, f delivers an intermediate score for each SOP that is solved. The goal of a decision maker for a DOP is not to minimize the scores f_i at each iteration i, but to minimize the overall accumulated score [4]. A DOP is thus defined as:

$$\min \sum_{i=0}^{n} \mathbb{E}(f_i(s_i, u_i))$$

where $\mathbb{E}$ is the expected value of individual SOP scores f_i.

Simulation Optimization Simulation Optimization Problems (SOPs) are a special case of optimization problems in which a mathematical model of the system to optimize is used to evaluate f. This model is called a simulation. The properties of the simulation might be partially unknown or treated entirely as a black box in simulation optimization [5]. A simulation generates a set of indicators $Z = (X, Y, u)$ from external variables Y, an internal state X, and a control variable u. The external variables Y represent the state of the environment. The internal state X can depend on previous calculations in the simulation or provide randomness to simulate nondeterminism. It is typically assumed that X and Y cannot be directly influenced other than by providing different values for u. In an SOP, f maps the indicators to a numerical value $f(Z(X, Y, u)) \in \mathbb{R}$.

Simulation Optimization is typically employed when an algebraic description of the problem is not available and the problem has non-deterministic properties [5]. This is often the case in traffic modeling problems, in which the actions of different actors need to be modeled and predicted. Simulation Optimization is thus employed in a variety of traffic-related fields, from high-level planning for tasks like traffic light sequence control [6] to trajectory planning for autonomous vehicles [7].

2.2 Connected Vehicles

Once an optimal emergency corridor is generated by the RGS Controller, it needs to be communicated to surrounding vehicles. This is achieved through Vehicle-to-Everything (V2X) communication. This subsection will give an overview of the communication technologies used in V2X and provide definitions for relevant V2X messages.

V2X is an umbrella term for different forms of communication involving vehicles. Two major aspects of V2X are Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I) communication. To partake in V2X communication, a vehicle needs to be equipped with an **onboard unit** (OBU). Vehicles with an OBU are referred to as **connected vehicles**. Infrastructure, like traffic lights or roadside sensors, uses a **roadside unit** (RSU) for V2X communication. Infrastructure can also serve as a relay for V2X messages to extend their range by transmitting them between RSUs.

There are two communication methods currently being discussed for V2X in Europe. The first is ITS-G5 [8] (Intelligent Transport Systems, 5 GHz), wireless communication in the 5.85–5.925 GHz frequency range, also referred to as Dedicated Short-Range Communications (DSRC). It is based on the IEEE 802.11 standard, which is used in WiFi networks. However, ITS-G5 allows Ad-Hoc communication between two endpoints without a central authority. Cellular-V2X (C-V2X) is a competing approach based on cellular LTE communication. It can use existing cellular infrastructure in combination

with RSUs. Both technologies have performed similarly in comparative evaluations, with ITS-G5 achieving slightly lower latencies [9]. Both technologies have an effective communication range of up to 400 meters in an urban environment [10]. Several factors have an impact on the range and reliability of V2X transmissions, such as blocked lines of sight, interferences from other signal sources, and the movement of communicating vehicles. Approaches like multihop routing [11] can mitigate these issues; they however require a network of RSUs and/or cellular infrastructure in the C-V2X case.

To build applications that utilize V2X, the European Telecommunications Standards Institute (ETSI) defines standardized message definitions for different use cases. An entity capable of sending or receiving V2X messages, like infrastructure with an RSU or a connected vehicle, is referred to as a station. The Cooperative Awareness Message [12] (CAM) contains information about the state of a station. The contents differ depending on the station type. If sent by a connected vehicle, the CAM contains position, heading, speed, vehicle type, and dimensions. Stations can be made aware of the presence of other vehicles without tracking them with their own sensors through a CAM. The Collective Perception Message [13] (CPM) allows the sharing of tracking information. It contains a list of objects perceived by a station, typically through sensors like cameras, radar, or LiDAR (**Li**ght **D**etection **a**nd **R**anging). Through CPMs, objects that do not transmit their own positions via CAMs, such as non-connected vehicles or pedestrians, can be accounted for in V2X applications. The Decentralized Environmental Notification Message [14] (DENM) contains notifications about abnormal circumstances, like traffic conditions, weather hazards, or roadwork sites. Information about the location of such events is written into the location container of the DENM. The `PathPredicted2` field of the DENM's location container is especially relevant to the RGS use case, as it allows the definition of an area based on the trajectory of a vehicle.

2.3 Related Work

V2X messages are still in the process of widespread adoption. Today, emergency vehicles still mostly rely on classical devices to enact their special permissions when on an emergency response. They typically use an audio-visual signal like an emergency siren. In Germany, emergency vehicles additionally have electronic means for traffic signal priority requests (TSP). These requests are performed at individual intersections by sending an R09 Telegram [15] or a V2X message [16]. This subsection explores more sophisticated technical approaches to improve rescue mobility from contemporary research.

Any traffic control system is typically evaluated in a traffic simulation before it is deployed in practice. Such evaluations rely on a robust model of emergency mobility, the special permissions of emergency vehicles, and the reactions of other drivers. The traffic simulator SUMO (**S**imulation of **U**rban **MO**bility [17]) offers such a model [2] based on traffic data analysis, which is widely used for modeling emergency mobility. Kuzmic and Rudolph [18] have developed a simulator in the Unity 3D game engine. It models sensors and communication of self-driving vehicles when forming an emergency corridor on motorways.

Multiple approaches aim to improve rescue mobility by changing traffic light sequences

(TLS) beyond typical TSP. One solution is to provide a "green wave" on the path of the emergency vehicle; traffic lights are set to green ahead of time, allowing other vehicles to clear intersections. This reduces the need for emergency corridor formation and reduces response time. Yadav et al. [19] propose a deep reinforcement learning-based route selection algorithm that considers adaptable traffic lights. The fastest route is chosen based on the traffic situation while controlling traffic lights encountered by the emergency vehicle. An evaluation of their approach in SUMO shows that it can reduce emergency vehicle travel time by 25%. Rajak et al. [20] propose a networked TLS control system that creates a "green wave" along an entire route. An emergency vehicle requests green lights along its route at a central server, which then communicates with corresponding traffic lights. Their evaluation results again show a significant impact on response time by TLS control. Bieker-Walz [2] has also implemented an algorithm that controls multiple traffic lights on an emergency route. Its evaluation on synthetic and real-world scenarios in SUMO has also resulted in significantly lower travel time for emergency vehicles. Ruppe et al. [21] propose a decentralized system that incorporates route information into TLS control. Combined with overtaking strategies for specific situations, the system allows for precise preemption with minimal impact on surrounding traffic, while reducing the crossing time for emergency vehicles at controlled intersections. Their results have been verified in long-term field tests.

Apart from TLS control, rescue mobility research focuses on individual vehicles. This is especially relevant for autonomous vehicles, as they need to be capable of reacting to emergency situations. Current approaches include detecting emergency vehicles with the visual sensors and microphones [22] [23]. Others focus on V2X communication to inform vehicles of an incoming emergency vehicle. The impact of V2X communication on rescue corridor formation has been explored by Ashish et al. [24]. They showed that the increased range of V2X transmissions can be beneficial in emergency corridor formation over the audio-visual emissions of an emergency siren. Their work was evaluated in SUMO and has shown lower emergency travel times when using V2X communication instead of a siren only. Kumar et al. [25] propose an architecture that issues alerts to emergency drivers and other vehicles. This system uses V2X messages to communicate and can incorporate information from local infrastructure. Drivers are warned of pedestrians on roads, impending collisions, and congested roads. The project AORTA[2] also aims to improve rescue mobility through V2X communication. However, as of writing, no publications or technical details are available.

Study Project A 6-month study project preceded this thesis to plan aspects of the RGS controller. The following points were established during the project:

- An optimization problem solved by the RGS controller was defined. As the traffic state changes over time, the controller shall provide an emergency corridor that keeps the crossing time of the emergency vehicle as low as possible. Secondary

[2]AORTA is a project similar to the RGS in scope and goals according to its website: `https://www.projekt-aorta.de/`

objectives are as few as possible traffic rule violations and a stable solution over time.

- Based on the constraints provided by the RGS as implemented for the GAIA-X4 AMS Project and the problem definition, an architecture for the controller was designed. The controller has to interface with other components and accept standardized messages. It must allow for different optimization strategies to be used.

- A set of evaluation scenarios was conceptualized based on an expert interview [26] with Jonas Klemmt, a researcher at the Brunswick fire brigade. The scenarios resemble situations typically encountered in emergency responses and situations that are especially challenging.

- A fitness landscape analysis was conducted on one instance of the problem in a preliminary implementation of an evaluation scenario. The selection of optimization algorithms was influenced by the results.

- It was decided to implement and evaluate the Downhill-Simplex Algorithm, a Memetic Algorithm, and a Reinforcement Learning Agent in the RGS controller. This selection is based on literature research and the insights provided by the fitness landscape analysis.

- SUMO was selected as the traffic simulator for the RGS controller. It strikes a balance between performance and model accuracy that is adequate for the presented use case. It also provides an implementation of rescue mobility, as mentioned above.

The aspects mentioned in the listing will be further elaborated on in the relevant sections. At this point, they shall illustrate how the work was divided between the study project and this thesis.

3 System Design

The proposed system is to enhance rescue mobility by planning rescue corridors ahead of time and communicating them to connected vehicles. This section will first explain the overall pipeline of the RGS, followed by a definition of the optimization problem that needs to be solved to generate a rescue corridor. Next, the architecture of the RGS controller, the centerpiece of this thesis, is examined. Finally, changes made to the vehicle model in the simulation are discussed. These changes are required to have simulated vehicles react to communicated emergency corridors.

3.1 RGS Pipeline architecture

The RGS is being developed as a part of the GAIA-X4 AMS (Advanced Mobility Services) project. It is to provide connected vehicles with the capability to clear emergency corridors with the support of local infrastructure, possibly before an emergency vehicle is in visible or audible range. This is an especially interesting prospect for fully autonomous vehicles, which would otherwise have to rely on onboard sensors. The RGS provides a warning in a standardized format to which connected and autonomous vehicles can react.

In the experimental setup used in the project, the information relevant to this process comes from two sources: a roadside LIDAR sensor and the emergency vehicle itself. This subsection will detail how this information is processed and transmitted, what happens with the output provided by the RGS controller, and what implications the pipeline has for the controller's design. The pipeline architecture is shown in Figure 1.

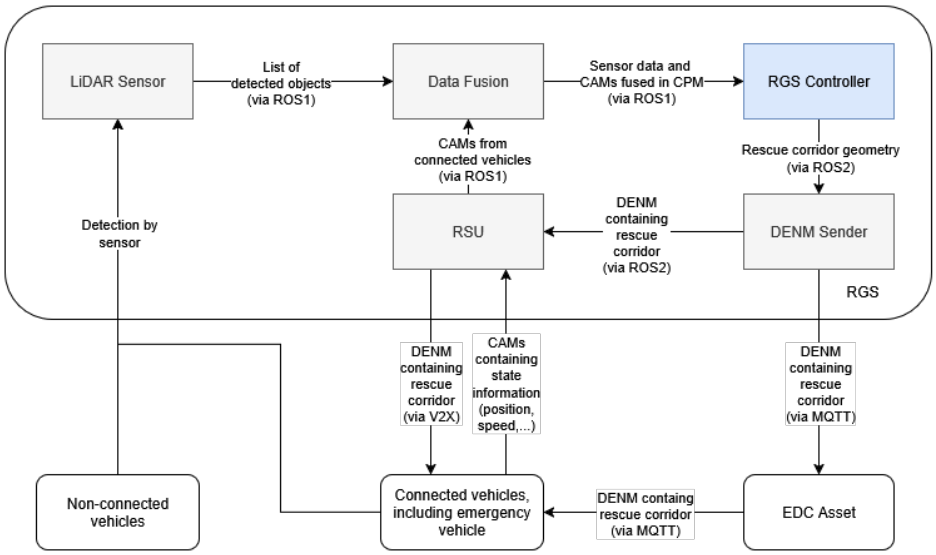

Figure 1: The RGS architecture. Communication paths are simplified, and the V2X-communication case is assumed.

Requirements The RGS is designed to be used as a general-purpose program that can generate messages aiding rescue mobility in an urban environment, regardless of the concrete scenario, traffic density, or other factors. There are, however, still conditions that have to be met for the RGS to be deployed effectively.

- An emergency vehicle must be approaching an area of interest, in which a rescue corridor is to be formed.
- The emergency vehicle must be capable of sending CAM messages and do so as it approaches the area of interest.
- The area of interest must be fitted with the infrastructure to send and receive V2X messages.
- The area of interest must be fitted with the infrastructure to run the RGS controller or transmit inputs and outputs to a remote location, such as Mobile Edge Computing [27].
- Other vehicles in the area of interest must be able to react to incoming DENM messages. If not all vehicles are capable of doing so, system performance might degrade. If none can react to DENMs, the system has no effect.

From CAM to DENM If all conditions are met, the RGS starts to work. As the emergency vehicle approaches the area of interest, it sends a CAM message to notify the RGS of its presence. The CAM is received by an RSU and transmitted to the infrastructure running the RGS. In the Gaia-X4 AMS case, the RGS controller, the data fusion module, and all relevant communication are performed on a computer that is connected by wire to the LiDAR sensor and the RSU. The messaging functionality of the open-source Robotic Operating System (ROS) [28] is used for internal communication between components of the RGS. ROS implements messages based on the publish-subscribe pattern: A system can contain multiple components called nodes. Any node in a system can publish messages to a topic, to which other nodes can listen and retrieve relevant messages. CAMs received by the RSU are thus published on a ROS topic.

At the same time, the LiDAR sensor continuously tracks moving objects in its vicinity. Objects detected by the sensor are referred to as tracks. The sensor publishes a list of its current tracks on a ROS topic in regular intervals. Both the track list and incoming CAMs are received by the data fusion module. It combines the information from both sources by associating sensor tracks and CAM positions based on their distance to each other [29]. The resulting list of objects is written into a CPM and published to a ROS topic for the RGS controller to read.

The LiDAR sensor and the data fusion module use ROS1, which is incompatible with the newer ROS2. However, components in the pipeline after the RGS controller use ROS2. The main logic of the controller runs on a ROS2 node. To bridge the gap between ROS1 and ROS2, the controller includes a translator node that runs in ROS1. It receives the fused CPM from the ROS1 topic, extracts relevant information, and publishes it on an

MQTT [30] topic. MQTT is another publish-subscribe messaging protocol that allows the ROS nodes of different versions to communicate with each other.

The controller then generates a rescue corridor based on the CPM. The details of the controller's functionality are laid out later in this section. For now, it is simply assumed that it receives the traffic state as a CPM and generates DENM containing an emergency corridor. The corridor is encoded as a series of points with offsets defining its width and written into the `PathPredicted` field of the DENM's location container. They define a polygon that represents an emergency corridor that vehicles are to clear so the emergency vehicle can pass safely.

The generated DENM is published on a ROS2 topic. The DENM Sender listens to that topic and further processes the DENM. It can then be transferred back to the RSU and sent to connected vehicles in the vicinity. The DENM is simultaneously published to an EDC (Eclipse Dataspace Connector [31]) Asset in a Gaia-X dataspace. Vehicles can retrieve DENM messages from this EDC regardless of their own location and V2X range limitations. This allows for an even earlier planning of maneuvers.

The state of the environment and the output of the controller form a feedback loop. The traffic state in the area of interest is registered by sensors and messages from connected vehicles. The data fusion generates a CPM for the controller, which in turn generates an optimal rescue corridor. It is sent out as a DENM, to which connected vehicles react, changing the traffic state. Non-connected vehicles might be present in the area of interest. While they cannot receive DENM messages and thus do not have instructions on how to form the requested rescue corridor, they are still accounted for in the optimization loop if sensors detect them.

Communication method While the Gaia-X-based communication was successfully tested in the Gaia-X4 AMS project, the V2X use case was chosen for the design of the RGS controller for multiple reasons. Compared to the GAIA-X case, using only V2X enables purely local operation. There is only direct communication between the local infrastructure and vehicles, and no need for remote connections. The Gaia-X case requires an internet connection for vehicles to retrieve DENMs. Communication range also has a major impact on the size of the traffic scenarios that need to be simulated to obtain rescue corridors. The further away the emergency vehicle is from the area of interest, the larger and thus computationally expensive the required simulation becomes.

An additional problem is unknown driver intentions. If the area of interest contains a single intersection i_0, each vehicle v can take $r_{i_0}(o_v^0)$ routes, where o_v^j is the direction from which v approaches i_j and r_{i_j} gives the number of reachable routes from that direction. Each additional intersection i_j in the area of interest increases the number of possible routes for each vehicle v by $r_{i_j}(o_v^j)$ for each route that ended with o_v^j. If, in the worst case, all intersections in the area of interest are pairwise connected to each other, this leads to an exponential increase in possible routes for each vehicle. Research on estimating driver intentions on single intersections has been conducted [32], and traffic flow data can help to predict routes in real-world scenarios [33]. However, these methods do not guarantee accuracy; thus, not knowing driver intentions still adds a large degree of uncertainty that

scales with the size of the simulated area of interest and increases the problem difficulty.

3.2 Problem Definition

Before the details of the RGS controller are laid out, the problem that it solves has to be defined. As established beforehand, the input for the controller is the state of the traffic in the area of interest. The controller shall output a rescue corridor through which the emergency vehicle can pass and other vehicles must stay clear of. It is to be noted that the RGS system is designed to be capable of handling multiple emergency vehicles simultaneously, as long as they are moving through the area of interest on the same route. From an optimization perspective, this makes little difference, as input and output remain the same, and multiple emergency vehicles are expected to follow the same corridor. Thus, only the single emergency vehicle case is considered in this thesis.

The general approach to the problem is to model the state of the environment in a simulation, observe the reactions of vehicles to certain actions, and generate a rescue corridor based on these observations. t_0 shall be defined as the point in time at which the formation of an emergency corridor is requested through the infrastructure by a fused CPM. At any given $t > t_0$, the controller has to solve the simulation optimization problem P_t:

$$P_t : \min f; f(u) = g(Z(X_t, Y_t, u))$$

Target Function P_t is a minimization problem, which is about finding the minimum value for a target function f using the decision variable u. f is evaluated in a simulation. Taking the decision variable u, its internal state X_t at the time t, such as random vehicle-specific behavior, and the set of external variables Y_t, which in this case is the traffic state as defined in the CPM at t, the simulation can calculate a set of indicators Z. These indicators include positions, speeds, trajectories, and various other properties of vehicles in the simulation. As the controller is only interested in a subset of Z, the selection and weighting function g is defined as:

$$g = a \cdot t_{emg} + b \cdot num_v + c \cdot dist_E(u, u_{t_{DENM}})$$

with:

- $a, b \in [0, 1]$. These are control parameters to determine how much influence their respective terms have on the evaluation.
- t_{emg} as the time the emergency vehicle takes to reach its destination, which usually is the end of the simulated scenario. This is the main goal of the optimization.
- num_v as the number of accumulated rule violations committed by all vehicles during the simulation. It is incremented by 1 for each vehicle that currently commits a

violation at each simulation step. Violations include standing or moving on sidewalks, moving on opposite direction lanes, or moving past red traffic lights. This is used as a secondary criterion to order results with otherwise similar evaluations.

- $c = 0$ if $t < t_{DENM}$, otherwise $c \in \mathbb{R}+$. c is a control parameter, t_{DENM} is a point in time after t_0 at which the controller sends the first DENM, so that vehicles in the area of interest can start their evasive maneuvers.

- $dist_E(u, u_{t_{DENM}})$ as the Euclidean distance between the current solution u and the solution $u_{t_{DENM}}$ first sent at t_{DENM}. The Euclidean distance is chosen because it not only takes into account how many values in u have been changed, like the Hamming distance, but also by how much these values have been changed, giving a better indication of how "far" u is from $u_{t_{DENM}}$. This term reflects the requirement for the controller to output similar solutions. The need for stable solutions over time was outlined during field evaluations in the GAIA-X4 AMS project. It was shown that autonomous vehicles would struggle with frequently changing rescue corridor DENMs.

Previous considerations include the accumulated loss times of other vehicles as another criterion for the value of g. Loss time is defined as the time a vehicle spends standing or moving below its maximum allowed speed. The intention was to account for the negative impacts of the emergency corridor on other vehicles in the target function. It was eventually discarded for multiple reasons: firstly, situations were observed in simulations where low loss times could encourage other violations, such as non-emergency vehicles crossing red traffic lights. As safety is regarded as more important than time loss for non-emergency vehicles, such encouragements were deemed unacceptable. Secondly, in complex scenarios, especially at intersections, vehicles would accumulate high loss times regardless of the actions of the controller. The impact of the loss times would then scale with t_{emg}, making an additional criterion obsolete.

Since num_v is already a secondary metric in the target function, designed to be a "tie-breaker" for similarly performing solutions u, it was decided not to include a third and possibly contradictory metric with the loss times.

Decision Variables The ability of the controller to influence the environment comes from the rescue corridor in the DENM. Thus, the decision variable u of the optimization problem is the shape of the rescue corridor. In previous work, a format for u as a sequence of lateral positions has been established with the following properties:

$$
\begin{aligned}
&u = (p_0, ..., p_n) \\
&p_i \in [0..s_i] \qquad \forall i \in [0..n] \\
&|p_j - p_{j+1}| \leq l \quad \forall j \in [0..n-1]
\end{aligned}
$$

with n as the number of decision points p_i along the route, s_i as the maximum number of available lateral positions at p_i, and l as the maximum change of lateral positions

between two decision points. The decision points each have a fixed distance d_p to the preceding and following points. The total number of decision points n is thus calculated as $\frac{len_r}{d_p}$, with len_r as the length of the route.

The decision points intuitively break the route down into segments, during each of which the emergency vehicle will attempt to take a given lateral position on the road. The limit l for a maximum change between two points exists to prevent extreme maneuvers.

Dynamic Optimization Problem By the previously established definition of dynamic optimization problems [4], the goal of the system at any point in time t_i is to minimize the expected value of all future evaluations of f. For the defined optimization problem, the actual performance of the solution in the environment is the relevant optimization goal. In deployment, the environment is the real world, whereas in evaluation, it is abstracted by a simulation running in real-time. The RGS controller provides a solution u_{t_i} at a given t_i to solve P_{t_i}. Its goal is, however, to solve the optimization problem P':

$$P' = \min g'; g' = a \cdot t'_{emg} + b \cdot num'_v$$

where t'_{emg} and num'_v are the respective values taken from the environment, not the prediction in an instance of P. The parameters a and b remain the same as before. It is assumed that a solution u_{t_i} which performs well on P_{t_i} will lead to a good performance in P', since P_{t_i} is modeled after the state of the environment at t_i. However, the simulation used in P_{t_i} might not be able to fully capture the state of the environment, or its prediction might differ from reality. This can happen due to different causes, such as incomplete information about driver intentions, non-connected vehicles in the area of interest, or incorrect predictions about the evasive maneuvers taken by vehicles. It is thus insufficient to solve only one instance of P and assume the environment would behave exactly as predicted. The controller has to constantly update its prediction based on the current state of the environment to factor in any unforeseen changes and deliver an optimal solution to P'.

3.3 Controller Architecture

The RGS controller[3] is the optimization component of the RGS pipeline and the primary focus of this work. It receives CPMs containing the positions of vehicles in the area of interest and generates an emergency corridor encoded in a DENM. Whenever a new CPM arrives, it is treated as the current input for the controller. The controller generates DENMs as long as the received CPMs contain an emergency vehicle in the area of interest.

The RGS controller can receive input in two ways: from an external source or an outer simulation. In case of an external source, the controller operates in the RGS pipeline as described above. The fused CPM is received by a ROS1 node that publishes its contents to an MQTT topic, to which the controller listens.

[3]The source code of the RGS controller is available at: `https://scm.cms.hu-berlin.de/trappejo/rgs_controller`

The second case is used in testing and evaluation. A simulation instance, the **outer simulation**, runs in real time and periodically generates CPMs from its current state, which are then passed to the RGS controller. Generated DENMs from the controller are passed back to the outer simulation, in which the emergency vehicle follows the DENM's trajectory and other vehicles react accordingly. The outer simulation represents the real world for the controller in this case.

The component is named controller because the problem it solves is a controlling problem: the input changes over time as vehicles move, and the output, the emergency corridor, must be adjusted accordingly. An architecture inspired by the controlling algorithms, specifically model predictive control [34], was thus chosen: A simulation is used to predict the future traffic state and generate a recommended trajectory from that prediction. The emergency vehicle follows the predicted path, which leads to a new traffic state on which the controller bases further predictions.

In the preceding study project, different traffic simulators were examined, and SUMO (**S**imulation of **U**rban **MO**bility [17]) was chosen for this project, as it offers a reasonable tradeoff between performance and model accuracy.

Programming Language The controller is restricted to being written in either Python or C++, since only these languages are officially supported by ROS2. Both languages offer advantages: C++ is generally faster, being a compiled language, whereas Python is interpreted at runtime. Python is less verbose and does not require recompiling when code is changed. Potential performance gains from using C++ were examined by comparing runtimes of SUMO's interface libsumo and algorithm parameter generation.

In a comparative test with 100 simulations of a complex traffic scenario, libsumo was called from both Python and C++. In each simulation step, the testing program would change the lateral alignments of vehicles to emulate interaction between the controller and the simulation. The C++ program completed 100 simulation runs in 55.4 seconds, whereas the Python script surprisingly only took 41.9 seconds for the same task. The test was repeated with 500 simulation runs, confirming the results (281.9 seconds in C++, 209.0 seconds in Python). In a smaller scenario with 1000 runs each, the programs completed in about equal times (C++ in 29.4 seconds, Python in 29.2 seconds). This behavior is unexpected, as the Python implementation of libsumo should call the same underlying C++ library as a C++ program. Due to these findings, the controller uses the Python interface of libsumo.

Based on typical simulation times in SUMO, the hypothesis was formed that the time the controller spent on simulating parameters to asses their performance would be significantly greater than the time it would need for other calculations, such as parameter generation.

To evaluate the impact of language on parameter generation, the times that one of the optimization methods, Downhill-Simplex (section 4.1), uses for its parameter generation were examined. The parameter generation times were then compared to the evaluation times of parameters using the simulation when implemented in Python. An evaluation run of a simple scenario was completed in 29.246 seconds. In that time, the controller

ran 1482 simulations. The Simplex Optimization strategy class spent a total of 0.157 seconds on calculating new parameters, while it was waiting for their evaluation for 29.053 seconds. The difference between evaluation and generation time becomes even more apparent when looking at the first time steps of the run, when the controller has to simulate almost the entire scenario to evaluate a parameter. The further the emergency vehicle advances, the shorter the simulations performed by the controller become on average. The generation of the initial simplex consisting of 9 distinct parameters took 0.0005 seconds, whereas its evaluation took 0.1170 seconds. More complex simulation scenarios further widen the gap between parameter generation and evaluation time: the generation of 14 initial parameters in a complex scenario took the Simplex Optimization strategy 0.0005 seconds. They were evaluated in 1.1835 seconds. It was thus concluded that using C++ for parameter generation could not provide meaningful speedups to the optimization, as the vast majority of the computation time required by the controller is used when evaluating parameters in the simulation.

Python's ability to quickly apply changes proved helpful in integration tests. Using C++ would not offer significant performance improvements, since it would not speed up the simulation and would have little to no effect on the far less computationally expensive parameter generation. Thus, Python was chosen as the language for the controller for its ease of use.

With the required technologies selected, an architecture for the controller was developed. The three core tasks communication, simulation and optimization are distributed in individual classes. The following paragraphs will detail the class structure of the RGS controller, which is also laid out in Figure 2 as a class diagram.

MessageHandler The *MessageHandler* class handles communication of the RGS controller: it receives CPMs via MQTT and publishes DENMs via ROS2. If an external CPM source is used, the *MessageHandler* is the entry point to the program. It is started via Docker Compose, together with a ROS1 Node, which transmits received CPMs via MQTT. If an outer simulation is used, an *OuterSim* serves as the entry point. The *OuterSim* creates a *MessageHandler* and directly communicates with it.

SimManager CPM contents received by the *MessageHandler* are in both cases passed to the *SimManager*, which is the central coordinating class in the RGS controller. The SimManager only saves the most recent CPM, which is considered the current state of the outer simulation or the real world. This state is updated whenever a new CPM arrives, and any simulations are thus run with the most recent available observed state. When the first CPM is passed to the *SimManager*, it requests a set of initial parameters from the *Optimizer*, for each of which, a simulation is started.

Optimizer The *Optimizer* manages results of simulations for the *SimManager*. It evaluates completed simulations according to the target function g, caches simulation results, and serves as an interface to the optimization strategy for the *SimManager*. Upon

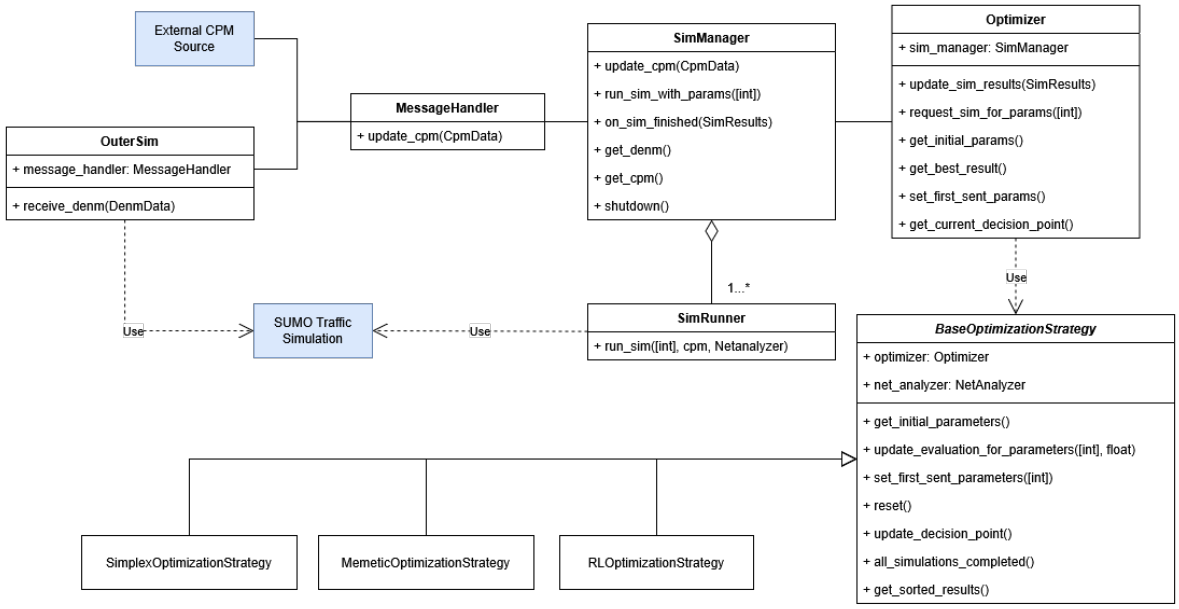

Figure 2: Class diagram of the RGS controller as it is implemented. Utility classes, such as V2X-Message Data objects, the NetAnalyzer, and SimResults, as well as members and methods not relevant to the core functionality, are omitted for simplicity. The optimization strategies Simplex (Downhill-Simplex), Memetic (Memetic Algorithm), and RL (Reinforcement Learning) are explained in detail in section 4 and thus kept minimal in this diagram.

request by the *SimManager*, the *Optimizer* lets its optimization strategy generate a set of initial parameters when the first CPM arrives.

The initial parameters are passed back to the *SimManager* and evaluated using a *SimRunner*. Parallelization in libsumo requires the use of separate processes, as done in Python's `multiprocessing` module. The *SimRunner* thus utilizes a pool of processes to run multiple *SimRunners* simultaneously. The speedup achieved through `multiprocessing` was examined by running 1000 iterations of a low-complexity evaluation scenario; the results are displayed in Figure 3. The performance gain of using SUMO with `multiprocessing` is not directly proportional to the number of available processors, but an improvement is still present.

SimRunner A *SimRunner* receives a parameter u and the current CPM and returns a *SimResults* object upon completion. The *SimRunner* first spawns all vehicles from the CPM into the simulation and then runs it as fast as the available computational resources allow it. During each simulation step, the emergency vehicles in the simulation adapt their lateral position based on u. Any other vehicles attempt to clear the path defined by u. Statistics about the run, such as the loss times of the vehicles and the number of rule violations committed, are collected in the *SimResults* object. Should a collision occur before the emergency vehicle reaches the end of the simulated scenario, a

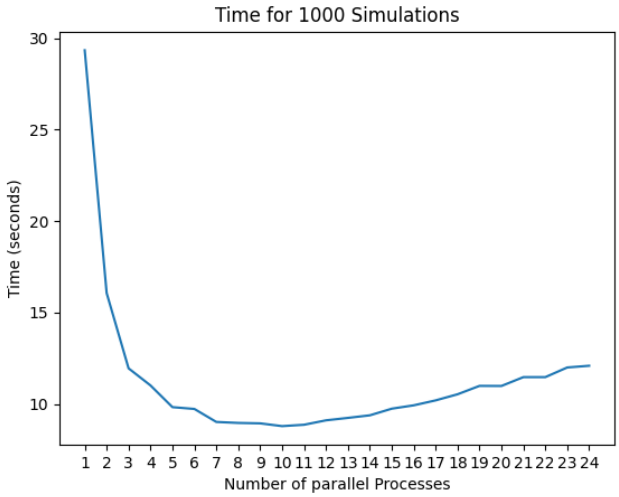

Figure 3: Time to finish 1000 simulations with a given number of parallel processes. The test system has a 12-core CPU, including virtual cores. The best time (8.79 seconds) was achieved using 11 parallel processes.

corresponding flag is set in *SimResults*, the run is terminated, and regarded as a failure. The same happens if the run does not complete within a predefined simulation time t_{sim}. Simulation time is the time that has passed within the simulation, not the actual computation time needed by the *SimRunner*. This serves as a detection mechanism for deadlocks and to save computation time. Through experimentation, t_{sim} was defined as 120 seconds. Even in the most complex evaluation scenarios, taking more than 120 seconds cannot be regarded as an acceptable result. If the simulation does not result in a failure, a DENM is generated based on the trajectory of the emergency vehicle in the simulation run. The DENM is appended to the *SimResults* object. Once the simulation is finished or truncated, the *SimResults* object is returned to the *SimManager*, which passes it to the *Optimizer* for evaluation.

Result Caching The target function g is then evaluated on this result by the *Optimizer* and the score $g(u)$ is saved in a cache. Each score has a validity duration d_{val} and is removed from the cache as soon as d_{val} expires. This is done because DENM messages are sent in fixed intervals. Whenever the *MessageHandler* requests a DENM from the *Optimizer* through the *SimManager*, the current best result with a valid d_{val} is returned. The validity duration is a tradeoff between good and recent solutions. If d_{val} is too high, the controller risks sending an obsolete DENM, based on a state that is no longer accurate, but scored better than more recent evaluations. If d_{val} is too low, a bad solution might be sent, as potentially better ones are still being simulated at the time of the DENM request.

d_{val} was chosen to be $= 1/f_{DENM} * 2$ seconds, where f_{DENM} is the frequency at which DENMs are published. With f_{DENM} as 2Hz, a value that proved appropriate in field tests,

this results in a validity duration of 1 second. In this way, each evaluated solution has a chance to appear in a DENM, while the state each DENM is based on is at maximum $1s + t_{sim}(u)$ old, where $t_{sim}(u)$ is the simulation time that was needed to evaluate u.

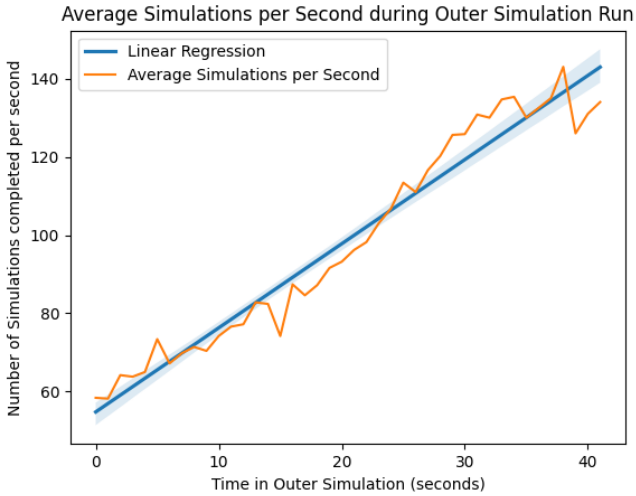

Figure 4: Simulation runs completed by the controller in a medium-complexity scenario over time. The results are averaged over 5 iterations of the outer simulation on the same scenario, and a linear regression line is fitted (blue). As the outer simulation progresses, the simulations performed by the controller become shorter, increasing the throughput.

To examine the implications of d_{val} on the optimization algorithms, the throughput of simulations achieved in practice on a medium-complexity scenario was examined. With full utilization of parallelization, the controller can complete between 60 and 140 simulations per second, as seen in Figure 4. An *OptimizationStrategy* has to request a simulation for its current best solution no later than one second after the last request of the at that time best solution. This is especially relevant for optimization algorithms that operate in cycles. If one cycle of an algorithm takes more than 60 parameter evaluations, the controller risks having none of the current best solutions in the cache when a DENM is requested.

OptimizationStrategy Evaluated parameters u and their scores $g(u)$ are passed back to the *OptimizationStrategy*. The actual implementations of strategies will be discussed in the next section, so *OptimizationStrategy* is assumed to be an implementation of the base class *BaseOptimizationStrategy*. This base class bundles functionality that can be used by all strategies. Among them is its own caching system independent from the one used by the *Optimizer*. The implementation of a strategy can decide for itself how long solutions shall be seen as valid for a given parameter u. This is useful for algorithms operating in phases, when prior evaluations might be needed again in later phases, and there is no guarantee that the phase completes before d_{val} of a needed solution expires.

The *OptimizationStrategy* is also notified when $u_{t_{DENM}}$ is sent out, allowing it to adapt its behavior and reevaluate cached solutions. This might be necessary, as the target function now changes by considering the distance of any solution u to $u_{t_{DENM}}$.

The *BaseOptimizationStrategy* also keeps track of the current size of the problem by monitoring which decision points have been passed already by the emergency vehicle in the outer simulation or the real world, as this is relevant to most strategies. Typically, a strategy will request simulations, wait until they are evaluated, and generate new parameters based on the results.

To summarize, the *MessageHandler* receives CPMs containing the current traffic state in the area of interest and passes those to the *Optimizer* through the *SimManager*. The *Optimizer* generates parameters according to its *OptimizationStrategy*, which are evaluated in the simulation, and then passed back to the *OptimizationStrategy* with their scores, guiding further optimization decisions. The *MessageHandler* periodically sends DENMs based on the currently best result available.

3.4 Simulation Adaptations

As mentioned previously, both the *OuterSim* and the *SimRunner* use SUMO as their traffic simulator, which provides a model for emergency mobility [2]. Emergency vehicles in SUMO have special rights, such as disobeying speed limits, overtaking other vehicles on the right, crossing red traffic lights, and disregarding right of way at intersections. Emergency vehicles in SUMO also have a blue light device, which makes other vehicles react to them: within 25 meters, they will attempt to form a rescue lane to the right of the left-most lane on a road. Vehicles can be configured to only react to the blue light device with a given probability, which is set to 0.577 by default.

While these features offer a reasonable baseline for simulating emergencies, they do not fully cover the requirements of the RGS. In SUMO's model, vehicles only react to the emergency vehicle itself. For the RGS use case, the vehicles have to react to a DENM they receive. The RGS controller thus uses customized behavior for all vehicles in its simulations.

The core of the optimization lies in controlling the lateral movement of the emergency vehicle. Lateral movement can be simulated in SUMO using the sublane model: each lane is divided into sublanes with a given width. A vehicle can inhabit multiple sublanes at a time if it is wider than a single sublane. If a vehicle moves to a sublane p, it will attempt to keep p at the center of its lateral position.

From the perspective of the emergency vehicles, a parameter u consists of a series of sublane changes. Whenever the emergency vehicle passes a decision point, it will attempt to change its sublane to the current p_i in u when between decision points i and $i + 1$. The emergency vehicle will still perform small sublane changes independently of u, to avoid collisions or perform overtaking maneuvers. It also retains its special rights from SUMO's base emergency model explained above, except for the blue light device.

SUMO uses the Krauß car-following model [35] for vehicle movements, which is a collision-free model, meaning that in a normal SUMO simulation, collisions should not occur. However, the heavy interference with the behavior of vehicles caused by the RGS

controller can, in some cases, lead to collisions. One observed example is oncoming traffic: When the emergency vehicle receives a DENM that commands it to move to an opposite direction lane, it will sometimes collide with oncoming vehicles if they are too close and cannot react in time. Any simulation involving a collision is regarded as a failure.

An early hypothesis during the development of the RGS controller was that collisions in the outer simulation would not occur. Inner simulations would predict them, and the controller would select a different solution. This assumption proved to be false. There are different possible causes for collisions in the outer simulation. The RGS controller generally does not know the intentions of vehicles other than the emergency vehicle. Vehicles might behave differently in the predictions of the controller than in the outer simulation or the real world, for example, by taking a different route than anticipated. The time between CPM arrival and DENM retrieval also plays a role in solution accuracy. Finally, the CPM format might not be able to transfer all relevant information between the two layers of simulation, leading to differing states.

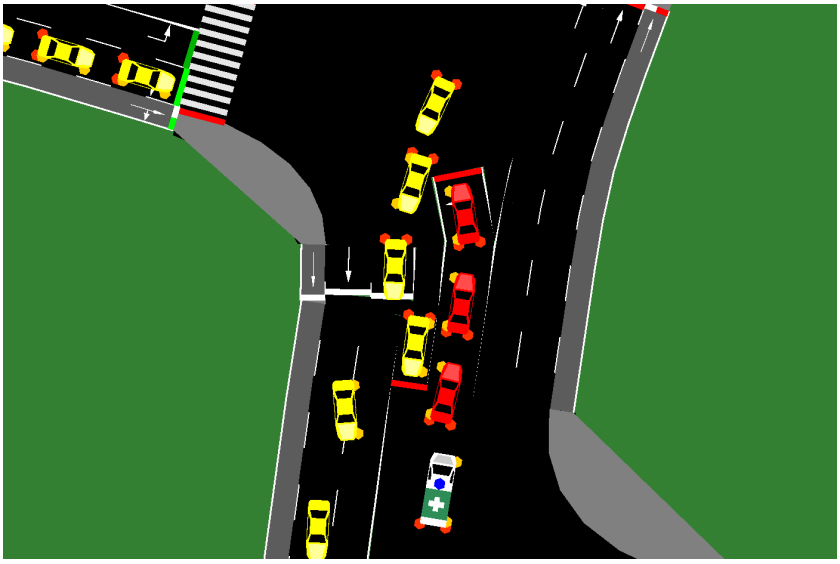

Figure 5: A deadlock situation illustrated in SUMO. The red lines mark traffic lights. The red vehicles attempt to move to the left, but their path is blocked by the yellow vehicles. The leading yellow vehicle coming from the north attempts to turn left from its point of view. Even when both traffic lights turn green, neither the leading red nor the leading yellow vehicles can advance.

Simulations can also fail due to not completing, and thus resulting in a timeout. Timeouts can be caused by deadlock situations sometimes encountered in intersection scenarios. A deadlock occurs when vehicles block an intersection in a way that progress is no longer possible. In the RGS controller, deadlocks have been observed when vehicles are being "pushed" onto an intersection by the DENM. If vehicles are standing at a traffic light and are blocking the desired emergency corridor in the DENM, they will attempt to move onto the intersection regardless, as described above. This can lead to a situation as illustrated in Figure 5. SUMO typically handles deadlocks by teleporting vehicles to

their destination at a blocked intersection. Since this behavior is not realistic in the RGS use case, it has been disabled in the RGS controller. Instead, a deadlock is assumed after a certain amount of time has passed, and the simulation is regarded as a failure.

4 Optimization Algorithms

This section introduces the three optimization strategies evaluated in the RGS controller. For each strategy, the motivation for its inclusion is presented and its functionality explained. Relevant implementation details are also provided, for example, how the strategies utilize parallelization.

4.1 Downhill-Simplex

The Downhill-Simplex Algorithm, also known as the Nelder-Mead Algorithm [36], is a numerical minimization technique that converges to a local optimum of a target function. The algorithm evaluates a set of points, called a simplex, and gradually moves it towards better target function values. It either expands towards well-performing regions of the search space or becomes smaller to move away from poor-performing regions.

The target function landscape is generally assumed to be smooth for the Downhill-Simplex Algorithm to work well. This means that solutions close to each other should have similar target function values. General assumptions on this matter are hard to make for the emergency corridor optimization problem, as the solution landscape is different for each instance. A fitness landscape analysis conducted in the preceding study project indicated a smooth target function landscape in the examined instance. While this cannot be assumed as true for all instances of the problem, the simplicity of the Downhill-Simplex Method and the ability to work without a gradient of the target function make it an appealing candidate for the RGS controller.

The Downhill-Simplex algorithm needs an initial simplex to function. Takenaga et al. [37] have examined the impact of different simplex initialization methods on the performance of the algorithm. They conclude that the regular simplex initialization method performs best in most applications. It creates points from a single initial parameter, which is at the center of the first simplex. This initial parameter is the path through the center of the road at each decision point. In the regular initialization method, points are placed around the central point in a manner that they all have the same distance to the central point. They also each have the same distance to each other point in the initial simplex. Takenaga et al. recommend covering a volume as large as possible in the initial simplex for best search performance. The simplex consists of $n + 1$ points, where n is the dimension of the parameters u.

After an initial simplex is formed and evaluated, it is ordered by the target function values of its points. Let S be the current simplex ordered by target function value. u_0 is the best parameter in S, which has $j = n + 1$ elements:

$$g(u_0) \leq g(u_1) \leq ... \leq g(u_j), i \in [0, j], u_i \in S$$

A reflection point u_j^r is formed by taking the worst point u_j in the simplex. The algorithm first creates a centroid from the remaining points in S.

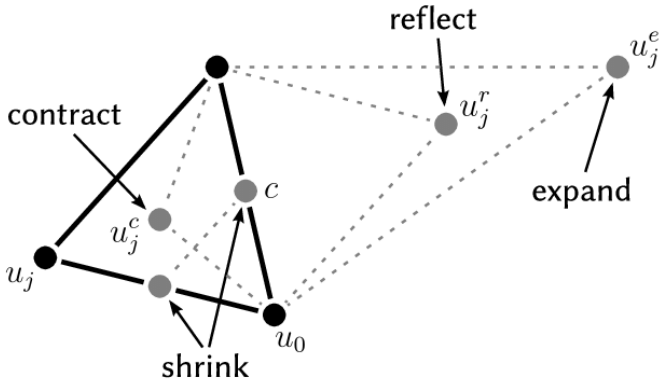

Figure 6: The operations performed by the Downhill-Simplex algorithm as they would be performed in a two-dimensional problem. With u_j as the worst point in S, the centroid c is formed from the remaining points. c determines the direction of the reflection, resulting in u_j^r. The expansion point u_j^e and contraction point u_j^c are located on a line formed by u_j and c. A two-dimensional problem is used for illustration purposes; the actual problem solved by the RGS controller typically has a higher dimension. Original image from [38], with adapted point designations.

$$c = \frac{1}{n} \sum_{i=0}^{n} u_i, u_i \in S$$

Intuitively, a line is drawn from u_j to the centroid c, which is the center of all remaining points $u_i, i \neq j$. The reflection point u_j^r is then placed on the extension of that line, on the opposing side of the centroid. It has the same distance to c as u_j. The reflection, along with other operations of the Downhill-Simplex algorithm, is visualized in Figure 6. It is calculated as:

$$u_j^r = c + \alpha(c - u_j)$$

where α is an algorithm parameter with a default value of 1. Based on the evaluation of u_j^r, an expansion point u_j^e is created if u_j^r is a better solution than the previous best point u_0, so if $g(u_j^r) < g(u_0)$:

$$u_j^e = u_j^r + \gamma(u_j^r - c), \gamma > 0$$

The default value [39] for the algorithm parameter $\gamma = 1$ is used in the implementation. The expansion point u_j^e is thus created if the reflection point shows a direction in which

the solutions are better than the current best solution, to move the simplex even further into a presumably better-performing region of the solution space. Should this assumption prove true, if $g(u_j^e) < g(u_j^r)$, u_j^e is chosen as the replacement point u_j', which replaces u_j in the next iteration.

If the reflection point is an improvement over the current best point u_0, but still better than u_j, it becomes the replacement point u_j'. If u_j^r is not an improvement over u_0 and is worse than the next worse point u_{j-1}, a contraction point u_j^c is generated. This serves as a mechanism to move the simplex away from poor-performing regions of the solution space. u_j^c is defined as:

$$u_j^c = c + \beta(\tilde{u_j} - c), 1 > \beta > 0$$

where $\tilde{u_j} = u_j^r$ if $g(u_j^r) < g(u_j)$ and $\tilde{u_j} = u_j$ otherwise. β is an algorithm parameter with a default value of $1/2$. Should the contraction point be better than u_j, it becomes the replacement point u_j'. In this case, the reflection probed a worse region of the solution space, and the simplex is thus moved away from it, but also shrunk in the process.

If a replacement point u_j' was assigned, it replaces u_j in S, the new simplex $S' = u_j' \cup u_i, i \in [0, n]$ is reordered by target function value, and the procedure is restarted by generating a reflection point from the worst point in S'. Since the worst point of the simplex has been replaced, and at least one new point is present in the simplex in this case, the iteration will generate a different worst point u_j' and centroid c' each, leading to a new reflection point.

If the contraction point u_j^c is not an improvement over u_j, the simplex is instead shrunk towards the current best point u_0, so that:

$$\begin{aligned} S' =& [u_0, (\tau u_0 + (1-\tau)u_1), (\tau u_0 + (1-\tau)u_2), \\ & ..., (\tau u_0 + (1-\tau)u_{j-1}), (\tau u_0 + (1-\tau)u_j)], \\ & 0 < \tau < 1 \end{aligned}$$

The algorithm parameter τ is set to the default value [39] of $\tau = 1/2$. This case implies that no improvement could be found through expansion and that the contraction failed by delivering a worse solution than u_j. This is usually a rare case, since contraction should provide a better solution in a smooth landscape in most cases [36]. S is then shrunk in the hope of finding a smoother landscape. When the process is restarted with a shrunken S' and a new reflection point is generated, it is located closer to other points, and the algorithm explores a smaller, hopefully smoother part of the solution landscape.

Both when shrinking and contracting the simplex, the volume covered by S' becomes smaller. The algorithm ultimately converges to a single solution. This brings the risk of becoming stuck in an optimum. This is addressed by covering a large area of the solution space with the initial simplex and by using a reinitialization procedure. If the simplex converges to a solution, a reinitialization is performed. The solution to which the algorithm has converged is chosen as the initial parameter, and a new simplex is

generated using the initialization method [37]. This is necessary to keep the controller simulating new parameters and potentially explore new solutions, should the input change. A converged simplex with only one distinct solution otherwise does not generate any new points and thus has no chance of evaluating different solutions.

During testing, situations where all evaluated solutions in one iteration of the algorithm yielded invalid results were encountered. These solutions would either cause a collision of two vehicles or a timeout in the simulation. Such cases are evaluated with a score of $g(u) = \infty$. In this situation, if all initial parameters $u \in S$ and reflections u^r have the same score of ∞, S will be shrunk, since there are no indications for expansion, reflection, or contraction. Doing so, the algorithm risks being stuck in a part of the solution landscape that has only invalid solutions and thus never returning a valid solution. If such a case is identified, a reinitialization with a randomly generated u as the initial parameter is performed.

In its original implementation, the Downhill-Simplex algorithm only evaluates one parameter at a time, as calculations always depend on the results of previous evaluations. Since evaluation is far more computationally expensive than parameter generation in the problem at hand, a parallel implementation [39] of the algorithm is used, allowing the controller to evaluate up to p parameters simultaneously, where p is the minimum of n and the number of available processors. The parallel Downhill-Simplex algorithm reflects the worst p points in each iteration and evaluates them. Then, for each of the p worst points, expansion or contraction points are calculated if indicated by the previous results. Finally, a new simplex is generated. The p worst points are replaced by better points from their expansions and contractions if applicable, or the simplex is shrunk towards the best point if no improvements were found in other directions. The dimension of the solution space n is a limit to parallelization. On lower-dimensional instances of the problem, there are thus fewer simulations evaluated in parallel. When the number of remaining decision points n is smaller than the number of available processors, the algorithm cannot utilize the full available parallelization potential.

4.2 Memetic Algorithm

Memetic Algorithms (MA) are a further development of Genetic Algorithms (GA) [40] by incorporating problem knowledge through local search. This subsection will briefly explain the concepts of GA and MA and give insight into the considerations taken for the MA implementation used in the RGS controller.

GAs are a family of metaheuristic optimization algorithms. They keep a set of candidate solutions that are evaluated and altered in order to find better results, without requiring knowledge about the concrete optimization problem. A set of candidate solutions is referred to as a population. By considering multiple candidates at the same time, the algorithm gains a global perspective on the solution landscape. As the name Genetic Algorithms implies, they are inspired by evolutionary biology: Once a population is evaluated, different operators are applied to it, which are modeled after biological phenomena. These typically include selection (choosing candidates for recombination based on some criteria), recombination (creating a new offspring candidate from multiple

parent candidates), and mutation (randomly altering candidates). A typical GA starts with an initial population, which can be a set of randomly chosen candidates or the result of another initialization algorithm. The initial population is then evaluated against the target function. The candidates are sorted by target function value, the best solutions are selected, and new candidates are created using the aforementioned operators. The resulting set of candidates is evaluated again, and the process is repeated. One such cycle of altering and selecting candidates is referred to as a generation.

Memetic algorithms make use of problem knowledge by adding a local search operator. Solutions from the neighborhood of each candidate in the population are frequently evaluated and can replace candidates in the generation if they perform better. A memetic algorithm was selected for the RGS controller due to multiple factors. Due to their distinct phases, GAs in general have great potential for parallelization. The available computational power can be utilized to its full potential with a GA, as the number of candidates in a generation is typically larger than the number of available processors.

The global perspective of a GA was also assumed to be useful. Instead of moving all observed candidates towards a single solution, like Downhill-Simplex, GAs keep a diverse set of solutions and aim to improve upon good features in a population. As new features are constantly injected through the mutation operator, the risk of converging to a single, suboptimal solution is reduced. Due to the dynamic nature of the problem, the evaluations of solutions might change over time. Thus, observing many solutions simultaneously allows for fast reactions to changing environmental states.

MAs specifically introduce an additional local search operator. This fits the problem structure of the dynamic optimization problem well, as solutions generated after t_{DENM} are evaluated based on their difference from the first sent solution $u_{t_{DENM}}$. The local search operator is assumed to generate solutions that are closer to ones in the current generation, thus they are more likely to be close to $u_{t_{DENM}}$.

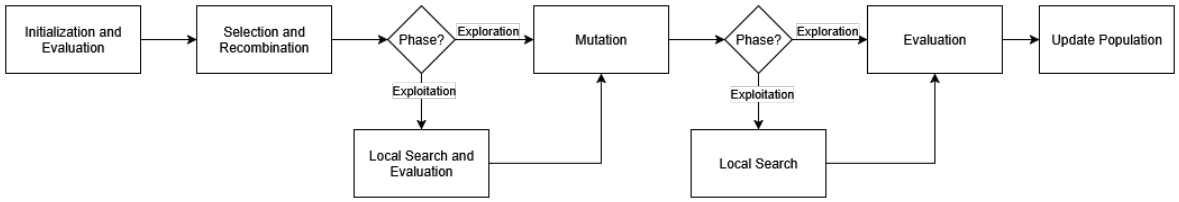

Figure 7: The steps of the memetic algorithm as used in the RGS controller. Depending on the phase, local search is applied to members of the intermediate population.

The memetic algorithm implementation used in the RGS controller follows this general architecture as laid out in Figure 7. It operates in two phases titled **exploration** and **exploitation**. The procedure starts in the exploration phase when the first CPM is received. During this first phase, the algorithm operates like a typical Genetic Algorithm and does not perform local search between steps. This is done to save computation time during the exploration phase, as each local search candidate has to be evaluated in the simulation before the algorithm can proceed. Instead, all candidates of a generation are only evaluated before the population is updated and a new generation is started. The

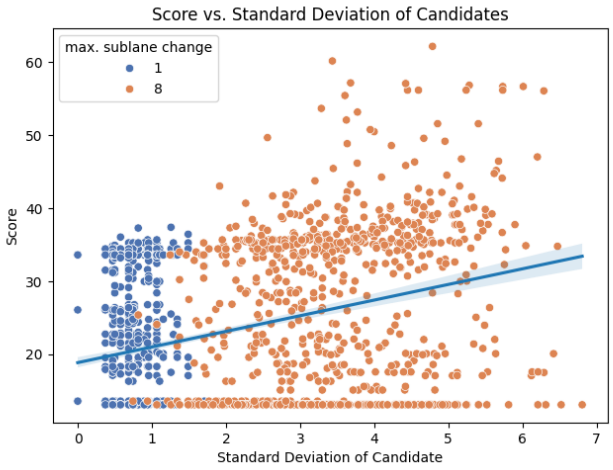

Figure 8: Score and Standard Deviation of Candidates with different values for l, evaluated on Scenario 2 with average traffic density. The blue line is a linear regression line fitted to the results, implying that a higher standard deviation leads to a higher, thus worse score for the evaluated candidates.

phase switches to exploitation when the DENM has been sent out. From then on, local search is included in the process.

Initialization A first population is generated in the Initialization step, half of which consists of randomly generated candidates. The other half is generated based on an assumption about traffic: lateral movement slows down a vehicle. Especially with high resolution of sublanes and a high value for l, the maximum sublane change between two steps, random candidates tend to involve many sublane changes. This hypothesis could be confirmed by sampling random candidates with $l = 1$ and $l = 8$ and comparing their standard deviations, as shown in Figure 8.

The other half of the initial population was thus filled with candidates representing straight paths along the available route, as it is assumed that they offer good starting points for solutions with a low t_{emg}. The question may arise why the other half of the population is still chosen randomly or why a low l-value is not simply used for all generated parameters. The assumption about a low standard deviation of a parameter has only been proven on small examples like Scenario 2. Situations the RGS Controller can encounter, such as a left turn when currently on the right-most lane, require lateral movement by multiple sublanes. To keep variety in the population, random candidates are still included.

Once generated, the initial population is evaluated, and the algorithm proceeds.

Selection The first step for each new population is Selection. This operator chooses candidates on which the next generation will be based, typically those with high fitness values. Parameters are selected randomly and compete in a tournament, the winner

of which becomes eligible for recombination. For the tournament, n_{trn} members of the current population are selected, where n_{trn} is a configurable parameter. The selected member with the highest fitness, or the lowest target function value, wins. The higher the value for n_{trn} is, the more likely it is that an above-average candidate will win the tournament. This is also referred to as a higher selection pressure. A higher selection pressure typically means the algorithm converges to a solution faster, with the risk of it being a suboptimal solution. In the exploration phase, $n_{trn} = 2$, whereas $n_{trn} = 3$ in the exploitation phase.

Recombination When a sufficient number of parameters has been selected according to the selection rate, the Recombination operator is applied. Recombination generates a new candidate from existing solutions, hoping to preserve good features of both candidates. While a range of sophisticated approaches [41] exist for recombination, they usually require evaluation of intermediate candidates, which is computationally expensive. The RGS controller thus uses a simple crossover operator that creates two offspring from two random parents from the previously generated selection.

Local Search If in the exploration phase, the algorithm now performs the first iteration of local search on the current population. To do so, for each candidate u, a set of random neighbors $neigh(u)$ is generated and evaluated in the simulation along with u itself. A neighbor of u is defined as a valid solution u' that differs from u in exactly one decision point. To limit the computational impact of the local search, $neigh(u)$ has a size of 4 for each u. After the evaluation, each u is replaced in the current population with its best performing neighbor $u' \in neigh(u)$, if $g(u') < g(u)$.

Mutation In the next step, the mutation operator is applied. Candidates are randomly selected and altered to introduce new features into the population. The mutation operator chooses one decision point from each selected candidate u and changes it to a random value within its possible bounds. The newly generated candidates are added to the population. If the algorithm is in the exploitation phase, another local search iteration is conducted. Regardless of phase, the population is then evaluated.

Mutation and neighbor generation use the same basic operation; they alter an existing candidate to generate new features. The key difference between mutation and local search is that neighbors are evaluated before a candidate is added to the population, and only one of them is eventually added. Mutated candidates are added without prior evaluation. This guarantees new features being injected into the population, even if mutated candidates perform worse than the solutions they are based on.

Update Population The current population pop_n now consists of modified candidates from the last population pop_{n-1}. In the next step, both are merged into a new population pop_{n+1} using the random replacement strategy [42]: pop_{n-1} is sorted by fitness value and each $u_i \in pop_{n-1}, i \in 0, .., size(pop_{n-1})$ is replaced with a probability of $\frac{i}{size(pop_{n-1})}$. The replacing candidate is always the best candidate of pop_n that has not yet been added

to pop_{n+1}. With this step, a generation is complete, and the algorithm is restarted, as pop_{n+1} goes into the Selection and Recombination step.

The MA implemented in the RGS controller has a generation size of 48 candidates. Trails showed that larger generations would lead to unsatisfying results, as no more than a single generation would complete before t_{DENM} in complex scenarios. Since operations are applied on an entire generation at once, parallelization can be used to its full extent if the number of available processors is smaller than the generation size.

4.3 Reinforcement Learning

Reinforcement Learning (RL) is a subfield of Machine Learning in which an agent is trained to make decisions through repeated interaction with an environment. A sequence of such interactions is called an **episode**. At a given timestep t, the agent receives a state of the environment, also known as an **observation** s_t, and a **reward** r_t that quantifies the agent's performance. The agent then outputs an **action** a_t that influences the environment, leading to a new state, thus a new observation and reward, as displayed in Figure 9. This process is repeated until the episode ends. The goal of the agent is to establish a **policy** θ that recommends actions based on observations to maximize the reward accumulated over an episode.

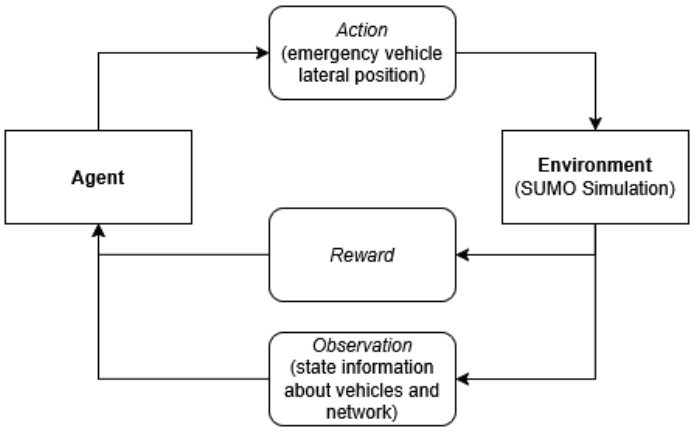

Figure 9: General concept of reinforcement learning as it is used in the RGS controller. The agent observes the current state of the simulation at each decision point and generates a new lateral position for the emergency vehicle.

The way RL works can be well-mapped to simulation optimization problems. Simulations calculate discrete time steps, in between which observations can be generated and actions can be applied. And while the computational cost of training an RL agent can be large, its usage is computationally inexpensive compared to other optimization algorithms. These properties make RL an appealing method for the RGS controllers' optimization problem.

Deep Reinforcement Learning One possible approach to decision-making in an RL setup is through a neural network. This process is called Deep Reinforcement Learning (DRL). A neural network consists of nodes called neurons organized in layers. Each neuron is connected to the previous and following layers, and each connection is weighted. A neuron is activated when the sum of all incoming connections exceeds an activation threshold called a bias. The connection weights and biases can be adjusted. A neural network has an input layer to which data is applied. It also has an output layer indicating the decision result. Between them are hidden layers that guide the decision-making process. The neurons in the input layer react to the input data. If an input value surpasses the bias of an input neuron, it fires, meaning that it generates a signal to all neurons on the next layer. This signal is weighted by the network's weights, potentially leading to new activations in that layer. In this way, signals are propagated through the network to the output layer. The neurons of the output layer fire if their bias is exceeded by the sum of the incoming signals. The activated neurons on the output layer are then typically interpreted as a decision made by the neural network.

The current policy θ of the agent is characterized by the weights and biases of the neural network. A policy can be deterministic and always decide on the same action a_t for a given state s_t. It can also be stochastic, meaning that for any given state, an action is chosen randomly following a probability distribution dictated by θ. The probability for an action a_t being taken form a state s_t is denoted as $\pi_\theta(a_t|s_t)$.

Policy Gradient Methods To make decisions, a neural network has to be tuned to a specific problem in a process referred to as training. In DRL, a neural network is trained with reinforcement learning, meaning that θ is gradually changed during training. There are different techniques to accomplish this. Proximal Policy Optimization (PPO) [43] was selected for the RGS controller, as it has been successfully used in similar traffic simulation problems [7] [44] [45]. PPO is a policy gradient method; it estimates the gradient of the policy with regard to the achieved reward to improve performance. Intuitively, a function maps the parameters θ of the neural network to the reward these parameters will generate. The reward received with different θ values is observed, and θ is moved in the direction where the highest reward is expected. Multiple episodes are run in a batch, and the policy is updated based on the results. Policy gradient methods generally use a loss function L^{PG} to choose the direction in which the policy shall be improved. It quantifies the difference between expected and actual rewards achieved by a policy and is defined as:

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t[\log \pi_\theta(a_t|s_t)\hat{A}_t]$$

$\hat{A}_t$ is the advantage function, the difference between the estimated and actual rewards of an action a_t. $L^{PG}(\theta)$ thus returns a positive value if an action has performed better than anticipated, and the likelihood $\pi_\theta(a_t|s_t)$ with which a_t is chosen from s_t is increased according to the result. The opposite happens should the action perform worse than expected. The expectation $\hat{\mathbb{E}}_t[...]$ indicates the observed advantage over a batch of episodes.

The log operator is used to reduce the amount of change applied in a single step.

However, policy gradient methods using L^{PG} can still change their policy radically in a single update. The Trust Region Policy Optimization (TRPO) [46] algorithm, on which PPO is based, aims to avoid such potentially destructive updates. It considers the ratio r_t between the old policy $\pi_{\theta_{old}}$ and the current π_θ. TRPOs uses loss function L^{CPI}:

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t] = \hat{\mathbb{E}}_t[r_t(\theta)\hat{A}_t]$$

L^{CPI} is subject to a KL-divergence[4] constraint that prevents changes between updates from becoming too large. This constraint holds the KL-divergence of the old and new policies against a pre-defined threshold. While this constraint limits the size of updates, it adds complexity to the algorithm and increases computation time during training.

PPO builds upon the intuition of TRPO, but instead of the KL-constraint, it builds a clipping operator directly into its loss function L^{CLIP}:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)]$$

ϵ is an algorithm parameter with a default value of 0.2. The clipping function in combination with the minimum operator in L^{CLIP} makes sure that large performance gains in training do not lead to overly enthusiastic updates on the policy. It also ensures that large decreases in performance by recent changes can be corrected by returning to the old policy in larger steps.

PPO uses an actor-critic configuration using two neural networks, which is outlined in Figure 10. The actor network represents the policy and is trained using L^{CLIP}. The critic network provides the reward estimation used in the advantage function $\hat{A}$. It learns from the error in its estimation compared to the observed rewards. PPO runs minibatches of a small number of episodes each. After a minibatch, the advantage function is evaluated, and the policy is updated. The critic network can be updated in parallel. It is also possible to have the actor and critic share the same network, which then requires a combined loss function that accounts for the advantage function.

PPO in the RGS Controller The RGS Controller uses the `sable-baselines3` [49] Implementation of PPO. The agent was trained in a custom `gymnasium` [50] environment. An episode consists of one SUMO simulation run. During each simulation timestep, the agent receives an observation and the current reward from the environment, and generates an action. The `gymnasium` framework offers boilerplate code for training and allows easy integration of different reinforcement learning algorithms for possible future extensions of the RGS controller.

[4]Kullback-Leibler divergence $D_{KL}(Q||P) = \sum_{x \in X} P(x) \log \frac{P(x)}{Q(x)}$ [47], which quantifies the difference between probability distributions Q and P on a set of variables X

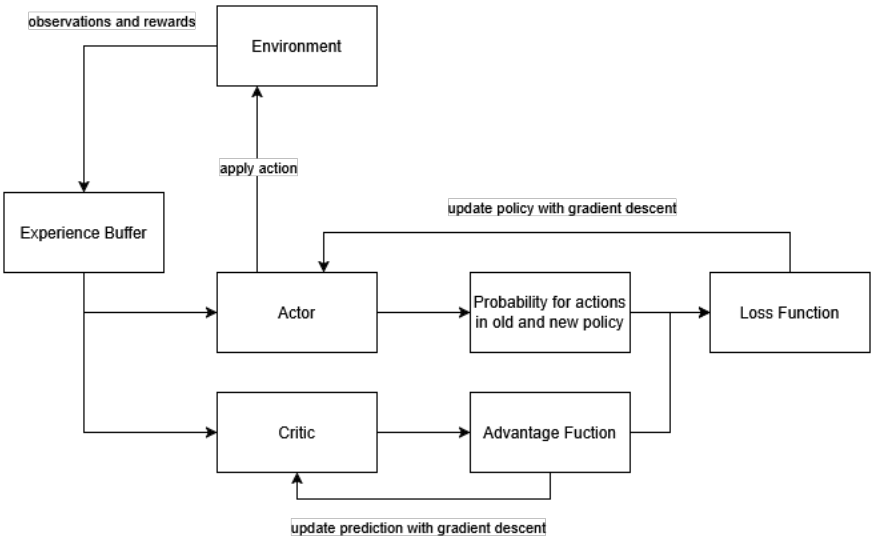

Figure 10: The actor-critic-architecture of PPO. Batches of observations and rewards are stored in the experience buffer and evaluated by the actor and critic for their corresponding purposes. Figure based on [48].

The observation is based on the perspective of the emergency vehicle. If multiple emergency vehicles are present in a scenario, the perspective of the leading vehicle is taken. For each other vehicle, the position relative to the emergency vehicle is saved along with speed, heading, and the lateral position on the road. All values are normalized to the interval $[0, 1]$. If a vehicle is not on a road segment belonging to the emergency vehicles route, its lateral position is set to -1. This allows the agent to discern vehicles on the route from those that are not. Vehicles behind the emergency vehicle are ignored in the observation. The observation also includes the width of the upcoming road segments.

The reward function is based on the objective function g. It assigns a small negative reward for each simulation step in which the emergency vehicle is in the simulation, and a small negative reward for each traffic rule violation. Whenever a collision occurs, the agent receives a large negative reward. Upon successful completion, when the emergency vehicle leaves the simulation before a timeout occurs, a large positive reward is awarded. Thus, the only way to achieve a positive reward at all is to successfully complete the scenario.

For training, a set of synthetic scenarios that resemble traffic situations from the evaluation scenarios was designed. They differ from the evaluation scenarios in length, number of lanes, and layout, but depict similar traffic situations. The number of vehicles and their spawn positions are randomized to generate varied training data and avoid overfitting the agent to specific scenarios. A random scenario is selected for each training episode, and the agent does not directly receive the information about which scenario it is currently in. The model was trained for $2 \cdot 10^7$ timesteps, after which convergence was reached, as shown in Figure 11.

The reinforcement learning *OptimizationStartegy* implementation in the RGS controller

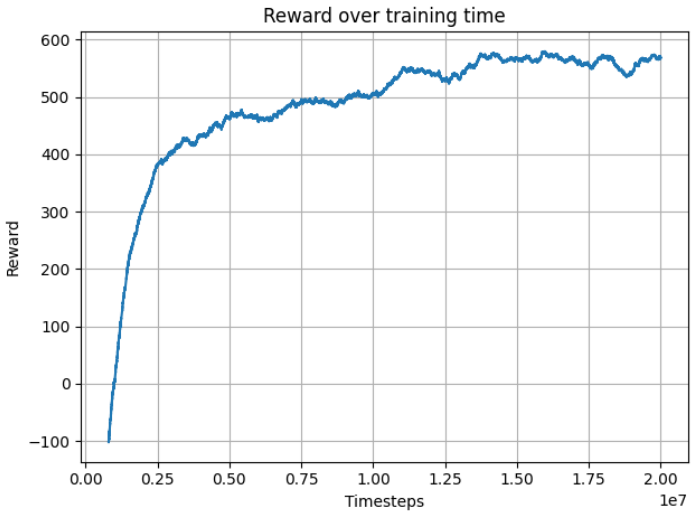

Figure 11: The reward received by the RGS controller's PPO model during its training process. The reward is displayed as a moving average over the last 10000 timesteps for better readability.

uses the trained PPO model to generate solutions based on the current CPM. Whenever a CPM arrives and is passed to the *OptimizationStartegy* by the *Optimizer*, it is loaded into SUMO and an observation is generated. The observation serves as the input to the PPO model, which then generates a solution. This solution is then evaluated using the *SimRunner* as in other strategies. As the model generated by PPO is stochastic, the RGS controller generates multiple solutions from a single observation, which can be evaluated in parallel.

5 Experimental Setup

This section describes the experiments conducted to answer the research questions of this thesis. It first presents the different evaluation scenarios and relevant parameters for the RGS controller as used in the evaluation. Finally, the experiments are explained in detail.

5.1 Evaluation Scenarios

The evaluation scenarios were designed based on an expert interview [26] conducted during the preceding study project. They represent situations typically encountered in emergency responses. Each optimization algorithm is evaluated on each scenario, in two different traffic demand configurations: average traffic density and heavy traffic density. In the context of this evaluation, average traffic density means 25 vehicles per km of road, whereas high traffic density means 55 vehicles per km for all synthetic scenarios. The estimations for traffic densities are based on traffic camera experiments [51]. The traffic densities for the real-world scenario, the Tostmannplatz intersection in Brunswick, are based on measurements conducted by the city of Brunswick and publicized in a traffic demand map[5]. The scenarios are designed around a t_{DENM}, the point in time at which the first DENM is sent, of 5 seconds.

Figure 12: A section of scenario 1, showing the two lanes of the road and the sidewalks. The emergency vehicle overtakes other vehicles on the left lane. They are marked green, as they are evading to the right.

Scenario 1 The first scenario consists of a straight, two-lane one-way road segment with a length of 600 meters. Sidewalks are present for evading vehicles or the emergency vehicle to use. The speed on the road is limited to 50km/h, while the emergency vehicle may move with up to 100km/h. The purpose of this scenario is to test whether the emergency vehicle can overtake all other vehicles before they reach the end of the road. In a real traffic scenario, the need for an emergency corridor at an intersection or obstacle further down the road could be prevented by such overtaking maneuvers, saving valuable time for the emergency vehicle. Scenario 1 is visualized in Figure 12.

[5]The traffic demand map (German: Verkehrsmengenkarte) of Brunswick is available at: `https://www.braunschweig.de/leben/stadtplan_verkehr/verkehrsplanung/verkehrsmengenkarten.php`

The vehicles added in this scenario are placed within 50-300m of the start of the road. Vehicles placed further behind will likely leave the simulation before the emergency vehicle has a chance to overtake them, thus invalidating the purpose of the scenario. The first 50 meters are left free to give the controller initial time to calculate a first solution u_0. With the defined traffic density values, scenario 1 has 6 vehicles in the average-density case and 14 vehicles in the high-density case.

Figure 13: The traffic light in scenario 2. The emergency vehicle can pass through a corridor formed in the middle of the road, as shown. It can alternatively use the sidewalks, which requires other vehicles to move further towards the center of the road.

Scenario 2 The second scenario presents the emergency vehicle with a typical obstacle encountered in urban areas: a traffic light. It is placed on the 200m mark of the one-way road segment, which again has two lanes and sidewalks, as seen in Figure 13. This scenario aims to examine the behavior in the direct vicinity of the traffic light; thus, the road is only 300m long in total. The goal in this scenario is to have the emergency vehicle get past slow-moving or even standing vehicles as fast as possible. The speed constraints are the same as in scenario 1.

Again, the first 50m of the road are clear of other vehicles. They are placed up to 150m in front of the traffic light. The traffic light is red for 30 seconds. Should the emergency vehicle be unable to pass it before it turns green, it will suffer a large penalty in the t_{emg} component of the target function. The average traffic density case thus results in 4 vehicles, and the high-density case in 8 vehicles.

Figure 14: The blocked road segment in scenario 3. The entry of the segment has potential for collisions if multiple vehicles enter it carelessly.

Scenario 3 In this scenario, the emergency vehicle again has to pass an obstacle in front of it; this time, one of the two lanes is blocked on a part of the road segment, as displayed in Figure 14. The one-way segment has a total length of 300m, two lanes and sidewalks, but between 200m and 250m from the start, the right lane and sidewalk are blocked. Additionally, the speed limit on the one-lane segment is reduced to 30km/h, while the emergency vehicle may go up to 60km/h on it. The same speed limits as in the previous scenarios apply to the other segments. This scenario simulates difficult traffic situations like construction sites, where space for maneuvering is limited.

As in scenario 2, vehicles are placed between 50m and 200m from the start of the scenario, again with 4 vehicles being present in the average-density case and 8 in the high-density case.

Figure 15: A section of the two-lane road in scenario 4. With no sidewalks available, the emergency vehicle has to use the opposite direction lane for overtaking maneuvers.

Scenario 4 Scenario 4 features a 600m two-lane road segment similar to that in Scenario 1, except that there are no sidewalks and it is a two-way road. The emergency vehicle can move into the opposite direction lane to overtake vehicles on its own lane. Careless movement on the opposite lane can however result in a collision. The controller needs to find a balance of risk and reward, weighing time savings against potential collisions, which would render the entire simulation run unsuccessful.

Using the same argument as in Scenario 1, vehicles are placed on the right lane between 50m and 300m. Since the traffic density is valid for the entire road, 3 vehicles are present on the right lane in the average-density case, and 7 are present in the high-density case. Vehicles are placed only on the first third of the left lane, which goes in the opposite direction of the emergency vehicles' route. This gives the emergency vehicle opportunities to overtake at the beginning of the scenario. This results in 3 vehicles being present on the left lane in the average-density case and 6 in the high-density case.

Scenario 5 Combining elements of previous scenarios, scenario 5 simulates a junction of two two-lane two-way roads with a traffic light. The scenario thus consists of four 150m segments connected by an intersection in the middle. Each segment is 90° apart from its neighbors. Sidewalks are present in the entire scenario, and opposite-direction lanes can be used for overtaking maneuvers. The emergency vehicle approaches the intersection from the south, and the traffic light is configured to be red for 30 seconds for vehicles coming from the north or south. There are three variations of this scenario with different

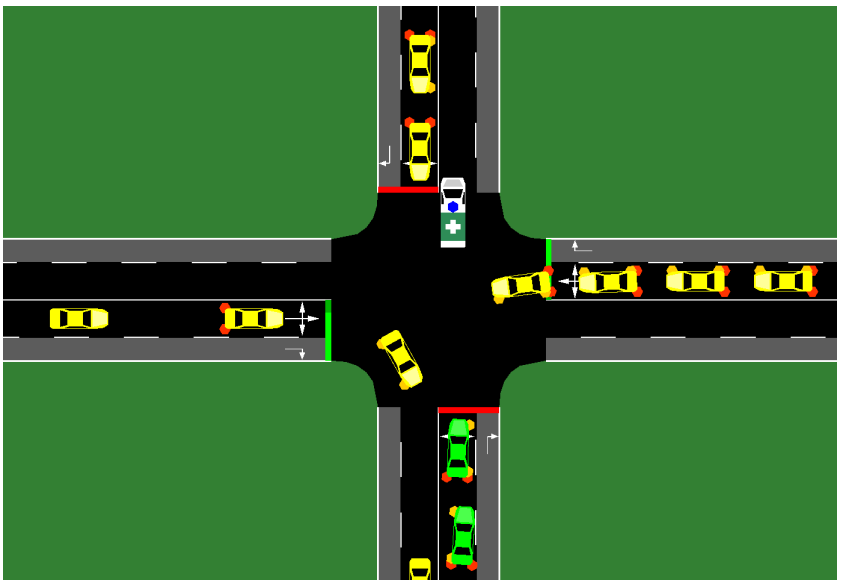

Figure 16: The central intersection in scenario 5. All roads in the scenario are two-way roads with one lane in each direction. Oncoming traffic has to be considered in overtaking maneuvers, while blocking the intersection needs to be prevented, as it can result in a deadlock.

routes for the emergency vehicle: a straight crossing, a turn to the right, and a turn to the left.

Vehicles are placed on all segments with random routes assigned to them for additional variation. To keep the first 50m of the southern segment free, it holds 3 vehicles in the average-density case, while all other segments hold 4, resulting in a total of 15 vehicles. In the high-density case, 8 vehicles are present on each segment except for the southern one, which holds 6, resulting in 30 total vehicles. Due to constraints in the simulation, the vehicles have to be spawned with a lower starting speed in the high-density case.

Scenario 6 The last scenario is based on the real-world intersection of the streets Bienroder Weg, Mergestraße, and Riekestraße near the Tostmannplatz in Brunswick, Germany. Its SUMO implementation was provided by the DLR. All roads are bidirectional. The north-south axis has separated lanes, meaning overtaking on the opposite lane is not possible there. Sidewalks are present throughout the entire scenario. The emergency vehicle again approaches from the south, allowing for three variations of the scenario, with it going east, north, or west. The traffic light on the main junction is configured to turn green for vehicles coming from the south after 62 seconds, with all other directions turning green before. Emergency vehicles are capable of requesting a green light at intersections, and V2X technology can be used to influence traffic light programs intelligently. One possibility for that would be the integration with DLR's SIRENE [21]. However, the presence of such systems is not guaranteed. Thus, in this scenario, the controller has to manage the situation without support from the infrastructure beyond the detection

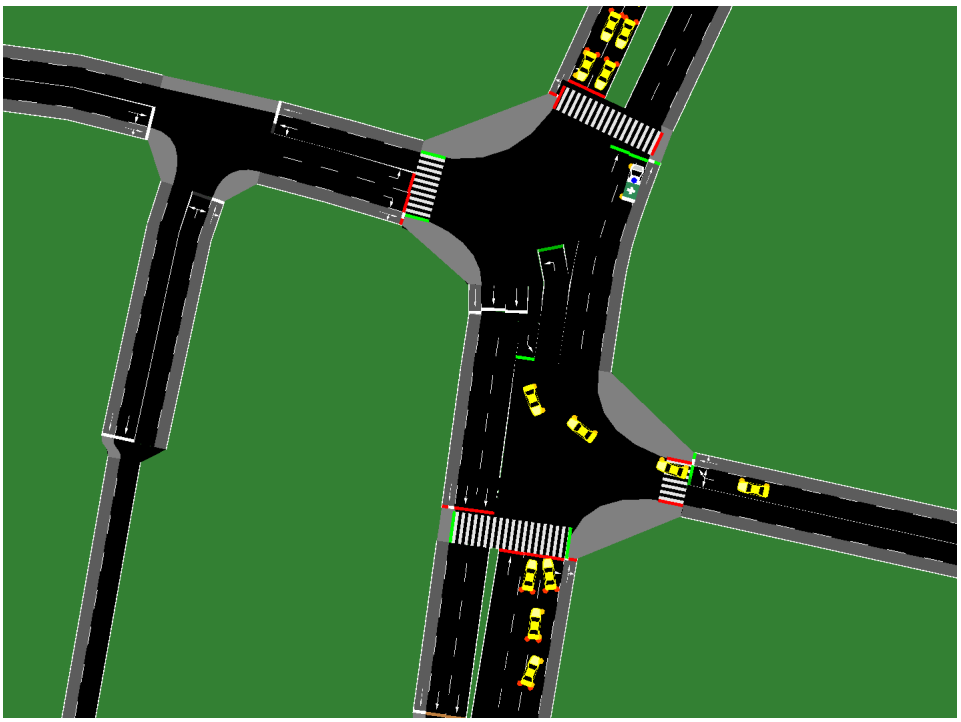

Figure 17: The Tostmannplatz intersection in Braunschweig modeled in SUMO. Different turning lanes in each direction, a mix of structurally separated lanes and two-way roads, and multiple chained traffic signals make this the most challenging scenario in the evaluation suite.

of vehicles. The scenario combines elements of previous scenarios, including sidewalks, single-lane segments, two-way roads, varying speed limits, and a traffic light intersection with random routes for passing vehicles.

As mentioned above, the traffic density calculation for this scenario is based on a traffic map provided by the city of Brunswick. The traffic map shows the traffic demand q, so the number of vehicles passing a road segment within 24 hours on an average work day. This number was converted to traffic density k (number of vehicles per km) with the formula $k = \frac{q}{v}$, where v is the average speed at the corresponding segment. For the average-density case, k was calculated and applied to the segment length as in previous scenarios. For the high-density case, the average-density numbers were multiplied by a factor of 2,2 that matches the difference between average and high density observed in [51]. The resulting number of vehicles present for different densities with corresponding segment lengths is listed in Table 1.

Segment	Length	Vehicles (average density)	Vehicles (high density)
North	252m	3	6
East	216m	1	3
South	152m*	4	8
West	166m	1	2

Table 1: Number of vehicles for each traffic density and road segment in Scenario 6. The southern segment is 202m long in total; however, as in the other scenarios, vehicles are not placed on the first 50m of the segment. The traffic map does not provide data for the eastern segment; the values are thus estimated based on similar roads in the vicinity for which data is available.

An additional scenario was generated from the DLR site at Brunswick airport from OpenStreetMap data. It was used during integration tests and the demonstration of the GAIA-X4 AMS project. Due to its simplicity and similarity to the synthetic scenario 2, it was omitted from the evaluation suite.

5.2 Parameters for Controller and Simulation

This subsection describes how parameters for the simulation and the RGS controller have been determined for the evaluation. The parameters are the same in each evaluation scenario and for each optimization algorithm, as the RGS controller should offer a general-purpose method for generating rescue corridors.

Simulation Parameters The relevant parameters for the simulation are the step length t_{step}, the distance between decision points d_p, the maximum sublane change between two decision points l, and the lateral resolution $w_{sublane}$. The simulation step length t_{step} is given in seconds and represents the time simulated in each discrete step of the simulation. Maneuvers are applied between simulation steps, meaning that a lower t_{step} allows for more fine-grained movement, but makes the simulation computationally more expensive. SUMO's default value of $t_{step} = 1s$ proved to be insufficient for the RGS controller, often resulting in the inability of vehicles to take proper evasive actions. Values as low as $t_{step} = 0.2s$ have been experimented with, but were discarded due to long simulation times, leading to poor optimization performance. Ultimately, $t_{step} = 0.4s$ was found to be a feasible compromise between computation speed and simulation accuracy.

The distance between decision points in meters d_p determines the size of the optimization problem, as the number of decision points n is calculated by dividing the remaining length of the emergency vehicles' route by d_p. It also determines the granularity of the resulting emergency corridor. It is defined as $d_p = 40m$ for the RGS controller. In combination with a lateral resolution, or sublane width, of $w_{sublane} = 0.4m$, this value showed sufficient maneuverability for the emergency vehicle while keeping the problem small enough for optimization strategies to provide solutions. The maximum sublane change between two decision points l is defined as 8 sublanes, implying that a solution u contains a maximum lateral movement of 3.2 meters between two decision points.

This value is again a tradeoff. The maximum lateral movement of vehicles in SUMO is limited to allow for realistic maneuvers. At high speeds, the emergency vehicle is unable to laterally traverse 8 sublanes within 40m. If l were lower, the maneuverability of the emergency vehicle in critical situations at low speeds, for example, at traffic lights, would be severely limited.

Target Function Parameters The target function parameters (a, b, c) are one of the most relevant parts of the configuration. They are used by the outer simulation to assess the performance of an evaluation run and by the RGS controller to evaluate candidates generated by the optimization strategies. a determines the weight of the time the emergency vehicle needs to exit the simulation, t_{emg}. b determines the weight of the total number of rule violations by all vehicles num_v. c determines how much a solution is penalized for its distance to the first sent solution $u_{t_{DENM}}$ in the RGS controller.

As the emergency vehicle time t_{emg} is the central optimization goal, a is defined as 1, and other parameters are defined relative to it. b is tied to t_{step}, as rule violations are counted in each simulation step. The value for b was chosen to be $t_{step}/2$. This implies that for each two violations committed, t_{emg} would have to be at least one simulation step better to achieve an equal score.

The Euclidean distance is used to quantify the difference between a solution u and the first sent solution $u_{t_{DENM}}$. u receives a penalty of $d \cdot dist(u, u_{t_{DENM}})$ on its score. A very low value for c would encourage the controller to send solutions that are only marginally better than the first solution $u_{t_{DENM}}$, possibly forcing vehicles to move after having already formed a valid emergency corridor. Too high values for c make the controller unable to react to changes in the traffic situation and force it to generate a good solution by t_{DENM}, as any better solution generated later will be disproportionally penalized. Experimental results showed that large c values would lead to poor performance in complex scenarios. Algorithms would be encouraged to converge towards $u_{t_{DENM}}$ and mostly disregard other optimization goals. Large deviations at single decision points are already taken into account by the square operator in the Euclidean distance, meaning that such deviations would quickly add up to a large penalty in the score. For the evaluation, c was defined as 1, which was found to offer a reasonable tradeoff between consistent and good solutions.

5.3 Definition of Experiments

In order to answer **RQ1** and evaluate the optimization strategies on synthetic scenarios, the RGS controller is run with an outer simulation on scenarios 1-5. To answer **RQ2** and discuss performance on real-world scenarios, the same setup is used on scenario 6. The evaluation runs for these two cases are combined in **Experiment 1**. The testing machine in all experiments has a Ryzen 5 3600X CPU with 6 physical and 6 additional virtual cores operating at a frequency of 3.8 GHz, and 16GB of RAM.

To ensure statistical significance of the results, each simulation configuration is run 50 times. A simulation configuration is defined by the strategy employed, the scenario, the traffic density, and the scenario variant, if applicable. Variants exist only for scenarios 5 and 6; they represent different routes the emergency vehicle takes. The routes are defined

by the direction the emergency vehicle turns at the intersection from its point of view; they can be either **straight**, **left**, or **right**. An example of a simulation configuration would thus be the Downhill-Simplex Algorithm, scenario 5, left turn, average traffic density.

A set of baseline results was generated for each scenario in Experiment 1. These baselines are DENMs based on typical best practices for rescue corridors and were created by a static analysis of the evaluation scenarios. For each scenario, the underlying road network was analyzed without regard for any specific traffic density. The rescue corridor is thus typically located to the right of the left-most lane. Comparing the strategies to baselines reveals whether a dynamic optimization approach, as proposed with the RGS controller, outperforms a static, rules-based approach, which could be implemented with less effort. The baseline solution is run 50 times for each configuration. It is treated as a DENM that is contentiously sent with the same emergency vehicle trajectory, starting at t_{DENM}.

To evaluate the different strategies' performance when some vehicles do not cooperate and address **RQ3**, a separate experiment was conducted, designated as **Experiment 2**. Two scenarios were chosen for this experiment: Scenario 2, as it provides a simple representation of the general emergency corridor formation problem, and Scenario 6 in the straight crossing variant, providing a real-world example. Both scenarios are run with high traffic density. The configurations in this experiment consist of a strategy, a scenario, and a cooperation probability p_{coop}. At the beginning of an outer simulation, each vehicle other than the emergency vehicle is defined to be either cooperative or not cooperative according to p_{coop}. In the case of $p_{coop} = 0.8$, a vehicle will be cooperative with a probability of 0.8. The state of cooperation lasts for the entire outer simulation run. The RGS controller has no knowledge of p_{coop} and internally assumes all vehicles to behave cooperatively. The possible values for p_{coop} are 1.0, 0.9, 0.8, 0.7, and 0.5. Each configuration is again run 50 times.

Perfect communication is assumed in all experiments, meaning that any DENM generated by the RGS controller will be received by all vehicles in the scenario within one simulation step. When the DENM sender is assumed to be at the center of each scenario, this assumption is reasonable, as V2X messages can be reliably transmitted at ranges of up to 400m in urban environments [10].

6 Experimental Results

This section presents the results of the experiments defined in section 5. The results of Experiment 1 are examined first, detailing strategy performance on synthetic scenarios and real-world scenarios, followed by an analysis of the results under aspects not covered by the research questions. The impact of varying degrees of cooperation analyzed in Experiment 2 is presented afterwards. Throughout this section, the figures use the following terms for optimization strategies: the Downhill-Simplex algorithm is referred to simply as *simplex*, the memetic algorithm as *memetic*, and reinforcement learning is abbreviated as *rl*.

6.1 Relative Performance of Optimization Strategies

As described in the previous section, the optimization strategies were compared to each other in different scenarios in Experiment 1. Due to the outer simulation, which emulates the real world in the RGS controller, running in real time, and the high number of configurations, the experiment had a high runtime, taking over 33 hours to complete. This subsection contains figures with aggregated results; the results of individual evaluation scenarios can be found in the appendix.

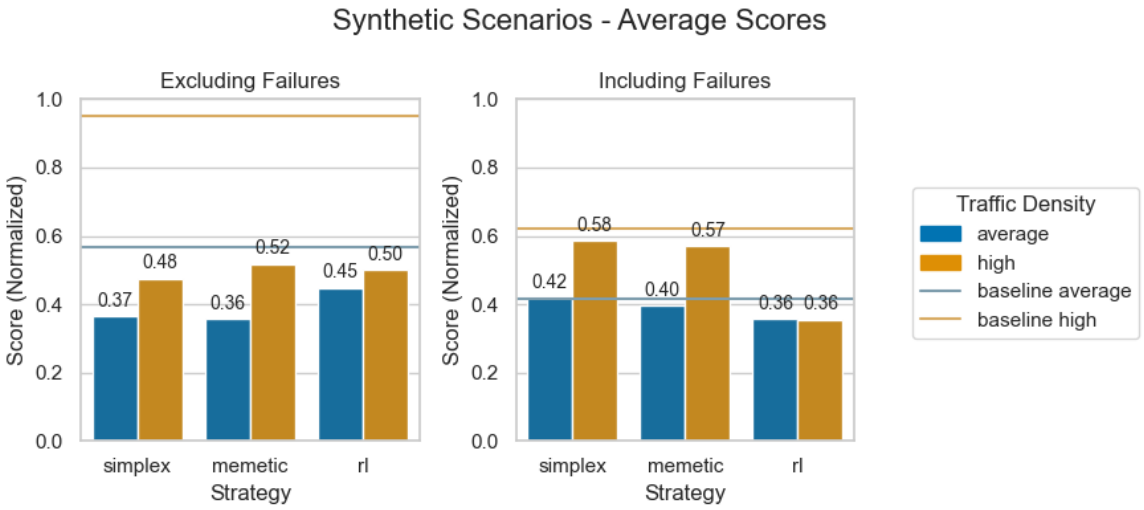

Figure 18: The normalized scores achieved by the optimization strategies in the synthetic scenarios, compared to the baseline and grouped by traffic density. The two subplots show scores excluding and including failed simulation runs.

Synthetic Scenarios The score each algorithm reached in the synthetic scenarios is displayed in Figure 18. A lower score implies better performance. The results are aggregated over all synthetic scenarios and normalized for each scenario, meaning that the best achieved score by any strategy in a scenario is given the value 0, whereas the worst achieved score in a scenario has a normalized score value of 1. The bars denote the

mean normalized score over all scenarios achieved by the respective strategy, also given as a numeric value on top of the bar. They are grouped by traffic density. The horizontal lines represent the mean baseline values for each traffic density, again, aggregated over all synthetic scenarios. The two subplots represent results either including or excluding failures. Any outer simulation run resulting in a failure, either a collision or a timeout, is evaluated with a normalized score of 1.

In scenarios 1-4, also referred to as the basic scenarios, which do not contain inherent randomness in the form of randomized routes for vehicles, the baseline leads to deterministic results. The memetic algorithm and the reinforcement learning agent are themselves non-deterministic. While the Downhill-Simplex algorithm is theoretically deterministic, the parallelization can lead to non-deterministic behavior. Depending on the order in which solutions are simulated by the RGS controller and the time simulations take to complete, they might be based on different CPMs, leading to the possibility of a different result.

As can be seen in Figure 18, all optimization strategies outperform the baseline solution in synthetic scenarios by a large margin, as long as only non-failing results are considered. However, when including failures, the difference between baseline and optimization strategies becomes much smaller. The Downhill-Simplex algorithm does not perform better than the baseline in average traffic density situations and only has a slight advantage in high traffic density situations. The situation is similar for the memetic algorithm, which outperforms the baseline by less than 10% in the aggregated results. Only the reinforcement learning agent is capable of beating the baseline and the other strategies by a significant margin, and only in heavy traffic density configurations. It is important to note that the baseline results did not produce any failures in scenarios 1-4, which only include one possible route for each vehicle.

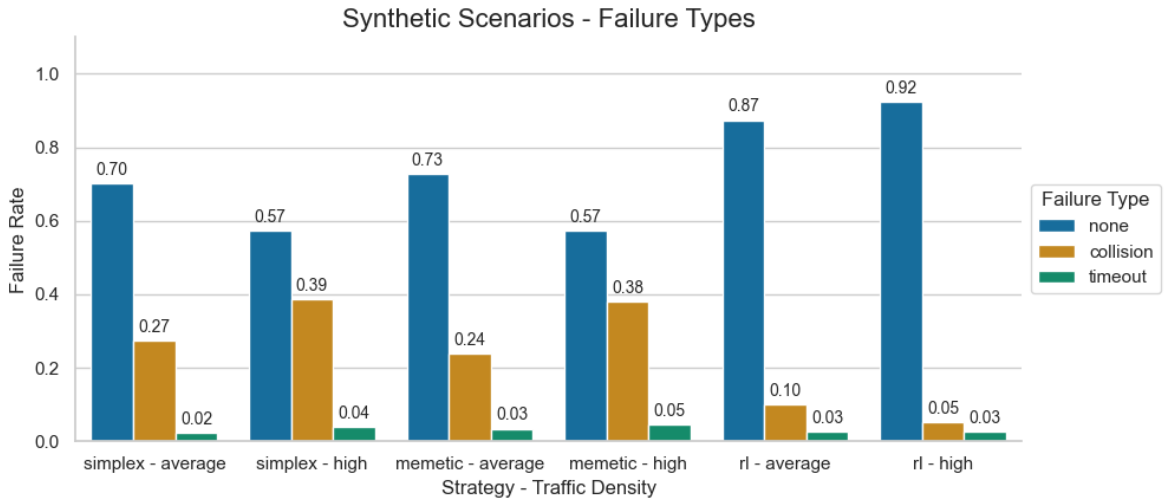

Figure 19: Rates of failure types among the simulation results in the synthetic scenarios, by optimization strategy and traffic density.

More details on failures during Experiment 1 are displayed in Figure 19. For each

combination of optimization strategy and traffic density, the figure contains grouped bars showing the percentage of failures of each type that occurred in the respective simulation configurations on synthetic scenarios.

The relatively low rate of failures achieved by the reinforcement learning agent explains its relatively good performance when also considering failed simulation runs. Another interesting observation is that both the Downhill-Simplex algorithm and the memetic algorithm struggle with higher traffic density. For both strategies, the rate of collisions increased by over 10% when switching from average to high traffic density. Reinforcement Learning, in contrast, managed to reduce its already low collision rate of 10% by half in high-density configurations.

These results warrant further investigation. In scenarios 1, 2, 3, and the straight variant of scenario 5, reinforcement learning achieved lower failure rates in average-density configurations than in high-density ones (see Figures 33, 35, 37, and 41). When looking at the achieved scores, it becomes apparent that reinforcement learning performs better in high-density configurations when including failures because it can produce fewer failures in those configurations. The results that exclude failures show that reinforcement learning does not generally perform better in high traffic density configurations, which can be observed in the aggregated results. The reinforcement learning model was trained on random traffic densities, ranging from the average to high values in the evaluation scenarios. The traffic density usually encountered by the agent in training was thus higher than the average, but lower than the high density in the evaluation. The lower failure rate in high traffic density achieved by reinforcement learning could be an indicator of possible overfitting of the model.

For the other strategies, higher traffic density usually correlates with a higher failure rate in the synthetic scenarios. There are exceptions, such as scenario 4 for the Memetic algorithm (see Figure 39), but in most configurations, this relationship holds.

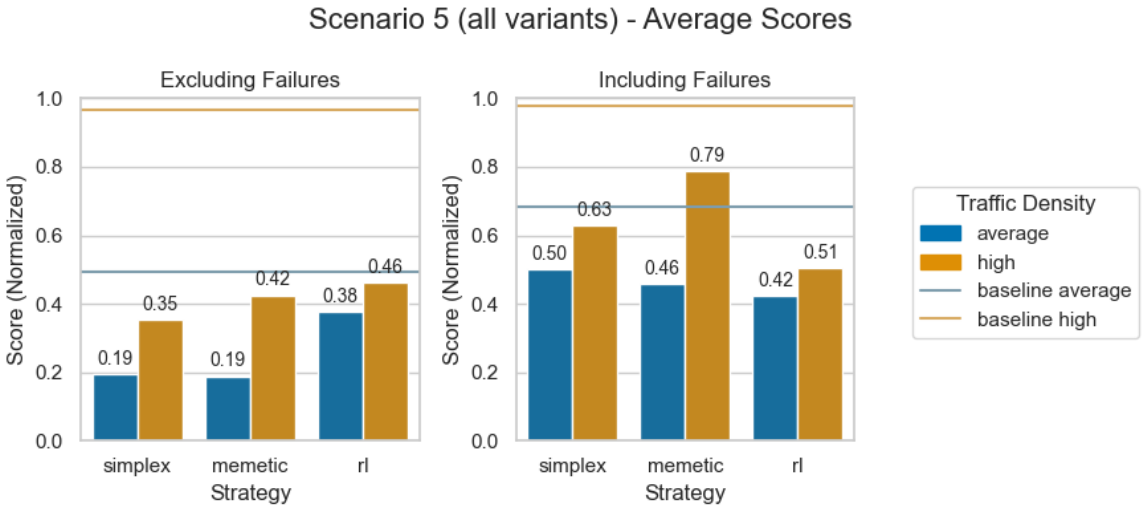

Figure 20: The normalized scores achieved by the optimization strategies in scenario 5, compared to the baseline and grouped by traffic density.

Failure rates influence the scores achieved by strategies in individual scenarios. This becomes apparent when investigating the basic scenarios 1-4, which each consist of a variation of a straight road with no intersections or possibilities for alternating routes. In scenarios 2, 3, and 4, the three strategies have similar performance to each other when only considering successful runs (see Figures 34, 36, and 38), the results however drastically differ when failures are included. In scenarios 1 and 2, reinforcement learning outperforms other strategies due to low failure rates. Downhill-Simplex performs better than Memetic in scenario 2, while performing worse in scenario 4. Reinforcement learning again performs better than others in the high-density case of scenario 3, but significantly worse in the low-density case. While a clear trend across the basic scenarios is not visible, it can be said that reinforcement learning delivers the best results when including failures due to generally low failure rates compared to other strategies.

Contrasting the basic scenarios, the different variants of scenario 5 include multiple possible routes, as the scenario represents an intersection of two roads. The routes for all non-emergency vehicles are assigned randomly at the beginning of the simulation run. Hence, this is the first scenario in which the baseline solution produced different results over multiple simulation runs, as the vehicles react to the solution differently, depending on their assigned route. As shown in Figure 20, the baseline values differ when failures are included, as some evaluations of the baseline solution resulted in failures.

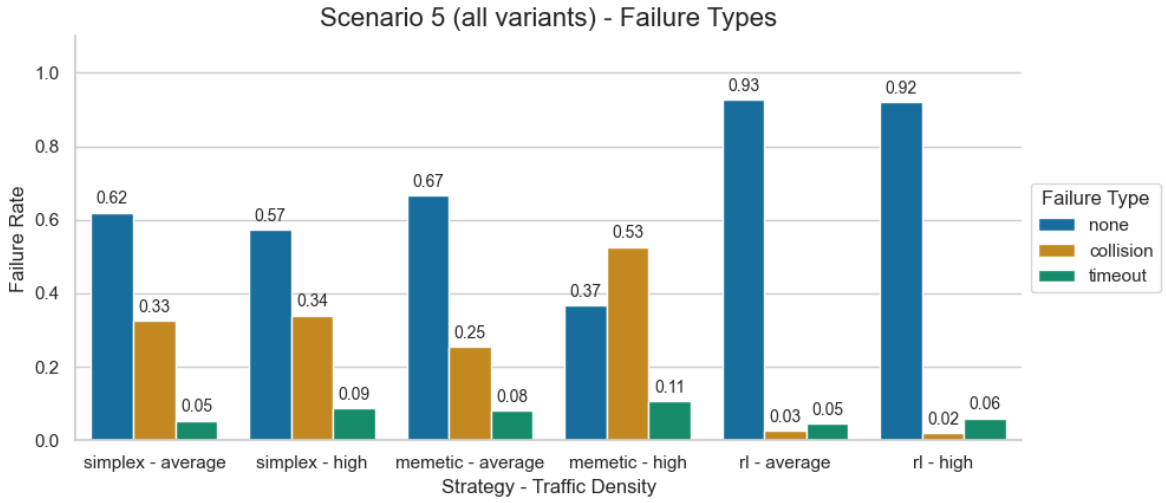

Figure 21: Rates of failure types among the simulation results in scenario 5, by optimization strategy and traffic density.

The scores in scenario 5 show that the Downhill-Simplex and memetic algorithms outperform reinforcement learning when excluding failures. This is especially significant in the low traffic density case. However, due to a low failure rate, as illustrated in Figure 21, reinforcement learning again manages to achieve better results when failures are included. In the high traffic density case, simulation runs controlled by the memetic algorithm only have a 37% success rate, compared to 92% for reinforcement learning.

While the failures produced by reinforcement learning are predominantly timeouts, the failures of the other strategies are mainly collisions.

The results of scenario 5 confirm previous observations, in that reinforcement learning performs well on synthetic scenarios, mainly due to having a lower rate of failures than other strategies. Downhill-Simplex and Memetic perform similarly to each other and can outperform reinforcement learning, but will more often lead to failed simulations, especially in high traffic density configurations. Other than in previous scenarios, Downhill-Simplex and Memetic deliver better results than reinforcement learning as long as failures are excluded.

Real-World Scenario As in scenario 5, the baseline in scenario 6 is influenced by the non-deterministic properties of the scenario and produces failed simulation runs. The results differ from the synthetic scenarios, as seen in Figure 22. The Downhill-Simplex and memetic algorithms outperform reinforcement learning both when excluding and including failed runs. When excluding failures, reinforcement learning performs worse than the baseline in high traffic density, whereas the other strategies beat the baseline. The memetic algorithm performs slightly better than Downhill-Simplex in high-traffic density. The gap between strategies remains similar when failures are included, with reinforcement learning again being outperformed by the others, and Memetic having a slight advantage over Downhill-Simplex in high traffic density.

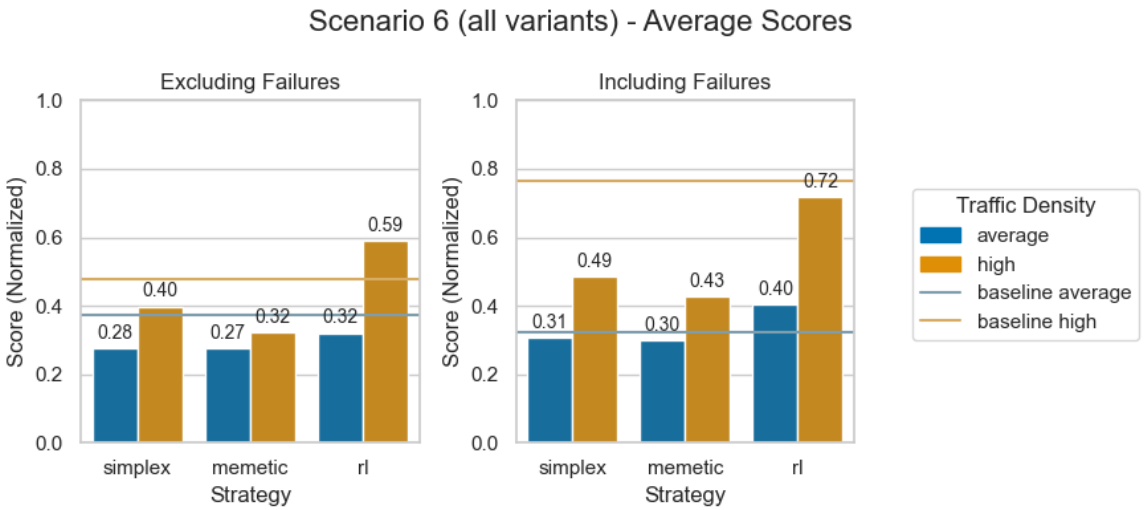

Figure 22: The normalized scores achieved by the optimization strategies in the Tostmannplatz scenario, compared to the baseline and grouped by traffic density.

The failure rates in scenario 6 also show a different picture than in previous scenarios. They are visualized in Figure 23. Reinforcement learning has higher failure rates than the other strategies in both traffic densities. As in scenario 5, a large amount of the failures produced by reinforcement learning are timeouts, typically caused by deadlock situations.

Examination of individual variants of Scenario 6 shows that a large portion of reinforcement learning's comparatively poor performance stems from the right turn variant (see

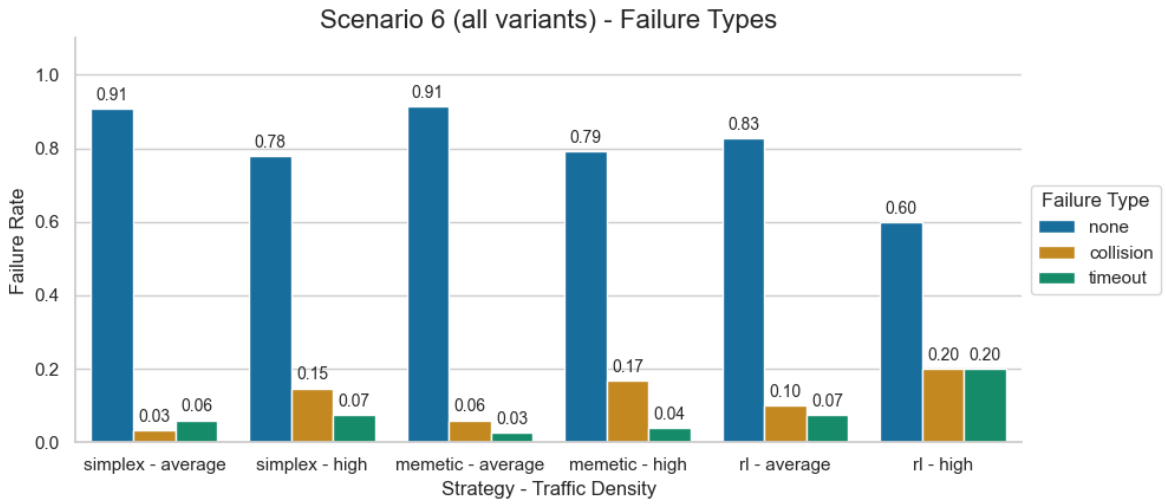

Figure 23: Rates of failure types among the simulation results in the Tostmannplatz scenario, by optimization strategy and traffic density.

Figure 50). Interestingly, reinforcement learning performs poorly even when excluding failures. In the high traffic density case, the mean scores of the other strategies (50.41 for Downhill-Simplex and 53.53 for Memetic) are less than half of the mean score achieved by reinforcement learning, which is 124.96. In the left turn variant, the other strategies again perform better than reinforcement learning. The advantage gets amplified when failures are included, as reinforcement learning has a 52% failure rate in high traffic density configurations (see Figure 49). The straight variant has the most balanced result, but shows the same trend as the turning variants: Memetic performs slightly better than Downhill-Simplex, and both outperform reinforcement learning.

In conclusion, the real-world scenario 6 contradicts the implications of the results on synthetic scenarios, in which reinforcement learning performed best due to a lower rate of failed simulations. As the results for scenario 5 already hint at, reinforcement learning struggles with more complex scenarios, generating worse-performing solutions. While a relatively low failure rate could mitigate this flaw in scenario 5, this is no longer the case in scenario 6, where reinforcement learning generates higher amounts of failures than the other strategies.

Further Observations Not all properties of the strategies are captured by the target function value and failure rates. The dataset generated for experiment 1 offers other insights relevant to assessing them. One such insight is the use of computational resources by each strategy. It can be measured as the number of simulations run by the RGS controller during one outer simulation run. Figure 24 shows the average number of simulations completed per second over all evaluation runs in experiment 1.

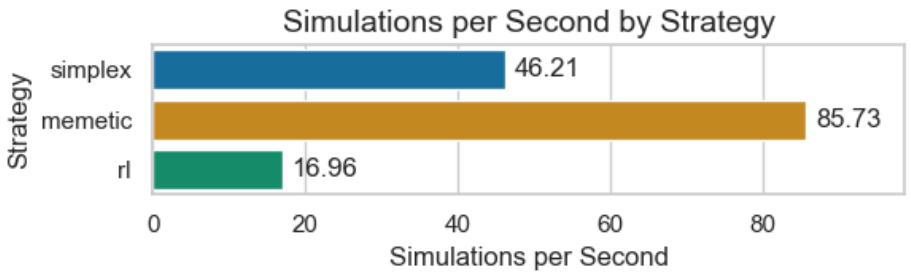

Figure 24: Mean Simulations completed per second by the RGS Controller over all simulation runs, by strategy.

On average, the memetic algorithm completes the most simulations per second. The Downhill-Simplex algorithm completes slightly more than half of that. This can be explained by the limited degree of parallelization that is possible in the Downhill-Simplex algorithm. While the memetic strategy can evaluate as many solutions in parallel as needed, Downhill-Simplex can only run up to $n + 1$ simulations in parallel, where n is the current dimension of the problem, which equals the number of remaining decision points. While the reinforcement learning agent can also utilize parallelization, it may sometimes produce identical solutions from a single observation, which will only be simulated once. Due to being pretrained on the problem, reinforcement learning does not require simulations to explore the solution space of the current problem instance; the generated solutions are only evaluated to sort and validate them. Thus, reinforcement learning requires fewer simulations than the other strategies.

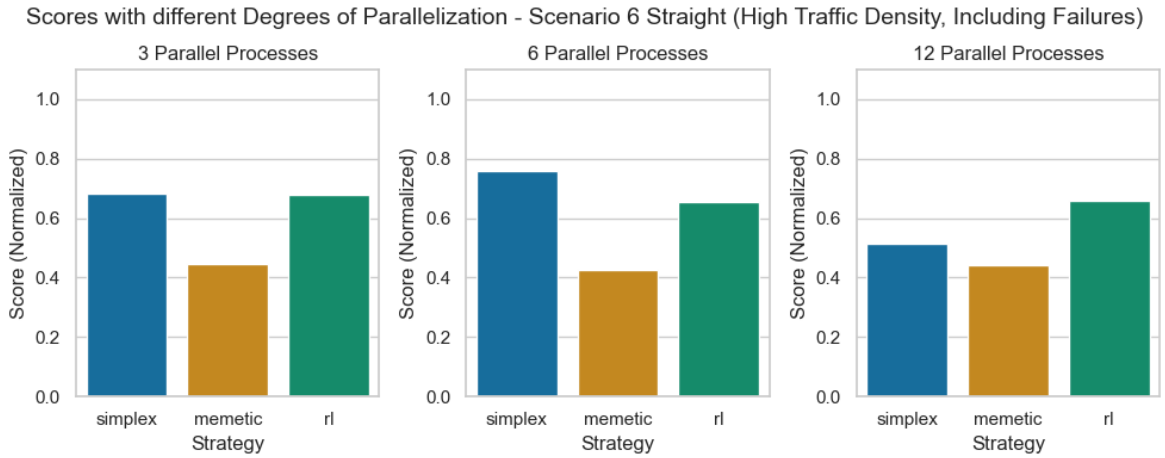

Figure 25: Normalized scores by strategy for a straight crossing in Scenario 6 with high traffic density at different numbers of available parallel processes.

Interestingly, the performance of the memetic algorithm remains almost unaffected by a reduction of available parallel processes, as shown in Figure 25. The results were gathered on the straight variant of scenario 6 with heavy traffic, as it is one of the

computationally most demanding scenarios. The size of the multiprocessing pool was reduced to simulate a lower number of available processors. Downhill-Simplex displays greatly reduced performance with a lower degree of parallelization. When the number of available processors becomes lower than the dimension of the problem, Downhill-Simplex converges more slowly and thus likely delivers a worse solution at t_{DENM}.

Another interesting aspect is the stability of solutions. The target function term $c \cdot dist_E(u, u_{t_{DENM}})$ encourages stable solutions throughout an outer simulation run. It was examined how much the solutions provided over time differ from the initial solution $u_{t_{DENM}}$. The average normalized distance of all solutions received by the outer simulation after t_{DENM} within one simulation run is used as a measure of solution stability. The lower this distance is, the closer solutions sent by the RGS controller are to $u_{t_{DENM}}$ on average. Figure 26 shows the solution stability achieved by the strategies in each scenario.

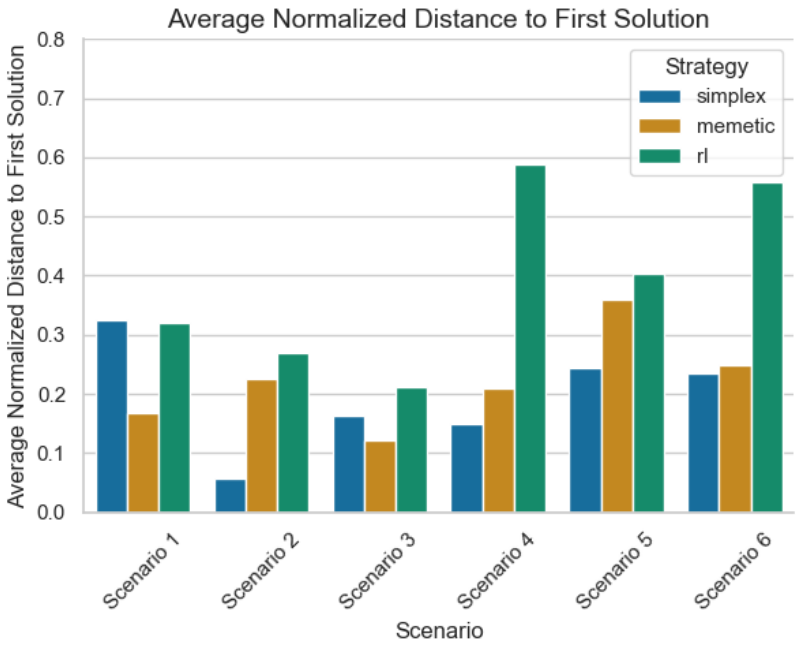

Figure 26: Mean distance of all solutions sent by the RGS controller from $u_{t_{DENM}}$. Normalized for each scenario, grouped by strategy. A lower average normalized distance from the first solution implies a higher stability of subsequent solutions.

It can be seen that reinforcement learning generally provides less stable solutions than the other strategies. Downhill-Simplex provides the most stable solutions in all scenarios except for 1 and 3. Reinforcement learning is less stable because the model was trained to generate solutions purely depending on the state of the environment to retain flexibility. The results do not show a correlation between mean solution distance and score achieved in a scenario. Reinforcement learning outperformed other strategies in most synthetic scenarios, but performed worse in scenario 6. Reinforcement learning also has the least stable solutions in all scenarios other than scenario 1.

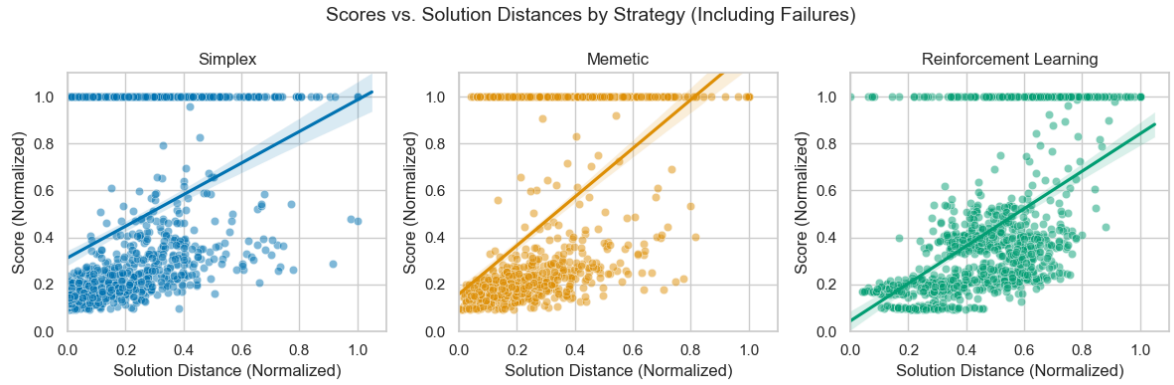

Figure 27: The x-axis of each plot shows the mean normalized distance of received solutions to $u_{t_{DENM}}$. The y-axis shows the normalized score. Each point represents one configuration.

The relationship between score and solution stability of individual simulation runs is examined in Figure 27. For each configuration in experiment 1, the normalized score achieved is put in relation to the normalized mean solution distance. The three subplots display this relationship for each strategy, which is further illustrated by a linear regression line fitted to the results.

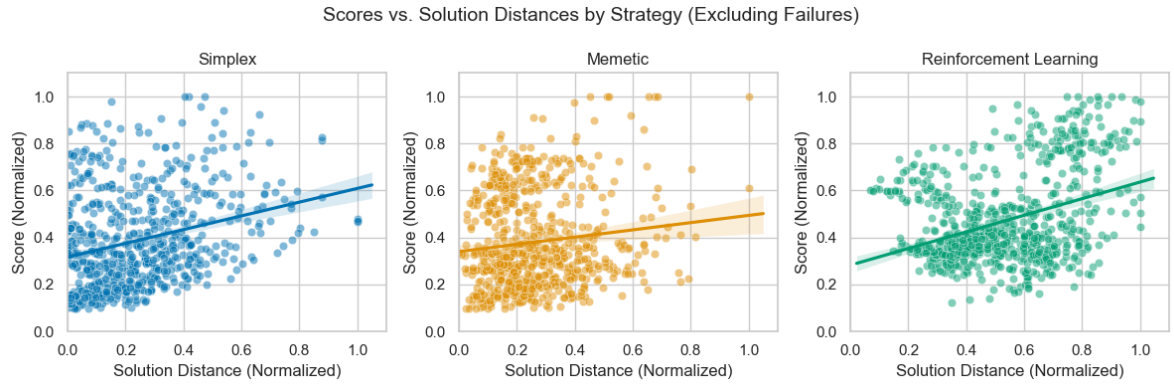

Figure 28: The x-axis shows normalized solution distance, the y-axis shows the normalized score. Each point represents one configuration. The normalization is applied based on the maximum achieved score in each scenario, resulting in a wider distribution of data points than in Figure 27.

All strategies show a similar trend: an increasing mean solution distance leads to a worse score. The data points with a score of 1.0 are mostly configurations that resulted in a failure. For all strategies, the relative amount of failed configurations increases with a larger mean solution distance. The results without failures are visualized in Figure 28. The regression lines are flatter, indicating a weaker relation between solution stability and score. This implies that solution stability does lead to relatively fewer failures, but it does not guarantee increased scores for valid solutions.

6.2 Impact of non-cooperative Vehicles on Optimization

The strategies' ability to cope with non-cooperative vehicles was evaluated in experiment 2. Figure 29 shows the results of outer simulation runs with varying cooperation probabilities p_{coop} for vehicles in Scenario 2. The left subplot displays the mean normalized score achieved by strategies in 50 simulation runs for p_{coop} values between 0.5 and 1.0 when failing configurations are ignored. The right subplot also displays the mean scores for cooperation probabilities, in addition to the failure rates of the strategies at respective p_{coop} values.

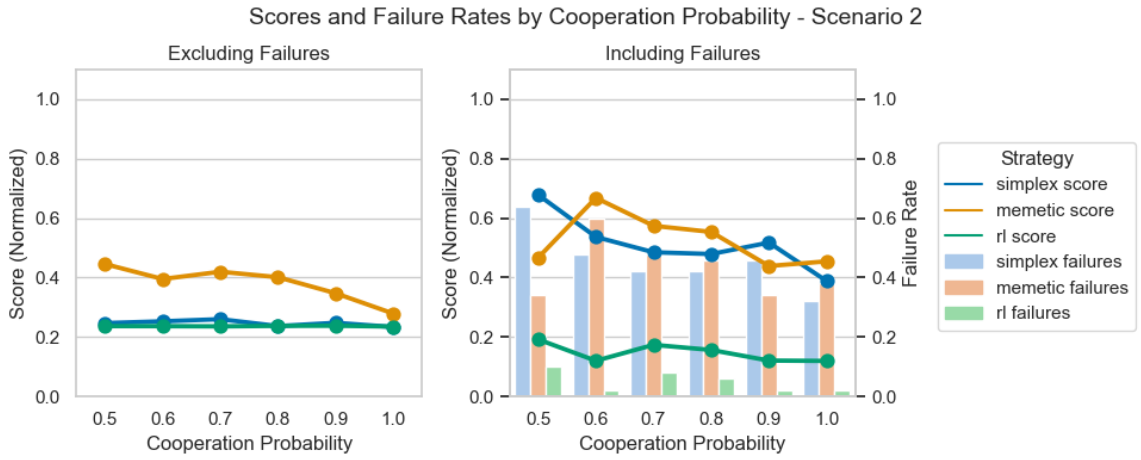

Figure 29: Normalized Scores at varying p_{coop} values for Scenario 2 with high traffic density, by strategy. The right plot includes failed simulation runs. It additionally displays failure rates as bars for each strategy and p_{coop} value.

The results and failure rates at $p_{coop} = 1$ match the values from experiment 1 (see Figures 34 and 35). For decreasing p_{coop} values, a close correlation between score and failure rate can be observed. The scores that exclude failures show that Downhill-Simplex and Reinforcement Learning are mostly unaffected by non-cooperative vehicles in Scenario 2, whereas the performance of the memetic algorithm deteriorates with a decreasing p_{coop} value. The differences in performance of the strategies observed in scenario 2 with full cooperation remain when p_{coop} decreases: reinforcement learning outperforms the other strategies when failures are considered.

An examination of individual results shows reasons why Downhill-Simplex and reinforcement learning seem to be unaffected by a decreasing p_{coop} value when excluding failures. Almost all successful solutions provided by these strategies include the same route for the emergency vehicle: it passes the other vehicles at the traffic light on the left, utilizing the sidewalk. When successful, this approach barely slows down the emergency vehicle, and thus enables constant scores regardless of p_{coop}. The significantly lower failure rate of reinforcement learning stems from its solutions mostly choosing the left-most sublane. The reinforcement learning agent receives a high penalty for collisions during training, and is thus expected to behave cautiously in situations that could cause them.

Downhill-Simplex chooses a route slightly closer to the center of the road. While that route is marginally faster if vehicles cooperate, which the RGS controller assumes, it leads to a higher risk of collisions if vehicles do not adjust their lateral position. The fewer vehicles cooperate, the higher the chance of a collision becomes, which is reflected in the increasing failure rate of Downhill-Simplex with decreasing p_{coop}. Similarly to Downhill-Simplex, the memetic algorithm often chooses a route that would work well when other vehicles fully cooperate, but leads to collisions with non-cooperative vehicles. However, the memetic algorithm appears to be less prone to converging to the same solution in every simulation run. The Downhill-Simplex algorithm will quickly reach a situation in which its simplex is small, and exploring new solutions in different regions of the solution space becomes unlikely. Reinforcement learning generates solutions based on trained behavior and the current observation, also increasing the likelihood of similar solutions over multiple simulation runs. In contrast, the memetic algorithm keeps a large population of candidates. In some simulation runs, a different solution than the described route along the left sidewalk is selected and sent to vehicles in the outer simulation. When successful, these routes have been observed to often lead to the emergency vehicle getting stuck before the traffic light. This explains the higher scores for successful runs.

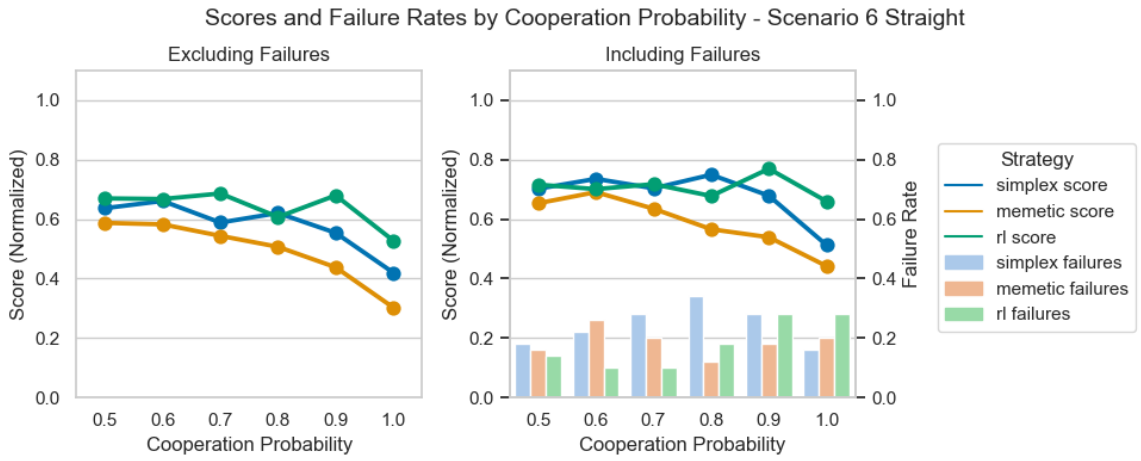

Figure 30: Normalized Scores at varying p_{coop} values for the Tostmannplatz scenario with high traffic density, by strategy. The right plot includes failed simulation runs and displays failure rates as bars for each strategy and p_{coop} value.

The other scenario evaluated for experiment 2 was scenario 6 in the straight crossing variant. An overall trend is easier to identify than in the previous scenario, especially in the successful runs: as p_{coop} decreases, the score worsens for all strategies, as shown in Figure 30. The difference between $p_{coop} = 1$ and $p_{coop} = 0.9$ is significant for all strategies. The mean score of Downhill-Simplex changes from 0.42 to 0.55 at $p_{coop} = 0.9$. For Memetic, it changes from 0.3 to 0.44, and for reinforcement learning from 0.53 to 0.68. As in experiment 1, the memetic algorithm outperforms other strategies when all vehicles cooperate in the configuration (see Figure 46). At $p_{coop} = 0.5$, memetic still has a slight advantage, but the scores of all strategies are closer to each other. The results

indicate a weak correlation between failure rates and scores. However, no overall trend in failure rates is visible over changing p_{coop} values.

One of the most curious aspects of the scenario 6 results is the performance of reinforcement learning, specifically at $p_{coop} = 0.9$. Reinforcement learning generates worse solutions at $p_{coop} = 0.9$ than for any other p_{coop} value, both when excluding and including failures. A correlation to failure rates can explain this observation for the latter case. To gain more insight into the results, the distribution of scores at different p_{coop} values in scenario 6 is visualized in Figure 31.

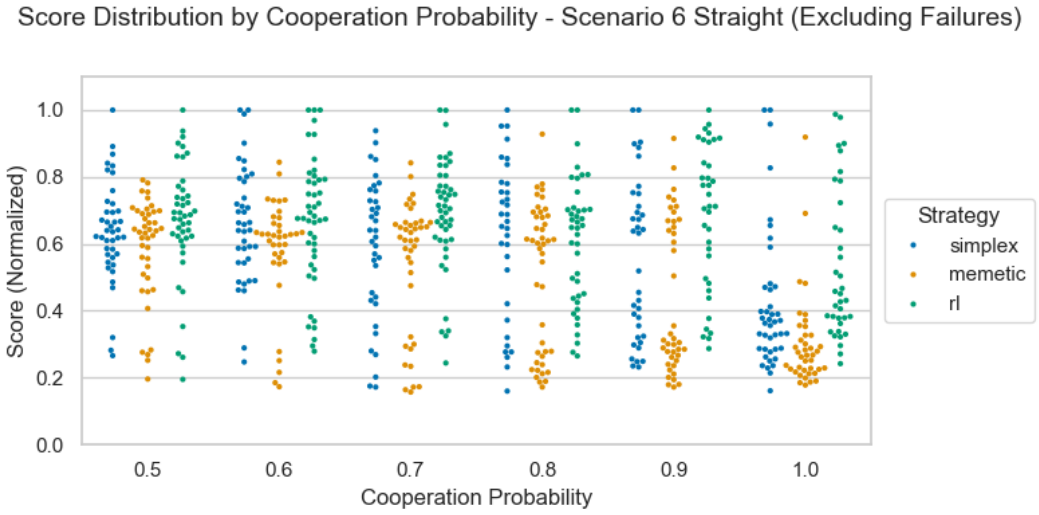

Figure 31: Normalized scores on the y-axis are plotted against different p_{coop} values on the x-axis. Points represent configurations and their respective scores at a certain p_{coop} and are grouped by strategy. Failed simulations are not included.

It can be seen that under full cooperation, most simulation runs controlled by Downhill-Simplex or Memetic have a normalized score between 0.2 and 0.4. At lower cooperation probabilities, most results for all strategies are located in a score range of 0.5 to 0.8. Scenario 6 represents a traffic light-controlled intersection with a long waiting time for the direction the emergency vehicle approaches from. When the RGS controller generates successful solutions for this scenario, two outcomes are possible: Either the emergency vehicle passes the intersection while the traffic light is still red, or it gets stuck in traffic and has to wait until the road ahead is cleared when the traffic light turns green. The visible clusters in the distribution diagram represent these two cases. The relatively low scores below 0.4 mark runs in which the emergency vehicle was able to transition the traffic light smoothly, whereas higher scores indicate that the vehicle got stuck.

The results achieved by the reinforcement learning algorithm are more equally distributed at any p_{coop} the those of other strategies. For the case of $p_{coop} = 0.9$, a small number of configurations have resulted in unusually high scores, leading to a higher mean score. The distribution plot shows that a cooperation chance of 0.9 poses problems for all strategies, as a significant amount of the configurations at that value resulted in a

poor score. At $p_{coop} = 0.7$, the majority of simulation runs for each strategy resulted in a situation in which the emergency vehicle likely could not traverse the intersection before the traffic light turned green.

It can thus be said that the results achieved by all strategies start to deteriorate quickly when non-cooperative vehicles are present. Even if vehicles cooperate with a chance of 90%, the strategies' performance was observed to decrease significantly. In the case of the memetic algorithm, scores increased by 46% at $p_{coop} = 0.9$ compared to the full cooperation case.

7 Discussion

This section discusses and interprets the results of the experiments. It also gives recommendations on further research towards networked rescue mobility based on the findings made in this thesis.

The results of experiment 1 are used to answer research question 1 and evaluate the performance of optimization strategies on synthetic scenarios. As detailed in the last section, reinforcement learning outperformed other strategies, as long as failed runs are considered and given a poor score. Reinforcement learning produced an exceptionally low failure rate and achieved good results, especially in high traffic density. The other strategies could partially deliver better scores if failures were not taken into account. For a verdict on the performance, the implications of failed scenarios must first be discussed.

The RGS system does not directly control vehicles in a real-world situation. Connected and autonomous vehicles receive the emergency corridor geometry, but evasive maneuvers must be executed by the driver or the vehicle itself. The situations that lead to failures in outer simulations during the experiments might thus not occur in real life. Both human drivers and autonomous vehicles would prioritize their safety during evasive maneuvers, thereby reducing the likelihood of colliding with other evading vehicles or oncoming traffic. Many deadlock situations encountered in the outer simulation could be resolved by drivers that are not constrained by SUMO's routes and lanes. However, failures in the outer simulation can also be interpreted as potentially dangerous situations. If two vehicles collided in the outer simulation, they might have gotten very close to each other under similar circumstances in the real world.

Under these considerations, it can be said that reinforcement learning provides the safest solutions in synthetic scenarios, but struggles with more complex scenarios like the intersection in scenario 5. It generally outperforms other strategies in simple scenarios. Downhill-Simplex and the memetic algorithm provide solutions that achieve better target function scores in complex scenarios, as long as failures are ignored.

The results of the real-world scenario 6 show a poorer performance of reinforcement learning than in the synthetic scenarios. The Downhill-Simplex and memetic algorithms performed well in scenario 6, providing both lower failure rates and better scores than reinforcement learning. Memetic performed best in the experiment, as it provided better-scoring solutions in high traffic density than Downhill-Simplex.

In combination with the unusually good scores achieved by RL in high traffic density configurations, it seems plausible that the RGS controller's reinforcement learning agent has been overfitted to training data, or that the training data does not contain enough information to learn generalizable behavior from it. This could be addressed with a larger, more diverse training dataset, including real-world scenarios, in future research. The model used in the RGS controller has possibly reached what is known as an irreducible error [52] with the generated training data. This implies that further performance improvements require a larger training dataset; more research in this direction is thus recommended. Due to the good results in synthetic scenarios and the low computational requirements at runtime, reinforcement learning is a promising approach for rescue corridor optimization despite its shortcomings in parts of this evaluation.

Should the issues faced by reinforcement learning not be resolved, the experimental results suggest using the memetic algorithm or a similar approach. Memetic performs similarly to Downhill-Simplex in synthetic scenarios and better in scenario 6, especially in high traffic density. It has the additional advantage of not being negatively influenced by a reduced degree of parallelization. In the parallel Downhill-Simplex, the number of available processors determines the number of points in the simplex that can be evaluated at the same time. Its performance thus decreases when the number of processors becomes lower than the problem dimension.

The strategies manage to perform at least as good the static baseline in synthetic scenarios on average. If failures are not considered, the strategies mostly provide significantly better solutions overall. There are exceptions; for example, scenario 6, in which reinforcement learning performs worse than the baseline when failures are excluded, or scenario 4, in which all strategies perform worse than the baseline if failures are included. Scenario 4 models a two-lane road with oncoming traffic. The static solution keeps the emergency vehicle on the right lane for the entire duration of the scenario. This guarantees a failure-free completion. Some solutions provided by the strategies result in better scores, as overtaking maneuvers are performed. Others result in collisions with oncoming traffic. When aggregating the results, the collisions damage the score more than the slightly better solutions improve it. As the exclusion of failed results on scenario 4 shows, better solutions than the static solution exist (see Figure 38). Overall, strategies can provide better results than the static baseline in most cases. This proves the viability of a dynamic optimization approach, even if it does not outperform static solutions in all scenarios.

Experiment 2 provides insights into the robustness of the strategies towards non-cooperative vehicles. The scores a strategy achieves at p_{coop} values smaller than 1 are related to the performance of the respective strategy at full cooperation. The results from scenario 2 might lead to the conclusion that reinforcement learning is less affected by non-cooperative vehicles than other strategies. As detailed in section 6.2, certain properties of the solutions provided by reinforcement learning led to a comparatively good performance and an especially low failure rate. The results of scenario 6 in experiment 2 show that reinforcement learning cannot generally be assumed to perform better than other strategies when non-cooperative vehicles are present. Instead, it becomes apparent that all strategies struggle even with low amounts of non-cooperative vehicles, and optimization performance rapidly decreases.

The high sensitivity of optimization strategies to non-cooperative vehicles is concerning. Connected vehicles are currently just in the process of adoption, and non-connected vehicles will likely remain on the streets for decades, significantly reducing the potential usefulness of an emergency corridor optimization system like the RGS. The experimental results show that any strategy in the RGS controller struggles when its predictions are no longer accurate. A mechanism to inform the controller about vehicles that have received a DENM and will react to it could help to improve prediction quality. It is thus proposed to study the effects of vehicles sending an acknowledgment message after receiving a DENM with a rescue corridor. While this form of request-response communication is currently not provided by V2X, it could be realized by including the latest received emergency

corridor in the CAM that connected vehicles send regularly.

Findings from field tests should also be considered in the design of future rescue corridor optimization systems. While the experimental results do not suggest a correlation between score and solution stability, field tests conducted during the GAIA-X4 AMS project have shown a high importance of stable solutions in practice. Autonomous vehicles in the real world react more slowly to the DENM than they do in the simulation. Frequently changing DENM corridors posed little issues for vehicles in SUMO, whereas in field tests, vehicles were partially unable to react accordingly if the DENM changed quickly. The field tests also highlighted the need for high-fidelity simulation scenarios when deploying the RGS. The OpenStreetMap-based SUMO scenario used in GAIA-X4 AMS was not accurate enough for DENM generation and required manual readjustment. Autonomous vehicles in the field tests also used different algorithms to react to the emergency corridor. Details about these algorithms should be implemented in future optimization systems to improve prediction accuracy.

While using a more complex simulation for prediction is not feasible with current computational resources, SUMO has limitations that need to be considered for future research in this field. Sidewalks are modeled as lanes with a lower speed limit and access restrictions in the evaluation scenarios. Obstacles typically present on sidewalks, such as lamp posts or trash cans, are not modeled, even though they would influence rescue corridor formation in the real world. As explained above, the driver model used in the experiments can also lead to unrealistic outcomes. It is therefore advised to pursue more accurate models of both the environment and vehicle behavior in future research.

8 Conclusion

To conclude this thesis, this section answers the research questions and summarizes the findings made.

Research question 1 aims to evaluate the comparative performance of the optimization strategies on synthetic scenarios. The reinforcement learning strategy performs better than others in the synthetic evaluation scenarios. This is achieved mainly due to a low failure rate. When failed runs are not considered, the Downhill-Simplex and memetic algorithms outperform reinforcement learning in scenario 5, which models an intersection and is the most complex of the synthetic scenarios. Unexpected results, such as better performance on higher traffic density and the performance difference of reinforcement learning between synthetic and real-world scenarios, raise concerns about potential overfitting of the reinforcement learning agent.

The performance of strategies in real-world scenarios is examined by research question 2 and evaluated in scenario 6. The scenario is modeled after the Tostmannplatz in Brunswick and uses real traffic demand data. In this scenario, the memetic algorithm performed best, followed by Downhill-Simplex, and reinforcement learning performed worst. Not only did reinforcement learning provide worse scores in this scenario, but it also resulted in higher failure rates. The Downhill-Simplex and memetic algorithms achieved similar failure rates and near-equal scores in average traffic density situations. Memetic proved superior to the other strategies in high traffic density.

Research question 3 concerns itself with the influence of non-cooperative vehicles on optimization strategies. An experiment on the synthetic scenario 2 provided inconclusive results. However, the evaluation of the real-life scenario 5 showed that all strategies suffer from reduced performance when non-cooperative vehicles are involved. Even a small amount of non-cooperative vehicles leads to a relatively large loss in optimization performance. It is therefore recommended to establish a confirmation protocol for the emergency corridor DENM to improve prediction accuracy.

The results gathered from the experiments based on the research questions allow comparisons to be made between the optimization strategies. However, when choosing an algorithm for the rescue corridor optimization task, other considerations should not be disregarded. Further examinations of the experimental results revealed that the memetic algorithm uses the most processing power due to its high utilization of parallelization, but retains its performance even when the number of parallel processes is reduced. The Downhill-Simplex and Memetic algorithms both provide more stable solutions than reinforcement learning on average. While the experimental results do not show a strict correlation between scores and simulation stability, field tests have shown that stable solutions are valuable.

Apart from the experimental results, this thesis provides an architecture and implementation for a dynamic optimization system for rescue mobility optimization. The RGS controller is extensible and compatible with standardized V2X messages. While possible improvements to the system have been uncovered during the evaluation, such as a more accurate model for vehicle behavior, it provides a foundation for future research and smart rescue mobility applications.

References

[1] Peter Sefrin. Hilfsfrist versus versorgungszeit. *Notarzt*, 31(02):72–74, 2015.

[2] Laura Bieker-Walz. Verkehrsmanagement für Einsatzfahrzeuge. 2021.

[3] Arnaud Braud, Gaël Fromentoux, Benoit Radier, and Olivier Le Grand. The road to european digital sovereignty with gaia-x and idsa. *IEEE network*, 35(2):4–5, 2021.

[4] Haobo Fu, Peter R. Lewis, Bernhard Sendhoff, Ke Tang, and Xin Yao. What are dynamic optimization problems? In *2014 IEEE Congress on Evolutionary Computation (CEC)*, pages 1550–1557, Beijing, China, July 2014. IEEE.

[5] Satyajith Amaran, Nikolaos V. Sahinidis, Bikram Sharda, and Scott J. Bury. Simulation optimization: A review of algorithms and applications, June 2017. arXiv:1706.08591.

[6] Chun-Wei Tsai, Tzu-Chi Teng, Jian-Ting Liao, and Ming-Chao Chiang. An effective hybrid-heuristic algorithm for urban traffic light scheduling. *Neural Computing and Applications*, 33, 12 2021.

[7] Rodrigo Gutiérrez-Moreno, Rafael Barea, Elena López-Guillén, Javier Araluce, and Luis M. Bergasa. Reinforcement Learning-Based Autonomous Driving at Intersections in CARLA Simulator. *Sensors*, 22(21):8373, November 2022.

[8] EN 302 663 V1.3.1. Intelligent Transport Systems (ITS); ITS-G5 Access layer specification for Intelligent Transport Systems operating in the 5 GHz frequency band. Standard, European Telecommunications Standards Institute, October 2019.

[9] Fabian De Ponte Müller, Ibrahim Rashdan, Martin Schmidhammer, and Stephan Sand. Its-g5 and c-v2x link level performance measurements. In *ITS World Congress 2021*, 2021.

[10] Waigand et al. Kaul, Schieben. *Projektergebnisse des 5G-Reallabor in der Mobilitätsregion Braunschweig-Wolfsburg : Schlussbericht des Verbundvorhabens*. Deutsches Zentrum für Luft- und Raumfahrt e.V. *DLR*, Köln, 2023.

[11] Yilin Li, Jian Luo, Richard A. Stirling-Gallacher, Zhongfeng Li, and Giuseppe Caire. Multihop routing for data delivery in V2X networks. *CoRR*, abs/1901.04962, 2019.

[12] EN 302 637-2 V1.4.1. Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service. Standard, European Telecommunications Standards Institute, April 2019.

[13] TS 103 324 V2.1.1. Intelligent Transport System (ITS); Vehicular Communications; Basic Set of Applications; Collective Perception Service; Release 2. Standard, European Telecommunications Standards Institute, June 2023.

[14] TS 103 831 V2.2.1. Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Decentralized Environmental Notification Service; Release 2. Standard, European Telecommunications Standards Institute, April 2024.

[15] Hellmut Schumacher. Technische anforderungen an rechnergesteuerte betriebsleitsysteme - Übertragungsverfahren datenfunk; ergänzung 2 "datensatz für meldesysteme", 1990.

[16] Michael Klöppel-Gersdorf and Joerg Holfeld. A tool for v2x infrastructure placement. In *International Conference on Vehicle Technology and Intelligent Transport Systems 2025*, 2025.

[17] Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, and Evamarie Wießner. Microscopic traffic simulation using sumo. In *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018.

[18] Jurij Kuzmic and Günter Rudolph. Unity 3d simulator of autonomous motorway traffic applied to emergency corridor building. In *IoTBDS*, pages 197–204, 2020.

[19] Shalini Yadav and Rahul Rishi. Algorithm for creating optimized green corridor for emergency vehicles with minimum possible disturbance in traffic. 13(1):84–95.

[20] Biru Rajak, Shrabani Mallick, and Dharmender Singh Kushwaha. Creating a dynamic real time green corridor and assessing its impact on normal traffic flow. 171:2–11.

[21] Sten Ruppe and Ronald Nippold. Bevorrechtigung von einsatzfahrzeugen mittels einer dezentralen lsa-steuerung. In *Abschlusspräsentation Forschungsprojekt SIRENE*, April 2021.

[22] Aparajit Garg, Anchal Kumar Gupta, Divyansh Shrivastava, Yash Didwania, and Prayash Jyoti Bora. Emergency vehicle detection by autonomous vehicle. *International Journal of Engineering Research & Technology (IJERT)*, 8(05):190–194, 2019.

[23] Hongyi Sun, Xinyi Liu, Kecheng Xu, Jinghao Miao, and Qi Luo. Emergency Vehicles Audio Detection and Localization in Autonomous Driving, October 2021. arXiv:2109.14797.

[24] Amit Ashish, R Gandhiraj, and Manoj Panda. V2x based emergency corridor for safe and fast passage of emergency vehicle. In *2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, pages 1–7, 2021.

[25] Nitish Kumar, Hershita Shukla, and P. Rajalakhsmi. V2X Enabled Emergency Vehicle Alert System, March 2024. arXiv:2403.19402.

[26] Jonas Klemmt; Researcher at Feuerwehr Braunschweig. Interview, Mai 2024.

[27] Michael Till Beck, Sebastian Feld, Claudia Linnhoff-Popien, and Uwe Pützschler. Mobile edge computing: Ein enabler für latenzsensitive mobilfunk-services. 39(2):108–114.

[28] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, 2009.

[29] Simon Steuernagel. Track-to-track association and fusion for cooperative automotive environment perception, February 2019.

[30] OASIS Standard. Mqtt version 3.1. 1. *URL http://docs. oasis-open. org/mqtt/mqtt/v3*, 1:29, 2014.

[31] Tobias Dam, Lukas Daniel Klausner, Sebastian Neumaier, and Torsten Priebe. A survey of dataspace connector implementations.

[32] Alberto Díaz-Álvarez, Miguel Clavijo, Felipe Jiménez, and Francisco Serradilla. Inferring the driver's lane change intention through LiDAR-based environment analysis using convolutional neural networks. 21(2):475.

[33] Ning Ye, Zhong-qin Wang, Reza Malekian, Qiaomin Lin, and Ru-chuan Wang. A method for driving route predictions based on hidden markov model. 2015:1–12.

[34] KS Holkar and Laxman M Waghmare. An overview of model predictive control. *International Journal of control and automation*, 3(4):47–63, 2010.

[35] S. Krauß. Microscopic modeling of traffic flow: Investigation of collision free vehicle dynamics. Technical report, 1998. LIDO-Berichtsjahr=1999,.

[36] J. A. Nelder and R. Mead. A Simplex Method for Function Minimization. *The Computer Journal*, 7(4):308–313, January 1965.

[37] Shintaro Takenaga, Yoshihiko Ozaki, and Masaki Onishi. Practical initialization of the Nelder–Mead method for computationally expensive optimization problems. *Optimization Letters*, 17(2):283–297, March 2023.

[38] Jade Yu Cheng and Thomas Mailund. Ancestral population genomics using coalescence hidden Markov models and heuristic optimisation algorithms. *Computational Biology and Chemistry*, 57:80–92, August 2015.

[39] Donghoon Lee and Matthew Wiswall. A Parallel Implementation of the Simplex Function Minimization Routine. *Computational Economics*, 30(2):171–187, August 2007.

[40] John H. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. Complex adaptive systems. MIT Press, nachdr. edition.

[41] Pablo Moscato and Carlos Cotta. A Gentle Introduction to Memetic Algorithms. In Fred Glover and Gary A. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57, pages 105–144. Kluwer Academic Publishers, Boston, 2003.

[42] Jim Smith and Frank Vavak. Replacement strategies in steady state genetic algorithms: Static environments. *Foundations of genetic algorithms*, 5:219–233, 1999.

[43] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms.

[44] Fei Ye, Xuxin Cheng, Pin Wang, Ching-Yao Chan, and Jiucai Zhang. Automated Lane Change Strategy using Proximal Policy Optimization-based Deep Reinforcement Learning, May 2020. arXiv:2002.02667.

[45] Xiaochang Chen, Jieqiang Wei, Xiaoqiang Ren, Karl H. Johansson, and Xiaofan Wang. Automatic Overtaking on Two-way Roads with Vehicle Interactions Based on Proximal Policy Optimization. In *2021 IEEE Intelligent Vehicles Symposium (IV)*, pages 1057–1064, Nagoya, Japan, July 2021. IEEE.

[46] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust Region Policy Optimization, April 2017. arXiv:1502.05477.

[47] S. Kullback and R. A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79 – 86, 1951.

[48] Sanjna Siboo, Anushka Bhattacharyya, Rashmi Naveen Raj, and S. H. Ashwin. An empirical study of ddpg and ppo-based reinforcement learning algorithms for autonomous driving. *IEEE Access*, 11:125094–125108, 2023.

[49] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[50] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. Gymnasium: A standard interface for reinforcement learning environments.

[51] Zijian Hu, William H. K. Lam, S. C. Wong, Andy H. F. Chow, and Wei Ma. Turning Traffic Monitoring Cameras into Intelligent Sensors for Traffic Density Estimation, October 2021. arXiv:2111.00941.

[52] Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory F. Diamos, Heewoo Jun, Hassan Kianinejad, Md. Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically. *CoRR*, abs/1712.00409, 2017.

Appendix

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und noch nicht für andere Prüfungen eingereicht habe. Sämtliche Quellen einschließlich Internetquellen, die unverändert oder abgewandelt wiedergegeben werden, insbesondere Quellen für Texte, Grafiken, Tabellen und Bilder, sind als solche kenntlich gemacht. Mir ist bekannt, dass bei Verstößen gegen diese Grundsätze ein Verfahren wegen Täuschungsversuchs bzw. Täuschung eingeleitet wird.

Berlin, den 14. Juli 2025

Figures

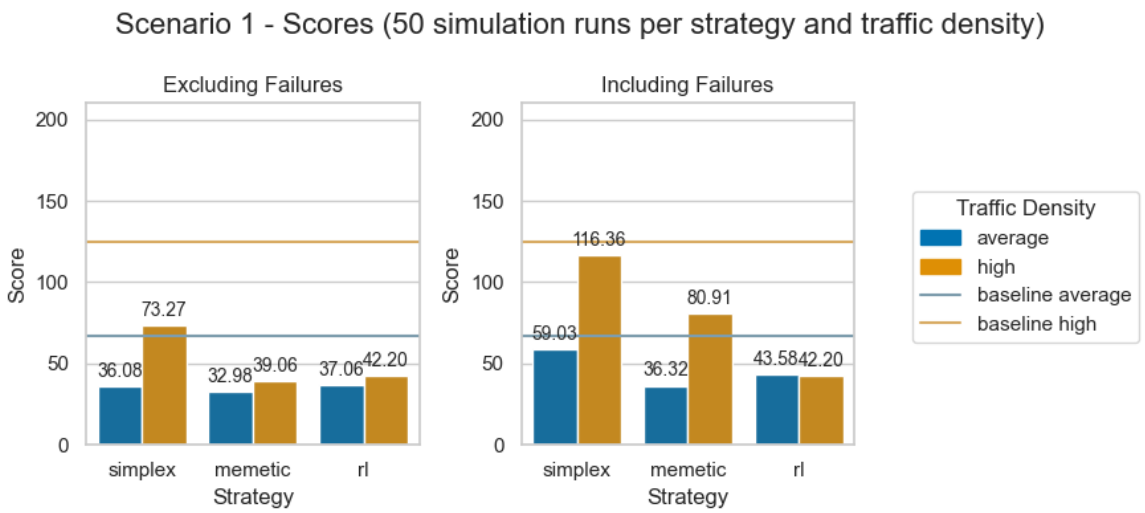

Figure 32

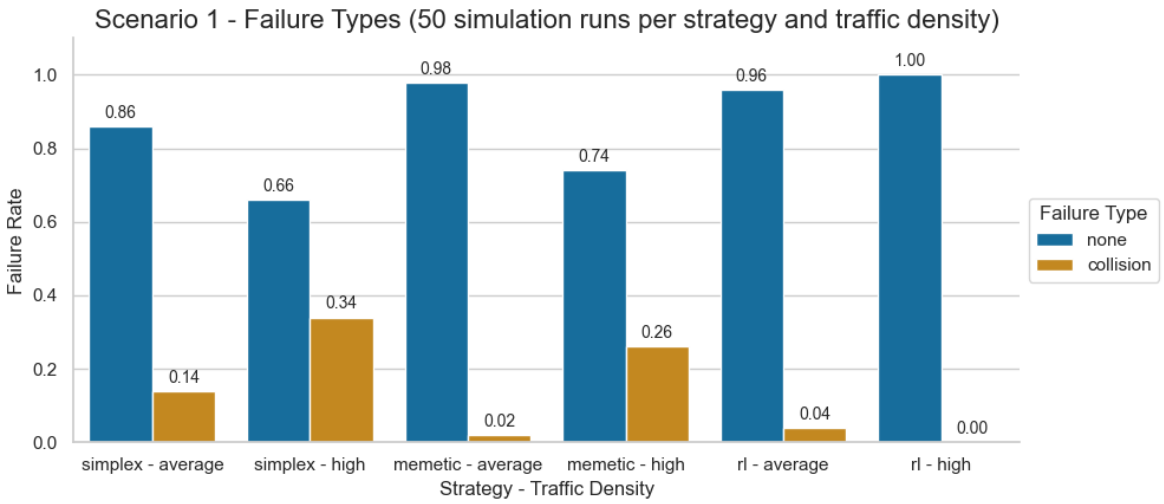

Figure 33

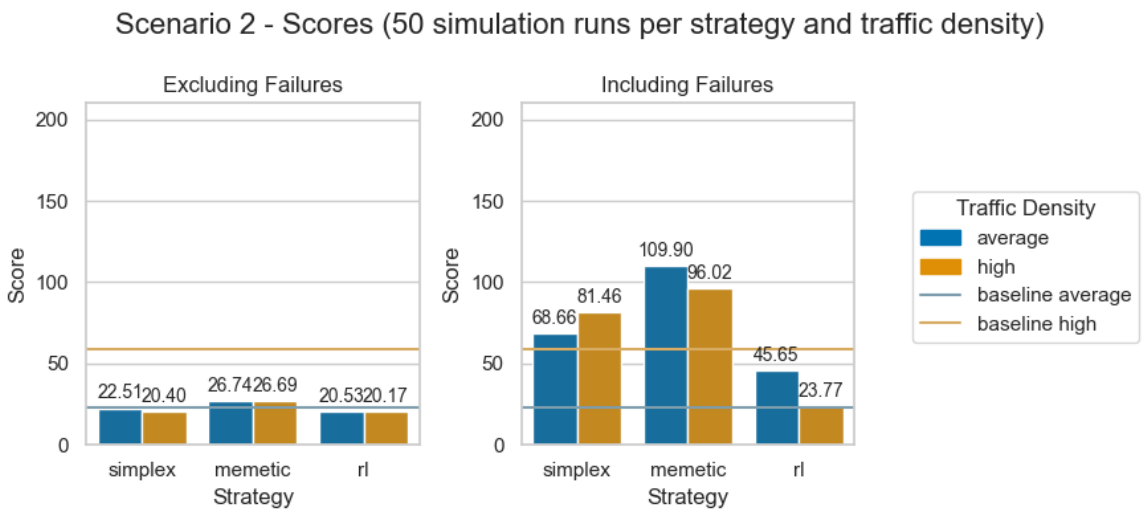

Figure 34

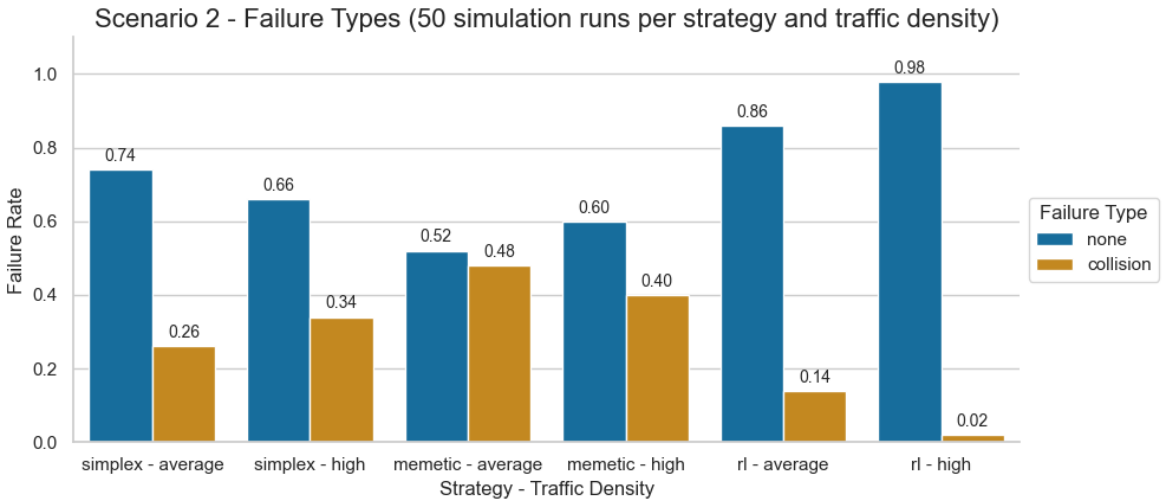

Figure 35

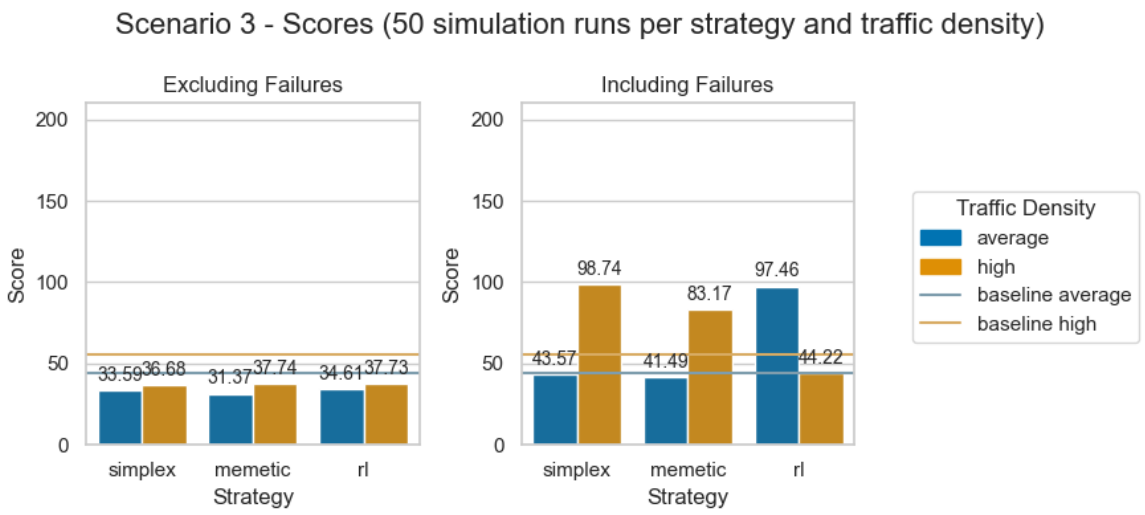

Figure 36

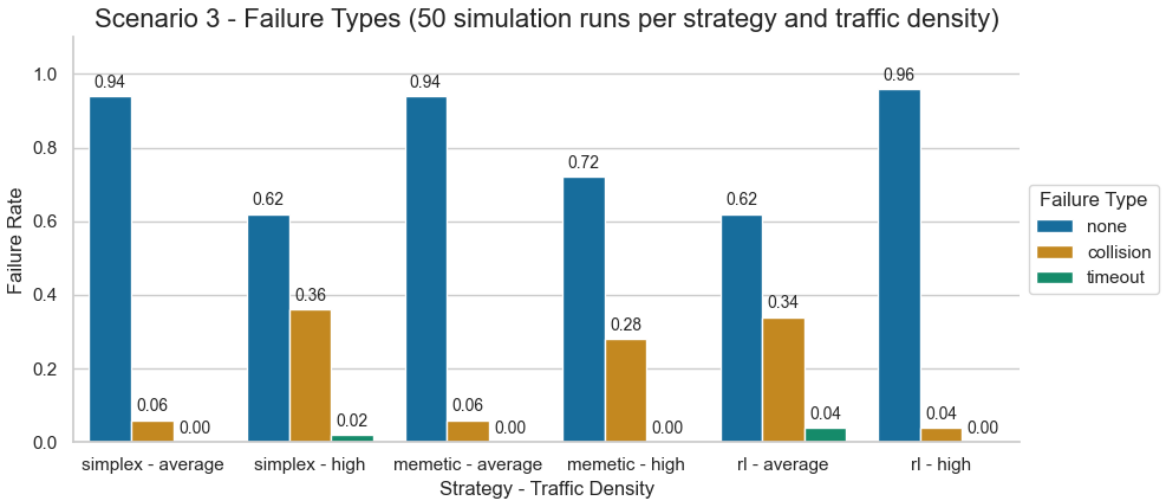

Figure 37

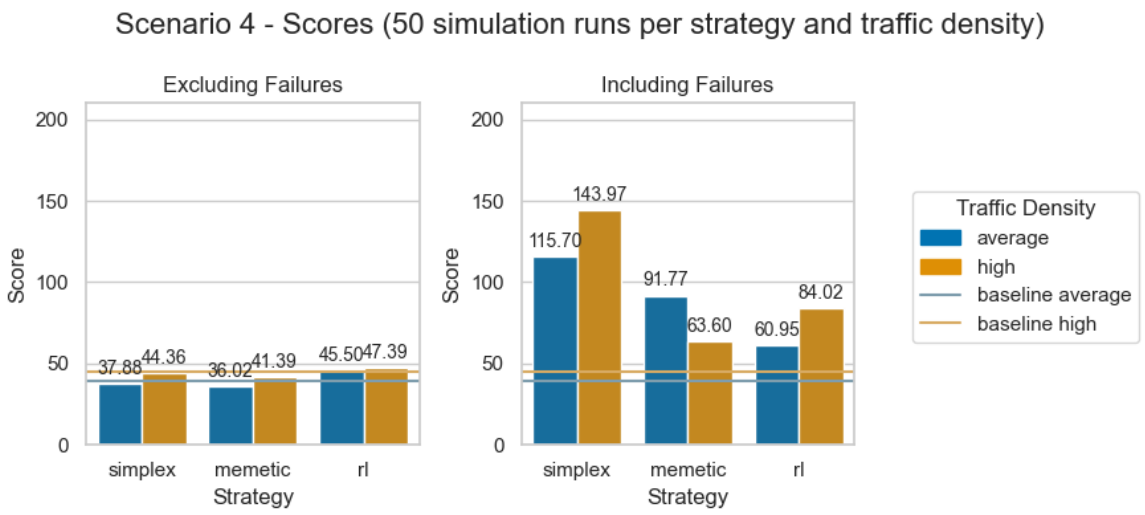

Figure 38

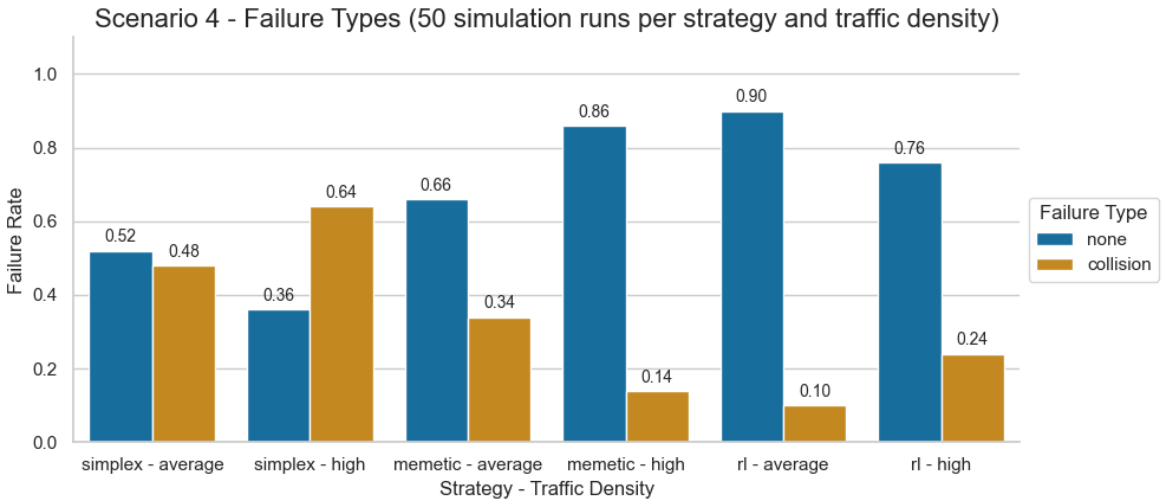

Figure 39

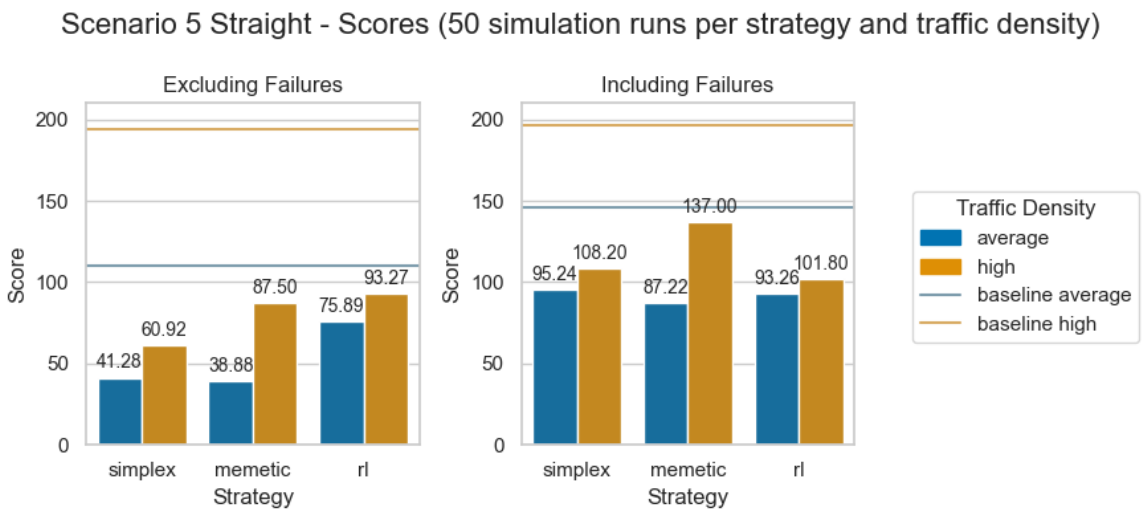

Figure 40

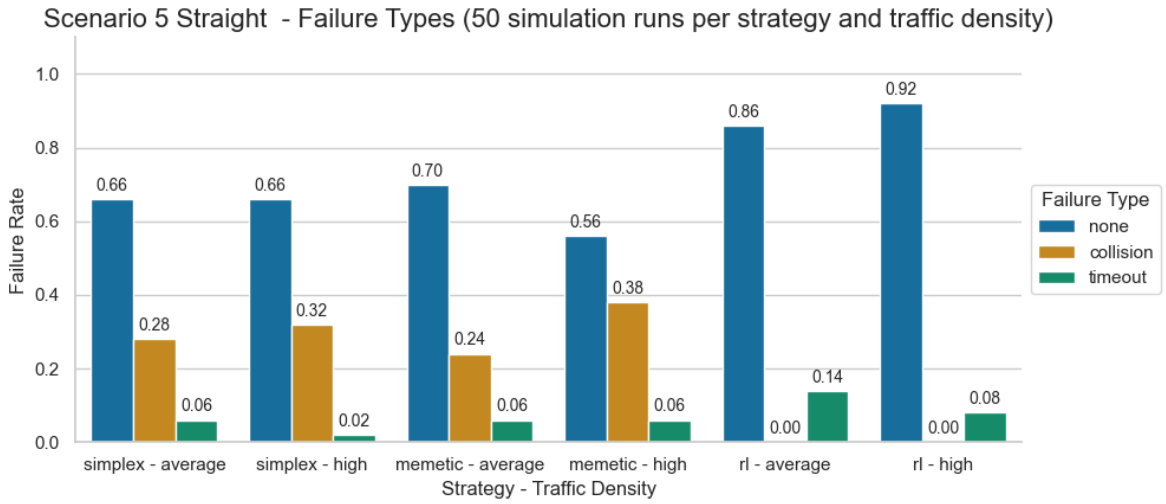

Figure 41

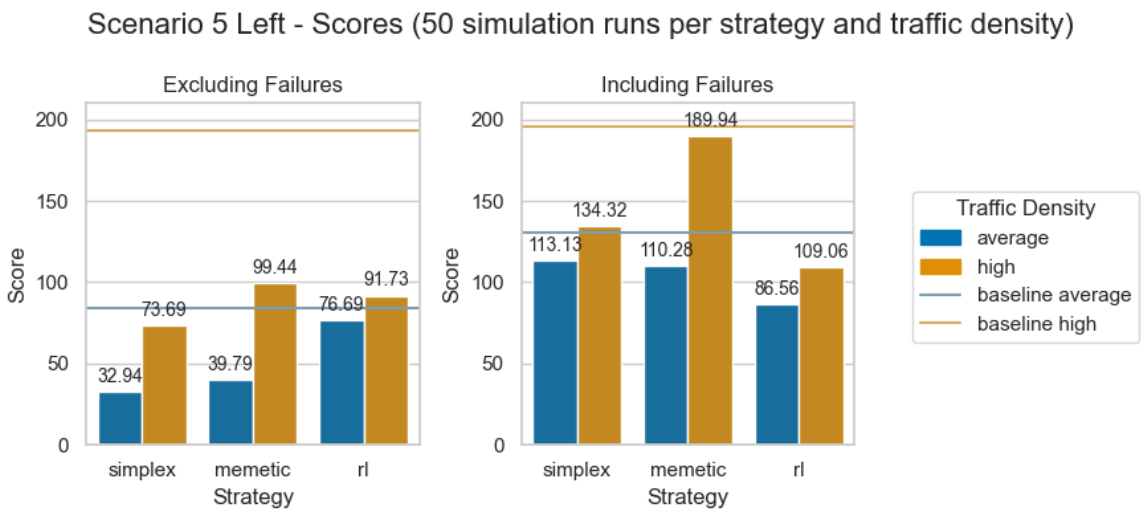

Figure 42

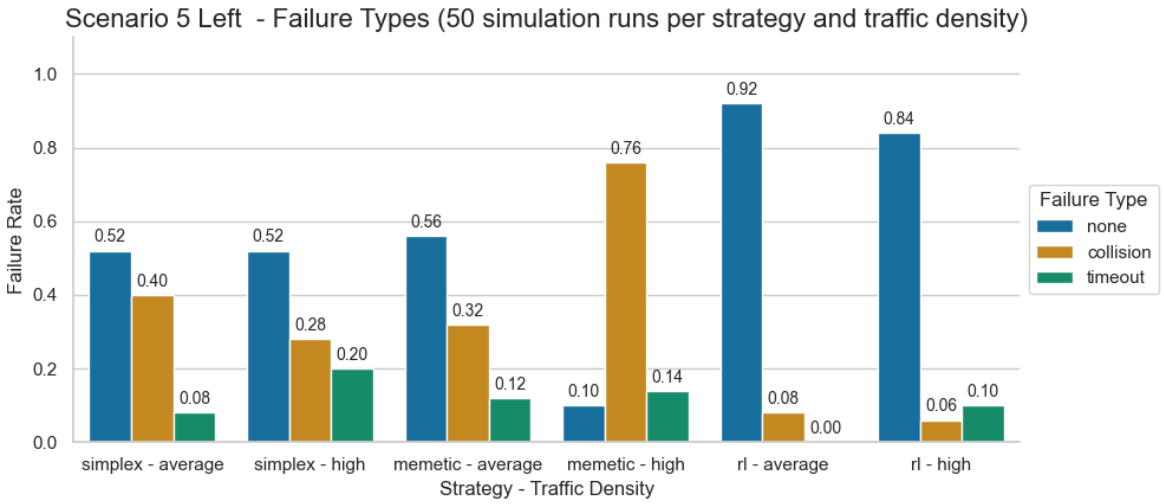

Figure 43

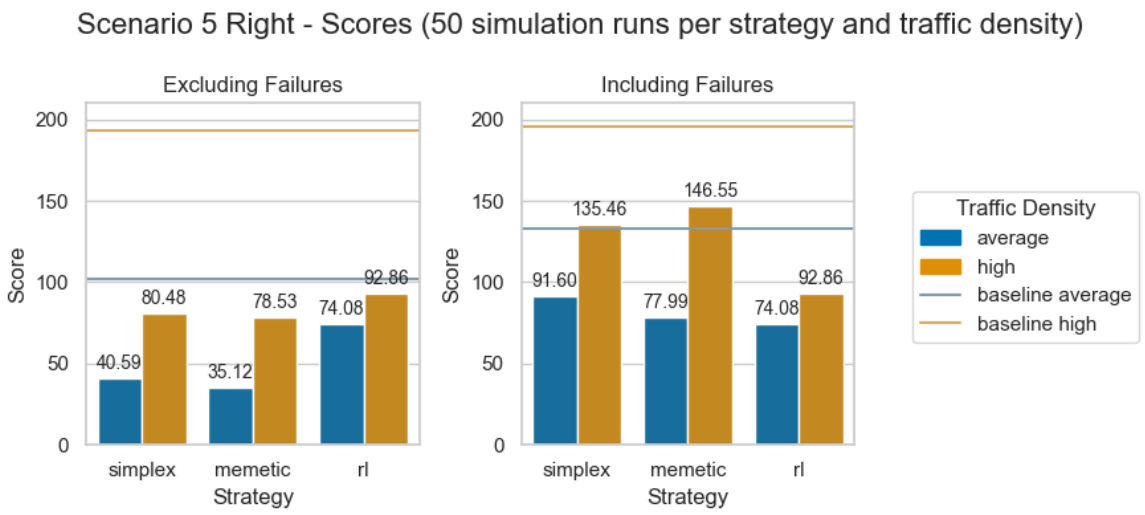

Figure 44

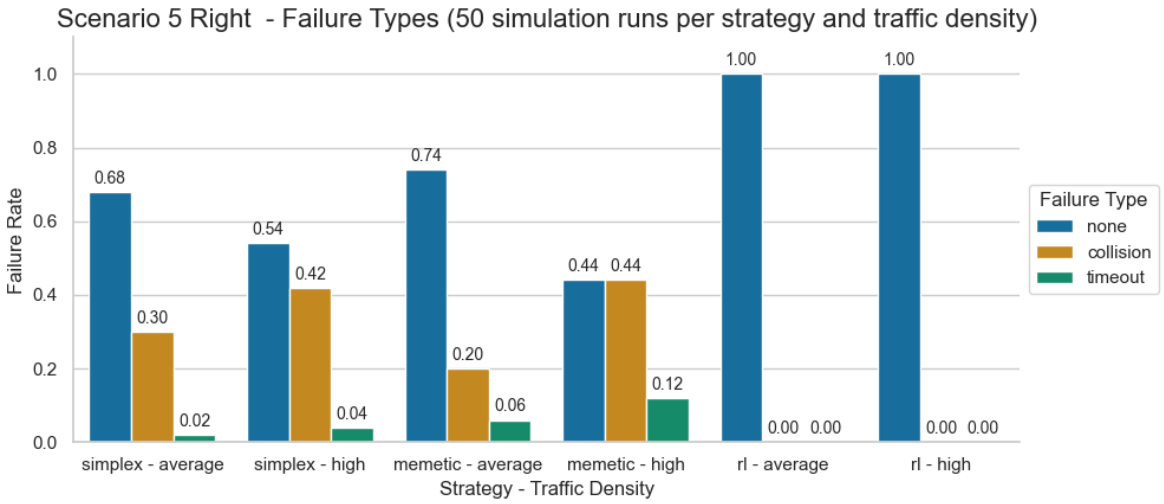

Figure 45

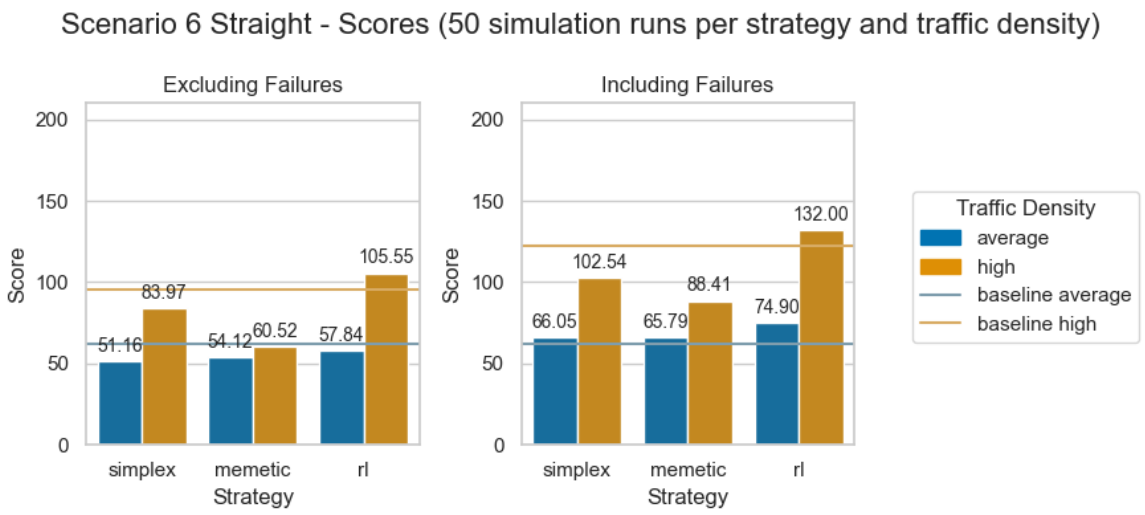

Figure 46

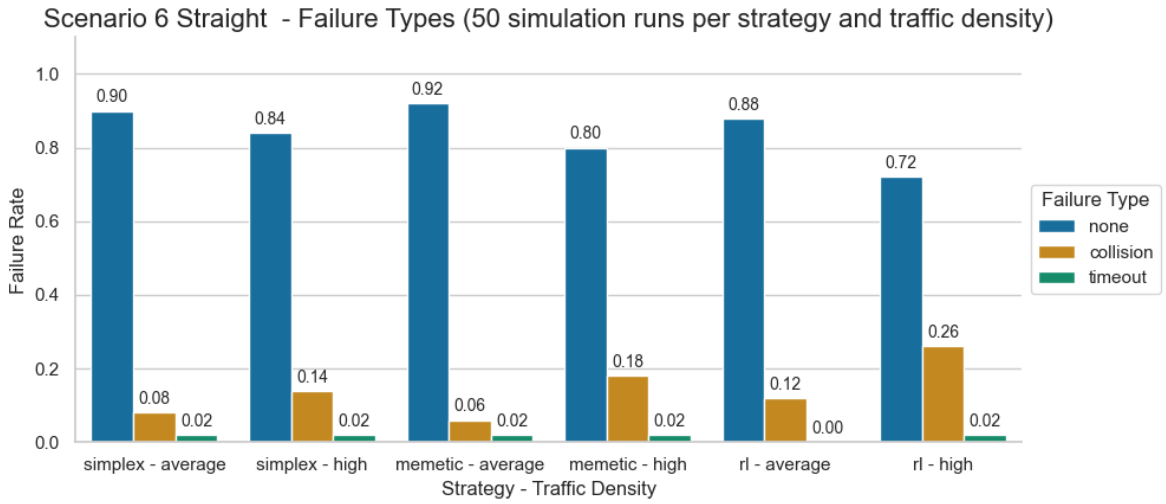

Figure 47

Figure 48

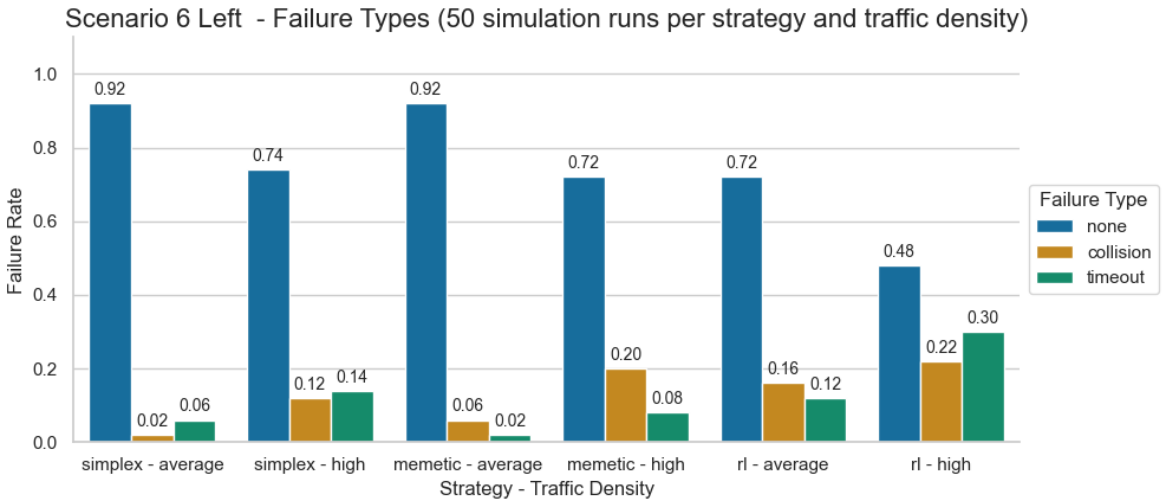

Figure 49

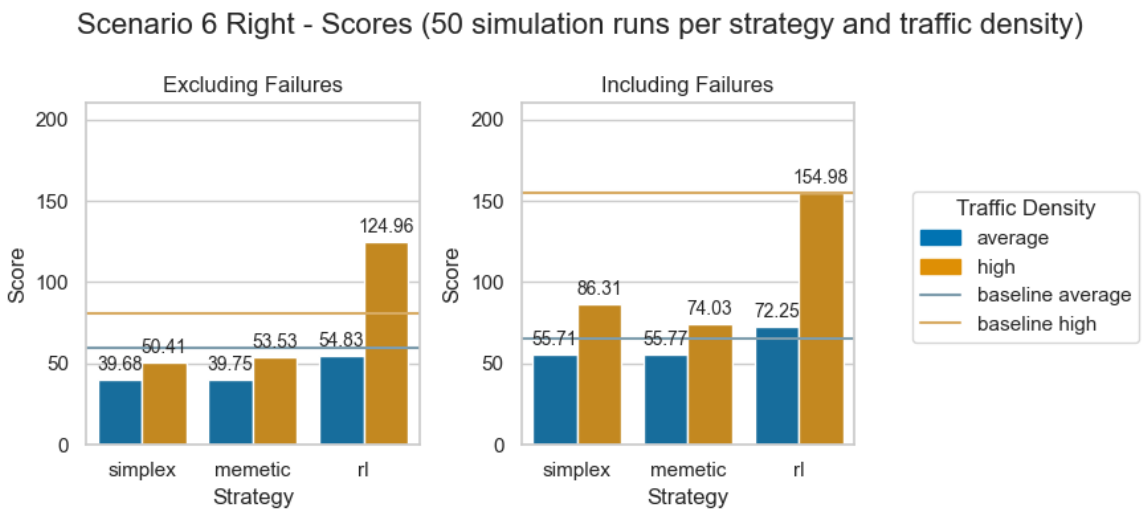

Figure 50

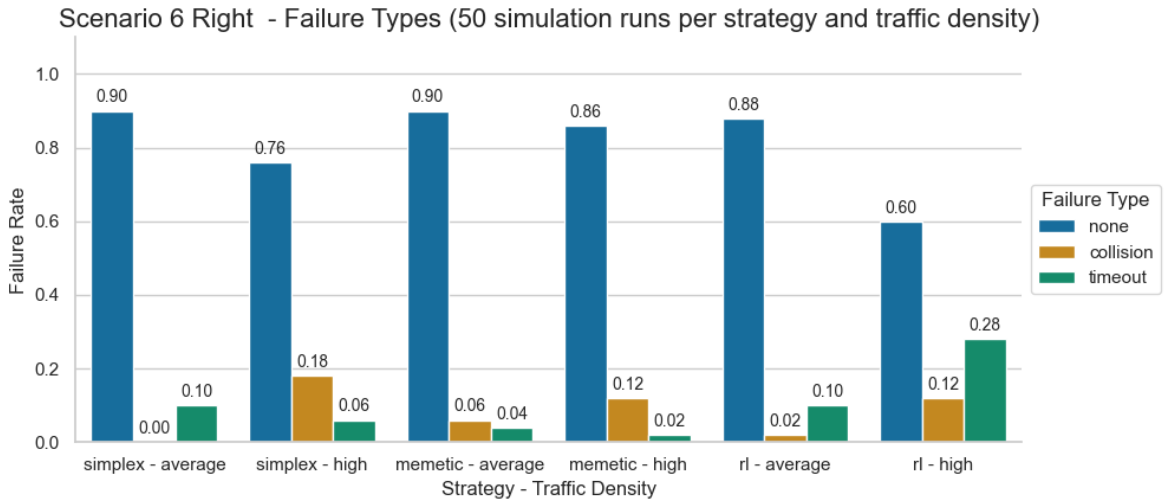

Figure 51